

32nd International Symposium on Theoretical Aspects of Computer Science

STACS'15, March 4–7, 2015, Garching, Germany

Edited by

Ernst W. Mayr
Nicolas Ollinger



Editors

Ernst W. Mayr	Nicolas Ollinger
Fakultät für Informatik	LIFO
Technische Universität München	Université d'Orléans
mayr@in.tum.de	nicolas.ollinger@univ-orleans.fr

ACM Classification 1998

F.1.1 Models of Computation, F.2.2 Nonnumerical Algorithms and Problems, F.4.1 Mathematical Logic, F.4.3 Formal Languages, G.2.1 Combinatorics, G.2.2 Graph Theory

ISBN 978-3-939897-78-1

Published online and open access by

Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at <http://www.dagstuhl.de/dagpub/978-3-939897-78-1>.

Publication date

February, 2015

Bibliographic information published by the Deutsche Nationalbibliothek

The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at <http://dnb.d-nb.de>.

License

This work is licensed under a Creative Commons Attribution 3.0 Unported license (CC-BY 3.0): <http://creativecommons.org/licenses/by/3.0/legalcode>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.STACS.2015.i

ISBN 978-3-939897-78-1

ISSN 1868-8969

<http://www.dagstuhl.de/lipics>

LIPICs – Leibniz International Proceedings in Informatics

LIPICs is a series of high-quality conference proceedings across all fields in informatics. LIPICs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

Editorial Board

- Susanne Albers (TU München)
- Chris Hankin (Imperial College London)
- Deepak Kapur (University of New Mexico)
- Michael Mitzenmacher (Harvard University)
- Madhavan Mukund (Chennai Mathematical Institute)
- Catuscia Palamidessi (INRIA)
- Wolfgang Thomas (RWTH Aachen)
- Pascal Weil (*Chair*, CNRS and University Bordeaux)
- Reinhard Wilhelm (Saarland University)

ISSN 1868-8969

<http://www.dagstuhl.de/lipics>

■ Foreword

The Symposium on Theoretical Aspects of Computer Science (STACS) conference series is an international forum for original research on theoretical aspects of computer science. Typical areas are (cited from the call for papers for this year's conference):

algorithms and data structures, including: parallel, distributed, approximation, and randomized algorithms, computational geometry, cryptography, algorithmic learning theory, algorithmic game theory, analysis of algorithms; automata and formal languages; computational complexity, parameterized complexity, randomness in computation; logic in computer science, including: semantics, specification and verification, rewriting and deduction; current challenges, for example: natural computing, quantum computing, mobile and net computing.

STACS is held alternately in France and in Germany. This year's conference (taking place March 4–7 in Garching near Munich) is the 32nd in the series. Previous meetings took place in Paris (1984), Saarbrücken (1985), Orsay (1986), Passau (1987), Bordeaux (1988), Paderborn (1989), Rouen (1990), Hamburg (1991), Cachan (1992), Würzburg (1993), Caen (1994), München (1995), Grenoble (1996), Lübeck (1997), Paris (1998), Trier (1999), Lille (2000), Dresden (2001), Antibes (2002), Berlin (2003), Montpellier (2004), Stuttgart (2005), Marseille (2006), Aachen (2007), Bordeaux (2008), Freiburg (2009), Nancy (2010), Dortmund (2011), Paris (2012), Kiel (2013), and Lyon (2014).

The interest in STACS has remained at a high level over the past years. The STACS 2015 call for papers led to 235 submissions with authors from 39 countries. Each paper was assigned to three program committee members who, at their discretion, asked external reviewers for reports. The committee selected 55 papers during a three-week electronic meeting held in November/December. For the first time within the STACS conference series, there was also a rebuttal period during which authors could submit remarks to the PC concerning the reviews of their papers. As co-chairs of the program committee, we would like to sincerely thank all its members and the many external referees for their valuable work. In particular, there were intense and interesting discussions. The overall very high quality of the submissions made the selection a difficult task.

This year, the conference includes two tutorials. We would like to express our thanks to the speakers Felix Brandt (TUM) and Paul Goldberg (Oxford) for these tutorials, as well as to the invited speakers, Sanjeev Arora (Princeton), Manuel Bodirsky (Dresden), and Peter Sanders (Karlsruhe). Special thanks also go to Andrei Voronkov for his EasyChair software (<http://www.easychair.org>). Moreover, we would like to warmly thank Christine Lissner and Ernst Bayer for continuous help throughout the conference organization.

We would also like to thank Marc Herbstritt and Michael Wagner from the Dagstuhl/LIPIcs team for assisting us in the publication process and the final production of the proceedings. These proceedings contain extended abstracts of the accepted contributions and abstracts of the invited talks and the tutorials. The authors retain their rights and make their work available under a Creative Commons license. The proceedings are published electronically by Schloss Dagstuhl – Leibniz-Center for Informatics within their LIPIcs series.

STACS 2015 has received funds and help from the Deutsche Forschungsgemeinschaft (DFG), for which we are very grateful.

Munich and Orléans, February 2015

Ernst W. Mayr and Nicolas Ollinger



■ Conference Organization

Program Committee

Andris Ambainis	FPM, U Riga
Hagit Attiya	CS, Technion, Haifa
Johannes Blömer	CS, U Paderborn
Mikołaj Bojańczyk	II, U Warsaw
Tomas Brazdil	Masaryk U, Brno
Niv Buchbinder	SOR, Tel Aviv U
Anuj Dawar	CL, U Cambridge
Adrian Dumitrescu	CS, U Wisconsin-Milwaukee
Matthias Englert	DIMAP/DCS, U Warwick
Funda Ergun	SCS SFU and SoIC Indiana U
Fedor Fomin	IN, U Bergen
Tobias Friedrich	FMI, FSU Jena
Christian Glaßer	II, U Würzburg
Etienne Grandjean	GREYC, Caen
Tomasz Jurdzinski	U Wroclaw
Manfred Kufleitner	FMI, U Stuttgart
Jerome Leroux	CNRS, LaBRI, Bordeaux
Ernst W. Mayr	TUM, München (co-chair)
Peter Bro Miltersen	CS, U Aarhus
Nicolas Ollinger	LIFO, Orléans (co-chair)
Sylvain Perifel	LIAFA, U Paris Diderot
Jayalal Sarma	IIT, Madras
Nicolas Schabanel	CNRS, LIAFA, Paris 7
Lutz Schröder	FAU Erlangen-Nürnberg
Dimitrios M. Thilikos	CNRS, LIRMM and U Athens
Gerhard Woeginger	TUE, Eindhoven

Local Organization Committee

Ernst W. Mayr, TUM, München (chair)
Christine Lissner, TUM, München
Ernst Bayer, TUM, München



■ External Reviewers

Sebastian Abshoff
Oswin Aichholzer
Helmut Alt
Vikraman Arvind
James Aspnes
Mohamed Faouzi Atig
Erfan Sadeqi Azer
Golnaz Badkobeh
Christel Baier
Valeriy Balabanov
János Balogh
Evangelos Bampas
Hideo Bannai
Régis Barbanchon
Rafael Barbosa
Leonid Barenboim
Laurent Bartholdi
Surender Baswana
Tugkan Batu
Florent Becker
Petra Berenbrink
Attila Bernáth
Valérie Berthé
Dietmar Berwanger
Randeep Bhatia
Binay Bhattacharya
Arnab Bhattacharyya
Marcin Bieńkowski
Olivier Bodini
Manuel Bodirsky
Andrej Bogdanov
Bernard Boigelot
Udi Boker
Guillaume Bonfante
Paul Bonsma
Adam Bould
Andreas Brandstadt
Simina Brânzei
Sascha Brauer
Michael Bremner
Karl Bringmann
Joshua Brody
Véronique Bruyère
Kevin Buchin
Kathrin Bujna
Jannis Bulian
Benjamin Burton
Jarosław Byrka
Daniel Cabarcas
Gruia Calinescu
Olivier Carton
Parinya Chalermsook
Jérémy Chalopin
Witold Charatonik
Krishnendu Chatterjee
Arkadev Chattopadhyay
Dimitris Chatzidimitriou
Kaustuv Chaudhuri
Ke Chen
Shahar Chen
Christine Cheng
Otfried Cheong
Mahdi Cheraghchi
Ferdinando Cicalese
Francisco Claude
Lorenzo Clemente
Ilan Cohen
Dinu Coltuc
Nadia Creignou
Maxime Crochemore
Marek Cygan
Artur Czumaj
Jurek Czyżowicz
Shantanu Das
Samir Datta
Mark de Berg
Bart de Keijzer
Nicolas de Rugy-Altherre
Ronald de Wolf
Emmanuel Delucchi
Dariusz Dereniowski
Krishnamorthy Dinesh
Michael Dinitz
Itai Dinur
Shahar Dobzinski
Laurent Doyen
Anne Driemel
Léo Ducas
Fabien Durand
Christoph Dürr
Zdeňek Dvořák
Stefan Dziembowski
Rüdiger Ehlers
Kord Eickmeyer

32nd International Symposium on Theoretical Aspects of Computer Science (STACS'15).
Editors: Ernst W. Mayr, Nicolas Ollinger



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Martina Eikel
Yuval Emek
Alina Ene
David Eppstein
Leah Epstein
Bruno Escoffier
William Evans
Yuri Faenza
John Fearnley
Moran Feldman
Nathanaël Fijalkow
Aris Filos-Ratsikas
Samuel Fiorini
Lila Fontes
Vojtěch Forejt
Mathew Francis
Robert Fraser
Anna Frid
Oliver Friedmann
Alan Frieze
Travis Gagie
Anahi Gajardo
Jakub Gajarský
Iftah Gamzu
Robert Ganian
Pierre Ganty
Leszek Gąsieniec
Serge Gaspers
Philippe Gaucher
Paweł Gawrychowski
Guido Gherardi
Anirban Ghosh
Panos Giannopoulos
Archontia Giannopoulou
Hugo Gimbert
Tomasz Gogacz
Stefan Göller
Petr Golovach
Daniel Gonçalves
David Gosset
Dominique Gouyou-Beauchamps
Sathish Govindarajan
Vineet Goyal
Serge Grigorieff
Joshua Grochow
Peter Günther
Heng Guo
Rishi Gupta
Shalmoli Gupta
Christoph Haase
Magnús M. Halldórsson
Sean Hallgren
Michał Hańćkowiak
Kristoffer Arnsfelt Hansen
Thomas Dueholm Hansen
Sariel Har-Peled
Thomas Hayes
Pinar Heggernes
Lauri Hella
Benjamin Hellouin de Ménibus
Monika Henzinger
Frédéric Herbretreau
Ulrich Hertrampf
Cameron Donnay Hill
Jeff Hirst
Petr Hliněný
Martin Hofer
Seok-Hee Hong
Florian Horn
Pavel Hrubes
Andreas Hülsing
Paul Hunter
John Iacono
Rasmus Ibsen-Jensen
Sungjin Im
Radu Iosif
Rani Izsak
Bart M. P. Jansen
Emmanuel Jeandel
Stacey Jeffery
Anders Jensen
Artur Jeź
Łukasz Jeź
Ajay Joneja
Mark Jones
Antoine Joux
Jakob Juhnke
Tomasz Jurkiewicz
Mark Kaminski
Marcin Kamiński
Mamadou Moustapha Kanté
Michael Kapralov
Juhani Karhumäki
Shiva Kasiviswanathan
Jonathan Kausch
Akitoshi Kawamura
Edon Kelmendi
Iordanis Kerenidis
Eun Jung Kim
Shelby Kimmel

Valerie King
Hartmut Klauck
Bartek Klin
Peter Kling
Hirotada Kobayashi
Yusuke Kobayashi
Johannes Koebler
Pascal Koiran
Stavros Kolliopoulos
Balagopal Komarath
Eryk Kopczyński
Swastik Kopparty
Sajin Koroth
Guy Kortsarz
Adrian Kosowski
Robin Kothari
Lukasz Kowalik
Daniel Kral
Dieter Kratsch
Jan Krčál
Jan Křetínský
Stephan Kreutzer
Sebastian Krinninger
R. Krithika
Anton Krohmer
Antonín Kučera
Ravi Kumar
Piyush Kurur
Eyal Kushilevitz
Martin Kutrib
Roman Kuznets
Jakub Łacki
Peter Lammich
Michael Lampis
Kasper Green Larsen
Yanfang Le
Bastien Le Gloannec
Thierry Lecroq
Axel Legay
Daniel Lemire
Hendrik W. Lenstra
Anthony Leverrier
Asaf Levin
Nutan Limaye
Vincent Limouzy
Gennadij Liske
Maciej Liśkiewicz
Xiao Liu
Daniel Lokshtanov
Florian Lonsing
Krzysztof Loryś
Michael Ludwig
Frédéric Magniez
Meena Mahajan
Johann Makowsky
Ritankar Mandal
Spyridon Maniatis
Rajsekar Manokaran
Sabrina Mantaci
Bodo Manthey
Alberto Marchetti-Spaccamela
Jerzy Marcinkowski
Russell Martin
Dániel Marx
Tomas Masopust
Kevin Matulef
Elvira Mayordomo
Arne Meier
Daniel Meister
Stefan Mengel
George Mertzios
Pierre-Étienne Meunier
Friedhelm Meyer auf der Heide
Othon Michail
Henryk Michalewski
Matúš Mihalák
Samuel Mimram
Matteo Mio
Neeldhara Misra
Tal Mizrahi
Matthias Mnich
Morteza Monemizadeh
Walter Morris
Benjamin Moseley
Amer Mouawad
Jean-Yves Moyen
Yannis Moysoglou
Marcin Mucha
Partha Mukhopadhyay
Wolfgang Mulzer
Daniel Nagaj
Viswanath Nagarajan
Alberto Naibo
N. S. Narayanaswamy
Meghana Nasre
Gonzalo Navarro
Alantha Newman
Calvin Newport
Phong Nguyen
Patrick K. Nicholson

Nicolas Nisse	Oded Regev
Petr Novotný	Vojtěch Řehák
Zeev Nutov	Eric Remila
Jan Obdržálek	Pierre-Alain Reynier
Alexander Okhotin	Gaétan Richard
Alberto Ordóñez	David Richerby
Sebastian Ordyniak	Liam Roditty
Sigal Oren	Martin Roetteler
Rotem Oshman	Heiko Röglin
Yota Otachi	Adi Rosen
Youssef Oualhadj	Günter Rote
Kenta Ozeki	Sasanka Roy
Katarzyna Paluch	Alan Roytman
Konstantinos Panagiotou	Michał Rózański
Fahad Panolan	Philipp Rümmer
Evanthia Papadopoulou	Ignaz Rutter
Charles Paperman	Aleksi Saarela
Mike Paterson	Benjamin Sach
Boaz Patt-Shamir	Sigve Hortemo Sæther
Christophe Paul	Ville Salo
Daniel Paulusma	Arnaud Sangnier
Arno Pauly	Piotr Sankowski
Emmanuel Paviot-Adet	Kanthi Sarpatwar
Ami Paz	Ignasi Sau
Lehilton L. C. Pedrosa	Nitin Saurabh
Andrzej Pelc	Saket Saurabh
Pablo Pérez-Lantero	Guido Schaefer
Dominique Perrin	Marcus Schaefer
Leonid Petrov	Patrick Scharpfenecker
Giovanni Pighizzini	Christian Scheffer
Michał Pilipczuk	Christian Scheideler
Chris Pinkau	Sven Schewe
Marek Piotrów	Maximilian Schlund
Thomas Place	Markus L. Schmid
Sebastian Pokutta	Dominique Schmitt
Valentin Polishchuk	Sylvain Schmitz
Natacha Portier	Henning Schnoor
Cristian Prisacariu	Roy Schwartz
Ariel Procaccia	Luc Segoufin
Kirk Pruhs	Géraud Sénizergues
Simon Puglisi	Olivier Serre
Mikaël Rabie	Jiří Sgall
M. S. Ramanujan	Paul Shafer
Narad Rampersad	Chintan Shah
Ramyaa Ramyaa	Mordechai Shalom
Mickael Randour	Asaf Shapira
B. V. Raghavendra Rao	John Shareshi
Baharak Rastegari	Alexander Shen
Saurabh Ray	Arseny Shur
Jean-Florent Raymond	Anastasios Sidiropoulos

Laurent Simon
Rakesh Sinha
Naveen Sivadasan
Alexander Skopalik
Shakhar Smorodinsky
Christian Sohler
Shay Solomon
Eric Sopena
Troels Bjerre Sørensen
Jiří Srba
Srikanth Srinivasan
Grzegorz Stachowiak
Gawiejnowicz Stanisław
Daniel Stefankovic
Eckhard Steffen
Damien Stehlé
Darren Strash
Howard Straubing
Hsin-Hao Su
Scott Summers
Grégoire Sutre
Stefan Szeider
Luis Tabera
Avishay Tal
Navid Talebanfard
Tami Tamir
Pingzhong Tang
Till Tantau
Gabor Tardos
Hanjo Täubig
Sébastien Tavenas
Balder ten Cate
Lidia Tendera
Véronique Terrier
Raghunath Tewari
Abhradeep Thakurta
Johan Thapper
Guillaume Theyssier
Erez Timnat
Alexander Tiskin
Stefan Toman
Jacobo Torán
Eric Torng
Dave Touchette
Craig Tovey
Henry Towsner
Ashutosh Trivedi
Torsten Ueckerdt
Seeun Umboh
Pierre Valarcher
Leo van Iersel
Erik Jan van Leeuwen
Dieter van Melkebeek
Rob van Stee
Anke van Zuylen
Adi Vardi
Shai Vardi
Sergei Vassilvitskii
Sander Verdonschot
José Verschae
Aravindan Vijayaraghavan
Tobias Walter
Haitao Wang
Justin Ward
Xiangzhi Wei
Jeremias Weihmann
Armin Weiss
Matthias Westermann
James Wilson
Maximilian Witek
Philipp Woelfel
Alexander Wolff
Damien Woods
James Worrell
Thomas Worsch
Zhilin Wu
Christian Wulff-Nilsen
Tim Wylie
Mingyu Xiao
G Xu
Li Yan
Yuichi Yoshida
Victor Zamaraev
Meirav Zehavi
Marc Zeitoun
Jie Zhang
Yuan Zhou

■ Contents

Invited talks

Overcoming Intractability in Unsupervised Learning <i>Sanjeev Arora</i>	1
The Complexity of Constraint Satisfaction Problems <i>Manuel Bodirsky</i>	2
Parallel Algorithms Reconsidered <i>Peter Sanders</i>	10

Tutorials

Computational Social Choice <i>Felix Brandt</i>	19
Algorithmic Game Theory <i>Paul Goldberg</i>	20

Regular contributions

The Minimum Oracle Circuit Size Problem <i>Eric Allender, Dhiraj Holden, and Valentine Kabanets</i>	21
Graph Searching Games and Width Measures for Directed Graphs <i>Saeed Akhoondian Amiri, Lukasz Kaiser, Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz</i>	34
Subset Sum in the Absence of Concentration <i>Per Austrin, Petteri Kaski, Mikko Koivisto, and Jesper Nederlof</i>	48
On Sharing, Memoization, and Polynomial Time <i>Martin Avanzini and Ugo Dal Lago</i>	62
Proof Complexity of Resolution-based QBF Calculi <i>Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota</i>	76
Welfare Maximization with Friends-of-Friends Network Externalities <i>Sayan Bhattacharya, Wolfgang Dvořák, Monika Henzinger, and Martin Starnberger</i>	90
Markov Decision Processes and Stochastic Games with Total Effective Payoff <i>Endre Boros, Khaled Elbassioni, Vladimir Gurvich, and Kazuhisa Makino</i>	103
Advice Complexity for a Class of Online Problems <i>Joan Boyar, Lene M. Favrholdt, Christian Kudahl, and Jesper W. Mikkelsen</i>	116
Las Vegas Computability and Algorithmic Randomness <i>Vasco Brattka, Guido Gherardi, and Rupert Hözl</i>	130
Understanding Model Counting for β -acyclic CNF-formulas <i>Johann Brault-Baron, Florent Capelli, and Stefan Mengel</i>	143

32nd International Symposium on Theoretical Aspects of Computer Science (STACS'15).

Editors: Ernst W. Mayr, Nicolas Ollinger



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Parameterized Complexity Dichotomy for Steiner Multicut <i>Karl Bringmann, Danny Hermelin, Matthias Mnich, and Erik Jan van Leeuwen</i> ..	157
Solving Totally Unimodular LPs with the Shadow Vertex Algorithm <i>Tobias Brunsch, Anna Großwendt, and Heiko Röglin</i>	171
Improved Local Search for Geometric Hitting Set <i>Norbert Bus, Shashwat Garg, Nabil H. Mustafa, and Saurabh Ray</i>	184
Arc Diagrams, Flip Distances, and Hamiltonian Triangulations <i>Jean Cardinal, Michael Hoffmann, Vincent Kusters, Csaba D. Tóth, and Manuel Wettstein</i>	197
Tractable Probabilistic μ -Calculus That Expresses Probabilistic Temporal Logics <i>Pablo Castro, Cecilia Kilmurray, and Nir Piterman</i>	211
Tribes Is Hard in the Message Passing Model <i>Arkadev Chattopadhyay and Sagnik Mukhopadhyay</i>	224
Network Design Problems with Bounded Distances via Shallow-Light Steiner Trees <i>Markus Chimani and Joachim Spoerhase</i>	238
Combinatorial Expressions and Lower Bounds <i>Thomas Colcombet and Amaldev Manuel</i>	249
Construction of μ -Limit Sets of Two-dimensional Cellular Automata <i>Martin Delacourt and Benjamin Hellouin de Menibus</i>	262
Derandomized Graph Product Results Using the Low Degree Long Code <i>Irit Dinur, Prahladh Harsha, Srikanth Srinivasan, and Girish Varma</i>	275
Space-efficient Basic Graph Algorithms <i>Amr Elmasry, Torben Hagerup, and Frank Kammer</i>	288
Pattern Matching with Variables: Fast Algorithms and New Hardness Results <i>Henning Fernau, Florin Manea, Robert Mercas, and Markus L. Schmid</i>	302
Approximating the Generalized Terminal Backup Problem via Half-integral Multiflow Relaxation <i>Takuro Fukunaga</i>	316
On Matrix Powering in Low Dimensions <i>Esther Galby, Joël Ouaknine, and James Worrell</i>	329
The Complexity of Recognizing Unique Sink Orientations <i>Bernd Gärtner and Antonis Thomas</i>	341
New Geometric Representations and Domination Problems on Tolerance and Multitolerance Graphs <i>Archontia C. Giannopoulou and George B. Mertzios</i>	354
Comparing 1D and 2D Real Time on Cellular Automata <i>Anaël Grandjean and Victor Poupet</i>	367
Tropical Effective Primary and Dual Nullstellensätze <i>Dima Grigoriev and Vladimir V. Podolskii</i>	379
Upper Tail Estimates with Combinatorial Proofs <i>Jan Hązła and Thomas Holenstein</i>	392

Minimum Cost Flows in Graphs with Unit Capacities <i>Andrew V. Goldberg, Haim Kaplan, Sagi Hed, and Robert E. Tarjan</i>	406
Inductive Inference and Reverse Mathematics <i>Rupert Hölzl, Sanjay Jain, and Frank Stephan</i>	420
Dynamic Planar Embeddings of Dynamic Graphs <i>Jacob Holm and Eva Rotenberg</i>	434
On the Information Carried by Programs about the Objects They Compute <i>Mathieu Hoyrup and Cristóbal Rojas</i>	447
Communication Complexity of Approximate Matching in Distributed Graphs <i>Zengfeng Huang, Božidar Radunović, Milan Vojnović, and Qin Zhang</i>	460
Stochastic Scheduling of Heavy-tailed Jobs <i>Sungjin Im, Benjamin Moseley, and Kirk Pruhs</i>	474
On Finding the Adams Consensus Tree <i>Jesper Jansson, Zhaoxian Li, and Wing-Kin Sung</i>	487
Flip Distance Is in FPT Time $\mathcal{O}(n + k \cdot c^k)$ <i>Iyad Kanj and Ge Xia</i>	500
New Pairwise Spanners <i>Telikepalli Kavitha</i>	513
Multi- k -ic Depth Three Circuit Lower Bound <i>Neeraj Kayal and Chandan Saha</i>	527
Automorphism Groups of Geometrically Represented Graphs <i>Pavel Klavík and Peter Zeman</i>	540
Correlation Clustering and Two-edge-connected Augmentation for Planar Graphs <i>Philip N. Klein, Claire Mathieu[‡], and Hang Zhou[‡]</i>	554
Extended Formulation Lower Bounds via Hypergraph Coloring? <i>Stavros G. Kolliopoulos and Yannis Moysoglou</i>	568
Lempel-Ziv Factorization May Be Harder Than Computing All Runs <i>Dmitry Kosolobov</i>	582
Visibly Counter Languages and Constant Depth Circuits <i>Andreas Krebs, Klaus-Jörn Lange, and Michael Ludwig</i>	594
Optimal Decremental Connectivity in Planar Graphs <i>Jakub Łącki and Piotr Sankowski</i>	608
Testing Small Set Expansion in General Graphs <i>Angsheng Li and Pan Peng</i>	622
Paid Exchanges are Worth the Price <i>Alejandro López-Ortiz, Marc P. Renault, and Adi Rosén</i>	636
Undecidability in Binary Tag Systems and the Post Correspondence Problem for Five Pairs of Words <i>Turlough Neary</i>	649

Separation and the Successor Relation <i>Thomas Place and Marc Zeitoun</i>	662
Computing 2-Walks in Polynomial Time <i>Andreas Schmid and Jens M. Schmidt</i>	676
Towards an Isomorphism Dichotomy for Hereditary Graph Classes <i>Pascal Schweitzer</i>	689
Existential Second-order Logic over Graphs: A Complete Complexity-theoretic Classification <i>Till Tantau</i>	703
The Returning Secretary <i>Shai Vardi</i>	716
Homomorphism Reconfiguration via Homotopy <i>Marcin Wrochna</i>	730
Computing Downward Closures for Stacked Counter Automata <i>Georg Zetsche</i>	743

Overcoming Intractability in Unsupervised Learning

Sanjeev Arora

Computer Science Department, Princeton University
arora@princeton.edu

Abstract

Unsupervised learning – i.e., learning with unlabeled data - is increasingly important given today's data deluge. Most natural problems in this domain – e.g. for models such as mixture models, HMMs, graphical models, topic models and sparse coding/dictionary learning, deep learning – are NP-hard. Therefore researchers in practice use either heuristics or convex relaxations with no concrete approximation bounds. Several nonconvex heuristics work well in practice, which is also a mystery.

The talk will describe a sequence of recent results whereby rigorous approaches leading to polynomial running time are possible for several problems in unsupervised learning. The proof of polynomial running time usually relies upon nondegeneracy assumptions on the data and the model parameters, and often also on stochastic properties of the data (average-case analysis). We describe results for topic models, sparse coding, and deep learning. Some of these new algorithms are very efficient and practical – e.g. for topic modeling.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, I.2 Artificial Intelligence

Keywords and phrases machine learning, unsupervised learning, intractability, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.1

Category Invited Talk



© Sanjeev Arora;

licensed under Creative Commons License CC-BY

32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).

Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 1–1

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



SYMPOSIUM
ON THEORETICAL
ASPECTS
OF COMPUTER
SCIENCE

The Complexity of Constraint Satisfaction Problems*

Manuel Bodirsky

Institut für Algebra, Technische Universität Dresden, Germany
Manuel.Bodirsky@tu-dresden.de

Abstract

The *tractability conjecture* for constraint satisfaction problems (CSPs) describes the constraint languages over a finite domain whose CSP can be solved in polynomial-time. The precise formulation of the conjecture uses basic notions from universal algebra. In this talk, we give a short introduction to the universal-algebraic approach to the study of the complexity of CSPs. Finally, we discuss attempts to generalise the tractability conjecture to large classes of constraint languages over infinite domains, in particular for constraint languages that arise in qualitative temporal and spatial reasoning.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases constraint satisfaction, universal algebra, model theory, clones, temporal and spatial reasoning

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.2

Category Invited Talk

1 The Constraint Satisfaction Problem

Constraint satisfaction problems are computational problems that can be formalised in several equivalent ways. A mathematically convenient way is to view CSPs as structural homomorphism problems, as follows. Fix a structure Γ with a finite relational signature τ . The domain of Γ need not be finite for the following computational problem to be well-defined.

► **Definition 1** ($\text{CSP}(\Gamma)$). The constraint satisfaction problem for Γ , denoted by $\text{CSP}(\Gamma)$, is the computational problem to decide for a given *finite* τ -structure A whether there exists a homomorphism to Γ .

The fixed structure Γ is often referred to as the *constraint language* of the constraint satisfaction problem, since we choose from the relations in Γ to formulate our constraints in the input structure A . We give some concrete examples of CSPs.

1. Graph n -colorability can be formulated as $\text{CSP}(K_n)$ where K_n is the complete loopless graph on n vertices.
2. The question whether a given finite digraph is acyclic, i.e., does not contain a directed cycle, can be formulated as $\text{CSP}(\mathbb{Q}; <)$.
3. The question whether a given directed graph has a vertex bipartition such that both parts are acyclic can be formulated as $\text{CSP}(\mathbb{N}; E)$ where

$$E := \{(a, b) \in \mathbb{N}^2 \mid a < b \text{ or } (a - b) \text{ is odd}\}.$$

* The author has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013 Grant Agreement no. 257039).

4. $\text{CSP}(\mathbb{R}; \leq, A, O)$ for $A := \{(a, b, c) \in \mathbb{R}^3 \mid a + b = c\}$ and $O := \{1\}$ is essentially the feasibility problem for linear programs (see [5]).

The list can be prolonged easily, and contains a variety of problems that appeared in the literature throughout theoretical computer science.

There is a great amount of work about the computational complexity of $\text{CSP}(\Gamma)$ when Γ is a finite structure (i.e., has a finite domain), stimulated by the following dichotomy conjecture.

► **Conjecture 1** (Feder and Vardi [18]). *Let Γ be a finite structure with a finite relational signature. Then $\text{CSP}(\Gamma)$ is in P or NP-complete.*

2 The Universal-Algebraic Approach

The central notion for the universal algebraic approach is the notion of a *polymorphism* of a constraint language Γ . A polymorphism of Γ is a homomorphism h from finite powers of Γ into Γ . In other words, when h has arity k , then we require for all relations R of Γ and $(a_1^1, \dots, a_n^1) \in R, \dots, (a_1^k, \dots, a_n^k) \in R$ that $(h(a_1^1, \dots, a_1^k), \dots, h(a_n^1, \dots, a_n^k)) \in R$. Unary polymorphisms are also known as *endomorphisms*. Thus, polymorphisms generalise endomorphisms, and endomorphisms generalise automorphisms of Γ . We write $\text{Pol}(\Gamma)$ for the set of all polymorphisms of Γ , and $\text{Aut}(\Gamma)$ for the set of all polymorphisms of Γ .

The following result for structures with a finite domain, which relies on a fundamental theorem in universal algebra [19, 16], hints at the relevance of polymorphisms for CSPs.

► **Theorem 2** ([23]). *Let Γ_1 and Γ_2 be finite structures with the same domain and finite relational signatures such that $\text{Pol}(\Gamma_1) \subseteq \text{Pol}(\Gamma_2)$. Then there is a deterministic linear-time many-one reduction from $\text{CSP}(\Gamma_2)$ to $\text{CSP}(\Gamma_1)$.*

Theorem 2 has an important advancement, Theorem 3 below, which is particularly important when we want to reduce between CSPs where the constraint languages have different domains. Let us first mention that the set $\text{Pol}(\Gamma)$ is a *function clone*. A function clone is a set \mathcal{S} of functions of finite arity that

- is closed under *composition*: for k -ary $g \in \mathcal{S}$ and l -ary $f_1, \dots, f_k \in \mathcal{S}$ the l -ary function $g(f_1, \dots, f_k)$ given by $(x_1, \dots, x_l) \mapsto g(f_1(x_1, \dots, x_l), \dots, f_k(x_1, \dots, x_l))$ is also in \mathcal{S} , and
- contains the *projections* π_i^k given by $(x_1, \dots, x_k) \mapsto x_i$.

A map $\xi: \text{Pol}(\Gamma_1) \rightarrow \text{Pol}(\Gamma_2)$ is called a *clone homomorphism* if for all $g, f_1, \dots, f_k \in \text{Pol}(\Gamma_1)$

$$\xi(g(f_1, \dots, f_k)) = \xi(g)(\xi(f_1), \dots, \xi(f_k))$$

and $\xi(\pi_i^k) = \pi_i^k$ for all $1 \leq i \leq k$. A *clone isomorphism* is a bijective clone homomorphism.

► **Theorem 3.** *Suppose that Γ_1 and Γ_2 are finite structures with finite relational signature such that there exists a clone isomorphism between $\text{Pol}(\Gamma_1)$ to $\text{Pol}(\Gamma_2)$. Then $\text{CSP}(\Gamma_1)$ and $\text{CSP}(\Gamma_2)$ are equivalent under deterministic linear-time many-one reductions.*

3 The Finite Domain Tractability Conjecture

Theorem 3 from the previous section tells us that the computational complexity of $\text{CSP}(\Gamma)$ is coded into the equations that hold on the polymorphisms. We even have a candidate equation that might characterise the CSPs in P.

► **Theorem 4** ([17, 28, 21]). *Suppose that Γ is a finite structure. Then Γ has a Taylor¹ polymorphism f , that is, when f has arity n then it satisfies for every $i \leq n$ an equation of the form*

$$\forall x, y. f(x_1, \dots, x_n) = f(y_1, \dots, y_1),$$

where $x_1, \dots, x_n, y_1, \dots, y_n \in \{x, y\}$ and $x_i \neq y_i$, or there is a structure Γ' obtained from Γ by dropping all but finitely many relations such that $\text{CSP}(\Gamma')$ is NP-complete.

The condition given in Theorem 4 has been improved recently: the existence of a Taylor polymorphism is equivalent to the existence of an operation that satisfies an equation that is much easier to grasp.

► **Theorem 5** ([1]). *A finite structure Γ has a Taylor polymorphism if and only if it has a cyclic polymorphism f , that is, f has arity $n \geq 2$ and satisfies*

$$\forall x_1, \dots, x_n. f(x_1, \dots, x_n) = f(x_2, \dots, x_n, x_1).$$

The following conjecture has been made in different form by Bulatov, Jeavons, and Krokhin [17]; the formulation given below is equivalent by well-known facts. The conjecture complements Theorem 4, and its truth would settle the dichotomy conjecture.

► **Conjecture 2** (Tractability Conjecture). *Let Γ be a finite structure with finite relational signature and a Taylor (or, equivalently, cyclic) polymorphism. Then $\text{CSP}(\Gamma)$ is in P.*

4 Infinite Domains

The universal-algebraic approach can be generalised to constraint languages Γ over infinite domains. This generalisation is most straightforward when the automorphism group of Γ is large, in the following sense.

► **Definition 6.** A permutation group G on a set X is called *oligomorphic* if the componentwise action of G on X^n has only finitely many orbits, for all $n \in \mathbb{N}$.

An example of an oligomorphic permutation group is the automorphism group of $(\mathbb{Q}; <)$. Countable structures Γ with an oligomorphic permutation group are well-known to model-theorists: by a theorem independently due to Ryll-Nardzewski, Engeler, and Svenonius (see, e.g., [22]), these are precisely the countable structures that are ω -categorical, that is, Γ has the property that all countable models of the first-order theory of Γ are isomorphic to Γ .

A versatile method to construct ω -categorical structures is via Fraïssé-limits, and taking reducts, which we briefly recall here. We need the standard notion of homogeneity (sometimes called ultrahomogeneity) from model theory. A structure Γ is called *homogeneous* if all isomorphisms between finite substructures can be extended to automorphisms of Γ . Homogeneous structures with finite relational signature are ω -categorical [22]. Homogeneous structures are uniquely given by their *age*, which is the class of finite structures that embed into them. The age of a homogeneous structure must have the amalgamation property (we again refer to [22]), and every amalgamation class \mathcal{C} gives rise to a homogeneous structure of age \mathcal{C} . The fundamental model theory of homogeneous structures goes back to Fraïssé, and hence the unique homogeneous structure for a given amalgamation class is called the *Fraïssé-limit* of this class.

¹ Note that, contrary to what can often be found in the literature, in our definition of Taylor operations, we do not insist on idempotency of f .

A *reduct* of a structure Δ is a structure Γ on the same domain such that all relations of Γ are first-order definable (without parameters) in Δ . For example, the structure $(\mathbb{Q}; \text{Betw})$ where $\text{Betw} := \{(x, y, z) \mid x < y < z \vee z < y < x\}$ (the so-called *Betweenness relation*) is a reduct of $(\mathbb{Q}; <)$. Reducts of homogeneous structures need not be homogeneous, but reducts of ω -categorical structures remain ω -categorical.

When Γ is ω -categorical, then the complexity of Γ is still coded into the polymorphisms.

► **Theorem 7** ([8]). *Let Γ_1 and Γ_2 be ω -categorical structures with the same domain and finite relational signatures such that $\text{Pol}(\Gamma_1) = \text{Pol}(\Gamma_2)$. Then Γ_1 and Γ_2 are equivalent under deterministic linear-time many-one reductions.*

An example of a permutation group which is *not* oligomorphic is the automorphism group of the structure $(\mathbb{N}; E)$ discussed in the introduction: it has infinitely many orbits in its componentwise action on \mathbb{N}^2 . However, in this case it is easy to come up with a structure that has precisely the same CSP, but whose automorphism group *is* oligomorphic: let Q_1, Q_2 be a partition of \mathbb{Q} such that both Q_1 and Q_2 are dense in \mathbb{Q} , and consider the structure $(\mathbb{Q}; E')$ where

$$E' := \{(a, b) \in \mathbb{Q}^2 \mid a < b \text{ or } a \in Q_1 \Leftrightarrow b \in Q_2\} .$$

This is a frequent phenomenon: many computational problems in temporal and spatial reasoning can be formulated as CSPs, but often some extra care is needed to show that they can be formulated with ω -categorical constraint languages. A necessary and sufficient Myhill-Nerode-type condition that characterises the CSPs that can be formulated with an ω -categorical constraint language can be found in [3]. An example of a structure that does not satisfy the mentioned Myhill-Nerode-type condition of Example 4, in the introduction. Hence, $\text{CSP}(\mathbb{R}; A, O)$ (which is essentially the feasibility problem for linear programs) cannot be formulated as $\text{CSP}(\Gamma)$ with an ω -categorical constraint language.

We do not know whether Theorem 3 remains valid for ω -categorical structures Γ , that is, whether the isomorphism type of the polymorphism clone of Γ determines the complexity of $\text{CSP}(\Gamma)$. However, the theorem can be rescued by a slight modification.

► **Theorem 8** ([12]). *Suppose that Γ_1 and Γ_2 are ω -categorical structures with finite relational signature such that there exists a clone isomorphism between $\text{Pol}(\Gamma_1)$ and $\text{Pol}(\Gamma_2)$ which is also a homeomorphism. Then $\text{CSP}(\Gamma_1)$ and $\text{CSP}(\Gamma_2)$ are equivalent under deterministic linear-time many-one reductions.*

The homeomorphic requirement in Theorem 8 is with respect to the *topology of pointwise convergence* on the space of all functions of finite arity, which is defined as follows. For elements a, b_1, \dots, b_k of the domain D , define $F_{a, b_1, \dots, b_k} := \{f \mid f(b_1, \dots, b_k) = a\}$. Then the topology of pointwise convergence is the smallest topology where the open sets include $\{F_{a, b_1, \dots, b_k} \mid k \in \mathbb{N}, a, b_1, \dots, b_k \in D\}$. It is a basic fact that a clone \mathcal{C} is closed in this space, $\bar{\mathcal{C}} = \mathcal{C}$, if and only if it is the polymorphism clone of a structure.

5 A general tractability conjecture

Cyclic polymorphisms do not characterise the tractability of the CSP for ω -categorical structures: a simple counterexample is the structure $(\mathbb{N}; \neq, I_4)$ where I_4 is the quaternary relation defined as $I_4 := \{(a, b, c, d) \in \mathbb{N}^4 \mid a = b \Rightarrow c = d\}$. The automorphism group of this structure is the set of all permutations of \mathbb{N} , which is clearly oligomorphic. The polymorphisms of this structure are precisely all functions that are composed from injective functions and

projections. Hence, the clone does not contain cyclic operations. But $\text{CSP}(\mathbb{N}; \neq, I_4)$ is easily seen to be in P; see [6].

The structure $(\mathbb{N}; \neq, I_4)$ has polymorphisms that are almost as good as cyclic operations: every binary injective operation f will be a polymorphism, and we can always pick an injection i from $\mathbb{N} \rightarrow \mathbb{N}$ such that the following holds:

$$\forall x_1, x_2. f(x_1, x_2) = i(f(x_2, x_1)) .$$

We also have to describe an obstruction to general algorithmic results for the class of all ω -categorical structures. Henson [20] constructed uncountably many homogeneous directed graphs Γ , and all of these directed graphs have distinct CSPs. Since there are only countably many algorithms, there must be directed graphs in this class with an undecidable CSP. There are also CSPs of various intermediate complexities [2]. All of Henson's digraphs have a binary polymorphism f and endomorphisms e_1, e_2 satisfying

$$\forall x_1, x_2. e_1(f(x_1, x_2)) = e_2(f(x_2, x_1)) ,$$

that is, from a universal-algebraic perspective, they all 'look like easy CSPs', but they are not.

Henson's directed homogeneous graphs are based on forbidding *infinite* families of finite structures. On the other hand, the ω -categorical structures that appear 'in nature' (either in mathematics or to formulate computational problems as CSPs) can typically be described by forbidding only finitely many finite structures. More formally, we say that a homogeneous structure Γ is *finitely bounded* if there exists a finite set \mathcal{F} of finite structures such that the age of Γ is given as the class of all finite structures that do not embed any of the structures from \mathcal{F} . We now generalise the tractability conjecture by modifying the idea of Taylor polymorphisms so that it involves outside applications of endomorphisms, as follows.

► **Conjecture 3.** *Let Γ be the reduct of a finitely bounded homogeneous structure. If Γ has a polymorphism f of arity $n \geq 2$ such that for every $i \leq n$ there are endomorphisms e_1, e_2 and $x_1, \dots, x_n, y_1, \dots, y_n \in \{x, y\}$ with $x_i \neq y_i$ such that f satisfies*

$$e_1(f(x_1, \dots, x_n)) = e_2(f(y_1, \dots, y_n))$$

then $\text{CSP}(\Gamma)$ is in P. Otherwise, $\text{CSP}(\Gamma)$ is NP-complete.

The conjecture has been verified for several classes of ω -categorical structures:

- All reducts of $(\mathbb{Q}; <)$ in [7];
- All reducts of the Random graph (the Fraïssé-limit of the class of all finite graphs) in [10];
- All reducts of the homogeneous equivalence relation with infinitely many infinite classes in [15].

The strongest tool we have for attacking this conjecture will be introduced in the next section.

6 Ramsey Theory

The complexity classification results for ω -categorical structures mentioned in Section 5 rely on results from structural Ramsey theory. We say that a homogeneous structure Γ is *Ramsey* if for all finite substructures A and B of Γ and every colouring of the embeddings of A into Γ with finitely many colours, there exists an embedding $e: B \rightarrow \Gamma$ such that all embeddings of A into $e(B)$ have the same color. Examples of homogeneous Ramsey structures are

- $(\mathbb{Q}; <)$;
- the *ordered* Random graph, and other generically ordered Fraïssé-limits of so-called *free amalgamation classes* (examples are ordered versions of the Henson digraphs) [27];
- the *convexly ordered* homogeneous binary branching C-relation, and other tree-like structures [25, 26];
- the *lexicographically ordered* vector space over a finite field (see [24]);
- the *lexicographically ordered* atomless Boolean algebra (see [24]).

The fact that a structure is Ramsey can be exploited when analysing its automorphism group, endomorphism monoid, or polymorphism clone. Our usage of Ramsey theory is almost exclusively via the concept of *canonical functions*. For simplicity, we explain this concept for unary functions only; however, the ideas generalize straightforwardly to finitary functions; see [9] for an in-depth introduction to the method of canonical functions. A function $f: \Gamma \rightarrow \Gamma$ is called *canonical* if for all $\beta \in \text{Aut}(\Gamma)$ we have $f \circ \beta \in \{\alpha f \mid \alpha \in \text{Aut}(\Gamma)\}$. When Γ is an ordered Ramsey structure, then an arbitrary function ‘looks as a canonical function on large parts of the domain’: formally, for every function f over the domain of Γ , there exists a canonical function g in $\overline{\{\alpha f \beta \mid \alpha, \beta \in \text{Aut}(\Gamma)\}}$ – the *canonisation lemma*. In practice, we often use a generalisation of canonisation involving constants – we refer to [9] for details. Suppose now that Γ is homogeneous in a finite relational signature. Then there are only finitely many behaviours of canonical functions, and this is essential to break classification arguments dealing with endomorphisms (and polymorphisms) into finitely many cases. We hope that canonical functions and canonization can be used to reduce Conjecture 3 to Theorem 4 and Conjecture 2.

The method of canonical functions has been used extensively in [7, 13, 4, 10, 9, 15, 11], in two contexts: complexity classification of CSPs and classification of reducts of homogeneous structures.

When is it possible to apply this method to analyse the endomorphisms (and polymorphisms) of \mathfrak{C} ? We do not need \mathfrak{C} to be Ramsey, it suffices that \mathfrak{C} has a homogeneous expansion with finite relational signature which is Ramsey. The following question is therefore of essential importance.

► **Question 1** ([14]). *Is it true that every homogeneous structure with finite relational signature has a homogeneous Ramsey expansion with finite relational signature?*

Similar in spirit, we ask the following.

► **Question 2** ([14]). *Can every ω -categorical structure be expanded to an ω -categorical structure which is Ramsey?*

These questions are closely related to recent research in topological dynamics – we refer to a recent survey article for more on this connection [29]. A positive answer to Question 1 would imply that the method of Ramsey theory and canonical functions can be used to approach the tractability conjecture from Section 5 in general.

References

- 1 Libor Barto and Marcin Kozik. Absorbing subalgebras, cyclic terms and the constraint satisfaction problem. *Logical Methods in Computer Science*, 8/1(07):1–26, 2012.
- 2 Manuel Bodirsky and Martin Grohe. Non-dichotomies in constraint satisfaction complexity. In Luca Aceto, Ivan Damgard, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, and Igor Walukiewicz, editors, *Proceedings of the International Colloquium*

- on Automata, Languages and Programming (ICALP), Lecture Notes in Computer Science, pages 184–196. Springer Verlag, July 2008.
- 3 Manuel Bodirsky, Martin Hils, and Barnaby Martin. On the scope of the universal-algebraic approach to constraint satisfaction. *Logical Methods in Computer Science (LMCS)*, 8(3:13), 2012. An extended abstract that announced some of the results appeared in the proceedings of Logic in Computer Science (LICS'10).
 - 4 Manuel Bodirsky, Peter Jonsson, and Trung Van Pham. The reducts of the homogeneous binary branching C-relation. Preprint arXiv:1408.2554, 2014.
 - 5 Manuel Bodirsky, Peter Jonsson, and Timo von Oertzen. Essential convexity and complexity of semi-algebraic constraints. *Logical Methods in Computer Science*, 8(4), 2012. An extended abstract about a subset of the results has been published under the title *Semilinear Program Feasibility* at ICALP'10.
 - 6 Manuel Bodirsky and Jan Kára. The complexity of equality constraint languages. *Theory of Computing Systems*, 3(2):136–158, 2008. A conference version appeared in the proceedings of Computer Science Russia (CSR'06).
 - 7 Manuel Bodirsky and Jan Kára. The complexity of temporal constraint satisfaction problems. *Journal of the ACM*, 57(2):1–41, 2009. An extended abstract appeared in the Proceedings of the Symposium on Theory of Computing (STOC'08).
 - 8 Manuel Bodirsky and Jaroslav Nešetřil. Constraint satisfaction with countable homogeneous templates. *Journal of Logic and Computation*, 16(3):359–373, 2006.
 - 9 Manuel Bodirsky and Michael Pinsker. Reducts of Ramsey structures. *AMS Contemporary Mathematics, vol. 558 (Model Theoretic Methods in Finite Combinatorics)*, pages 489–519, 2011.
 - 10 Manuel Bodirsky and Michael Pinsker. Schaefer's theorem for graphs. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*, pages 655–664, 2011. Preprint of the long version available at arxiv.org/abs/1011.2894.
 - 11 Manuel Bodirsky and Michael Pinsker. Minimal functions on the random graph. *Israel Journal of Mathematics*, 200(1):251–296, 2014.
 - 12 Manuel Bodirsky and Michael Pinsker. Topological Birkhoff. *Transactions of the American Mathematical Society*, 2014. To appear (electronic version is published). Preprint arxiv.org/abs/1203.1876.
 - 13 Manuel Bodirsky, Michael Pinsker, and András Pongrácz. The 42 reducts of the random ordered graph. Preprint arXiv:1309.2165, 2013.
 - 14 Manuel Bodirsky, Michael Pinsker, and Todor Tsankov. Decidability of definability. *Journal of Symbolic Logic*, 78(4):1036–1054, 2013. A conference version appeared in the Proceedings of LICS 2011, pages 321–328.
 - 15 Manuel Bodirsky and Michał Wrona. Equivalence constraint satisfaction problems. In *Proceedings of Computer Science Logic*, volume 16 of *LIPICS*, pages 122–136. Dagstuhl Publishing, September 2012.
 - 16 V. G. Bodnarčuk, L. A. Kalužnin, V. N. Kotov, and B. A. Romov. Galois theory for Post algebras, part I and II. *Cybernetics*, 5:243–539, 1969.
 - 17 Andrei A. Bulatov, Andrei A. Krokhin, and Peter G. Jeavons. Classifying the complexity of constraints using finite algebras. *SIAM Journal on Computing*, 34:720–742, 2005.
 - 18 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. *SIAM Journal on Computing*, 28:57–104, 1999.
 - 19 David Geiger. Closed systems of functions and predicates. *Pacific Journal of Mathematics*, 27:95–100, 1968.
 - 20 C. Ward Henson. Countable homogeneous relational systems and categorical theories. *Journal of Symbolic Logic*, 37:494–500, 1972.

- 21 David Hobby and Ralph McKenzie. *The structure of finite algebras*, volume 76 of *Contemporary Mathematics*. American Mathematical Society, 1988.
- 22 Wilfrid Hodges. *A shorter model theory*. Cambridge University Press, Cambridge, 1997.
- 23 P. G. Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science*, 200:185–204, 1998.
- 24 Alexander Kechris, Vladimir Pestov, and Stevo Todorćevic. Fraissé limits, Ramsey theory, and topological dynamics of automorphism groups. *Geometric and Functional Analysis*, 15(1):106–189, 2005.
- 25 Klaus Leeb. *Vorlesungen über Pascaltheorie*, volume 6 of *Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung*. Friedrich-Alexander-Universität Erlangen-Nürnberg, 1973.
- 26 Keith R. Milliken. A Ramsey theorem for trees. *Journal of Combinatorial Theory, Series A*, 26(3):215 – 237, 1979.
- 27 Jaroslav Nešetřil and Vojtěch Rödl. The partite construction and Ramsey set systems. *Discrete Mathematics*, 75(1-3):327–334, 1989.
- 28 Walter Taylor. Varieties obeying homotopy laws. *Canadian Journal of Mathematics*, 29:498–527, 1977.
- 29 Lionel Nguyen Van Thé. A survey on structural ramsey theory and topological dynamics with the Kechris-Pestov-Todorćevic correspondence in mind. *Accepted for publication in Zb. Rad. (Beogr.)*, 2014. Preprint arXiv:1412.3254v2.

Parallel Algorithms Reconsidered

Peter Sanders

Karlsruhe Institute of Technology
Karlsruhe, Germany
sanders@kit.edu

Abstract

Parallel algorithms have been a subject of intensive algorithmic research in the 1980s. This research almost died out in the mid 1990s. In this paper we argue that it is high time to reconsider this subject since a lot of things have changed. First and foremost, parallel processing has moved from a niche application to something mandatory for any performance critical computer applications. We will also point out that even very fundamental results can still be obtained. We give examples and also formulate some open problems.

1998 ACM Subject Classification F.2 Analysis of Algorithms and Problem Complexity, F.1.2 Parallelism and Concurrency

Keywords and phrases parallel algorithm, algorithm engineering, communication efficient algorithm, polylogarithmic time algorithm, parallel machine model

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.10

Category Invited Talk

1 Introduction

Parallel algorithms were a hot topic in the 1980 but then the subject almost died. For example, a quick, subjective count of the parallel algorithm papers in STOC 1985, 1990, 1995, 2000, 2005, 2010, 2014 gave 13, 8, 11, 6, 1, 1, 6 papers respectively. The left column of the following table gives a number of interrelated very strong reasons why this happened. However, if you also look at the right column, you see that these reasons are not relevant any more.

Parallel computing was in practice used rarely because parallel computers were expensive and hard to program due to exotic hardware and software.	Today, parallel hardware is everywhere (see Figure 1). Even smart phones have quad-core processors. The latest Intel server processors support up to 18 cores. With multiple sockets and hardware multithreading, this already ranges into three digit numbers of threads. Graphics processors increasingly used for general purpose computing (GPGPU) have thousands of cores. For example, the NVidia GTX 980 card has 2048 cores and needs a number of hardware threads at least an order of magnitude larger to achieve full performance.
---	---

For most programmers, it was easier to wait until the microprocessor industry provided new processor designs that translate the additional transistor budget due to Moore's law into higher **clock frequency** and higher instruction parallelism.

The actual applications of parallel computers were mostly **numerical** simulations that needed little of the results developed by the algorithm theory community. Exceptions (that almost prove the rule) can be found for lower level aspects like network topologies, e.g. [21].

The machine models used by theorists, like the **PRAM** model were widely criticized as too remote from practice.

Most companies specializing in parallel computers did go bankrupt.

In the late 1990s, the **Internet boom** (aka Dot-com bubble) drew parallel algorithms researchers into startups (e.g., Akamai) and into new research fields related to emerging internet applications (e.g., algorithmic game theory).

These observations indicate that parallel algorithms should be an even hotter topic than in the 1980s. It seems that today the theory community is lagging behind an important trend. One way to explain this lack of enthusiasm is the hypothesis that, perhaps, researchers may have done a very thorough job in the past and discovered almost all the really interesting parallel algorithms that are there to discover. The main purpose of the remainder of this paper is to refute this hypothesis.

First it should be noted, that in the last two decades there have been important trends in computer science that have a largely unaddressed parallel processing aspect:

- There has been a lot of work on processing large data sets in the presence of *memory hierarchies* (e.g., [23, 43]). Many of the techniques developed there, e.g., time forward processing (e.g., [10, 11]), do not readily translate to parallel processing and thus pose important open problems.

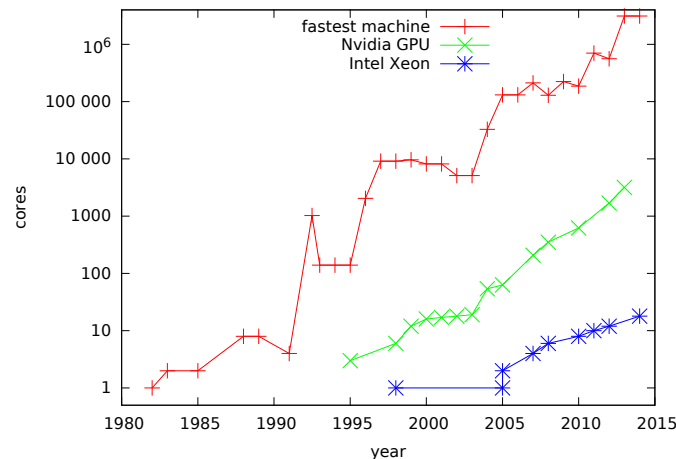
This stopped when processor design ran against the “power wall” – it is no longer feasible to significantly increase processor clock speeds since this increases the energy consumption to a point where energy costs are too high and where cooling becomes too expensive [16]. For example, in 2004, Intel presented the Pentium 4 Prescott microarchitecture that increased clock frequency and energy consumption but *not* benchmark performance compared to previous models. A short time later, the Netburst line of microarchitectures used for the Pentium 4 was discontinued and Intel started to design more conservative cores putting more and more of them on the same chip.

Now, we have to look for parallelization opportunities in every performance critical application since this is the only way to exploit the available hardware. Moreover, the Big Data boom has produced many new applications outside numerical simulations where massively parallel processing is crucial. Furthermore, the methodology of algorithm engineering (e.g. [29]) makes it easier to bridge gaps between theory and practice.

Message passing models (e.g., [41, 33]) or memory hierarchies [4] avoid some of the pitfalls of PRAMs. Moreover, PRAM algorithms are often not that impractical if we avoid the fallacy of speeding up computations at the cost of highly inefficient computing.

Now, parallel computers are mainstream products of the big players. Big data and cloud is at the core of the business of some of the worlds most profitable companies. Computer games proved to be a killer application (almost literally), catalyzing the use of architectures (GPUs) that would otherwise have been dismissed as too exotic and cumbersome.

Some of these people and a new generation of researchers now look at parallel algorithms from a fresh Big Data perspective. Indeed, in 2014 there were 6 STOC papers on parallel algorithms again.



■ **Figure 1** Number of processors (cores) in the worlds fastest supercomputers [40], Nvidia GPUs [44], and Intel Xeon server processors (single chip) [45].

- Data movement in memory hierarchies is vertical data movement between memory units at different levels. Horizontal data movements between processors in a distributed memory machine is an equally important related problem but has been studied much less. An important difference is that horizontal communication volume can be sublinear in the input size if we manage to solve problems by predominantly local computation. The resulting area of *communication efficient algorithms* is full of interesting open problems [33].
- *Streaming* algorithms [18] have explicitly been developed to allow processing large data sets. However, the basic model for streaming algorithms is inherently sequential and needs parallel generalizations. See also [33].
- Many *succinct data structures* (e.g. [17]) have been designed to handle large data sets. However, for many of them it is not clear how to construct them efficiently in parallel.
- *Smoothed analysis* [37] is a sound way to explain why certain algorithms for hard problems are efficient in practice. However, few parallel algorithms have been analyzed in this framework.
- *Fixed-parameter algorithms* (e.g., [25]) study efficient algorithms for “easy” instances of hard problems. Few of these algorithms have been parallelized so far.
- There has been some early work on parallel *approximation algorithms* [22] but very little on parallelizing the vast number of approximation algorithms studied since then. It is particularly surprising that even the intensive work on scheduling parallel processors has seen very few algorithms for doing that in parallel [3, 31].
- Application areas like *bioinformatics* or *computational finance* recently had large impact on algorithmic research. Many of the investigated problems require parallel algorithms to be useful in practice.
- The *big data* boom brought a large number of new applications into focus, in particular, algorithms for *data analysis* and *machine learning* become important.
- The *energy consumption* of computations is becoming more important than running time. this should become important for algorithm design, in particular for *exascale computing* where the computer architects are already basing most of their design decisions on energy consumption (e.g. [20]).

- Applications on exascale computers and Big data also require *fault tolerance*. Building that already into the algorithms is a promising research area. The algorithm theory community has done some work on *resilient algorithms* [14] that can survive certain memory corruptions but is has not embraced fault tolerant parallel algorithms. This is surprising because fault tolerance is actually *easier* to achieve in a parallel system since intact processors may step in for faulty ones.

2 Examples from our Work

In order to illustrate that there is a bonanza of quite fundamental results on parallel algorithms still to be found, we describe a selection of our results on parallel algorithms published since 2013.

2.1 Sorting

Sorting is one of the most intensively studied algorithmic problems. It is of particular interest to parallel computing since sorting is often used to bring data together that has to be processed together. We were able to obtain several quite fundamental new results on sorting.

String sorting. is practically important since many big data applications have variable length keys. The theoretical challenge here is to exploit that only *distinguishing prefixes* need to be inspected – indeed sequential string sorting needs work only linear in the total distinguishing prefix size. We found *no* previous work on parallel string sorting except some PRAM algorithms always inspecting the entire input which are thus work-inefficient. We adapted parallel sorting algorithms for atomic objects so that they only inspect the distinguishing prefixes [8, 7].

Malleable sorting. In practice, parallel programs have to share resources (e.g. processors) with other programs. Therefore, the amount of available resources for a particular program may vary over time in an online fashion. Thus parallel algorithms should be able to dynamically adapt to the amount of available resources. We have studied this phenomenon for the example of sorting and show that this yields advantages over leaving this adaptivity to the operating system [15].

Massively parallel sorting. There are many asymptotically efficient sorting algorithms running in polylogarithmic time. However, all these algorithms require the data to be moved at least a logarithmic number of times. On the other hand, there are algorithms that need to move data only once which makes them much more practical for sorting large inputs on distributed memory machines. However, these algorithms need a linear number of message startups on the critical path which makes them impractical for large machines. We have designed algorithms that interpolate between these to extremes – moving the data k times reduces the critical path length to $kp^{1/k}$ [5]. There were similar algorithms but none with a comparable worst case guarantee.

2.2 Data Structures

There has been a lot of work on concurrent data structures (e.g. [19]). However, much of this is very slow in the worst case since contention of operations competing for the same

place in the data structure can occur. It turns out that these problems can sometimes be avoided by relaxing the data structure semantics or by considering bulk operations.

Relaxed Priority Queues. support concurrent insertions and deletions of elements that have near minimum values. We have designed a very simple such data structure (MultiQueue) based on multiple sequential priority queues. Insertions go to random queues and deletions take the minimum from two randomly sampled queues [28]. This data structure considerably outperforms much more complicated previous data structures.

Approximate Membership. Bloom filters save communication volume by providing a space efficient data structure for approximate membership queries. However, there is surprisingly little work on distributed Bloom filters. For example, a recent survey on Bloom filters in distributed systems [39] mentions no less than 23 variants but none that truly distributes the data structure over multiple processors and thus scales to the largest data sets. We have designed such a structure and apply it for communication efficient duplicate detection and database join [33].

2.3 Graph Algorithms

Multi-objective Shortest Paths. is an intensively studied problem of high practical relevance where parallelization is attractive since it is computationally much more expensive than the standard single-objective case. While the latter problem is difficult to parallelize in the worst case, we have shown that the additional work due to the added objectives is easy to parallelize. Indeed, a very simple generalization of Dijkstra's well known single-objective algorithm turns out to be a scalable parallel algorithm requiring the same number of n iterations [32]. This algorithm also works well in practice [13]. Another interesting aspect of this problem is that it combines graphs and computational geometry.

Maximal matchings. We give a simple linear work polylogarithmic time algorithm for computing maximal matchings in [9]. The algorithm also computes 1/2-approximations of weighted matchings and works well in practice.

Graph partitioning. asks for partitioning the vertex set of a graph into k pieces of about equal size such that the number of cut edges is small. This is a frequently needed (NP-hard) problem that is particularly important for processing graphs in parallel. Our partitioner KaHIP [34] yields the highest quality world wide for a large spectrum of inputs including some of the largest inputs considered so far [1, 24]. The algorithms used are complex heuristics combining many techniques. What is interesting from an algorithm theory point of view is that the practical quality improvements we achieve are in large parts due to integrating solvers for graph theoretical subproblems for which polynomial time algorithms are known. For example, this includes maximum flows, strongly connected components, negative cycle detection, or edge coloring.

2.4 Linear Algebra

One criticism of classical PRAM algorithms is not so much founded in the machine model but in the strive for polylogarithmic execution time even at the cost of inefficient algorithms. One such example are algorithms for matrix inversion and related problems. Theoretical research has found polylogarithmic time inefficient algorithms whereas the algorithms used

in practice perform a near optimal amount of work yet need time $\Omega(n)$ where n is the matrix dimension. We found an asymptotically efficient polylogarithmic time algorithm that also works well in practice [35]. The algorithm combines Strassen's recursive algorithm [38] with an inefficient algorithm based on Newton's method [26]. Overall, this reduces matrix inversion to a polylogarithmic number of matrix multiplications. Since the inefficient algorithm is only applied to small subproblems, the overall algorithm is efficient.

3 Selected Open Problems

To further underline that there is a lot to be done, we give a list of open problems selected for being quite fundamentally interesting and spanning a wide range of topics. The ordering is roughly from rather specific questions to quite general problem areas with a large number of concrete possible projects.

1. *Priority Queues*: Show probabilistic quality guarantees for the MultiQueues from [28] or design a comparably fast data structure with provable guarantees.
2. *Strongly Connected Components*: Is there a polylogarithmic time, work efficient algorithm for finding strongly connected components? (Similar questions can be asked for many graph problems.)
3. *Matchings*: Give a linear work polylogarithmic time parallel algorithm for $(1 - \epsilon)$ -approximation of weighted matchings – perhaps by parallelizing [12]. Even the unweighted case, or the $2/3 - \epsilon$ -approximation algorithms like [27] would be an interesting result.
4. *Data Exchange*: There has been intensive work on routing in a wide spectrum of networks. However, even very simple models have wide open problems. For example, consider a half-duplex fully connected model: any one of p processors can move a data packet to any other processor in one step. However, at any time, a processor can only be involved in a single communication. Let h denote the maximum number of packets any processor is involved in. Delivering the data directly is equivalent to edge coloring of multigraphs and thus takes about $\frac{3}{2}h$ steps in the worst case [36]. We show that routing the packets on detours can lower this to about $\frac{6}{5}h$ [30]. In the very special case that one fourth of the processors need not communicate at all, this can be reduced to h steps [2]. An interesting conjecture is that $\approx h$ steps also suffice as long as the total number of packets is at most $\frac{3}{8}hp$.
5. *Solving Systems of Linear Equations*: Is there a polylogarithmic time algorithm with work $O(n^3)$ that solves a system of equations $Ax = b$ in a comparably stable way as Gaussian elimination? The matrix inversion algorithm from [35] is not stable enough for all applications, in particular if A is not symmetric. On the other hand, Gaussian elimination is P-complete [42].
6. *Compressed Text Indexing*: Develop a polylogarithmic time work-efficient algorithm for constructing compressed suffix arrays or related data structures.
7. *Parallel Paging*: A lot of work has been done on the sequential paging problem [6] – given a single sequence of data block accesses, decide, which of them should be kept in cache at what time to minimize the number of block transfers between slow memory and cache. Very little is known on parallel paging. For example, given n such sequences representing tasks, schedule them on p processors such that the bottleneck number of block transfers is minimized. We may want to distinguish between shared and private caches, online and offline strategies, ...
8. *Energy Efficient Computing*: Find a simple model with high predictive value for the actual energy efficiency of (parallel) algorithms.

9. *Communication Efficient Algorithms*: For your favorite algorithmic problem, find out whether there exists a communication efficient parallel algorithm for it.
10. *Parallel Streaming Algorithms*: Assume data arrives in data streams to p processors with limited *local* memory. How can you approximate important information about the data while keeping the communication volume small? The concrete problem considered could be any problem previously considered in sequential streaming algorithms.

References

- 1 Yaroslav Akhremtsev, Peter Sanders, and Christian Schulz. (semi-)external algorithms for graph partitioning and clustering. In *Alenex 2015*, 2015.
- 2 Eric Anderson, Joseph Hall, Jason Hartline, Mick Hobbes, Anna Karlin, Jared Saia, Ram Swaminathan, and John Wilkes. Algorithms for data migration. *Algorithmica*, 57(2):349–380, 2010.
- 3 Richard J. Anderson, Ernst W. Mayr, and Manfred K. Warmuth. Parallel approximation algorithms for bin packing. *Inf. Comput.*, 82(3):262–277, 1989.
- 4 Lars Arge, Michael T Goodrich, Michael Nelson, and Nodari Sitchinava. Fundamental parallel algorithms for private-cache chip multiprocessors. In *20th Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 197–206. ACM, 2008.
- 5 Michael Axtmann, Timo Bingmann, Peter Sanders, and Christian Schulz. Practical massively parallel sorting—basic algorithmic ideas. *arXiv preprint arXiv:1410.6754*, 2014.
- 6 A. L. Belady. A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, 5:78–101, 1966.
- 7 Timo Bingmann, Andreas Eberle, and Peter Sanders. Engineering parallel string sorting. *arXiv preprint arXiv:1403.2056*, 2014.
- 8 Timo Bingmann and Peter Sanders. Parallel string sample sort. In *21st European Symposium on Algorithms (ESA)*, volume 8125 of *LNCS*, pages 169–180. Springer, 2013.
- 9 Marcel Birn, Vitaly Osipov, Peter Sanders, Christian Schulz, and Nodari Sitchinava. Efficient parallel and external matching. In *Europar*, LNCS. Springer, 2013.
- 10 Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter. External memory graph algorithms. In *6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 139–149, 1995.
- 11 R. Dementiev, P. Sanders, D. Schultes, and J. Sibeyn. Engineering an external memory minimum spanning tree algorithm. In *IFIP TCS*, pages 195–208, Toulouse, 2004.
- 12 Ran Duan and Seth Pettie. Approximating maximum weight matching in near-linear time. In *51st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 673–682. IEEE, 2010.
- 13 Stephan Erb, Moritz Kobitzsch, and Peter Sanders. Parallel bi-objective shortest paths using weight-balanced B-trees with bulk updates. In *13th Symposium on Experimental Algorithms (SEA)*, 2014.
- 14 Irene Finocchi, Fabrizio Grandoni, and Giuseppe F Italiano. Designing reliable algorithms in unreliable memories. In *Algorithms-ESA 2005*, volume 3669 of *LNCS*, pages 1–8. Springer, 2005.
- 15 Patrick Flick, Peter Sanders, and Jochen Speck. Malleable sorting. In *27th International Symposium on Parallel & Distributed Processing (IPDPS)*, pages 418–426. IEEE, 2013.
- 16 Michael Flynn and Patrick Hung. Microprocessor design issues: thoughts on the road ahead. *Micro, IEEE*, 25(3):16–31, 2005.
- 17 Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35(2):378–407, 2005.

- 18 Monika Henzinger, Prabhakar Raghavan, and Sridar Rajagopalan. Computing on data streams. *External Memory Algorithms: DIMACS Workshop External Memory and Visualization, May 20-22, 1998*, 50:107, 1999.
- 19 M. Herlihy and N. Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann, 2008.
- 20 Peter Kogge et al. Exascale computing study: Technology challenges in achieving exascale systems. Technical Report CSE-TR-2008-13, U. of Notre Dame / DARPA IPTO, 2008.
- 21 Charles E Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *Computers, IEEE Transactions on*, 100(10):892–901, 1985.
- 22 Ernst W. Mayr. Parallel approximation algorithms. Technical Report STm-CS-88-1225, Stanford University, 1988.
- 23 U. Meyer, P. Sanders, and J. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2625 of *LNCS Tutorial*. Springer, 2003.
- 24 Henning Meyerhenke, Peter Sanders, and Christian Schulz. Parallel graph partitioning for complex networks. In *29th IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2015. to appear.
- 25 Rolf Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.
- 26 Victor Pan and John Reif. Fast and efficient parallel solution of dense linear systems. *Computers & Mathematics with Applications*, 17(11):1481 – 1491, 1989.
- 27 S. Pettie and P. Sanders. A simpler linear time $2/3 - \epsilon$ approximation for maximum weight matching. *Information Processing Letters*, 91(6):271–276, 2004.
- 28 Hamza Rihani, Peter Sanders, and Roman Dementiev. Multiqueues: Simpler, faster, and better relaxed concurrent priority queues. *CoRR*, abs/1411.1209, 2014.
- 29 P. Sanders. Algorithm engineering – an attempt at a definition. In *Efficient Algorithms*, volume 5760 of *LNCS*, pages 321–340. Springer, 2009.
- 30 P. Sanders and R. Solis-Oba. How helpers hasten h -relations. *Journal of Algorithms*, 41:86–98, 2001.
- 31 P. Sanders and J. Speck. Exact parallel malleable scheduling for tasks with concave speedup functions. In *25th IEEE International Parallel and Distributed Processing Symposium*, pages 1156–1166, 2011.
- 32 Peter Sanders and Lawrence Mandow. Parallel label-setting multi-objective shortest path search. In *27th IEEE International Parallel & Distributed Processing Symposium*, pages 215–224, 2013.
- 33 Peter Sanders, Sebastian Schlag, and Ingo Müller. Communication efficient algorithms for fundamental big data problems. In *IEEE Int. Conf. on Big Data*, 2013.
- 34 Peter Sanders and Christian Schulz. Think locally, act globally: Highly balanced graph partitioning. In *Experimental Algorithms*, pages 164–175. Springer Berlin Heidelberg, 2013.
- 35 Peter Sanders, Jochen Speck, and Raoul Steffen. Work-efficient matrix inversion in polylogarithmic time. In *25th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 214–221. ACM, 2013.
- 36 C. E. Shannon. A theorem on colouring lines of a network. *J. Math. Phys.*, 28(148–151), 1949.
- 37 D. Spielman and S.-H. Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004.
- 38 Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969. 10.1007/BF02165411.
- 39 Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. Theory and practice of Bloom filters for distributed systems. *Communications Surveys & Tutorials, IEEE*, 14(1):131–155, 2012.

- 40 TOP500 lists. <http://www.top500.org>. for data 1993–, Accessed: 2014-12-04.
- 41 L. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1994.
- 42 Stephen A Vavasis. Gaussian elimination with pivoting is p-complete. *SIAM journal on discrete mathematics*, 2(3):413–423, 1989.
- 43 Jeffrey Scott Vitter. Algorithms and data structures for external memory. *Foundations and Trends® in Theoretical Computer Science*, 2(4):305–474, 2008.
- 44 Wikipedia – list of Nvidia graphics processing units. en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units. Accessed: 2014-12-04, sum of number of cores for different purposes.
- 45 Wikipedia – list of Intel Xeon microprocessors. en.wikipedia.org/wiki/List_of_Intel_Xeon_microprocessors. Accessed: 2014-12-04.

Computational Social Choice*

Felix Brandt

Faculty of Informatics
TU München, Germany
brandtf@in.tum.de

Abstract

Over the past few years there has been a lively exchange of ideas between computer science, in particular *theoretical computer science* and *artificial intelligence*, on the one hand and economics, in particular *game theory* and *social choice*, on the other. This exchange goes in both directions and has produced active research areas such as *algorithmic game theory* and *computational social choice*.

Social choice theory concerns the formal analysis and design of methods for aggregating possibly conflicting preferences such as in voting, assignment, or matching problems. Much of the work in classic social choice theory has focused on results concerning the formal possibility and impossibility of aggregation functions that combine desirable properties.

This tutorial provided an overview of central results in social choice theory with a special focus on axiomatic characterizations as well as computational aspects. While some aggregation functions can be easily computed, others have been shown to be computationally intractable (e.g., NP-hard or #P-hard). Topics that were covered in this tutorial included

- (i) rational choice theory,
- (ii) Arrow's impossibility theorem,
- (iii) tournament solutions (such as the top cycle, the uncovered set, the Banks set, or the tournament equilibrium set), and
- (iv) randomized social choice functions.

The overarching theme were escape routes from negative results such as Arrow's impossibility theorem.

1998 ACM Subject Classification F.2 Theory of Computation: Analysis of Algorithms and Problem Complexity, J.4 Computer Applications: Social and Behavioral Sciences – Economics

Keywords and phrases social choice theory, economics, algorithms, theory

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.19

Category Tutorial

* This work was partially supported by the Deutsche Forschungsgemeinschaft under grant BR 2312/7-2.



© Felix Brandt;

licensed under Creative Commons License CC-BY

32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).

Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 19–19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



SYMPOSIUM
ON THEORETICAL
ASPECTS
OF COMPUTER
SCIENCE

Algorithmic Game Theory*

Paul W. Goldberg

Department of Computer Science
Oxford University, United Kingdom
Paul.Goldberg@cs.ox.ac.uk

Abstract

Game theory studies mathematical models of interactions amongst self-interested entities. A “solution concept” means a description of the outcome of a game, and it is important that it should be defined in such a way that a solution always exists (every game should have an outcome). Nash’s famous theorem that mixed-strategy equilibria are guaranteed to exist, resulted in *Nash equilibrium* being the most prominent solution concept in game theory.

As a result, computational challenges of the form “given a game, find a solution”, have the property that we are searching for something whose existence is guaranteed (they are *total search problems*). Moreover, these solutions belong to the complexity class NP, since it is usually straightforward to check whether a proposed solution is correct (an incorrect one will admit a profitable deviation by one or more of the players, and this is usually easy to find). However, in versions of the problem that appear to be computationally hard, we cannot apply NP-completeness, due to a result of Megiddo saying that total search problems cannot be NP-complete unless NP is equal to co-NP.

In this tutorial, which is intended for people familiar with NP-completeness, I give an overview of the alternative notions of computational hardness that apply to game-theoretic solution concepts. I discuss the complexity class PPAD (introduced by Papadimitriou) which captures the computational complexity of various classes of games that don’t seem to be solvable in polynomial time. I also mention the complexity classes PLS and FIXP, and the kinds of games that they apply to.

Suppose, alternatively, that we have a polynomial-time algorithm that applies to some given class of games. A follow-up question is whether there exist algorithms that find a solution via processes that reflect decentralised selfish behaviour. This is because a solution concept arguably remains unrealistic if it can be efficiently computed, but only using a highly centralised algorithm. In the second half of the tutorial I present some results on learning dynamics for equilibrium computation, and mention recent work on communication complexity and query complexity.

I discuss some research directions and open problems, such as the following. What are the prospects for proving that PPAD is as hard as NP? How about algorithms that find improved *approximate* Nash equilibria? 2-player games are easy to solve in practice, using the Lemke-Howson algorithm, so is there a satisfying mathematical sense in which 2-player games are easy to solve? (For example, a sense in which Lemke-Howson works “most of the time”?)

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, J.4 Economics

Keywords and phrases equilibrium, non-cooperative games, computational complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.20

Category Invited Talk

* This work was partially supported by EPSRC under grant EP/K01000X/1.

The Minimum Oracle Circuit Size Problem

Eric Allender¹, Dhiraj Holden², and Valentine Kabanets³

- 1 Department of Computer Science, Rutgers University
Piscataway, NJ, USA
allender@cs.rutgers.edu
- 2 Department of Computer Science, California Institute of Technology
Pasadena, CA, USA
dholden@caltech.edu
- 3 School of Computing Science, Simon Fraser University, Burnaby, BC, Canada
kabanets@cs.sfu.ca

Abstract

We consider variants of the Minimum Circuit Size Problem MCSP, where the goal is to minimize the size of *oracle* circuits computing a given function. When the oracle is QBF, the resulting problem MCSP^{QBF} is known to be complete for PSPACE under ZPP reductions. We show that it is *not* complete under logspace reductions, and indeed it is not even hard for TC^0 under uniform AC^0 reductions. We obtain a variety of consequences that follow if oracle versions of MCSP are hard for various complexity classes under different types of reductions. We also prove analogous results for the problem of determining the resource-bounded Kolmogorov complexity of strings, for certain types of Kolmogorov complexity measures.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Kolmogorov complexity, minimum circuit size problem, PSPACE, NP-intermediate sets

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.21

1 Introduction

The Minimum Circuit Size Problem (MCSP) asks to decide, for a given truth table f of a Boolean function and a parameter s , whether f is computable by a Boolean circuit of size at most s . MCSP is a well-known example of a problem in NP that is widely believed to be intractable, although it is not known to be NP-complete. MCSP is known to be hard for the complexity class SZK under BPP-Turing reductions [4], which provides strong evidence for intractability. On the other hand, Kabanets and Cai showed [11] that if MCSP is NP-complete under the “usual” sort of polynomial-time reductions, then $\text{EXP} \not\subseteq \text{P/poly}$. This can not be interpreted as strong evidence against NP-completeness – since it is widely conjectured that $\text{EXP} \not\subseteq \text{P/poly}$ – but it does indicate that it may be difficult to provide an NP-completeness proof.

However, there are other ways to define what the “usual” sort of reductions are: e.g., logspace, (uniform) TC^0 , AC^0 , or NC^0 . The overwhelming majority of problems that are known to be NP-complete are, in fact, NP-complete under very restricted kinds of reductions. Can we rule out NP-hardness of MCSP under such reductions?

Very recently, Murray and Williams [13] have shown that MCSP is not even P-hard under uniform NC^0 reductions. Can MCSP be NP-hard under slightly stronger reductions, e.g., uniform AC^0 reductions? We suspect that the answer is ‘No’, but so far we (like Murray



© Eric Allender, Dhiraj Holden, and Valentine Kabanets;
licensed under Creative Commons License CC-BY
32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).
Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 21–33



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



SYMPOSIUM
ON THEORETICAL
ASPECTS
OF COMPUTER
SCIENCE

and Williams) can only show that P-hardness of MCSP under uniform AC^0 , TC^0 , or logspace reductions would imply new (likely) complexity lower bounds (in the spirit of [11]).

The main focus of the present paper is an *oracle version* of MCSP, denoted $MCSP^A$ for a language A , which asks to decide for a given truth table f and a parameter s if f is computable by an A -oracle circuit of size at most s . We prove a number of implications of hardness of $MCSP^A$ for various choices of the oracle A , and various reductions. In particular, we prove for a PSPACE-complete A that $MCSP^A$ is *not* P-hard under uniform AC^0 reductions.

The results presented here (along with the results recently reported by Murray and Williams [13]) are the first results giving unlikely consequences that would follow if variants of MCSP or the various oracle circuit minimization problems are hard under a natural notion of reducibility. We also show that analogous results hold in the Kolmogorov complexity setting due to the correspondence between circuit size and Kolmogorov complexity, using the minimum-KT complexity problem defined in this paper.

Below we provide a summary of our main results.

1.1 Our results

Most of our results follow the template:

If $MCSP^A$ is hard for a complexity class \mathcal{C} under reductions of type \mathcal{R} , then complexity statement \mathcal{S} is true.

Table 1 below states our results for different instantiations of A , \mathcal{C} , \mathcal{R} , and \mathcal{S} ; note that $\mathcal{S} = \perp$ means that the assumption is false, i.e., $MCSP^A$ is *not* \mathcal{C} -hard under \mathcal{R} -reductions. Throughout, we assume that the reader is familiar with complexity classes such as NP, PP, PSPACE, NEXP, etc. We denote the polynomial hierarchy by PH, and its linear-time version (linear-time hierarchy) by LTH. The Counting Hierarchy, denoted CH, is the union of the classes PP, PP^{PP} , etc.

■ **Table 1** Summary of main results: If $MCSP^A$ is \mathcal{C} -hard under \mathcal{R} , then \mathcal{S} . The last column shows the theorem where the result is stated in the paper.

oracle A	class \mathcal{C}	reductions \mathcal{R}	statement \mathcal{S}	Theorem
PH-hard	TC^0	uniform AC^0	\perp	Theorem 20
any	TC^0	uniform AC^0	$LTH \not\subseteq io\text{-}SIZE^A[2^{\Omega(n)}]$	Lemma 21
any	TC^0	uniform AC^0	$NP^A \not\subseteq SIZE^A[\text{poly}]$	Corollary 24
any in CH	P	uniform TC^0	$P \neq PP$	Corollary 13
\emptyset	P	logspace	$P \neq PSPACE$	Corollary 14
QBF	P	logspace	$EXP = PSPACE$	Corollary 18
QBF	NP	logspace	$NEXP = PSPACE$	Theorem 17
QBF	PSPACE	logspace	\perp	Corollary 19
EXP-complete	NP	polytime	$NEXP = EXP$	Theorem 15

For the most restricted reductions, uniform AC^0 , we get that $MCSP^A$ is not TC^0 -hard for any oracle A such that $PH \subseteq SIZE^A[\text{poly}]$ (Theorem 20), e.g., for $A = \oplus P$ (Corollary 23). For any oracle A , we conclude new circuit lower bounds for the linear-time hierarchy and for NP^A (Lemma 21 and Corollary 24¹).

¹ Prior to our work, Murray and Williams have shown that if $SAT_{\leq m}^{AC^0} MCSP$, then $NP \not\subseteq P/\text{poly}$ [13]. Their result is similar to (and is implied by) our Corollary 24 for the case of $A = \emptyset$.

If MCSP is P-hard under uniform TC^0 or logspace reductions, then P is different from PP or from PSPACE (Corollaries 13 and 14).

One of the more interesting oracle circuit minimization problems is $MCSP^{QBF}$. It was shown in [3] that $MCSP^{QBF}$ is complete for PSPACE under ZPP-Turing reductions, but the question of whether it is complete for PSPACE under more restrictive reductions was left open. For most natural complexity classes \mathcal{C} above PSPACE, there is a corresponding oracle circuit minimization problem (which we will sometimes denote $MCSP^{\mathcal{C}}$) that is known to be complete under P/poly reductions, but is not known to be complete under more restrictive reductions [3]. For the particular case of $\mathcal{C} = PSPACE$, we denote this as $MCSP^{QBF}$. We show that $MCSP^{QBF}$ is not PSPACE-complete under logspace reductions (Corollary 19). Furthermore, it is not even TC^0 -hard under uniform AC^0 reductions (Theorem 20).

Finally, for even more powerful oracles A , we can handle even general polynomial-time reductions. We show that if $SAT_{\leq m}^p \leq MCSP^{EXP}$, then $EXP = NEXP$ (Theorem 15).

We believe that MCSP is not TC^0 -hard under even *nonuniform* AC^0 reductions. While we are unable to prove this, we can rule out restricted AC^0 reductions for a certain gap version of MCSP. Define *gap-MCSP* as follows: Given a truth table f and a parameter s , output ‘Yes’ if f requires circuit size s , and output ‘No’ if f can be computed by a circuit of size at most $s/2$. Call a mapping from n -bit strings to m -bit strings $\alpha(n)$ -*stretching* if $m \leq n \cdot \alpha(n)$, for some function $\alpha : \mathbb{N} \rightarrow \mathbb{R}^{\geq 0}$.

We prove that gap-MCSP is *not* TC^0 -hard under nonuniform AC^0 reductions that are $n^{1/31}$ -stretching (Theorem 27).

1.2 Related work

The most closely related is the recent paper by Murray and Williams [13], which also considers the question whether MCSP is NP-complete under weak reductions, and proves a number of conditional and unconditional results. The main unconditional result is that MCSP is *not* TC^0 -hard under uniform NC^0 reductions (or more generally, under $O(n^{1/2-\epsilon})$ -time projections, for every $\epsilon > 0$); we give an alternative proof of this result (Theorem 25). For conditional results, [13] shows that if MCSP is NP-hard under uniform AC^0 reductions, then $NP \not\subseteq P/poly$ and $E \not\subseteq io\text{-SIZE}[2^{\Omega(n)}]$ (also implied by our Corollary 24 and Lemma 21), and that NP-hardness of MCSP under general polynomial-time reductions implies $EXP \neq ZPP$.

$MCSP$, $MCSP^{QBF}$ and other oracle circuit minimization problems are closely related to notions of resource-bounded Kolmogorov complexity. Briefly, a small (oracle) circuit is a short description of the string that represents the truth-table of the function computed by the circuit. Notions of resource-bounded Kolmogorov complexity were presented and investigated in [3] that are roughly equivalent to (oracle) circuit size.

In particular, there is a space-bounded notion of Kolmogorov complexity, KS , such that the set of KS -random strings (denoted R_{KS}) is complete for PSPACE under ZPP reductions. It is shown in [3] that R_{KS} is not even hard for TC^0 under AC^0 reductions, and R_{KS} is not hard for PSPACE under logspace-Turing reductions. The proof of this non-hardness result also carries over to show that a set such as $\{f : f \text{ is the truth table of a function on } n \text{ variables that has QBF circuits of size at most } 2^{n/2}\}$ is also not hard for TC^0 under AC^0 reductions, and is not hard for PSPACE under logspace-Turing reductions. However it does *not* immediately carry over to $MCSP^{QBF}$, which is defined as $\{(f, i) : f \text{ is the truth table of a function on } n \text{ variables that has QBF circuits of size at most } i\}$; similarly it does not carry over to the set $\{(x, i) : KS(x) \leq i\}$. Also, the techniques presented in [3] have not seemed to provide any tools to derive consequences assuming completeness results for oracle circuit minimization problems for oracles less powerful than PSPACE. We should point out,

however, that [3] proves a result similar to (and weaker than) our Lemma 21 in the context of time-bounded Kolmogorov complexity: if R_{KT} is TC^0 -hard under AC^0 many-one reductions, then $PH \not\subseteq SIZE \left[2^{n^{o(1)}} \right]$.

1.3 Our techniques

To illustrate our proof techniques, let us sketch a proof of one of our results: If MCSP is P-hard under uniform logspace reductions, then $P \neq PSPACE$ (Corollary 14).

The proof is by contradiction. Suppose that $P = PSPACE$. Our logspace reduction maps n -bit instances of QBF to n^c -bit instances (f, s) of MCSP so that each bit of f is computable in $O(\log n)$ space.

1. Imagine that our reduction is given as input a *succinct* version of QBF, where some $\text{poly}(\log n)$ -size circuit D on each $\log n$ -bit input $1 \leq i \leq n$ computes the i th bit of the QBF instance. It is not hard to see that our reduction, given the circuit D , can compute each bit of f in $\text{poly}(\log n)$ space. Thus the Boolean function with the truth table f is computable by a $PSPACE = P$ algorithm (which also has the circuit D as an input). It follows that this function f is computable by some polynomial-size Boolean circuit.
2. Next, since we know that f has at most polynomial circuit complexity, to decide the MCSP instance (f, s) , we only need to consider the case where $s < \text{poly}$ (since for big values of s , the answer is ‘Yes’). But deciding such MCSP instances (which we call *succinct MCSP*) is possible in Σ_2^P : guess a circuit of size at most s , and verify that it agrees with the given polynomial-size circuit for f on all inputs.
3. Finally, since $\Sigma_2^P \subseteq PSPACE = P$, we get that our succinct MCSP instances can be decided in P . The reduction from succinct QBF to succinct MCSP is also in $PSPACE = P$. Hence, succinct QBF is in P . But, succinct QBF is EXSPACE-complete, and so we get the collapse $EXSPACE = P$, contradicting the hierarchy theorems.

In step (1) of the sketched proof, the uniformity of an assumed reduction to MCSP is used to argue that the truth table f produced by the reduction is in fact “easy” to compute uniformly. The uniform complexity of computing the function f is roughly the “exponential” analogue of the uniform complexity of the reduction. For circuit classes such as AC^0 and TC^0 , we use the well-known connection between the “exponential” analog of uniform AC^0 and PH , and between the “exponential” analog of uniform TC^0 and CH .

We use the uniform easiness of the function f to conclude that f has small circuit complexity (and hence our reduction actually outputs instances of *succinct MCSP*). To get that conclusion, we need to assume (or derive) the collapse to P/poly of the uniform complexity class that contains f ; in our example above, we got it from the assumption that $PSPACE = P$.

Step (2) exploits the fact that succinct MCSP does *not* become “exponentially harder” (unlike the usual succinct versions of hard problems), but is actually computable in Σ_2^P .

In Step (3), we combine the algorithm for our reduction and the algorithm for succinct MCSP to get an “efficient” algorithm for the succinct version of the input problem (succinct QBF in our example). Since the succinct version of the input problem *does* become exponentially harder than its non-succinct counterpart, we get some impossible collapse (which can be disproved by diagonalization).

We use this style of proof for all our results involving reductions computable by uniform TC^0 and above. However, for the case of uniform AC^0 (and below), we get stronger results by replacing the diagonalization argument of Step (3) with the nonuniform AC^0 circuit lower bound for PARITY [10].

Remainder of the paper. We state the necessary definitions and auxiliary results in Section 2. Our main results are proved in Section 3, and some generalizations are given in Section 4. We give concluding remarks in Section 5.

2 Definitions

► **Definition 1.** The minimum circuit size problem MCSP, as defined in [11], is defined as $\{(f, s) \mid f \text{ has circuits of size } s\}$, where f is a string of length 2^m encoding the entire truth-table of some m -variate Boolean function. (Versions of this problem have been studied long prior to [11]. See [4, 17] for a discussion of this history.) We will also consider the analogous problem for circuits with oracles, the Minimum A -Circuit Size problem MCSP^A , defined analogously, where instead of ordinary circuits, we use circuits that also have oracle gates that query the oracle A . When A is a standard complete problem for some complexity class \mathcal{C} , we may refer to this as $\text{MCSP}^{\mathcal{C}}$.

We will not need to be very specific about the precise definition of the “size” of a circuit. Our results hold if the “size” of a circuit is the number of gates (including oracle gates), or the number of “wires”, or the number of bits used to describe a circuit in some standard encoding. It is perhaps worth mentioning that the different versions of MCSP that one obtains using these different notions of “size” are not known to be efficiently reducible to each other.

Circuit size relative to oracle A is polynomially-related to a version of time-bounded Kolmogorov complexity, denoted KT^A , which was defined and studied in [3].

► **Definition 2.** $\text{KT}^A(x) = \min\{|d| + t : \forall b \in \{0, 1, *\} \forall i \leq |x| + 1 \ U^A(d, i, b) \text{ accepts in } t \text{ steps iff } x_i = b\}$. Here, U is some fixed universal Turing machine, which has random access to the oracle A and to the input string (or “description”) d ; x_i denotes the i -th symbol of x , where $x_{|x|+1} = *$.

By analogy to MCSP^A , we define the “minimum KT problem”:

► **Definition 3.** $\text{MKTP}^A = \{(x, i) \mid \text{KT}^A(x) \leq i\}$.

All of our results that deal with MCSP^A also apply to MKTP^A .

We wish to warn the reader that one’s intuition can be a poor guide, when judging how MCSP^A and MCSP^B compare to each other, for given oracles A and B . For instance, it is known that MCSP^{SAT} ZPP-Turing reduces to MCSP^{QBF} [3], but no deterministic reduction is known. Similarly, no efficient reduction of any sort is known between MCSP and MCSP^{SAT} . Some of our theorems derive consequences from the assumption that MCSP^{SAT} is hard for some complexity class under AC^0 reductions. Although one might suspect that this is a weaker hypothesis than assuming that MCSP is hard for the same complexity class under AC^0 reductions – certainly the best upper bound for MCSP^{SAT} is worse than the best known upper bound for MCSP – nonetheless we are not able to derive the same consequences assuming only that MCSP is hard. For essentially all time- and space-bounded complexity classes \mathcal{C} that contain PSPACE, $\text{MCSP}^{\mathcal{C}}$ is complete for \mathcal{C}/poly under P/poly reductions [3, 6], but uniform reductions are known only for two cases [3]: when $\mathcal{C} = \text{PSPACE}$ (MCSP^{QBF} is complete for PSPACE under ZPP reductions) and when $\mathcal{C} = \text{EXP}$ (MCSP^{EXP} is complete for EXP under NP-Turing reductions).

2.1 Succinct Problems

The study of succinct encodings of computational problems was introduced by [9, 16], and has been studied since then by [18, 7], among others. Succinct encodings play an important role in the proofs of our main results.

► **Definition 4.** Given a language L , we define the succinct version of L (denoted $\text{succ}.L$) to be the language $\{C \mid \text{tt}(C) \in L\}$ where C is a Boolean Circuit and $\text{tt}(C)$ is the truth-table for C .

It will be necessary for us to consider “succinctly-presented” problems, where the circuit that constitutes the succinct description is itself an *oracle* circuit:

► **Definition 5.** Given a language L and an oracle A , we define the A -succinct version of L (denoted $A\text{-succ}.L$) to be the language $\{C \mid \text{tt}(C) \in L\}$ where C is a Boolean Circuit with oracle gates, and $\text{tt}(C)$ is the truth-table for C , when it is evaluated with oracle A . If $A = \emptyset$, we denote this language as $\text{succ}.L$.

The typical situation that arises is that the succinct version of a problem A has exponentially greater complexity than A . In particular, this happens when A is complete for a complexity class under “logtime reductions”.

► **Definition 6.** We say that a function f can be computed in logarithmic time if there exists a random-access Turing machine that, given (x, i) , computes the i th bit $f(x)$ in time $O(\log |x|)$.

Building on prior work of [16, 9, 18], Balcázar, Lozano, and Torán presented a large list of complexity classes $(\mathcal{C}_1, \mathcal{C}_2)$, where \mathcal{C}_1 is defined in terms of some resource bound $B(n)$ and \mathcal{C}_2 is defined in the same way, with resource bound $B(2^n)$, such that if a set A is complete for \mathcal{C}_1 under logtime reductions, then $\text{succ}.A$ is complete for \mathcal{C}_2 under polynomial-time many-one reductions [7].

Somewhat surprisingly, the complexity of succ.MCSP appears *not* to be exponentially greater than that of MCSP. (Related observations were made earlier by Williams [19].)

► **Theorem 7.** $\text{succ.MCSP} \in \Sigma_2^P$

Proof. We present an algorithm in Σ_2^P that decides succ.MCSP . Given an instance of succinct MCSP C , note that $C \in \text{succ.MCSP}$ iff z is a string of the form $(f, s) \in \text{MCSP}$, where $z = \text{tt}(C)$. By definition, $|z|$ must be a power of 2, say $|z| = 2^r$, and $|f|$ must also be a power of 2, say $|f| = 2^m$ for some $m < r$. Note also that if $s > |f| = 2^m$, then (f, s) should obviously be accepted, since every m -variate Boolean function has a circuit of size 2^m . To be precise, we will choose one particular convention for encoding the pair (f, s) ; other reasonable conventions will also yield a Σ_2^P upper bound. Let us encode (f, s) as a string of length 2^{m+1} , where the first 2^m bits give the truth table for f , and the second 2^m bits give s in binary. Note that this means that C has $m + 1$ input variables, and hardwiring the high-order input bit of C to 0 results in a circuit C' for f (of size at most $|C|$).

Using this encoding, the “interesting” instances (f, s) are of the form where the second half of the string is all zeros, except possibly for the low-order m bits (encoding a number $s \leq 2^m = |f|$). The low-order m bits can be computed deterministically in polynomial time, given C , by evaluating C on inputs $1^{m+1-\log m} 0^{\log m}, 1^{m+1-\log m} 0^{-1+\log m} 1, \dots, 1^{m+1}$. Let the number encoded by the low-order m bits be s' . Then C (an encoding of (f, s)) is in succ.MCSP iff

- there is some bit position j corresponding to one of the high-order $2^m - m$ bits of s such that $C(j) = 1$, or
- there exists a circuit D of size at most s' such that, for all i , $D(i) = C'(i)$ and for all bit positions j corresponding to one of the high-order $2^m - m$ bits of s , $C(j) = 0$ (and thus $s = s'$).

It is easily seen that this can be checked in Σ_2^p . ◀

Because this proof relativizes, we obtain:

► **Corollary 8.** *Let A and B be oracles such that $B \leq_T^p A$. Then $B\text{-succ.MCSP}^A$ is in $(\Sigma_2^p)^A$.*

Proof. We use the same encoding as in Theorem 7. Thus, an oracle circuit C encoding an instance (f, s) (where f is an m -ary function) has $m + 1$ input variables, and hardwiring the high-order input bit of C to 0 results in an oracle circuit C' (with oracle B) for f (of size at most $|C|$). But if $B \leq_T^p A$, then this also gives us an oracle circuit C'' (with oracle A) for f (of size at most $|C|^k$ for some k), where we can obtain C'' from C in polynomial time.

Then C (an encoding of (f, s)) is in $B\text{-succ.MCSP}^A$ iff

- there is some bit position j corresponding to one of the high-order $2^m - m$ bits of s such that $C^B(j) = 1$, or
- there exists a circuit D of size at most s' such that, for all i , $D^A(i) = C''^A(i)$ and for all bit positions j corresponding to one of the high-order $2^m - m$ bits of s , $C^B(j) = 0$ (and thus $s = s'$).

It is easily seen that this can be checked in $(\Sigma_2^p)^A$. ◀

An analogous result also holds for MKTP^A.

► **Theorem 9.** *Let A and B be oracles such that $B \leq_T^p A$. Then $B\text{-succ.MKTP}^A$ is in $(\Sigma_2^p)^A$.*

2.2 Constant-Depth Reductions

► **Proposition 10.** *Suppose that f is a uniform AC^0 reduction from a problem A to a problem B . Let C be an instance of succ.A . Then, the language $\{(C, i) \mid \text{the } i\text{th bit of } f(tt(C)) \text{ is } 1\}$ is in LTH (the linear-time hierarchy).*

Proof. Consider the unary version of the above language: $\{1^{(C,i)} \mid \text{the } i\text{th bit of } f(tt(C)) \text{ is } 1\}$; we claim that this language is in uniform AC^0 . To see this, note that after computing the length of the input (in binary), and thus obtaining a description of C (of length $\log n$), an AC^0 algorithm can compute each bit of $tt(C)$. For instance, the i th bit of $tt(C)$ can be computed by guessing a bit vector of length $\log n$ recording the value of each gate of C on input i , and then verifying that all of the guessed values are consistent. Once the bits of $tt(C)$ are available, then the AC^0 algorithm computes $f(tt(C))$.

The result is now immediate, from [5, Proposition 5], which shows that the rudimentary languages (that is, the languages in the linear-time version LTH of the polynomial-time hierarchy PH) are precisely the sets whose unary encodings are in Dlogtime-uniform AC^0 . ◀

By an entirely analogous argument, we obtain:

► **Proposition 11.** *Suppose that f is a uniform TC^0 reduction from a problem A to a problem B . Let C be an instance of succ.A . Then, the language $\{(C, i) \mid \text{the } i\text{th bit of } f(tt(C)) \text{ is } 1\}$ is in CH.*

3 Main Results

3.1 Conditional collapses and separations of complexity classes

Our first theorem shows that significant conclusions follow if MCSP is hard for P under AC^0 reductions.

► **Theorem 12.** *If there is any set A in the polynomial hierarchy such that $MCSP^A$ (or $MKTP^A$) is hard for P under AC^0 reductions, then $P \neq NP$.*

Proof. We present only the proof for $MCSP^A$; the proof for $MKTP^A$ is identical. Suppose that $P = NP$ and $MCSP^A$ is hard for P under AC^0 reductions. Thus, there is a family $\{C_n\}$ of AC^0 circuits reducing SAT to $MCSP^A$, such that $C_n(\phi) = f(\phi)$, where f is the reduction function and ϕ is an instance of SAT.

Now we claim that $\text{succ.SAT} \leq_m^p \text{succ.MCSP}^A$. To see this, consider an instance D of succ.SAT (that is, a circuit D on n variables that, when given input i , outputs the i th bit of a SAT instance of size 2^n). This problem has been shown to be complete for NEXP[15]. By Proposition 10, we have that the language $\{(D, i) \mid \text{the } i\text{th bit of } f(\text{tt}(D)) \text{ is } 1\}$ is in PH. By our assumption that $P = NP$, we have that this language is in P . Let E_m be a family of circuits deciding this language. The function that takes input D and outputs $E_{|(D,n)|}$ (with D hardwired in) is a polynomial-time reduction from succ.SAT to succ.MCSP^A , which is in $(\Sigma_2^p)^A$, by Corollary 8. Since $A \in P$ (by our assumption that $P = NP$), we have that $\text{NEXP} \subseteq P$, which is a contradiction. ◀

► **Corollary 13.** *If there is any set $A \in \text{CH}$ such that $MCSP^A$ (or $MKTP^A$) is hard for P under TC^0 reductions, then $P \neq \text{PP}$.*

Due to space limitations, this proof and several others are omitted. A more complete version may be found on ECCC.

► **Corollary 14.** *Suppose that MCSP (or MKTP) is hard for P under logspace many-one reductions. Then $P \neq \text{PSPACE}$.*

► **Theorem 15.** *Suppose that $MCSP^{\text{EXP}}$ is hard for NP under polynomial-time reductions. Then $\text{NEXP} = \text{EXP}$.*

Proof. Let f be the reduction taking an instance of SAT to an instance of $MCSP^{\text{EXP}}$. We construct a reduction from succ.SAT to $B\text{-succ.MCSP}^{\text{EXP}}$ for some $B \in \text{EXP}$.

Consider the language $L = \{(C, i) \mid \text{the } i\text{th bit of } f(\phi_C) \text{ is } 1\}$, where ϕ_C is the formula described by the circuit C , viewed as an instance of succ.SAT with n input variables. We can decide L in exponential time because we can write down ϕ_C in exponential time, and then we can compute $f(\phi_C)$ in exponential time because f is a poly-time reduction on an exponentially large instance. Let $\{D_m\}$ be a family of oracle circuits for L , using an oracle for an EXP-complete language B . Thus the mapping $C \mapsto D_{|C|+n}$ is a polynomial-time reduction from succ.SAT to $B\text{-succ.MCSP}^{\text{EXP}}$, which is in $(\Sigma_2^p)^{\text{EXP}} = \text{EXP}$ (see, e.g., [6, Theorem 24]), and thus $\text{EXP} = \text{NEXP}$. ◀

► **Corollary 16.** *Consider Levin's time-bounded Kolmogorov complexity measure Kt [12]. Suppose that $\{(x, i) : Kt(x) \leq i\}$ is hard for NP under polynomial-time reductions. Then $\text{NEXP} = \text{EXP}$.*

► **Theorem 17.** *If $MCSP^{\text{QBF}}$ or $MKTP^{\text{QBF}}$ is hard for NP under logspace reductions, then $\text{NEXP} = \text{PSPACE}$.*

► **Corollary 18.** *If MCSP^{QBF} (or MKTP^{QBF}) is hard for P under logspace reductions, then $\text{EXP} = \text{PSPACE}$.*

Proof. The proof is identical to the proof of the preceding theorem, with NP replaced by P , and with NEXP replaced by EXP . ◀

If we carry out a similar argument, replacing NP with PSPACE , we obtain the contradiction $\text{EXPSPACE} = \text{PSPACE}$, yielding the following.

► **Corollary 19.** *Neither MCSP^{QBF} nor MKTP^{QBF} is hard for PSPACE under logspace reductions.*

3.2 Impossibility of uniform AC^0 reductions

► **Theorem 20.** *For any language A that is hard for PH under P/poly reductions, MCSP^A is not hard for TC^0 under uniform AC^0 reductions.*

The theorem will follow from the next lemma. Recall that LTH (linear-time hierarchy) stands for the linear-time version of the polynomial-time hierarchy PH .

► **Lemma 21.** *Suppose that, for some language A , MCSP^A is TC^0 -hard under uniform AC^0 reductions. Then $\text{LTH} \not\subseteq \text{io-SIZE}^A[2^{\Omega(n)}]$.*

Proof. It is shown in [1, Theorems 5.1 and 6.2] that if a set is hard for any class \mathcal{C} that is closed under TC^0 reductions under uniform AC^0 reductions, then it is hard under length-increasing (uniform AC^0)-uniform NC^0 reductions. (Although Theorems 5.1 and 6.2 in [1] are stated only for sets that are *complete* for \mathcal{C} , they do hold also assuming only hardness [2], using exactly the same proofs.) Here, the notion “ AC^0 -uniform NC^0 ” refers to NC^0 circuits with the property that direct connection language $DCL = \{(n, t, i, j) \mid \text{gate } i \text{ of } F_n \text{ has type } t \text{ and has an edge leading from gate } j\}$ with n in unary is in $\text{Dlogtime-uniform AC}^0$.

Hence, if MCSP^A is hard for TC^0 under uniform AC^0 reductions, then we get that PARITY is reducible to MCSP^A under a length-increasing (uniform AC^0)-uniform NC^0 reduction. Such a reduction R maps PARITY instances $x \in \{0, 1\}^n$ to MCSP^A instances (f, s) , where f is the truth table of a Boolean function, $f \in \{0, 1\}^m$, for some m such that $n \leq m \leq n^{O(1)}$, and $0 \leq s \leq m$ is the size parameter in binary, and hence $|s| \leq O(\log n)$.

Being the output of an NC^0 reduction, the binary string s depends on at most $O(\log n)$ bits in the input string x . Imagine fixing these bits in x to achieve the minimum value of the parameter s . Denote this minimum value of s by v . (We do not need for v to be efficiently computable in any sense.) We get a *nonuniform* NC^0 reduction from PARITY on $n - O(\log n) \geq n/2$ bit strings to MCSP^A with the size parameter fixed to the value v .

► **Claim 22.** *For any language A and any $0 \leq v \leq m$, MCSP^A on inputs $f \in \{0, 1\}^m$, with the size parameter fixed to v , is solved by a DNF formula of size $O(m \cdot 2^{v^2 \log v})$.*

Proof of Claim 22. Each A -oracle circuit of size v on $\log m$ inputs can be described by a binary string of length at most $O(v^2 \log v)$, since each of v gates has at most v inputs. Thus, there are at most $2^{O(v^2 \log v)}$ Boolean functions on $\log m$ inputs that are computable by A -oracle circuits of size at most v . Checking if any one of these truth tables equals to the input truth table f can be done by a DNF, where we take an OR over all easy functions, and for each easy function we use an AND gate to check equality to the input f . ◀

We conclude that PARITY on $n/2$ -bit strings is solvable by AC^0 circuits of depth 3 and size $O(m \cdot 2^{v^2 \log v})$. Indeed, each bit of the truth table f is computable by an NC^0 circuit, and hence by a DNF (and a CNF) of constant size. Plugging in these DNFs (or CNFs) for the bits of f into the DNF formula from Claim 22 yields the required depth-3 AC^0 circuit for PARITY on inputs of length at least $n/2$.

Next, since PARITY on m -bit strings requires depth-3 AC^0 circuits of size at least $2^{\Omega(\sqrt{m})}$ [10], we get that $v \geq n^{1/5}$. Hence, on input 0^n , our *uniform* NC^0 reduction produces (f, s) where f is the truth table of a Boolean function on r -bit inputs that has A -oracle circuit complexity at least $v \geq n^{1/5} \geq 2^{\epsilon r}$, for some $\epsilon > 0$.

Finally, since the NC^0 reduction is (uniform AC^0)-uniform, we get that the Boolean function whose truth table is f is computable in LTH. ◀

Proof of Theorem 20. Towards a contradiction, suppose that $MCSP^A$ is TC^0 -hard under uniform AC^0 reductions. Then, by Lemma 21, there is a language $L \in PH$ that requires A -oracle circuit complexity $2^{\Omega(n)}$ almost everywhere. However, since A is PH-hard under P/poly reductions, we get that $L \in SIZE^A[\text{poly}]$. A contradiction. ◀

► **Corollary 23.** $MCSP^{\oplus P}$ is not TC^0 -hard under uniform AC^0 reductions.

► **Corollary 24.** Suppose that, for some oracle A , $MCSP^A$ is TC^0 -hard under uniform AC^0 reductions. Then $NP^A \not\subseteq SIZE^A[\text{poly}]$.

► **Remark.** Murray and Williams [13] prove results similar to (and implied by) our Lemma 21 and Corollary 24 for the case of the empty oracle $A = \emptyset$. Namely, they show that if $MCSP$ is NP-hard under uniform AC^0 reductions, then $NP \not\subseteq P/\text{poly}$ and $E \not\subseteq \text{io-SIZE}[2^{\Omega(n)}]$.

Finally, we observe that the ideas in our proof of Lemma 21 yield an alternate proof of the result by Murray and Williams [13] that PARITY is not reducible to $MCSP$ via “local” $O(n^{1/2-\epsilon})$ -time reductions. We prove the version for polylogtime-uniform NC^0 reductions, but the same argument applies also to the “local” reductions of [13].

► **Theorem 25** ([13]). *There is no polylogtime-uniform NC^0 reduction from PARITY to $MCSP$.*

Proof. Suppose there is such a reduction. Similarly to the proof of Lemma 21, we conclude that this NC^0 reduction maps 0^n to an $MCSP$ instance (f, s) where f is the truth table of a Boolean function on $r := O(\log n)$ inputs that requires exponential circuit size $s \geq 2^{\Omega(r)}$. On the other hand, since our NC^0 reduction is polylogtime-uniform, the Boolean function with the truth table f is computable in P, and hence in $SIZE[\text{poly}]$. A contradiction. ◀

3.3 Gap $MCSP$

For $0 < \epsilon < 1$, we consider the following *gap version* of $MCSP$, denoted ϵ -gap $MCSP$: Given (f, s) , output ‘Yes’ if f requires circuits of size at least s , and output ‘No’ if f can be computed by a circuit of size at most $(1 - \epsilon)s$.

For $\alpha : \mathbb{N} \rightarrow \mathbb{R}^+$, call a mapping $R : \{0, 1\}^n \rightarrow \{0, 1\}^m$ α -stretching if $m \leq \alpha(n) \cdot n$. We will prove that there is no n^δ -stretching nonuniform AC^0 reduction from PARITY to ϵ -gap $MCSP$, for certain parameters $0 < \epsilon, \delta < 1$. First, we rule out nonuniform NC^0 reductions.

► **Theorem 26.** *For every $n^{-1/6} < \epsilon < 1$ and for every constant $\delta < 1/30$, there is no n^δ -stretching (nonuniform) NC^0 reduction from PARITY to ϵ -gap $MCSP$.*

Proof. Towards a contradiction, suppose there is an n^δ -stretching NC^0 reduction from PARITY on inputs $x \in \{0, 1\}^n$ to ϵ -gap MCSP instances (f, s) . Fix to zeros all $O(\log n)$ bit positions in the string x that determine the value of the size parameter s . As in the proof of Lemma 21, we get an NC^0 reduction from PARITY on at least $n/2$ bits y to the ϵ -gap MCSP instance with the size parameter fixed to some value $v \geq n^{1/5}$.

By our assumption, $|f| \leq n \cdot n^\delta$. Since each bit of f is computable by an NC^0 circuit, we get that each bit of f depends on at most c bits in the input y . The total number of pairs (i, j) where f_i depends on bit y_j is at most $c \cdot |f|$. By averaging, there is a bit y_j , $1 \leq j \leq n/2$, that influences at most $c|f|/(n/2) \leq 2cn^\delta$ bit positions in the string f .

Fix y so that all bits are 0 except for y_j (which is set to 1). This y is mapped by our NC^0 reduction to the truth table f' that is computable by a circuit of size at most $(1 - \epsilon)v$. On the other hand, flipping the bit y_j to 0 forces the reduction to output a truth table f'' of circuit complexity at least v . But, y_j influences at most $2cn^\delta$ positions in f' , and so the circuit complexity of f'' differs from that of f' by at most $O(n^\delta \log n)$ gates (as we can just construct a “difference” circuit of that size that is 1 on the at most $2cn^\delta$ affected positions of f'). We get $\epsilon v \leq O(n^\delta \log n)$, which is impossible when $\delta < 1/30$. ◀

Now we extend Theorem 26 to the case of nonuniform AC^0 reductions.

► **Theorem 27.** *For every $n^{-1/7} < \epsilon < 1$ and for every constant $\delta < 1/31$, there is no n^δ -stretching (nonuniform) AC^0 reduction from PARITY to ϵ -gap MCSP.*

Proof. Towards a contradiction, suppose there is a n^δ -stretching AC^0 reduction from PARITY on n -bit strings to the ϵ -gap MCSP. We will show that this implies the existence of an NC^0 reduction with parameters that contradict Theorem 26 above.

► **Claim 28.** *For every constant $\gamma > 0$, there exist a constant $a > 0$ and a restriction of our AC^0 circuit satisfying the following: (1) each output of the restricted circuit depends on at most a inputs, and (2) the number of unrestricted variables is at least $n^{1-\gamma}$.*

Proof of Claim 28. Recall that a random p -restriction of n variables x_1, \dots, x_n is defined as follows: for each $1 \leq i \leq n$, with probability p , leave x_i unrestricted, and with probability $1-p$, set x_i to 0 or 1 uniformly at random. By Håstad’s Switching Lemma [10], the probability that a given CNF on n variables with bottom fan-in at most t does not become a decision tree of depth at most r after being hit with a random p -restriction is at most $(5pt)^r$.

For an AC^0 circuit of size n^k and depth d , set $p := (5a)^{-1}n^{-2k/a}$ for some constant $a > 0$ to be determined. Applying this random p -restriction d times will reduce the original circuit to a decision tree of depth a with probability at least $1 - dn^k(5pa)^a > 3/4$. The expected number of unrestricted variables at the end of this process is $p^d n \geq \Omega(n/n^{2kd/a}) = \Omega(n/n^{\gamma'})$, for $\gamma' := 2kd/a$. By Chernoff bounds, the actual number of unrestricted variables is at least $1/2$ of the expectation with probability at least $3/4$.

Thus, with probability at least $1/2$, we get a restriction that makes the original AC^0 circuit into an NC^0 circuit on at least $n/n^{2\gamma'}$ variables, where each output of the new circuit depends on at most a input variables. Setting $\gamma := 2\gamma'$, we get that $a = (4kd)/\gamma$. ◀

We get an NC^0 reduction from PARITY on $n' := n^{1-\gamma}$ variables to ϵ -gap MCSP. This reduction is at most $(n')^{(\delta+\gamma)/(1-\gamma)}$ -stretching. Choose $0 < \gamma < (1/31)^2$ so that $(\delta + \gamma)/(1 - \gamma) < 1/30$, and $\epsilon > n^{-1/7} > (n')^{-1/6}$. Finally, appeal to Theorem 26 for contradiction. ◀

4 Generalizations

Theorem 12 gives consequences of MCSP being hard for P. The property of P that is exploited in the proof is that the polynomial hierarchy collapses to P if $\text{NP} = \text{P}$. (This is required, so that we can efficiently a circuit that computes bits of the reduction, knowing only that it is in the polynomial hierarchy.)

The next theorem formalizes this observation:

► **Theorem 29.** *Let \mathcal{C} be any class such that if $\text{NP} = \mathcal{C}$, then $\text{PH} = \mathcal{C}$. If there is a set $A \in \text{PH}$ that is hard for \mathcal{C} under \leq_T^P reductions such that MCSP^A (or MKTP^A) is hard for \mathcal{C} under uniform AC^0 reductions, then $\text{NP} \neq \mathcal{C}$.*

► **Corollary 30.** *Let A be any set in the polynomial hierarchy. If MCSP^A (or MKTP^A) is hard for $\text{AC}^0[6]$ under AC^0 reductions, then $\text{AC}^0[6] \neq \text{NP}$.*

Recall that SZK denotes the class of languages with Statistical Zero-Knowledge proofs.

► **Corollary 31.** *Let A be any set in the polynomial hierarchy that is hard for SZK under \leq_T^P reductions. If MCSP^A is hard for SZK under AC^0 reductions, then $\text{SZK} \neq \text{NP}$.*

Proof. SZK is closed under complementation [14]. Thus if NP is equal to the class of languages in SZK, then $\text{coNP} = \text{NP} = \text{SZK}$ and PH collapses to SZK. Thus SZK satisfies the hypothesis of Theorem 29. ◀

Similarly, we can state the following theorem about TC^0 reductions.

► **Theorem 32.** *Let \mathcal{C} be any class such that if $\text{PP} = \mathcal{C}$, then $\text{CH} = \mathcal{C}$. If there is a set $A \in \text{CH}$ that is hard for \mathcal{C} under \leq_T^P reductions such that MCSP^A (or MKTP^A) is hard for \mathcal{C} under uniform TC^0 reductions, then $\text{PP} \neq \mathcal{C}$.*

Fenner, Fortnow, and Kurtz [8] introduced several complexity classes, including SPP and WPP that are “low for PP”, in the sense that $\text{PP} = \text{PP}^{\text{SPP}} = \text{PP}^{\text{WPP}}$. Thus we obtain the following corollary:

► **Corollary 33.** *Let A be any set in the counting hierarchy that is hard for WPP under \leq_T^P reductions. If MCSP^A is hard for WPP (or SPP) under uniform TC^0 reductions, then $\text{WPP} \neq \text{PP}$ (respectively $\text{SPP} \neq \text{PP}$).*

5 Discussion

The contrast between Theorem 12 and Corollary 18 is stark. Theorem 12 obtains a very unsurprising consequence from the assumption that MCSP is hard for P under a very restrictive class of reductions, while Corollary 18 obtains a very unlikely collapse from the assumption that the apparently much harder problem MCSP^{QBF} is hard for P under a much less restrictive class of reductions. Yet, the absence of any known efficient reduction from MCSP to MCSP^{QBF} means that we have been unable to obtain any *unlikely* consequences by assuming that MCSP is hard for P. We believe that it should be possible to provide evidence that MCSP is not hard for P, and we pose this as an open question for further research.

Acknowledgments This research was supported in part by NSF grants CCF-1064785 and CCF-1423544, and by an NSERC Discovery Grant. Some of this work was carried out at the 2014 Dagstuhl Workshop on Algebra in Computational Complexity (Dagstuhl Seminar 14391). We also acknowledge helpful discussions with Ryan Williams, Chris Umans, Manindra Agrawal, and Mitsunori Ogihara.

References

- 1 Manindra Agrawal. The isomorphism conjecture for constant depth reductions. *Journal of Computer and System Sciences*, 77(1):3–13, 2011.
- 2 Manindra Agrawal. Personal Communication, 2014.
- 3 Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM Journal on Computing*, 35(6):1467–1493, 2006.
- 4 Eric Allender and Bireswar Das. Zero knowledge and circuit minimization. In *Mathematical Foundations of Computer Science (MFCS)*, volume 8635 of *Lecture Notes in Computer Science*, pages 25–32. Springer, 2014.
- 5 Eric Allender and Vivek Gore. On strong separations from AC^0 . In Jin-Yi Cai, editor, *Advances in Computational Complexity Theory*, volume 13 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 21–37. AMS Press, 1993.
- 6 Eric Allender, Michal Koucký, Detlef Ronneburger, and Sambuddha Roy. The pervasive reach of resource-bounded Kolmogorov complexity in computational complexity theory. *Journal of Computer and System Sciences*, 77:14–40, 2010.
- 7 José L Balcázar, Antoni Lozano, and Jacobo Torán. The complexity of algorithmic problems on succinct instances. In *Computer Science*, pages 351–377. Springer, 1992.
- 8 Stephen A. Fenner, Lance Fortnow, and Stuart A. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48(1):116–148, 1994.
- 9 Hana Galperin and Avi Wigderson. Succinct representations of graphs. *Information and Control*, 56(3):183–198, 1983.
- 10 Johan Håstad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 6–20, 1986.
- 11 Valentine Kabanets and Jin-Yi Cai. Circuit minimization problem. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 73–79. ACM, 2000.
- 12 Leonid Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Information and Control*, 61:15–37, 1984.
- 13 Cody Murray and Ryan Williams. On the (non) NP-hardness of computing circuit complexity. In *Electronic Colloquium on Computational Complexity (ECCC)*, 2014. TR14-164.
- 14 Tatsuaki Okamoto. On relationships between statistical zero-knowledge proofs. *Journal of Computer and System Sciences*, 60(1):47–108, 2000.
- 15 Christos H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.
- 16 Christos H. Papadimitriou and Mihalis Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71(3):181–185, 1986.
- 17 Boris A. Trakhtenbrot. A survey of Russian approaches to perebor (brute-force searches) algorithms. *IEEE Annals of the History of Computing*, 6(4):384–400, 1984.
- 18 Klaus W Wagner. The complexity of combinatorial problems with succinct input representation. *Acta Informatica*, 23(3):325–356, 1986.
- 19 Ryan Williams. <http://cstheory.stackexchange.com/questions/10320/succinct-problems-in-mathsf/10546#10546>, 2012.

Graph Searching Games and Width Measures for Directed Graphs

Saeed Akhoondian Amiri¹, Łukasz Kaiser², Stephan Kreutzer¹, Roman Rabinovich^{*1}, and Sebastian Siebertz¹

- 1 Logic and Semantic, Technische Universität Berlin
Ernst-Reuter-Platz 7, 10587 Berlin, Germany
{saeed.akhoondianamiri,stephan.kreutzer,roman.rabinovich,sebastian.siebertz}@tu-berlin.de
- 2 LIAFA, CNRS & Université Paris Diderot[†], lukaszkaiser@google.com

Abstract

In cops and robber games a number of cops tries to capture a robber in a graph. A variant of these games on undirected graphs characterises tree width by the least number of cops needed to win. We consider cops and robber games on digraphs and width measures (such as DAG-width, directed tree width or D-width) corresponding to them. All of them generalise tree width and the game characterising it.

For the DAG-width game we prove that the problem to decide the minimal number of cops required to capture the robber (which is the same as deciding DAG-width), is PSPACE-complete, in contrast to most other similar games. We also show that the cop-monotonicity cost for directed tree width games cannot be bounded by any function. As a consequence, D-width is not bounded in directed tree width, refuting a conjecture by Safari.

A large number of directed width measures generalising tree width has been proposed in the literature. However, only very little was known about the relation between them, in particular about whether classes of digraphs of bounded width in one measure have bounded width in another. In this paper we establish an almost complete order among the most prominent width measures with respect to mutual boundedness.

1998 ACM Subject Classification G.2.m Graph Theory

Keywords and phrases cops and robber games, directed graphs, DAG-width

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.34

1 Introduction

Graph searching games, also known as *cops and robber* or *pursuit-evasion* games, are an important type of games on graphs and digraphs studied intensively in the literature. While there are many different forms of graph searching games, the basic idea is always that a number of *searchers* tries to find or catch a *fugitive* hiding in the vertices or edges of a graph or digraph. See Section 2 for details of the games used in this paper and see [15] for an introduction and [12] for a comprehensive survey of graph searching games.

Graph searching games have originally been introduced to model the search of rescuers trying to find a miner lost in a mine after some accident. Any graph searching game defines a natural graph invariant assigning to every graph the minimal number of cops needed to

* Partially supported by ESF, <http://www.esf.org>.

† Currently at Google Inc.



© Saeed A. Amiri, Łukasz Kaiser, Stephan Kreutzer, Roman Rabinovich, and Sebastian Siebertz; licensed under Creative Commons License CC-BY

32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).

Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 34–47



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



guarantee capture of the robber. It has subsequently emerged that these graph invariants are closely related to width measures such as directed or undirected *tree* or *path width* studied in graph structure theory (see e.g. [8, 14, 2]).

In particular, a type of strategies for the cops called *monotone* strategies often corresponds exactly to decompositions such as *tree* or *path decompositions* and hence concepts such as *tree width* etc. can equivalently be defined by a particular variant of graph searching games. In this paper we will therefore treat strategies for the cops and corresponding graph or digraph decompositions equivalently, emphasising either of the two views whenever it seems more appropriate.

Graph searching games can be defined on undirected or directed graphs. On undirected graphs, the *visible robber game* defines exactly tree width, the *invisible robber game* defines path width and a variation of the invisible game called the *inert robber game* again defines tree width. These games can be generalised to digraphs in two different ways. In this way we naturally obtain games on digraphs corresponding to directed width measures such as *directed tree width* [14], *DAG-width* [4], *Kelly-width* [13] or *directed path width* [2]. We will therefore refer to these games as the *directed tree width games*, *DAG-width games* etc.

Structural width measures such as tree and path width have found important applications in algorithms and complexity theory. In view of the correspondence between graph searching games and such structural decomposition based width measures, a natural question arising is the problem of determining for a given graph or digraph the minimal number of cops that guarantees to capture the robber in a particular game variant. For most game variants including tree or path width games (both directed and undirected) one can show that this problem is in NP: decompositions, i.e. monotone winning strategies for the cops (see below), are of size polynomial in the input graph and one can therefore simply guess such a strategy and verify the correctness of the guess. In this way for most game variants relevant in this context it was shown that they can be decided in NP and they are usually NP-complete. Only the complexity of DAG-width games was left as an open problem as there the corresponding decompositions are DAG-like and hence not obviously seen to be polynomial.

Surprisingly, in this paper we show that deciding the DAG-width of a digraph is not only not in NP (under standard complexity theoretical assumptions), it is in fact PSPACE-complete and therefore exhibits the worst case complexity of such games. This result is quite unexpected and especially surprising as such a high complexity was to date only exhibited by a form of graph searching games called *domination games* (see [10, 9, 16]). In these games, each cop not only occupies his current vertex but a whole neighbourhood of fixed radius, which essentially allows to simulate set quantification making the problem PSPACE-complete.

The DAG-width game, however, is a straight forward translation of the NP-complete game for tree width to digraphs and to the best of our knowledge this is the only graph searching game with the usual capturing condition that exhibits such a complexity.

As a consequence of the proof technique used to prove this result we also show that there are classes of graphs for which any DAG decomposition of optimal width must contain a super polynomial number of bags. (If $\text{NP} \neq \text{PSPACE}$, this would follow from the previous result, but we show this unconditionally.) Furthermore, we obtain that there cannot be a polynomial time approximation algorithm for DAG-width with only an additive error.

As explained above, the cop number of graph searching game variants is very closely related, and often equivalent, to standard width measures for graphs and digraphs. In the literature on digraph width measures a significant number of width measures have been proposed as directed analogue of undirected tree width. Among these are directed tree width [14], DAG-width [4], Kelly-width [13] and D-width [23]. Furthermore, there are some

game variants such as cop-monotone directed tree width games and non-monotone DAG-width games, for which no corresponding width measure has been defined. The obvious question is how these different measures compare to each other, i.e. whether a class of digraphs of bounded width in one measure has bounded width in another. For some pairs of digraph width measures the relation has been determined, but to date there is no clear picture. In particular, the relation between DAG-width, Kelly-width, D-width and cop-monotone directed tree width games is not known. As the second main result of this paper we establish a nearly complete order among these width measures. The most difficult part hereby is to show that any class of digraphs of bounded Kelly-width also has bounded DAG-width.

A crucial concept in graph searching games is *monotonicity*. A strategy for the cops is *robber-monotone* if vertices unavailable for the robber at a position of a play never become available later on and it is *cop-monotone* if the cops never go back to a vertex they have left before. Monotone strategies are particularly well behaved and in fact, cop-monotone strategies are very similar to decompositions such as tree or path decompositions. A highly desirable property of a particular variant of graph searching games therefore is that the number of cops needed to catch the robber on a graph or digraph G with a monotone strategy is the same (or at least bounded in) the number of cops needed with any strategy. The number of extra cops needed for monotone strategies is called the *cop- or robber-monotonicity cost* of the game.

This *monotonicity* problem has driven the field of graph searching games from the very beginning, see e.g. [18, 5, 24, 6, 2, 11, 25, 14, 1, 27, 26, 12, 19, 7]. For undirected graphs, the monotonicity problem is by now well understood and most natural graph searching variants are indeed monotone. For directed graphs, the situation is very different. The games corresponding to directed path width are cop- and robber-monotone [2]. The games for directed tree width are not robber- and not cop-monotone, but a robber-monotone strategy requires at most three times the number of cops [14]. However, many important problems regarding monotonicity on directed graphs are still wide open.

Among the most important open problems in this respect are the questions whether the cop-monotonicity cost for the game corresponding to directed tree width can be bounded by any function and whether the robber-monotonicity cost for the games corresponding to DAG-width or Kelly-width can be bounded. In [23], it has been conjectured that the cop-monotonicity cost for directed tree width games is bounded, but the problem was left open to date. In this paper we refute this conjecture by showing that there is a class of graphs such that on every digraph in this class, 4 cops have a robber-monotone winning strategy in the directed tree width game, but the number of cops needed for cop-monotone winning strategies is unbounded.

As a technical tool to show that DAG-width is bounded in Kelly-width we introduce another notion of monotonicity for DAG-width games that we call *weak monotonicity*. The core of the argument is to show that any weakly monotone strategy in the DAG-width game can be translated into a fully monotone strategy with only a quadratic increase in the number of cops. We can then show that strategies in the games corresponding to Kelly-width can be translated into weakly monotone strategies in the DAG-width game and hence into monotone strategies.

While this relation between Kelly and DAG-width is the most explicit application of this concept of weak monotonicity, we believe that weak monotonicity will have many more applications. In particular, as explained above, the outstanding open problem in the area of digraph searching games is the monotonicity for DAG-width (and Kelly-width) games. These games have been shown to be non-monotone in [17]. More precisely, in [17] it was

shown that there are classes of digraphs on which the cops need $4/3$ times as many cops for a monotone strategy than for an unrestricted strategy. However, all attempts to use the techniques developed in [17] to show that the monotonicity costs cannot be bounded by any constant, or any function at all, have failed. Our result on weak monotonicity proves that these attempts are doomed to fail as the non-monotone strategy used by cops in the examples in [17] is in fact weakly monotone. We therefore believe that weak monotonicity will prove to be a valuable step towards a solution of the monotonicity problem of DAG-width games. And indeed this was the original motivation for introducing weak monotonicity in [21]. It is worth mentioning that the weakly monotone DAG-width games have a corresponding decomposition. These weak DAG decompositions approximate DAG decompositions and always have size polynomial in the size of the graph.

Our contributions. The main results of this paper are the following.

- We show that deciding the DAG-width of a graph, or equivalently deciding the number of cops needed to win the corresponding monotone graph searching game, is PSPACE-complete.
- We show that there are graphs for which no DAG decomposition of polynomial size exist whose width is at most an additive constant away from the optimal width.
- We refute a conjecture by Safari [23] by showing that the cop-monotonicity costs for the graph searching games corresponding to directed tree width are unbounded. As a consequence, we obtain that D-width is not bounded by any function in the directed tree width. In fact, D-width is not even bounded by any function in the number of cops needed in the cop-monotone directed tree width game. Furthermore, we also show that D-width cannot even be bounded by any function in the DAG-width and in the Kelly-width.
- We show that DAG-width can be bounded by a quadratic function in the Kelly-width. Together with the previous results, we obtain an almost complete classification of the directed width measures proposed in the literature.

2 Preliminaries

We assume familiarity with basic concepts of graph theory and refer to [8] for background. All graphs in this paper are finite, directed and simple, i.e. they do not have loops or multiple edges between the same pair of vertices. Undirected graphs are digraphs with a symmetric edge relation. We write \bar{G} for the underlying undirected graph of G . If G is a graph, then $V(G)$ is its set of vertices and $E(G)$ is its set of edges. For a set $X \subseteq V(G)$ we write $G[X]$ for the subgraph of G induced by X and $G - X$ for $G[V(G) \setminus X]$. The set of vertices reachable from a set $V' \subseteq V(G)$ is denoted $\text{Reach}_G(V')$. If $V' = \{v\}$, we also write $\text{Reach}_G(v)$. A *strongly connected component* of a digraph G is a maximal subgraph C of G which is strongly connected, i.e. between any pair $u, v \in V(C)$ there are directed paths from u to v and from v to u . All components of digraphs considered in this paper will be strong and hence we simply speak of *components*.

2.1 Graph Searching Games

A graph searching game (also known as cops and robber game and pursuit-evasion game) is played on a graph G by a team of cops and a robber. The robber and each cop occupy a vertex of G . Hence, a current game position can be described by a pair (C, v) , where C is the set of vertices occupied by cops and v is the current robber position. At the beginning

the robber chooses an arbitrary vertex v and the game starts at position (\emptyset, v) . The game is played in rounds. In each round, from a position (C, v) the cops first announce their next move, i.e. the set $C' \subseteq V(G)$ of vertices that they will occupy next. Based on the triple (C, C', v) the robber chooses his new vertex v' . This completes a round and the play continues at position (C', v') . Variations of graph searching games are obtained by restricting the moves allowed for the cops and the robber. In all game variants considered here, from a position (C, C', v) , i.e. when the cops move from their current position C to C' and the robber is on v , the robber has exactly the same choice of moves from any vertex in the component of $G - C$ containing v . We will therefore describe game positions by a pair (C, R) , or a triple (C, C', R) , where C, C' are as before and R induces a component of $G - C$.

A graph searching game on G is specified by a tuple $\mathcal{G} = (\text{Pos}(G), \text{Moves}(G), \text{Mon})$, where $\text{Pos}(G)$ describes the set of possible positions, $\text{Moves}(G)$ the set of legal moves and Mon specifies the monotonicity condition used. In all game variants considered here, the set $\text{Pos}(G)$ of positions is $\text{Pos}_c \cup \text{Pos}_r$ where $\text{Pos}_c = \{(C, R) : C \subseteq V(G), R \subseteq V(G) \text{ induces a component of } G - C\}$ are cop positions and $\text{Pos}_r = \{(C, C', R) : C, C' \subseteq V(G) \text{ and } R \subseteq V(G) \text{ induces a component of } G - C\}$ are robber positions.

As far as legal moves are concerned, we distinguish between two different types of games, called *reachability* and *component* games. In both cases the cops moves are

$$\text{Moves}_c(G) := \{((C, R), (C, C', R)) : (C, R) \in \text{Pos}_c, (C, C', R) \in \text{Pos}_r\}.$$

The difference is in the definition of the set of possible robber moves.

Reachability game

In the *reachability game*, we define $\text{Moves}(G)$ as $\text{ReachMoves}(G)$, where

$$\begin{aligned} \text{ReachMoves}(G) := & \text{Moves}_c(G) \cup \{((C, C', R), (C', R')) : (C, C', R) \in \text{Pos}_r, \\ & (C', R') \in \text{Pos}_c \text{ and } R' \text{ is a component of } G - C' \text{ such that } R' \subseteq \text{Reach}_{G-(C \cap C')}(R)\}. \end{aligned}$$

In other words, the robber can run along any directed path in the digraph which does not contain a cop from $C \cap C'$ (i.e. one that remains on the board).

Component game

In the *component game*, we define $\text{Moves}(G)$ as $\text{CompMoves}(G)$, where

$$\begin{aligned} \text{CompMoves}(G) := & \text{Moves}_c(G) \cup \{((C, C', R), (C', R')) : (C, C', R) \in \text{Pos}_r, \\ & (C', R') \in \text{Pos}_c \text{ and } R' \text{ is a component of } G - C' \text{ such that } R \\ & \text{and } R' \text{ are subsets of the same component of } G - (C \cap C')\}. \end{aligned}$$

That means, in the component game, the robber can only run to a new vertex within the strongly connected component of $G - (C \cap C')$ that contains his current position.

Monotonicity

The component Mon is a set of finite plays. The cops win all plays $(C_0, R_0), (C_0, C_1, R_0), (C_1, R_1), \dots$ in Mon where $R_i = \emptyset$ for some i (and the play stops here) and the robber wins all other plays. Usually Mon describes cop- or robber-monotonicity: $\text{Mon} \in \{\text{cm}(G) \cup \text{rm}(G)\}$. A play $(C_0, R_0), (C_0, C_1, R_0), (C_1, R_1), \dots$ is

- in $\text{cm}(G)$, called *cop-monotone*, if for all $i, j, k \geq 0$ with $i < j < k$ we have $C_i \cap C_k \subseteq C_j$,

- in $\text{rm}(G)$, called *robber-monotone*, if $R_{i+1} \subseteq R_i$ for all i .

Cop-monotonicity means that the cops never reoccupy vertices. Robber-monotonicity means that once the robber cannot reach a vertex, he will never be able to reach it in the future. A strategy for the cops is *cop-* or *robber-monotone* if all plays consistent with that strategy are cop- or robber-monotone, respectively.

By combining reachability or component games with monotonicity conditions we obtain a range of different graph searching games. It follows immediately from the definition that on every digraph the cops have a winning strategy in each of the graph searching games defined above by simply placing a cop on every vertex. For a given digraph G , we are therefore interested in the minimal number k such that the cops have a winning strategy in which no cop position C_i contains more than k vertices.

► **Definition 2.1.** Let Fin be the set of all finite plays. For every digraph G and for

$$X \in \{\text{dtw}, \text{cmdtw}, \text{rmdtw}, \text{nmDAG}, \text{cmDAG}, \text{DAG}\}$$

let $cn_G(X)$ be the minimal number of cops that have a winning strategy in the game $\mathbb{G}_G(X)$ where

- $\mathbb{G}(\text{dtw}, G) := (\text{Pos}(G), \text{CompMoves}(G), \text{Mon} = \text{Fin})$,
- $\mathbb{G}(\text{cmdtw}, G) := (\text{Pos}(G), \text{CompMoves}(G), \text{Mon} = \text{cm}(G))$,
- $\mathbb{G}(\text{rmdtw}, G) := (\text{Pos}(G), \text{CompMoves}(G), \text{Mon} = \text{rm}(G))$,
- $\mathbb{G}(\text{nmDAG}, G) := (\text{Pos}(G), \text{ReachMoves}(G), \text{Mon} = \text{Fin})$,
- $\mathbb{G}(\text{cmDAG}, G) := (\text{Pos}(G), \text{ReachMoves}(G), \text{Mon} = \text{cm}(G))$,
- $\mathbb{G}(\text{DAG}, G) := (\text{Pos}(G), \text{ReachMoves}(G), \text{Mon} = \text{rm}(G))$.

It follows immediately from the definitions that, for all digraphs G ,

$$\begin{aligned} cn_G(\text{dtw}) &\leq cn_G(\text{cmdtw}), cn_G(\text{rmdtw}) \text{ and} \\ cn_G(\text{cmdtw}), cn_G(\text{rmdtw}) &\leq cn_G(\text{nmDAG}) \leq cn_G(\text{DAG}), cn_G(\text{cmDAG}). \end{aligned} \tag{1}$$

The number $cn_G(\text{cmdtw}) - cn_G(\text{dtw})$ is called the *cop-monotonicity cost* for the component game on G . Robber-monotonicity cost as well as monotonicity cost for other game variants are defined analogously.

2.2 Decompositions and Widths

Most of the games described in Definition 2.1 can be characterised by widths of decompositions of the graphs. In the following let G be an arbitrary graph. For $v, w \in V(G)$ we write $v \leq w$ if $w \in \text{Reach}_G(v)$ and $v < w$ if, additionally, $v \neq w$.

Directed tree width [22, 14] was the first generalisation of tree width to digraphs. For $X, Y \subseteq V(G)$ we say that X is Y -normal if X is a union of components of $G - Y$. An *arboreal decomposition* of G is a triple (R, X, W) where R is a directed tree with edges oriented away from the root and $X = \{X_e : e \in E(R)\}$ and $W = \{W_r : r \in V(R)\}$ are collections of sets of vertices of G such that

- (i) W is a partition of $V(G)$ into nonempty sets and
- (ii) if $e = (t, s) \in E(R)$, then $W_{\geq e}$ is X_e -normal where $W_{\geq e} = \bigcup\{W_r : r \in V(R), r \geq s\}$.

The *width* of (R, X, W) is $\max_{r \in V(R)} |W_r \cup \bigcup_{e \sim r} X_e| - 1$ where $e \sim r$ means that r is incident with e . The *directed tree width* of G is the least width of an arboreal decomposition of G .

DAG-width was defined in [3] and simultaneously in [20]. A *DAG decomposition* of G is a tuple (D, B) where D is a DAG and $B = \{B_d : d \in V(D)\}$ is a set of bags, i.e. subsets of $V(G)$, such that

1. $\bigcup_{d \in V(D)} B_d = V(G)$,
2. for all $a, b, c \in D$, if $a < b < c$, then $B_a \cap B_c \subseteq B_b$,
3. for every root $r \in V(D)$, $\text{Reach}_G(B_{\geq r}) = B_{\geq r}$ where $B_{\geq r} = \bigcup_{r \leq d} B_d$,
4. for each $(a, b) \in E(D)$, $\text{Reach}_{G-(B_a \cap B_b)}(B_{\geq b} \setminus B_a) = B_{\geq b} \setminus B_a$.

The width of (D, B) is $\max_{d \in V(D)} |B_d|$ and its size is $|V(D)|$. The *DAG-width* $\text{DAG-w}(G)$ of G is the minimal width of a DAG decomposition of G .

Kelly-width is a complexity measure for digraphs introduced in [13]. Similarly to tree width, Kelly-width can be defined by a decomposition, by a graph searching game or by an elimination order. We choose the latter definition. An *elimination order* \triangleleft for a graph $G = (V, E)$ is a linear order on V . For a vertex v define $V_{\triangleright v} := \{u \in V : v \triangleleft u\}$. The *support* of a vertex v with respect to \triangleleft is

$$\text{supp}_{\triangleleft}(v) := \{u \in V : v \triangleleft u \text{ and there is } v' \in \text{Reach}_{G-V_{\triangleright v}}(v) \text{ with } (v', u) \in E\}.$$

The *width* of an elimination order \triangleleft is $\max_{v \in V} |\text{supp}_{\triangleleft}(v)|$. The *Kelly-width* $\text{Kelly-w}(G)$ of G is one plus the minimum width of an elimination order of G .

In [23], Safari suggests D-width as another structural complexity measure. Let G be a graph. A *D-decomposition* of G is a pair $(T, (X_t)_{t \in V(T)})$ where T is an undirected tree and $X_t \subseteq V(G)$ for all $t \in V(T)$ is a set of bags such that for all $v \in V(G)$ the set $\{t \in V(T) : v \in X_t\}$ is non-empty and connected in T and for every edge $(s, t) \in E(T)$ and every strongly connected component C of $G - (X_s \cap X_t)$, either $V(C) \subseteq \bigcup_{r \in V(T_s)} X_r$ or $V(C) \subseteq \bigcup_{r \in V(T_t)} X_r$, where T_s, T_t are the two connected components of $T - \{(s, t), (t, s)\}$. The width of $(T, (X_t))$ is $\max_{t \in V(T)} |X_t|$. The *D-width* of G , $\text{D-w}(G)$, is the minimum of the widths of all D-decompositions of G .¹

2.3 Known Relations between Cop Numbers and Widths

We are interested in the question which cop numbers and widths are bounded in terms of which (other) cop numbers and widths. For instance, for DAG-width and Kelly-width we want to know whether there is a function $f: \mathbb{N} \rightarrow \mathbb{N}$ such that for all graphs G we have $\text{DAG-w}(G) \leq f(\text{Kelly-w}(G))$. Besides bounds from the inequalities in (1) the following relations are known.

► **Theorem 2.2** ([14, 4, 23]²). *Let G be a graph.*

1. $\text{dtw}(G)$, $cn_G(\text{dtw})$ and $cn_G(\text{rmdtw})$ are within factor 3 of each other.
2. $cn_G(\text{DAG}) = cn_G(\text{cmDAG}) = \text{DAG-w}(G)$.
3. $cn_G(\text{dtw}) \leq 2cn_G(\text{cmdtw}) + 1 < \text{D-w}(G)$.

This allows us to call $\mathbb{G}(\text{dtw}, G)$ the directed tree width game and $\mathbb{G}(\text{DAG}, G)$ the *DAG-width game*.

Berwanger et al. present a class of graphs certifying that DAG-width is not bounded in directed tree width. The same holds if we substitute directed tree width by D-width and/or DAG-width by Kelly-width (using the same class of graphs).

► **Theorem 2.3** ([4]). *There is a class of graphs G_n such that $cn_{G_n}(\text{dtw}) = cn_{G_n}(\text{cmdtw}) = 2$ and $cn_{G_n}(\text{rmdtw}) = \text{D-w}(G_n) = 1$, but $\text{DAG-w}(G)$ and $\text{Kelly-w}(G)$ are not bounded.*

¹ In [23] the width is $\max_{t \in V(T)} |X_t| - 1$.

² The second inequality in (3) is not proven in the cited works, but easy to prove, see Appendix.

3 The Complexity of DAG-width and the DAG-width Game

For all widths W considered in our work except DAG-width it is easy to see that the problem, given a graph G and a natural number k , whether $W(G) \leq k$, is in NP. The reason is that the size of the corresponding decompositions is polynomial in the size of G . Because on graphs with a symmetric edge relation all widths considered here are equal to tree width and tree width is NP-hard, the width measures here are NP-complete. We show that the situation with DAG-width is different, however. It turns out that DAGW, the problem whether $\text{DAG-w}(G) \leq k$, is PSPACE-complete and, moreover, for some graphs, there are no decompositions of polynomial size even if we allow a constant additive error in the width.

► **Theorem 3.1.** *DAGW is PSPACE-complete.*

Proof sketch. The easier part is to show that DAGW is in PSPACE. Due to the robber-monotonicity, the length of every play in $\mathbb{G}(\text{DAG}, G)$ is linear in $|G|$. Hence the winner of the game can be determined in alternating PTIME (by simply simulating the game) and thus in PSPACE.

For the hardness, we reduce QBF, which is PSPACE-complete, to DAGW. A *quantified boolean formula* φ is of the form $\varphi = Q_1 X_1 \dots Q_r X_r \psi(X_1, \dots, X_r)$ where Q_i is either \forall or \exists and ψ is a propositional formula in CNF with variables from $\mathcal{X} = \{X_1, \dots, X_r\}$. A formula $\exists X \psi(X)$ is true if there is a value $\beta(X) \in \{0, 1\}$ for X such that ψ is true. A formula $\forall X \psi(X)$ is true if for both values $\beta(X) \in \{0, 1\}$ for X , ψ is true.

It is very well known that deciding QBF, the problem whether a given quantified formula is true, is PSPACE-complete. The idea of our reduction is to simulate the choice of a truth value for a variable by a quantifier in the game $\mathbb{G}(\text{DAG}, S_\varphi)$, where S_φ is some graph constructed from φ . The choices are stored as vertices occupied by cops using the monotonicity of the game. These cops only reflect the history of the play and do not change the flow of the remaining play.

Let $\varphi = Q_1 X_1 Q_2 X_2 \dots Q_r X_r \psi(X_1, \dots, X_r)$ be a quantified boolean formula. The graph S_φ is constructed inductively level by level, each of which corresponds to some X_i .

If φ has no variables, then if φ is true, S_φ is a single vertex, and if φ is false, S_φ is a 2-clique. Then one cop wins if, and only if, φ is true. Otherwise we start the construction of S_φ with a gadget F_ψ . It has a vertex v and for every clause $C = L_1 \vee L_2 \vee \dots \vee L_{r(C)}$ an $r(C)$ -clique K^C with vertices $v_1^C, v_2^C, \dots, v_{r(C)}^C$. The edges go from v to every vertex of K^C and back, i.e. we have edges (v, v_i^C) and (v_i^C, v) for all clauses C and all $i \in \{1, \dots, r(C)\}$.

For $j = r, r-1, \dots, 1$ we construct graphs S_φ^j such that $S_\varphi^1 = S_\varphi$. For convenience, let $S_\varphi^{r+1} = F_\psi$.

Assume that S_φ^{j+1} has already been constructed. Then S_φ^j is the following graph. There are two cases. If $Q_j = \exists$, then the vertex set is

$$V(S_\varphi^j) = V_{\exists}(j) = V(S_\varphi^{j+1}) \cup A(j) \cup B(j) \cup C_0(j) \cup C_1(j) \cup M(j) \cup D(j) \cup \{c_0(j), c_1(j)\}$$

where $|A(j)| = |B(j)| = |D(j)| = 2$, $|C_i(1)| = |M(1)| = 4$, $|C_i(k+1)| = |M(k+1)| = |M(k)| + 3$ for all $k \in \{2, \dots, j-1\}$ and $i \in \{0, 1\}$. Furthermore, $B(j) = \{b_0(j), b_1(j)\}$. We set $N(j) = M(j) \cup D(j)$.

The set of edges is

$$\begin{aligned}
E(S_\varphi^j) &= E(S_\varphi^{j+1}) \cup \binom{N(j)}{2} \cup \bigcup_{i=0}^1 \binom{C_i(j)}{2} \cup \binom{A(j)}{2} \\
&\cup \bigcup_{i=0}^1 \left((N(j) \times \{c_i(j)\}) \cup (\{c_i(j)\} \times C_i(j)) \cup (C_i(j) \times D(j)) \cup (C_i(j) \times \{b_i(j)\}) \right) \\
&\cup (B(j) \times A(j)) \cup (A(j) \times B(j)) \cup (A(j) \times M(j)) \\
&\cup (N(j) \times V(S_\varphi^{j+1})) \cup (A(j) \times V(S_\varphi^{j+1})) \cup (V(S_\varphi^{j+1}) \times A(j)) \cup E(j).
\end{aligned}$$

Hereby, for a set X , the notation $\binom{X}{2}$ means $\{(a, b) \in X^2 : a \neq b\}$ and $E(j)$ is the set of edges connecting F_ψ to the new level defined as follows. Let K^C be a clique in F_ψ corresponding to the clause $C = L_1 \vee \dots \vee L_r$. If $X_j = L_i$, then $(v_i^C, b_1(j)) \in E(j)$. If $\neg X_j = L_i$, then $(v_i^C, b_0(j)) \in E(j)$. Otherwise (i.e. if X_j does not appear in C) $\{(v_i^C, b_0(j)), (v_i^C, b_1(j))\} \subseteq E(j)$.

In the second case $Q_j = \forall$. Then $V(S_\varphi(j)) = V_\forall(j) = V_\exists(j) \setminus \{c_0(j), c_1(j)\}$ and the edges are as in an existential level (including edges connecting the level and F_ψ), but edges containing $c_i(j)$ are replaced by edges $\bigcup_{i=0}^1 N(j) \times C_i(j)$. In other words, the paths that lead from $N(j)$ to $C_i(j)$ through $c_i(j)$ are replaced by direct edges.

One can show that $r + 1$ cops win on S_φ if, and only if, the formula φ is true. The main ingredient of the proof is that in the cops and robber game on S_φ , the cops can expel the robber from a level ℓ only in one way up to irrelevant changes. Hereby, exactly $\ell - 3$ cops remain free for use in the next level $\ell - 3$. They occupy $N(\ell)$, the robber goes to one of the $C_i(\ell)$ and a cop is placed to $b_i(\ell)$. If now the robber remains in $C_i(\ell)$, he is captured there by the cops from $M(\ell)$, so he goes to $A(\ell)$ or to the next level. In any case the cops from $D(\ell)$ move to $A(\ell)$ and the robber is in the next level $\ell - 3$. Note that the cops occupy $A(\ell)$ (blocking the robber in lower levels) and one vertex from $B(\ell)$. This one vertex encodes the choice for the value of the variable from φ that corresponds to that level. In universal levels it is the robber who makes the choice and in the existential these are the cops.

If the level is universal, the robber determines which vertex from $B(\ell)$ will be occupied by deciding in which $C_i(\ell)$ he goes after the cops occupy $N(\ell)$. In the existential level, the cops can determine in which $C_i(\ell)$ the robber must go. If they want $b_{1-i}(\ell)$ to be occupied when the robber leaves level ℓ , they place a cop on $c_i(\ell)$ before occupying $N(\ell)$. Then the cops expel the robber from $C_i(\ell)$ if he is there and occupy $N(\ell)$. The robber goes to $C_{1-i}(\ell)$ (all paths to $C_i(\ell)$ are blocked) or directly to $A(\ell) \cup B(\ell) \cup S_\varphi^{\ell-3}$. In any case the cop from $c_i(\ell)$ moves to $b_{1-i}(\ell)$. If the robber was in $C_{1-i}(\ell)$ and remains there, he is captured by the cops from $M(\ell)$ as before, so after $b_{1-i}(\ell)$ is occupied, the robber is in $A(\ell)$ and after the cops from $D(\ell)$ occupy $A(\ell)$, he is in the next level.

When the robber leaves the last level and proceeds to F_ψ , one cop remains free and goes to v . The robber chooses a clique K^C corresponding to the clause C in ψ . At this point, the value for X_j from C is $\alpha(X_j) = i$ if and only if a cop occupies $b_i(j)$. Furthermore, the construction of edges between F_ψ and the levels guarantees that $\alpha \models C$ if and only if the cop from $B(j)$ can be reused without violating robber-monotonicity. Finally, the cops capture the robber in K^C if and only if they have one free cop. Summing up, the cops win if and only if φ is true. \blacktriangleleft

We can change the construction of S_φ to obtain graphs that have no polynomial size DAG decomposition of width that differs from the optimal one in at most a fixed additive constant. We replace F_ψ in S_φ by a single vertex, make every level universal and adjust the

sizes of $A(\ell)$, $B(\ell)$ and $D(\ell)$ by setting $|A(\ell)| = |B(\ell)| = |D(\ell)| = \lfloor \frac{\ell}{\log \ell} \rfloor$. Then a careful calculation of used cops proves the following theorem.

► **Theorem 3.2.** *There is no polynomial size approximation of an optimal DAG decomposition of $G_n(s, t)$ with an additive constant error.*

4 Comparing Width Measures with Respect to Generality

By Theorem 2.2, directed tree width and the robber-monotone variant of the corresponding game are bounded in each other. One would expect that the same holds for the cop-monotone variant. This was implicitly assumed by Safari in [23] who conjectured that D-width and directed tree width are the same. Note that by Theorem 2.2, $s \cdot cn_G(\text{dtw}) + 1 \leq \text{D-w}(G)$, so if $\text{dtw}(G) = \text{D-w}(G)$, then the cop-monotonicity cost for directed tree width is zero. We show, however, that it is not only positive, but, moreover, cannot be bounded by any function.

► **Theorem 4.1.** *There is a class $\{G_n : n > 2\}$ of graphs such that for all $n > 2$, $cn_G(\text{dtw}) = cn_{G_n}(\text{rmdtw}) \leq 4$ and $cn_{G_n}(\text{cmdtw}) \geq n$.*

Proof. Let $n > 2$. We inductively define a sequence of graphs G_n^m and sets of marked vertices $M(G_n^m) \subseteq V(G_n^m)$ for $m \in \{1, \dots, n+1\}$. We then define G_n as G_n^{n+1} .

First G_n^1 is an edgeless graph with a single vertex and $M(G_n^1) = V(G_n^1)$, i.e. the vertex of G_n^1 is marked. Assume that $(G_n^m, M(G_n^m))$ has been constructed. Let T_ℓ^d denote the complete undirected tree of branching degree d and depth ℓ (the depth is the maximum number of vertices on the path from the root to a leaf). One part of G_n^{m+1} is a copy of T_{n+2}^{n+1} , which has $(n+1)^{n+2}$ leaves v_s for $s \in \{1, \dots, (n+1)^{n+2}\}$. The graph G_n^{m+1} is the disjoint union of T_{n+2}^{n+1} and $n \cdot (n+1)^{n+2}$ copies $H_j^{m+1}(v_s)$ of G_n^m where $j \in \{1, \dots, n\}$ and $s \in \{1, \dots, (n+1)^{n+2}\}$ plus some additional edges which we describe next. We denote the subgraph of G_n^{m+1} induced by T_{n+2}^{n+1} by $T(G_n^{m+1})$ and the root of $H_j^{m+1}(v_s)$ by $r(H_j^{m+1}(v_s))$ for all m, j and s .

For every leaf $v \in \{v_s : 1 \leq s \leq (n+1)^{n+2}\}$ of $T(G_n^{m+1})$ there is an undirected edge from v to the root of $H_i^{m+1}(v)$. Let $x_i^{m+1}(v)$ be the i th vertex on the path from the root of $T(G_n^{m+1})$ to v . For all leaves v of $T(G_n^{m+1})$ and all $1 \leq i \leq n$ we add directed edges from $x_i^{m+1}(v)$ to all marked vertices $M(H_i^{m+1}(v))$ of $H_i^{m+1}(v)$. Finally, for all leaves v of $T(G_n^{m+1})$ and all leaves of $H_i^{m+1}(v)$ we add a directed edge to v . We define $M(G_n^{m+1}) := V(T(G_n^{m+1}))$.

Let us describe a non-cop-monotone winning strategy for 4 cops on G_n . Observe that $G_n = G_n^{n+1}$ is an undirected tree with additional edges that connect only vertices of the same branch. In particular, for each subgraph $H_j^i(v)$, if the robber is in $H_j^i(v)$ and the cops block the root of $T(H_j^i(v))$ and $x_j^{i+1}(v)$, then the robber cannot leave $H_j^i(v)$ as he cannot re-enter $H_j^i(v)$.

The cops chase the robber from the root of G_n downwards. In $T(G_n)$, two cops suffice for that. Consider a position where the cops just expelled the robber from $T(G_n)$. The robber is in some $H_j^n(v)$ and the cops occupy v and its predecessor w . Now the cop from w goes to $x_j^n(v)$ (here non-cop-monotonicity occurs) and a third cop occupies $r(H_j^n(v))$. The cop on $r(H_j^n(v))$ together with the cop on $x_j^n(v)$ block all paths from $T(G_n)$ to the robber in $H_j^n(v) - r(H_j^n(v))$, so the cops on v and w are not needed any more. These two cops chase the robber down the tree further, while the other cop remains on $r(H_j^n(v))$ and $x_j^n(v)$.

In general, assume for some $i < n, j$ and v (j and v are new), the robber is blocked in $H_j^i(v)$ by cops on v , on $r(H_{j'}^{i+1}(v'))$ and on $x_j^{i+1}(v)$. Hereby j' and v' are such that $r(H_{j'}^{i+1}(v'))$ is on the path from $r(H_j^i(v))$ to the root of G_n . Now the cop from the predecessor of v goes to $x_j^{i+1}(v)$ (again non-monotonicity occurs). Then the cop from $r(H_{j'}^{i+1}(v'))$ goes to $r(H_j^i(v))$.

These two cops block all paths from $G_n - H_j^i(v)$ to $H_j^i(v) - r(H_j^i(v))$. Hence the other two cops can chase the robber down the tree further. Finally the robber is captured in some leaf of G_n .

Now we construct a robber strategy that wins against all cop-monotone strategies for n cops if $n > 2$. For a vertex v and subtree T of G_n we say that T is a subtree of v if the root of T is a direct successor of v . The robber resides on a vertex of $T(G_n)$ that has the least distance to the root of G_n as long as this is possible. When a cop occupies his vertex v the robber proceeds to a directed successor of v such that the subtree of v is cop free. Such a successor always exists due to the high branching degree of $T(G_n)$. When the robber reaches a leaf w_n of $T(G_n)$, every vertex on the path from the root of G_n to w_n has been occupied by a cop. As the length of the path is greater than the number of cops, there is a vertex $x_{i_n}^n(w_n)$ that has been left by a cop. When a cop occupies w_n , the robber goes to $G_{i_n}^n(w_n)$. Now on $G_{i_n}^n(w_n)$ (which is isomorphic to G_n^{n-1}) the robber plays in the same way as on G_n and so on recursively for each m on $G_{i_m}^m(w_m)$. Note that until the robber is captured, there is a path from this vertex to a leaf of G_n and then to all already chosen w_j .

Consider a position when the robber arrives at a leaf v of G_n and a cop is landing on this vertex. Then at most $n - 1$ cops are on the graph and there is some j such that there is no cop in $T(G_{i_j}^j(w_j))$. Thus there is a cop free path from v to w_j , then to $x_{i_j}^j(w_j)$ within $T(G_{i_j}^j(w_j))$ and then via $x_{i_{j-1}}^{j-1}(w_{j-1})$, $x_{i_{j-2}}^{j-2}(w_{j-2})$, \dots , $x_{i_2}^2(w_2)$ back to v . Note that all those x -vertices are not occupied by cops by construction of the robber strategy. Thus the robber can return to w_j and play from w_j as before. In this way the robber will never be captured. \blacktriangleleft

► **Corollary 4.2.** *D-width is not bounded in directed tree width.*

There is yet another reason why Corollary 4.2 holds. In the definition of D-width we have the condition that the set of bags containing a vertex v is connected in the decomposition tree. This implies cop-monotonicity in the directed tree width game. Moreover, this forbids the existence of two distinct plays such that the cops are placed on v in both plays, but not in their common prefix. However, one can construct graphs where this restriction leads to an unbounded blow up of the number of needed cops. As the DAG-width and the Kelly-width of those graphs are bounded, we obtain the following theorem³.

► **Theorem 4.3.** *D-w(G) is bounded neither in $cn_G(\text{cmdtw})$, nor in DAG-w(G), nor in Kelly-w(G). More precisely, there is a class of graphs G_n such that 3 cops have a cop- and robber-monotone winning strategy in the directed tree width and DAG-width games on each G_n and $\text{Kelly-w}(G_n) = 4$, but $\text{D-w}(G_n) \geq n$.*

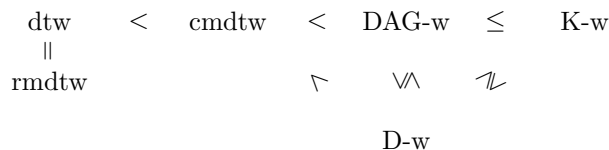
4.1 Kelly-width is Bounded in DAG-width

We show that DAG-width is bounded in Kelly-width by a quadratic function.

► **Theorem 4.4.** *If $\text{Kelly-w}(G) = k + 1$, then $\text{DAG-w}(G) \leq 72k^2 + 42k + 18$.*

In order to prove this we introduce a weaker notion of robber-monotonicity for the DAG-width games. Then we show that with a quadratic number of additional cops one can turn a winning cop strategy for the game with weak monotonicity into a winning strategy for the game with strong (i.e. usual) monotonicity. By a construction from [13], if $\text{Kelly-w}(G) = k$,

³ See the appendix for a proof.



■ **Figure 1** The boundedness relation between different measures. “=” means mutually bounded, “<” means bounded only in one direction, “≤” at least in one direction, “≰” not bounded in any direction.

then $2k - 1$ cops have a (possibly non-monotone) winning strategy in the DAG-width game. We observe that this strategy is, in fact, weakly monotone and thus can be converted into a strongly monotone one.

Weak monotonicity relaxes the winning condition for the cops, so that they win more plays. Formally, for a digraph G we define the set $\text{wm}(G)$ as the set of all finite plays $(C_0, R_0), (C_0, C_1, R_0), (C_1, R_1), \dots$ such that the following condition is satisfied. For all i let $c(i) := C_{i+1} \cap R_i$ be the cops which move into the component of $G - C_i$ currently used by the robber. We call these cops the *chasers*. All other cops being placed, i.e. the cops in $(C_{i+1} \setminus C_i) \setminus c(i)$ are *guards*. The play $(C_0, R_0), (C_0, C_1, R_0), (C_1, R_1), \dots$ is *weakly monotone* if for all i and all j with $j < i$, no vertex in $c(j)$ is reachable by a directed path from any vertex in R_i in $G - (C_i \cap C_{i+1})$. That is, for weak monotonicity we only require monotonicity in the cops that are used to shrink the robber space but not in the cops placed outside of the component to block the paths to previous cop positions. The set $\text{wm}(G)$ is the set of all weakly monotone plays on G . The *weakly monotone game* is the game defined by $\mathbb{G}(\text{wmDAGW}, G) = (\text{Pos}(G), \text{ReachMoves}(G), \text{Mon} = \text{wm}(G))$.

► **Lemma 4.5.** $cn_G(\text{wmDAG}) \leq 18 \cdot cn_G(\text{DAG})^2 + 3 \cdot cn_G(\text{DAG})$.

As, clearly, $cn_G(\text{DAG}) \leq cn_G(\text{wmDAG})$, we obtain that weakening the monotonicity in the DAG-width game does not change the boundedness of $cn_G(\text{DAG})$.

We remark that it is possible to define a decomposition corresponding to the weakly monotone game. Unlike DAG decompositions a *weak* DAG decomposition is always of polynomial size in the size of G . Hence we have an NP-algorithm that computes a succinct representation of a DAG decomposition whose width is at most quadratically worse than the optimum.

► **Lemma 4.6.** *If $\text{Kelly-w}(G) = k + 1$, then $cn_G(\text{wmDAG}) \leq 2k + 1$.*

The other direction, i.e. whether DAG-width is bounded in Kelly-width, is the last open question in our scheme.

We obtain the picture shown in Figure 1. The only blank spot is the strictness of the inequality $\text{DAG-w}(G) \leq \text{Kelly-w}(G)$, i.e. whether Kelly-width is a function of DAG-width. It was conjectured that Kelly-width and DAG-width differ by at most a constant factor [13, Conjecture 30]. However, methods we used to show a weaker version of one direction of the conjecture do not seem to apply for the other direction.

References

- 1 I. Adler. Directed tree-width examples. *J. Comb. Theory, Ser. B*, 97(5):718–725, 2007.
- 2 J. Barát. Directed Path-width and Monotonicity in Digraph Searching. *Graphs and Comb.*, 22(2), 2006.

- 3 D. Berwanger, A. Dawar, P. Hunter, and S. Kreutzer. DAG-width and parity games. In *STACS '06*, volume 3884 of *LNCS*. Springer, 2006.
- 4 D. Berwanger, A. Dawar, P. Hunter, S. Kreutzer, and J. Obdržálek. The DAG-width of directed graphs. *J. Comb. Theory*, 102(4):900–923, 2012.
- 5 D. Bienstock and P. Seymour. Monotonicity in graph searching. *Journal of Algorithms*, 12(2):239–245, 1991.
- 6 N. Dendris, L. Kirousis, and D. Thilikos. Fugitive search games on graphs and related parameters. *Theoretical Computer Science*, 172(1–2):233 – 254, 1997.
- 7 D. Dereniowski. From pathwidth to connected pathwidth. In *28th Symposium on Theoretical Aspects of Computer Science (STACS)*, 2011.
- 8 R. Diestel. *Graph Theory, 4th Edition*. Springer, 2012.
- 9 F. Fomin, P. Golovach, and D. Thilikos. Approximation algorithms for domination search. In Klaus Jansen and Roberto Solis-Oba, editors, *Approximation and Online Algorithms (WAOA)*, volume 6534 of *Lecture Notes in Computer Science*, pages 130–141. Springer Berlin / Heidelberg, 2011.
- 10 F. Fomin, D. Kratsch, and H. Müller. On the domination search number. *Discrete Applied Mathematics*, 127(3):565–580, 2003.
- 11 F. Fomin and D. Thilikos. On the monotonicity of games generated by symmetric submodular functions. *Discrete Applied Mathematics*, 131(2):323 – 335, 2003.
- 12 F. Fomin and D. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, 2008.
- 13 P. Hunter and S. Kreutzer. Digraph measures: Kelly decompositions, games, and orderings. *Theor. Comput. Sci.*, 399(3), 2008.
- 14 T. Johnson, N. Robertson, P. Seymour, and R. Thomas. Directed Tree-Width. *J. Comb. Theory, Ser. B*, 82(1), 2001.
- 15 S. Kreutzer. Graph searching games. In Krzysztof R. Apt and Erich Grädel, editors, *Lectures in Game Theory for Computer Scientists*, chapter 7, pages 213–263. CUP, 2011.
- 16 S. Kreutzer and S. Ordyniak. Distance-d-domination games. In *34th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, 2009.
- 17 S. Kreutzer and S. Ordyniak. Digraph decompositions and monotonicity in digraph searching. *Theor. Comput. Sci.*, 412(35):4688–4703, 2011.
- 18 A. S. LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM*, 40:224–245, 1993.
- 19 F. Mazoit and N. Nisse. Monotonicity of non-deterministic graph searching. *Theor. Comput. Sci.*, 399(3):169–178, 2008.
- 20 J. Obdržálek. *Algorithmic analysis of parity games*. PhD thesis, School of Informatics, University of Edinburgh, 2006.
- 21 R. Rabinovich. *Graph Complexity Measures and Monotonicity*. PhD thesis, RWTH Aachen University, 2013.
- 22 B. Reed. Introducing directed tree-width. *Electronic Notes in Discrete Mathematics*, 3:222 – 229, 1999.
- 23 M. A. Safari. D-width: A more natural measure for directed tree width. In Joanna Jedrzejowicz and Andrzej Szepietowski, editors, *MFCS*, volume 3618 of *Lecture Notes in Computer Science*, pages 745–756. Springer, 2005.
- 24 P. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *J. Comb. Theory Ser. B*, 58(1), 1993.
- 25 Y. Stamatiou and D. Thilikos. Monotonicity and inert fugitive search games. In *6th Twente Workshop on Graphs and Combinatorial Optimization CTW 1999*. University of Twente, Enschede, 1999.

- 26 B. Yang and Y. Cao. Monotonicity of strong searching on digraphs. *J. Comb. Optim.*, 14(4):411–425, 2007.
- 27 B. Yang and Y. Cao. On the monotonicity of weak searching. In *COCOON*, pages 52–61, 2008.

Subset Sum in the Absence of Concentration

Per Austrin¹, Petteri Kaski², Mikko Koivisto³, and
Jesper Nederlof⁴

- 1 School of Computer Science and Communication, KTH Royal Institute of Technology, Sweden
austrin@csc.kth.se
- 2 Helsinki Institute for Information Technology HIIT & Department of Computer Science, Aalto University, Finland
petteri.kaski@aalto.fi
- 3 Helsinki Institute for Information Technology HIIT & Department of Computer Science, University of Helsinki, Finland
mikko.koivisto@helsinki.fi
- 4 Department of Mathematics and Computer Science, Technical University of Eindhoven, The Netherlands
jespernederlof@hotmail.com

Abstract

We study the exact time complexity of the SUBSET SUM problem. Our focus is on instances that lack additive structure in the sense that the sums one can form from the subsets of the given integers are not strongly concentrated on any particular integer value. We present a randomized algorithm that runs in $O(2^{0.3399n} B^4)$ time on instances with the property that no value can arise as a sum of more than B different subsets of the n given integers.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems, F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics

Keywords and phrases subset sum, additive combinatorics, exponential-time algorithm, homomorphic hashing, Littlewood–Offord problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.48

1 Introduction

Given integers $a_1, a_2, \dots, a_n \in \mathbb{Z}$ and a target integer $t \in \mathbb{Z}$ as input, the NP-complete SUBSET SUM problem asks whether there exists a subset $S \subseteq [n]$ with $\sum_{j \in S} a_j = t$. Despite the apparent simplicity of the problem statement, to date there has been modest progress on exact algorithms [11] for SUBSET SUM. Indeed, from a worst-case performance perspective the fastest known algorithm runs in $O^*(2^{n/2})$ time¹ and dates back to the 1974 work of Horowitz and Sahni [13]. Improving the worst-case running time is a well-established open problem [28, §53].

Improved algorithms are known in special cases where one assumes control on the additive structure in the *steps*² a_1, a_2, \dots, a_n . In particular this is the case if we assume

¹ The $O^*(\cdot)$ notation suppresses a multiplicative factor polynomial in the input size.

² The term *step* originates from the study of the Littlewood–Offord problem in additive combinatorics, see e.g. Tao and Vu [26] and [25, §7]. In this context the integers $\mathbf{a} = (a_1, a_2, \dots, a_n)$ define the step-lengths of an n -step random walk on \mathbb{Z} given by the random variable $S(\mathbf{a}) = \sum_{i=1}^n a_i \epsilon_i$, where the values $\epsilon_1, \epsilon_2, \dots, \epsilon_n \in \{-1, 1\}$ are selected independently and uniformly at random. (The SUBSET

that the possible sums that one can form out of the steps are supported by a small set of values. Assuming such additive structure is available, a number of algorithmic techniques exist to improve upon the Horowitz–Sahni bound, ranging from Bellman’s classical dynamic programming [5] to algebraization [16, 18] and to parameterized techniques [9].

Given that additive structure enables improved algorithms, it would thus be natural to expect, *a priori*, that the worst-case instances are the ones that lack any additive structure. In a cryptographic context such instances are frequently assumed to be *random*. Yet, recently Howgrave-Graham and Joux [14] made a breakthrough by showing that random instances can in fact be solved in $O^*(2^{0.337n})$ time; a correct derivation of this bound and an improvement to $O^*(2^{0.291n})$ are given by Becker, Coron, and Joux [4].

The results of Joux *et al.* raise a number of intriguing combinatorial and algorithmic questions. Given that improved algorithms exist for random instances, precisely what types of hard instances remain from a worst-case analysis perspective? What are the *combinatorial* properties that the hard instances must have? Conversely, from an algorithms standpoint one would like to narrow down precisely what *intrinsic* property of the steps a_1, a_2, \dots, a_n enables algorithm designs such as the Joux *et al.* results. Such a quest for intrinsic control is further motivated by the relatively recent identification of a large number of natural dichotomies between *structure* and *randomness* in combinatorics (cf. Tao [22, 23, 24], Trevisan [27], and Bibak [6]). Do there exist algorithm designs that capitalize on *pseudorandomness* (absence of structure) to improve over the worst-case performance?

In this paper we seek to take algorithmic advantage of the *absence* of additive structure in instances of SUBSET SUM. A prerequisite to such a goal is to have a combinatorial parameter that measures the extent of additive structure in an instance. The close relationship between the SUBSET SUM problem and the *Littlewood–Offord problem* in additive combinatorics [25, §7] suggests that one should study measures of additive structure in the context of the latter.

We show that such transfer is indeed possible, and leads to improved exact algorithms for pseudorandom instances of SUBSET SUM. Our measure of choice for pseudorandomness is the *concentration probability* employed, for example, by Tao and Vu [26] in the context of inverse theorems in Littlewood–Offord theory. We use an equivalent but fully combinatorial definition that will be more convenient in the context of SUBSET SUM. Define the set function $a : 2^{[n]} \rightarrow \mathbb{Z}$ for all $X \subseteq [n]$ by $a(X) = \sum_{j \in X} a_j$. We now assume that we have a uniform upper bound on the size of the preimages $a^{-1}(u) = \{X \subseteq [n] : a(X) = u\}$ for $u \in \mathbb{Z}$. For $B \geq 1$ we say that the instance a_1, a_2, \dots, a_n has *B-bounded concentration* if for all $u \in \mathbb{Z}$ it holds that $|a^{-1}(u)| \leq B$.³ That is, no value $u \in \mathbb{Z}$ may occur as a sum $a(X) = u$ for more than B different subsets $X \subseteq [n]$. The extreme case $B = 1$ captures the notion of *additive independence* or *dissociativity* in additive combinatorics [25, §4.32]. The other extreme case $B = 2^n$ is achieved by $a_1 = a_2 = \dots = a_n = 0$, and more generally, an instance of high density (see §1.2) automatically has high concentration.

Our main result is that all instances *without* strong additive structure (without exponential concentration of sums) can be solved faster than the Horowitz–Sahni time bound $O^*(2^{n/2})$. A quantitative claim is as follows.⁴

SUM problem is equivalent to asking whether the outcome $S(\mathbf{a}) = s$ has positive probability for $s = 2t - \sum_{j=1}^n a_j$.

³ In terms of the Littlewood–Offord random walk, B -bounded concentration is equivalent to the assertion that for all $u \in \mathbb{Z}$ the probability of the outcome $S(\mathbf{a}) = u$ is at most $B/2^n$.

⁴ The running time versus the concentration bound in Theorem 1 can be sharpened somewhat; our subsequent analysis enables a more precise smooth tradeoff curve between the concentration bound B and the running time of the algorithm captured by Equation (5) in what follows. When the instance

► **Theorem 1.** *There exists a randomized algorithm for SUBSET SUM that with probability $1 - o(1)$ solves instances with B -bounded concentration in time $O^*(2^{0.3399n} B^4)$.*

Theorem 1 shows that we can afford quite considerable additive structure (exponential concentration of sums) in an instance, and still remain below the Horowitz–Sahni $O^*(2^{n/2})$ worst-case upper bound. Here we should note that we do not show how to *decide* whether a given instance has B -bounded concentration. Yet, by invoking our algorithm with $B = 1, 2, 4, \dots$ we can conclude within the time bound in Theorem 1 that we have decided the instance correctly with high probability *or* the instance does not have B -bounded concentration. Thus, we are left with a dichotomy of possibilities from the perspective of worst-case algorithm design:

Either worst-case instances *must* have considerable additive structure (and the Horowitz–Sahni bound thus remains uncontested), *or* SUBSET SUM has improved worst-case algorithms.

1.1 Methodology and Contributions

Our main challenge is to show that having control *only* on the concentration parameter B enables algorithmic implications. Here our approach is to alleviate the randomness assumptions on one of the aforementioned Joux *et al.* algorithm designs by injecting extra entropy into the algorithm.

In particular, we view the choice of the moduli in the algorithm as defining a *family of hash functions*. We show that control only on B suffices for probabilistic control on the extent to which a fixed solution is witnessed in the codomain of a *random* hash function. Thus, we obtain an algorithm design by (i) selecting a random hash function, (ii) selecting a random element in the codomain, and (iii) searching the preimage for solutions.

The Howgrave-Graham–Joux two-level design. To highlight our contribution in more detail, let us begin with a high-level review of a Howgrave-Graham–Joux [14] design, adapted from the careful presentation by Becker [3, §4.2], that we then proceed to analyze and instrument in what follows. The algorithm searches for a solution S by building the solution from four disjoint parts

$$S = S_1 \cup S_2 \cup S_3 \cup S_4 \quad \text{with} \quad |S_1| = |S_2| = |S_3| = |S_4| = |S|/4. \quad (1)$$

The search is executed along what can be viewed as a two-level recursion tree, which is best analyzed from a top-down perspective. (The actual algorithm will operate in the reverse direction from the bottom up when searching for S .) The first level splits S into two halves T and $S \setminus T$ for some $T \in \binom{S}{|S|/2}$. The second level then splits T and $S \setminus T$ into halves. That is, T splits to L and $T \setminus L$ for some $L \in \binom{T}{|S|/4}$, and $S \setminus T$ splits to R and $(S \setminus T) \setminus R$ for some $R \in \binom{S \setminus T}{|S|/4}$. The triple (T, L, R) now naturally splits S into four parts

$$S_1 = L, \quad S_2 = T \setminus L, \quad S_3 = R, \quad S_4 = (S \setminus T) \setminus R. \quad (2)$$

The advantage of the tree-like design over a flat design (1) is that subtrees at T and $S \setminus T$ are (essentially) independent and the T -component of the three-tuple (T, L, R) gives direct control on what happens at nodes T and $S \setminus T$ in the tree. The design now controls the size of

does not have B -bounded concentration, the algorithm will run in the claimed time in the parameters n and B , but in this case the algorithm is not guaranteed to find a solution with high probability.

the search space by fixing two $|S|/2$ -bit coprime moduli, M_1 and M_2 , and then *constraining uniformly at random* the congruence classes of the values $a(T)$, $a(L)$, and $a(R)$, which are constrained modulo M_1M_2 , M_1 , and M_1 , respectively.⁵ A clever application of a theorem of Nguyen, Shparlinski, and Stern [19, Theorem 3.2] on the distribution of modular sums across the ensemble of all instances a_1, a_2, \dots, a_n then enables Howgrave-Graham and Joux to control on the size of the search space on all but an exponentially negligible fraction of instances to obtain $O^*(2^{0.337n})$ running time on random instances. This is the starting point of our present work.

Our contributions. In contrast to the Howgrave-Graham–Joux analysis, our objective is to *localize* the ensemble analysis to individual instances with control only on the instance-specific additive structure captured by B . Our central observation is that we can obtain further probabilistic control on the algorithm by studying how the selection of the moduli M_1 and M_2 affects the search space. In particular it makes sense to treat the selection of M_1 and M_2 as the task of selecting a hash function from a formal family of hash functions h_{M_1, M_2} with

$$h_{M_1, M_2}(T, L, R) = (a(T) \bmod M_1M_2, a(L) \bmod M_1, a(R) \bmod M_1) \in \mathbb{Z}_{M_1M_2} \times \mathbb{Z}_{M_1} \times \mathbb{Z}_{M_1}.$$

Our main contribution is now that, assuming control only on B , and letting $M_1 = p$ and $M_2 = q$ to be distinct random primes, we find that any fixed solution S is extensively witnessed in the codomain $\mathbb{Z}_{pq} \times \mathbb{Z}_p \times \mathbb{Z}_p$ of a random $h_{p,q}$ (Lemma 3). Furthermore, we show that the size of the search space traversed by (a minor instrumentation of) the two-level Howgrave-Graham–Joux design is bounded with essentially the same running time guarantee (Lemma 4).⁶ Here we stress that our main contribution is analytical (Lemma 3) and in explicating the relevance of the hash family that enables us to inject entropy into the algorithm to cope with control on B only. In essence, our contribution is in localizing the outliers substantially deviating from the ensemble average (Nguyen *et al.* [19, Theorem 3.2]) to instances with substantial additive structure (exponential concentration of sums).

1.2 Related Work

The complexity of a SUBSET SUM instance is commonly measured by the *density*, defined as $n/(\log_2 \max_i a_i)$. From the perspective of worst-case guarantees, the Schroeppe–Shamir algorithm [21] that uses $O^*(2^{n/2})$ time and $O^*(2^{n/4})$ space remains uncontested for instances of density at most 2. However, improved space–time tradeoffs have been discovered recently first for random instances by Dinur, Dunkelman, Keller, and Shamir [8], and then generalized to worst-case instances by Austrin, Kaski, Koivisto, and Määttä [2]. For worst-case instances with density greater than 2, Bellman’s classical dynamic programming algorithm [5] remains the fastest. If allowed only polynomial space, improving the $O^*(2^n)$ -time exhaustive search

⁵ Because the set function a is modular and a solution satisfies $a(S) = t$, control on T, L, R gives control on the corresponding right siblings $S \setminus T, T \setminus L, (S \setminus T) \setminus R$ and hence gives control on the entire search.

⁶ Observe that our bound $O^*(2^{0.3399n} B^4)$ is worse than the $O^*(2^{0.337n})$ obtainable for random instances with a corrected version of the original Howgrave-Graham and Joux [14] design (as described by Becker, Coron, and Joux [4] and Becker [3, §4.2]). This results from the fact that we analyze only the *two-level* design relative to control on B , whereas $O^*(2^{0.337n})$ time would require the analysis of a more intricate three-level design for a gain in the third decimal digit of the exponent of the running time. Our contribution in the present work should be viewed as not optimizing the base of the exponential via increasingly careful constructions but rather showing that control on B alone is sufficient to go below the Horowitz–Sahni bound by a slightly more refined analysis of the Howgrave-Graham–Joux design.

algorithm for density at most 1 is an open problem, while for density $1 + \Theta(1)$ there is an improved algorithm by Lokshtanov and Nederlof [18].

Impagliazzo and Naor [15] show that, with respect to polynomial time solvability, random instances are the hardest when the density is close to 1. As already mentioned, for such instances the $O^*(2^{n/2})$ time bound was recently improved to $O^*(2^{0.337n})$ [14] and subsequently to $O^*(2^{0.291n})$ [4, 3]. Interestingly, almost all instances of density at most 0.94 can be reduced to the problem of finding shortest non-zero vectors in lattices (cf. Lagarias and Odlyzko [17] and Coster *et al.* [7]). Flaxman and Przydatek's algorithm [10] solves random instances in expected polynomial time if the density is $\Omega(n/\log_2 n)$, or equivalently, $\log_2 \max_i a_i = O((\log_2 n)^2)$.

Random instances have been studied also for other NP-hard problems; see, for example, Achlioptas's survey on random satisfiability [1]. We are not aware of prior work on exact algorithms that make use of pseudorandomness (*absence* of structure) and improve over the worst case in such an explicit way as we do in the present work.

2 Preliminaries

This section makes a review of standard notation and results used in this paper.

We write $[n]$ for the set $\{1, 2, \dots, n\}$. For a finite set U , we write 2^U for the set of all subsets of U and $\binom{U}{k}$ for the set of all subsets of U of size k . For $0 \leq \sigma \leq 1$ let $H(\sigma) = -\sigma \log_2 \sigma - (1-\sigma) \log_2 (1-\sigma)$ be the binary entropy function with $H(0) = H(1) = 0$. For all integers $n \geq 1$ and $0 \leq \sigma \leq 1$ such that σn is an integer, we have by Stirling's formula [20] that $\binom{n}{\sigma n} \leq 2^{nH(\sigma)}$.

We write \mathbb{Z} for the set of integers and $\mathbb{Z}_{\geq 1}$ for the set of all positive integers. We will need the following weak version of the Prime Number Theorem [12, p. 494, Eq. (22.19.3)].

► **Lemma 2.** *For all large enough integers b it holds that there exist at least $2^b/b$ prime numbers p in the interval $2^b < p < 2^{b+1}$.*

For a modulus $M \in \mathbb{Z}_{\geq 1}$ and $x, y \in \mathbb{Z}$, we write $x \equiv y \pmod{M}$, or $x \equiv_M y$ for short, to indicate that M divides $x - y$.

For a logical proposition P , we write $[P]$ to indicate a 1 if P is true and a 0 if P is false.

3 The Algorithm

This section proves Theorem 1. Suppose we are given an instance $a_1, a_2, \dots, a_n, t \in \mathbb{Z}$ as input. We assume that the instance has B -bounded concentration.

3.1 Preprocessing and Parameters of the Input

By resorting to routine randomized preprocessing (detailed in the full version) and then invoking the main algorithm (to be described) a polynomial (in the original input size) number of times, we may assume that the input to the main algorithm has the following structure:

- (a) the input a_1, a_2, \dots, a_n, t consists of positive integers only,
- (b) $a_1 + a_2 + \dots + a_n + t \leq 2^{\tau n}$ for $\tau > 0$ a constant independent of n ,
- (c) the solution $S \subseteq [n]$, if any, has size $s = |S|$ that is known to us,
- (d) $n/100 \leq s \leq n/2$,
- (e) both n and s are multiples of 8, and
- (f) the instance has B -bounded concentration.

Define $\sigma = s/n$. In particular, $1/100 \leq \sigma \leq 1/2$.

3.2 The Hash Functions

We are interested in discovering the set S (if such a set exists) by assembling it from four equally-sized disjoint parts. Recalling our discussion in §1.1, we will follow a tree-based design with domain

$$\mathcal{D}(S) = \left\{ (T, L, R) : T \in \binom{S}{s/2}, L \in \binom{T}{s/4}, R \in \binom{S \setminus T}{s/4} \right\}.$$

Since we are always analyzing a fixed arbitrary solution S , we suppress S and simply write \mathcal{D} . The following family of hash functions seeks to witness at least one split from \mathcal{D} with high probability. Towards this end, for $p, q \in \mathbb{Z}_{\geq 1}$ define the function

$$h_{p,q} : \mathcal{D} \rightarrow \mathbb{Z}_{pq} \times \mathbb{Z}_p \times \mathbb{Z}_p$$

for all $(T, L, R) \in \mathcal{D}$ by

$$h_{p,q}(T, L, R) = (a(T) \bmod pq, a(L) \bmod p, a(R) \bmod p). \quad (3)$$

Our main lemma shows that \mathcal{D} is indeed extensively witnessed in the codomain $\mathbb{Z}_{pq} \times \mathbb{Z}_p \times \mathbb{Z}_p$. The sizes of p and q will be judiciously chosen as follows. Let $p_* = \binom{s/2}{s/4}/B$ and $q_* = \binom{s}{s/2}/(Bp_*)$.⁷ Let $\lambda = \log(p_*)/n \approx \sigma/2 - \log(B)/n$.

► **Lemma 3.** *Let p and q be independently chosen random primes in the range $[p_*, 2p_*]$ and $[q_*, 2q_*]$, respectively. Then with probability at least $1/2$, it holds that $|h_{p,q}(\mathcal{D})| \geq 2^{-(19+3\tau/\lambda)} s^{-6} p^3 q$.*

Observe in particular that the codomain has size $p^3 q$. We will prove Lemma 3 in §4.

3.3 The Search Subroutine

Once p and q have been fixed, we need a compatible search subroutine that carries out the search in a random preimage of $h_{p,q}^{-1}$. Towards this end, let us assume that p and q are fixed and coprime.

The high-level structure of the search subroutine is captured in the following lemma.

► **Lemma 4.** *Suppose that $k_T \in \mathbb{Z}_{pq}$, $k_L \in \mathbb{Z}_p$, and $k_R \in \mathbb{Z}_p$ have been chosen independently and uniformly at random. Then, there exists a randomized algorithm that searches for subsets $S \in \binom{[n]}{s}$ such that both of the following requirements hold:*

- (i) $a(S) = t$; and
- (ii) there exist sets $T \in \binom{S}{s/2}$, $L \in \binom{S}{s/4}$, and $R \in \binom{S \setminus T}{s/4}$ such that we have

$$a(T) \equiv k_T \pmod{pq}, \quad a(L) \equiv k_L \pmod{p}, \quad \text{and} \quad a(R) \equiv k_R \pmod{p}.$$

For every choice of k_T, k_L, k_R , and every subset S satisfying these conditions, the algorithm finds S with probability at least $1/n^2$ (over the internal randomness of the algorithm).

The expected running time of the algorithm over the choice of (k_T, k_L, k_R) is

$$O(n^2 \cdot 2^{\frac{1}{2}H(\sigma/4)} + n^2 \cdot 2^{H(\sigma/4)n}/p + n^2 B \cdot 2^{2H(\sigma/4)n}/(p^2 q)). \quad (4)$$

The running time bound holds uniformly for all coprime choices of p and q .

The proof of this lemma is given in §5.

⁷ For completeness, we round p_* to 1 if $p_* < 1$. However, in this case the running time becomes $\Omega\left(\binom{s/2}{s/4}^4\right)$, rendering the algorithm slow and uninteresting.

3.4 Completing the Proof of Theorem 1

We now combine Lemma 3 and Lemma 4 to yield the algorithm design for Theorem 1.

The algorithm starts by selecting a random independent pair p, q of primes with $p \in [p_*, 2p_*]$ and $q \in [q_*, 2q_*]$ and then selects uniform and independent $k_T \in \mathbb{Z}_{pq}$, $k_L \in \mathbb{Z}_p$, $k_R \in \mathbb{Z}_p$. Then we run the algorithm of Lemma 4 with these parameters. If we find a solution, the algorithm reports that solution. Otherwise the algorithm reports that the instance has no solution.

Let us now analyze the success probability and running time of the algorithm. The output of the algorithm is always correct if the instance has no solution, so let us assume that the instance has a solution S . Let $\alpha = 2^{-(19+3\tau/\lambda)}s^{-6}$, and note that α is inversely polynomial in the input size. From Lemma 3 we have that with probability at least $1/2$ we obtain a pair of primes p, q such that $|h_{p,q}(\mathcal{D})| \geq \alpha p^3 q$. Conditioning on this event, we have that $(k_T, k_L, k_R) \in h_{p,q}(\mathcal{D})$ with probability at least α . Conditioned on $(k_T, k_L, k_R) \in h_{p,q}(\mathcal{D})$, the algorithm of Lemma 4 finds the solution S with probability at least $1/n^2$. In total, the probability that the algorithm finds a solution S is at least $\alpha' := \alpha/(2n^2)$.

The algorithm runs in expected time

$$T = n^2 \cdot O\left(2^{\frac{1}{2}H(\sigma/4)n} + 2^{H(\sigma/4)n}/p + B \cdot 2^{2H(\sigma/4)n}/(p^2q)\right).$$

By Markov's inequality, with probability at most $\alpha'/2$ it runs in time at most $2T/\alpha'$, so even if we terminate it after $2T/\alpha'$ steps, it still has an $\alpha'/2$ chance of finding S . To amplify this to $1 - o(1)$ we repeat the whole procedure n/α' times, and the overall running time for the algorithm is (suppressing terms polynomial in n)

$$\begin{aligned} 2nT/(\alpha')^2 &= O^*\left(2^{\frac{1}{2}H(\sigma/4)n} + 2^{H(\sigma/4)n}/p + B \cdot 2^{2H(\sigma/4)n}/(p^2q)\right) \\ &= O^*\left(2^{\frac{1}{2}H(\sigma/4)n} + B2^{(H(\sigma/4)-\sigma/2)n} + B^4 \cdot 2^{(2H(\sigma/4)-3\sigma/2)n}\right), \end{aligned} \quad (5)$$

where we used that $p \geq p_* = \binom{s/2}{s/4}/B \geq 2^{s/2}/(Bs)$ and $q \geq q_* = \binom{s}{s/2}/(Bp_*) \geq 2^{s/2}/(Bs)$.

The first two summands in (5) are maximized in the range $0 \leq \sigma \leq 1/2$ when $\sigma = 1/2$ and are both bounded by $B2^{0.3n}$. The third summand is maximized when $\sigma = 4/9$ where it is roughly $B^4 2^{0.3399n}$, giving the claimed time bound in Theorem 1. The proof of Theorem 1 is now complete.

4 Analysis of the Hash Function

This section proves Lemma 3.

4.1 Size of the Image Under Bounded Collisions

Our first objective is to bound the size of the image $h_{p,q}(\mathcal{D})$ from below subject to the assumption that the parameters p, q have ‘‘few collisions’’ in a sense to be made precise under the assumptions in Lemma 5.

For a subset $X \subseteq S$, a modulus $M \in \mathbb{Z}_{\geq 1}$, and $k \in \mathbb{Z}_M$, let us count the number of halves of X that land in the congruence class of k modulo M by

$$f_{M,k}(X) = \left| \left\{ Y \in \binom{X}{\lfloor |X|/2 \rfloor} : a(Y) \equiv k \pmod{M} \right\} \right|. \quad (6)$$

Let us say that a set $X \subseteq S$ is γ -well-spread relative to a modulus $M \in \mathbb{Z}$ if

$$C_M(X) = \sum_{k \in \mathbb{Z}_M} f_{M,k}(X)^2 \leq \gamma \frac{\binom{|X|}{\lfloor |X|/2 \rfloor}^2}{M}. \quad (7)$$

Note that if $a(Y)$ over $Y \in \binom{X}{|X|/2}$ is evenly distributed over all M modular classes, we would have $C_M(X) = \binom{s}{s/2}^2 / M$.

► **Lemma 5.** *Let $p, q \in \mathbb{Z}_{\geq 1}$ be fixed so that*

- (i) *S is γ -well-spread relative to the modulus pq , and*
- (ii) *for at least half of all $T \in \binom{S}{s/2}$, it holds that both T and $S \setminus T$ are γ -well-spread relative to the modulus p .*

Then $|h_{p,q}(\mathcal{D})| \geq p^3 q / (2\gamma^3)$.

Proof. Let \mathcal{T}_p consist of all $T \in \binom{S}{s/2}$ such that both T and $S \setminus T$ are γ -well-spread relative to p . Let us write \mathcal{D}_p for the subfamily of \mathcal{D} consisting of all triples $(T, L, R) \in \mathcal{D}$ such that $T \in \mathcal{T}_p$. We thus have by assumption (ii) that

$$|\mathcal{D}_p| \geq \frac{1}{2} |\mathcal{D}| = \frac{1}{2} \binom{s}{s/2} \binom{s/2}{s/4}. \quad (8)$$

It suffices to establish the conclusion for $m = |h_{p,q}(\mathcal{D}_p)|$. Towards this end, let us analyze collisions of $h_{p,q}$ on \mathcal{D}_p . Let \mathcal{C}_p consist of all pairs $(T_1, L_1, R_1), (T_2, L_2, R_2) \in \mathcal{D}_p$ with $h_{p,q}(T_1, L_1, R_1) = h_{p,q}(T_2, L_2, R_2)$.

We start with a routine quadratic bound. Let c_1, c_2, \dots, c_m be the sizes of preimages of $h_{p,q}$ on \mathcal{D}_p . We have $|\mathcal{C}_p| = \sum_{i=1}^m c_i^2$ and $|\mathcal{D}_p| = \sum_{i=1}^m c_i$. By the Cauchy–Schwarz inequality we thus have

$$|h_{p,q}(\mathcal{D}_p)| = m \geq |\mathcal{D}_p|^2 / |\mathcal{C}_p|. \quad (9)$$

The claim thus follows by (8) and (9) if we can obtain sufficient control on $|\mathcal{C}_p|$. Recalling (3) and (6), we have

$$\begin{aligned} |\mathcal{C}_p| &= \sum_{\substack{T_1 \in \mathcal{T}_p \\ T_2 \in \mathcal{T}_p}} [a(T_1) \equiv_{pq} a(T_2)] \sum_{\substack{L_1 \in \binom{T_1}{s/4} \\ L_2 \in \binom{T_2}{s/4}}} [a(L_1) \equiv_p a(L_2)] \sum_{\substack{R_1 \in \binom{S \setminus T_1}{s/4} \\ R_2 \in \binom{S \setminus T_2}{s/4}}} [a(R_1) \equiv_p a(R_2)] \\ &= \sum_{\substack{T_1 \in \mathcal{T}_p \\ T_2 \in \mathcal{T}_p}} [a(T_1) \equiv_{pq} a(T_2)] \sum_{\ell \in \mathbb{Z}_p} f_{p,\ell}(T_1) f_{p,\ell}(T_2) \sum_{\ell \in \mathbb{Z}_p} f_{p,\ell}(S \setminus T_1) f_{p,\ell}(S \setminus T_2) \\ &\leq \underbrace{\sum_{\substack{T_1 \in \mathcal{T}_p \\ T_2 \in \mathcal{T}_p}} [a(T_1) \equiv_{pq} a(T_2)]}_{(a)} \underbrace{\max_{\substack{T_1 \in \mathcal{T}_p \\ T_2 \in \mathcal{T}_p}} \sum_{\ell \in \mathbb{Z}_p} f_{p,\ell}(T_1) f_{p,\ell}(T_2)}_{(b)} \underbrace{\max_{\substack{T_1 \in \mathcal{T}_p \\ T_2 \in \mathcal{T}_p}} \sum_{\ell \in \mathbb{Z}_p} f_{p,\ell}(S \setminus T_1) f_{p,\ell}(S \setminus T_2)}_{(c)}. \end{aligned}$$

Using (6) we can bound (a) from above by

$$\sum_{\substack{T_1 \in \mathcal{T}_p \\ T_2 \in \mathcal{T}_p}} [a(T_1) \equiv_{pq} a(T_2)] \leq \sum_{k \in \mathbb{Z}_{pq}} f_{pq,k}(S)^2 = C_{pq}(S).$$

Using the Cauchy–Schwarz inequality we can bound (b) from above by

$$\max_{\substack{T_1 \in \mathcal{T}_p \\ T_2 \in \mathcal{T}_p}} \sum_{\ell \in \mathbb{Z}_p} f_{p,\ell}(T_1) f_{p,\ell}(T_2) \leq \max_{T \in \mathcal{T}_p} \sum_{\ell \in \mathbb{Z}_p} f_{p,\ell}(T)^2 = \max_{T \in \mathcal{T}_p} C_p(T).$$

This bound on (b) applies also to (c) because we have $T \in \mathcal{T}_p$ if and only if $S \setminus T \in \mathcal{T}_p$. Combining the bounds on (a)–(c) and then using the conditions of S and every $T \in \mathcal{T}_p$ being

γ -well-spread, we conclude that

$$|\mathcal{C}_p| \leq C_{pq}(S) \cdot \left(\max_{T \in \mathcal{T}_p} C_p(T) \right)^2 \leq \gamma \frac{\binom{s}{s/2}}{pq} \left(\gamma \frac{\binom{s/2}{s/4}}{p} \right)^2 = \frac{\gamma^3 |\mathcal{D}|}{p^3 q}. \quad (10)$$

The lemma follows now follows from (8), (9), and (10). \blacktriangleleft

4.2 Bounded Collisions Happen with High Probability

This section shows that well-spread moduli are a high-probability event for $p, q \in \mathbb{Z}_{\geq 1}$ selected from an appropriate random ensemble.

Besides control on collisions over a modulus (7), we require control on collisions over the integers. This control is available via bounded concentration. For all $X \subseteq S$ define

$$C_\infty(X) = \sum_{Y_1, Y_2 \in \binom{X}{|X|/2}} [a(Y_1) = a(Y_2)]. \quad (11)$$

► **Lemma 6.** *For all $X \subseteq S$ we have $C_\infty(X) \leq \binom{|X|}{|X|/2} B$.*

Proof. From (11) we have that there are at most $\binom{|X|}{|X|/2}$ ways to select Y_1 . By B -bounded concentration, there are at most B ways to select an $Y_2 \in a^{-1}(a(Y_1))$. \blacktriangleleft

For the next lemma, we recall the parameter τ from §3.1, satisfying $a_1 + \dots + a_n + t \leq 2^{\tau n}$.

► **Lemma 7.** *Let \mathcal{M} be a nonempty set of integers each of which has all prime factors at least $2^{\lambda n}$ for some $\lambda > 0$. Suppose that we select an $M \in \mathcal{M}$ uniformly at random. Then, for any $T \subseteq S$ it holds that*

$$\mathbb{E}_{M \in \mathcal{M}} [C_M(T)] \leq \binom{|T|}{|T|/2} B + \frac{\binom{|T|}{|T|/2}^2}{|\mathcal{M}|} \cdot 2^{\tau/\lambda}.$$

Proof. Fix a nonempty $T \subseteq S$. For an arbitrary $M \in \mathcal{M}$ we have from (7) and (6) that

$$C_M(T) = \sum_{L_1, L_2 \in \binom{T}{|T|/2}} [a(L_1) \equiv_M a(L_2)] = \sum_{L_1, L_2 \in \binom{T}{|T|/2}} [M \text{ divides } a(L_1) - a(L_2)].$$

By linearity of expectation thus

$$\mathbb{E}_{M \in \mathcal{M}} [C_M(T)] = \sum_{L_1, L_2 \in \binom{T}{|T|/2}} \Pr_{M \in \mathcal{M}} [M \text{ divides } a(L_1) - a(L_2)]. \quad (12)$$

The terms in the sum (12) split into two cases. If $a(L_1) = a(L_2)$, then M divides $a(L_1) - a(L_2)$ with probability 1. From (11) we observe that the sum of these terms is exactly $C_\infty(T)$. Apply Lemma 6 to bound $C_\infty(T)$ from above. If $a(L_1) \neq a(L_2)$, then we observe that, by virtue of preprocessing, $|a(L_1) - a(L_2)| \leq 2^{\tau n}$. This implies that $a(L_1) - a(L_2)$ has at most τ/λ prime factors (with repetition) of size at least $2^{\lambda n}$. Any M that divides $a(L_1) - a(L_2)$ must be created from these τ/λ factors and hence there are at most $2^{\tau/\lambda}$ possible values for M . \blacktriangleleft

For the next lemma, recall that $p_* = \binom{s/2}{s/4}/B$ and $q_* = \binom{s}{s/2}/(Bp_*)$.

► **Lemma 8.** *With probability at least $1/2$, a random independent pair (p, q) of primes from $[p_*, 2p_*] \times [q_*, 2q_*]$ satisfies the assumptions of Lemma 5 with $\gamma = 2^{6+\tau/\lambda} s^2$, where $\lambda = \frac{\log p_*}{n}$.*

Proof. Let us start with assumption (ii) in Lemma 5. Take \mathcal{M} to be the set of primes p in the interval $[p_*, 2p_*]$. Define $Z(p) = \mathbb{E}_{T \in \binom{S}{s/2}} [C_p(T)]$. We then have

$$\begin{aligned} \mathbb{E}_{p \in \mathcal{M}} [Z(p)] &= \mathbb{E}_{p \in \mathcal{M}} [\mathbb{E}_{T \in \binom{S}{s/2}} [C_p(T)]] = \mathbb{E}_{T \in \binom{S}{s/2}} [\mathbb{E}_{p \in \mathcal{M}} [C_p(T)]] \\ &\leq \binom{s/2}{s/4} B + \frac{\binom{s/2}{s/4}^2}{|\mathcal{M}|} \cdot 2^{\tau/\lambda} = \binom{s/2}{s/4}^2 \left(\frac{1}{p_*} + \frac{2^{\tau/\lambda}}{|\mathcal{M}|} \right), \end{aligned}$$

where the inequality is Lemma 7. Applying Markov's inequality and $|\mathcal{M}| \geq p_*/\log(p_*)$ (due to Lemma 2), we conclude that with probability $\geq 3/4$ over a random $p \in \mathcal{M}$ it holds that

$$Z(p) \leq 4 \binom{s/2}{s/4}^2 \cdot \frac{1 + \log(p_*)}{p_*} 2^{\tau/\lambda} \leq 8s 2^{\tau/\lambda} \frac{\binom{s/2}{s/4}^2}{p}.$$

Conditioning on such a p , at least a $3/4$ fraction of $T \in \binom{S}{s/2}$ satisfies $C_p(T) \leq 32s 2^{\tau/\lambda} \binom{s/2}{s/4}^2 / p$. Since $T \in \binom{S}{s/2}$ if and only if $S \setminus T \in \binom{S}{s/2}$, we get by the union bound that half of all $T \in \binom{S}{s/2}$ satisfy

$$C_p(T) \leq 32s 2^{\tau/\lambda} \binom{s/2}{s/4}^2 / p, \quad C_p(S \setminus T) \leq 32s 2^{\tau/\lambda} \binom{s/2}{s/4}^2 / p. \quad (13)$$

We thus conclude that that assumption (ii) in Lemma 5 holds with probability at least $3/4$ over $p \in \mathcal{M}$.

Next let us consider assumption (i) in Lemma 5. We now take \mathcal{M} to be the set of products pq of primes p in the interval $[p_*, 2p_*]$ and primes q in the interval $[q_*, 2q_*]$. By Lemma 2, we have $|\mathcal{M}| \geq \frac{p_* q_*}{\log(p_*) \log(q_*)} \geq p_* q_* / s^2$, so Lemma 7 implies

$$\mathbb{E}_{pq \in \mathcal{M}} [C_{pq}(S)] \leq \binom{s}{s/2} B + \binom{s}{s/2}^2 \frac{s^2 2^{\tau/\lambda}}{p_* q_*} = \binom{s}{s/2}^2 \left(\frac{1}{p_* q_*} + \frac{s^2 2^{\tau/\lambda}}{p_* q_*} \right).$$

Markov's inequality then implies that with probability at least $3/4$ over $pq \in \mathcal{M}$, we have

$$C_{pq}(S) \leq 64s^2 2^{\tau/\lambda} \frac{\binom{s}{s/2}^2}{pq}. \quad (14)$$

Taking the union bound, we conclude that with probability at least $1/2$ over the choice of p, q both assumptions (i) and (ii) in Lemma 5 hold, with $\gamma = 64s^2 2^{\tau/\lambda}$. \blacktriangleleft

4.3 Combining the Two Parts

By Lemma 8, we have for a random independent pair p, q that S is γ -well-spread relative to the modulus pq , and for at least half of all $T \in \binom{S}{s/2}$, it holds that both T and $S \setminus T$ are γ -well-spread relative to the modulus p , with $\gamma = 2^{6+\tau/\lambda} s^2$. Then Lemma 5 gives that $|h_{p,q}(\mathcal{D})| \geq p^3 q / (2(2^{6+\tau/\lambda} s^2)^3) = 2^{-(19+3\tau/\lambda)} s^{-6} p^3 q$.

5 The Search Subroutine

Our goal in this section is to build the search subroutine in Lemma 4. We build the subroutine from the bottom up, starting in §5.1 from subroutines that build bottom-level candidate partial solutions of size $s/4$, then in §5.2 proceeding to the mid-level subroutines that assemble the candidate partial solutions of size $s/2$, and finally arrive in §5.3 at root-level node that assembles all the sets S of size s required by Lemma 4.

5.1 Subroutine for Bottom-Level Nodes

Let $p \in \mathbb{Z}_{\geq 1}$ be fixed. The following subroutine is executed in the four bottom-level nodes.

► **Lemma 9.** *There is a randomized algorithm for listing solutions $Z \in \binom{[n]}{s/4}$ to $a(Z) \equiv k \pmod{p}$ with the following properties:*

- (i) *For every fixed k , every solution Z gets listed by the algorithm with probability at least $1/\sqrt{n}$ (over the internal randomness of the algorithm).*
- (ii) *If k is picked uniformly at random, the expected running time of the algorithm over the choice of k is $O(n^2 \binom{n/2}{s/8} + n^2 \binom{n/2}{s/8}^2 / p)$ and the expected number of solutions found is at most $\binom{n/2}{s/8}^2 / p$.*

Proof. The algorithm starts by picking a random subset $\mathcal{N} \in \binom{[n]}{n/2}$.

Next, we construct $\mathcal{L} = \binom{\mathcal{N}}{s/8}$, $\mathcal{R} = \binom{[n] \setminus \mathcal{N}}{s/8}$, the subsets of \mathcal{N} and $[n] \setminus \mathcal{N}$ of size $s/8$. For each $X \in \mathcal{L} \cup \mathcal{R}$ we compute the residue of $a(X)$ modulo p and sort the two lists in increasing order of residue.

Initialize \mathcal{S} to an empty list. Using the sorted lists, for each $j \in \mathbb{Z}_p$ do the following. Iterate over all $X \in \mathcal{L}$ such that $a(X) \equiv j \pmod{p}$ and over all $Y \in \mathcal{R}$ such that $a(Y) \equiv k - j \pmod{p}$. (Note that we do this implicitly by simultaneously scanning the two lists, not with an explicit loop that considers each j in turn.) For each such pair we append $X \cup Y$ to \mathcal{S} .

Let us now analyze success probability and running time.

For success probability, note that any fixed solution $Z \in \binom{[n]}{s/4}$ gets split perfectly by \mathcal{N} (i.e., $|\mathcal{N} \cap Z| = s/8$) with probability $\binom{n/2}{s/8}^2 / \binom{n}{s/4} \geq 1/\sqrt{n}$ over the choice of \mathcal{N} . Whenever this happens, the algorithm finds Z .

For the running time, constructing the lists \mathcal{L} and \mathcal{R} can be done with brute force in $O(n^2 \binom{n/2}{s/8})$ time and space.⁸ The running time of the merge step is bounded by $O(n^2)$ times the number of solutions, which on average over a random $k \in \mathbb{Z}_p$ is $\binom{n/2}{s/8}^2 / p$ (since the total number of solutions over all $k \in \mathbb{Z}_p$ is simply $|\mathcal{L}| \cdot |\mathcal{R}|$). ◀

5.2 Subroutine for Mid-Level Nodes

We now proceed to the subroutine executed by the two mid-level nodes. Let $M = pq$ for two distinct primes p, q .

► **Lemma 10.** *There is a randomized algorithm for listing solutions $(X, Y) \in \binom{[n]}{s/4}^2$ to $a(X) + a(Y) \equiv k_M \pmod{M}$ and $a(X) \equiv k_p \pmod{p}$ with $X \cap Y = \emptyset$, with the following properties:*

- (i) *For every fixed k_M, k_p , every solution (X, Y) gets listed by the algorithm with probability at least $1/n$ (over the internal randomness of the algorithm).*
- (ii) *If k_M, k_p are picked uniformly at random from \mathbb{Z}_M and \mathbb{Z}_p , respectively, the expected running time of the algorithm over the choice of k_M, k_p is $O(n^2 \binom{n/2}{s/8} + n^2 \binom{n/2}{s/8}^2 / p + n^2 \binom{n/2}{s/8}^4 / (Mp))$ and the expected number of solutions found is at most $\binom{n/2}{s/8}^4 / (Mp)$.*

Proof. The algorithm starts by picking $k_p \in \mathbb{Z}_p$ uniformly at random. It then executes the algorithm of Lemma 9 twice, with parameters $k = k_p$ and $k = (k_M - k_p) \pmod{p}$, yielding

⁸ We are sorting $2^{\Omega(n)}$ items and comparing each pair of items takes $\Omega(n)$ time.

two lists \mathcal{L} (of solutions $Z \in \binom{n}{s/8}$ to $a(Z) \equiv k_p \pmod{p}$) and \mathcal{R} (of solutions $Z \in \binom{n}{s/8}$ to $a(Z) \equiv k_M - k_p \pmod{p}$).

We now merge these in a similar way as the proof of Lemma 9, except that we filter out all (X, Y) which are not disjoint and simply don't add them to the result.

The success probability follows because the two calls to the bottom-level algorithm are independent in terms of the internal randomness used by the calls.

For the running time, note that the two parameters k_p and $(k_M - k_p) \bmod p$ to the bottom-level algorithm are both uniformly random so that the expected sizes (over k_p and k_M) of \mathcal{L} and \mathcal{R} are at most $\binom{n/2}{s/8}^2 / p$. Furthermore, since p divides M , $k_M \bmod p$ is uniformly random and the two parameters are independent, implying that the expectation of $|\mathcal{L}| \cdot |\mathcal{R}|$ is at most $\binom{n/2}{s/8}^4 / p^2$.

By the Chinese Remainder Theorem, $k_M \bmod q$ is independent of $k_M \bmod p$, so conditioned on k_p , $k_M \bmod p$, \mathcal{L} and \mathcal{R} , the expected running time of the merge step is $O(|\mathcal{L}| \cdot |\mathcal{R}| / q)$. Thus in overall expectation over the choice of k_p and k_M , the running time is $\binom{n/2}{s/8}^4 / (p^2 q)$. ◀

5.3 The Root-Level Node

We are now ready to complete the proof of Lemma 4.

Proof of Lemma 4. Apply the algorithm of Lemma 10 twice, first with parameters $(k_M, k_p) = (k_T, k_L)$, then with parameters $(k_M, k_p) = ((t - k_T) \bmod M, k_R)$. Thus we get a list \mathcal{L} of solutions $(X, Y) \in \binom{[n]}{s/4}^2$ such that $X \cap Y = \emptyset$, $a(X) + a(Y) \equiv k_M \pmod{M}$ and $a(X) \equiv k_p \pmod{p}$. Similarly we have a list \mathcal{R} of solutions $(Z, W) \in \binom{[n]}{s/4}^2$ such that $Z \cap W = \emptyset$, $a(Z) + a(W) \equiv t - k_M \pmod{M}$, $a(Z) \equiv k_R \pmod{p}$.

We now merge these lists to construct solutions $(X, Y, Z, W) \in \binom{[n]}{s}$ to $a(X) + a(Y) + a(Z) + a(W) = t$, and if $(X \cup Y) \cap (Z \cup W) = \emptyset$, we output the solution $X \cup Y \cup Z \cup W$.

For every fixed choice of (k_T, k_L, k_R) and solution $X \cup Y \cup Z \cup W$, the probability that we find the solution is at least $1/n^2$ by independence of the internal randomness employed by the two applications of Lemma 10.

By B -bounded concentration, for any choice $a(X) + a(Y)$ from \mathcal{L} , there are at most B solutions (Z, W) from \mathcal{R} to $a(Z) + a(W) = t - a(X) - a(Y)$. This implies that the merge step runs in time $O(n^2(|\mathcal{L}| + |\mathcal{R}|)B)$. Since in expectation over the targets k the sizes of \mathcal{L} and \mathcal{R} are bounded by $\binom{n/2}{s/8}^4 / (Mp)$, it follows that the merge step runs in expected time $O(n^2 \binom{n/2}{s/8}^4 B / (Mp))$.

Substituting $s = \sigma n$ and bounding the binomial coefficients from above with Stirling's formula [20], we obtain the desired running time bound (4) required by Lemma 4. This completes the proof of Lemma 4. ◀

Acknowledgements This research was funded in part by the European Research Council, Advanced Grant 226203 and Swedish Research Council, Grant 621-2012-4546 (P.A.), the European Research Council, Starting Grant 338077 “Theory and Practice of Advanced Search and Enumeration” (P.K.), the Academy of Finland, Grant 276864 “Supple Exponential Algorithms” (M.K.), and by the NWO VENI project 639.021.438 (J.N.).

References

- 1 Dimitris Achlioptas. Random satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 245–270. IOS Press, 2009.
- 2 Per Austrin, Petteri Kaski, Mikko Koivisto, and Jussi Määttä. Space-time tradeoffs for Subset Sum: An improved worst case algorithm. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *ICALP (1)*, volume 7965 of *Lecture Notes in Computer Science*, pages 45–56. Springer, 2013.
- 3 Anja Becker. *The Representation Technique: Application to Hard Problems in Cryptography*. PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, 2012.
- 4 Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 364–385. Springer, 2011.
- 5 Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, N. J., 1957.
- 6 Khodakhast Bibak. Additive combinatorics: With a view towards computer science and cryptography - an exposition. In Jonathan M. Borwein, Igor Shparlinski, and Wadim Zudilin, editors, *Number Theory and Related Fields*, pages 99–128. Springer, 2013.
- 7 Matthijs J. Coster, Antoine Joux, Brian A. Lamacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved low-density subset sum algorithms. *Computational Complexity*, 2:111–128, 1992.
- 8 Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 719–740. Springer, 2012.
- 9 Michael R. Fellows, Serge Gaspers, and Frances A. Rosamond. Parameterizing by the number of numbers. *Theory Comput. Syst.*, 50(4):675–693, 2012.
- 10 Abraham D. Flaxman and Bartosz Przydatek. Solving medium-density subset sum problems in expected polynomial time. In Volker Diekert and Bruno Durand, editors, *STACS 2005*, volume 3404 of *Lecture Notes in Computer Science*, pages 305–314. Springer Berlin Heidelberg, 2005.
- 11 Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.
- 12 Godfrey H. Hardy and Edward M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, Oxford, sixth edition, 2008. Revised by D. R. Heath-Brown and J. H. Silverman, With a foreword by Andrew Wiles.
- 13 Ellis Horowitz and Sartaj Sahni. Computing partitions with applications to the knapsack problem. *J. Assoc. Comput. Mach.*, 21:277–292, 1974.
- 14 Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 235–256. Springer, 2010.
- 15 Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9(4):199–216, 1996.
- 16 Petteri Kaski, Mikko Koivisto, and Jesper Nederlof. Homomorphic hashing for sparse coefficient extraction. In Dimitrios M. Thilikos and Gerhard J. Woeginger, editors, *IPEC*, volume 7535 of *Lecture Notes in Computer Science*, pages 147–158. Springer, 2012.
- 17 Jeffrey C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *J. ACM*, 32(1):229–246, 1985.
- 18 Daniel Lokshtanov and Jesper Nederlof. Saving space by algebraization. In Leonard J. Schulman, editor, *STOC*, pages 321–330. ACM, 2010.

- 19 Phong Q. Nguyen, Igor E. Shparlinski, and Jacques Stern. Distribution of modular sums and the security of the server aided exponentiation. In Kwok-Yan Lam, Igor Shparlinski, Huaxlong Wang, and Chaoping Xing, editors, *Cryptography and Computational Number Theory*, volume 20 of *Progress in Computer Science and Applied Logic*, pages 331–342. Birkhäuser, 2001.
- 20 Herbert Robbins. A remark on Stirling’s formula. *The American Mathematical Monthly*, 62(1):26–29, 1955.
- 21 Richard Schroepel and Adi Shamir. A $T = O(2^{n/2})$, $S = O(2^{n/4})$ algorithm for certain NP-complete problems. *SIAM J. Comput.*, 10(3):456–464, 1981.
- 22 Terence Tao. The dichotomy between structure and randomness, arithmetic progressions, and the primes. In *International Congress of Mathematicians. Vol. I*, pages 581–608. Eur. Math. Soc., Zürich, 2007.
- 23 Terence Tao. Structure and randomness in combinatorics. In *FOCS*, pages 3–15. IEEE Computer Society, 2007.
- 24 Terence Tao. *Structure and Randomness*. American Mathematical Society, Providence, RI, 2008.
- 25 Terence Tao and Van Vu. *Additive Combinatorics*, volume 105 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, Cambridge, 2006.
- 26 Terence Tao and Van Vu. A sharp inverse Littlewood-Offord theorem. *Random Structures Algorithms*, 37(4):525–539, 2010.
- 27 Luca Trevisan. Pseudorandomness in computer science and in additive combinatorics. In *An irregular mind*, volume 21 of *Bolyai Soc. Math. Stud.*, pages 619–650. János Bolyai Math. Soc., Budapest, 2010.
- 28 Gerhard J. Woeginger. Exact algorithms for NP-hard problems: A survey. In Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi, editors, *Combinatorial Optimization*, volume 2570 of *Lecture Notes in Computer Science*, pages 185–208. Springer, 2001.

On Sharing, Memoization, and Polynomial Time*

Martin Avanzini and Ugo Dal Lago

Università di Bologna & INRIA, Sophia Antipolis
martin.avanzini@uibk.ac.at and dallago@cs.unibo.it

Abstract

We study how the adoption of an evaluation mechanism with sharing and memoization impacts the class of functions which can be computed in polynomial time. We first show how a natural cost model in which lookup for an already computed result has no cost is indeed invariant. As a corollary, we then prove that the most general notion of ramified recurrence is sound for polynomial time, this way settling an open problem in implicit computational complexity.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.3.2 Semantics of Programming Languages, F.4.1 Mathematical Logic, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases implicit computational complexity, data-tiering, polynomial time

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.62

1 Introduction

Traditionally, complexity classes are defined by giving bounds on the amount of resources algorithms are allowed to use while solving problems. This, in principle, leaves open the task of understanding the *structure* of complexity classes. As an example, a given class of functions is not necessarily closed under composition or, more interestingly, under various forms of recursion. When the class under consideration is not too large, say close enough to the class of *polytime computable functions*, closure under recursion does not hold: iterating over an efficiently computable function is not necessarily efficiently computable, e.g. when the iterated function grows more than linearly. In other words, characterizing complexity classes by purely recursion-theoretical means is non-trivial.

In the past twenty years, this challenge has been successfully tackled, by giving *restricted* forms of recursion for which not only certain complexity classes are closed, but which *precisely* generate the class. This has been proved for classes like PTIME, PSPACE, the polynomial hierarchy PH, or even smaller ones like NC. A particularly fruitful direction has been the one initiated by Bellantoni and Cook, and independently by Leivant, which consists in restricting the primitive recursive scheme by making it *predicative*, thus forbidding those nested recursive definitions which lead outside the classes cited above. Once this is settled, one can tune the obtained scheme by either adding features (e.g. parameter substitutions) or further restricting the scheme (e.g. by way of linearization).

Something a bit disappointing in this field is that the expressive power of the simplest (and most general) form of predicative recurrence, namely *simultaneous* recurrence on *generic algebras* is unknown. If algebras are restricted to be *string* algebras, or if recursion is not simultaneous, soundness for polynomial time computation is known to hold [21, 16]. The two soundness results are obtained by quite different means, however: in presence of trees, one is

* This work was partially supported by FWF project number J 3563 and by French ANR project Elica ANR-14-CE25-0005.

forced to handle *sharing* [16] of common sub-expressions, while simultaneous definitions by recursion requires a form of *memoization* [21].

In this paper, we show that sharing and memoization can indeed be reconciled, and we exploit both to give a new invariant time cost model for the evaluation of rewrite systems. This paves the way towards polytime soundness for simultaneous predicative recursion on generic algebras, thus solving the open problem we were mentioning. More precisely, with the present paper we make the following contributions:

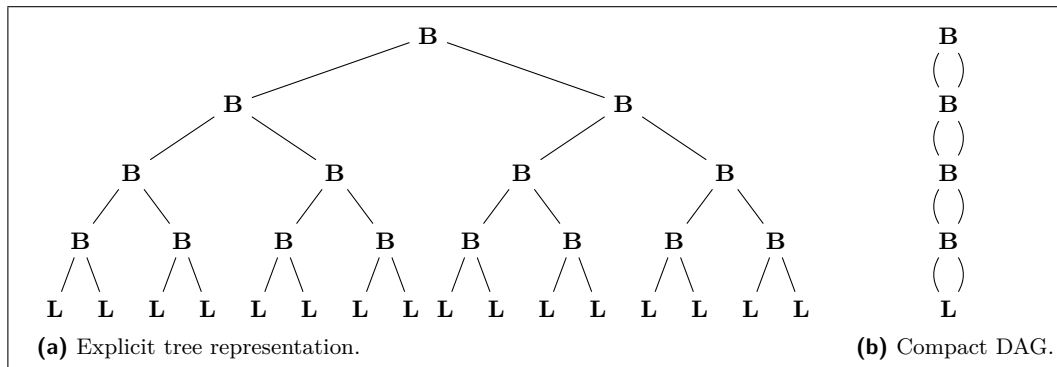
1. We define a simple functional programming language. The domain of the defined functions is a free algebra formed from constructors. Hence we can deal with functions over strings, lists, but also trees (see Section 3). We then extend the underlying rewriting based semantics with *memoization*, i.e. intermediate results are automatically tabulated to avoid expensive re-computation (Section 4). As standard for functional programming languages such as Haskell or OCaml, data is stored in a *heap*, facilitating *sharing* of common sub-expression. To measure the *runtime* of such programs, we employ a novel cost model, called *memoized runtime complexity*, where each function application counts one time unit, but lookups of tabulated calls do not have to be accounted.
2. Our *invariance theorem* (see Theorem 17) relates, within a polynomial overhead, the memoized runtime complexity of programs to the cost of implementing the defined functions on a classical model of computation, e.g. *Turing* or *random access machines*. The invariance theorem thus confirms that our cost model truthfully represents the computational complexity of the defined function.
3. We extend upon Leivant's notion of *ramified recursive functions* [20] by allowing definitions by *generalised ramified simultaneous recurrence* (*GRSR* for short). We show that the resulting class of functions, defined over arbitrary free algebras have, when implemented as programs, polynomial memoized runtime complexity (see Theorem 21). By our invariance theorem, the function algebra is sound for polynomial time, and consequently GRSR characterizes the class of polytime computable functions.

An extended version of this paper with more details, including all proofs, is also available [4].

1.1 Related Work

That predicative recursion *on strings* is sound for polynomial time, even in presence of simultaneous recursive definitions, is known for a long time [9]. Variations of predicative recursion have been later considered and proved to characterize classes like PH [10], PSPACE [23], EXPTIME [3] or NC [12]. Predicative recursion on trees has been claimed to be sound for polynomial time in the original paper by Leivant [20], the long version of which only deals with strings [21]. After fifteen years, the non-simultaneous case has been settled by the second author with Martini and Zorzi [16]; their proof, however, relies on an ad-hoc, infinitary, notion of graph rewriting. Recently, ramification has been studied in the context of a simply-typed λ -calculus in an unpublished manuscript [17]; the authors claim that a form of ramified recurrence on trees captures polynomial time; this, again, does not take simultaneous recursion into account.

The formalism presented here is partly inspired by the work of Hoffmann [19], where sharing and memoization are shown to work well together in the realm of term graph rewriting. The proposed machinery, although powerful, is unnecessarily complicated for our purposes. Speaking in Hoffmann's terms, our results require a form of full memoization, which *is* definable in Hoffmann's system. However, most crucially for our concerns, it is unclear how the overall system incorporating full memoization can be implemented efficiently, if at all.



■ **Figure 1** Complete Binary Tree of Height Four, as Computed by $\text{tree}(\mathbf{S}^4(\mathbf{0}))$.

2 The Need for Sharing and Memoisation

This Section is an informal, example-driven, introduction to ramified recursive definitions and their complexity. Our objective is to convince the reader that those definitions do *not* give rise to polynomial time computations if naively evaluated, and that sharing and memoization are *both* necessary to avoid exponential blowups.

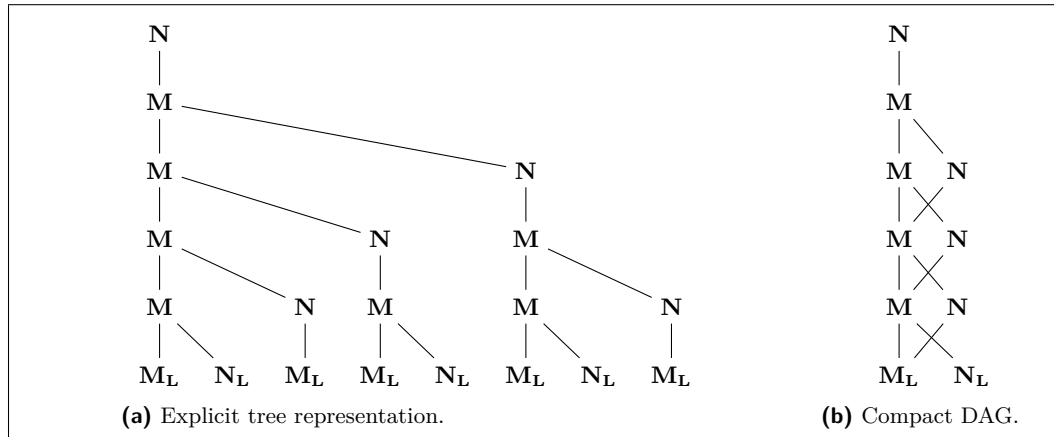
In Leivant’s system [21], functions and variables are equipped with a *tier*. Composition must preserve tiers and, crucially, in a function defined by primitive recursion the tier of the recurrence parameter must be higher than the tier of the recursive call. This form of *ramification* of functions effectively tames primitive recursion, resulting in a characterisation of the class of *polytime computable functions*.

Of course, ramification also controls the growth rate of functions. However, as soon as we switch from strings to a domain where tree structures are definable, this control is apparently lost. For illustration, consider the following definition:

$$\text{tree}(\mathbf{0}) = \mathbf{L} \quad \text{tree}(\mathbf{S}(n)) = \text{br}(\text{tree}(n)) \quad \text{br}(t) = \mathbf{B}(t, t) .$$

The function tree is defined by primitive recursion, essentially from basic functions. As a consequence, it is easily seen to be ramified in the sense of Leivant. Even though the number of recursive steps is linear in the input, the result of $\text{tree}(\mathbf{S}^n(\mathbf{0}))$ is the complete binary tree of height n . As thus the length of the output is exponential in the one of its input, there is, at least apparently, little hope to prove tree a polytime function. The way out is *sharing*: the complete binary tree of height n can be compactly represented as a *directed acyclic graph* (DAG for short) of linear size (see Figure 1). Indeed, using the compact DAG representation it is easy to see that the function tree is computable in polynomial time. This is the starting point of [16], in which general ramified recurrence is proved sound for polynomial time. A crucial observation here is that *not only* the output’s size, but also the total amount of work can be kept under control, thanks to the fact that evaluating a primitive recursive definition on a compactly represented input can be done by constructing an isomorphic DAG of recursive calls.

This does not scale up to *simultaneous* ramified recurrence. The following example computes the genealogical tree associated with *Fibonacci’s rabbit problem* for $n \in \mathbb{N}$ generations. Rabbits come in pairs. After one generation, each *baby* rabbit pair (\mathbf{N}) matures. In each



■ **Figure 2** Genealogical Rabbit Tree up to the Sixth Generation, as Computed by $\text{rabbits}(\mathbf{S}^6(\mathbf{0}))$.

generation, an *adult* rabbit pair (**M**) bears one pair of babies.

$$\begin{aligned} \text{rabbits}(\mathbf{0}) &= \mathbf{N}_L & \mathbf{a}(\mathbf{0}) &= \mathbf{M}_L & \mathbf{b}(\mathbf{0}) &= \mathbf{N}_L \\ \text{rabbits}(\mathbf{S}(n)) &= \mathbf{b}(n) & \mathbf{a}(\mathbf{S}(n)) &= \mathbf{M}(\mathbf{a}(n), \mathbf{b}(n)) & \mathbf{b}(\mathbf{S}(n)) &= \mathbf{N}(\mathbf{a}(n)) . \end{aligned}$$

The function `rabbits` is obtained by case analysis from the functions `a` and `b`, which are defined by *simultaneous* primitive recursion: the former recursively calls itself *and* the latter, while the latter makes a recursive call to the former. The output of $\text{rabbits}(\mathbf{S}^n(\mathbf{0}))$ is tightly related to the sequence of Fibonacci numbers: the number of nodes at depth i is given by the i^{th} Fibonacci number. Hence the output tree has exponential size in n but, again, can be represented compactly (see Figure 2). This does not suffice for our purposes, however. In presence of simultaneous definitions, indeed, avoiding re-computation of previously computed values becomes more difficult, the trick described above does not work, and the key idea towards that is the use of *memoization*.

What we prove in this paper is precisely that sharing and memoization can indeed be made to work together, and that they together allow to prove polytime soundness for all ramified recursive functions, also in presence of tree algebras and simultaneous definitions.

3 Preliminaries

General Ramified Simultaneous Recurrence

Let \mathbb{A} denote a (*finite and untyped*) signatures of constructors $\mathbf{c}_1, \dots, \mathbf{c}_k$, each equipped with an arity $\text{ar}(\mathbf{c}_i)$. In the following, the set of terms $\mathcal{T}(\mathbb{A})$ over the signature \mathbb{A} , defined as usual, is also denoted by \mathbb{A} if this does not create ambiguities. We are interested in total functions from $\mathbb{A}^n = \underbrace{\mathbb{A} \times \dots \times \mathbb{A}}_{n \text{ times}}$ to \mathbb{A} .

- **Definition 1.** The following are so-called *basic functions*:
 - For each constructor \mathbf{c} , the *constructor function* $\mathbf{f}_{\mathbf{c}} : \mathbb{A}^{\text{ar}(\mathbf{c})} \rightarrow \mathbb{A}$ for \mathbf{c} , defined as follows: $\mathbf{f}_{\mathbf{c}}(x_1, \dots, x_{\text{ar}(\mathbf{c})}) = \mathbf{c}(x_1, \dots, x_{\text{ar}(\mathbf{c})})$
 - For each $1 \leq n \leq m$, the (m, n) -*projection function* $\Pi_n^m : \mathbb{A}^m \rightarrow \mathbb{A}$ defined as follows: $\Pi_n^m(x_1 \dots, x_m) = x_n$.

$\frac{}{\mathbf{f}_c \triangleright \mathbb{A}_n^{\text{ar}(c)} \rightarrow \mathbb{A}_n}$	$\frac{}{\prod_m^n \triangleright \mathbb{A}_{p_1} \times \dots \times \mathbb{A}_{p_m} \rightarrow \mathbb{A}_{p_n}}$	$\frac{\mathbf{f}_i \triangleright \mathbb{A}_p^{\text{ar}(c_i)} \times \mathbf{A} \rightarrow \mathbb{A}_m}{\text{case}(\{\mathbf{f}_i\}_{1 \leq i \leq k}) \triangleright \mathbb{A}_p \times \mathbf{A} \rightarrow \mathbb{A}_m}$
$\frac{\mathbf{f} \triangleright \mathbb{A}_{p_1} \times \dots \times \mathbb{A}_{p_n} \rightarrow \mathbb{A}_m \quad \mathbf{g}_i \triangleright \mathbf{A} \rightarrow \mathbb{A}_{p_i}}{\mathbf{f} \circ (\mathbf{g}_1, \dots, \mathbf{g}_n) \triangleright \mathbf{A} \rightarrow \mathbb{A}_m}$		$\frac{\mathbf{f}_i^j \triangleright \mathbb{A}_p^{\text{ar}(c_i)} \times \mathbb{A}_m^{n \cdot \text{ar}(c_i)} \times \mathbf{A} \rightarrow \mathbb{A}_m \quad p > m}{\text{simrec}(\{\mathbf{f}_i^j\}_{1 \leq i \leq k, 1 \leq j \leq n}) \triangleright \mathbb{A}_p \times \mathbf{A} \rightarrow \mathbb{A}_m}$

■ **Figure 3** Tiering as a Formal System.

► **Definition 2.**

- Given a function $\mathbf{f} : \mathbb{A}^n \rightarrow \mathbb{A}$ and n functions $\mathbf{g}_1, \dots, \mathbf{g}_n$, all of them from \mathbb{A}^m to \mathbb{A} , the *composition* $\mathbf{h} = \mathbf{f} \circ (\mathbf{g}_1, \dots, \mathbf{g}_n)$ is a function from \mathbb{A}^m to \mathbb{A} defined as follows: $\mathbf{h}(\vec{x}) = \mathbf{f}(\mathbf{g}_1(\vec{x}), \dots, \mathbf{g}_n(\vec{x}))$.
- Suppose given the functions \mathbf{f}_i where $1 \leq i \leq k$ such that for some m , $\mathbf{f}_i : \mathbb{A}^{\text{ar}(c_i)} \times \mathbb{A}^n \rightarrow \mathbb{A}$. Then the function $\mathbf{g} = \text{case}(\{\mathbf{f}_i\}_{1 \leq i \leq k})$ defined by *case distinction* from $\{\mathbf{f}_i\}_{1 \leq i \leq k}$ is a function from $\mathbb{A} \times \mathbb{A}^n$ to \mathbb{A} defined as follows: $\mathbf{g}(c_i(\vec{x}), \vec{y}) = \mathbf{f}_i(\vec{x}, \vec{y})$.
- Suppose given the functions \mathbf{f}_i^j , where $1 \leq i \leq k$ and $1 \leq j \leq n$, such that for some m , $\mathbf{f}_i^j : \mathbb{A}^{\text{ar}(c_i)} \times \mathbb{A}^{n \cdot \text{ar}(c_i)} \times \mathbb{A}^m \rightarrow \mathbb{A}$. The functions $\{\mathbf{g}_j\}_{1 \leq j \leq n} = \text{simrec}(\{\mathbf{f}_i^j\}_{1 \leq i \leq k, 1 \leq j \leq n})$ defined by *simultaneous primitive recursion* from $\{\mathbf{f}_i^j\}_{1 \leq i \leq k, 1 \leq j \leq n}$ are all functions from $\mathbb{A} \times \mathbb{A}^m$ to \mathbb{A} such that for $\vec{x} = x_1, \dots, x_{\text{ar}(c_i)}$,

$$\mathbf{g}_j(c_i(\vec{x}), \vec{y}) = \mathbf{f}_i^j(\vec{x}, \mathbf{g}_1(x_1, \vec{y}), \dots, \mathbf{g}_1(x_{\text{ar}(c_i)}, \vec{y}), \dots, \mathbf{g}_n(x_1, \vec{y}), \dots, \mathbf{g}_n(x_{\text{ar}(c_i)}, \vec{y}), \vec{y}) .$$

We denote by $\text{SIMREC}(\mathbb{A})$ the class of *simultaneous recursive functions over* \mathbb{A} , defined as the smallest class containing the basic functions of Definition 1 and that is closed under the schemes of Definition 2.

Tiering, the central notion underlying Leivant’s definition of *ramified recurrence*, consists in attributing *tiers* to inputs and outputs of some functions among the ones constructed as above, with the goal of isolating the polytime computable ones. Roughly speaking, the role of tiers is to single out “a copy” of the signature by a level: this level permits to control the recursion nesting. Tiering can be given as a formal system, in which judgments have the form $\mathbf{f} \triangleright \mathbb{A}_{p_1} \times \dots \times \mathbb{A}_{p_{\text{ar}(\mathbf{f})}} \rightarrow \mathbb{A}_m$ for $p_1, \dots, p_{\text{ar}(\mathbf{f})}, m$ natural numbers and $\mathbf{f} \in \text{SIMREC}(\mathbb{A})$. The system is defined in Figure 3, where \mathbf{A} denotes the expression $\mathbb{A}_{q_1} \times \dots \times \mathbb{A}_{q_k}$ for some $q_1, \dots, q_k \in \mathbb{N}$. Notice that composition preserves tiers. Moreover, recursion is allowed only on inputs of tier higher than the tier of the function (in the case $\mathbf{f} = \text{simrec}(\{\mathbf{f}_i^j\}_{1 \leq i \leq k, 1 \leq j \leq n})$, we require $p > m$).

► **Definition 3.** We call a function $\mathbf{f} \in \text{SIMREC}(\mathbb{A})$ definable by *general ramified simultaneous recurrence* (*GRSR* for short) if $\mathbf{f} \triangleright \mathbb{A}_{p_1} \times \dots \times \mathbb{A}_{p_{\text{ar}(\mathbf{f})}} \rightarrow \mathbb{A}_m$ holds.

► **Remark.** Consider the *word algebra* $\mathbb{W} = \{\epsilon, \mathbf{a}, \mathbf{b}\}$ consisting of a constant ϵ and two unary constructors \mathbf{a} and \mathbf{b} , which is in bijective correspondence to the set of binary words. Then the functions definable by ramified simultaneous recurrence over \mathbb{W} includes the ramified recursive functions from Leivant [21], and consequently all polytime computable functions.

► **Example 4.**

1. Consider $\mathbb{N} := \{\mathbf{0}, \mathbf{S}\}$ with $\text{ar}(\mathbf{0}) = 0$ and $\text{ar}(\mathbf{S}) = 1$, which is in bijective correspondence to the set of natural numbers. We can define addition $\text{add} : \mathbb{N}_i \times \mathbb{N}_j \rightarrow \mathbb{N}_j$ for $i > j$, by

$$\text{add}(\mathbf{0}, y) = \Pi_1^1(y) = y \quad \text{add}(\mathbf{S}(x), y) = (\mathbf{f}_S \circ \Pi_2^3)(x, \text{add}(x, y), y) = \mathbf{S}(\text{add}(x, y)) ,$$

using general simultaneous ramified recursion, i.e. $\{\text{add}\} = \text{simrec}(\{\{\Pi_1^1, \mathbf{f}_S \circ \Pi_2^3\}\})$.

$$\boxed{
\begin{array}{c}
\frac{\mathbf{f} \in \mathcal{F} \quad t_i \downarrow v_i \quad \mathbf{f}(v_1, \dots, v_k) \downarrow v}{\mathbf{f}(t_1, \dots, t_k) \downarrow v} \qquad \frac{\mathbf{c} \in \mathcal{C} \quad t_i \downarrow v_i}{\mathbf{c}(t_1, \dots, t_k) \downarrow \mathbf{c}(v_1, \dots, v_k)} \\
\\
\frac{\mathbf{f}(p_1, \dots, p_k) \rightarrow r \in \mathcal{R} \quad \forall i. p_i \sigma = v_i \quad r \sigma \downarrow v}{\mathbf{f}(v_1, \dots, v_k) \downarrow v}
\end{array}
}$$

■ **Figure 4** Operational Semantics for Program $(\mathcal{F}, \mathcal{C}, \mathcal{R})$.

2. Let $\mathbb{T} := \{\mathbf{N}_L, \mathbf{M}_L, \mathbf{N}, \mathbf{M}\}$, where $\text{ar}(\mathbf{N}_L) = \text{ar}(\mathbf{M}_L) = 0$, $\text{ar}(\mathbf{N}) = 1$ and $\text{ar}(\mathbf{M}) = 2$. Then we can define the functions $\text{rabbits} : \mathbb{N}_i \rightarrow \mathbb{T}_j$ for $i > j$ from Section 2 by composition from the following two functions, defined by simultaneous ramified recurrence.

$$\begin{array}{ll}
\mathbf{a}(0) = \mathbf{M}_L & \mathbf{a}(\mathbf{S}(n)) = (\mathbf{f}_M \circ (\Pi_2^3, \Pi_3^3))(n, \mathbf{a}(n), \mathbf{b}(n)) = \mathbf{M}(\mathbf{a}(n), \mathbf{b}(n)) \\
\mathbf{b}(0) = \mathbf{N}_L & \mathbf{b}(\mathbf{S}(n)) = (\mathbf{f}_N \circ \Pi_3^3)(n, \mathbf{a}(n), \mathbf{b}(n)) = \mathbf{N}(\mathbf{a}(n)) .
\end{array}$$

3. We can define a function $\#\text{leafs} : \mathbb{T} \rightarrow \mathbb{N}$ by simultaneous primitive recursion which counts the number of leafs in \mathbb{T} -trees as follows.

$$\begin{array}{ll}
\#\text{leafs}(\mathbf{N}_L) = \mathbf{S}(0) & \#\text{leafs}(\mathbf{M}_L) = \mathbf{S}(0) \\
\#\text{leafs}(\mathbf{N}(t)) = \#\text{leafs}(t) & \#\text{leafs}(\mathbf{M}(l, r)) = \text{add}(\#\text{leafs}(l), \#\text{leafs}(r)) .
\end{array}$$

However, this function *cannot* be ramified, since add in the last equation requires different tiers. Indeed, having a ramified recursive function $\#\text{leafs} : \mathbb{T}_i \rightarrow \mathbb{N}_1$ (for some $i > 1$) defined as above would allow us to ramify $\text{fib} = \#\text{leafs} \circ \text{rabbits}$ which on input n computes the n^{th} Fibonacci number, and is thus an exponential function.

Computational Model, Syntax and Semantics

We introduce a simple, *rewriting based*, notion of program for computing functions over term algebras. Let \mathcal{V} denote a set of *variables*. Terms over a signature \mathcal{F} that include variables from \mathcal{V} are denoted by $\mathcal{T}(\mathcal{F}, \mathcal{V})$. A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ is called *linear* if each variable occurs at most once in t . The set of *subterms* $\text{STs}(t)$ of a term t is defined by $\text{STs}(t) := \{t\}$ if $t \in \mathcal{V}$ and $\text{STs}(t) := \bigcup_{1 \leq i \leq \text{ar}(f)} \text{STs}(t_i) \cup \{t\}$ if $t = f(t_1, \dots, t_{\text{ar}(f)})$. A *substitution*, is a finite mapping σ from variables to terms. By $t\sigma$ we denote the term obtained by replacing in t all variables $x \in \text{dom}(\sigma)$ by $\sigma(x)$.

► **Definition 5.** A *program* \mathbf{P} is given as a triple $(\mathcal{F}, \mathcal{C}, \mathcal{R})$ consisting of two disjoint signatures \mathcal{F} and \mathcal{C} of *operation symbols* f_1, \dots, f_m and *constructors* $\mathbf{c}_1, \dots, \mathbf{c}_n$ respectively, and a *finite* set \mathcal{R} of *rules* $l \rightarrow r$ over terms $l, r \in \mathcal{T}(\mathcal{F} \cup \mathcal{C}, \mathcal{V})$. For each rule, the *left-hand side* l is of the form $f_i(p_1, \dots, p_k)$ for some i , where the *patterns* p_j consist only of variables and constructors, and all variables occurring in the *right-hand side* r also occur in the left-hand side l .

We keep the program $\mathbf{P} = (\mathcal{F}, \mathcal{C}, \mathcal{R})$ fixed throughout the following. Moreover, we require that \mathbf{P} is *orthogonal*, that is, the following two requirements are met:

1. *left-linearity*: the left-hand sides l of each rule $l \rightarrow r \in \mathcal{R}$ is *linear*; and
2. *non-ambiguity*: there are no two rules with overlapping left-hand sides in \mathcal{R} .

Orthogonal programs define a class of deterministic first-order functional programs, see e.g. [6]. The domain of the defined functions is the constructor algebra $\mathcal{T}(\mathcal{C})$. Correspondingly, elements of $\mathcal{T}(\mathcal{C})$ are called *values*, which we denote by v, u, \dots .

In Figure 4 we present the operational semantics, realizing standard *call-by-value* evaluation order. The statement $t \downarrow v$ means that the term t *reduces* to the value v . We say that P computes the function $\mathbf{f} : \mathcal{T}(\mathcal{C})^k \rightarrow \mathcal{T}(\mathcal{C})$ if there exists an operation $f \in \mathcal{F}$ such that $\mathbf{f}(v_1, \dots, v_k) = v$ if and only if $f(v_1, \dots, v_k) \downarrow v$ holds for all inputs $v_i \in \mathcal{T}(\mathcal{C})$.

► **Example 6** (Continued from Example 4). The definition of `rabbits` from Section 2 can be turned into a program P_R over constructors of \mathbb{N} and \mathbb{T} , by *orienting* the underlying equations from left to right and replacing applications of functions $\mathbf{f} \in \{\mathbf{rabbits}, \mathbf{a}, \mathbf{b}\}$ with corresponding operation symbols $f \in \{\mathbf{rabbits}, \mathbf{a}, \mathbf{b}\}$. For instance, concerning the function `a`, the defining equations are turned into $\mathbf{a}(\mathbf{0}) \rightarrow \mathbf{M}_L$ and $\mathbf{a}(\mathbf{S}(n)) \rightarrow \mathbf{M}(\mathbf{a}(n), \mathbf{b}(n))$.

The example hints at a systematic construction of programs P_f computing functions $\mathbf{f} \in \text{SIMREC}(\mathbb{A})$, which can be made precise [4].

Term Graphs

We borrow key concepts from *term graph rewriting* (see e.g. the survey of Plump [25] for an overview) and follow the presentation of Barendregt et al. [8]. A *term graph* T over a signature \mathcal{F} is a *directed acyclic graph* whose nodes are labeled by symbols in $\mathcal{F} \cup \mathcal{V}$, and where outgoing edges are ordered. Formally, T is a triple $(N, \text{succ}, \text{lab})$ consisting of *nodes* N , a *successors function* $\text{succ} : N \rightarrow N^*$ and a *labeling function* $\text{lab} : N \rightarrow \mathcal{F} \cup \mathcal{V}$. We require that term graphs are *compatible* with \mathcal{F} , in the sense that for each node $o \in N$, if $\text{lab}_T(o) = f \in \mathcal{F}$ then $\text{succ}_T(o) = [o_1, \dots, o_{\text{ar}(f)}]$ and otherwise, if $\text{lab}_T(o) = x \in \mathcal{V}$, $\text{succ}_T(o) = []$. In the former case, we also write $T(o) = f(o_1, \dots, o_{\text{ar}(f)})$, the latter case is denoted by $T(o) = x$. We define the *successor relation* \rightarrow_T on nodes in T such that $o \rightarrow_T p$ holds iff p occurs in $\text{succ}(o)$, if p occurs at the i^{th} position we also write $o \xrightarrow{i}_T p$. Throughout the following, we consider only *acyclic* term graphs, that is, when \rightarrow_T is acyclic. Hence the *unfolding* $[o]_T$ of T at node o , defined by $[o]_T := x$ if $T(o) = x \in \mathcal{V}$, and otherwise $[o]_T := f([o_1]_T, \dots, [o_k]_T)$ where $T(o) = f(o_1, \dots, o_k)$, results in a finite term. We called the term graph T *rooted* if there exists a unique node o , the *root* of T , with $o \rightarrow_T^* p$ for every $p \in N$. We denote by $T|_o$ the *sub-graph* of T *rooted* at o . Consider a symbol $f \in \mathcal{F}$ and nodes $\{o_1, \dots, o_{\text{ar}(f)}\} \subseteq N$ of T . The extension S of T by a fresh node $o_f \notin N$ with $S(o_f) = f(o_1, \dots, o_{\text{ar}(f)})$ is denoted by $T \uplus \{o_f \mapsto f(o_1, \dots, o_{\text{ar}(f)})\}$. We write $f(T|_{o_1}, \dots, T|_{o_{\text{ar}(f)}})$ for the term graph $S|_{o_f}$.

For two rooted term graphs $T = (N_T, \text{succ}_T, \text{lab}_T)$ and $S = (N_S, \text{succ}_S, \text{lab}_S)$, a mapping $m : N_T \rightarrow N_S$ is called *morphic* in $o \in N_T$ if (i) $\text{lab}_T(o) = \text{lab}_S(m(o))$ and (ii) $o \xrightarrow{i}_T p$ implies $m(o) \xrightarrow{i}_S m(p)$ for all appropriate i . A *homomorphism* from T to S is a mapping $m : N_T \rightarrow N_S$ that (i) maps the root of T to the root of S and that (ii) is morphic in all nodes $o \in N_T$ not labeled by a variable. We write $T \succcurlyeq_m S$ to indicate that m is, possibly an extension of, a homomorphism from T to S .

Every term t is trivially representable as a *canonical tree* $\Delta(t)$ unfolding to t , using a fresh node for each occurrence of a subterm in t . For t a linear term, to each variable x in t we can associate a *unique node* in $\Delta(t)$ labeled by x , which we denote by o_x . The following proposition relates matching on terms and homomorphisms on trees. It essentially relies on the imposed linearity condition.

► **Proposition 7** (Matching on Graphs). *Let t be a linear term, T be a term graph and let o be a node of T .*

1. *If $\Delta(t) \succcurlyeq_m T|_o$ then there exists a substitution σ such that $t\sigma = [o]_T$.*
2. *Vice versa, if $t\sigma = [o]_T$ holds for some substitution σ then there exists a homomorphism $\Delta(t) \succcurlyeq_m T|_o$.*

$$\boxed{
\begin{array}{l}
\frac{f \in \mathcal{F} \quad (C_{i-1}, t_i) \Downarrow_{n_i} (C_i, v_i) \quad (C_k, f(v_1, \dots, v_k)) \Downarrow_n (C_{k+1}, v) \quad m = n + \sum_{i=1}^k n_i}{(C_0, f(t_1, \dots, t_k)) \Downarrow_m (C_{k+1}, v)} \text{ (Split)} \\
\frac{\mathbf{c} \in \mathcal{C} \quad (C_{i-1}, t_i) \Downarrow_{n_i} (C_i, v_i) \quad m = \sum_{i=1}^k n_i}{(C_0, \mathbf{c}(t_1, \dots, t_k)) \Downarrow_m (C_k, \mathbf{c}(v_1, \dots, v_k))} \text{ (Con)} \quad \frac{(f(v_1, \dots, v_k), v) \in C}{(C, f(v_1, \dots, v_k)) \Downarrow_0 (C, v)} \text{ (Read)} \\
\frac{(f(v_1, \dots, v_k), v) \notin C \quad f(p_1, \dots, p_k) \rightarrow r \in \mathcal{R} \quad \forall i. p_i \sigma = v_i \quad (C, r\sigma) \Downarrow_m (D, v)}{(C, f(v_1, \dots, v_k)) \Downarrow_{m+1} (D \cup \{(f(v_1, \dots, v_k), v)\}, v)} \text{ (Update)}
\end{array}
}$$

■ **Figure 5** Cost Annotated Operational Semantics with Memoization for Program $(\mathcal{F}, \mathcal{C}, \mathcal{R})$.

Here, the substitution σ and homomorphism m satisfy $\sigma(x) = [m(o_x)]_T$ for all variables x in t .

4 Memoization and Sharing, Formally

To implement *memoization*, we make use of a *cache* C which stores results of intermediate functions calls. A *cache* C is modeled as a set of tuples $(f(v_1, \dots, v_{\text{ar}(f)}), v)$, where $f \in \mathcal{F}$ and $v_1, \dots, v_{\text{ar}(f)}$ as well as v are values.

Figure 5 collects the *memoizing operational semantics* with respect to the program $P = (\mathcal{F}, \mathcal{C}, \mathcal{R})$. Here, a statement $(C, t) \Downarrow_m (D, v)$ means that starting with a cache C , the term t reduces to the value v with updated cache D . The natural number m indicates the *cost* of this reduction. The definition is along the lines of the standard semantics (Figure 4), carrying the cache throughout the reduction of the given term. The last rule of Figure 4 is split into two rules (Read) and (Update). The former performs a read from the cache, the latter the reduction in case the corresponding function call is not tabulated, updating the cache with the computed result. Notice that in the semantics, a read is attributed zero cost, whereas an update is accounted with a cost of one. Consequently the cost m in $(C, t) \Downarrow_m (D, v)$ refers to the number of non-tabulated function applications.

► **Lemma 8.** *We have $(\emptyset, t) \Downarrow_m (C, v)$ for some $m \in \mathbb{N}$ and cache C if and only if $t \Downarrow v$.*

The lemma confirms that the call-by-value semantics of Section 3 is correctly implemented by the memoizing semantics. To tame the growth rate of values, we define *small-step semantics* corresponding to the memoizing semantics, facilitating sharing of common sub-expressions.

Small-Step Semantics with Memoization and Sharing

To incorporate sharing, we extend the pairs (C, t) by a *heap*, and allow *references* to the heap both in terms and in caches. Let Loc denote a countably infinite set of *locations*. We overload the notion of *value*, and define *expressions* e and (*evaluation*) *contexts* E according to the following grammar:

$$\begin{aligned}
v &:= \ell \mid \mathbf{c}(v_1, \dots, v_k); \\
e &:= \ell \mid \langle f(\ell_1, \dots, \ell_k), e \rangle \mid f(e_1, \dots, e_k) \mid \mathbf{c}(e_1, \dots, e_k); \\
E &:= \square \mid \langle f(\ell_1, \dots, \ell_k), E \rangle \mid f(\ell_1, \dots, \ell_{i-1}, E, e_{i+1}, \dots, e_k) \mid \mathbf{c}(\ell_1, \dots, \ell_{i-1}, E, e_{i+1}, \dots, e_k).
\end{aligned}$$

Here, $\ell_1, \dots, \ell_k, \ell \in \text{Loc}$, $f \in \mathcal{F}$ and $\mathbf{c} \in \mathcal{C}$ are k -ary symbols. An expression is a term including references to values that will be stored on the heap. The additional construct $\langle f(\ell_1, \dots, \ell_k), e \rangle$ indicates that the partially evaluated expression e descends from a call

$$\begin{array}{c}
\frac{(f(\ell_1, \dots, \ell_k), \ell) \notin D \quad f(p_1, \dots, p_k) \rightarrow r \in \mathcal{R} \quad T := \Delta(f(p_1, \dots, p_k))}{T \geq_m f(H \setminus \ell_1, \dots, H \setminus \ell_k) \quad \sigma_m := \{x \mapsto m(\ell_x) \mid \ell_x \in \text{Loc}, T(\ell_x) = x \in \mathcal{V}\}} \text{(apply)} \\
\frac{}{(D, H, E[f(\ell_1, \dots, \ell_k)]) \rightarrow_{\mathbf{R}} (D, H, E[\langle f(\ell_1, \dots, \ell_k), r \sigma_m \rangle])} \\
\frac{(f(\ell_1, \dots, \ell_k), \ell) \in D}{(D, H, E[f(\ell_1, \dots, \ell_k)]) \rightarrow_{\mathbf{r}} (D, H, E[\ell])} \text{(read)} \\
\frac{}{(D, H, E[\langle f(\ell_1, \dots, \ell_k), \ell \rangle]) \rightarrow_{\mathbf{s}} (D \cup \{(f(\ell_1, \dots, \ell_k), \ell)\}, H, E[\ell])} \text{(store)} \\
\frac{(H', \ell) = \text{merge}(H, \mathbf{c}(\ell_1, \dots, \ell_k))}{(D, H, E[\mathbf{c}(\ell_1, \dots, \ell_k)]) \rightarrow_{\mathbf{m}} (D, H', E[\ell])} \text{(merge)}
\end{array}$$

■ **Figure 6** Small Step Semantics with Memoization and Sharing for Program $(\mathcal{F}, \mathcal{C}, \mathcal{R})$.

$f(v_1, \dots, v_k)$, with arguments v_i stored at location ℓ_i on the heap. A context E is an expression with a unique *hole*, denoted as \square , where all sub-expression to the left of the hole are references pointing to values. This syntactic restriction is used to implement a *left-to-right, call-by-value* evaluation order. We denote by $E[e]$ the expression obtained by replacing the hole in E by e .

A *configuration* is a triple (D, H, e) consisting of a *cache* D , *heap* H and expression e . Unlike before, the cache D consists of pairs of the form $(f(\ell_1, \dots, \ell_k), \ell)$ where instead of values, we store references $\ell_1, \dots, \ell_k, \ell$ pointing to the heap. The heap H is represented as a (multi-rooted) term graph H with nodes in Loc and constructors \mathcal{C} as labels. If ℓ is a node of H , then we say that H stores at location ℓ the value $[\ell]_H$ obtained by unfolding H starting from location ℓ . We keep the heap in a *maximally shared* form, that is, $H(\ell_a) = \mathbf{c}(\ell_1, \dots, \ell_k) = H(\ell_b)$ implies $\ell_a = \ell_b$ for two locations ℓ_a, ℓ_b of H . Thus crucially, values are stored once only, by the following lemma.

► **Lemma 9.** *Let H be a maximally shared heap with locations ℓ_1, ℓ_2 . If $[\ell_1]_H = [\ell_2]_H$ then $\ell_1 = \ell_2$.*

The operation $\text{merge}(H, \mathbf{c}(\ell_1, \dots, \ell_k))$, defined as follows, is used to extend the heap H with a constructor \mathbf{c} whose arguments point to ℓ_1, \dots, ℓ_k , retaining maximal sharing. Let ℓ_f be the first location not occurring in the nodes N of H (with respect to an arbitrary, but fixed enumeration on Loc). For $\ell_1, \dots, \ell_k \in N$ we define

$$\text{merge}(H, \mathbf{c}(\ell_1, \dots, \ell_k)) := \begin{cases} (H, \ell) & \text{if } H(\ell) = \mathbf{c}(\ell_1, \dots, \ell_k), \\ (H \cup \{\ell_f \mapsto \mathbf{c}(\ell_1, \dots, \ell_k)\}, \ell_f) & \text{otherwise.} \end{cases}$$

Observe that the first clause is unambiguous on maximally shared heaps.

Figure 6 collects the small step semantics with respect to a program $P = (\mathcal{F}, \mathcal{C}, \mathcal{R})$. We use $\rightarrow_{\mathbf{rsm}}$ to abbreviate the relation $\rightarrow_{\mathbf{r}} \cup \rightarrow_{\mathbf{s}} \cup \rightarrow_{\mathbf{m}}$ and likewise we abbreviate $\rightarrow_{\mathbf{R}} \cup \rightarrow_{\mathbf{rsm}}$ by $\rightarrow_{\mathbf{Rrsm}}$. Furthermore, we define $\rightarrow_{\mathbf{R}/\mathbf{rsm}} := \rightarrow_{\mathbf{rsm}}^* \cdot \rightarrow_{\mathbf{R}} \cdot \rightarrow_{\mathbf{rsm}}^*$. Hence the *m-fold composition* $\rightarrow_{\mathbf{R}/\mathbf{rsm}}^m$ corresponds to a $\rightarrow_{\mathbf{Rrsm}}$ -reduction with precisely m applications of $\rightarrow_{\mathbf{R}}$.

It is now time to show that the model of computation we have just introduced fits our needs, namely that it faithfully simulates big-step semantics as in Figure 5 (itself a correct implementation of call-by-value evaluation from Section 3). This is proved by first showing how big-step semantics can be *simulated* by small-step semantics, later proving that the latter is in fact *deterministic*.

In the following, we denote by $[e]_H$ the term obtained from e by following pointers to the

heap, ignoring the annotations $\langle f(\ell_1, \dots, \ell_k), \cdot \rangle$. Formally, we define

$$[e]_H := \begin{cases} f([e_1]_H, \dots, [e_k]_H) & \text{if } e = f(e_1, \dots, e_k), \\ [e']_H & \text{if } e = \langle f(\ell_1, \dots, \ell_k), e' \rangle. \end{cases}$$

Observe that this definition is well-defined as long as H contains all locations occurring in e (a property that is preserved by $\rightarrow_{\text{Rrsm}}$ -reductions). An *initial configuration* is a configuration of the form (\emptyset, H, e) with H a maximally shared heap and $e = f(v_1, \dots, v_k)$ an expression unfolding to a function call. Notice that the arguments v_1, \dots, v_k are allowed to contain references to the heap H .

► **Lemma 10 (Simulation).** *Let (\emptyset, H, e) be an initial configuration. If $(\emptyset, [e]_H) \Downarrow_m (C, v)$ holds for $m \geq 1$ then $(\emptyset, H, e) \rightarrow_{\text{R/rsm}}^m (D, G, \ell)$ for a location ℓ in G with $[\ell]_G = v$.*

The next lemma shows that the established simulation is *unique*, that is, there is exactly one derivation $(\emptyset, H, e) \rightarrow_{\text{R/rsm}}^m (D, G, \ell)$. Here, a relation \rightarrow is called *deterministic on a set* A if $b_1 \leftarrow a \rightarrow b_2$ implies $b_1 = b_2$ for all $a \in A$.

► **Lemma 11 (Determinism).** *The relation $\rightarrow_{\text{Rrsm}}$ is deterministic on all configurations reachable from initial configurations.*

► **Theorem 12.** *Suppose $(\emptyset, f(v_1, \dots, v_k)) \Downarrow_m (C, v)$ holds for a reducible term $f(v_1, \dots, v_k)$. Then for each initial configuration (\emptyset, H, e) with $[e]_H = f(v_1, \dots, v_k)$, there exists a unique sequence $(\emptyset, H, e) \rightarrow_{\text{R/rsm}}^m (D, G, \ell)$ for a location ℓ in G with $[\ell]_G = v$.*

Proof. As $f(v_1, \dots, v_k)$ is reducible, it follows that $m \geq 1$. Hence the theorem follows from Lemma 10 and Lemma 11. ◀

Invariance

Theorem 12 tells us that a term-based semantics (in which sharing is *not* exploited) can be simulated step-by-step by another, more sophisticated, graph-based semantics. The latter's advantage is that each computation step does not require copying, and thus does not increase the size of the underlying configuration too much. This is the key observation towards *invariance*: the number of reduction step is a sensible cost model from a complexity-theoretic perspective. Precisely this will be proved in the remaining of the section.

Define the *size* $|e|$ of an expression recursively by $|\ell| := 1$, $|f(e_1, \dots, e_k)| := 1 + \sum_{i=1}^k |e_i|$ and $|\langle f(\ell_1, \dots, \ell_k), e \rangle| := 1 + |e|$. In correspondence we define the *weight* $\text{wt}(e)$ by ignoring locations, i.e. $\text{wt}(\ell) := 0$. Recall that a reduction $(D_1, H_1, e_1) \rightarrow_{\text{R/rsm}}^m (D_2, H_2, e_2)$ consists of m applications of \rightarrow_{R} , all possibly interleaved by \rightarrow_{rsm} -reductions. As a first step, we thus estimate the overall length of the reduction $(D_1, H_1, e_1) \rightarrow_{\text{R/rsm}}^m (D_2, H_2, e_2)$ in m and the size of e_1 . Set $\Delta := \max\{|r| \mid l \rightarrow r \in \mathcal{R}\}$. The following serves as an intermediate lemma.

► **Lemma 13.** *The following properties hold:*

1. *If $(D_1, H_1, e_1) \rightarrow_{\text{rsm}} (D_2, H_2, e_2)$ then $\text{wt}(e_2) < \text{wt}(e_1)$.*
2. *If $(D_1, H_1, e_1) \rightarrow_{\text{R}} (D_2, H_2, e_2)$ then $\text{wt}(e_2) \leq \text{wt}(e_1) + \Delta$.*

Then essentially an application of the *weight gap principle* [18], a form of *amortized cost analysis*, binds the overall length of an $\rightarrow_{\text{R/rsm}}^m$ -reduction suitably.

► **Lemma 14.** *If $(D_1, H_1, e_1) \rightarrow_{\text{R/rsm}}^m (D_2, H_2, e_2)$ then $(D_1, H_1, e_1) \rightarrow_{\text{Rrsm}}^n (D_2, H_2, e_2)$ for $n \leq (1 + \Delta) \cdot m + \text{wt}(e)$ and $\Delta \in \mathbb{N}$ a constant depending only on \mathcal{P} .*

Define the size of a configuration $|(D, H, e)|$ as the sum of the sizes of its components. Here, the size $|D|$ of a cache D is defined as its cardinality, similar, the size $|H|$ of a heap is defined as the cardinality of its set of nodes. Notice that a configuration (D, H, e) can be straightforwardly encoded within logarithmic space-overhead as a string $\lceil (D, H, e) \rceil$, i.e. the length of the string $\lceil (D, H, e) \rceil$ is bounded by a function in $O(\log(n) \cdot n)$ in $|(D, H, e)|$, using constants to encode symbols and an encoding of locations logarithmic in $|H|$. Crucially, a step in the small-step semantics increases the size of a configuration only by a constant.

► **Lemma 15.** *If $(D_1, H_1, e_1) \rightarrow_{\text{Rrsm}} (D_2, H_2, e_2)$ then $|(D_2, H_2, e_2)| \leq |(D_1, H_1, e_1)| + \Delta$.*

► **Theorem 16.** *There exists a polynomial $p : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ such that for every initial configuration (\emptyset, H_1, e_1) , a configuration (D_2, H_2, e_2) with $(\emptyset, H_1, e_1) \rightarrow_{\text{R/rsm}}^m (D_2, H_2, e_2)$ is computable from (\emptyset, H_1, e_1) in time $p(|H_1| + |e_1|, m)$.*

Proof. It is tedious, but not difficult to show that the function which implements a step $c \rightarrow_{\text{Rrsm}} d$, i.e. which maps $\lceil c \rceil$ to $\lceil d \rceil$, is computable in polynomial time in $\lceil c \rceil$, and thus in the size $|c|$ of the configuration c . Iterating this function at most $n := (1 + \Delta) \cdot m + |(\emptyset, H_1, e_1)|$ times on input $\lceil (\emptyset, H_1, e_1) \rceil$, yields the desired result $\lceil (D_2, H_2, e_2) \rceil$ by Lemma 14. Since each iteration increases the size of a configuration by at most the constant Δ (Lemma 15), in particular the size of each intermediate configuration is bounded by a linear function in $|(\emptyset, H_1, e_1)| = |H_1| + |e_1|$ and n , the theorem follows. ◀

Combining Theorem 12 and Theorem 16 we thus obtain the following.

► **Theorem 17.** *There exists a polynomial $p : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ such that for $(\emptyset, f(v_1, \dots, v_k)) \Downarrow_m (C, v)$, the value v represented as DAG is computable from v_1, \dots, v_k in time $p(\sum_{i=1}^k |v_i|, m)$.*

Theorem 17 thus confirms that the cost m of a reduction $(\emptyset, f(v_1, \dots, v_k)) \Downarrow_m (C, v)$ is a suitable cost measure. In other words, the *memoized runtime complexity* of a function f , relating input size $n \in \mathbb{N}$ to the maximal cost m of evaluating f on arguments v_1, \dots, v_k of size up to n , i.e. $(\emptyset, f(v_1, \dots, v_k)) \Downarrow_m (C, v)$ with $\sum_{i=1}^k |v_i| \leq n$, is an *invariant cost model*.

► **Example 18** (Continued from Example 6). Reconsider the program P_R and the evaluation of a call $\text{rabbits}(\mathbf{S}^n(\mathbf{0}))$ which results in the genealogical tree v_n of height $n \in \mathbb{N}$ associated with *Fibonacci's rabbit problem*. Then one can show that $\text{rabbits}(\mathbf{S}^n(\mathbf{0})) \Downarrow_m v_n$ with $m \leq 2 \cdot n + 1$. Crucially here, the two intermediate functions \mathbf{a} and \mathbf{b} defined by simultaneous recursion are called only on proper subterms of the input $\mathbf{S}^n(\mathbf{0})$, hence in particular the rules defining \mathbf{a} and \mathbf{b} respectively, are unfolded at most n times. As a consequence of the bound on m and Theorem 17 we obtain that the function `rabbits` from the introduction is polytime computable.

► **Remark.** Strictly speaking, our DAG representation of a value v , viz the part of the final heap reachable from a corresponding location ℓ , is not an encoding in the classical, complexity theoretic setting. Different computations resulting in the same value v can produce different DAG representations of v , however, these representations differ only in the naming of locations. Even though our encoding can be exponentially compact in comparison to a linear representation without sharing, it is not exponentially more *succinct* than a reasonable encoding for graphs (e.g. representations as circuits, see Papadimitriou [24]). In such succinct encodings not even equality can be decided in polynomial time. Our form of representation does clearly not fall into this category. In particular, in our setting it can be easily checked in polynomial time that two DAGs represent the same value.

5 GRSR is Sound for Polynomial Time

Sometimes (e.g., in [11]), the first step towards a proof of soundness for ramified recursive systems consists in giving a proper bound precisely relating the size of the result and the size of the inputs. More specifically, if the result has tier n , then the size of it depends polynomially on the size of the inputs of tier higher than n , but only *linearly*, and in very restricted way, on the size of inputs of tier n . Here, a similar result holds, but size is replaced by *minimal shared size*.

The *minimal shared size* $\|v_1, \dots, v_k\|$ for a *sequence* of elements $v_1, \dots, v_k \in \mathbb{A}$ is defined as the number of subterms in v_1, \dots, v_k , i.e. the cardinality of the set $\bigcup_{1 \leq i \leq k} \text{STs}(v_i)$. Then $\|v_1, \dots, v_k\|$ corresponds to the number of locations necessary to store the values v_1, \dots, v_k on a heap (compare Lemma 9). If \mathbf{A} is the expression $\mathbb{A}_{n_1} \times \dots \times \mathbb{A}_{n_m}$, n is a natural number, and \vec{t} is a sequence of m terms, then $\|\vec{t}\|_{\mathbf{A}}^{>n}$ is defined to be $\|t_{i_1}, \dots, t_{i_k}\|$ where i_1, \dots, i_k are precisely those indices such that $n_{i_1}, \dots, n_{i_k} > n$. Similarly for $\|\vec{t}\|_{\mathbf{A}}^{\leq n}$.

► **Proposition 19 (Max-Poly).** *If $\mathbf{f} \triangleright \mathbf{A} \rightarrow \mathbb{A}_n$, then there is a polynomial $p_{\mathbf{f}} : \mathbb{N} \rightarrow \mathbb{N}$ such that $\|\mathbf{f}(\vec{v})\| \leq \|\vec{v}\|_{\mathbf{A}}^{\leq n} + p_{\mathbf{f}}(\|\vec{v}\|_{\mathbf{A}}^{>n})$.*

Once we know that ramified recursive definitions are not too fast-growing for the minimal shared size, we know that all terms around do not have a too-big minimal shared size. As a consequence:

► **Proposition 20.** *If $\mathbf{f} \triangleright \mathbf{A} \rightarrow \mathbb{A}_n$, then there is a polynomial $p_{\mathbf{f}} : \mathbb{N} \rightarrow \mathbb{N}$ such that for every $v, (\emptyset, \mathbf{f}(\vec{v})) \Downarrow_m (C, v)$, with $m \leq p_{\mathbf{f}}(\|\vec{v}\|)$.*

The following, then, is just a corollary of Proposition 20 and Invariance (Theorem 17).

► **Theorem 21.** *Let $\mathbf{f} : \mathbb{A}_{p_1} \times \dots \times \mathbb{A}_{p_k} \rightarrow \mathbb{A}_m$ be a function defined by general ramified simultaneous recursion. There exists then a polynomial $p_{\mathbf{f}} : \mathbb{N}^k \rightarrow \mathbb{N}$ such that for all inputs v_1, \dots, v_k , a DAG representation of $\mathbf{f}(v_1, \dots, v_k)$ is computable in time $p_{\mathbf{f}}(|v_1|, \dots, |v_k|)$.*

► **Example 22 (Continued from Example 18).** In Example 4 we indicated that the function `rabbits` : $\mathbb{N} \rightarrow \mathbb{T}$ from Section 2 is definable by GRSR. As a consequence of Theorem 21, it is computable in polynomial time, e.g. on a Turing machine. Similar, we can prove the function `tree` from Section 2 polytime computable.

6 Conclusion

In this work we have shown that simultaneous ramified recurrence on generic algebras is sound for polynomial time, resolving a long-lasting open problem in implicit computational complexity theory. We believe that with this work we have reached the *end of a quest*. Slight extensions, e.g. the inclusion of *parameter substitution*, lead outside polynomial time as soon as simultaneous recursion over trees is permissible.

Towards our main result, we introduced the notion of memoized runtime complexity, and we have shown that this cost model is invariant under polynomial time. Crucially, we use a compact DAG representation of values to control duplication, and tabulation to avoid expensive re-computations. To the authors best knowledge, our work is the first where sharing and memoization are reconciled, in the context of implicit computational complexity theory. Both techniques have been extensively employed, however separately. Essentially relying on sharing, the invariance of the unitary cost model in various rewriting based models of computation, e.g. the λ -calculus [1, 15, 2] and term rewrite systems [14, 5] could be proved.

Several works (e.g. [22, 13, 7]) rely on memoization, employing a measure close to our notion of memoized runtime complexity. None of these works integrate sharing, instead, inputs are either restricted to strings or dedicated bounds on the size of intermediate values have to be imposed. We are confident that our second result is readily applicable to resolve such restrictions.

References

- 1 B. Accattoli and U. Dal Lago. On the Invariance of the Unitary Cost Model for Head Reduction. In *Proc. of 23rd RTA*, volume 15 of *LIPICs*, pages 22–37. Dagstuhl, 2012.
- 2 B. Accattoli and U. Dal Lago. Beta Reduction is Invariant, Indeed. In *Joint Proc. of 23rd CSL and 29th LICS*, page 8. ACM, 2014.
- 3 T. Arai and N. Eguchi. A New Function Algebra of EXPTIME Functions by Safe Nested Recursion. *TOCL*, 10(4), 2009.
- 4 M. Avanzini and U. Dal Lago. On Sharing, Memoization, and Polynomial Time. Technical report, University of Bologna, 2014. Available at <http://arxiv.org/abs/1501.00894>.
- 5 M. Avanzini and G. Moser. Closing the Gap Between Runtime Complexity and Polytime Computability. In *Proc. of 21st RTA*, volume 6 of *LIPICs*, pages 33–48. Dagstuhl, 2010.
- 6 F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
- 7 P. Baillot, U. Dal Lago, and J.-Y. Moyén. On Quasi-interpretations, Blind Abstractions and Implicit Complexity. *MSCS*, 22(4):549–580, 2012.
- 8 H. P. Barendregt, M. v. Eekelen, J. R. W. Glauert, J. R. Kennaway, M. J. Plasmeijer, and M. R. Sleep. Term Graph Rewriting. In *PARLE (2)*, volume 259 of *LNCS*, pages 141–158. Springer, 1987.
- 9 S. Bellantoni. *Predicative Recursion and Computational Complexity*. PhD thesis, University of Toronto, 1992.
- 10 S. Bellantoni. Predicative Recursion and the Polytime Hierarchy. In *Feasible Mathematics II*. Birkhäuser Boston, 1994.
- 11 S. Bellantoni and S. Cook. A new Recursion-Theoretic Characterization of the Polytime Functions. *CC*, 2(2):97–110, 1992.
- 12 G. Bonfante, R. Kahle, J.-Y. Marion, and I. Oitavem. Recursion Schemata for NCK. In *Proc. of 22nd CSL*, volume 5213 of *LNCS*, pages 49–63. Springer, 2008.
- 13 G. Bonfante, J.-Y. Marion, and J.-Y. Moyén. Quasi-interpretations: A Way to Control Resources. *Theoretical Computer Science*, 412(25):2776–2796, 2011.
- 14 U. Dal Lago and S. Martini. Derivational Complexity is an Invariant Cost Model. In *Revised Selected Papers of 1st FOPARA*, volume 6324 of *LNCS*, pages 100–113. Springer, 2009.
- 15 U. Dal Lago and S. Martini. On Constructor Rewrite Systems and the Lambda Calculus. *LMCS*, 8(3):1–27, 2012.
- 16 U. Dal Lago, S. Martini, and M. Zorzi. General Ramified Recurrence is Sound for Polynomial Time. In *Proc. of 1st DICE*, volume 23 of *EPTCS*, pages 47–62, 2010.
- 17 N. Danner and J. S. Royer. Ramified Structural Recursion and Corecursion. *CoRR*, abs/1201.4567, 2012.
- 18 N. Hirokawa and G. Moser. Automated Complexity Analysis Based on the Dependency Pair Method. In *Proc. of 4th IJCAR*, volume 5195 of *LNAI*, pages 364–380. Springer, 2008.
- 19 B. Hoffmann. Term Rewriting with Sharing and Memoization. In *Proc. of 3rd ALP*, volume 632 of *LNCS*, pages 128–142. Springer, 1992.
- 20 D. Leivant. Stratified Functional Programs and Computational Complexity. In *Proc. of 20th POPL*, pages 325–333. ACM, 1993.

- 21 D. Leivant. Ramified Recurrence and Computational Complexity I: Word Recurrence and Poly-time. In *Feasible Mathematics II*, volume 13, pages 320–343. Birkhäuser Boston, 1995.
- 22 J.-Y. Marion. Analysing the Implicit Complexity of Programs. *IC*, 183:2–18, 2003.
- 23 I. Oitavem. Implicit Characterizations of Pspace. In *PTCS*, pages 170–190, 2001.
- 24 C. H. Papadimitriou. *Computational Complexity*. Addison Wesley Longman, second edition, 1995.
- 25 D. Plump. Essentials of Term Graph Rewriting. *ENTCS*, 51:277–289, 2001.

Proof Complexity of Resolution-based QBF Calculi

Olaf Beyersdorff¹, Leroy Chew¹, and Mikoláš Janota²

¹ School of Computing, University of Leeds, UK

² INESC-ID, Lisbon, Portugal

Abstract

Proof systems for quantified Boolean formulas (QBFs) provide a theoretical underpinning for the performance of important QBF solvers. However, the proof complexity of these proof systems is currently not well understood and in particular lower bound techniques are missing. In this paper we exhibit a new and elegant proof technique for showing lower bounds in QBF proof systems based on strategy extraction. This technique provides a direct transfer of circuit lower bounds to lengths of proofs lower bounds. We use our method to show the hardness of a natural class of parity formulas for Q-resolution and universal Q-resolution. Variants of the formulas are hard for even stronger systems as long-distance Q-resolution and extensions. With a completely different lower bound argument we show the hardness of the prominent formulas of Kleine Büning et al. [34] for the strong expansion-based calculus IR-calc. Our lower bounds imply new exponential separations between two different types of resolution-based QBF calculi: proof systems for CDCL-based solvers (Q-resolution, long-distance Q-resolution) and proof systems for expansion-based solvers ($\forall\text{Exp}+\text{Res}$ and its generalizations IR-calc and IRM-calc). The relations between proof systems from the two different classes were not known before.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems: Complexity of proof procedures

Keywords and phrases proof complexity, QBF, lower bound techniques, separations

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.76

1 Introduction

Proof complexity studies the complexity of theorem proving in various formal systems, providing both sharp lower and upper bounds for the size of proofs of important combinatorial statements. One motivation for this research comes from its close connection to fundamental questions in computational complexity, and this connection has been present since the very beginnings of the field [20]. Another motivation is the tremendous success of SAT solvers, which today solve huge industrial instances of the NP-hard SAT problem with even millions of variables. Proof complexity provides the main theoretical tool for an understanding of the power and limitations of these algorithms. As most modern SAT solvers are based on resolution, this proof system has received a key attention; and many ingenious techniques have been devised to understand the complexity of resolution proofs (cf. [40, 17] for surveys).

During the last decade there has been a great interest and research activity to extend the success of SAT solvers to the more expressive *quantified Boolean formulas (QBF)*. Due to its PSPACE completeness (even for restricted versions [2]), QBF is far more expressive than SAT and thus applies to further fields such as formal verification or planning [38, 7, 21]. As for SAT solvers, runs of QBF solvers produce witnesses of unsatisfiability (proofs), and there has been a lot of interest in the correspondence between the formal systems and solvers.

In particular, Kleine Büning et al. [34] define a resolution-like calculus called *Q-resolution* (Q-Res). There are several extensions of Q-Res; notably *long-distance Q-resolution* (LD-Q-



© Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota;

licensed under Creative Commons License CC-BY

32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).

Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 76–89

Leibniz International Proceedings in Informatics



LIPIcs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



Res) [3], which is more powerful than the standard Q-Res [22]. Q-Res and its extensions are important as they model QBF solving based on CDCL [24]. While Q-Res can only resolve on existential variables, the proof system *QU-Res*, introduced by Van Gelder [43], also allows to resolve on universal variables. Combining universal and long-distance Q-resolution, Balabanov et al. [4] recently considered the system LQU^+ -Res. Apart from CDCL, another main approach to QBF-solving is through *expansion of quantifiers* [14, 6, 28]. Recently, a proof system $\forall\text{Exp}+\text{Res}$ was introduced with the motivation to trace expansion-based QBF solvers [29]. $\forall\text{Exp}+\text{Res}$ also uses resolution, but is conceptually very different from Q-Res.

In the recent work [8] two further proof systems *IR-calc* and *IRM-calc* are introduced, which unify the CDCL and expansion based approaches in the sense that *IR-calc* simulates both Q-Res and $\forall\text{Exp}+\text{Res}$. The system *IRM-calc* enhances *IR-calc* and additionally simulates long-distance Q-resolution. While *IR-calc* and *IRM-calc* are quite powerful, they still preserve the property of strategy extraction, which is important for verifying runs of QBF solvers.

In general, it is fair to say that the complexity and relations between QBF proof systems are not well understood. In particular, in sharp contrast to propositional proof complexity, we currently lack lower bound techniques for QBF proof systems.¹

Our contributions

In this paper we aim towards a significantly better understanding of proof complexity of QBF proof systems. Our main contributions are the following:

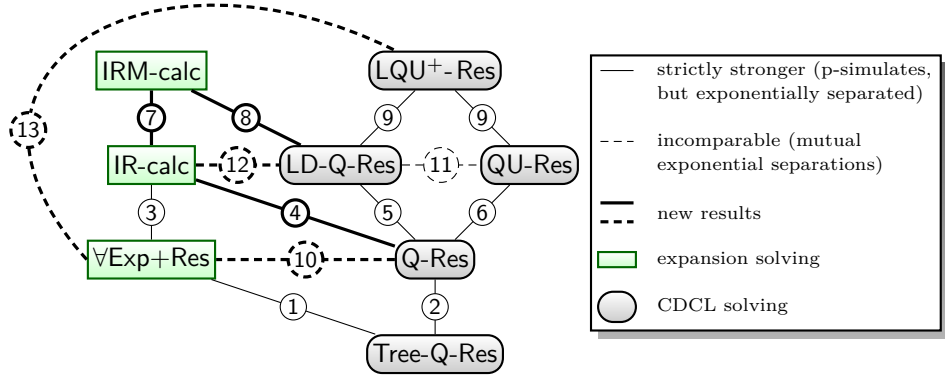
1. A new lower bound method based on strategy extraction. We exhibit a new method to obtain lower bounds to the proof size in QBF proof systems, which directly allows to transfer circuit lower bounds to size of proof lower bounds. This method is based on the property of *strategy extraction*, which is known to hold for many resolution-based QBF proof systems. A QBF proof system has strategy extraction if given a refutation of a false QBF φ it is possible to efficiently compute a winning strategy for the universal player for φ .

The basic idea of our method is both conceptually simple and elegant: If we know that a family φ_n of false QBFs requires large winning strategies, then proofs of φ_n must be large in all proof systems with feasible strategy extraction. Now we need suitable formulas φ_n . Starting with a language L – for which we know (or conjecture) circuit lower bounds – we construct a family of false QBFs φ_n such that every winning strategy of the universal player for φ_n will have to compute L for inputs of length n . Consequently, a circuit lower bound for L directly translates into a lower bound for the winning strategy and therefore the proof size.

This immediately implies conditional lower bounds. However, if carefully implemented, our method also yields *unconditional lower bounds*. For Q-Res (and QU-Res) it is known that strategy extraction is computationally easy [3]; it is in fact possible in AC^0 as we verify here. Using the hardness of parity for AC^0 we can therefore construct formulas $Q\text{PARITY}_n$ that require exponential-size proofs in Q-Res (and QU-Res).

Conceptually, our lower bound method via strategy extraction is similar to the feasible interpolation technique [35], which is one of the most successful techniques in classical proof complexity. In feasible interpolation, circuit lower bounds are also translated into proof size lower bounds. However, feasible interpolation only works for formulas of a special syntactic

¹ We note the very recent game technique for tree-like Q-Res [9], inspired by [10, 11, 12]. Further, [5] introduces a technique that lifts *known* hardness results for Q-Res to stronger systems by modifying the formula. We use that idea in Sec. 5.



■ **Figure 1** The simulation order of QBF resolution systems (references in the table below).

form, while our technique directly applies to arbitrary languages. It is a long-standing belief in the proof complexity community that there exists a direct connection between progress for showing lower bounds in circuit complexity and for proof systems (cf. [19]). For QBF proof systems our technique makes such a connection very explicit.

2. Lower bounds for QBF proof systems. Our new lower bound method directly gives a new lower bound for Q-Res for the parity formulas. In addition, we transfer this lower bound to the stronger systems LD-Q-Res and QU-Res by arguing that neither long-distance nor universal resolution gives any advantage on a suitable modification of the parity formulas.

For the strong system IR-calc from [8] we show that the strategy extraction method is not

	Simulation/Separation		Incomparable	
1	[31]	[31]	10	[30], Cor. 16
2	by Def.	[16]	11	[4]
3	[8]	[30], [8]	12	Cor. 8, Cor. 24
4	[8]	Cor. 16	13	Cor. 8, Cor. 24
5	by Def.	[22]		
6	by Def.	[43]		
7	by Def.	[8], Cor. 8		
8	[8]	Cor. 24		
9	by Def.	[4]		

directly applicable (at least for unconditional bounds in the way we use it here). However, we use a completely different lower bound argument to obtain an exponential lower bound for the well-known formulas KBKF(t) of Kleine Büning, Karpinski and Flögel [34] in IR-calc. In the same work [34], where Q-Res was introduced, these formulas were suggested as hard formulas for Q-Res. In fact, a number of further separations of QBF proof systems builds on this [22, 4]. Here we

show in a technically involved counting argument that the formulas are even hard for IR-calc. As IR-calc simulates Q-Res [8] we obtain as a by-product a formal proof of the hardness of KBKF(t) in Q-Res.

3. Separations between QBF proof systems. Our lower bounds imply a number of new separations and incomparability results. The two main new results are: (i) IR-calc does not simulate LD-Q-Res; (ii) LQU⁺-Res does not simulate ∇Exp+Res. Both are in fact exponential separations. Item (i) is obtained from the lower bound for KBKF(t), while (ii) follows from the lower bound on a variant of the parity formulas. In contrast to separations by KBKF(t), all separations derived from (ii) even hold for formulas of bounded quantifier complexity. Together with previous simulation results these imply many further separations.

Figure 1 depicts the simulation order of QBF resolution systems together with the separations. Combined with previous simulations and separations (cf. the table accompanying Fig. 1) this yields an almost complete understanding of the simulation order of QBF resolution systems.

2 Preliminaries

A *literal* is a Boolean variable or its negation. If l is a literal, $\neg l$ denotes the complementary literal, i.e. $\neg\neg x = x$. A *clause* is a disjunction of literals and a *term* is a conjunction of literals. The empty clause is denoted by \perp , which is semantically equivalent to false. A formula in *conjunctive normal form* (CNF) is a conjunction of clauses. For a literal $l = x$ or $l = \neg x$, we write $\text{var}(l)$ for x and extend this notation to $\text{var}(C)$ for a clause C .

$$\frac{}{C} \text{ (Axiom)} \quad \frac{D \cup \{u\}}{D} \text{ (\forall-Red)} \quad \frac{D \cup \{u^*\}}{D} \text{ (\forall-Red}^*)$$

C is a clause in the matrix. Literal u is universal and $\text{lv}(u) \geq \text{lv}(l)$ for all $l \in D$.

$$\frac{C_1 \cup U_1 \cup \{x\} \quad C_2 \cup U_2 \cup \{\neg x\}}{C_1 \cup C_2 \cup U} \text{ (Res)}$$

We consider four instantiations of the Res-rule:
S \exists R: x is existential.
 If $z \in C_1$, then $\neg z \notin C_2$. $U_1 = U_2 = U = \emptyset$.
S \forall R: x is universal. Otherwise same conditions as S \exists R.
L \exists R: x is existential.
 If $l_1 \in C_1, l_2 \in C_2$, $\text{var}(l_1) = \text{var}(l_2) = z$ then $l_1 = l_2 \neq z^*$. U_1, U_2 contain only universal literals with $\text{var}(U_1) = \text{var}(U_2)$. $\text{ind}(x) < \text{ind}(u)$ for each $u \in \text{var}(U_1)$.
 If $w_1 \in U_1, w_2 \in U_2$, $\text{var}(w_1) = \text{var}(w_2) = u$ then $w_1 = \neg w_2$, $w_1 = u^*$ or $w_2 = u^*$. $U = \{u^* \mid u \in \text{var}(U_1)\}$.
L \forall R: x is universal. Otherwise same conditions as L \exists R.

■ **Figure 2** The rules of CDCL-based proof systems.

Quantified Boolean Formulas (QBFs) [33] extend propositional logic with quantifiers with the standard semantics that $\forall x. \Psi$ is satisfied by the same truth assignments as $\Psi[0/x] \wedge \Psi[1/x]$ and $\exists x. \Psi$ as $\Psi[0/x] \vee \Psi[1/x]$. Unless specified otherwise, we assume that QBFs are in *closed prenex* form with a CNF *matrix*, i.e., we consider the form $Q_1 X_1 \dots Q_k X_k. \phi$, where X_i are pairwise disjoint (ordered) sets of variables; $Q_i \in \{\exists, \forall\}$ and $Q_i \neq Q_{i+1}$. The formula ϕ is in CNF and is defined only on variables $X_1 \cup \dots \cup X_k$. The propositional part ϕ is called the *matrix* and the rest the *prefix*. If $x \in X_i$, we say that x is at *level* i and write $\text{lv}(x) = i$; we write $\text{lv}(l)$ for $\text{lv}(\text{var}(l))$. In contrast to the level, the *index* $\text{ind}(x)$ provides the more detailed information on the actual position of x in the prefix, i.e. all variables are indexed by $1, \dots, n$ from left to right.

Often it is useful to think of a QBF $Q_1 X_1 \dots Q_k X_k. \phi$ as a *game* between the *universal* and the *existential player*. In the i -th step of the game, the player Q_i assigns values to all the variables X_i . The existential player wins the game iff the matrix ϕ evaluates to 1 under the assignment constructed in the game. The universal player wins iff the matrix ϕ evaluates to 0. Given a universal variable u with index i , a *strategy* for

$$\frac{\{l^{\{\tau\}} \mid l \in C, l \text{ exist.}\} \cup \{\tau(l) \mid l \in C, l \text{ univ.}\}}{} \text{ (Ax)}$$

C is a clause from the matrix and τ is an assignment to all universal variables.

$$\frac{C_1 \vee x^\tau \quad C_2 \vee \neg x^\tau}{C_1 \cup C_2} \text{ (Res)}$$

■ **Figure 3** The rules of $\forall\text{Exp}+\text{Res}$ [31].

u is a function from all variables of index $< i$ to $\{0, 1\}$. A QBF is false iff there exists a

winning strategy for the universal player, i.e. if the universal player has a strategy for all universal variables that wins any possible game [25][1, Sec. 4.2.2][37, Chap. 19].

A *proof system* [20] for a language L over Γ is a poly-time computable function $f : \Gamma^* \rightarrow \Gamma^*$ with $\text{rng}(f) = L$. If $f(x) = y$ then x is called an f -proof for y . For $L = \text{QBF}$ we speak of a *QBF proof system*. In our systems here, proofs are sequences of clauses; a *refutation* is a proof deriving \perp . A proof system S for L *simulates* a proof system P for L if there exists a polynomial p such that for all P -proofs π of x there is an S -proof π' of x with $|\pi'| \leq p(|\pi|)$.

Resolution-based calculi for QBF. We now give a brief overview of the main existing resolution-based calculi for QBF. We start by describing the proof systems modelling *CDCL-based QBF solving*; their rules are summarized in

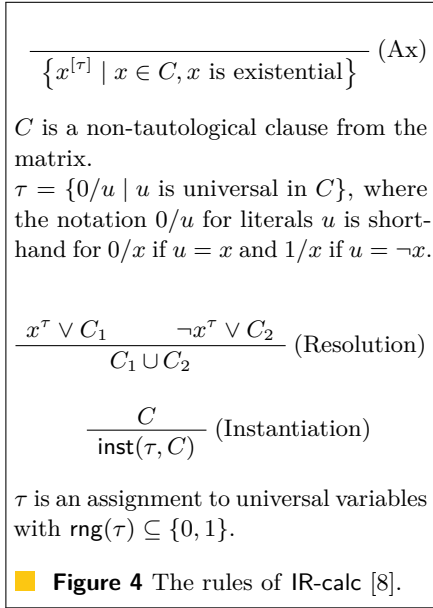


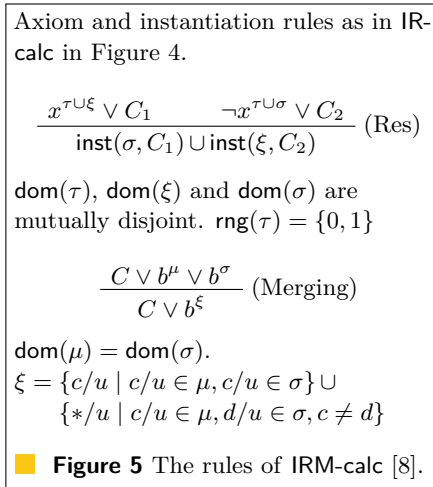
Figure 2. The most basic and important system is *Q-resolution (Q-Res)* by Kleine Büning et al. [34]. It is a resolution-like calculus that operates on QBFs in prenex form with CNF matrix. In addition to the axioms, Q-Res comprises the resolution rule $\text{S}\exists\text{R}$ and universal reduction $\forall\text{-Red}$ (cf. Fig. 2).

Long-distance resolution (LD-Q-Res) appears originally in the work of Zhang and Malik [44] and was formalized into a calculus by Balabanov and Jiang [3]. It merges complementary literals of a universal variable u into the special literal u^* . These special literals prohibit certain resolution steps. In particular, different literals of a universal variable u may be merged only if $\text{lv}(x) < \text{lv}(u)$, where x is the resolution variable. LD-Q-Res uses the rules $\text{L}\exists\text{R}$, $\forall\text{-Red}$ and $\forall\text{-Red}^*$.

QU-resolution (QU-Res) [43] removes the restriction from Q-Res that the resolved variable must be an existential variable and allows resolution of uni-

versal variables. The rules of QU-Res are $\text{S}\exists\text{R}$, $\text{S}\forall\text{R}$ and $\forall\text{-Red}$. *LQU⁺-Res* [4] extends LD-Q-Res by allowing short and long distance resolution pivots to be universal, however, the pivot is never a merged literal z^* . LQU⁺-Res uses the rules $\text{L}\exists\text{R}$, $\text{L}\forall\text{R}$, $\forall\text{-Red}$ and $\forall\text{-Red}^*$.

The second type of calculi models *expansion-based QBF solving*. These calculi are based on *instantiation* of universal variables: $\forall\text{Exp}+\text{Res}$ [31], IR-calc, and IRM-calc [8]. All these calculi operate on clauses that comprise only existential variables from the original QBF, which are additionally *annotated* by a substitution to some universal variables, e.g. $\neg x^{0/u_1 1/u_2}$. For any annotated literal l^σ , the substitution σ must not make assignments to variables at a higher quantification level than l , i.e. if $u \in \text{dom}(\sigma)$, then u is universal and $\text{lv}(u) < \text{lv}(l)$. To preserve this invariant, we use the *auxiliary notation* $l^{[\sigma]}$, which for an existential literal l and an assignment σ to the universal variables filters out all assignments that are not permitted, i.e. $l^{[\sigma]} = l^{\{c/u \in \sigma \mid \text{lv}(u) < \text{lv}(l)\}}$.



The simplest instantiation-based calculus we consider is the calculus $\forall\text{Exp}+\text{Res}$, whose rules are presented in Figure 3. The system IR-calc extends $\forall\text{Exp}+\text{Res}$ by enabling partial assignments in annotations. To do so, we utilize the auxiliary operations of *completion* and *instantiation*. For assignments τ and μ , we write $\tau \preceq \mu$ for the assignment σ defined as follows: $\sigma(x) = \tau(x)$ if $x \in \text{dom}(\tau)$, otherwise $\sigma(x) = \mu(x)$ if $x \in \text{dom}(\mu)$. The operation $\tau \preceq \mu$ is called *completion* because μ provides values for variables not defined in τ . The operation is associative and therefore we can omit parentheses. For an assignment τ and an annotated clause C the function $\text{inst}(\tau, C)$ returns the annotated clause $\{l^{[\sigma \preceq \tau]} \mid l^\sigma \in C\}$. The system IR-calc is defined in Figure 4.

The calculus IRM-calc further extends IR-calc by enabling annotations containing $*$. The rules of the calculus IRM-calc are presented in Figure 5. The symbol $*$ may be introduced by the merge rule, e.g. by collapsing $x^{0/u} \vee x^{1/u}$ into $x^{*/u}$. The calculus IR-calc p-simulates $\forall\text{Exp}+\text{Res}$ as well as Q-Res . The calculus IRM-calc p-simulates IR-calc as well as LD-Q-Res [8].

3 A lower bound in IR-calc for the formulas of Kleine Büning et al.

Our first main result is a proof complexity analysis of a well-known family of formulas $\text{KBKF}(t)$ first defined by Kleine Büning et al. [34]. Here we prove that the $\text{KBKF}(t)$ formulas are hard for IR-calc , which is stronger than Q-Res (Cor. 16, [8, Thm 6]). This provides the first non-trivial lower bound for IR-calc , and further even separates the system from LD-Q-Res .

► **Definition 1** (Kleine Büning, Karpinski and Flögel [34]). The formula $\text{KBKF}(t)$ has prefix $\exists y_0, y_{1,0}, y_{1,1} \forall x_1 \exists y_{2,0}, y_{2,1} \forall x_2 \dots \forall x_{t-1} \exists y_{t,0}, y_{t,1} \forall x_t \exists y_{t+1} \dots y_{t+t}$ and matrix clauses

$$\begin{array}{ll} C_- = \{\neg y_0\} & C_0 = \{y_0, \neg y_{1,0}, \neg y_{1,1}\} \\ C_i^0 = \{y_{i,0}, x_i, \neg y_{i+1,0}, \neg y_{i+1,1}\} & C_i^1 = \{y_{i,1}, \neg x_i, \neg y_{i+1,0}, \neg y_{i+1,1}\} \quad \text{for } i \in [t-1] \\ C_t^0 = \{y_{t,0}, x_t, \neg y_{t+1}, \dots, \neg y_{t+t}\} & C_t^1 = \{y_{t,1}, \neg x_t, \neg y_{t+1}, \dots, \neg y_{t+t}\} \\ C_{t+i}^0 = \{x_i, y_{t+i}\} & C_{t+i}^1 = \{\neg x_i, y_{t+i}\} \quad \text{for } i \in [t]. \end{array}$$

Let us verify that the $\text{KBKF}(t)$ formulas are indeed false QBFs and – at the same time – provide some intuition about them. The existential player starts by playing $y_0 = 0$ because of clause C_- . Clause C_0 forces the existential player to set one of $y_{1,0}, y_{1,1}$ to 0. Assume the existential chooses $y_{1,0} = 0$ and $y_{1,1} = 1$. If the universal player tries to win, he will counter with $x_1 = 0$, thus forcing the existential player again to set one of $y_{2,0}, y_{2,1}$ to 0. This continues for t rounds, leaving in each round a choice of $y_{i,0} = 0$ or $y_{i,1} = 0$ to the existential player, to which the universal counters by setting x_i accordingly. Finally, the existential player is forced to set one of y_{t+1}, \dots, y_{t+t} to 0. This will contradict one of the clauses $C_{t+1}^0, C_{t+1}^1, \dots, C_{2t}^0, C_{2t}^1$, and the universal player wins.

It is clear from this explanation, that the existential player has exponentially many choices and the universal player likewise needs to uniquely counter to all these choices to win. The aim of this section is to show that IR-calc and therefore Q-Res in some sense need to go through all these exponentially many options in order to refute the formula, thus forcing IR-calc and Q-Res proofs of exponential size.

Syntactically, $\text{KBKF}(t)$ are *existential Horn formulas*, i.e., they contain at most one positive existential literal per clause. In fact, they even have a stronger property: C_- is the only clause without a head (a positive existential literal). We will strengthen this in the next lemma by a simple modification such that now all clauses have a head.

► **Lemma 2.** *We can transform every IR-calc refutation π of $\text{KBKF}(t)$ into a IR-calc proof π' of y_0 from $\text{KBKF}(t) \setminus \{\neg y_0\}$. We perform this by: (i) deleting every instance of the axiom*

$\{\neg y_0\}$; (ii) for every clause without a positive existential literal we add the literal y_0 to the clause with the empty annotation.

After this transformation, which preserves proof length, we can focus on proofs of y_0 from $\text{KBKF}(t) \setminus \{\neg y_0\}$. Exploiting that all axioms now contain exactly one positive literal we show a number of invariants, which hold for all clauses in all IR-calc proofs of the formulas.

► **Lemma 3.** *Let C be an annotated clause in an IR-calc proof of y_0 from $\text{KBKF}(t) \setminus \{\neg y_0\}$. Then the following invariants hold for C :*

1. C has exactly one positive literal $y_{h,a}^A$ for $h \leq t$ or y_h^A for $h > t$ (or y_0 with no annotation). We call this unique literal the head of C and use the indices h and a also in the following invariants to denote its position as well as A for its annotation.
2. If, for some $j \in [2t]$, $b \in \{0, 1\}$ and B some annotation, $\neg y_{j,b}^B \in C$ (or $\neg y_j^B \in C$), then $j > h$. i.e. literals in the body are always at a higher quantification level than the head.
3. If $\neg y_{j,b}^B \in C$ (or $\neg y_j^B \in C$), then $A \cup \{a/x_h\} \subseteq B$, where all extra annotations in B are of the form c_k/x_k for $k > h$. This invariant acts vacuously for $h > t$ where the clauses contain no negative literals.
4. If $\neg y_{j,b}^B \in C$ (or $\neg y_j^B \in C$) then for all k , $h \leq k < j$ (or $h \leq k \leq t$, when $j > t$) there is $c_k \in \{0, 1\}$ such that $c_k/x_k \in B$.
5. If $\neg y_{j,b}^B \in C$ with $j \leq t$, then for $k \in [t]$, $d \in \{0, 1\}$ and D some annotation, there is no $\neg y_{k,d}^D \in C$ nor $\neg y_{t+k}^D \in C$ such that $B \cup \{b/x_j\} \subseteq D$.

We will now give the overall idea of our lower bound argument. For a clause C we define a set $\Sigma(C)$ of annotations associated with C . Our lower bound argument then rests on counting the set $\Sigma(C)$ as we progress through the proof. More precisely, we show that axioms have empty Σ and that instantiation steps do not change Σ at all. In a resolution step $\frac{D_1}{C} \frac{D_2}{C}$, the set $\Sigma(C)$ either equals $\Sigma(D_1) \cup \Sigma(D_2)$ or grows by exactly one new element. In some sense, we only make progress in the proof in the latter case, and we need exponentially many resolution steps of this kind. Putting everything together we find that by the end of the proof we must have collected all the exponentially many annotations in $\Sigma(y_0)$, implying an exponential lower bound to the proof length (Theorem 6).

We now just give the skeleton of the formal argument. We start with the definition of Σ .

► **Definition 4.** Let C be a clause in an IR-calc proof of y_0 from $\text{KBKF}(t) \setminus \{\neg y_0\}$. We define the set $\Sigma(C)$ of complete annotations (to all x_i) by the following rules.

1. $\Sigma(C) = \emptyset$ when $C = \{y_{t+j}^B\}$ (type-1 clause).
Assume now that C is not type-1 and has the head $y_{h,a}^A$.
2. $\Sigma(C) = \emptyset$ when some x_j , $j < h$ is not given a value in A (type-2 clause).
3. Otherwise (type-3 clause), $\Sigma(C)$ is defined by the following process of adding and removing assignments according to C , which now has complete annotations for each literal by Invariant 4. We start by initialising $\Sigma(C)$ as all complete annotations X to x_1, \dots, x_t such that $A \cup \{a/x_h\} \subseteq X$ (if y_0 is the head we add the complete set of annotations). For each $\neg y_{j,b}^B \in C$ with $j \leq t$ we remove from $\Sigma(C)$ all complete annotations X such that $B \cup \{b/x_j\} \subseteq X$. Invariant 5 ensures that annotations will not be deleted twice here. Finally, we remove annotations B for all $y_j^B \in C$ with $j > t$ (note that B is necessarily complete by Invariants 3 and 4).

For type-3 clauses C , $\Sigma(C)$ counts the complete annotations (and their corresponding literals) resolved away, negative literals are required to be removed and positive literals increase Σ because they can be used to remove a negative literal by resolving. It works by

making each $y_{i,j}$ worth twice as much as $y_{i+1,k}$ because of the C_i^j axioms. Types 1 and 2 are special cases.

The next lemma is the key to our lower bound.

- **Lemma 5.** *Let C be a clause in an IR-calc proof from $\text{KBKF}(t) \setminus \{-y_0\}$.*
1. *If C is the instantiation of an axiom, then $\Sigma(C) = \emptyset$.*
 2. *If C is derived by instantiating D , then $\Sigma(C) = \Sigma(D)$.*
 3. *Let C be derived by resolving D_1 and D_2 . Let \sqcup denote disjoint union. If D_1 is a type-3 clause that is resolved with the type-1 clause $D_2 = \{y_{t+j}^B\}$ for $j > 0$ and there is no $k > 0$, $k \neq j$ such that $\neg y_{t+k}^B \in D_1$, then $\Sigma(C) = \Sigma(D_1) \sqcup \Sigma(D_2) \sqcup \{B\} = \Sigma(D_1) \sqcup \{B\}$. Otherwise $\Sigma(C) = \Sigma(D_1) \sqcup \Sigma(D_2)$.*

We can now deduce that all proofs of $\text{KBKF}(t)$ in IR-calc are of at least exponential size.

- **Theorem 6.** *All proofs of $\text{KBKF}(t)$ in IR-calc have length at least 2^t .*

Since IR-calc simulates Q-Res [8], we get as a corollary the hardness of $\text{KBKF}(t)$ for Q-Res as already stated in [34].

- **Corollary 7.** *All proofs of $\text{KBKF}(t)$ in Q-Res are of at least exponential size.*

As the formulas $\text{KBKF}(t)$ are easy for long-distance and universal resolution [22, 43] we obtain the following exponential separations.

- **Corollary 8.** *IR-calc does neither simulate LD-Q-Res nor QU-Res.*

4 Lower bounds for Q-Res and QU-Res via strategy extraction

This section shows a new and conceptually very different lower bound for QU-Res (and thus for Q-Res). This lower bound constitutes in fact a new lower bound technique that is widely applicable (cf. Sec. 6). We illustrate this technique here with an exponential lower bound for parity formulas in QU-Res. This provides a separation between QU-Res and $\forall\text{Exp}+\text{Res}$.

The lower bound argument hinges on strategy extraction, which is a widely used paradigm in QBF solving and proof systems. We recall that QU-Res admits strategy extraction via a computationally very restricted model, namely decision lists.

- **Definition 9** (decision list [39]). A *decision list* $D = (t_1, c_1), \dots, (t_n, c_n)$ is a finite sequence of pairs where t_i is a term and $c_i \in \{0, 1\}$ is a Boolean constant. Additionally, the last term is the empty term (equivalent to true). For an assignment μ , a decision list D evaluates to c_i if i is the least index such that $\mu \models t_i$, in such case we say that (t_i, c_i) *triggers* under μ .

Winning strategies in form of decision lists can be efficiently extracted from QU-Res proofs:

- **Theorem 10** (Balabanov, Jiang, Widl [3, 4]). *Given a Q-Res or QU-Res refutation π of QBF ϕ , there exists a winning strategy for the universal player for ϕ , such that each of its strategies for the universal variables is computable by a decision list of size polynomial in $|\pi|$.*

Balabanov et al. use a different form than decision lists, but it is semantically equivalent. We deem decision lists as more intuitive for our purposes. Note that that under our definition, a strategy for a universal variable may take as input outputs of strategy functions of smaller index (similarly as in the strategy construction by Goultiaeva et al. [25]).

The general idea behind our lower bound technique is as follows. First, we observe that we can define a family of QBFs ϕ_f , such that every winning strategy of the universal player

must compute a unique Boolean function f (Lemma 12). If we know that strategy extraction is possible by a weak computational model, say AC^0 , we can carefully choose the Boolean formula ϕ_f such that the unique winning strategy f cannot be computed by AC^0 circuits. As the extracted strategy is polynomial in the proof, this implies a lower bound on the proof size. Thus we immediately turn circuit lower bounds to lower bounds for the proof size.

We will now implement this idea for the *parity function* $\text{PARITY}(x_1, \dots, x_n) = x_1 \oplus \dots \oplus x_n$, which is the classical example of a function not computable in AC^0 .

► **Theorem 11** (Furst, Saxe, Sipser [23], Håstad [26]). $\text{PARITY} \notin \text{AC}^0$. *In fact, every non-uniform family of bounded-depth circuits computing PARITY is of exponential size.*

We first observe how to construct a QBF that forces a unique winning strategy.

► **Lemma 12.** *Consider the QBF $\exists x_1, \dots, x_n \forall z. (z \vee \phi_f) \wedge (\neg z \vee \neg \phi_f)$, where ϕ_f is a propositional formula depending only on the variables x_1, \dots, x_n . Let $f : 2^n \rightarrow \{0, 1\}$ be a Boolean function that returns 1 iff ϕ_f evaluates to true. Then there is a unique strategy for the universal player for z , which is $z \leftarrow f$.*

We will now use this idea specifically for the parity function. Consider the QBF $\Phi = \exists X \forall z \exists T. (F^+ \wedge F^-)$ where F^+ is a CNF encoding of $z \vee \text{PARITY}(X)$ and F^- encodes $\neg z \vee \neg \text{PARITY}(X)$. Both F^+ and F^- use additional variables in T . More precisely, for $N > 1$ define QPARITY_N as follows. Let $\text{xor}(o_1, o_2, o)$ be the set of clauses $\{\neg o_1 \vee \neg o_2 \vee \neg o, o_1 \vee o_2 \vee \neg o, \neg o_1 \vee o_2 \vee o, o_1 \vee \neg o_2 \vee o\}$, which defines o to be $o_1 \oplus o_2$. Define QPARITY_N as

$$\exists x_1, \dots, x_N \forall z \exists t_2, \dots, t_N. \text{xor}(x_1, x_2, t_2) \cup \bigcup_{i=3}^N \text{xor}(t_{i-1}, x_i, t_i) \cup \{z \vee t_N, \neg z \vee \neg t_N\}.$$

Note that since we want to encode parity in CNF, i.e. a bounded-depth formula, and $\text{PARITY} \notin \text{AC}^0$, we need to use further existential variables (recall that existential AC^0 characterises all of NP). Choosing existential variables t_i to encode the prefix sums $x_1 \oplus \dots \oplus x_i$ of the parity $x_1 \oplus \dots \oplus x_N$ provides the canonical CNF formulation of parity.

To use the lower bound of Theorem 11 we need to verify that QU-Res enables strategy extraction in AC^0 . This holds as decision lists can be turned into bounded-depth circuits.

► **Lemma 13.** *If f_D can be represented as a polynomial-size decision list D , then $f_D \in \text{AC}^0$.*

Proof. Let $S = \{i \mid (t_i, 1) \in D\}$ be the indices of all pairs in D with 1 as the second component. Observe that f_D evaluates to 1 under μ iff one of the t_i with $i \in S$ triggers under μ . For each t_i with $i \in S$ construct a function $f_i = t_i \wedge \bigwedge_{l=1}^{i-1} \neg t_l$. Construct a circuit for the function $\bigvee_{i \in S} f_i$, which is equal to f_D and is computable in AC^0 as all t_i are just terms. ◀

We can now put everything together and turn the circuit lower bound of Theorem 11 into a lower bound for proof size in QU-Res.

► **Theorem 14.** *Any QU-Res refutation of QPARITY_N is of exponential size in N .*

Proof. By Lemma 12 there is a unique strategy for the variable z in QPARITY_N , which is the PARITY function on N variables. From Theorem 10, there is a polynomial-time algorithm for constructing a decision list D_N from any QU-Res refutation of QPARITY_N . Such decision list can be converted in polynomial time into a circuit with bounded depth by Lemma 13. Hence, the decision list must be of exponential size in N due to Theorem 11. ◀

In contrast to this lower bound, the parity formulas are easy in $\forall \text{Exp} + \text{Res}$.

► **Lemma 15.** *The formulas QPARITY_N have polynomial-size $\forall \text{Exp} + \text{Res}$ refutations.*

Proof sketch. Expand z in both polarities, which generates the clauses $\text{xor}(x_1, x_2, t_2^{0/z}) \cup \bigcup_{i=3}^N \text{xor}(t_{i-1}^{0/z}, x_i, t_i^{0/z}) \cup \{t_N^{0/z}\}$ and $\text{xor}(x_1, x_2, t_2^{1/z}) \cup \bigcup_{i=3}^N \text{xor}(t_{i-1}^{1/z}, x_i, t_i^{1/z}) \cup \{\neg t_N^{1/z}\}$.

Inductively, for $i = 2, \dots, N$ derive clauses representing $t_i^{0/z} = t_i^{1/z}$. This lets us derive a contradiction using the clauses $t_N^{0/z}$ and $\neg t_N^{1/z}$. ◀

Theorem 14 together with Lemma 15 immediately give the following separations.

► **Corollary 16.** *Q-Res and QU-Res do not simulate $\forall\text{Exp+Res}$, IR-calc, IRM-calc.*

This also has consequences for the complexity of strategy extraction in $\forall\text{Exp+Res}$.

► **Corollary 17.** *Winning strategies for $\forall\text{Exp+Res}$ cannot be computed in AC^0 . This even holds when the system $\forall\text{Exp+Res}$ is restricted to formulas with constant quantifier complexity.*

Note, however, that strategy extraction for IRM-calc is in P due to [8, Thm. 4].

5 Extending the lower bound to LD-Q-Res and LQU⁺-Res

We now aim to extend the lower bound from the previous section to stronger QBF proof systems using long-distance resolution. For this we cannot directly use the strategy extraction method from the last section. However, we will slightly modify the parity formulas and then reduce the hardness of those in the stronger systems to the hardness of QPARITY in Q-Res. As the modified formulas remain easy for $\forall\text{Exp+Res}$, these lower bounds imply many new separations between the proof systems involved.

We start by extending the lower bound to LD-Q-Res, which will provide a separation of LD-Q-Res and $\forall\text{Exp+Res}$. For this we consider a variant of the parity formulas from the last section. Let $\text{xor}_l(o_1, o_2, o, z)$ be the set of clauses $\{z \vee \neg o_1 \vee \neg o_2 \vee \neg o, z \vee o_1 \vee o_2 \vee \neg o, z \vee \neg o_1 \vee o_2 \vee o, z \vee o_1 \vee \neg o_2 \vee o\}$ (xor_l defines o to be equal to $o_1 \oplus o_2$ if $z = 0$). The formulas LQPARITY_N are constructed from QPARITY_N by replacing each occurrence of $\text{xor}(\dots)$ by two copies $\text{xor}_l(\dots, z)$ and $\text{xor}_l(\dots, \neg z)$. It is easy to verify that the same arguments as for QPARITY in Section 4 also apply to LQPARITY, yielding:

► **Proposition 18.** *The formulas LQPARITY_N have polynomial-size $\forall\text{Exp+Res}$ refutations, but require exponential-size Q-Res refutations.*

We now want to show that LQPARITY is hard for LD-Q-Res by arguing that long-distance steps do not help to refute these formulas. In the next two lemmas we will show that this actually applies to all QBFs Φ meeting the following condition.

► **Definition 19.** We say that z is *completely blocked* in a QBF Φ , if all clauses of Φ contain the universal variable z and some existential literal l such that $\text{lv}(z) < \text{lv}(l)$.

► **Lemma 20.** *Let Φ be a QBF and z be completely blocked in Φ . Let further C be a clause derived from Φ by LD-Q-Res. If C contains some existential literal l such that $\text{lv}(z) < \text{lv}(l)$, then $z \in C$ or $\neg z \in C$, or $z^* \in C$.*

► **Lemma 21.** *Let Φ be a QBF such that z is completely blocked in Φ and let π be a refutation of Φ such that the variable z is \forall -reduced as early as possible. Then the derivation of the empty clause in π does not contain z^* in any of its clauses.*

This enables us to prove the hardness of LQPARITY in LD-Q-Res.

► **Theorem 22.** *Any refutation of LQPARITY_N in LD-Q-Res is exponential in N .*

Proof. Any LD-Q-Res refutation π can be in polynomial time translated into a refutation π' such that \forall -reductions are carried out as soon as possible (such a refutation has clauses that are equal to the clauses of π or some universal literals are missing). From Lemma 21, the derivation of \perp in π' contains no occurrences of the merged literal z^* , hence any such clauses can be removed from the refutation. Therefore π' is in fact also a Q-Res refutation. Hence, π must be exponential in N due to Proposition 18. \blacktriangleleft

Our next goal is to extend the lower bound for the parity formulas for the system LQU^+ -Res, which enables both long-distance and universal resolution. For such we again modify the formula $QPARITY$, using a similar technique as in [4]. The trick is essentially to double the universal literals so they form tautological clauses when resolved. This way resolution on universal variables does not give any advantage.

We define formulas $QUPARITY_N$ from $LQPARITY_N$ as follows: replace the universal quantifier $\forall z$ by two new quantifiers $\forall z_1 \forall z_2$ and replace all occurrences of the literal z by $z_1 \vee z_2$ and likewise of $\neg z$ by $\neg z_1 \vee \neg z_2$. It is clear that these formulas are false as the universal player should play both z_1 and z_2 as they would z in $QPARITY$. In a similar argument as for $LQPARITY$ we now show that neither long-distance nor universal resolution steps help to refute $QUPARITY$

► **Theorem 23.** $QUPARITY_N$ require exponential-size refutations in LQU^+ -Res.

As $QUPARITY_N$ still remains easy for $\forall\text{Exp}+\text{Res}$ in a proof similar to Lemma 15 we get the following separations.

► **Corollary 24.** LQU^+ -Res does not simulate $\forall\text{Exp}+\text{Res}$, IR-calc, and IRM-calc.

6 Strategy extraction as a general lower bound technique

The results of Sect. 4 can be vastly generalised. We say that a QBF proof system P has *strategy extraction in complexity class C* if from each proof π of a QBF φ , a winning strategy for the universal player, i.e. strategies for all universal variables, can be computed from π in C .

Let L be a language in Σ_k^P/poly for some $k \geq 0$. Let $L = \{x \in \Sigma^* \mid \exists y_1 \forall y_2 \dots Qy_k. A(x, y)\}$, where A is a predicate computable in P/poly . We can thus compute A by a sequence of polynomial-size circuits A_n . The computation of each such circuit A_n can be described by a CNF $C_n(\bar{x}, \bar{y}, \bar{w})$, where \bar{x} are the propositional variables associated with the input x , $\bar{y}_1, \dots, \bar{y}_k$ are the propositional variables for the witnesses y_1, \dots, y_k , and \bar{w} are auxiliary propositional variables describing the gates of the circuit A_n .

Now let $\Phi_{L,n}(\bar{x}, \bar{y}_1, \dots, \bar{y}_k, z, \bar{w}) = \exists \bar{x} \forall z \exists \bar{y}_1 \forall \bar{y}_2 \dots Q \bar{y}_k \exists \bar{w}. (z \leftrightarrow C_n(\bar{x}, \bar{y}_1, \dots, \bar{y}_k, \bar{w}))$. Clearly, this is a false QBF as it expresses that x is both in and outside L . Moreover, from the construction of the formula it is clear that the only winning strategy for the universal player is to play $z = 1 - \chi_L(x)$, where χ_L is the characteristic function of L , and to supply arbitrary values for the remaining universal variables \bar{y}_2 etc. Therefore each winning strategy for the universal player for Φ_L will have to compute the characteristic function of L . This immediately yields conditional lower bounds for proof systems with strategy extraction:

► **Theorem 25.** Let P be QBF proof system with strategy extraction in P/poly . Then P is not polynomially bounded, unless $\text{PH} \subseteq P/\text{poly}$.

Note that the assumption $\text{PH} \not\subseteq P/\text{poly}$ is considered very weak. In fact, even $\text{NP} \cap \text{coNP} \subseteq P/\text{poly}$ is considered unlikely as factoring is in $\text{NP} \cap \text{coNP}$. Also by the Karp-Lipton

theorem [32], $\text{NP} \subseteq \text{P/poly}$ implies that the polynomial hierarchy collapses to the second level, and there are even stronger Karp-Lipton collapse consequences known (cf. [18, 13]).

Theorem 25 can be applied e.g. to IRM-calc, which has strategy extraction in P [8].

► **Corollary 26.** *IRM-calc is not polynomially bounded unless $\text{PH} \subseteq \text{P/poly}$.*

If the proof system allows for strategy extraction via weaker models, then we can improve the conditional lower bounds to unconditional lower bounds, possibly even exponential. We exemplify this paradigm in our next results.

► **Theorem 27.** *Let P be a QBF proof system.*

1. *Let P have strategy extraction in a complexity class C such that the non-uniform version of C is strictly weaker than NP/poly . Then P is not polynomially bounded.*
2. *If P has strategy extraction in AC^0 , then P requires exponential-size proofs, even for formulas of bounded quantifier complexity.*

Our previous Theorem 14 is an instance of item 2 of Theorem 27. In contrast, we can show that the method of strategy extraction is not effective for $\forall\text{Exp}+\text{Res}$ (and therefore neither for IR-calc nor IRM-calc), because all formulas that are potentially hard via the strategy extraction method are easy for $\forall\text{Exp}+\text{Res}$, similarly as in Lemma 15.

► **Proposition 28.** *For every language $L \in \text{P/poly}$ the formulas $\Phi_{L,n}$ have polynomial-size $\forall\text{Exp}+\text{Res}$ refutations.*

We remark that the same method of constructing short $\forall\text{Exp}+\text{Res}$ proofs does not work once we have further universal or existential variables in the formulas, i.e. if L is a language from a level Σ_i^P or Π_i^P with $i \geq 1$.

7 Conclusion

We have shown new lower bounds for Q-Res, IR-calc, LD-Q-Res and LQU⁺-Res, and thereby settled the relative complexity of the main resolution-based QBF calculi. This reveals an almost complete picture of the simulation order of these proof systems (cf. Fig. 1). Most importantly, our results show striking separations between all proof systems modelling CDCL-based QBF solving vs. proof systems modelling expansion-based solving. This provides theoretical evidence that these two paradigms for QBF-solving are indeed complementary and should enhance the power of the solvers when carefully used in conjunction.

Two specific questions that remain open are to show explicit lower bounds for natural QBF formulas in IRM-calc and to fully explore the relationship of this system to universal resolution. With respect to lower bounds for IRM-calc we remark that it is easy to transfer classical resolution lower bounds to this system (e.g., use the existentially closed version of the pigeonhole principle) and thereby improve Corollary 26 to an unconditional lower bound. However, it would be interesting to find meaningful classes of QBFs that are hard for IRM-calc. Regarding the relationship to universal resolution we leave open whether IRM-calc can simulate LQU⁺-Res (but conjecture incomparability of the systems).

A more general and challenging open problem is to determine the extent of the applicability of our new lower bound method via strategy extraction. Here we have shown that this method is very effective for $\exists\forall\exists$ -formulas in Q-Res, but fails for exactly these formulas in expansion-based systems as $\forall\text{Exp}+\text{Res}$ and stronger. Is it possible to use the technique for different types of QBFs even for unconditional lower bounds for stronger QBF proof systems? An open question remains how these techniques apply to the recent systems Q(D)-resolution [42] and QRAT [27].

Acknowledgments This work was partially supported by CMU-Portugal grant AMOS (CMUP-EPB/TIC/0049/2013), FCT grant POLARIS (PTDC/EIA-CCO/123051/2010), INESC-ID's multiannual PIDDAC funding PEst-OE/EEI/LA0021/2011, EPSRC grant EP/L024233/1, and a grant from the John Templeton Foundation. The second author was supported by a Doctoral Training Grant from EPSRC.

References

- 1 Sanjeev Arora and Boaz Barak. *Computational Complexity – A Modern Approach*. Cambridge University Press, 2009.
- 2 Albert Atserias and Sergi Oliva. Bounded-width QBF is PSPACE-complete. *J. Comput. Syst. Sci.*, 80(7):1415–1429, 2014.
- 3 Valeriy Balabanov and Jie-Hong R. Jiang. Unified QBF certification and its applications. *Formal Methods in System Design*, 41(1):45–65, 2012.
- 4 Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In Sinz and Egly [41], pages 154–169.
- 5 Valeriy Balabanov, Magdalena Widl, and Jie-Hong R. Jiang. QBF resolution systems and their proof complexities. In *SAT*, pages 154–169, 2014.
- 6 Marco Benedetti. Evaluating QBFs via symbolic Skolemization. In Franz Baader and Andrei Voronkov, editors, *LPAR*, volume 3452, pages 285–300. Springer, 2004.
- 7 Marco Benedetti and Hratch Mangassarian. QBF-based formal verification: Experience and perspectives. *JSAT*, 5(1-4):133–191, 2008.
- 8 Olaf Beyersdorff, Leroy Chew, and Mikoláš Janota. On unification of QBF resolution-based calculi. In *MFCS, II*, pages 81–93, 2014.
- 9 Olaf Beyersdorff, Leroy Chew, and KartEEK Sreenivasaiyah. A game characterisation of tree-like Q-resolution size. In *LATA*. Springer, 2015.
- 10 Olaf Beyersdorff, Nicola Galesi, and Massimo Lauria. A lower bound for the pigeonhole principle in tree-like resolution by asymmetric prover-delayer games. *Information Processing Letters*, 110(23):1074–1077, 2010.
- 11 Olaf Beyersdorff, Nicola Galesi, and Massimo Lauria. A characterization of tree-like resolution size. *Information Processing Letters*, 113(18):666–671, 2013.
- 12 Olaf Beyersdorff, Nicola Galesi, and Massimo Lauria. Parameterized complexity of DPLL search procedures. *ACM Transactions on Computational Logic*, 14(3), 2013.
- 13 Olaf Beyersdorff and Sebastian Müller. A tight Karp-Lipton collapse result in bounded arithmetic. *ACM Transactions on Computational Logic*, 11(4), 2010.
- 14 Armin Biere. Resolve and expand. In *SAT*, pages 238–246, 2004.
- 15 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.
- 16 Maria Luisa Bonet, Juan Luis Esteban, Nicola Galesi, and Jan Johannsen. On the relative complexity of resolution refinements and cutting planes proof systems. *SIAM J. Comput.*, 30(5):1462–1484, 2000.
- 17 Samuel R. Buss. Towards NP-P via proof complexity and search. *Ann. Pure Appl. Logic*, 163(7):906–917, 2012.
- 18 Jin-Yi Cai. $S_2^P \subseteq ZPP^{NP}$. *Journal of Computer and System Sciences*, 73(1):25–35, 2007.
- 19 Stephen A. Cook and Phuong Nguyen. *Logical Foundations of Proof Complexity*. Cambridge University Press, 2010.
- 20 Stephen A. Cook and Robert A. Reckhow. The relative efficiency of propositional proof systems. *The Journal of Symbolic Logic*, 44(1):36–50, 1979.
- 21 Uwe Egly, Martin Kronegger, Florian Lonsing, and Andreas Pfandler. Conformant planning as a case study of incremental QBF solving. *CoRR*, abs/1405.7253, 2014.

- 22 Uwe Egly, Florian Lonsing, and Magdalena Widl. Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In McMillan et al. [36], pages 291–308.
- 23 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.
- 24 Enrico Giunchiglia, Paolo Marin, and Massimo Narizzano. Reasoning with quantified boolean formulas. In Biere et al. [15], pages 761–780.
- 25 Alexandra Goultiaeva, Allen Van Gelder, and Fahiem Bacchus. A uniform approach for generating proofs and strategies for both true and false QBF formulas. In Toby Walsh, editor, *IJCAI*, pages 546–553. IJCAI/AAAI, 2011.
- 26 Johan Håstad. *Computational Limitations of Small-depth Circuits*. MIT Press, Cambridge, MA, USA, 1987.
- 27 Marijn Heule, Martina Seidl, and Armin Biere. A unified proof system for QBF preprocessing. In *Automated Reasoning – 7th International Joint Conference, IJCAR*, volume 8562, pages 91–106. Springer, 2014.
- 28 Mikoláš Janota, William Klieber, João Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. In Alessandro Cimatti and Roberto Sebastiani, editors, *SAT*, volume 7317, pages 114–128. Springer, 2012.
- 29 Mikoláš Janota, Radu Grigore, and Joao Marques-Silva. On QBF proofs and preprocessing. In McMillan et al. [36], pages 473–489.
- 30 Mikoláš Janota and Joao Marques-Silva. $\forall\text{Exp}+\text{Res}$ does not P-Simulate Q-resolution. International Workshop on Quantified Boolean Formulas, 2013.
- 31 Mikoláš Janota and Joao Marques-Silva. On propositional QBF expansions and Q-resolution. In M. Järvisalo and A. Van Gelder, editors, *SAT*, pages 67–82. Springer, 2013.
- 32 Richard M. Karp and Richard J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proc. 12th ACM Symposium on Theory of Computing*, pages 302–309. ACM Press, 1980.
- 33 Hans Kleine Büning and Uwe Bubeck. Theory of quantified boolean formulas. In Biere et al. [15], pages 735–760.
- 34 Hans Kleine Büning, Marek Karpinski, and Andreas Flögel. Resolution for quantified Boolean formulas. *Inf. Comput.*, 117(1):12–18, 1995.
- 35 Jan Krajíček. Interpolation theorems, lower bounds for proof systems and independence results for bounded arithmetic. *The Journal of Symbolic Logic*, 62(2):457–486, 1997.
- 36 Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors. *Logic for Programming, Artificial Intelligence, and Reasoning, LPAR*. Springer, 2013.
- 37 Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- 38 Jussi Rintanen. Asymptotically optimal encodings of conformant planning in QBF. In *AAAI*, pages 1045–1050. AAAI Press, 2007.
- 39 Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.
- 40 Nathan Segerlind. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 13(4):417–481, 2007.
- 41 Carsten Sinz and Uwe Egly, editors. *Theory and Applications of Satisfiability Testing - SAT*, volume 8561. Springer, 2014.
- 42 Friedrich Slivovsky and Stefan Szeider. Variable dependencies and Q-resolution. In Sinz and Egly [41], pages 269–284.
- 43 Allen Van Gelder. Contributions to the theory of practical quantified Boolean formula solving. In Michela Milano, editor, *CP*, volume 7514, pages 647–663. Springer, 2012.
- 44 Lintao Zhang and Sharad Malik. Conflict driven learning in a quantified Boolean satisfiability solver. In *ICCAD*, pages 442–449, 2002.

Welfare Maximization with Friends-of-Friends Network Externalities

Sayan Bhattacharya^{*1}, Wolfgang Dvořák², Monika Henzinger², and
Martin Starnberger²

- 1 Institute of Mathematical Sciences, Chennai, India.
- 2 University of Vienna, Faculty of Computer Science, Austria

Abstract

Online social networks allow the collection of large amounts of data about the influence between users connected by a friendship-like relationship. When distributing items among agents forming a social network, this information allows us to exploit network externalities that each agent receives from his neighbors that get the same item. In this paper we consider Friends-of-Friends (2-hop) network externalities, i.e., externalities that not only depend on the neighbors that get the same item but also on neighbors of neighbors. For these externalities we study a setting where multiple different items are assigned to unit-demand agents. Specifically, we study the problem of welfare maximization under different types of externality functions. Let n be the number of agents and m be the number of items. Our contributions are the following: (1) We show that welfare maximization is APX-hard; we show that even for step functions with 2-hop (and also with 1-hop) externalities it is NP-hard to approximate social welfare better than $(1 - 1/e)$. (2) On the positive side we present (i) an $O(\sqrt{n})$ -approximation algorithm for general concave externality functions, (ii) an $O(\log m)$ -approximation algorithm for linear externality functions, and (iii) an $(1 - 1/e)^{\frac{1}{6}}$ -approximation algorithm for 2-hop step function externalities. We also improve the result from [6] for 1-hop step function externalities by giving a $(1 - 1/e)/2$ -approximation algorithm.

1998 ACM Subject Classification F.2 Analysis of algorithms and problem complexity, G.1.2 Approximation

Keywords and phrases network externalities, welfare maximization, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.90

1 Introduction

Assume you have to form a committee and need to decide whom to choose as a member. It seems like a good strategy to select members from your network that are well-connected to the whole field so that not only the knowledge of the actual members but also of their whole network can be called upon when needed. Along the same vein assume you want to play a multiplayer online game but you do not have enough friends who are willing to play with you. Then it is a good idea to ask these friends to contact their friends whether they are willing to play as well. Both these settings can be modeled by a social network graph and in both settings not the *direct* (or *1-hop*) neighbors alone, but instead the 1-hop neighbors in combination with the *neighbors of neighbors* (or *2-hop neighbors*) are the decisive factor. Note that the 2-hop neighborhoods cannot be modeled by 1-hop neighborhoods through the

* The work was done while the author was at the University of Vienna, Faculty of Computer Science.



insertion of an additional edge (to the neighbor of the neighbor) as we require that every *participating* neighbor of a neighbor is adjacent to a *participating* neighbor. In the above example, we can only get the opinion of a contact of a contact if we asked the contact before. In the same way, the participation of a friend of a friend will only be possible if there is a participating friend that invites him.

There has been a large body of work by social scientists and, in the last decade, also by computer scientists (see e.g., the influential paper by Kempe, Kleinberg, and Tardos [18] and its citations) to model and analyze the effect of 1-hop neighborhoods. The study of 2-hop neighborhoods has received much less attention (see e.g., [10, 17]). This is surprising as a recent study [14] of the Facebook network shows that the median Facebook user has 31k people as “friends of friends” and due to some users with very large friend lists, the average number of friends-of-friends reaches even 156k. Thus, even if each individual friend of a friend has only a small influence on a Facebook user, in aggregate the influence of the friends-of-friends might be large and should not be ignored.

We, therefore, initiate the study of the influence of 2-hop neighborhoods in the popular *assignment* setting, where items are assigned to users whose values for the item depend on who else in their neighborhood has the item. There is a large body of work on mechanisms and pricing strategies for this problem with a single [5, 15, 4, 1, 7, 19, 11, 3, 13] or multiple items [8, 2, 6, 12, 21, 22, 20, 16] when the valuation function of a user depends *solely* on the 1-hop neighborhood of a user and the user itself. All this work assumes that there is an infinite supply of items (of each type if there are different items) and the users have unit-demand, that is, they want to buy only *one* item. This is frequently the case, for example, if the items model competing products or if the user has to make a binary decision between participating or not participating. In the above examples, this requirement would model that each user can only be in one committee or play one game at a time.

Thus, we study the allocation of items to users in a setting with 2-hop network externalities, where the valuation that a user derives from the products depends on herself, her 1-hop, and her 2-hop neighborhood with the goal of maximizing the social welfare of the allocation. The prior work that is most closely related to our work is the work by Bhargat et al. [6], where they study the multi-item setting with 1-hop externality functions and give approximation algorithms for different classes of externality functions. For linear externalities they give a $1/64$ -approximation algorithm and for step function externalities they get an approximation ratio of $(1 - 1/e)/16 \approx 0.04$. Additionally they present a $2^{O(d)}$ -approximation algorithm for convex externalities that are bounded by polynomials of degree d and a polylogarithmic approximation algorithm for submodular externalities.

1.1 Our Results

The Model: Let $G = (V, E)$ be an undirected graph modeling the social network. Consider any agent $j \in V$ who receives item $i \in I$, and let $S_{ij} \subseteq V \setminus \{j\}$ denote the (2-hop) *support* of agent j for item i : this is the set of agents who contribute towards the valuation of j . Specifically, an agent $j' \in V \setminus \{j\}$ belongs to the set S_{ij} iff j' gets item i and the following condition holds: either j' is a neighbor of j (i.e., $(j, j') \in E$), or j and j' have a common neighbor j'' who also gets item i . The valuation received by agent j is equal to $\lambda_{ij} \cdot \text{ext}_{ij}(|S_{ij}|)$, where λ_{ij} is the agent’s *intrinsic valuation* and $\text{ext}_{ij}(|S_{ij}|)$ is her 2-hop *externality* for item i . The goal is to compute an assignment of items to the agents that maximizes the social welfare, which is defined as the sum of the valuations obtained by the agents.

We study three types of 2-hop externality functions, namely concave, linear and step function externalities.

Step-function externalities: Consider a game requiring a minimal or fixed number of players (larger than two), e.g., Bridge or Canasta, then the externality is a step function. For step functions we show that it is NP-hard to approximate the social welfare within a factor of $(1 - 1/e)$. The result holds for 1-hop and 2-hop externalities. We also show that the problem remains APX-hard when the number of items is restricted to 2. Then we give an $(1 - 1/e)/6 \approx 0.1$ -approximation algorithm for 2-hop step function externalities. Note that this is within a factor of $1/6$ of the hardness bound. Our technique also leads to a *combinatorial* $(1 - 1/e)/2 \approx 0.3$ -approximation algorithm for 1-hop step function externalities, improving the approximation ratio of the *LP-based* algorithm in [6].

Linear externalities: First we show that social welfare maximization for linear 2-hop externality functions is NP-hard.¹ Then we give an $O(\log n)$ -approximation algorithm for linear 2-hop externalities. For these externality functions we can relax the unit-demand requirement. Specifically, we can handle the setting where each user j can buy up to c_j different items, where c_j is a parameter given in the input.²

Concave externalities: We give an $O(\sqrt{n})$ -approximation algorithm when the externality functions $ext_{ij}(\cdot)$ are concave and monotone.

Extensions: Our algorithms for linear and concave externalities can be further generalized to allow a weighting of 2-hop neighbors so that 2-hop neighbors have a lower weight than 1-hop neighbors. This can be useful if it is important that the influence of 2-hop neighbors does not completely dominate the influence of the 1-hop neighbors.

Techniques: The main challenge in dealing with 2-hop externalities is as follows. Fix an agent j who gets an item i , and let $V_i \subseteq V$ denote the set of all agents who get item i . Recall that the agent j 's externality is given by $ext_{ij}(|S_{ij}|)$, where the set S_{ij} is called the support of agent j . The problem is that $|S_{ij}|$, as a function of $V_i \setminus \{j\}$, is not submodular. This is in sharp contrast with the 1-hop setting, where the support for the agent's externality comes only from the set of her 1-hop neighbors who receive item i .

All the mechanisms in [6] use the same basic approach: First solve a suitable LP-relaxation and then round its values independently for each item i . In the 2-hop setting, however, the lack of submodularity of the support size (as described above) leads to many dependencies in the rounding step. Nevertheless, we show how to extend the technique in [6] to achieve the approximation algorithm for linear 2-hop externality functions, using a novel LP. We further give a simple combinatorial algorithm with an approximation guarantee of $O(\sqrt{n})$ for 2-hop concave externalities. For this, we show that either an $\Omega(1/\sqrt{n})$ -fraction of the optimal social welfare comes from a single item, or we can reduce our problem to a setting with 1-hop step function externalities by losing an $(1 - \Omega(1/\sqrt{n}))$ -fraction of the objective.

Our approach for 2-hop step functions is different. We use a novel decomposition of the graph into a maximal set of disjoint connected sets of size 3, 2, and 1. We say an assignment is *consistent* if it assigns all the nodes (i.e., users) in the same connected set the same item.

¹ Theorem 3.1 in [6] claims that the welfare maximization problem for linear 1-hop externality functions in complete graphs is MaxSNP-hard, which would imply our result, but, as we show in the full version, this claim is not true.

² This is also true for the results in [6]. In both results, the assumption is that the valuation functions are additive over the items.

We show first that restricting ourself to consistent assignments reduces the maximum welfare by at most a factor of $1/6$. Finally, we show that finding the optimal consistent assignment is equal to maximizing social welfare in a scenario where agents are not unit demand, do not influence each other, and have valuation functions that are fractionally subadditive in the items they get assigned. For the latter we use the $(1 - 1/e)$ -approximation algorithm by Feige [9].

2 Notations and Preliminaries

We are given a simple undirected graph $G = (V, E)$ with $|V| = n$ nodes. Each node $j \in V$ in this graph is an agent, and there is an edge $(j, j') \in E$ iff the agents j and j' are friends with each other. There is a set of m items $I = \{1, \dots, m\}$. Each item is available in unlimited supply, and each agent wants to get at most one item. An *assignment* $\mathcal{A} : V \rightarrow I$ specifies the item received by every agent, and under this assignment, $u_j(\mathcal{A}, G)$ gives the *valuation* of an agent $j \in V$. Our goal is to find an assignment that maximizes the *social welfare* $\sum_{j \in V} u_j(\mathcal{A}, G)$, i.e., the sum of the valuations of the agents.

Let $F_j^1(G)$ (resp. $F_j^2(G)$) be the 1-hop (resp. 2-hop) neighborhood of node j .

$$F_j^1(G) = \{j' \in V : (j, j') \in E\}, \quad F_j^2(G) = \bigcup_{j' \in F_j^1(G)} F_{j'}^1(G) \setminus (F_j^1(G) \cup \{j\}).$$

Define $V_i(\mathcal{A}, G) = \{j \in V : \mathcal{A}(j) = i\}$ to be the set of agents who receive item $i \in I$ under the assignment \mathcal{A} . Let $N_j^1(i, \mathcal{A}, G) = F_j^1(G) \cap V_i(\mathcal{A}, G)$ denote the set of agents in $F_j^1(G)$ who receive item i under the assignment \mathcal{A} . Further, let $N_j^2(i, \mathcal{A}, G) = F_j^2(G) \cap V_i(\mathcal{A}, G) \cap \left(\bigcup_{j'' \in N_j^1(i, \mathcal{A}, G)} F_{j''}^1(G)\right)$ denote the set of agents in $F_j^2(G)$ who receive item i under the assignment \mathcal{A} and are adjacent to some node in $N_j^1(i, \mathcal{A}, G)$.

The *support* of an agent $j \in V$ for item $i \in I$ is defined as $S_{ij}(\mathcal{A}, G) = N_j^1(i, \mathcal{A}, G) \cup N_j^2(i, \mathcal{A}, G)$. This is the set of agents contributing towards the valuation of j for item i . Let λ_{ij} be the *intrinsic valuation* of agent j for item i , and let $ext_{ij}(|S_{ij}(\mathcal{A}, G)|)$ be the *externality* of the agent for the same item. The agent's valuation from the assignment \mathcal{A} is given by the following equality.

$$u_j(\mathcal{A}, G) = \lambda_{\mathcal{A}(j), j} \cdot ext_{\mathcal{A}(j), j}(|S_{\mathcal{A}(j), j}(\mathcal{A}, G)|).$$

We consider three types of externalities in this paper.

► **Definition 1.** In *concave externality* it holds that $ext_{ij}(t)$ is a monotone and concave function of t , with $ext_{ij}(0) = 0$, for every item $i \in I$ and agent $j \in V$.

► **Definition 2.** In *linear externality* it holds that for all $j \in V$, $i \in I$ and every nonnegative integer t , we have $ext_{ij}(t) = t$.

We extend the step function definition of [6] as follows to 2-hop neighborhoods.

► **Definition 3.** For integer $s \geq 1$, in *s-step function externality* it holds that for all $j \in V$, $i \in I$ and every nonnegative integer t , we have $ext_{ij}(t)$ is 1 if $t \geq s$ and 0 otherwise.

We omit the symbol G from these notations if the underlying graph is clear from the context. Some proofs are omitted due to space restrictions but are provided in a full version available at <http://eprints.cs.univie.ac.at/4240/1/paper-full.pdf>.

3 An $O(\sqrt{n})$ -Approximation for Concave Externalities

For the rest of this section, we fix the underlying graph G , and assume that the agents have concave externalities as per Definition 1. We also fix the intrinsic valuations λ_{ij} and the externality functions $ext_{ij}(\cdot)$.

- Let $\mathcal{A}^* \in \arg \max_{\mathcal{A}} \left\{ \sum_{j \in V} u_j(\mathcal{A}) \right\}$ be an assignment that maximizes the social welfare, and let $\text{OPT} = \sum_{j \in V} u_j(\mathcal{A}^*)$ be the optimal social welfare.
- Let $X^* = \{j \in V : |S_{\mathcal{A}^*(j),j}(\mathcal{A}^*)| \geq \sqrt{n}\}$ be the set of agents with support size at least \sqrt{n} under the assignment \mathcal{A}^* , and let $Y^* = V \setminus X^*$ be the remaining set of agents.

Since X^* and Y^* partition the set of agents V , there can be two possible cases. Half of the social welfare under \mathcal{A}^* is coming (1) either from the agents in X^* , or (2) from the agents in Y^* . Lemma 4 shows that in the former case there is a *uniform assignment*, where every agent gets the same item, that retrieves $1/(2\sqrt{n})$ -fraction of the optimal social welfare. We consider the latter case in Lemma 5, and reduce it to a problem with 1-hop externalities.

► **Lemma 4.** *If $\sum_{j \in X^*} u_j(\mathcal{A}^*) \geq \text{OPT}/2$, then there is an item $i \in I$ such that $\sum_{j \in V} u_j(\mathcal{A}^i) \geq \text{OPT}/(2\sqrt{n})$, where \mathcal{A}^i is the assignment that gives item i to every agent in V , that is, $\mathcal{A}^i(j) = i$ for all $j \in V$.*

Proof. Define the set of items $I(X^*) = \bigcup_{j \in X^*} \{\mathcal{A}^*(j)\}$.

We claim that $|I(X^*)| \leq \sqrt{n}$. To see why the claim holds, let $V_i^* = \{j \in V : \mathcal{A}^*(j) = i\}$ be the set of agents who receive item i under \mathcal{A}^* . Now, fix any item $i \in I(X^*)$, and note that, by definition, there is an agent $j \in X^*$ with $\mathcal{A}^*(j) = i$. Thus, we have $|V_i^*| \geq |S_{ij}(\mathcal{A}^*)| \geq \sqrt{n}$. We conclude that $|V_i^*| \geq \sqrt{n}$ for every item $i \in I(X^*)$. Since $\sum_{i \in I(X^*)} |V_i^*| \leq |V| = n$, it follows that $|I(X^*)| \leq \sqrt{n}$.

To conclude the proof of the lemma, we now make the following observations.

$$\begin{aligned} \sum_{j \in X^*} u_j(\mathcal{A}^*) &= \sum_{i \in I(X^*)} \sum_{j \in X^* : \mathcal{A}^*(j) = i} u_j(\mathcal{A}^*) \leq |I(X^*)| \cdot \max_{i \in I(X^*)} \left(\sum_{j \in X^* : \mathcal{A}^*(j) = i} u_j(\mathcal{A}^*) \right) \\ &\leq \sqrt{n} \cdot \max_{i \in I(X^*)} \left(\sum_{j \in X^* : \mathcal{A}^*(j) = i} u_j(\mathcal{A}^i) \right) \leq \sqrt{n} \cdot \max_{i \in I(X^*)} \left(\sum_{j \in V} u_j(\mathcal{A}^i) \right) \end{aligned}$$

The lemma holds since $\text{OPT}/(2\sqrt{n}) \leq \sum_{j \in X^*} u_j(\mathcal{A}^*)/\sqrt{n} \leq \max_{i \in I(X^*)} \left(\sum_{j \in V} u_j(\mathcal{A}^i) \right)$. ◀

For every item $i \in I$ and agent $j \in V$, we now define the externality function $\hat{ext}_{ij}(t)$ and the valuation function $\hat{u}_j(\mathcal{A})$.

$$\hat{ext}_{ij}(t) = \begin{cases} ext_{ij}(1) & \text{if } t \geq 1; \\ 0 & \text{otherwise.} \end{cases} \quad \hat{u}_j(\mathcal{A}) = \lambda_{\mathcal{A}(j),j} \cdot \hat{ext}_{ij}(|N_j^1(i, \mathcal{A})|) \quad (1)$$

Clearly, for every assignment $\mathcal{A} : V \rightarrow I$, we have $0 \leq \sum_{j \in V} \hat{u}_j(\mathcal{A}) \leq \sum_{j \in V} u_j(\mathcal{A})$. Also note that the valuation function $\hat{u}_j(\cdot)$ depends only on the 1-hop neighborhood of the agent j . Specifically, if an agent j gets an item i , then her valuation is $\lambda_{ij} \cdot ext_{ij}(1)$ if at least one of her 1-hop neighbors also gets the same item i , and zero otherwise. Bhargat et al. [6] gave an LP-based $O(1)$ -approximation for finding an assignment $\mathcal{A} : V \rightarrow I$ that maximizes the social welfare in this setting (also see Section 5 for a combinatorial algorithm). In the lemma below, we show that if the agents in Y^* contribute sufficiently towards OPT under the assignment \mathcal{A}^* , then by losing an $O(\sqrt{n})$ -factor in the objective, we can reduce our original problem to the one where the externalities are $\hat{ext}_{ij}(\cdot)$ and the valuations are $\hat{u}_j(\cdot)$.

► **Lemma 5.** *If $\sum_{j \in Y^*} u_j(\mathcal{A}^*) \geq \text{OPT}/2$, then $\sum_{j \in V} \hat{u}_j(\mathcal{A}^*) \geq \text{OPT}/(2\sqrt{n})$.*

Proof. Consider a node $j \in Y^*$ that makes nonzero contribution towards the objective (i.e., $u_j(\mathcal{A}^*) > 0$) and suppose that it gets items i (i.e., $\mathcal{A}^*(j) = i$). Since $u_j(\mathcal{A}^*) > 0$, we have $S_{ij}(\mathcal{A}^*) = N_j^1(i, \mathcal{A}^*) \cup N_j^2(i, \mathcal{A}^*) \neq \emptyset$, which in turn implies that $N_j^1(i, \mathcal{A}^*) \neq \emptyset$. Thus, we have $\hat{u}_j(\mathcal{A}^*) = \lambda_{ij} \cdot \text{ext}_{ij}(1)$. Since $|S_{ij}(\mathcal{A}^*)| \leq \sqrt{n}$ and $\text{ext}_{ij}(\cdot)$ is a concave function, we have $\text{ext}_{ij}(1) \geq \text{ext}_{ij}(|S_{ij}(\mathcal{A}^*)|)/|S_{ij}(\mathcal{A}^*)| \geq \text{ext}_{ij}(|S_{ij}(\mathcal{A}^*)|)/\sqrt{n}$. Multiplying both sides of this inequality by λ_{ij} , we conclude that $\hat{u}_j(\mathcal{A}^*) \geq u_j(\mathcal{A}^*)/\sqrt{n}$ for all agents $j \in Y^*$ with $u_j(\mathcal{A}^*) > 0$. In contrast, if $u_j(\mathcal{A}^*) = 0$, then the inequality $\hat{u}_j(\mathcal{A}^*) \geq u_j(\mathcal{A}^*)/\sqrt{n}$ is trivially true. Thus, summing over all $j \in Y^*$, we infer that $\sum_{j \in Y^*} \hat{u}_j(\mathcal{A}^*, G) \geq \sum_{j \in Y^*} u_j(\mathcal{A}^*, G)/\sqrt{n} \geq \text{OPT}/(2\sqrt{n})$. The lemma now follows since $\sum_{j \in V} \hat{u}_j(\mathcal{A}^*, G) \geq \sum_{j \in Y^*} \hat{u}_j(\mathcal{A}^*, G)$. ◀

The algorithm for concave externalities. We run two procedures. Procedure (1) returns an assignment $\mathcal{A}' \in \arg \max_{i \in I} \left(\sum_{j \in V} u_j(\mathcal{A}^i) \right)$, where $\mathcal{A}^i(j) = i$ for all $i \in I$ and $j \in V$. Procedure (2) invokes the algorithm in [6] and returns an assignment \mathcal{A}'' such that $\sum_{j \in V} \hat{u}_j(\mathcal{A}'') \geq (1/\alpha) \cdot \max_{\mathcal{A}} \left(\sum_{j \in V} \hat{u}_j(\mathcal{A}) \right)$ for some constant $\alpha \geq 1$, where the function $\hat{u}_j(\cdot)$ is defined as in equation 1. Our algorithm now compares these two assignments \mathcal{A}' and \mathcal{A}'' and returns the one that gives maximum social welfare, i.e., we output an assignment $\mathcal{A}''' \in \arg \max_{\mathcal{A} \in \{\mathcal{A}', \mathcal{A}''\}} \left(\sum_{j \in V} u_j(\mathcal{A}) \right)$.

► **Theorem 6.** *The algorithm described above gives an $O(\sqrt{n})$ -approximation for social welfare under 2-hop, concave externalities.*

Proof. Recall the notations introduced in the beginning of Section 3. Since the set of agents V is partitioned into $X^* \subseteq V$ and $Y^* = V \setminus X^*$, either $\sum_{j \in X^*} u_j(\mathcal{A}^*) \geq \text{OPT}/2$ or $\sum_{j \in Y^*} u_j(\mathcal{A}^*) \geq \text{OPT}/2$. In the former case, Lemma 4 guarantees that $\sum_{j \in \mathcal{A}'''} u_j(\mathcal{A}''') \geq \sum_{j \in \mathcal{A}'} u_j(\mathcal{A}') \geq \text{OPT}/(2\sqrt{n})$. In the latter case, by Lemma 5 we have $\sum_{j \in \mathcal{A}'''} u_j(\mathcal{A}''') \geq \sum_{j \in \mathcal{A}''} u_j(\mathcal{A}'') \geq \sum_{j \in \mathcal{A}''} \hat{u}_j(\mathcal{A}'') \geq \sum_{j \in \mathcal{A}^*} \hat{u}_j(\mathcal{A}^*)/\alpha \geq \text{OPT}/(2\alpha\sqrt{n})$. Since α is a constant, we conclude that the social welfare returned by our algorithm is always within an $O(\sqrt{n})$ -factor of the optimal social welfare. ◀

4 An $O(\log m)$ -Approximation for Linear Externalities

In this section, we assume that the input graph $G = (V, E)$ is of the following form. The set V is partitioned into three groups V_1, V_2 and V_3 . Further, an edge in E either connects a node in V_1 with a node in V_2 , or connects a node in V_2 with a node in V_3 . Our goal is to assign the items to the agents in such a way as to maximize the social welfare from the set V_1 . We refer to this problem as RESTRICTED-WELFARE.

► **Theorem 7.** *Any α -approximation algorithm for the RESTRICTED-WELFARE problem can be converted into an $O(\alpha)$ -approximation algorithm for the welfare-maximization problem in general graphs with linear (or even concave) externalities.*

Consider the LP below. Here, the variable $\alpha(i, j, k)$ indicates if both the agents $j \in V_1$ and $k \in F_j^1$ received item $i \in I$. If this variable is set to one, then agent j gets one unit of externality from agent k . Similarly, the variable $\beta(i, j, l)$ indicates if both the agents $j \in V_1, l \in V_3 \cap F_j^2$ received item $i \in I$ and there is at least one agent $k \in F_j^1 \cap F_l^1$ who also received the same item. If this variable is set to one, then agent j gets one unit of externality from agent l . Clearly, the total valuation of agent j for item i is given by $\sum_{k \in V_2 \cap F_j^1} \lambda_{ij} \cdot \alpha(i, j, k) + \sum_{l \in V_3 \cap F_j^2} \lambda_{ij} \cdot \beta(i, j, l)$. Summing over all the items and all the agents in V_1 , we see that the LP-objective encodes the social welfare of the set V_1 .

$$\text{Maximize: } \sum_{j \in V_1} \sum_{i \in I} \lambda_{ij} \cdot \left(\sum_{k \in V_2 \cap F_j^1} \alpha(i, j, k) + \sum_{l \in V_3 \cap F_j^2} \beta(i, j, l) \right) \quad (2)$$

$$\beta(i, j, l) \leq \min\{w(i, l), y(i, j)\} \quad \forall i \in I, j \in V_1, l \in V_3 \cap F_j^2 \quad (3)$$

$$\beta(i, j, l) \leq \sum_{k \in F_j^1 \cap F_l^1} z(i, k) \quad \forall i \in I, j \in V_1, l \in V_3 \cap F_j^2 \quad (4)$$

$$\alpha(i, j, k) \leq \min\{y(i, j), z(i, k)\} \quad \forall i \in I, j \in V_1, k \in V_2 \cap F_j^1 \quad (5)$$

$$\sum_i y(i, j) \leq 1, \sum_i z(i, k) \leq 1, \sum_i w(i, l) \leq 1 \quad \forall j, k, l \quad (6)$$

$$0 \leq z(i, k), y(i, j), w(i, l), \alpha(i, j, k), \beta(i, j, l) \quad \forall i, j, k, l \quad (7)$$

The variables $y(i, j)$, $z(i, k)$ and $w(i, l)$ respectively indicate if an agent $j \in V_1$, $k \in V_2$, $l \in V_3$ received item $i \in I$. Constraints 6 state that an agent can get at most one item. Constraint 5 says that if $\alpha(i, j, k) = 1$, then both $y(i, j)$ and $z(i, k)$ must also be equal to one. Constraint 3 states that if $\beta(i, j, l) = 1$, then both $y(i, j)$ and $w(i, l)$ must also be equal to one. Finally, note that if an agent $l \in V_3$ contributes one unit of externality to an agent $j \in V_1$ for an item $i \in I$, then there must be some agent $k \in F_j^1 \cap F_l^1$ in V_2 who received the same item. This condition is encoded in constraint 4. Thus, we have the following lemma.

► **Lemma 8.** *The LP is a valid relaxation of the RESTRICTED-WELFARE problem.*

Before proceeding towards the rounding scheme, we perform a preprocessing step as described in the next lemma.

► **Lemma 9.** *In polynomial time, we can get a feasible solution to the LP that gives an $O(\log m)$ approximation to the optimal objective, and ensures that each $\alpha(i, j, k), \beta(i, j, l), y(i, j), w(i, l) \in \{0, \gamma\}$ for some real number $\gamma \in [0, 1]$, and that each $z(i, k) \leq \gamma$.*

We now present the rounding scheme for LP (see Algorithm 1). Here, the set W_i denotes the set of agents that have not yet been assigned any item when the rounding scheme enters the FOR loop for item i (see Step 2). Note that the sets T_i might overlap, but these conflicts are resolved in Line 9 by intersecting T_i with W_i , which is disjoint with all previous $T_j, j < i$.

Algorithm 1 Rounding Scheme for LP

1. In accordance with Lemma 9, compute a feasible solution to the LP.
Set $T_0 \leftarrow \emptyset$, and $W_0 \leftarrow V = V_1 \cup V_2 \cup V_3$.
 2. FOR all items $i \in I = \{1, \dots, m\}$:
 3. Set $W_i \leftarrow W_{i-1} \setminus T_{i-1}$, and $T_i \leftarrow \emptyset$.
 4. Pick a value η_i uniformly at random from $[0, 1]$.
 5. IF $\eta_i \leq \gamma$:
 6. FOR all nodes $j \in V_1$:
IF $y(i, j) = \gamma$, then with probability $1/4$, set $T_i \leftarrow T_i \cup \{j\}$.
 7. FOR all nodes $l \in V_3$:
IF $w(i, l) = \gamma$, then with probability $1/4$, set $T_i \leftarrow T_i \cup \{l\}$.
 8. FOR all nodes $k \in V_2$:
With probability $z(i, k)/(4\gamma)$, set $T_i \leftarrow T_i \cup \{k\}$.
 9. Assign item i to all nodes in $W_i \cap T_i$, i.e., set $\mathcal{A}(t) \leftarrow i$ for all $t \in W_i \cap T_i$.
 10. RETURN the (random) assignment \mathcal{A} .
-

► **Lemma 10.** *For all $t \in V$ and all $i \in I$, we have $\mathbf{P}[t \in W_i] \geq 3/4$. Thus, $\mathbf{P}[\{t_1, t_2, t_3\} \subseteq W_i] \geq 1/4$ for all $t_1, t_2, t_3 \in V$.*

Proof. We will prove the lemma for a node in V_1 , the argument extends to $V_2 \cup V_3$.

Fix any node $j \in V_1$ and any item $i \in I$, and consider an indicator random variable $\Gamma_{i'j}$ that is set to one iff $j \in T_{i'}$. It is easy to check that $\mathbf{E}[\Gamma_{i'j}] = y(i', j)/4$ for all items $i' \in I$. By constraint 6 and linearity of expectation, we thus have: $\mathbf{E}[\sum_{i' < i} \Gamma_{i'j}] = \sum_{i' < i} y(i', j)/4 \leq 1/4$. Applying Markov's inequality, we get $\mathbf{P}[\sum_{i' < i} \Gamma_{i'j} = 0] \geq 3/4$. In other words, with probability at least $3/4$, we have that $j \notin T_{i'}$ for all $i' < i$. Under this event, we must have $j \in W_i$.

We have $\mathbf{P}[t \notin W_i] \leq 1/4$ for all $t \in \{t_1, t_2, t_3\}$. $\mathbf{P}[\{t_1, t_2, t_3\} \subseteq W_i] \geq 1/4$ now follows from applying union-bound over these three events. \blacktriangleleft

In the first step, when we find a feasible solution to the LP in accordance with Lemma 9, we lose a factor of $O(\log m)$ in the objective. Below, we will show that the remaining steps in the rounding scheme result in a loss of at most a constant factor in the approximation ratio.

For all items $i \in I$, nodes $j \in V_1$, and nodes $k \in F_j^1, l \in F_j^2$, we define the random variables $X(i, j, k)$ and $Y(i, j, l)$. Their values are determined by the outcome \mathcal{A} of our randomized rounding. To be more specific, we have that $X(i, j, k) = 1$ if both j and k receive item i , and $X(i, j, k) = 0$ otherwise. Further, $Y(i, j, l) = 1$ if both j and l receive item i and there is some node in $F_j^1 \cap F_l^1$ that also received item i , and $Y(i, j, l) = 0$ otherwise. Now, the valuation of any agent $j \in V_1$ from the (random) assignment \mathcal{A} is:

$$u_j(\mathcal{A}) = \sum_{i \in I} \left(\sum_{k \in F_j^1} \lambda_{ij} \cdot X(i, j, k) + \sum_{l \in F_j^2} \lambda_{ij} \cdot Y(i, j, l) \right) \quad (8)$$

We will analyze the expected contribution of the rounding scheme to each term in the LP-objective. Towards this end, we prove the following lemmas.

► **Lemma 11.** *For all $i \in I, j \in V_1, k \in F_j^1$, we have $\mathbf{E}_{\mathcal{A}}[X(i, j, k)] \geq \delta \cdot \alpha(i, j, k)$, where $\delta > 0$ is a sufficiently small constant.*

► **Lemma 12.** *For all $i \in I, j \in V_1, l \in F_j^2$, we have $\mathbf{E}_{\mathcal{A}}[Y(i, j, l)] \geq \delta \cdot \beta(i, j, l)$, where δ is a sufficiently small constant.*

Proof. Fix an item $i \in I$, a node $j \in V_1$ and a node $l \in F_j^2$. If $\beta(i, j, l) = 0$ the lemma is trivially true. Otherwise suppose for the rest of the proof that $\beta(i, j, l) = y(i, j) = w(i, l) = \gamma$.

Let \mathcal{E}_i be the event that $\eta_i \leq \gamma$ (see Step 4 in Algorithm 1). Let $Z(i, k)$ be an indicator random variable that is set to one iff node $k \in V_2$ is included in the set T_i by our rounding scheme (see Step 8 in Algorithm 1). We have:

$$\mathbf{P}[\mathcal{E}_i] = \gamma, \text{ and } \mathbf{P}[Z(i, k) = 1 \mid \mathcal{E}_i] = z(i, k)/4\gamma \text{ for all } k \in V_2 \quad (9)$$

Thus, conditioned on the event \mathcal{E}_i , the expected number of common neighbors of j and l who are included in the set T_i is given by

$$\mu_i := \mathbf{E} \left[\sum_{k \in F_j^1 \cap F_l^1} Z(i, k) \mid \mathcal{E}_i \right] = \sum_{k \in F_j^1 \cap F_l^1} z(i, k)/4\gamma \geq 1/4 \quad (10)$$

Note that conditioned on the event \mathcal{E}_i , the random variables $Z(i, k)$ are mutually independent. Thus, applying Chernoff bound on Equation 10, we infer that with constant probability, at least one common neighbor of j and l will be included in the set T_i . To be more precise, define $T_{i,j,l} = T_i \cap F_j^1 \cap F_l^1$. For some sufficiently small constant δ_1 , we have:

$$\mathbf{P} \left[T_{i,j,l} \neq \emptyset \mid \mathcal{E}_i \right] = \mathbf{P} \left[\sum_{k \in F_j^1 \cap F_l^1} Z(i, k) > 0 \mid \mathcal{E}_i \right] \geq 1 - e^{-1/8} = \delta_1 \quad (11)$$

Let $\mathcal{E}_{i,j,l}$ be the event that the following two conditions hold simultaneously: (a) $T_{i,j,l} \neq \emptyset$, AND (b) j, l , and an arbitrary node from $T_{i,j,l}$ —each of these three nodes is included in W_i . Now, Equation 11 and Lemma 10 imply that $\mathbf{P}[\mathcal{E}_{i,j,l} | \mathcal{E}_i] \geq \delta_2$ for $\delta_2 = \delta_1/4$. Putting all these observations together, we obtain that $\mathbf{P}[Y(i, j, l) = 1] = \mathbf{P}[\mathcal{E}_i] \cdot \mathbf{P}[\mathcal{E}_{i,j,l} | \mathcal{E}_i]$. $\mathbf{P}[j, l \in T_i | \mathcal{E}_{i,j,l} \cap \mathcal{E}_i] = \gamma \cdot \delta_2 \cdot (1/4) \cdot (1/4) = \delta \cdot \gamma = \delta \cdot \beta(i, j, l)$ for $\delta = \delta_2/16$. \blacktriangleleft

► **Theorem 13.** *The rounding scheme in Algorithm 1 gives an $O(\log m)$ -approximation to the RESTRICTED-WELFARE problem.*

Proof. In the first step, when we find a feasible solution to the LP in accordance with Lemma 9, we lose a factor of $O(\log m)$ in the objective. At the end of the remaining steps, the expected valuation of an agent $j \in V_1$ is given by:

$$\begin{aligned} \mathbf{E}_{\mathcal{A}}[u_j(\mathcal{A})] &= \sum_{i \in I} \lambda_{ij} \cdot \left(\sum_{k \in F_j^1} \mathbf{E}_{\mathcal{A}}[X(i, j, k)] + \sum_{l \in F_l^1} \mathbf{E}_{\mathcal{A}}[Y(i, j, l)] \right) \\ &= \Theta \left(\sum_{i \in I} \lambda_{ij} \cdot \left(\sum_{k \in F_j^1} \alpha(i, j, k) + \sum_{l \in F_l^1} \beta(i, j, l) \right) \right) \end{aligned}$$

The first equality follows from linearity of expectation, while the second equality follows from Lemma 11 and Lemma 12. Thus, the expected valuation of any agent in V_1 is within a constant factor of the fractional valuation of the same agent under the feasible solution to the LP obtained at the end of Step 1 (see Algorithm 1). Summing over all the agents in V_1 , we get the theorem. \blacktriangleleft

We can generalize the above approach to the following setting: Each user j is given an integer c_j and can be assigned up to c_j different items (each at most once). For this we replace for each item i and node j the constraint $\sum_i y(i, j) \leq 1$ by the two constraints $\sum_i y(i, j) \leq c_j$ and $y(i, j) \leq 1$ and adapt the proof of Lemma 10.

Finally, we state NP-hardness for linear externalities, not only in the 2-hop setting but also for 1-hop.³

► **Theorem 14.** *Maximizing social welfare under linear externalities is NP-hard.*

5 Constant Approximation for Step Function Externalities

In this section, our goal is to maximize the social welfare when the agents have general step function externalities, i.e., to receive externality an agent needs a certain number of 1- and 2-hop neighbors having the same product. We will show that no constant factor approximation is possible unless a bound on the number of neighbors an agent needs to receive externality is given. Thus we consider the case of 2-step function externalities, where only two neighbors are needed (see Definition 3) and give a $\frac{1}{6} \cdot (1 - 1/e)$ -approximation algorithm for this problem. Notice that if we consider step functions that just require one neighbor the problem reduces to the 1-hop step function scenario in [6]. However, our algorithm gives a $\frac{1}{2} \cdot (1 - 1/e)$ -approximation for this scenario improving the result in [6].

In the following we assume 2-step function externalities. Let $G_{V'}$ denote the subgraph induced by $V' \subseteq V$. For the rest of this section, the term “triple” will refer to any (unordered)

³ Theorem 3.1 in [6] claims that the welfare maximization problem for linear 1-hop externality functions in complete graphs is MaxSNP-hard, which would imply our result, but this claim is not true.

set of three nodes $T = \{j_1, j_2, j_3\}$ such that G_T is connected. Similarly, the term “pair” will refer to any (unordered) set of two nodes $\{j_1, j_2\}$ that are connected by an edge in E .

We first compute a maximal collection of mutually disjoint triples in the graph G . We denote this collection by \mathcal{T} , and let $V(\mathcal{T}) = \bigcup_{T \in \mathcal{T}} T \subseteq V$. The graph $G_{V \setminus V(\mathcal{T})}$, by definition, consists of a mutually disjoint collection of pairs (say \mathcal{P}) and a set of isolated nodes (say B). We thus have the following lemma.

► **Lemma 15.** *In $G = (V, E)$, there is no edge that connects a node $j \in B$ with another node in B or with a node belonging to a pair in \mathcal{P} . Furthermore, there is no edge that connects two nodes j, j' belonging to two different pairs $P, P' \in \mathcal{P}$.*

► **Definition 16.** An assignment \mathcal{A} is *consistent* iff two agents get the same item whenever they belong to the same triple or the same pair. To be more specific, for all $j, j' \in V$, we have that $\mathcal{A}(j) = \mathcal{A}(j')$ if either (a) $j, j' \in T$ for some triple $T \in \mathcal{T}$ or (b) $\{j, j'\} \in \mathcal{P}$.

The next lemma shows that by losing a factor of 6 in the approximation ratio, we can focus on maximizing the social welfare via a consistent assignment.

► **Lemma 17.** *The social welfare from the optimal consistent assignment is at least $(1/6) \cdot \text{OPT}$, where OPT is the maximum social welfare over all assignments.*

Proof. Let \mathcal{A}^* be an assignment (not necessarily consistent) that gives maximum social welfare. We convert it into a (random) consistent assignment \mathcal{A} as follows. For each triple $\{j_1, j_2, j_3\} \in \mathcal{T}$, we pick one of the items $\mathcal{A}^*(j_1), \mathcal{A}^*(j_2), \mathcal{A}^*(j_3)$ uniformly at random, and assign that item to all the three agents j_1, j_2, j_3 . Similarly, for each pair $\{j_1, j_2\} \in \mathcal{P}$, we pick one of the items $\mathcal{A}^*(j_1), \mathcal{A}^*(j_2)$ uniformly at random, and assign that item to both the agents j_1, j_2 . The events corresponding to different triples and pairs are mutually independent. Finally, the remaining agents (those who are in B) get the same items as in \mathcal{A}^* . It is easy to see that the resulting assignment \mathcal{A} is consistent. We claim that $\mathbf{E}[u_j(\mathcal{A})] \geq (1/6) \cdot u_j(\mathcal{A}^*)$ for all $j \in V$. To prove this claim, we consider three cases.

Case 1 ($j \in B$): Let $\mathcal{A}^*(j) = i$. Since $j \in B$, it always gets the same item under \mathcal{A} , i.e., $\mathcal{A}(j) = i$. Now, if $u_j(\mathcal{A}^*) = 0$, then the claim is trivially true. Otherwise it must be the case that $\mathcal{A}^*(j') = i$ for some neighbor j' of j . Since $j \in B$, this neighbor j' must be part of some triple $T \in \mathcal{T}$ (see Lemma 15). With probability at least $1/3$ all the three nodes in T are assigned item i under \mathcal{A} and at least two nodes of T are in the 2-hop neighborhood of j . In that event j gets the same valuation as in \mathcal{A}^* , and we have that $\mathbf{E}[u_j(\mathcal{A})] \geq (1/3) \cdot u_j(\mathcal{A}^*)$.

Case 2 (j belongs to a pair in \mathcal{P}): Consider the pair $P = \{j, j'\} \in \mathcal{P}$, which has j and another node (say j') as its members. Let $\mathcal{A}^*(j) = i$. As in Case 1, if $u_j(\mathcal{A}^*) = 0$, then the claim is trivially true. Otherwise it must be the case that there exists a node j'' with $\mathcal{A}^*(j'') = i$ such that j'' is either a neighbor of j or a neighbor of j' . Since $\{j, j'\} \in \mathcal{P}$, this agent j'' must be part of some triple $T \in \mathcal{T}$ (see Lemma 15). Let \mathcal{E}_1 be the event that all the three nodes in T are assigned item i under \mathcal{A} . Similarly, let \mathcal{E}_2 be the event that both the nodes $j, j' \in P$ get the same item i under \mathcal{A} . Since these two events are mutually independent, we have that $\mathbf{P}[\mathcal{E}_1 \cap \mathcal{E}_2] \geq (1/3) \cdot (1/2) = 1/6$, and in the event $\mathcal{E}_1 \cap \mathcal{E}_2$, we have $u_j(\mathcal{A}) = u_j(\mathcal{A}^*)$. It follows that $\mathbf{E}[u_j(\mathcal{A})] \geq (1/6) \cdot u_j(\mathcal{A}^*)$.

Case 3 (j belongs to a triple in \mathcal{T}): Consider the triple $T = \{j, j', j''\} \in \mathcal{T}$ which has, besides j , two other nodes (say j' and j'') as its members. With probability at least $1/3$, all these three nodes are assigned item $\mathcal{A}^*(j)$ under \mathcal{A} , and in this event we have $u_j(\mathcal{A}) \geq u_j(\mathcal{A}^*)$. It follows that $\mathbf{E}[u_j(\mathcal{A})] \geq (1/3) \cdot u_j(\mathcal{A}^*)$.

Now, we take a sum of the inequalities $\mathbf{E}[u_j(\mathcal{A})] \geq (1/6) \cdot u_j(\mathcal{A}^*)$ over all agents $j \in V$, and by linearity of expectation infer that the expected social welfare under the consistent assignment \mathcal{A} is within a factor of 6 of the optimal social welfare. This concludes the proof of the lemma. \blacktriangleleft

Next, we will give an $(1 - 1/e)$ -approximation algorithm for finding a consistent assignment of items that maximizes the social welfare. Along with Lemma 17, this will imply the main result of this section (see Theorem 20).

We use the term “resource” to refer to either a pair $P \in \mathcal{P}$ or an agent $j \in B$. Let $\mathcal{R} = \mathcal{P} \cup B$ denote the set of all resources. We say that a resource $r \in \mathcal{R}$ *neighbors* a triple $T \in \mathcal{T}$ iff in the graph $G = (V, E)$ either (a) $r = \{j, j'\} \in \mathcal{P}$ and some node in $\{j, j'\}$ is adjacent to some node in T , or (b) $r = j \in B$ and j is adjacent to some node in T . We slightly abuse the notation (see Section 2) and let $N(T) \subseteq \mathcal{R}$ denote the set of resources that are neighbors of $T \in \mathcal{T}$.

By definition, every consistent assignment ensures that if two agents belong to the same triple in \mathcal{T} (resp. the same pair in \mathcal{P}), then both of them get the same item. We say that *the item is assigned to a triple (resp. resource)*. Note that the triples do not need externality from outside. To be more specific, the contribution of a triple $T \in \mathcal{T}$ to the social welfare is always equal to $\sum_{j \in T} \lambda_{i,j}$, where i is the item assigned to T . Resources, however, do need outside externality, which by Lemma 15 can come only from a triple in \mathcal{T} .

► **Lemma 18.** *In a consistent assignment, if a resource $r \in \mathcal{R}$ makes a positive contribution to the social welfare, then it neighbors some triple $T_r \in \mathcal{T}$, and both the resource r and the triple T_r receive the same item.*

Proof. If a resource contributes a nonzero amount to the social welfare, then it must receive nonzero externality from the assignment. By Lemma 15, such externality can come only from a triple in \mathcal{T} . The lemma follows. \blacktriangleleft

Thus, given a consistent assignment \mathcal{A} consider the following mapping $T_{\mathcal{A}}(r)$ of a resource $r \in \mathcal{R}$ to triples in \mathcal{T} in accordance with Lemma 18: If the resource r makes zero contribution towards the social welfare (a case not covered by the lemma), then we let $T_{\mathcal{A}}(r)$ be any arbitrary triple from \mathcal{T} . Otherwise $T_{\mathcal{A}}(r)$ denotes an (arbitrary) neighboring triple of \mathcal{T} that receives the same item as r . We say that the triple $T_{\mathcal{A}}(r)$ *claims* the resource r .

For ease of exposition, let $\lambda_{i,r}(T)$ be the valuation of the resource r when both the resource and the triple T that claims it get item $i \in I$, i.e.,

$$\lambda_{i,r}(T) = \begin{cases} \lambda_{i,j} + \lambda_{i,j'} & \text{if } r = \{j, j'\} \in \mathcal{P} \text{ and } r \in N(T); \\ \lambda_{i,j} & \text{if } r = j \in B \text{ and } r \in N(T); \\ 0 & \text{if } r \notin N(T). \end{cases}$$

Now, any consistent assignment \mathcal{A} can be interpreted as follows. Under such an assignment, every triple $T \in \mathcal{T}$ claims the subset of the resources $S_T = \{r \in \mathcal{R} : T_{\mathcal{A}}(r) = T\}$; the subsets corresponding to different triples being mutually exclusive. A triple T and the resources in S_T all get the same item (say $i \in I$). The valuation obtained from them is $u_T(S_T, i) = \sum_{j \in T} \lambda_{i,j} + \sum_{r \in S_T} \lambda_{i,r}(T)$.

If our goal is to maximize the social welfare, then, naturally, for every triple T , we will pick the item that maximizes $u_T(S_T, i)$, thereby extracting a valuation of $u_T(S_T) = \max_i u_T(S_T, i)$. The next lemma shows that this function is fractionally subadditive.

► **Lemma 19.** *The function $u_T(S_T)$ is fractionally subadditive in S_T .*

The preceding discussion shows that the problem of computing a consistent assignment for welfare maximization is equivalent to the following setting. We have a collection of triples \mathcal{T} , and a set of resources \mathcal{R} . We will distribute these resources amongst the triples, i.e., every triple T will get a subset $S_T \subseteq \mathcal{R}$, and these subsets will be mutually exclusive. The goal is to maximize the sum $\sum_{T \in \mathcal{T}} u_T(S_T)$, where the functions $u_T(\cdot)$'s are fractionally subadditive. By a celebrated result of Feige [9], we can get an $(1 - 1/e)$ -approximation algorithm for this problem if we can implement the following subroutine (called *demand oracle*) in polynomial time: Each resource r is given a “cost” $p(r)$ and we need to determine for each triple T a set of resources S_T^* that maximizes $u_T(S_T) - \sum_{r \in S_T} p(r)$ over all sets S_T . Such a demand oracle can be implemented in polynomial time using a simple greedy algorithm for each T and each item i : Add a resource r to S_T^* iff $\lambda_{i,r}(T) > p(r)$. The result of the approximation algorithm assigns each triple T a subset S_T and we then pick the item i that maximizes $u_T(S_T, i)$ over all items i . Together with Lemma 15, this implies the theorem stated below.

► **Theorem 20.** *We can get a polynomial-time $\frac{1}{6} \cdot (1 - 1/e)$ -approximation algorithm for the problem of maximizing social welfare under 2-step function externalities.*

The algorithm can be easily adapted for 1-hop step function externalities. The difference is that instead of computing a maximal collection \mathcal{T} of mutually disjoint triples, one computes a maximal collection of mutually disjoint pairs.

► **Theorem 21.** *We can get a polynomial-time $\frac{1}{2} \cdot (1 - 1/e)$ -approximation algorithm for maximizing social welfare under 1-step function externalities.*

Finally, we present our hardness results for step functions. By a reduction from MAX INDEPENDENT SET we can show that, for unbounded s , there is no constant factor approximation. The main idea is that we modify the graph such that we replace each edge by a path of length three and each of the original nodes j wants a different item, while j can only get positive externalities when having a support of $2\delta_j$ (δ_j the node degree of j). The valuations of the newly introduced nodes are set to 0. That is, nodes that are adjacent in the original graph have two common neighbors in the constructed graph, want different items, need all their neighbors as support, and thus only one of them can have positive valuation.

► **Theorem 22.** *For any $\varepsilon > 0$ the problem of maximizing social welfare under arbitrary s -step function externalities is not approximable within $O(n^{1/4-\varepsilon})$ unless $\text{NP} = \text{P}$, and not approximable within $O(n^{1/2-\varepsilon})$ unless $\text{NP} = \text{ZPP}$.*

Second, we show that maximizing social welfare under 2-step function externalities is APX-hard and thus no PTAS can exist. This is by a reduction from MAX COVERAGE. The APX-hardness for two items is by a reduction from SAT.

► **Theorem 23.** *The problem of maximizing social welfare under step function externalities is APX-hard, in particular, there is no polynomial time $1 - \frac{1}{e} + \epsilon$ -approximation algorithm (unless $\text{P} = \text{NP}$). Furthermore, the problem remains APX-hard (although with a larger constant) even if there are only two items.*

Acknowledgments The research leading to these results has received funding from the European Union’s Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 317532 and from the Vienna Science and Technology Fund (WWTF) through project ICT10-002.

References

- 1 Hessameddin Akhlaghpour, Mohammad Ghodsi, Nima Haghpanah, Vahab S. Mirrokni, Hamid Mahini, and Afshin Nikzad. Optimal iterative pricing over social networks. In *6th WINE*, pages 415–423, 2010.
- 2 Noga Alon, Michal Feldman, Ariel D. Procaccia, and Moshe Tennenholtz. A note on competitive diffusion through social networks. *Inf. Process. Lett.*, 110(6):221–225, 2010.
- 3 Nima Anari, Shayan Ehsani, Mohammad Ghodsi, Nima Haghpanah, Nicole Immorlica, Hamid Mahini, and Vahab S. Mirrokni. Equilibrium pricing with positive externalities. *Theor. Comput. Sci.*, 476:1–15, 2013.
- 4 David Arthur, Rajeev Motwani, Aneesh Sharma, and Ying Xu. Pricing strategies for viral marketing on social networks. In *5th WINE*, pages 101–112, 2009.
- 5 Bernard Bensaid and Jean-Philippe Lesne. Dynamic monopoly pricing with network externalities. *Int. J. of Industrial Organization*, 14(6):837–855, 1996.
- 6 Anand Bhalgat, Sreenivas Gollapudi, and Kamesh Munagala. Mechanisms and allocations with positive network externalities. In *13th EC*, pages 179–196, 2012.
- 7 Sayan Bhattacharya, Dmytro Korzhyk, and Vincent Conitzer. Computing a profit-maximizing sequence of offers to agents in a social network. In *8th WINE*, pages 482–488, 2012.
- 8 Pradeep Dubey, Rahul Garg, and Bernard De Meyer. Competing for customers in a social network: The quasi-linear case. In *2nd WINE*, pages 162–173, 2006.
- 9 Uriel Feige. On maximizing welfare when utility functions are subadditive. *SIAM J. Comput.*, 39(1):122–142, 2009.
- 10 Scott L. Feld. Why your friends have more friends than you do. *American J. of Sociology*, 96(6):1464–1477, 1991.
- 11 Dimitris Fotakis and Paris Siminelakis. On the efficiency of influence-and-exploit strategies for revenue maximization under positive externalities. In *8th WINE*, pages 270–283, 2012.
- 12 Sanjeev Goyal and Michael Kearns. Competitive contagion in networks. In *44th STOC*, pages 759–774, 2012.
- 13 Nima Haghpanah, Nicole Immorlica, Vahab S. Mirrokni, and Kamesh Munagala. Optimal auctions with positive network externalities. *ACM Trans. Economics and Comput.*, 1(2):13:1–13:24, 2013.
- 14 Keith N. Hampton, Lauren Sessions Goulet, Cameron Marlow, and Lee Rainie. Why most facebook users get more than they give. *Pew Internet & American Life Project*, 2012.
- 15 Jason Hartline, Vahab S. Mirrokni, and Mukund Sundararajan. Optimal marketing strategies over social networks. In *17th WWW*, pages 189–198, 2008.
- 16 Xinran He and David Kempe. Price of anarchy for the n-player competitive cascade game with submodular activation functions. In *9th WINE*, pages 232–248, 2013.
- 17 Matthew O. Jackson and Brian W. Rogers. Meeting strangers and friends of friends: How random are social networks? *American Economic Review*, 97(3):890–915, 2007.
- 18 David Kempe, Jon M. Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *9th KDD*, pages 137–146, 2003.
- 19 Vahab S. Mirrokni, Sebastien Roch, and Mukund Sundararajan. On fixed-price marketing for goods with positive network externalities. In *8th WINE*, pages 532–538, 2012.
- 20 Sunil Simon and Krzysztof R. Apt. Choosing products in social networks. In *8th WINE*, pages 100–113, 2012.
- 21 Reiko Takehara, Masahiro Hachimori, and Maiko Shigeno. A comment on pure-strategy nash equilibria in competitive diffusion games. *Inf. Process. Lett.*, 112(3):59–60, 2012.
- 22 Vasileios Tzoumas, Christos Amanatidis, and Evangelos Markakis. A game-theoretic analysis of a competitive diffusion process over social networks. In *8th WINE*, pages 1–14, 2012.

Markov Decision Processes and Stochastic Games with Total Effective Payoff *

Endre Boros¹, Khaled Elbassioni², Vladimir Gurvich¹, and Kazuhisa Makino⁴

- 1 MSIS Dep. of RBS and RUTCOR, Rutgers University; 100 Rockafeller Road, Piscataway, NJ 08854-8054, USA
{endre.boros,vladimir.gurvich}@rutgers.edu
- 2 Masdar Institute of Science and Technology, P.O. Box 54224, Abu Dhabi, UAE
kelbassioni@masdar.ac.ae
- 3 Research Institute for Mathematical Sciences (RIMS) Kyoto University, Kyoto 606-8502, Japan
makino@kurims.kyoto-u.ac.jp

Abstract

We consider finite Markov decision processes (MDPs) with undiscounted *total* effective payoff. We show that there exist uniformly optimal pure stationary strategies that can be computed by solving a polynomial number of linear programs. We apply this result to two-player zero-sum stochastic games with perfect information and undiscounted total effective payoff, and derive the existence of a saddle point in uniformly optimal pure stationary strategies.

1998 ACM Subject Classification G.3 Probability and Statistics, G.1.6 Optimization

Keywords and phrases Markov decision processes, undiscounted stochastic games, linear programming, mean payoff, total payoff

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.103

1 Introduction

1.1 Basic concepts

1.1.1 Markov decision processes

We will consider Markov decision processes (MDPs) with *total effective payoff*. Let $G = (V, E)$ be a *finite* directed graph (digraph) in which loops and multiple arcs are allowed. The vertices $v \in V$ are called positions (or states) and the arcs $e \in E$ are called *moves* (or transitions). The vertex-set V is partitioned into two subsets $V = V_W \cup V_R$ that correspond to white and random positions, controlled respectively, by a player (decision maker), who will be called MAX, and by nature. Let us denote by $E(u)$ the set of arcs leaving u and assume that $E(u) \neq \emptyset$ in every position $u \in V$.

For all random positions $u \in V_R$ we are given probabilities $p(u, v) > 0$ for all random moves $(u, v) \in E(u)$ such that $\sum_{(u,v) \in E(u)} p(u, v) = 1$. There is also a *local reward* function $r : E \rightarrow \mathbb{Z}$ given. The triplet $\Gamma = (G, p, r)$ will be called an MDP.

* This research was partially supported by the Scientific Grant-in-Aid from Ministry of Education, Science, Sports and Culture of Japan. The first author also thanks for partial support the National Science Foundation (Grant and IIS-1161476).

1.1.2 Strategies

The vertices represent the states of a finite state dynamical system. If at time t the system is in state $v_t = u \in V_W$ then the controller (MAX) chooses (as an action) one of the outgoing arcs $(u, v) \in E(u)$ with some probability and the system moves with this probability to $v_{t+1} = v$. If $v_t = u \in V_R$, then the system moves to $v_{t+1} = v$ with probability $p(u, v)$ (MAX has no influence over this move.)

A strategy (policy) of MAX is a mapping \mathfrak{s} that for every possible $v_t = u \in V_W$ provides a probability distribution over $E(u)$. These probabilities may depend, in general, not only on u and t but also on the entire history of the system up to time t . If these probabilities take only values 0 and 1, then the strategy \mathfrak{s} is called *pure*; if these probabilities depend only on the current state u , then \mathfrak{s} is called *stationary*. A pure stationary strategy is also called *positional*. We shall denote by \mathfrak{S} the set of all possible strategies and by $\tilde{\mathfrak{S}}$ the set of all positional strategies.

Once MAX chooses a strategy $\mathfrak{s} \in \mathfrak{S}$, and we fix an initial state v_0 , the above process produces a series of states $v_t(\mathfrak{s}) \in V$, $t = 0, 1, \dots$, which generally are random variables for $t > 0$. We associate to such a process the sequence of expected local rewards

$$a_t(\mathfrak{s}) = \mathbb{E}_{\mathfrak{s}}[r(v_t(\mathfrak{s}), v_{t+1}(\mathfrak{s}))] \quad \text{for } t = 0, 1, \dots,$$

and set $a(\mathfrak{s}) = \langle a_0(\mathfrak{s}), a_1(\mathfrak{s}), \dots \rangle$. For simplicity we will omit in the sequel the argument \mathfrak{s} and write v_t and $\mathbb{E}_{\mathfrak{s}}(r(v_t, v_{t+1}))$ rather than $v_t(\mathfrak{s})$ and $\mathbb{E}_{\mathfrak{s}}[r(v_t(\mathfrak{s}), v_{t+1}(\mathfrak{s}))]$ for $t = 0, 1, \dots$

1.1.3 Effective payoffs

We consider an effective payoff function $\pi : \mathbb{R}^* \rightarrow \bar{\mathbb{R}}$, where $\bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ and \mathbb{R}^* standardly denotes the set of all real sequences. The objective of MAX is to find a strategy $\mathfrak{s} \in \mathfrak{S}$ such that $\pi(a(\mathfrak{s})) = \pi_{\mathfrak{s}}(v_0)$ is as large as possible. A strategy \mathfrak{s} is called *uniformly optimal* if $\pi_{\mathfrak{s}}(v_0) \geq \pi_{\mathfrak{s}'}(v_0)$ for any strategy $\mathfrak{s}' \in \mathfrak{S}$ and any initial position $v_0 \in V$.

In this paper we consider the following two effective payoff functions:

$$\phi_{\mathfrak{s}}(v_0) = \liminf_{T \rightarrow \infty} \frac{1}{T+1} \sum_{t=0}^T \mathbb{E}_{\mathfrak{s}}[r(v_t, v_{t+1})], \quad (1)$$

$$\psi_{\mathfrak{s}}(v_0) = \liminf_{T \rightarrow \infty} \frac{1}{T+1} \sum_{t=0}^T \sum_{j=0}^t \mathbb{E}_{\mathfrak{s}}[r(v_j, v_{j+1})]. \quad (2)$$

The first one, called *mean payoff*, is classic [12, 4]. The second one, called *total payoff* or *total reward*, was introduced by Thuijsman and Vrieze [27], as a “refinement” of the mean payoff. Let us note however that in fact total payoff MDPs can be shown to include mean payoff MDPs as a special case.

We note that in many earlier works the effective payoff of a play was defined as the *sum* of all local rewards assigned to the moves of this play. Yet, evaluation of the infinite plays may constitute a problem. For that reason, in most of the papers an assumption has to be made such as termination with probability one [7, 9, 25, 3, 31, 30]; in fact definition (2) is a generalization of the sum of local rewards, taking properly into account how to handle cycling in an infinite (non-terminating) play; see Section 1.3.

For an MDP Γ , payoff function π , and a node u , let us define

$$\pi_{\Gamma}(u) = \sup_{\mathfrak{s} \in \mathfrak{S}} \pi_{\mathfrak{s}}(u),$$

as the *value* of the MDP at node u .

1.1.4 Stochastic games with perfect information: The BWR model

We also consider the following natural and standard generalization. Assume that the finite vertex set V of a given finite directed graph $G = (V, E)$ is partitioned into three (rather than two) subsets $V = V_B \cup V_W \cup V_R$ that correspond to *black*, *white*, and *random* positions, controlled respectively, by two players, MIN and MAX, and nature.

Analogously to MDPs, we can define strategies for the players, and denote by \mathfrak{S}_B and \mathfrak{S}_W the sets of strategies of MIN and MAX, respectively. Given a pair of strategies $\mathfrak{s} = (\mathfrak{s}_B, \mathfrak{s}_W)$ of the players and an initial vertex $v_0 \in V$, we can associate a sequence of expected rewards $\mathbb{E}_{\mathfrak{s}}[r(v_t(\mathfrak{s}), v_{t+1}(\mathfrak{s}))]$ to these, just like we did for MDPs. The objectives of MIN and MAX are to minimize and respectively maximize the expected effective payoff $\pi_{\mathfrak{s}_B, \mathfrak{s}_W}(v_0) = \pi_{\mathfrak{s}}(v_0)$.

Given a stochastic game with a fixed initial position v_0 , a *saddle point* is defined as a pair of strategies $\mathfrak{s}_B^* \in \mathfrak{S}_B$ and $\mathfrak{s}_W^* \in \mathfrak{S}_W$ such that

$$\pi_{\mathfrak{s}_B^*, \mathfrak{s}_W}(v_0) \leq \pi_{\mathfrak{s}_B^*, \mathfrak{s}_W^*}(v_0) \leq \pi_{\mathfrak{s}_B, \mathfrak{s}_W^*}(v_0) \quad \text{for all } \mathfrak{s}_B \in \mathfrak{S}_B \text{ and } \mathfrak{s}_W \in \mathfrak{S}_W. \quad (3)$$

If such a pair exists, the quantity $\pi_{\mathfrak{s}_B^*, \mathfrak{s}_W^*}(v_0)$ is called the value of the game at node v_0 . The saddle point $(\mathfrak{s}_B^*, \mathfrak{s}_W^*)$ is called *uniform (subgame perfect)* if the above inequalities hold for all initial positions $v_0 \in V$.

For $\pi = \phi$, such a model was first mentioned in [13], and it was shown in [6] that it is polynomially equivalent with stochastic games with perfect information [12]. For $\pi = \psi$, this model is the same as the one introduced in [27] in case of perfect information. The concept was further developed in [8, 28].

1.2 Main results

We first consider total-payoff MDPs and prove the following result.

► **Theorem 1.** *In every MDP with total effective payoff, $\pi = \psi$, MAX possesses a uniformly optimal positional strategy. Moreover, such a strategy, together with the optimal value can be found in polynomial time.*

For mean payoff MDPs, the analogous result is well-known, see, e.g. [16, 4, 7, 23]. In fact there are several known approaches to construct the optimal stationary strategies. For instance, a polynomial-time algorithm to solve mean payoff MDPs is based on solving two associated linear programs, see, e.g., [7].

Our approach for proving Theorem 1 is inspired by a result of [28]. We extend this result to characterize the existence of pure and stationary optima within *all* possible strategies by the feasibility of an associated linear system. Next, we show that this system is always feasible and a solution can be obtained by solving a polynomial number of linear programming problems.

► **Remark.** If there are no random nodes in the MDP, then a uniformly optimal stationary strategy can be found by a combinatorial algorithm that solves a polynomial number of minimum mean-cycle problems [18]; we omit the details from this version.

► **Theorem 2.** *Every BWR-game with total effective payoff, $\pi = \psi$, has a saddle point in uniformly optimal positional strategies.*

For the mean payoff games with perfect information the above result is well-known [12, 22].

Let us note that there may be no stationary best response against a non-stationary strategy of the opponent. However, for the case of total payoff BWR-games, Theorem 1

implies that for any stationary strategy of a player there is a pure stationary best response (among all strategies) of the opponent. This fact implies that it is enough to construct a saddle point within the family of positional strategies. This latter can be shown by using the discounted formulation of the game.

1.3 Applications of the total payoff

1.3.1 Total payoff MDPs/games with a terminating condition

This is the special case of MDPs/ stochastic games with one special *terminal* state, which is absorbing (that is, $p(t, t) = 1$) and cost free (that is, $r(t, t) = 0$). The payoff function, which is also sometimes called “Total Payoff” is defined as the sum

$$\theta_{\mathfrak{s}}(v_0) = \liminf_{T \rightarrow \infty} \sum_{t=0}^T \mathbb{E}_{\mathfrak{s}}[r(v_t, v_{t+1})].$$

This type of MDPs/games have been considered under different names, such as *stochastic shortest path problems/games*, *first passage problems* and *transient programming problems* [1, 2, 3, 7, 14, 25, 29, 31, 30]. This can be thought of as a generalization of the classical (deterministic) shortest path problem on graphs, with the difference that, at each node, one should select a probability distribution over successor nodes, out of a given set of probability distributions. The objective is that the chosen *random* path leads to the terminal node with probability one and with the smallest expected cost. In order to establish the existence of optimal stationary strategies that can be derived by the solutions of Bellman-type equations, several assumptions have been made in earlier works, most notably, the existence of a *proper* stationary strategy, i.e., one that guarantees termination from every state with probability 1. Note that for such a proper strategy \mathfrak{s} , the resulting Markov chain contains exactly one absorbing class, namely the terminal node, and in this case, it is not hard to see that the values obtained from sum payoff $\theta_{\mathfrak{s}}$ and the total payoff $\psi_{\mathfrak{s}}$ are the same. Thus total payoff MDPs/games considered in this paper can be thought of as a generalization of shortest path problems/games, when we do not assume that there is a single terminal.

1.3.2 The shortest path interdiction problem (SPIP)

This is the special case of shortest path games when there are no random nodes. More precisely, in this problem, edges have positive lengths and there is a dedicated terminal vertex to which the minimizer tries to find a short path, while the opponent tries to block such paths. It is easy to see that if we add a loop with zero length on the terminal vertex then the total payoff ψ will be exactly the length of the path for every terminating path, and will be $+\infty$ otherwise.

The problem was introduced by Fulkerson and Harding [10]; see a short survey by Israely and Wood [17]. The simplest version is as follows: Given a digraph $G = (V, E)$, with weighted arcs $r : E \rightarrow \mathbb{Z}_+$, and two vertices $s, t \in V$, eliminate (at most) k arcs of E to maximize the length of a shortest (s, t) -path. While this problem is APX-hard [20], the following *vertex-wise budget* SPIP is tractable [21, 20]: we are given a budget allowing to eliminate (at most) $k(v)$ arcs going from each state $v \in V$. This version was considered in [21], where an efficient interdiction algorithm was obtained. Given a digraph $G = (V, E)$, an integer-valued local cost function $r : E \rightarrow \mathbb{Z}_+$, a constraint $k(v)$ in every vertex $v \in V$, and an initial vertex s , this algorithm finds in quadratic time an interdiction that maximizes simultaneously the

lengths of all shortest paths from s to each vertex $v \in V$. The execution time is just slightly larger than for the classic Dijkstra shortest path algorithm.

Waving the non-negativity condition from the latter version, we obtain another interesting relation: In this case, the SPIP becomes equivalent [21] with solving mean payoff BW-games (no random nodes). Although the latter problem is known to be in the intersection of NP and co-NP [19, 32], yet, it is not known to be polynomial.

1.3.3 Scheduling with and/or precedence constraints

[24] is another application of the total payoff with $r \geq 0$. Given a digraph $G = (V, E)$, whose states are interpreted as jobs, the and/or precedence constraints require that some jobs $u \in V$ cannot be started before *all* immediate predecessors (v such that $(v, u) \in E$) are completed, while some other jobs $w \in V$ cannot be started before *at least one* immediate predecessor is complete. It is easy to see that this model is equivalent with a total reward BW-game which has nonnegative local rewards. For this problem [24] provides a polynomial time algorithm.

2 Characterization of pure stationary optima in total MDPs

Our proof of Theorem 1 is based on strengthening a result of Thuijsman and Vrieze (Theorem 5.3 in [28]) which gives a sufficient and necessary condition for a general total reward stochastic game to have a saddle point when both players are *restricted to stationary strategies*. In case of MDPs, this amounts to the feasibility of a linear program of the form that will be described in Section 3. In this section, we show that the existence of a solution for this LP implies in fact the existence of an optimal solution in positional strategies, even if each player is allowed to choose from the space of all, possibly history-dependent, strategies. Our proof relies heavily on the concept of a *potential transformation* and relating the total and mean effective payoffs of a transformed game to those in the original game.

2.1 Potential transformation

Let us consider a mapping $x : V \rightarrow \mathbb{R}$, whose values $x(v)$ will be called *potentials*, and define the transformed reward local function $r[x] : E \rightarrow \mathbb{R}$ as:

$$r[x](u, v) = r(u, v) - x(u) + x(v), \quad \text{where } (u, v) \in E. \quad (4)$$

Potential transforms were first introduced in 1958 by Gallai [11], then applied to stochastic games in 1966 by Hoffman and Karp [15] and to B -games (that is, min mean-cycles) in 1978 by Karp [18].

Given a potential transformation x , and an MDP $\Gamma = (G, p, r)$, let us denote by $\phi[x]$ (similarly, $\psi[x]$) the optimal effective payoff vectors in the transformed MDP $\Gamma[x] = (G, p, r[x])$. Let us further associate to such a potential vector the quantity $\mathcal{M}(x) = 2 \max_{v \in V} |x(v)|$.

Let us also introduce

$$\widehat{\phi}_s[x](v_0) = \limsup_{T \rightarrow \infty} \frac{1}{T+1} \sum_{t=0}^T \mathbb{E}_s[r[x](v_t, v_{t+1})], \quad \text{and}$$

$$\widehat{\psi}_s[x](v_0) = \limsup_{T \rightarrow \infty} \frac{1}{T+1} \sum_{t=0}^T \sum_{i=0}^t \mathbb{E}_s[r[x](v_i, v_{i+1})],$$

and for $x = \mathbf{0}$ write $\widehat{\phi}_s[0](v_0) = \widehat{\phi}_s(v_0)$, and analogously $\widehat{\psi}_s[0](v_0) = \widehat{\psi}_s(v_0)$.

► **Fact 1** (see, e.g., [6]). For any MDP Γ , there exists a potential y such that, if $v_0 = v \in V$ is the initial vertex and $t \in \mathbb{Z}_+$, then

- (i) $\mathbb{E}_{\mathfrak{s}}[r[y](v_t, v_{t+1})] \leq \phi_{\Gamma}(v)$ for any arbitrary strategy \mathfrak{s} ;
- (ii) $\mathbb{E}_{\mathfrak{s}^*}[r[y](v_t, v_{t+1})] = \phi_{\Gamma}(v)$ for some positional strategy \mathfrak{s}^* .

2.2 Characterization of pure and stationary optima

Let us start with a few useful properties connecting mean payoff and total payoff values.

► **Lemma 3.** *If for a strategy $\mathfrak{s} \in \mathfrak{S}$ and initial vertex $v_0 \in V$ we have $\phi_{\mathfrak{s}}(v_0) > 0$, then $\psi_{\mathfrak{s}}(v_0) = +\infty$. Analogously, if we have $\widehat{\phi}_{\mathfrak{s}}(v_0) < 0$, then $\widehat{\psi}_{\mathfrak{s}}(v_0) = -\infty$.*

► **Lemma 4.** *Assume that $\sup_{\mathfrak{s}} \phi_{\mathfrak{s}}(v) \leq 0$ for all $v \in V$, and denote by y a corresponding potential transformation as in Fact 1. Then we have the following relations hold for all strategies $\mathfrak{s} \in \mathfrak{S}$ and initial vertices $v_0 \in V$:*

$$\phi_{\mathfrak{s}}(v_0) = \phi_{\mathfrak{s}}[y](v_0) \leq 0, \quad (5a)$$

$$\widehat{\phi}_{\mathfrak{s}}(v_0) = \widehat{\phi}_{\mathfrak{s}}[y](v_0) \leq 0, \quad (5b)$$

and

$$\psi_{\mathfrak{s}}(v_0) \leq \widehat{\psi}_{\mathfrak{s}}(v_0) \leq \mathcal{M}(y) < \infty. \quad (5c)$$

► **Lemma 5.** *Assume that $\sup_{\mathfrak{s}} \phi_{\mathfrak{s}}(v) \leq 0$ for all $v_0 \in V$. Then if $\phi_{\mathfrak{s}}(v_0) < 0$ for a strategy $\mathfrak{s} \in \mathfrak{S}$, then $\widehat{\psi}_{\mathfrak{s}}(v_0) = -\infty$.*

The following corollary of Lemma 3 and Fact 1 states that the total payoff in an MDP is not finite if the mean payoff is not zero.

► **Corollary 6.** *For an MDP and a node u , we have*

$$\begin{aligned} \phi_{\Gamma}(u) > 0 &\implies \psi_{\Gamma}(u) = \widehat{\psi}_{\Gamma}(u) = +\infty, \\ \phi_{\Gamma}(u) < 0 &\implies \psi_{\Gamma}(u) = \widehat{\psi}_{\Gamma}(u) = -\infty. \end{aligned}$$

► **Lemma 7.** *Assume that $\sup_{\mathfrak{s}} \phi_{\mathfrak{s}}(v) \leq 0$ for all $v \in V$, and that $\mathfrak{s} \in \mathfrak{S}$ is a strategy with initial vertex v_0 such that $\psi_{\mathfrak{s}}(v_0)$ is finite. Then we have*

$$\phi_{\mathfrak{s}}(v_0) = \widehat{\phi}_{\mathfrak{s}}(v_0) = 0.$$

For brevity, we will use the following notation throughout the rest of this section: Given a mapping $f : E(u) \rightarrow \mathbb{R}$ and a subset $F \subseteq E(u)$ we write

$$M_{(u,v) \in F}[f] = \begin{cases} \max_{(u,v) \in F} f(u, v), & \text{for } u \in V_W, \\ \text{avg}_{(u,v) \in F} f(u, v), & \text{for } u \in V_R, \end{cases}$$

where $\text{avg}_{(u,v) \in F}(f(v, u)) := \sum_{(u,v) \in F} p(u, v) f(u, v)$.

► **Theorem 8.** *For a total reward MDP $\Gamma = (G, P, r)$, the following two statements are equivalent:*

- (i) *the value vector ψ_{Γ} exists, is finite, and MAX possesses a uniformly optimal positional strategy (optimal among all strategies);*

(ii) the following set of equations has a (finite) solution for variables $\mu, x \in \mathbb{R}^V$, $\alpha \in \mathbb{R}_+$:

$$\mu(u) = M_{(u,v) \in E(u)}[r(u,v) + \mu(v)] \quad \text{for all } u \in V, \quad (6a)$$

$$\mu(u) = M_{(u,v) \in E(u)}[\alpha r(u,v) + x(v) - x(u)] \quad \text{for all } u \in V, \quad (6b)$$

$$\mu(u) = M_{(u,v) \in \text{EXT}(u)}[\alpha r(u,v) + x(v) - x(u)] \quad \text{for all } u \in V_W, \quad (6c)$$

where, for a vertex $u \in V_W$, $\text{EXT}(u)$ denotes the set of arcs in $E(u)$ attaining equality in (6a).

Let us remark that the series of Lemmas we used to prove the above theorem remain true if we replace in the definitions of ϕ and ψ the operator \liminf with \limsup . Thus, Theorem 8 also holds with this modified definition, too. Consequently, switching the controller to a "minimizer" an analogous theorem will hold, since we can obtain this situation by changing the sign of all local rewards, switching to \limsup in the definitions of ϕ and ψ , and then applying the above theorem with a "maximizer." This observation will be useful for using the "symmetry" between the players in proving Theorem 2.

3 LP formulation

Our purpose in this section is to show that in a total reward MDP, the optimal solution can always be realized by a positional strategy that can be obtained in polynomial time. One of the main ingredients in this proof is the treatment of the case when

$$\phi_\Gamma(u) = 0 \quad \forall u \in V. \quad (\text{A})$$

In this section we shall assume that the above condition holds, and show that in this case the optimal solution can be obtained via solving a small series of linear programs. To arrive to the proof of this statement, we need a series of technical lemmas.

Based on the idea of [28] let us associate to Γ the following linear programming problem $\text{LP}(\alpha)$, where $\alpha \in \mathbb{R}$ is a real parameter. Recall that $E(u) = \{(u,v) \in E \mid v \in N^+(u)\}$, where $N^+(u)$ is the set of out-neighbors of vertex u .

$$\min \sum_{u \in V} y(u)$$

$$s.t. \quad (7a)$$

$$y(u) \geq r(u,v) + y(v) \quad \forall u \in V_W, (u,v) \in E(u) \quad (7b)$$

$$y(u) \geq \text{avg}_{v \in N^+(u)} (r(u,v) + y(v)) \quad \forall u \in V_R \quad (7c)$$

$$y(u) \geq \alpha r(u,v) - x(u) + x(v) \quad \forall u \in V_W, (u,v) \in E(u) \quad (7d)$$

$$y(u) \geq \text{avg}_{v \in N^+(u)} (\alpha r(u,v) - x(u) + x(v)) \quad \forall u \in V_R. \quad (7e)$$

The main idea is to show that this LP has an optimal solution satisfying conditions (6a)-(6c) of Theorem 8 (with $y(u) = \mu(u)$). For this we need to show that, starting from an arbitrary optimal solution (x, y) , we can construct another optimal solution (x^*, y^*) such that for all $u \in V_W$, there is an arc $(u, v) \in E$ such that the inequalities (7b) and (7d), corresponding to this arc, are tight at (x^*, y^*) .

Given a feasible solution (x, y) of $\text{LP}(\alpha)$, let us denote by $I^u(y)$ the set of arcs $(u, v) \in E(u)$ for which (7b) holds with equality, and let $J_\alpha^u(x, y)$ denote the set of arcs $(u, v) \in E(u)$ for which (7d) is an equality. Furthermore, let us denote by $I^R(y)$ the set of vertices $u \in V_R$ for

which (7c) holds with equality, and let $J_\alpha^R(x, y)$ denote the set of vertices $u \in V_R$ for which (7e) holds with equality.

In view of Theorem 8, it will be enough to show the following:

► **Theorem 9.** *Under Assumption (A), if $\alpha > 0$ is large enough then $LP(\alpha)$ has an optimal solution (x^*, y^*) such that*

$$\emptyset \neq J_\alpha^u(x^*, y^*) \subseteq I^u(y^*) \text{ and } J_\alpha^R(x^*, y^*) = I^R(y^*) = V_R.$$

To arrive to the proof of this claim, we need several technical lemmas. Let us first show that this linear program has a finite optimum whenever α is nonnegative. We break this claim into two lemmas:

► **Lemma 10.** *Problem $LP(\alpha)$ is feasible, if $\alpha \geq 0$.*

► **Lemma 11.** *Problem $LP(\alpha)$ is bounded.*

Let us then denote by $Z(\alpha)$ the optimum value of $LP(\alpha)$.

► **Corollary 12.** *The value $Z(\alpha)$ exists and is finite for all $\alpha \geq 0$.*

Strengthening Lemma 11, we can get an explicit lower bound on $Z(\alpha)$, of *polynomial bit-length* in terms of the input size (assuming rational input), as follows.

► **Lemma 13.** *For any feasible solution (x, y) in $LP(\alpha)$, $\alpha \geq 0$, and for any vertex $u \in V$ and any strategy $\mathfrak{s} \in \mathfrak{S}$, we have $y(u) \geq \psi_{\mathfrak{s}}(u)$.*

► **Corollary 14.** *There exists a real $L \in \mathbb{R}$ such that we have $L \leq Z(\alpha)$ for all $\alpha \geq 0$.*

Proof. By Lemma 13 we have $\sum_{u \in V} y(u) \geq \sum_{u \in V} \psi_{\mathfrak{s}}(u)$, for all feasible solutions (x, y) of $LP(\alpha)$, $\alpha \geq 0$ and for any strategy $\mathfrak{s} \in \mathfrak{S}$. Let us now fix a uniformly optimal stationary strategy \mathfrak{s} of the mean-payoff MDP (which we know to exist, see, e.g., [23]). It was shown in [28] that under the assumption (A) we have $\psi_{\mathfrak{s}}(u)$ finite for all vertices $u \in V$ (see Proposition 1 in Section 5.2 for an explicit formula). Consequently, $L = \sum_{u \in V} \psi_{\mathfrak{s}}(u)$ is a finite lower bound (of polynomial bit-length) on the objective function value of any feasible solution of $LP(\alpha)$ for any $\alpha \geq 0$. ◀

► **Lemma 15.** *There exists a finite real α_0 (of polynomial bit-length in terms of the input size), such that $Z(\alpha) = Z(\alpha_0)$ for all $\alpha > \alpha_0$.*

► **Lemma 16.** *Let us consider $\alpha \geq \alpha_0$ and denote by (x^*, y^*) an arbitrary optimal solution of $LP(\alpha)$. Then, we have $I^R(y^*) = V_R$ and $I^u(y^*) \neq \emptyset$ for all $u \in V_W$.*

To arrive to a proof of Theorem 9, which is the main aim of this section, it will not be enough simply to take an optimal solution of $LP(\alpha)$ for a large enough value of α , e.g., for $\alpha \geq \alpha_0$. While the optimal values in y^* will be indeed optimal in the MDP, the additional conditions of Theorem 9 call for a careful selection of an optimal x^* . In fact $LP(\alpha)$ typically has many optimal solutions, even if we fix the values in y^* , and the rest of the proof will focus on showing how can we find efficiently an appropriate x^* satisfying all conditions of Theorem 9.

To this end let us fix an optimal solution (x^*, y^*) of $LP(\alpha)$ for some $\alpha \geq \alpha_0$, and consider the polyhedron $X_\alpha(y^*)$ defined as the set of feasible $x \in \mathbb{R}^V$ vectors in the following system of inequalities:

$$\begin{aligned} 0 &\geq \alpha r(u, v) - y^*(u) - x(u) + x(v) && \forall u \in V_W, (u, v) \in I^u(y^*) \\ 0 &\geq \operatorname{avg}_{v \in N^+(u)} (\alpha r(u, v) - y^*(u) - x(u) + x(v)) && \forall u \in V_R. \end{aligned}$$

Note that out of the inequalities of (7d) we have included only those to which the corresponding inequalities in (7b)-(7c) are tight at y^* . Since $x^* \in X_\alpha(y^*)$, this set is a nonempty, closed convex set.

► **Lemma 17.** *For all $x \in X_\alpha(y^*)$ there exists a finite $\Delta(x) \geq 0$ such that $(x + \Delta y^*, y^*)$ is feasible in $LP(\alpha + \Delta)$ for all $\Delta \geq \Delta(x)$.*

Given a vector $x \in X_\alpha(y^*)$ let us call a vertex $u \in V_R$ *tight* if $u \in J_\alpha^R(x, y^*)$. Analogously, we call a vertex $u \in V_W$ *tight* if $0 = \alpha r(u, v) - y^*(u) - x(u) + x(v)$ for some arc $(u, v) \in I^u(y^*)$. Let us finally denote by $T(x)$ the set of tight vertices. We will be done if we show that there is a potential vector $x \in X_\alpha(y^*)$ such that $T(x) = V$, and which can be found by linear programming.

Let us define the set of vertices which belong to all tight sets:

$$U = \bigcap_{x \in X_\alpha(y^*)} T(x).$$

► **Lemma 18.** *If $\alpha \geq \alpha_0$, then $U \neq \emptyset$.*

► **Lemma 19.** *For all vertices $w \in V$ we can test if $w \in U$, and if not, find $x_w \in X_\alpha(y^*)$ such that $w \notin T(x_w)$ in polynomial time.*

► **Corollary 20.** *For each $\alpha \geq 0$ we can find the set $U \subseteq V$, and a vector $\bar{x} \in X_\alpha(y^*)$ such that $U = T(\bar{x})$ in polynomial time.*

► **Lemma 21.** *For all $x \in X_\alpha(y^*)$ and for all $v \notin T(x)$ there exists a small $\epsilon > 0$ such that for the vector*

$$x'(u) = \begin{cases} x(u) & \text{if } u \neq v, \\ x(u) - \epsilon & \text{if } u = v \end{cases}$$

we have $x' \in X_\alpha(y^)$.*

We shall prove next, with the above lemma in mind, that there exists a vector in $X_\alpha(y^*)$, if $\alpha \geq \alpha_0$, at which all vertices are tight. To this end let us consider the set U and the vector \bar{x} , as in Corollary 20, and the following linear programming problem:

$$\max \sum_{u \in V} z(u) \quad \text{s.t.} \quad (\bar{x} - z) \in X_\alpha(y^*), \quad z \geq 0, \quad z(u) = 0 \quad \forall u \in U. \quad (\text{LPZ})$$

Let us note that in this linear program α , r , y^* , and \bar{x} are all constants, just like the $z(u) = 0$ values for $u \in U$, and hence $z(v)$ for $v \in V \setminus U$ are the only variables.

► **Lemma 22.** *If $\alpha \geq \alpha_0$ then problem (LPZ) has a finite optimum.*

► **Corollary 23.** *If $\alpha \geq \alpha_0$, and \bar{z} is an optimum solution of (LPZ), then $T(\bar{x} - \bar{z}) = V$.*

Proof of Theorem 9. For an $\alpha' \geq \alpha_0$, let y^* be optimal in $LP(\alpha')$, let \bar{x} be as in Corollary 20 and \bar{z} as in Corollary 23, and define $x^* = \bar{x} - \bar{z} + \Delta(\bar{x} - \bar{z})y^*$ and $\alpha = \alpha' + \Delta(\bar{x} - \bar{z})$. Then, by Lemma 17 and Corollary 23 it follows that (x^*, y^*) is an optimal solution in $LP(\alpha)$, satisfying all conditions of the theorem. ◀

4 General MDPs

In this section we extend the result of the previous section to the more general case when $\phi_\Gamma(u) \neq 0$ for some $u \in V$.

► **Lemma 24.** *Let u denote a vertex with $\phi_\Gamma(u) \leq 0$, and let \mathfrak{s} denote a strategy in \mathfrak{S} . If, starting from initial vertex $v_0 = u$ strategy \mathfrak{s} uses with positive probability an arc (v, w) such that $v \in V_W$ and $0 \geq \phi_\Gamma(v) > \phi_\Gamma(w)$, then we have $\psi_\mathfrak{s}(v_0) = -\infty$.*

Let us introduce a new MDP $\Gamma' = (G' = (V, E'), p, r')$ obtained from $\Gamma = (G, p, r)$ as follows:

1. Delete all the arcs (u, v) from G such that $u \in V_W$ and $\phi_\Gamma(u) > \phi_\Gamma(v)$
2. Define $r'(u, v) = r(u, v) - \phi_\Gamma(u)$ for all the remaining arcs (u, v) .

Let us denote by E' the set of arcs of G' , and by $E'(u)$ the set of arcs in G' leaving vertex $u \in V$. Clearly, $E'(u) = E(u)$ for $u \in V_R$.

Let us note that we have $\phi_\Gamma(u) = \phi_\Gamma(v)$ for all $(u, v) \in E'(u)$, $u \in V_W$, since MAX could not have an arc $(u, v) \in E(u)$, $u \in V_W$ such that $\phi_\Gamma(u) < \phi_\Gamma(v)$, and all arcs going down in value are removed in Γ' . Let us also note that $\phi_{\Gamma'}(u) = 0$ for all vertices u .

It is easy to see that an optimal strategy with respect to the mean payoff function ϕ in Γ' is also optimal in Γ . We shall prove below in two lemmas that the same essentially holds in positional strategies with respect to the total payoff function ψ .

► **Lemma 25.** *Fix an initial vertex $v_0 = u$ such that $\phi_\Gamma(u) = 0$. Then any strategy \mathfrak{s} in Γ' satisfies $\mathbb{E}_\mathfrak{s}[r'(v_j, v_{j+1})] = \mathbb{E}_\mathfrak{s}[r(v_j, v_{j+1})]$.*

Since $\phi_{\Gamma'}(u) = 0$ for all $u \in V$, Theorems 8 and 9 imply that Γ' possesses a uniformly optimal positional strategy \mathfrak{s}^* with respect to the total payoff function ψ .

► **Lemma 26.** *\mathfrak{s}^* is also optimal in Γ .*

Proof. Let u be an initial vertex. By Lemmas 24 and 25, \mathfrak{s}^* is optimal in Γ , if u satisfies $\phi_\Gamma(u) = 0$. On the other hand, if $\phi_\Gamma(u) > 0$ (resp., < 0), let us note that $\phi_\Gamma(v) = \phi_\Gamma(w)$ if $v \in V_W$ and $(v, w) \in E'(v)$, and $\phi_\Gamma(v) = \text{avg}_{(v,w) \in E'(v)} \phi_\Gamma(w)$ if $v \in V_R$. This implies that $\mathbb{E}_\mathfrak{s}[\phi_\Gamma(v_t)] = \phi_\Gamma(v_0)$ for all t . Since by our construction we have, for any strategy $\mathfrak{s} \in \mathfrak{S}$,

$$\mathbb{E}_\mathfrak{s}[r(v_t, v_{t+1})] = \mathbb{E}_\mathfrak{s}[r'(v_t, v_{t+1})] + \mathbb{E}_\mathfrak{s}[\phi_\Gamma(v_t)] = \mathbb{E}_\mathfrak{s}[r'(v_t, v_{t+1})] + \phi_\Gamma(v_0) \quad (8)$$

and $\psi_\mathfrak{s}(v)$ is finite for all $v \in V$, the equality $\psi_\Gamma(v_0) = +\infty$ (resp., $-\infty$) follows. Indeed, if $\phi_{\Gamma'}(v_0) > 0$, then (8) implies for $\mathfrak{s} = \mathfrak{s}^*$ that $\psi_{\mathfrak{s}^*}(v_0) = +\infty$; on the other hand, if $\phi_{\Gamma'}(v_0) < 0$, then (8) together with Lemma 24 imply for any $\mathfrak{s} \in \mathfrak{S}$ that $\psi_\mathfrak{s}(v_0) = -\infty$. ◀

5 Two-player zero-sum games with perfect information (BWR-games)

We now turn our attention to two-person zero-sum stochastic games with perfect information and total effective payoff.

5.1 Discounted BWR-games

Let β be a number in $(0, 1]$ called the *discount factor*. *Discounted mean payoff* stochastic games were introduced by Shapley [26] and have payoff function:

$$\phi_{\mathfrak{s}}^{\beta}(v_0) = (1 - \beta) \sum_{j=0}^{\infty} \beta^j \mathbb{E}_{\mathfrak{s}}[r_{\mathfrak{s}}(v_t, v_{t+1})], \quad (9)$$

where $a(\mathfrak{s}) = \langle \mathbb{E}_{\mathfrak{s}}[r_{\mathfrak{s}}(v_0, v_1)], \mathbb{E}_{\mathfrak{s}}[r_{\mathfrak{s}}(v_1, v_2)], \dots \rangle$ is the sequence of expected rewards incurred at steps $0, 1, \dots$ of the play, according to the pair of strategies $\mathfrak{s} = (\mathfrak{s}_B, \mathfrak{s}_W)$.

Discounted games, in general, are easier to solve, due to the fact that a standard value iteration is in fact a fast converging contraction. Hence, they are widely used in the literature of stochastic games together with the above limit equality. In fact, for mean payoff BW-games with n vertices and *integral* rewards of maximum absolute value R it is known [32] that for two pairs of stationary strategies $\mathfrak{s}, \mathfrak{s}' \in \widehat{\mathfrak{G}}$ we have $\phi_{\mathfrak{s}}^{\beta}(u) < \phi_{\mathfrak{s}'}^{\beta}(u)$ if and only if $\phi_{\mathfrak{s}}(u) < \phi_{\mathfrak{s}'}(u)$ whenever $1 - \beta \leq \frac{1}{4n^3R}$.

If the discount factor β is strictly less than 1, we obtain the following result, which follows essentially from [26].

► **Fact 2** ([26]). A BWR-game with the discounted mean payoff function ϕ^{β} has a saddle point in uniformly optimal positional strategies, for all $0 < \beta < 1$.

We show in the next subsection that the same pair of stationary strategies form a uniform Nash equilibrium with respect to the total payoff ψ , if β is sufficiently close enough to 1.

5.2 Existence of a saddle point in positional strategies

When the mean payoff values are zero, there is an explicit formula for computing the total reward values, corresponding to a stationary strategy, as a function of the limiting probability matrix. To write this formula, we need first to introduce some notation. Given a BWR-game $\Gamma = (G, p, r)$ and a pair of positional strategies $\mathfrak{s} = (\mathfrak{s}_B, \mathfrak{s}_W)$, we obtain a weighted Markov chain $\Gamma_{\mathfrak{s}} = (P_{\mathfrak{s}}, r)$ with transition matrix $P_{\mathfrak{s}}$ in the obvious way:

$$p_{\mathfrak{s}}(u, v) = \begin{cases} 1 & \text{if } u \in V_W \cup V_B \text{ and } (u, v) \text{ is chosen by } \mathfrak{s}; \\ 0 & \text{if } u \in V_W \cup V_B \text{ and } (u, v) \text{ is not chosen by } \mathfrak{s}; \\ p(v, u) & \text{if } v \in V_R. \end{cases}$$

We define the *expected local reward* $r_{\mathfrak{s}} : V \rightarrow \mathbb{R}$, corresponding to the pair \mathfrak{s} as

$$r_{\mathfrak{s}}(u) = \begin{cases} r(u, v) & \text{if } u \in V_W \cup V_B \text{ and } (u, v) \text{ is chosen by } \mathfrak{s}; \\ \sum_{(u, v) \in E(u)} p(u, v) r(u, v) & \text{if } v \in V_R. \end{cases}$$

Finally, we will denote by $Q_{\mathfrak{s}}$ the (unique) limiting average probability matrix satisfying $Q_{\mathfrak{s}}P_{\mathfrak{s}} = P_{\mathfrak{s}}Q_{\mathfrak{s}} = Q_{\mathfrak{s}}$. Note that $\phi_{\mathfrak{s}} = Q_{\mathfrak{s}}r_{\mathfrak{s}}$ and $\phi_{\mathfrak{s}}^{\beta} = (1 - \beta)(I - \beta P_{\mathfrak{s}})^{-1}r_{\mathfrak{s}}$.

► **Proposition 1** ([28]). If \mathfrak{s} is stationary strategy such that $\phi_{\mathfrak{s}} = \mathbf{0}$, then $\psi_{\mathfrak{s}} = (I - P_{\mathfrak{s}} + Q_{\mathfrak{s}})^{-1}r_{\mathfrak{s}}$, where I is the $|V| \times |V|$ identity matrix.

To prove our main result for BWR-games (Theorem 2), it will be enough to consider games in which $\phi_{\Gamma}(u) = 0$ for all $u \in V$.

► **Theorem 27**. Consider an undiscounted BWR-game Γ such that $\phi_{\Gamma} = \mathbf{0}$. Then there is a uniformly optimal pair of positional strategies $(\mathfrak{s}_B, \mathfrak{s}_W)$ satisfying:

$$\pi_{\mathfrak{s}_B^*, \mathfrak{s}_W}(v_0) \leq \pi_{\mathfrak{s}_B^*, \mathfrak{s}_W^*}(v_0) \leq \pi_{\mathfrak{s}_B, \mathfrak{s}_W^*}(v_0) \quad \text{for all } \mathfrak{s}_B \in \widehat{\mathfrak{G}}_B, \mathfrak{s}_W \in \widehat{\mathfrak{G}}_W \text{ and for all } v_0 \in V.$$

If $|V| = n$, all rewards are integral with maximum absolute value R , and all transition probabilities are rational with maximum common denominator $D > 0$, then such a saddle point can be found by solving a discounted game with $\beta = 1 - \frac{1}{(nD)^{O(n^2)}R}$.

Proof. We start with the following claim.

► **Claim 1.** Let $\mathfrak{s} = (\mathfrak{s}_B, \mathfrak{s}_W)$ be a pair of positional strategies such that $\phi_{\mathfrak{s}}(v) = 0$ for all $v \in V$. Then, we have

$$\lim_{\beta \rightarrow 1^-} \frac{\psi_{\mathfrak{s}} - (I - \beta P_{\mathfrak{s}})^{-1} r_{\mathfrak{s}}}{1 - \beta} = P_{\mathfrak{s}}(I - P_{\mathfrak{s}} + Q_{\mathfrak{s}})^{-2} r_{\mathfrak{s}}.$$

$$\text{Let } \gamma = \min_{u \in V} \min_{\substack{\mathfrak{s}, \mathfrak{s}' \in \widehat{\mathfrak{S}} \\ \psi_{\mathfrak{s}}(u) \neq \psi_{\mathfrak{s}'}(u)}} |\psi_{\mathfrak{s}}(u) - \psi_{\mathfrak{s}'}(u)| \quad \text{and} \quad \kappa = \max_{u \in V} \max_{\mathfrak{s} \in \widehat{\mathfrak{S}}} |P_{\mathfrak{s}}(I - P_{\mathfrak{s}} + Q_{\mathfrak{s}})^{-2} r_{\mathfrak{s}}|.$$

Standard estimation arguments (see, e.g., [5]) give $\gamma \geq \frac{1}{(nD)^{O(n^2)}}$ and $\kappa \leq (nD)^{O(n^2)}R$.

Claim 1 implies that, for any sufficiently small $\epsilon > 0$, there exists a $\beta(\epsilon) \in (0, 1)$ such that, for all pairs of positional strategies $\mathfrak{s} \in \widehat{\mathfrak{S}}$, we have

$$\|(1 - \beta(\epsilon))\psi_{\mathfrak{s}} - \phi_{\mathfrak{s}}^{\beta(\epsilon)}\|_{\infty} < (1 - \beta(\epsilon))^2(\epsilon + \kappa) \leq 2(1 - \beta(\epsilon))^2\kappa. \quad (10)$$

Let us choose ϵ such that $\beta(\epsilon) > 1 - \frac{\gamma}{4\kappa}$. Then for any two pairs of positional strategies $\mathfrak{s}, \mathfrak{s}' \in \widehat{\mathfrak{S}}$, such that $\psi_{\mathfrak{s}}(u) > \psi_{\mathfrak{s}'}(u)$, we have $\psi_{\mathfrak{s}}(u) - \psi_{\mathfrak{s}'}(u) \geq \gamma$. On the other hand, by (10), we get

$$\left| (1 - \beta(\epsilon))\psi_{\mathfrak{s}}(u) - \phi_{\mathfrak{s}}^{\beta(\epsilon)}(u) \right| < 2(1 - \beta(\epsilon))^2\kappa \quad \text{and} \quad \left| (1 - \beta(\epsilon))\psi_{\mathfrak{s}'}(u) - \phi_{\mathfrak{s}'}^{\beta(\epsilon)}(u) \right| < 2(1 - \beta(\epsilon))^2\kappa.$$

Consequently, by our choice of ϵ , $\phi_{\mathfrak{s}}^{\beta(\epsilon)} > \phi_{\mathfrak{s}'}^{\beta(\epsilon)}$ follows, proving the claim of the theorem. ◀

Proof of Theorem 2. First assume that $\phi_{\Gamma}(u) = 0$ for all $u \in V$. Then Theorem 27 implies the existence of saddle point $\mathfrak{s}^* = (\mathfrak{s}_B^*, \mathfrak{s}_W^*)$, among uniformly optimal positional strategies. Since, by Theorem 1, the best response in the MDP obtained by fixing MAX's strategy to \mathfrak{s}_W^* (resp., MIN's strategy to \mathfrak{s}_B^*) is positional, it follows that \mathfrak{s}^* is a saddle point among all strategies of the two players. The case when $\phi_{\Gamma}(u) \neq 0$ for some $u \in V$ is handled using the same approach used in Section 4. ◀

References

- 1 D. P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987.
- 2 D. P. Bertsekas and J. N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991.
- 3 D. P. Bertsekas and H. Yuz. Stochastic shortest path problems, under weak conditions, lids report 2909. Technical report, MIT, 2013.
- 4 D. Blackwell. Discrete dynamic programming. *Ann. Math. Statist.*, 33:719–726, 1962.
- 5 E. Boros, K. Elbassioni, V. Gurvich, and K. Makino. A pumping algorithm for ergodic stochastic mean payoff games with perfect information. In *Proc. 14th IPCO*, volume 6080 of *LNCS*, pages 341–354. Springer, 2010.
- 6 E. Boros, K. Elbassioni, V. Gurvich, and K. Makino. On canonical forms for zero-sum stochastic mean payoff games. *Dynamic Games and Applications*, 3(2):128–161, 2013.
- 7 C. Derman. *Finite State Markov decision processes*. Academic Press, New York and London, 1970.

- 8 J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, Berlin, 1996.
- 9 O. Friedmann, T. D. Hansen, and U. Zwick. Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In *STOC*, pages 283–292, 2011.
- 10 D.R. Fulkerson and G.C. Harding. Maximizing the minimum source-sink path subject to a budget constraint. *Mathematical Programming*, 13:116–118, 1977.
- 11 T. Gallai. Maximum-minimum Sätze über Graphen. *Acta Mathematica Academiae Scientiarum Hungaricae*, 9:395–434, 1958.
- 12 D. Gillette. Stochastic games with zero stop probabilities. In M. Dresher, A. W. Tucker, and P. Wolfe, editors, *Contribution to the Theory of Games III*, volume 39 of *Annals of Mathematics Studies*, pages 179–187. Princeton University Press, 1957.
- 13 V.A. Gurvich, A.V. Karzanov, and L.G. Khachiyan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *USSR Comput. Math. Math. Phys.*, 28:85–91, 1988.
- 14 O. O. Hernández-Lerma and J.-B. Lasserre. *Further topics on discrete-time Markov control processes*. Applications of mathematics. Springer, New York, 1999.
- 15 A. J. Hoffman and R. M. Karp. On non-terminating stochastic games. *Management Science*, 12:359–370, 1966.
- 16 R. A. Howard. *Dynamic programming and Markov processes*. Technology press and Willey, New York, 1960.
- 17 E. Israeli and R. K. Wood. Shortest-path network interdiction. *Networks*, 40(2):97–111, 2002.
- 18 R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Math.*, 23:309–311, 1978.
- 19 A. V. Karzanov and V. N. Lebedev. Cyclical games with prohibition. *Mathematical Programming*, 60:277–293, 1993.
- 20 L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, V. Gurvich, G. Rudolf, and J. Zhao. On short paths interdiction problems: Total and node-wise limited interdiction. *Theory Comput. Syst.*, 43(2):204–233, 2008.
- 21 L. Khachiyan, V. Gurvich, and J. Zhao. Extending dijkstra’s algorithm to maximize the shortest path by node-wise limited arc interdiction. In *CSR*, pages 221–234, 2006.
- 22 T. M. Liggett and S. A. Lippman. Stochastic games with perfect information and time-average payoff. *SIAM Review*, 4:604–607, 1969.
- 23 H. Mine and S. Osaki. *Markovian decision process*. American Elsevier Publishing Co., New York, 1970.
- 24 R. H. Möhring, M. Skutella, and F. Stork. Scheduling with and/or precedence constraints. *SIAM J. Comput.*, 33(2):393–415, 2004.
- 25 S. D. Patek and D. P. Bertsekas. Stochastic shortest path games. *SIAM Journal on Control and Optimization*, 37:804–824, 1997.
- 26 L. Shapley. Stochastic games. *Proc. Nat. Acad. Sci. USA*, 39:1095–1100, 1953.
- 27 F. Thuijsman and O. J. Vrieze. The bad match, a total reward stochastic game. *Operations Research Spektrum*, 9:93–99, 1987.
- 28 F. Thuijsman and O. J. Vrieze. Total reward stochastic games and sensitive average reward strategies. *Journal of Optimization Theory and Applications*, 98:175–196, 1998.
- 29 P. Whittle. *Optimization over Time*. John Wiley & Sons, Inc., New York, NY, USA, 1982.
- 30 H. Yu and D. P. Bertsekas. Q-learning and policy iteration algorithms for stochastic shortest path problems. *Annals OR*, 208(1):95–132, 2013.
- 31 H. Yuz. Stochastic shortest path games and q-learning, lids report 2875. Technical report, MIT, 2011.
- 32 U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1-2):343 – 359, 1996.

Advice Complexity for a Class of Online Problems*

Joan Boyar, Lene M. Favrholt, Christian Kudahl, and Jesper W. Mikkelsen

University of Southern Denmark

{joan,lenem,kudahl,jesperwm}@imada.sdu.dk

Abstract

The advice complexity of an online problem is a measure of how much knowledge of the future an online algorithm needs in order to achieve a certain competitive ratio. We determine the advice complexity of a number of hard online problems including independent set, vertex cover, dominating set and several others. These problems are hard, since a single wrong answer by the online algorithm can have devastating consequences. For each of these problems, we show that $\log\left(1 + \frac{(c-1)^{c-1}}{c^c}\right)n = \Theta(n/c)$ bits of advice are necessary and sufficient (up to an additive term of $O(\log n)$) to achieve a competitive ratio of c . This is done by introducing a new string guessing problem related to those of Emek et al. (TCS 2011) and Böckenhauer et al. (TCS 2014). It turns out that this gives a powerful but easy-to-use method for providing both upper and lower bounds on the advice complexity of an entire class of online problems.

Previous results of Halldórsson et al. (TCS 2002) on online independent set, in a related model, imply that the advice complexity of the problem is $\Theta(n/c)$. Our results improve on this by providing an exact formula for the higher-order term. Böckenhauer et al. (ISAAC 2009) gave a lower bound of $\Omega(n/c)$ and an upper bound of $O((n \log c)/c)$ on the advice complexity of online disjoint path allocation. We improve on the upper bound by a factor of $\log c$. For the remaining problems, no bounds on their advice complexity were previously known.

1998 ACM Subject Classification F.1.2 Models of Computation (online computation)

Keywords and phrases online algorithms, advice complexity, asymmetric string guessing, advice complexity class AOC, covering designs

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.116

1 Introduction

An online problem is an optimization problem in which the input is divided into small pieces, usually called requests, arriving sequentially. Throughout the paper, we let n denote the number of requests. An online algorithm must serve each request without any knowledge of future requests, and the decisions made by the online algorithm are irrevocable. The goal is to minimize or maximize some objective function. Traditionally, competitive analysis is used to measure the quality of an online algorithm: The solution produced by the algorithm is compared to the solution produced by an optimal offline algorithm, OPT , which knows the entire request sequence in advance. While competitive analysis has been very successful and led to the design of many interesting online algorithms, it sometimes gives overly pessimistic results. Comparing an online algorithm, which knows nothing about the future, to an optimal offline algorithm, which knows the entire input, can be rather crude.

* This work was partially supported by the Villum Foundation and the Danish Council for Independent Research, Natural Sciences. Most proofs have been omitted due to space restrictions. These can be found in the full version of the paper [9].



© Joan Boyar, Lene M. Favrholt,
Christian Kudahl, and Jesper W. Mikkelsen;
licensed under Creative Commons License CC-BY

32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).

Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 116–129



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



As an example, consider the classical problem of finding a maximum independent set in a graph. Suppose that, at some point, an online algorithm decides to include a vertex v in its solution. It then turns out that all forthcoming vertices in the graph are connected to v and no other vertices. Thus, the online algorithm cannot include any of these vertices. On the other hand, **OPT** knows the entire graph, and so it rejects v and instead takes all forthcoming vertices. In fact, one can easily show that no online algorithm, even if we allow randomization, can obtain a competitive ratio better than $\Omega(n)$ for this problem.

This paper studies the maximum independent set problem and other similarly hard online problems. Many papers have studied special cases or relaxed versions of such problems, see e.g., [12, 20–22, 25, 33]. Often it is the online constraint that is relaxed, resulting in various semi-online models. The notion of advice complexity offers a quantitative and standardized, i.e., problem independent, way of relaxing the online constraint.

Advice complexity. The main idea of advice complexity is to provide an online algorithm, **ALG**, with some advice bits. These bits are provided by a trusted oracle, **O**, which has unlimited computational power and knows the entire request sequence. In the first model proposed [15], the advice bits were given as answers (of varying lengths) to questions posed by **ALG**. One difficulty with this model is that using at most 1 bit, three different options can be encoded (giving no bits, a 0, or a 1). This problem was addressed by the model proposed in [16], where the oracle is required to send a fixed number of advice bits per request. However, for the problems we consider, one bit per request is enough to guarantee an optimal solution, and so this model is not applicable. Instead, we will use the “advice-on-tape” model [7], which allows for a sublinear number of advice bits while avoiding the problem of encoding information in the length of each answer. Before the first request arrives, the oracle prepares an *advice tape*, an infinite binary string. The algorithm **ALG** may, at any point, read some bits from the advice tape. The advice complexity of **ALG** is the maximum number of bits read by **ALG** for any input sequence of at most a given length. When advice complexity is combined with competitive analysis, the central question is: How many bits of advice are necessary and sufficient to achieve a given competitive ratio c ?

► **Definition 1** (Advice complexity [7, 24] and competitive ratio [26, 32]). The input to an online problem, **P**, is a request sequence $\sigma = (r_1, \dots, r_n)$. An *online algorithm with advice*, **ALG**, computes the output $y = (y_1, \dots, y_n)$, under the constraint that y_i is computed from φ, r_1, \dots, r_i , where φ is the content of the advice tape. The *advice complexity*, $b(n)$, of **ALG** is the largest number of bits of φ read by **ALG** over all possible inputs of length at most n .

Each possible output for **P** is associated with a *score*. For a request sequence σ , $\text{ALG}(\sigma)$ ($\text{OPT}(\sigma)$) denotes the score of the output computed by **ALG** (**OPT**) when serving σ . If **P** is a maximization problem, then **ALG** is $c(n)$ -*competitive* if there exists a constant α such that $\text{OPT}(\sigma) \leq c(n) \cdot \text{ALG}(\sigma) + \alpha$ for all request sequences σ of length at most n . If **P** is a minimization problem, then **ALG** is $c(n)$ -*competitive* if there exists a constant α such that $\text{ALG}(\sigma) \leq c(n) \cdot \text{OPT}(\sigma) + \alpha$ for all request sequences σ of length at most n . In both cases, if the inequality holds with $\alpha = 0$, we say that **ALG** is *strictly* $c(n)$ -*competitive*. For $c \geq 1$, the *advice complexity*, $f(n, c)$, of a problem **P** is the smallest possible advice complexity of a c -competitive online algorithm for **P**.

We only consider deterministic online algorithms (with advice). Note that both the advice read and the competitive ratio may depend on n , but, for ease of notation, we often write b and c instead of $b(n)$ and $c(n)$. Also, by this definition, $c \geq 1$, for both minimization and maximization problems. Lower and upper bounds on the advice complexity have been obtained for many problems, see e.g., [2, 4–8, 10, 11, 14–16, 18, 19, 24, 27, 28, 30, 31].

Online string guessing. In [5, 16], the advice complexity of the following string guessing problem, SG, is studied: For each request, which is simply empty and contains no information, the algorithm tries to guess a single bit (or more generally, a character from some finite alphabet). The correct answer is either revealed as soon as the algorithm has made its guess (known history), or all of the correct answers are revealed together at the very end of the request sequence (unknown history). The goal is to guess correctly as many bits as possible. This problem was first introduced (under the name generalized matching pennies) in [16], where a lower bound for randomized algorithms with advice was given. In [5], the lower bound was improved for the case of deterministic algorithms. In fact, the lower bound given in [5] is tight up to lower-order terms. While SG is rather uninteresting in the view of traditional competitive analysis, it is very useful in an advice complexity setting. Indeed, it has been shown that the string guessing problem can be reduced to many classical online problems, thereby giving lower bounds on the advice complexity for these problems. This includes bin packing [11], the k -server problem [19], list update [10], metrical task system [16], set cover [5] and a certain version of maximum clique [5].

Our contribution. In this paper, we introduce a new *asymmetric string guessing* problem, ASG, formally defined in Section 2. The rules are similar to those of the original string guessing problem with an alphabet of size two, but the score function is asymmetric: If the algorithm answers 1 and the correct answer is 0, then this counts as a single wrong answer (as in the original problem). On the other hand, if the algorithm answers 0 and the correct answer is 1, the solution is infeasible and has an infinite penalty. This asymmetry in the score function forces the algorithm to be very cautious when making its guesses. It turns out that ASG captures, in a very precise way, the hardness of problems such as online independent set and online vertex cover.

We give lower and upper bounds on the advice complexity of the new asymmetric string guessing problem, ASG. The bounds are tight up to an additive term of $O(\log n)$. More precisely, if b is the number of advice bits necessary and sufficient to achieve a (strict)¹ competitive ratio of $c > 1$, then we show that²

$$\frac{1}{e \ln 2} \frac{n}{c} - \Theta(\log n) \leq b = \log \left(1 + \frac{(c-1)^{c-1}}{c^c} \right) n \pm \Theta(\log n) \leq \frac{n}{c} + \Theta(\log n).$$

This holds for all variants of the asymmetric string guessing problem (minimization/-maximization and known/unknown history). See Figure 1 on page 122 for a graphical plot.

We introduce a class, AOC, of online problems. The class AOC essentially consists of those problems which can be reduced to ASG. In particular, for any problem in AOC, our upper bound on the advice complexity for ASG applies. This is one of the few known examples of a general technique for constructing online algorithms with advice, which works for an entire class of problems.

On the hardness side, we show that several online problems, including vertex cover, cycle finding, dominating set, independent set, set cover and disjoint path allocation (described in Section 5) are AOC-complete, that is, they have the same advice complexity as ASG. We

¹ The upper bound holds for being strictly c -competitive, while the lower bound also holds for being c -competitive. For the lower bound, the constant hidden in $\Theta(\log n)$ depends on the additive constant α of the c -competitive algorithm.

² We only consider $c > 1$ since in order to be strictly 1-competitive, an algorithm needs to correctly guess every single bit. It is easy to show that this requires n bits of advice (see e.g. [5]). By Remark 9, this also gives a lower bound for being 1-competitive.

prove this by providing reductions from ASG to each of these problems. The reductions preserve the competitive ratio and only increase the advice read by an additive term of $O(\log n)$. Thus, we obtain bounds on the advice complexity of each of these problems which are essentially tight.

As a key step in obtaining our results, we establish a connection between the advice complexity of ASG and the size of covering designs (a well-studied object from the field of combinatorial designs).

Comparison with previous results. The original string guessing problem, SG, can be viewed as a maximization problem, the goal being to correctly guess as many of the n bits as possible. Clearly, OPT always obtains a profit of n . With a single bit of advice, an algorithm can achieve a strict competitive ratio of 2: The advice bit simply indicates whether the algorithm should always guess 0 or always guess 1. This is in stark contrast to ASG, where linear advice is needed to achieve any constant competitive ratio. On the other hand, for both SG and ASG, achieving a constant competitive ratio $c < 2$ requires linear advice. However, the exact amount of advice required to achieve a particular competitive ratio $c < 2$ is larger for ASG than for SG. See Figure 1 for a graphical comparison.

The problems online independent set and online disjoint path allocation, which we show to be AOC-complete, have previously been studied in the context of advice complexity or similar models. In [7], the advice complexity of online disjoint path allocation is considered. It is shown that a strictly c -competitive algorithm must read at least $\frac{n+2}{2c} - 2$ bits of advice. On the other hand, the authors show that for any $c \geq 2$, there exists a strictly c -competitive online algorithm reading at most $\min \left\{ n \log \left(\frac{c}{(c-1)^{c/(c-1)^c}} \right), \frac{n \log n}{c} \right\} + 3 \log n + O(1)$ bits of advice. We remark that $(n \log c)/c < n \log (c/(c-1)^{(c-1)/c}) < 2(n \log c)/c$, for $c \geq 2$. Thus, this older upper bound is $\Theta((n \log c)/c)$ and at least a factor of $2 \log c$ away from the lower bound. We improve on these bounds, removing the gap between the upper and lower bound.

In [21], the online independent set problem is considered in a multi-solution model. In this model, an online algorithm is allowed to maintain multiple solutions. The algorithm knows (a priori) the number n of vertices in the input graph, and the model is parametrized by a function $r(n)$. Whenever a vertex v is revealed, the algorithm can include v in at most $r(n)$ different solutions (some of which might be new solutions with v as the first vertex). At the end, the algorithm outputs the solution containing the most vertices. The multi-solution model is closely related to the advice complexity model. Simple conversions allow one to translate both upper and lower bounds between the two models almost exactly, up to an additive term of $O(\log n)$. Doing so, the results of [21] can be summarized as follows: For any $c \geq 1$, there is a strictly c -competitive independent set algorithm reading at most $\lceil n/c \rceil + O(\log n)$ bits of advice. On the other hand, any strictly c -competitive algorithm for independent set must read at least $\frac{n}{2c} - \log n$ bits of advice. Our improvement for this problem consists of determining the exact coefficient of the higher-order term.

One important feature of the framework that we introduce in this paper is that obtaining tight bounds on the advice complexity for problems like online independent set and online disjoint path allocation becomes very easy. We remark that the reductions we use to show the hardness of these two problems reduce instances of ASG to instances of online independent set (resp. disjoint path allocation) that are identical to the hard instances used in [21] (resp. [7]). What enables us to improve the previous bounds, even though we use the same hard instances, is that we have a detailed analysis of the advice complexity of ASG at our disposal.

Related work. The advice complexity of online disjoint path allocation has also been studied as a function of the length of the path (as opposed to the number of requests), see [3, 7]. The advice complexity of online independent set on bipartite graphs and on sparse graphs has been determined in [14]. The advice complexity of an online set cover problem [1] has been studied in [27]. However, the version of online set cover that we consider is different and so our results and those of [27] are incomparable.

Preliminaries. Let \log denote the binary logarithm \log_2 and \ln the natural logarithm \log_e . By a *string* we always mean a bit string. For a string $x \in \{0, 1\}^n$, we denote by $|x|_1$ the Hamming weight of x (that is, the number of 1s in x) and we define $|x|_0 = n - |x|_1$. Also, we denote the i 'th bit of x by x_i , so that $x = x_1x_2 \dots x_n$. For $n \in \mathbb{N}$, define $[n] = \{1, 2, \dots, n\}$. For a subset $Y \subseteq [n]$, the *characteristic vector* of Y is the string $y = y_1, \dots, y_n \in \{0, 1\}^n$ such that, for all $i \in [n]$, $y_i = 1$ if and only if $i \in Y$. For $x, y \in \{0, 1\}^n$, we write $x \sqsubseteq y$ if $x_i = 1 \Rightarrow y_i = 1$ for all $1 \leq i \leq n$.

If the oracle needs to communicate some integer m to the algorithm, and if the algorithm does not know of any upper bound on m , the oracle needs to use a self-delimiting encoding. For instance, the oracle can write $\lceil \log m \rceil$ in unary (a string of 1's followed by a 0) before writing m itself in binary. In total, this encoding uses $O(\log m)$ bits. Slightly more efficient encodings exist, see e.g. [6].

2 Asymmetric String Guessing

In this section, we formally define the asymmetric string guessing problem. There are four variants of the problem, one for each combination of minimization/maximization and known/unknown history. Collectively, these four problems will be referred to as ASG. We have deliberately tried to mimic the definition of the string guessing problem SG from [5].

► **Definition 2.** The *minimum asymmetric string guessing problem with unknown history*, MINASGU, has input $(?, \dots, ?, x)$, where $x \in \{0, 1\}^n$, for some $n \in \mathbb{N}$. For $1 \leq i \leq n$, round i proceeds as follows:

1. The algorithm receives request $?_i$ which contains no information.
2. The algorithm answers y_i , where $y_i \in \{0, 1\}$.

The *output* $y = y_1 \dots y_n$ computed by the algorithm is *feasible*, if $x \sqsubseteq y$. Otherwise, y is *infeasible*. The *cost* of a feasible output is $|y|_1$, and the cost of an infeasible output is ∞ .

► **Definition 3.** The *minimum asymmetric string guessing problem with known history*, MINASGK, has input $x = (?, x_1, \dots, x_n)$, where $x \in \{0, 1\}^n$, for some $n \in \mathbb{N}$. For $1 \leq i \leq n$, round i proceeds as follows:

1. If $i > 1$, the algorithm learns the correct answer, x_{i-1} , to the request in the previous round.
2. The algorithm answers $y_i = f(x_1, \dots, x_{i-1}) \in \{0, 1\}$, where f is a function defined by the algorithm.

The *output* $y = y_1 \dots y_n$ computed by the algorithm is *feasible*, if $x \sqsubseteq y$. Otherwise, y is *infeasible*. The *cost* of a feasible output is $|y|_1$, and the cost of an infeasible output is ∞ .

We collectively refer to MINASGK and MINASGU as MINASG. The string x in either version of MINASG will be referred to as the *input string* or the *correct string*. Note that the number of requests in both versions of MINASG is $n + 1$, since there is a final request that does not require any response from the algorithm. This final request ensures that the

entire string x is eventually known. For simplicity, we will measure the advice complexity of MINASG as a function of n (this choice is not important as it changes the complexity by at most one bit).

Without advice the situation is the following. For any deterministic MINASG algorithm which sometimes answers 0, there exists an input string on which the algorithm gets a cost of ∞ . However, if an algorithm always answers 1, the input string could consist solely of 0s. Thus, no deterministic algorithm can achieve a finite competitive ratio. One can easily show that the same holds for any randomized algorithm.

We now give a simple algorithm for MINASG which reads $O(n/c)$ bits of advice and achieves a strict competitive ratio of $\lceil c \rceil$.

► **Theorem 4.** *For any $c \geq 1$, there is a strictly $\lceil c \rceil$ -competitive algorithm for MINASGU which reads $\lceil \frac{n}{c} \rceil + O(\log(n/c))$ bits of advice.*

Proof. Let $x = x_1 \dots x_n$ be the input string. The oracle encodes $p = \lceil n/c \rceil$ in a self-delimiting way, which requires $O(\log(n/c))$ bits of advice. For $0 \leq j < p$, define $C_j = \{x_i : i \equiv j \pmod{p}\}$. These p sets partition the input string, and the size of each C_j is at most $\lceil n/p \rceil \leq \lceil c \rceil$. The oracle writes one bit, b_j , for each set C_j . If C_j contains only 0s, b_j is set to 0. Otherwise, b_j is set to 1.

The algorithm learns p and the bits b_0, \dots, b_{p-1} from the advice tape. In round i , the algorithm answers with the bit $b_{i \bmod p}$. Clearly, this algorithm is strictly $\lceil c \rceil$ -competitive. ◀

The definition of the maximization version of ASG is similar to the definition of MINASG:

► **Definition 5.** The *maximum asymmetric string guessing problem with unknown history*, MAXASGU, is identical to the MINASGU problem with unknown history, except that the score function is different. In the MAXASGU problem, the score of a feasible output y is $|y|_0$. The score of an infeasible output is $-\infty$. The goal is to maximize the score. The *maximum asymmetric string guessing problem with known history*, MAXASGK, is defined similarly.

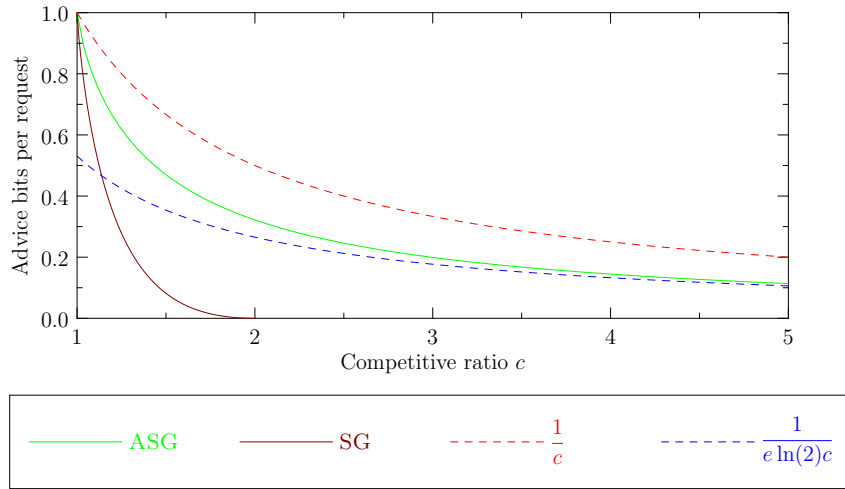
We collectively refer to the two maximization problems as MAXASG.

An algorithm for MAXASG without advice cannot attain a finite competitive ratio. If such an algorithm would ever answer 0 in some round, an adversary would let the correct answer be 1 and the algorithm's output would be infeasible. On the other hand, answering 1 in every round gives an output with a profit of zero.

► **Theorem 6.** *For any $c \geq 1$, there is a strictly $\lceil c \rceil$ -competitive algorithm for MAXASGU which reads $\lceil n/c \rceil + O(\log n)$ bits of advice.*

Proof. The oracle partitions the input string $x = x_1 \dots x_n$ into $\lceil c \rceil$ disjoint blocks, each containing (at most) $\lceil \frac{n}{c} \rceil$ consecutive bits. Note that there must exist a block where the number of 0s is at least $|x|_0 / \lceil c \rceil$. The oracle uses $O(\log n)$ bits to encode the index i at which this block starts and the index i' at which it ends. Furthermore, the oracle writes the string $x_i \dots x_{i'}$ onto the advice tape, which requires at most $\lceil \frac{n}{c} \rceil$ bits, since this is the largest possible size of a block. The algorithm learns the string $x_i \dots x_{i'}$ and answers accordingly in rounds i to i' . In all other rounds, the algorithm answers 1. ◀

In the following sections, we determine the amount of advice necessary and sufficient to achieve some (strict) competitive ratio $c > 1$. It turns out that the algorithms from Theorems 4 and 6 use the asymptotically smallest possible number of advice bits, but the coefficient in front of the term n/c can be improved.



■ **Figure 1** The upper solid line (green) shows the number of advice bits per request which are necessary and sufficient for obtaining a (strict) competitive ratio of c for ASG (ignoring lower-order terms). The lower solid line (brown) shows the same number for the original string guessing problem SG [5]. The dashed lines are the functions $1/c$ and $1/(e \ln(2)c)$.

3 Advice Complexity of ASG

In order to determine the advice complexity of ASG, we will use some basic results from the theory of combinatorial designs. Let $v \geq k \geq t$ be integers. A (v, k, t) -covering design is a family of k -subsets (called *blocks*) of a v -set, X , such that any t -subset of X is contained in at least one block. The *size* of a covering design, D , is the number of blocks in D . The *covering number*, $C(v, k, t)$, is the smallest possible size of a (v, k, t) -covering design. The connection to ASG is that for inputs to MINASG where the number of 1s is t , an (n, ct, t) -covering design can be used to obtain a strictly c -competitive algorithm. Many papers have been devoted to the study of covering numbers. See [13] for a survey. We make use of the following bounds on the size of a covering design:

► **Lemma 7** (Erdős, Spencer [17]). *For all natural numbers $v \geq k \geq t$,*

$$\frac{\binom{v}{t}}{\binom{k}{t}} \leq C(v, k, t) \leq \frac{\binom{v}{t}}{\binom{k}{t}} \left(1 + \ln \binom{k}{t} \right) \quad (1)$$

We will state the obtained advice complexity bounds in terms of the following function:

$$B(n, c) = \log \left(1 + \frac{(c-1)^{c-1}}{c^c} \right) n \quad (2)$$

For $c > 1$, we show that $B(n, c) \pm O(\log n)$ bits of advice are necessary and sufficient to achieve a (strict) competitive ratio of c , for any version of ASG. See Figure 1 for a graphical view. It can be shown that $n/(a \cdot c) \leq B(n, c) \leq n/c$, where $a = e \ln(2)$. In particular, if $c = o(n/\log n)$, we see that $O(\log n)$ becomes a lower-order term. Thus, for this range of c , we determine exactly the higher-order term in the advice complexity of ASG. Since this is the main focus of our paper, we will consider $O(\log n)$ a lower-order term. The case where $c = \Omega(n/\log n)$ is treated in the full version of the paper [9].

3.1 Advice Complexity of MINASG

The following theorems show that covering numbers are closely related to the amount of advice needed for an algorithm, ALG , to achieve a strict competitive ratio of c for MINASG.

► **Theorem 8.** *For any $c > 1$, there exists a strictly c -competitive algorithm for MINASGU and MINASGK reading b bits of advice, where*

$$b = \log \left(\max_{t: \lfloor ct \rfloor < n} C(n, \lfloor ct \rfloor, t) \right) + O(\log n) = B(n, c) + O(\log n).$$

Proof (sketch). Let $x = x_1 \dots x_n$ be an input string to MINASG and set $t = |x|_1$. The oracle encodes n and t in a self-delimiting way. If $t = 0$ or $ct \geq n$, it is trivial to output a solution of the desired quality. In all other cases, ALG computes an optimal $(n, \lfloor ct \rfloor, t)$ -covering design by making a systematic enumeration of all possible solutions (using lexicographic order, say). Note there must exist a $\lfloor ct \rfloor$ -block Y from this covering design such that the characteristic vector y of Y satisfies $x \sqsubseteq y$. Using $\lceil \log C(n, \lfloor ct \rfloor, t) \rceil$ bits of advice, the oracle can encode the index of Y . A lengthy calculation using (1) allows us to express this upper bound in terms of the function $B(n, c)$ defined in (2). ◀

Note that an algorithm for MINASGU reading $b(n)$ bits of advice can only produce $2^{b(n)}$ different outputs, one for each possible advice string. Consider the set of input strings $I_{n,t} = \{x \in \{0, 1\}^n : |x|_1 = t\}$, and let $Y_{n,t}$ be the corresponding set of output strings produced by the algorithm. If the algorithm is strictly c -competitive, then $Y_{n,t}$ can be converted into an $(n, \lfloor ct \rfloor, t)$ -covering design. This gives a lower bound of $\log (\max_{t: \lfloor ct \rfloor < n} C(n, \lfloor ct \rfloor, t))$ on the advice needed to achieve a strict competitive ratio of c for MINASGU.

Recall that for MINASGK, the output produced by an algorithm can depend on both the advice read and the correct answer to requests in previous rounds. In particular, the proof of the lower bound for MINASGU sketched above breaks down for MINASGK. However, by using a more complicated argument, Theorem 10 gives a lower bound of $B(n, c) - O(\log n)$ on the advice needed to achieve a strict competitive ratio of c for MINASGK. Note that this matches the upper bound from Theorem 8 up to an additive term of $O(\log n)$.

Before proving Theorem 10, we remark that there is a close connection between results on the competitive ratio and the strict competitive ratio:

► **Remark 9.** Suppose that a MINASG algorithm, ALG , is c -competitive. By definition, there exists a constant, α , such that $\text{ALG}(\sigma) \leq c \cdot \text{OPT}(\sigma) + \alpha$. Then, one can construct a new algorithm, ALG' , which is *strictly* c -competitive and uses $O(\log n)$ additional advice bits as follows: Use $O(\log n)$ bits of advice to encode the length n of the input and use $\alpha \cdot \lceil \log n \rceil = O(\log n)$ bits of advice to encode the index of (at most) α rounds in which ALG guesses 1 but where the correct answer is 0. Clearly, ALG' can use this additional advice to achieve a strict competitive ratio of c . This also means that a lower bound of b on the number of advice bits required to be *strictly* c -competitive implies a lower bound of $b - O(\log n)$ advice bits for being c -competitive (where the constant hidden in $O(\log n)$ depends on the additive constant α of the c -competitive algorithm). We will use this observation in Theorem 10. Note that the same technique can be used for MAXASG.

► **Theorem 10.** *For any $c > 1$, a c -competitive algorithm for MINASGK or MINASGU must read at least b bits of advice, where*

$$b \geq \log \left(\max_{t: \lfloor ct \rfloor < n} \frac{\binom{n}{t}}{\binom{\lfloor ct \rfloor}{t}} \right) - O(\log n) = B(n, c) - O(\log n) \tag{3}$$

Proof (sketch). Fix n, c and let b be the maximum number of advice bits read by the strictly c -competitive algorithm, **ALG**, over all inputs of length n . Suppose, by way of contradiction, that there exists some t such that $\lfloor ct \rfloor < n$ and $b < \log\left(\binom{n}{t} / \binom{\lfloor ct \rfloor}{t}\right)$. Let $I_{n,t}$ be the set of input strings of length n and Hamming weight t . We achieve the desired contradiction by showing that there is some input string in $I_{n,t}$ on which the algorithm incurs a cost of at least $\lfloor ct \rfloor + 1$.

Note that $|I_{n,t}| = \binom{n}{t}$. By the pigeonhole principle, there must exist some advice string, φ , such that the set of input strings, $I_{n,t}^\varphi \subseteq I_{n,t}$, of length n and Hamming weight t for which **ALG** reads the advice φ has size $|I_{n,t}^\varphi| > \binom{\lfloor ct \rfloor}{t}$.

We consider the computation of **ALG**, when reading the advice φ , as a game between **ALG** and an adversary, **ADV**. From the advice, **ALG** learns that the input string belongs to $I_{n,t}^\varphi$. Because of the known history, at the beginning of round i , **ALG** also knows the first $i - 1$ bits of the input string. We say that a string $s \in I_{n,t}^\varphi$ is *alive* in round i if the first $i - 1$ bits of s are identical to those revealed in the first $i - 1$ rounds. If, in round i , there exists some string s such that s is still alive and $s_i = 1$, then **ALG** must answer 1. If not, **ADV** could pick s as the input string and hence **ALG** would incur a cost of ∞ . On the other hand, if no such s exists in round i , we may assume (without loss of generality) that **ALG** answers 0. Intuitively, **ADV** wants to maximize the number of rounds where **ALG** is forced to answer 1 but where **ADV** can still give 0 as the correct answer.

Suppose that in some round, there are m strings from $I_{n,t}^\varphi$ which are alive. Let h be the number of 1s that have yet to be revealed in each of these strings (this is well-defined since all strings from $I_{n,t}^\varphi$ have the same number of 1s). We let $L_1(m, h)$ denote the minimum cost that the adversary can force **ALG** to incur in the remaining rounds when starting from this situation. The proof is finished by showing that for any $m, h \geq 1$,

$$L_1(m, h) \geq \min \left\{ d : m \leq \binom{d}{h} \right\}. \quad (4)$$

Indeed, it follows from (4) that $L_1(|I_{n,t}^\varphi|, t) \geq \lfloor ct \rfloor + 1$. By induction on m and h , one can show that (4) is true by using the following adversary strategy: Let m the number of strings alive in the current round, i . Furthermore, let m_0 be the number of the m alive strings for which the i th bit is 0, and let $m_1 = m - m_0$. If $m_0 = m$, then **ADV** has to choose 0 as the correct bit in round i . If $m_0 < m$, let d_1 be the smallest integer such that $m_1 \leq \binom{d_1}{h-1}$ and let d be the smallest integer such that $m \leq \binom{d}{h}$. If $d_1 \leq d + 1$, the adversary chooses 1 as the correct bit in round i and otherwise it chooses 0.

By Remark 9, the first inequality of (3) follows. The last equality follows from a simple but lengthy calculation showing that $\log\left(\max_{t: \lfloor ct \rfloor < n} \binom{n}{t} / \binom{\lfloor ct \rfloor}{t}\right) = B(n, c) - O(\log n)$. ◀

3.2 Advice Complexity of MAXASG

The advice complexity of MAXASG is the same as that of MINASG, up to an additive $O(\log n)$ term. This is not immediately obvious, but one can show that computing a c -competitive solution for MAXASG, on input strings where the number of 0s is u , requires roughly the same amount of advice as computing a c -competitive solution for MINASG, on input strings where the number of 1s is $\lceil u/c \rceil$.

► **Theorem 11.** For MAXASGU and MAXASGK and for any $c > 1$,

$$b = B(n, c) \pm O(\log n) \quad (5)$$

bits of advice are necessary and sufficient to achieve a (strict) competitive ratio of c .

4 The Complexity Class AOC

In this section, we define a class of problems, AOC, and show that for each problem, P , in AOC, the advice complexity of P is at most that of ASG.

► **Definition 12.** A problem P is in AOC (*Asymmetric Online Covering*) if it can be defined as follows: The input to an instance of P consists of a sequence of n requests $\sigma = (r_1, \dots, r_n)$ and possibly one final dummy request. An algorithm for P computes a binary output string $y = y_1 \dots y_n \in \{0, 1\}^n$, where $y_i = f(r_1, \dots, r_i)$ for some function f .

For minimization (maximization) problems, the score function s maps a pair (σ, y) of input and output to a cost (profit) in $\mathbb{N} \cup \{\infty\}$ ($\mathbb{N} \cup \{-\infty\}$). For an input σ and an output y , y is *feasible* if $s(\sigma, y) \in \mathbb{N}$. Otherwise, y is *infeasible*. There must exist at least one feasible output. Let $S_{\min}(\sigma)$ ($S_{\max}(\sigma)$) be the set of those outputs that minimize (maximize) s for a given input σ .

If P is a minimization problem, then for every input σ , the following must hold:

1. For a feasible output y , $s(\sigma, y) = |y|_1$.
2. An output y is feasible if there exists a $y' \in S_{\min}(\sigma)$ such that $y' \sqsubseteq y$.
If there is no such y' , the output may or may not be feasible.

If P is a maximization problem, then for every input σ , the following must hold:

1. For a feasible output y , $s(\sigma, y) = |y|_0$.
2. An output y is feasible if there exists a $y' \in S_{\max}(\sigma)$ such that $y' \sqsubseteq y$.
If there is no such y' , the output may or may not be feasible.

The dummy request is a request that does not require an answer and is not counted when we count the number of requests. Most of the problems that we consider will not have such a dummy request, but it is necessary to make sure that ASG belongs to AOC.

The input σ to a problem P in AOC can contain any kind of information. However, for each request, an algorithm for P only needs to make a binary decision. If the problem is a minimization problem, it is useful to think of answering 1 as accepting the request and answering 0 as rejecting the request (e.g., vertices in a vertex cover). The output is guaranteed to be feasible if the accepted requests are a superset of the requests accepted in some optimal solution. If the problem is a maximization problem, it is useful to think of answering 0 as accepting the request and answering 1 as rejecting the request (e.g., vertices in an independent set). The output is guaranteed to be feasible if the accepted requests are a subset of the requests accepted in an optimal solution.

The key point of Definition 12 is that an ASGU algorithm works for every problem in AOC. Thus, by Theorems 8 and 11, we get the following upper bound on the advice complexity for problems in AOC.

► **Theorem 13.** *Let P be a problem in AOC. There exists a strictly c -competitive online algorithm for P reading b bits of advice, where*

$$b = \log \left(1 + \frac{(c-1)^{c-1}}{c^c} \right) n + O(\log n) = B(n, c) + O(\log n).$$

For all variants of ASG, we know that this upper bound is tight up to an $O(\log n)$ term. This leads us to the following definition of completeness.

► **Definition 14.** A problem P is *AOC-complete* if P belongs to AOC and if, for all $c > 1$, any c -competitive algorithm for P must read at least b bits of advice, where

$$b = \log \left(1 + \frac{(c-1)^{c-1}}{c^c} \right) n - O(\log n) = B(n, c) - O(\log n).$$

The constant hidden in $O(\log n)$ in Definition 14 is allowed to depend on the additive constant α of the c -competitive algorithm.

Note that the advice complexity of an AOC-complete problem must be identical to the upper bound from Theorem 13, up to a lower-order term of $O(\log n)$. By Theorems 10 and 11, all of MINASGU, MINASGK, MAXASGU and MAXASGK are AOC-complete. When we show that some problem P is AOC-complete, we do this by giving a reduction from a known AOC-complete problem to P , which preserves the competitive ratio and increases the number of advice bits by at most $O(\log n)$. ASGK is especially well-suited as a starting point for such reductions. We allow for an additional $O(\log n)$ bits of advice in Definition 14 in order to be able to use the reduction between the strict and non-strict competitive ratios as explained in Remark 9 and in order to encode some natural parameters of the problem, such as the input length or the score of an optimal solution. For most values of c , it seems reasonable to allow these additional advice bits. However, it does mean that for $c = \Omega(n/\log n)$, the requirement in the definition of AOC-complete is vacuously true.

5 Applications

By definition, showing that a problem is AOC-complete gives (almost) tight bounds on its advice complexity. We show that several natural online problems are AOC-complete.

Many of the problems that we consider are graph problems. Unless otherwise mentioned, the problems are studied in the *vertex-arrival model*. In this model, the vertices of an unknown graph are revealed one by one. That is, in each round, a vertex is revealed together with all edges connecting it to previously revealed vertices. For the problems we study in the vertex-arrival model, whenever a vertex, v , is revealed, an online algorithm ALG must (irrevocably) decide if v should be included in its solution or not. The individual graph problems are defined by specifying the set of feasible solutions. For all of the problems, the cost (or profit) of a feasible solution is the number of vertices in that solution. The cost (profit) of an infeasible solution is ∞ ($-\infty$). The problems we consider in this model are:

- ONLINE VERTEX COVER. A solution is feasible if all edges in the input graph have an endpoint at some vertex in the solution. The problem is a minimization problem.
- ONLINE CYCLE FINDING. A solution is feasible if the subgraph induced by the vertices in the solution contains a cycle. We assume that the presented graph always contains a cycle. The problem is a minimization problem.
- ONLINE DOMINATING SET. A solution is feasible if each vertex in the input graph is in the solution or has a neighbor in the solution. The problem is a minimization problem.
- ONLINE INDEPENDENT SET. A solution is feasible if no two vertices in the solution are neighbors. The problem is a maximization problem.

We also consider the following online problems. Again, the cost (profit) of an infeasible solution is ∞ ($-\infty$).

- ONLINE DISJOINT PATH ALLOCATION. A path, P , is given. Each request is a subpath of P and must immediately be either accepted or rejected. A solution is feasible if the accepted subpaths are edge disjoint. The profit of a feasible solution is the number of accepted paths. The problem is a maximization problem.

- ONLINE SET COVER (set-arrival version). A finite set U known as the *universe* is given. The input is a sequence of finite subsets of U , (A_1, \dots, A_n) , where $\cup_{1 \leq i \leq n} A_i = U$. On arrival, a subset must either be accepted or rejected. Denote by S the set of indices of the subsets in some solution. The solution is feasible if $\cup_{i \in S} A_i = U$. The cost of a feasible solution is the number of accepted subsets. The problem is a minimization problem.

Note that the offline version of the problems we study have very different properties. Finding the shortest cycle in a graph can be done in polynomial time. There is a 2-approximation algorithm for finding a minimum vertex cover³. No $o(\log n)$ -approximation algorithm exists for finding a minimum set cover (or a minimum dominating set), unless $P = NP$ [29]. For any $\varepsilon > 0$, no $n^{1-\varepsilon}$ -approximation algorithm exists for finding a maximum independent set, unless $ZPP = NP$ [23].

The following theorem shows that all of the problems defined above are AOC-complete.

► **Theorem 15.** *For the problems ONLINE VERTEX COVER, ONLINE CYCLE FINDING, ONLINE DOMINATING SET, ONLINE SET COVER, ONLINE INDEPENDENT SET and ONLINE DISJOINT PATH ALLOCATION and for any $c > 1$, possibly a function of the input length n ,*

$$b = \log \left(1 + \frac{(c-1)^{c-1}}{c^c} \right) n \pm O(\log n)$$

bits of advice are necessary and sufficient to achieve a (strict) competitive ratio of c .

It is easy to check that each of these problems belongs to AOC. To show completeness, we use reductions from ASG. Here, we sketch the reduction from MINASG to ONLINE VERTEX COVER⁴. For an input string $x = x_1 \dots x_n \in \{0, 1\}^n$, define $G_x = (V, E)$ as follows: $V = \{v_1, \dots, v_n\}$ and $E = \{(v_i, v_j) : x_i = 1 \text{ and } i < j\}$. Furthermore, let $V_1 = \{v_i : x_i = 1\}$. The vertices will be revealed in the order (v_1, \dots, v_n) . Note that $V_1 \setminus \{v_n\}$ is a minimum vertex cover of G_x . Also, if an algorithm rejects just a single vertex from V_1 , it must accept all forthcoming vertices in order to produce a feasible solution. Using these observations, one can show that ONLINE VERTEX COVER is AOC-complete. The details, and the reductions for the remaining problems, can be found in the full version [9].

6 Conclusion and Open Problems

As with the original string guessing problem SG [5, 16], we have shown that ASG is a useful tool for determining the advice complexity of online problems. It seems plausible that one could identify other variants of online string guessing and obtain classes similar to AOC. This could lead to an entire hierarchy of string guessing problems and related classes.

More concretely, there are various possibilities of generalizing ASG. One could associate some positive weight to each bit x_i in the input string. The goal would then be to produce a feasible output of minimum (or maximum) weight. Such a string guessing problem would model minimum weight vertex cover (or maximum weight independent set). Note that for MAXASG, the algorithm from Theorem 6 works in the weighted version. However, the same is not true for any of the algorithms we have given for MINASG. Thus, it remains an open problem if $O(n/c)$ bits of advice suffice to achieve a competitive ratio of c for the weighted version of MINASG.

³ We emphasize that the 2-approximation algorithm which greedily covers the edges (by selecting both endpoints) one by one cannot be used in the online vertex-arrival model.

⁴ The graph used in this reduction is identical to the graph used in [21] to show hardness of ONLINE INDEPENDENT SET in the multi-solution model.

References

- 1 Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM J. Comput.*, 39(2):361–370, 2009.
- 2 Kfir Barhum. Tight bounds for the advice complexity of the online minimum steiner tree problem. In *SOFSEM*, pages 77–88, 2014.
- 3 Kfir Barhum, Hans-Joachim Böckenhauer, Michal Forišek, Heidi Gebauer, Juraj Hromkovič, Sacha Krug, Jasmin Smula, and Björn Steffen. On the power of advice and randomization for the disjoint path allocation problem. In *SOFSEM*, pages 89–101, 2014.
- 4 Maria Paola Bianchi, Hans-Joachim Böckenhauer, Juraj Hromkovič, and Lucia Keller. Online coloring of bipartite graphs with and without advice. *Algorithmica*, 70(1):92–111, 2014.
- 5 Hans-Joachim Böckenhauer, Juraj Hromkovič, Dennis Komm, Sacha Krug, Jasmin Smula, and Andreas Sprock. The string guessing problem as a method to prove lower bounds on the advice complexity. *Theor. Comput. Sci.*, 2014.
- 6 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Kráľovič, and Richard Kráľovič. On the advice complexity of the k-server problem. In *ICALP (1)*, pages 207–218, 2011.
- 7 Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Kráľovič, Richard Kráľovič, and Tobias Mömke. On the advice complexity of online problems. In *ISAAC*, pages 331–340, 2009.
- 8 Hans-Joachim Böckenhauer, Dennis Komm, Richard Kráľovič, and Peter Rossmanith. The online knapsack problem: Advice and randomization. *Theor. Comput. Sci.*, 527:61–72, 2014.
- 9 Joan Boyar, Lene M. Favrholdt, Christian Kudahl, and Jesper W. Mikkelsen. The Advice Complexity of a Class of Hard Online Problems. *arXiv*, 1408.7033 (cs.DS), August 2014.
- 10 Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. On the list update problem with advice. In *LATA*, pages 210–221, 2014.
- 11 Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. Online bin packing with advice. In *STACS*, pages 174–186, 2014. Full paper to appear in *Algorithmica*.
- 12 Marc Demange and Vangelis Th. Paschos. On-line vertex-covering. *Theor. Comput. Sci.*, 332(1-3):83–108, 2005.
- 13 Jeffrey H. Dinitz and Douglas R. Stinson, editors. *Contemporary Design Theory: a Collection of Surveys*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, New York, 1992.
- 14 Stefan Dobrev, Rastislav Kráľovič, and Richard Kráľovič. Independent set with advice: The impact of graph knowledge - (extended abstract). In *WAOA*, pages 2–15, 2012.
- 15 Stefan Dobrev, Rastislav Kráľovič, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO - Theor. Inf. Appl.*, 43(3):585–613, 2009.
- 16 Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theor. Comput. Sci.*, 412(24):2642–2656, 2011.
- 17 Paul Erdős and Joel Spencer. *Probabilistic Methods in Combinatorics*. Academic Press, 1974.
- 18 Michal Forišek, Lucia Keller, and Monika Steinová. Advice complexity of online coloring for paths. In *LATA*, pages 228–239, 2012.
- 19 Sushmita Gupta, Shahin Kamali, and Alejandro López-Ortiz. On advice complexity of the k-server problem under sparse metrics. In *SIROCCO*, pages 55–67, 2013.
- 20 Magnús M. Halldórsson. Online coloring known graphs. *Electronic J. of Combinatorics*, 7(R7), 2000.
- 21 Magnús M. Halldórsson, Kazuo Iwama, Shuichi Miyazaki, and Shiro Taketomi. Online independent sets. *Theor. Comput. Sci.*, 289(2):953–962, 2002.
- 22 Magnús M. Halldórsson and Hadas Shachnai. Return of the boss problem: Competing online against a non-adaptive adversary. In *FUN*, pages 237–248, 2010.

- 23 Johan Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Math.*, 182(1):105–142, 1999.
- 24 Juraj Hromkovič, Rastislav Kráľovič, and Richard Kráľovič. Information complexity of online problems. In *MFCS*, pages 24–36, 2010.
- 25 Sandy Irani. Coloring inductive graphs on-line. *Algorithmica*, 11(1):53–72, 1994.
- 26 Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.
- 27 Dennis Komm, Richard Kráľovič, and Tobias Mömke. On the advice complexity of the set cover problem. In *CSR*, pages 241–252, 2012.
- 28 Shuichi Miyazaki. On the advice complexity of online bipartite matching and online stable marriage. *Inf. Process. Lett.*, 114(12):714–717, 2014.
- 29 Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *STOC*, pages 475–484, 1997.
- 30 Marc P. Renault, Adi Rosén, and Rob van Stee. Online algorithms with advice for bin packing and scheduling problems. *CoRR*, abs/1311.7589, 2013.
- 31 Sebastian Seibert, Andreas Sprock, and Walter Unger. Advice complexity of the online coloring problem. In *CIAC*, pages 345–357, 2013.
- 32 Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.
- 33 Wen-Guey Tzeng. On-line dominating set problems for graphs. In D.-Z. Ding and Pardalos, editors, *Handbook of Combinatorial Optimization*. Kluwer Academic Publishers, Boston, 1998.

Las Vegas Computability and Algorithmic Randomness*

Vasco Brattka^{1,2}, Guido Gherardi², and Rupert Hölzl³

- 1 Department of Mathematics & Applied Mathematics,
University of Cape Town,
South Africa,
Vasco.Brattka@cca-net.de
- 2 Faculty of Computer Science,
Universität der Bundeswehr München,
Germany,
Guido.Gherardi@gmail.com
- 3 Department of Mathematics,
National University of Singapore,
Republic of Singapore,
r@hoelzl.fr

Abstract

In this article we try to formalize the question “What can be computed with access to randomness?” We propose the very fine-grained Weihrauch lattice as an approach to differentiate between different types of computation with access to randomness. In particular, we show that a natural concept of Las Vegas computability on infinite objects is more powerful than mere oracle access to a Martin-Löf random object. As a concrete problem that is Las Vegas computable but not computable with access to a Martin-Löf random oracle we study the problem of finding Nash equilibria.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Weihrauch degrees, weak König’s lemma, Las Vegas computability, algorithmic randomness, Nash equilibria

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.130

1 Introduction

Studying how access to sources of random information can enable or simplify the computation of certain mathematical objects is a recurring theme of theoretical computer science. This is particularly evident in the context of complexity theory, where for example the difficult question of whether P is equal to BPP has been studied for a long time—so far without an answer.

But the initial question can also be studied in more general settings. Many different versions have been studied in the field of algorithmic randomness. Here, the main subject is to find the correct formalization of what a random object is. The field has very strong interactions with the subject of computability theory (or, in the older terminology, recursion

* Vasco Brattka is supported by the National Research Foundation of South Africa. Rupert Hölzl was supported by a Feodor Lynen postdoctoral research fellowship of the Alexander von Humboldt Foundation and is supported by the Ministry of Education of Singapore through grant R146-000-184-112 (MOE2013-T2-1-062).

theory), and it has been extensively studied what computational properties random objects possess (see [17, 12] for recent monographs on algorithmic randomness).

It can be argued that this approach better represents the original question of what can be computed with access to randomness than, for example, the complexity-theoretic approach; the argument being that space or time bounds are not considered, meaning we are getting a better idea of the real computational *content* of random objects—as opposed to a gauge of their ability to *speed up* a computation until it can be performed within polynomial time. For this reason, the results from algorithmic randomness and computability theory certainly are of high importance. But in this article we will argue that one further ingredient is missing to capture best the intuitive idea behind the above question.

When thinking about the question in the setting of algorithmic randomness, maybe the first classic result that comes to mind is the following well-known theorem (this and the following Theorem 2 are discussed in [17, 12]).

► **Theorem 1** (Kučera-Gács Theorem). *Every sequence $A \in 2^{\mathbb{N}}$ is computable from some Martin-Löf random sequence $X \in 2^{\mathbb{N}}$.*

Informally this means that random objects compute everything. While it may seem at first that this result settles the initial question once and for all, it is not in fact a satisfactory answer. This is because it is not actually the randomness of X that is used to compute A ; if this were the case then X could be replaced with any other “similarly random” set Y and it would still compute A . But this is not so: it is well known that the Turing upper cone of any non-computable set has measure 0 (we will discuss this in more detail in a moment).

This is a clear indication that in fact the computation of A does not really use the randomness of X . Instead, by studying the proof of the theorem, it becomes apparent that X is in fact “tailor-made” (or, perhaps, “tailor-chosen”) to compute A . The randomness only enters the picture insofar as in order to be able to choose an X that computes A we need a large class of sets to choose from, and the class MLR happens to be large enough. Other sufficiently rich classes can be imagined where the same construction would be possible, a trivial example being the class of sets generated from Martin-Löf random sequences by inserting the symbol 0 in every second place.

So the Kučera-Gács Theorem does not settle our initial question, or only a very weak formalization of it. What we would rather like to know is what can be computed given access to *any* sufficiently random sequence. A second possible approach to the question might be Sacks’ Measure Theorem.

► **Theorem 2** (Sacks’ Measure Theorem). *If $A \in 2^{\mathbb{N}}$ is computable from every X in a subset of $2^{\mathbb{N}}$ of positive measure, then A is computable.*

Once again it might seem that this settles the question, as this states that randomness cannot compute anything of interest (i.e., that cannot be computed without randomness in any case). But again, this is not quite the answer to the question we are investigating. Sacks’ theorem only applies if we want to compute a single set A . This is because the proof relies essentially on a majority vote argument.

But there are many very valid settings where this is not the case: often we are given a mathematical problem and want to find a solution to it, and we want to know whether randomness can help us to find such a solution. For a given instance of such a task there may be many admissible solutions; and each of these solutions may have a low probability of being produced by a Turing machine, so that a majority vote mechanic would fail.

To overcome this limitation we therefore need to work within a framework that is more involved while still holding on to the ideas of computability theory. This new framework is

provided by the Weihrauch degrees. The general idea here is to think about a mathematical task as a black box to which we pass an encoding of an instance of the mathematical problem at hand, and the black box has to return an encoding of *one of the* admissible solutions. One example might be a black box NASH which, given a bi-matrix game (A, B) , will produce one of the possible Nash equilibria (x, y) of this game.

Once having established this framework we can now ask questions such as the following: Assume we have a black box solving a certain problem \mathcal{A} . Can we use it to solve another problem \mathcal{B} ? The approach here is heavily inspired by many-one reducibility in computability theory, but applies (in general) to infinite objects instead of finite ones: we code an instance B of problem \mathcal{B} into a valid instance A of problem \mathcal{A} . We then run the black box for \mathcal{A} on A . The black box provides a solution for A . We then need to convert the solution for A back into a solution for B .

The reducibility sketched above is called *Weihrauch reducibility* and is *uniform*. That is, the pre-processing and post-processing steps in the above sketch are required to be performed by a pair of Turing machines uniformly for all instances of \mathcal{B} . The reducibility induces a rich lattice of mathematical problems of different difficulties.

In this article we will let \mathcal{A} be a source of randomness, and then study which problems \mathcal{B} are reducible to it. The randomness source \mathcal{A} will be multi-valued, that is, while it is obliged to output random objects, it has free choice as to which specific random object it outputs. A problem \mathcal{B} is only reducible to \mathcal{A} if the pair of Turing machines is able to uniformly produce a solution for the given instances B of \mathcal{B} , no matter which choice the black box for \mathcal{A} makes. The idea is that then the operations by the Turing machines can rely on no property of the black box's output *except on its randomness* to fulfill their task.¹ It is for this reason that we believe this approach correctly formalizes the initial question of “What can be computed with access to randomness?”

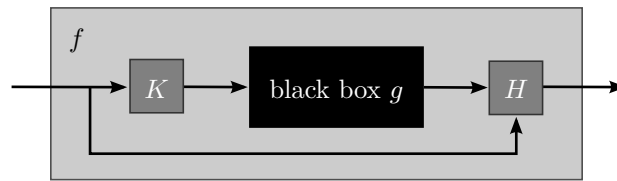
More concretely, we will identify two types of computations with access to random sources; we will determine their location in the Weihrauch lattice; and we will separate them by showing that one of them is significantly stronger than the other.

For the purposes of this extended abstract we focus on a number of key results and include only proof sketches; we also use a minimum of mathematical formalism. The present article focuses primarily on the interaction between computable analysis and algorithmic randomness; the results are presented in a form targeting a computer science audience. We rely to a large degree on computable analysis results from our forthcoming related article [7]. That article, in contrast to the present one, uses the full computable analysis formalism, presents the proofs of its results with all details, and contains much additional material.

2 The Weihrauch Lattice

The original definition of Weihrauch reducibility is due to Klaus Weihrauch [24] and has been studied for many years. More recently it has been noticed that a certain variant of this reducibility yields a lattice that is very suitable for the classification of the uniform

¹ For the sake of precision we should add that this idea is only perfectly realized for the randomness source MLR we study below; and only to a lesser degree by the randomness sources derived from WWKL, where the Turing machines actually can rely on some (very limited) additional property of the black box's output, namely the fact that it is guaranteed to be contained in some positive measure set. One could interpret the differences in computational strength of the different randomness sources studied in this article as stemming from this distinction.



■ **Figure 1** A visualization of the Weihrauch reducibility of f to g via Turing machines K and H .

computational content of mathematical problems (see [13, 20, 19, 6, 5, 4, 8, 11]). The basic reference for all notions from computable analysis is Weihrauch’s textbook [25].

Formally, the Weihrauch lattice is formed by equivalence classes of partial multi-valued functions $f : \subseteq X \rightrightarrows Y$ on represented spaces X, Y . A multi-valued function is considered as a computational problem in the sense that for every $x \in \text{dom}(f)$ the goal is to find *some* $y \in f(x)$. A typical mathematical problem f would be the problem of solving some type of equation or to determine Nash equilibria, as mentioned above. In this case $\text{dom}(f)$ would contain the admissible instances x of the problem and for each instance x the set $f(x)$ would contain the corresponding solutions.

A *represented space* (X, δ) is a set X together with a surjective partial map $\delta : \subseteq \mathbb{N}^{\mathbb{N}} \rightarrow X$ that assigns *names* $p \in \mathbb{N}^{\mathbb{N}}$ to points $\delta(p) = x \in X$. These representations allow us to describe computations on all representable spaces using Turing machines that operate on the names corresponding to points in the space. In order to keep the presentation here as accessible as possible, we will not make any technical use of the machinery of representations and we refer the reader to [7] for all details.

The intuition behind Weihrauch reducibility is that $f \leq_w g$ holds if there is a computational procedure for solving f when allowed a single application of the computational resource g . We make this somewhat more precise.

► **Definition 3** (Weihrauch reducibility). Let $f : \subseteq X \rightrightarrows Y$ and $g : \subseteq W \rightrightarrows Z$ be multi-valued functions on represented spaces. Then f is called *Weihrauch reducible* to g , in symbols $f \leq_w g$, if f can be simulated through an oracle computational process that accesses the oracle g exactly once; that is, if given a name \hat{x} of an input $x \in \text{dom}(f)$ it is effectively possible to compute a name \hat{y} of some value $y \in f(x)$ consulting the oracle function g exactly on one input $w \in \text{dom}(g)$. Formally, the computation works by having a pair of two Turing machines K and H . K operates on \hat{x} to generate \hat{w} , a name for w . The result z of the evaluation of g on w will again be represented by a name \hat{z} . Then the second Turing machine H operates on \hat{z} to generate \hat{y} , the name representing the final output y .

The concept of Weihrauch reducibility can be seen as a variant of many-one reducibility for multi-valued functions on infinite objects. The informal definition above is equivalent to the more formal definition in [7] via [23, Theorem 7.2].

Weihrauch reducibility induces a lattice with a rich and very natural algebraic structure. We briefly summarize some of these algebraic operations for mathematical problems $f : \subseteq X \rightrightarrows Y$ and $g : \subseteq W \rightrightarrows Z$:

- $f \times g$ is the *product* of f and g and represents the parallel evaluation of problem f on some input x and g on some input w .
- $f \sqcup g$ is the *coproduct* of f and g and represents the alternative evaluation of f on some input x or g on some input w (where the input set is the disjoint union $X \sqcup W$ and the output set is $Y \sqcup Z$).

- If $Z = X$, then $f \circ g : \subseteq W \rightrightarrows Y$ is the composition of f and g . Note that since f and g can be partial the following careful definition is needed.

$$\begin{aligned} \text{dom}(f \circ g) &:= \{w \in W : g(w) \subseteq \text{dom}(f)\} \text{ and} \\ (f \circ g)(w) &:= \{y \in Y : (\exists x \in X)(y \in f(x) \text{ and } x \in g(w))\}. \end{aligned}$$

- $f * g := \sup\{f_0 \circ g_0 : f_0 \leq_W f \text{ and } g_0 \leq_W g\}$ is the *compositional product* and represents the consecutive usage of the problem f after the problem g .
- $f^* := \bigsqcup_{n=0}^{\infty} f^n$ is the *finite parallelization* and allows an evaluation of the n -fold product f^n for some arbitrary given $n \in \mathbb{N}$.

The coproduct $f \sqcup g$ is the supremum in the Weihrauch lattice (and we do not specify the infimum operation here). The lattice is not complete as infinite suprema do not need to exist, but the supremum $f * g$ always exists. The finite parallelization is a closure operator in the Weihrauch lattice. Further details can be found in [7].

3 Las Vegas Computability

Randomized algorithms have been studied in the discrete setting for a long time (see for instance Motwani and Raghavan [15]), but very little is known for computations on infinite objects. We will now present a formalization of Las Vegas computability with probabilistic Turing machines that is very close to Babai's original understanding of this concept [3].

► **Definition 4** (Las Vegas computability). A partial multi-valued function $f : \subseteq X \rightrightarrows Y$ on represented spaces is called *Las Vegas computable*, if there exists a Turing machine M that given some input $p \in \mathbb{N}^{\mathbb{N}}$ and some auxiliary input $r \in 2^{\mathbb{N}}$ (the “random advice”) computes f in the following way. The machine M either produces some infinite output $q \in \mathbb{N}^{\mathbb{N}}$ or stops after a finite number of steps and signals a failure with the following additional constraints:

1. if p is a name of some input $x \in \text{dom}(f)$ and M does not fail, then q has to be a name of a correct output $y \in f(x)$,
2. for any fixed such p there is a positive probability that for some random advice r the machine M does not fail.

The aforementioned condition can be made precise by requiring that the set S_p of successful random advices r for which the machine computes without failure has positive measure $\mu(S_p) > 0$ for every fixed p that is a name of some $x \in \text{dom}(f)$. Here μ denotes the uniform measure on $2^{\mathbb{N}}$.

The essential feature of a Las Vegas computation is that it produces a result with positive probability, and if it produces a result, then this result is correct. Hence, the correctness of the computation is never compromised, only the success of the computation is subject to randomization.

We mention that Las Vegas computability as defined above is a refinement of the definition of non-deterministically computable functions, as originally introduced by Martin Ziegler [26] and further studied in [4]. The difference between Las Vegas computability and non-deterministic computability is that in the former case one asks for a positive probability that a piece of advice is successful whereas in the latter case one just asks for the existence of a successful piece of advice.

We now show that the class of Las Vegas computable functions is closed under composition, which means that this class is “reasonable” in a certain sense. The proof of the following theorem is a refined version of the corresponding proof for non-deterministic functions in [4] with an additional invocation of Fubini's Theorem.

► **Theorem 5** (Independent Choice). *The class of Las Vegas computable multi-valued functions on represented spaces is closed under composition.*

Proof sketch. Let $f : \subseteq Y \rightrightarrows Z$ and $g : \subseteq X \rightrightarrows Y$ be Las Vegas computable, witnessed by two probabilistic Turing machines M_f and M_g . We describe the construction of a suitable probabilistic Turing machine M for $f \circ g$. Given a name p of some $x \in \text{dom}(f \circ g)$ as an input for M , we just interpret the random advice of M as a pair $\langle r, s \rangle \in 2^{\mathbb{N}}$ and we use s as random advice in order to simulate M_g on input p and then we simulate M_f on the corresponding output with advice r . The composed machine M fails if either of the simulations of the two machines M_g or M_f fails in this process. It is clear that the composed machine will produce a correct result for $f \circ g$ if it does not fail. We now have to look at the success probabilities of the corresponding machines, i.e., at the measures of the following sets:

- S_p is the set of successful advices s of M_g on input p ;
- $R_{p,s}$ is the set of successful advices r of M_f on the output q that M_g produces on input p with advice s ;
- T_p is the set of successful advices $\langle r, s \rangle$ of M on input p .

We know that $\mu(S_p) > 0$ for all p that are names of elements of $\text{dom}(g)$ and $\mu(R_{p,s}) > 0$ for all combinations of p, s that lead to an output q of M_g that is the name of an element of $\text{dom}(f)$. We need to prove $\mu(T_p) > 0$ for all p that are names of elements of $\text{dom}(f \circ g)$. We obtain

$$T_p = \{\langle r, s \rangle \in 2^{\mathbb{N}} : s \in S_p \text{ and } r \in R_{p,s}\},$$

which implies by the Theorem of Fubini for measurable sets and (strict) monotonicity of the integral for non-negative functions that

$$\mu(T_p) = \int_{S_p} \mu(R_{p,s}) \, d\mu > 0.$$

Here the integrand is understood to be the function $s \mapsto \mu(R_{p,s})$. This proves that M satisfies the necessary conditions. ◀

4 Weak Weak König's Lemma

In this section we will see that we can also characterize Las Vegas computability with the help of Weihrauch reducibility and Weak Weak König's Lemma. Weak König's Lemma is a principle that has been intensively studied in reverse mathematics [22]. The classical lemma of König says (in its weak version) that every infinite binary tree has an infinite path. Here we understand Weak König's Lemma as the mathematical problem $\text{WKL} : \subseteq \text{Tr} \rightrightarrows 2^{\mathbb{N}}, T \mapsto [T]$ that maps an infinite binary tree $T \subseteq 2^{<\mathbb{N}}$ to an infinite path $p \in [T]$ of this tree. By Tr we denote the set of all binary trees (represented via their characteristic functions) and by $[T]$ we denote the set of infinite paths of such a tree. We assume that $\text{dom}(\text{WKL})$ is the set of infinite binary trees. In [4] it was proved that

$$f \leq_{\text{w}} \text{WKL} \iff f \text{ is non-deterministically computable,}$$

and here we prove a similar result for the so-called Weak Weak König's Lemma and Las Vegas computability. Weak Weak König's Lemma has also been introduced in reverse mathematics and for us it is the problem to find an infinite path in a binary tree of positive measure, i.e., $\text{WWKL} : \subseteq \text{Tr} \rightrightarrows 2^{\mathbb{N}}, T \mapsto [T]$, which is the restriction of WKL to the set $\text{dom}(\text{WWKL}) := \{T \in \text{Tr} : \mu([T]) > 0\}$. Using this notation we obtain the following result.

► **Theorem 6** (Las Vegas computability). *For $f : \subseteq X \rightrightarrows Y$ we obtain:*

$$f \leq_W \text{WWKL} \iff f \text{ is Las Vegas computable.}$$

Proof sketch. There is a computable map from infinite binary trees to closed sets given by $T \mapsto [T]$ and this map has a computable multi-valued right inverse. Here trees $T \subseteq 2^{<\mathbb{N}}$ are represented via their characteristic functions $\text{cf}_T : 2^{<\mathbb{N}} \rightarrow \{0, 1\}$ and closed sets $A \subseteq 2^{\mathbb{N}}$ are represented by negative information, for instance by an enumeration of sufficiently many balls $w2^{\mathbb{N}}$ that exhaust the complement of A . Now the fact that f is Las Vegas computable means that f can be computed by a probabilistic machine that takes advantage of a random input $r \in 2^{\mathbb{N}}$. For any fixed input p this machine can also identify the unsuccessful advices, which means that the set $S_p \subseteq 2^{\mathbb{N}}$ of successful advices is co-c.e. closed in p , i.e., we can compute it with respect to negative information. This information can be converted into the characteristic function of a tree T with $[T] = S_p$ and this yields a reduction $f \leq_W \text{WWKL}$, since $\mu(S_p) > 0$. Vice versa any such reduction can be converted into a corresponding Las Vegas machine. ◀

Using a result of Jockusch and Soare that generalizes Theorem 2, one can prove $\text{WKL} \not\leq_W \text{WWKL}$ (see [10, Theorem 20]). This means that WWKL is strictly below WKL .

► **Proposition 7.** $\text{WWKL} <_W \text{WKL}$.

We can also rephrase this result as follows.

► **Corollary 8.** *Every Las Vegas computable multi-valued function is non-deterministically computable, but there are non-deterministically computable multi-valued functions that are not Las Vegas computable.*

In [7] we prove (with a finite extension argument) that determining zeros of continuous functions with sign changes is a concrete problem that is non-deterministically computable but not Las Vegas computable. We close this section by mentioning another important observation.

► **Proposition 9.** $\text{WWKL} \equiv_W \text{WWKL}^*$.

Proof. The reduction $\text{WWKL} \leq_W \text{WWKL}^*$ is obvious. For the other direction, notice that for any two multi-valued functions f and g we have $f \times g = (\text{id} \times g) \circ (f \times \text{id})$ where $f \times \text{id} \leq_W f$ and $\text{id} \times g \leq_W g$. Then Theorems 6 and 5 applied to $f = g = \text{WWKL}$ show $\text{WWKL}^2 \leq_W \text{WWKL}$. Iteration of this argument concludes the proof. ◀

5 Dependence on the Probability

Next one could ask whether there is a difference between the setting described so far (i.e., the setting where we only demand positivity of the success probabilities $\mu(S_p)$) and a setting where one demands fixed minimum probabilities $\mu(S_p) > \varepsilon$ for some $\varepsilon > 0$.

This question can and has already been studied in the form of a corresponding restriction of Weak Weak König's Lemma: Dorais et al. [11] have introduced the problem $\varepsilon\text{-WWKL}$, which is WWKL restricted to $\text{dom}(\varepsilon\text{-WWKL}) := \{T \in \text{Tr} : \mu([T]) > \varepsilon\}$, i.e., to trees whose sets of infinite paths have measure larger than ε . It is clear that if the lower probability bound ε decreases, then one can compute at least as much as before, i.e., the map $\varepsilon \mapsto \varepsilon\text{-WWKL}$ is anti-monotone with regards to Weihrauch reducibility. We have proved that it is even strictly anti-monotone, a result that has independently been obtained by Dorais et al. [11]. Our proof

is essentially a combination of a combinatorial argument that is based on a certain version of the pigeonhole principle, combined with a topological and measure-theoretic reasoning. We just sketch the idea.

► **Theorem 10.** $\varepsilon \geq \delta \iff \varepsilon\text{-WWKL} \leq_W \delta\text{-WWKL}$ for $\varepsilon, \delta \in [0, 1]$.

Proof idea. We only need to prove “ \Leftarrow ”. Let $\varepsilon < \delta$. Then there are positive integers $a < b$ with $\varepsilon < \frac{a}{b} < \delta$. We consider the problem $C_{a,b}$ of finding a point in a closed subset $A \subseteq \{0, \dots, b-1\}$ (given by negative information) of cardinality $|A| \geq a$. One can easily prove that $C_{a,b} \leq_W \varepsilon\text{-WWKL}$; and a more involved argument based on a corresponding pigeonhole principle shows $C_{a,b} \not\leq_W \delta\text{-WWKL}$. This proves $\varepsilon\text{-WWKL} \not\leq_W \delta\text{-WWKL}$. ◀

The aforementioned result can be interpreted such that probability amplification fails for Las Vegas computable functions. Intuitively, this is because we are dealing with infinite computations and even if we perform two randomized computations in parallel we need to start producing some definite output without ever knowing whether one of the involved computations might turn out to be a failure at some later stage.

At one extreme end of the probability spectrum we have 0-WWKL, which is identical to WWKL and hence it represents Las Vegas computations. On the other hand, we have 1-WWKL, which is easily seen to be computable (since every *closed* set $A \subseteq 2^{\mathbb{N}}$ of measure 1 needs to be the full space $A = 2^{\mathbb{N}}$). However, there is still a non-computable problem below all the $\varepsilon\text{-WWKL}$ with $\varepsilon \in [0, 1)$ that is of interest to us, and in a certain sense it is the infimum of all problems $\varepsilon\text{-WWKL}$: this is $(1 - *)\text{-WWKL} : \subseteq \text{Tr}^{\mathbb{N}} \rightrightarrows 2^{\mathbb{N}}$, defined by

$$(1 - *)\text{-WWKL}((T_n)_{n \in \mathbb{N}}) := \bigsqcup_{n \in \mathbb{N}} (1 - 2^{-n})\text{-WWKL}(T_n),$$

where $\text{dom}((1 - *)\text{-WWKL}) := \{(T_n)_{n \in \mathbb{N}} \in \text{Tr}^{\mathbb{N}} : (\forall n \in \mathbb{N}) \mu([T_n]) > 1 - 2^{-n}\}$. Intuitively, this problem is the following: given a sequence of infinite binary trees $(T_n)_{n \in \mathbb{N}}$ with $\mu([T_n]) > 1 - 2^{-n}$ for all n , we want to find one infinite path $p \in [T_n]$ in *one* of the trees T_n . One easily obtains the following corollary from Theorem 10.

► **Corollary 11.** $(1 - *)\text{-WWKL} <_W \varepsilon\text{-WWKL}$ for every $\varepsilon \in [0, 1)$.

6 Algorithmic Randomness

With the arguments in the previous sections we were able to identify the Weihrauch degree of WWKL as that of a natural kind of randomized computation. Having done this, we are now able to locate this type of randomized computation in the Weihrauch lattice and to compare it with other types. Another natural type is computation with access to Martin-Löf oracles. We recall that this has been extensively studied from a non-uniform perspective in computability theory (see [17, 12]). But of course, here we will again take the Weihrauch lattice perspective: we ask what can be reduced to the principle $\text{MLR} : \subseteq 2^{\mathbb{N}} \rightrightarrows 2^{\mathbb{N}}$, which maps an arbitrary input $x \in 2^{\mathbb{N}}$ to an output $y \in 2^{\mathbb{N}}$ that is Martin-Löf random relative to x . In fact, we call the functions $f : \subseteq X \rightrightarrows Y$ with

$$f \leq_W \text{MLR}$$

Martin-Löf computable and they can be seen as those functions that are computable on a *Martin-Löf machine*, i.e., on a machine that can request a Martin-Löf random sequence (relative to the input) exactly once during the course of its computation. Now the obvious question is: how does the power of Martin-Löf machines compare to Las Vegas machines?

We will see that in the Weihrauch lattice MLR is strictly weaker than WWKL and that the distance in the lattice is in fact quite large. This follows from the following result.

► **Theorem 12** (Martin-Löf computability). $\text{MLR} <_{\text{W}}(1 - *)\text{-WWKL}$.

Proof sketch of Theorem 12. We recall (see [17, 12]) that there is a universal Martin-Löf test, which is a computable sequence $(U_i)_{i \in \mathbb{N}}$ of c.e. open sets $U_i \subseteq 2^{\mathbb{N}}$ such that $\mu(U_i) < 2^{-n}$ and $\bigcap_{i=0}^{\infty} U_i$ is exactly the set of all sequences which are not Martin-Löf random. Hence, each complement $A_i := 2^{\mathbb{N}} \setminus U_i$ is a co-c.e. closed set with $\mu(A_i) > 1 - 2^{-n}$ and each A_i only contains Martin-Löf random sequences. Hence, we can compute a corresponding sequence $(T_i)_{i \in \mathbb{N}}$ of infinite binary trees with $[T_i] = A_i$. Upon input of this sequence, $(1 - *)\text{-WWKL}$ yields a Martin-Löf random sequence. The entire argument can be relativized, i.e., it also works in the presence of some oracle $p \in 2^{\mathbb{N}}$. This yields the reduction $\text{MLR} \leq_{\text{W}}(1 - *)\text{-WWKL}$, and this reduction is strict according to [9, Lemma 7.4]. ◀

Together with Corollary 11 the previous theorem yields the following statement which constitutes one of the central results of this article.

► **Corollary 13.** *Every Martin-Löf computable multi-valued function is also Las Vegas computable, but there are Las Vegas computable multi-valued functions which are not Martin-Löf computable.*

Notice the stark contrast with reverse mathematics, where the principles WWKL and MLR are equivalent over RCA_0 . This follows essentially from a formalization of a theorem of Kučera [14]; a detailed alternate proof, which even works over the weaker proof system RCA_0^* , has been given by Avigad et al. [2, Theorem 3.1].

In the next section we will see a concrete example of a problem that is Las Vegas computable, but not Martin-Löf computable.

7 Nash Equilibria

To show that Las Vegas computability is more than just a purely theoretical notion, we will give a concrete example of a useful mathematical task that can be performed with it but not with the weaker types of randomized computation studied in this article.

To this end we have proved (based on results of Arno Pauly) that there is a Las Vegas algorithm to compute Nash equilibria. We recall from [19, 18] that a pair $A, B \in \mathbb{R}^{m \times n}$ of $m \times n$ -matrices is called a *bi-matrix game*. Any vector $s = (s_1, \dots, s_m) \in \mathbb{R}^m$ with $s_i \geq 0$ for all $i = 1, \dots, m$ and $\sum_{j=1}^m s_j = 1$ is called a *mixed strategy*. By S^m we denote the set of these mixed strategies of dimension m . Then a *Nash equilibrium* is a pair $(x, y) \in S^n \times S^m$ of strategies such that

1. $x^T A y \geq w^T A y$ for all $w \in S^n$ and
2. $x^T B y \geq x^T B z$ for all $z \in S^m$.

Nash [16] proved that for any bi-matrix game there exists a Nash equilibrium. By $\text{NASH}_{n,m} : \mathbb{R}^{m \times n} \times \mathbb{R}^{m \times n} \rightrightarrows \mathbb{R}^n \times \mathbb{R}^m$ we denote the corresponding problem

$$\text{NASH}_{n,m}(A, B) := \{(x, y) \in \mathbb{R}^n \times \mathbb{R}^m : (x, y) \text{ is a Nash equilibrium for } (A, B)\}$$

of finding a Nash-equilibrium for an $m \times n$ bi-matrix game and by $\text{NASH} := \bigsqcup_{n,m \in \mathbb{N}} \text{NASH}_{n,m}$ we denote the coproduct of all such games for finite $m, n \in \mathbb{N}$. This means intuitively that NASH is the following problem: given a bi-matrix game (A, B) of arbitrary known dimension

$m \times n$, find a Nash equilibrium (x, y) of (A, B) . Pauly [19, 21] proved the following theorem asserting that the problem NASH is Weihrauch equivalent to the idempotent closure of robust division RDIV on the unit interval.

► **Theorem 14** (Nash equilibria). $\text{NASH} \equiv_{\text{W}} \text{RDIV}^*$.

Robust division is the multi-valued function $\text{RDIV} : [0, 1]^2 \rightrightarrows [0, 1]$ with

$$\text{RDIV}(x, y) := \begin{cases} \left\{ \frac{x}{\max(x, y)} \right\} & \text{if } y > 0, \\ [0, 1] & \text{otherwise.} \end{cases}$$

In other words, RDIV is essentially the problem of computing the fraction $\frac{x}{y}$ within $[0, 1]$, where the result is allowed to be arbitrary in case that the denominator y is zero. It is easy to see that RDIV is discontinuous and hence not computable. The result $\text{NASH} \equiv_{\text{W}} \text{RDIV}^*$ means that one can compute Nash equilibria with a certain number of parallel robust divisions (where the number of divisions needed is known a priori). The intuition behind this is that robust division can be used to solve linear equations and linear inequalities in a compact domain and repeated operations of this type can lead (in a rather involved way) to a Nash equilibrium. In order to prove that Nash equilibria are Las Vegas computable it suffices by Proposition 9 to show that RDIV is Las Vegas computable, i.e., $\text{RDIV} \leq_{\text{W}} \text{WWKL}$.

► **Proposition 15.** $\text{RDIV} \leq_{\text{W}} \text{WWKL}$.

We do not give a full proof but describe the idea behind the Las Vegas algorithm for robust division RDIV informally (we note that $r \in [0, 1]$ can be used equivalently instead of $r \in 2^{\mathbb{N}}$ as random advice):

1. Given $x, y \in [0, 1]$ and a random advice $r \in [0, 1]$, we aim to compute the fraction $z = \frac{x}{\max(x, y)}$.
2. We guess that r is a correct solution, in particular $r = z$ if $y > 0$, and we start to output longer and longer approximations of r (in form of rational intervals $[a, b]$ with $a < r < b$).
3. Simultaneously, we try to find out whether $y > 0$, which we will eventually recognize, if this is correct.
4. As soon as we find that $y > 0$, we can compute the true result $z = \frac{x}{\max(x, y)}$ and in this case we start to produce approximations of z as output.
5. If at some stage we find that the best approximation $[a, b]$ of r that was already produced as output is incompatible with z , i.e., if $z \notin [a, b]$, then we stop the computation and indicate that it failed.

This algorithm produces a correct result $z \in \text{RDIV}(x, y)$ whenever it does not fail. It can only fail if $y > 0$. In the moment when the algorithm detects $y > 0$, then there is still a positive probability $\mu([a, b]) = b - a > 0$ that the random advice r , which has only been approximated up to $[a, b]$ so far, is compatible with the true result $z = \frac{x}{\max(x, y)}$. Hence, the above algorithm constitutes a Las Vegas algorithm for robust division. Altogether we obtain the following.

► **Corollary 16.** $\text{NASH} \leq_{\text{W}} \text{WWKL}$.

The above algorithm for robust division only yields success probabilities arbitrarily close to zero (depending on when it is recognized that the denominator is positive). In fact, one can prove that robust division (and hence Nash equilibria) cannot be computed with any fixed positive success probability.

► **Proposition 17.** $\text{RDIV} \not\leq_W \varepsilon\text{-WWKL}$ for $\varepsilon > 0$.

In particular, this implies the following by Corollary 11 and Theorem 12.

► **Corollary 18.** *Computing Nash equilibria is Las Vegas computable but not Martin-Löf computable.*

Also note that Proposition 17 implies that the reducibility in Corollary 16 is strict.

8 Conclusions

In this paper we have introduced the class of Las Vegas computable functions (for computations with infinite objects), and we have proved that it is strictly included in between the classes of Martin-Löf computable functions and non-deterministically computable functions. As a natural example of a Las Vegas computable problem that is not Martin-Löf computable, we have discussed the problem of finding Nash equilibria. Principles very closely related to MLR and WWKL turned out to be equivalent to each other in the non-uniform and more coarse-grained setting of reverse mathematics [14, 2]. Hence, reverse mathematics could not uncover the fine-grained uniform distinctions that we have made with the help of the Weihrauch lattice.

Having presented our above results, we should at the closure of this article mention other existing work in which the question “What can be computed with access to randomness?” was studied in other contexts, and clarify how these settings differ from ours. The work concerning the complexity-theoretic setting may be the most well-known one. There, similar motivations to the ones behind our present article have led, among other questions, to the study of the inclusion chain $\text{P} \subseteq \text{ZPP} \subseteq \text{BPP} \subseteq \text{NP}$. The central open question in this area is whether $\text{P} = \text{BPP}$. The setting differs very much from ours: First of all, it is discrete, that is, the objects computed are finite strings. Furthermore, resource bounds are present in this setting; they are in fact required to make this study interesting, as without them there is no computability-theoretic difference between deterministic and randomized computation.

Note that there is also a body of work by Allender et al. in the complexity-theoretic setting (see, for example, [1]). While it may seem closely related to the initial question at first, we would like to point out that the computations studied there are not truly taking advantage of randomness as a resource for computation. Rather, in this work, computations are made using reductions to the sets R_C , R_K , and so on. These are the sets containing the information about which finite strings are compressible and which are not; therefore the work by Allender et al. can be thought of as not being about computation from *randomness*, but rather about computation from the *knowledge about what is random*.

Once we leave the domain of complexity theory, that is, give up resource bounds on the computations, we need to look at the computation of infinite objects if we want to find interesting insights into the power of randomness. In this area the important distinction is then that between uniform and non-uniform computation. This paper has focused on the levels of computation power that are needed in the uniform, infinite setting. If one wants to look at the *non-uniform* counterpart of our work then that is the work on WWKL and MLR in reverse mathematics, as cited above.

Acknowledgments. The authors would like to thank the referees for their detailed feedback.

References

- 1 Eric Allender, Harry Buhrman, Michal Koucký, Dieter van Melkebeek, and Detlef Ronneburger. Power from random strings. *SIAM Journal on Computing*, 35(6):1467–1493, 2006.
- 2 Jeremy Avigad, Edward T. Dean, and Jason Rute. Algorithmic randomness, reverse mathematics, and the dominated convergence theorem. *Annals of Pure and Applied Logic*, 163(12):1854–1864, 2012.
- 3 László Babai. Monte-Carlo algorithms in graph isomorphism testing. Technical Report No. 79-10, Université de Montréal, Département de Mathématique et de Statistique, 1979.
- 4 Vasco Brattka, Matthew de Brecht, and Arno Pauly. Closed choice and a uniform low basis theorem. *Annals of Pure and Applied Logic*, 163(8):986–1008, 2012.
- 5 Vasco Brattka and Guido Gherardi. Effective choice and boundedness principles in computable analysis. *The Bulletin of Symbolic Logic*, 17(1):73–117, 2011.
- 6 Vasco Brattka and Guido Gherardi. Weihrauch degrees, omniscience principles and weak computability. *The Journal of Symbolic Logic*, 76(1):143–176, 2011.
- 7 Vasco Brattka, Guido Gherardi, and Rupert Hölzl. Probabilistic computability and choice. July 2014. Preliminary version available at <http://arxiv.org/abs/1312.7305>.
- 8 Vasco Brattka, Guido Gherardi, and Alberto Marcone. The Bolzano-Weierstrass theorem is the jump of weak König’s lemma. *Annals of Pure and Applied Logic*, 163(6):623–655, 2012.
- 9 Vasco Brattka, Matthew Hendtlass, and Alexander P. Kreuzer. On the uniform computational content of computability theory. January 2015. Preliminary version available at <http://arxiv.org/abs/1501.00433>.
- 10 Vasco Brattka and Arno Pauly. Computation with advice. In Xizhong Zheng and Ning Zhong, editors, *CCA 2010, Proceedings of the Seventh International Conference on Computability and Complexity in Analysis*, Electronic Proceedings in Theoretical Computer Science, pages 41–55, 2010.
- 11 François G. Dorais, Damir D. Dzhafarov, Jeffrey L. Hirst, Joseph R. Mileti, and Paul Shafer. On uniform relationships between combinatorial problems. *Transactions of the AMS*, 2014. Accepted for publication. Preliminary version available at <http://arxiv.org/abs/1212.0157>.
- 12 Rodney G. Downey and Denis R. Hirschfeldt. *Algorithmic randomness and complexity*. Theory and Applications of Computability. Springer, New York, 2010.
- 13 Guido Gherardi and Alberto Marcone. How incomputable is the separable Hahn-Banach theorem? *Notre Dame Journal of Formal Logic*, 50(4):393–425, 2009.
- 14 Antonín Kučera. Measure, Π_1^0 -classes and complete extensions of PA. In Heinz-Dieter Ebbinghaus, Gert H. Müller, and Gerald E. Sacks, editors, *Recursion Theory Week. Proceedings of the Conference Held at the Mathematisches Forschungsinstitut in Oberwolfach, April 15–21, 1984*, volume 1141 of *Lecture Notes in Mathematics*, pages 245–259. Springer, Berlin, 1985.
- 15 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, 1995.
- 16 John Nash. Non-cooperative games. *Annals of Mathematics*, 54:286–295, 1951.
- 17 André Nies. *Computability and Randomness*, volume 51 of *Oxford Logic Guides*. Oxford University Press, New York, 2009.
- 18 Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, Cambridge, 2007.
- 19 Arno Pauly. How incomputable is finding Nash equilibria? *Journal of Universal Computer Science*, 16(18):2686–2710, 2010.
- 20 Arno Pauly. On the (semi)lattices induced by continuous reducibilities. *Mathematical Logic Quarterly*, 56(5):488–502, 2010.

- 21 Arno Pauly. *Computable Metamathematics and its Application to Game Theory*. PhD thesis, University of Cambridge, Computer Laboratory, Clare College, Cambridge, 2011.
- 22 Stephen G. Simpson. *Subsystems of Second Order Arithmetic*. Perspectives in Logic, Association for Symbolic Logic. Cambridge University Press, Poughkeepsie, 2009.
- 23 Nazanin R. Tavana and Klaus Weihrauch. Turing machines on represented sets, a model of computation for analysis. *Logical Methods in Computer Science*, 7(2:19):1–21, 2011.
- 24 Klaus Weihrauch. The degrees of discontinuity of some translators between representations of the real numbers. Technical Report TR-92-050, International Computer Science Institute, Berkeley, July 1992.
- 25 Klaus Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.
- 26 Martin Ziegler. Real hypercomputation and continuity. *Theory of Computing Systems*, 41(1):177–206, 2007.

Understanding Model Counting for β -acyclic CNF-formulas

Johann Brault-Baron^{*1}, Florent Capelli^{†2}, and Stefan Mengel^{‡3}

1 LSV UMR 8643, ENS Cachan and Inria, France

2 IMJ UMR 7586 - Logique, Université Paris Diderot, France

3 LIX UMR 7161, École Polytechnique, France

Abstract

We show that #SAT on β -acyclic CNF-formulas can be solved in polynomial time. In contrast to previous algorithms for other structurally restricted classes of formulas, our algorithm does not proceed by dynamic programming. Instead, it works along an elimination order, solving a weighted version of constraint satisfaction. We give evidence that this deviation from more standard algorithms is no coincidence by showing that it is outside of the framework recently proposed by Sæther et al. (SAT 2014) which subsumes all other structural tractability results for #SAT known so far.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases model counting, hypergraph acyclicity, structural tractability

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.143

1 Introduction

The propositional model counting problem #SAT is, given a CNF-formula F , to count the satisfying assignments of F . #SAT is the canonical #P-complete problem and is thus central to the area of counting complexity. Moreover, many important problems in artificial intelligence research reduce to #SAT (see e.g. [19]), so there is also great interest in the problem from a practical point of view.

Unfortunately, #SAT is computationally very hard: even for very restricted CNF-formulas, e.g. monotone 2-CNF-formulas, the problem is #P-hard and in fact even #P-hard to approximate [19]. Thus the focus of research in finding tractable classes of #SAT-instances has turned to so-called *structural* classes, which one gets by assigning a graph or hypergraph to a CNF-formula and then restricting the class of (hyper)graphs considered. The general idea is that if the (hyper)graph associated with an instance has a tree-like decomposition that is “nice” enough, e.g. a tree decomposition of small width, then there is a dynamic programming algorithm that solves #SAT for the instance. In the recent years, many such dynamic programming algorithms for ever more general classes of graphs and hypergraphs have been found, see e.g. [13, 21, 18, 22, 7].

It had been an open question for some time how far this approach could be pushed, until very recently Sæther, Telle and Vatshelle, in a striking contribution [20], introduced a new

* This work has received support from the French Agence Nationale de la Recherche, AGGREG project reference ANR-14-CE25-0017-01.

† Partially supported by ANR Blanc CompA ANR-13-BS02-0001.

‡ Partially supported by a grant from Qualcomm.



framework for #SAT. This framework which we call STV-framework is centered around a new width measure called PS-width and aims to formalize the most general class for which efficient dynamic programming for #SAT is possible (see Section 2.5 for details). We consider the STV-framework to be a very convincing formalization, delineating the limits of dynamic programming for #SAT. This belief is supported by the fact that in the full version of this paper we show that the STV-framework gives a uniform explanation for *all* previously known structural tractability results for #SAT.

In this article, we focus on β -acyclic CNF-formulas, i.e., formulas whose associated hypergraph is β -acyclic. There are several different reasonable ways of defining acyclicity of hypergraphs that have been proposed [12, 11], and β -acyclicity is the most general acyclicity notion generally considered in the literature for which #SAT could be tractable (see the discussions in [17, 7]). The complexity of #SAT for β -acyclic formulas is left as an open problem in [7] because it is interesting for several reasons: First, up to this paper, it was the only structural class of formulas for which we know that SAT is tractable [17] without this directly generalizing to a tractability result for #SAT. This is because the algorithm of [17] does *not* proceed by dynamic programming but uses resolution, a technique that is known to generally not generalize to counting. Moreover, β -acyclicity can be generalized to a width-measure [15], so there is hope that a good algorithm for β -acyclic formulas might generalize to wider classes for which even the status for SAT is left as an open problem in [17]. Since decomposition techniques based on hypergraph acyclicity tend to be more general than graph-based techniques [14], this might lead to large, new classes of tractable #SAT-instances.

The contribution of this paper is twofold: First, we show that #SAT on β -acyclic formulas is tractable. In fact, we show that a more general counting problem which we call *weighted counting for constraint satisfaction with default values*, for short #CSP_{def}, is tractable on β -acyclic hypergraphs. We remark that there is another line of research on #CSP, the counting problem related to constraint satisfaction, where dichotomy theorems for weighted #CSP depending on fixed constraint languages are proven, see e.g. [5, 6]. We stress that we do *not* assume that the relations of our instances are fixed but we consider them as part of the input. Thus our results and those on fixed constraint languages are completely unrelated. Instead, the structural restrictions we consider are similar to those considered e.g. in [9], but since we allow clauses, resp. relations, of unbounded arity, our results and those of [9] are incomparable as well.

We note that our algorithm is very different in style from the algorithms for structural #SAT in the literature. Instead of doing dynamic programming along a decomposition, we proceed along a vertex elimination order which is more similar to the approach to SAT in [17]. But in contrast to using well-understood resolution techniques, we develop from scratch a procedure to update the weights of our #CSP_{def} instance along the elimination order. Our algorithm is non-obvious and novel, but it is relatively easy to write down and its correctness is easy to prove. Indeed, most of the work in the full version of this paper is spent on showing the polynomial runtime bound which requires a thorough understanding of how the weights of instances evolve during the algorithm. Unfortunately, these considerations cannot be presented in this version of this paper due to space restrictions.

Our second contribution is showing that our tractability result is not covered by the STV-framework, which, as discussed before, covers all other known structural tractability results for #SAT. In fact, we show that from [20] we cannot even get subexponential runtime bounds for β -acyclic #SAT. This can be seen as an explanation for why the algorithm for β -acyclic #SAT is so substantially different from the algorithms from the literature. We

feel that the deviation from the usual dynamic programming techniques is not a coincidence but instead due to the fact that β -acyclic #SAT is the first known tractable class which is not explained by the unifying framework of [20]. Thus, our algorithm indeed introduces a new algorithmic technique for #SAT that allows the solution of instances that could not be solved with techniques known before.

2 Preliminaries and notation

2.1 Weighted counting for constraint satisfaction with default values

Let D and X be two sets. D^X denotes the set of functions from X to D . We think of X as a set of variables and of D as a domain, and thus we call $a \in D^X$ an *assignment* to the variables X . A *partial assignment* to the variables X is a mapping in D^Y where $Y \subseteq X$. If $a \in D^X$ and $Y \subseteq X$, we denote by $a|_Y$ the *restriction* of a onto Y . For $a \in D^X$ and $b \in D^Y$, we write $a \sim b$ if $a|_{X \cap Y} = b|_{X \cap Y}$ and if $a \sim b$, we denote by $a \cup b$ the mapping in $D^{X \cup Y}$ with $(a \cup b)(x) = a(x)$ if $x \in X$ and $(a \cup b)(x) = b(x)$ otherwise. Let $a \in D^X$, $y \notin X$ and $d \in D$. We write $a \oplus_y d := a \cup \{y \mapsto d\}$. We denote by \mathbb{Q}_+ the set of nonnegative rationals.

► **Definition 1.** A *weighted constraint with default value* $c = (f, \mu)$ on variables X and domain D is a function $f : S \rightarrow \mathbb{Q}_+$ with $S \subseteq D^X$ and $\mu \in \mathbb{Q}_+$. $S = \text{supp}(c)$ is called the *support* of c , $\mu(c) = \mu$ is called the *default value* and we denote by $\text{var}(c) = X$ the *variables* of c . We define the *size* $|c|$ of the constraint c to be $|c| := |S| \cdot |\text{var}(c)|$. The constraint c naturally induces a total function on D^X , also denoted by c , defined by $c(a) := f(a)$ if $a \in S$ and $c(a) := \mu$ otherwise.

To ease the notation, when a is an assignment to a set $X \supseteq \text{var}(c)$, we make the convention $c(a) = c(a|_{\text{var}(c)})$. Observe that we do not assume $\text{var}(c)$ to be non-empty. A constraint whose set of variables is empty has only one possible value in its support: the value associated with the empty assignment (the assignment that assigns no variable).

Since we only consider weighted constraints with default value in this paper, we will only say *weighted constraint* where the default value is always implicitly understood. Note that we restrict ourselves to non-negative weights, because non-negativity will be crucial in the proofs. This is not a problem in our context, non-negative numbers are sufficient to encode #SAT as we will see in Section 2.3. We however use rational numbers for convenience and all our results can be extended easily to non-negative algebraic numbers.

► **Definition 2.** The problem $\#\text{CSP}_{\text{def}}$ is the problem of computing, given a finite set I of weighted constraints on domain D , the *partition function*

$$w(I) = \sum_{a \in D^{\text{var}(I)}} \prod_{c \in I} c(a),$$

where $\text{var}(I) := \bigcup_{c \in I} \text{var}(c)$.

The *size* $\|I\|$ of a $\#\text{CSP}_{\text{def}}$ -instance I is defined to be $\|I\| := \sum_{c \in I} |c|$. Its *structural size* $s(I)$ of I is defined to be $s(I) := \sum_{c \in I} |\text{var}(c)|$.

Note that the size of an instance as defined above roughly corresponds to that of an encoding in which the non-default values, i.e., the values on the support, are given by listing the support and the associated values in one table for each relation. We consider this convention as very natural and indeed it is near to the conventions in database theory and artificial intelligence.

Given an instance I , it will be useful to refer to subinstances of I , that is a set $J \subseteq I$. We will also refer to partition function of subinstances under some partial assignment, that is, the partition function of J where some of its variables are forced to a certain value. To this end, for $a \in D^W$, with $W \subseteq \text{var}(I)$, and $J \subseteq I$ with $V' = \text{var}(J)$ we define

$$w(J, a) := \sum_{\substack{b \in D^{V'} \\ a \sim b}} \prod_{c \in J} c(b).$$

2.2 Graphs and hypergraphs associated to CNF-formulas

We use standard notation for graphs which can e.g. be found in [10]. A *hypergraph* $\mathcal{H} = (V, E)$ consists of a finite set V and a set E of non-empty subsets of V . The elements of V are called *vertices* while the elements of E are called *edges*. As usual for graphs, we sometimes denote the vertex set of a hypergraph \mathcal{H} by $V(\mathcal{H})$ and the edge set of \mathcal{H} by $E(\mathcal{H})$. The size of a hypergraph is defined to be $\|\mathcal{H}\| = \sum_{e \in E(\mathcal{H})} |e|$.

We denote by $\mathcal{H} \setminus v$ the hypergraph we get from \mathcal{H} after deleting v from $V(\mathcal{H})$ and all edges $e \in E(\mathcal{H})$ and then deleting the empty edge if it occurs.

We are interested in structural restrictions of the problem $\#\text{CSP}_{\text{def}}$. i.e., we restrict the way the variables interact in the different constraints. To formalize this, we introduce the hypergraph associated to an instance of $\#\text{CSP}_{\text{def}}$: The hypergraph $\mathcal{H}(I)$ associated to $\#\text{CSP}_{\text{def}}$ -instance I is the hypergraph $\mathcal{H}(I) := (\text{var}(I), E_I)$ where $E_I := \{\text{var}(c) \mid c \in I\}$. The hypergraph of a CNF-formula is defined as $\mathcal{H}(F) := (\text{var}(F), E_F)$ where $E_F := \{\text{var}(C) \mid C \in \text{cla}(F)\}$ where $\text{var}(F)$ denotes the set of variables of F and $\text{cla}(F)$ denotes the set of clauses of F .

The incidence graph $I(\mathcal{H})$ of a hypergraph $\mathcal{H} = (V, E)$ is the bipartite graph with the vertex set $V \cup E$ and an edge between $v \in V, e \in E$ if and only if $v \in e$. Similarly, the incidence graph $I(F)$ of a CNF-formula F has the vertex set $\text{var}(F) \cup \text{cla}(F)$ and $x \in \text{var}(F)$ and $C \in \text{cla}(F)$ are connected by an edge if and only if x appears in C .

2.3 Relation to $\#\text{SAT}$

We show in this section how we can encode $\#\text{SAT}$ into $\#\text{CSP}_{\text{def}}$ -instances with the same hypergraphs.

Classically, in CSP, all the solutions to a constraint are explicitly listed. For a CNF-formula however, each clause with n variables has $2^n - 1$ solutions, which would lead to a CSP-representation exponentially bigger than the CNF-formula. One way of dealing with this is encoding CNF-formulas into CSP-instances by listing all assignments that are *not* solution of a constraint, see e.g. [8]. In this encoding, each clause has only one counter-example and the corresponding CSP-instance is roughly of the same size as the CNF-formula.

The strength of the CSP with default values is that it can easily embed both representations. This leads to a polynomial reduction from $\#\text{SAT}$ to $\#\text{CSP}_{\text{def}}$.

► **Lemma 3.** *Given a CNF-formula F one can construct in polynomial time a $\#\text{CSP}_{\text{def}}$ -instance I on variables $\text{var}(F)$ and domain $\{0, 1\}$ such that*

- $\mathcal{H}(F) = \mathcal{H}(I)$,
- for all $a \in \{0, 1\}^{\text{var}(F)}$, a is a solution of F if and only if $\prod_{c \in I} c(a) = 1$, and 0 otherwise, and
- $s(I) = \|I\| = |F|$.

Proof. For each clause C of F , we define a constraint c with default value 1 whose variables are the variables of C and such that $\text{supp}(c) = \{a\}$ and $c(a) = 0$, where a is the only assignment of $\text{var}(C)$ that is not a solution of C . It is easy to check that this construction has the above properties. \blacktriangleleft

2.4 β -acyclicity of hypergraphs

In this section we introduce the characterizations of β -acyclicity of hypergraphs we will use in this paper. We remark that there are many more characterizations, see e.g. [12, 3, 4].

► **Definition 4.** Let \mathcal{H} be a hypergraph. A vertex $x \in V(\mathcal{H})$ is called a *nest point* if $\{e \in E(\mathcal{H}) \mid x \in e\}$ forms a sequence of sets increasing with respect to inclusion, that is $\{e \in E(\mathcal{H}) \mid x \in e\} = \{e_1, \dots, e_k\}$ with $e_i \subseteq e_{i+1}$ for $i \in \{1, \dots, k-1\}$.

A β -elimination order for \mathcal{H} is defined inductively as follows:

- If $\mathcal{H} = \emptyset$, then only the empty tuple is a β -elimination order for \mathcal{H} .
- Otherwise, (x_1, \dots, x_n) is a β -elimination for \mathcal{H} if x_1 is a nest point of \mathcal{H} and (x_2, \dots, x_n) is a β -elimination order for $\mathcal{H} \setminus x_1$.

A hypergraph \mathcal{H} called β -acyclic if and only if there exists a β -elimination order for \mathcal{H} .

One can easily show [4] that removing a nest point in a hypergraph does not change its β -acyclicity. Since deciding if a vertex is a nest point could be done in polynomial time, a greedy elimination of nest points yields a polynomial time algorithm to test the β -acyclicity of a hypergraph and to compute a β -elimination order if it exists.

We will also make use of another equivalent characterization of β -acyclic hypergraphs. A graph G is defined to be *chordal bipartite* if it is bipartite and every cycle of length at least 6 in G has a chord.

► **Theorem 5 ([1]).** *A hypergraph is β -acyclic if and only if its incidence graph is chordal bipartite.*

We say that a $\#\text{CSP}_{\text{def}}$ -instance I is β -acyclic if $\mathcal{H}(I)$ is β -acyclic and we use an analogous convention for $\#\text{SAT}$. Note that the incidence graph of an instance I and that of its hypergraph in general do not coincide, because I might contain several constraints with the same sets of variables. But with Theorem 5, it is not hard to see that the incidence graph of an instance I is chordal bipartite if and only if the incidence graph of the hypergraph of I is chordal bipartite, so we can interchangeably use both notions of incidence graphs in this paper without changing the class of instances.

Using Lemma 3 gives the following easy corollary.

► **Corollary 6.** *$\#\text{SAT}$ is polynomial time reducible to $\#\text{CSP}_{\text{def}}$. Moreover, $\#\text{SAT}$ restricted to β -acyclic formulas is polynomial time reducible to $\#\text{CSP}_{\text{def}}$ restricted to β -acyclic instances.*

2.5 Width measures of graphs and CNF-Formulas

In this section we introduce several width measures on graphs and CNF-formulas that are used when relating our algorithm for β -acyclic $\#\text{CSP}_{\text{def}}$ to the framework of Sæther, Telle and Vatshelle [20]. Readers only interested in the algorithmic part of this paper may safely skip to Section 3.

We consider several width notions that are mainly defined by branch decompositions. For an introduction into this area and many more details see [23]. For a tree T we denote by

$L(T)$ the set of the leaves of T and by $V(T)$ the set of vertices of T . A *branch decomposition* (T, δ) of a graph $G = (V, E)$ consists of a subcubic tree T , i.e., a tree in which every vertex has degree at most 3, and a bijection δ between $L(T)$ and V . For convenience we often identify $L(T)$ and V . Moreover, it is often convenient to see a branch decomposition as rooted tree, and as this does not change any of the notions we define (see [23]), we generally follow this convention. For every $x \in V(T)$ we define T_x be the subtree of T rooted in x . From x we get a partition or *cut* of V into two sets defined by $(L(T_x), V \setminus L(T_x))$. For a set $X \subseteq V$ we often write \bar{X} for $V \setminus X$.

Given a symmetric function $f : 2^V \times 2^V \rightarrow \mathbb{R}$ we define the f -width of a branch decomposition (T, δ) to be $\max_{x \in V(T)} f(L(T_x), V \setminus L(T_x))$, i.e., the f -width is the maximum value of f over all cuts of the vertices of T . The f -branch width of a graph G is defined as the minimum f -width of all branch decompositions of G .

Given a graph $G = (V, E)$ and a cut (X, \bar{X}) of V , we define $G[X, \bar{X}]$ to be the graph with vertex set V and edge set $\{uv \mid u \in X, v \in \bar{X}, uv \in E\}$.

We will use at several places the well-known notion of treewidth of a graph G , denoted by $\mathbf{tw}(G)$. Instead of working with the usual definition of treewidth (see e.g. [2]), it is more convenient for us to work with the strongly related notion of *Maximum-Matching-width* (for short *MM-width*) introduced by Vatselle [23]. The MM-width of a graph G , denoted by $\mathbf{mmw}(G)$, is defined as the f -branch width of G for the function f that, given a cut (X, \bar{X}) of G , computes the size of the maximum matching of $G[X, \bar{X}]$. MM-width and treewidth are linearly related [23, p. 28].

► **Lemma 7.** *Let G be a graph, then $\frac{1}{3}(\mathbf{tw}(G) + 1) \leq \mathbf{mmw}(G) \leq \mathbf{tw}(G) + 1$.*

The *Maximum-Induced-Matching-width* (for short *MIM-width*) is another width measure of graphs that we will use extensively: The MIM-width of a graph G , denoted by $\mathbf{mimw}(G)$, is defined as the f -branch width of G for the function f that, given a cut (X, \bar{X}) of G , computes the size of the maximum induced matching of $G[X, \bar{X}]$.

Given a CNF-formula F , we say that a set of clauses $\mathcal{C} \subseteq \text{cla}(F)$ is *projection satisfiable* if there is an assignment to F that satisfies all clauses in \mathcal{C} and no clause in $\text{cla}(F) \setminus \mathcal{C}$. The PS-value of F is defined to be the number of projection satisfiable subsets of $\text{cla}(F)$. Let F be a CNF-formula, $X \subseteq \text{var}(F)$ and $\mathcal{C} \subseteq \text{cla}(F)$. Then we denote by $F_{X, \mathcal{C}}$ the formula we get from F by deleting first every clause not in \mathcal{C} and then every variable not in X .

Let $I(F)$ be the incidence graph of F and let (A, \bar{A}) be a cut of $I(F)$. Let $X := \text{var}(F) \cap A$, $\bar{X} := \text{var}(F) \cap \bar{A}$, $\mathcal{C} := \text{cla}(F) \cap A$ and $\bar{\mathcal{C}} := \text{cla}(F) \cap \bar{A}$. Let $ps(A, \bar{A})$ be the maximum of the PS-values of $F_{X, \bar{\mathcal{C}}}$ and $F_{\bar{X}, \mathcal{C}}$. Then the PS-width of a branch decomposition (T, δ) of $I(F)$ is defined as the ps-branch width of (T, δ) . Moreover, the PS-width of F , denoted $\mathbf{psw}(F)$, is defined to be the ps-branch width of $I(F)$.

Let us try to give an intuition why we believe that PS-width is a good notion to model the limits of tractable dynamic programming for #SAT: The dynamic programming algorithms in the literature typically proceed by cutting instances into subinstances and then iteratively solving the instance along these cuts. During this process, some information has to be propagated between the subinstances. Intuitively, a minimum amount of such information is which sets of clauses are already satisfied by certain assignments and which clauses still have to be satisfied later in the process. In doing this, the individual clauses can be “bundled together” if they are satisfied by an assignment simultaneously. The number such bundles is exactly the PS-width of a cut, so we feel that PS-width is a good formalization of the minimum amount of information that has to be propagated during dynamic programming in the style of the algorithms from the literature.

Not only is PS-width in our opinion a good measure for the limits of dynamic programming, but Sæther, Telle and Vatshelle also showed that it allows efficient solving of #SAT.

► **Theorem 8** ([20]). *Given a CNF-formula F with n variables and m clauses and of size s , and a branch decomposition (T, δ) of the incidence graph $I(F)$ of F with PS-width k , one can count the number of satisfying assignments of F in time $O(k^3 s(m + n))$.*

We admit that the intuition explained above is rather vague and informal, so the reader might or might not share it, but we stress that it is supported more rigorously by the fact that all known tractability results from the literature that were shown by dynamic programming can be explained by a combination of PS-width and Theorem 8 (see the full version of this paper).

3 The algorithm

In this section we describe an algorithm that, given an instance I of #CSP_{def} on domain D and a nest point x of $\mathcal{H}(I)$, constructs in a polynomial number of arithmetic operations an instance I' such that $\mathcal{H}(I') = \mathcal{H}(I) \setminus x$, $\|I'\| \leq \|I\|$ and $w(I) = |D|w(I')$. We then explain that if I is β -acyclic, we can iterate the procedure to compute $w(I)$ in a polynomial number of arithmetic operations.

To make the presentation of the algorithm more clear, we will first consider a special case before presenting the algorithm for the general case.

3.1 The special case of nested constraints

In this section we consider #CSP_{def}-instances whose variable scopes are nested, i.e., instances of the form $I = \{c_1, \dots, c_p\}$ where $\text{var}(c_1) \subseteq \dots \subseteq \text{var}(c_p)$. Note that these instances are β -acyclic, but of course there are β -acyclic instances not of this form which will be treated in the next section. Let us first sketch the idea behind the algorithm.

Fix an instance I as above and let $x \in \text{var}(c_1)$. Observe that x chosen this way is a nest point of $\mathcal{H}(I)$. We want to eliminate x from I to get an instance I' such that

$$w(I) = |D|w(I'). \tag{1}$$

For each constraint c_i , the instance I' will have a constraint c'_i in the variables $\text{var}(c_i) \setminus \{x\}$.

The idea behind the computation of the weights of I' is as follows: For every subinstance $I_i = \{c_1, \dots, c_i\}$ we want for all assignments a to $\text{var}(I')$ that $w(I_i, a) = |D|w(I'_i, a)$ where $I'_i = \{c'_1, \dots, c'_i\}$. Note that (1) follows directly from this. Since a is an assignment to all the variables of I' , we have $w(I'_i, a) = \prod_{j=1}^i c'_j(a)$. So let us compute the weights of the c'_j .

For $i = 1$ we have

$$c'_1(a) = w(I'_1, a) = \frac{w(I_1, a)}{|D|} = \frac{\sum_{d \in D} c_1(a \oplus_x d)}{|D|}.$$

For $i > 1$ we have $|D| \prod_{j=1}^i c'_j(a) = |D|w(I'_i, a) = w(I_i, a) = \sum_{d \in D} \prod_{j=1}^i c_j(a \oplus_x d)$. It follows that

$$c'_i(a) = \frac{\sum_{d \in D} \prod_{j=1}^i c_j(a \oplus_x d)}{|D| \prod_{j=1}^{i-1} c'_j(a)} = \frac{\sum_{d \in D} \prod_{j=1}^i c_j(a \oplus_x d)}{|D|w(I'_{i-1}, a)}.$$

By construction we have $|D|w(I'_{i-1}, a) = w(I_{i-1}, a) = \sum_{d \in D} \prod_{j=1}^{i-1} c_j(a \oplus_x d)$. The case where the denominator is zero will be dealt with in the proof Theorem 9.

Choosing the weights for the c'_i as described above, yields an instance I' that satisfies (1). Iterating this process then yields an algorithm to solve the instance I . The following theorem formalizes the above discussion, applies some arithmetic simplifications and analyzes the number of arithmetic operations performed in the procedure.

► **Theorem 9.** *Let I be a set of weighted constraints on domain D of the form $I = \{c_1, \dots, c_p\}$ where $\text{var}(c_1) \subseteq \dots \subseteq \text{var}(c_p)$. Let $x \in \text{var}(c_1)$.*

We define a new instance $I' := \{c'_1, \dots, c'_p\}$ such that $c'_i := (f'_i, \mu_i)$ is the weighted constraint on variables $\text{var}(c'_i) = \text{var}(c_i) \setminus \{x\}$, with default value $\mu(c_i)$ and $\text{supp}(c'_i) := \{a \in D^{\text{var}(c'_i)} \mid \exists d \in D, (a \oplus_x d) \in \text{supp}(c_i)\}$. Moreover, for all $a \in \text{supp}(c'_i)$, define

$$f'_1(a) := \frac{\sum_{d \in D} c_1(a \oplus_x d)}{|D|}, \text{ and } f'_i(a) := \frac{\sum_{d \in D} \prod_{j=1}^i c_j(a \oplus_x d)}{\sum_{d \in D} \prod_{j=1}^{i-1} c_j(a \oplus_x d)}, \text{ for } i > 1$$

if the denominator is non-zero and $f'_i(a) := 0$ otherwise.

Then $\mathcal{H}(I') = \mathcal{H}(I) \setminus x$, $\|I'\| \leq \|I\|$ and $w(I) = |D|w(I')$. Moreover, one can compute I' with $O(p\|I\|)$ arithmetic operations.

Proof. Observe first that for $a \in \text{supp}(c'_i)$ and $j \leq i$ and $d \in D$, the assignment $a \oplus_x d$ assigns values to all variables of c_j , since $\text{var}(c_j) \subseteq \text{var}(c_i)$ and thus all terms defined above are well-defined.

$\mathcal{H}(I') = \mathcal{H}(I) \setminus x$ is obvious because for every i we have $\text{var}(c'_i) = \text{var}(c_i) \setminus \{x\}$.

Moreover, $\|I'\| \leq \|I\|$ because for every i , $|c'_i| \leq |c_i|$ since $|\text{supp}(c'_i)| = |\{a \in D^{\text{var}(c'_i)} \mid \exists d \in D, (a \oplus_x d) \in \text{supp}(c_i)\}| \leq |\text{supp}(c_i)|$.

Inspired by the discussion at the beginning of this section, we now show by induction on i that for all $a \in D^{\text{var}(c'_i)}$,

$$|D| \prod_{j=1}^i c'_j(a) = \sum_{d \in D} \prod_{j=1}^i c_j(a \oplus_x d).$$

For $i = 1$, let $a \in D^{\text{var}(c'_1)}$. If $a \in \text{supp}(c'_1)$, this statement is clear from the definition.

If $a \notin \text{supp}(c'_1)$, then for all $d \in D$, $(a \oplus_x d) \notin \text{supp}(c_1)$. Thus $c_1(a \oplus_x d) = \mu_1$ for all d and consequently $\sum_{d \in D} c_1(a \oplus_x d) = |D|\mu_1 = |D|c'_1(a)$.

Now suppose that the result holds for i . Let $a \in D^{\text{var}(c'_i)}$. Then we get by induction

$$|D| \prod_{j=1}^{i+1} c'_j(a) = \left(\sum_{d \in D} \prod_{j=1}^i c_j(a \oplus_x d) \right) c'_{i+1}(a).$$

First, assume that $\sum_{d \in D} \prod_{j=1}^i c_j(a \oplus_x d) = 0$. Since this is a sum of non-negative rationals, we have that for all d , $\prod_{j=1}^i c_j(a \oplus_x d) = 0$. Thus, $\prod_{j=1}^{i+1} c_j(a \oplus_x d) = 0$ for all d and it follows that $\sum_{d \in D} \prod_{j=1}^{i+1} c_j(a \oplus_x d) = 0$ which proves the claim.

Now assume that $\sum_{d \in D} \prod_{j=1}^i c_j(a \oplus_x d) \neq 0$. If $a \in \text{supp}(c'_{i+1})$, then by definition of c'_{i+1} , the claim follows directly.

If $a \notin \text{supp}(c_{i+1})$, then $\prod_{j=1}^{i+1} c_j(a \oplus_x d) = \mu_{i+1} \prod_{j=1}^i c_j(a \oplus_x d)$ for all d . Thus

$$\sum_{d \in D} \prod_{j=1}^{i+1} c_j(a \oplus_x d) = \mu_{i+1} \sum_{d \in D} \prod_{j=1}^i c_j(a \oplus_x d) = c'_{i+1}(a) \sum_{d \in D} \prod_{j=1}^i c_j(a \oplus_x d),$$

which establishes the claim for $i + 1$.

Applying the result for $i = p$, we get

$$|D|w(I', a) = |D| \prod_{j=1}^p c'_j(a) = \sum_{d \in D} \prod_{j=1}^p c_j(a \oplus_x d) = w(I, a).$$

It follows directly that $w(I) = |D|w(I')$ as desired.

We now analyze the number of arithmetic operations we make in the construction of I' . Clearly, if we have computed the $\sum_{d \in D} \prod_{j=1}^i c_j(a \oplus_x d)$ for all $i \leq p$ and $a \in \text{supp}(c'_i)$ then we can compute $c'_i(a)$ with one division. Thus we need to do p divisions. Moreover, if we have computed $\prod_{j=1}^i c_j(a \oplus_x d)$, then we only need one more multiplication to compute $\prod_{j=1}^{i+1} c_j(a \oplus_x d)$.

Now, we prove by induction on i that $\prod_{j=1}^i c_j(a \oplus_x d)$ can take at most $1 + \sum_{j=1}^i |c_j|$ different values when a varies. This is trivial for $i = 0$. Now remark that if $a \oplus_x d \notin \text{supp}(c_i)$, then $\prod_{j=1}^i c_j(a \oplus_x d) = \mu_i \prod_{j=1}^{i-1} c_j(a \oplus_x d)$, thus by induction, we have at most $1 + \sum_{j=1}^{i-1} |c_j|$ different values for $\prod_{j=1}^i c_j(a \oplus_x d)$. Moreover, there are at most $|\text{supp}(c_i)| \leq |c_i|$ other values for $a \oplus_x d \in \text{supp}(c_i)$, which completes the induction.

It follows that we have to compute at most $O(p\|I\|)$ different values for the $\prod_{j=1}^i c_j(a \oplus_x d)$ which can be done with $O(p\|I\|)$ multiplications. Now if i is fixed, for all a , the sum $\sum_{d \in D} \prod_{j=1}^i c_j(a \oplus_x d)$ has at most $1 + \sum_{j=1}^i |c_j|$ different terms that are already computed. Thus we only need $O(\|I\|)$ operations to compute each of them. As there are p different sums to compute, we can do everything with $O(p\|I\|)$ arithmetic operations. ◀

3.2 The general case

In this section, we extend Theorem 9 to the case where we have a general nest point in a $\#\text{CSP}_{\text{def}}$ -instance. This will allow us to solve $\#\text{CSP}_{\text{def}}$ for all β -acyclic instances.

In the following, for $x \in \text{var}(I)$, we denote by $I(x) = \{c \in I \mid x \in \text{var}(c)\}$.

► **Theorem 10.** *Let I be a set of weighted constraints on domain D and x a nest point of $\mathcal{H}(I)$. Let $I(x) = \{c_1, \dots, c_p\}$ with $\text{var}(c_1) \subseteq \dots \subseteq \text{var}(c_p)$. Let $I' = \{c' \mid c \in I\}$ where*

- if $c \notin I(x)$ then $c' := c$
- if $c = c_i$, then $c'_i := (f'_i, \mu_i)$ is the weighted constraint on variables $\text{var}(c'_i) = \text{var}(c_i) \setminus \{x\}$, with default value $\mu(c_i)$ and $\text{supp}(c'_i) := \{a \in D^{\text{var}(c'_i)} \mid \exists d \in D, (a \oplus_x d) \in \text{supp}(c_i)\}$. Moreover, for all $a \in \text{supp}(c'_i)$, define

$$f'_1(a) := \frac{\sum_{d \in D} c_1(a \oplus_x d)}{|D|} \text{ and } f'_i(a) := \frac{\sum_{d \in D} \prod_{j=1}^i c_j(a \oplus_x d)}{\sum_{d \in D} \prod_{j=1}^{i-1} c_j(a \oplus_x d)}, \text{ for } i > 1$$

if the denominator is non-zero and $f'_i(a) := 0$ otherwise.

Then $\mathcal{H}(I') = \mathcal{H}(I) \setminus x$, $\|I'\| \leq \|I\|$ and $w(I) = |D|w(I')$. Moreover, one can compute I' with a $O(p\|I(x)\|)$ arithmetic operations.

Proof. Note first that the definition of the c'_i is identical to the construction in Theorem 9.

Let us explain why I' is well-defined. As x is a nest point, we can write $I(x) = \{c_1, \dots, c_p\}$ with $\text{var}(c_1) \subseteq \dots \subseteq \text{var}(c_p)$. If two constraints have the same variables, we choose an arbitrary order for them. Note that in the full version we will choose a specific order that ensures that the algorithm runs in polynomial time on a RAM, but in this proof any order will do.

$\mathcal{H}(I') = \mathcal{H}(I) \setminus x$ and $\|I'\| \leq \|I\|$ is shown as before. Moreover, the bound on the number of arithmetic operations follows from Theorem 9.

We have $w(I) = \sum_{a \in D^{\text{var}(I) \setminus \{x\}}} \sum_{d \in D} \prod_{i=1}^p c_i(a \oplus_x d) \prod_{c \notin I(x)} c(a \oplus_x d)$. For $c \notin I(x)$ we have $c(a \oplus_x d) = c(a) = c'(a)$ by definition. Moreover, we have seen in the proof of Theorem 9 that $\sum_{d \in D} \prod_{i=1}^p c_i(a \oplus_x d) = |D| \prod_{i=1}^p c'_i(a)$. It follows that

$$w(I) = |D| \sum_{a \in D^{\text{var}(I) \setminus \{x\}}} \prod_{i=1}^p c'_i(a) \prod_{c \notin I(x)} c'(a) = |D| w(I').$$

◀

Our algorithm for $\#\text{CSP}_{\text{def}}$ iteratively applies the procedure of Theorem 10 along a β -elimination order. Clearly, this only uses a polynomial number of arithmetic operations, because applying Theorem 10 only needs a polynomial number of arithmetic operations and the size of the instances is decreasing. Note that this does not directly give that the algorithm runs in polynomial time. We iteratively multiply and divide numbers which could lead to an iterative squaring effect and result in numbers of exponential bitsize. However, in the full version of this paper, we show that this actually does not happen: The weights computed in each step are essentially a ratio of weights of two subinstances of the original instance. It follows that the bitsize of the numerator and the denominator of all numbers is polynomially bounded and thus all arithmetic operations can be done in polynomial time. Implementing this yields the following result.

► **Theorem 11.** *There exists an algorithm that, given a β -acyclic instance I of $\#\text{CSP}_{\text{def}}$, computes $w(I)$ in polynomial time.*

Combining Theorem 11 and Corollary 6 we get the main tractability result for $\#\text{SAT}$.

► **Corollary 12.** *$\#\text{SAT}$ on β -acyclic CNF-formulas can be solved in polynomial time.*

4 Relation to the STV-framework

In this section we compare our algorithmic result for $\#\text{SAT}$ on β -acyclic hypergraphs to the framework proposed by Sæther, Telle and Vatshelle in [20] which we call for short the STV-framework. In the full version of this paper we show that the STV-framework gives a uniform explanation of all tractability results for $\#\text{SAT}$ in the literature, extending the results of [20]. We see this as strong evidence that the STV-framework is indeed a good formalization of the intuitive notion of “dynamic programming for $\#\text{SAT}$ ”.

To show that our result is not covered by the STV-framework, we now show that it cannot give any subexponential time algorithms for β -acyclic $\#\text{SAT}$. To this end, we prove an exponential lower bound on the PS-width of β -acyclic CNF-formulas.

We start off with a simple lemma.

We remind the reader that a CNF-formula F is called *monotone* if all variables appear only positively in F .

► **Lemma 13.** *For every bipartite graph G there is a monotone CNF-formula F such that F has the incidence graph G and $\text{psw}(F) \geq 2^{\text{mimw}(G)/2}$.*

Proof. We construct F by choosing arbitrarily one color class of G to represent clauses and the other one to represent variables. This choice then uniquely yields a monotone formula where a clause C contains a variable x if and only if x is connected to C by an edge in G .

Let (T, δ) be a branch decomposition of G and F . Let t be a vertex of T with cut (A, \bar{A}) . Set $X := \text{var}(F) \cap A$, $\bar{X} := \text{var}(F) \cap \bar{A}$, $\mathcal{C} := \text{cla}(F) \cap A$ and $\bar{\mathcal{C}} := \text{cla}(F) \cap \bar{A}$. Moreover, let M be a maximum induced matching of $G[A, \bar{A}]$ and let V_M be the end vertices of M .

First assume that $|\mathcal{C} \cap V_M| \geq |\bar{\mathcal{C}} \cap V_M|$. Let C_1, \dots, C_k be the clauses in $\mathcal{C} \cap V_M$ and let x_1, \dots, x_k be variables in $\bar{X} \cap V_M$. Note that $k \geq |M|/2$. Since M is an induced matching, every clause C_i contains exactly one of the variables x_j , and we assume w.l.o.g. that C_i contains x_i . Let a be an assignment to the x_i and let a' be the extended assignment of \bar{X} that we get by assigning 0 to all other variables. Then a' satisfies in $F_{\bar{X}, \mathcal{C}}$ exactly the clauses C_i for which $a(x_i) = 1$ since the formula is monotone. Since there are 2^k assignments to the x_i , we have $|PS(F_{\bar{X}, \mathcal{C}})| \geq 2^k \geq 2^{|M|/2}$.

For $|\mathcal{C} \cap V_M| \leq |\bar{\mathcal{C}} \cap V_M|$ it follows symmetrically that $|PS(F_{X, \bar{\mathcal{C}}})| \geq 2^{|M|/2}$. Consequently, we have in either case that the PS-width of F is at least $2^{|M|/2}$ and the claim follows. ◀

We will now define for every graph G a graph G' . The construction will be such that, if G is chosen in the right way, then G' will be chordal bipartite and of high MIM-width. Combining this with Lemma 13 yields a class of β -acyclic CNF-formulas of high PS-width. Since PS-width is the crucial parameter in the STV-framework, this shows that β -acyclic #SAT cannot be solved efficiently by this framework.

Given a graph $G = (V, E)$ we define $G' = (V', E')$ as follows:

- for every $v \in V$ there are two vertices $x_v, y_v \in V'$,
 - for every edge $e = uv \in E$ there are four vertices $p_{e,u}, q_{e,u}, p_{e,v}, q_{e,v} \in V'$,
 - every $u, v \in V$ we add the edge $x_v y_u$ to E' , and
 - for every edge $e = uv \in E$ we add the edges $p_{e,u} q_{e,u}, p_{e,v} q_{e,v}, x_u p_{e,u}, y_v q_{e,u}, x_v p_{e,v}, y_u q_{e,v}$.
- These are all vertices and edges of G' .

► **Lemma 14.** *G' is chordal bipartite.*

► **Lemma 15.** *Let G be bipartite. Then $\text{tw}(G) \leq 6\text{mimw}(G')$.*

Proof. Let (T', δ') be a branch decomposition of G' . Let $A, B \subseteq V(G)$ be the two colour classes of G . We construct a branch decomposition (T, δ) of G by deleting the leaves labeled with $p_{e,u}, q_{e,u}, p_{e,v}, q_{e,v}$, and those labeled x_v for $v \in A$ or with y_v for $v \in B$. Then we delete all internal vertices of T' that have become leaves by these deletions until we get a branch decomposition T with the leaves x_v for $v \in B$ and y_v for $v \in A$. For the leaves of T we define $\delta(t) := v$ where $v \in V$ is such that $\delta'(t) = x_v$ or $\delta'(t) = y_v$. The result (T, δ) is a branch decomposition of G .

Let t be a vertex of T with the corresponding cut (X, \bar{X}) . Let $M \subseteq E$ be a matching in $G[X, \bar{X}]$. Let (X', \bar{X}') be the cut of t in (T', δ') . Let $e = uv \in M$, then x_u and y_v are on different sides of the cut X' and they are connected by the path $x_u p_{e,u} q_{e,u} y_v$. Consequently, there is at least one edge along this path in $G'[X', \bar{X}']$. Choose one such edge arbitrarily.

Let M' be the set of edges we have chosen for the different edges in M . Let M'_x be the set of edges in M' that do not have an end vertex y_v and let M'_y be the set of edges in M' that do not have an end vertex x_v . Let M'' be the bigger of these two sets. Since $e' \in M'$ can only have an end vertex x_v or y_u but not both, we have $|M'_x| + |M'_y| \geq |M'|$ and thus $|M''| \geq |M'|/2$.

We claim that M'' is an induced matching in G' . Clearly, M' is a matching because M is one. Consequently, $M'' \subseteq M'$ is also a matching. We now show that M'' is also induced. By way of contradiction, assume this were not true. Then there must be two adjacent vertices $u, v \in V'$ that are end vertices of edges in M'' but not in the same edge in M'' . If $u = p_{e',w}$ for some $e' \in E$ and $w \in V$, then v must be x_w . But then by construction of M' , the vertex

w must be incident to two edges in M which contradicts M being a matching. Similarly, we can rule out that v is $q_{e,w}$. Thus, u must be x_w or y_w and v must be $x_{w'}$ or $y_{w'}$. Since x_w and $x_{w'}$ are in the same colour class of G' , they are not adjacent. Similarly y_w and $y_{w'}$ are not adjacent. Consequently, we may assume that $u = x_w$ and $v = y_{w'}$. But then they cannot both be an endpoint of an edge in M'' by construction of M'' . Thus M'' is induced.

By Lemma 7 we know that there is a $t \in T$ with cut (X, \bar{X}) such that we can find a matching M of size at least $\frac{\mathbf{tw}(G)}{3}$ in $G[X, \bar{X}]$. By the construction above the corresponding cut (X', \bar{X}') yields an induced matching of size $\frac{\mathbf{tw}(G)}{6}$ in $G'[X', \bar{X}']$. This completes the proof. \blacktriangleleft

The connection between expansion and treewidth (see [16]) yields the following lemma.

► **Lemma 16.** *There is a family \mathcal{G} of graphs and constants $c > 0$ and $d \in \mathbb{N}$ such that for every $G \in \mathcal{G}$ the graph G has maximum degree d and we have $\mathbf{tw}(G) \geq c|E(G)|$.*

► **Corollary 17.** *There is a family \mathcal{G}' of chordal bipartite graphs and a constant c such that for every graph $G \in \mathcal{G}'$ we have $\mathbf{mimw}(G) \geq c|V(G)|$.*

Proof. Let \mathcal{G} be the class of Lemma 16. We first transform every graph $G \in \mathcal{G}$ into a bipartite one G_1 by subdividing every edge, i.e., by introducing for each edge $e = uv$ a new vertex w_e and by replacing e by uw_e and w_ev . It is well-known that subdividing edges does not decrease the treewidth of a graph (see e.g. [10]), and thus $\mathbf{tw}(G) \leq \mathbf{tw}(G_1)$. Moreover, $|E(G_1)| = 2|E(G)|$, and thus $\mathbf{tw}(G_1) \geq \frac{1}{2}c|E(G_1)|$. Now let $\mathcal{G}' = \{G_1 \mid G \in \mathcal{G}\}$. Then the graphs in \mathcal{G}' are chordal bipartite by Lemma 14 and the bound on the MIM-width follows by combining Lemma 16 and Lemma 15. \blacktriangleleft

Combining Corollary 17 and Lemma 13 yields the main result of this section.

► **Corollary 18.** *There is a family of monotone β -acyclic CNF-formulas of PS-width $2^{\Omega(n)}$ where n is the number of variables in the formulas.*

Since the runtime in Theorem 8 depends linearly on the PS-width, we get that the STV-framework cannot prove subexponential runtime bounds for $\#\text{SAT}$ on β -acyclic formulas.

5 Conclusion

We have shown that β -acyclic $\#\text{SAT}$ can be solved in polynomial time, a question left open in [7]. Our algorithm does not follow the dynamic programming approach that was used in all other structural tractability results that were known before, and as we have seen this is no coincidence. Instead, β -acyclic $\#\text{SAT}$ lies outside the STV-framework of [20] that explains all earlier results in a uniform way.

We close this paper with several open problems that we feel should be explored in the future. First, our algorithm for $\#\text{SAT}$ is specifically designed for the case of β -acyclic formulas, but we feel that the techniques developed might possibly be extended to other classes of hypergraphs that one can characterize by elimination orders. In this direction, it would be interesting to see if hypergraphs of bounded β -hypertree width, a width measure generalizing β -acyclicity proposed in [15], can be characterized by elimination orders and if such a characterization can be used to solve $\#\text{SAT}$ on the respective instances. Note that this case lies outside of the STV-framework, therefore dynamic programming without new ingredients is unlikely to work. Also, even the complexity of deciding SAT on instances of bounded β -hypertree width is an open problem [17].

It might also be interesting to generalize our algorithm to solve cases for which we already have polynomial time algorithms. For example, is there any uniform explanation for tractability of bounded cliquewidth $\#SAT$ and β -acyclic $\#SAT$, similarly to the way in which the framework of [20] explains tractability for all previously known results?

Finally, we feel that, although we have shown that the STV-framework does not explain all tractability results for $\#SAT$, it is still a framework that should be studied in the future. We believe that there are still many classes to be captured by it and thus we see a better understanding of the framework as an important goal for future research. One question is the complexity of computing branch decompositions of (approximately) minimal MIM-width or PS-width. Alternatively, one could try to find more classes of bipartite graphs for which one can efficiently compute branch decompositions of small MIM-width. This would then directly extend the knowledge on structural classes of CNF-formulas for which dynamic programming can efficiently solve $\#SAT$.

References

- 1 G. Ausiello, A. D'Atri, and M. Moscarini. Chordality properties on graphs and minimal conceptual connections in semantic data models. *J. Comput. Syst. Sci.*, 33(2):179–202, 1986.
- 2 H.L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1-2):1–21, 1993.
- 3 A. Brandstädt, V.B. Le, and J.P. Spinrad. *Graph Classes: A Survey*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.
- 4 J. Brault-Baron. Hypergraph Acyclicity Revisited. *ArXiv e-prints*, March 2014.
- 5 A. Bulatov, M. Dyer, L.A. Goldberg, M. Jalsenius, M. Jerrum, and D. Richerby. The complexity of weighted and unweighted $\#CSP$. *Journal of Computer and System Sciences*, 78(2):681–688, March 2012.
- 6 J.-Y. Cai and X. Chen. Complexity of counting CSP with complex weights. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, page 909–920, New York, NY, USA, 2012. ACM.
- 7 F. Capelli, A. Durand, and S. Mengel. Hypergraph Acyclicity and Propositional Model Counting. In *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference*, pages 399–414, 2014.
- 8 D.A. Cohen, M.J. Green, and C. Houghton. Constraint representations and structural tractability. In *Principles and Practice of Constraint Programming - CP 2009*, pages 289–303, 2009.
- 9 V. Dalmau and P. Jonsson. The complexity of counting homomorphisms seen from the other side. *Theor. Comput. Sci.*, 329(1-3):315–323, 2004.
- 10 Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- 11 D. Duris. Some characterizations of γ and β -acyclicity of hypergraphs. *Inf. Process. Lett.*, 112(16):617–620, 2012.
- 12 R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30(3):514–550, 1983.
- 13 E. Fischer, J.A. Makowsky, and E.V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511–529, 2008.
- 14 G. Gottlob, N. Leone, and F. Scarcello. A comparison of structural CSP decomposition methods. *Artif. Intell.*, 124(2):243–282, 2000.
- 15 G. Gottlob and R. Pichler. Hypergraphs in Model Checking: Acyclicity and Hypertree-Width versus Clique-Width. *SIAM Journal on Computing*, 33(2), 2004.

- 16 M. Grohe and D. Marx. On tree width, bramble size, and expansion. *J. Comb. Theory, Ser. B*, 99(1):218–228, 2009.
- 17 S. Ordyniak, D. Paulusma, and S. Szeider. Satisfiability of acyclic and almost acyclic CNF formulas. *Theoretical Computer Science*, 481:85–99, 2013.
- 18 D. Paulusma, F. Slivovsky, and S. Szeider. Model Counting for CNF Formulas of Bounded Modular Treewidth. In *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013*, pages 55–66, 2013.
- 19 D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1–2):273 – 302, 1996.
- 20 S. Hortemo Sæther, J.A. Telle, and M. Vatshelle. Solving MaxSAT and #SAT on structured CNF formulas. In *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference*, pages 16–31, 2014.
- 21 M. Samer and S. Szeider. Algorithms for propositional model counting. *Journal of Discrete Algorithms*, 8(1):50–64, 2010.
- 22 F. Slivovsky and S. Szeider. Model Counting for Formulas of Bounded Clique-Width. In *Algorithms and Computation - 24th International Symposium, ISAAC 2013*, pages 677–687, 2013.
- 23 M. Vatshelle. *New Width Parameters of Graphs*. PhD thesis, University of Bergen, 2012.

Parameterized Complexity Dichotomy for Steiner Multicut*

Karl Bringmann¹, Danny Hermelin², Matthias Mnich³, and Erik Jan van Leeuwen⁴

- 1 Institute of Theoretical Computer Science, ETH Zurich, Zurich, Switzerland
karlb@inf.ethz.ch
- 2 Ben Gurion University of the Negev, Israel
hermelin@bgu.ac.il
- 3 Universität Bonn, Bonn, Germany
mmnich@uni-bonn.de
- 4 Max Planck Institut für Informatik, Saarbrücken, Germany
erikjan@mpi-inf.mpg.de

Abstract

We consider the STEINER MULTICUT problem, which asks, given an undirected graph G , a collection $\mathcal{T} = \{T_1, \dots, T_t\}$, $T_i \subseteq V(G)$, of terminal sets of size at most p , and an integer k , whether there is a set S of at most k edges or nodes such that of each set T_i at least one pair of terminals is in different connected components of $G \setminus S$. This problem generalizes several well-studied graph cut problems, in particular the MULTICUT problem, which corresponds to the case $p = 2$. The MULTICUT problem was recently shown to be fixed-parameter tractable for parameter k [Marx and Razgon, Bousquet et al., STOC 2011]. The question whether this result generalizes to STEINER MULTICUT motivates the present work.

We answer the question that motivated this work, and in fact provide a dichotomy of the parameterized complexity of STEINER MULTICUT on general graphs. That is, for any combination of k , t , p , and the treewidth $\text{tw}(G)$ as constant, parameter, or unbounded, and for all versions of the problem (edge deletion and node deletion with and without deletable terminals), we prove either that the problem is fixed-parameter tractable or that the problem is hard (W[1]-hard or even (para-)NP-complete). Among the many results in the paper, we highlight that:

- The edge deletion version of STEINER MULTICUT is fixed-parameter tractable for parameter $k + t$ on general graphs (but has no polynomial kernel, even on trees).
- In contrast, both node deletion versions of STEINER MULTICUT are W[1]-hard for the parameter $k + t$ on general graphs.
- All versions of STEINER MULTICUT are W[1]-hard for the parameter k , even when $p = 3$ and the graph is a tree plus one node.

Since we allow k , t , p , and $\text{tw}(G)$ to be any constants, our characterization includes a dichotomy for STEINER MULTICUT on trees (for $\text{tw}(G) = 1$) as well as a polynomial time versus NP-hardness dichotomy (by restricting $k, t, p, \text{tw}(G)$ to constant or unbounded).

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases graph cut problems, Steiner cut, fixed-parameter tractability

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.157

* K. Bringmann is supported by the ETH Zurich Postdoctoral Fellowship Program. D. Hermelin has received funding from the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement number 631163.11, and by the ISRAEL SCIENCE FOUNDATION (grant No. 551145/14).



© Karl Bringmann, Danny Hermelin,
Matthias Mnich, and Erik Jan van Leeuwen;
licensed under Creative Commons License CC-BY

32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).

Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 157–170

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



1 Introduction

Graph cut problems are among the most fundamental problems in algorithmic research. The classic result in this area is the polynomial-time algorithm for the s - t cut problem of Ford and Fulkerson [19] (independently proven by Elias et al. [17] and Dantzig and Fulkerson [13]). This result inspired a research program to discover the computational complexity of this problem and of more general graph cut problems. One well-studied generalization of the s - t cut problem is the MULTICUT problem, in which we want to disconnect t pairs of nodes instead of just one pair. In a recent major advance of the research program on graph cut problems, Bousquet et al. [3] and Marx and Razgon [32] showed that MULTICUT is fixed-parameter tractable in the size k of the cut only, meaning that it has an algorithm running in time $f(k) \cdot \text{poly}(|V(G)|)$ for some function f , resolving a longstanding problem in parameterized complexity (with many papers [30, 33, 21, 31] building up to this result).

In this paper, we continue the research program on generalized graph cut problems, and consider the STEINER MULTICUT problem. This problem was proposed by Klein et al. [25], and appears in several versions, depending on whether we want to delete edges or nodes, and whether we are allowed to delete terminal nodes. Formally, these versions of the STEINER MULTICUT problem are defined as follows:

{EDGE, NODE, RESTR. NODE} STEINER MULTICUT

Input: An undirected graph G with terminal sets $T_1, \dots, T_t \subseteq V(G)$, and integer $k \in \mathbb{N}$.

Task: Find a set S of k {edges, nodes, non-terminal nodes} such that for $i = 1, \dots, t$ and at least one pair $u, v \in T_i$ there is no $u - v$ path in $G \setminus S$.

Observe that MULTICUT is the special case of STEINER MULTICUT in which each terminal set has size two. In general, the terminal sets of STEINER MULTICUT can have arbitrary size, and we use p to denote $\max_i |T_i|$.

The complexity of STEINER MULTICUT has been investigated extensively, but so far only from the perspective of approximability. This line of work was initiated by Klein et al. [25], who gave an LP-based $O(\log^3(kp))$ -approximation algorithm. The approximability of several variations of the problem has also been considered [35, 20, 2]; in particular, Garg et al. [20] give an $O(\log t)$ -approximation algorithm for MULTICUT. On the hardness side, even MULTICUT is APX-hard [12, 4] and cannot be approximated within any constant factor assuming the Unique Games Conjecture [6]. We also remark that Steiner cuts (the case when $t = 1$) are of interest: they are an ingredient in several LP-based approximation algorithms (for example for STEINER FOREST [1, 26]) and there is a connection to the number of edge-disjoint Steiner trees that each connect all terminals [29]. To the best of our knowledge, however, STEINER MULTICUT in its general form has not yet been considered from the perspective of parameterized complexity.

1.1 Our Contributions

In this paper, we fully chart the (parameterized) complexity landscape of STEINER MULTICUT according to k , t , p (defined as above), and the treewidth $\text{tw}(G)$. For all three versions of STEINER MULTICUT, for each possible combination of k , t , p , and $\text{tw}(G)$, where each may be either chosen as a constant, a parameter, or unbounded, we consider the complexity of STEINER MULTICUT. We show a complete dichotomy: either we provide a fixed-parameter algorithm with respect to the chosen parameters, or we prove a W[1]-hardness or (para-) NP-completeness result that rules out a fixed-parameter algorithm (unless many canonical NP-complete problems have subexponential- or polynomial-time algorithms respectively).

■ **Table 1** Summary of known and new complexity results for STEINER MULTICUT, where new results are marked with *; the other entries are either known or follow easily from known results in the literature. Only maximal FPT results and minimal $W[\cdot]$ - or NP-hardness results are listed; empty cells are dominated by other results. E.g. EDGE STEINER MULTICUT with parameter t is hard, since it is already NP-hard for $t = 3, p = 2$. For NODE STEINER MULTICUT, one also has to apply the rule that $k < t$ (see Section 2) to generate a full characterization of all cases. Tree diagrams of this table are offered in the full version of this paper.

constants	params	EDGE STEINER MC	NODE STEINER MC	RESTR. NODE STEINER MC
k	—	poly (Sect. 2)	poly (Sect. 2)	poly (Sect. 2)
$t \leq 2$	—	poly (Sect. 2)		poly (Sect. 2)
$t = 3, p = 2$	—	NP-h [12]		NP-h [12]
—	k, t	*FPT (Thm. 1)	*W[1]-h (Thm. 2)	*W[1]-h (Thm. 2)
—	k, p, t		FPT (Sect. 2)	FPT (Sect. 2)
t	k			FPT (Sect. 2)
$p = 2$	k	FPT [3, 32]	FPT [3, 32]	FPT [3, 32]
$p = 3, \text{tw} = 2$	k	*W[1]-h (Thm. 3)	*W[1]-h (Thm. 3)	*W[1]-h (Thm. 3)
—	t, tw	*FPT (Thm. 13)	*FPT (Thm. 13)	*FPT (Thm. 13)
$\text{tw} = 1$	—		*poly (Thm. 15)	
$\text{tw} = 1$	k	*W[2]-h (Thm. 16)		*W[2]-h (Thm. 16)
$\text{tw} = 1$	k, p	*FPT (Thm. 18)		*FPT (Thm. 18)
$\text{tw} = 1, p = 2$	—	NP-h [4]		NP-h [4]
$\text{tw} = 2, p = 2$	—		NP-h [4]	

The dichotomy is composed of three main results, along with many smaller ones (see Table 1). These three main results are stated in the three theorems below:

► **Theorem 1.** EDGE STEINER MULTICUT is fixed-parameter tractable for parameter $k + t$.

► **Theorem 2.** NODE STEINER MULTICUT and RESTR. NODE STEINER MULTICUT are $W[1]$ -hard for the parameter $k + t$.

► **Theorem 3.** NODE STEINER MULTICUT, EDGE STEINER MULTICUT, and RESTR. NODE STEINER MULTICUT are $W[1]$ -hard for the parameter k , even if $p = 3$ and $\text{tw}(G) = 2$.

Observe the sharp gap described by Theorem 1 and Theorem 2 between the parameterized complexity of the edge deletion version versus the node deletion version; this gap does not exist for the MULTICUT problem. Also note that Theorem 3 implies that the fixed-parameter algorithms for MULTICUT for parameter k [3, 32] do not generalize to STEINER MULTICUT.

To obtain the fixed-parameter algorithm of Theorem 1, we have to avoid the brute-force choice of a pair of separated terminals of each terminal set: Although one can trivially reduce every instance of the EDGE STEINER MULTICUT problem to at most $\binom{p}{2}^t$ instances of MULTICUT parameterized by k , this only yields an $f(k) \cdot n^{O(t)}$ -time algorithm (for unbounded p). Our contribution in Theorem 1 is that we improve on this simple algorithm and obtain a runtime of $f(k, t) \cdot n^{O(1)}$. We give two independent proofs of Theorem 1:

- Our first proof uses a variation of the recent technique of Chitnis et al. [8] known as randomized contractions (even though the technique actually yields deterministic algorithms). The rough idea of the algorithm is to first determine a large subgraph G'

of the input graph G , such that G' has no small cut and only has a small interface (i.e. a small number of vertices that connect the subgraph to the rest of the graph). We can then branch on the behavior of a solution on the interface to determine a set $U \subseteq E(G')$ of ‘useless’ edges, in the sense that when U is contracted in G a smallest solution (of size at most k) persists in the remaining graph. By iterating this procedure, we can reduce the size of the graph until it is small enough to be handled by exhaustive enumeration. Our algorithm is similar to the one for EDGE MULTIWAY CUT-UNCUT in the paper by Chitnis et al. [8]; however, in contrast to that problem, there seems to be no straightforward projection of the instance onto G' in our case, and therefore more involved arguments are needed to determine the set U .

- Our second proof (presented only in the full version) is based on several novel structural lemmas that show that a minimal edge Steiner cut can be decomposed into important separators and minimal s - t cuts. Using a branching strategy, we ascertain the topology of the decomposition that is promised by the structural lemmas. Since there are only few important separators of bounded size [30, 7, 32] and all relevant minimal s - t cuts lie in a graph of bounded treewidth (following the treewidth reduction techniques of Marx et al. [31]), we can then optimize over important separators and minimal s - t cuts.

The advantage of the first algorithm over the second is that it runs in single-exponential time, instead of double-exponential time. However, the second algorithm is slightly faster in terms of n . Moreover, as part of the correctness proof of the second algorithm, we present some structural lemmas that give additional insight into the properties of the cut, which may be of independent interest. Therefore, we present both algorithms in the full version, but only show the first algorithm in this extended abstract.

The $W[1]$ -hardness results of Theorem 2 and 3 all rely on reductions from the MULTICOLORED CLIQUE problem [18]. For the proof of Theorem 3, we introduce a novel intermediate problem, NAE-INTEGGER-3-SAT, which is an integer variant of the better known NOT-ALL-EQUAL-3-SAT problem. We show that NAE-INTEGGER-3-SAT is $W[1]$ -hard parameterized by the number of variables. This is a powerful starting point for parameterized hardness reductions and should turn out to be useful to prove the hardness of other problems.

To complete our dichotomy, we chart the full (parameterized) complexity of STEINER MULTICUT on trees, that is, for graphs G with $\text{tw}(G) = 1$ (these results are mostly deferred to the full version of this paper). In fact, some of the hardness results that we prove for STEINER MULTICUT on general graphs even hold for trees. We also show that many of the results for trees do not carry over to graphs of bounded treewidth, the only exception being a fixed-parameter algorithm for parameters $\text{tw}(G) + t$.

We remark that our characterization induces a polynomial time vs. NP-hardness dichotomy for STEINER MULTICUT, i.e., for any choice of $k, p, t, \text{tw}(G)$ as any constants or unbounded (and all three problem variants), we either prove that STEINER MULTICUT is in \mathbf{P} or that it is NP-hard. This characterization can be obtained from Table 1 by considering all its polynomial time and NP-hardness results as well as using the rule that any fixed-parameter algorithm induces a polynomial-time algorithm by setting all parameters to $O(1)$.

1.2 Related Work

We already mentioned several results on the special case of STEINER MULTICUT when $p = 2$ (MULTICUT) [30, 33, 21, 3, 32, 31]. MULTICUT is itself a generalization of MULTIWAY CUT, also known as MULTITERMINAL CUT, where the goal is to delete k edges or nodes to separate all terminals from each other. This problem is NP-complete even for three terminals [12]

and has been extensively studied from a parameterized point of view (see, e.g., the work of Cao et al. [5] or Cygan et al. [11]). The parameterized complexity of many different other graph cut problems has also been considered in recent years [16, 8, 24, 30]. On trees, we only mention here that EDGE MULTICUT and RESTR. NODE MULTICUT remain NP-hard [4], but are fixed-parameter tractable [23, 22]. In contrast, NODE MULTICUT has a polynomial-time algorithm on trees [4].

Organization. We begin our exposition in Section 2 by giving easy results for certain parameter combinations of STEINER MULTICUT. Thereafter, we present our fixed-parameter algorithm for EDGE STEINER MULTICUT (Theorem 1) in Section 3. Following this, in the two subsequent sections, we present our $W[1]$ -hardness proofs: the proof of Theorem 3 in Section 4, and of Theorem 2 in Section 5. Section 6 then focuses on trees to complete our dichotomy. We conclude with some discussion and open problems in Section 7. The full version of this paper is included as an appendix; there we also define basic notions of parameterized complexity.

2 Easy and Known Results

In this section, we collect easy and known results about the STEINER MULTICUT problem. Some of these results are scattered throughout the literature, while others are new. First, observe that whenever the cut size k is constant, we can solve the problem in polynomial time by simply guessing the desired set S of at most k edges or nodes.

Furthermore, NODE STEINER MULTICUT is trivially solvable when $t \leq k$, as in this case we may simply delete an arbitrary terminal node from each set T_i , resulting in a solution of size at most k ; thus, any instance is always a “yes”-instance in this case.

We may reduce STEINER MULTICUT to $\binom{p}{2}^t$ instances of MULTICUT by branching for each terminal set over its separated terminals. Since MULTICUT is in FPT for parameter k , we obtain a fixed-parameter algorithm for STEINER MULTICUT for parameter $k + t + p$. Also, since $\binom{p}{2}^t \leq n^{O(t)}$, STEINER MULTICUT is in FPT for parameter k and any constant t .

Now, MULTICUT on instances with $t = 1$ (i.e. instances that have only one terminal pair $|T_1| = \{s, t\}$) is polynomial-time solvable by running an $s - t$ cut algorithm. For $t = 2$ a result by Yannakakis et al. [34, Lemma 1] also yields a polynomial time algorithm for MULTICUT. Again by branching over the separated terminals in both terminal sets, we obtain a polynomial time algorithm for STEINER MULTICUT for $t \leq 2$.

When there are three or more terminal sets, then STEINER MULTICUT generalizes MULTIWAY CUT and thus is NP-complete [12] even when $p = 2$.

We next show that RESTR. NODE STEINER MULTICUT is as least as hard as NODE STEINER MULTICUT. Therefore, whenever NODE STEINER MULTICUT is $W[1]$ -hard (or NP-hard) for a certain combination of parameters, then so is RESTR. NODE STEINER MULTICUT.

► **Lemma 4.** *Any instance of NODE STEINER MULTICUT can be reduced in polynomial time to an instance of RESTR. NODE STEINER MULTICUT with the same parameter values k , t , p , and $\text{tw}(G)$.*

Proof. Take an instance (G, \mathcal{T}, k) , $\mathcal{T} = \{T_1, \dots, T_t\}$, of NODE STEINER MULTICUT and transform it to an instance of RESTR. NODE STEINER MULTICUT by adding for each terminal node $v \in T_1 \cup \dots \cup T_t$ a new pendant node v' . Then replace v by v' in every terminal set T_i . It is easy to see that the original instance admits a node cut of size k if and only if the new instance admits a node cut of size k that does not use any terminal nodes. ◀

3 Tractability for Edge Deletion and Parameter $k + t$

In this section, we prove Theorem 1, namely that EDGE STEINER MULTICUT parameterized by $k + t$ is fixed-parameter tractable. The proof uses the technique of randomized contractions pioneered by Chitnis et al. [8]; a second, independent proof is deferred to the full version. Later we will see that Theorem 1 is “maximal”, in the sense that EDGE STEINER MULTICUT is $W[1]$ -hard parameterized by k or t alone (this follows from Theorem 16 and the fact that even EDGE MULTICUT is NP-hard when $t = 3$ [12] respectively), that the corresponding node deletion problem is $W[1]$ -hard parameterized by $k + t$ (Theorem 2), and that there exists no polynomial kernel for EDGE STEINER MULTICUT parameterized by $k + t$ (Theorem 17).

We first state some notions and supporting lemmas from the paper of Chitnis et al. [8], which are needed to make our proof work. The *identification* of two vertices $v, w \in V(G)$ results in a graph G' by removing v, w , adding a new vertex vw , and if v or w is an endpoint of an edge, then we replace this endpoint by vw . Note that the identification of two vertices does not remove any edges, and generally results in a multigraph (with parallel edges and self-loops). Without confusion, we may sometimes refer to vw by its old names v or w .

The *contraction* of an edge $(v, w) \in E(G)$ results in a graph G' by removing all edges between v and w , and then identifying v and w . This is also known as contraction without removing parallel edges. Again, the result of a contraction is generally a multigraph. Given a set $F \subseteq E(G)$ of edges that induce a connected subgraph of G with $a + 1$ vertices of which v is one, after contracting all edges of F , we say that a vertices were contracted *onto* v .

► **Definition 5** ([8]). Given two integers a, b , an (a, b) -good edge separation of a connected graph G is a partition (V_1, V_2) of $V(G)$ such that $|V_1|, |V_2| > a$, $G[V_1]$ and $G[V_2]$ are connected, and the number of edges between V_1 and V_2 is at most b .

We now define several notions and prove a few lemmas that are implicit in the work of Chitnis et al. [8].

► **Definition 6.** A b -bordered subgraph of G is a connected induced subgraph G' of G such that in G at most b vertices of $V(G')$ have an edge to a vertex of $V(G) \setminus V(G')$. We call the vertices of G' that have an edge in G to a vertex of $V(G) \setminus V(G')$ the *border vertices* of G' .

► **Lemma 7.** Given a connected graph G and two integers a, b (b even), one can find in time $2^{O(\min\{a, b\} \log(a+b))} |V(G)|^4 \log |V(G)|$ a b -bordered subgraph of G that does not admit an $(a, b/2)$ -good edge separation.

Let G be a connected graph and let a be an integer. Given a set $F \subseteq E(G)$, let G_F denote the graph obtained from G by contracting all edges of F , and then identifying into a single vertex (which we denote by h_F) all vertices onto which at least a vertices were contracted. Observe that G_F is potentially a multigraph, and that h_F might not exist.

A subset Y of the edges of a connected graph G is a *separator* if $G \setminus Y$ has more than one connected component. The set Y is a *minimal separator* if there is no $Y' \subset Y$ such that $G \setminus Y'$ has the same connected components as $G \setminus Y$.

► **Lemma 8** ([8]). Let G be a connected graph, let a, b be two integers (b even), and let $F \subseteq E(G)$ with $|F| \leq b/2$. If G admits no $(a, b/2)$ -good edge separation, then $G \setminus F$ has at most $(b/2) + 1$ connected components, of which at most one has more than a vertices.

► **Lemma 9.** Let G be a connected graph, let a, b be any two integers (b even) such that G does not admit an $(a, b/2)$ -good edge separation and such that $|V(G)| > a(b/2 + 1)$, and let $Y \subseteq E(G)$ with $|Y| \leq b/2$ be a minimal separator. In time $2^{O(b \log(a+b))} |E(G)| \log |E(G)|$

one can find a family \mathcal{F} of $2^{O(b \log(a+b))} \log |E(G)|$ subsets of $E(G)$ that contains a set $F_0 \subseteq E(G)$ with the following properties: (1) $F_0 \cap Y = \emptyset$, (2) h_{F_0} exists in G_{F_0} , (3) h_{F_0} is the identification of a subset of the vertices of a connected component of $G \setminus Y$, and (4) for each connected component C of $G_{F_0} \setminus \{h_{F_0}\}$, Y either contains all edges of $G_{F_0}[C \cup \{h_{F_0}\}]$ or none of these edges.

It is important to observe that the construction of the family \mathcal{F} does not require knowledge of Y itself, beyond that it has size at most $b/2$. Moreover, note that all edges of Y are present in G_{F_0} , as $F_0 \cap Y = \emptyset$.

We are now ready to describe the algorithm for EDGE STEINER MULTICUT for the parameter $k + t$. The basic intuition is to find a part of the graph that does not have a (q, k) -good edge separation for some q , but that only has a small number of border vertices. In this part of the graph we find and contract a set of edges that are provably not part of some smallest edge Steiner multicut. We repeat this procedure until the graph is small enough to be handled by an exhaustive enumeration algorithm.

Consider an instance (G, \mathcal{T}, k) with $\mathcal{T} = \{T_1, \dots, T_t\}$ of EDGE STEINER MULTICUT. We may assume that G is connected. Let q be an integer determined later (q will depend on k and t only). We assume that $|E(G)| > q$, or we can use exhaustive enumeration to solve the problem in $tq^{O(k)}$ time.

We apply the algorithm of Lemma 7 to find a $2k$ -bordered subgraph G' of G that does not admit a (q, k) -good edge separation. Let B denote the set of border vertices of G' . Note that possibly $G' = G$, in which case $B = \emptyset$. The idea is now to determine a set of edges of G' that is not used by some optimal solution.

Let S be a smallest edge Steiner multicut of (G, \mathcal{T}) . Let G'' denote the graph $(B \cup (V(G) \setminus V(G')), E(G) \setminus E(G'))$. Observe that $E(G')$ and $E(G'')$ partition $E(G)$. Let $S' = S \cap E(G')$ and let $S'' = S \cap E(G'')$. We call a terminal set *active* if it is not separated in $G \setminus S'$. We call border vertices $b, b' \in B$ *paired* if there is a path between b and b' in $G'' \setminus S''$. Note that this defines an equivalence relation on B . We call an equivalence class B' of this relation *i -active* if the terminal set T_i is active and the component of $G'' \setminus S''$ that contains B' contains a terminal of T_i . Intuitively, this information suffices to compute a set $Z \subseteq E(G')$ such that $Z \cup S''$ is a smallest edge Steiner multicut of (G, \mathcal{T}) . Then we could contract (in G) all other edges of G' to get a smaller instance, and repeat this until the instance is small enough to be solved by exhaustive enumeration.

Of course, we do not know S , and thus we do not know this equivalence relation on B nor which classes are i -active for each $i = 1, \dots, t$. However, we can branch over all possibilities. In each branch, we find a small set of edges, which we mark. At the end, we contract (in G) the set of edges of G' that were not marked, and thus reduce the size of the instance. We will prove that in one of the branches, we mark a smallest set Z of edges (of size at most k) such that $(Z \cup S'')$ is a smallest edge Steiner multicut (of size at most k) of (G, \mathcal{T}) . Therefore, after contraction, a smallest edge Steiner multicut (of size at most k) of (G, \mathcal{T}) persists (if such a cut existed in the first place). In particular, we argue that we mark Z in the branch with the active classes \mathcal{T}^S , the equivalence relation \mathcal{B}^S , and i -active classes \mathcal{B}_i^S of \mathcal{B}^S for $i = 1, \dots, |\mathcal{T}^S|$ that are induced by S .

The algorithm proceeds by branching over all possibilities. Let $\mathcal{T}' = \{T'_1, \dots, T'_{t'}\}$ be an arbitrary subset of \mathcal{T} , let \mathcal{B} be an arbitrary equivalence relation on B , and let \mathcal{B}_i denote an arbitrary subset of \mathcal{B} for $i = 1, \dots, t'$. We say that we made the *right choice* if $\mathcal{T}' = \mathcal{T}^S$, $\mathcal{B} = \mathcal{B}^S$, and $\mathcal{B}_i = \mathcal{B}_i^S$ for $i = 1, \dots, t'$. The algorithm considers two cases.

Case 1: If $|V(G')| \leq q(k + 1)$, then we can essentially use exhaustive enumeration. Let \tilde{G} be the graph obtained from G' by identifying two border vertices if they are in the same

equivalence class of \mathcal{B} . This also makes each \mathcal{B}_i a set of vertices, which by abuse of notation, we denote by \mathcal{B}_i as well. For $i = 1, \dots, t'$, let \tilde{T}_i be equal to $\mathcal{B}_i \cup (T'_i \cap (V(G') \setminus B))$. Then $\tilde{\mathcal{T}} = \{\tilde{T}_1, \dots, \tilde{T}_{t'}\}$. We verify that no terminal set in $\tilde{\mathcal{T}}$ is a singleton; otherwise, we can continue with the next branch.

► **Lemma 10.** *Assume we made the right choice. Then S' is an edge Steiner multicut of $(\tilde{G}, \tilde{\mathcal{T}})$. Moreover, for any edge Steiner multicut X of $(\tilde{G}, \tilde{\mathcal{T}})$, $X \cup S''$ is an edge Steiner multicut of (G, \mathcal{T}) .*

Now the algorithm uses exhaustive enumeration to find a smallest edge Steiner multicut of $(\tilde{G}, \tilde{\mathcal{T}})$ of size at most k (if one exists) in $t(qk)^{O(k)}$ time. Mark this set of edges in G' .

By Lemma 10, if we made the right choice, we mark a set Z such that $Z \cup S''$ is a smallest edge Steiner multicut of (G, \mathcal{T}) .

Case 2: If $|V(G')| > q(k+1)$, then a more complicated approach is needed, because we cannot just use exhaustive enumeration. In fact, we cannot work with $(\tilde{G}, \tilde{\mathcal{T}})$ directly here, as \tilde{G} might have a (q, k) -good edge separation, even though G' does not.

We proceed as follows. Apply Lemma 9 with $a = q$ and $b = 2k$ to G' with respect to $Y := S'$, and let \mathcal{F} be the resulting family. Note that $Y = S'$ is a minimal separator, $|V(G')| > q(k+1) = a(b/2 + 1)$, and G' has no $(a, b/2)$ -good edge separation, and thus the lemma indeed applies. Consider an arbitrary $F \in \mathcal{F}$. We augment our definition of the *right choice* by adding the condition that $F = F_0$, where F_0 is the family that Lemma 9 promises exists in \mathcal{F} . Now find G'_F . If h_F does not exist in G'_F , then we proceed to the next set F , as Lemma 9 promises that h_F exists if we made the right choice.

We call a set $X \subseteq E(G'_F)$ an *all-or-nothing cut* if for each connected component C' of $G'_F \setminus \{h_F\}$, X either contains all edges of $G'_F[C' \cup \{h_F\}]$ or none of these edges. Note that Lemma 9 promises that S' is an all-or-nothing cut in G'_F for $F = F_0 \in \mathcal{F}$.

Let \mathcal{C} denote the set of connected components of $G'_F \setminus \{h_F\}$ that contain a vertex onto which a border vertex was contracted. Let $Y \subseteq \bigcup_{C \in \mathcal{C}} E(G'_F[C \cup \{h_F\}])$ be an arbitrary set of edges that contains for each $C \in \mathcal{C}$ either all edges of $G'_F[C \cup \{h_F\}]$ or none of these edges. Note that Y is basically an all-or-nothing cut restricted to the edges induced by \mathcal{C} . The algorithm will consider all possible choices of Y . We augment our definition of the *right choice* again, by adding the condition that $Y = Y_0$, where $Y_0 := S' \cap (\bigcup_{C \in \mathcal{C}} E(G'_F[C \cup \{h_F\}]))$.

Now the algorithm deletes all edges in Y and contracts any edges in $(\bigcup_{C \in \mathcal{C}} E(G'_F[C \cup \{h_F\}])) \setminus Y$. Denote the resulting graph by H_F . Let \hat{G} be the graph obtained from H_F by identifying two border vertices if they are in the same equivalence class of \mathcal{B} . This also compresses each \mathcal{B}_i into a set of vertices, which by abuse of notation, we denote by \mathcal{B}_i as well. For $i = 1, \dots, t'$, let \hat{T}_i be equal to $\mathcal{B}_i \cup (T'_i \cap (V(\hat{G}) \setminus B))$. Then $\hat{\mathcal{T}}$ consists of all \hat{T}_i that are not already separated in H_F . We verify that no terminal set in $\hat{\mathcal{T}}$ is a singleton; otherwise, we can continue with the next branch.

► **Lemma 11.** *Assume that we made the right choice. Then $S' \setminus Y$ is an edge Steiner multicut of $(\hat{G}, \hat{\mathcal{T}})$ that is an all-or-nothing cut. Moreover, for any edge Steiner multicut X of $(\hat{G}, \hat{\mathcal{T}})$, $Y \cup X \cup S''$ is an edge Steiner multicut of (G, \mathcal{T}) .*

We now aim to find a smallest edge Steiner multicut X of $(\hat{G}, \hat{\mathcal{T}})$ that is an all-or-nothing cut. Let $\{C'_1, \dots, C'_u\}$ be the set of connected components of $\hat{G} \setminus \{h_F\}$. Let $\hat{\mathcal{T}}|_i$ denote the set of terminal sets in $\hat{\mathcal{T}}$ that are separated if one removes all edges of $E(\hat{G}[\{h_F\} \cup C'_i])$ from G'_F . Define $z[\mathcal{U}, i]$, where $\mathcal{U} \subseteq \hat{\mathcal{T}}$ and $1 \leq i \leq u$, as the size of the smallest all-or-nothing cut

of the terminal sets in \mathcal{U} using only edges in or going out of C'_1, \dots, C'_i . Then for any $\mathcal{U} \subseteq \hat{\mathcal{T}}$,

$$z[\mathcal{U}, 1] = \begin{cases} \infty & \text{if } \mathcal{U} \not\subseteq \hat{\mathcal{T}}|_1 \\ |E(G'_F[\{h_F\} \cup C'_1])| & \text{otherwise (i.e. if } \mathcal{U} \subseteq \hat{\mathcal{T}}|_1) \end{cases}$$

and for $i > 1$, $z[\mathcal{U}, i] = \min \left\{ z[\mathcal{U}, i-1], |E(G'_F[\{h_F\} \cup C'_i])| + z[\mathcal{U} \setminus \hat{\mathcal{T}}|_i, i-1] \right\}$. Note that $z[\hat{\mathcal{T}}, u]$ holds the size of the smallest edge Steiner multicut of $(\hat{G}, \hat{\mathcal{T}})$ that is an all-or-nothing cut (if one exists). Finding the set achieving this smallest size is straightforward from the dynamic-programming table. Finally, over all choices of F and all choices of Y , mark in G' the smallest set of edges that was found if it has size at most k .

By Lemma 11, if we made the right choice, we mark a set Z such that $Z \cup S''$ is a smallest edge Steiner multicut of (G, \mathcal{T}) .

In both cases, let M be the set of marked edges. Now we contract all unmarked edges $E(G') \setminus M$ in G . Let \tilde{G} denote the resulting graph; note that in general \tilde{G} is a multigraph. Each time we contract an edge between two vertices u and v , we replace u and v by uv in all terminal sets in \mathcal{T} . Let $\tilde{\mathcal{T}}$ denote the resulting set of terminal sets. Observe that if a terminal set \tilde{T}_i in $\tilde{\mathcal{T}}$ is a singleton set and T_i was not a singleton set in \mathcal{T} , then we can answer “no”. Now it remains to prove that (G, \mathcal{T}, k) is a “yes”-instance if and only if $(\tilde{G}, \tilde{\mathcal{T}}, k)$ is. For this, it suffices to note that an edge Steiner multicut of $(\tilde{G}, \tilde{\mathcal{T}})$ corresponds directly to an edge Steiner multicut of (G, \mathcal{T}) , and that for any smallest edge Steiner multicut S of (G, \mathcal{T}) there is a smallest edge Steiner multicut $Z \cup (S \setminus E(G'))$ of (G, \mathcal{T}) such that $Z \subseteq M$.

Since there are at most $2k$ border vertices and t terminal sets, there are $r = 2^{O(kt \log k)}$ different branches that we consider for \mathcal{T}' , \mathcal{B} , and \mathcal{B}_i , and in each we mark at most k edges. Choose $q = rk + 1$. Now note that $|E(G')| \geq q$. If $G = G'$, then this follows from the assumption that $|E(G)| > q$. If $G \neq G'$, then G' was obtained after considering multiple (q, k) -good separations. Hence, $|V(G')| > q$, and since G' is connected, $|E(G')| \geq q$. Since $q = rk + 1$, at least one edge of G' was not marked and thus contracted. Therefore, $|V(G)|$ decreases by at least one, and the entire procedure finishes after at most $|V(G)|$ iterations.

To analyze the running time, note that the dynamic-programming algorithm requires time $2^{O(t)k} + O(t|E(G)|)$. The family \mathcal{F} contains $2^{O(k^2 t \log k)} \log |E(G)|$ sets and can be constructed in $2^{O(k^2 t \log k)} |E(G)| \log |E(G)|$ time. Hence, Case 2 runs in $2^{O(k^2 t \log k)} |E(G)| \log |E(G)|$ time. Case 1 runs in $2^{O(k^2 t \log k)}$ time. Since there are $r = 2^{O(kt \log k)}$ different branches for \mathcal{T}' , \mathcal{B} , and \mathcal{B}_i that we consider, and it takes $2^{O(k^2 t \log k)} |V(G)|^4 \log |V(G)|$ time to find a $2k$ -bordered subgraph that does not admit a (q, k) -good separation, each iteration takes $2^{O(k^2 t \log k)} |V(G)|^4 \log |V(G)|$ time. Since there are at most $|V(G)|$ iterations, the total running time is $2^{O(k^2 t \log k)} |V(G)|^5 \log |V(G)|$. Actually, using the recurrence outlined by Chitnis et al. [8], one can show a bound on the running time of $2^{O(k^2 t \log k)} |V(G)|^4 \log |V(G)|$.

4 Steiner Multicuts for Graphs of Bounded Treewidth

In this section, we consider STEINER MULTICUT on graphs of bounded treewidth. To start the exposition, we note that EDGE STEINER MULTICUT and RESTR. NODE STEINER MULTICUT are NP-complete for trees and NODE STEINER MULTICUT is NP-complete on series-parallel graphs [4], which are graphs of treewidth two. This means that any efficient algorithm for STEINER MULTICUT on graphs of bounded treewidth needs an additional parameter.

We first show Theorem 3, namely that all variants of STEINER MULTICUT for the parameter k are W[1]-hard, even if $p = 3$ and $\text{tw}(G) = 2$ (but t is unbounded). The graph G is in fact a tree plus one node. We then contrast this result by showing that the problem is fixed-parameter tractable on bounded treewidth graphs when t is a parameter.

The reduction for Theorem 3 is from an intermediate problem, (MONOTONE) NAE-INTEGGER-3-SAT. In this problem, we are given variables x_1, \dots, x_k that each take a value in $\{1, \dots, n\}$ and clauses C_1, \dots, C_m of the form $\text{NAE}(x_{i_1} \leq a_1, x_{i_2} \leq a_2, x_{i_3} \leq a_3)$, $a_1, a_2, a_3 \in \{1, \dots, n\}$, which is satisfied if not all three inequalities are true and not all are false (i.e., they are “not all equal”). The goal is to find an assignment of the variables that satisfies all given clauses. We remark that NAE-INTEGGER-3-SAT generalizes MONOTONE NAE-3-SAT (by restriction to $n = 2$), and that NAE-INTEGGER-3-SAT can be solved in time $O(m \cdot n^k)$, by enumerating all assignments.

► **Lemma 12.** *NAE-INTEGGER-3-SAT is $W[1]$ -hard for parameter k .*

Proof. Let (G, k) be an instance of MULTICOLORED CLIQUE [18]. We use V_i to denote the set of vertices of color i , $n_i = |V_i|$, and $E_{i,j}$ to denote the set of edges with one endpoint in V_i and the other in V_j . We create an instance of NAE-INTEGGER-3-SAT on variables x_i , one for each color $1 \leq i \leq k$, and y_{ij} , one for each pair of colors $1 \leq i < j \leq k$. We identify the vertices V_i with the integers $\{1, \dots, n_i\}$ in an arbitrary way. We restrict x_i to $\{1, \dots, n_i\}$ using the clause $\text{NAE}(x_i \leq 0, x_i \leq 0, x_i \leq n_i)$, and write $x_i = u$ if the number x_i corresponds to vertex u . Analogously, we can identify the edges $uv \in E_{i,j}$ with numbers in $\{1, \dots, |E_{i,j}|\}$ and write $y_{ij} = uv$ if we pick the number corresponding to edge uv . Consider the following constraints (for any edge uv , with u of color i and v of color j), $y_{ij} = uv \Rightarrow x_i = u$, $y_{ij} = uv \Rightarrow x_j = v$. If we can encode these constraints with NAE-clauses, then any satisfying assignment of the constructed NAE-INTEGGER-3-SAT instance corresponds to a clique in G , as all chosen pairs y_{ij} correspond to edges, and edges sharing a color i picked the same vertex x_i . We focus on the first constraint; the second is similar. Note that the constraint is equivalent to $y_{ij} = uv \Rightarrow x_i \geq u$, $y_{ij} = uv \Rightarrow x_i \leq u$. Again, w.l.o.g., we focus on the first of these constraints. It is equivalent to $y_{ij} < uv \vee y_{ij} > uv \vee x_i \geq u$, which in turn can be written as $\text{NAE}(y_{ij} < uv, y_{ij} > uv, x_i \geq u)$, since $y_{ij} < uv, y_{ij} > uv$ cannot both be true. Note that we can replace any inequality $x < a$ by $x \leq a - 1$ (and similarly for $x > a$). Hence, we can encode all desired constraints if we may use “ \leq ” and “ \geq ” inequalities, not only “ \leq ” inequalities, as is the case in the definition of NAE-INTEGGER-3-SAT.

In the remainder of this proof, we reduce NAE-INTEGGER-3-SAT with “ \leq ” and “ \geq ” inequalities to the original variant with only “ \leq ” inequalities. Given any instance of NAE-INTEGGER-3-SAT with both types of inequalities, for any variable x we introduce a new variable \bar{x} . For any $1 \leq v \leq n$, we add the constraint $\text{NAE}(x \leq v, x \leq v, \bar{x} \leq n - v)$. This enforces $\bar{x} = n + 1 - x$. Finally, we replace any inequality $x \geq v$ by $\bar{x} \leq n + 1 - v$. This yields an equivalent NAE-INTEGGER-3-SAT instance with only “ \leq ” inequalities. ◀

Proof of Theorem 3. We first give a reduction from NAE-INTEGGER-3-SAT to EDGE STEINER MULTICUT (satisfying $p = 3$ and $\text{tw}(G) = 2$). Consider an instance of NAE-INTEGGER-3-SAT on variables x_1, \dots, x_k taking values in $\{1, \dots, n\}$ with clauses C_1, \dots, C_m . Take k paths consisting of n edges and identify their start nodes (to a common node s) and end nodes (to a common node t), respectively. The resulting graph G has $\text{tw}(G) = 2$, since it is not a tree, but becomes a tree after deleting s (or t). Let v_j^i be the j -th node on the i -th path from s to t , so that $v_0^i = s$ and $v_n^i = t$. For each clause $\text{NAE}(x_{i_1} \leq a_1, x_{i_2} \leq a_2, x_{i_3} \leq a_3)$ we introduce a terminal set $\{v_{a_1}^{i_1}, v_{a_2}^{i_2}, v_{a_3}^{i_3}\}$ (note that we can assume $0 \leq a_j \leq n$ without loss of generality). Further, we let $\{s, t\}$ be a terminal set and set the cut size to k , i.e., we allow to delete k edges. This finishes the construction. In order to separate s from t we need to cut at least one edge of each of the k paths that connect s and t , and because the cut size is k we have to delete *exactly one* edge per path. Say we delete the x_i -th edge on the i -th path. This splits G into two components, one containing s and the other containing t . Note that

we separate nodes v_j^i and $v_{j'}^{i'}$ by cutting at $x_i \leq j$ and $x_{i'} > j'$ (or with both inequalities the other way round), since then v_j^i is in the t -component and $v_{j'}^{i'}$ in the s -component. Hence, the following are equivalent:

- the terminal set $\{v_{a_1}^{i_1}, v_{a_2}^{i_2}, v_{a_3}^{i_3}\}$ is disconnected;
- some pair of nodes in this set is disconnected;
- among the inequalities $x_{i_j} \leq a_j$, $j = 1, 2, 3$, one is true and one is false;
- the clause $\text{NAE}(x_{i_1} \leq a_1, x_{i_2} \leq a_2, x_{i_3} \leq a_3)$ is satisfied.

Therefore, the given NAE-INTEGGER-3-SAT instance is equivalent to the constructed EDGE STEINER MULTICUT instance.

We can adapt this construction to prove hardness of NODE STEINER MULTICUT; hardness of RESTR. NODE STEINER MULTICUT then follows from Lemma 4. ◀

We contrast the above theorem with the following result by showing that STEINER MULTICUT is fixed-parameter tractable for the parameter t if the graph has bounded treewidth, through an MSOL-formula.

► **Theorem 13.** NODE STEINER MULTICUT, EDGE STEINER MULTICUT, and RESTR. NODE STEINER MULTICUT are fixed-parameter tractable for the parameter $t + \text{tw}(G)$.

5 Hardness for Cutsizes k and Number of Terminal Sets t

In this section, we consider the STEINER MULTICUT problem on general graphs parameterized by $k+t$. We show that both node deletion versions of the problem, NODE STEINER MULTICUT and RESTR. NODE STEINER MULTICUT, are $W[1]$ -hard for this parameter.

► **Theorem 14.** NODE STEINER MULTICUT and RESTR. NODE STEINER MULTICUT are $W[1]$ -hard for the parameter $k+t$.

Proof. We present a parameterized reduction from the MULTICOLORED CLIQUE problem [18] to NODE STEINER MULTICUT. Let (H, k) be an instance of MULTICOLORED CLIQUE, and let V_i and $E_{i,j}$ be as in the definition of MULTICOLORED CLIQUE. We then create the following instance of NODE STEINER MULTICUT. First, we subdivide each edge of H , and let $N_{i,j}$ denote the set of nodes that were created when subdividing the edges of $E_{i,j}$. Then, add a complete graph C with $2k$ nodes, where we denote the nodes of C by c_1, \dots, c_{2k} , and make all nodes of V_i adjacent to c_{2i-1} and c_{2i} for each $i = 1, \dots, k$. Let G denote the resulting graph. Observe that $G[V(H)]$ and $G[\bigcup_{i,j} N_{i,j}]$ are both independent sets of G . We then create terminal sets $T_i = V_i \cup \{c_{2i-1}\}$ and $T'_i = V_i \cup \{c_{2i}\}$, and terminal sets $T_{i,j} = N_{i,j} \cup V(C)$. Let $\mathcal{T} = \{T_i, T'_i \mid i = 1, \dots, k\} \cup \{T_{i,j} \mid i \neq j, i, j = 1, \dots, k\}$. Then the created instance is (G, \mathcal{T}, k) .

Suppose that (H, k) is a “yes”-instance of MULTICOLORED CLIQUE, and let K denote a clique of H such that $V(K) \cap V_i \neq \emptyset$ for each i . Pick a node $v_i \in V(K) \cap V_i$ and let $S = \{v_i \mid i = 1, \dots, k\}$. Observe that v_i disconnects terminal sets T_i and T'_i . Further, if we let $n_{i,j}$ denote the subdivision node of the edge $(v_i, v_j) \in E(H)$, then v_i and v_j disconnect $n_{i,j}$ from the rest of $T_{i,j}$. Finally, $|S| = k$. Therefore, (G, \mathcal{T}, k) is a “yes”-instance of NODE STEINER MULTICUT.

Suppose that (G, \mathcal{T}, k) is a “yes”-instance of NODE STEINER MULTICUT, and let $S \subseteq V(G)$ denote a node Steiner multicut of G with respect to terminal sets \mathcal{T} such that $|S| \leq k$. We claim that $H[S]$ is a multicolored clique of H . First, observe that to disconnect T_i and T'_i , we need that $S \cap T_i \neq \emptyset$ and $S \cap T'_i \neq \emptyset$. Since $|S| \leq k$, we know that $S \cap T_i \cap T'_i \neq \emptyset$ for each $i = 1, \dots, k$. This implies that $|S| = k$, that $S \subseteq V(H)$, and that $S \cap V_i \neq \emptyset$ for each

$i = 1, \dots, k$. It remains to show that $H[S]$ is a clique. Let v_i denote the node in $S \cap V_i$. Suppose that $(v_i, v_j) \notin E(H)$ for some i, j . Then consider the terminal set $T_{i,j}$, and observe that for any node $n \in N_{i,j}$ at least one endpoint of the edge corresponding to n is not in S . Since $S \cap V(C) = \emptyset$, this implies that $T_{i,j}$ is not disconnected by S , a contradiction. It follows that $H[S]$ is a clique, and thus (H, k) is a “yes”-instance of MULTICOLORED CLIQUE.

For RESTR. NODE STEINER MULTICUT hardness follows from the statement for NODE STEINER MULTICUT and Lemma 4. ◀

6 Steiner Multicuts in Trees

We state the following results on trees; the proofs are in the full version. The first theorem generalizes the algorithm for NODE MULTICUT on trees [4].

► **Theorem 15.** NODE STEINER MULTICUT *can be decided in linear time on trees.*

The next theorems follow from a reduction from HITTING SET [15, 14].

► **Theorem 16.** EDGE STEINER MULTICUT *and* RESTR. NODE STEINER MULTICUT *are* $W[2]$ -*hard on trees for the parameter* k .

► **Theorem 17.** EDGE STEINER MULTICUT *and* RESTR. NODE STEINER MULTICUT *have no polynomial kernel on trees for the parameter* $k + t$, *unless the polynomial hierarchy collapses to the third level.*

Note that the above theorem complements the FPT result of Theorem 1.

The final theorem uses a branching strategy in which an incident edge or neighbor needs to be chosen for the lowest common ancestor of the terminals in any terminal set.

► **Theorem 18.** EDGE STEINER MULTICUT *and* RESTR. NODE STEINER MULTICUT *are fixed-parameter tractable on trees for the parameter* $k + p$.

7 Discussion

We provided a comprehensive computational complexity analysis of the STEINER MULTICUT problem with respect to fundamental parameters, culminating in either a fixed-parameter algorithm or a $W[1]$ -hardness result for *every* combination of parameters. This way, we generalize known tractability results for special cases of STEINER MULTICUT, and chart the boundary of tractability for other cases. See Table 1 for a complete overview.

We leave several interesting questions for future research. A possible extension is to consider directed graphs. Already MULTICUT is $W[1]$ -hard in this case [32] for parameter cut size k , even on acyclic directed graphs [27]. On the other hand, MULTICUT is fixed-parameter tractable for the parameter $k + t$ in directed acyclic graphs [27]. It would be interesting whether this result generalizes to STEINER MULTICUT.

Another possible extension is to investigate which problems admit polynomial kernels. While we have resolved many kernelization questions in the full version of this paper, several open problems remain, in particular whether there is a polynomial kernel for the parameters $k + t + p$ on general graphs. Answers in this research direction might shed new light on some long-standing open questions [9] on the existence of polynomial kernels for MULTICUT for parameter $k + t$ (currently, only a kernel of size $k^{O(\sqrt{t})}$ is known [28], and there is no kernel of size polynomial in k only [10]).

Acknowledgements. We thank Magnus Wahlström for an insight that helped in proving Lemma 12.

References

- 1 Ajit Agrawal, Philip Klein, and R. Ravi. When trees collide: an approximation algorithm for the generalized Steiner problem on networks. *SIAM J. Comput.*, 24(3):440–456, 1995.
- 2 Adi Avidor and Michael Langberg. The multi-multiway cut problem. *Theoret. Comput. Sci.*, 377(1-3):35–42, 2007.
- 3 Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. In *Proc. STOC 2011*, pages 459–468, New York, NY, USA, 2011. ACM.
- 4 Gruia Călinescu, Cristina G. Fernandes, and Bruce Reed. Multicuts in unweighted graphs and digraphs with bounded degree and bounded tree-width. *J. Algorithms*, 48(2):333–359, 2003.
- 5 Yixin Cao, Jianer Chen, and Jia-Hao Fan. An $O^*(1.84^k)$ parameterized algorithm for the multiterminal cut problem. *Information Processing Letters*, 114(4):167–173, 2014.
- 6 Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating multicut and sparsest-cut. *Comput. Complexity*, 15(2):94–114, 2006.
- 7 Jianer Chen, Yang Liu, and Songjian Lu. An improved parameterized algorithm for the minimum node multiway cut problem. *Algorithmica*, 55(1):1–13, 2009.
- 8 Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michał Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. In *Proc. FOCS 2012*, pages 460–469, Washington, DC, USA, 2012. IEEE Computer Society.
- 9 Marek Cygan, Łukasz Kowalik, and Marcin Pilipczuk. Open problems from workshop on kernels, April 2013. <http://worker2013.mimuw.edu.pl/slides/worker-opl.pdf>.
- 10 Marek Cygan, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Magnus Wahlström. Clique cover and graph separation: new incompressibility results. *ACM Transactions on Computation Theory (TOCT)*, 6(2), 2014.
- 11 Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. On multiway cut parameterized above lower bounds. *ACM Transactions on Computation Theory (TOCT)*, 5(1), 2013.
- 12 E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, 1994.
- 13 G. B. Dantzig and D. R. Fulkerson. On the max-flow min-cut theorem of networks. In *Linear inequalities and related systems*, Annals of Mathematics Studies, no. 38, pages 215–221. Princeton University Press, Princeton, N. J., 1956.
- 14 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Incompressibility through colors and IDs. In *Proc. ICALP 2009*, volume 5555 of *Lecture Notes Comput. Sci.*, pages 378–389. Springer, Berlin, 2009.
- 15 R. G. Downey and M. R. Fellows. *Parameterized complexity*. Monographs in Computer Science. Springer-Verlag, New York, 1999.
- 16 Rodney G. Downey, Vladimir Estivill-Castro, Michael Fellows, Elena Prieto, and Frances A. Rosamond. Cutting up is hard to do: The parameterised complexity of k -cut and related problems. *Electr. Notes Theor. Comput. Sci.*, 78(0):209 – 222, 2003.
- 17 P. Elias, A. Feinstein, and C.E. Shannon. A note on the maximum flow through a network. *Information Theory, IRE Transactions on*, 2(4):117–119, 1956.
- 18 Michael R. Fellows, Danny Hermelin, Frances Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theoret. Comput. Sci.*, 410(1):53–61, 2009.

- 19 L. R. Ford, Jr. and D. R. Fulkerson. Maximal flow through a network. *Canad. J. Math.*, 8:399–404, 1956.
- 20 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM J. Comput.*, 25(2):235–251, 1996.
- 21 Sylvain Guillemot. FPT algorithms for path-transversal and cycle-transversal problems. *Discrete Optim.*, 8(1):61–71, 2011.
- 22 Jiong Guo, Falk Hüffner, Erhan Kenar, Rolf Niedermeier, and Johannes Uhlmann. Complexity and exact algorithms for multicut. In *Proc. SOFSEM 2006*, volume 3831 of *Lecture Notes Comput. Sci.*, pages 303–312, Berlin, 2006. Springer.
- 23 Jiong Guo and Rolf Niedermeier. Fixed-parameter tractability and data reduction for multicut in trees. *Networks*, 46(3):124–135, 2005.
- 24 Ken-ichi Kawarabayashi and Mikkel Thorup. The minimum k -way cut of bounded size is fixed-parameter tractable. In *Proc. FOCS 2011*, pages 160–169. IEEE Computer Soc., Los Alamitos, CA, 2011.
- 25 Philip N. Klein, Serge A. Plotkin, Satish Rao, and Éva Tardos. Approximation algorithms for Steiner and directed multicuts. *J. Algorithms*, 22(2):241–269, 1997.
- 26 Jochen Könemann, Stefano Leonardi, Guido Schäfer, and Stefan H. M. van Zwam. A group-strategyproof cost sharing mechanism for the Steiner Forest game. *SIAM J. Comput.*, 37(5):1319–1341, 2008.
- 27 Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Magnus Wahlström. Fixed-parameter tractability of multicut in directed acyclic graphs. In *Proc. ICALP 2012*, volume 7391 of *Lecture Notes Comput. Sci.*, pages 581–593. Springer, Berlin, 2012.
- 28 Stefan Kratsch and Magnus Wahlström. Representative sets and irrelevant vertices: New tools for kernelization. In *Proc. FOCS 2012*, pages 450–459, 2012.
- 29 Lap Chi Lau. An approximate max-Steiner-tree-packing min-Steiner-cut theorem. *Combinatorica*, 27(1):71–90, 2007.
- 30 Dániel Marx. Parameterized graph separation problems. *Theoret. Comput. Sci.*, 351(3):394–406, 2006.
- 31 Dániel Marx, Barry O’Sullivan, and Igor Razgon. Finding small separators in linear time via treewidth reduction. *ACM Trans. Algorithms*, 9(4):30:1–30:35, October 2013.
- 32 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. In *Proc. STOC 2011*, pages 469–478, New York, NY, USA, 2011. ACM.
- 33 Mingyu Xiao. Simple and improved parameterized algorithms for multiterminal cuts. *Theory Comput. Syst.*, 46(4):723–736, 2010.
- 34 Mihalis Yannakakis, Paris C. Kanellakis, Stavros S. Cosmadakis, and Christos H. Papadimitriou. Cutting and partitioning a graph after a fixed pattern (extended abstract). In *Proc. ICALP 1983*, volume 154 of *Lecture Notes Comput. Sci.*, pages 712–722, London, UK, UK, 1983. Springer-Verlag.
- 35 Bo Yu and Joseph Cheriyan. Approximation algorithms for feasible cut and multicut problems. In *Proc. ESA 1995*, volume 979 of *Lecture Notes Comput. Sci.*, pages 394–408. Springer, Berlin, 1995.

Solving Totally Unimodular LPs with the Shadow Vertex Algorithm*

Tobias Brunsch, Anna Großwendt, and Heiko Röglin

Department of Computer Science
University of Bonn, Germany
{brunsch,grosswen,roeglin}@cs.uni-bonn.de

Abstract

We show that the shadow vertex simplex algorithm can be used to solve linear programs in strongly polynomial time with respect to the number n of variables, the number m of constraints, and $1/\delta$, where δ is a parameter that measures the flatness of the vertices of the polyhedron. This extends our recent result that the shadow vertex algorithm finds paths of polynomial length (w.r.t. n , m , and $1/\delta$) between two given vertices of a polyhedron [4].

Our result also complements a recent result due to Eisenbrand and Vempala [6] who have shown that a certain version of the random edge pivot rule solves linear programs with a running time that is strongly polynomial in the number of variables n and $1/\delta$, but independent of the number m of constraints. Even though the running time of our algorithm depends on m , it is significantly faster for the important special case of totally unimodular linear programs, for which $1/\delta \leq n$ and which have only $O(n^2)$ constraints.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases linear optimization, simplex algorithm, shadow vertex method

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.171

1 Introduction

The shadow vertex algorithm is a well-known pivoting rule for the simplex method that has gained attention in recent years because it was shown to have polynomial running time in the model of smoothed analysis [8]. Recently we have observed that it can also be used to find short paths between given vertices of a polyhedron [4]. Here short means that the path length is $O(\frac{mn^2}{\delta^2})$, where n denotes the number of variables, m denotes the number of constraints, and δ is a parameter of the polyhedron that we will define shortly.

Our result left open the question whether or not it is also possible to solve linear programs in polynomial time with respect to n , m , and $1/\delta$ by the shadow vertex simplex algorithm. In this article we resolve this question and introduce a variant of the shadow vertex simplex algorithm that solves linear programs in strongly polynomial time with respect to these parameters.

For a given matrix $A = [a_1, \dots, a_m]^T \in \mathbb{R}^{m \times n}$ and vectors $b \in \mathbb{R}^m$ and $c_0 \in \mathbb{R}^n$ our goal is to solve the linear program $\max\{c_0^T x \mid Ax \leq b\}$. We assume without loss of generality that $\|c_0\| = 1$ and $\|a_i\| = 1$ for every row a_i of the constraint matrix.

► **Definition 1.** The matrix A satisfies the δ -distance property if the following condition holds: For any $I \subseteq \{1, \dots, m\}$ and any $j \in \{1, \dots, m\}$, if $a_j \notin \text{span}\{a_i \mid i \in I\}$

* This research was supported by ERC Starting Grant 306465 (BeyondWorstCase).

then $\text{dist}(a_j, \text{span}\{a_i \mid i \in I\}) \geq \delta$. In other words, if a_j does not lie in the subspace spanned by the $a_i, i \in I$, then its distance to this subspace is at least δ .

We present a variant of the shadow vertex simplex algorithm that solves linear programs in strongly polynomial time with respect to n, m , and $1/\delta$, where δ denotes the largest δ' for which the constraint matrix of the linear program satisfies the δ' -distance property. (In the following theorems, we assume $m \geq n$. If this is not the case, we use a method described in [3] to add irrelevant constraints so that A has rank n . Hence, for instances that have fewer constraints than variables, the parameter m should be replaced by n in all bounds.)

► **Theorem 2.** *There exists a randomized variant of the shadow vertex simplex algorithm (described in Section 2) that solves linear programs with n variables and m constraints satisfying the δ -distance property using $O(\frac{mn^3}{\delta^2} \cdot \log(\frac{1}{\delta}))$ pivots in expectation if a basic feasible solution is given. A basic feasible solution can be found using $O(\frac{m^5}{\delta^2} \cdot \log(\frac{1}{\delta}))$ pivots in expectation.*

We stress that the algorithm can be implemented without knowing the parameter δ . From the theorem it follows that the running time of the algorithm is strongly polynomial with respect to the number n of variables, the number m of constraints, and $1/\delta$ because every pivot can be performed in time $O(mn)$ in the arithmetic model of computation (see Section 2.2).¹

Let $A \in \mathbb{Z}^{m \times n}$ be an integer matrix and let $A' \in \mathbb{R}^{m \times n}$ be the matrix that arises from A by scaling each row such that its norm equals 1. If Δ denotes an upper bound for the absolute value of any sub-determinant of A , then A' satisfies the δ -distance property for $\delta = 1/(\Delta^2 n)$ [4]. For such matrices A Phase 1 of the simplex method can be implemented more efficiently and we obtain the following result.

► **Theorem 3.** *For integer matrices $A \in \mathbb{Z}^{m \times n}$, there exists a randomized variant of the shadow vertex simplex algorithm (described in Section 2) that solves linear programs with n variables and m constraints using $O(mn^5 \Delta^4 \log(\Delta + 1))$ pivots in expectation if a basic feasible solution is given, where Δ denotes an upper bound for the absolute value of any sub-determinant of A . A basic feasible solution can be found using $O(m^6 \Delta^4 \log(\Delta + 1))$ pivots in expectation.*

Theorem 3 implies in particular that totally unimodular linear programs can be solved by our algorithm with $O(mn^5)$ pivots in expectation if a basic feasible solution is given and with $O(m^6)$ pivots in expectation otherwise.

Besides totally unimodular matrices there are also other classes of matrices for which $1/\delta$ is polynomially bounded in n . Eisenbrand and Vempala [6] observed, for example, that $\delta = \Omega(1/\sqrt{n})$ for edge-node incidence matrices of undirected graphs with n vertices. One can also argue that δ can be interpreted as a condition number of the matrix A in the following sense: If $1/\delta$ is large then there must be an $(n \times n)$ -submatrix of A of rank n that is almost singular.

1.1 Related Work

Shadow vertex simplex algorithm

We will briefly explain the geometric intuition behind the shadow vertex simplex algorithm. For a complete and more formal description, we refer the reader to [2] or [8]. Let us consider

¹ By *strongly polynomial with respect to n, m , and $1/\delta$* we mean that the number of steps in the arithmetic model of computation is bounded polynomially in n, m , and $1/\delta$ and the size of the numbers occurring during the algorithm is polynomially bounded in the encoding size of the input.

the linear program $\max\{c_0^T x \mid Ax \leq b\}$ and let $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ denote the polyhedron of feasible solutions. Assume that an initial vertex x_1 of P is known and assume, for the sake of simplicity, that there is a unique optimal vertex x^* of P that maximizes the objective function $c_0^T x$. The shadow vertex pivot rule first computes a vector $w \in \mathbb{R}^n$ such that the vertex x_1 minimizes the objective function $w^T x$ subject to $x \in P$. Again for the sake of simplicity, let us assume that the vectors c_0 and w are linearly independent.

In the second step, the polyhedron P is projected onto the plane spanned by the vectors c_0 and w . The resulting projection is a (possibly open) polygon P' and one can show that the projections of both the initial vertex x_1 and the optimal vertex x^* are vertices of this polygon. Additionally, every edge between two vertices x and y of P' corresponds to an edge of P between two vertices that are projected onto x and y , respectively. Due to these properties a path from the projection of x_1 to the projection of x^* along the edges of P' corresponds to a path from x_1 to x^* along the edges of P .

This way, the problem of finding a path from x_1 to x^* on the polyhedron P is reduced to finding a path between two vertices of a polygon. There are at most two such paths and the shadow vertex pivot rule chooses the one along which the objective $c_0^T x$ improves.

Finding short paths

In [4] we considered the problem of finding a short path between two given vertices x_1 and x_2 of the polyhedron P along the edges of P . Our algorithm is the following variant of the shadow vertex algorithm: Choose two vectors $w_1, w_2 \in \mathbb{R}^n$ such that x_1 uniquely minimizes $w_1^T x$ subject to $x \in P$ and x_2 uniquely maximizes $w_2^T x$ subject to $x \in P$. Then project the polyhedron P onto the plane spanned by w_1 and w_2 in order to obtain a polygon P' . Let us call the projection π . By the same arguments as above, it follows that $\pi(x_1)$ and $\pi(x_2)$ are vertices of P' and that a path from $\pi(x_1)$ to $\pi(x_2)$ along the edges of P' can be translated into a path from x_1 to x_2 along the edges of P . Hence, it suffices to compute such a path to solve the problem. Again computing such a path is easy because P' is a two-dimensional polygon.

The vectors w_1 and w_2 are not uniquely determined, but they can be chosen from cones that are determined by the vertices x_1 and x_2 and the polyhedron P . We proved in [4] that the expected path length is $O(\frac{mn^2}{\delta^2})$ if w_1 and w_2 are chosen randomly from these cones. For totally unimodular matrices this implies that the diameter of the polyhedron is bounded by $O(mn^4)$, which improved a previous result by Dyer and Frieze [5] who showed that for this special case paths of length $O(m^3 n^{16} \log(mn))$ can be computed efficiently.

Additionally, Bonifas et al. [1] proved that in a polyhedron defined by an integer matrix A between any pair of vertices there exists a path of length $O(\Delta^2 n^4 \log(n\Delta))$ where Δ is the largest absolute value of any sub-determinant of A . For the special case that A is a totally unimodular matrix, this bound simplifies to $O(n^4 \log n)$. Their proof is non-constructive, however.

Geometric random edge

Eisenbrand and Vempala [6] have presented an algorithm that solves a linear program $\max\{c_0^T x \mid Ax \leq b\}$ in strongly polynomial time with respect to the parameters n and $1/\delta$. Remarkably the running time of their algorithm does not depend on the number m of constraints. Their algorithm is based on a variant of the random edge pivoting rule. The algorithm performs a random walk on the vertices of the polyhedron whose transition probabilities are chosen such that it quickly attains a distribution close to its stationary distribution.

In the stationary distribution the random walk is likely at a vertex x_c that optimizes an objective function $c^T x$ with $\|c_0 - c\| < \frac{\delta}{2n}$. The δ -distance property guarantees that x_c and the optimal vertex x^* with respect to the objective function $c_0^T x$ lie on a common facet. This facet is then identified and the algorithm is run again in one dimension lower. This is repeated at most n times until all facets of the optimal vertex x^* are identified. The number of pivots to identify one facet of x^* is proven to be $O(n^{10}/\delta^8)$. A single pivot can be performed in polynomial time but determining the right transition probabilities is rather sophisticated and requires to approximately integrate a certain function over a convex body.

Let us point out that the number of pivots of our algorithm depends on the number m of constraints. However, Heller showed that for the important special case of totally unimodular linear programs $m = O(n^2)$ [7]. Using this observation we also obtain a bound that depends polynomially only on n for totally unimodular matrices.

Combinatorial linear programs

Éva Tardos has proved in 1986 that combinatorial linear programs can be solved in strongly polynomial time [9]. Here combinatorial means that A is an integer matrix whose largest entry is polynomially bounded in n . Her result implies in particular that totally unimodular linear programs can be solved in strongly polynomial time, which is also implied by Theorem 3. However, the proof and the techniques used to prove Theorem 3 are completely different from those in [9].

1.2 Our Contribution

We replace the random walk in the algorithm of Eisenbrand and Vempala by the shadow vertex algorithm. Given a vertex x_0 of the polyhedron P we choose an objective function $w^T x$ for which x_0 is an optimal solution. As in [4] we choose w uniformly at random from the cone determined by x_0 . Then we randomly perturb each coefficient in the given objective function $c_0^T x$ by a small amount. We denote by $c^T x$ the perturbed objective function. As in [4] we prove that the projection of the polyhedron P onto the plane spanned by w and c has $O(\frac{mn^2}{\delta^2})$ edges in expectation. If the perturbation is so small that $\|c_0 - c\| < \frac{\delta}{2n}$, then the shadow vertex algorithm yields with $O(\frac{mn^2}{\delta^2})$ pivots a solution that has a common facet with the optimal solution x^* . We follow the same approach as Eisenbrand and Vempala and identify the facets of x^* one by one with at most n calls of the shadow vertex algorithm.

The analysis in [4] exploits that the two objective functions possess the same type of randomness (both are chosen uniformly at random from some cones). This is not the case anymore because every component of c is chosen independently uniformly at random from some interval. This changes the analysis significantly and introduces technical difficulties that we address in this article.

The problem when running the simplex method is that a feasible solution needs to be given upfront. Usually, such a solution is determined in Phase 1 by solving a modified linear program with a constraint matrix A' for which a feasible solution is known and whose optimal solution is feasible for the linear program one actually wants to solve. There are several common constructions for this modified linear program, it is, however, not clear how the parameter δ is affected by modifying the linear program. To solve this problem, Eisenbrand and Vempala [6] have suggested a method for Phase 1 for which the modified constraint matrix A' satisfies the δ -distance property for the same δ as the matrix A . However, their method is very different from usual textbook methods and needs to solve m different linear programs to find an initial feasible solution for the given linear program. We show that also

one of the usual textbook methods can be applied. We argue that $1/\delta$ increases by a factor of at most \sqrt{m} and that Δ , the absolute value of any sub-determinant of A , does not change at all in case one considers integer matrices. In this construction, the number of variables increases from n to $n + m$.

1.3 Outline and Notation

In the following we assume that we are given a linear program $\max\{c_0^T x \mid Ax \leq b\}$ with vectors $b \in \mathbb{R}^m$ and $c_0 \in \mathbb{R}^n$ and a matrix $A = [a_1, \dots, a_m]^T \in \mathbb{R}^{m \times n}$. Moreover, we assume that $\|c_0\| = \|a_i\| = 1$ for all $i \in [m]$, where $[m] := \{1, \dots, m\}$ and $\|\cdot\|$ denotes the Euclidean norm. This entails no loss of generality since any linear program can be brought into this form by scaling the objective function and the constraints appropriately. For a vector $x \in \mathbb{R}^n \setminus \{0^n\}$ we denote by $\mathcal{N}(x) = \frac{1}{\|x\|} \cdot x$ the normalization of vector x .

For a vertex v of the polyhedron $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ we call the set of row indices $B_v = \{i \in \{1, \dots, m\} \mid a_i \cdot v = b_i\}$ *basis* of v . Then the *normal cone* C_v of v is given by the set

$$C_v = \left\{ \sum_{i \in B_v} \lambda_i a_i \mid \lambda_i \geq 0 \right\}.$$

We will describe our algorithm in Section 2.1 where we assume that the linear program is non-degenerate, that A has full rank n , and that the polyhedron P is bounded. We have already described in Section 3 of [4] that the linear program can be made non-degenerate by slightly perturbing the vector b . This does not affect the parameter δ because δ depends only on the matrix A . In Section 3 we analyze our algorithm and prove Theorem 2. In the full version of this article [3] we discuss why we can assume that A has full rank and why P is bounded. There are, of course, textbook methods to transform a linear program into this form. However, we need to be careful that this transformation does not change δ . Moreover in [3] we discuss how Phase 1 of the simplex method can be implemented and we give an alternative definition of δ and discuss some properties of this parameter. Due to space limitations most proofs are omitted. They can also be found in [3].

2 Algorithm

Given a linear program $\max\{c_0^T x \mid Ax \leq b\}$ and a basic feasible solution x_0 , our algorithm randomly perturbs each coefficient of the vector c_0 by at most $1/\phi$ for some parameter ϕ to be determined later. Let us call the resulting vector c . The next step is then to use the shadow vertex algorithm to compute a path from x_0 to a vertex x_c which maximizes the function $c^T x$ for $x \in P$. For $\phi > \frac{2m^{3/2}}{\delta}$ one can argue that the solution x has a facet in common with the optimal solution x^* of the given linear program with objective function $c_0^T x$. Then the algorithm is run again on this facet one dimension lower until all facets that define x^* are identified.

Section 2.1 presents the shadow vertex algorithm, the main building block of our algorithm. Details of the identification and reduction to an optimal facet are provided in the full version of this paper. In Section 2.2 we discuss the running time of a single pivot step of the shadow vertex algorithm.

2.1 The Shadow Vertex Method

In this section we assume that we are given a linear program of the form $\max\{c_0^T x \mid x \in P\}$, where $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ is a bounded polyhedron (i.e., a polytope), and a basic feasible solution $x_0 \in P$. We assume $\|c_0\| = \|a_i\| = 1$ for all rows a_i of A . Furthermore, we assume that the linear program is non-degenerate.

Due to the assumption $\|c_0\| = 1$ it holds $c_0 \in [-1, 1]^n$. Our algorithm slightly perturbs the given objective function $c_0^T x$ at random. For each component $(c_0)_i$ of c_0 it chooses an arbitrary interval $I_i \subseteq [-1, 1]$ of length $1/\phi$ with $(c_0)_i \in I_i$, where ϕ denotes a parameter that will be given to the algorithm. Then a random vector $c \in [-1, 1]^n$ is drawn as follows: Each component c_i of c is chosen independently uniformly at random from the interval I_i . We denote the resulting random vector by $\text{pert}(c_0, \phi)$. Note that we can bound the norm of the difference $\|c_0 - c\|$ between the vectors c_0 and c from above by $\frac{\sqrt{n}}{\phi}$.

The shadow vertex algorithm is given as Algorithm 1. It is assumed that ϕ is given to the algorithm as a parameter. We will discuss later how we can run the algorithm without knowing this parameter. Let us remark that the Steps 5 and 6 in Algorithm 1 are actually not executed separately. Instead of computing the whole projection P' in advance, the edges of P' are computed on the fly one after another.

Algorithm 1 Shadow Vertex Algorithm

- 1: Generate a random perturbation $c = \text{pert}(c_0, \phi)$ of c_0 .
 - 2: Determine n linearly independent rows u_k^T of A for which $u_k^T x_0 = b_k$.
 - 3: Draw a vector $\lambda \in (0, 1]^n$ uniformly at random.
 - 4: Set $w = -[u_1, \dots, u_n] \cdot \lambda$.
 - 5: Use the function $\pi : x \mapsto (c^T x, w^T x)$ to project P onto the Euclidean plane and obtain the shadow vertex polygon $P' = \pi(P)$.
 - 6: Walk from $\pi(x_0)$ along the edges of P' in increasing direction of the first coordinate until a rightmost vertex \tilde{x}_c of P' is found.
 - 7: Output the vertex x_c of P that is projected onto \tilde{x}_c .
-

Note that

$$\|w\| \leq \sum_{k=1}^n \lambda_k \cdot \|u_k\| \leq \sum_{k=1}^n \lambda_k \leq n,$$

where the second inequality follows because all rows of A are assumed to have norm 1.

The Shadow Vertex Algorithm yields a path from the vertex x_0 to a vertex x_c that is optimal for the linear program $\max\{c^T x \mid x \in P\}$ where $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$. The following theorem (whose proof can be found in Section 3) bounds the expected length of this path, i.e., the number of pivots.

► **Theorem 4.** *For any $\phi \geq \sqrt{n}$ the expected number of edges on the path output by Algorithm 1 is $O\left(\frac{mn^2}{\delta^2} + \frac{m\sqrt{n}\phi}{\delta}\right)$.*

Since $\|c_0 - c\| \leq \frac{\sqrt{n}}{\phi}$ choosing $\phi > \frac{2n^{3/2}}{\delta}$ suffices to ensure $\|c_0 - c\| < \frac{\delta}{2n}$. This implies (see [3]) that, for such a choice of ϕ , the vertex x_c has a facet in common with the optimal solution of the linear program $\max\{c_0^T x \mid x \in P\}$ and we can reduce the dimension of the linear program as discussed in [3]. This step is repeated at most n times. It is important that we can start each repetition with a known feasible solution because the transformation in [3] maps the optimal solution of the linear program of repetition i onto a feasible solution with

which repetition $i+1$ can be initialized. Together with Theorem 4 this implies that an optimal solution of the linear program can be found by performing in expectation $O\left(\frac{mn^3}{\delta^2} + \frac{mn^{3/2}\phi}{\delta}\right)$ pivots if a basic feasible solution x_0 and the right choice of ϕ are given. We will refer to this algorithm as *repeated shadow vertex algorithm*.

Since δ is not known to the algorithm, the right choice for ϕ cannot easily be computed. Instead we will try values for ϕ until an optimal solution is found. For $i \in \mathbb{N}$ let $\phi_i = 2^i n^{3/2}$. First we run the repeated shadow vertex algorithm with $\phi = \phi_0$ and check whether the returned solution is an optimal solution for the linear program $\max\{c_0^T x \mid x \in P\}$. If this is not the case, we run the repeated shadow vertex algorithm with $\phi = \phi_1$, and so on. We continue until an optimal solution is found. For $\phi = \phi_{i^*}$ with $i^* = \lceil \log_2(1/\delta) \rceil + 2$ this is the case because $\phi_{i^*} > \frac{2n^{3/2}}{\delta}$.

Since $\phi_{i^*} \leq \frac{8n^{3/2}}{\delta}$, in accordance with Theorem 4, each of the at most $i^* = O(\log(1/\delta))$ calls of the repeated shadow vertex algorithm uses in expectation

$$O\left(\frac{mn^3}{\delta^2} + \frac{mn^{3/2}\phi_{i^*}}{\delta}\right) = O\left(\frac{mn^3}{\delta^2}\right).$$

pivots. Together this proves the first part of Theorem 2. The second part follows from [3], where it is proven that Phase 1 can be realized with increasing $1/\delta$ by at most \sqrt{m} and increasing the number of variables from n to $n+m \leq 2m$. This implies that the expected number of pivots of each call of the repeated shadow vertex algorithm in Phase 1 is $O(m(n+m)^3 \sqrt{m}^2 / \delta^2) = O(m^5 / \delta^2)$. Since $1/\delta$ can increase by a factor of \sqrt{m} , the argument above yields that we need to run the repeated shadow vertex algorithm at most $i^* = O(\log(\sqrt{m}/\delta))$ times in Phase 1 to find a basic feasible solution. By setting $\phi_i = 2^i \sqrt{m}(n+m)^{3/2}$ instead of $\phi_i = 2^i (n+m)^{3/2}$ this number can be reduced to $i^* = O(\log(1/\delta))$ again.

Theorem 3 follows from Theorem 2 using the following fact from [4]: Let $A \in \mathbb{Z}^{m \times n}$ be an integer matrix and let $A' \in \mathbb{R}^{m \times n}$ be the matrix that arises from A by scaling each row such that its norm equals 1. If Δ denotes an upper bound for the absolute value of any sub-determinant of A , then A' satisfies the δ -distance property for $\delta = 1/(\Delta^2 n)$. Additionally in [3] it is proven that Phase 1 can be realized without increasing Δ but with increasing the number of variables from n to $n+m \leq 2m$. Substituting $1/\delta = \Delta^2 n$ in Theorem 2 almost yields Theorem 3 except for a factor $O(\log(\Delta^2 n))$ instead of $O(\log(\Delta + 1))$. This factor results from the number i^* of calls of the repeated shadow vertex algorithm. The desired factor of $O(\log(\Delta + 1))$ can be achieved by setting $\phi_i = 2^i n^{5/2}$ if a basic feasible solution is known and $\phi_i = 2^i (n+m)^{5/2}$ in Phase 1.

2.2 Running Time of the Repeated Shadow Vertex Algorithm

So far we have only discussed the number of pivots. Let us now calculate the actual running time of our algorithm. For an initial basic feasible solution x_0 the repeated shadow vertex algorithm repeats the following three steps until an optimal solution is found. Initially let $P' = P$.

Step 1: Run the shadow vertex algorithm for the linear program $\max\{c^T x \mid x \in P'\}$, where $c = \text{pert}(c_0, \phi)$. We will denote this linear program by LP' .

Step 2: Let x_c denote the returned vertex in Step 1, which is optimal for the objective function $c^T x$. Identify an element a'_i of x_c that is in common with the optimal basis.

Step 3: Calculate an orthogonal matrix $Q \in \mathbb{R}^{n \times n}$ that rotates a'_i into the first unit vector e_1 and set LP' to the projection of the current LP' onto the orthogonal complement. Let P' denote the polyhedron of feasible solutions of LP' .

First note that the three steps are repeated at most n times during the algorithm. In Step 1 the shadow vertex algorithm is run once. Step 1 to Step 4 of Algorithm 1 can be performed in time $O(m)$ as we assumed P to be non-degenerate (this implies P' to be non-degenerate in each further step). Step 5 and Step 6 can be implemented with strongly polynomial running time in a tableau form, described in [2]. The tableau can be set up in time $O((m-d)d^3) = O(mn^3)$ where d is the dimension of P' . By Theorem 1 of [2] we can identify for a vertex on a path the row which leaves the basis and the row which is added to the basis in order to move to the next vertex in time $O(m)$ using the tableau. After that, the tableau has to be updated. This can be done in $O((m-d)d) = O(mn)$ steps. Using this and Theorem 4 we can compute the path from x_0 to x_c in expected time $O(mn^3 + mn \cdot (\frac{mn^2}{\delta^2} + \frac{m\sqrt{n}\phi}{\delta})) = O(\frac{m^2n^3}{\delta^2} + \frac{m^2n^{3/2}\phi}{\delta})$. Using that $\phi \leq \frac{8n^{3/2}}{\delta}$, as discussed above, yields a running time of $O(\frac{m^2n^3}{\delta^2})$.

Once we have calculated the basis of x_c we can easily compute the element a_i that is also an element of the optimal basis. Assume the rows a'_1, \dots, a'_n are the basis of x_c . Eisenbrand and Vempala discuss in [6] that we can solve the system of linear equations $[a'_1, \dots, a'_n]\mu = c$ and choose the row for which the coefficient μ_i is maximal. Then a'_i is part of the optimal basis. As a consequence, Step 2 can be performed in time $O(n^3)$. Moreover solving a system of linear equations is possible in strongly polynomial time using Gaussian elimination.

In Step 3, we compute an orthogonal matrix $Q \in \mathbb{R}^{d \times d}$ such that $e_1 Q = a_i$. Since Q is orthogonal we obtain $e_1 = a_i Q^T$ and thus, the first row of Q is given by a_i . Hence, it is sufficient to compute an orthonormal basis including a_i . This is possible in strongly polynomial time $O(d^3) = O(n^3)$ using the Gram-Schmidt process.

Since all Steps are repeated in this order at most n times we obtain a running time $O(\frac{m^2n^4}{\delta^2})$ for the repeated shadow vertex algorithm.

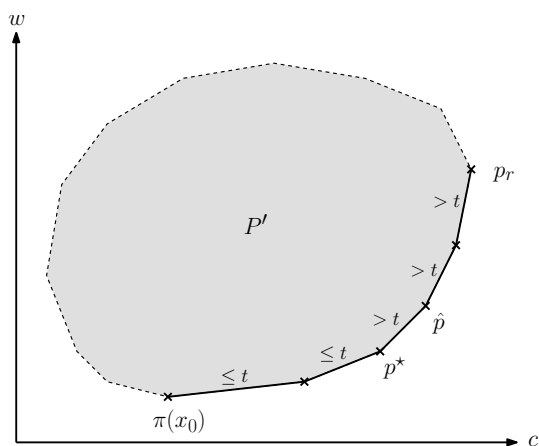
► **Theorem 5.** *The repeated shadow vertex algorithm has a running time of $O(\frac{m^2n^4}{\delta^2})$.*

The entries of both c and λ in Algorithm 1 are continuous random variables. In practice it is, however, more realistic to assume that we can draw a finite number of random bits. In the full version of this paper we show that our algorithm only needs to draw $\text{poly}(\log m, n, \log(1/\delta))$ random bits in order to obtain the expected running time stated in Theorem 2 if δ (or a good lower bound for it) is known. However, if the parameter δ is not known upfront and only discrete random variables with a finite precision can be drawn, we have to modify the shadow vertex algorithm. This will give us an additional factor of $O(n)$ in the expected running time.

3 Analysis of the Shadow Vertex Algorithm

For given linear functions $L_1: \mathbb{R}^n \rightarrow \mathbb{R}$ and $L_2: \mathbb{R}^n \rightarrow \mathbb{R}$ we denote by $\pi = \pi_{L_1, L_2}$ the function $\pi: \mathbb{R}^n \rightarrow \mathbb{R}^2$, given by $\pi(x) = (L_1(x), L_2(x))$. Note that n -dimensional vectors can be treated as linear functions. By $P' = P'_{L_1, L_2}$ we denote the projection $\pi(P)$ of the polytope P onto the Euclidean plane, and by $R = R_{L_1, L_2}$ we denote the path from the bottommost vertex of P' to the rightmost vertex of P' along the edges of the lower envelope of P' .

Our goal is to bound the expected number of edges of the path $R = R_{c, w}$, which is random since c and w are random. Each edge of R corresponds to a slope in $(0, \infty)$. These slopes are pairwise distinct with probability one (see Lemma 7). Hence, the number of edges of R equals the number of distinct slopes of R .



■ **Figure 1** Slopes of the vertices of R .

► **Definition 6.** For a real $\varepsilon > 0$ let \mathcal{F}_ε denote the event that there are three pairwise distinct vertices z_1, z_2, z_3 of P such that z_1 and z_3 are neighbors of z_2 and such that

$$\left| \frac{w^T \cdot (z_2 - z_1)}{c^T \cdot (z_2 - z_1)} - \frac{w^T \cdot (z_3 - z_2)}{c^T \cdot (z_3 - z_2)} \right| \leq \varepsilon.$$

Note that if event \mathcal{F}_ε does not occur, then all slopes of R differ by more than ε . Particularly, all slopes are pairwise distinct. First of all we show that event \mathcal{F}_ε is very unlikely to occur if ε is chosen sufficiently small. The proof of the following lemma is almost identical to the corresponding proof in [4] except that we need to adapt it to the different random model of c .

► **Lemma 7.** *The probability of event \mathcal{F}_ε tends to 0 for $\varepsilon \rightarrow 0$.*

Let p be a vertex of R , but not the bottommost vertex $\pi(x_0)$. We call the slope s of the edge incident to p to the left of p *the slope of p* . As a convention, we set the slope of $\pi(x_0)$ to 0 which is smaller than the slope of any other vertex p of R .

Let $t \geq 0$ be an arbitrary real, let p^* be the rightmost vertex of R whose slope is at most t , and let \hat{p} be the right neighbor of p^* , i.e., \hat{p} is the leftmost vertex of R whose slope exceeds t (see Figure 1). Let x^* and \hat{x} be the neighboring vertices of P with $\pi(x^*) = p^*$ and $\pi(\hat{x}) = \hat{p}$. Now let $i = i(x^*, \hat{x}) \in [m]$ be the index for which $a_i^T x^* = b_i$ and for which \hat{x} is the (unique) neighbor x of x^* for which $a_i^T x < b_i$. This index is unique due to the non-degeneracy of the polytope P . For an arbitrary real $\gamma \geq 0$ we consider the vector $\tilde{w} := w - \gamma \cdot a_i$.

► **Lemma 8** (Lemma 9 of [4]). *Let $\tilde{\pi} = \pi_{c, \tilde{w}}$ and let $\tilde{R} = R_{c, \tilde{w}}$ be the path from $\tilde{\pi}(x_0)$ to the rightmost vertex \tilde{p}_r of the projection $\tilde{\pi}(P)$ of polytope P . Furthermore, let \tilde{p}^* be the rightmost vertex of \tilde{R} whose slope does not exceed t . Then $\tilde{p}^* = \tilde{\pi}(x^*)$.*

Let us reformulate the statement of Lemma 8 as follows: The vertex \tilde{p}^* is defined for the path \tilde{R} of polygon $\tilde{\pi}(R)$ with the same rules as used to define the vertex p^* of the original path R of polygon $\pi(P)$. Even though R and \tilde{R} can be very different in shape, both vertices, p^* and \tilde{p}^* , correspond to the same solution x^* in the polytope P , that is, $p^* = \pi(x^*)$ and $\tilde{p}^* = \tilde{\pi}(x^*)$.

Lemma 8 holds for any vector \tilde{w} on the ray $\vec{r} = \{w - \gamma \cdot a_i \mid \gamma \geq 0\}$. As $\|w\| \leq n$ (see Section 2.1), we have $w \in [-n, n]^n$. Hence, ray \vec{r} intersects the boundary of $[-n, n]^n$ in a unique point z . We choose $\tilde{w} = \tilde{w}(w, i) := z$ and obtain the following result.

► **Corollary 9.** Let $\tilde{\pi} = \pi_{c, \tilde{w}(w, i)}$ and let \tilde{p}^* be the rightmost vertex of path $\tilde{R} = R_{c, \tilde{w}(w, i)}$ whose slope does not exceed t . Then $\tilde{p}^* = \tilde{\pi}(x^*)$.

Note that Corollary 9 only holds for the right choice of index $i = i(x^*, \hat{x})$. However, the vector $\tilde{w}(w, i)$ can be defined for any vector $w \in [-n, n]^n$ and any index $i \in [m]$. In the remainder, index i is an arbitrary index from $[m]$.

We can now define the following event that is parameterized in i , t , and a real $\varepsilon > 0$ and that depends on c and w .

► **Definition 10.** For an index $i \in [m]$ and a real $t \geq 0$ let \tilde{p}^* be the rightmost vertex of $\tilde{R} = R_{c, \tilde{w}(w, i)}$ whose slope does not exceed t and let y^* be the corresponding vertex of P . For a real $\varepsilon > 0$ we denote by $E_{i, t, \varepsilon}$ the event that the conditions

- $a_i^T y^* = b_i$ and
- $\frac{w^T(\hat{y} - y^*)}{c^T(\hat{y} - y^*)} \in (t, t + \varepsilon]$, where \hat{y} is the neighbor y of y^* for which $a_i^T y < b_i$,

are met. Note that the vertex \hat{y} always exists and that it is unique since the polytope P is non-degenerate.

Let us remark that the vertices y^* and \hat{y} , which depend on the index i , equal x^* and \hat{x} if we choose $i = i(x^*, \hat{x})$. For other choices of i , this is, in general, not the case.

Observe that all possible realizations of w from the line $L := \{w + x \cdot a_i \mid x \in \mathbb{R}\}$ are mapped to the same vector $\tilde{w}(w, i)$. Consequently, if c is fixed and if we only consider realizations of λ for which $w \in L$, then vertex \tilde{p}^* and, hence, vertex y^* from Definition 10 are already determined. However, since w is not completely specified, we have some randomness left for event $E_{i, t, \varepsilon}$ to occur. This allows us to bound the probability of event $E_{i, t, \varepsilon}$ from above (see proof of Lemma 12). The next lemma shows why this probability matters.

► **Lemma 11 (Lemma 12 from [4]).** For any $t \geq 0$ and $\varepsilon > 0$ let $A_{t, \varepsilon}$ denote the event that the path $R = R_{c, w}$ has a slope in $(t, t + \varepsilon]$. Then, $A_{t, \varepsilon} \subseteq \bigcup_{i=1}^m E_{i, t, \varepsilon}$.

With Lemma 11 we can now bound the probability of event $A_{t, \varepsilon}$. The proof of the next lemma is almost identical to the proof of Lemma 13 from [4]. The only differences to Lemma 13 from [4] are that we can now use the stronger upper bound $\|c\| \leq 2$ instead of $\|c\| \leq n$ and that we have more carefully analyzed the case of large t .

► **Lemma 12.** For any $\phi \geq \sqrt{n}$, any $t \geq 0$, and any $\varepsilon > 0$ the probability of event $A_{t, \varepsilon}$ is bounded by

$$\Pr[A_{t, \varepsilon}] \leq \frac{2mn^2\varepsilon}{\max\{\frac{n}{2}, t\} \cdot \delta^2} \leq \frac{4mn\varepsilon}{\delta^2}.$$

► **Lemma 13.** For any interval I let X_I denote the number of slopes of $R = R_{c, w}$ that lie in the interval I . Then, for any $\phi \geq \sqrt{n}$,

$$\mathbf{E}[X_{(0, n]}] \leq \frac{4mn^2}{\delta^2}$$

Proof. For a real $\varepsilon > 0$ let \mathcal{F}_ε denote the event from Definition 6. Recall that all slopes of R differ by more than ε if \mathcal{F}_ε does not occur. For $t \in \mathbb{R}$ and $\varepsilon > 0$ let $Z_{t, \varepsilon}$ be the random variable that indicates whether R has a slope in the interval $(t, t + \varepsilon]$ or not, i.e., $Z_{t, \varepsilon} = 1$ if $X_{(t, t + \varepsilon]} > 0$ and $Z_{t, \varepsilon} = 0$ if $X_{(t, t + \varepsilon]} = 0$.

Let $k \geq 1$ be an arbitrary integer. We subdivide the interval $(0, n]$ into k subintervals. If none of them contains more than one slope then the number $X_{(0, n]}$ of slopes in the

interval $(0, n]$ equals the number of subintervals for which the corresponding Z -variable equals 1. Formally

$$X_{(0,n]} \leq \begin{cases} \sum_{i=0}^{k-1} Z_{i \cdot \frac{n}{k}, \frac{n}{k}} & \text{if } \mathcal{F}_{\frac{n}{k}} \text{ does not occur,} \\ m^n & \text{otherwise.} \end{cases}$$

This is true because $\binom{m}{n-1} \leq m^n$ is a worst-case bound on the number of edges of P and, hence, of the number of slopes of R . Consequently,

$$\begin{aligned} \mathbf{E} [X_{(0,n]}] &\leq \sum_{i=0}^{k-1} \mathbf{E} [Z_{i \cdot \frac{n}{k}, \frac{n}{k}}] + \mathbf{Pr} [\mathcal{F}_{\frac{n}{k}}] \cdot m^n = \sum_{i=0}^{k-1} \mathbf{Pr} [A_{i \cdot \frac{n}{k}, \frac{n}{k}}] + \mathbf{Pr} [\mathcal{F}_{\frac{n}{k}}] \cdot m^n \\ &\leq \sum_{i=0}^{k-1} \frac{2mn^2 \cdot \frac{n}{k}}{\frac{n}{2}\delta^2} + \mathbf{Pr} [\mathcal{F}_{\frac{n}{k}}] \cdot m^n = \frac{4mn^2}{\delta^2} + \mathbf{Pr} [\mathcal{F}_{\frac{n}{k}}] \cdot m^n. \end{aligned}$$

The second inequality stems from Lemma 12. Now the lemma follows because the bound on $\mathbf{E} [X_{(0,n]}]$ holds for any integer $k \geq 1$ and since $\mathbf{Pr} [\mathcal{F}_\varepsilon] \rightarrow 0$ for $\varepsilon \rightarrow 0$ in accordance with Lemma 7. \blacktriangleleft

In [4] we only computed an upper bound for the expected value of $X_{(0,1]}$. Then we argued that the same upper bound also holds for the expected value of $X_{(1,\infty)}$. In order to see this, we simply exchanged the order of the objective functions in the projection π . Then any edge with a slope of $s > 1$ becomes an edge with slope $\frac{1}{s} < 1$. Hence the number of slopes in $[1, \infty)$ equals the number of slopes in $(0, 1]$ in the scenario in which the objective functions are exchanged. Due to the symmetry in the choice of the objective functions in [4] the same analysis as before applies also to that scenario.

We will now also exchange the order of the objective functions $w^T x$ and $c^T x$ in the projection. Since these objective functions are not anymore generated by the same random experiment, a simple argument as in [4] is not possible anymore. Instead we have to go through the whole analysis again. We will use the superscript -1 to indicate that we are referring to the scenario in which the order of the objective functions is exchanged. In particular, we consider the events $\mathcal{F}_\varepsilon^{-1}$, $A_{t,\varepsilon}^{-1}$, and $E_{i,t,\varepsilon}^{-1}$ that are defined analogously to their counterparts without superscript except that the order of the objective functions is exchanged. The proof of the following lemma is analogous to the proof of Lemma 7.

► **Lemma 14.** *The probability of event $\mathcal{F}_\varepsilon^{-1}$ tends to 0 for $\varepsilon \rightarrow 0$.*

► **Lemma 15.** *For any $\phi \geq \sqrt{n}$, any $t \geq 0$, and any $\varepsilon > 0$ the probability of event $A_{t,\varepsilon}^{-1}$ is bounded by*

$$\mathbf{Pr} [A_{t,\varepsilon}^{-1}] \leq \frac{2mn^{3/2}\varepsilon\phi}{\max\{1, \frac{nt}{2}\} \cdot \delta} \leq \frac{2mn^{3/2}\varepsilon\phi}{\delta}.$$

► **Lemma 16.** *For any interval I let X_I^{-1} denote the number of slopes of $R_{w,c}$ that lie in the interval I . Then*

$$\mathbf{E} [X_{(0,1/n]}^{-1}] \leq \frac{2m\sqrt{n}\phi}{\delta}.$$

Proof. As in the proof of Lemma 13 we define for $t \in \mathbb{R}$ and $\varepsilon > 0$ the random variable $Z_{t,\varepsilon}^{-1}$ that indicates whether $R_{w,c}$ has a slope in the interval $(t, t + \varepsilon]$ or not. For any integer $k \geq 1$

we obtain

$$\begin{aligned} \mathbf{E} \left[X_{\left(0, \frac{1}{n}\right]}^{-1} \right] &\leq \sum_{i=0}^{k-1} \mathbf{E} \left[Z_{i, \frac{1}{kn}, \frac{1}{kn}}^{-1} \right] + \mathbf{Pr} \left[\mathcal{F}_{\frac{1}{kn}}^{-1} \right] \cdot m^n \\ &= \sum_{i=0}^{k-1} \mathbf{Pr} \left[A_{i, \frac{1}{kn}, \frac{1}{kn}}^{-1} \right] + \mathbf{Pr} \left[\mathcal{F}_{\frac{1}{kn}}^{-1} \right] \cdot m^n \\ &\leq \sum_{i=0}^{k-1} \frac{2mn^{3/2}\phi}{kn\delta} + \mathbf{Pr} \left[\mathcal{F}_{\frac{1}{k2^\ell\sqrt{n}}}^{-1} \right] \cdot m^n = \frac{2m\sqrt{n}\phi}{\delta} + \mathbf{Pr} \left[\mathcal{F}_{\frac{1}{k2^\ell\sqrt{n}}}^{-1} \right] \cdot m^n. \end{aligned}$$

The second inequality stems from Lemma 15. Now the lemma follows because the bound holds for any integer $k \geq 1$ and $\mathbf{Pr} \left[\mathcal{F}_\varepsilon^{-1} \right] \rightarrow 0$ for $\varepsilon \rightarrow 0$ in accordance with Lemma 14. ◀

The following corollary directly implies Theorem 4.

► **Corollary 17.** *The expected number of slopes of $R = R_{c,w}$ is*

$$\mathbf{E} [X_{(0,\infty)}] = \frac{4mn^2}{\delta^2} + \frac{2m\sqrt{n}\phi}{\delta}.$$

Proof. We divide the interval $(0, \infty)$ into the subintervals $(0, n]$ and (n, ∞) . Using Lemma 13, Lemma 16, and linearity of expectation we obtain

$$\begin{aligned} \mathbf{E} [X_{(0,\infty)}] &= \mathbf{E} [X_{(0,n]}] + \mathbf{E} [X_{(n,\infty)}] = \mathbf{E} [X_{(0,n]}] + \mathbf{E} \left[X_{\left(0, \frac{1}{n}\right]}^{-1} \right] \\ &\leq \frac{4mn^2}{\delta^2} + \frac{2m\sqrt{n}\phi}{\delta}. \end{aligned}$$

In the second step we have exploited that by definition $X_{(a,b)} = X_{(1/b, 1/a)}^{-1}$ for any interval (a, b) . ◀

4 Conclusions

We have shown that the shadow vertex algorithm can be used to solve linear programs possessing the δ -distance property in strongly polynomial time with respect to n , m , and $1/\delta$. The bound we obtained in Theorem 2 depends quadratically on $1/\delta$. Roughly speaking, one term $1/\delta$ is due to the fact that the smaller δ the less random is the objective function $w^\top x$. This term could in fact be replaced by $1/\delta(B)$ where B is the matrix that contains only the rows that are tight for x . The other term $1/\delta$ is due to our application of the principle of deferred decisions in the proof of Lemma 12. The smaller δ the less random is $w(Z)$.

For packing linear programs, in which all coefficients of A and b are non-negative and one has $x \geq 0$ as additional constraint, it is, for example, clear that $x = 0^n$ is a basic feasible solution. That is, one does not need to run Phase 1. Furthermore as in this solution without loss of generality exactly the constraints $x \geq 0$ are tight, $\delta(B) = 1$ and one occurrence of $1/\delta$ in Theorem 2 can be removed.

Acknowledgments The authors would like to thank Friedrich Eisenbrand and Santosh Vempala for providing detailed explanations of their paper and the anonymous reviewers for valuable suggestions how to improve the presentation.

References

- 1 Nicolas Bonifas, Marco Di Summa, Friedrich Eisenbrand, Nicolai Hähnle, and Martin Niemeier. On sub-determinants and the diameter of polyhedra. In *Proceedings of the 28th ACM Symposium on Computational Geometry (SoCG)*, pages 357–362, 2012.
- 2 Karl Heinz Borgwardt. *A probabilistic analysis of the simplex method*. Springer-Verlag New York, Inc., New York, NY, USA, 1986.
- 3 Tobias Brunsch, Anna Großwendt, and Heiko Röglin. Solving totally unimodular LPs with the shadow vertex algorithm. *CoRR*, abs/1412.5381, 2014.
- 4 Tobias Brunsch and Heiko Röglin. Finding short paths on polytopes by the shadow vertex algorithm. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 279–290, 2013.
- 5 Martin E. Dyer and Alan M. Frieze. Random walks, totally unimodular matrices, and a randomised dual simplex algorithm. *Mathematical Programming*, 64:1–16, 1994.
- 6 Friedrich Eisenbrand and Santosh Vempala. Geometric random edge. *CoRR*, abs/1404.1568, 2014.
- 7 I. Heller. On linear systems with integral valued solutions. *Pacific Journal of Mathematics*, 7(3):1351–1364, 1957.
- 8 Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *Journal of the ACM*, 51(3):385–463, 2004.
- 9 Éva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986.

Improved Local Search for Geometric Hitting Set*

Norbert Bus¹, Shashwat Garg², Nabil H. Mustafa³, and Saurabh Ray⁴

- 1 Université Paris-Est, Laboratoire d'Informatique Gaspard-Monge.
busn@esiee.fr
- 2 Indian Institute of Technology, Delhi.
garg.shashwat@gmail.com
- 3 Laboratoire d'Informatique Gaspard-Monge, Université Paris-Est, Equipe A3SI, ESIEE Paris.
mustafan@esiee.fr
- 4 Computer Science Department, New York University, Abu Dhabi.
saurabh.ray@nyu.edu

Abstract

Over the past several decades there has been steady progress towards the goal of polynomial-time approximation schemes (PTAS) for fundamental geometric combinatorial optimization problems. A foremost example is the geometric hitting set problem: given a set P of points and a set \mathcal{D} of geometric objects, compute the minimum-sized subset of P that hits all objects in \mathcal{D} . For the case where \mathcal{D} is a set of disks in the plane, a PTAS was finally achieved in 2010, with a surprisingly simple algorithm based on local-search. Since then, local-search has turned out to be a powerful algorithmic approach towards achieving good approximation ratios for geometric problems (for geometric independent-set problem, for dominating sets, for the terrain guarding problem and several others).

Unfortunately all these algorithms have the same limitation: local search is able to give a PTAS, but with large running times. That leaves open the question of whether a better understanding – both combinatorial and algorithmic – of local search and the problem can give a better approximation ratio in a more reasonable time. In this paper, we investigate this question for hitting sets for disks in the plane. We present tight approximation bounds for $(3, 2)$ -local search and give an $(8 + \epsilon)$ -approximation algorithm with expected running time $\tilde{O}(n^{2.34})$; the previous-best result achieving a similar approximation ratio gave a 10-approximation in time $O(n^{15})$ – that too just for unit disks. The techniques and ideas generalize to $(4, 3)$ local search. Furthermore, as mentioned earlier, local-search has been used for several other geometric optimization problems; for all these problems our results show that $(3, 2)$ local search gives an 8-approximation and no better¹. Similarly $(4, 3)$ -local search gives a 5-approximation for all these problems.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases hitting sets, Delaunay triangulation, local search, disks, geometric algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.184

* The work of Nabil H. Mustafa in this paper has been partially supported by the grant ANR SAGA (JCJC-14-CE25-0016-01).

¹ This is assuming the use of the standard framework. Improvement of the approximation factor by using additional properties specific to the problem may be possible.

1 Introduction

The minimum hitting set problem is one of the most fundamental combinatorial optimization problems: given a range space (P, \mathcal{D}) consisting of a set P and a set \mathcal{D} of subsets of P called the *ranges*, the task is to compute the smallest subset $S \subseteq P$ that has a non-empty intersection with each of the ranges in \mathcal{D} . If there are no restrictions on the set system \mathcal{D} , then it is known that it is NP-hard to approximate the minimum hitting set within a logarithmic factor of the optimal [24]. A natural occurrence of the hitting set problem occurs when the range space \mathcal{D} is derived from geometry. Unfortunately, for most natural geometric range spaces, computing the minimum-sized hitting set remains NP-hard. For example, even the (relatively) simple case where \mathcal{D} is a set of unit disks in the plane is strongly NP-hard [18]. Therefore fast algorithms for computing provably good approximate hitting sets for geometric range spaces have been intensively studied for the past three decades. Since there is little hope of computing the minimum-sized hitting set for general geometric problems in polynomial time, effort has turned to approximating the optimal solution.

In this paper we will consider a fundamental case for geometric hitting sets, where the geometric objects are arbitrary radius disks in the plane (or halfspaces in \mathbb{R}^3). This has been the subject of a long line of investigation, for more than two decades. The case when all the disks have the same radius is easier, and has been studied in a series of works [9, 6, 10, 12, 13]. The problem becomes harder when the disk radii can be arbitrary. A first break-through for this problem came in 1994, when Bronnimann and Goodrich [8] proved the following interesting connection between the hitting-set problem, and ϵ -nets²: let (P, \mathcal{D}) be a range-space for which we want to compute a minimum hitting set. If one can compute an ϵ -net of size c/ϵ for the ϵ -net problem for (P, \mathcal{D}) in polynomial time, then one can compute a hitting set of size at most $c \cdot \text{OPT}$ for (P, \mathcal{D}) , where OPT is the size of the optimal (smallest) hitting set, in polynomial time. A shorter, simpler proof was given by Even *et al.* [15]. Recently, Agarwal and Pan [5] presented an algorithm that can compute hitting-sets for disks from ϵ -nets in time $O(n \log^6 n)$.

Local search. There is a fundamental limitation of the above technique: it *cannot* give better than constant-factor approximations. The reason is that the technique reduces the problem of computing a minimum size hitting set to the problem of computing a minimum sized ϵ -net. And it is known that for some constant $c \geq 2$, there do not exist ϵ -nets of size smaller than c/ϵ – even for halfspaces in 2D. This limitation was the main barrier towards better quality algorithms until the usefulness of local search algorithms was introduced.

There has been recent progress in breaking or improving the constant-approximation barriers for many geometric problems using very similar applications of local-search; e.g., independent set of non-piercing rectangles [4], independent set of pseudodisks [11], dominating sets in disk intersection graphs [17], terrain guarding problem [19] and several other problems. For the hitting set problem on (P, \mathcal{D}) , local-search works as follows: start with any hitting set $S \subseteq P$, and repeatedly decrease the size of S , if possible, by replacing k points of S with $< k$ points of $P \setminus S$. Call such an algorithm a $(k, k - 1)$ -local search algorithm. Mustafa and Ray [22] showed that a $(k, k - 1)$ -local search algorithm for the hitting set problem gives a $(1 + c/\sqrt{k})$ -approximation, for a fixed constant c , when the ranges are disks, or more generally, pseudo-disks in \mathbb{R}^2 . The running time of their algorithm to compute a $(1 + \epsilon)$ -approximation is $O(n^{O(1/\epsilon^2)})$.

² Given (P, \mathcal{D}) , an ϵ -net is a subset $S \subseteq P$ such that $D \cap S \neq \emptyset$ for all $D \in \mathcal{D}$ with $|D \cap P| \geq \epsilon n$.

■ **Table 1** Summary of previous work.

<i>Congruent disks</i>			<i>Arbitrary disks</i>		
	Quality	Time		Quality	Time
Călinsecu <i>et al.</i> [9]	108	$O(n^2)$	Bronniman <i>et al.</i> [8]	$O(1)$	$O(n^3)$
Ambhul <i>et al.</i> [6]	72	$O(n^2)$	Mustafa <i>et al.</i> [22]	$(1 + \epsilon)$	$n^{O(1/\epsilon^2)}$
Carmi <i>et al.</i> [10]	38	$O(n^6)$	Agarwal <i>et al.</i> [3]	$O(\log n)$	$\tilde{O}(n)$
Claude <i>et al.</i> [12]	22	$O(n^6)$	Agarwal <i>et al.</i> [5]	$O(1)$	$\tilde{O}(n)$
Fraser <i>et al.</i> [13]	18	$O(n^2)$	This paper	$8 + \epsilon$	$\tilde{O}(n^{7/3})$
Acharyya <i>et al.</i> [1]	$(9 + \epsilon)$	$O(n^{3+12/\epsilon})$	This paper	$5 + \epsilon$	$\tilde{O}(n^{3.75})$

Our Contributions. Both these approaches have to be evaluated on the questions of computational efficiency as well as approximation quality. In spite of all the progress, there remains a large gap between quality and efficiency – mainly due to the ugly trade-offs between running times and approximation factors; see Table 1 for the current state of the art. The algorithm of Agarwal and Pan [5] that rounds via ϵ -nets gives an $\tilde{O}(n)$ -time algorithm, but the constant in the approximation depends on the constant in the size of ϵ -nets, which is large. For disks in the plane, the current best size of ϵ -net is at least $40/\epsilon$ [23], yielding at best a 40 -approximation algorithm. At the other end, the $(k, k - 1)$ -local search algorithm [22] can compute solutions arbitrarily close to the optimal, but it is extremely inefficient, even for reasonable approximation factors. For example, it takes time $O(n^{66})$ to compute a 3-approximation [16]. Furthermore, note that any attempts at progress on improving local search must take into account that the hitting set problem for even unit disks is $W[1]$ -hard [20]; so it is unlikely that algorithms exist that do not have a dependency on $1/\epsilon$ in the exponent.

Therefore in this paper we undertake a closer study of $(k, k - 1)$ -local search for small values of k . Table 1 states our contributions. As our first result, we determine the exact limits of $(3, 2)$ -local search:

► **Theorem (Proof in Section 2).** A $(3, 2)$ -local search algorithm returns a 8-approximation to the minimum hitting set. Furthermore, there exist a set P of points and a set \mathcal{D} of disks where $(3, 2)$ local-search does not return hitting-sets of size less than 8 factor of the optimal hitting set.

Remark: In fact this immediately implies improved bounds for many other local search algorithms; e.g., it implies that the $(3, 2)$ -local search gives 8-approximation to the independent set of pseudodisks, dominating sets in disk intersection graphs, terrain guarding problem. We leave the details of these further applications to the full paper.

A straightforward algorithm for $(3, 2)$ -local search proceeds as follows: each $(3, 2)$ improvement step tries all $O(n^5)$ 5-tuples, and for each checks if it is indeed an improvement in time $O(n)$. The total number of steps for the whole algorithm can be $O(n)$, giving a $O(n^7)$ naive running time. We show how to perform this search more efficiently:

► **Theorem (Proof in Section 3).** A $(3, 2)$ -local search can be performed in expected time $O(n^{2.34})$.

In fact, these techniques can be generalized for larger values of k . For example, it can be shown that $(4, 3)$ -local search gives a 5-approximation in time $\tilde{O}(n^{3.75})$. As the details are similar, we leave the proof for the full version of the paper.

2 Analysis of Quality for Local Search

Let R be a region in the plane. We say that a point $p \in \mathbb{R}^2$ hits R if $p \in R$, and that a set of points X hits a set of regions \mathcal{R} if each region in \mathcal{R} is hit by some point in X . We denote by $\mathcal{H}(P, \mathcal{R})$ the set system $(P, \{R \cap P : R \in \mathcal{R}\})$ induced by P and \mathcal{R} . A hitting set for $\mathcal{H}(P, \mathcal{R})$ is a subset of P which hits \mathcal{R} . A hitting set of the smallest cardinality is called the minimum hitting set and its size is denoted $\text{OPT}(P, \mathcal{R})$ (or simply OPT when it is clear from the context). From now onwards, P denotes a set of points and \mathcal{D} denotes a set of (circular) disks in the plane. Our goal is to compute a hitting set for $\mathcal{H}(P, \mathcal{D})$ of a small size efficiently.

The analysis of the approximation factor achieved by a $(k, k-1)$ -local search depends on the following theorem on planar bipartite graphs.

► **Theorem 1.** [11, 22] *Let $G = (R, B, E)$ be a bipartite planar graph on red and blue vertex sets R and B , such that for every subset $B' \subseteq B$ of size at most k , where k is a large enough number, $|N_G(B')| \geq |B'|$. Then, $|B| \leq (1 + c/\sqrt{k})|R|$, where c is a constant.*

Here $N_G(B')$ denotes the set of neighbors of the vertices in B' in G . The proof of the above theorem, which relies on planar graph separators, requires k to be quite large, thereby limiting the practical utility of the above theorem. A priori, it is not clear whether the theorem holds at all for small values of k . For instance, one can easily see that for $k = 2$ there is no upper bound on $|B|/|R|$ (e.g., consider complete bipartite graph where B is arbitrarily large and $|R| = 2$). However, for $k = 3$, we show a small bound of 8 on $|B|/|R|$, and then prove that it is, in fact, optimal.

► **Theorem 2.** *Let $G = (R, B, E)$ be a bipartite planar graph on red and blue vertex sets R and B , such that for every subset $B' \subseteq B$ of size at most 3, $|N_G(B')| \geq |B'|$. Then, $|B| \leq 8|R|$ and this bound is tight.*

Proof. Let $n_b = |B|$ and $n_r = |R|$. Our goal is to prove that $n_b \leq 8n_r$. Note that no vertex in B can have degree 0, otherwise the neighborhood of such a vertex is of size 0, violating the conditions of the theorem. We make a new graph G' by adding edges in G to all vertices of B which have degree 1 in G . This can always be done while maintaining the planarity and bipartiteness of the graph as any such vertex v must lie in a face which has at least two vertices of R , at least one of which is not adjacent to v . Thus in G' every vertex in B has degree at least 2. Let n_{b_2} be the number of vertices of B which have degree 2 and $n_{b_{\geq 3}} = n_b - n_{b_2}$ be the number of vertices of B which have degree at least 3 in G' . Since G' is planar and bipartite the number of edges in $G' \leq 2(n_b + n_r)$. This implies that $2n_{b_2} + 3n_{b_{\geq 3}} \leq 2n_b + 2n_r$. Since $n_b = n_{b_2} + n_{b_{\geq 3}}$, we obtain $n_{b_{\geq 3}} \leq 2n_r$.

We now show that $n_{b_2} \leq 6n_r$. To do that we construct a graph H with vertex set R as follows: two vertices $r_1 \in R$ and $r_2 \in R$ are adjacent in H iff there is at least one vertex $b \in B$ of degree 2 which is adjacent to both r_1 and r_2 in G' . Note that H is planar since the edge between r_1 and r_2 can be routed via one such b . Note that for the same pair $\{r_1, r_2\}$ there cannot be three vertices $b_1, b_2, b_3 \in B$ of degree 2 each that are adjacent to both r_1 and r_2 since in that case the neighborhood of the set $\{b_1, b_2, b_3\}$ is of size 2 violating the conditions of the theorem. Therefore, each vertex $b \in B$ of degree 2 corresponds to an edge in H and each edge has at most two vertices in B that correspond to it. Since the number of edges in H is at most $3|R| = 3n_r$, we conclude that $n_{b_2} \leq 6n_r$. Thus $n_b = n_{b_2} + n_{b_{\geq 3}} \leq 6n_r + 2n_r = 8n_r$. ◀

We now show that the bound given above is tight. However, that still leaves open the possibility that, by exploiting other properties of disks, a $(3, 2)$ -local search could give a

better approximation for the problem of computing minimum hitting sets for disks in the plane. The following theorem rules this out.

► **Theorem 3.** *For any $\delta > 0$, there exists an integer n_0 such that one can construct a set \mathcal{D} of disks in the plane, a set of points P , $|P| \geq n_0$, and a subset $B \subseteq P$ s.t. i) B is a hitting set for $\mathcal{H}(P, \mathcal{D})$, ii) $|B| \geq (8 - \delta)\text{OPT}$ and iii) there are no subsets $X \subseteq B$ and $Y \subseteq P \setminus B$, $|Y| < |X| \leq 3$, s.t. $(B \setminus X) \cup Y$ is a hitting set for $\mathcal{H}(P, \mathcal{D})$.*

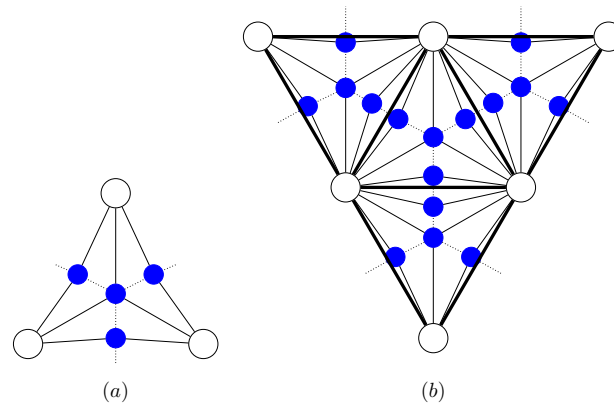
Proof. We first construct a bipartite graph $G = (R, B, E)$ that satisfies the conditions of Theorem 2 and $|B| \geq (8 - \delta)|R|$. Let L be the triangular lattice, and take a large equilateral triangle Δ aligned with the edges of L (so that $L \cap \Delta$ triangulates Δ) and containing many faces of the lattice. Then replace each face of the lattice by the block of the type shown in Figure 1(a). The corner vertices (unshaded) of the block map to the corner vertices of the face, while the other vertices (shaded) in the block lie in the interior of the face. Let R be the set of vertices of L lying in Δ and let B be the set of vertices lying in the interior of the faces in $L \cap \Delta$. The blocks together define a bipartite graph (see Figure 1(b) for a small example with four blocks put together). Note that each face in $L \cap \Delta$ contains four points of B , and if Δ is large enough, the number of faces of L in Δ is nearly twice the number of vertices of L in Δ . The size of Δ can be chosen, depending on δ , such that the number of faces of L is at least $(2 - \delta/4)$ times the number of vertices of L . Thus we get that $|B| \geq (8 - \delta)|R|$. It can be verified by inspection that there is no subset of B of size at most 3 with a smaller neighborhood. This shows that the bound in Theorem 2 is tight within additive constants.

Now, we extend G to a triangulation by including the dotted edges in the blocks. Note that there are some dotted edges going between blocks. We also put an additional vertex in the outer face and connect it to all vertices in the outer face of G (i.e. we stellate the outer face). The resulting graph, call it Ξ , is triangulated (i.e., each face is of size 3) and furthermore it is 4-connected since, as can be verified by inspection, there is no separating triangle (a non-facial cycle of length 3). By a theorem of Dillencourt and Smith (Theorem 3.5 in [14]), there exists an embedding of Ξ in the plane so that Ξ is the Delaunay triangulation of its vertices. Abusing notation, we refer to the embedding as Ξ and we refer to the embedding of a vertex v in Ξ as v .

Let R and B thus be the two sets of points. We set $P = R \cup B$, and construct \mathcal{D} by taking for each edge e in G a disk that contains exactly the two end points of e among all the vertices in Ξ . This is possible because Ξ is now a Delaunay triangulation of the points in P . By construction, each disk in \mathcal{D} contains exactly one point from each of the sets R and B and thus both the sets are hitting sets for $\mathcal{H}(P, \mathcal{D})$. Since OPT is the size of the smallest hitting set, $\text{OPT} \leq |R|$ and therefore $|B| \geq (8 - \delta)\text{OPT}$. Consider a local improvement step where we seek to decrease the size of the hitting set B by removing some subset $X \subseteq B$ of size at most 3 and adding a smaller set Y outside B (i.e., $Y \subseteq R$) so that $(B \setminus X) \cup Y$ is a hitting set for \mathcal{D} . Let x be one of the points in X . Observe that then all neighbors of x in G must be in Y since for each neighbor y of x , there is a disk in \mathcal{D} which contains only the two points x and y among all the points in $R \cup B$. This means that $|Y| \geq |N_G(X)|$. Since for any X of size at most 3, $|N_G(X)| \geq |X|$, we have that $|Y| \geq |X|$ implying that such a local improvement is not possible. ◀

As mentioned before, a $(4, 3)$ -local search gives a 5 approximation (proof in the full version of the paper).

► **Theorem 4.** *Let $G = (R, B, E)$ be a bipartite planar graph on red and blue vertex sets R and B , such that for every subset $B' \subseteq B$ of size at most 4, $|N_G(B')| \geq |B'|$. Then, $|B| \leq 5|R|$.*



■ **Figure 1** Unshaded vertices correspond to red and shaded to blue vertices. The dotted lines show a triangulation. The edges of L are drawn in bold, while the dotted edges and the edges of the lattice L are not part of the graph. We tile the triangles in (a) as shown in (b). The ratio of shaded to unshaded vertices goes to 8 as size of the tiling is increased. Connecting the vertices at the boundary of the tiling to a new vertex gives a 4-connected graph.

3 An Algorithm for (3, 2) Local Search

Our algorithm is based on local search. It starts with a hitting set and repeatedly tries to make local improvements. Let S be a hitting set for $\mathcal{H}(P, \mathcal{D})$. Let $X \subseteq S$ and $Y \subseteq P$. We say that (X, Y) is a *local improvement pair* with respect to S and $\mathcal{H}(P, \mathcal{D})$ if $|Y| < |X|$ and $(S \setminus X) \cup Y$ is a hitting set for $\mathcal{H}(P, \mathcal{D})$. Such a local improvement reduces the size of the hitting set by $|X| - |Y|$. We will refer to this quantity as the *profit* of the local improvement and the local improvement pair. We say that $X \subseteq S$ is *locally improvable* with respect to S and $\mathcal{H}(P, \mathcal{D})$ if there exists a $Y \subseteq P$ such that (X, Y) is a local improvement pair. If (X, Y) is a local improvement pair, we say that Y can *locally replace* X .

Let S be a hitting set for $\mathcal{H}(P, \mathcal{D})$. For any $s \in S$, we denote by $\mathcal{D}(s)$ the set of disks in \mathcal{D} that are hit by s but not by any other point in S . We will call the disks in $\mathcal{D}(s)$ the *personal disks* of s . We will denote the region $\bigcap_{D \in \mathcal{D}(s)} D$ by $R(s)$ and call it the *personal region* of s . The notations $\mathcal{D}(s)$ and $R(s)$ are always with respect to a set system $\mathcal{H}(P, \mathcal{D})$ and a hitting set S . These things that are not explicit in the notation will be clear from the context. We also extend the same definitions for sets of points: for a set $X \subseteq S$, let $\mathcal{D}(X)$ be the set of disks in \mathcal{D} which contain only points of X . We call these the *personal disks* of X . The *personal region* of X is $R(X) = \bigcap_{D \in \mathcal{D}(X)} D$. A set of regions \mathcal{R} are *pseudodisks* if they are simply connected and the boundaries of every pair of regions in \mathcal{R} intersect at most twice.

Preparing for the algorithm. We prove a few results useful for describing the algorithm.

► **Lemma 5.** *Let S be a hitting set for $\mathcal{H}(P, \mathcal{D})$. If $|S| > 8 \cdot \text{OPT}(P, \mathcal{D}) + 3t$, for some integer $t \geq 0$, then there exist $t + 1$ disjoint subsets X_0, \dots, X_t of S , each of which is of size 3 and is locally improvable with respect to S in $\mathcal{H}(P, \mathcal{D})$.*

Proof. The proof is by induction. The statement is true for $t = 0$: if there is no locally improvable set X of size 3, then taking $B = S$ and $R = O$, where O is the optimal hitting set for $\mathcal{H}(P, \mathcal{D})$ and applying Theorem 2, we get that $|S| \leq 8\text{OPT}(P, \mathcal{D})$. Assume inductively that the lemma is true for $t - 1$, and let the t disjoint sets of S be X_0, \dots, X_{t-1} . It remains

to construct the set X_t . Let $Z = \bigcup_{i=0}^{t-1} X_i$. Let $P' = P \setminus Z$, $\mathcal{D}' = \{D \in \mathcal{D} : D \cap Z = \emptyset\}$ and $S' = S \setminus Z$. Clearly S' is a hitting set for $\mathcal{H}(P', \mathcal{D}')$. Moreover,

► **Claim 1.** $\text{OPT}(P', \mathcal{D}') \leq \text{OPT}(P, \mathcal{D})$.

Proof. Take any hitting set A for $\mathcal{H}(P, \mathcal{D})$. Then any point $a \in A$ that hits a disk in \mathcal{D}' must belong to P' : otherwise $a \in P \setminus P' = Z$, and we had constructed \mathcal{D}' by removing all the disks hit by Z from \mathcal{D} . Therefore all the points in A hitting \mathcal{D}' belong to P' , and form a hitting set in P' for $\mathcal{H}(P', \mathcal{D}')$ of size at most $|A|$. ◀

Therefore, $|S'| = |S| - 3t > 8 \cdot \text{OPT}(P, \mathcal{D}) \geq 8 \cdot \text{OPT}(P', \mathcal{D}')$, and any hitting set for $\mathcal{H}(P, \mathcal{D})$ contains a hitting set for $\mathcal{H}(P', \mathcal{D}')$. Now, using the Theorem 2 for $t = 0$ on S' and (P', \mathcal{D}') , the fact that $|S'| > 8 \cdot \text{OPT}(P', \mathcal{D}')$ implies that there is set $X_t \subseteq S'$ of size 3 and a set $Y \subseteq P'$ of size 2 such that $(S' \setminus X_t) \cup Y$ is a hitting set for $\mathcal{H}(P', \mathcal{D}')$. This means that $(S' \setminus X_t) \cup Y \cup Z$ is a hitting set for $\mathcal{H}(P, \mathcal{D})$ since all disks in $\mathcal{D} \setminus \mathcal{D}'$ intersect Z . In other words, $(S \setminus X_t) \cup Y$ is a hitting set for $\mathcal{H}(P, \mathcal{D})$ since $S' \cup Z = S$ and $X_t \cap Z = \emptyset$. That is, X_t is locally improvable with respect to S in $\mathcal{H}(P, \mathcal{D})$. Since $X_t \subseteq S'$ and $S' \cap Z = \emptyset$, X_t is disjoint from the other X_i 's. ◀

The following key structural property is crucial (due to shortage of space, proof omitted):

► **Lemma 6.** *Let S be a hitting set for $\mathcal{H}(P, \mathcal{D})$. Then the personal regions of the points in S form a collection of pseudodisks.*

► **Lemma 7.** *Let S be a hitting set for $\mathcal{H}(P, \mathcal{D})$. Suppose that we are given two sets $X \subseteq S$ and $Y \subseteq P$ such that $|Y| = O(1)$, $|X| > 4|Y|$ and for each $x \in X$, Y hits $\mathcal{D}(x)$, the personal disks of x . Then there exists a set $X' \subseteq X$ of size $\Omega(|X|)$ such that (X', Y) is a local improvement pair with respect to S and $\mathcal{H}(P, \mathcal{D})$. Furthermore, given X and Y , X' can be computed in time $O(|X| \log |X|)$.*

Proof. Consider the Delaunay triangulation of the points in X , and let X' be an independent-set in this Delaunay graph. First we show that $(S \setminus X') \cup Y$ is a hitting set for $\mathcal{H}(P, \mathcal{D})$. Consider a disk D that is not hit by $S \setminus X'$. Since D is hit by S (S being a hitting set for $\mathcal{H}(P, \mathcal{D})$), D contains at least one point of X' . If D contains exactly one point $x' \in X'$ then D is hit by Y since $D \in \mathcal{D}(x')$ and Y hits $\mathcal{D}(x')$. Otherwise, D contains at least two points of X' , in which case it must contain some point of $x \in X \setminus X' \subseteq S \setminus X'$ since X' is an independent set in the Delaunay triangulation of X (and by the fact that subgraph of the Delaunay graph induced by the set of points of X inside any disk is connected).

The Delaunay triangulation can be constructed in $O(|X| \log |X|)$ time. If $|X| \leq 5|Y|$, i.e. $|X| = O(1)$, we find an independent set of size at least $\lceil |X|/4 \rceil > |Y|$ in the Delaunay graph in $O(1)$ time by brute force; the existence of such an independent set follows from the 4-color theorem on planar graphs. If $|X| > 5|Y|$, we compute a 5-coloring of the Delaunay graph in $O(|X|)$ time and take the largest color class as X' . Thus $|X'| \geq \lceil |X|/5 \rceil > |Y|$.

Therefore $|X'| > |Y|$, and so (X', Y) is a local improvement pair. ◀

The next two lemmas show that one can efficiently preprocess disks to answer containment queries in logarithmic time. Due to the shortage of space, the proofs are omitted.

► **Lemma 8.** *Let \mathcal{D} be a set of m disks in the plane having a common intersection region, say R . Then the boundary of R is composed of $O(m)$ circular arcs, and can be computed in $O(m \log m)$ expected time. We can also construct, in $O(m \log m)$ time, a data structure which, for any given query point q , answers whether $q \in R$ in $O(\log m)$ time.*

► **Lemma 9.** *Let P be a set of n points in the plane and let \mathcal{D} be a set of pseudodisks, the boundary of each being composed of circular arcs. For any constant C , we can compute, for each $p \in P$ that lies in at most C pseudodisks, the exact set of pseudodisks it hits in $O(n \log m)$ time, where m is the total number of arcs in all the pseudodisks.*

The Algorithm. We now describe our algorithm for computing a small hitting set for $\mathcal{H}(P, \mathcal{D})$. We first compute a hitting set S of size $O(\text{OPT})$ [5]. We also assume that we know the value of $\text{OPT} = \text{OPT}(P, \mathcal{D})$ although it suffices to guess the value of OPT within a $(1 + \epsilon)$ factor which can be done in $O(1/\log(1 + \epsilon))$ guesses since we know OPT within a constant factor. Throughout this section, we will use n as the total input size. We therefore upper bound $|P|$ and $|\mathcal{D}|$ by n .

Next, for a given $\epsilon > 0$, we prune the input so that no point is contained in more than $\Delta = n/(\epsilon \cdot \text{OPT})$ disks. This can be done by iterating over each point $p \in P$ and computing the number of disks $\mathcal{D}' \subseteq \mathcal{D}$ that contain p . If $|\mathcal{D}'| \geq \Delta$, remove the disks in \mathcal{D}' from \mathcal{D} and add the point p to the set Q (which is initially empty). Note that as we go over the points the set \mathcal{D} changes but we do not change the value of Δ . Since each time we add a point to Q , we remove at least Δ disks from \mathcal{D} , $|Q| \leq n/\Delta = \epsilon \cdot \text{OPT}$. We can thus add the set Q to our hitting set at the cost of an added ϵ in our approximation factor. This preprocessing procedure takes $O(n^2)$ time (this will not be the bottleneck of our algorithm).

After preprocessing, we pass P and \mathcal{D} to Algorithm 1 which we describe now. It requires an initial hitting set S of size $O(\text{OPT})$ which we obtain from [5]. The goal of Algorithm 1 is to compute a hitting set whose size is at most $(8 + \epsilon) \cdot \text{OPT}$. We compute a value $t = |S| - 8 \cdot \text{OPT}$ which indicates how far we are from the solution we seek. As we will see, when t is large, progress can be made quickly. However as we approach the quantity $8 \cdot \text{OPT}$, progress becomes slower and slower. The algorithm uses only local improvements of the type (X, Y) where $|Y| \leq 2$. Throughout the algorithm we maintain for each $D \in \mathcal{D}$, the number of points, N_D , it contains from S . Initially computing N_D for each disk takes $O(n^2)$ time. After that we need to update these quantities only when a local improvement (X, Y) happens. We update N_D as follows: $N_D = N_D - |D \cap X| + |D \cap Y|$. Since $|Y|$ is always at most 2 in our algorithm, naively this takes time $O(n|X|)$ for updating all disks in \mathcal{D} . Since such a local improvement decreases the size of the hitting set S by $|X| - 2 = \Omega(|X|)$, the overhead for maintaining N_D is $O(n)$ per improvement. Let $\text{LocallyImprove}(X, Y)$ be the procedure that updates S to $(S \setminus X) \cup Y$ and updates N_D for each disk as mentioned above.

In each iteration of the **while** loop in Algorithm 1, we first construct a range reporting data structure [2] for the points in S so that given any disk D , we can find the set of points in $D \cap S$ in time $O(\log n + |D \cap S|)$. We then use this data structure to compute the personal disks of each $s \in S$ as follows. Iterate over each disk $D \in \mathcal{D}$ and if $N_D = 1$, use the reporting data structure to find the single point $s \in S$ that is contained by D . We then add D to the (initially empty) list of personal disks of s . Since each query takes $O(\log n)$ time, the total time taken to compute the personal disks is $O(n \log n)$. If we find some point $s \in S$ for which $\mathcal{D}(s) = \emptyset$, we can just remove s from the current hitting set. In other words we do a local improvement $(\{s\}, \emptyset)$.

The algorithm iterates over the points of P in random order, considering the possibility of replacing each point in a local-improvement step. Say the current point being considered is p_1 ; the goal is to find a point p_2 so that $\{p_1, p_2\}$ can replace a large set $X \subseteq S$, i.e., a local improvement pair $(X, \{p_1, p_2\})$ of large profit. If we can find such a profitable local improvement, we make the improvement, exit from the **for** loop, and continue with the next iteration of the **while** loop. Otherwise, we continue with the next point in the random

Algorithm 1: Algorithm for $(8 + \epsilon)$ -approximation.

Data: A point set P , a set of disks \mathcal{D} , a hitting set S of $\mathcal{H}(P, \mathcal{D})$ with $|S| = O(\text{OPT}(\mathcal{H}(P, \mathcal{D})))$, the size of the optimal hitting set $\text{OPT} = \text{OPT}(\mathcal{H}(P, \mathcal{D}))$, and a parameter $\epsilon > 0$.

- 1 For each disk $D \in \mathcal{D}$ compute $N_D = |D \cap S|$ // takes $O(n^2)$ time
- 2 **while** $t = |S| - 8 \cdot \text{OPT} > \epsilon \cdot \text{OPT}$ **do**
- 3 Construct a range reporting data structure for S for disk ranges
- 4 For each $s \in S$ compute $\mathcal{D}(s) = \{D \in \mathcal{D} : D \cap s = \{s\}\}$ // use range reporting
- 5 **if** $\mathcal{D}(s) = \emptyset$ for some $s \in S$ **then**
- 6 LocallyImprove($\{s\}, \emptyset$) // s is dropped from the hitting set
- 7 **continue** // with the next iteration of the while loop on line 2.
- 8 $\pi =$ A random permutation of the points in P
- 9 **for** $i = 1$ to $|P|$ **do**
- 10 $p_1 = \pi_i$
- 11 **for** each $s \in S$ **do**
- 12 Compute: $\mathcal{D}'(s) = \{D \in \mathcal{D}(s) : p_1 \notin D\}$, $R'(s) = \bigcap_{D \in \mathcal{D}'(s)} D$
- // The above loop takes $O(n \log n)$ time
- 13 Let $\mathcal{R}' = \{R'(s) : s \in S\}$ // \mathcal{R}' is a set of pseudodisks
- 14 $M = \{s \in S : R'(s) = \emptyset\}$
- 15 **for** each $p \in P$ **do**
- 16 Compute $\alpha(p)$ s.t. $0.9 \cdot \text{depth}(p, \mathcal{R}') \leq \alpha(p) \leq \text{depth}(p, \mathcal{R}')$
- // $\text{depth}(p, \mathcal{R}')$ denotes the number of regions in \mathcal{R}' containing p
- 17 Let $q = \arg \max_{p \in P} \alpha(p)$
- 18 Set $\beta = \max\{\sqrt{t}, \epsilon t \cdot \text{OPT}/n\}$
- 19 **if** $|M|/5 + \alpha(q) \geq \frac{i\beta}{cn \log n}$ **then**
- 20 Compute $S'(q) = \{s \in S : q \in R'(s)\}$ // Note that $|S'(q)| = \text{depth}(q, \mathcal{R}')$
- 21 **if** $|S'(q) \cup M| \geq 9$ **then**
- 22 Compute an independent set $X \subseteq S'(q) \cup M$ in the Delaunay triangulation of $S'(q)$ of size at least 3 and $\Omega(|S'(q) \cup M|)$
- // $O(n \log n)$ time
- 23 LocallyImprove($X, \{p_1, p_2 = q\}$)
- 24 **break** // exit for loop
- 25 **else**
- 26 For each $p_2 \in P$, set $S'(p_2) = \{s \in S : p_2 \in R'(s)\}$ // $O(n \log n)$ time
- // Since $|S'(q) \cup M| \leq 8$, $|S'(p_2) \cup M| \leq \lceil 8/0.9 \rceil = 8$ for all $p_2 \in P$
- 27 Enumerate all pairs (X, p_2) where $p_2 \in P$, $X \subseteq S'(p_2) \cup M$ and $|X| \leq 3$
- 28 **if** for any (X, p_2) enumerated, $(X, \{p_1, p_2\})$ is a local improvement pair **then**
- 29 LocallyImprove($X, \{p_1, p_2\}$)
- 30 **break** // exit for loop

ordering. For any pair of points $Y = \{p, q\} \subseteq P$, denote by $\rho(Y)$ the number of points in S all of whose personal disks are hit by Y . For a point $p \in P$, we use $\rho(p)$ to denote $\max_{q \in P \setminus S} \rho(\{p, q\})$. Call a point $p \in P$ *useful* if there exists some $q \in P$ so that for some $X \subseteq S$, $(X, \{p, q\})$ is a local improvement pair.

Analysis of the Algorithm.

► **Lemma 10.** *If p_1 is useful, we can compute in $O(n \log^2 n)$ time a local improvement of profit $\Omega(\rho(p_1))$.*

Proof. Let us start by considering how we could compute $\rho(p_1)$. In order to compute $\rho(p_1)$, we need to find a point q so that the number of points $s \in S$ whose personal disks are hit by $\{p_1, q\}$ is maximized. To do this, we first compute for each $s \in S$, the set $\mathcal{D}'(s)$ of disks in $\mathcal{D}(s)$ that are not hit by p_1 . For each $s \in S$, we then construct the region $R'(s)$ by taking the intersection of the disks in $\mathcal{D}'(s)$. Let $\mathcal{R}' = \{R'(s) : s \in S\}$. For some $s \in S$, $\mathcal{D}'(s)$ may be empty and consequently some of the regions in \mathcal{R}' are empty. Let $M = \{s \in S : \mathcal{D}'(s) = \emptyset\}$. The personal disks of the points in M are hit by p_1 alone. The regions in \mathcal{R}' define an arrangement of pseudodisks (Lemma 6). In this arrangement we seek to find a point $q \in P$ of maximum depth. However, instead of finding a point with maximum depth, we find a point whose depth is within a constant factor of the maximum. We construct, in $O(n \log n)$ time, an approximate depth query data structure for the pseudodisks in \mathcal{R}' using Corollary 5.9 in [7] with a constant $\epsilon' \leq 0.1$. Then for each point $p \in P$, compute a value $\alpha(p)$ s.t. $0.9 \text{ depth}(p, \mathcal{R}') \leq \alpha(p) \leq \text{depth}(p, \mathcal{R}')$ where $\text{depth}(p, \mathcal{R}')$ denotes the depth of p in the arrangement of regions in \mathcal{R}' . This takes $O(\log^2 n)$ time per point and so the overall time taken is $O(n \log^2 n)$. We then take the point p with the maximum $\alpha(p)$ as q . Observe that $|M| + \alpha(q) = \Theta(\rho(p_1))$. We first compute the set $S'(q) = \{s \in S : q \in R'(s)\}$. Note that $|S'(q)| = \text{depth}(q, \mathcal{R}') \geq \alpha(q)$. There are two cases to consider:

Case 1: $|S'(q) \cup M| > 8$. In this case, we set $p_2 = q$ and let $Y = \{p_1, p_2\}$. Using Lemma 7 (note here that $|S'(q) \cup M| > |Y|$), we can find a subset $X \subseteq S'(q) \cup M$ so that $X = \Omega(|S'(q) \cup M|)$ so that $(X, \{p_1, p_2\})$ is a local improvement pair. Note that $|X|$ is $\Omega(\rho(p_1))$. Thus in this case, we conclude that p_1 is useful and indeed we have found a local improvement that decreases the size of the current hitting set by $\Omega(\rho(p_1))$.

Case 2: $|S'(q) \cup M| \leq 8$. In this case $|S'(p)| \leq \lceil 8/0.9 \rceil = 8$ for all $p \in P$. This means that $\rho(p_1) = O(1)$ and we just need to find one set X of size 3 and a point p_2 so that $(X, \{p_1, p_2\})$ is a local improvement pair. Using Lemma 9, we compute the set $S'(p)$ for all $p \in P$ in $O(n \log n)$ expected time. For each $p_2 \in P$, we need to check if there is any subset X in $S'(p_2) \cup M$ of size 3 so that $(X, \{p_1, p_2\})$ is a local improvement pair. Since $|S'(p_2) \cup M| \leq 8$, there are at most $\binom{8}{3}$ subsets $X \subseteq S'(p_2) \cup M$ for which we need to check if $(X, \{p_1, p_2\})$ is a local improvement pair. Thus there are $O(n)$ pairs of the form $(X, \{p_1, p_2\})$, where $|X| = 3$, that we need to check. For a particular pair of this form, we basically need to verify that all the disks in \mathcal{D} whose intersection with S is a subset of X are hit by either p_1 or p_2 . To make things simpler, we first remove from \mathcal{D} all the disks that are hit by p_1 and obtain a set $\mathcal{D}' \subseteq \mathcal{D}$. Now, we need to verify for all disks in \mathcal{D}' whose intersection with X is a subset of X that they are hit by p_2 . All the $O(n)$ pairs can be checked in $O(n \log n)$ time as follows.

We construct a data structure that will help us do the checking for all the $O(n)$ pairs of the form $(X, \{p_1, p_2\})$. We have already constructed a range reporting data structure on S for disk ranges. Additionally, use a dictionary data structure (based on balanced binary trees) in which the keys are subsets of S of size at most 3 and the value corresponding to a key U is a list of disks $D \in \mathcal{D}'$ s.t. $D \cap S = U$. We start with an empty dictionary. We

then go over each disk $D \in \mathcal{D}'$ one by one and if $N_D \leq 3$, we use the range reporting data structure to get $U = D \cap S$ in $O(\log n)$ time. We search the dictionary for U and if it is found, we add D to its list. If no entry is found, we create an entry for U with a single element d in its list. Note that since the number of (≤ 3)-sets that can be obtained from set of n points by intersecting it with a set of disks is linear in the number of points [21], the number of distinct keys in the dictionary is $O(n)$. We go over each key U and construct the region $R'(U)$ by taking the intersection of all the disks in the list associated with U . Note that $R'(U)$ can be constructed in $O(m \log m)$ time where m is the size of the list associated with U using Lemma 8. Since each disk is in the list of at most one U , the overall time is $O(n \log n)$. In the same amount of time, for each key U , we set up a data structure that allows us to check if a query point q is in $R'(U)$ using Lemma 8. Now, to check if a pair $(X, \{p_1, p_2\})$ is an improvement pair, we go over all subsets $U \subseteq X$ and check if $p_2 \in R'(U)$. The time spent for any pair is now $O(\log n)$. Therefore checking all the $O(n)$ pairs takes $O(n \log n)$ time.

If none of the checked pairs give local improvement, we conclude that p_1 is not useful. ◀

Next we show how to find a profitable improvement. Let $\beta = \max\{\sqrt{t}, \epsilon t \cdot \text{OPT}/n\}$.

► **Lemma 11.** *There exists a $k > 0$ such that there are at least $\Omega(\beta/k)$ useful points $p \in P$ with $\rho(p) \geq k$.*

Proof. By Lemma 5, there exists $\Omega(t)$ local improvement pairs $(X_0, Y_0), \dots$ where the X_i 's are disjoint subsets of S but the Y_i 's need not be disjoint. Each X_i is of size 3 and each Y_i is of size 2. For any pair of points $Y = \{p_1, p_2\} \subseteq P$, if (X_i, Y) is a local improvement pair among the $\Omega(t)$ pairs, then we say that X_i is a triple assigned to the pair Y . Define the weight of Y as the number of triples assigned to it and denote it by $W(Y)$. The total weight of all pairs is then $\Omega(t)$.

Call a pair Y to be of type i if $2^{i-1} \leq W(Y) < 2^i$, for $i = 1, \dots, O(\log t)$. If $W(Y) = 0$ then we say that Y is of type 0. Since the total weight of all pairs is $\Omega(t)$, there must be some $j > 0$ so that the total weight of the pairs of type j is $\Omega(t/2^j)$. Let $Q = \bigcup_Y \{Y \mid Y \text{ is of type } j\}$.

There are two lower bounds on the size of Q . First, since the total weight of the pairs of type j is $\Omega(t/2^j)$, and each pair has weight less than 2^j , the number of pairs is $\Omega(t/2^{2j})$, and hence $|Q| = \Omega(\sqrt{t}/2^j)$. On the other hand, for any local improvement pair (X_i, Y) where Y is of type j , take any point $x \in X_i$. Since $\mathcal{D}(x)$ is non-empty, any disk $D \in \mathcal{D}(x)$ contains at least one point in Y . Therefore any such local improvement pair leads to an incidence between a point in Q and a disk in \mathcal{D} . Note that since the X_i 's are disjoint these are distinct incidences. Thus there are $\Omega(t/2^j)$ incidences. Since by assumption no point in P , and therefore no point in Q , is in more than $n/(\epsilon \cdot \text{OPT})$ disks in \mathcal{D} , we have that $|Q| = \Omega(\epsilon t \cdot \text{OPT}/2^j n)$.

Therefore, $|Q| = \Omega(\max\{\epsilon t \cdot \text{OPT}/2^j n, \sqrt{t}/2^j\}) = \Omega(\beta/2^j)$. Observe that each $p \in Q$ is useful and $\rho(p) \geq 3 \cdot 2^j$. The lemma is therefore true for $k = 2^j$. ◀

Running time. Preprocessing takes $O(n^2)$ time but this is dominated by the running time of Algorithm 1. Consider a single iteration of the `while` loop in Algorithm 1. If we find some point $s \in S$ for which $\mathcal{D}(s) = \emptyset$, we drop s from the current hitting set. This way we have improved the size of the hitting set at the cost of $O(n \log n)$ time. The total time spent on such improvements is at most $O(\text{OPT } n \log n) = O(n^2 \log n)$.

Otherwise, call a single iteration of the `while` loop *lucky* if the following is true:

$$\exists i \text{ such that the point } \pi_i \text{ is useful and } \frac{i}{\rho(\pi_i)} \leq \frac{Cn}{\beta}$$

for some constant C .

► **Claim 2.** Probability that any iteration of the `while` loop is lucky is at least $1/2$.

Proof. By Lemma 11, there exists a k such that there are $\Omega(\beta/k)$ useful points, say the set U , with $\rho(p) \geq k$. Consider the smallest index i s.t. $\pi_i \in U$. The expected value of i is $O(nk/\beta)$. Therefore, with probability at least $\frac{1}{2}$, $i \leq Cnk/\beta$ for some large enough C . Then,

$$\frac{i}{\rho(\pi_i)} \leq \frac{Ckn/\beta}{k} = \frac{Cn}{\beta}$$

◀

► **Claim 3.** For a lucky iteration of the `while` loop, let λ be the reduction in size of the current hitting set, and σ the time spent in this iteration. Then $\sigma/\lambda \leq Cn^2 \log^2 n/\beta$.

Proof. As we go over the points in random order, for the current point $\nu = \pi_i$, we estimate $\rho(\nu)$ which allows us to check if $i/\rho(\nu) \leq Cn/\beta$. If so, assuming that the point ν is useful, we decrease the size of the current hitting set by $\Omega(\rho(\nu))$. If $i/\rho(\nu) > Cn/\beta$ or we discover that ν is not useful we move to the next point in the random order. However, since the iteration of the `while` loop is lucky, we will find some point $\nu = \pi_i$ which is useful and for which $i/\rho(\nu) \leq Cn/\beta$. For this point ν , we find a local improvement involving ν of value $\Omega(\rho(\nu))$ and the current iteration of the `while` loop ends. The total time spent in this iteration is $\sigma = O(i \cdot n \log^2 n)$ since we have seen i points so far and for each point we spend $O(n \log^2 n)$ time. The reduction in the size of the current hitting set is $\lambda = \Omega(\rho(\nu))$. Thus $\sigma/\lambda \leq \frac{i}{\rho(\nu)} \cdot n \log^2 n \leq Cn^2 \log^2 n/\beta$. ◀

Since any iteration of the `while` loop is lucky with probability at least 0.5, we can assume that all the iterations are lucky (running time affected by a factor of 2).

► **Claim 4.** The expected time taken to halve t is $O(n^{7/3} \log^2 n \epsilon^{-1/3})$.

Proof. Claim 3 tells us that the amortized amount of time spent for the reducing the size of the current hitting set by 1 is $O(n^2 \log^2 n/\beta)$. Since β is an increasing function of t , this decreases with t . However, t does not change by more than a factor of 2 until it is halved. So, the expected time for t to be halved is $O(t/2 \cdot n^2 \log^2 n/\beta)$. Now, $t/\beta = \min\{\sqrt{t}, n/(\epsilon \cdot \text{OPT})\}$. Since $t = O(\text{OPT})$, $t/\beta = O(\min\{\sqrt{\text{OPT}}, n/(\epsilon \cdot \text{OPT})\}) = O((n/\epsilon)^{1/3})$. Thus the expected time to halve t is $O(n^{7/3} \log^2 n \epsilon^{-1/3})$. ◀

Since the initial value of t is $O(\text{OPT})$, there are $O(\log 1/\epsilon)$ halving rounds until $t \leq \epsilon \cdot \text{OPT}$. Thus, the expected running time of the Algorithm 1 is $O(n^{7/3} \log^2 n \epsilon^{-1/3} \log(1/\epsilon))$. Finally, since we need to run Algorithm 1 for $O(1/\log(1+\epsilon))$ guesses for OPT , the overall running time is $O(n^{7/3} \log^2 n \epsilon^{-1/3} \log(1/\epsilon)/\log(1+\epsilon))$. For a fixed small value of ϵ , this is $O(n^{7/3} \log^2 n)$.

Acknowledgments. We thank the anonymous reviewers for the insightful feedback.

References

- 1 Rashmishnata Acharyya, Manjanna Basappa, and Gautam K. Das. Unit disk cover problem in 2D. In *Computational Science and Its Applications - ICCSA 2013 - 13th International Conference, Ho Chi Minh City, Vietnam, June 24-27, 2013, Proceedings, Part II*, pages 73–85, 2013.
- 2 Peyman Afshani and Timothy M. Chan. Optimal halfspace range reporting in three dimensions. In *SODA*, pages 180–186, 2009.

- 3 Pankaj K. Agarwal, Esther Ezra, and Micha Sharir. Near-linear approximation algorithms for geometric hitting sets. *Algorithmica*, 63(1-2):1–25, 2012.
- 4 Pankaj K. Agarwal and Nabil H. Mustafa. Independent set of intersection graphs of convex objects in 2D. *Comput. Geom.*, 34(2):83–95, 2006.
- 5 Pankaj K. Agarwal and Jiangwei Pan. Near-linear algorithms for geometric hitting sets and set covers. In *Symposium on Computational Geometry*, page 271, 2014.
- 6 Christoph Ambühl, Thomas Erlebach, Matús Mihalák, and Marc Nunkesser. Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In *APPROX-RANDOM*, pages 3–14, 2006.
- 7 Boris Aronov and Sarel Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38(3):899–921, 2008.
- 8 H. Bronnimann and M. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete & Computational Geometry*, 14(4):463–479, 1995.
- 9 Gruia Călinescu, Ion I. Mandoiu, Peng-Jun Wan, and Alexander Zelikovsky. Selecting forwarding neighbors in wireless ad hoc networks. *MONET*, 9(2):101–111, 2004.
- 10 P. Carmi, M. Katz, and N. Lev-Tov. Covering points by unit disks of fixed location. In *ISAAC*, pages 644–655, 2007.
- 11 Timothy M. Chan and Sarel Har-Peled. Approximation algorithms for maximum independent set of pseudo-disks. In *Symposium on Computational Geometry*, pages 333–340, 2009.
- 12 Francisco Claude, Gautam K. Das, Reza Dorrigiv, Stephane Durocher, Robert Fraser, Alejandro López-Ortiz, Bradford G. Nickerson, and Alejandro Salinger. An improved line-separable algorithm for discrete unit disk cover. *Discrete Math., Alg. and Appl.*, 2(1):77–88, 2010.
- 13 Gautam K. Das, Robert Fraser, Alejandro Lopez-Ortiz, and Bradford G. Nickerson. On the discrete unit disk cover problem. *International Journal on Computational Geometry and Applications*, 22(5):407–419, 2012.
- 14 Michael B. Dillencourt and Warren D. Smith. Graph-theoretical conditions for inscribability and delaunay realizability. *Discrete Mathematics*, 161(1–3):63–77, 1996.
- 15 G. Even, D. Rawitz, and S. Shahar. Hitting sets when the VC-dimension is small. *Inf. Process. Lett.*, 95:358–362, 2005.
- 16 Robert Fraser. *Algorithms for Geometric Covering and Piercing Problems*. PhD thesis, University of Waterloo, 2012.
- 17 Matt Gibson and Imran A. Pirwani. Algorithms for dominating set in disk graphs: Breaking the $\log n$ barrier. In *ESA*, pages 243–254, 2010.
- 18 Dorit S. Hochbaum and Wolfgang Maass. Fast approximation algorithms for a nonconvex covering problem. *J. Algorithms*, 8(3):305–323, 1987.
- 19 Erik Krohn, Matt Gibson, Gaurav Kanade, and Kasturi R. Varadarajan. Guarding terrains via local search. *JoCG*, 5(1):168–178, 2014.
- 20 Daniel Marx. Parameterized complexity of independence and domination on geometric graphs. In *Parameterized and Exact Computation, Second International Workshop, IWPEC 2006, Zurich, Switzerland, September 13-15, 2006, Proceedings*, pages 154–165, 2006.
- 21 Jiri Matousek. *Lectures in Discrete Geometry*. Springer-Verlag, New York, NY, 2002.
- 22 Nabil H. Mustafa and Saurabh Ray. Improved results on geometric hitting set problems. *Discrete & Computational Geometry*, 44(4):883–895, 2010.
- 23 Evangalia Pyrga and Saurabh Ray. New existence proofs for epsilon-nets. In *Proceedings of Symposium on Computational Geometry*, pages 199–207, 2008.
- 24 Ran Raz and Muli Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *STOC*, pages 475–484, 1997.

Arc Diagrams, Flip Distances, and Hamiltonian Triangulations

Jean Cardinal^{*1}, Michael Hoffmann^{†2}, Vincent Kusters^{†2},
Csaba D. Tóth³, and Manuel Wettstein^{‡2}

1 Université libre de Bruxelles (ULB), Belgium

jcardin@ulb.ac.be

2 Department of Computer Science, ETH Zürich, Switzerland

{hoffmann,vincent.kusters,manuelwe}@inf.ethz.ch

3 California State University Northridge, Los Angeles, CA, USA

cdtoth@acm.org

Abstract

We show that every triangulation (maximal planar graph) on $n \geq 6$ vertices can be flipped into a Hamiltonian triangulation using a sequence of less than $n/2$ combinatorial edge flips. The previously best upper bound uses 4-connectivity as a means to establish Hamiltonicity. But in general about $3n/5$ flips are necessary to reach a 4-connected triangulation. Our result improves the upper bound on the diameter of the flip graph of combinatorial triangulations on n vertices from $5.2n - 33.6$ to $5n - 23$. We also show that for every triangulation on n vertices there is a simultaneous flip of less than $2n/3$ edges to a 4-connected triangulation. The bound on the number of edges is tight, up to an additive constant. As another application we show that every planar graph on n vertices admits an arc diagram with less than $n/2$ biarcs, that is, after subdividing less than $n/2$ (of potentially $3n - 6$) edges the resulting graph admits a 2-page book embedding.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases graph embeddings, edge flips, flip graph, separating triangles

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.197

1 Introduction

An *arc diagram* (Figure 1) is a drawing of a graph in which vertices are represented by points on a horizontal line, called the *spine*, and edges are drawn either as one halfcircle (*proper arc*) or as a sequence of halfcircles centered on the line (forming a smooth Jordan arc). In a *proper arc diagram* all arcs are proper. Arc diagrams have been used and studied in many contexts since their first appearance in the mid-sixties [3, 24]. They constitute a well-studied geometric representation in graph drawing [14] that occurs, for instance, in the study of crossing numbers [1, 6] and universal point sets for circular arc drawings [4].

Bernhart and Kainen [5] proved that a planar graph admits a *plane* (i.e., crossing-free) proper arc diagram if and only if it can be augmented to a Hamiltonian planar graph by

* Partially supported by the ESF EUROCORES programme EuroGIGA, CRP ComPoSe.

† Partially supported by the ESF EUROCORES programme EuroGIGA, CRP GraDR and the Swiss National Science Foundation, SNF Project 20GG21-134306.

‡ Partially supported by the ESF EUROCORES programme EuroGIGA, CRP ComPoSe and the Swiss National Science Foundation, SNF Project 20GG21-134318/1.



© Jean Cardinal, Michael Hoffmann, Vincent Kusters,
Csaba D. Tóth, and Manuel Wettstein;
licensed under Creative Commons License CC-BY

32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).

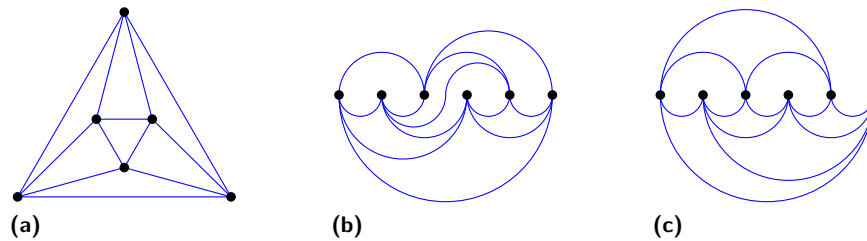
Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 197–210

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany





■ **Figure 1** A plane straight-line drawing (a), an arc diagram (b) and a proper arc-diagram (c) of the same graph.

adding new edges. Such planar graphs are also called *subhamiltonian*, and they are NP-hard to recognize [26]. A Hamiltonian cycle in the augmented graph directly yields a feasible order for the vertices on the spine. Every planar graph can be subdivided into a subhamiltonian graph with at most one subdivision vertex per edge [22]. Consequently, every planar graph admits a plane *biarc diagram* in which each edge is either a proper arc or the union of two halfcircles (a *biarc*); one above and one below the spine. Di Giacomo et al. [15] showed that every planar graph even admits a *monotone* biarc diagram in which every biarc is *x-monotone*—such an embedding is also called a *2-page topological book embedding*. See [14] for various other applications of subhamiltonian subdivisions of planar graphs.

Eppstein [13] said: “Arc diagrams (with one arc per edge) are very usable and practical but can only handle a subset of planar graphs.” Using biarcs allows to represent all planar graphs, but adds to the complexity of the drawing. Hence it is a natural question to ask: How close can we get to a proper arc diagram, while still being able to represent all planar graphs? A natural measure of complexity is the number of biarcs used.

Previous methods for subdividing an n -vertex planar graph into a subhamiltonian graph use at most one subdivision per edge [14, 15, 19, 22], consequently the number of biarcs in an arc diagram is bounded by the number of edges. Our main goal in this paper is to tighten the upper and lower bound on the minimum number of biarcs in an arc diagram (or, alternatively, the number of subdivision vertices in a subhamiltonian subdivision) of a planar graph with n vertices. Minimizing the number of biarcs is clearly NP-hard, since the number of biarcs is zero if and only if the graph is subhamiltonian.

Our results. We show that the number of biarcs can be bounded by n , even when they are restricted to be monotone. Although previous methods can be shown to yield less than the trivial $3n - 6$ biarcs [19], or ensure monotonicity [15], we give the first proof that both properties can be guaranteed simultaneously. The algorithm is similar to the canonical ordering-based method of Di Giacomo et al. [15].

► **Theorem 1.** *Every planar graph on $n \geq 4$ vertices admits an arc diagram using at most $n - 4$ biarcs, all of which are monotone.*

For arbitrary (not necessarily monotone) biarcs we achieve better bounds. Our main tool is relating subhamiltonian planar graphs to edge flips in triangulations. A *flip* in a triangulation involves switching the diagonal of a quadrilateral made of two adjacent facial triangles. We consider *combinatorial* flips, which can be regarded as an operation on an abstract graph. The *flip graph* induced by flips on the set of all triangulations on n vertices, and the corresponding *flip distance* between two triangulations, have been the topic of extensive research [9, 11]. For instance, the flip diameter restricted to the interior of a convex polygon is equivalent to rotation distance of binary trees [25, 23].

We prove that in every triangulation there exists a set of less than $2n/3$ edges that can be flipped *simultaneously* so that the resulting triangulation is 4-connected, and that this bound is tight up to an additive constant (Section 4). Since by Tutte's Theorem every 4-connected planar graph is Hamiltonian, we can transform every planar graph into a subhamiltonian graph by subdividing at most $2n/3$ edges. The fact that a single simultaneous flip can make a triangulation 4-connected has already been established by Bose et al. [8]. However, they do not give any bound on the number of flipped edges.

► **Theorem 2.** *Every maximal planar graph on $n \geq 6$ vertices can be transformed into a 4-connected maximal planar graph using a simultaneous flip of at most $\lfloor (2n - 7)/3 \rfloor$ edges.*

► **Theorem 3.** *For every $i \in \mathbb{N}$, there is a maximal planar graph G_i on $n_i = 3i + 4$ vertices such that no simultaneous flip of less than $(2n_i - 8)/3 = 2i$ edges results in a 4-connected graph.*

Finally, we prove an upper bound on the flip distance of a planar triangulation to Hamiltonicity, that is, on the worst-case number of successive flips required to reach a Hamiltonian triangulation (Section 5). Given the hardness of determining whether a given planar graph is Hamiltonian, we should not expect a nice characterization of (non-)Hamiltonicity. Hence, in the context of planar graphs, 4-connectivity is often used as a substitute because by Tutte's Theorem it is a sufficient condition for Hamiltonicity. Bose et al. [10] gave a tight bound (up to an additive constant) of $3n/5$ flips to transform a given triangulation on n vertices into a 4-connected triangulation. We show that fewer flips are sufficient to guarantee Hamiltonicity. Obviously, the target triangulation is not 4-connected in general, which means it possibly contains separating triangles.

► **Theorem 4.** *Every maximal planar graph on $n \geq 6$ vertices can be transformed into a Hamiltonian maximal planar graph using a sequence of at most $\lfloor (n - 3)/2 \rfloor$ edge flips.*

In this case we do not have a matching lower bound. The best lower bound we know can be obtained using *Kleetopes* [16]. These are convex polytopes that are generated from another convex polytope by replacing every face by a small pyramid. In the language of planar graphs, we start from a 3-connected planar graph and for every face add a new vertex that is connected to all vertices on the boundary of the face. If the graph we start from has enough faces, then the added vertices form a large independent set so that the resulting graph is not Hamiltonian. Aichholzer et al. [2] describe such a construction explicitly in the context of flipping a triangulation to a Hamiltonian triangulation, but state the asymptotics only. A precise counting reveals the following figures.

► **Theorem 5.** *For every $i \in \mathbb{N}$, there is a maximal planar graph G_i on $n_i = 3i + 8$ vertices such that no sequence of less than $(n_i - 8)/3 = i$ edge flips produces a Hamiltonian graph and no set of less than $(n_i - 8)/3 = i$ subdivision vertices produces a subhamiltonian graph.*

Our proof for Theorem 4 is constructive, and each flip in the sequence involves an edge of the initial graph G and is incident to a separating triangle of G . Some of these edges may be incident to a common facial triangle, so they cannot always be flipped simultaneously. However, we show that if we *subdivide* each of these edges (instead of successively flipping them), we obtain a subhamiltonian graph. Combined with the characterization of Bernhart and Kainen [5], this yields a new bound on the number of biarcs.

► **Corollary 6.** *Every planar graph on $n \geq 6$ vertices admits a biarc diagram with at most $\lfloor (n - 3)/2 \rfloor$ biarcs.*

As another corollary of Theorem 4, we establish a new upper bound on the diameter of the flip graph of all triangulations on n vertices, improving on the previous best bound of $5.2n - 33.6$ by Bose et al. [10]. Mori et al. [21] showed that any two Hamiltonian triangulations on n vertices can be transformed into each other by at most $\max\{4n - 20, 0\}$ flips. Combined with Theorem 4, this implies the following.

► **Corollary 7.** *Every two triangulations on $n \geq 6$ vertices can be transformed into each other using at most $5n - 23$ edge flips.*

Due to space constraints, many proofs have to be omitted.

2 Notation

A *drawing* of a graph G in \mathbb{R}^2 maps the vertices into distinct points in the plane and maps each edge to a Jordan arc between (the images of) the two vertices that is disjoint from (the image of) any other vertices. To avoid notational clutter it is common to identify vertices and edges with their geometric representation. A drawing is called *plane* (or an *embedding*) if no two edges intersect except at a possible common endpoint. Only planar graphs admit plane drawings, but not every drawing of a planar graph is plane. A *maximal planar* graph on n vertices is a planar graph with $3n - 6$ edges. In this paper the term *triangulation* is used as a synonym for maximal planar graph.¹

In a plane drawing of a triangulation G , every face (including the outer face) is bounded by three edges. Hence, every triangulation with $n \geq 4$ vertices is 3-connected [12][Lemma 4.4.5]. Every 3-connected planar graph has a topologically unique plane drawing, apart from the choice of the outer face. Specifically, the facial triangles are precisely the nonseparating chordless cycles of G in every plane drawing [12][Proposition 4.2.7]. Consequently, G has a well-defined dual graph G^* (independent of the drawing): the vertices of G^* correspond to the faces of G , and two vertices of G^* are adjacent if and only if the corresponding faces share an edge. A triangle of G that is not facial is called a *separating* triangle and its removal disconnects the graph.

A graph is *Hamiltonian* if it contains a cycle through all vertices. By a famous theorem of Tutte, all 4-connected planar graphs are Hamiltonian. For triangulations, 4-connectivity is equivalent to the absence of separating triangles. A vertex or an edge is *incident* to a triangle T in a graph if it is a vertex or edge of T .

A triangulation G can be partitioned into a *4-block tree* \mathcal{B} . Each vertex of \mathcal{B} is either a maximal 4-connected component of G or a subgraph of G that is isomorphic to K_4 . Two vertices of \mathcal{B} are adjacent if they share a separating triangle of G . The 4-block tree is similar to the standard block-tree for 2-connected components, but the generalization of the notion “component” to higher connectivity is not straightforward in general. For a triangulation, however, the 4-block tree is well-defined and can be computed in linear time and space [18].

Flips. Consider an edge ab of a triangulation G and let abc and adb denote the two incident facial triangles. The *flip* of ab replaces the edge ab by the edge cd . If this operation produces a triangulation (i.e., if the edge cd is not already present in G), we call ab *flippable*².

¹ In contrast, a maximal plane *straight-line* drawing may have fewer edges, depending on the number of points on the convex hull.

² We consider *combinatorial flips*, as opposed to *geometric flips* defined for straight-line plane drawings, where an edge is flippable if and only if the quadrilateral formed by the two incident facial triangles is convex.

A closely related concept is the *simultaneous flip* of a set F of flippable edges in a triangulation $G = (V, E)$, which is defined as follows. For $e \in F$ denote by $c(e)$ the edge created by flipping e in G , and let $C(F) = \bigcup_{e \in F} c(e)$. Then the simultaneous flip of F in G results in the graph $G' = (V, (E \setminus F) \cup C(F))$. Bose et al. [8] introduced this notion and showed that the result of a simultaneous flip is a triangulation if every facial triangle of G is incident to at most one edge from F and the edges $c(e)$, for $e \in F$, are all distinct and not present in E .

3 Biarc Diagrams

► **Lemma 8.** *If a planar graph G has a simultaneous flip of k edges such that the resulting graph is Hamiltonian, then G admits an arc diagram with at most k biarcs.*

Proof. Let H be a Hamiltonian graph obtained from G by simultaneously flipping an edge set E_1 to E_2 with $|E_1| = k$. Without loss of generality, assume that E_1 is a minimal set of edges that must be flipped in order to obtain a Hamiltonian graph. Consequently, every Hamiltonian cycle C in H passes through all k edges in E_2 . If we subdivide each edge in E_2 , we obtain a Hamiltonian graph H' . Now consider the graph G' obtained from G by subdividing each edge in E_1 , and identify the subdivision vertices of the corresponding edges in G' and H' . Notice that the union of G' and H' is a plane graph that contains H' , hence it is Hamiltonian. Consequently G' is subhamiltonian. By the characterization of Bernhart and Kainen [5], G admits an arc diagram with k biarcs, as claimed. ◀

In order to obtain a general statement about arc diagrams from Lemma 8, we need a bound on the number of edges to simultaneously flip in a given graph in order to make it Hamiltonian. Even the existence of such a simultaneous flip—regardless of the number of edges involved—is not obvious to begin with. For instance, consider a vertex that has linear degree in a triangulation T_1 and constant degree in a triangulation T_2 . As a single simultaneous flip can only change about half of the edges incident to a vertex, at least a logarithmic number of simultaneous flips is required to transform T_1 into T_2 [8].

Bose et al. [8] showed that every triangulation on $n \geq 6$ vertices can be transformed to a 4-connected (hence Hamiltonian) triangulation by a single simultaneous flip. However, no bound is known on the number of flipped edges, which leaves us with the trivial bound of $(2n - 4)/2 = n - 2$. Note that the resulting bound on the number of biarcs is similar to the one from Theorem 1, but there we could guarantee that all biarcs are monotone. Using Lemma 8 we do not have any control over the type of biarcs used.

The obvious open question is: Can we give a better bound on the number of edges needed in a simultaneous flip to a Hamiltonian triangulation?

4 Simultaneous Flip Distance to 4-connectivity

In this section we determine the maximum number of edges needed to transform an n -vertex triangulation into a Hamiltonian triangulation using a single simultaneous flip. Consider a triangulation $G = (V, E)$. As there is no 4-connected triangulation on fewer than six vertices, suppose that G has at least six vertices. We would like to transform G into a 4-connected triangulation by simultaneously flipping a set $F \subset E$ of edges such that all separating triangles are destroyed and none created. We use the following criterion by Bose et al. [8] to check whether a simultaneous flip produces a 4-connected triangulation.

► **Lemma 9** (Bose et al. [8]). *Let F be a set of edges in a triangulation G such that no two edges in F are incident to a common triangle, every edge in F is incident to a separating triangle, and for every separating triangle T there is at least one edge in F that is incident to T . Then F is simultaneously flippable in G and the resulting triangulation is 4-connected.*

For a simultaneously flippable set F of edges no face of the triangulation is incident to more than one edge. Recall that the edges of a triangulation G and its dual G^* are in one-to-one correspondence. Consequently, the dual edges of F form a matching in G^* . As all faces of a triangulation are triangles, G^* is cubic (3-regular). Moreover, every triangulation on $n \geq 4$ vertices is 3-connected and so its dual is 2-edge-connected (bridgeless). By a famous theorem of Tait the following statement is equivalent to the Four-Color Theorem:

► **Theorem 10** (Tait [7] Chapter 11). *Every bridgeless cubic planar graph admits a partition of the edge set into three perfect matchings.*

In particular, this applies to the dual of a triangulation. Call a set $F \subseteq E$ of edges of a triangulation $G = (V, E)$ a (perfect) *dual matching* if the corresponding set of edges in the dual graph G^* forms a (perfect) matching of G^* . While it is clear that a perfect dual matching contains exactly one edge of each facial triangle, this is not obvious for separating triangles. But it follows from a simple parity argument, as the following lemma shows.³

► **Lemma 11.** *Every perfect dual matching of a triangulation G contains an edge of every triangle of G .*

The last missing bit to prove Theorem 2 is an upper bound on the number of edges in a triangulation that can be incident to separating triangles.

► **Lemma 12.** *At most $2n - 7$ edges of a maximal planar graph on $n \geq 4$ vertices are incident to separating triangles. This bound is the best possible.*

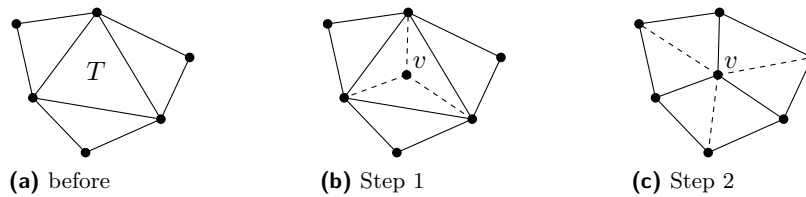
► **Theorem 2.** *Every maximal planar graph on $n \geq 6$ vertices can be transformed into a 4-connected maximal planar graph using a simultaneous flip of at most $\lfloor (2n - 7)/3 \rfloor$ edges.*

Proof. Consider a maximal planar graph G on n vertices. By Theorem 10 the $3n - 6$ edges of G can be partitioned into three perfect dual matchings $M_1, M_2,$ and M_3 , of $n - 2$ edges each. Let M'_i , for $i \in \{1, 2, 3\}$, denote the dual matching that results from removing all edges from M_i that are not incident to any separating triangle. By Lemma 12 at most $2n - 7$ edges of G are incident to separating triangles. Therefore, one of $M'_1, M'_2,$ and M'_3 contains at most $\lfloor (2n - 7)/3 \rfloor$ edges. By Lemma 9 these edges are simultaneously flippable and the resulting graph is 4-connected. ◀

5 Flip Distance to Hamiltonicity

With regard to arc diagrams, there is actually no reason to insist that the triangulation be 4-connected. In order to apply Lemma 8 we only need the triangulation to be Hamiltonian. Hence the obvious question: Can we always find a simultaneous flip of fewer than $2n/3$ edges to obtain a Hamiltonian triangulation? In this section we go one step further and in addition lift the restriction that the flip be simultaneous. Instead, an arbitrary sequence

³ Bose et al. [8] derive this property from the explicit Tait coloring. The statement here is slightly more general because it holds for every perfect dual matching.



■ **Figure 2** Example of a dummy flip.

of edge flips is allowed. In this case tight bounds are known if the goal is to obtain a 4-connected triangulation. Bose et al. [10] showed that $\lfloor (3n - 9)/5 \rfloor$ flips are always sufficient and sometimes $(3n - 10)/5$ flips are necessary to transform a given triangulation on n vertices into a 4-connected triangulation.

In general, a non-simultaneous flip sequence has no direct implication for arc diagrams. But if only edges of the original triangulation are flipped, then we can subdivide those edges rather than flipping them. In the resulting arc diagram only the subdivided edges may appear as biarcs. But a bound on the flip distance to a Hamiltonian triangulation is of independent interest. For instance, it is directly related to the current best upper bound on the diameter of the flip graph of combinatorial triangulations [10, 20, 21]. The argument uses a single so-called canonical triangulation and shows that every triangulation can be transformed into this canonical triangulation in two steps: First at most $\lfloor (3n - 9)/5 \rfloor$ flips are needed to obtain a 4-connected triangulation and then an additional at most $2n - 15$ flips are needed to transform any 4-connected triangulation into the canonical one. Combining two such flip sequences yields an upper bound of $5.2n - 33.6$ on the diameter of the flip graph [10]. The bound of $2n - 15$ flips for the second step is actually tight [20]. The corresponding bound for a triangulation that is Hamiltonian (but not necessarily 4-connected) is slightly worse only: It can be transformed into the canonical triangulation using at most $2n - 10$ flips [21]. Hence our focus is to improve the first step by showing that fewer flips are needed to guarantee a Hamiltonian triangulation than a 4-connected one.

► **Theorem 4.** *Every maximal planar graph on $n \geq 6$ vertices can be transformed into a Hamiltonian maximal planar graph using a sequence of at most $\lfloor (n - 3)/2 \rfloor$ edge flips.*

Proof outline. The proof is constructive and consists of two steps. In a first step we apply a sequence of elementary operations that transform a triangulation G into a 4-connected triangulation G' . An elementary operation is either a usual edge flip or a *dummy flip*, where a facial triangle T is subdivided into three triangles by inserting a new (dummy) vertex and then all three edges of T are flipped. All this will be done in such a way that G' becomes 4-connected and, therefore, contains a Hamiltonian cycle H' . We then remove all dummy vertices and construct a Hamiltonian cycle H'' resembling H' in the resulting triangulation G'' . Finally, we argue that G'' can be obtained from G with at most $n/2$ (usual) edge flips. Specifically, we show that each dummy flip can be implemented using at most two edge flips.

Dummy flips. Given a triangulation G on $n \geq 4$ vertices and a facial triangle T of G , a dummy flip of T transforms G as follows (Figure 2): First, insert a new (dummy) vertex v in the interior of face T and connect it to all three vertices of T . Note that T becomes a separating triangle in the resulting graph. Second, flip all three edges of T in an arbitrary order. Similarly to the usual flip operation, a dummy flip may create multiple edges. But we

will use this operation in specific situations only—as specified in the lemma below—where we can show that it produces a triangulation (that is, no multiple edges).

► **Lemma 13.** *Let G be a maximal planar graph and let T be a facial triangle of G such that every edge of T is incident to a separating triangle of G . Then the dummy flip operation of T in G produces no double edges and no new separating triangles.*

Step 1: 4-connectivity. Our main lemma to establish Theorem 4 is the following.

► **Lemma 14.** *Every maximal planar graph on $n \geq 6$ vertices can be transformed into a 4-connected maximal planar graph by a sequence of f flip and d dummy flip operations, for some $f, d \in \mathbb{N}$, such that $f + 2d \leq (n - 3)/2$.*

Recall that there are triangulations on n vertices that contain $\lfloor (3n - 9)/5 \rfloor$ pairwise edge-disjoint separating triangles [10, 17]. In this case, we need to flip away at least one edge from each separating triangle to reach 4-connectivity. Considering that a dummy flip operation flips three edges, the parameters in Lemma 14 satisfy $f + 3d \geq \lfloor (3n - 9)/5 \rfloor$. The crucial claim in Lemma 14 is that $f + 2d \leq (n - 3)/2$ is possible, and later we will show how to replace each dummy flip by two usual flips rather than three (Lemma 22).

The rest of this section is devoted to the proof of Lemma 14. We describe an algorithm that, given a triangulation G on $n \geq 6$ vertices, returns a sequence of f flip and d dummy flip operations that produces a 4-connected graph. The bound $6f + 12d \leq 3n - 9$ is established via the following charging scheme. Each edge of G , with the exception of the three edges of the outer face, receives one unit of credit. Each edge flip costs six units and each dummy flip costs fifteen units.

4-Block Decomposition. In our algorithm, we recursively process 4-connected subgraphs using the 4-block tree \mathcal{B} of G . By fixing an (arbitrary) plane embedding of G , we make \mathcal{B} a rooted tree such that the root is the 4-block that contains the boundary of the outer face of G . Every separating triangle T of G corresponds to an edge between two 4-blocks, where the parent lies in the exterior of T (plus T) and the child lies in the interior of T (plus T). For a 4-block G_i in \mathcal{B} denote by T_i the outer face of G_i , and denote by n_i the number of vertices of G_i minus three (the vertices of T_i). An edge of G_i is called an *interior* edge if it is not incident to the outer face T_i . For each 4-block G_i in \mathcal{B} we maintain counters f_i and d_i that denote the number of flips and dummy flips, respectively, that were used within G_i during the course of the algorithm. Initially $f_i = d_i = 0$, for every vertex G_i of \mathcal{B} .

The algorithm computes the sequence of flip and dummy flip operations incrementally, and maintains a current triangulation produced by the operations. Both the graph G and the 4-block decomposition \mathcal{B} change dynamically: when we flip an edge e of some separating triangle(s), all 4-blocks containing edge e merge into a single 4-block. At the end of the algorithm, the tree \mathcal{B} consists of a single 4-block that corresponds to the 4-connected graph G' . In order to avoid notational clutter, we always denote the current 4-block tree by \mathcal{B} . As an invariant (detailed below) we maintain that at each node of \mathcal{B} the number of interior edges (ignoring dummy edges) balances the cost of operations that were spent in this 4-block. As \mathcal{B} evolves, so does the graph $\mathcal{G}(\mathcal{B})$ represented by \mathcal{B} . This graph is the union of all nodes (4-blocks) in \mathcal{B} , where for any edge of \mathcal{B} the vertices and edges of the common triangle in the two endpoints (4-blocks) are identified.

Main loop. At every step, we take an arbitrary 4-block G_i on the penultimate level of \mathcal{B} , that is, G_i is not a leaf but all of its children are leaves. Let C_i denote the set of indices c

such that G_c is a child of G_i in \mathcal{B} , and denote $\mathcal{T}_i = \{T_c \mid c \in C_i\}$. The algorithm selects a sequence of edges of G_i to be flipped (or dummy flipped) in order to merge G_i with G_c , for all $c \in C_i$, into a new 4-block G_z . Denote the resulting 4-block tree by \mathcal{B}' . If no edge of T_i is flipped, then G_z is a leaf of \mathcal{B}' . But if an edge of T_i is flipped, then G_z may be an interior node of \mathcal{B}' .

If an edge e of T_i is flipped and G_i is not the root of \mathcal{B} , then more blocks may merge into G_z : The edge e is definitely shared with the parent of G_i in \mathcal{B} , but it may be shared with further ancestors as well. In addition, the edge e may belong to (at most) one sibling G_s of G_i and some descendants of G_s as well. We denote by J the set of all j such that G_j is a leaf of \mathcal{B} that is merged into G_z . Similarly, denote by Q the set of all q such that G_q is an interior vertex of \mathcal{B} that is merged into G_z , and denote by Q^+ the set of indices $q \in Q$ such that $f_q + d_q > 0$. Note that neither J nor Q are empty, because $C_i \subseteq J$ and $i \in Q$. However, we may have $Q^+ = \emptyset$.

Algorithmic preliminaries. In each iteration, we flip the edges of a dual matching of G_i (a 4-connector, defined below), but if \mathcal{T}_i forms a checkerboard (defined below), we substitute three of these flip operations by one dummy flip.

A 4-connector for G_i is a dual matching of G_i that contains precisely one edge from every triangle in \mathcal{T}_i . A 4-connector always exists because every perfect dual matching (Theorem 10) is a 4-connector (Lemma 11). A *minimum 4-connector* is a 4-connector of minimum cardinality. By Lemma 9 we can flip the edges of a 4-connector in an arbitrary order, and the 4-blocks G_c , for all $c \in C_i$, will merge into G_i . We say that \mathcal{T}_i is a *checkerboard* if the triangles in \mathcal{T}_i are pairwise edge-disjoint and every interior edge of G_i belongs to some triangle in \mathcal{T}_i . If \mathcal{T}_i is a checkerboard, then we perform a dummy flip on a triangle F that is selected according to the following lemma.

► **Lemma 15.** *If \mathcal{T}_i is a checkerboard, then G_i has a facial triangle F that is adjacent to three triangles in \mathcal{T}_i that are not all adjacent to T_i .*

Algorithm 4CONNECT(G). Given a triangulation G , fix an arbitrary embedding of G . This embedding defines a rooted 4-block tree \mathcal{B} . While \mathcal{B} is not a singleton, do: (1) Consider an arbitrary vertex G_i at the penultimate level of \mathcal{B} . (2) Find a minimum 4-connector M for G_i that contains a maximum number of edges of T_i (that is, one, if possible). (3) If \mathcal{T}_i is *not* a checkerboard, then flip the edges of M in an arbitrary order. (4) Otherwise, let F be an arbitrary facial triangle as in Lemma 15. First apply a dummy flip to F . For each of the three triangles from \mathcal{T}_i adjacent to F , remove the incident edge from M , and flip all remaining edges of M in an arbitrary order. (5) Finally, update \mathcal{B} and $\mathcal{G}(\mathcal{B})$.

Correctness of the Algorithm. We show that the above algorithm turns an input triangulation G on n vertices into a 4-connected triangulation using a sequence of f flip and d dummy flip operations, for some $f, d \in \mathbb{N}$, such that $f + 2d \leq (n - 3)/2$. By Lemmata 9, 13, and 15, the operations described in the algorithm can be performed. In every step of the algorithm at least two nodes of the 4-block tree are merged. Therefore, after a finite number of steps we are left with a block tree that consists of a single 4-block G' .

► **Observation 16.** *For each vertex v created by a dummy flip operation in algorithm 4CONNECT(G), subsequent operations do not modify the six facial triangles incident to v .*

Free edges. It remains to bound the number of flip and dummy flip operations performed by the algorithm. An edge within some 4-block G_i of \mathcal{B} is *free* if it is not incident to any separating triangle of $\mathcal{G}(\mathcal{B})$. Free edges are a good measure of progress for our algorithm because our final goal is to arrive at a state where all edges of $\mathcal{G}(\mathcal{B})$ are free.

Invariants. As an invariant we maintain that every vertex G_i of \mathcal{B} satisfies the following condition:

- (F1) If G_i is the only vertex of \mathcal{B} , then it has at least $6f_i + 15d_i + 3$ free edges.
- (F2) If G_i is a leaf of \mathcal{B} that is not the root of \mathcal{B} , then G_i has at least $6f_i + 15d_i + 3$ free interior edges.
- (F3) If G_i is an interior vertex of \mathcal{B} , then either $f_i = d_i = 0$ or G_i has at least $6f_i + 15d_i + 1$ free interior edges.

Initially, (F2) holds for every leaf G_i of \mathcal{B} because all of the interior $3(n_i + 3) - 6 - 3 = 3n_i$ edges are free, $n_i \geq 1$, and $f_i = d_i = 0$. Trivially, (F3) holds for every interior vertex G_i of \mathcal{B} because $f_i = d_i = 0$. Having a certain number of edges in a plane graph implies having a certain number of vertices, as quantified by the following lemma.

Invariant maintenance. It remains to show that each step of the algorithm maintains invariants (F1)–(F3). We start bounding the size of a minimum 4-connector M of G_i .

► **Lemma 17.** *Let M be a minimum 4-connector for G_i that contains the maximum number of edges of T_i .*

- (1) *If s edges of G_i are each incident to two triangles from \mathcal{T}_i , then $|M| \leq |C_i| - \lceil s/3 \rceil$.*
- (2) *If the triangles in \mathcal{T}_i are pairwise edge-disjoint and $f_i + d_i > 0$, then $|M| \leq n_i - 2f_i - 5d_i$.*

In both cases, equality is possible only if M contains an edge of T_i .

Proof. (1) Partition the edge set of G_i into three dual matchings by Theorem 10. One of them, say D , contains at least $\lceil s/3 \rceil$ of the s edges that are incident to two triangles from \mathcal{T}_i . If every triangle in \mathcal{T}_i selects a unique incident edge from D , we obtain a 4-connector $R \subseteq D$ of size at most $|C_i| - \lceil s/3 \rceil$. The minimality of M yields $|M| \leq |R|$.

(2) By (F3), G_i has at least $6f_i + 15d_i + 1$ free interior edges. Therefore, at least one of the three perfect dual matchings guaranteed by Theorem 10 contains at least $\lceil (6f_i + 15d_i + 1)/3 \rceil = 2f_i + 5d_i + 1$ free interior edges. After removal of all those edges, the resulting dual matching R is still a 4-connector. By the minimality of M we have $|M| \leq |R| \leq (n_i + 1) - (2f_i + 5d_i + 1) = n_i - 2f_i - 5d_i$.

If M contains no edge of T_i , consider again the 4-connector D from above. It is obtained by removing free interior edges from a perfect dual matching of G_i . Hence D contains an edge of T_i . But then by the choice of M we know that D is not a minimum 4-connector and so we have $|M| \leq |C_i| - \lceil s/3 \rceil - 1$ and $|M| \leq n_i - 2f_i - 5d_i - 1$, respectively. ◀

► **Lemma 18.** *Let M be a minimum 4-connector for G_i and suppose that \mathcal{T}_i is not a checkerboard. Then after flipping the edges in M , the resulting 4-block G_z contains at least $6f_z + 15d_z + 3\lceil s/3 \rceil + |Q^+|$ free interior edges, where s denotes the number of edges of G_i that are incident to two triangles from \mathcal{T}_i .*

► **Lemma 19.** *Suppose that G_i together with all its children in \mathcal{B} is merged into a leaf G_z of \mathcal{B}' using f flips and d dummy flips. Then G_z contains at least $6(f_z - f_i - f) + 15(d_z - d_i - d) + 3n_i + 3|C_i| + 3|Q| - 3$ free interior edges.*

► **Lemma 20.** *Suppose that $f_i = d_i = 0$ and G_i along with all its children in \mathcal{B} is merged into an **interior node** G_z of \mathcal{B}' using f flips and d dummy flips. Then G_z contains at least $6(f_z - f) + 15(d_z - d) + 3n_i + 3|C_i| + 1$ free interior edges.*

► **Lemma 21.** *Suppose that \mathcal{T}_i is a **checkerboard**. Then G_z fulfills invariants (F1)–(F3).*

Case analysis. We now use Lemmata 18–21 to show that every step of algorithm 4CONNECT maintains (F1)–(F3). By Lemma 18 we may suppose in the following that the triangles in \mathcal{T}_i are pairwise edge-disjoint. Because if they are not, then the 4-block G_z obtained by flipping the edges in M by Lemma 18 fulfills one of (F1), (F2), or (F3), depending on the status of G_z in \mathcal{B}' . If \mathcal{T}_i is a checkerboard, then we are done by Lemma 21. Hence suppose that \mathcal{T}_i is not a checkerboard. Together with the fact that the triangles in \mathcal{T}_i are pairwise edge-disjoint, it follows that G_i has at least one free interior edge. As this edge appears in one of the three perfect dual matchings of Theorem 10, we conclude that $|M| \leq n_i$. For the remainder of the analysis we distinguish four cases.

Case 0: G_z is the only node of \mathcal{B}' . Then by Lemma 18 there are at least $6f_z + 15d_z + 3\lceil s/3 \rceil + |Q^+| \geq 6f_z + 15d_z$ free interior edges in G_z . Together with the three free non-interior edges of T_z this proves (F1).

Case 1: G_z is an interior vertex of \mathcal{B}' . If $Q^+ \neq \emptyset$, then (F3) holds by Lemma 18. Hence suppose that $Q^+ = \emptyset$ and so in particular $f_i = d_i = 0$. Using Lemma 20 with $f = |M|$ and $d = 0$ we obtain at least $6(f_z - f) + 15(d_z - d) + 3n_i + 3|C_i| + 1 \geq 6(f_z - f) + 15d_z + 3f + 3f + 1 = 6f_z + 15d_z + 1$ free interior edges in G_z , which proves (F3).

Case 2: G_z is a leaf of \mathcal{B}' (but not the only node) and $f_i + d_i > 0$. We distinguish two subcases. If M contains no edge of T_i , then Lemma 17(2) yields $f = |M| \leq n_i - 2f_i - 5d_i - 1$. By Lemma 19, we find at least

$$\begin{aligned} & 6(f_z - f_i - f) + 15(d_z - d_i) + 3n_i + 3|C_i| + 3|Q| - 3 \\ & \geq 6f_z - 6f_i - 3f - 3(n_i - 2f_i - 5d_i - 1) + 15(d_z - d_i) + 3n_i + 3f + 3|Q| - 3 \\ & = 6f_z + 15d_z + 3|Q| \end{aligned}$$

free interior edges in G_z . Since $i \in Q$, we have $|Q| \geq 1$ and (F2) follows.

Otherwise, M contains an edge of T_i . Then the parent G_p of G_i in \mathcal{B} is merged into G_z . Lemma 17(2) yields $f = |M| \leq n_i - 2f_i - 5d_i$. By Lemma 19 we find at least

$$\begin{aligned} & 6(f_z - f_i - f) + 15(d_z - d_i) + 3n_i + 3|C_i| + 3|Q| - 3 \\ & \geq 6f_z - 6f_i - 3f - 3(n_i - 2f_i - 5d_i) + 15(d_z - d_i) + 3n_i + 3f + 3|Q| - 3 \\ & = 6f_z + 15d_z + 3(|Q| - 1) \end{aligned}$$

free interior edges in G_z . Since $\{i, p\} \subseteq Q$, we have $|Q| \geq 2$ and (F2) follows.

Case 3: G_z is a leaf of \mathcal{B}' (but not the only node) and $f_i = d_i = 0$. We distinguish two subcases. If $|M| \leq n_i - 1$, then Lemma 19 guarantees $6f_z - 3f - 3f + 15d_z + 3n_i + 3|C_i| + 3|Q| - 3 \geq 6f_z - 3(n_i - 1) - 3f + 15d_z + 3n_i + 3f + 3|Q| - 3 = 6f_z + 15d_z + 3|Q|$ free interior edges in G_z , which together with $i \in Q$ proves (F2).

Otherwise, we have $f = |M| = n_i$. We claim that M contains an edge of T_i (otherwise \mathcal{T}_i would be a checkerboard). Suppose that M does not contain any edge of T_i . Then no

edge of T_i is incident to any triangle from \mathcal{T}_i . (Otherwise, we could replace the edge in M that is incident to such a triangle by the edge shared with T_i . As the triangles in \mathcal{T}_i are pairwise edge-disjoint, the replaced edge is incident to only one triangle from \mathcal{T}_i . The result is a 4-connector of the same size as M , but with an edge of T_i . This contradicts our choice of M .) Therefore, all $3n_i$ interior edges of G_i are incident to triangles in \mathcal{T}_i , and so \mathcal{T}_i is a checkerboard. This proves our claim.

As M contains an edge of T_i , the parent G_p of G_i in \mathcal{B} is merged into G_z as well. By Lemma 19 we find at least $6(f_z - n_i) + 15d_z + 3n_i + 3n_i + 3|Q| - 3 \geq 6f_z + 15d_z + 3(|Q| - 1)$ free interior edges in G_z , which together with $\{i, p\} \subseteq Q$ proves (F2).

Summary. In all cases we have shown that the resulting 4-block tree \mathcal{B}' satisfies our invariant. Thus the resulting 4-connected graph G' has $n + d$ vertices and at least $6f + 15d + 3$ edges, where f and d denote the number of flip and dummy flip operations, respectively, that were executed during the algorithm. Being a maximal planar graph, G' contains exactly $3(n + d) - 6$ edges. Therefore, $6f + 15d + 3 \leq 3(n + d) - 6$ and so $2f + 4d \leq n - 3$, as required. This completes the proof of Lemma 14.

Step 2: Eliminating dummy vertices. At this stage we have a 4-connected planar graph G' . By Tutte's Theorem such a graph is Hamiltonian, so consider some Hamiltonian cycle H' of G' . It remains to argue how G' and H' can be used to obtain a short sequence of edge flips that transform the original graph G into a Hamiltonian graph G'' . The following lemma in combination with Lemma 14 completes the proof for Theorem 4.

► **Lemma 22.** *Suppose that G' has been obtained from G using f flips and d dummy flips. Then G can be transformed into a Hamiltonian maximal planar graph using at most $f + 2d$ edge flips.*

Proof of Corollary 6. The following analogue of Lemma 22, combined with Lemma 14 and the characterization of Bernhart and Kainen [5], proves Corollary 6.

► **Lemma 23.** *Suppose that G' has been obtained from G using f subdivisions and d dummy flips. Then G can be subdivided into a subhamiltonian graph using at most $f + 2d$ subdivision vertices (at most one per edge).*

Acknowledgements This work began at the 12th Gremo's Workshop on Open Problems (GWOP), June 30–July 4, 2014, in Val Sinestra (GR), Switzerland. We thank all participants for the productive and positive atmosphere, and in particular Radoslav Fulek, Anna Gundert, Malte Milatz, Bettina Speckmann, Sebastian Stich, and Tibor Szabó for inspiring discussions.

References

- 1 Bernardo M. Ábrego, Oswin Aichholzer, Silvia Fernández-Merchant, Pedro Ramos, and Gelasio Salazar. Shellable drawings and the cylindrical crossing number of K_n . *Discrete Comput. Geom.*, 52(4):743–753, 2014.
- 2 Oswin Aichholzer, Clemens Huemer, and Hannes Krasser. Triangulations without pointed spanning trees. *Comput. Geom. Theory Appl.*, 40(1):79–83, 2008.
- 3 T. Alastair and J. Nicholson. Permutation procedure for minimising the number of crossings in a network. *Proc. IEE*, 115(1):21–26, 1968.

- 4 Patrizio Angelini, David Eppstein, Fabrizio Frati, Michael Kaufmann, Sylvain Lazard, Tamara Mchedlidze, Monique Teillaud, and Alexander Wolff. Universal point sets for drawing planar graphs with circular arcs. *Journal of Graph Algorithms and Applications*, 18(3):313–324, 2014.
- 5 Frank Bernhart and Paul C. Kainen. The book thickness of a graph. *J. Combin. Theory, Ser. B* 27:320–331, 1979.
- 6 J. Blažek and M. Koman. A minimal problem concerning complete plane graphs. In M. Fiedler, editor, *Theory of graphs and its applications*, pages 113–117. Czech. Acad. of Sci., 1964.
- 7 John Adrian Bondy and U. S. R. Murty. *Graph Theory*, volume 244 of *Graduate texts in Mathematics*. Springer, Berlin, 2008.
- 8 Prosenjit Bose, Jurek Czyzowicz, Zhicheng Gao, Pat Morin, and David R. Wood. Simultaneous diagonal flips in plane triangulations. *J. Graph Theory*, 54(4):307–330, 2007.
- 9 Prosenjit Bose and Ferran Hurtado. Flips in planar graphs. *Comput. Geom. Theory Appl.*, 42(1):60–80, 2009.
- 10 Prosenjit Bose, Dana Jansens, André van Renssen, Maria Saumell, and Sander Verdonschot. Making triangulations 4-connected using flips. *Comput. Geom. Theory Appl.*, 47(2):187–197, 2014.
- 11 Prosenjit Bose and Sander Verdonschot. A history of flips in combinatorial triangulations. In *Computational Geometry—XIV Spanish Meeting on Computational Geometry, EGC 2011*, volume 7579 of *LNCS*, pages 29–44. Springer, Berlin, 2012.
- 12 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Berlin, 4 edition, 2010.
- 13 David Eppstein. Universal point sets for planar graph drawings with circular arcs. Presentation at the 25th Canadian Conference on Computational Geometry, Waterloo, Canada, <http://www.ics.uci.edu/~eppstein/pubs/AngEppFra-CCCG-13-slides.pdf>, 2013.
- 14 Emilio Di Giacomo, Walter Didimo, and Giuseppe Liotta. Spine and radial drawings. In Roberto Tamassia, editor, *Handbook of Graph Drawing and Visualization*, chapter 8, pages 247–284. CRC press, Boca Raton, FL, 2013.
- 15 Emilio Di Giacomo, Walter Didimo, Giuseppe Liotta, and Stephen K. Wismath. Curve-constrained drawings of planar graphs. *Comput. Geom. Theory Appl.*, 30(1):1–23, 2005.
- 16 Branko Grünbaum. *Convex Polytopes*, volume 221 of *Graduate texts in Mathematics*. Springer, Berlin, 2003.
- 17 Jochen Harant, Mirko Horňák, and Zdzisław Skupień. Separating 3-cycles in plane triangulations. *Discrete Math.*, 239(1):127–136, 2001.
- 18 Goos Kant. A more compact visibility representation. *Int. J. Comput. Geometry Appl.*, 7(3):197–210, 1997.
- 19 Michael Kaufmann and Roland Wiese. Embedding vertices at points: Few bends suffice for planar graphs. *J. Graph Algorithms Appl.*, 6(1):115–129, 2002.
- 20 Hideo Komuro. The diagonal flips of triangulations on the sphere. *Yokohama Math. J.*, 44:115–122, 1997.
- 21 Ryuichi Mori, Atsuhiko Nakamoto, and Katsuhiko Ota. Diagonal flips in Hamiltonian triangulations on the sphere. *Graphs Combin.*, 19(3):413–418, 2003.
- 22 János Pach and Rephael Wenger. Embedding planar graphs at fixed vertex locations. *Graphs Combin.*, 17:717–728, 2001.
- 23 Lionel Pournin. The diameter of associahedra. *Adv. Math.*, 259:13–42, 2014.
- 24 Thomas L. Saaty. The minimum number of intersections in complete graphs. *Proceedings of the National Academy of Sciences of the United States of America*, 52:688–690, 1964.
- 25 Daniel D. Sleator, Robert E. Tarjan, and William P. Thurston. Rotation distance, triangulations, and hyperbolic geometry. *J. Amer. Math. Soc.*, 1:647–682, 1988.

- 26 Avi Wigderson. The complexity of the Hamiltonian circuit problem for maximal planar graphs. Technical Report 298, Princeton University, 1982.

Tractable Probabilistic μ -Calculus That Expresses Probabilistic Temporal Logics*

Pablo Castro^{1,2}, Cecilia Kilmurray^{1,2}, and Nir Piterman³

- 1 Departamento de Computación, FCEFQyN, Universidad Nacional de Río Cuarto, Río Cuarto, Argentina
`{ckilmurray, pcastro}@dc.exa.unrc.edu.ar`
- 2 Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
- 3 Department of Computer Science, University of Leicester, Leicester, UK
`nir.piterman@leicester.ac.uk`

Abstract

We revisit a recently introduced probabilistic μ -calculus and study an expressive fragment of it. By using the probabilistic quantification as an atomic operation of the calculus we establish a connection between the calculus and obligation games. The calculus we consider is strong enough to encode well-known logics such as PCTL and PCTL*. Its game semantics is very similar to the game semantics of the classical μ -calculus (using parity obligation games instead of parity games). This leads to an optimal complexity of $\text{NP} \cap \text{co-NP}$ for its finite model checking procedure. Furthermore, we investigate a (relatively) well-behaved fragment of this calculus: an extension of PCTL with fixed points. An important feature of this extended version of PCTL is that its model checking is only exponential w.r.t. the alternation depth of fixed points, one of the main characteristics of Kozen's μ -calculus.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases μ -calculus, probabilistic logics, model checking, game semantics

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.211

1 Introduction

In recent years, probabilistic model checking has received an increasing attention in the area of system verification; tools like PRISM [8] and LiQuor [4] enable the automatic verification of quantitative properties of systems (and a lot more); these kinds of properties are essential for the verification of network protocols, critical systems and randomized algorithms, to name a few examples.

Some of the most prominent probabilistic temporal logics used for model checking are PCTL, the probabilistic counterpart of CTL, and PCTL*, the probabilistic counterpart of CTL*. In particular, PCTL has a clear semantics, and its model checking procedure can be performed in polynomial time. The definition of a probabilistic μ -calculus that provides a unifying formalism for probabilistic temporal logics has been an active field of research in the area, such a formalism could provide to probabilistic model checking the same benefits as those given by Kozen's μ -calculus to qualitative model checking. The μ -calculus [12] is a powerful temporal logic that combines many useful features. It generalizes modal logic by adding fixpoint operators, it has a compact, extremely powerful, and very pleasing mathematical

* This work was partially supported by FP7-PEOPLE-IRESES-2011 MEALS project and EPSRC EP/L007177/1 project.

theory, its model checking problem is polynomial in the length of the formula and only exponential in its alternation depth [7]. Most of the temporal logics used in computer science can be encoded into fragments of it; and, in addition, it has strong connections to two-player games and automata theory, which lead to optimal decision procedures for it.

Here, we revisit the probabilistic μ -calculus introduced by Mio and Simpson in [14, 15], however, we suggest to use probabilistic quantification as an atomic operation. The resulting probabilistic μ -calculus (named μ^p -calculus) enjoys many of the qualities of the discrete μ -calculus. We show that the logic is expressive enough to capture PCTL and PCTL*. We establish a tight connection between our logic and the recently introduced obligation parity games [3]. In particular, we provide a game semantics for μ^p -calculus using such games. When considering finite-state model checking, the games provide an optimal decision procedure in $\text{NP} \cap \text{co-NP}$ (compared with 3EXPTIME for the logic of Mio and Simpson); where optimality is w.r.t. model checking the discrete μ -calculus, which has the same complexity. In contrast to the “normal” μ -calculus, we lose the connection between the alternation depth of the formula and the complexity of model checking. We also propose a well-behaved fragment of μ^p -calculus, this logic is mainly an extension of PCTL with fixpoints, we prove that the complexity of model checking for this fragment is only exponential in the alternation depth of quantifiers; as mentioned above, this is an important characteristic of standard μ -calculus.

The paper is organized as follows. In Section 2 we introduce the basic definitions needed to tackle the rest of the paper. The probabilistic μ -calculus is introduced in Section 3 and then its expressivity is investigated. We then present the game semantics in Section 4. In Section 5 we show that a well-known “hard” problem in $\text{NP} \cap \text{co-NP}$ can be reduced to model checking formulas of μ^p -calculus with only one fixpoint operator. A well-behaved fragment of this logic is described in Section 6. Finally, we discuss related work and add final remarks.

2 Preliminaries

In this section we briefly introduce some basic concepts. We denote the set of real numbers between 0 and 1 as $[0, 1]$. Given a set S we denote by $\vec{0}(S)$ the function $\vec{0}(S)(s) = 0$ for every $s \in S$ and by $\vec{1}(S)$ the function $\vec{1}(S)(s) = 1$ for every $s \in S$. When S is clear from the context we write $\vec{0}$ and $\vec{1}$. Given a universe U and a subset $S \subseteq U$ we write χ_S for the function $\chi_S(s) = 1$ if $s \in S$ and $\chi_S(s) = 0$ for $s \notin S$.

A *Kripke structure* over a set AP of atomic propositions is a tuple $K = \langle S, R, L, s_0 \rangle$, where S is a (countable) set of locations, $R \subseteq S \times S$ is a relation such that for every $s \in S$ we have that $R(s)$ is finite, $L : AP \rightarrow 2^S$ is a labeling function and $s_0 \in S$ is an initial location. A *Markov chain* over a set AP of atomic letters is a tuple $M = \langle K, P \rangle$, where K is a Kripke structure and $P : R \rightarrow (0, 1]$ is such that for every $s \in S$ we have $\sum_{(s, s') \in R} P(s, s') = 1$. Sometimes it will be convenient to consider $P : S \times S \rightarrow [0, 1]$ by associating $P(s, s') = 0$ for every $(s, s') \notin R$. For a location $s \in S$ we denote by M_s the Markov chain obtained from M by setting s to the initial location. A path $\pi = s_0, s_1, \dots$ is a finite or infinite sequence of locations such that for every $0 \leq i < n$ we have $P(s_i, s_{i+1}) > 0$. If $\pi = s_0, \dots, s_n$ is finite, we denote by $\text{measure}_M(\pi) = \prod_{i=0}^{n-1} P(s_i, s_{i+1})$ the measure of (the set of infinite paths that extend) π . Given a (Borel) set of paths Π starting from the same state s , we denote by $\text{measure}_M(\Pi)$ the measure of Π . Note that every Markov chain can be interpreted as a Kripke structure by looking on the embedded Kripke structure.

PCTL formulas over a set AP are defined as state formulas (Φ) and path formulas (Ψ)

as follows. Let $J = \{>, \geq\} \times [0, 1]$ be the set of bounds.

$$\Phi ::= p_i \mid \neg p_i \mid \Phi_1 \vee \Phi_2 \mid \Phi_1 \wedge \Phi_2 \mid \mathcal{P}_J(\Psi) \quad \Psi ::= \bigcirc \Phi \mid \Phi \mathcal{U} \Phi \mid \Phi \mathcal{W} \Phi$$

Here \mathcal{W} is the weak until (i.e., it allows the first operand to hold forever). As usual we introduce the abbreviations F and G . State formulas are *formulas*. The semantics of PCTL associates with every formula a set of states. We denote by $\llbracket \varphi \rrbracket_M$ the set of states of M that satisfy φ . For every path formula φ and state s of M , $\text{measure}_M(s, \varphi)$ is the measure of paths starting in s that satisfy φ . The semantics and intuitions of PCTL formulas are as usual, see [1].

We define μ -calculus over Kripke structures with the following syntax.

$$\Phi ::= p_i \mid \neg p_i \mid X_i \mid \diamond \Phi \mid \square \Phi \mid \Phi_1 \vee \Phi_2 \mid \Phi_1 \wedge \Phi_2 \mid \mu X_i. \Phi \mid \nu X_i. \Phi$$

Where $p_i \in AP$, $\mathcal{V} = \{X_0, X_1, \dots\}$ is an enumerable set of variables, and $X_i \in \mathcal{V}$. The notions of *open* and *closed* formulas are as usual. The semantics of a μ -calculus formula over a Kripke structure $K = \langle S, R, L, s_0 \rangle$ is given w.r.t. assignments to variables in \mathcal{V} . An assignment $\rho : \mathcal{V} \rightarrow (S \rightarrow \{0, 1\})$ associates a function from the states to $\{0, 1\}$ with every variable in \mathcal{V} . Given an assignment ρ we set $\rho[f/X]$ to be the assignment that associates the function f with X and $\rho(Y)$ with every $Y \neq X$. We use the notation of a function into $\{0, 1\}$ instead of set notation to facilitate the discussion in the rest of the paper. The semantics of a formula φ in structure K with respect to assignment ρ , denoted $\llbracket \varphi \rrbracket_K^\rho$, is defined as follows.

$$\begin{aligned} \llbracket p_i \rrbracket_K^\rho &= \chi_{L(p_i)} & \llbracket \neg p_i \rrbracket_K^\rho &= 1 - \chi_{L(p_i)} \\ \llbracket X \rrbracket_K^\rho &= \rho(X) & \llbracket \varphi_1 \vee \varphi_2 \rrbracket_K^\rho &= \max(\llbracket \varphi_1 \rrbracket_K^\rho, \llbracket \varphi_2 \rrbracket_K^\rho) \\ \llbracket \diamond \varphi \rrbracket_K^\rho &= \lambda s. \max_{(s, s') \in R} \llbracket \varphi \rrbracket_K^\rho(s') & \llbracket \varphi_1 \wedge \varphi_2 \rrbracket_K^\rho &= \min(\llbracket \varphi_1 \rrbracket_K^\rho, \llbracket \varphi_2 \rrbracket_K^\rho) \\ \llbracket \square \varphi \rrbracket_K^\rho &= \lambda s. \min_{(s, s') \in R} \llbracket \varphi \rrbracket_K^\rho(s') & \llbracket \mu X. \varphi \rrbracket_M^\rho &= \text{lfp}(\llbracket \varphi \rrbracket_M^{\rho[f/X]}) \\ \llbracket \mu X. \varphi \rrbracket_M^\rho &= \text{lfp}(\llbracket \varphi \rrbracket_M^{\rho[f/X]}) & \llbracket \nu X. \varphi \rrbracket_M^\rho &= \text{gfp}(\llbracket \varphi \rrbracket_M^{\rho[f/X]}) \end{aligned}$$

Note that the semantics of a formula where all variables are bound by fixpoint operators is independent of the assignment ρ . The interested reader is referred to [16] for an in-depth introduction to μ -calculus.

3 A Probabilistic μ -Calculus

In this section we present our version of probabilistic μ -calculus (denoted μ^p -calculus). Unlike the “normal” μ -calculus, μ^p -calculus is two sorted. We distinguish between qualitative formulas (that get values in $\{0, 1\}$) and quantitative formulas (that get values in $[0, 1]$).¹ Although the logic is a subset of the probabilistic μ -calculus of Mio and Simpson [15] we give a direct definition of its semantics without relying on their results.

Given an enumerable set of variables $\mathcal{V} = \{X_0, X_1, \dots\}$, the syntax of the logic is given by the following grammar, where Ψ are qualitative formulas, and Φ are quantitative formulas.

$$\begin{aligned} J &::= \{>, \geq\} \times [0, 1] \\ \Psi &::= p_i \mid \neg p_i \mid \Psi_1 \vee \Psi_2 \mid \Psi_1 \wedge \Psi_2 \mid [\Phi]_J \mid \nu X_i. \Psi \mid \mu X_i. \Psi \\ \Phi &::= \Psi \mid X_i \mid \Phi_1 \vee \Phi_2 \mid \Phi_1 \wedge \Phi_2 \mid \diamond \Phi \mid \square \Phi \mid \bigcirc \Phi \mid \nu X_i. \Phi \mid \mu X_i. \Phi \end{aligned} \tag{1}$$

¹ This is not to be confused with qualitative PCTL, where the bounds are restricted to ≥ 1 and > 0 .

We say that variable X_i is *bound* in $\sigma X_i.\varphi(X_i)$ for $\sigma \in \{\mu, \nu\}$. A variable that is not bound is *free*. A *formula* is a *qualitative formula* with no free variables. That is, at the top level we consider only formulas that can be evaluated to $\{0, 1\}$. Note that we add to the existential and universal next operators of μ -calculus the (probabilistic) next operator and the probabilistic quantification operator.

The semantics of a formula ψ over a Markov chain M is defined with respect to an interpretation ρ , which associates a function from states to real values in $[0, 1]$ with each variable appearing in ψ . Formally, for $\rho : \mathcal{V} \rightarrow (S \rightarrow [0, 1])$ the semantics $\llbracket \psi \rrbracket_M^\rho : S \rightarrow [0, 1]$ is defined as follows:

$$\begin{array}{ll} \llbracket p_i \rrbracket_M^\rho = \chi_{L(p_i)} & \llbracket \neg p_i \rrbracket_M^\rho = 1 - \chi_{L(p_i)} \\ \llbracket X \rrbracket_M^\rho = \rho(X) & \\ \llbracket \varphi_1 \vee \varphi_2 \rrbracket_M^\rho = \max(\llbracket \varphi_1 \rrbracket_M^\rho, \llbracket \varphi_2 \rrbracket_M^\rho) & \llbracket \varphi_1 \wedge \varphi_2 \rrbracket_M^\rho = \min(\llbracket \varphi_1 \rrbracket_M^\rho, \llbracket \varphi_2 \rrbracket_M^\rho) \\ \llbracket \bigcirc \varphi \rrbracket_M^\rho = \lambda s. \sum_{s'} P(s, s') \llbracket \varphi \rrbracket_M^\rho(s') & \llbracket [\varphi]_J \rrbracket_M^\rho = (\llbracket \varphi \rrbracket_M^\rho(s) J ? 1 : 0) \\ \llbracket \diamond \varphi \rrbracket_M^\rho = \lambda s. \max_{(s, s') \in R} \llbracket \varphi \rrbracket_M^\rho(s') & \llbracket \square \varphi \rrbracket_M^\rho = \lambda s. \min_{(s, s') \in R} \llbracket \varphi \rrbracket_M^\rho(s') \\ \llbracket \mu X.\varphi \rrbracket_M^\rho = \text{lfp}(\llbracket \varphi \rrbracket_M^{\rho[f/X]} \rrbracket) & \llbracket \nu X.\varphi \rrbracket_M^\rho = \text{gfp}(\llbracket \varphi \rrbracket_M^{\rho[f/X]} \rrbracket) \end{array}$$

That is, the value of the probabilistic next is the average value over successors and the probabilistic quantification compares the value with the given bound. Even though the semantics is quite similar to the semantics of μ -calculus the former is restricted to functions of the type $f : S \rightarrow \{0, 1\}$ and here the functions are $f : S \rightarrow [0, 1]$. That is, functions associate real values with states.

It is simple to see that all these transformers are monotonic. In particular, if $\rho_1 \leq \rho_2$, that is for every $X \in \mathcal{V}$ and every $s \in S$ we have $\rho_1(X)(s) \leq \rho_2(X)(s)$, then $\llbracket \varphi \rrbracket_M^{\rho_1} \leq \llbracket \varphi \rrbracket_M^{\rho_2}$. For instance, consider a formula of the form $[\varphi]_J$. We have to show that, if $\llbracket [\varphi]_J \rrbracket_M^{\rho_1}(s) = 1$, then $\llbracket [\varphi]_J \rrbracket_M^{\rho_2} = 1$. However, if $\rho_1 \leq \rho_2$ it follows that $\llbracket \varphi \rrbracket_M^{\rho_1} \leq \llbracket \varphi \rrbracket_M^{\rho_2}$. So, if $\llbracket \varphi \rrbracket_M^{\rho_1} J$, then also $\llbracket \varphi \rrbracket_M^{\rho_2} J$. It follows from the Knaster-Tarski theorem that fixed-points are well defined.

It is possible to show that our calculus is closed under negation. For this, we need to consider the usual dualizations between the standard operators. In addition, the probabilistic next is its own dual and the probabilistic quantification has to be replaced with the dual probabilistic quantification. That is, $[\cdot]_{>1-p}$ is the dual of $[\cdot]_{\geq p}$ and $[\cdot]_{\geq 1-p}$ is the dual of $[\cdot]_{>p}$. We now show that the definition of qualitative formulas is indeed justified.

► **Lemma 1.** *For every qualitative formula φ we have $\llbracket \varphi \rrbracket_M^\rho \in \{0, 1\}$.*

Proof. We can show that the semantics of all operators in the qualitative fragment are functions whose range is $\{0, 1\}$. This holds trivially for propositions. Given two functions whose range is $\{0, 1\}$ clearly, min and max return such functions as well.

For $[\varphi]_J$ this follows directly from the definition. ◀

3.1 Expressing the μ -Calculus

We show that μ^p -calculus is strong enough to express the μ -calculus over the embedded Kripke structure without using the existential and universal next operators. We include this construction mostly as justification for the hardness of model checking the μ^p -calculus over finite-state Markov chains.

Given a μ -calculus formula φ , let $p(\varphi)$ denote the formula obtained from φ by the following recursive transformation.

$$\begin{array}{lll} p(p_i) = p_i & p(\psi_1 \vee \psi_2) = p(\psi_1) \vee p(\psi_2) & p(\diamond \psi) = [\bigcirc p(\psi)]_{>0} \\ p(\neg p_i) = \neg p_i & p(\psi_1 \wedge \psi_2) = p(\psi_1) \wedge p(\psi_2) & p(\square \psi) = [\bigcirc p(\psi)]_{\geq 1} \\ p(X) = X & p(\mu X.\psi) = \mu X.p(\psi) & p(\nu X.\psi) = \nu X.p(\psi) \end{array}$$

That is, we replace the existential next operator by a probabilistic quantification of more than 0, and the universal next operator by a probabilistic quantification of at least 1.

► **Lemma 2.** *For every Markov chain $M = \langle K, P \rangle$ we have $\llbracket p(\varphi) \rrbracket_M^\rho = \llbracket \varphi \rrbracket_K^\rho$.*

We notice that, in general, it is not clear how to express the universal and existential next operators without including them explicitly. This is because the $[\cdot]_J$ operator also resets the value to 0 or 1. An additional comment regarding these operators is included in Section 5. It follows that μ^p -calculus is strong enough to express all standard temporal logics such as CTL, LTL, and CTL*.

3.2 Expressing PCTL

We show that μ^p -calculus can express PCTL. Given a PCTL formula φ , let $t(\varphi)$ denote the formula obtained from φ by the following recursive transformation.

$$\begin{aligned} t(p_i) &= p_i & t(\psi_1 \vee \psi_2) &= t(\psi_1) \vee t(\psi_2) & t(\mathcal{P}_J(\psi)) &= [t(\psi)]_J \\ t(\neg p_i) &= \neg p_i & t(\psi_1 \wedge \psi_2) &= t(\psi_1) \wedge t(\psi_2) & t(\bigcirc \psi) &= \bigcirc t(\psi) \\ t(\psi_1 \mathcal{U} \psi_2) &= \mu X . t(\psi_2) \vee (t(\psi_1) \wedge \bigcirc X) & t(\psi_1 \mathcal{W} \psi_2) &= \nu X . t(\psi_2) \vee (t(\psi_1) \wedge \bigcirc X) \end{aligned}$$

That is, we use fixpoint operators to unwind until and weak until operators in the standard way this is done with CTL and μ -calculus. We note that this construction is essentially identical to the encoding of CTL in μ -calculus, which is used also in [15] (though the main complexity in their construction is in expressing the probabilistic quantification, which is part of the syntax in our setting). Due to its importance we include it in full here.

► **Lemma 3.** *For every Markov chain M and PCTL formula φ we have $\llbracket \varphi \rrbracket_M = \llbracket t(\varphi) \rrbracket_M^\rho$.*

The conversion of PCTL* to μ^p -calculus is also possible. As for PCTL, it is essentially identical to the translation of CTL* to μ -calculus, with the caveat that we have to replace nondeterministic automata by deterministic automata. The usage of deterministic automata is, similarly, required for the handling of LTL for probabilistic model checking [2]. We note that this implies that PCTL* is expressible (through the same construction with the additional encoding of the probabilistic thresholds) also in the probabilistic μ -calculus of Mio and Simpson.

4 Game Semantics

First, we describe the intuition behind the game semantics, and only then formally define the games. Given a formula φ and a Markov chain $M = \langle K, P \rangle$, where $K = \langle S, R, L, s^{in} \rangle$, we define a game whose configurations correspond to locations of M and subformulas of φ . The semantics is defined in terms of a two-player stochastic obligation parity game [3]. Such games include configurations of players 0 and 1 as well as probabilistic configurations. The winning condition is a combination of a parity condition and obligations (for how much player 0 has to win) on some configurations. Player 0 is the *verifier*, who tries to prove that the formula holds, and player 1 is the *refuter*, who tries to prove that the formula does not hold. Each configuration has a valuation for each player. In general, the value of a configuration, denoted by $val_i(s, \varphi)$ for $i \in \{0, 1\}$, is a value in $[0, 1]$; $val_i(s, \varphi) = 1$ means that player i wins (completely) from a configuration. For every qualitative (sub)formula the value of (s, φ) is either 0 or 1. Intuitively, if $val_0(s, \varphi) = 1$, then the formula is true in M . For propositions, (s, p) , player 0 wins when $s \in L(p)$ and she loses otherwise (configurations

with $\neg p$ are dual). Configurations $(s, \varphi_1 \vee \varphi_2)$ are verifier configurations, and she chooses a successor (s, φ_i) . Configurations $(s, \varphi_1 \wedge \varphi_2)$ are refuter configurations, and she selects a successor (s, φ_i) .

For a fixpoint $\sigma \in \{\mu, \nu\}$, from configuration $(s, \sigma X.\varphi)$ the game progresses to (s, φ) ; while from configurations (s, X) the game progresses to $(s, \sigma X.\varphi)$ where $\sigma X.\varphi$ is the subformula binding X . Interesting cases are the probabilistic operators: from configuration $(s, [\varphi]_J)$ the game progresses with no choice to (s, φ) . However, the former configurations have the obligation J associated with them. That is, from these obligation states player 0 wins completely (value 1) if she wins with a value satisfying J from the successor configuration. These three types of configurations (fixpoint related and probabilistic quantification) are deterministic configurations. We associate them with the probabilistic player and assign the probability 1 to the single successor. The next operators are treated as follows. A configuration of the form $(s, \diamond\varphi)$ is a verifier configuration from where she chooses a successor s' of s and moves to configuration (s', φ) . A configuration of the form $(s, \square\varphi)$ is a refuter configuration from where she chooses a successor s' of s and moves to configuration (s', φ) . A configuration of the form $(s, \bigcirc\varphi)$ is a probabilistic configuration with successors (s', φ) for every successors s' of s . Furthermore, the probability of $((s, \bigcirc\varphi), (s', \varphi))$ is $\kappa(s, s')$. It follows that the only (meaningful) probabilistic configurations are those corresponding to the probabilistic next of the calculus. The parity condition in the game arises from the alternation depth of formulas.

4.1 Parity Obligation Games

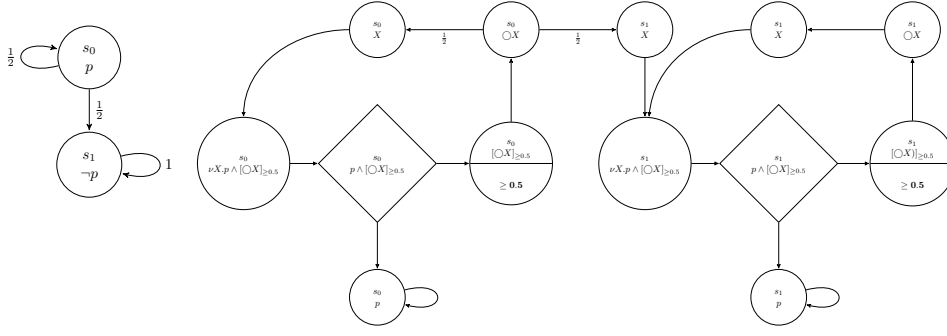
We give a short introduction to obligation parity games. The notion of winning (and value) in an obligation game is quite involved and we refer the reader to [3] for an in-depth introduction.

A parity obligation game is $G = (V, (V_0, V_1, V_p), E, \kappa, \mathcal{G})$, where V is a set of configurations, V_0 , V_1 , and V_p form a partition of V to player 0, player 1, and stochastic configurations, respectively, $E \subseteq V \times V$ is the set of edges, κ associates a probabilistic distribution with the edges leaving every configuration in V_p , i.e., for every $v \in V_p$ we have $\sum_{(v, v') \in E} \kappa(v, v') = 1$ and for every $(v, v') \notin E$ we have $\kappa(v, v') = 0$, and $\mathcal{G} = (c, O)$ is the winning condition, where $c : V \rightarrow [0..m]$ is a parity priority function, with m as its *index*, and $O : V \rightarrow \{\perp\} \cup (\{>, \geq\} \times [0, 1])$ is the obligation function. A configuration v such that $O(v) \neq \perp$ is called an *obligation configuration*.

► **Theorem 4.** [3] *For every configuration $v \in V$ there is a value $val_i(v) \in [0, 1]$ such that $val_0(v) + val_1(v) = 1$. Furthermore, for every obligation configuration v we have $val_i(v) \in \{0, 1\}$. For a configuration v of a finite parity obligation game, one can decide whether $val_i(v) \geq r$ in $NP \cap co-NP$ and $val_i(v)$ can be computed in exponential time.*

4.2 Model Checking Game

We are now ready to formally define the model checking games. Let $sub(\varphi)$ denote the subformulas of φ according to the grammar in (1). We use the notion of *alternation depth* as defined, e.g., in [7]. Roughly speaking, the alternation depth of a formula is a measure of its complexity. Essentially, it is the largest number of μ and ν alternations that appear in the formula. Furthermore, let d be $ad(\varphi)$, with every subformula φ' of φ we can associate a *color* $c(\varphi')$ as follows. If $\varphi' = \nu X.\psi$ then $c(\varphi') = 2(d - ad(\varphi'))$. If $\varphi' = \mu X.\psi$ then $c(\varphi') = 2(d - ad(\varphi')) + 1$. For every other formula φ' we set $c(\varphi') = 2d - 1$. It follows that $c(\varphi')$ is in the range $[0..2d - 1]$.



■ **Figure 1**
Markov chain M .

■ **Figure 2** The game $G_{M,\varphi}$.

► **Definition 5.** Consider a Markov chain $M = \langle K, P \rangle$, where $K = \langle S, R, L, s^{in} \rangle$ and a formula φ . We define the game $G_{M,\varphi} = (V, E, (V_0, V_1, V_p), \kappa, \mathcal{G})$ as follows:

- $V = \{(s, \psi) \mid s \in S \wedge \psi \in \text{sub}(\varphi)\}$,
 - $V_0 = \{(s, \psi_1 \vee \psi_2), (s, \diamond\psi)\}$, $V_1 = \{(s, \psi_1 \wedge \psi_2), (s, \square\psi)\}$, and $V_p = V \setminus (V_0 \cup V_1)$,
 - $E = \{((s, p), (s, p)), ((s, \neg p), (s, \neg p)) \mid p \text{ is a proposition}\} \cup \{((s, [\psi]_J), (s, \psi))\}$
 $\cup \{((s, \psi_1 \vee \psi_2), (s, \psi_i)) \mid i \in \{1, 2\}\} \cup \{((s, \psi_1 \wedge \psi_2), (s, \psi_i)) \mid i \in \{1, 2\}\}$
 $\cup \{((s, \diamond\psi), (s', \psi)) \mid P(s, s') > 0\} \cup \{((s, \square\psi), (s', \psi)) \mid P(s, s') > 0\}$
 $\cup \{((s, \bigcirc\psi), (s', \psi)) \mid P(s, s') > 0\} \cup \{((s, \sigma X.\psi), (s, \psi)) \mid \sigma \in \{\nu, \mu\}\}$
 $\cup \{((s, X), (s, \sigma X.\psi)) \mid \sigma X.\psi \text{ is the subformula binding } X \text{ and } \sigma \in \{\mu, \nu\}\}$
 - $\kappa((s, \bigcirc\psi)(s', \psi)) = P(s, s')$, and $\kappa((s, \psi)(s, \psi')) = 1$ for every other (s, ψ) and $((s, \psi), (s, \psi')) \in E$.
 - $\mathcal{G} = (c, O)$, where $O(s, [\psi]_J) = J$ and $O(s, \psi) = \perp$ for every other formula;
- $$c(s, \psi) = \begin{cases} c(\psi) & \text{If } \psi \text{ is not a proposition.} \\ 0 & \text{If } (\psi = p \text{ and } s \in L(p)) \text{ or } (\psi = \neg p \text{ and } s \notin L(p)) \\ 1 & \text{If } (\psi = p \text{ and } s \notin L(p)) \text{ or } (\psi = \neg p \text{ and } s \in L(p)) \end{cases}$$

Let us present a simple example to obtain a first taste of μ^p -calculus and its game semantics. Consider Markov chain M in Fig. 1 and the formula $\varphi : \nu X.p \wedge [\bigcirc X]_{\geq 0.5}$. The alternation depth of φ is 1. It follows that $c(s_0, p) = 0$, $c(s_1, p) = 1$, and for every other configuration $c(v) = 0$. The game obtained from φ and M is shown in Fig. 2. In this graphic, we use circles to denote probabilistic configurations and diamonds to denote player 1 configurations. Note that there are no player 0 configurations in this game. The only configurations with obligations are $(s_0, [\bigcirc X]_{\geq 0.5})$ and $(s_1, [\bigcirc X]_{\geq 0.5})$. Let us calculate the value of $(s_0, \nu X.p \wedge [\bigcirc X]_{\geq 0.5})$, the unique successor of this configuration is a configuration where the refuter plays. The configuration (s_0, p) is colored 0 as $p \in L(s_0)$. Thus, the refuter should avoid this sink state as it is winning for verifier and select the other successor. This is a probabilistic configuration with obligation $\geq \frac{1}{2}$. Then note that player 0 can ensure that with probability at least $\frac{1}{2}$ she either wins by reading (s_0, p) or gets to the same obligation configuration, with color 0 the minimal in the loop. Player 0 can repeat this pattern forever. It follows that player 0 meets her obligation and that the value of $(s_0, [\bigcirc X]_{\geq 0.5})$ is 1. We conclude that $\text{val}_0(s_0, \nu X.p \wedge [\bigcirc X]_{\geq 0.5}) = 1$. Thus, the formula holds over this structure. Intuitively, there is a location where p holds and for at least $\frac{1}{2}$ of its successors the same property holds again.

The following theorem shows that these games capture the semantics of μ^p -calculus.

► **Theorem 6.** For every Markov chain M , every location s , and every formula φ we have $\llbracket \varphi \rrbracket_M^p(s) = \text{val}_0(s, \varphi)$, where $\text{val}_0(s, \varphi)$ is the value of configuration (s, φ) in game $G_{M,\varphi}$.

► **Corollary 7.** *Given a finite Markov chain M and a formula φ we can decide whether $\llbracket \varphi \rrbracket_M^\rho = 1$ in $\text{NP} \cap \text{co-NP}$.*

Proof. From Theorem 4 we can determine whether the value of configuration (s, φ) in $G_{M, \varphi}$ is at least one in $\text{NP} \cap \text{co-NP}$. The size of $G_{M, \varphi}$ is polynomial in the size of M and in the size of φ . The result follows. ◀

We note that the game captures also the semantics of quantitative subformulas. It follows that for a quantitative subformula ψ we can decide whether $\llbracket \psi \rrbracket_M^\rho(s) > p$ in $\text{NP} \cap \text{co-NP}$ and compute it in exponential time.

5 Hardness of Model Checking

As we have shown in Section 3, there is a simple translation from the μ -calculus to our logic. The exact complexity of model checking the μ -calculus is a long standing open problem. It is well-known that its complexity lies in $\text{UP} \cap \text{co-UP}$ [11] and is equivalent to the complexity of solving parity games [7]. However, the complexity arises from the alternation of fixpoint operators. Here, we show that in our logic already the fraction that uses only the least fixpoint (and only one fixpoint) is as hard as some of the “hard” problems known to be in $\text{NP} \cap \text{co-NP}$ but not known to be in P.

5.1 Two-player Stochastic Reachability Games

A two-player stochastic reachability game is $G = (V, (V_0, V_1, V_p), E, \kappa, T)$, where V , V_0 , V_1 , V_p , E , and κ are just like in parity obligation games and $T \subseteq V$ is a set of target configurations. A strategy for player 0 is $\sigma : V_0 \rightarrow V$ such that for every $v \in V_0$ we have $(v, \sigma(v)) \in E$. A strategy for player 1 is defined similarly. We intentionally consider only deterministic memoryless strategies². Given strategies σ and π for players 0 and 1, respectively, the Markov chain $G_{\sigma, \pi}$ is the result of fixing the choices of the players according to their strategies. For a configuration $v \notin T$, let $\Pi_v = \{v\} \cdot V^* \cdot T \cdot V^\omega$ be the set of paths that start in v and visit T . Then, the value of a configuration $v \in V \setminus T$ for player 0 is $\text{val}_0(v) = \sup_\sigma \inf_\pi \text{measure}_{G_{\sigma, \pi}}(\Pi_v)$.

► **Theorem 8.** [6, 11, 17] *For every configuration $v \in V \setminus T$ deciding if $\text{val}_0(v) > p$ for some $p \in [0, 1]$ is in $\text{NP} \cap \text{co-NP}$. The decision problem of whether a configuration in a 2-player parity/mean-payoff/discounted is winning for player 0 can be reduced to deciding $\text{val}_0(v) > p$.*

5.2 Encoding Games as Model Checking

Consider a two-player stochastic reachability game $G = (V, (V_0, V_1, V_p), E, \kappa, T)$, a configuration $v \in V \setminus T$ and a value $p \in [0, 1]$. We show how to construct a Markov chain M_G and a formula φ_R such that $\llbracket \varphi_R \rrbracket_{M_G}^\rho(s_0) = 1$ iff $\text{val}_0(v) > p$, where s_0 is the initial state of M_G . Let $M_G = \langle K, P \rangle$ be a Markov chain, where $K = \langle V, E, L, v \rangle$, and $P(v, v')$ is $\kappa(v, v')$ if $v \in V_p$ and $P(v, v') = \frac{1}{|E(v)|}$ otherwise³. The labeling L uses four propositions: p_0, p_1 , and p_p

² It is well known that in two-player stochastic reachability games there are optimal deterministic memoryless strategies for both players [6].

³ Or indeed, every distribution that associates non-zero probability with exactly the successors of v .

marking configurations of player 0, player 1, and stochastic, and p_g marking configurations in T as the goal.

Let $\psi_R = p_g \vee ((p_p \rightarrow \bigcirc X) \wedge (p_0 \rightarrow \diamond X) \wedge (p_1 \rightarrow \square X))$. Then $\varphi_R = [\mu X. \psi_R]_{>q}$.

► **Lemma 9.** $\llbracket \varphi_R \rrbracket_{M_G}^\rho(v) = 1$ iff $\text{val}_0(v) > q$.

► **Corollary 10.** *Model checking alternation free μ^p -calculus formulas is as hard as solving parity/mean-payoff/discounted games.*

We note that this result relies on the usage of the existential and universal next operators. Indeed, the proof relies on our ability to “keep” the value of existential and universal configurations in the original game in the formula. We do not know whether it is possible to prove a similar result for a calculus without the existential and universal next operators. We suspect that these next operators increase the expressive power of the logic. We also do not know if by removing these two operators the “normal” complexity hierarchy of the μ -calculus that relies on alternation depth is introduced. We note that parity obligation games can clearly encode the reachability of stochastic games. Thus, showing that the μ^p -calculus without existential and universal next operators enjoys the same hierarchy would require other techniques for model checking this calculus. A hardness result that does not use the existential and universal next operators is by encoding the μ -calculus in μ^p -calculus, as we do in Subsection 3.1. This hardness result does rely on fixpoint alternation.

We note that a similar encoding can represent the value of an obligation game (with finitely many different obligation values) as the result of model checking a μ^p -calculus formula over a Markov chain. As before, the structure of the game is encoded into the Markov chain. The encoding is more involved as we have to include propositions that will identify the exact obligations of configurations. Using these additional propositions the correct probabilistic quantification can be included in the formula. The structure of the formula is very similar to the classical encoding of the solution of parity games as μ -calculus model checking. That is, a prefix with fixpoints binding the variables according to the parity condition followed by a body that includes the association of configurations with player 0, player 1, or probabilistic (as above) with the inclusion of probabilistic quantification as well. We leave further details of this construction as future work.

6 μ -PCTL

We now introduce a fragment of μ^p -calculus that is expressive enough for encoding PCTL and whose model checking is exponential only w.r.t. alternations of quantifiers. Thus, for formulas with a bounded number of fixpoint alternations the model checking of this fragment is polynomial. We believe that this logic may serve as a basis for defining other useful extensions of PCTL.

Let AP be a set $\{p_0, p_1, \dots\}$ of atomic propositions and let $\mathcal{V} = \{X_0, X_1, X_2, \dots\}$ be an enumerable set of variables; the sets Φ and Ψ of location and path formulas, respectively, are mutually recursively defined as follows:

$$\begin{aligned} J &::= \{>, \geq\} \times [0, 1] \\ \Phi &::= p_i \mid \neg p_i \mid X_i \mid \Phi_1 \vee \Phi_2 \mid \Phi_1 \wedge \Phi_2 \mid [\Psi]_J \mid \nu X_i. \Phi \mid \mu X_i. \Phi \\ \Psi &::= X\Phi \mid \Phi \mathcal{U} \Phi \mid \Phi \mathcal{W} \Phi \end{aligned}$$

We assume that in every formula there is no repetition of bound variables; it is straightforward to see that every formula can be rewritten to satisfy this requirement. In general,

we are interested in formulas in which all variables are bound. The definition of alternation depth is as before.

The semantics of this logic can be straightforwardly obtained from the semantics for μ^p -calculus given in Section 3, taking into account the fixpoint semantics of path operators; and similarly for its game semantics. That is, we replace $X\psi$ by $\bigcirc\psi$, $\psi_1 \mathcal{U} \psi_2$ by $\mu X.\psi_2 \vee (\psi_1 \wedge \bigcirc X)$, and $\psi_1 \mathcal{W} \psi_2$ by $\nu X.\psi_2 \vee (\psi_1 \wedge \bigcirc X)$.

Before presenting the model-checking algorithm we introduce some further notations. We use a collection of (global) set variables $S_i \in 2^S$, where each variable S_i represents the valuation of a variable X_i appearing in the formula. Let c_0, c_1, \dots be a set of fresh propositions, and we denote by $M[c_i \leftarrow S_i]$ the structure over $AP \cup \{c_1, \dots, c_n\}$ obtained from M by setting $L(c_i) = S_i$. For the formula φ , let $\varphi[X_j \leftarrow c_j]$ be the formula obtained from φ , by replacing every reference to X_j by c_j .

We are now ready to present the model-checking algorithm for μ -PCTL. Our procedure, called *eval*, is presented as Algorithm 1. The procedure takes a Markov chain $M = \langle S, R, L, s_0 \rangle$ and a μ -PCTL formula φ and returns the set of states satisfying φ . We assume that variables S_i , where X_i is bound by a least fixpoint, are initialized to the empty set; and variables S_i , where X_i is bound by a greatest fixpoint, are initialized to the set of all states S . This algorithm uses the well-known way of calculating fixed points by using the Knaster-Tarski theorem and it assumes a polynomial model checking algorithm for PCTL (denoted *evalPCTL*).

The algorithm is similar to that proposed in [7] to model check standard μ -calculus, fixed points are calculated in the standard way, new constants are used for reducing subformulas to PCTL formulas, and we only reset the values of variables when the nesting of two different fixed points are found, otherwise previous calculation of fixed points are employed; to do so, we use some auxiliary functions: *Parent*(φ_i) returns the fixpoint σX_j surrounding φ_i such that X_j appears free in that formula, and *OpenSub*(φ_i) returns the set subformulas of φ_i that are bound by the same fixpoint operators and in which X_i is free. Notice that formulas of the form $[\Psi]_J$ are handled by *evalPCTL* after replacing fixpoint variables by propositions.

► **Theorem 11.** *For a formula φ , $s \in \text{eval}(M, \varphi)$ iff $\llbracket \varphi \rrbracket_M^{\rho}(s) = 1$.*

We note that this procedure is exponential only w.r.t. alternation depth. Thus, if the alternation depth is fixed the procedure is polynomial.⁴

► **Theorem 12.** *Procedure *eval* runs in time $O(|M|^k \cdot |\phi|^{\frac{3}{2}ad(\phi)+1})$, where the constant k depends on the model checker used for PCTL formulas.*

Furthermore, we prove that this fragment of μ^p -calculus is strictly more expressive than PCTL.

► **Theorem 13.** *μ -PCTL is strictly more expressive than PCTL.*

Proof. Consider the formula $\nu Y.p \wedge [XY]_{>0}$, one can see that it is equivalent to the CTL formula EGp. Theorem 14.45 in [1] shows that there is no qualitative PCTL formula that is

⁴ We also note that if a similar approach would be applied to finite obligation parity games the result would be an exponential number of calls to an NP \cap co-NP algorithm. Indeed, the search for the sets of obligations that can be used to satisfy other obligations can follow the same search pattern by using maximal and minimal fixpoints. However, checking that each obligation is met, which corresponds to the PCTL model checking in *eval*, would be a solution of a finite turn-based stochastic parity-reachability game, which is in NP \cap co-NP.

Input: A Markov Chain M and a formula ϕ

Output: Set of states satisfying ϕ

```

1 switch the form of  $\phi$  do
2   case  $\phi$  is a PCTL formula return  $evalPCTL(M, \phi)$  ;
3   case  $\phi = p_i$  return  $L(p_i)$  ;
4   case  $\phi = c_i$  return  $S_i$  ;
5   case  $\phi = \phi_1 \wedge \phi_2$  return  $eval(M, \phi_1) \cap eval(M, \phi_2)$  ;
6   case  $\phi = \phi_1 \vee \phi_2$  return  $eval(M, \phi_1) \cup eval(M, \phi_2)$  ;
7   case  $\phi = \nu X_i. \phi'$ 
8     if  $Parent(\phi) = \mu X_j$  then
9       forall the  $\nu X_k \in OpenSub(\phi)$  do  $S_k = S$ ;
10    end
11    repeat
12       $S'_i = S_i$ ;
13       $S_i = eval(M[c_i \leftarrow S_i], \phi'[X_i \leftarrow c_i])$ ;
14    until  $S_i = S'_i$ ;
15    return  $S_i$ ;
16  end
17  case  $\phi = \mu X_i. \phi'$ 
18    if  $Parent(\phi) = \nu X_j$  then
19      forall the  $\mu X_k \in OpenSub(\phi)$  do  $S_k = \emptyset$ ;
20    end
21    repeat
22       $S'_i = S_i$ ;
23       $S_i = eval(M[c_i \leftarrow S_i], \phi'[X_i \leftarrow c_i])$ ;
24    until  $S_i = S'_i$ ;
25    return  $S_i$ ;
26  end
27 endsw

```

Algorithm 1: Recursive Procedure $eval$

equivalent to it. It is possible to extend their proof to cover also quantitative probabilistic quantification of PCTL. Thus, formula $\nu Y.p \wedge [XY]_{>0}$ cannot be expressed in PCTL. ◀

To summarize, μ -PCTL formulas with bounded alternation depth admit a polynomial model-checking procedure, μ -PCTL is more expressive than PCTL. Finally, note that μ -PCTL may be particularly useful to capture properties about repeating patterns of executions with measure 0. For instance, the formula $\nu X.p \wedge [OX]_{\geq 0.5}$ allows one to separate the model of Figure 1 from the model obtained from it by removing the loop in state s_0 . We leave as further work a careful investigation of this logic.

7 Related Work

Several attempts have been made to extend the features of Kozen's μ -calculus to the realm of logics characterizing Markov chains. Huth and Kwiatkowska and, independently, McIver and Morgan considered qualitative μ -calculi over Markov chains [9, 13]. Their definition replaced union by maximum (*max*) and intersection by minimum (*min*) defining a basic probabilistic calculus. The semantics of a formula was changed from a Boolean value of

$\{0, 1\}$ to a real value in $[0, 1]$. Their logic, however, does not capture popular probabilistic temporal logics such as PCTL [14]. In particular, these logics do not include the probabilistic quantification central to the notion of PCTL and also did not allow to capture a single probabilistic quantification surrounding an LTL formula. Cleaveland et al. extend the calculus of Huth and Kwiatkowska by adding probabilistic quantification and allowing a finite number of nesting of probabilistic quantifications [5]. In particular, they do not allow interaction between fixpoint operators and probabilistic quantification. This restriction makes reasoning about the logic simple by repeating a finite number of times the evaluation of the simpler logic of Huth and Kwiatkowska. The resulting logic allows to express PCTL (and PCTL^{*}). At the same time, it limits the expressive power of the logic: it cannot express the μ -calculus over the embedded Kripke structure, or even the CTL formula EGp , which we saw can be expressed in μ -PCTL (and consequently in μ^p -calculus). Both types of μ -calculus are subsets of μ^p -calculus.

Recently, Mio and Simpson [15] suggested an extended quantitative μ -calculus that includes various options for join and meet. They include the *max* and *min* suggested previously, but also include some standard operators in Łukasiewicz logics such as \oplus and \odot , that have similar pleasing mathematical properties and are generalizations of Boolean disjunction and conjunction. In order to capture probabilistic quantification they also include explicit multiplication by a rational constant. The resulting logic enjoys some of the mathematical properties of the μ -calculus, allowing one to express PCTL probabilistic quantification, for instance. Using the operators \oplus and \odot as atomic operators results in several shortcomings. The best algorithms for model checking for this logic are either non-elementary or (by reduction to first-order theory of the reals) triple exponential. Probabilistic quantification is expressed as a combination of a fixpoint of one of the new operators along with multiplication by constants. Another advantage of our logic over that of Mio and Simpson is that we can syntactically recognize formulas that are qualitative. Furthermore, not directly relevant for the μ -calculus, the game semantics associated with it includes a construct called “independent product” and it is not known whether games with this feature are determined for general Borel winning conditions. We note that Mio and Simpson define their logic on Markov decision processes (MDPs) and not over Markov chains. All the results we presented above generalize to MDPs. There are no additional technical difficulties in carrying the proofs over. We have chosen to present our work on MDPs to simplify presentation and to be consistent with the large body of work on model checking Markov chains and PCTL that we are familiar with.

8 Final Remarks

We have presented a probabilistic μ -calculus that uses probabilistic quantification as an atomic operation. Our main goal is to provide a unifying formalism into which the probabilistic temporal logics used in model checking can be encoded. We have shown that PCTL and PCTL^{*} can be captured in this calculus, and we note that similar results can be obtained for other probabilistic logics such as probabilistic linear temporal logic. We have proved some interesting results for this logic; in particular, its model checking problem is in $NP \cap co-NP$ and it admits a simple game semantics. Furthermore, we presented a simple fragment of this logic which we believe may be important for expressing properties that are not expressible in other probabilistic logics, in particular, those predicating about executions with measure 0, we leave as a further work a deeper investigation of this fragment.

The discrete μ -calculus is intrinsically linked to alternating parity tree automata. We

believe that a similar connection exists between μ^p -calculus and p-automata [10]. We leave the consideration of this connection as future work.

References

- 1 C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
- 2 A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *15th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 1995.
- 3 K. Chatterjee and N. Piterman. Obligation blackwell games and p-automata. Technical report, arXiv:1206.5174, 2012.
- 4 F. Ciesinski and C. Baier. LiQuor: A tool for qualitative and quantitative linear time analysis of reactive systems. In *QEST*, pages 131–132. IEEE Computer Society, 2006.
- 5 R. Cleaveland, S. Purushothaman Iyer, and M. Narasimha. Probabilistic temporal logics via the modal mu-calculus. *Theor. Comput. Sci.*, 342(2-3):316–350, 2005.
- 6 A. Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992.
- 7 E.A. Emerson and C. Lei. Efficient model checking in fragments of the μ -calculus. In *LICS*. IEEE Computer Society, 1986.
- 8 A. Hinton, M.Z. Kwiatkowska, G. Norman, and D. Parker. PRISM: a tool for automatic verification of probabilistic systems. In *TACAS*, volume 3920 of *Lecture Notes in Computer Science*. Springer-Verlag, 2006.
- 9 M. Huth and M.Z. Kwiatkowska. Quantitative analysis and model checking. In *12th IEEE Symposium on Logic in Computer Science*, pages 111–122. IEEE Computer Society, 1997.
- 10 M. Huth, N. Piterman, and D. Wagner. p-automata: New foundations for discrete-time probabilistic verification. *Performance Evaluation*, 69(7–8):356–378, 2012.
- 11 M. Jurdzinski. Deciding the winner in parity games is in $UP \cap co-UP$. *Inf. Process. Lett.*, 68(3):119–124, 1998.
- 12 D. Kozen. Results on the propositional μ -calculus. In *Automata, Languages and Programming*, volume 140 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1982.
- 13 A. McIver and C. Morgan. Results on the quantitative μ -calculus $qM\mu$. *ACM Trans. Comput. Log.*, 8(1), 2007.
- 14 M. Mio. *Game Semantics for Probabilistic μ -Calculus*. PhD thesis, University of Edinburgh, 2012.
- 15 M. Mio and A. Simpson. Łukasiewicz μ -calculus. In *FICS*, 2013.
- 16 K. Scheider. *Verification of Reactive Systems: Formal Methods and Algorithms*. Springer, 2004.
- 17 U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Theor. Comput. Sci.*, 158(1&2):343–359, 1996.

Tribes Is Hard in the Message Passing Model*

Arkadev Chattopadhyay¹ and Sagnik Mukhopadhyay²

1 Tata Institute of Fundamental Research, Mumbai
India

arkadev.c@tifr.res.in

2 Tata Institute of Fundamental Research, Mumbai
India

sagnik@tifr.res.in

Abstract

We consider the point-to-point message passing model of communication in which there are k processors with individual private inputs, each n -bit long. Each processor is located at the node of an underlying undirected graph and has access to private random coins. An edge of the graph is a private channel of communication between its endpoints. The processors have to compute a given function of all their inputs by communicating along these channels. While this model has been widely used in distributed computing, strong lower bounds on the amount of communication needed to compute simple functions have just begun to appear.

In this work, we prove a tight lower bound of $\Omega(kn)$ on the communication needed for computing the Tribes function, when the underlying graph is a star of $k + 1$ nodes that has k leaves with inputs and a center with no input. A lower bound on this topology easily implies comparable bounds for others. Our lower bounds are obtained by building upon the recent information theoretic techniques of Braverman et al. ([4], FOCS'13) and combining it with the earlier work of Jayram, Kumar and Sivakumar ([10], STOC'03). This approach yields information complexity bounds that are of independent interest.

1998 ACM Subject Classification F.1.1 Models of computation, E.4.8 Coding and Information theory, F.2.2. Analysis of algorithms and Problem Complexity

Keywords and phrases communication complexity, Tribes, information complexity, direct-sum

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.224

1 Introduction

The classical model of 2-party communication was introduced in the seminal work of Yao[18], motivated by problems of distributed computing. This model has proved to be of fundamental importance (see the book by Kushilevitz and Nisan [13]) and forms the core of the vibrant subject of communication complexity. It is fair to say that the wide applicability of this model to different areas of computer science cannot be over-emphasized.

However, a commonly encountered situation in distributed computing is one where there are multiple processors, each holding a private input, that are connected by an underlying communication graph. An edge of the graph corresponds to a private channel of communication between the endpoints. There are k processors located on distinct nodes of the graph that want to compute a function of their joint inputs. In such a networked scenario, a very natural question is to understand how much total communication is needed

* A. Chattopadhyay is partially supported by a Ramanujan Fellowship of the DST and S. Mukhopadhyay is supported by a TCS Fellowship.



to get the function computed. The classical 2-party model is just a special case where the graph is an edge connecting two processors.

Among others, this model has also been called the Number-in-hand multiparty point-to-point message passing model of communication. Apart from distributed computing, this model is used in secure multiparty computation. The study of the communication cost in the model was most likely introduced by Dolev and Feder [6] and further worked on by Duris and Rolim [8]. These early works focused on deterministic communication. There has been renewed interest in the model because it arguably better captures many of today's networks that is studied in various distributed models: models for map-reduce [11, 9], massively parallel model for computing conjunctive queries [3, 12], distributed models of learning [1] and in core distributed computing [7]. However, there were no known systematic techniques of proving lower bounds on the cost of randomized communication protocols that exploited the *non-broadcast* nature of the *private channels* of communication in the model. Recently, there has been a flurry of work developing new techniques for proving lower bounds on communication. Phillips, Verbin and Zhang [14] introduced the method of symmetrization to prove strong bounds for a variety of functions. Their technique was further developed in the works of Woodruff and Zhang [15, 16, 17].

All these works considered the co-ordinator model, a special case, that was introduced in the early work of [6]. In the co-ordinator model, the underlying graph has the star topology with $k + 1$ nodes. There are k leaves, each holding an n -bit input. Each of the k leaf-nodes is connected to the center of the star. The node at the center has no input and is called the co-ordinator. The following two simple observations about the model will be relevant for this work: every function can be trivially computed using $O(nk)$ bits of communication by having each of the k players send their inputs to the co-ordinator who then outputs the answer. It is also easily observed that the co-ordinator model can simulate a communication protocol on an arbitrary topology having k nodes with at most a $\log k$ factor blow-up in the total communication cost.

A key lesson learnt from our experience with the classical 2-party model is that an excellent indicator of our understanding of a model is our ability to prove lower bounds for the widely known Set-Disjointness problem in the model. Indeed, as surveyed in [5], several new and fundamental lower bound techniques have emerged from efforts to prove lower bounds for this function. Further, the lower bound for Set-Disjointness, is what drives many of the applications of communication complexity to other domains. While the symmetrization technique of Phillips et.al and its refinements by Woodruff and Zhang proved several lower bounds, no strong lower bounds for Set-Disjointness were known until recently in the k -processor co-ordinator model. In this setting, the relevant definition of Set-Disjointness is the natural generalization of its 2-party definition: view the n -bit inputs of the k processors as a $k \times n$ Boolean matrix where the i th row corresponds to the Processor i 's input. The Set-Disjointness function outputs 1 iff there exists a column of this matrix that has no zeroes.

In an important development, Braverman et al. [4] proved a tight $\Omega(kn)$ lower bound for Set-Disjointness in the co-ordinator model. Their approach is to build up new information complexity tools for this model that is a significant generalization of the 2-party technique of Bar-Yossef et al. [2]. In this work, we further develop this information complexity method for the co-ordinator model by considering another natural and important function, known as Tribes $_{m,\ell}$. In this function, the n -bit input to each processor is grouped into m blocks, each of length ℓ . Thus, the overall $k \times n$ input matrix splits up into m sub-matrices A_1, \dots, A_m , each of dimension $k \times \ell$. Tribes outputs 1 iff the Set-Disjointness function outputs 1 on each

sub-matrix A_i . This obviously imparts a direct-sum flavor to the problem of determining the complexity of the Tribes function in the following sense: a naive protocol will solve Tribes by simultaneously running an optimal protocol for Set-Disjointness on each of the m instances A_1, \dots, A_m . Is this strategy optimal?

This question was answered in the affirmative for the 2-party model by Jayram, Kumar and Sivakumar [10] when they proved an $\Omega(n)$ lower bound on the randomized communication complexity of the Tribes function. Their work delicately extended the information theoretic tools of Bar-Yossef et.al [2]. Interestingly, it also exhibited the power of the information complexity approach. There was no other known technique to establish a tight lower bound on the Tribes function¹.

In this work, we show that the naive strategy for solving Tribes is optimal also in the co-ordinator model:

► **Theorem 1.** *In the k -processor co-ordinator model, every bounded error randomized protocol solving the Tribes $_{m,n}$ function, has communication cost $\Omega(m\ell k)$, for every $k \geq 2$.*

We prove this by extending and simplifying the information complexity approach of [4] and the earlier work of [10]. It is worth noting that our bounds in Theorem 1 hold for all values of k . In particular, this also yields a lower bound for Set-Disjointness for all values of k . The earlier bound of Braverman et al. only worked if $k = \Omega(\log n)$.

2 Overview & Comparison with Previous Work

We first provide a quick overview of our techniques and contributions. We follow this up with a more detailed description, elaborating on the main steps of the argument.

Brief Summary: Recall that the Tribes $_{m,n}$ function can be written as an m -fold AND of Disj $_n$ instances. One possible way to show that Tribes $_{m,n}$ is hard in message-passing model is to show that any protocol evaluating Tribes $_{m,n}$ must evaluate all the Disj $_n$ instances. This suffices to argue that Tribes $_{m,n}$ is m times as hard as Disj $_n$. By now it is well known that information complexity provides a convenient framework to realize such direct sum arguments. In order to do so, one needs to define a distribution on inputs that is entirely supported on the ones of the m Set-Disjointness instances of Tribes. This was the general strategy of Jayram et al. [10] in the 2-party context. However, the first problem one encounters is to define an appropriate hard distribution and a right notion of information cost such that Disjointness has high information cost of $\Omega(k\ell)$ under that distribution *in the co-ordinator model*. This turns out to be a delicate and involved step. Various natural information costs do not work as observed by Phillips et al. [14]. Here, we are helped by the work of Braverman et al. [4]. They come up with an appropriate distribution τ and an information cost measure IC^0 . However, we face some problems in using them. The first is that τ happens to be (almost) entirely supported on the zeroes of Set-Disjointness. Taking ideas from [10], we modify τ to get a distribution μ supported exclusively on the ones of Set-Disjointness. Roughly speaking, to sample from μ , we first sample from τ and then pick a random column of the sampled input and force it to all ones. Intuitively, the idea is that the all ones column is being well hidden at a random spot. The intuition is, if τ was hard, μ should also remain hard. It turns

¹ This is not surprising. Two other successful techniques, the discrepancy and the corruption method, both yield lower bounds on the non-deterministic complexity. On the other hand, Tribes and its complement, on n -bit inputs, both have only \sqrt{n} non-deterministic complexity.

out that we prove the hardness of μ directly from scratch. To do so we appropriately modify the information cost measure IC^0 to IC so that it yields high information complexity under μ . Here, we use an idea of [10].

However, proving that IC is high for protocols when inputs are sampled according to μ raises new technical challenges. The first challenge is to prove a direct sum result on the information complexity of protocols as measured by IC . We do not know how to do that. Here we borrow ideas from [10] to introduce a new information measure, $\text{PIC}(f)$ which is a lower bound on $\text{IC}(f)$ and will be explained in relevant section. We show that $\text{PIC}(\text{Disj}_n)$ is at least $\Omega(\ell \cdot \text{PIC}(\text{Disj}_2))$. Implementing this step is a novelty of this work. The final challenge is to prove that $\text{IC}(\text{Disj}_2)$ is $\Omega(k)$. We again do that by first simplifying some of the lemmas of [4] and extending them using some ideas from the work of [10].

More Detailed Account: Among the many possible ways to define information cost of a protocol, the definition we work with stems from the inherent structure of the communication model. As evident from the previous discussion, in the model of communication we are interested in, the co-ordinator can see the whole transcript of the protocol but cannot see the inputs. On the other hand, the processors can only see a local view of the transcript - the message that is passed to them and the message they send - along with their respective inputs. From the point of view of the co-ordinator, who has no input, the information revealed by the transcript about the input can be expressed by $\mathbb{I}[X : \Pi(X)]$. This is small for the protocol where the co-ordinator goes around probing each player on each coordinate to see whether any player has 0 in it and gives up once she finds such a player. (We call it Protocol A). It is not hard to see that the information cost can only be as high as $O(n \log k)$ for protocol A. A relevant information cost measure from the point of view of processor i is $\mathbb{I}[X^{-i} : \Pi(X) | X^i]$ which measures how much information processor i learns about other inputs from the transcript. It turns out that this information cost is also very small for the protocol where all the processors send their respective inputs to the co-ordinator (We call this protocol as protocol B). Here $\mathbb{I}[X^{-i} : \Pi(X) | X^i]$ is 0 for all i . What is worth noticing is that in both protocols, if we consider the sum of the two information costs, i.e., $\mathbb{I}[X : \Pi(X)] + \sum_i \mathbb{I}[X^{-i} : \Pi(X) | X^i]$, it is $\Omega(nk)$ which is the kind of bound we are aiming for.

This cost trade-off was first observed in [14] but they were unable to prove a lower bound for Disj_n in this model of communication. Braverman et al [4] solved this problem by coming up with the following notion of information complexity. Let $(\mathbf{X}, \mathbf{M}, \mathbf{Z})$ be distributed jointly according to some distribution τ . The information cost of a protocol Π with respect to τ is defined as, $\text{IC}_\tau^0(\Pi) = \sum_{i \in [k]} \left[\mathbb{I}_{\tau}[\mathbf{X}^i : \Pi^i(\mathbf{X}) | \mathbf{M}, \mathbf{Z}] + \mathbb{I}_{\tau}[\mathbf{M} : \Pi^i(\mathbf{X}) | \mathbf{X}^i, \mathbf{Z}] \right]$.

Conditioning on the auxiliary random variables \mathbf{M} and \mathbf{Z} serves the following purpose: Even though the distribution τ is a non-product distribution, it can be thought of as a convex combination of product distributions, one for each specific values of \mathbf{M} and \mathbf{Z} . It is well-known by now that such convex combination facilitates proving direct-sum like result.

The desired properties of the distribution τ are as follows. First, the distribution should have enough entropy to make it hard for the players to encode their inputs cheaply and send it across to the co-ordinator. Such an encoding is attempted in protocol B. Second, the distribution should be supported on inputs which have only a few 0's in each column of Disj_n . This makes sure that the co-ordinator has to probe $\Omega(k)$ processors in each column before he finds a 0 in that column. This attempt of probing was undertaken by the co-ordinator in protocol A. The first property can be individually satisfied by setting each processor's input to be 0 or 1 with equal probability in each column. The second property can also be

individually satisfied by taking a random processor for each column and giving it a 0 and giving 1 to rest of the processors as their inputs. Let \mathbf{Z}_j denote the processor whose bit was fixed to 0 in column j . The hard distribution for Disj_n is a convex combination of these two distributions. The way it is done is by setting a Bernoulli random variable \mathbf{M}_j for each of the column j which acts as a switch, i.e., if $\mathbf{M}_j = 0$ the input to the column j is sampled from the first distribution, otherwise it is sampled from the second distribution. \mathbf{M}_j takes value 0 with probability $2/3$. We define $\mathbf{M} = \langle \mathbf{M}_1, \dots, \mathbf{M}_\ell \rangle$ and $\mathbf{Z} = \langle \mathbf{Z}_1, \dots, \mathbf{Z}_\ell \rangle$.

At this point it is interesting to go back to the definition of IC^0 and try to see the implication of each term in the definition. For the coordinator, $\sum_i \mathbb{I}[\mathbf{X}^i : \Pi^i(\mathbf{X}) \mid \mathbf{M}, \mathbf{Z}]$ represents the amount of information revealed about the inputs of the processors by the transcript. For convenience, we can assume that \mathbf{M} is with co-ordinator. We can do this without loss of generality as the co-ordinator can sample $O(\log k)$ inputs from column j and conclude the value of \mathbf{M}_j from it, for any j . This amount of communication is okay for us as we are trying to show a lower bound of $\Omega(nk)$. However note that we cannot assume that the processors have the knowledge of \mathbf{M} . Had that been the situation, the processors would have employed protocol A or protocol B in column j depending on the value of the \mathbf{M}_j . The value of $\mathbb{I}[\mathbf{X}^i : \Pi^i(\mathbf{X}) \mid \mathbf{M}, \mathbf{Z}]$, in this protocol, would have been small. So we need to make sure that we charge the processors for their effort to know the value of \mathbf{M} . This is taken care by the second term in the definition of IC^0 i.e., $\mathbb{I}[\mathbf{M} : \Pi^i(\mathbf{X}) \mid \mathbf{X}^i, \mathbf{Z}]$. Braverman et al. [4] used this notion of information complexity to achieve the $\Omega(\ell k)$ lower bound for the information cost of Disj_n with respect to the hard distribution.

As mentioned before, we, however, need the hard distribution ζ for $\text{Tribes}_{m,n}$ to be entirely supported on 1s of Disj_n . But the distribution τ described above is supported on 0s of Disj_n . Here we borrow ideas from [10] and design a distribution μ by selecting a random column for the Disj_n instances and planting an all 1 input in it. We denote the random co-ordinate by \mathbf{W} . It is easy to verify that μ is a distribution supported in 1s of Disj_n . We set the hard distribution for $\text{Tribes}_{m,n}$ to be an m -fold product distribution $\zeta = \mu^m$ denoted by the random variables $\langle \bar{\mathbf{X}}, \bar{\mathbf{M}}, \bar{\mathbf{Z}}, \bar{\mathbf{W}} \rangle$. It is to be noted that a correct protocol should work well for all inputs, not necessarily for the inputs coming from the distribution ζ . This property will be crucially used in later part of the proof. The modification of the input distribution from τ to μ and subsequently to ζ calls for changing the definition of the information complexity to suit our purpose. We define information complexity as follows which we will use in this paper.

► **Definition 2.** Let $(\bar{\mathbf{X}}, \bar{\mathbf{M}}, \bar{\mathbf{Z}}, \bar{\mathbf{W}})$ be distributed jointly according to ζ . The information cost of a protocol Π with k processors in NIH point-to-point coordinator model with respect to ζ is defined as,

$$\text{IC}_\zeta(\Pi) = \sum_{i \in [k]} \left[\mathbb{I}[\bar{\mathbf{X}}^i : \Pi^i(\bar{\mathbf{X}}) \mid \bar{\mathbf{M}}, \bar{\mathbf{Z}}, \bar{\mathbf{W}}] + \mathbb{I}[\bar{\mathbf{M}} : \Pi^i(\bar{\mathbf{X}}) \mid \bar{\mathbf{X}}^i, \bar{\mathbf{Z}}, \bar{\mathbf{W}}] \right]. \quad (1)$$

For a function $f : \mathbf{X} \rightarrow \mathcal{R}$, the information complexity of the function is defined as, $\text{IC}_{\zeta, \delta}(f) = \inf_{\Pi} \text{IC}_\mu(\Pi)$, where the infimum is taken over all δ -error protocol Π for f .

By doing this, we are able to bound the information complexity of $\text{Tribes}_{m,n}$ as m -times that of Disj_n . Although non-trivial, this step can be accomplished by exploiting the proof techniques used in [4]. The next step is to bound the information complexity of Disj_n , which turns out to be difficult for two reasons. First, the distribution μ is no more a 0 distribution for Disj_n . We get around this by defining a new information complexity measure, - which

we call as partial information complexity - to show that the partial information complexity of Disj_n on distribution μ is at least $(\ell - 1)$ -times that of Disj_2 . This is one of the main technical contributions of our paper. See Section 4.1 for details. The second hurdle we face is bounding the information complexity of Disj_2 . Here we combine ideas from [10, 4] to conclude that the partial information complexity of Disj_2 is $\Omega(k)$. This is the second main technical contribution of this paper, which is explained in Section 4.2. Finally we give a simple argument in Section 5 to show that $\text{IC}_\zeta(\Pi)$ lower bounds the communication cost of Π where Π is any correct protocol for $\text{Tribes}_{n,n}$.

3 Preliminaries

Communication complexity. In this work, we are mainly interested in multiparty communication *number-in-hand* model. In this model of computation, the input is distributed between k players P_1, \dots, P_k who jointly wish to compute a function f on the combined input by communication with each other.

We work with randomized protocol where the players have access to private coins. (Though it might seem like that the public coin protocol can yield better upper bound, it can be noted that all the proofs can be modified to give the same result for public coin model.) The standard notion of private coin randomized communication complexity is adopted here, where we look at the worst-case communication of the protocol when the protocol is allowed to make only δ error (bounded away from $1/2$) on each input. Here the probability is taken over the private coin tosses of the players. For more details, readers are referred to [13].

Information theory. We will quickly go through the information theoretic definitions and facts we need. For a random variable X taking value in the sample space Ω according to the distribution $p(\cdot)$, the entropy of X , denoted as $\mathcal{H}(X)$, is defined as $\mathcal{H}(X) = \sum_{x \in \Omega} \Pr[X = x] \log \frac{1}{\Pr[X=x]} = \mathbf{E}_x \left[\log \frac{1}{p(x)} \right]$.

For two random variables X and Y , the conditional entropy of X given Y is defined as $\mathcal{H}(X|Y) = \mathbf{E}_{x,y} \left[\log \frac{1}{p(x|y)} \right]$.

Informally, the entropy of a random variable measures the uncertainty associated with it. Conditioning on another random variable, i.e., knowing the value that another random variable takes can only decrease the uncertainty of the former one. This notion is captured in the following fact that $\mathcal{H}(X|Y) \leq \mathcal{H}(X)$ where the equality is achieved when X is independent of Y . Given two random variables X and Y with joint distribution $p(x, y)$ we can talk about how much information one random variable reveals about the other random variable. The mutual information, as it is called, between X and Y is defined as $\mathbb{I}[X : Y] = \mathcal{H}(X) - \mathcal{H}(X|Y)$.

It is to be noted that the mutual information is a symmetric quantity, though it might not be obvious from the definition itself. From the previous discussion, it is easy to see that the mutual information is a non-negative quantity. As before, we can also define conditional mutual information as $\mathbb{I}[X : Y|Z] = \mathcal{H}(X|Z) - \mathcal{H}(X|Y, Z)$.

The following chain rule of mutual information will be crucially used in our proof.

$$\mathbb{I}[X_1, \dots, X_n : Y] = \sum_{i \in [n]} \mathbb{I}[X_i : Y | X_{i-1}, \dots, X_1]. \quad (2)$$

It is to be noted that the chain rule of mutual information will also work when conditioned on random variable Z .

► **Remark.** Consider a permutation $\sigma : [n] \rightarrow [n]$. The following observation will be useful in our proof.

$$\mathbb{I}[X_1, \dots, X_n : Y] = \sum_{i \in [n]} \mathbb{I}[X_{\sigma(i)} : Y | X_{\sigma(i-1)}, \dots, X_{\sigma(1)}]. \quad (3)$$

We will use the following lemma regarding mutual information.

► **Lemma 3.** *Consider random variables A, B, C and D . If A is independent of B given D then,*

$$\mathbb{I}[A : B, C | D] = \mathbb{I}[A : C | B, D], \quad (4)$$

and

$$\mathbb{I}[A : C | B, D] \geq \mathbb{I}[A : C | D]. \quad (5)$$

4 Lower Bound for Tribes _{m,n} in Message Passing Model

Here, in the first subsection, we will show two direct-sum results. In the first step we bound the information complexity of Tribes _{m,n} in terms of that of Disj _{n} . It is to be noted that the proof technique of [2] falls short of proving any lower bound on the information complexity measure we have defined - mainly because of the fact the information complexity measure consists of sum two different mutual information terms for each processor, and it is not clear that one can come up with lower bounds for both the terms simultaneously. This problem has already been attended to in [4] and the proof we present here resembles the proof technique used by them. For completeness we include the proof in this paper. In the second step, we will bound the information complexity of Disj _{n} in terms of Disj₂. This step is more difficult and a straight-forward application of the direct-sum argument of [4] will not work. First we use ideas from [10] to define partial information complexity measure which is more convenient to work with. Then we come up with a novel direct-sum argument for partial information complexity measure.

In Section 4.2, we show that the information complexity of Disj₂ is $\Omega(k)$. We manage to show this by combining ideas from [4, 10].

4.1 Direct Sum

In this section we prove that the information cost of computing Tribes _{m,n} is m times the information cost of computing Disj _{n} . The proof is almost the same proof as in [4] where the authors have used a direct sum theorem to show that the information cost of computing Disj _{n} is ℓ times the information cost of computing k -bit AND _{k} . Before going into details we need the following definitions which we will borrow from [10].

Consider $f : \mathcal{D}^m \rightarrow \mathcal{R}$ can be written as $f(X) = g(h(X_1), \dots, h(X_m))$ where $X = \langle X_1, \dots, X_m \rangle$, $X_i \in \mathcal{D}$ and $h : \mathcal{D} \rightarrow \mathcal{R}$. In other words, f is g -decomposable with primitive h .

► **Definition 4** (Collapsing distribution). We call $X \in \mathcal{D}^m$ be a collapsing input for f if for any $i \in [m]$ and $y \in \mathcal{D}$, we have $f(X(i, y)) = h(y)$. Any distribution ζ supported entirely on collapsing inputs on f is called a collapsing distribution of f .

► **Definition 5** (Projection). Given a distribution ν specified by random variable (D_1, \dots, D_k) and a subset S of $[k]$, we call the projection of ν on $(D_i)_{i \in S}$, denoted as $\nu \downarrow_{(D_i)_{i \in S}}$, the marginal distribution of $(D_i)_{i \in S}$ induced by ν .

The proof is by reduction: we will show that given a protocol Π for $\text{Tribes}_{m,n}$ and a collapsing distribution $\mu = \zeta_\ell^m$, we can construct a protocol Π' for Disj_n such that it computes Disj_n with the same error probability as that of Π and the information complexity of Π is m times that of Disj_n .

► **Theorem 6.** *Let $\mu = \zeta_\ell^m$ be a collapsing distribution for $\text{Tribes}_{m,n}$ partitioned by \mathbf{M}, \mathbf{Z} and \mathbf{W} as described before. Then*

$$\text{IC}_\mu(\text{Tribes}_{m,n}) \geq m \cdot \text{IC}_{\zeta_\ell}(\text{Disj}_n). \quad (6)$$

As mentioned before, the proof of Theorem 6 works out nicely by adapting the proof techniques of [4] and is omitted in this version.

Now our goal is to connect the information cost of Disj_n under ζ_ℓ to information cost of AND_k . So a natural attempt is to prove a theorem like Theorem 6 for reduction from Disj_n to AND_k . Unfortunately this is not possible. Recall that $\text{Disj}_n(X) = \bigvee_{i=1}^\ell \bigwedge_{j=1}^k X_i^j$. Hence for a collapsing distribution each of the AND_k s should evaluate to 0, which is not the case for the distribution ζ_ℓ .

Inspired by [10], we define the following measure of information cost, namely, partial information cost. Let Π be a protocol for Disj_n . The partial information cost of Π is defined as,

$$\text{PIC}(\Pi) = \sum_{i=1}^k (\mathbb{I}[\mathbf{M}_{-\mathbf{W}} : \Pi^i(\mathbf{X}) \mid \mathbf{X}^i, \mathbf{Z}, \mathbf{W}] + \mathbb{I}[\mathbf{X}_{-\mathbf{W}}^i : \Pi^i(\mathbf{X}) \mid \mathbf{M}, \mathbf{Z}, \mathbf{W}]). \quad (7)$$

The random variable $\mathbf{M}_{-\mathbf{W}}$ denotes \mathbf{M} with its \mathbf{W} -th coordinate removed. Similarly, $\mathbf{X}_{-\mathbf{W}}^i$ denotes \mathbf{X}^i with its \mathbf{W} -th coordinate removed. The partial information complexity of Disj_n is the partial information cost of the best protocol computing Disj_n . It is easy to see that the partial information complexity of any function f lower bounds the information complexity of f .

We prove the following theorem.

► **Theorem 7.** *Let ζ_ℓ be the distribution over the inputs of Disj_n partitioned by $\mathbf{M}, \mathbf{Z}, \mathbf{W}$ as described before. Then*

$$\text{PIC}_{\zeta_\ell}(\text{Disj}_n) \geq (\ell - 1) \cdot \text{PIC}_{\zeta_2}(\text{Disj}_2). \quad (8)$$

Here we will show the following reduction analogous to our previous reduction from $\text{Tribes}_{m,n}$ to Disj_n . Given a protocol Π' for Disj_n and distribution ζ_ℓ (as described in Section 2, we will come up with a protocol Π'' for Disj_2 such that the partial information cost of Π'' w.r.t. ζ_ℓ is $1/(\ell - 1)$ times the partial information cost of Π' w.r.t. ζ_2 .

Let us describe the construction of the protocol Π'' . On an input $u = \langle u_1, u_2 \rangle$ for Disj_2 , the processors and the coordinator sample a $k \times \ell$ random matrix $\mathbf{X}(u)$ in the following way.

1. The coordinator samples \mathbf{P} and \mathbf{Q} uniformly at random from $[\ell]$ such that $\mathbf{P} < \mathbf{Q}$.
2. The coordinator samples $\mathbf{Z}_{-\{\mathbf{P}, \mathbf{Q}\}} = (\mathbf{Z}_i)_{i \in [\ell] \setminus \{\mathbf{P}, \mathbf{Q}\}}$, where each $\mathbf{Z}_i \in_R [k]$, and sends it to all the processors.
3. The coordinator samples a number \mathbf{R} uniformly at random from $\{0, \dots, \ell - 2\}$ and then samples a subset $\mathbf{T} \subseteq [\ell] \setminus \{\mathbf{P}, \mathbf{Q}\}$ uniformly at random from all sets of size \mathbf{R} that do not contain \mathbf{P}, \mathbf{Q} . Then the coordinator samples $\mathbf{M}_t \sim \text{Bin}(1/3)$ for all $t \in \mathbf{T}$ and sends them to all the processors. The processors use their private randomness to sample \mathbf{X}_t for each

column t in \mathbf{T} in the following way: The input of the Z_t -th processor is fixed to 0 in \mathbf{X}_t and the other processors get 1 if $\mathbf{M}_t = 1$, otherwise, if $\mathbf{M}_t = 0$, they get 0 or 1 uniformly at random. We will call this input sampling procedure as `IpSample`.

4. For the rest of the columns, the coordinator samples the inputs according to `IpSample` and sends the requisite inputs to the respective processors.
5. The processors form the input $\mathbf{X} \equiv \mathbf{X}(u, \mathbf{P}, \mathbf{Q})$ (i.e., $\mathbf{X}_{\mathbf{P}} = u_1$ and $\mathbf{X}_{\mathbf{Q}} = u_2$) and run the protocol Π' for Disj_n with \mathbf{X} as input.

► **Observation 8.** Consider the tuple $(\mathbf{U}, \mathbf{N}, \mathbf{V}, \mathbf{S})$ distributed according to ζ_2 . If \mathbf{U} is given as input to protocol Π'' , then $(\mathbf{X}, \mathbf{M}, \mathbf{Z}, \mathbf{W})$ is distributed according to ζ_ℓ , where \mathbf{W} is the unique all 1's coordinate in \mathbf{X} . Here $\mathbf{W} = \mathbf{P}$ if $\mathbf{V} = 1$ and $\mathbf{W} = \mathbf{Q}$ if $\mathbf{V} = 2$.

Next we prove the following lemma connecting the information cost of Π' for Disj_n and that of Π'' for Disj_2 . This lemma implies the Theorem 7.

► **Lemma 9.**

$$\mathbb{I}_{(\mathbf{U}, \mathbf{N}, \mathbf{V}, \mathbf{S}) \sim \zeta_2} [\mathbf{U}_{-\mathbf{V}}, \Pi''^i(\mathbf{U}) \mid \mathbf{N}, \mathbf{V}, \mathbf{S}] \leq \frac{1}{\ell - 1} \mathbb{I}_{(\mathbf{X}, \mathbf{M}, \mathbf{W}, \mathbf{Z}) \sim \zeta_\ell} [\mathbf{X}_{-\mathbf{W}}, \Pi'^i(\mathbf{X}) \mid \mathbf{M}, \mathbf{W}, \mathbf{Z}], \quad (9)$$

and

$$\mathbb{I}_{(\mathbf{U}, \mathbf{N}, \mathbf{V}, \mathbf{S}) \sim \zeta_2} [\mathbf{N}_{-\mathbf{V}}, \Pi''^i(\mathbf{U}) \mid \mathbf{U}^i, \mathbf{V}, \mathbf{S}] \leq \frac{1}{\ell - 1} \mathbb{I}_{(\mathbf{X}, \mathbf{M}, \mathbf{W}, \mathbf{Z}) \sim \zeta_\ell} [\mathbf{M}_{-\mathbf{W}}, \Pi'^i(\mathbf{X}) \mid \mathbf{X}^i, \mathbf{W}, \mathbf{Z}]. \quad (10)$$

Proof. We consider the LHS of Equation (10). The view of processor i of the transcript of protocol Π'' , denoted as $\Pi''^i(\mathbf{U})$, is given as follows.

$$\Pi''^i(\mathbf{U}) = \langle \mathbf{P}, \mathbf{Q}, \mathbf{Z}_{-\mathbf{P}, \mathbf{Q}}, \mathbf{R}, \mathbf{T}, \mathbf{M}_{\mathbf{T}}, \mathbf{X}_{\mathbf{T} \setminus \{\mathbf{P}, \mathbf{Q}\}}^i, \Pi'(\mathbf{X}(\mathbf{P}, \mathbf{Q}, \mathbf{U})) \rangle. \quad (11)$$

So the LHS of Equation (10) can be written as

$$\begin{aligned} & \mathbb{I}_{(\mathbf{U}, \mathbf{N}, \mathbf{V}, \mathbf{S}) \sim \zeta_2} [\mathbf{N}_{-\mathbf{V}} : \Pi''^i(\mathbf{U}) \mid \mathbf{U}^i, \mathbf{V}, \mathbf{S}] \\ &= \mathbb{I}_{(\mathbf{U}, \mathbf{N}, \mathbf{V}, \mathbf{S}) \sim \zeta_2} [\mathbf{N}_{-\mathbf{V}} : \mathbf{P}, \mathbf{Q}, \mathbf{Z}_{-\mathbf{P}, \mathbf{Q}}, \mathbf{T}, \mathbf{M}_{\mathbf{T}}, \mathbf{R}, \mathbf{X}_{\mathbf{T} \setminus \{\mathbf{P}, \mathbf{Q}\}}^i, \Pi'^i(\mathbf{X}(\mathbf{P}, \mathbf{Q}, \mathbf{U})) \mid \mathbf{U}^i, \mathbf{V}, \mathbf{S}] \\ &= \mathbb{I}_{(\mathbf{U}, \mathbf{N}, \mathbf{V}, \mathbf{S}) \sim \zeta_2} [\mathbf{N}_{-\mathbf{V}} : \Pi'^i(\mathbf{X}(\mathbf{P}, \mathbf{Q}, \mathbf{U})) \mid \mathbf{P}, \mathbf{Q}, \mathbf{Z}_{-\mathbf{P}, \mathbf{Q}}, \mathbf{T}, \mathbf{M}_{\mathbf{T}}, \mathbf{R}, \mathbf{X}_{\mathbf{T} \setminus \{\mathbf{P}, \mathbf{Q}\}}^i, \mathbf{U}^i, \mathbf{V}, \mathbf{S}] \\ & \hspace{15em} [\text{Lemma 3 eqn. (4)}] \\ &= \mathbb{I}_{(\mathbf{U}, \mathbf{N}, \mathbf{V}, \mathbf{S}) \sim \zeta_2} [\mathbf{N}_{-\mathbf{V}} : \Pi'^i(\mathbf{X}) \mid \mathbf{P}, \mathbf{Q}, \mathbf{R}, \mathbf{T}, \mathbf{M}_{\mathbf{T}}, \mathbf{Z}, \mathbf{V}, \mathbf{X}_{\mathbf{T}}^i] \\ & \hspace{15em} [\text{Combining } (\mathbf{U}^i, \mathbf{X}_{\mathbf{T} \setminus \{\mathbf{P}, \mathbf{Q}\}}^i \text{ and } (\mathbf{Z}_{-\mathbf{P}, \mathbf{Q}}, \mathbf{S})] \\ &\leq \mathbb{I}_{(\mathbf{U}, \mathbf{N}, \mathbf{V}, \mathbf{S}) \sim \zeta_2} [\mathbf{N}_{-\mathbf{V}} : \Pi'^i(\mathbf{X}) \mid \mathbf{P}, \mathbf{Q}, \mathbf{R}, \mathbf{T}, \mathbf{M}_{\mathbf{T}}, \mathbf{Z}, \mathbf{V}, \mathbf{X}^i] \\ & \hspace{15em} [\text{Lemma 3 eqn. (5), } \mathbf{X}_{\mathbf{S}}^i \text{ ind. of } \mathbf{N}_{-\mathbf{V}}] \end{aligned}$$

[\mathbf{V} takes value in 1 and 2 uniformly at random. Hence we can write it as follows.]

$$\begin{aligned} &= \frac{1}{2} \mathbb{I}_{(\mathbf{X}, \mathbf{M}, \mathbf{Z}) \sim \zeta_\ell \downarrow \mathbf{X}, \mathbf{M}, \mathbf{Z}} [\mathbf{M}_{\mathbf{P}} : \Pi'^i(\mathbf{X}) \mid \mathbf{P}, \mathbf{Q}, \mathbf{R}, \mathbf{T}, \mathbf{M}_{\mathbf{T}}, \mathbf{Z}, \mathbf{V} = 2, \mathbf{X}^i] \\ & \quad + \frac{1}{2} \mathbb{I}_{(\mathbf{X}, \mathbf{M}, \mathbf{Z}) \sim \zeta_\ell \downarrow \mathbf{X}, \mathbf{M}, \mathbf{Z}} [\mathbf{M}_{\mathbf{Q}} : \Pi'^i(\mathbf{X}) \mid \mathbf{P}, \mathbf{Q}, \mathbf{R}, \mathbf{T}, \mathbf{M}_{\mathbf{T}}, \mathbf{Z}, \mathbf{V} = 1, \mathbf{X}^i]. \quad (12) \end{aligned}$$

Consider the first mutual information term.

$$\begin{aligned}
& \mathbb{I}[\mathbf{M}_{\mathbf{P}} : \Pi^{i_i}(\mathbf{X}) \mid \mathbf{P}, \mathbf{Q}, \mathbf{R}, \mathbf{T}, \mathbf{M}_{\mathbf{T}}, \mathbf{Z}, \mathbf{V} = 2, \mathbf{X}^i] \\
&= \frac{2}{\ell(\ell-1)} \sum_{p < q} \mathbb{I}[\mathbf{M}_p : \Pi^{i_i}(\mathbf{X}) \mid \mathbf{P} = p, \mathbf{Q} = q, \mathbf{R}, \mathbf{T}, \mathbf{M}_{\mathbf{T}}, \mathbf{Z}, \mathbf{V} = 2, \mathbf{X}^i] \\
&= \frac{2}{\ell(\ell-1)^2} \sum_{p < q} \sum_{r=0}^{\ell-2} \sum_{t:|t|=r} \Pr[\mathbf{T} = t] \mathbb{I}[\mathbf{M}_p : \Pi^{i_i}(\mathbf{X}) \mid p, q, r, t, \mathbf{M}_t, \mathbf{Z}, \mathbf{V} = 2, \mathbf{X}^i] \\
&= \frac{2}{\ell(\ell-1)^2} \sum_{p < q} \sum_{r=0}^{\ell-2} \sum_{t:|t|=r} \frac{(\ell-r-2)!r!}{(\ell-2)!} \mathbb{I}[\mathbf{M}_p : \Pi^{i_i}(\mathbf{X}) \mid p, q, r, t, \mathbf{M}_t, \mathbf{Z}, \mathbf{V} = 2, \mathbf{X}^i].
\end{aligned}$$

We can safely drop the conditioning $\mathbf{P} = p, \mathbf{R} = r, \mathbf{T} = t$ and $\mathbf{V} = 2$ in the following way. It is easy to see $\mathbf{R} = r, \mathbf{T} = t$ is implied by \mathbf{M}_t . \mathbf{M}_p implies $\mathbf{P} = p$. Moreover, given $(p, q), \mathbf{V} = 2$ is equivalent to $\mathbf{W} = p$. So we can write,

$$\begin{aligned}
& \mathbb{I}[\mathbf{M}_{\mathbf{P}} : \Pi^{i_i}(\mathbf{X}) \mid \mathbf{P}, \mathbf{Q}, \mathbf{R}, \mathbf{T}, \mathbf{M}_{\mathbf{T}}, \mathbf{Z}, \mathbf{V} = 2, \mathbf{X}^i] \\
&= \frac{2}{(\ell-1)! \ell(\ell-1)} \sum_q \sum_{p:p < q} \sum_{r=0}^{\ell-2} \sum_{t:|t|=r} ((\ell-r-2)!r!) \mathbb{I}[\mathbf{M}_p : \Pi^{i_i}(\mathbf{X}) \mid \mathbf{W} = q, \mathbf{M}_t, \mathbf{Z}, \mathbf{X}^i].
\end{aligned} \tag{13}$$

Similarly, the second mutual information term of Equation (12) term can be written in the following way.

$$\begin{aligned}
& \mathbb{I}[\mathbf{M}_{\mathbf{Q}} : \Pi^{i_i}(X) \mid \mathbf{P}, \mathbf{Q}, \mathbf{R}, \mathbf{T}, \mathbf{M}_{\mathbf{T}}, \mathbf{Z}, \mathbf{V} = 1, \mathbf{X}^i] \\
&= \frac{2}{(\ell-1)! \ell(\ell-1)} \sum_q \sum_{p:p < q} \sum_{r=0}^{\ell-2} \sum_{t:|t|=r} ((\ell-r-2)!r!) \mathbb{I}[\mathbf{M}_q : \Pi^{i_i}(\mathbf{X}) \mid \mathbf{W} = p, \mathbf{M}_t, \mathbf{Z}, \mathbf{X}^i].
\end{aligned} \tag{14}$$

Combining Equation (12), (13), (14), we get,

$$\begin{aligned}
& \mathbb{I}(\mathbf{U}, \mathbf{N}, \mathbf{V}, \mathbf{S})_{\sim \zeta_2} [\mathbf{N}_{-\mathbf{V}}, \Pi^{i_i}(\mathbf{U}) \mid \mathbf{U}^i, \mathbf{V}, \mathbf{S}] \\
&\leq \frac{1}{(\ell-1)! \ell(\ell-1)} \sum_{q'} \sum_{p':p' \neq q'} \sum_{r=0}^{\ell-2} \sum_{t:|t|=r} ((\ell-r-2)!r!) \mathbb{I}[\mathbf{M}_{p'} : \Pi^{i_i}(X) \mid \mathbf{W} = q', \mathbf{M}_t, \mathbf{Z}, \mathbf{X}^i]
\end{aligned}$$

The number of permutations of $[\ell] \setminus q$ where the $r+1$ th element is p' and the first r elements constitute the set t is $(\ell-r-2)!r!$. Hence we can write the previous summation as follows,

$$\begin{aligned}
&= \frac{1}{(\ell-1)! \ell(\ell-1)} \sum_{q'} \sum_{\sigma \in \mathcal{S}_{[\ell] \setminus q'}} \sum_{i \in [\ell] \setminus q'} \mathbb{I}[\mathbf{M}_{\sigma(i)} : \Pi^{i_i}(\mathbf{X}) \mid \mathbf{M}_{\{\sigma(1), \dots, \sigma(i-1)\}}, \mathbf{Z}, \mathbf{W} = q', \mathbf{X}^i] \\
&= \frac{1}{(\ell-1)! \ell(\ell-1)} \sum_{q'} \sum_{\sigma \in \mathcal{S}_{[\ell] \setminus q'}} \mathbb{I}[\mathbf{M}_{-q'} : \Pi^{i_i}(\mathbf{X}) \mid \mathbf{Z}, \mathbf{W} = q', \mathbf{X}^i] \\
&\hspace{15em} \text{[Using chain rule of information, Eq. (3)]} \\
&= \frac{1}{\ell-1} \sum_{q'} \frac{1}{\ell} \mathbb{I}_{(\mathbf{X}, \mathbf{M}, \mathbf{Z}) \sim \zeta_{\ell} \downarrow \mathbf{X}, \mathbf{M}, \mathbf{Z}} [\mathbf{M}_{-q'} : \Pi^{i_i}(\mathbf{X}) \mid \mathbf{Z}, \mathbf{W} = q', \mathbf{X}^i] \\
&= \frac{1}{\ell-1} \mathbb{I}_{(\mathbf{X}, \mathbf{M}, \mathbf{Z}, \mathbf{W}) \sim \zeta_{\ell}} [\mathbf{M}_{-\mathbf{W}} : \Pi^{i_i}(\mathbf{X}) \mid \mathbf{Z}, \mathbf{W}, \mathbf{X}^i].
\end{aligned} \tag{15}$$

Equation (9) can be proved in the similar way and therefore omitted. \blacktriangleleft

4.2 Lower Bounding Disj_2

In this section we prove the following.

► **Theorem 10.** $\text{IC}(\text{Disj}_2) \geq \text{PIC}(\text{Disj}_2) \geq \Omega(1)$.

This, combined with Theorem 7 and Theorem 6 will imply a $\Omega(m\ell k)$ lower bound on the switched information complexity of $\text{Tribes}_{m,n}$ which is the lower bound on $R_\delta(\text{Tribes}_{m,n})$ we aimed for.

Notation. By \bar{e} we mean the all 1 vector of size k . By $\bar{e}_{i,j}$, we mean the boolean vector of size k where all entries are 1 except the entries in index i and j . Similarly, \bar{e}_i is the boolean vector where all entries are 1 except that of index i . $\Pi[i, x, m, z; \bar{e}_i]$ implies the transcript of the protocol Π on the following Disj_2 instance: the input of the first column comes from the distribution specified by $\mathbf{M} = m$, $\mathbf{Z} = z$ and $\mathbf{X}_1^i = x$ and the input of the second column is \bar{e}_i . Abusing notation slightly, $\Pi^i[x, m, z; \bar{e}_i]$ represents processor- i 's view of the transcript $\Pi[i, x, m, z; \bar{e}_i]$.

Hellinger distance. For probability distributions P and Q supported on a sample space Ω , the Hellinger distance between P and Q , denoted as $h(P, Q)$, is defined as, $h(P, Q) = \frac{1}{\sqrt{2}} \|\sqrt{P} - \sqrt{Q}\|_2 = 1 - F(P, Q)$, where $F(P, Q) = \sum_{\omega \in \Omega} \sqrt{P(\omega)Q(\omega)}$ is also known as Bhattacharya coefficient. Below we will state a fact (without proof) about Hellinger distance.

► **Fact 11** ([2]). *Let Π be a δ -error protocol for function f . For inputs x and y such that $f(x) \neq f(y)$, we have,*

$$h(\Pi(x), \Pi(y)) \geq \frac{1 - \delta}{\sqrt{2}}. \quad (16)$$

The following lemmas are generalization of their two-party analogues.

► **Lemma 12** (*k*-party cut-paste). *For any randomized protocol Π computing $f : X^k \rightarrow \{0, 1\}$ and for any $x, y \in X^k$ and for some i and j ,*

$$h(\Pi(x_i x_j x_{-i,j}, y_i y_j y_{-i,j})) = h(\Pi(x_i y_j x_{-i,j}, y_i x_j y_{-i,j})). \quad (17)$$

► **Lemma 13** (Pythagorean). *For any randomized protocol Π and for any input $x, y \in X^k$ and for some i and j ,*

$$2h^2(\Pi(x_i x_j x_{-i,j}, y_i y_j y_{-i,j})) \geq h^2(\Pi(x_i x_j x_{-i,j}, x_i y_j y_{-i,j})) + h^2(\Pi(y_i x_j x_{-i,j}, y_i y_j y_{-i,j})). \quad (18)$$

Following structural properties are generalizations of analogous properties shown in [4]. Simpler proofs will be included in full version.

► **Lemma 14** (Diagonal). *For $i \neq j$*

$$h^2(\Pi^i[0, 0, j; \bar{e}], \Pi^i[1, 1, z; \bar{e}]) \geq \frac{1}{2} h^2(\Pi^i(\bar{e}_{i,j}; \bar{e}), \Pi^i(\bar{e}_j; \bar{e})). \quad (19)$$

► **Lemma 15** (Global-to-local). *For $i \neq j$*

$$h^2(\Pi[i, 0, 0, z; \bar{e}], \Pi[i, 1, 0, z; \bar{e}]) = h(\Pi^i[0, 0, z; \bar{e}], \Pi^i[1, 0, z; \bar{e}]), \quad (20)$$

and

$$h(\Pi(\bar{e}_{i,j}; \bar{e}), \Pi(\bar{e}_i; \bar{e})) = h(\Pi^i(\bar{e}_{i,j}; \bar{e}), \Pi^i(\bar{e}_i; \bar{e})). \quad (21)$$

Now we are ready to prove the partial information cost of Disj_2 is $\Omega(k)$. We consider processor i and fix a value $j \neq i$.

► **Claim 16** ([4]).

$$(1) \mathbb{I}[\mathbf{M}_{-\mathbf{W}} : \Pi^i \mid \mathbf{X}^i, \mathbf{Z} = j, \mathbf{W} = 2] \geq \frac{2}{3} h^2(\Pi^i[1, 0, j; \bar{e}], \Pi^i[1, 1, j; \bar{e}]), \quad (22)$$

$$(2) \mathbb{I}[\mathbf{X}_{-\mathbf{W}}^i : \Pi^i \mid \mathbf{M}, \mathbf{Z} = j, \mathbf{W} = 2] \geq \frac{2}{3} h^2(\Pi^i[0, 0, j; \bar{e}], \Pi^i[1, 0, j; \bar{e}]), \quad (23)$$

$$(3) \mathbb{I}[\mathbf{M}_{-\mathbf{W}} : \Pi^i \mid \mathbf{X}^i, \mathbf{Z} = j, \mathbf{W} = 1] \geq \frac{2}{3} h^2(\Pi^i[\bar{e}; 1, 0, j], \Pi^i[\bar{e}; 1, 1, j]), \quad (24)$$

$$(4) \mathbb{I}[\mathbf{X}_{-\mathbf{W}}^i : \Pi^i \mid \mathbf{M}, \mathbf{Z} = j, \mathbf{W} = 1] \geq \frac{2}{3} h^2(\Pi^i[\bar{e}; 0, 0, j], \Pi^i[\bar{e}; 1, 0, j]). \quad (25)$$

Using Cauchy-Schwarz and triangle inequality, we can write the following.

$$\begin{aligned} & \sum_i \mathbb{I}[\mathbf{M}_{-\mathbf{W}} : \Pi^i \mid \mathbf{X}^i, \mathbf{Z}, \mathbf{W}] + \mathbb{I}[\mathbf{X}_{-\mathbf{W}}^i : \Pi^i \mid \mathbf{M}, \mathbf{Z}, \mathbf{W}] \\ & \geq \frac{1}{3k} \sum_i \sum_{j:i \neq j} [h^2(\Pi^i[1, 1, j; \bar{e}], \Pi^i[0, 0, j; \bar{e}]) + h^2(\Pi^i[\bar{e}; 1, 1, j], \Pi^i[\bar{e}; 0, 0, j])] \\ & \hspace{20em} \text{[Claim 16]} \\ & \geq \frac{1}{6k} \sum_i \sum_{j:i \neq j} [h^2(\Pi^i(\bar{e}_{i,j} \cdot \bar{e}), \Pi^i(\bar{e}_j \cdot \bar{e})) + h^2(\Pi^i(\bar{e}\bar{e}_{i,j}), \Pi^i(\bar{e}\bar{e}_j))]. \quad \text{[Lemma 14]} \\ & \geq \frac{1}{6k} \sum_i \sum_{j:i \neq j} [h^2(\Pi(\bar{e}_{i,j} \cdot \bar{e}), \Pi(\bar{e}_j \cdot \bar{e})) + h^2(\Pi(\bar{e}\bar{e}_{i,j}), \Pi(\bar{e}\bar{e}_j))] \quad \text{[Lemma 15]} \\ & \geq \frac{1}{24k} \sum_{i \neq j} [h^2(\Pi(\bar{e}_i \cdot \bar{e}), \Pi(\bar{e}_j \cdot \bar{e})) + [h^2(\Pi(\bar{e}\bar{e}_i), \Pi(\bar{e}\bar{e}_j))] \quad \text{[Recounting \& Tr. ineq.]} \\ & = \frac{1}{24k} \sum_{i \neq j} [h^2(\Pi(\bar{e} \cdot \bar{e}), \Pi(\bar{e}_{i,j} \cdot \bar{e})) + [h^2(\Pi(\bar{e}\bar{e}), \Pi(\bar{e}\bar{e}_{i,j}))] \quad \text{[Lemma 12]} \\ & \geq \frac{1}{48k} \sum_{i \neq j} [h^2(\Pi(\bar{e} \cdot \bar{e}_{i,j}), \Pi(\bar{e}_{i,j} \cdot \bar{e}))] \quad \text{[Cauchy-Schwarz \& triangle inequality]} \\ & \geq \frac{1}{96k} \sum_{i \neq j} [h^2(\Pi(\bar{e} \cdot \bar{e}_{i,j}), \Pi(\bar{e}_j \cdot \bar{e}_i)) + h^2(\Pi(\bar{e}_{i,j} \bar{e}), \Pi(\bar{e}_i \cdot \bar{e}_j))] \quad \text{[Lemma 13]} \\ & = \frac{k-1}{384} (1-\delta)^2 = \Omega(k). \quad \text{[Fact 11]} \end{aligned}$$

5 Putting Everything Together

In this section we show randomized communication complexity of any function f is lower bounded by the information complexity of f .

► **Theorem 17.** *For any distribution μ over the inputs,*

$$R_\epsilon(\text{Tribes}_{m,n}) = \Omega(\text{IC}_\mu(\text{Tribes}_{m,n})). \quad (26)$$

This follows from the fact that the expected length of any instantaneous q -ary code for a random variable X is at least $\mathcal{H}(X)/\log q$. We omit the proof for space constraint. Using Theorem 6, 7, 10 and 17, it is not hard to see that Theorem 1 follows.

References

- 1 Maria-Florina Balcan, Avrim Blum, Shai Fine, and Yishay Mansour. Distributed learning, communication complexity and privacy. In Shie Mannor, Nathan Srebro, and Robert C. Williamson, editors, *COLT 2012 - The 25th Annual Conference on Learning Theory, June 25-27, 2012, Edinburgh, Scotland*, pages 26.1–26.22. JMLR.org, 2012.
- 2 Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. In *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*, pages 209–218. IEEE Computer Society, 2002.
- 3 Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. In Richard Hull and Wenfei Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 273–284. ACM, 2013.
- 4 Mark Braverman, Faith Ellen, Rotem Oshman, Toniann Pitassi, and Vinod Vaikuntanathan. A tight bound for set disjointness in the message-passing model. In *FOCS*, pages 668–677. IEEE Computer Society, 2013.
- 5 Arkadev Chattopadhyay and Toniann Pitassi. The story of set disjointness. *SIGACT News*, 41(3):59–85, 2010.
- 6 Danny Dolev and Tomás Feder. Determinism vs. nondeterminism in multiparty communication complexity. *SIAM J. Comput.*, 21(5):889–895, 1992.
- 7 Andrew Drucker, Fabian Kuhn, and Rotem Oshman. The communication complexity of distributed task allocation. In Darek Kowalski and Alessandro Panconesi, editors, *ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012*, pages 67–76. ACM, 2012.
- 8 Pavol Duris and José D. P. Rolim. Lower bounds on the multiparty communication complexity. *J. Comput. Syst. Sci.*, 56(1):90–95, 1998.
- 9 Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In Takao Asano, Shin-Ichi Nakano, Yoshio Okamoto, and Osamu Watanabe, editors, *Algorithms and Computation - 22nd International Symposium, ISAAC 2011, Yokohama, Japan, December 5-8, 2011. Proceedings*, volume 7074 of *Lecture Notes in Computer Science*, pages 374–383. Springer, 2011.
- 10 T. S. Jayram, Ravi Kumar, and D. Sivakumar. Two applications of information complexity. In Lawrence L. Larmore and Michel X. Goemans, editors, *STOC*, pages 673–682. ACM, 2003.
- 11 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In Moses Charikar, editor, *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 938–948. SIAM, 2010.
- 12 Paraschos Koutris and Dan Suciu. Parallel evaluation of conjunctive queries. In Maurizio Lenzerini and Thomas Schwentick, editors, *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2011, June 12-16, 2011, Athens, Greece*, pages 223–234. ACM, 2011.
- 13 Eyal Kushilevitz and Noam Nisan. *Communication complexity*. Cambridge University Press, 1997.
- 14 Jeff M. Phillips, Elad Verbin, and Qin Zhang. Lower bounds for number-in-hand multiparty communication complexity, made easy. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 486–501. SIAM, 2012.
- 15 David P. Woodruff and Qin Zhang. Tight bounds for distributed functional monitoring. In Howard J. Karloff and Toniann Pitassi, editors, *Proceedings of the 44th Symposium on*

- Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 941–960. ACM, 2012.
- 16 David P. Woodruff and Qin Zhang. When distributed computation is communication expensive. In Yehuda Afek, editor, *Distributed Computing - 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14-18, 2013. Proceedings*, volume 8205 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2013.
 - 17 David P. Woodruff and Qin Zhang. An optimal lower bound for distinct elements in the message passing model. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 718–733. SIAM, 2014.
 - 18 Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings of the 11th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 209–213. ACM, 1979.

Network Design Problems with Bounded Distances via Shallow-Light Steiner Trees

Markus Chimani¹ and Joachim Spoerhase²

1 Theoretical Computer Science, Osnabrück University, Germany
markus.chimani@uni-osnabrueck.de

2 Institute of Computer Science, University of Würzburg, Germany
joachim.spoerhase@uni-wuerzburg.de

Abstract

In a directed graph G with non-correlated edge lengths and costs, the *network design problem with bounded distances* asks for a cost-minimal spanning subgraph subject to a length bound for all node pairs. We give a bi-criteria $(2 + \varepsilon, O(n^{0.5+\varepsilon}))$ -approximation for this problem. This improves on the currently best known linear approximation bound, at the cost of violating the distance bound by a factor of at most $2 + \varepsilon$.

In the course of proving this result, the related problem of *directed shallow-light Steiner trees* arises as a subproblem. In the context of directed graphs, approximations to this problem have been elusive. We present the first non-trivial result by proposing a $(1 + \varepsilon, O(|R|^\varepsilon))$ -approximation, where R is the set of terminals.

Finally, we show how to apply our results to obtain an $(\alpha + \varepsilon, O(n^{0.5+\varepsilon}))$ -approximation for *light-weight directed α -spanners*. For this, no non-trivial approximation algorithm has been known before. All running times depends on n and ε and are polynomial in n for any fixed $\varepsilon > 0$.

1998 ACM Subject Classification G.2.1 Combinatorial algorithms, G.2.2 Graph algorithms, Network problems

Keywords and phrases network design, approximation algorithm, shallow-light spanning trees, spanners

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.238

1 Introduction

We consider the following network design problem introduced by Dodis and Khanna [6]:

► **Definition 1** (Directed Network Design with Bounded Distances). Given a directed graph $G = (V, E)$, an edge cost function $c: E \rightarrow \mathbb{N}$, an edge length function $\ell: E \rightarrow \mathbb{N}$, and a length bound $L \in \mathbb{N}$. We ask for a spanning subgraph H of G of minimum cost (with respect to c) such that for each node pair u, v the distance in H (with respect to ℓ) is at most L .

Generally, for a given graph $G = (V, E)$, we let $n := |V|$ and $m := |E|$; $\bar{\ell}_H(u, v)$ denotes the lengths of the shortest u - v path in $H \subseteq G$ with respect to ℓ . For uniform edge costs and lengths, Dodis and Khanna [6] devise an $O(\log n \log L)$ -approximation. For non-uniform edge costs, they show $\Omega(2^{\log^{1-\varepsilon} n})$ -hardness of approximation, and propose an $O(n \log L)$ -approximation under the restriction that the edge lengths are polynomially bounded. Up to now, no improved algorithm is known.

In this paper (Section 2), we give an algorithm for this problem, without any of the above restrictions and without ratio-dependency on L , achieving essentially a performance ratio $O(\sqrt{n})$ while violating the distance bound L by a factor of at most $2 + \varepsilon$.

► **Theorem 2.** *There is a bi-criteria $(2 + \varepsilon, O(n^{1/2+\varepsilon}))$ -approximation for the above directed network design problem with bounded distances.*

As a starting point, our algorithm uses a two-stage approach originally proposed by Feldman et al. [8] for directed Steiner forest, which has later been reused for directed spanners [3, 5, 2]. We divide the considered node pairs into *thin* and *thick* pairs. We settle the former by LP-rounding, as we have to cover certain cuts w.r.t. shortest paths. For the latter, we sample nodes and construct short in- and out-trees for each of them. This latter part is a main technical challenge: In contrast to the case of sparse spanners, we cannot simply use shortest-path trees, as they could have arbitrarily high costs. To solve this issue, we turn our attention to a second problem, which is also of independent interest:

► **Definition 3** (Directed Shallow-Light Steiner Trees). Given a directed graph $G = (V, E)$, an edge cost function $c: E \rightarrow \mathbb{N}$, an edge length function $\ell: E \rightarrow \mathbb{N}$, a distinguished root node $r \in V$, and a set $R \subseteq V$ of terminals with distance bounds $d: R \rightarrow \mathbb{N}$. We ask for an r -rooted subtree T of G of minimum cost (with respect to c) such that for any terminal $v \in R$ the distance $\bar{\ell}_T(r, v)$ in T (with respect to ℓ) is at most $d(v)$.

Kortsarz and Peleg [11] gave an $O(|R|^\varepsilon)$ -approximation for undirected graphs with uniform edge lengths and uniform distance bounds. The directed problem with non-uniform edge costs has formerly been considered in [12], where a bi-criteria $(2, O(\log n))$ -approximation for directed shallow-light *spanning* trees (that is, $R = V$) was proposed. Unfortunately, the proof has an error¹, and there has not been any progress on the problem since. We propose the first non-trivial result for the general directed problem (cf. Section 3). In fact, at the cost of violating the length bounds by a factor of at most $(1 + \varepsilon)$, we obtain the same approximation ratio as [11], but for directed graphs and without the restrictions to uniform lengths and costs:

► **Theorem 4.** *There is a bi-criteria $(1 + \varepsilon, |R|^\varepsilon)$ -approximation for directed shallow-light Steiner trees.*

Finally (Section 4), we give a further application of our shallow-light Steiner tree result:

► **Definition 5** (Light-Weight Directed α -Spanners). Given a directed graph $G = (V, E)$, an edge cost function $c: E \rightarrow \mathbb{N}$, an edge length function $\ell: E \rightarrow \mathbb{N}$, and a stretch factor $\alpha \geq 1$. We ask for a spanning subgraph H of G of minimum cost (with respect to c) such that for each node pair u, v the distance $\bar{\ell}_H(u, v)$ in H (with respect to ℓ) is at most $\alpha \cdot \bar{\ell}_G(u, v)$, i.e., α times their distance in G .

As of now, this problem has only been successfully tackled for undirected graphs [13, 1]. Its directed variant remained an interesting open problem [5]². We give the first non-trivial result:

► **Theorem 6.** *There is a bi-criteria $(\alpha + \varepsilon, O(n^{1/2+\varepsilon}))$ -approximation for light-weight directed α -spanners.*

¹ Verified by personal communication with J. Naor.

² As mentioned in the corresponding slides, available online.

2 Network Design with Bounded Pairwise Distance

We build our solution network as the union of subgraphs. We say such a subgraph *settles* a node pair (u, v) , if it includes a path connecting u to v complying with the distance bound. As sketched above, the overall scheme of our approximation algorithm is to classify node pairs into two categories. Let $(u, v) \in V \times V$ be any node pair, and \mathcal{P}_{uv}^L the set of all u - v paths of length at most L . We denote with $V_{uv} := \bigcup_{P \in \mathcal{P}_{uv}^L} V(P)$ and $E_{uv} := \bigcup_{P \in \mathcal{P}_{uv}^L} E(P)$ the nodes and edges, respectively, contained in any such path. The node pair (u, v) is called *thin* if $|V_{uv}| \leq \sqrt{n}$ and *thick* otherwise. We settle node pairs based on this classification. However, we will never explicitly compute any $\mathcal{P}_{uv}^L, V_{uv}, E_{uv}$ nor any node-pair classifications. They are only of interest for the approximation proof. We note that the concept of this classification is lifted from Feldman et al. [8]. The handling of the thin pairs follows the idea of anti-spanners by Berman et al. [2], as it can be made to work in our context, see below. Successfully tackling the thick pairs, however, is a technical challenge and requires our result on shallow-light trees (see Section 3). Let OPT denote the value of the optimum solution to the full problem.

2.1 Thin Pairs

2.1.1 Path-based LP

We consider the following path-based LP relaxation of the problem, requiring an exponential number of variables. Let $\mathcal{P}^L := \bigcup_{(u,v) \in V \times V} \mathcal{P}_{uv}^L$.

$$\begin{aligned}
\min \quad & \sum_{e \in E} c_e x_e, \quad \text{s.t.} \\
& \sum_{P \in \mathcal{P}_{u,v}^L} f_P \geq 1 \quad \forall (u, v) \in V \times V \\
& \sum_{P \in \mathcal{P}_{u,v}^L, P \ni e} f_P \leq x_e \quad \forall e \in E, (u, v) \in V \times V \\
& x_e \geq 0, \quad f_P \geq 0 \quad \forall e \in E, \quad \forall P \in \mathcal{P}^L
\end{aligned} \tag{1}$$

Its dual can be written as:

$$\begin{aligned}
\max \quad & \sum_{(u,v) \in V \times V} \alpha_{uv}, \quad \text{s.t.} \\
& \sum_{e \in P} \beta_{uv}^e \geq \alpha_{uv} \quad \forall (u, v) \in V \times V, P \in \mathcal{P}_{uv}^L \\
& \sum_{(u,v) \in V \times V} \beta_{uv}^e \leq c_e \quad \forall e \in E \\
& \beta_{uv}^e \geq 0, \quad \alpha_{uv} \geq 0 \quad \forall e \in E, (u, v) \in V \times V
\end{aligned} \tag{2}$$

LP (1) has an exponential number of variables. Below, we argue that we can get a PTAS for this LP by an approach analogous to the one proposed in [5]. Let $\varepsilon > 0$. We first consider the dual LP (2). This LP has a polynomial number of variables but an exponential number of constraints. We use the ellipsoid method to get an approximate solution to it. The separation oracle works as follows. (We do not consider the constraints $\sum_{(u,v) \in V \times V} \beta_{uv}^e \leq c_e$ since there are only polynomially many of these.) For each fixed $(u, v) \in V \times V$, we consider variables the β_{uv}^e as edge weights. Thus, determining whether a constraint is violated for some $P \in \mathcal{P}_{uv}^L$ amounts to checking whether α_{uv} is at most the weight of a lightest u - v path (under weights β_{uv}^e) whose length (under edge lengths ℓ) is bounded by L . Already this

necessary subproblem (*length-bounded shortest path*) is NP-hard. However, Hassin [9], later sped up by Ergun et al. [7], describes an FPTAS. Assume we run the ellipsoid algorithm by using this approximate separation oracle with error parameter ε . Then, we end up with an optimum solution to the *restricted* dual LP, which has only constraints for paths $P \in \mathcal{P}^L$ that we included when running the ellipsoid algorithm. Since we used an FPTAS for the separation oracle, the constraints that we did not include can be violated by a factor at most $1 - \varepsilon$. That is, we have $\sum_{e \in P} \beta_{uv}^e \geq (1 - \varepsilon)\alpha_{uv}$ for all paths $P \in \mathcal{P}^L$ that we did not include. Hence, if we set $\alpha'_{uv} = (1 - \varepsilon)\alpha_{uv}$ we obtain a feasible solution to the original dual LP that is $(1 - \varepsilon)$ -approximate with respect to the optimum solution of the restricted dual. Now suppose that we solve the *restricted* primal LP where we only include the (polynomially many) variables that correspond to constraints of the restricted dual. Then the optimum solution to this LP is at most $1/(1 - \varepsilon)$ times larger than the optimum solution to the original dual (and hence the original primal) since the restricted dual LP is the dual to the restricted primal LP and since the original dual is $(1 - \varepsilon)$ -approximate to the restricted dual.

2.1.2 Randomized LP Rounding

We describe an algorithm that computes a subgraph $H_1 \subseteq G$ where the distance $\bar{\ell}_{H_1}(u, v)$ is at most L for every thin pair (u, v) . The algorithm first solves the above LP within a ratio of $1 + \varepsilon$. Then each edge e is sampled with probability $\min(\gamma \cdot x_e, 1)$ where $\gamma := \sqrt{n} \cdot \log n$. The cost of H_1 is $O(\gamma(1 + \varepsilon)\text{OPT})$. We have to show that this algorithm creates a feasible solution with high probability.

► **Definition 7.** Let (u, v) be a thin pair, $C \subseteq E$ a set of edges, and $G_C := (V, E \setminus C)$. We say C is a *u-v-stretching cut* if $\bar{\ell}_{G_C}(u, v) \leq L$ for all $C' \subset C$ but $\bar{\ell}_{G_C}(u, v) > L$.

► **Lemma 8.** Let $H = (V, E')$ be a subgraph of G and (u, v) a thin pair. H settles (u, v) if and only if each *u-v-stretching cut* contains at least one edge of E' .

Proof. If there is a *u-v-stretching cut* C that contains no edge of E' then $E' \subseteq E \setminus C$ and hence $\bar{\ell}_H(u, v) \geq \bar{\ell}_{G_C}(u, v) > L$. Conversely, if H does not settle (u, v) then $\bar{\ell}_H(u, v) > L$ and hence $E \setminus E'$ would contain a *u-v-stretching cut* C , which clearly has no edge of E' . ◀

► **Lemma 9.** For each thin pair (u, v) the number of *u-v-stretching cuts* is at most $\sqrt{n}^{\sqrt{n}}$.

Proof. Consider some *u-v-stretching cut* C and let T be a shortest path tree in the graph $H_C := (V_{uv}, E_{uv} \setminus C)$ rooted at u . Let $\bar{\ell}_T(w)$ denote the distance from u to w in T . If there is no *u-w* path in H_C then $\bar{\ell}_T(w) := \infty$. We show that $C = \{wx \in E_{uv} \mid \bar{\ell}_T(w) + \ell(wx) < \bar{\ell}_T(x)\}$, which implies that C is uniquely determined by T .

Consider an edge $wx \in E_{uv}$ such that $\bar{\ell}_T(w) + \ell(wx) < \bar{\ell}_T(x)$. Then $wx \in C$ because T is a shortest path tree in H_C .

Now, let $wx \in C$. Because $C' := C \setminus \{wx\}$ is not a *u-v* stretching cut there is a *u-v* path in $H_{C'} := (V_{uv}, E_{uv} \setminus C')$ of length at most L . This path must use the edge wx and has length $\bar{\ell}_T(w) + \ell(wx) + \bar{\ell}_{H_C}(x, v)$. Since H_C has no *u-v* path of length at most L we can conclude that $\bar{\ell}_{H_C}(u, x) + \bar{\ell}_{H_C}(x, v) > L$ and therefore $\bar{\ell}_T(w) + \ell(wx) < \bar{\ell}_{H_C}(u, x) = \bar{\ell}_T(x)$.

Hence the *u-v-stretching cut* C is uniquely determined by the tree T . We now count the number of rooted trees in H_C . For every node in such an out-tree there are \sqrt{n} possibilities to choose its parent node. Hence the total number of rooted trees and therefore the number of *u-v-stretching cuts* can be upper bounded by $\sqrt{n}^{\sqrt{n}}$. ◀

► **Lemma 10.** The above algorithm settles each thin pair with high probability.

Proof. By Lemma 8, it suffices to show that for every thin pair (u, v) and every u - v stretching cut C there is an edge from H_1 in C with high probability.

For every such cut C the LP value $\sum_{e \in C} x_e$ must be at least 1. This holds because every u - v path in $\mathcal{P}_{u,v}^L$ must contain at least one edge of C , since the total flow sent along these paths is at least 1 and since $\sum_{e \in C} x_e$ is an upper bound on this total flow because of the constraints $\sum_{P \in \mathcal{P}_{u,v}^L, P \ni e} f_P \leq x_e$ in the LP. If $\gamma \cdot x_e \geq 1$ for some $e \in C$ then $e \in E(H_1)$. Otherwise, the probability that none of the edges in C is sampled is at most

$$\prod_{e \in C} (1 - \gamma x_e) \leq \prod_{e \in C} e^{-\gamma x_e} = e^{-\gamma \sum_{e \in C} x_e} \leq e^{-\gamma} \leq n^{-\sqrt{n}}.$$

By Lemma 9, the total number of stretching cuts is at most $n^2 \sqrt{n}^{\sqrt{n}}$. Hence the probability that at least one stretching cut contains no edge of H_1 is at most $\sqrt{n}^{-\Omega(\sqrt{n})}$. ◀

2.2 Thick Pairs and Overall Algorithm

We now describe an algorithm to settle all thick pairs. The algorithm samples a set of $\delta = 3\sqrt{n} \log n$ many nodes of G . For each node u in this set, the algorithm determines a u -rooted shallow-light Steiner tree T_u by means of the algorithm described in Section 3 and summarized in Theorem 4. As input for this algorithm we use the graph G , the edge costs c and the edge lengths ℓ as in the instance of the network design problem; the root is the node u and the set R of terminals are all $V \setminus \{u\}$; we use L as the distance bound for each node. Similarly, the algorithm computes an in-tree rooted at u such that for each node the distance to u is at most L . This can be accomplished by computing a shallow-light Steiner tree T' in the graph G' arising from G by reversing all edges and then reversing the edges of T' . The output H_2 of the process is the union of all these spanning trees.

Our overall algorithm then returns $H_1 \cup H_2$, the union of the solution for the thin and the thick pairs, respectively. We are now ready to prove the following theorem:

► **Theorem 1 (Revisited).** *The above algorithm is a $(2 + \varepsilon, O(n^{1/2+\varepsilon}))$ -approximation algorithm for the directed network design problem with bounded distances (cf. Definition 1). The running time depends on n and ε and is polynomial in n for any fixed $\varepsilon > 0$.*

Proof. We first show that the algorithm outputs a feasible solution with high probability. In the light of Lemma 10, it remains to show that all thick pairs are settled with high probability. A thick pair (u, v) is settled if the above algorithm samples a node r from the set V_{uv} . In this case, the inclusion of the r -rooted in-tree and the r -rooted out-tree guarantees the existence of a u - v path of length at most $2(1 + \varepsilon)L$: we travel from u to r and then from r to v . Since for any thick pair its set V_{uv} contains at least \sqrt{n} many nodes, the probability that none of the δ many sampled nodes are from V_{uv} can be bounded by

$$\left(1 - \frac{1}{\sqrt{n}}\right)^\delta \leq e^{-3 \log n} = \frac{1}{n^3}.$$

Since there are at most n^2 thick pairs the claim follows.

We now analyze the cost of the algorithm. The cost of the procedure for settling thin pairs is $\gamma(1 + \varepsilon)\text{OPT}$ since every edge is sampled with probability at most γ times higher than its LP value. Now observe that every tree constructed in the procedure for thick pairs has cost at most $O(n^\varepsilon)\text{OPT}$. This follows from the fact that the optimum solution to the network design problem ensures the existence of a feasible solution to the problem of finding the rooted subtrees, and that the algorithm from Section 3 is an $O(n^\varepsilon)$ -approximation algorithm. Since the number of such trees constructed by the algorithm is $O(\delta)$ the ratio of the algorithm is bounded by $O(\delta n^\varepsilon + \gamma) = O(n^{1/2+\varepsilon})$. ◀

3 Directed Shallow-Light Steiner Trees

Let T be a rooted out-tree, i.e., its edges are directed from the root towards the leaves. A *branch node* is a node with out-degree larger than 1; as a special case, we always consider the root node to be a branch node. We say T is an i -level tree if no path from the root to any leaf contains more than i branch nodes.

Let $T \subseteq G$ be any out-tree, subgraph of a complete digraph G , with an arbitrary number of levels. Clearly, we can find a related out-tree with the same root and leaves requiring at most i levels, for any given i . If the edges have metric weights, a very general result by Helvig et al. [10] relates the weights of these two trees:

► **Lemma 2** (Helvig et al. [10]). *Let T be a rooted subtree of weight $c(T)$ with k leaves in a metrically-weighted complete digraph, and T_i the cheapest subtree with the same root and leaves and at most i levels. We have $c(T_i) \leq 2i(k/2)^{1/i}c(T)$.*

A typical application of this lemma is the following: Assuming metric edge weights, any digraph can be considered complete by adding artificial edges corresponding to paths in G . Consider any optimization problem whose solution is a tree. We can establish an approximation algorithm for it by first finding an approximation for the best p -level solution, for some p . We can then apply the lemma to obtain an approximation ratio to the original non-level-restricted problem. In our application, we have non-correlated edge costs and lengths. However, in order to apply the lemma, it suffices to observe that if there is a node pair (u, w) without any edge uw of length at most $\ell(uv) + \ell(vw)$, for any node v , we could (conceptually) insert an edge with this length and cost $c(uv) + c(vw)$ representing this u - v - w path. Observe that this would, in general, result in multiple edges connecting the same node pair, with different length/cost combinations. We do not need to explicitly consider these additional edges. In our algorithm, we will directly identify the corresponding paths meeting at branch nodes. Furthermore, by adding edges of zero length and cost, we can in the following always assume that there is an optimum solution where all terminals appear as leaves.

3.1 Algorithm

As mentioned above, there is an FPTAS [9, 7] to solve the problem of finding the cheapest (with respect to edge costs c) path from a node u to a node v of length at most D (with respect to edge length ℓ). We denote the result of this FPTAS by $\text{MINCOSTPATH}(u, v, D)$.

Our algorithm employs a recursive greedy strategy, which has been originally invented by Zelikovsky [14]. It has later been applied by Kortsarz and Peleg [11] to undirected Shallow-Light Steiner Trees. Specifically, they give an $(2 + \varepsilon, O(|R|^\varepsilon))$ -approximation for undirected graphs with uniform edge lengths and uniform distance bounds. Charikar et al. [4] reuse this strategy for directed Steiner trees (without distance bounds) and obtain an $O(|R|^\varepsilon)$ -approximation algorithm, devising a particularly elegant analysis of recursive greedy.

Our algorithm uses five parameters, cf. Algorithm 1. The graph G , costs c , and lengths ℓ remain unchanged over all recursive calls to the procedure and are hence not explicitly included in these parameters. The algorithm operates in *levels* given by parameter $i \leq n$. The higher the level, the better the approximation guarantee. Parameters r , R , and d denote the root, the terminal set, and the vector of distance bounds, respectively. Parameter $k \leq |R|$ specifies the minimum number of terminals out of R , the resulting tree has to span (while meeting the distance bounds). Setting $k = |R|$, the algorithm outputs a feasible directed shallow-light Steiner tree.

Level $i = 1$ of the algorithm works as follows. For all terminals $t \in R$, the algorithm computes an r - t path P_t by $\text{MINCOSTPATH}(r, t, d(t))$. Clearly, P_t respects the length bound $d(t)$. The resulting tree consists of the union of the k cheapest (w.r.t. c) of these paths.³

For $i > 1$ we employ a greedy strategy to obtain a feasible solution T . Let the *relative cost* of a tree T' spanning k' terminals be defined as $\varrho(T') := c(T')/k'$. Starting with empty T , we iteratively compute a subtree T_{best} of low relative cost $\varrho(T_{\text{best}})$, add it to T , remove the newly spanned terminals from R , and adjust k accordingly.

In order to compute T_{best} , the algorithm exhaustively tests all nodes v and all values $k' \leq k$ to compute a cheap tree T' rooted at v that spans at least k' terminals. (Note, that k is adjusted by the algorithm.) These trees T' are computed by applying the algorithm recursively but for level $i - 1$. To obtain an r -rooted tree we connect r to v by a path P . This requires to adjust the distance bounds accordingly in the above mentioned recursive calls. An issue that arises here is that the necessary properties of path P are not clear a priori. In general, we may not be able to use the shortest path (w.r.t. ℓ) as this might be too expensive (w.r.t. c) to give a low relative cost.

To this end, we consider every possible path length up to $\ell(E)$, where the latter denotes the total length of all edges. This becomes tractable when we allow for a relative error of up to $(1 + \varepsilon)$: we evaluate a geometrically increasing sequence of length bounds $(1 + \varepsilon)^j$, for non-negative integrals j , and determine for each of these bounds the cheapest path P_j respecting it.

3.2 Analysis

Let $\mathcal{G} := (G, c, \ell, r, R, d)$ be a directed shallow-light Steiner tree problem instance as defined above. For the related problem of a *k -terminal directed shallow light Steiner tree (k -DSLST)* we are given an instance (\mathcal{G}, k) , $k \leq |R|$, and ask for the cheapest directed shallow light Steiner tree subject to any k -element subset of R . We observe that $k = |R|$ gives the original problem. An *$f(k)$ -partial approximation* for k -DSLST is a procedure that finds a tree T that is rooted at r , contains $1 \leq k' \leq k$ terminals of R , and has relative cost $\varrho(T) \leq f(k) \cdot c(T^*)/k$. Here, $c(T^*)$ is the cost of an optimum solution to k -DSLST.

We will show later (cf. Lemma 4) that the core of our algorithm in fact constitutes such a partial approximation. This allows us to adapt a lemma by Charikar et al. [4] to obtain an approximation to the original problem, as summarized in the following lemma. While their result is dealing with Steiner trees and does hence not consider length restrictions, their proof is versatile enough to be carried out in an identical fashion for our following situation: Let $\mathcal{P}(\mathcal{G}, k)$ be a partial approximation routine. We construct an approximation algorithm $\mathcal{A}(\mathcal{G}, k)$ as follows: First, $\mathcal{A}(\mathcal{G}, k)$ calls $\mathcal{P}(\mathcal{G}, k)$ which yields a tree T' spanning some terminals R' . If $|R'| = k$, we are done. Otherwise, $\mathcal{A}(\mathcal{G}, k)$ returns the union of T' and the tree T'' resulting from $\mathcal{A}(\mathcal{G}'', k'')$ where \mathcal{G}'' is the problem instance with reduced terminal set $R \setminus R'$ and $k'' := k - |R'|$.

³ As a side note, observe that one may be tempted to assume that some of these paths may coincide in the beginning, thus giving rise to a branch node where the paths start to differ. We would hence, inadvertently, construct a tree with more than one level. We do not need to care about this issue: Firstly, in our cost computation (of the upper bound) we assume the worst case, i.e., that such common subpaths do not exist; if they would, the cost would only decrease, thus improving the approximative solution. Secondly, we can always (implicitly) consider the metric closure of G (with multiedges for different length-vs.-cost combinations); in this case we always find distinct paths.

Algorithm 1 Approximation of a directed shallow-light Steiner tree for (G, c, ℓ, r, R, d)

```

1: procedure SHALLOWLIGHT( $i, r, R, d, k$ )
2:   if no  $k$  terminals in  $R$  respect the distance bounds from  $r$  then
3:     return  $\emptyset$ 
4:   if  $i = 1$  then
5:     for each terminal  $t \in R$  do
6:        $P_t \leftarrow \text{MINCOSTPATH}(r, t, d(t))$ 
7:     let  $R'$  be the set of  $k$  terminals with minimum  $c(P_t)$ 
8:     return  $\bigcup_{t \in R'} P_t$ 
9:    $T \leftarrow \emptyset$ 
10:  while  $k > 0$  do
11:     $T_{\text{best}} \leftarrow \emptyset$ 
12:    for each  $v \in V$  and each  $k', 1 \leq k' \leq k$  do
13:      for  $j = 0, \dots, \lceil \log_{1+\varepsilon} \ell(E) \rceil$  do
14:         $P_j \leftarrow \text{MINCOSTPATH}(r, v, (1+\varepsilon)^j)$ 
15:         $d'(u) \leftarrow d(u) - \frac{\ell(P_j)}{1+\varepsilon}$  for each  $u \in V$ 
16:         $T' \leftarrow \text{SHALLOWLIGHT}(i-1, v, R, d', k') \cup P_j$ 
17:        if  $\varrho(T_{\text{best}}) > \varrho(T')$  then  $T_{\text{best}} \leftarrow T'$ 
18:       $T \leftarrow T \cup T_{\text{best}}$ 
19:       $k \leftarrow k - |R \cap V(T_{\text{best}})|$ 
20:       $R \leftarrow R - V(T_{\text{best}})$ 
21:  return  $T$ 

```

► **Lemma 3** (Adaptation of Charikar et al. [4]). *Given an $f(k)$ -partial approximation $\mathcal{P}(\mathcal{G}, k)$ and an algorithm $\mathcal{A}(\mathcal{G}, k)$ as described above. If $f(x)/x$ is a decreasing function in x , then \mathcal{A} is a $g(k)$ -approximation, with $g(k) = \int_0^k (f(x)/x) dx$.*

In the light of \mathcal{P} and \mathcal{A} , the identification of T_{best} in Algorithm 1 corresponds to \mathcal{P} while the outer while loop resembles \mathcal{A} . It remains to show that our algorithm meets the criteria of an $f(k)$ -partial approximation with $f(x)/x$ being a decreasing function. At its core, the proof strategy is similar to Charikar et al., but we have to carefully consider our length restrictions and violations within the recursion.

► **Lemma 4.** *Consider SHALLOWLIGHT(i, r, R, d, k) (Alg. 1), which iteratively computes T . Let $\bar{T} := T_{\text{best}}$ be any tree incorporated in the current solution (line 18). It violates the length bounds by a factor of at most $(1 + \varepsilon)$. For $i \geq 2$, \bar{T} 's relative cost $\varrho(\bar{T})$ is at most $(i - 1)$ times the relative cost $\varrho^* := \varrho_{\bar{R}, \bar{k}}^*$ of the optimum solution $T^* := T_{\bar{R}, \bar{k}}^*$ to \bar{k} -DSLST with i levels, where \bar{R} and \bar{k} are the values for R and k currently used by the algorithm, respectively.*

Proof. Observe that, for $i > 1$, \bar{T} consists of an r - v path \bar{P} and a tree (computed recursively) with at most $i - 1$ levels rooted at v . We prove the lemma by induction on i .

First consider the length property of \bar{T} . For $i = 1$, it trivially holds by the direct application of the FPTAS (line 6). For $i \geq 2$, we can bound the length of \bar{P} by $(1 + \varepsilon)^j < \ell(\bar{P}) \leq (1 + \varepsilon)^{j+1}$. By line 15, the permissible length for a connection from v to some node u in $\bar{T} \setminus \bar{P}$ is bounded by $d'(u) \leq d(u) - (1 + \varepsilon)^j$. By induction, we will violate this bound by a factor of at most $(1 + \varepsilon)$, i.e., the length of a connection between r and u in \bar{T} will be at most $(1 + \varepsilon)^{j+1} + (1 + \varepsilon)(d(u) - (1 + \varepsilon)^j) = (1 + \varepsilon)d(u)$.

Now, consider the cost property. It holds for $i = 2$. Assume $i \geq 3$ and that the claim holds for all level restrictions less than i . Let v denote a *level-child* of r with respect to T^* , i.e., all inner nodes of the path $P_{j,v}$ between r and v in T^* are of degree 2. The subtree $T_v \subset T^*$ rooted at v has (at most) $i - 1$ levels. (By augmenting G with sufficient 0-cost 0-length edges, we can assume that T_v^* has precisely $i - 1$ levels.) Let $c_{j,v}$ and $\ell_v \leq (1 + \varepsilon)^j$ denote the cost and length of $P_{j,v}$, respectively. Let C_v denote the cost of T_v and k_v the number of terminals in T_v . In the following, consider the node v^* , level-child of r in T^* , with minimal $\varrho_{v^*} := (c_{j,v^*} + C_{v^*})/k_{v^*} < \varrho^*$.

At some point at level i , our algorithm will also consider node v^* and number k_{v^*} . The computed r - v^* path may be up to $(1 + \varepsilon)\ell_{v^*} \leq (1 + \varepsilon)^{j+1}$ long. We investigate the behavior of $\text{SHALLOWLIGHT}(i - 1, v^*, R, d', k_{v^*})$. It returns an $(i - 1)$ -level tree S that is, again, iteratively constructed. Let S' be the tree incorporated into S by the algorithm such that the current S now contains at least $k_{v^*}/(i - 1)$ terminals for the first time. Let S_0, S_1 be the solution trees before and after adding S' , respectively. Furthermore, let s_0, s_1 be the number of \bar{R} -nodes covered by S_0, S_1 , respectively. Observe that $s_1 \geq k_{v^*}/(i - 1)$.

Consider the nodes not covered before S' : $|T_{v^*} \cap \bar{R}| \geq k_{v^*} - s_0 = k_{v^*} - k_{v^*}/(i - 1) = \frac{i-2}{i-1}k_{v^*}$. Since we can cover all these nodes at cost at most C_{v^*} , we have an upper bound of $\frac{i-1}{i-2}C_{v^*}/k_{v^*}$ on the relative cost for the uncovered terminals. By our induction hypothesis, we know that we will hence find a solution—violating the length restrictions by at most a factor of $(1 + \varepsilon)$ —with relative cost at most $(i - 2)\frac{i-1}{i-2}C_{v^*}/k_{v^*}$ for S' . This upper bound naturally holds for each subtree that is incorporated into S before S' . Consequently, the relative cost of S_1 is also at most $(i - 1)C_{v^*}/k_{v^*}$.

Observe that our algorithm will not only compute $\text{SHALLOWLIGHT}(i - 1, v^*, R, d', k_{v^*})$ but also $\text{SHALLOWLIGHT}(i - 1, v^*, R, d', s_1)$. Observe the equally modified length restrictions d' . In the latter case, the algorithm will stop after adding S' to S , returning this S as its $(i - 1)$ -level solution tree of relative cost $\varrho(S) \leq (i - 2)C_{v^*}/k_{v^*}$. On level i , this S will be joined with the computed path \bar{P} of cost at most that of P_{j,v^*} (with corresponding j) and violating the length constraints by at most $(1 + \varepsilon)$ as discussed above. Together, they form a tree T' with $\varrho(T') = \varrho(S) + c_{j,v^*}/s_1 \leq (i - 2)C_{v^*}/k_{v^*} + c_{j,v^*}/(k_{v^*}/(i - 1)) \leq (i - 1)(c_{j,v^*} + C_{v^*})/k_{v^*} = (i - 1)\varrho_{v^*} = (i - 1)\varrho^*$. \blacktriangleleft

We are now able to prove the approximation result for directed shallow-light Steiner trees.

► **Theorem 5 (Revisited).** *The above algorithm is a bi-criteria $(1 + \varepsilon_1, O(|R|^{\varepsilon_2}))$ -approximation for directed shallow-light Steiner trees: for arbitrary small $\varepsilon_1, \varepsilon_2 > 0$, it gives a solution at most $O(|R|^{\varepsilon_2})$ times more expensive than the optimum, while violating the length constraints by a factor of at most $(1 + \varepsilon_1)$. For fixed ε_2 , its runtime is polynomial in the input size and ε_1 .*

Proof. Lemma 4 shows that each chosen T_{best} on level i has a relative cost of at most $(i - 1)$ the relative-cost-optimum i -level tree w.r.t. \bar{R}, \bar{k} . By Lemma 2, the latter approximates the optimum tree without level restrictions. So, overall, each T_{best} is a $(i - 1)2i(\bar{k}/2)^{1/i}$ -partial approximation for k -DSLST. By Lemma 3, this gives a $g(k)$ -approximation for k -DSLST with

$$g(k) = \int_0^k \left((i - 1)2i(\bar{x}/2)^{1/i}/x \right) dx = \frac{2i^2(i - 1)}{2^{1/i}} k^{1/i}.$$

We hence have an $O(|R|^{\varepsilon_2})$ -approximation for directed shallow-light Steiner trees ($=|R|$ -DSLST)—w.r.t. violating the length bounds by at most a factor of $(1 + \varepsilon_1)$ —by choosing a suitable i inversely correlated to ε_2 .

Consider the running time of our algorithm. MINCOSTPATH is an FPTAS with running time $O(mn/\varepsilon_1)$ [7]. Consider any call to SHALLOWLIGHT w.r.t. some i, k . For $i = 1$, it requires $O(|R|nm/\varepsilon_1)$ time. Otherwise, we may add $O(k)$ different trees T_{best} and the block of lines 14–17 is repeated $O(nk^2 \log \ell(E))$ times. Overall, any run of the procedure (disregarding recursive calls) requires $O(n^2mk^2 \log \ell(E)/\varepsilon_1)$ time. For overall i levels, there are $O(n^{i-1}k^{2i-2})$ recursive invocations, inducing an overall runtime of $O(n^{i+1}mk^{2i} \log \ell(E)/\varepsilon_1)$. Clearly, $\log \ell(E)$, the logarithm of the sum of all edge lengths, is polynomially bounded by the input size, and, by choice of i above, i is directly correlated to (and only dependent on) $1/\varepsilon_2$. ◀

4 Conclusions: Light-Weight Directed Spanners

We conclude with sketching another application of our shallow-light Steiner tree result. We obtain a bi-criteria approximation algorithm for light-weight directed α -spanners (cf. Definition 5). To the best of our knowledge no non-trivial result is known for this problem.

We employ a two-stage approach similar to the one used for directed sparse spanners [5, 2] and for our network design problem in Section 2. Thin and thick pairs are defined analogously to Section 2. Thin pairs can be settled as in [2] as only the linearity of the objective function is used there. For settling thick pairs, a set of $\Theta(\sqrt{n} \log n)$ many nodes is sampled. In the case of sparse spanners [2] it is sufficient to compute a shortest path in-tree and a shortest path out-tree for each of these sampled nodes, and take the union of these trees. Since each of these trees has at most $n - 1$ edges, which is clearly a lower bound on OPT, the total cost for this stage is $\tilde{O}(\sqrt{n} \cdot \text{OPT})$. It is shown that this procedure settles all thick pairs with high probability. In the case of light-weight spanners we compute a directed shallow-light spanning tree for each sampled node. More precisely, let u be the sampled node. We compute a shallow-light spanning tree T rooted at u such that for each node $v \in V$ its distance $\bar{\ell}_T(u, v)$ is at most $\alpha \cdot \bar{\ell}_G(u, v)$. Since the optimum solution to the spanner problem ensures the existence of a feasible solution to this problem, we can compute such a tree of cost at most $O(n^\varepsilon \text{OPT})$ using Theorem 4. Analogously, we can compute an in-tree with root u and the respective distance bounds. The total cost of the union of all such spanning trees is $O(n^{1/2+\varepsilon} \text{OPT})$.

Unfortunately, the resulting solution is not necessarily feasible since the stretch factor α may be violated. We can still argue that the solution gives a bi-criteria approximation with bounded stretch factor. To see this, consider a thick pair (u, v) and assume that we sample a node z such that there is a u - v path visiting z of length at most $\alpha \cdot \bar{\ell}_G(u, v)$. Hence $\bar{\ell}_G(u, z) + \bar{\ell}_G(z, v) \leq \alpha \bar{\ell}_G(u, v)$. Using the paths provided by the shallow-light in-tree and the shallow-light out-tree computed by our algorithm we can find a path of length at most $(\alpha + \varepsilon)\alpha \bar{\ell}_G(u, z) + (\alpha + \varepsilon)\alpha \bar{\ell}_G(z, v) \leq (\alpha + \varepsilon)\alpha \bar{\ell}_G(u, v)$ in our output graph. We have:

► **Theorem 6 (Revisited).** *The above algorithm is a bi-criteria $(\alpha + \varepsilon, O(n^{1/2+\varepsilon}))$ -approximation for light-weight directed α -spanners. The running time depends on n and ε and is polynomial in n for any fixed $\varepsilon > 0$.*

References

- 1 Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993.
- 2 Piotr Berman, Arnab Bhattacharyya, Konstantin Makarychev, Sofya Raskhodnikova, and Grigory Yaroslavtsev. Approximation algorithms for spanner problems and directed steiner forest. *Inf. Comput.*, 222:93–107, 2013.

- 3 Arnab Bhattacharyya, Elena Grigorescu, Kyomin Jung, Sofya Raskhodnikova, and David P. Woodruff. Transitive-closure spanners. *SIAM J. Comput.*, 41(6):1380–1425, 2012.
- 4 Moses Charikar, Chandra Chekuri, To-Yat Cheung, Zuo Dai, Ashish Goel, Sudipto Guha, and Ming Li. Approximation algorithms for directed steiner problems. *J. Algorithms*, 33(1):73–91, 1999. (preliminary version appeared at SODA’98).
- 5 Michael Dinitz and Robert Krauthgamer. Directed spanners via flow-based linear programs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing (STOC’11)*, pages 323–332, 2011.
- 6 Yevgeniy Dodis and Sanjeev Khanna. Designing networks with bounded pairwise distance. In *Proc. 21st Ann. ACM Symposium on Theory of Computing (STOC’99)*, pages 750–759, 1999.
- 7 Funda Ergun, Rakesh Sinha, and Lisa Zhang. An improved FPTAS for restricted shortest path. *Information Processing Letters*, 83:287–291, 2002.
- 8 Moran Feldman, Guy Kortsarz, and Zeev Nutov. Improved approximation algorithms for directed steiner forest. *J. Comput. Syst. Sci.*, 78(1):279–292, 2012.
- 9 Refael Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17(1):36–42, 1992.
- 10 C. S. Helvig, G. Robins, and A. Zelikovsky. An improved approximation scheme for the group steiner problem. *Networks*, 37(1):8–20, 2001.
- 11 Guy Kortsarz and David Peleg. Approximating shallow-light trees. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA ’97)*, pages 103–110, 1997.
- 12 Joseph Naor and Baruch Schieber. Improved approximations for shallow-light spanning trees. In *38th Annual Symposium on Foundations of Computer Science (FOCS’97)*, pages 536–541, 1997.
- 13 David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.
- 14 A. Zelikovsky. A series of approximation algorithms for the acyclic directed steiner tree problem. *Algorithmica*, 18:99–110, 1997.

Combinatorial Expressions and Lower Bounds

Thomas Colcombet and Amaldev Manuel

LIAFA, Université Paris-Diderot

{thomas.colcombet, amal}@liafa.univ-paris-diderot.fr

Abstract

A new paradigm, called combinatorial expressions, for computing functions expressing properties over infinite domains is introduced. The main result is a generic technique, for showing indefinability of certain functions by the expressions, which uses a result, namely Hales-Jewett theorem, from Ramsey theory. An application of the technique for proving inexpressibility results for logics on metafinite structures is given. Some extensions and normal forms are also presented.

1998 ACM Subject Classification F.1.1 Models of Computation, G.2.1 Combinatorics

Keywords and phrases expressions, lower bound, indefinability

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.249

1 Introduction

In this paper, we study the computational power of parallel devices that have an unlimited access to Boolean computations, as well as access to an infinite domain of ‘data’ (for instance integers or positive integers in what follows) under the restriction of a limited ‘bandwidth’. This limitation formally states that only a bounded number of data can be manipulated simultaneously. This is in contrast to the operations over Boolean values that have unlimited input size.

A typical model of this form consists of finite circuits—we call them *combinatorial expressions* throughout the paper—, of bounded depth, in which gates are of two kinds: gates with unbounded domain using arbitrary operations of fan-in at most two (for instance the binary gcd, the binary sum, product, or even non-computable functions), and gates using inputs ranging over a finite domain with unrestricted fan-in (for instance disjunctions, conjunctions of unbounded fan-in, or majority gates).

We use these devices for studying problems that have sequences of data as input. Typical such problems are:

- does a sequence of positive integers (data) have a gcd of one?
- does a sequence of integers sum to zero?
- are all the integers in a sequence distinct?

The motivation of the authors in studying such devices arose from proofs of lower bounds for logics over data-words. The essence of these lower bounds can be easily captured by reduction to lower bounds over combinatorial expressions. Independently of this motivation, we believe that the objects presented here deserve a study on their own. We chose to include here a simpler application to indefinability results in metafinite model theory.

Contributions Our contributions concerning such models go in several directions.

- We introduce the model of combinatorial expressions, show some normal forms for them.
- We prove indefinability results for these expressions: Indefinability of functions (i.e., maps from tuples of data to data) using the pigeonhole principle, and indefinability of problems



© Thomas Colcombet and Amaldev Manuel;
licensed under Creative Commons License CC-BY
32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).
Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 249–261



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



SYMPOSIUM
ON THEORETICAL
ASPECTS
OF COMPUTER
SCIENCE

- (i.e., maps of tuples of data to $\{0,1\}$) using reductions to problems of ‘window definability’ (see immediately below).
- We introduce the questions of ‘window definability’. Window definable properties are properties that can be described as Boolean combinations of properties over subsets of the inputs (namely windows). Using combinatorial arguments from Ramsey theory, namely Hales-Jewett theorem, we show that some problems are window undefinable.
 - We study the added expressive power when expressions are furthermore allowed to use selection gates.
 - Finally, we apply these techniques for proving some undefinability results over metafinite structures (these are finite structures, in which tuples can take values in some fixed infinite domain).

Related works This work is of course related to circuit complexity (see for instance [6]), and more precisely to families of circuits of bounded depth, since the object we are manipulating can be seen as circuits. However, the expressive power of the models that we study is very different. Indeed, not only, our combinatorial expressions can deal with data ranging over an infinite domain, but furthermore, even Boolean gates are not restricted to a simple families like Boolean connectives.

It was brought to our attention that our lower bound result is related to a result of Pascal Tesson stating that testing whether k subsets of $[n]$ form a partition requires a non-constant communication complexity in the k -party ‘input on the forehead’ model [7, 8]. The two results also make an analogous use of the Hales-Jewett theorem.

There are other families of machines that have been extended to infinite domains, this is in particular the subject of study of algebraic complexity (in particular [1, 5], or [9] for circuits). However, the principle of these branches of work is to see how allowing machines to have primitive capabilities to perform computations in a (infinite) field changes their expressive power. The fact that the infinite domain is equipped of an algebraic (field) structure, changes radically the expressive power, and in particular relates this branch of research to the study of polynomials.

Organisation of the paper In Section 2, we present combinatorial expressions and some motivating examples, which is then followed by a normal form theorem for the expressions and a simple undefinability result. Section 3 contains the main contribution of this paper, namely the undefinability results using the Hales-Jewett theorem. In Section 4, an extended class of expressions is presented and a normal form theorem is given for this class. Section 5 presents an application of the undefinability result to metafinite logics. In Section 6 we discuss some interesting directions for future work and conclude.

Acknowledgments The Hales-Jewett theorem was brought to our attention by Srikanth Srinivasan, and the link to multiparty communication complexity by Frederic Magniez.

2 Combinatorial expressions and normal form

The aim of this section is to introduce the objects of our study, namely *combinatorial expressions*. As usual, \mathbb{Z} (*resp.* \mathbb{N}) is the set of (*resp.* non-negative) integers, and $[n]$ denotes the set $\{1, \dots, n\}$.

2.1 Combinatorial expressions

Combinatorial expressions are built by composing partial maps over a *data domain* \mathcal{D} which is an infinite set. Typical instances of data domains are integers (\mathbb{Z}), natural numbers (\mathbb{N}), words over a finite alphabet (A^* where A is a finite alphabet) etc. A variable X has *range* $E \subseteq \mathcal{D}$, abbreviated as $X : E$, if the set of values over which it ranges is E . We assume an infinite supply of variables for each range E . A map $f : E_1 \times \cdots \times E_k \rightarrow F$, where $E_1, \dots, E_k, F \subseteq \mathcal{D}$, has *arity* k , *domain* $E_1 \times \cdots \times E_k$ and *range* F . The *image* of the map f is the set of values in F that f maps to, i.e. the set $\{f(a_1, \dots, a_k) \mid a_1 \in E_1, \dots, a_k \in E_k\}$. The expressions are built using two specific classes of functions, namely:

- *binary functions* — when $k = 2$, and,
- *finitary functions* — when each of E_1, \dots, E_k is finite.

A binary function has a bounded arity but may have an unbounded input domain, for example the usual addition on naturals $+$: $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ is binary. On the other hand a finitary function has a finite input domain, but no restriction on the arity. An example of a finitary function is the Boolean conjunction over k inputs $\bigwedge_k : \{0, 1\}^k \rightarrow \{0, 1\}$. Now we formally define combinatorial expressions.

► **Definition 1.** *Combinatorial expressions* are defined inductively;

- a variable $X : E$ is a combinatorial expression with range E , and,
- if $f : E_1 \times \cdots \times E_k \rightarrow F$ is a binary or finitary function, and t_1, \dots, t_k are combinatorial expressions with ranges E_1, \dots, E_k respectively, then $f(t_1, \dots, t_k)$ is a combinatorial expression with range F .

Let t be a combinatorial expression that contains (possibly vacuously) the variables $\bar{X} = X_1 : E_1, \dots, X_n : E_n$. We indicate the variables of t by the notation $t(\bar{X})$. For the valuation $\bar{a} = a_1, \dots, a_n, a_i \in E_i$ of the variables \bar{X} , the value of the expression t , denoted as $t(\bar{a})$, is defined in the obvious way; if t is a variable X_i then $t(\bar{a}) = a_i$, and if $t = f(t_1, \dots, t_k)$ then $t(\bar{a}) = f(t_1(\bar{a}), \dots, t_k(\bar{a}))$. Assume $F \subseteq \mathbb{N}$ is the range of the expression t . Naturally t defines a map $t : \bar{a} \rightarrow t(\bar{a})$ from the set $E_1 \times \cdots \times E_n$ to the set F . Like in the case of functions, the *image* of the term t is the set of *output* values of t , i.e. the set $\{t(\bar{a}) \mid \bar{a} \in E_1 \times \cdots \times E_n\}$. Given a map $m : \mathcal{D}^n \rightarrow \mathcal{D}$ we say the map is *realised* by an expression t if t defines the map m .

Next we introduce the notion of *depth* of an expression. For a variable X the depth is 0. For an expression $f(t_1, \dots, t_k)$ the depth is 1 more than the maximum of depths of t_1, \dots, t_k .

► **Definition 2 (Family of combinatorial expressions).** Fix a sequence $X_1 : \mathcal{D}, X_2 : \mathcal{D}, \dots$ of variables. A family of combinatorial expressions is a sequence of expressions $(t_n)_{n \in \mathbb{N}} = t_1, t_2, \dots$ where t_n is an expression over the variables X_1, \dots, X_n .

A family of combinatorial expressions defines a map $(t_n)_{n \in \mathbb{N}} : a_1, \dots, a_n \rightarrow t_n(a_1, \dots, a_n)$ from \mathcal{D}^* (all finite sequences over \mathcal{D}) to \mathcal{D} . The family $(t_n)_{n \in \mathbb{N}}$ is of *constant depth* if there is a $k \in \mathbb{N}$ such that each expression t_n is of depth at most k .

Given a map $m : \mathcal{D}^* \rightarrow \mathcal{D}$, we say m is *realisable* (by a constant depth family) if there is a family of combinatorial expressions (of constant depth) $(t_n)_{n \in \mathbb{N}}$ that defines m . A particular case is when the range of the map m is restricted to a set of size two, without loss of generality we assume it is $\{0, 1\}$; in this case we say $(t_n)_{n \in \mathbb{N}}$ realises the *property* (or *problem*) $\{a_1, \dots, a_n : m(a_1, \dots, a_n) = 1\}$.

► **Example 3.** Some examples of combinatorial expressions and families are given below. We take the domain \mathcal{D} to be the set of natural numbers \mathbb{N} .

1. Fix a number $k \in \mathbb{N}$. Let $m : \mathbb{N}^* \rightarrow \{0, \dots, k\}$ be the map $m : a_1, \dots, a_n \rightarrow (\sum_i a_i) \bmod k$. The map m is realised by the family $(f_n(g(X_1), \dots, g(X_n)))_{n \in \mathbb{N}}$, where $f_n : \{0, \dots, k\}^n \rightarrow \{0, \dots, k\}$ is the finitary function $f_n : a_1, \dots, a_n \rightarrow (\sum_i a_i) \bmod k$, and $g : \mathbb{N} \rightarrow \{0, \dots, k\}$ is the binary function (by abuse of notation) $g : i \rightarrow i \bmod k$. This family has depth 2.
2. Let P_1 be the set of all finite sequences of non-zero naturals. This property is realised by the family $(\bigwedge_n(\text{zero}(X_1), \dots, \text{zero}(X_n)))_{n \in \mathbb{N}}$, where $\text{zero} : \mathbb{N} \rightarrow \{0, 1\}$ is the binary function that maps precisely all the non-zero naturals to 1, and \bigwedge_n is the finitary function that defines the Boolean conjunction on n inputs. This family has depth 2.
3. Let P_2 be the property $\{a_1, \dots, a_n \in \mathbb{N}^* : a_i \neq a_j\}$. Let $\text{neq} : \mathbb{N} \times \mathbb{N} \rightarrow \{0, 1\}$ be the binary function that has value 1 precisely when the inputs differ. Then, the property P_2 is realised by the family (of depth 2) $(\bigwedge_{n \in \mathbb{N}} (\bigwedge_{(i,j) \in \binom{[n]}{2}} \text{neq}(X_i, X_j)))_{n \in \mathbb{N}}$ where the expression $t_{ij} = \text{neq}(X_i, X_j)$.
4. We claim that for any map $m : \mathbb{N}^* \rightarrow \mathbb{N}$ there is a family $(t_n)_{n \in \mathbb{N}}$ of *logarithmic depth* (i.e. the expression t_n has depth at most $\log n$) realising it. Let p_1, p_2, \dots be an enumeration of the prime numbers. For each prime p_i , let $p_i\text{-exp} : x \rightarrow p_i^x$ be the exponential function with base p_i . Define the binary function $u_n : \mathbb{N} \rightarrow \mathbb{N}$ as

$$u_n(x) = m(a_1, \dots, a_n) \text{ where } a_i \text{ is the exponent of } p_i \text{ in } x$$

Finally let $\pi_n(X_1, \dots, X_n)$ be the expression (of $\log n$ -depth) that computes the product of the variables X_1, \dots, X_n . Then, the family of expressions

$$(u_n(\pi_n(p_1\text{-exp}(X_1), \dots, p_n\text{-exp}(X_n))))_{n \in \mathbb{N}}$$

realises the map m .

Example 3.4 implies that the class of maps realised by families of expressions of logarithmic depth is degenerate, i.e. every map is realisable. Hence for interesting results one has to consider the class of families of sub-logarithmic depth. In the following we study the class of families of expressions of constant depth and prove that it is non-degenerate, i.e. there are maps and properties that are not realisable.

2.2 Normal form and definability

In the rest of the section we exhibit a normal form for the expressions. First we introduce the important notion of semantic equivalence of expressions.

► **Definition 4** (Equivalence of expressions). Two expressions $t_1(\bar{X})$ and $t_2(\bar{X})$ over the variables $\bar{X} = X_1 : E_1, \dots, X_n : E_n$ are *equivalent* if $t_1(\bar{a}) = t_2(\bar{a})$ for all $\bar{a} = a_1, \dots, a_n$, $a_i \in E_i$.

We introduce some notation. For an expression t , we denote the range and image of t by $\text{range}(t)$ and $\text{image}(t)$ respectively. Assume $\bar{t} = t_1, \dots, t_n$ is a finite sequence of expressions. We define $\text{len}(\bar{t}) = n$ and $\text{range}(\bar{t}) = \text{range}(t_1) \times \dots \times \text{range}(t_n)$. If $\bar{s} = s_1, \dots, s_m$ and $\bar{t} = t_1, \dots, t_n$ are two sequences of expressions, then \bar{s}, \bar{t} denotes the sequence $s_1, \dots, s_m, t_1, \dots, t_n$. An expression t is a *binary expression* if it consists only of binary functions.

The normal form theorem is obtained by transforming the expressions and for that we use the idea of pairing, i.e. encoding pairs of elements from \mathcal{D} as an element in \mathcal{D} . We use the following fact from set theory.

► **Fact 1** (See [4], Chapter 3). *If A is an infinite set, there exists an injective map from $A \times A$ to A .*

Fix an injective map π from $\mathcal{D} \times \mathcal{D}$ to \mathcal{D} . For elements $a_1, a_2 \in \mathcal{D}$ we let $\langle a_1, a_2 \rangle$ denote the element $\pi(a_1, a_2) \in \mathcal{D}$. Similarly for subsets $E_1, E_2 \in \mathcal{D}$ we let $\langle E_1, E_2 \rangle$ denote the set $\{\langle a_1, a_2 \rangle \mid a_1 \in E_1, a_2 \in E_2\}$.

► **Theorem 5 (Normal form).** *For every expression $t(\bar{X})$ of depth $\ell \in \mathbb{N}$ there exists an equivalent expression of the form*

$$b(r(\bar{X}), f(\bar{s}(\bar{X}))) \tag{1}$$

where b is a binary function, f is a finitary function, $r(\bar{X})$ is a binary expression of depth at most ℓ , and $\bar{s}(\bar{X})$ is a sequence of binary expressions of depth at most ℓ . Moreover, if the image of $t(\bar{X})$ is finite then there exists an equivalent expression of the form

$$f(\bar{s}(\bar{X})) \tag{2}$$

where f is a finitary function and $\bar{s}(\bar{X})$ is a sequence of binary expressions of depth at most ℓ .

The normal form theorem allows us to formulate arguments about the expressive power of our model. We next give such an application.

► **Proposition 1.** *Let $m : \mathcal{D}^{2^k+1} \rightarrow \mathcal{D}$ be a map that satisfies the following property: (†) For any index $i \in [2^k + 1]$ and any $n \in \mathbb{N}$ there exist values $a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_{2^k+1} \in \mathcal{D}$ such that the set $\{m(a_1, \dots, a_{i-1}, b, a_{i+1}, \dots, a_{2^k+1}) \mid b \in \mathcal{D}\}$ is of size at least n . Then the map m is not realisable by an expression of depth at most k .*

Proof. The proof is an application of the pigeonhole principle along with the normal form theorem. Let m be a map that satisfies the Property (†).

Let $\bar{X} = X_1, \dots, X_{2^k+1}$ be the input variables with range \mathcal{D} . Assume there is an expression $t(\bar{X})$ of depth k realising the map m . Using the normal form theorem we obtain an equivalent expression $t'(\bar{X})$ of the form

$$b(r(\bar{X}), f(\bar{s}(\bar{X})))$$

where b is a binary, f is finitary, $r(\bar{X})$ is a binary expression of depth at most k and $\bar{s}(\bar{X})$ is a sequence of binary expressions of depth at most k . Let the image of the function f be of size n . Since the binary expression $r(\bar{X})$ has depth at most k , there is a variable X_i that is not used by the expression $r(\bar{X})$. Consider the following set of input tuples;

$$\begin{array}{cccccccc} a_1, & \dots, & a_{i-1}, & b_1, & a_{i+1}, & \dots & a_{2^k+1} \\ a_1, & \dots, & a_{i-1}, & b_2, & a_{i+1}, & \dots & a_{2^k+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_1, & \dots, & a_{i-1}, & b_m, & a_{i+1}, & \dots & a_{2^k+1} \end{array}$$

where $a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_{2^k+1} \in \mathcal{D}$ and $b_1, b_2, \dots, b_m \in \mathcal{D}$, $m > n$ are such that

$$|\{m(a_1, \dots, a_{i-1}, b_i, a_{i+1}, \dots, a_{2^k+1}) \mid i \in [m]\}| > n$$

and for each $j \neq l$ it is the case that

$$m(a_1, \dots, a_{i-1}, b_j, a_{i+1}, \dots, a_{2^k+1}) \neq m(a_1, \dots, a_{i-1}, b_l, a_{i+1}, \dots, a_{2^k+1}) .$$

Existence of $a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_{2^k+1} \in \mathcal{D}$ and $b_1, b_2, \dots, b_m \in \mathcal{D}$ are guaranteed by the Property (†). The tuples differ only on the input variable X_i . Hence on all these inputs the

binary expression $r(\bar{X})$ has the same output. Moreover, since f has a finite image of size n and the number of input tuples is more than n , by pigeonhole principle there exist two input tuples on which f has the same output. Let them be

$$\begin{matrix} a_1, & \dots, & a_{i-1}, & b_{j_1}, & a_{i+1}, & \dots & a_{2^k+1}, \\ a_1, & \dots, & a_{i-1}, & b_{j_2}, & a_{i+1}, & \dots & a_{2^k+1}. \end{matrix}$$

It follows that on these two inputs the expressions $r(\bar{X})$ and $f(\bar{s}(\bar{X}))$ have the same output and hence the function b also has the same output. But clearly the map m differs on these inputs which is a contradiction. Hence the claim is established. ◀

► **Corollary 6.** *The following maps are not realised by expressions of depth at most k .*

1. $gcd : (\mathbb{N} \setminus \{0\})^{2^k+1} \rightarrow \mathbb{N} \setminus \{0\}$ defined as $gcd : a_1, a_2, \dots, a_{2^k+1} \rightarrow gcd(a_1, a_2, \dots, a_{2^k+1})$,
2. $sum : \mathbb{Z}^{2^k+1} \rightarrow \mathbb{Z}$ defined as $sum : a_1, a_2, \dots, a_{2^k+1} \rightarrow a_1 + \dots + a_{2^k+1}$.

Proof. By virtue of the previous proposition, it is enough to establish the Property (†) for each of the maps m .

1. For each $i \in 2^k + 1$ and $n \in \mathbb{N}$, we define the following tuples ;

$$\begin{matrix} 2^{n+1}, & \dots, & 2^{n+1}, & 2, & 2^{n+1}, & \dots, & 2^{n+1} \\ 2^{n+1}, & \dots, & 2^{n+1}, & 2^2, & 2^{n+1}, & \dots, & 2^{n+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 2^{n+1}, & \dots, & 2^{n+1}, & 2^{n+1} & 2^{n+1}, & \dots, & 2^{n+1} \end{matrix}$$

$$\underbrace{\hspace{10em}}_{i-1 \text{ times}} \qquad \underbrace{\hspace{10em}}_{2^k + 1 - i \text{ times}}$$

It is straightforward to check that the image of the map gcd on these tuples is of size $n + 1$.

2. As before, for each $i \in 2^k + 1$ and $n \in \mathbb{N}$, we define the following tuples ;

$$\begin{matrix} 0, & \dots, & 0, & 1, & 0, & \dots, & 0 \\ 0, & \dots, & 0, & 2, & 0, & \dots, & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0, & \dots, & 0, & n+1 & 0, & \dots, & 0 \end{matrix}$$

$$\underbrace{\hspace{10em}}_{i-1 \text{ times}} \qquad \underbrace{\hspace{10em}}_{2^k + 1 - i \text{ times}}$$

It is easily verified that the image of the map sum on these tuples is of size $n + 1$. ◀

Before concluding this section let us note that the arguments we used in Proposition 1 does not work for proving indefinability of maps with a finite image, for instance the map $P_{gcd=1} : (\mathbb{N} \setminus \{0\})^* \rightarrow \{0, 1\}$ defined as $P_{gcd=1}(a_1, a_2, \dots, a_m) = 1$ iff $gcd(a_1, \dots, a_m) = 1$. In the next section we develop advanced techniques for handling such maps.

3 Window-definability and indefinability

In this section, we provide the necessary material for showing that some problems are not expressible by combinatorial expressions of bounded depth. This is different from the indefinability result that we have seen before, Proposition 1, which was dealing with the indefinability of functions that have an infinite/unbounded image while the maps we consider

in this section have an image of size 2. For this, we slightly depart from the above framework, and introduce the notion of window-definability.

In the following we use the notation A^B to denote the set of all vectors/sequences over A indexed by the set B . Let us fix a finite set of variables, $\mathcal{V} = \{X_1, \dots, X_k\}$ ranging over \mathcal{D} . A *window* (over \mathcal{V}) is a subset of \mathcal{V} . Given a valuation of the variables $\bar{a} \in \mathcal{D}^{\mathcal{V}}$, its restriction to a window W is denoted $\bar{a}|_W$. Two valuations v, v' are *W-equivalent* if $v|_W = v'|_W$, i.e., indistinguishable ‘through the window W ’. From now, \mathcal{W} designates a fixed set of windows. A problem $P \subseteq \mathcal{D}^{\mathcal{V}}$ is *W-definable* if it can be described as a Boolean combination of languages of the form

$$\{\bar{a} \in \mathcal{D}^{\mathcal{V}} \mid \bar{a}|_W \in S\} \quad \text{for some } S \subseteq \mathcal{D}^W.$$

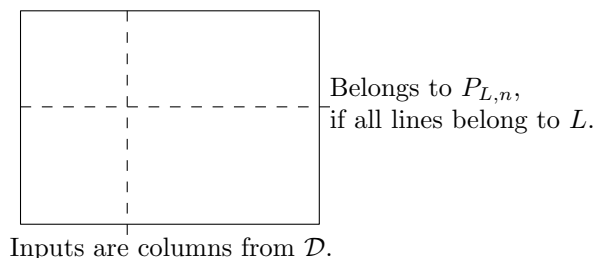
Such a Boolean combination is called a *W-definition*. In other words, the membership to P is entirely determined given finitely many properties of the input ‘seen through the windows’. Of course, if $\mathcal{V} \in \mathcal{W}$, then all problems are \mathcal{W} -definable. We are interested in understanding the notion of \mathcal{W} -definability when this is not the case (i.e. $\mathcal{V} \notin \mathcal{W}$). The *size of a W-definition* is the number of sets of the above form it uses. A problem is *W, k-definable* if there is a \mathcal{W} -definition for it of size at most k . Two problems P, R are *W, k-separable* if there is a \mathcal{W}, k -definable set D such that $P \cap D = \emptyset$ and $R \subseteq D$.

These notions are related to the above sections thanks to the following lemma:

► **Lemma 7.** *The problems definable by combinatorial expressions of depth k are \mathcal{W} -definable, where \mathcal{W} is the set of all windows of size at most 2^k .*

Proof. By the normal form theorem, any expression of depth k deciding a property (since it has a finite image) is equivalent to an expression of the form $f(\bar{s}(\bar{X}))$ where f is a finitary function, and $\bar{s}(\bar{X})$ is a sequence of binary expressions of depth at most k . First of all we observe that we can assume that every binary expression in $\bar{s}(\bar{X})$ outputs a Boolean value, in other words f computes a Boolean function. Notice that there is no loss of generality here since any finitary function can be converted to a Boolean function by increasing the number of inputs. Now the claim follows by observing that each binary expression $s_i(\bar{X})$ in $\bar{s}(\bar{X})$ corresponds to a set of the form $\{\bar{a} \in \mathcal{D}^{\mathcal{V}} \mid \bar{a}|_W \in S\}$ where the window W is of size at most 2^k (namely the variables used by the expression $s_i(\bar{X})$). ◀

We shall now head toward indefinability results. For this, we use an exact characterization of the definability for some special forms of problems: *rectangle problems*. In such problems, data are column vectors. Hence, tuples of data as input can be seen as rectangles. We are interested in relating the \mathcal{W} -definability of the set of rectangles such that every line belongs to some given set of valid rows L , to some simpler properties of L .



Formally, fix an alphabet A , and a *line property* $L \subseteq A^{\mathcal{V}}$. Then, given a positive integer n , consider the domain $\mathcal{D}_n = A^n$ (understood as ‘columns’), and define the problem $P_{L,n} \subseteq \mathcal{D}_n^{\mathcal{V}}$

consisting of these rectangles such that every line belongs to L :

$$P_{L,n} = \{\bar{a} \in \mathcal{D}_n^\mathcal{V} \mid \pi_i(\bar{a}) \in L \text{ for all } i \in [n]\},$$

where $\pi_i(a_1 \dots a_n) = a_i$ is extended component-wise to tuples indexed by \mathcal{V} .

Our Theorem 8, just below, relates the \mathcal{W} -definability of these problems to the property of L being \mathcal{W} -closed, that we define now.

An element $\bar{a} \in A^\mathcal{V}$ belongs to the \mathcal{W} -closure of $L \subseteq A^\mathcal{V}$, denoted $\bar{L}^\mathcal{W}$, if for all $W \in \mathcal{W}$ there is some $\bar{b} \in L$ such that \bar{a} and \bar{b} are W -equivalent. The set L is \mathcal{W} -closed if it is equal to its \mathcal{W} -closure.

The interesting examples are more the negative ones: consider for instance $A = \{0, 1\}$, $L \subseteq A^\mathcal{V}$ the set of tuples that contain at least one occurrence of 1, and \mathcal{W} to be $2^\mathcal{V} \setminus \mathcal{V}$, then L is not \mathcal{W} -closed since $\bar{a} = 0^\mathcal{V}$ does not belong to L , but for all windows $W \in \mathcal{W}$ we can define \bar{b} to be 0 over W and 1 elsewhere. This \bar{b} is W -equivalent to \bar{a} , and since it contains at least one occurrence of 1, it belongs to L .

► **Theorem 8.** *For all $L \subseteq A^\mathcal{V}$,*

- *if L is \mathcal{W} -closed then there is some k such that all $P_{L,n}$ has a \mathcal{W}, k -definition for all positive integers n , and*
- *if L is not \mathcal{W} -closed, then for all k , there exists n such that $P_{L,n}$ has no \mathcal{W}, k -definition.*

Proof of the first item. Assume that L is \mathcal{W} -closed, this means that:

$$L = \{\bar{a} \in A^\mathcal{V} \mid \bar{a}|_W \in L|_W \text{ for all } W \in \mathcal{W}\}, \quad \text{where } L|_W = \{\bar{a}|_W \mid \bar{a} \in L\}.$$

Consider now the set $R_{L,n} \subseteq (\mathcal{D}_n)^\mathcal{V}$ that contains $u \in (\mathcal{D}_n)^\mathcal{V}$ if $u|_W \in (L|_W)^n$ for all $W \in \mathcal{W}$. By definition, $R_{L,n}$ is $\mathcal{W}, |\mathcal{W}|$ -definable. Let us show that $R_{L,n} = P_{L,n}$.

Consider some $\bar{a} \in (\mathcal{D}_n)^\mathcal{V}$. Then $\bar{a} \in P_{L,n}$ if and only if for all $i \in [n]$ and all $W \in \mathcal{W}$, $\pi_i(\bar{a}|_W) \in L|_W$, if and only if for all $W \in \mathcal{W}$ and all $i \in [n]$, $\pi_i(\bar{a}|_W) \in L|_W$, if and only if $\bar{a} \in R_{L,n}$. Hence $P_{L,n} = R_{L,n}$ is $\mathcal{W}, |\mathcal{W}|$ -definable. ◀

Before being able to prove the second item, we need to introduce the deep combinatorial theorem of Hales-Jewett. In this theorem, a *combinatorial line* of B^n (for some finite set B and some positive integer n) is a set of the form $\ell = \{u[b] \mid b \in B\}$ for some $u \in (B^*x)^+B^*$, where $u[b]$ denotes u in which b has been substituted to all occurrences of x .

► **Theorem 9** (Hales-Jewett [3]). *Given some finite sets B and C , there is a positive integer n such that for all maps χ from B^n to C there exists a χ -monochromatic combinatorial line ℓ , i.e., there is $c \in C$ such that $\chi(v) = c$ for all $v \in \ell$.*

We now use a this theorem for establishing the second item of Theorem 8. We establish in fact the following stronger lemma.

► **Lemma 10.** *If $\bar{a} \in \bar{L}^\mathcal{W} \setminus L$, then for all k , there exists n such that $P_{L,n}$ and the set $P_{L \cup \{\bar{a}\},n} \setminus P_{L,n}$ cannot be \mathcal{W}, k -separated.*

Hence, one cannot separate ‘all lines are in L ’ from ‘all lines are in $L \cup \{\bar{a}\}$ and there is a line equal to \bar{a} ’. It is easy to see that Lemma 10 implies the second item of Theorem 8. Indeed, L being non \mathcal{W} -closed means that there exists $\bar{a} \in \bar{L}^\mathcal{W} \setminus L$. Assuming for the sake of contradiction that $P_{L,n}$ would be \mathcal{W}, k -definable would thus imply that $P_{L,n}$ and $P_{L \cup \{\bar{a}\},n} \setminus P_{L,n}$ would be \mathcal{W}, k -separated by $P_{L,n}$ itself. A contradiction.

Proof. Assume that $P_{L,n}$ and $Q_{\bar{a},n} = P_{L \cup \{\bar{a}\},n} \setminus P_{L,n}$ are \mathcal{W}, k -separable.

Our *first step* consists in showing that the language that separates $P_{L,n}$ from $Q_{\bar{a},n}$ can be derived from a coloring function χ of the inputs to some set C (the size of which does not depend on n). Let us assume that the \mathcal{W}, k -separability is witnessed by a Boolean combination of the sets $R_i = \{\bar{c} \in (\mathcal{D}_n)^\mathcal{V} \mid \bar{c}|_{W_i} \in S_i\}$ for $i \in [k]$ where $W_i \in \mathcal{W}$ and $S_i \subseteq (\mathcal{D}_n)^W$. Consider now the set $C = [2]^{[k]}$ (note that it does not depend on n), and the map χ from $(\mathcal{D}_n)^\mathcal{V}$ to C defined by

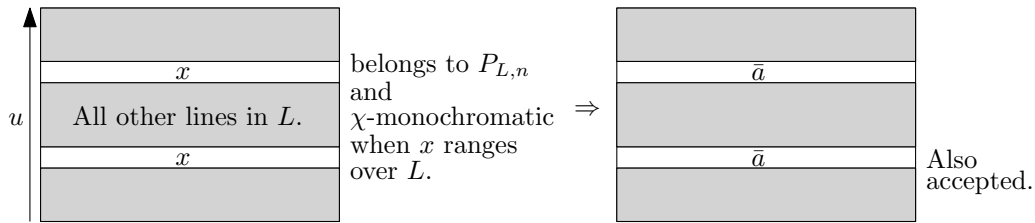
$$\text{for all } i \in [k] \text{ and } \bar{c} \in (\mathcal{D}_n)^\mathcal{V}, \quad \chi(\bar{c})_i = \delta_{\bar{c}|_{W_i} \in S_i} = \begin{cases} 1 & \text{if } \bar{c}|_{W_i} \in S_i, \\ 0 & \text{otherwise.} \end{cases}$$

This map stores all the relevant information concerning the membership in each of the R_i 's. The \mathcal{W}, k -separability means that whenever $\bar{c} \in P_{L,n}$ and $\bar{c}' \in Q_{\bar{a},n}$, $\chi(\bar{c}) \neq \chi(\bar{c}')$ (\star).

We shall now lay ground for the use of Hales-Jewett. This theorem will be used on rows: a rectangle of the problem will be seen as a sequence of rows, one on top of the previous one. This is different from the input itself, since each variable accounts for one column. We will allow to use these two points of view by implicitly identifying elements from $(A^\mathcal{V})^n$ (sequence of rows of length \mathcal{V}) with elements from $(A^n)^\mathcal{V} = (\mathcal{D}_n)^\mathcal{V}$ (sequence of columns of depth n). Under this identification, we can for instance write $L^n = P_{L,n}$, since $P_{L,n}$ consists of these inputs such that every line belongs to L .

We can now apply the Hales-Jewett theorem, using $B = L$ and C as defined during the first step, thus getting a number n . The first step provides us with a coloring χ from B^n to C . Finally, the theorem of Hales-Jewett states the existence of a χ -monochromatic combinatorial line $\ell = \{u[\bar{b}] \mid \bar{b} \in L\}$ for $u \in (L \cup \{x\})^n$, of color $c \in C$.

The principle of the rest of the proof is shown in the following picture:



Let us prove that $\chi(u[\bar{a}]) = c$ (This is a contradiction to (\star) since $u[\bar{a}] \in Q_{\bar{a},n}$, thus completing the proof).

For all $i \in [k]$, we have:

$$\begin{aligned} \chi(u[\bar{a}])_i &= \delta_{u[\bar{a}]|_{W_i} \in S_i} && \text{(by definition of } \chi) \\ &= \delta_{u[\bar{b}]|_{W_i} \in S_i} \quad \text{for some } \bar{b} \in L && \text{(since } \bar{a} \in \bar{L}^\mathcal{W}) \\ &= \chi(u[\bar{b}])_i && \text{(by definition of } \chi) \\ &= c_i. && \text{(since } u[\bar{b}] \in \ell \text{ and hence is mapped to } c \text{ by } \chi) \end{aligned}$$

Hence $\chi(u[\bar{a}]) = c$. ◀

We can now derive other indefinability results from Theorem 8.

► **Lemma 11.** Consider a set of windows \mathcal{W} such that $\mathcal{V} \notin \mathcal{W}$.

- For \mathcal{D} being the positive integers, then the set $P_{\text{gcd}=1}$ of inputs of gcd one is not \mathcal{W} -definable.
- For \mathcal{D} being the integers, the set $P_{\Sigma=0}$ of inputs of null sum is not \mathcal{W} -definable.

Proof of first item. Let k be a positive integer. Let us show that $P_{\text{gcd}=1}$ is not \mathcal{W}, k -definable.

Let $A = \{0, 1\}$ and $L = A^{\mathcal{V}} \setminus \{0\}^{\mathcal{V}}$. We have seen that this L was not \mathcal{W} -closed. Hence, according to Theorem 8, there is n such that the rectangle problem $P_{L,n}$ is not \mathcal{W}, k -definable. Let p_1, \dots, p_n be n distinct prime numbers. Define the map f from $\mathcal{D}_n = \{0, 1\}^n$ to positive integers by $f(a_1 \dots a_n) = p_1^{a_1} p_2^{a_2} \dots p_n^{a_n}$. Clearly, given an input tuple $\bar{a} \in (\mathcal{D}_n)^{\mathcal{V}}$, $\text{gcd}_{v \in \mathcal{V}} f(a(v))$ is 1 if and only if for all prime numbers $p = p_1, \dots, p_n$, $f(a(v))$ is not divisible by p for some $v \in \mathcal{V}$, or equivalently, if all lines in \bar{a} (seen as a rectangle) contains a 0. Hence $P_{L,n} = \{\bar{a} \mid \text{gcd}_{v \in \mathcal{V}} f(a(v))\}$.

Thus, assume that $P_{\text{gcd}=1}$ would be \mathcal{W}, k -definable, then the problem $P_{L,n}$ would also be \mathcal{W}, k -definable. This is a contradiction. ◀

Proof of second item. The approach is similar. Let us assume without loss of generality that $|\mathcal{V}| \geq 2$. Let k be a positive integer. Let us show that $P_{\Sigma=0}$ is not \mathcal{W}, k -definable.

Let $A = \{0, 1, -1\}$ and L be the tuples $\bar{b} \in A^{\mathcal{V}}$ that sum to 0. This set is not \mathcal{W} -closed. Indeed, consider a tuple \bar{a} that consists only of 0's but for one occurrence of 1. For all windows $W \in \mathcal{W}$, either the occurrence of 1 does not occur in W , and \bar{a} is W -equivalent to the null tuple which belongs to L , or there is some occurrence of 0 that occurs outside W , and by switching this 0 into a -1 yields once more a W -equivalent tuple in L . Hence, according to Theorem 8, for n sufficiently large, $P_{L,n}$ is not \mathcal{W}, k -definable.

Let us consider now some $\lambda > |\mathcal{V}|$, and the map from $\mathcal{D}_n = A^n$ to integers defined by $f(a_1 \dots a_n) = \sum_{i \in [n]} \lambda^i a_i$. Thanks to the choice of a sufficiently large λ , for all inputs $\bar{a} \in (\mathcal{D}_n)^{\mathcal{V}}$, $\sum_{v \in \mathcal{V}} f(a(v)) = 0$ if and only if all rows in \bar{a} sum to 0, i.e., if and only if $\bar{a} \in P_{L,n}$.

Thus, assume that $P_{\Sigma=0}$ would be \mathcal{W}, k -definable, then the problem $P_{L,n}$ would also be \mathcal{W}, k -definable. This is a contradiction. ◀

► **Corollary 12.** *The problems of null sum and of gcd one over more than 2^k inputs are not recognizable by combinatorial expressions of depth at most k .*

4 Selection functions

So far in our expressions we allowed only functions with bounded domain and unbounded arity, or bounded arity and unbounded domain. It is natural to ask if the class of expressions can be extended without being degenerate (i.e. not accepting all maps from \mathcal{D}^* to \mathcal{D}). In this section we present the class of selection functions that are similar to the *multiplexer gates* in digital circuits. Intuitively a selection function takes m values $a_1, \dots, a_m \in \mathcal{D}$ and a number i in $[m]$ as input and outputs the value a_i , i.e. it selects the i th input. To keep the discussion simple, let us assume that \mathcal{D} contains the natural numbers \mathbb{N} .

► **Definition 13.** Formally, a selection function sel_m of arity $m \in \mathbb{N}$ is a function of the form

$$sel_m : \left(\prod_{i \in [m]} E_i \right) \times [m] \rightarrow \bigcup_{i \in [m]} E_i$$

where each $E_i \subseteq \mathcal{D}$ such that $sel_m(a_1, \dots, a_m, j) = a_j$ for $a_1 \in E_1, \dots, a_m \in E_m, j \in [m]$.

We define the *extended class of combinatorial expressions* inductively as follows: every combinatorial expression belongs to the extended class. Further more, if $sel_m : \left(\prod_{i \in [m]} E_i \right) \times [m] \rightarrow \bigcup_{i \in [m]} E_i$ is a selection function and t_1, \dots, t_m, t are expressions in the extended class

with ranges $E_1, \dots, E_m, [m]$ respectively then $sel_m(t_1, \dots, t_m, t)$ is an expression with range $\bigcup_{i \in [m]} E_i$ that belongs to the extended class of terms.

In the following we prove that the extended class of expressions have the same expressive power. First we prove a normal form theorem for the extended class.

► **Theorem 14** (Normal form theorem for extended class of expressions). *For every expression $t(\bar{X})$ of depth $\ell \in \mathbb{N}$ there exists an equivalent expression of the form*

$$sel_m(\bar{r}(\bar{X}), f(\bar{s}(\bar{X})))$$

where $m \in \mathbb{N}$, $\bar{r}(\bar{X})$ is a sequence of binary expressions of depth at most ℓ , f is a finitary function, and $\bar{s}(\bar{X})$ is a sequence of binary expressions of depth at most ℓ . Moreover if the image of $t(\bar{X})$ is finite then there exists an equivalent expression of the form

$$f(\bar{s}(\bar{X}))$$

where f is a finitary function and \bar{s} is a sequence of binary expressions of depth at most ℓ .

An immediate consequence of the above result is that for defining functions from \mathcal{D}^* to \mathcal{D} with finite image, selection functions are useless. This situation is not different in general also. From Example 4 it follows that,

► **Remark.** The function $sel_{2^k+1} : \mathcal{D}^{2^k+1} \times [2^k + 1] \rightarrow \mathcal{D}$ is definable by a combinatorial expression (that does not use selection functions) of depth $k + 1$.

However selection functions add succinctness as the following propositions shows.

► **Proposition 2.** *The function $sel_{2^k+1} : \mathcal{D}^{2^k+1} \times [2^k + 1] \rightarrow \mathcal{D}$ is not definable by a combinatorial expression of depth k .*

Proof. The proof is very close to the proof of Proposition 1. Assume there is a combinatorial expression $t(X_1, \dots, X_{2^k+1}, y)$ that defines the function sel_{2^k+1} . By the normal form theorem we transform t into an equivalent expression of the form

$$b(r(X_1, \dots, X_{2^k+1}, y), f(\bar{s}(X_1, \dots, X_{2^k+1}, y)))$$

where b is binary, f is finitary, and r and \bar{s} are binary expressions of depth k . Since the expression r has depth k , there is a variable X_i that is not present in r . Choose an element $a \in \mathcal{D}$ and consider the inputs $S = \{(x_1, \dots, x_{2^k+1}, i) \in \mathcal{D}^{2^k+2} \mid x_i \in \mathcal{D}, \forall j \neq i, x_j = a\}$. Choose inputs $\bar{u}, \bar{v} \in S$ such that $\bar{u} \neq \bar{v}$ and $f(\bar{s}(\bar{u})) = f(\bar{s}(\bar{v}))$. Such inputs \bar{u} and \bar{v} exist by pigeonhole principle (since S is infinite while the image of f is finite). Observe that $b(r(\bar{u}), f(\bar{s}(\bar{u}))) = b(r(\bar{v}), f(\bar{s}(\bar{v})))$ contradicting the fact that $sel_{2^k+1}(\bar{u}) \neq sel_{2^k+1}(\bar{v})$. Hence the claim is proved. ◀

5 Application in metafinite logics

In this section we describe an application of our undefinability results, namely to prove inexpressibility results for logics on metafinite structures. *Metafinite model theory* was initiated by Grädel and Gurevich [2] in order ‘to extend the approach and methods of finite model theory beyond finite structures’. A *metafinite structure* \mathfrak{M} is a triple $\langle \mathfrak{A}, \mathfrak{B}, \rho \rangle$ where \mathfrak{A} is a finite first order structure, \mathfrak{B} is a first order structure (typically infinite) and ρ is a weight function from the domain of \mathfrak{A} to the domain \mathfrak{B} (the original definition allows a finite number of weight functions of different arity). For an example consider the structure

$\mathfrak{M} = \langle \mathfrak{A}, \mathfrak{B}, \rho \rangle$ where $\mathfrak{A} = ([n], \leq)$ is a finite linear order, $\mathfrak{B} = (\mathbb{N}, +, \times)$ is the natural numbers with arithmetic, and ρ is a map from $[n]$ to \mathbb{N} . In short, \mathfrak{M} represents a sequence of natural numbers. Such kind of *weighted* structures arise naturally in several areas of computer science. An important such case concerns databases, where some data naturally range over infinite/unbounded domains. When considering logics (for instance first order logic) expressing properties over metafinite structures, quantifications are assumed to only range over the finite structure \mathfrak{A} , but the formulas can access the structure \mathfrak{B} via the functions ρ and the use of the relations in \mathfrak{B} . In [2] several theorems from finite model theory are lifted to the case of metafinite models.

We now show that our undefinability results can be used to derive undefinability results in this context. We consider structures of the following form; $\mathfrak{M} = \langle \mathfrak{A}, \mathfrak{B}, \rho \rangle$ where $\mathfrak{A} = ([n], \leq)$ is a finite linear order, \mathfrak{B} is the natural numbers \mathbb{N} with all possible relations and functions of all arity (denoted as \mathbb{N}^*), and ρ is a map from the positions in \mathfrak{A} to \mathbb{N} . We consider the *monadic second order logic* on these structures which has the following syntax and semantics: we have first order variables x, y, \dots and set variables X, Y, \dots that range over positions and sets of positions in the structure \mathfrak{A} respectively. When x is a variable then $\rho(x)$ is a term over the structure \mathfrak{B} , and if t_1, \dots, t_k are terms over the structure \mathfrak{B} and f is a function in \mathfrak{B} of arity k then $f(t_1, \dots, t_k)$ is a term over the structure \mathfrak{B} . If R is a relation of arity k in \mathfrak{B} and t_1, \dots, t_k are terms over \mathfrak{B} , then $R(t_1, \dots, t_k)$ is an atomic formula of the logic. The only other atomic formulas are of the form $x \leq y$. The rest of the formulas of the logic are defined inductively: $\varphi_1 \vee \varphi_2, \neg \varphi_1, \exists x. \varphi_1, \exists X. \varphi_1$ are formulas when φ_1, φ_2 are formulas. The semantics of terms and formulas are defined in the obvious way (see [2], Definition 3.1).

Using our inexpressibility result we prove the following theorem:

► **Theorem 15.** *The set of all structures $\mathfrak{M} = \langle ([n], \leq), \mathbb{N}^*, \rho \rangle$ that satisfy the property*

$$\gcd(\rho(1), \dots, \rho(n)) = 1 \tag{3}$$

is not definable in monadic second order logic.

Before concluding, we note the following. The first remark is that by similar arguments we can also prove undefinability of the property $\sum (\rho(1), \dots, \rho(n)) = 0$ in monadic second order logic. Secondly, since our construction of the expression depends only on the fact that the quantification over the structure \mathfrak{A} is finite, the theorem also holds for any finite structure \mathfrak{A} , i.e. any finite signature not necessarily the signature (\leq) , and any logical formalism where the quantification over \mathfrak{A} is finite – in particular higher order logics.

6 Conclusion

In this work we introduced a formalism of expressions that take inputs from an infinite domain. The expressions are shown to have equivalent expressions in a normal form which allows to prove undefinability results using regularity lemmas from combinatorics. We point out some interesting avenues for further exploration. Firstly in this paper we have placed no restriction on the size of the circuits. But it seems that some of the results in Section 3 point towards the possibility of finer undefinability results that take into account the size of the expressions. Secondly we have defined our domain \mathcal{D} to be infinite. It is also an interesting to investigate expressive power of families of expressions when the data domain is unbounded yet finite, and grows asymptotically with the input size.

References

- 1 Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.
- 2 Erich Grädel and Yuri Gurevich. Metafinite model theory. *Information and Computation*, 140(1):26 – 81, 1998.
- 3 R.L. Graham, B.L. Rothschild, and J.H. Spencer. *Ramsey Theory*. A Wiley-Interscience publication. Wiley, 1990.
- 4 T. Jech. *Set Theory: The Third Millennium Edition, Revised and Expanded*. Springer Monographs in Mathematics. Springer, 2003.
- 5 Pascal Koiran. A weak version of the blum, shub, and smale model. *Journal of Computer and System Sciences*, 54(1):177 – 189, 1997.
- 6 Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhauser Verlag, Basel, Switzerland, Switzerland, 1994.
- 7 Pascal Tesson. *Computational Complexity Questions Related to Finite Monoids and Semigroups*. PhD thesis, School of Computer Science, McGill University, Montreal, 2003.
- 8 Pascal Tesson. An application of the haes-jewett theorem to multiparty communication complexity. Extract from the PhD Thesis, 2004.
- 9 L. G. Valiant. Completeness classes in algebra. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 249–261, New York, NY, USA, 1979. ACM.

Construction of μ -Limit Sets of Two-dimensional Cellular Automata

Martin Delacourt¹ and Benjamin Hellouin de Menibus²

- 1 CMM, Universidad de Chile, Beauchef 851, Edificio Norte – Piso 7, Santiago, CHILE
- 2 I2M, Aix-Marseille Université, CMI, 39 rue F. Joliot Curie, 13453 Marseille Cedex 13, FRANCE

Abstract

We prove a characterisation of μ -limit sets of two-dimensional cellular automata, extending existing results in the one-dimensional case. These sets describe the typical asymptotic behaviour of the cellular automaton, getting rid of exceptional cases, when starting from the uniform measure.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases cellular automata, dynamical systems, μ -limit sets, subshifts, measures

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.262

1 Introduction

Cellular automata are discrete dynamical systems defined by a local rule, introduced in the 40s by John von Neumann [12]. They model a large variety of discrete systems and are linked with various areas of mathematics or computer science, in particular computation theory, complex systems, ergodic theory and combinatorics.

One of the main catalysts of the study of cellular automata was their surprisingly complex and organised behaviours, even when iterated on configurations with no particular structure (e.g. chosen at random). To formalise these observations, many authors tried to describe their asymptotic behaviour by considering the limit set, which is the set of configurations that can be reached after arbitrarily many steps. These sets were shown to have potentially high computational complexity [11, 1], and any nontrivial property on them is undecidable [9]. Nevertheless, the problem of characterising which subshifts can be limit sets of CA remains open.

In 2000, Kůrka and Maass argued that limit sets did not provide a good description of empirical observations and introduced instead a measure-theoretical version [10]. The idea of μ -limit sets is to choose the initial configuration at random, according to some probability measure μ , and to consider all patterns whose probability to appear does not tend to 0. In the one-dimensional case, similar results of high complexity and undecidability were found [4, 3, 6, 2]. Another approach was developed in [5], considering the limit probability measure, with similar results.

In this article, we consider the two-dimensional case and prove a characterisation of all subshifts that can be μ -limit sets of CA for μ the uniform Bernoulli measure. The method is constructive and inspired by the one-dimensional constructions in [2, 5].



2 Definitions

2.1 Cellular automata on two dimensions

► **Definition 1** (Configurations, patterns, cylinders). Let \mathcal{A} be a finite alphabet. We introduce $\mathcal{A}^{\mathbb{Z}^2}$ the set of (two-dimensional) **configurations**. Denote \mathcal{A}^* the set of finite **patterns**, that is, any element of $\mathcal{A}^{\mathbb{U}}$ for some $\mathbb{U} \subset_{\text{finite}} \mathbb{Z}^2$ (denote $\mathbb{U} = \text{supp}(u)$ the **support** of the pattern u). Such a pattern is said to be square or rectangular if its support is.

Given $u \in \mathcal{A}^*$ and $i, j \in \mathbb{Z}^2$, define the cylinder $[u]_{i,j} = \{x \in \mathcal{A}^{\mathbb{Z}^2} \mid x_{(i,j)+\text{supp}(u)} = u\}$.

Endowed with the product topology, $\mathcal{A}^{\mathbb{Z}^2}$ is a compact and metrisable space. A distance inducing this topology is:

$$\forall x, y \in \mathcal{A}^{\mathbb{Z}^2}, d_C(x, y) = 2^{-\Delta(x, y)} \quad \text{where } \Delta(x, y) = \min\{|i| + |j| \mid i, j \in \mathbb{Z}^2, x_{i,j} \neq y_{i,j}\}$$

The **frequency** of a pattern $u \in \mathcal{A}^*$ in another pattern $v \in \mathcal{A}^*$ is defined as:

$$\text{Freq}(u, v) = \frac{\#\left\{ (i, j) \in \text{supp}(v) : \begin{array}{l} (i, j) + \text{supp}(u) \subseteq \text{supp}(v) \\ v_{(i,j)+\text{supp}(u)} = u \end{array} \right\}}{\#\{(i, j) \in \text{supp}(v) : (i, j) + \text{supp}(u) \subseteq \text{supp}(v)\}}, \quad 0 \text{ if it is undefined.}$$

► **Definition 2** (Shift actions). Define the two shifts actions $\sigma_{\uparrow}, \sigma_{\rightarrow} : \mathcal{A}^{\mathbb{Z}^2} \rightarrow \mathcal{A}^{\mathbb{Z}^2}$ by:

$$\forall x \in \mathcal{A}^{\mathbb{Z}^2}, i, j \in \mathbb{Z}^2, \quad \sigma_{\rightarrow}(x)_{i,j} = x_{i-1,j} \quad \text{and} \quad \sigma_{\uparrow}(x)_{i,j} = x_{i,j-1}.$$

► **Definition 3** (Cellular automata). A (two-dimensional) **cellular automaton** is a continuous action $F : \mathcal{A}^{\mathbb{Z}^2} \rightarrow \mathcal{A}^{\mathbb{Z}^2}$ that commutes with σ_{\rightarrow} and σ_{\uparrow} . Equivalently, it can be defined by a **local rule** $\bar{F} : \mathcal{A}^{\mathbb{U}_F} \rightarrow \mathcal{A}$, where $\mathbb{U}_F \subset \mathbb{Z}^2$ is a finite **neighbourhood**, in the sense that

$$\forall x \in \mathcal{A}^{\mathbb{Z}^2}, i, j \in \mathbb{Z}^2, F(x)_{i,j} = \bar{F}((x_{(i,j)+u})_{u \in \mathbb{U}_F}).$$

This equivalence is known as the Curtis-Hedlund-Lyndon theorem [7].

2.2 Probability measures

► **Definition 4** (Probability measures on $\mathcal{A}^{\mathbb{Z}^2}$). Let \mathfrak{B} be the Borel sigma-algebra of $\mathcal{A}^{\mathbb{Z}^2}$. Denote by $\mathcal{M}(\mathcal{A}^{\mathbb{Z}^2})$ the set of probability measures on $\mathcal{A}^{\mathbb{Z}^2}$ defined on the sigma-algebra \mathfrak{B} . Let $\mathcal{M}_{\sigma}(\mathcal{A}^{\mathbb{Z}^2})$ be the $\sigma_{\uparrow}, \sigma_{\rightarrow}$ -**invariant probability measures** on $\mathcal{A}^{\mathbb{Z}^2}$, that is to say the measures $\mu \in \mathcal{M}(\mathcal{A}^{\mathbb{Z}^2})$ such that $\mu(\sigma_{\uparrow}^{-1}(B)) = \mu(\sigma_{\rightarrow}^{-1}(B)) = \mu(B)$ for all $B \in \mathfrak{B}$. For a continuous application $F : \mathcal{A}^{\mathbb{Z}^2} \rightarrow \mathcal{A}^{\mathbb{Z}^2}$, denote $F\mu$ the image of the measure μ by F : $F\mu(X) = \mu(F^{-1}(X))$.

► **Definition 5** (Bernoulli measure). The Bernoulli measure $\mu_{\lambda} \in \mathcal{M}_{\sigma}(\mathcal{A}^{\mathbb{Z}^2})$ associated with a vector $\lambda = (\lambda_a) \in [0; 1]^{\mathcal{A}}$ such that $\sum_{a \in \mathcal{A}} \lambda_a = 1$ is defined as:

$$\forall u \in \mathcal{A}^{\mathbb{U}}, \mu_{\lambda}([u]) = \prod_{(i,j) \in \mathbb{U}} \lambda_{u_{i,j}}.$$

► **Definition 6** (μ -limit set). Let $F : \mathcal{A}^{\mathbb{Z}^2} \rightarrow \mathcal{A}^{\mathbb{Z}^2}$ be a CA and μ an initial probability measure. The μ -limit set of F $L_{\mu}(F)$ is defined by:

$$u \in L_{\mu}(F) \iff F^t \mu([u]) \xrightarrow[t \rightarrow \infty]{} 0.$$

2.3 Compatibility

The standard Turing machine model has access to a one-dimensional working tape than can be infinite on one or both sides. We consider in this paper that the machines have access to a two-dimensional tape infinite in all directions, in order to simplify some constructions. The only difference is that the computing head, when reading the current state and the letter on the tape at its current location, has the ability to move in four different directions: $\uparrow, \downarrow, \rightarrow, \leftarrow$. This model remains exactly as powerful as a Turing machine.

► **Definition 7** (Computable sequence of patterns). A sequence of patterns $(u_n)_{n \in \mathbb{N}} \in (\mathcal{A}^*)^{\mathbb{N}}$ is computable if there exists a Turing machine that, given as input an integer n written in binary, stops and outputs u_n .

In the previous definition, the Turing machine's alphabet contains at least \mathcal{A} and $\{0, 1\}$. We can assume the input is written left to right on row 0 surrounded by a special blank state.

► **Proposition 8.** Let $F : \mathcal{A}^{\mathbb{Z}^2} \rightarrow \mathcal{A}^{\mathbb{Z}^2}$ be a CA and $\mu \in \mathcal{M}_\sigma(\mathcal{A}^{\mathbb{Z}^2})$ be the uniform Bernoulli measure. Then there is a computable sequence of square patterns $(w_i)_{i \in \mathbb{N}}$ such that

$$u \in L_\mu(F) \iff \text{Freq}(u, w_i) \xrightarrow{i \rightarrow \infty} 0.$$

The sequence is built using de Bruijn tori, a combinatorial object constructed explicitly in [8]. Due to space constraints, the proof is in the appendix.

3 Main theorem

► **Theorem 9.** Let μ be the uniform Bernoulli measure over \mathcal{A} and $(w_i)_{i \in \mathbb{N}}$ a computable sequence of square patterns. Then there exists an alphabet $\mathcal{B} \supseteq \mathcal{A}$ and a cellular automaton F over \mathcal{B} such that:

$$u \in L_\mu(F) \iff \text{Freq}(u, w_i) \xrightarrow{i \rightarrow \infty} 0.$$

This theorem along with Proposition 8 characterises all μ -limit sets when μ the uniform Bernoulli measure. The proof of the theorem relies on an explicit construction; that is, we prove the result effectively by describing the CA.

Similarly to what was done for one-dimensional CA in [2, 5], the idea is, starting from some random configuration according to a measure μ , to build a partition of connected subsets of the plane using auxiliary states. In each subset, independently, each w_i is computed successively and concatenated copies of it are written over all the subset. To ensure the density of auxiliary states tends to 0, they merge progressively in a controlled manner, offering more space for computation.

4 Construction

4.1 Overview

First, we present a sketch of the different steps of the construction corresponding to a computable sequence of patterns $(w_i)_{i \in \mathbb{N}}$. The alphabet \mathcal{B} is the product of different layers, each layer being used for a different auxiliary process, plus two special states (seed and heart). The **main layer** is the writing layer whose alphabet is \mathcal{A} ; each other layer uses a different alphabet containing a blank symbol $\#$ corresponding to the absence of information. Hence we have $\mathcal{A} \subset \mathcal{B}$ up to the bijection $a \leftrightarrow (a, \#, \dots, \#)$.

- Colonising the space: Section 4.2.
Starting from a random configuration drawn according to μ , we first want to “clean” the randomly generated content of the auxiliary layers. \mathcal{B} contains a **seed** state $\boxed{*}$. Each seed, at time 1, erases the contents of a small area around it and give birth to membranes growing in every direction except when they meet other membranes. They erase all information contained in the auxiliary layers and membranes faking life which are recognised with the help of age counters.
- Internal metabolism: partitioning the cleaned space. Section 4.3.1.
Each seed gives birth to a heart \heartsuit that will be the core of a living organism. Every organism owns an age counter making sure they are all synchronised. Regularly, the organism around each living heart grows in each direction until it meets a fellow organism, thus claiming its territory.
- Internal metabolism: fighting for survival. Section 4.3.2.
Organisms need to become larger and larger through time, so we regularly remove some of the hearts. When two hearts are too close, one of them is removed to ensure that the distance between hearts is large and tends to infinity.
- Internal metabolism: Computing and writing. Sections 4.3.4 and 4.3.5.
In each organism, when the territory is established, some word w_n is computed and then written all over the territory. Copies of w_n thus cover the cleaned surface.

Throughout this article, t refers to the number of steps since time 0.

4.2 Colonisation of the space

4.2.1 Growing squares

There is a particular **seed** state $\boxed{*}$ that is only present in the initial configuration. It is the only relevant information in the initial configuration. Every occurrence of $\boxed{*}$ triggers the birth at time 1 and subsequent growth of a living square-shaped membrane (initially forming a 5×5 cells square).

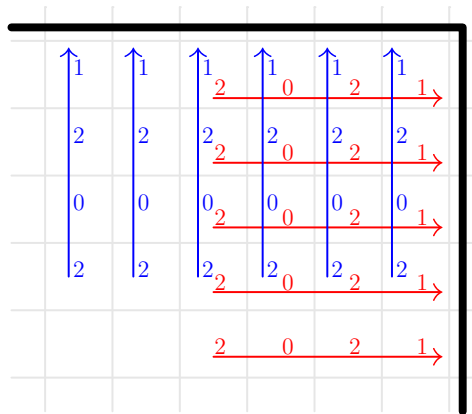
If seeds are too close from each other and do not have enough space to form the initial organism, the northernmost seed is destroyed (westernmost in case of a tie).

A layer of the alphabet, called **cleaning layer** is dedicated to the membrane growth and cleaning process. The membrane spreads slowly to the outside, thanks to a respiration process that “pushes” the membrane to the outside. A membrane is a boundary between its inside and the outside, thus defining the direction in which it expands. To each point of the membrane is associated a binary counter that keeps track of its age (see Figure 1).

► **Definition 10** (Redundant binary basis). Let $c = c_{n-1} \dots c_0 \in \{0, 1, 2\}^n$ be a counter. The **value** of c is $\sum_{i=0}^{n-1} c_i 2^i$ (reverse order). Since $2 = 10$, 2 can be seen as a 0 with a carry.

At each step, the counters are incremented by adding one to the least significant bit and the carries are propagated along the counter, which can be done in a local manner ($02 \rightarrow 10, 12 \rightarrow 20$).

If the membrane has sides of length n , there are n such counters on each side with the same value, with superpositions of two of them in the cells near the corner. As they grow, they need more than one cell and form a band of growing width along the membrane as shown in Figure 1. For a living membrane, the counters are created with value 0 at step $t = 1$, ensuring their age is the current time minus 1. In the other cases, the membrane and counters already existed at time $t = 0$ (with value at least 0), which means they appear older than living membranes.



■ **Figure 1** Corner of a membrane extending to the north and the east.

This counter is used to control the speed of the membrane. The respiration consists in taking a step forward (according to the direction of the membrane) each time the age of the counter is the exact square of an integer. The successive squares are computed under the counter, on the computation layer, using a space $O(\log t)$ if t is the age of the membrane.

We can define three kinds of membranes:

- Living membranes** which were created by a seed, and whose counters all have value $t - 1$;
- Dead membranes** which have some incoherence (not closed, different counter values, no square computation...) and self-destruct when realising it;
- Zombie membranes** which are perfectly coherent despite not being created by a seed, and whose counters all have the same value $t' > t - 1$.

The content of any cell outside a membrane is deleted, except for the encounter of another membrane. In this case the comparison process starts. The reason membranes spread slowly is to limit the interferences between the growing and comparison processes.

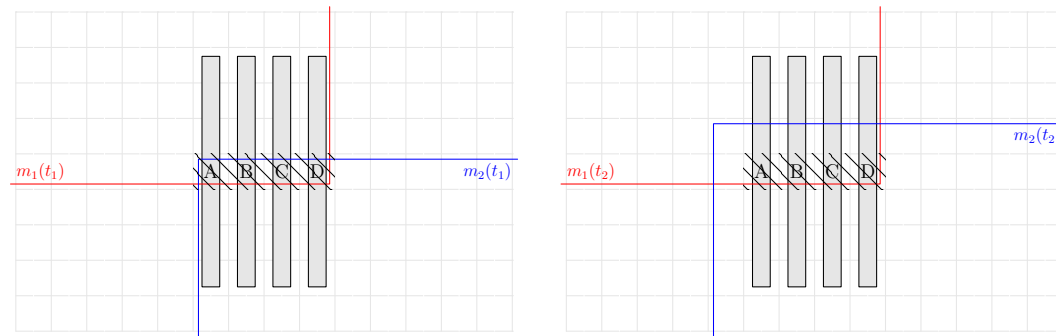
4.2.2 Comparison

When two membranes meet, membranes fight for survival, which is only granted to the youngest. Indeed, we saw that only living membranes can have age $t - 1$, all other membranes' counters having value greater than t . Comparing the age of both counters is achieved on a dedicated **comparison layer**.

When membranes meet, two special states \square are written on the comparison layer to trigger the process on each side. Each of them progresses along its corresponding counter, copying the value of the counter on the comparison layer. Incrementation and carry propagation continue in the original counter. However, it is not necessary to increment and propagate carries in the copied counter since both sides would increase by the same amount during the comparison anyway. During the copy into the comparison layer, all carries are taken into account and resolved, so that we have two pure binary counters at the end of the process.

Both copied counters progress towards the encounter point at speed 1 and a comparison is performed bit-by-bit, starting from the least significant. When the last bits of the counters arrive, we can decide which counter corresponds to the youngest membrane.

As shown in Figure 2, if at time t_1 two membranes meet, comparison of the age of counters takes place at each contact cell. Here the same process takes place at cells A , B , C and D .



■ **Figure 2** At time t_1 , the membranes m_1 and m_2 meet on cells A , B , C and D . The counters are represented by grey areas. At t_2 , when the comparison is finished, one of the squares may have grown (here m_2).

► **Proposition 11.** *During a comparison process, a living membrane may grow only once (including the initial growth that triggered the comparison)*

Proof. If the comparison process started at time t_0 , the counters of a living membrane have length less than $\log(t_0)$. The comparison process takes at most twice as many steps as the length of the counter. The respiration process happens when t is a perfect square. Therefore the time between two successive growths, at time t_0 or later, is at least $\lceil \sqrt{t_0} \rceil$ steps. ◀

Let us consider the various possible results:

The membranes have the same age: they are both alive or both zombie. In any case, both membranes turn into a single one as shown in Figure 3. Some \boxtimes symbols are written at the corners, so that, when both sides grow again, they remember they are part of the same membrane.

A membrane is younger: the oldest one is zombie and can be safely destroyed. A death signal \boxtimes spread in both directions along the oldest membrane, erasing it. The surviving membrane resumes its growth, with its age counters still accurate. The same happens if a membrane grows twice, disrupting the comparison process.

Notice that only the membrane and not the "insides" of the zombie are cleaned since it can contain other living membranes. None of the signals or processes described in the following sections can enter or leave a membrane, or interact with it or counters, except if explicitly mentioned.

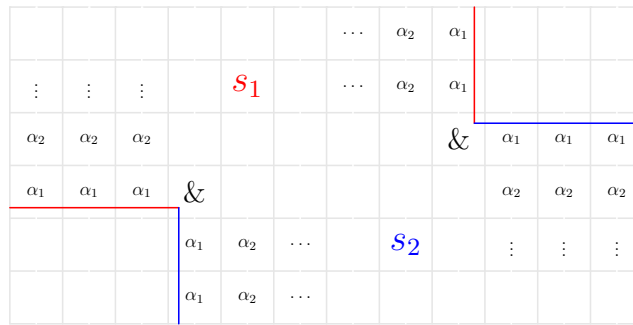
For $t \in \mathbb{N}$, denote

$$Pr(t) = \{F^t(c) \mid c \in \mathcal{B}^{\mathbb{Z}^2}, \exists(i, j) \in \mathbb{Z}^2, d_\infty((i, j), (0, 0)) \leq \lfloor \sqrt{t} \rfloor, c_{ij} = \boxtimes\}$$

the set of images of configurations containing a seed \boxtimes at distance $\lfloor \sqrt{t} \rfloor$ at most of $(0, 0)$. As μ is the uniform Bernoulli measure, the following lemma is clear:

► **Lemma 12.** $F^t \mu(Pr(t)) = 1 - (1 - \mu(\boxtimes))^{(2\sqrt{t}+1)^2} \rightarrow_t 1$

This means that, with probability 1, for almost any configuration the central cell eventually belongs to the insides of a living membrane.



■ **Figure 3** At the end of the comparison, if membrane counters share a common value, the common part of their boundaries is erased and $\&$ symbols mark the corners.

4.3 Working in the clean surface

We now consider only the protected area, which is the union of all insides of living membranes. Thus every construction presented in this section remains inside this area and stops if it reaches the membrane. They take place on four new layers: the **age**, **partitioning**, **computing** and **writing** layers.

At some time $t_n = K^{2n}$, $n \in \mathbb{N}$ for some integer K that will be specified later, various operations are performed simultaneously inside all membranes. First, a simulated Turing machine computes w_n . Then, repeated copies of w_n are copied everywhere inside the membrane. Meanwhile, the heart checks that it is not too close to a neighbour, and one of them is deleted if it is the case.

These operations occurs between times t_n and $t_{n+1} - 1$, which is called the n^{th} generation.

4.3.1 Claiming its territory

At time 1, while creating a membrane, each seed \boxtimes transforms itself into a **heart** \heartsuit . Any heart is the centre of an **organism** to which it provides life. At the same time, a binary counter is given to each heart, thus giving it the knowledge of its age. This age is exactly the same for any heart inside a living membrane. This counter is the only thing contained in the age layer.

In the rest of this section, only the partitioning layer is concerned.

At time t_n , every heart send signals at speed 1 in each direction until they meet a fellow signal, in which case they disappear and the symbol \boxplus is written where they met. These signals erase everything on the partitioning and computing layers but disappear if they reach a membrane. In this case, \boxplus is written along the membrane. The **territory** of the heart $H \in \mathbb{Z}^2$ is the largest set of 4-connected cells containing H that does not contain the symbol \boxplus . An organism is composed of a heart and its territory.

Simultaneously, at $t = t_n$, signals leave H and draw the **body** of H : a square of size $2n + 1$ centred in H . The body is supposed to be entirely in the territory of H ; if not, the organism is in conflict with every other organism whose body intersects its own. At the end of each generation, we make sure there does not remain any conflict by removing some of the hearts.

Thus, the global dynamics partition the protected space by redefining territories during each generation, then resolve conflicts: during the n^{th} generation, the distance between two surviving hearts is at least $2n - 1$ (remember we use the distance d_∞).

4.3.2 Choosing its destiny

In this section, we describe conflicts. To get organisms larger and larger through time, we want them to contain at least their entire body, whose size depends of the current generation. We need as well to control the growth of the organisms, preventing them from being too large. Indeed, we have to write the computed pattern all over the organism before the beginning of the next generation. Thus, if at some step a chain of conflicts between organisms appears, we do not want to erase all hearts simultaneously.

To avoid this, we add an algorithmic device and give to each heart some bit of information with the constraint that these bits have to be mutually independent at any given time. Then, for each conflict between two organisms, we choose the one to delete thanks to the sum of their two random bits.

First, we use two versions of the state \boxtimes in the initial configuration: \boxtimes_0 and \boxtimes_1 . This bit is transmitted to \heartsuit which has two versions \heartsuit_0 or \heartsuit_1 . In both cases, we keep the notations \boxtimes and \heartsuit when the value of the bit does not matter. The bit is also known by the whole boundary of the corresponding organism.

Second, note that, given some heart H living at generation n , the conflicting hearts are at distance $2n - 1$ or $2n$ or they would have conflicted before. Thus, they all belong to a square of side $4n + 1$ centred in H . The distance between each other is also $2n - 1$ or $2n$, hence there are at most 8 simultaneous conflicts, one at most in each eighth part of the plane centred in H : NNE, ENE, ESE, SSE, SSW, WSW, WNW and NNW.

To ensure that the independence property remains true, a heart provides some fresh information to its killer when it is deleted. Hence, we give 8 other binary bits to each seed, and therefore to each heart. Each eighth part of the territory's boundary carries one of these reserve bits alongside with the main one.

During the n^{th} generation, when two organisms O and O' of hearts respectively \heartsuit_b at (x, y) and $\heartsuit_{b'}$ at (x', y') meet, the sum $\beta = b \oplus b'$ is computed where the boundaries meet. If $\beta = 0$ then the northernmost heart wins (westernmost in case of a tie) and the other way around if $\beta = 1$. Then the boundary of the killed organism (say O') transmits its reserve bit b_r to the winner whose main bit becomes $b \oplus b_r$. If some organism kills many others simultaneously (at most 8), it sums all transmitted reserve bits to its own. The key point is that all main bits are and remain independent. This is ensured since the reserve bits are not used until they pass to the winner.

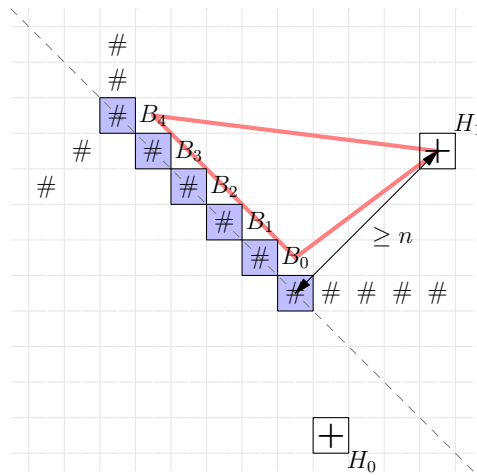
On the other hand, a death signal is sent to the heart of the loser, which dies at the reception. This does not interrupt the processes of computation or copying that will be described later, but the organism will never grow again and signals from other hearts will erase it during the next generation.

► **Definition 13.** Define the *radius* r of an organism as the largest distance from a cell inside its territory to its heart. The territory of the organism is hence bounded by $4r^2$.

► **Lemma 14.** *There exists a constant K , such that $p_n \rightarrow_n 1$, where p_n is the probability that at least one living heart remains in a square of radius K^n during the n^{th} generation.*

Proof. Denote $q_n, n \in \mathbb{N}$ the probability for a cell to be a living heart during generation n . For $n = 0$, $q_0 > 0$ is a constant given by μ . Then, during each generation $k \leq n$, a heart survives with probability at least $(1/2)^8$ ($1/2$ for each conflict). Hence $q_n \geq q_0 * (1/2)^{8n}$.

Two different cells have each independently probability q_n to be a heart as long as there is no chain of conflicts between them. At generation n , they have been affected only by hearts at distance $\sum_{k=0}^n k \leq n^2$ at most. So there are $d_n = \lfloor (2K^n + 1)/(2n^2 + 1) \rfloor^2$ independent cells in a square of radius K^n .



■ **Figure 4** Two hearts H_1 and H_2 are conflicting. Cells A_0 to A_4 form the common boundary of their territory. The red triangle is a set of cells inside the territory of H_1 .

Now we have $1 - p_n \leq (1 - q_n)^{d_n}$. This tends to 0 for $K \geq 17$. ◀

This lemma means that we only need to consider organisms of radius less than K^n . The other ones are sufficiently sparse.

► **Definition 15.** An organism is said to be *healthy* during the n^{th} generation when its radius is less than K^n (K being given in the previous lemma).

4.3.3 Shape of organisms

► **Lemma 16.** If a cell A is in the organism of heart H , then each cell B such that $d_\infty(B, H) \leq d_\infty(A, H) - d_\infty(A, B)$ is in the same organism.

Proof. The triangle inequality gives the result automatically, for any other heart H' :

$$d_\infty(B, H) \leq d_\infty(A, H) - d_\infty(A, B) \leq d_\infty(A, H') - d_\infty(A, B) \leq d_\infty(B, H').$$

► **Lemma 17.** $F^{t_n} \mu(\llbracket \# \rrbracket \cap Pr(t_n)) = O(1/n)$

Proof. Given $n \in \mathbb{N}$, consider the set of cells containing state $\#$ at time t_{n+1} within the protected area. It is possible to cut this set into horizontal, vertical or diagonal segments such that each one of them is the common boundary of two specific hearts. When two hearts claim their territory, they send signals in every direction at speed one. These signals may eventually cross to give birth to the boundary. Except if they cross exactly in their corners (four cells for each organism, which is negligible), the length of their common boundary is at least 2. Consider one of these boundary segments containing cells $\{A_0, A_1, \dots, A_k\}$ and denote H_0 and H_1 the associated hearts.

The proof is illustrated on Figure 4 in the case of a diagonal segment. Denote d the line supporting the segment, as $d_\infty(H_0, H_1) \geq 2n, \exists j \in \{0, 1\}$ such that $d_\infty(H_j, d) \geq n$. Denote O_j the organism centred in H_j . Since A_0, A_1, \dots, A_k are on the boundary of O_j , there exist distinct points B_0, B_1, \dots, B_{k-1} adjacent to A_0, A_1, \dots, A_k and inside O_j .

► **Claim 18.** Every cell inside the triangle $B_0 B_{k-1} H_j$ is inside O_j .

Proof. For any such cell x , there exists $l \in [0, k - 1]$ such that $d_\infty(H_j, B_l) = d_\infty(H_j, x) + d_\infty(x, B_l)$. Hence, using Lemma 16, x belongs to O_j . ◀

There are $\lfloor (k - 1)(n - 1)/2 \rfloor$ cells in the triangle $B_0 B_{k-1} H_j$, which means that for each cell of the boundary segment, we produced $O(n)$ cells inside an organism.

Any cell inside an organism can be attached this way to two segments at most (the border of the triangle can be shared). Thus, for any cell containing $\boxed{\#}$, there are at least $\Theta(n)$ cells that do not contain $\boxed{\#}$. Hence $F^{t_n} \mu(\boxed{\#}) \cap Pr(t_n) = O(1/n)$. ◀

4.3.4 Computing

In this section, we deal only with the computing layer. At time $t_n, n \in \mathbb{N}$, the same computation starts around each heart. While signals leave the heart to determine the boundaries of their territory, other signals draw the limits of a square of side \sqrt{n} whose down-left corner is the heart. This is the space allowed for computation. The heart creates a Turing machine head and the computation starts. It has to remain in this space and halt in less than K^{2n} .

Without loss of generality, we can choose a computable sequence of patterns $(w_i)_{i \in \mathbb{N}}$ such that w_n is the pattern computed during the n^{th} generation. Indeed, we can transform the original sequence by repeating each pattern until there is enough space and time to compute the following one. Denote \mathbb{U}_n the support of w_n and l_n its size: $\mathbb{U}_n = \text{supp}(w_n) = [0, l_n] \times [0, l_n]$. Considering the space allowed for computation, we have that $l_n \leq \sqrt{n}$.

4.3.5 Copying

Finally, we consider the copying layer. After computing a pattern on the computing layer of an organism, we write copies of it over the whole territory of this organism.

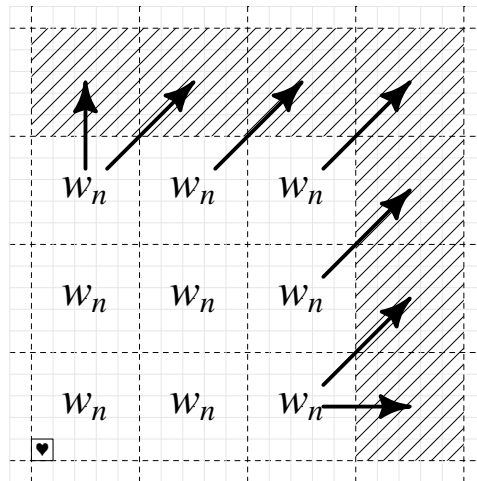
During the n^{th} generation, the computation takes less than K^{2n} steps, which leaves $K^{2n+2} - K^{2n}$ steps before t_{n+1} . We show that this is enough to write periodic copies of the result all over the organism, as long as the organism is healthy.

Consider an organism of heart $H = (x_H, y_H)$ during generation n . We first write 4 copies of w_n around H at $(x_H - l_n, y_H - l_n) + \mathbb{U}_n$, $(x_H - l_n, y_H) + \mathbb{U}_n$, $(x_H, y_H - l_n) + \mathbb{U}_n$ and $H + \mathbb{U}_n$. To copy a square, a machine copies all the states sequentially. First, the sides of the squares are marked on the copying layer with a state \boxed{G} (this takes $O(l_n)$ steps using counters initialised with value l_n), then the machine needs $2l_n$ steps to go to the copy emplacement, make the copy and come back. There are l_n^2 cells to copy, so the whole process of copying a square takes $O(l_n^3)$ steps.

Starting with these 4 copies of w_n , 4 different copying processes take place, each one in its quarter of the plane: north-east, north-west, south-west and south-east. We only detail the process in the north-east quarter.

The base square is copied along the vertical and horizontal axes until it reaches the limit of the territory. Simultaneously, each of these copies replicates itself in diagonal towards the north-east. This way, the whole territory is eventually covered with copies of the computed pattern w_n . The set of states \boxed{G} draw a grid of step l_n . The copying process is actually a wave starting at the heart of the organism and extending the area where the pattern w_n is written. See Figure 5.

► **Lemma 19.** *For any healthy organism, copying takes less than $O(nK^n)$ time steps during the n^{th} generation.*



■ **Figure 5** The square pattern is copied all over the whole territory both on axes and along diagonals, starting from the heart.

Proof. Consider a healthy organism, as the radius is bounded by K^n and the grid step is l_n , there are sequences of at most K_n/l_n square copies to do in each quarter. Each one of these copies requires $O(l_n^3)$ steps, hence the total copy time is $O(nK^n)$ (recall $l_n \leq \sqrt{n}$). ◀

► **Lemma 20.** *During the n^{th} generation, any cell in a healthy organism that was not reached by the copying process is at distance \sqrt{n} or less of the boundary of the territory.*

Proof. Again, we prove it in the north-east quarter, the proof is symmetric in the other cases. Take a cell A in the territory of a healthy organism and at distance more than l_n of the boundary of the territory. A is in a square S of the \mathbb{G} grid (or would be by extending the grid). Thanks to the hypothesis we know that S entirely belongs to the organism. The copy process reached S , arriving from a square S' at the south, east or south-east of S depending of the position of S . Now, according to Lemma 16, S' entirely belongs to the organism.

This way, we can go recursively all the way back to the heart, and the copy process is necessarily successful at each step. ◀

5 Proof of the main theorem

We saw in previous sections that a configuration tends to contain only healthy organisms, and that computing and copying can be both achieved in less than $t_{n+1} - t_n$ time steps in a healthy organism. From this we now conclude.

Proof. Given a sequence $(w_n)_{n \in \mathbb{N}}$, we build the cellular automaton F over the alphabet \mathcal{B} as described in the previous sections.

Suppose $t = t_{n+1} - 1$, $n \in \mathbb{N}$. First, if $s \in \mathcal{B} \setminus \mathcal{A}$, a cell can have state s if it is:

- outside the protected area, use Lemma 12;
- outside a healthy organism, use Lemma 14;
- in the border of a healthy organism, use Lemma 17;
- in the computation area of an organism, which are negligible since this area is a square of side \sqrt{n} in territories that contain a square of side n ;

- in the grid drawn in each territory (states \square), negligible as well since the grid occupies less than $4l_n$ cells in each square of side l_n .

Therefore $L_\mu(F) \subseteq \mathcal{A}^*$.

Now, we show that we only need to consider the squares of the grid entirely included in a healthy organism. As we said before, it is enough to consider healthy organisms. Every square that is only partially inside a healthy field is located into a band of width less than \sqrt{n} adjacent to the boundary of the field, hence there are at most $O(1/\sqrt{n})$ such cells thanks to Lemma 12. As we forced $i \leq \sqrt{n}$, we can effectively neglect those partial squares. In any other square, thanks to Lemma 20, we know that the copy was achieved successfully.

For all these reasons, for a square pattern u , $F^{t_n} \mu([u]) \sim_{n \rightarrow \infty} \text{Freq}(u, w_n)$.

Moreover, during the n^{th} generation, while the copying process is engaged but not finished, some part of the main layer contains copies of w_n and the rest is still filled with copies of w_{n-1} . Hence, for some $0 \leq \alpha \leq 1$:

$$F^t \mu([u]) \sim_{n \rightarrow \infty} (\alpha F^{t_n} \mu([u]) + (1 - \alpha) F^{t_n} \mu([u])).$$

Perspectives

As for the one-dimensional case, we have a characterisation of all subshifts that are μ -limit sets of CA. Some corollaries can be derived from this result, but the main open problem is to generalise it to larger classes of measures. In dimension 1, the difference is that there is no need for a trick such as the one used in Section 4.3.2 to resolve conflicts while avoiding erasing too many hearts. As this trick only works with the uniform Bernoulli measure, hence, a better understanding of the dynamics of disappearance of the hearts should allow to generalise the result.

References

- 1 Alexis Ballier, Pierre Guillon, and Jarkko Kari. Limit sets of stable and unstable cellular automata. *Fundam. Inf.*, 110(1-4):45–57, January 2011.
- 2 Laurent Boyer, Martin Delacourt, Victor Poupet, Mathieu Sablik, and Guillaume Theyssier. μ -limit sets of cellular automata from a computational complexity perspective. *CoRR*, abs/1309.6730, 2014.
- 3 Laurent Boyer, Martin Delacourt, and Mathieu Sablik. Construction of μ -limit sets. In *JAC*, pages 76–87, 2010.
- 4 Laurent Boyer, Victor Poupet, and Guillaume Theyssier. On the complexity of limit sets of cellular automata associated with probability measures. In *MFCS*, pages 190–201, 2006.
- 5 Benjamin Hellouin de Menibus and Mathieu Sablik. Characterisation of sets of limit measures after iteration of a cellular automaton on an initial measure. *CoRR*, abs/1301.1998, 2013.
- 6 Martin Delacourt. Rice’s theorem for μ -limit sets of cellular automata. In *ICALP (2)*, pages 89–100, 2011.
- 7 Gustav A. Hedlund. Endomorphisms and automorphisms of the shift dynamical system. *Mathematical Systems Theory*, 3(4):320–375, 1969.
- 8 Glenn Hurlbert and Garth Isaak. On the de bruijn torus problem. *Journal of Combinatorial Theory, Series A*, 64(1):50 – 62, 1993.
- 9 J. Kari. Rice’s theorem for the limit sets of cellular automata. *Theoretical Computer Science*, 127:229–254, 1994.

- 10 P. Kůrka and A. Maass. Limit Sets of Cellular Automata Associated to Probability Measures. *Journal of Statistical Physics*, 100(5-6):1031–1047, 2000.
- 11 Alejandro Maass. On the sofic limit sets of cellular automata. *Ergodic Theory and Dynamical Systems*, 15:663–684, 7 1995.
- 12 John von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

Derandomized Graph Product Results Using the Low Degree Long Code

Irit Dinur^{*1}, Prahladh Harsha², Srikanth Srinivasan³, and Girish Varma^{†4}

- 1 Weizmann Institute of Science, Israel.
irit.dinur@weizmann.ac.il
- 2 Tata Institute of Fundamental Research, India.
prahladh@tifr.res.in
- 3 Department of Mathematics, IIT Bombay, India.
srikanth@math.iitb.ac.in
- 4 Tata Institute of Fundamental Research, India.
girishrv@tifr.res.in

Abstract

In this paper, we address the question of whether the recent derandomization results obtained by the use of the low-degree long code can be extended to other product settings. We consider two settings: (1) the graph product results of Alon, Dinur, Friedgut and Sudakov [*GAF*A, 2004] and (2) the “majority is stablest” type of result obtained by Dinur, Mossel and Regev [*SICOMP*, 2009] and Dinur and Shinkar [In *Proc. APPROX*, 2010] while studying the hardness of approximate graph coloring.

In our first result, we show that there exists a considerably smaller subgraph of $K_3^{\otimes R}$ which exhibits the following property (shown for $K_3^{\otimes R}$ by Alon *et al.*): independent sets close in size to the maximum independent set are well approximated by dictators.

The “majority is stablest” type of result of Dinur *et al.* and Dinur and Shinkar shows that if there exist two sets of vertices A and B in $K_3^{\otimes R}$ with very few edges with one endpoint in A and another in B , then it must be the case that the two sets A and B share a single influential coordinate. In our second result, we show that a similar “majority is stablest” statement holds good for a considerably smaller subgraph of $K_3^{\otimes R}$. Furthermore using this result, we give a more efficient reduction from Unique Games to the graph coloring problem, leading to improved hardness of approximation results for coloring.

1998 ACM Subject Classification G.2.2 Graph Theory, F.1.3 Reducibility and completeness

Keywords and phrases graph product, derandomization, low degree long code, graph coloring

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.275

1 Introduction

The discovery of the low-degree long code (aka short code) by Barak *et al.* [2] has over the last year led to several more efficient inapproximability reductions [2, 5, 9, 13, 15]. The low-degree long code is a derandomization of the long code in the following sense. Given a finite field \mathbb{F} , the long code of a string $x \in \mathbb{F}^n$ is the evaluation of every \mathbb{F} -valued function on

* Irit Dinur’s research is supported by ERC-Stg grant number 239985.

† Girish Varma’s research is supported by Google Ph.D. Fellowship in Algorithms. Part of the work was done when the author was visiting the Weizmann Institute of Science, Israel.

\mathbb{F}^n at the point x while the degree d long code of x is the evaluation of every n -variate polynomial of total degree at most d at the point x . The crucial observation of Barak *et al.* [2] was that the optimal testing results for Reed-Muller codes [3, 10] proved that the low-degree long code could be used as a surrogate for the long code in several inapproximability results. In this paper, we ask if we can extend this application of low-degree long code to other product settings. In particular, we prove the following two results. (1) We show that result due to Alon *et al.* [1] on the size of maximum independent sets in product graphs can be derandomized (Theorem 1.2). (2) We show that the “majority is stablest” type of result obtained by Dinur *et al.* [7] and Dinur and Shinkar [8] can be derandomized (Theorem 1.4).

1.1 Derandomized graph products

As a first application, we consider the following graph product result due to Alon *et al.* [1]. Consider the undirected weighted graph K_3 on the three vertices $V = \{0, 1, 2\}$ and edges weighted as follows: $W(f, f') = 1/2$ iff $f' \neq f \in \{0, 1, 2\}$. Let $K_3^{\otimes R}$ be the graph with vertex set $V^{\otimes R}$ and weights-matrix the R -wise tensor of the matrix W . Clearly, for any $i \in [R]$ and $a \in \{0, 1, 2\}$, the set $V_{i,a} := \{v \in V^{\otimes R} : v_i = a\}$ is an independent set in $K_3^{\otimes R}$ of fractional size $1/3$ since K_3 does not have any self loops. We call such an independent set a *dictator* for obvious reasons. Alon *et al.* [1] showed that these are the maximal independent sets in $K_3^{\otimes R}$ and in fact any independent set of size close to the maximum is close to a dictator.

► **Theorem 1.1** ([1]). *Let A be an independent set in $K_3^{\otimes R}$ of size $\delta 3^R$. Then,*

1. $\delta \leq 1/3$.
2. $\delta = 1/3$ iff A is a dictator.
3. If $\delta \geq 1/3 - \varepsilon$, then A is $O(\varepsilon)$ -close to a dictator. That is, there is a dictator A' such that $|A \Delta A'| = O(\varepsilon 3^R)$.

Note that the above graph has 3^R vertices. Our first result (Theorem 1.2) shows that there exists a considerably smaller subgraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of $K^{\otimes R}$ with only $3^{\text{poly}(\log R)}$ vertices that has the same properties. In order to describe the subgraph, it will be convenient to think of K_3 as having vertex set \mathbb{F}_3 and

$$W(f, f') = \Pr_{p \in \mathbb{F}_3, a \in \{1, 2\}} [f' = f + a(p^2 + 1)].$$

Let $\mathcal{P}_{r,d}$ be the set of polynomials on r variables over \mathbb{F}_3 of total degree at most d and individual degrees of the variables at most 2. Let r and d be two parameters and let $R = 3^r$. Note that $V^{\otimes R}$ can be identified with $\mathcal{P}_{r,2r}$, since $\mathcal{P}_{r,2r}$ is the set of all functions from \mathbb{F}_3^r to \mathbb{F}_3 . The subgraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is as follows: $\mathcal{V} := \mathcal{P}_{r,2d}$ and the edges are given by the weights-matrix defined below

$$\mathcal{W}(f, f') = \Pr_{p \in \mathcal{P}_{r,d}, a \in \{1, 2\}} [f' = f + a(p^2 + 1)].$$

Note that since $\mathcal{P}_{r,2d}$ is a subspace of dimension $r^{O(d)}$, the size of the vertex set is $3^{r^{O(d)}}$, which is considerably smaller than 3^R for constant d .

► **Theorem 1.2.** *There is a constant d for which the following holds. If A is an independent set of size $\delta |\mathcal{V}|$ in \mathcal{G} then*

1. $\delta \leq 1/3$.
2. $\delta = 1/3$ iff A is a dictator.
3. If $\delta \geq 1/3 - \varepsilon$ then A is $O(\varepsilon)$ -close to a dictator.

A crucial element in the proof of Theorem 1.1 is a hypercontractivity theorem for functions which do not have any heavy Fourier coefficients. Theorem 1.2 is proved by observing that a similar hypercontractivity theorem also holds good in the low-degree long code setting (see Lemma 3.4).

1.2 Derandomized “majority is stablest” result

While studying the hardness of approximate graph coloring, Dinur, Mossel and Regev [7] proved the following “majority is stablest” type of result: if there is a pair of subsets of vertices in $K_3^{\otimes R}$ of sufficiently large size such that the average weight of edges between them is small, then their indicator functions must have a common influential coordinate. Subsequently, Dinur and Shinkar [8] obtained the following quantitative improvement to the above theorem.

► **Theorem 1.3** ([8, Theorem 1.3]). *For all $\mu > 0$ there exists $\delta = \mu^{O(1)}$ and $k = O(\log 1/\mu)$ such that the following holds: For any two functions $A, B : \{0, 1, 2\}^R \rightarrow [0, 1]$ if*

$$\mathbb{E} A > \mu, \mathbb{E} B > \mu, \quad \text{and} \quad \mathbb{E}_{f, f'} A(f)B(f') \leq \delta^1$$

where f is chosen randomly from $V^{\otimes R}$ and f' is chosen with probability $W^{\otimes R}(f, f')$ then

$$\exists x \in [R] \text{ such that } \text{Inf}_x^{\leq k}(A) \geq \delta \text{ and } \text{Inf}_x^{\leq k}(B) \geq \delta.$$

Our second result (Theorem 1.4) shows that the above theorem can be derandomized to obtain a similar result for the subgraph \mathcal{G} . For defining influence for real valued functions on $\mathbb{P}_{r,2d}$, we note that the characters of $\mathbb{P}_{r,2d}$ are restrictions of characters of $\mathbb{F}_3^R \equiv \mathbb{P}_{r,2r}$. So the definition of influence for functions on \mathbb{F}_3^R also extends naturally to functions on $\mathbb{P}_{r,2d}$.

► **Theorem 1.4.** *For all $\mu > 0$ there exists $\delta = \mu^{O(1)}$, $k = O(\log 1/\mu)$, $d = O(\log 1/\mu)$ such that the following holds: For any two functions $A, B : \mathbb{P}_{r,2d} \rightarrow [0, 1]$ if*

$$\mathbb{E} A > \mu, \mathbb{E} B > \mu, \quad \text{and} \quad \mathbb{E}_{f, f'} A(f)B(f') \leq \delta$$

where f is chosen randomly from $\mathbb{P}_{r,2d}$, $f' = f + a(p^2 + 1)$, p are chosen randomly from $\mathbb{P}_{r,d}$ and $a \in_R \{1, 2\}$ then

$$\exists x \in \mathbb{F}_3^r \text{ such that } \text{Inf}_x^{\leq k}(A) \geq \delta \text{ and } \text{Inf}_x^{\leq k}(B) \geq \delta.$$

A similar derandomized “majority is stablest” result in the case of the noisy hypercube was proved by Barak *et al.* [2, Theorem 5.6] and they used the Meka-Zuckerman pseudorandom generators (PRGs) for polynomial threshold functions [14]. Kane and Meka [11] obtained a quantitative improvement over this derandomization by constructing an improved PRG for Lipschitz functions. Our setting is slightly more involved, (1) we have a two function version (ie., A and B) and (2) the underlying graph in K_3 and the corresponding noise operator in the derandomized setting has not necessarily positive eigenvalues. Yet, we manage to show that a derandomization still holds in this case too (using the Kane-Meka PRG). We conjecture that our derandomization can be further improved to obtain $d = O(\log \log 1/\mu)$.

¹ The hypothesis in the theorem statement of Dinur-Shinkar [8] requires $\mathbb{E}_{f, f'} A(f)B(f') = 0$, however it is easy to check that their theorem also holds good under the weaker hypothesis $\mathbb{E}_{f, f'} A(f)B(f') \leq \delta$.

1.2.1 Application to graph coloring

Using a version of Theorem 1.3 for another base graph on 4 vertices, Dinur and Shinkar proved a hardness result for graph coloring.

► **Definition 1.5** (Label Cover). An instance $G = (U, V, E, L, R, \{\pi_e\}_{e \in E})$ of a *Label Cover* consists of a bipartite graph (U, V, E) that is right regular along with a projection map $\pi_e : R \rightarrow L$ for every edge $e \in E$. Label Cover is a constraint satisfaction problem where the vertices in U are the variables taking values in L and vertices in V taking values in R . The instance is a *Unique Games* instance if $R = L$ and π_e is a permutation for all $e \in E$. Given a labeling $\ell : U \cup V \rightarrow L \cup R$, an edge $e = (u, v)$ is said to be satisfied if $\pi_e(\ell(v)) = \ell(u)$.

Dinur and Shinkar gave a reduction from an instance of Label Cover with n vertices, 2-to-1 constraints and label set of size R to a graph of size $n4^R$. Perfectly satisfiable instances were mapped to 4-colorable graphs. Instances for which any labeling can satisfy only an $s(n)$ fraction of edges were mapped to graphs which did not have any independent sets of size $\text{poly}(s(n))$. Since the size of the graph produced by the reduction is exponential in R , they needed to assume that $R = O(\log n)$, to get hardness results. We give a more efficient reduction using Theorem 1.4 from Label Cover instances for which the projection constraints have special form. Our reduction is simpler to describe for the case 3-colorable graphs and starts with Unique Games instances. Hence for getting hardness result, we need to assume a conjecture similar to the Unique Games Conjecture with specific parameters.

► **Conjecture 1.1** ($((c(n), s(n), r(n))$ -UG Conjecture). *It is NP-Hard to distinguish between unique games instances (U, V, E, R, Π) on n vertices and $R = \mathbb{F}_3^{r(n)}$ from the following cases:*

- *YES Case : There is a labeling and a set $S \subseteq V$ of size $(1 - c(n))|V|$ such that all edges between vertices in S are satisfied.*
- *NO Case : For any labeling, at most $s(n)$ fraction of edges are satisfied.*

Khot and Regev [12] proved that the Unique Games Conjecture implies that for any constants $c, s \in (0, 1/2)$ there is a constant r such that (c, s, r) -UG Conjecture is true. We also require that the constraints of the Unique Games instance are full rank linear maps.

► **Definition 1.6** (Linear constraint). A constraint $\pi : R \rightarrow L$ is a *linear constraint* of iff $R = L = \mathbb{F}_3^r$, and π is a linear map of rank r .

The theorem below is obtained by replacing the long code by the low degree long code of degree $d = O(\log 1/\mu)$ in the reduction of Dinur and Shinkar. For want of space, the description of this reduction is deferred to the full version [6, Appendix A].

► **Theorem 1.7.** *There is a reduction from (c, s, r) -Unique Games instances G with n vertices, label set \mathbb{F}_3^r and linear constraints to graphs \mathcal{G} of size $n3^{rO(\log 1/\mu)}$ where $\mu = \text{poly}(s)$ such that*

- *If G belongs to the YES case of (c, s, r) -UG Conjecture then there is a subgraph of \mathcal{G} with fractional size $1 - c$ that is 3-colorable.*
- *If G belongs to the NO case of (c, s, r) -UG Conjecture then \mathcal{G} does not have any independent sets of fractional size μ .*

Due to the improved efficiency of the reduction, we are able to get hardness results even if the label cover instances have super-polylogarithmic sized label sets of size at most $2^{2^{O(\sqrt{\log \log n})}}$, while the reduction due to Dinur and Shinkar only works if the label set is of size at most $O(\log^c n)$ for some constant c . More precisely, suppose the UG conjecture were true for soundness $s(n)$ and alphabet size $R = 3^r$ that satisfy $\log_3 R = r = s(n)^{O(1)}$. Then,

the result of Dinur and Shinkar rules out polynomial time algorithms that find an independent set of relative size $1/\text{poly}(\log \log N)$. On the other hand, under the same assumption, our reduction rules out polynomial time algorithms that find an independent set of relative size $1/2^{\text{poly}(\log \log N)}$.

► **Corollary 1.8.** *Let c, s, r be functions such that $r(n) = \text{poly}(1/s(n))$. Assuming (c, s, r) -UG Conjecture on instances with linear constraints, given a graph on N vertices which has an induced subgraph of relative size $1 - c$ that is 3-colorable, no polynomial time algorithm can find an independent set of fractional size $2^{-\text{poly}(\log \log N)}$.*

We remark that we can improve the conclusion if Theorem 1.4 can be proved even when $d = O(\log \log 1/\mu)$.

2 Preliminaries

2.1 Low degree polynomials

We will be working over the field \mathbb{F}_3 . Let $\mathcal{P}_{r,d}$ be the set of degree d polynomials on r variables over \mathbb{F}_3 , with individual variable degrees at most 2. Let $\mathfrak{F}_r := \mathcal{P}_{r,2r}$. Note that \mathfrak{F}_r is the set of all functions from \mathbb{F}_3^r to \mathbb{F}_3 . \mathfrak{F}_r is a \mathbb{F}_3 -vector space of dimension 3^r and $\mathcal{P}_{r,d}$ is a subspace of dimension $r^{O(d)}$. The Hamming distance between f and $g \in \mathfrak{F}_r$, denoted by $\Delta(f, g)$, is the number of inputs on which f and g differ. For $S \subseteq \mathfrak{F}_r$, define $\Delta(f, S) := \min_{g \in S} \Delta(f, g)$. We say that f is δ -far from S if $\Delta(f, S) \geq \delta$ and f is δ -close to S otherwise. Given $f, g \in \mathfrak{F}_r$, the dot product between them is defined as $\langle f, g \rangle := \sum_{x \in \mathbb{F}_3^r} f(x)g(x)$. For a subspace $S \subseteq \mathfrak{F}_r$, the dual subspace is defined as $S^\perp := \{g \in \mathfrak{F}_r : \forall f \in S, \langle g, f \rangle = 0\}$. The following theorem relating dual spaces is well known.

► **Lemma 2.1.** $\mathcal{P}_{r,d}^\perp = \mathcal{P}_{r,2r-d-1}$.

We need the following Schwartz-Zippel-like Lemma for degree d polynomials over \mathbb{F}_3 .

► **Lemma 2.2** (Schwartz-Zippel lemma [10, Lemma 3.2]). *Let $f \in \mathbb{F}_3[x_1, \dots, x_r]$ be a non-zero polynomial of degree at most d with individual degrees at most 2. Then $\Pr_{a \in \mathbb{F}_3^r} [f(a) \neq 0] \geq 3^{-d/2}$.*

The following lemma is an easy consequence of Lemma 2.2.

► **Lemma 2.3.** *If p is a uniformly random polynomial from $\mathcal{P}_{r,d}$ then as a string of length 3^r over the alphabet \mathbb{F}_3 , p is $3^{\lfloor \log((d+1)/2) \rfloor}$ -wise independent.*

2.2 Fourier analysis of functions on subspace of low degree polynomials

► **Definition 2.4** (Characters). A character of $\mathcal{P}_{r,d}$ is a function $\chi : \mathcal{P}_{r,d} \rightarrow \mathbb{C}$ such that

$$\chi(0) = 1 \text{ and } \forall f, g \in \mathcal{P}_{r,d}, \chi(f + g) = \chi(f)\chi(g).$$

The following lemma lists the basic properties of characters.

► **Lemma 2.5.** *Let $\{1, \omega, \omega^2\}$ be the cube roots of unity and for $\beta \in \mathfrak{F}_r, f \in \mathcal{P}_{r,d}, \chi_\beta(f) := \omega^{\langle \beta, f \rangle}$, where $\langle \beta, f \rangle := \sum_{x \in \mathbb{F}_3^r} \beta(x)f(x)$.*

- *The characters of $\mathcal{P}_{r,d}$ are $\{\chi_\beta : \beta \in \mathfrak{F}_r\}$.*
- *For $\beta \in \mathcal{P}_{r,d}^\perp, \chi_\beta$ is the constant 1 function.*
- *For any $\beta, \beta' \in \mathfrak{F}_r, \chi_\beta = \chi_{\beta'}$ if and only if $\beta - \beta' \in \mathcal{P}_{r,d}^\perp$.*

- For any β , let $|\beta|$ be the size of the set of inputs on which β is non-zero. For any distinct $\beta, \beta' \in \mathfrak{F}_r$ with $|\beta|, |\beta'| < 3^{\lfloor (d+1)/2 \rfloor} / 2$, $\chi_\beta \neq \chi_{\beta'}$ since $\beta + \beta' \notin \mathbb{P}_{r,d}^\perp$.
- $\forall \beta, \exists \beta'$ such that $\beta - \beta' \in \mathbb{P}_{r,d}^\perp$ and $|\beta'| = \Delta(\beta, \mathbb{P}_{r,d}^\perp)$ (i.e., the constant 0 function is (one of) the closest function to β' in $\mathbb{P}_{r,d}^\perp$). We call such a β' a minimum support function for the coset $\beta + \mathbb{P}_{r,d}^\perp$.
- Characters forms an orthonormal basis for the vector space of functions from $\mathbb{P}_{r,d}$ to \mathbb{C} , under the inner product $\langle A, B \rangle := \mathbb{E}_{f \in \mathbb{P}_{r,d}} [A(f)\overline{B(f)}]$
- Any function $A : \mathbb{P}_{r,d} \rightarrow \mathbb{C}$ can be uniquely decomposed as

$$A(f) = \sum_{\beta \in \Lambda_{r,d}} \widehat{A}(\beta) \chi_\beta(f) \text{ where } \widehat{A}(\beta) := \mathbb{E}_{g \in \mathbb{P}_{r,d}} [A(g) \overline{\chi_\beta(g)}], \quad (2.1)$$

and $\Lambda_{r,d}$ is the set of minimum support functions, one for each of the cosets in $\mathfrak{F}_r / \mathbb{P}_{r,d}^\perp$, with ties broken arbitrarily.

- Parseval's identity: For any function $A : \mathbb{P}_{r,d} \rightarrow \mathbb{C}$,

$$\sum_{\beta \in \Lambda_{r,d}} |\widehat{A}(\beta)|^2 = \mathbb{E}_{f \in \mathbb{P}_{r,d}} [|A(f)|^2]. \quad (2.2)$$

In particular, if $A : \mathbb{P}_{r,d} \rightarrow \{1, \omega, \omega^2\}$,

$$\sum_{\beta \in \Lambda_{r,d}} |\widehat{A}(\beta)|^2 = 1. \quad (2.3)$$

► **Definition 2.6 (Influence).** For a function $A : \mathbb{P}_{r,d} \rightarrow \mathbb{C}$ and a number $k < 3^{\lfloor (d+1)/2 \rfloor} / 2$, the degree k influence of $a \in \mathbb{F}_3^r$ is defined as

$$\text{Inf}_a^{\leq k}(A) = \sum_{\beta \in \Lambda_{r,d} : \beta(a) \neq 0 \text{ and } |\beta| \leq k} |\widehat{A}(\beta)|^2.$$

► **Definition 2.7 (Dictator).** A function $A : \mathbb{P}_{r,d} \rightarrow \mathbb{C}$ is a dictator if there exists $x \in \mathbb{F}_3^r$ and $\widehat{A}_0, \widehat{A}_1, \widehat{A}_2 \in \mathbb{C}$ such that A can be written as $A = \widehat{A}_0 + \widehat{A}_1 \chi_{e_x} + \widehat{A}_2 \chi_{2e_x}$ where $e_x : \mathbb{F}_3^r \rightarrow \mathbb{F}_3$ the indicator function for x .

The following lemma which follows from the results of Guruswami *et al.* [9], will be crucial for our proofs.

► **Lemma 2.8.** If $\alpha : \mathbb{F}_3^r \rightarrow \mathbb{F}_3$ such that $\Delta(\alpha, \mathbb{P}_{r,2d}^\perp) > 3^{d/2}$ then

$$\left| \mathbb{E}_{p \in \mathbb{P}_{r,d}} \chi_\alpha(p^2) \right| \leq 3^{-\Omega(3^{d/9})}.$$

Proof. By definition, $|\mathbb{E}_{p \in \mathbb{P}_{r,d}} \chi_\alpha(p^2)| = |\mathbb{E}_{p \in \mathbb{P}_{r,d}} \omega^{\langle \alpha, p^2 \rangle}|$. If $\alpha : \mathbb{F}_3^r \rightarrow \mathbb{F}_3$ is such that $\Delta(\alpha, \mathbb{P}_{r,2d}^\perp) > 3^{d/2}$ then for a random $p \in \mathbb{P}_{r,d}$, $\langle \alpha, p^2 \rangle$ is $3^{-\Omega(3^{d/9})}$ -close to the uniform distribution on \mathbb{F}_3 according to [9, Lemma 3.1 and 3.4]. ◀

3 Derandomized $K_3^{\otimes R}$

Alon *et al.* [1] proved Theorem 1.1 by using the following lemma.

► **Lemma 3.1.** *There is constant K such that the following holds: If $A : \mathbb{F}_3^R \rightarrow \{0, 1\}$ satisfies*

$$\sum_{|\alpha|>1} |\widehat{A}_\alpha|^2 \leq \varepsilon \text{ and } \widehat{A}_0 = \delta$$

then there exists a dictator $B : \mathbb{F}_3^R \rightarrow \{0, 1\}$ such that

$$\|A - B\|_2 \leq \frac{K\varepsilon}{\delta - \delta^2 - \varepsilon}.$$

The above lemma was proved using the following hypercontractive inequality.

► **Lemma 3.2.** *There is a constant C such that for any function $A : \mathbb{F}_3^R \rightarrow \mathbb{C}$ with $\widehat{A}_\alpha = 0$ when $|\alpha| > t$,*

$$\|A\|_4 \leq C^t \|A\|_2.$$

Our proof of Theorem 1.2 will use a similar lemma for functions on the subspace $P_{r,2d}$.

► **Lemma 3.3.** *There is a constant K such that the following holds: If $A : P_{r,2d} \rightarrow \{0, 1\}$ satisfies*

$$\sum_{|\alpha|>1} |\widehat{A}_\alpha|^2 \leq \varepsilon \text{ and } \widehat{A}_0 = \delta$$

then there exists a dictator $B : P_{r,2d} \rightarrow \{0, 1\}$ such that

$$\|A - B\|_2 \leq \frac{K\varepsilon}{\delta - \delta^2 - \varepsilon}.$$

The above lemma follows from hypercontractive inequalities over $P_{r,2d}$ stated below, in exactly the same way as Alon *et al.* proves Lemma 3.1 from Lemma 3.2.

► **Lemma 3.4.** *There is a constant C such that for $4t \leq 3^{d-1}$ and any function $A : P_{r,2d} \rightarrow \mathbb{C}$ with $\widehat{A}_\alpha = 0$ when $|\alpha| > t$,*

$$\|A\|_4 \leq C^t \|A\|_2.$$

Proof. Follows from Lemma 3.6 and Lemma 3.2. ◀

► **Definition 3.5 (Lift).** For a function $B : P_{r,2d} \rightarrow \mathbb{C}$ with the Fourier decomposition $B = \sum_{\alpha \in \Lambda_{r,d}} \widehat{B}_\alpha \chi_\alpha$, the lift of B denoted by B' is a function $B' : \mathfrak{F}_r \rightarrow \mathbb{C}$ with the Fourier decomposition $B' = \sum_{\alpha \in \Lambda_{r,d}} \widehat{B}_\alpha \chi_\alpha$. In the decomposition of B' , χ_α 's are functions with domain \mathfrak{F}_r .

► **Lemma 3.6.** *If $2kt \leq 3^{d-1}$ and $B : P_{r,2d} \rightarrow \mathbb{C}$ be a function such that $\widehat{B}_\alpha = 0$ when $|\alpha| > t$ then*

$$\|B\|_{2k} = \|B'\|_{2k}.$$

Proof. From the Lemma 2.2 and Lemma 2.1, we have that $\forall \alpha \in P_{r,2d}^\perp \setminus \{0\}$, $|\alpha| > 3^{d-1}$. So if $\exists \{\alpha_i, \beta_i\}_{i \in [k]}$ with $|\alpha_i|, |\beta_i| \leq t$, then

$$\sum_{i \in [k]} \alpha_i - \beta_i \in P_{r,2d}^\perp \Rightarrow \sum_{i \in [k]} \alpha_i - \beta_i = 0. \tag{3.1}$$

This is because $\sum_{i \in [t]} \alpha_i - \beta_i$ has support size at most $2kt < 3^{d-1}$. We use this fact to prove the theorem as follows:

$$\begin{aligned}
\|B\|_{2k}^{2k} &= \mathbb{E}_{f \in \mathbb{P}_{r,2d}} |B(f)|^{2k} = \mathbb{E}_{f \in \mathbb{P}_{r,2d}} \prod_{i \in [k]} B(f) \overline{B(f)} \\
&= \sum_{\alpha_1, \beta_1, \dots, \alpha_k, \beta_k \in \Lambda_{r,2d}} \left(\prod_{i \in [k]} \widehat{B}_{\alpha_i} \overline{\widehat{B}_{\beta_i}} \right) \mathbb{E}_{f \in \mathbb{P}_{r,2d}} \prod_{i \in [k]} \chi_{\alpha_i}(f) \overline{\chi_{\beta_i}(f)} \quad (\text{from (2.1)}) \\
&= \sum_{\substack{\alpha_1, \beta_1, \dots, \alpha_k, \beta_k \in \Lambda_{r,2d} \\ \sum_i \alpha_i - \beta_i \in \mathbb{P}_{r,2d}^\perp}} \prod_{i \in [k]} \widehat{B}_{\alpha_i} \overline{\widehat{B}_{\beta_i}} \\
&= \sum_{\substack{\alpha_1, \beta_1, \dots, \alpha_k, \beta_k \in \Lambda_{r,2d} \\ \sum_i \alpha_i - \beta_i = 0}} \prod_{i \in [k]} \widehat{B}_{\alpha_i} \overline{\widehat{B}_{\beta_i}} \quad (\text{from (3.1)}) \\
&= \sum_{\alpha_1, \beta_1, \dots, \alpha_k, \beta_k \in \Lambda_{r,2d}} \left(\prod_{i \in [k]} \widehat{B}_{\alpha_i} \overline{\widehat{B}_{\beta_i}} \right) \mathbb{E}_{f \in \mathfrak{F}_r} \prod_{i \in [k]} \chi_{\alpha_i}(f) \overline{\chi_{\beta_i}(f)} \\
&= \mathbb{E}_{f \in \mathfrak{F}_r} \prod_{i \in [k]} B'(f) \overline{B'(f)} = \mathbb{E}_{f \in \mathfrak{F}_r} |B'(f)|^{2k} = \|B'\|_{2k}^{2k}
\end{aligned}$$

◀

3.1 Proof of Theorem 1.2

Proof of 1. For $f \in V$, consider the set $\{f, f+1, f+2\} \subseteq V$. These sets form a partition of V and are triangles in the graph. Hence $\delta \leq 1/3$. ◀

Proof of 2. Let $A : \mathbb{P}_{r,2d} \rightarrow \{0, 1\}$ be the indicator set of the independent set of size $\delta|V|$. By Parseval's equation (2.2) and the fact that $\widehat{A}_0 = \delta$, we have that

$$\sum_{\alpha \in \Lambda_{r,2d} \setminus \{0\}} |\widehat{A}_\alpha|^2 = \delta - \delta^2. \quad (3.2)$$

Since A is an independent set,

$$\mathbb{E}_{p \in \mathbb{P}_{r,d}, a \in \mathbb{F}_3, f \in \mathbb{P}_{r,2d}} A(f) A(f + a(p^2 + 1)) = \sum_{\alpha \in \Lambda_{r,2d}} |\widehat{A}_\alpha|^2 \mathbb{E}_{p \in \mathbb{P}_{r,d}, a \in \mathbb{F}_3} \chi_\alpha(a(p^2 + 1)) = 0.$$

Taking the real parts of the equation on both sides and rearranging, we get

$$\sum_{\alpha \in \Lambda_{r,2d} \setminus \{0\}} |\widehat{A}_\alpha|^2 \operatorname{Re} \left(\mathbb{E}_{p \in \mathbb{P}_{r,d}} \chi_\alpha(p^2 + 1) \right) = -\delta^2. \quad (3.3)$$

Let T be a random variable such that $\Pr[T = \alpha] = |\widehat{A}_\alpha|^2 / (\delta - \delta^2)$ and X be the random variable $X(T) = \operatorname{Re}(\mathbb{E}_{p \in \mathbb{P}_{r,d}, a \in \mathbb{F}_3} \chi_\alpha(a(p^2 + 1)))$. From (3.2) and (3.3), we have that

$$\mathbb{E} X = \frac{-\delta}{1 - \delta}.$$

Since p is a random degree d polynomial, it is $3^{d/2}$ -wise independent from Lemma 2.3. So if

$|T| \leq 3^{d/2}$ then

$$\begin{aligned} & \left| \operatorname{Re} \left(\mathbb{E}_{p \in \mathbb{P}_{r,d}, \alpha \in \mathbb{F}_3} \chi_\alpha(a(p^2 + 1)) \right) \right| \\ &= \left| \frac{1}{2} \operatorname{Re} \left(\left(\frac{\omega^2 - 1}{3} \right)^{|\alpha|_1} \left(\frac{\omega - 1}{3} \right)^{|\alpha|_2} + \left(\frac{\omega - 1}{3} \right)^{|\alpha|_1} \left(\frac{\omega^2 - 1}{3} \right)^{|\alpha|_2} \right) \right| \leq \left(\frac{1}{\sqrt{3}} \right)^{|\alpha|} \end{aligned}$$

where $|\alpha|_a = \{x \in \mathbb{F}_3^r : \alpha(x) = a\}$.

If $|T| > 3^{d/2}$, we know from Lemma 2.8 that $|X(T)| \leq 3^{-\Omega(3^{d/9})}$.

Note that for T with $|T| = 1$, $X(T) = -1/2$. For T with $|T| = 2$, $X(T) \geq 0$. For T with $|T| \geq 3$, $X(T) \geq \frac{-1}{3\sqrt{3}}$. So if $\mathbb{E}X = -1/2$ then $\Pr[|T| = 1] = 1$. So A is a Boolean valued function with non zero Fourier coefficients of supports only 0 and 1. Using arguments similar to Proof of [1, Lemma 2.3], it can be shown that there is an $x \in \mathbb{F}_3^r$ such that $A(f)$ only depends on $f(x)$. ◀

Proof of 3. Suppose $\delta = 1/3 - \varepsilon$. First we show that most of Fourier weights are concentrated in the first two levels

► **Lemma 3.7.**

$$\sum_{\alpha \in \Lambda_{r,2d}: |\alpha| > 1} |\widehat{A}_\alpha|^2 \leq 2\varepsilon$$

Proof. Consider the random variables X and T defined in the Proof of 2. Since $\delta = 1/3 - \varepsilon$ and since $\varepsilon < 1/3$, $\mathbb{E}X = -1/2 + \varepsilon$. Let Y be the random variable $X + 1/2$. Note that $Y \geq 0$ and when $Y > 0$, $Y \geq 1/6$. Therefore by Markov, $\Pr[Y > 0] \leq 6\varepsilon$ and

$$\sum_{\alpha \in \Lambda_{r,2d}: |\alpha| > 1} |\widehat{A}_\alpha|^2 \leq (\delta - \delta^2) \Pr[Y > 0] \leq 2\varepsilon.$$

Then we use Lemma 3.3 to obtain the result. ◀

4 Derandomized Majority is Stablest

In this section, we prove Theorem 1.4. The graphs described in Theorem 1.4 and Theorem 1.3 can be viewed as Cayley graphs on a suitable group. For the proof, we will need bounds on the eigenvalues of these Cayley graphs. For a group G , \mathbb{R}^G denotes the vector space of real valued functions on G .

► **Definition 4.1** (Cayley Operator). For a group G with operation $+$, an operator $M : \mathbb{R}^G \rightarrow \mathbb{R}^G$ is a *Cayley operator* if there is a distribution μ on G such that for any function $A : G \rightarrow \mathbb{R}$,

$$(MA)(f) = \mathbb{E}_{\eta \in \mu} A(f + \eta).$$

It is easy to see that a character $\chi : G \rightarrow \mathbb{C}$ is an eigenvector of M with eigenvalue $\mathbb{E}_{\eta \in \mu} \chi(\eta)$.

► **Definition 4.2.** We define the following Cayley operators:

1. For the group \mathbb{F}_3 , let $T : \mathbb{R}^{\mathbb{F}_3} \rightarrow \mathbb{R}^{\mathbb{F}_3}$ be the Cayley operator corresponding to the distribution μ that is uniform on $\mathbb{F}_3 \setminus \{0\}$. Let λ be the second largest eigenvalue in absolute value of T .

2. For the group \mathfrak{F}_r , let $T_r : \mathbb{R}^{\mathfrak{F}_r} \rightarrow \mathbb{R}^{\mathfrak{F}_r}$ be the Cayley operator corresponding to the distribution μ_r that is uniform on $\{f \in \mathfrak{F}_r : f^{-1}(0) = \emptyset\}$. Let $\lambda_r(\alpha)$ be the eigenvalue of T_r corresponding to the eigenvector χ_α , for $\alpha \in \mathfrak{F}_r$.
3. For the group $\mathbb{P}_{r,2d}$, let $T_{r,d} : \mathbb{R}^{\mathbb{P}_{r,2d}} \rightarrow \mathbb{R}^{\mathbb{P}_{r,2d}}$ be the Cayley operator corresponding to the distribution $\mu_{r,2d}$ of choosing a uniformly random element in $\{p^2 + 1, -p^2 - 1\}$ where $p \in \mathbb{P}_{r,2d}$ is chosen uniformly at random. Let $\lambda_{r,d}(\alpha)$ be the eigenvalue of $T_{r,d}$ corresponding to χ_α , for $\alpha \in \mathfrak{F}_r$.
4. For the group $\mathbb{P}_{r,2d}$, let $S_{r,d} : \mathbb{R}^{\mathbb{P}_{r,2d}} \rightarrow \mathbb{R}^{\mathbb{P}_{r,2d}}$ be the Cayley operator corresponding to the distribution of $a \cdot \prod_{i=1}^d (\ell_i - 1)(\ell_i - 2)$ where ℓ_1, \dots, ℓ_d are linearly independent degree 1 polynomials chosen uniformly at random and a is randomly chosen from \mathbb{F}_3 . Let $\rho_{r,d}(\alpha)$ be the eigenvalue of $S_{r,d}$ corresponding to χ_α , for $\alpha \in \mathfrak{F}_r$.

Now we will list some known bounds of the eigenvalues of the above operators. It is easy to see that λ is a constant < 1 . Since \mathbb{F}_3^R can be identified with \mathfrak{F}_r , $T^{\otimes R}$ can be identified with T_r . Hence we have the following lemma.

► **Lemma 4.3.**

$$|\lambda_r(\alpha)| \leq |\lambda|^{|\alpha|}.$$

► **Lemma 4.4.** For $\alpha \in \Lambda_{r,2d}$,

$$|\lambda_{r,d}(\alpha)| \begin{cases} = |\lambda_r(\alpha)| & \text{if } |\alpha| \leq 3^{d/2} \\ \leq 3^{-3^{C_1 d}} & \text{otherwise.} \end{cases} \quad (4.1)$$

Proof. The first case follows from the fact that a random element η according to $\mu_{r,2d}$ (the distribution that defines $T_{r,d}$) is $3^{d/2}$ -wise independent (see Lemma 2.3) as a string of length 3^r over alphabet \mathbb{F}_3 . The latter case follows from Lemma 2.8. ◀

We will derive bounds on the eigenvalues of $S_{r,d}$ using the results of Haramaty *et al.* [10]. Haramaty *et al.* analyses the following test for checking whether a polynomial is of degree $2r - 2d - 1$: Choose a random affine subspace S of dimension $r - d$ and check if the polynomial is of degree $2r - 2d - 1$ on S . Note that for any $\alpha \in \mathbb{P}_{r,2r-2d-1}$ and subspace S of dimension $r - d$, $\sum_{x \in S} \alpha(x) = 0$. Hence this test is equivalent to choosing $\ell_1, \dots, \ell_d \in \mathbb{P}_{r,1}$ that are linearly independent and checking whether $\langle \alpha, \prod_{i=1}^d (\ell_i - 1)(\ell_i - 2) \rangle \neq 0$. Haramaty *et al.* proved the following lemma.

► **Lemma 4.5.** There exists constants C_1, C_2 such that

$$\Pr_{\ell_i} \left[\left\langle \alpha, \prod_{i=1}^d (\ell_i - 1)(\ell_i - 2) \right\rangle = 0 \right] \leq \max \left\{ 1 - \frac{C_1 \Delta(\alpha, \mathbb{P}_{r,2r-2d-1})}{3^d}, C_2 \right\}$$

where $\ell_1, \dots, \ell_d \in \mathbb{P}_{r,1}$ are random linearly independent polynomials.

► **Lemma 4.6.** There exists constants C'_1, C'_2 such that, for $\alpha \in \Lambda_{r,2d}$,

$$1 - \frac{2|\alpha|}{3^d} \leq |\rho_{r,d}(\alpha)| \leq \max \left\{ 1 - \frac{C'_1 \Delta(\alpha, \mathbb{P}_{r,2r-2d-1})}{3^d}, C'_2 \right\} \quad (4.2)$$

Proof. First we will prove the lower bound. By definition

$$\rho_{r,d}(\alpha) = \mathbb{E}_{\ell_i, a} \omega^{a \cdot \sum_x \alpha(x)} \prod_{i=1}^d (\ell_i(x) - 1)(\ell_i(x) - 2).$$

For any x in support of α , the probability that $\prod_{i=1}^d (\ell_i(x) - 1)(\ell_i(x) - 2) \neq 0$ is $1/3^d$. Hence by union bound, $\prod_{i=1}^d (\ell_i(x) - 1)(\ell_i(x) - 2) = 0$ for every x in support of α with probability $1 - |\alpha|/3^d$ and when this happens the expectation is 1. Also note that the quantity inside the expectation has absolute value 1.

For proving the upper bound we will use Lemma 4.5. Let p_{acc} be the probability mentioned in Lemma 4.5. Then

$$\rho_{r,d}(\alpha) = \mathbb{E}_{\ell_{i,a}} \omega^{\alpha \langle \prod_{i=1}^d (\ell_i - 1)(\ell_i - 2) \rangle} = p_{\text{acc}} + \frac{1 - p_{\text{acc}}}{2} (\omega + \omega^2) = \frac{3}{2} p_{\text{acc}} - \frac{1}{2}.$$

From the above equation and Lemma 4.5, the constants C'_1, C'_2 can be obtained. ◀

► **Lemma 4.7.** For $A, B : \mathbb{P}_{r,2d} \rightarrow [0, 1]$, let $A' := S_{r,d}^t A$ and similarly define B' . Then

$$|\langle A, T_{r,d} B \rangle - \langle A', T_{r,d} B' \rangle| \leq 2dt/3^d$$

Proof.

$$\begin{aligned} |\langle A, T_{r,d} B \rangle - \langle A', T_{r,d} B' \rangle| &\leq |\langle A, T_{r,d} B \rangle - \langle A, T_{r,d} B' \rangle| \\ &\quad + |\langle A, T_{r,d} B' \rangle - \langle A', T_{r,d} B' \rangle| \\ &= |\langle A - \mathbb{E} A, T_{r,d}(1 - S_{r,d}^t)(B - \mathbb{E} B) \rangle| \\ &\quad + |\langle T_{r,d}(1 - S_{r,d}^t)(A - \mathbb{E} A), B' - \mathbb{E} B' \rangle| \\ &\leq \|T_{r,d}(1 - S_{r,d}^t)(B - \mathbb{E} B)\| + \|T_{r,d}(1 - S_{r,d}^t)(A - \mathbb{E} A)\| \\ &\leq 2td/3^d \end{aligned}$$

The last step follows from the fact that the operators $T_{r,d}, (1 - S_{r,d}^t)$ have the same set of eigenvectors and the largest eigenvalue in absolute value of $T_{r,d}(1 - S_{r,d}^t)$ is $2td/3^d$ from Lemma 4.4 and Lemma 4.6. ◀

Theorem 1.4 will follow from the following lemma.

► **Lemma 4.8.** $\forall \varepsilon > 0, \exists k = O(1/\varepsilon^2), d = O(\log(1/\varepsilon))$ such that the following holds: if $A, B : \mathbb{P}_{r,2d} \rightarrow [0, 1]$ then $\exists \mathcal{A}, \mathcal{B} : \mathfrak{F}_r \rightarrow [0, 1]$ such that

1. $|\mathbb{E} A - \mathbb{E} \mathcal{A}|, |\mathbb{E} B - \mathbb{E} \mathcal{B}| \leq \varepsilon,$
2. For all $x \in \mathbb{F}_3^r, k' \leq k,$

$$\begin{aligned} \text{Inf}_x^{\leq k'}(\mathcal{A}) &\leq \text{Inf}_x^{\leq k'}(A) + \varepsilon \\ \text{Inf}_x^{\leq k'}(\mathcal{B}) &\leq \text{Inf}_x^{\leq k'}(B) + \varepsilon \end{aligned}$$

3. $|\langle A, T_{r,d} B \rangle - \langle \mathcal{A}, T_r \mathcal{B} \rangle| \leq \varepsilon.$

Proof of Theorem 1.4. We will show that if Theorem 1.4 is false then Theorem 1.3 is also false. First using Lemma 4.8 with parameter $\varepsilon = \mu^{O(1)}$, we obtain functions $\mathcal{A}, \mathcal{B} : \mathfrak{F}_r \rightarrow [0, 1]$ such that

1. $\mathbb{E} \mathcal{A}, \mathbb{E} \mathcal{B} \geq \mu - \varepsilon,$
2. For all $x \in \mathbb{F}_2^r, k' \leq k,$

$$\text{Inf}_x^{\leq k'}(\mathcal{A}) \leq \delta + \varepsilon \quad \text{and} \quad \text{Inf}_x^{\leq k'}(\mathcal{B}) \leq \delta + \varepsilon$$

3. $|\langle \mathcal{A}, T_r \mathcal{B} \rangle| \leq |\langle A, T_{r,d} B \rangle| + \varepsilon.$

Now applying Theorem 1.3 to the functions \mathcal{A}, \mathcal{B} , we obtain that $|\langle \mathcal{A}, T_r \mathcal{B} \rangle| \geq \delta'$, where $\delta' = \mu^{O(1)}$. Hence $|\langle A, T_{r,d} B \rangle| \geq \delta' - \varepsilon$, and we set the parameters $\delta = \delta' - \varepsilon, d = O(\log 1/\mu)$ and $k = O(\log 1/\mu)$. ◀

4.1 Proof of Lemma 4.8

For proving Lemma 4.8, crucially use the following lemma by Kane and Meka [11].

► **Lemma 4.9.** *Let $\xi : \mathbb{R} \rightarrow \mathbb{R}_+$ be the function $\xi(x) := (\max\{-x, x - 1, 0\})^2$. For any parameters $k \in \mathbb{N}$ and $\varepsilon \in (0, 1)$, there is a $d = O(\log(k/\varepsilon))$ such that the following holds: If the polynomial $P : \mathfrak{F}_r \rightarrow \mathbb{R}$ satisfies $\|P\| \leq 1$ and $\widehat{P}(\alpha) = 0$ for $\alpha \in \Lambda_{r,d}$ such that $|\alpha| > k$, then*

$$\left| \mathbb{E}_{f \in \mathfrak{F}_r} \xi(P(f)) - \mathbb{E}_{f \in \mathbb{P}_{r,d}} \xi(P(f)) \right| \leq \varepsilon.$$

► **Remark.** For proving Lemma 4.9, a generalization of [11, Lemma 4.1] to polynomials of the form $P : \{1, \omega, \omega^2\}^R \rightarrow \mathbb{R}$ is required. However, we observe that the polynomials we consider are real-valued $P : \mathfrak{F}_r \rightarrow \mathbb{R}$ and hence satisfy $\widehat{P}(\alpha) = \overline{P(-\alpha)}$.

Using this observation, the proof of [11, Lemma 4.1] generalizes to our setting (the above property is preserved throughout the proof). The result of [11] also requires an earlier result of Diakonikolas, Gopalana, Jaiswal, Servedio, and Viola [4] on fooling Linear Threshold functions (LTFs) with sample spaces of bounded independence. This proof also goes through for Thresholds of real-valued linear functions defined on variables that are uniformly distributed in $\{1, \omega, \omega^2\}$.²

Proof of Lemma 4.8. Let $t = \frac{3^d \log(10/\varepsilon)}{2k}$, and $A_1 = S_{r,d}^t A, B_1 = S_{r,d}^t B$. Then from Lemma 4.7

$$|\langle A, T_{r,d} B \rangle - \langle A_1, T_{r,d} B_1 \rangle| \leq 2dt/3^d \quad (4.3)$$

and similarly for B_1 . Let k be a number $< 3^{d/2}$ and $A_2 = \text{Re}(A_1^{\leq k})$. Using the fact that A_1 is real valued,

$$\|A_1 - A_2\| \leq \|A_1 - A_1^{\leq k}\| \leq (1 - 2k/3^d)^t \leq e^{-2tk/3^d} = \varepsilon/10 \quad (4.4)$$

Let $A_3 : \mathfrak{F}_r \rightarrow \mathbb{R}$ be defined as $A_3 := \text{Re}((A_1^{\leq k})')$ where $(A_1^{\leq k})'$ is the lift of $A_1^{\leq k}$. Since a random degree d polynomial is $3^{d/2}$ -wise independent,

$$\langle A_2, T_{r,d} B_2 \rangle = \langle A_3, T_r B_3 \rangle \quad (4.5)$$

Note that A_3 may not be a $[0, 1]$ -valued function. But since A is $[0, 1]$ -valued, so is A_1 . Let $\xi : \mathbb{R} \rightarrow \mathbb{R}_+$ be the function $\xi(x) := (\max\{-x, x - 1, 0\})^2$. Notice that $\mathbb{E}_f \xi \circ A(f)$ gives the ℓ_2^2 distance of A from $[0, 1]$ -valued functions. Using Lemma 4.9, for $d = O(\log(k/\varepsilon))$,

$$\left| \mathbb{E}_{f \in \mathbb{P}_{r,2d}} \xi(A_2(f)) - \mathbb{E}_{f \in \mathfrak{F}_r} \xi(A_3(f)) \right| \leq \varepsilon/10 \quad (4.6)$$

and similarly for B_2 . Hence there exists functions $\mathcal{A}, \mathcal{B} : \mathfrak{F}_r \rightarrow [0, 1]$ such that

1. $\|\mathbb{E} A - \mathbb{E} \mathcal{A}\| \leq \|A_1' - \mathcal{A}\| \leq \varepsilon$ (similarly for B),
2. For all $x \in \mathbb{F}_3^r, k' \leq k, \text{Inf}_x^{\leq k'}(\mathcal{A}) \leq \text{Inf}_x^{\leq k'}(A) + \varepsilon$ (similarly for B),
3. $|\langle A, T_{r,d} B \rangle - \langle \mathcal{A}, T_r \mathcal{B} \rangle| \leq \varepsilon$.

◀

² Such a function is the sign of a “linear polynomial” of the form $(\sum_{i=1}^R \alpha_i x_i + \overline{\alpha_i x_i}) - \theta$ for $\theta \in \mathbb{R}$.

References

- 1 Noga Alon, Irit Dinur, Ehud Friedgut, and Benny Sudakov. Graph products, fourier analysis and spectral techniques. *Geometric and Functional Analysis GFA*, 14(5):913–940, 2004.
- 2 Boaz Barak, Parikshit Gopalan, Johan Håstad, Raghu Meka, Prasad Raghavendra, and David Steurer. Making the long code shorter. In *Proc. 53th IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 370–379, 2012.
- 3 Arnab Bhattacharyya, Swastik Kopparty, Grant Schoenebeck, Madhu Sudan, and David Zuckerman. Optimal testing of Reed-Muller codes. In *Proc. 51st IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 488–497, 2010.
- 4 Ilias Diakonikolas, Parikshit Gopalan, Ragesh Jaiswal, Rocco A. Servedio, and Emanuele Viola. Bounded independence fools halfspaces. *SIAM J. Computing*, 39(8):3441–3462, 2010. (Preliminary version in *50th FOCS*, 2009).
- 5 Irit Dinur and Venkatesan Guruswami. PCPs via low-degree long code and hardness for constrained hypergraph coloring. In *Proc. 54th IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 340–349, 2013.
- 6 Irit Dinur, Prahladh Harsha, Srikanth Srinivasan, and Girish Varma. Derandomized graph product results using the low degree long code. arXiv:1411.3517, 2014.
- 7 Irit Dinur, Elchanan Mossel, and Oded Regev. Conditional hardness for approximate coloring. *SIAM J. Computing*, 39(3):843–873, 2009. (Preliminary version in *38th STOC*, 2006).
- 8 Irit Dinur and Igor Shinkar. On the conditional hardness of coloring a 4-colorable graph with super-constant number of colors. In Maria J. Serna, Ronen Shaltiel, Klaus Jansen, and José D. P. Rolim, editors, *Proc. 13th International Workshop on Randomization and Approximation Techniques in Computer Science (APPROX)*, volume 6302 of *LNCS*, pages 138–151. Springer, 2010.
- 9 Venkat Guruswami, Prahladh Harsha, Johan Håstad, Srikanth Srinivasan, and Girish Varma. Super-polylogarithmic hypergraph coloring hardness via low-degree long codes. In *Proc. 46th ACM Symp. on Theory of Computing (STOC)*, pages 614–623, 2014.
- 10 Elad Haramaty, Amir Shpilka, and Madhu Sudan. Optimal testing of multivariate polynomials over small prime fields. *SIAM J. Computing*, 42(2):536–562, 2013. (Preliminary version in *52nd FOCS*, 2011).
- 11 Daniel M. Kane and Raghu Meka. A PRG for Lipschitz functions of polynomials with applications to sparsest cut. In *Proc. 45th ACM Symp. on Theory of Computing (STOC)*, pages 1–10, 2013.
- 12 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\epsilon$. *J. Computer and System Sciences*, 74(3):335–349, 2008. (Preliminary version in *18th IEEE Conference on Computational Complexity*, 2003).
- 13 Subhash Khot and Rishi Saket. Hardness of coloring 2-colorable 12-uniform hypergraphs with $2^{(\log n)^{\Omega(1)}}$ colors. In *Proc. 55th IEEE Symp. on Foundations of Comp. Science (FOCS)*, pages 206–215, 2014.
- 14 Raghu Meka and David Zuckerman. Pseudorandom generators for polynomial threshold functions. *SIAM J. Computing*, 42(3):1275–1301, 2013. (Preliminary Version in *42nd STOC*, 2010).
- 15 Girish Varma. A note on reducing uniformity in Khot-Saket hypergraph coloring hardness reductions. arXiv:1408.0262, 2014.

Space-efficient Basic Graph Algorithms

Amr Elmasry¹, Torben Hagerup², and Frank Kammer²

- 1 Department of Computer Engineering and Systems
Alexandria University, Alexandria 21544, Egypt
elmasry@mpi-inf.mpg.de
- 2 Institut für Informatik, Universität Augsburg
86135 Augsburg, Germany
{hagerup,kammer}@informatik.uni-augsburg.de

Abstract

We reconsider basic algorithmic graph problems in a setting where an n -vertex input graph is read-only and the computation must take place in a working memory of $O(n)$ bits or little more than that. For computing connected components and performing breadth-first search, we match the running times of standard algorithms that have no memory restrictions, for depth-first search and related problems we come within a factor of $\Theta(\log \log n)$, and for computing minimum spanning forests and single-source shortest-paths trees we come close for sparse input graphs.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases graph algorithms, depth-first search, single-source shortest paths, register input model

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.288

1 Introduction

Motivated on the one hand by the increased prevalence of huge data collections (“big data”), and on the other hand by the emergence of small mobile devices and embedded systems that cannot be equipped with very large memories, recent years have seen a surge of interest in data structures and algorithms that treat memory space as a scarce resource.

The classical area of Turing machines that operate in logarithmic space is still very active [17, 18, 21, 22]. Practical concerns, however, lead us to focus on algorithms that run in near-linear time. Even for the fundamental s - t connectivity problem on directed graphs, the most space-efficient algorithm, due to Barnes et al. [7], needs $n/2^{O(\sqrt{\log n})}$ bits of memory when required to run in polynomial time. The bound is only slightly sublinear, and a nearly matching lower bound is known for the so-called NNJAG model [20]. In addition, Tompa [33] showed that certain natural algorithmic approaches to the problem require superpolynomial time if the number of bits available is $o(n)$. It therefore seems reasonable to accord algorithms that operate on general n -vertex graphs approximately n bits of working memory. In return, we would hope to match the time bounds of standard graph algorithms or to come close. For this to be possible, it is necessary to replace the Turing machine by a model closer to computing practice, and a number of such models have been proposed. Common to all of them is that access to the input is restricted in some way. In the *multi-pass streaming* model [27], the input can only be accessed in a purely sequential fashion, and the main goal is to minimize the number of passes over the input. Another model [9] allows the input items to be permuted but not destroyed, and in the



restore model [13], the input may be temporarily modified during a computation, but must be restored to its original state.

In this paper we employ the *register input model* of Frederickson [24]. It features a read-only input memory and a write-only output medium. The computation proper takes place in a working memory of limited size. When stating that a problem can be solved with a certain number of bits, what we mean is that the working memory comprises that many bits. The input and working memories are divided into words of w bits for a fixed parameter w , arithmetic and logical operations on w -bit words take constant time, and random access to the input and working memories is provided. In the context of inputs of n words, we assume, as is common, that $w = \Theta(\log n)$ and, in particular, that w is large enough to allow all words in the input and working memories to be addressed.

A number of results are known for the register input model. For the fundamental problem of sorting n items, Pagter and Rauhe [29] described a comparison-based algorithm that, for every given s with $\log n \leq s \leq n/\log n$, runs in $O(n^2/s)$ time using $O(s)$ bits, and a matching lower bound of $\Omega(n^2)$ for the time-space product was established by Beame [8] for the strong *branching-program model*. Other researchers have considered selection [10, 12, 23, 24, 28, 31] and various problems in computational geometry [2, 4, 5, 6, 11, 16]. With one exception, discussed below, we are not aware of previous work that reduces the working space needed to process n -vertex graphs below $\Theta(n \log n)$ bits with only a modest penalty in the running time.

1.1 New Results

We describe a number of algorithms for the register input model, all of which input a directed or undirected graph (plus, possibly, other items). When discussing graph algorithms below, we always use n and m to denote the number of vertices and the number of edges, respectively, in the input graph.

A focal point of our work is depth-first search (DFS) (Section 3) and its applications (Section 4). We first show that a DFS can be carried out in $O((n+m) \log n)$ time with $(\log_2 3 + \epsilon)n$ bits, for arbitrary fixed $\epsilon > 0$. A very similar result was found independently by Asano et al. [3]. They need cn bits, for an unspecified constant $c > 2$, or $\Theta(mn)$ time, however. Relaxing the space bound to $O(n)$ bits, we can perform the DFS in just $O((n+m) \log \log n)$ time. We also show how to achieve linear time with $O(n \log \log n)$ bits and how to interpolate between the two latter results with the same time-space product. Our main technique can be viewed as employing an “approximate runtime stack”: With just $O(\log \log n)$ bits rather than $\Theta(\log n)$ bits for each vertex on the stack, we store only an approximation of its stack entry and only an approximation of its position on the stack and show how to execute the DFS in the face of the resulting uncertainty.

Some applications of DFS process the output of a DFS in reverse order. This is rarely highlighted, since reversing a sequence is implicit when the whole sequence fits in memory. When this is not the case, however, the operation can become a bottleneck. While reversing a sequence in general may be more expensive, we show how to run a DFS in reverse with only a modest penalty of $O(n \log \log n)$ additional bits. This allows us to compute topological sortings and strongly connected components in linear time with $O(n \log \log n)$ bits. Here the main technique employed is to keep enough information about a DFS to restart it in the middle and to use this repeatedly to reverse small pieces of its output, produced in reverse order, one at a time. Although the connected components of an undirected graph are usually computed by means of DFS, in Section 5 we observe that this bottleneck can be avoided and show how to compute the connected components in $O(n+m)$ time with $O(n)$ bits and how to carry out a breadth-first search within the same bounds.

Section 6 describes space-efficient versions of the algorithms of Prim and Dijkstra for computing a minimum spanning forest (MSF) and a single-source shortest-paths (SSSP) tree, respectively. To describe the algorithms, we introduce the notion of a priority queue with a *deletion budget*. A priority queue with a deletion budget uses less space than a usual priority queue, but can be used only for a certain time before it must be *refilled*, i.e., initialized anew. We give two algorithms for the MSF problem. The first one runs with $O(n)$ bits in $O(n+m \log n)$ time. The second algorithm uses more space, namely $O(n \log(2+m/n))$ bits, but matches the running time of $O(m + n \log n)$ of usual implementations of Prim's algorithm. Despite the pronounced similarities between Prim's and Dijkstra's algorithms, the SSSP problem appears more difficult in a space-efficient setting because the vertices of the SSSP tree cannot be output and forgotten as they are computed; rather, their distances from the source are needed later in the computation. We cannot store the distances, and in order to recompute them with any degree of efficiency, we must remember the SSSP tree, which needs $\Theta(n \log(2 + m/n))$ bits. While this number of bits is $O(n)$ for sparse graphs with $m = O(n)$, it degrades to $\Theta(n \log n)$ for dense graphs with $m = \Theta(n^2)$. Assume, by way of example, that the input graph is sparse and that we want to use only $O(n)$ bits. Then we can recompute the distances from the SSSP tree in batches of size $\Theta(n/\log n)$ in $O(n)$ time. Since we need to do this for $\Theta(\log n)$ batches for each of $\Theta(\log n)$ refillings of the priority queue, the total time becomes $O(m+n(\log n)^2)$. More generally, if $O(n(\log(2+m/n)+s(n)))$ bits are available, we achieve a time bound of $O(m + n \log n + n((\log n)/s(n))^2)$.

2 Preliminaries

It is customary to distinguish between adjacency matrices and adjacency lists, but not to specify the input format of a graph algorithm in any greater detail. This is because linear time and a linear number of words of working memory are sufficient to convert between any two reasonable adjacency-list representations—e.g., the edges may be reordered by means of radix sorting. In our setting, where we want to get by with $o(n \log n)$ bits of working memory, we have to be more specific about the input format.

Let $G = (V, E)$ be an input graph with n vertices and m edges. As is common, we always assume that $V = \{1, \dots, n\}$ and that, given $u \in V$, we can access the set $N(u)$ of neighbors of u (if G is undirected) or of outneighbors of u (if G is directed). For some algorithms it suffices to be able to iterate over $N(u)$ in constant time per vertex, the archetypical functionality provided by adjacency lists. For most of our algorithms, however, we need random access to $N(u)$. More precisely, given u and an integer k with $1 \leq k \leq |N(u)|$, we need constant-time access to the k th element of $N(u)$. In such cases we will indicate that the input graph must be represented via *adjacency arrays*.

Some algorithms have additional special requirements. When we state that an adjacency-array representation of an undirected graph has *cross pointers*, what we mean is that, given a vertex u and the position in $N(u)$ of a neighbor v of u , in constant time we can find the position of u in $N(v)$. Our algorithm for computing the strongly connected components of a directed graph assumes that, given a vertex u , we have access not only to its outneighbors, but also to its inneighbors. We will formulate this by stating that the input graph must be represented with *in/out adjacency lists* or *arrays*. Our algorithm for the single-source shortest-paths problem uses in/out adjacency arrays, say, $N_{\text{in}}(u)$ and $N_{\text{out}}(u)$ for $u \in V$, and requires the arrays $N_{\text{out}}(u)$, for $u \in V$, to be sorted consistently with a linear order on V that is either the natural order $1, \dots, n$ or is specified in the input. We will say that the input graph must be represented with *sorted adjacency arrays*. In addition, for each

$(u, v) \in E$, given u and the position of v in $N_{\text{out}}(u)$, we must be able to find the position of u in $N_{\text{in}}(v)$ in constant time—again, we will say that the representation must have cross pointers.

An inconspicuous but crucial role is played in most of our algorithms by a special case, characterized in the following lemma, of a data structure developed in [26].

► **Lemma 2.1.** *For every fixed $n \in \mathbb{N} = \{1, 2, \dots\}$, there is a dictionary that can store a subset A of $\{1, \dots, n\}$, each $a \in A$ with a string h_a of satellite data of $O(\log n)$ bits, in $O(n + \sum_{a \in A} |h_a|)$ bits such that membership in A can be tested in constant time for each element of $\{1, \dots, n\}$, h_a can be inspected in constant time for each $a \in A$, elements with their satellite data can be inserted in and deleted from A in constant amortized time, an operation `some_id` that returns an (arbitrary) element of A is supported in constant time, and an operation `all_ids` that returns all elements of A is supported in $O(|A| + 1)$ time.*

We sometimes want to store for each vertex v in a graph with n vertices and m edges an index into the adjacency array of v . Jensen’s inequality and Lemma 2.1 show that this can be done with $O(n \log(2 + m/n))$ bits.

3 Depth-First Search

A DFS of a directed graph $G = (V, E)$ steps through the vertices of G and *processes* each in turn if its processing has not already begun. The processing of a vertex $u \in V$ consists in stepping through its outgoing edges and, for each such edge (u, v) , *exploring* (u, v) and, if the processing of v has not already started, processing v recursively. Every vertex is processed exactly once, and every edge is explored exactly once. When the processing of a vertex $u \in V$ starts, we say that u is *discovered*.

It is customary to use a stack to keep track of the vertices whose processing has begun, but not yet ended, with vertices that were discovered more recently appearing closer to the top of the stack. When a vertex is discovered, it is pushed on the stack, and when its processing terminates, it is again at the top of the stack, from which it is popped. Following Cormen et al. [14], we call a vertex *white* if it has not yet been discovered, *gray* if its processing is underway, and *black* if its processing has ended.

Whenever an edge (u, v) is explored, u is at the top of the stack. If the exploration of (u, v) causes v to be pushed on the stack above u , i.e., if v is white just prior to the exploration of (u, v) , v becomes gray at that point and will remain gray and immediately above u on the stack until v is popped and turns black. Thus, whenever a vertex v appears immediately above another vertex u on the stack, (u, v) is an edge of E and the first edge out of u whose head is neither black nor stored below v on the stack.

As described so far, depth-first search does not do anything useful—it is just an “empty control structure”. Applications of DFS therefore augment the basic scheme with additional computational steps. Such steps can be executed, e.g., at the beginning and/or at the end of each processing of a vertex and/or at the exploration of each edge. If they are phrased as application-dependent *user* procedures *preprocess*, *postprocess*, *preexplore* and *postexplore*, DFS can be expressed via the code fragment below, which denotes the outdegree of a vertex u by $\text{deg}(u)$ and its k th outneighbor by $N(u)[k]$, for $k = 1, \dots, \text{deg}(u)$.

DFS:

```
for  $u := 1$  to  $n$  do  $color[u] := white$ ;
for  $u := 1$  to  $n$  do if  $color[u] = white$  then  $process(u)$ ;
```

The procedure *process* is defined as follows:

```

process(u):
  color[u] := gray;
  preprocess(u);
  k := 1;
  while k ≤ deg(u) do
    v := N(u)[k];
    preexplore(u, v, color[v]);
    if color[v] = white then process(v);
    postexplore(u, v);
    k := k + 1;
  postprocess(u);
  color[u] := black;

```

We view the problem to be solved as that of executing the correct sequence of calls of *preprocess*, *postprocess*, *preexplore* and *postexplore*. Of course, we exclude the time and space requirements of these procedures from our resource bounds, and we often ignore them in what follows so as not to clutter the picture.

The execution of the procedure *DFS* uses $\Theta(n)$ bits of working memory for the array *color*. However, the implicit run-time stack needed to keep track of partially executed calls of *process* may require $\Theta(n \log n)$ bits. As a first step towards more space-efficient solutions, we eliminate the explicit recursion from the procedure *process* and reformulate it below to manage its own run-time stack. The latter, denoted by *S*, stores not just vertices, but pairs consisting of a vertex *u* and an integer that indicates the number of the next edge out of *u* to be explored. Pushing a pair (*u*, *k*) on *S* is written $S \leftarrow (u, k)$, popping the top entry from *S* and storing its components in *u* and *k* is written $(u, k) \leftarrow S$, and *S* is tested for being nonempty with $S \neq \emptyset$.

```

process(u):
   $S \leftarrow (u, 1)$ ;
  while  $S \neq \emptyset$  do
     $(u, k) \leftarrow S$ ;
    color[u] := gray;
    if k ≤ deg(u) then
       $S \leftarrow (u, k + 1)$ ;
      if color[N(u)[k]] = white then  $S \leftarrow (N(u)[k], 1)$ ;
    else color[u] := black;

```

Informally, the presence on *S* of an entry of the form (*u*, *k*) signals that at some point, namely when (*u*, *k*) again becomes the top entry of *S*, the algorithm will proceed to either process the *k*th edge out of *u* or discover that $k > \text{deg}(u)$. In the first case, (*u*, *k*) is replaced by (*u*, *k* + 1) as the top entry on *S*. Although this is formulated above in terms of standard stack operations as a pop followed by a conditional push, our discussion will instead pretend that it happens as a test of the value of a field in the top entry on *S* followed by—depending on the outcome—an increment of that value or a pop.

3.1 A Simple DFS Algorithm

An entry on *S* can be represented in $\Theta(\log n)$ bits and, in general, needs that much space. Since *S* may grow to contain as many as *n* entries, the algorithm stated above requires

$\Theta(n \log n)$ bits of working space. The goal in this section is to reduce the space requirements to little more than n bits while incurring only a logarithmic penalty in the time bound.

► **Theorem 3.1.** *For every constant $\epsilon > 0$, a DFS of a graph with n vertices and m edges can be performed in $O((n + m) \log n)$ time with at most $(\log_2 3 + \epsilon)n$ bits.*

Proof. Without loss of generality, assume below that n is larger than a certain constant. Take $\lambda = \log_2 3$. Among other data structures detailed below, we need the array *color* of n entries drawn from $\{\text{white}, \text{gray}, \text{black}\}$. It is well-known and easy to see that *color* can be realized in at most $(\lambda + \epsilon/3)n$ bits so that individual entries can be tested and set in constant time (assume without loss of generality that $3/\epsilon$ is an integer that divides n and store each group of $3/\epsilon$ consecutive color values in $\lceil \log_2(3^{3/\epsilon}) \rceil = \lceil 3\lambda/\epsilon \rceil$ bits).

Compute q as a positive integer with $q = \Theta(n/\log n)$, chosen so that $2q$ entries on S take up at most $(\epsilon/3)n$ bits. At any given time, we partition the entries on S into $O(\log n)$ segments as follows: The bottommost q entries form the first segment, the next q entries form the second segment, and so on, with the last segment usually containing fewer than q entries. Let us call the last (most recently pushed) entry within each segment its *trailer*.

We remember only a part S' of the full stack S . S' always consists of the one or two last (most recent) segments of S and therefore, by the choice of q , requires no more than $(\epsilon/3)n$ bits of storage. In addition, we store all trailers present on S on a separate stack T of $O((\log n)^2)$ bits. T and various simple variables together can be stored in fewer than $(\epsilon/3)n$ bits, so the total space requirements are bounded by $(\lambda + \epsilon)n$ bits.

The algorithm works with S' exactly as the usual DFS algorithm works with S (of course, additionally manipulating trailers as appropriate), except in the following two special events: (1) When S' already contains $2q$ entries and a new entry is to be pushed on S , first the older of the two segments present on S' is dropped to make room for a new segment. (2) When S' loses its last entry due to a pop but S (and hence T) is not empty, the one or two topmost segments of S are restored and placed on S' , after which the normal execution resumes.

The restoration of a segment is performed as follows: First all gray vertices are recolored white. Then the DFS is restarted from the beginning, except that black vertices remain black and that the process operates *quietly*, i.e., the user procedures *preprocess*, *postprocess*, *preexplore* and *postexplore* are not executed. The restoration is continued until the top entries on S' and T coincide, at which point one (if there is only one) or two segments of S will have been restored on S' .

To see that the restoration is correct, recall that if u and v are successive vertices on S , (u, v) is the first edge out of u whose head is neither black nor stored on S below v . Thus all vertices that are pushed on S' during the restoration, except for the last trailer, will simply skip over their first outgoing edges, those that point to gray or black vertices, and push the first white vertex encountered—the correct next vertex on the stack—while coloring it gray. In particular, no vertices are ever popped, so no restoration will be called for during the restoration. The last trailer will skip over edges to gray or black vertices and reach the edge that was the next edge to be explored in the original DFS, which is resumed at that precise point.

To bound the number of restorations, consider the potential function $\Phi = \max\{q - |S'|, 0\}$, where $|S'|$ is the number of entries currently stored on S' . $\Phi = q$ initially and $\Phi \geq 0$ at all times, no push increases Φ , a pop increases Φ by at most 1, and a restoration decreases Φ from q to 0. As each of the n vertices is popped only once, the number of restorations is bounded by $(q + n)/q = O(\log n)$. It is obvious that a restoration can be executed in $O(n + m)$ time. The computation outside of restorations also runs in $O(n + m)$ time, as it is basically a standard DFS, so the total time comes to $O((n + m) \log n)$. ◀

3.2 Depth-First Search in Linear Time

This section describes a DFS procedure that works in $O(n+m)$ time and uses $O(n \log \log n)$ bits. Several notions carry over from the previous section: segments of $q = \Theta(n/\log n)$ entries on S , the stack S' that contains only the last one or two segments on S , and the restoration of the topmost segment of S when S' becomes empty. There are two main new ideas, explained in the following.

The first idea is to carry out a restoration not by restarting the DFS from scratch, but by using the trailer of the segment just below the top segment of S as a starting point (if S contains just one segment, restart the DFS from the beginning). Thus the goal is to restore just the top segment without going through the process of reconstructing the segments below it only to throw them away immediately after.

The idea expressed in the previous paragraph meets with a difficulty. Recall that in the algorithm of Section 3.1, the restoration begins by recoloring all gray vertices white, so that they are again eligible for being pushed on the stack. Such an operation would be too expensive in the present context. Besides, what we need is something different: a recoloring that is applied only to the vertices in the top segment of S , since only these should be allowed to enter S' . We achieve a similar effect by numbering the segments consecutively from bottom to top and introducing a table D with an entry for each vertex in V . Whenever a vertex $u \in V$ is pushed on S' , the number of the segment that it enters is stored permanently in $D[u]$. Since all segment numbers are $O(\log n)$, D can be stored in $O(n \log \log n)$ bits. In addition, we temporarily switch the meaning of the colors white and gray for the vertices in the top segment of S for the duration of the restoration.

The restoration process is modified as follows: At each exploration of an edge (u, v) , v is pushed on S' exactly if $D[v]$ indicates that v belongs to the top segment on S and v is gray. If v is pushed, it is colored white to prevent it from being pushed again later in the restoration. When the restoration is complete, the vertices in the restored top segment are all white, and they are recolored gray (their “true” color) before the original DFS resumes.

The second new idea serves to speed up the search for the correct edge out of a vertex u on the stack during a restoration. Ideally, we would like to know the integer k such that the pair (u, k) is stored on S , but we do not have the space to remember this information for all vertices. Define a vertex to be *big* if its degree exceeds m/q . For the at most q big vertices we store the relevant pairs explicitly. More precisely, the part of S maintained on T is extended to include not only the trailers, but also all pairs (u, k) , where u is big. During a restoration, the part of T above and including the topmost trailer is accessed from bottom to top, in synchrony with the restoration, so that the need to search through the outgoing edges of big vertices is eliminated—the correct value of k is found in constant time. For other vertices we store a rough, $O(\log \log n)$ -bit approximation of the relevant k . Compute l as a positive integer with $l = \Theta(\log n)$. For each pair (u, k) stored on S with $\deg(u) \geq 1$, we extend the table entry $D[u]$ to contain also the integer $f_u = \lfloor (k-1)/g_u \rfloor$, where $g_u = \lceil \deg(u)/l \rceil$. Informally, f_u indicates the number of groups of g_u edges out of u that have been completely explored. The restoration is changed to skip the processing of edges in such groups.

For all $u \in V$ with $\deg(u) \geq 1$, $f_u \leq (k-1)/g_u \leq (k-1)l/\deg(u) \leq \deg(u)l/\deg(u) = O(\log n)$, so $O(n \log \log n)$ bits still suffice for the extended table D . During a restoration, the search for the correct edge out of a vertex u that is not big can now be performed in $O(g_u) = O(1 + m/(ql)) = O(1 + m/n)$ time. A single restoration carries out the search for q vertices and therefore takes $O((1 + m/n)q) = O((n+m)/\log n)$ time. As in the proof of Theorem 3.1 and for the same reasons, the number of restorations is $O(\log n)$ and the time spent outside of restorations is $O(n+m)$, so the total time for the DFS is $O(n+m)$.

► **Lemma 3.2.** *A DFS of a graph with n vertices and m edges, represented via adjacency arrays, can be performed in $O(n + m)$ time with $O(n \log \log n)$ bits.*

3.3 An Upper-Bound Time-Space Tradeoff for DFS

In this section we give a tradeoff between time and space for DFS. Except for the explicit constant factor indicated in the space bound of Theorem 3.1, Theorem 3.3 subsumes Theorem 3.1 and Lemma 3.2, but its proof draws heavily on arguments presented in their proofs.

► **Theorem 3.3.** *For every function $t : \mathbb{N} \rightarrow \mathbb{N}$ such that $t(n)$ can be computed within the resource bounds of this theorem (e.g., in $O(n)$ time using $O(n)$ bits), a DFS of a graph with n vertices and m edges, represented via adjacency arrays, can be performed in $O((n + m)t(n))$ time with $O(n + n(\log \log n)/t(n))$ bits.*

Proof. Begin by computing a positive integer r with $r = \Theta(1 + \frac{\log n}{t(n)})$. Divide S into segments of $q = \Theta(n/\log n)$ consecutive entries each, as usual, but additionally and in the same manner divide S into *big segments* of qr consecutive entries each. Thus each big segment consists of r consecutive usual segments.

We use an algorithm that is a combination of the two algorithms described in Sections 3.1 and 3.2. Its top-level structure is nearly identical to that of the algorithm of Theorem 3.1, except that it employs big segments in place of usual segments. Thus information is maintained about up to two big segments, and occasionally a big segment needs to be restored, which is done in linear time by recoloring all gray vertices white and repeating the computation quietly until the topmost trailer on T has been pushed on S' . The number of such *big restorations* is $O(n/(qr)) = O((\log n)/(1 + \frac{\log n}{t(n)})) = O(t(n))$, so the time spent in big restorations is $O((n + m)t(n))$.

Between big restorations, the algorithm proceeds almost exactly as that of Lemma 3.2. The only differences are that the table D is implemented not as a simple array, but with the dictionary of Lemma 2.1, and that the entry in D of a vertex u is deleted from D when u no longer belongs to one of the two topmost big segments on S . Because of this, the number of bits needed by D is $O(n + qr \log \log n) = O(n + \frac{n \log \log n}{\log n} (1 + \frac{\log n}{t(n)})) = O(n + n(\log \log n)/t(n))$. Of course, the restoration should classify a gray vertex without an entry in D as not belonging to the segment under restoration, i.e., such a vertex should not be pushed on S' . The runtime analysis of Section 3.2 carries over to the present context and shows the time spent outside of big restorations to be $O(n + m)$. ◀

4 Reverse DFS with Applications

► **Lemma 4.1.** *If we can carry out a DFS \mathcal{S} of a directed graph with n vertices and m edges in time $t(n, m)$ with $s(n, m)$ bits, we can output the reverse of a sequence of symbolic representations of the user calls executed by \mathcal{S} in $O(t(n, m))$ time with $O(s(n, m) + n \log \log n)$ bits.*

Proof. Assume without loss of generality that $n \leq s(n, m) \leq n \log n$. For times $t_0 < \dots < t_r$ such that t_0 and t_r are the times of the beginning and the end of \mathcal{S} , $r = O(n(\log n)/s(n, m))$, and the number of pushes and pops executed on the stack S of \mathcal{S} during the time interval $I_j = (t_{j-1}, t_j)$ is $O(s(n, m)/\log n)$, for $j = 1, \dots, r$, use a simulation of an execution of \mathcal{S} to compute r , to label each vertex v with the pair $(d[v], f[v]) \in \{1, \dots, r\}^2$ such that v becomes gray during $I_{d[v]}$ and black during $I_{f[v]}$, and to record, for $j = 1, \dots, r$, the top entries H and H' of S at times t_{j-1} and t_j , the value \widehat{H} at time t_{j-1} of the deepest stack entry that changes during I_j , and the first and last user calls, if any, executed during I_j .

For $j = r, \dots, 1$, we now simulate \mathcal{S} during I_j , record a part of the sequence of (symbolic representations of) user calls executed during I_j , and output the reverse of that sequence, completed with its missing parts, while keeping track of the vertex colors during the corresponding reverse DFS. The missing parts are those calls $\text{preexplore}(u, v, \text{color}[v])$ for which $\text{color}[v] \neq \text{white}$ and the calls $\text{postexplore}(u, v)$ that immediately follow them. Without these calls, the sequence of calls executed during I_j fits in $O(s(n, m))$ bits. On the other hand, it is easy to reconstruct the calls missing between two successive recorded calls, essentially by traversing corresponding pieces of adjacency lists or arrays, and to output them in reverse order, $O(n/\log n)$ calls at a time; the details are left to the reader.

To simulate \mathcal{S} from time t_{j-1} onwards, we need the coloring of each vertex v at time t_{j-1} , which can be deduced from $(d[v], f[v])$. We cannot construct the stack valid at time t_{j-1} in its entirety, but since \mathcal{S} is to be simulated only until t_j , it suffices to construct the part of S between \widehat{H} and H , inclusive. We do this by a stack restoration similarly as for DFS: Starting from \widehat{H} , each vertex steps through its adjacency list or array, skipping over black and gray outneighbors, until it encounters a first white outneighbor and pushes it on the stack. The recorded stack entries allow us to know exactly when to stop the stack restoration and when to stop the simulation pertaining to I_j . Apart from a constant-factor simulation overhead, the only significant resources needed are $O(s(n, m))$ bits used for the restored stack and for user calls, $O(n \log r) = O(n \log \log n)$ bits to store vertex labels and colors, and the time consumed by the stack reconstructions. For $j = r, \dots, 1$, if the stack reconstruction prior to the simulation for I_j pushes an entry other than \widehat{H} for a vertex v on S , we must have $f[v] = j$, since otherwise \widehat{H} could not appear at the top of S during I_j . Except for at most r vertices, every vertex is therefore pushed on S in at most one reconstruction, so the time needed for all reconstructions is within a constant factor of the time consumed by \mathcal{S} . ◀

In particular, we can output the vertices of a graph G in reverse postorder with respect to a DFS forest of G . If G is directed and acyclic, this order is a topological sorting of G [32].

► **Theorem 4.2.** *Within the time and space bounds of a DFS of G , up to a constant factor, plus $O(n \log \log n)$ bits, the vertices of a directed acyclic n -vertex graph G can be output in the order of a topological sorting of G .*

We define the SCC problem as follows: Given a directed n -vertex graph $G = (V, E)$ with c strongly connected components (SCCs), output a sequence $(u_1, k_1), \dots, (u_n, k_n)$, where $\{u_1, \dots, u_n\} = V$ and k_1, \dots, k_n is a nondecreasing sequence of integers such that $1 \leq k_i \leq c$ for $i = 1, \dots, n$ and $k_i = k_j$, for $1 \leq i, j \leq n$, exactly if u_i and u_j belong to the same SCC of G . Combining Lemma 4.1 with a DFS-based SCC algorithm whose main procedure steps through the vertices in reverse postorder [1], one can easily show the theorem below.

► **Theorem 4.3.** *If a DFS of a directed graph with n vertices and m edges, represented with in/out adjacency lists or arrays, can be carried out in $t(n, m)$ time with $s(n, m)$ bits, then, given a directed graph G with n vertices and m edges, represented in the same way, the SCC problem can be solved for G in $O(t(n, m))$ time with $O(s(n, m) + n \log \log n)$ bits.*

5 Computing Connected Components and Breadth-First Search

We consider the following variants of the connected-components and breadth-first search (BFS) problems: The input is an undirected graph $G = (V, E)$ and, in the case of BFS, a permutation $(\pi(1), \dots, \pi(n))$ of V . The output is a sequence $(u_1, k_1), \dots, (u_n, k_n)$, where $n = |V|$, $\{u_1, \dots, u_n\} = V$, and k_1, \dots, k_n is a nondecreasing sequence of integers with the

following property: For the connected-components problem, $1 \leq k_i \leq c$ for $i = 1, \dots, n$, where c is the number of connected components of G , and $k_i = k_j$, for $1 \leq i, j \leq n$, exactly if u_i and u_j belong to the same connected component of G . For BFS, k_i is the distance in G from u_i to the first vertex in the sequence $(\pi(1), \dots, \pi(n))$ that belongs to the same connected component as u_i , for $i = 1, \dots, n$.

► **Theorem 5.1.** *The connected-components and BFS problems for an undirected graph with n vertices and m edges can be solved in $O(n + m)$ time with $O(n)$ bits.*

Proof sketch. For the connected-components problem, we explore the graph using the same (white \rightarrow gray \rightarrow black) coloring as in the case of DFS. Instead of exploring an edge incident on the most recently discovered vertex, we pick an arbitrary gray vertex and explore all of its incident edges. When we run out of gray vertices, we instead process a white vertex after incrementing a components counter. If the set of gray vertices is stored in an instance of the dictionary of Lemma 2.1 with its *some_id* operation, the process can easily be carried out in linear time.

For the BFS problem, we refine the process by splitting the set of gray vertices in two, the sets of *inner-gray* and of *outer-gray* vertices. As long as there are inner-gray vertices, we process one of these, coloring its white neighbors outer-gray. When this is no longer the case, we increment a distance counter and recolor the outer-gray vertices inner-gray. When there are neither inner-gray nor outer-gray vertices, we set the distance counter to 0 and continue the process at the first white vertex in the sequence $(\pi(1), \dots, \pi(n))$. ◀

6 Priority Queues with a Deletion Budget and Their Applications

For our purposes, a *priority queue* is a data structure that maintains an initially empty collection of items, each with a unique *identification*, a *key* drawn from a totally ordered set, and arbitrary *satellite data*, under the operations *insert*, *extract_min* and *decrease_key*. The operation *insert* inserts a new item in the collection, *extract_min* returns an item whose key is minimal after deleting it from the collection, and a call *decrease_key*(v, d, p) replaces the current key d_v of the item with identification v by d and its current satellite data by p , provided that $d < d_v$, and does nothing if $d \geq d_v$. Our priority queue is nonstandard in two minor ways. First, satellite data are frequently not included in the specification of priority queues. And second, a call *decrease_key*(v, d, p) is usually considered legal only if d is smaller than the key of v before the call.

We will say that a priority queue has a *deletion budget* of b if it is guaranteed to work correctly until the end of the b th call of *extract_min* (but possibly not after that). Thus usual priority queues have infinite deletion budgets. The following general construction derives from a priority queue Q a priority queue Q_b with a smaller budget b : Initially, Q_b operates exactly as Q . Whenever the number of items stored in Q_b reaches $2b$, however, all the items are extracted from Q_b , their median is computed, b items with largest keys are thrown away, and the other b items are reinserted in Q_b . A call of *decrease_key* that refers to an item that was thrown away reinserts the item with its new key. To see the correctness of the construction, observe that if an item is thrown away and not later reinserted with a smaller key, Q can avoid returning it in one of the b first calls of *extract_min*, whereas an item whose key in Q_b is incorrect and therefore too large (the item must have been thrown away and later reinserted) is definitely not returned by Q_b in any of these calls. The main advantage of a priority queue with a small deletion budget is that it uses little space. By applying the construction above to a Fibonacci heap [25], augmented with an instance of

the dictionary of Lemma 2.1 that we use to map identifications of items to their positions in the Fibonacci heap and their keys and satellite data, we obtain:

► **Lemma 6.1.** *For every given $n, b \in \mathbb{N}$, there is a priority queue with deletion budget b for identifications drawn from $\{1, \dots, n\}$ with $O(\log n)$ -bit keys and $O(\log n)$ -bit satellite data that executes `insert` and `decrease_key` in constant amortized time and `extract_min` in $O(\log n)$ amortized time and that uses $O(n + b \log n)$ bits.*

6.1 Computing Minimum Spanning Forests

► **Theorem 6.2.** *Given an undirected graph G with n vertices, m edges and $O(\log n)$ -bit edge weights, represented with adjacency arrays and cross pointers, the edges of a minimum spanning forest of G can be output either in $O(n + m \log n)$ time with $O(n)$ bits or in $O(m + n \log n)$ time with $O(n \log(2 + m/n))$ bits.*

Proof sketch. We give a proof only for the more interesting case $m \geq n/2$. We run Prim's algorithm [30] with an instance Q_b of the priority queue of Lemma 6.1 with deletion budget $\Theta(n/\log n)$. Prim's algorithm grows a minimum spanning forest F one tree at a time, repeatedly adding to F a vertex outside of F that is closest to F . For each vertex v outside of F , Q_b stores the item (v, d_v, p_v) , where the key d_v is the smallest weight of an edge $\{u, v\}$ for which u is in F , and p_v , if $d_v < \infty$, is the position of u in the adjacency array $N(v)$ of v .

Q_b must be refilled $O(\log n)$ times. Between the refillings, the algorithm n times executes an `extract_min` operation on Q_b to obtain an item (v, d_v, p_v) and, if $d_v < \infty$, outputs the edge $\{u, v\}$, where u is the vertex in position p_v in $N(v)$. For each neighbor x of v outside of F , it also executes the operation `decrease_key`(x, d, p), where d is the weight of the edge $\{v, x\}$ and p is the position of v in $N(x)$ —which can be found by following a cross pointer. Outside of refillings of Q_b , the algorithm uses $O(m + n \log n)$ time.

Processing every edge in G , we can refill Q_b in $O(m)$ time, which shows the first part of the theorem. In order to be faster, we maintain for each vertex v outside of F the position in its adjacency array of a closest neighbor of v in F , i.e., the last component of the triple (v, d_v, p_v) . This needs $O(n \log(2 + m/n))$ bits and allows the refilling time to be lowered to $O(n)$, which shows the second part of the theorem. ◀

6.2 The Single-Source Shortest-Paths Problem

► **Theorem 6.3.** *For every function $s : \mathbb{N} \rightarrow \mathbb{N}$ such that $s(n)$ can be computed within the resource bounds of this theorem (e.g., in $O(n)$ time using $O(n)$ bits), the following problem can be solved in $O(m + n \log n + n((\log n)/s(n))^2)$ time with $O(n(\log(2 + m/n) + s(n)))$ bits: Given a directed graph $G = (V, E)$ with n vertices, m edges and nonnegative $O(\log n)$ -bit edge weights, represented with sorted in/out adjacency arrays and cross pointers, and a vertex $s^* \in V$ from which all vertices in G are reachable, compute a shortest-paths tree in G rooted at s^* , i.e., a tree that is the union, over all $v \in V$, of a shortest path in G from s^* to v .*

Proof sketch. Assume without loss of generality that $s(n) \leq \log n$. We run Dijkstra's algorithm [15, 19, 34] with an instance Q_b of the priority queue of Lemma 6.1 with deletion budget $\Theta(ns(n)/\log n)$. Dijkstra's algorithm grows an SSSP tree T rooted at s^* one vertex at a time, repeatedly adding to T a vertex outside of T that is closest to s^* . For each vertex v outside of T , Q_b stores the item (v, d_v, p_v) , where the key d_v is the infimum of the lengths of paths in G from s^* to v whose only vertex outside of T is v , and p_v , if $d_v < \infty$, is the position in the in-adjacency array $N_{\text{in}}(v)$ of v of the second-last vertex on a shortest such path.

Q_b must be refilled $O(\log n/s(n))$ times. Between the refillings, the algorithm n times executes an *extract_min* operation on Q_b to obtain an item (v, d_v, p_v) . It adds v to T and, if $v \neq s^*$, adds also the edge (u, v) , where u is the vertex in position p_v in $N_{\text{in}}(v)$. The edge (u, v) or the distance d_v from s^* to v may be output at this point; at any rate, the algorithm remembers (u, v) by storing p_v permanently with v . This allows v to find its parent u in T in constant time—an operation that we call *following a parent pointer*—and, over all vertices v , needs $O(n \log(2 + m/n))$ bits. For each outneighbor x of v outside of T , the algorithm also executes the operation *decrease_key* $(x, d_v + c, p)$, where c is the weight of the edge (v, x) and p is the position of v in $N_{\text{in}}(x)$, an operation known as *relaxing* the edge (v, x) . Outside of refillings of Q_b , the algorithm uses $O(m + n \log n)$ time.

To ease refillings, the algorithm maintains the following additional information: First, a list L of the vertices added to T since the previous refilling and their distances from s^* ($O(ns(n))$ bits). Second, for each vertex v outside of T , the integer p_v such that a triple of the form (v, d, p_v) was present in Q_b at the end of the previous refilling ($O(n \log(2 + m/n))$ bits). We call p_v the *old tentative parent pointer* of v .

In each refilling, the vertices outside of T are processed in $O(\log n/s(n))$ batches of $O(r)$ vertices each, where $r = ns(n)/\log n$. The batches must be consistent with the ordering of the adjacency arrays of G in the sense that if u and v are vertices such that v appears in a batch after that of u , v may not precede u in any adjacency array. The purpose of the processing of a batch is, for each vertex v in the batch, to insert the correct triple (v, d_v, p_v) in Q_b and to store p_v as the new old tentative parent pointer of v . We will show that a batch can be processed in $O(n)$ time plus a quantity that sums to $O(m + n \log n)$ over all refillings. Since there are $O(\log n/s(n))$ refillings and $O(\log n/s(n))$ batches to be processed in each refilling, we arrive at the overall time bound of $O(m + n \log n + n((\log n)/s(n))^2)$.

For each batch, we first recompute the items stored in Q_b for the vertices in the batch at the end of the previous refilling of Q_b . Since each vertex v in the batch already knows its old tentative parent pointer p_v , this amounts to computing the length of the path in T from s^* to the vertex u in position p_v in $N_{\text{in}}(v)$. This quantity could be found simply by following parent pointers from u to s^* , summing edge weights along the way. In the interest of efficiency, however, the vertices in the batch collaborate.

In a first phase, the vertices in the batch, one by one, emit a token that follows parent pointers, summing edge weights as it goes along, but marks the vertices that it passes and stops as soon as it reaches s^* or a vertex marked earlier by another token, after marking that vertex as a *branching vertex*. The total number of edges on the paths traversed by the tokens is bounded by $n - 1$ (T has no more edges), and the number of branching vertices is $O(r)$.

In a second phase, the vertices are processed in the same order as in the first phase, and each sends a token twice along the same path as in the first phase. The path ends either at s^* , at a distance of 0 from itself, or at a branching vertex which, at this point, will have been marked with its distance from s^* . Adding that distance to the known length of the path traversed, the vertex obtains its own distance from s^* . In the final traversal of the path by its token, the vertex helps later vertices in the batch by marking all branching vertices along the path with their distances from s^* . This is done by subtracting the edge weights encountered along the path from a variable initialized with the total length of the path.

What remains for the batch at this point is to relax all edges from vertices stored in L to vertices in the batch. Because the adjacency arrays are sorted consistently with the batches, this can be done in $O(m + n \log n)$ time over all refillings, which establishes the running time anticipated above. If the distances of branching vertices from s^* are stored in an instance of the dictionary of Lemma 2.1, the necessary additional space is $O(n + r \log n) = O(ns(n))$. ◀

References

- 1 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.
- 2 Tetsuo Asano, Kevin Buchin, Maike Buchin, Matias Korman, Wolfgang Mulzer, Günter Rote, and André Schulz. Reprint of: Memory-constrained algorithms for simple polygons. *Comput. Geom. Theory Appl.*, 47(3, Part B):469–479, 2014.
- 3 Tetsuo Asano, Taisuke Izumi, Masashi Kiyomi, Matsuo Konagaya, Hirotaka Ono, Yota Otachi, Pascal Schweitzer, Jun Tarui, and Ryuhei Uehara. Depth-first search using $O(n)$ bits. In *Proc. 25th International Symposium on Algorithms and Computation (ISAAC 2014)*, volume 8889 of *LNCS*, pages 553–564. Springer, 2014.
- 4 Tetsuo Asano, Wolfgang Mulzer, Günter Rote, and Yajun Wang. Constant-work-space algorithms for geometric problems. *J. Comput. Geom.*, 2(1):46–68, 2011.
- 5 Luis Barba, Matias Korman, Stefan Langerman, and Rodrigo I. Silveira. Computing a visibility polygon using few variables. *Comput. Geom. Theory Appl.*, 47(9):918–926, 2014.
- 6 Luis Barba, Matias Korman, Stefan Langerman, Rodrigo I. Silveira, and Kunihiko Sadakane. Space-time trade-offs for stack-based algorithms. In *Proc. 30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*, volume 20 of *LIPICs*, pages 281–292. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2013.
- 7 Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. A sublinear space, polynomial time algorithm for directed s - t connectivity. *SIAM J. Comput.*, 27(5):1273–1282, 1998.
- 8 Paul Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM J. Comput.*, 20(2):270–277, 1991.
- 9 Hervé Brönnimann, John Iacono, Jyrki Katajainen, Pat Morin, Jason Morrison, and Godfried Toussaint. Space-efficient planar convex hull algorithms. *Theor. Comput. Sci.*, 321(1):25–40, 2004.
- 10 Timothy M. Chan. Comparison-based time-space lower bounds for selection. *ACM Trans. Algorithms*, 6(2):Article 26, 2010.
- 11 Timothy M. Chan and Eric Y. Chen. Multi-pass geometric algorithms. *Discrete Comput. Geom.*, 37(1):79–102, 2007.
- 12 Timothy M. Chan, J. Ian Munro, and Venkatesh Raman. Faster, space-efficient selection algorithms in read-only memory for integers. In *Proc. 24th International Symposium on Algorithms and Computation (ISAAC 2013)*, volume 8283 of *LNCS*, pages 405–412. Springer, 2013.
- 13 Timothy M. Chan, J. Ian Munro, and Venkatesh Raman. Selection and sorting in the “restore” model. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 995–1004. SIAM, 2014.
- 14 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.
- 15 George B. Dantzig. On the shortest route through a network. *Manag. Sci.*, 6(2):187–190, 1960.
- 16 Omar Darwish and Amr Elmasry. Optimal time-space tradeoff for the 2D convex-hull problem. In *Proc. 22nd Annual European Symposium on Algorithms (ESA 2014)*, volume 8737 of *LNCS*, pages 284–295. Springer, 2014.
- 17 Bireswar Das, Samir Datta, and Prajakta Nimbhorkar. Log-space algorithms for paths and matchings in k -trees. *Theory Comput. Syst.*, 53(4):669–689, 2013.
- 18 Bireswar Das, Jacobo Torán, and Fabian Wagner. Restricted space algorithms for isomorphism on bounded treewidth graphs. *Inform. Comput.*, 217:71–83, 2012.
- 19 E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, 1959.

- 20 Jeff Edmonds, Chung Keung Poon, and Dimitris Achlioptas. Tight lower bounds for st -connectivity on the NNJAG model. *SIAM J. Comput.*, 28(6):2257–2284, 1999.
- 21 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *Proc. 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 143–152. IEEE Computer Society, 2010.
- 22 Michael Elberfeld and Ken-ichi Kawarabayashi. Embedding and canonizing graphs of bounded genus in logspace. In *Proc. 46th ACM Symposium on Theory of Computing (STOC 2014)*, pages 383–392. ACM, 2014.
- 23 Amr Elmasry, Daniel Dahl Juhl, Jyrki Katajainen, and Srinivasa Rao Satti. Selection from read-only memory with limited workspace. *Theor. Comput. Sci.*, 554:64–73, 2014.
- 24 Greg N. Frederickson. Upper bounds for time-space trade-offs in sorting and selection. *J. Comput. Syst. Sci.*, 34(1):19–26, 1987.
- 25 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, 1987.
- 26 Torben Hagerup and Frank Kammer. Dynamic data structures for the succinct RAM, 2015. In preparation.
- 27 J. I. Munro and M. S. Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12(3):315–323, 1980.
- 28 J. Ian Munro and Venkatesh Raman. Selection from read-only memory and sorting with minimum data movement. *Theor. Comput. Sci.*, 165(2):311–323, 1996.
- 29 Jakob Pagter and Theis Rauhe. Optimal time-space trade-offs for sorting. In *Proc. 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1998)*, pages 264–268. IEEE Computer Society, 1998.
- 30 R. C. Prim. Shortest connection networks and some generalizations. *Bell Syst. Tech. J.*, 36(6):1389–1401, 1957.
- 31 Venkatesh Raman and Sarnath Ramnath. Improved upper bounds for time-space trade-offs for selection. *Nord. J. Comput.*, 6(2):162–180, 1999.
- 32 Robert Tarjan. Finding dominators in directed graphs. *SIAM J. Comput.*, 3(1):62–89, 1974.
- 33 Martin Tompa. Two familiar transitive closure algorithms which admit no polynomial time, sublinear space implementations. *SIAM J. Comput.*, 11(1):130–137, 1982.
- 34 P. D. Whiting and J. A. Hillier. A method for finding the shortest route through a road network. *J. Oper. Res. Soc.*, 11(1/2):37–40, 1960.

Pattern Matching with Variables: Fast Algorithms and New Hardness Results

Henning Fernau¹, Florin Manea², Robert Mercas², and Markus L. Schmid¹

- ¹ Fachbereich IV – Abteilung Informatikwissenschaften, Universität Trier, D-54286 Trier, Germany, {Fernau, MSchmid}@uni-trier.de
- ² Kiel University, Department of Computer Science, D-24098 Kiel, Germany, {flm, rgm}@informatik.uni-kiel.de

Abstract

A pattern (i. e., a string of variables and terminals) maps to a word, if this is obtained by uniformly replacing the variables by terminal words; deciding this is \mathcal{NP} -complete. We present efficient algorithms¹ that solve this problem for restricted classes of patterns. Furthermore, we show that it is \mathcal{NP} -complete to decide, for a given number k and a word w , whether w can be factorised into k distinct factors; this shows that the injective version (i. e., different variables are replaced by different words) of the above matching problem is \mathcal{NP} -complete even for very restricted cases.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, F.4.3 Formal Languages

Keywords and phrases combinatorial pattern matching, combinatorics on words, patterns with variables, \mathcal{NP} -complete string problems

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.302

1 Introduction

In the context of this work, a *pattern* is a string that consists of *terminal symbols* (e. g., $\mathbf{a}, \mathbf{b}, \mathbf{c}$) and *variables* (e. g., x_1, x_2, x_3). The terminal symbols are treated as constants, while the variables are to be uniformly replaced by strings over the set of terminals (i. e., different occurrences of the same variable are replaced by the same string); thus, a pattern is mapped to a terminal word. For example, $x_1 \mathbf{a} b x_1 x_2 \mathbf{c} x_2 x_1$ can be mapped to $\mathbf{a} \mathbf{c} \mathbf{a} \mathbf{b} \mathbf{a} \mathbf{c} \mathbf{c} \mathbf{a} \mathbf{a} \mathbf{a} \mathbf{c}$ and $\mathbf{b} \mathbf{a} \mathbf{b} \mathbf{b} \mathbf{a} \mathbf{c} \mathbf{a} \mathbf{b}$ by the replacements $(x_1 \rightarrow \mathbf{a} \mathbf{c}, x_2 \rightarrow \mathbf{c} \mathbf{a} \mathbf{a})$ and $(x_1 \rightarrow \mathbf{b}, x_2 \rightarrow \mathbf{a})$, respectively.

Due to their simple definition, the concept of patterns (and how they map to words) emerges in various areas of theoretical computer science, such as language theory (pattern languages [2]), learning theory (inductive inference [2, 25, 27, 10], PAC-learning [20]), combinatorics on words (word equations [18, 24], unavoidable patterns [23]), pattern matching (generalised function matching [1, 26]), database theory (extended conjunctive regular path queries [4]), and we can also find them in practice in the form of extended regular expressions with backreferences [5, 14], used in programming languages like Perl, Java, Python, etc.

In all these different applications, the main purpose of patterns is to express combinatorial pattern matching questions. For instance, searching for a word w in a text t can be expressed as testing whether the pattern xwy can be mapped to t ; testing whether a word w contains

¹ The computational model we use is the standard unit-cost RAM with logarithmic word size. Also, all algorithms appearing in our time complexity evaluations are in base 2.

a k -repetition is equivalent to testing whether the pattern xy^kz can be mapped to w , etc. Not only problems of testing whether a given word contains a regularity or a motif of a certain form can be expressed by patterns, but also problems asking whether a word can be factorised in a specifically restricted manner can be modelled in this way. For instance, asking whether x^2y^3 can be mapped to w is equivalent to asking whether the word w can be factorised in two equal factors followed by three equal factors.

Unfortunately, deciding whether a given pattern can be mapped to a given word, the *matching problem*, is \mathcal{NP} -complete [2], which naturally severely limits the practical application of patterns. In fact, there are only few applications of patterns for which this problem does not play a central role and some computational tasks on patterns that have no apparent connection to the matching problem turn out to implicitly solve it anyway (e. g., this is the case for the task of computing so-called descriptive patterns for finite sets of words [2, 11]). A comprehensive multivariate analysis of the complexity of the matching problem [12, 13] demonstrates that the \mathcal{NP} -completeness also holds for strongly restricted variants of the problem. On the other hand, some subclasses of patterns are known for which the matching problem is in \mathcal{P} (this is obviously the case if the number of different variables in the patterns is bounded by a constant, but there are also more sophisticated structural parameters of patterns that can be exploited in order to solve the matching problem efficiently [28, 29]). Unfortunately, the existing polynomial time algorithms for these classes are fairly basic and they serve the mere purpose of proving containment in \mathcal{P} ; thus, they cannot be considered efficient in a practical sense. Therefore, we present here better algorithms for the known polynomial variants of the matching problem. While we consider our algorithms to be advanced and non-trivial, their running times have still an exponential dependency on certain parameters of patterns and, therefore, are acceptable only for strongly restricted classes of patterns. However, as can be concluded from the parameterised hardness results of [13], these exponential dependencies seem necessary under common complexity theoretical assumptions.

In some applications of patterns it might be necessary to require the mapping of variables to be injective (i. e., different variables are substituted by different objects), e. g., this is the case in the detection of duplications in programme code (see [3]). From a more general point of view, this injective version of the matching problem asks whether a word can be factorised in a certain way, such that some specific factors are not allowed to coincide. The special version of this problem where each two factors must be different has been investigated in [7] and is motivated by the problem of self-assembly of short DNA fragments into larger sequences, which is crucial for gene synthesis (see references in [7]). We show the \mathcal{NP} -completeness of the following natural combinatorial factorisation problem: given a number k and a word w , can w be factorised into at least k distinct factors? Besides the general insight into the hardness of computing a factorisation with distinct factors, this result also implies that even for the trivial patterns $x_1x_2 \cdots x_k$ the matching problem becomes \mathcal{NP} -complete if we require injectivity; thus, in terms of complexity, a clear borderline between the injective and the non-injective versions of the matching problem is established.

This paper is organised as follows. The next section contains basic definitions and then we give an overview of all our results in Section 3. In Section 4, we develop our algorithms for the matching problem and, in Section 5, we present the hardness result mentioned above.

2 Basic Definitions

For detailed definitions regarding combinatorics on words we refer to [22]. We denote our *alphabet* by Σ , the *empty word* by ε , and the *length* of a word w by $|w|$. For $w \in \Sigma^*$ and

each $1 \leq i \leq j \leq |w|$, let $w[i..j] = w[i] \cdots w[j]$, where $w[k]$ represents the *letter on position* k , for $1 \leq k \leq |w|$. The *catenation* of k words w_1, \dots, w_k is written $\Pi_{i=1,k} w_i$. If $w = w_i$ for all $1 \leq i \leq k$, this represents the k th *power* of w , denoted by w^k ; here, w is a *root* of w^k . We say that w is *primitive* if it cannot be expressed as a power $\ell > 1$ of any root.

For any $w \in \Sigma^+$, a *factorisation* of w is a tuple $p = (u_1, u_2, \dots, u_k) \in (\Sigma^+)^k$, $k \in \mathbb{N}$, with $w = u_1 u_2 \cdots u_k$. Every word u_i , $1 \leq i \leq k$, is called a *factor* (of p) or simply p -*factor*. Let $p = (u_1, u_2, \dots, u_k)$ be an arbitrary factorisation. We define its set of factors as $\text{sf}(p) = \{u_1, u_2, \dots, u_k\}$ and its size as $\text{s}(p) = k$. A factorisation p is *unique* if every factor is distinct, i. e., $\text{s}(p) = |\text{sf}(p)|$. For every $1 \leq i \leq \text{s}(p)$, $p(i) = u_i$ denotes the i th factor of p . As an example, we consider the factorisation $p = (\mathbf{a}, \mathbf{ba}, \mathbf{cba}, \mathbf{a}, \mathbf{ba}, \mathbf{a})$ of the word $w = \mathbf{abacbaabaa}$. We note that $\text{sf}(p) = \{\mathbf{a}, \mathbf{ba}, \mathbf{cba}\}$ and $\text{s}(p) = 6$. For the sake of readability, we sometimes represent a factorisation $(\mathbf{a}, \mathbf{ba}, \mathbf{cba}, \mathbf{a}, \mathbf{ba}, \mathbf{a})$ in the form $\mathbf{a} \mid \mathbf{ba} \mid \mathbf{cba} \mid \mathbf{a} \mid \mathbf{ba} \mid \mathbf{a}$.

Let $X = \{x_1, x_2, x_3, \dots\}$ and call every $x \in X$ a *variable*. For a finite alphabet of *terminals* $\Sigma \cap X = \emptyset$, we define $\text{PAT}_\Sigma = (X \cup \Sigma)^+$ and $\text{PAT} = \bigcup_\Sigma \text{PAT}_\Sigma$. Every $\alpha \in \text{PAT}$ is a *pattern* and every $w \in \Sigma^*$ is a (*terminal*) *word*. Given a sequence v , word or pattern, for the smallest sets $B \subseteq \Sigma$ and $Y \subseteq X$ with $v \in (B \cup Y)^*$, we denote $\text{alph}(v) = B$ and $\text{var}(v) = Y$. For any $x \in (\text{var}(\alpha) \cup \text{alph}(\alpha))$, $|\alpha|_x$ denotes the number of occurrences of x in α .

A *substitution* (for α) is a mapping $h : \text{var}(\alpha) \rightarrow \Sigma^+$. For every $x \in \text{var}(\alpha)$, we say that x is *substituted by* $h(x)$ and $h(\alpha)$ denotes the word obtained by substituting every occurrence of a variable x in α by $h(x)$ and leaving the terminals unchanged. If, for all $x, y \in \text{var}(\alpha)$, $x \neq y$ implies $h(x) \neq h(y)$, then h is *injective*. As an example, we consider the pattern $\beta = x_1 \mathbf{a} x_2 \mathbf{b} x_2 x_1 x_2$ and the words $u = \mathbf{bacbabbacbb}$, $v = \mathbf{abaabbabab}$. It can be verified that $h(\beta) = u$, where $h(x_1) = \mathbf{bacb}$, $h(x_2) = \mathbf{b}$ and $g(\beta) = v$, where $g(x_1) = g(x_2) = \mathbf{ab}$. Furthermore, h is injective, g is not and β cannot be mapped to v by an injective substitution.

The *matching problem*, denoted by MATCH, is to decide for a given pattern α and word w , whether there exists a substitution h with $h(\alpha) = w$. By inj-MATCH, we denote the variant of the matching problem where the substitution needs to be injective.² For any $P \subseteq \text{PAT}$, the *matching problem for* P is the matching problem, where the input patterns are from P .

A pattern α is *regular* if, for every $x \in \text{var}(\alpha)$, $|\alpha|_x = 1$, and the class of regular patterns is denoted by PAT_{reg} . For any $k \in \mathbb{N}$, a k -*variable* pattern is a pattern α that satisfies $|\text{var}(\alpha)| \leq k$ and a pattern β with $|\{x \in \text{var}(\beta) \mid |\beta|_x \geq 2\}| \leq k$ is a k -*repeated-variable* pattern. For every $k \in \mathbb{N}$, $\text{PAT}_{\text{var} \leq k}$ and $\text{PAT}_{\text{var} \leq k}^r$ denote the set of k -variable patterns and k -repeated-variable patterns, respectively. Let α be a pattern. For every $y \in \text{var}(\alpha)$, the *scope of* y in α is defined by $\text{sc}_\alpha(y) = \{i, i+1, \dots, j\}$, where i is the leftmost and j the rightmost occurrence of y in α . The scopes of some variables $y_1, y_2, \dots, y_k \in \text{var}(\alpha)$ *coincide* in α if $\bigcap_{1 \leq i \leq k} \text{sc}_\alpha(y_i) \neq \emptyset$. By $\text{scd}(\alpha)$, we denote the *scope coincidence degree* (scd for short) of α , which is the maximum number of variables in α such that their scopes coincide. For example, the scopes of all variables coincide in $\alpha_1 = x_1 x_2 x_1 x_2 x_3 x_1 x_2 x_3$, but the scopes of x_1 and x_3 do not coincide in $\alpha_2 = x_1 x_2 x_1 x_2 x_3 x_2 x_3 x_3$; thus, $\text{scd}(\alpha_1) = 3$ and $\text{scd}(\alpha_2) = 2$. For every $k \in \mathbb{N}$, let $\text{PAT}_{\text{scd} \leq k}$ denote the set of patterns α with $\text{scd}(\alpha) \leq k$. By definition, $\text{PAT}_{\text{scd} \leq 1}$ coincides with the class of *non-cross* patterns (see [29]), which we denote by PAT_{nc} .

The *one-variable blocks* in a pattern are contiguous blocks of occurrences of the same variable. For instance, the number of one-variable blocks in $\alpha = x_1 x_2 x_2 a x_2 x_2 x_2 x_3 a x_3 x_2 x_2 x_3 x_3$ is 7. A pattern α with m one-variable blocks can be written as $\alpha = w_0 \Pi_{i=1,m} (z_i^{k_i} w_i)$ with $z_i \in \text{var}(\alpha)$ for $i \in \{1, \dots, m\}$ and $z_i \neq z_{i+1}$, whenever $w_i = \varepsilon$ for $i \in \{1, \dots, m-1\}$.

² There exist variants of the matching problem where substitutions can also *erase* variables by mapping them to ε . In this work, we are not concerned with this variant of the problem.

3 Summary of Our Results

The classical and parametrised complexity of the matching problem for patterns has been recently investigated and is well understood (see [6, 12, 13, 26]). The most prominent subclasses of patterns for which it can be solved in polynomial time are the classes of patterns with a bounded number of (repeated) variables, of regular patterns, of non-cross patterns and of patterns with a bounded scope coincidence degree (see [2, 29, 28]). However, as mentioned in the introduction, the respective algorithms are rather poor considering their running times. For example, for patterns with a bounded number k of variables, the matching problem can be solved in $\mathcal{O}\left(\frac{n^{k-1}m}{(k-1)!}\right)$, where m and n are the lengths of the pattern and the word (see [17]). For patterns with a scd of at most k , an $\mathcal{O}(mn^{2(k+3)}(k+2)^2)$ time algorithm is given in [28], where m and n are the lengths of the pattern and the word, respectively, and the proof that the matching problem for non-cross patterns is in \mathcal{P} (see [29]) leads to an $\mathcal{O}(n^4)$ -time algorithm. Hence, we consider the following problem worth investigating.

► **Problem 1.** *Let K be a class of patterns for which the matching problem can be solved in polynomial time. Find an efficient algorithm that solves the matching problem for K .*

The main class of patterns we consider is that of patterns with bounded scope coincidence degree. Our first result in this setting concerns patterns where the scope coincidence degree is bounded by 1, or, in other words, non-cross patterns. In that case we show that we can decide whether a pattern α having m one-variable blocks matches a word w of length n in $\mathcal{O}(nm \log n)$ time; this is an important improvement over the previously available $\mathcal{O}(n^4)$ algorithm. Our algorithm is based on a general dynamic programming approach, and it tries to find, for certain prefixes of the pattern, the prefixes of the word that match them. While the general approach is rather simple, the details of the efficient implementation of this approach require a detailed combinatorial analysis of the possible matches. For instance, as a byproduct of our approach to the matching problem for PAT_{nc} , we obtain a stringology result that extends in a non-trivial manner a major result from [8], showing how the primitively rooted squares contained in a word of length n can be listed optimally in $\mathcal{O}(n \log n)$. Our result shows that given a word w of length n and a word v shorter than n , then w contains $\mathcal{O}(n \log n)$ factors of the form uvu with uv primitive, and all these factors can be found in $\mathcal{O}(n \log n)$ time. Again, this result is optimal, as it can be seen just by looking at the original case of primitively rooted squares, or factors of the form uvu with uv primitive and $v = \varepsilon$.

When considering general patterns with bounded scope coincidence degree, we show, using a similar dynamic programming approach, that the matching problem for $\text{PAT}_{\text{scd} \leq k}$ is solvable in $\mathcal{O}\left(\frac{n^{2k}m}{((k-1)!)^2}\right)$ time, where n is the length of the input word and m is, again, the number of one-variable blocks occurring in the pattern. One should note that in this case we were not able to use all the combinatorial insights shown for non-cross patterns (thus, the $\log n$ factor is replaced by an n factor in the evaluation of the time complexity), but, still, our algorithm is significantly faster than the previously known solution.

Another class of patterns we consider is $\text{PAT}_{\text{var} \leq k}^r$ of patterns with at most k repeated variables. For the basic case $k = 1$ we obtain that the matching problem is solved in $\mathcal{O}(n^2)$ time. Our algorithm is based on a non-trivial processing of the suffix array of the input word. Further, we use this result to show that the matching problem for the general class of patterns $\text{PAT}_{\text{var} \leq k}^r$ is solvable in $\mathcal{O}\left(\frac{n^{2k}}{((k-1)!)^2}\right)$ time, where n is the length of the input word. Note that our algorithm is better than the one that could have been obtained by using the fact that patterns with at most k repeated variables have the scd bounded by $k + 1$, and then direct applying our previous algorithm solving the matching problem for $\text{PAT}_{\text{scd} \leq k+1}$.

The classes of non-cross patterns and of patterns with a bounded scd or with a bounded number of repeated variables are of special interest, since for them we can compute so-called descriptive patterns (see [2, 29]) in polynomial time. A pattern α is *descriptive* (with respect to, say, non-cross patterns) for a finite set S of words if it can generate all words in S and there exists no other non-cross pattern that describes the elements of S in a better way. Computing a descriptive pattern, which is \mathcal{NP} -complete in general, means to infer a pattern common to a finite set of words, with applications for inductive inference of pattern languages (see [25]). For example, our algorithm for computing non-cross patterns can be used in order to obtain an algorithm that computes a descriptive non-cross pattern in time $\mathcal{O}(\sum_{w \in S} (m^2 |w| \log |w|))$, where m is the length of a shortest word of S (see [11] for details).

Our algorithms, except the ones for the basic cases of non-cross patterns and patterns with only one repeated variable, still have an exponential dependency on the number of repeated variables or the scd. Therefore, only for very low constant bounds on these parameters can these algorithms be considered efficient. Naturally, finding a polynomial time algorithm for which the degree of the polynomial does not depend on the number of repeated variables would be desirable. However, such an algorithm would also be a fixed parameter algorithm for the matching problem parameterised by the number of repeated variables and in [13] it has been shown that this parameterised problem is $W[1]$ -hard. This means that the existence of such an algorithm is very unlikely. Furthermore, since the number of repeated variables gives also an upper bound for the scd, the mentioned $W[1]$ -hardness result carries over to the case where the scd is a parameter and therefore it is just as unlikely to find an algorithm that is not exponential in the scd. This observation justifies the exponential dependency of our algorithms on the number of repeated variables and the scd.

As mentioned in the introduction, in certain settings it makes sense to require the mapping of variables to words to be injective. The current state of knowledge regarding the complexity of the matching problem suggests that this difference has no substantial impact; although, in [12] it is shown that MATCH is still \mathcal{NP} -complete if the alphabet size and the length of the words the variables are mapped to are bounded, whereas it is in \mathcal{P} if we additionally require injectivity. In contrast to this, we prove the following result, which gives strong evidence that inj-MATCH is generally much harder than the non-injective version.

► **Theorem 1.** *The following problem is \mathcal{NP} -complete: given a word w and a number k , is it possible to factorise w into at least k distinct factors?*

Consequently, the injective matching problem is \mathcal{NP} -complete even for the trivial patterns $x_1 x_2 \cdots x_k$, which means that, under the assumption $\mathcal{P} \neq \mathcal{NP}$, for *all* the above mentioned classes of patterns no polynomial time algorithms for the injective matching problem exist. In addition to this negative result for the matching problem, we also gain an important insight regarding the more general problem of factorising a string into distinct factors, which, as mentioned in the introduction, is motivated by computational biology. In [7], it is shown that it is \mathcal{NP} -complete to factorise a string into distinct factors with a bound on the length of the factors and, in this regard, our result shows that the \mathcal{NP} -completeness is preserved if the length bound is dropped and instead we have a lower bound on the number of factors.

4 Algorithmic Results

In this section we propose a series of algorithms for the matching problem for several classes of patterns. We begin by looking at classes where the number of repeated variables is bounded: we consider the basic classes where no variable or, more interestingly, only one variable is

repeated, and then investigate the class of patterns in which $k \geq 2$ variables are repeated. Then, we look at the more involved case of patterns with bounded scope coincidence degree. The basic case is, in this setting, PAT_{nc} , where the scope coincidence degree is upper bounded by 1; after presenting an algorithm solving the matching problem for nc-patterns, we analyse the general case when the upper bound of the scope coincidence degree is $k \geq 2$.

We assume for every input word w of length n that $\text{alph}(w) \subseteq \{1, \dots, n\}$ (i.e., the symbols are integers). This is a common assumption in algorithmics on words (see the discussion in [19]). Clearly, our reasoning holds canonically for constant alphabets, as well.

For a length n word w we can build in $\mathcal{O}(n)$ time the suffix tree and suffix array structures, as well as data structures allowing us to retrieve in $\mathcal{O}(1)$ time the length of the longest common prefix of any suffixes $w[i..n]$ and $w[j..n]$ of w , denoted $LCP_w(i, j)$ (the subscript w is omitted when there is no danger of confusion). These are *LCP* data structures (see, e.g., [19, 16]). Symmetrically we can build structures allowing us to retrieve in $\mathcal{O}(1)$ time the length of the longest common suffix of any two prefixes $w[1..i]$ and $w[1..j]$ of w , denoted $LCS(i, j)$.

The first case we approach is that of regular patterns. It was already known from [29] that the matching problem for such patterns can be solved in linear time, when the alphabet of the input word w is constant. However, it is not hard to see that the same time bound can be achieved in our setting (when the input words are over integer alphabets): the matching problem for PAT_{reg} is solvable in $\mathcal{O}(|w| + |\alpha|)$ time, where w is the input word and α the input pattern. More interesting is the case of patterns that contain one repeated variable.

► **Lemma 2.** *The matching problem for $\text{PAT}_{\text{var} \leq 1}^r$ is solvable in $\mathcal{O}(|w|(|w| + |\alpha|))$ time, where w is the input word and α is the input pattern.*

The main idea behind the algorithm used to obtain this result is to find an assignment for the repeating variable from the input pattern α , say x , such that all constant factors are well placed within the word, and then fill up (using a linear pattern matching algorithm to correctly align the terminal factors of the pattern inside the word) the remaining spaces with the help of the rest of the variables from the pattern, since they occur only once.

Finding the factors of the word which are images of the parts of the pattern that do not contain the repeated variable can be done in a quadratic time preprocessing.

To find a suitable assignment for x we first choose the length of its image $\ell \leq n$ and, in linear time, partition the suffix array of the word in several *clusters* (i.e., blocks of consecutive positions that are not extendable to the left or right) such that the suffixes contained in one cluster share a common prefix of length ℓ , that will correspond to the image of x . Note that if there exists a mapping of the pattern to the word, with the image of x of length ℓ , then it maps all factors starting with x to prefixes of elements in the same cluster. Essentially, to check if such a mapping exists we use a greedy processing of each cluster. Fixing the cluster, we also fix the image of x , denoted w_x in the following, and the suffixes in that cluster provide all the occurrences of w_x in w . Now, we can assume without loss of generality that α starts and ends with variables different from x . If this would not be the case, we just isolate the lengthwise maximal prefix α_p and suffix α_s of α that contain only occurrences of x and terminals; the images of these two factors of α are now known, as we know the image of x . So we check if the image of α_p is a prefix and the image of α_s is a suffix of w . If yes, we just have to check whether the rest of the pattern (α without α_p and α_s) maps to the rest of the word (w without the images of α_p and α_s); this puts us in the aforementioned case. Further, we sort the elements in the cluster in the order of their occurrence in the word. Then the i^{th} occurrence of x in α is mapped to the leftmost occurrence of w_x surrounded by the same terminal words as the variable x in α , such that the factor of α occurring between the i^{th} and $i - 1^{\text{th}}$ occurrence of x matches the factor of w found between the respective

images of these variables. As the variables x are considered from left to right and mapped, respectively, to occurrences of w_x that appear ordered in the cluster, we can implement the above strategy in $\mathcal{O}(k)$ time, where k represents the size of the cluster; the test whether we can correctly match the factors of α that do not contain x to factors of w is done using the information gathered during the preprocessing. We process in this way all clusters having at least as many elements as the number of repeated variables in the pattern. The total number of elements in these clusters is at most $n - \ell$ so this procedure takes $\mathcal{O}(n)$ time.

The time spent for each possible value for ℓ is $\mathcal{O}(n)$. Hence, in total our algorithm needs $\mathcal{O}(n^2)$ time to decide whether there exists an assignment that maps the pattern to the word.

To solve the matching problem for patterns with at most k repeated variables, we choose the images (starting and ending positions in w) of $k - 1$ of the k repeated variables, and then get a pattern with only one repeated variable. Further, we apply Lemma 2 on this pattern.

► **Theorem 3.** *The matching problem for $\text{PAT}_{\text{var} \leq k}^r$ is solvable in $\mathcal{O}\left(\frac{|w|^{2k}}{(k-1)!^2}\right)$ time, where w is the input word.*

We now consider the more involved case of patterns with a bounded scope coincidence degree. The following combinatorial results are well known (see, e. g., [8]).

► **Lemma 4** ([8]). *Let u_1, u_2 , and u_3 be primitive words, such that $|u_1| < |u_2| < |u_3|$ and u_i^2 are prefixes (suffixes) of a word w , for all $1 \leq i \leq 3$. Then $2|u_1| < |u_3|$. As a consequence, we have $|\{u|u \text{ primitive}, u^2 \text{ prefix (respectively, suffix) of } w\}| \leq 2 \log |w|$.*

Assume that $w \in \Sigma^*$ is of length n . For each $i \leq n$ we define the set

$$P_i = \{u \mid u \text{ is a primitive word such that } u^2 \text{ is a suffix of } w[1..i]\}.$$

Lemma 4 shows that $|P_i| \leq 2 \log n$ for all $1 \leq i \leq n$. Generally, we can represent the elements of P_i in various efficient manners (e. g., for each $u \in P_i$ it is enough to store its length).

► **Lemma 5** ([8]). *Let $w \in \Sigma^*$ be a word of length n . We can compute in $\mathcal{O}(n \log n)$ time all the sets P_i associated to w , with $i \in \{1, 2, \dots, n\}$.*

Note that in [8] there are examples of words of length n for which $\sum_{i \leq n} |P_i| \in \Theta(n \log n)$.

Next, we extend the results of Lemmas 4 and 5. Instead of primitively rooted squares, we consider words of the form uvu for some fixed word v , with uv primitive (or, equivalently, with vu primitive). It is not hard to show the following lemma.

► **Lemma 6.** *For a fixed v , let $u_1vu_1, u_2vu_2, u_3vu_3$ be prefixes (suffixes) of a word w such that $|u_1| < |u_2| < |u_3|$ and $u_i v$ are primitive for all $1 \leq i \leq 3$. Then $\frac{3|u_1|}{2} < |u_3|$. As a consequence, we have $|\{uvu|uv \text{ primitive}, uvu \text{ prefix (respectively, suffix) of } w\}| \in \mathcal{O}(\log |w|)$.*

Consider two words $w, v \in \Sigma^*$, with $|w| = n$. Following the case of the primitively rooted squares, for each $i \leq n$ we define the set

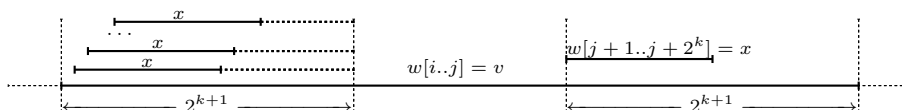
$$R_i^v = \{u \mid uvu \text{ is a suffix of } w[1..i] \text{ with } uv \text{ primitive}\}.$$

Again, R_i^v can be stored efficiently by the lengths of the words it contains. Clearly, $\sum_{i \leq n} |R_i^v| \in \mathcal{O}(n \log n)$. Moreover, as uvu with uv primitive is just a primitively rooted square when $v = \varepsilon$, it follows that for certain values of v , we have that $\sum_{i \leq n} |R_i^v| \in \Theta(n \log n)$.

The following result extends in a non-trivial manner the result of Lemma 5.

► **Lemma 7.** *Given two words $w, v \in \Sigma^*$, with $|w| = n$, we can compute in $\mathcal{O}(n \log n)$ time all the sets R_i^v associated to w , for $i \in \{1, 2, \dots, n\}$.*

We begin the high level description of the proof of this lemma with several preliminary facts. Given the word w , its dictionary of basic factors [9] is a data structure that associates



■ **Figure 1** Occurrences of x in $w[i - 2^{k+1}..i - 1]$: positions where u may start (Lemma 7).

labels to the factors of the form $w[i..i + 2^k - 1]$ (called basic factors), for $k \geq 0$ and $1 \leq i \leq n - 2^k + 1$, such that every two identical factors of the whole word get the same label and we can retrieve the label of such a factor in $\mathcal{O}(1)$ time. The dictionary of basic factors of a word of length n is constructed in $\mathcal{O}(n \log n)$ time. Looking deeper into the combinatorial structure of w we note that a basic factor $w[i..i + 2^k - 1]$ occurs either at most twice in a factor $w[j..j + 2^{k+1} - 1]$ or the positions where $w[i..i + 2^k - 1]$ occurs in $w[j..j + 2^{k+1} - 1]$ form an arithmetic progression of ratio $per(w[i..i + 2^k - 1])$ (see [21]). Hence, the occurrences of $w[i..i + 2^k - 1]$ in $w[j..j + 2^{k+1} - 1]$ can be presented in a compact manner: either at most two positions, or the starting position of the progression and its ratio. Using this property and the dictionary of basic factors one can produce in $\mathcal{O}(n \log n)$ a data structure that allows us to test the primitivity of each factor of w in $\mathcal{O}(1)$ time. Moreover, for a certain v , we can also produce in $\mathcal{O}(n \log n)$ time a data structure answering the following type of queries in $\mathcal{O}(1)$ time: “Given i and k return the compact representation of the occurrences of $w[i..i + 2^k - 1]$ in $w[i - |v| - 2^{k+1}..i - |v| - 1]$ ”.

Returning to the proof of the lemma, after the above preprocessing we find all the occurrences of v in w . Consider now one of these occurrences $w[i..j] = v$. We are searching all the prefixes u of $w[j + 1..n]$ that are also suffixes of $w[1..i - 1]$ with uv primitive. Checking naively each prefix of $w[j + 1..n]$ for these properties takes too long. Thus, we analyse simultaneously all the prefixes of $w[j + 1..n]$ that have the length between 2^k and 2^{k+1} , for $0 \leq k \leq \log n$. All these factors share as common prefix the basic factors $x = w[j + 1..j + 2^k]$. Using the data structures we constructed, we retrieve in $\mathcal{O}(1)$ time a compact representation of the occurrences of x in $w[i - 2^{k+1}..i - 1]$. We know that every possible candidate for the factor u we search for starts with one of these occurrences. Using a series of involved combinatorics on words insights, one can identify all the possible factors u that start on such a position and fulfil also the requirement that vu is primitive, in time proportional to their number. As in w there are at most $\mathcal{O}(n \log n)$ factors uvu fulfilling our requirements with each uniquely identified by the corresponding occurrence of v , and, moreover, the time we spend in analysing each occurrence of v is proportional to $\log n$ plus the number of valid factors uvu centred around that occurrence of v , the result of Lemma 7 follows.

We are now ready to solve the matching problem for non-cross patterns.

► **Theorem 8.** *The matching problem for PAT_{nc} is solvable in $\mathcal{O}(|w|m \log |w|)$ time, where w is the input word and m is the number of one-variable blocks occurring in the pattern.*

Let $\alpha \in \text{PAT}_{\text{nc}}$ be our pattern and $n = |w|$. If $\text{var}(\alpha) = \{x_1, x_2, \dots, x_\ell\}$, it is immediate that $\alpha = w_0 \prod_{k=1, \ell} (\alpha_k w_k)$, where for all $k \leq \ell$ we have $\text{var}(\alpha_k) = \{x_k\}$, α_k starts and ends with x_k , and w_k is a factor containing only terminals. We use a dynamic programming approach to test whether α matches w . More precisely, for each $i \leq \ell$ we identify all the prefixes $w[1..j]$ of w such that $w_0 \prod_{k=1, i-1} (\alpha_k w_k) \alpha_{i-1}$ matches $w[1..j]$. We briefly describe this approach.

First, assume that we know already all the positions j such that $w_0 \prod_{k=1, i-2} (\alpha_k w_k) \alpha_{i-1}$ matches $w[1..j]$. Clearly, in $\mathcal{O}(n)$ time we can find all the positions j where $w_0 \prod_{k=1, i-1} (\alpha_k w_k)$ matches $w[1..j]$: we just check whether $w[1..j]$ ends with w_{i-1} and, if so, whether the factor $w_0 \prod_{k=1, i-2} (\alpha_k w_k) \alpha_{i-1}$ matches $w[1..j - |w_{i-1}|]$. Then, we show how we can find in

$\mathcal{O}(np_i \log n)$ time the positions j such that $w_0 \prod_{k=1, i-1} (\alpha_k w_k) \alpha_i$ matches $w[1..j]$, where p_i is the number of one-variable blocks of α_i . To do this, we have to analyse the structure of α_i .

The simplest case is when $\alpha_i = x_i$. Then, $w_0 \prod_{k=1, i-1} (\alpha_k w_k) \alpha_i$ matches $w[1..j]$ if and only if there exists $j' < j$ such that $w_0 \prod_{k=1, i-1} (\alpha_k w_k)$ matches $w[1..j']$; finding all such positions j takes $\mathcal{O}(n)$ time, so our claim holds in this case.

Consider next the case when $\alpha_i = x_i^k$ with $k \geq 2$. In a first phase, for each position j and each primitively rooted square suffix t^2 of $w[1..j]$ we check whether t^k is a suffix of $w[1..j]$ and if $w_0 \prod_{k=1, i-1} (\alpha_k w_k)$ matches $w[1..j - k|t|]$; if both these checks are true, we conclude that $w_0 \prod_{k=1, i-1} (\alpha_k w_k) \alpha_i$ matches $w[1..j]$ when x_i is mapped to t , and we store this information. In this way, we found all the positions j such that $w_0 \prod_{k=1, i-1} (\alpha_k w_k) \alpha_i$ matches $w[1..j]$ when x_i is mapped to a primitive word; we just have to analyse the case when x_i is mapped to a non-primitive word. Now, for each position j (considered in increasing order) and each primitively rooted square suffix t^2 of $w[1..j]$, we check whether t^k is a suffix of $w[1..j]$ and, differently from the previous case, if $w_0 \prod_{k=1, i-1} (\alpha_k w_k) \alpha_i$ matches $w[1..j - k|t|]$ such that x_i is mapped to a power of t ; the dynamic programming approach ensures us that when j is considered we know whether $w_0 \prod_{k=1, i-1} (\alpha_k w_k) \alpha_i$ matches $w[1..j']$ such that x_i is mapped to a power of t' for all $j' < j$ and every t'^2 primitively rooted square suffix of $w[1..j']$. If both checks above return true, then we conclude that $w_0 \prod_{k=1, i-1} (\alpha_k w_k) \alpha_i$ matches $w[1..j]$ and x_i is mapped to a power of t ; if $w_0 \prod_{k=1, i-1} (\alpha_k w_k) \alpha_i$ matches $w[1..j - k|t|]$ with the image x_i being t^h , now we conclude that $w_0 \prod_{k=1, i-1} (\alpha_k w_k) \alpha_i$ matches $w[1..j]$ with x_i mapped to t^{h+1} . Clearly, this two-steps procedure returns all j 's such that $w_0 \prod_{k=1, i-1} (\alpha_k w_k) \alpha_i$ matches $w[1..j]$. The total time needed is $\mathcal{O}(n \log n)$, so our claim holds in this case, as well.

Finally, we consider the more complicated case of α_i containing at least one terminal. We let $\alpha_i = x_i^{\ell_0} \prod_{k=1, p_i} (w_{k,i} x_i^{\ell_{k,i}})$ be the decomposition of α_i in one-variable blocks. First, we assume that $\ell_{p_i, i} = 1$, so α_i ends with $x_i w_{p_i, i} x_i$; it may be the case that x_i is mapped to a word u such that $w_{p_i, i} u$ is primitive. Then, for each position j , we consider all the suffixes $uw_{p_i, i} u$ of $w[1..j]$ such that $w_{p_i, i} u$ is primitive (these factors are in $R_j^{w_{p_i, i}}$ and all of them can be identified in $\mathcal{O}(n \log n)$ according to Lemma 7). For each such suffix, we determine the factor u , the image of x_i . Next, in $\mathcal{O}(p_i)$ time we check whether the image γ_i of α_i under the substitution of x_i with u is a suffix of $w[1..j]$. If so, we then check whether $w_0 \prod_{k=1, i-1} (\alpha_k w_k)$ matches $w[1..j - |\gamma_i|]$, and, if our check is again true, conclude that $w_0 \prod_{k=1, i-1} (\alpha_k w_k) \alpha_i$ matches $w[1..j]$ when x_i is mapped to the u determined above. Further, we look at the case when x_i is mapped to a word u such that $w_{p_i, i} u$ is a repetition. For a position j , we consider each primitively rooted square suffix t^2 of $w[1..j]$. We can determine in constant time the exponent r_0 and the prefix t_0 of t such that $w_{p_i, i} = t^{r_0} t_0$; this means that $u = t_1 t^{r_1}$, where $t_0 t_1 = t$ but r_1 is not known. Just like before, we can check easily the cases when $r_1 \in \{0, 1\}$ (so, the value u to which x_i is mapped becomes fixed) in time $\mathcal{O}(p_i)$. In the following, let us assume that $r_1 \geq 2$ and, for simplicity, take $t_0 \neq \varepsilon$; thus, $t_1 \neq t$ and, as t_0 is known, so is t_1 . If $\ell_{p_i-1, i} \geq 2$, then the image u of x_i is uniquely determined: we just note that the word $uw_{p_i, i} u$ is $|t|$ -periodic, and cannot be extended with $|t|$ letters to the left without breaking the period, so we uniquely determine u by identifying the longest $|t|$ -periodic suffix w' of $w[1..j]$ and noting that $uw_{p_i, i} u$ is its longer suffix of the form $t_1 \{t\}^*$. Then, we can check again in $\mathcal{O}(p_i)$ whether α_i matches the suffix of $w[1..j]$ and continue just as we did before. Therefore, let us assume $\ell_{p_i-1, i} = 1$; if $w_{p_i-1, i} \notin \{t\}^* t_0$, then again we can determine the image of x_i by the same reasons, and we can continue similarly. This process continues in this manner, and we either get that the image of x_i is uniquely determined, or that $\alpha_i = x_i \prod_{k=1, p_i} (t^{s_k} t_0 x_i)$, so the image of α_i is $|t|$ -periodic. Fortunately, the latter case can be solved in the same manner as the case when α was just a repetition: we already know the positions j such

that $w_0 \prod_{k=1, i-1} (\alpha_k w_k) \alpha_i$ matches $w[1..j]$ when $w_{p_i, i} x_i$ is mapped to a power of t with lower exponent, so we just have to extend with powers of t of exponent equal to the number of occurrences of x_i in α_i , as we did before. The case when $t_0 = \varepsilon$ is treated similarly: either the image of x_i can be uniquely determined, or the image of both x_i and all the factors $w_{k, i}$ are powers of t and we can apply our previous dynamic programming approach. In the same manner, our last case, when $\ell_{p_i, i} \geq 2$ implies that either x_i is mapped to a primitive word t such that t^2 is a suffix of $w[1..j]$, or it is mapped to a power of such a word t . In both cases, an analysis similar to the above leads to the correct computation of all the positions j such that $w_0 \prod_{k=1, i-1} (\alpha_k w_k) \alpha_i$ matches $w[1..j]$. The total time needed for such an analysis is $\mathcal{O}(np_i \log n)$, as for each position j and each $t \in P_j$ we need to do $\mathcal{O}(p_i)$ steps, checking for each possible image of x_i that α_i is mapped correctly to a suffix $w[j' + 1..j]$ of $w[1..j]$, where $w_0 \prod_{k=1, i-1} (\alpha_k w_k)$ matched $w[1..j']$. Again, our claim holds.

It only remains to see that α matches to w if there exists a position j such that $w_0 \prod_{k=1, \ell-1} (\alpha_k w_k) \alpha_\ell$ matches $w[1..j]$ and $w[j + 1..n] = w_\ell$. The total time is, clearly, $\mathcal{O}\left(n \log n \left(\sum_{i=1, \ell} p_i\right)\right)$; summing up, the time complexity of our algorithm is $\mathcal{O}(nm \log n)$.

We now move on to the general case of patterns with bounded scope coincidence degree. The matching problem for $\text{PAT}_{\text{scd} \leq k}$ can be still solved by a dynamic programming approach.

► **Theorem 9.** *The matching problem for $\text{PAT}_{\text{scd} \leq k}$ is solvable in $\mathcal{O}\left(\frac{|w|^{2k} m}{((k-1)!)^2}\right)$ time, where w is the input word and m is the number of one-variable blocks occurring in the pattern.*

5 The Hardness of Factorising a Word into Distinct Factors

So far, we presented a series of upper bounds for the time needed to solve various matching problems. In this section, we prove the \mathcal{NP} -completeness of the following problem.

UNFACT

Instance: A word w and an integer $k \geq 1$.

Question: Does there exist a unique factorisation of w with size at least k ?

As shall be explained later on, this has implications on the injective version of the matching problem, which can be solved in $\mathcal{O}\left(\frac{n^{k-1} m}{(k-1)!}\right)$ (k and m are the numbers of variables and one-variable blocks, respectively), just as the general matching problem.

For the completeness result, we use the following as the base problem for our reduction.

3D-MATCH

Instance: An integer $\ell \in \mathbb{N}$ and a set $S \subseteq \{(p, q, r) \mid 1 \leq p < \ell + 1 \leq q < 2\ell + 1 \leq r \leq 3\ell\}$.

Question: Does there exist a subset S' of S with cardinality ℓ such that, for each two elements $(p, q, r), (p', q', r') \in S'$, $p \neq p', q \neq q'$ and $r \neq r'$?

An instance of 3D-MATCH is a set S of triples, the 3 components of which carry values from $\{1, 2, \dots, \ell\}$, $\{\ell + 1, \ell + 2, \dots, 2\ell\}$ and $\{2\ell + 1, 2\ell + 2, \dots, 3\ell\}$, respectively. A *solution* for (S, ℓ) is a selection of ℓ triples such that no two of them coincide in any component. Hence, for every $i \in \{1, 2, 3\}$, if we collect all the i^{th} components of the ℓ triples of a solution, then we get exactly the set $\{(i - 1)\ell + 1, (i - 1)\ell + 2, \dots, i\ell\}$. For the \mathcal{NP} -completeness of 3D-MATCH see [15].

We define a mapping g from 3D-MATCH to UNFACT. Let (S, ℓ) be an instance of 3D-MATCH, where $\ell \in \mathbb{N}$, $S = \{s_1, s_2, \dots, s_k\}$ with $s_i = (p_i, q_i, r_i)$, $1 \leq i \leq k$. Next, we construct a word w over the alphabet $\Sigma = \{\mathbf{a}, \mathbf{c}_i, \mathbf{\$}_i, \mathbf{b}_{i,j}, \mathbf{\%}_{i,j}, \mathbf{l}, \mathbf{\#}_i, \mathbf{\#}_0 \mid 1 \leq i \leq k, 1 \leq j \leq 4, 1 \leq l \leq 3\ell\}$. Let $v = v_1 v_2 \dots v_k$, where, for every $1 \leq i \leq k$, $v_i = \mathbf{c}_i p_i \mathbf{a} \mathbf{b}_{i,1} \mathbf{b}_{i,2} q_i \mathbf{a} \mathbf{b}_{i,3} \mathbf{b}_{i,4} r_i \mathbf{a} \mathbf{\$}_i$. Furthermore, we define $\hat{u} = 1 \mathbf{\#}_1 \dots \mathbf{\#}_{3\ell-2} (3\ell - 1) \mathbf{\#}_{3\ell-1} (3\ell) \mathbf{\#}_{3\ell}$ and $\bar{u} = \bar{u}_1 \bar{u}_2 \dots \bar{u}_k$, where,

for every $1 \leq i \leq k$, $\bar{u}_i = \mathbf{b}_{i,1} \%_{i,1} \mathbf{b}_{i,2} \%_{i,2} \mathbf{b}_{i,3} \%_{i,3} \mathbf{b}_{i,4} \%_{i,4}$. Finally, $u = \mathbf{a} \#_0 \widehat{u} \bar{u}$, $w = uv$, $\widehat{\ell} = 7\ell + 6(k - \ell) + |u|$ and $g(S, \ell) = (w, \widehat{\ell})$. This concludes the definition of the mapping g . In the following, let (S, ℓ) be a fixed instance of 3D-MATCH and $(w, \widehat{\ell}) = g(S, \ell)$.

We now explain the mapping g in an intuitive way. Every triple $s_i = (p_i, q_i, r_i)$ of S is represented by $v_i = \mathbf{c}_i p_i \mathbf{a} \mathbf{b}_{i,1} \mathbf{b}_{i,2} q_i \mathbf{a} \mathbf{b}_{i,3} \mathbf{b}_{i,4} r_i \mathbf{a} \mathbf{\$}_i$, where the factors $p_i \mathbf{a}$, $q_i \mathbf{a}$ and $r_i \mathbf{a}$ represent the single components. Each of the remaining symbols \mathbf{c}_i , $\mathbf{\$}_i$, $\mathbf{b}_{i,j}$, $1 \leq j \leq 4$, has exactly one occurrence in w ; thus, every factor that contains one of these will necessarily be distinct. Hence, the factors $p_i \mathbf{a}$, $q_i \mathbf{a}$ and $r_i \mathbf{a}$ are the only ones that may coincide in v_i and some v_j , $i \neq j$, and this is only the case if the triples s_i and s_j contain common elements.

We now define two special factorisations of the factors v_i , $1 \leq i \leq k$. The factorisation $\mathbf{c}_i p_i \mid \mathbf{a} \mathbf{b}_{i,1} \mid \mathbf{b}_{i,2} q_i \mid \mathbf{a} \mathbf{b}_{i,3} \mid \mathbf{b}_{i,4} r_i \mid \mathbf{a} \mathbf{\$}_i$ is called *safe* and the factorisation $\mathbf{c}_i \mid p_i \mathbf{a} \mid \mathbf{b}_{i,1} \mathbf{b}_{i,2} \mid q_i \mathbf{a} \mid \mathbf{b}_{i,3} \mathbf{b}_{i,4} \mid r_i \mathbf{a} \mid \mathbf{\$}_i$ is called *unsafe*. The safe factorisation contains only distinct factors, whereas the factors $p_i \mathbf{a}$, $q_i \mathbf{a}$ and $r_i \mathbf{a}$ of the unsafe factorisation may also occur in the unsafe factorisation of some v_j ; thus, the situation that s_i and s_j have common elements translates into the situation that the unsafe factorisations of v_i and v_j have common factors.

If $\{s_{t_1}, s_{t_2}, \dots, s_{t_\ell}\}$ is a solution of (S, ℓ) , then we can factorise all v_{t_i} , $1 \leq i \leq \ell$, into the unsafe factorisation, all other v_j , $j \notin \{t_1, t_2, \dots, t_\ell\}$, into the safe factorisation and the prefix u into $|u|$ individual factors. This yields a factorisation of w with $|u| + 7\ell + 6(k - \ell) = \widehat{\ell}$ factors and its uniqueness follows from the fact that $\{s_{t_1}, s_{t_2}, \dots, s_{t_\ell}\}$ is a solution of (S, ℓ) and that the symbols from u do not occur as single factors in v .

► **Lemma 10.** *If (S, ℓ) has a solution, then there is a unique factorisation of w with size $\widehat{\ell}$.*

Proving the converse of Lemma 10 is more difficult. The idea is to first show that if there exists a unique factorisation of w of size $\widehat{\ell}$, then there also exists one with at least the same size and the following properties: (1) no factor overlaps the boundaries between u and v or between some v_i and v_{i+1} , $1 \leq i \leq k - 1$, (2) u is split into $|u|$ factors. Property (1) is easily achieved by simply splitting the factors that may overlap the critical positions; this does only increase the number of factors and the uniqueness of the factorisations is guaranteed by the fact that the new factors must contain symbols with only one occurrence in w .

► **Lemma 11.** *If w has a unique factorisation f with size $\widehat{\ell}$, then, for some $\widehat{\ell}' \geq \widehat{\ell}$, there exists a unique factorisation f' of w of size $\widehat{\ell}'$, such that no f' -factor overlaps positions $|u|$ and $|u| + 1$ or positions $|uv_1 v_2 \cdots v_i|$ and $|uv_1 v_2 \cdots v_i| + 1$, for some i , $1 \leq i \leq k - 1$.*

Property (2) requires a more careful argument. If u is not split into $|u|$ factors, then in u there exists a factor $x\pi$, where x is a single symbol and π is some non-empty factor, and $x\pi$ is also a factor of the factorisation (with Lemma 11 we can assume that $x\pi$ lies inside of u). If $|\pi| \geq 2$, then we cut off x , which results in two factors x and π . Since $|\pi| \geq 2$, the factor π must contain a symbol with only one occurrence in w , which means that it is not repeated. If x is repeated, then this can only happen in some v_i and we can now show that the factor x must have a neighbour in v_i that starts with a symbol $y \in \{\mathbf{b}_{i,1}, \mathbf{b}_{i,2}, \mathbf{b}_{i,3}, \mathbf{b}_{i,4}, \mathbf{c}_i, \mathbf{\$}_i\}$. We now simply append x to this neighbour. If $y \in \{\mathbf{c}_i, \mathbf{\$}_i\}$, then the factor is distinct since \mathbf{c}_i and $\mathbf{\$}_i$ have only one occurrence in w . If, on the other hand, $y \in \{\mathbf{b}_{i,1}, \mathbf{b}_{i,2}, \mathbf{b}_{i,3}, \mathbf{b}_{i,4}\}$, then this new factor can only be repeated in u ; but all factors in u of size at least 2 contain a symbol that does not occur in v , thus, the newly constructed factor is distinct. If $|\pi| = 1$, then the situation is easier, since we can simply cut $x\pi$ into x and π and if one of these new factors is repeated, then we can append it to its other neighbour without producing a repeated factor.

► **Lemma 12.** *If w has a unique factorisation f of size $\widehat{\ell}$, then, for some $\widehat{\ell}' \geq \widehat{\ell}$, w has a unique factorisation f' of size $\widehat{\ell}'$, such that every single symbol of u is an f' -factor.*

Finally, we show the converse of Lemma 10. If there is a unique factorisation f of w of size $\widehat{\ell}$, then we can assume that it is of the form ensured by Lemmas 11 and 12. We can further conclude that if a single symbol of some v_i is a factor of f , then it is \mathfrak{c}_i or \mathfrak{s}_i , since otherwise it would be repeated in u . In particular, this means that no v_i can be split into more than 7 factors and if a v_i is split in exactly 7 factors, then this must be the safe factorisation defined above. Now if f splits T of the v_i , $1 \leq i \leq k$, into 7 factors and the remaining $k - T$ of the v_i , $1 \leq i \leq k$, into 6 or less factors, then f' splits w into at most $|u| + 7T + 6(k - T)$ factors. Since $\widehat{\ell} = 7\ell + 6(k - \ell) + |u| \leq |u| + 7T + 6(k - T)$ must be satisfied, we can conclude $\ell \leq T$, which means that at least ℓ factors v_i are factorised into the safe factorisation. This directly implies that the corresponding ℓ triples from S constitute a solution for (S, ℓ) .

Since the reduction is clearly polynomial, the main result, i. e., Theorem 14, follows.

► **Lemma 13.** *If there exists a unique factorisation of w of size $\widehat{\ell}$, then (S, ℓ) has a solution.*

► **Theorem 14.** *UNFACT is \mathcal{NP} -complete.*

We note that if a word has a unique factorisation of size k , then it also has a unique factorisation of size k' for all $1 \leq k' \leq k$. This is due to the fact that the uniqueness of a factorisation is preserved if we join a longest factor with one of its neighbours. In particular, this means that (w, k) is a positive UNFACT instance if and only if $x_1x_2 \cdots x_k$ matches w in an injective way; thus, we can conclude that inj-MATCH is \mathcal{NP} -complete for many classes of patterns for which its non-injective version can be easily solved in polynomial time.

► **Corollary 15.** *inj-MATCH is \mathcal{NP} -complete for PAT_{reg} , PAT_{nc} , $\text{PAT}_{\text{var} \leq k}^r$, $\text{PAT}_{\text{scd} \leq k}$, $k \geq 1$.*

We wish to point out that our proof of Theorem 14 requires an unbounded alphabet and it is open whether UNFACT is \mathcal{NP} -complete for fixed alphabets.³ Consequently, it does not imply that the injective matching problem for the classes of patterns mentioned in Corollary 15 is still \mathcal{NP} -complete if the alphabet is fixed. However, for the injective matching problem with a fixed alphabet, we can show a similar, but slightly weaker, result:

► **Theorem 16.** *inj-MATCH is \mathcal{NP} -complete for PAT_{nc} and $\text{PAT}_{\text{scd} \leq k}$, $k \geq 1$, if the alphabet is constant.*

Theorem 16 can also be proved by a reduction from 3D-MATCH. We shall give a definition of this reduction, but omit the proof of its correctness, and leave it to the reader.

Let (S, ℓ) be an instance of 3D-MATCH, where $\ell \in \mathbb{N}$, $S = \{s_1, s_2, \dots, s_k\}$ with $s_i = (p_i, q_i, r_i)$, $1 \leq i \leq k$. We define a word w over the alphabet $\Sigma = \{\mathfrak{a}, \mathfrak{b}, \mathfrak{s}, \mathfrak{c}, \#\}$ and a pattern α which uses the variables $x_{i,j}$, $1 \leq i \leq \ell$, $1 \leq j \leq 3$, and y_i, z_j , $1 \leq i \leq \ell + 1$, $1 \leq j \leq 2\ell + 2$. We first define the factors $\beta_i = x_{i,1}^2 x_{i,2}^2 x_{i,3}^2$, $1 \leq i \leq \ell$, $u_i = (\mathfrak{a}^{p_i} \mathfrak{b})^2 (\mathfrak{a}^{q_i} \mathfrak{b})^2 (\mathfrak{a}^{r_i} \mathfrak{b})^2$, $1 \leq i \leq k$, and $\#_i = (\#\mathfrak{c}^1 \#\mathfrak{c}^2 \# \cdots \#\mathfrak{c}^i \#)^m$, $1 \leq i \leq 2k + 2$, where $m = \max\{2k + 2, 3\ell\} + 1$. Then, in order to form α and w , these factors are combined as follows: $\alpha = z_1^m y_1 z_2^m \beta_1 z_3^m y_2 z_4^m \beta_2 z_5^m y_3 z_6^m \beta_3 \cdots \beta_\ell z_{2\ell+1}^m y_{\ell+1} z_{2\ell+2}^m$ and $w = \#_1 \mathfrak{s}_1 \#_2 u_1 \#_3 \mathfrak{s}_3 \#_4 u_2 \#_5 \mathfrak{s}_5 \#_6 u_3 \cdots u_k \#_{2k+1} \mathfrak{s}^{2k+1} \#_{2k+2}$.

A collection $\{s_{t_1}, s_{t_2}, \dots, s_{t_\ell}\}$ of ℓ elements from S translates into a substitution h with $h(\alpha) = w$ as follows. For every $1 \leq i \leq \ell$, the factor $z_{2i}^m \beta_i z_{2i+1}^m$ is mapped to $\#_{2t_i} u_{t_i} \#_{2t_i+1}$ and the variables y_l , $1 \leq l \leq \ell + 1$, with only one occurrence, are mapped to the remaining factors in between. Furthermore, it can be shown that if $\{s_{t_1}, s_{t_2}, \dots, s_{t_\ell}\}$ is a solution for (S, ℓ) , then h is injective. Proving the other direction is more difficult and requires a lemma which states that any substitution h with $h(\alpha) = w$ necessarily maps every β_i to some u_j .

³ As shown in [7], the variant where we require the factorisation to have short factors instead of a large size is \mathcal{NP} -complete also for fixed alphabets.

References

- 1 Amihood Amir and Igor Nor. Generalized function matching. *Journal of Discrete Algorithms*, 5:514–523, 2007.
- 2 Dana Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21:46–62, 1980.
- 3 Brenda S. Baker. Parameterized pattern matching: Algorithms and applications. *Journal of Computer and System Sciences*, 52:28–42, 1996.
- 4 Pablo Barceló, Leonid Libkin, Anthony W. Lin, and Peter T. Wood. Expressive languages for path queries over graph-structured data. *ACM Transactions on Database Systems*, 37, 2012.
- 5 Cezar Câmpeanu, Kai Salomaa, and Sheng Yu. A formal study of practical regular expressions. *International Journal of Foundations of Computer Science*, 14:1007–1018, 2003.
- 6 Raphaël Clifford, Aram W. Harrow, Alexandru Popa, and Benjamin Sach. Generalised matching. In *Proceedings of the 16th International Symposium on String Processing and Information Retrieval, SPIRE*, volume 5721 of *Lecture Notes in Computer Science*, pages 295–301, 2009.
- 7 Anne Condon, Ján Maňuch, and Chris Thachuk. The complexity of string partitioning. In *Proceedings of 23th Annual Symposium on Combinatorial Pattern Matching, CPM*, volume 7354 of *Lecture Notes in Computer Science*, pages 159–172, 2012.
- 8 Maxime Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Processing Letters*, 12(5):244–250, 1981.
- 9 Maxime Crochemore and Wojciech Rytter. Usefulness of the Karp-Miller-Rosenberg algorithm in parallel computations on strings and arrays. *Theoretical Computer Science*, 88(1):59–82, 1991.
- 10 Thomas Erlebach, Peter Rossmanith, Hans Stadtherr, Angelika Steger, and Thomas Zeugmann. Learning one-variable pattern languages very efficiently on average, in parallel, and by asking queries. *Theoretical Computer Science*, 261:119–156, 2001.
- 11 Henning Fernau, Florin Manea, Robert Mercas, and Markus L. Schmid. Revisiting Shinohara’s algorithm for computing descriptive patterns. Technical Report 14-3, Trier University, September 2014. https://www.uni-trier.de/fileadmin/fb4/INF/TechReports/descriptive_patterns_tech_report_Schmid.pdf.
- 12 Henning Fernau and Markus L. Schmid. Pattern matching with variables: A multivariate complexity analysis. In *Proceedings of the 24th Annual Symposium on Combinatorial Pattern Matching, CPM*, volume 7922 of *LNCS*, pages 83–94, 2013.
- 13 Henning Fernau, Markus L. Schmid, and Yngve Villanger. On the parameterised complexity of string morphism problems. In *Proceedings of the 33rd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, volume 24 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 55–66, 2013.
- 14 Jeffrey E. F. Friedl. *Mastering Regular Expressions*. O’Reilly, Sebastopol, CA, third edition, 2006.
- 15 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
- 16 Dan Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge University Press, New York, NY, USA, 1997.
- 17 Oscar H. Ibarra, Ting-Chuen Pong, and Stephen M. Sohn. A note on parsing pattern languages. *Pattern Recognition Letters*, 16:179–182, 1995.
- 18 Juhani Karhumäki, Wojciech Plandowski, and Filippo Mignosi. The expressibility of languages and relations by word equations. *Journal of the ACM*, 47:483–505, 2000.
- 19 Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *Journal of the ACM*, 53:918–936, 2006.

- 20 Michael Kearns and Leonard Pitt. A polynomial-time algorithm for learning k -variable pattern languages from examples. In *Proceedings of the 2nd Annual Conference on Learning Theory, COLT*, pages 57–71, 1989.
- 21 Tomasz Kociumaka, Jakub Radoszewski, Wojciech Rytter, and Tomasz Waleń. Efficient data structures for the factor periodicity problem. In *Proceedings of the 19th International Symposium on String Processing and Information Retrieval, SPIRE*, volume 7608 of *Lecture Notes in Computer Science*, pages 284–294, 2012.
- 22 M. Lothaire. *Combinatorics on Words*. Cambridge University Press, 1997.
- 23 M. Lothaire. *Algebraic Combinatorics on Words*, chapter 3. Cambridge University Press, Cambridge, New York, 2002.
- 24 Alexandru Mateescu and Arto Salomaa. Finite degrees of ambiguity in pattern languages. *RAIRO Informatique Théorique et Applications*, 28:233–253, 1994.
- 25 Yen K. Ng and Takeshi Shinohara. Developments from enquiries into the learnability of the pattern languages from positive data. *Theoretical Computer Science*, 397:150–165, 2008.
- 26 Sebastian Ordyniak and Alexandru Popa. A parameterized study of maximum generalized pattern matching problems. In *Proceedings of the 9th International Symposium on Parameterized and Exact Computation, IPEC*, 2014.
- 27 Daniel Reidenbach. Discontinuities in pattern inference. *Theoretical Computer Science*, 397:166–193, 2008.
- 28 Daniel Reidenbach and Markus L. Schmid. Patterns with bounded treewidth. *Information and Computation*, 239:87–99, 2014.
- 29 Takeshi Shinohara. Polynomial time inference of pattern languages and its application. In *Proceedings of the 7th IBM Symposium on Mathematical Foundations of Computer Science*, pages 191–209, 1982.

Approximating the Generalized Terminal Backup Problem via Half-integral Multiflow Relaxation

Takuro Fukunaga

National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan.
JST, ERATO, Kawarabayashi Large Graph Project, Japan.
takuro@nii.ac.jp

Abstract

We consider a network design problem called the generalized terminal backup problem. Whereas earlier work investigated the edge-connectivity constraints only, we consider both edge- and node-connectivity constraints for this problem. A major contribution of this paper is the development of a strongly polynomial-time $4/3$ -approximation algorithm for the problem. Specifically, we show that a linear programming relaxation of the problem is half-integral, and that the half-integral optimal solution can be rounded to a $4/3$ -approximate solution. We also prove that the linear programming relaxation of the problem with the edge-connectivity constraints is equivalent to minimizing the cost of half-integral multiflows that satisfy flow demands given from terminals. This observation implies a strongly polynomial-time algorithm for computing a minimum cost half-integral multiflow under flow demand constraints.

1998 ACM Subject Classification G.1.6 Optimization, G.2.2 Graph Theory

Keywords and phrases survivable network design, multiflow, LP rounding

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.316

1 Introduction

1.1 Generalized Terminal Backup Problem

The network design problem is the problem of constructing a low cost network that satisfies given constraints. It includes many fundamental optimization problems, and has been extensively studied. In this paper, we consider a network design problem called the *generalized terminal backup problem*, recently introduced by Bernáth and Kobayashi [3].

The generalized terminal backup problem is defined as follows. Let \mathbb{Q}_+ and \mathbb{Z}_+ denote the sets of non-negative rational numbers and non-negative integers, respectively. Let $G = (V, E)$ be an undirected graph with node set V and edge set E , $c: E \rightarrow \mathbb{Q}_+$ be an edge cost function, and let $u: E \rightarrow \mathbb{Z}_+$ be an edge capacity function. A subset T of V denotes the *terminal* node set in which each terminal t is associated with a connectivity requirement $r(t) \in \mathbb{Z}_+$. A solution is a multiple edge set on V containing at most $u(e)$ edges parallel to $e \in E$. The objective is to find a solution F that minimizes $\sum_{e \in F} c(e)$ under certain constraints. In Bernáth and Kobayashi [3], the subgraph (V, F) was required to contain $r(t)$ edge-disjoint paths that connect each $t \in T$ to other terminals. In addition to these edge-connectivity constraints, we consider node-connectivity constraints, under which the paths must be inner disjoint (i.e., disjoint in edges and nodes in $V \setminus T$) rather than edge-disjoint. To avoid confusion, we refer to the problem as *edge-connectivity terminal backup* when the edge-connectivity constraints are required, and as *node-connectivity terminal backup* when the node-connectivity constraints are imposed (removing “general” from the problem names



© Takuro Fukunaga;

licensed under Creative Commons License CC-BY

32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).

Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 316–328

Leibniz International Proceedings in Informatics

LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



makes no confusion because distinguishing edge-connectivity and node-connectivity implies that the connectivity requirement of a terminal is larger than one).

The generalized terminal backup problem models a natural data management situation. Suppose that each terminal represents a data storage server in a network, and $r(t)$ is the amount of data stored in the server at a terminal t . Backup data must be stored in servers different from that storing the original data. To this end, a sub-network that transfers data stored at one terminal to other terminals is required. We assume that edges can transfer a single unit of data per time unit. Hence, transferring data from terminal t to other terminals within one time unit requires $r(t)$ edge-disjoint paths from t to $T \setminus \{t\}$, which is represented by the edge-connectivity constraints. When nodes are also capacitated, $r(t)$ inner-disjoint paths are required; these requirements are met by the node-connectivity constraints.

The generalized terminal backup problem is interesting also from theoretical point of view. When $r \equiv 1$, the problem is called the terminal backup problem. Note that there is no difference between the edge- and the node-connectivity constraints when $r \equiv 1$. Anshelevich and Karagiozova [1] demonstrated that the terminal backup problem is reducible to the simplex matching problem, which is solvable in polynomial time. On the other hand, when $T = V$, the generalized terminal backup problem is equivalent to the capacitated b -edge cover problem with degree lower bound $b(v) = r(v)$ for $v \in V$. Since the capacitated b -edge cover problem admits a polynomial-time algorithm, the generalized terminal backup problem is solvable in polynomial time also when $T = V$. Therefore, we may naturally ask whether the generalized terminal backup problem is solvable in polynomial time. Bernáth and Kobayashi [3] proposed a polynomial-time algorithm for the uncapacitated case (i.e., $u(e) = +\infty$ for each $e \in E$) in the edge-connectivity terminal backup. Their result partially answers the above question, but their assumptions may be overly stringent in some situations; that is, their algorithm admits unfavorable solutions that select too many copies of a cheap edge. Moreover, their algorithm cannot deal with the node-connectivity constraints. Unfortunately, when the edge-capacities are bounded or node-connectivity constraints imposed, we do not know whether the generalized terminal backup problem is NP-hard or admits a polynomial-time algorithm. Instead, we propose approximation algorithms.

► **Theorem 1.** *There exist a strongly polynomial-time $4/3$ -approximation algorithm for the generalized terminal backup problem.*

The present study contributes two major advances to the generalized terminal backup problem.

- Bernáth and Kobayashi [3] discussed the generalized terminal backup problem in the uncapacitated setting with edge-connectivity constraints, noting that the problem in the capacitated setting is open. Here, we discuss the capacitated setting, and introduce the node-connectivity constraints.
- The generalized terminal backup problem can be formulated as the problem of covering skew supermodular biset functions, which is known to admit a 2-approximation algorithm. On the other hand, as stated in Theorem 1, we develop $4/3$ -approximation algorithms, that outperform this 2-approximation algorithm.

Let us explain the second advance more specifically. Given an edge set F and a nonempty subset X of V , let $\delta_F(X)$ denote the set of edges in F with one end node in X and the other in $V \setminus X$. Let $f: 2^V \rightarrow \mathbb{Z}_+$ be a function such that $f(X) = r(t)$ if $X \cap T = \{t\}$, and $f(X) = 0$ otherwise. By the edge-connectivity version of Menger's theorem, F satisfies the edge-connectivity constraints if and only if $|\delta_F(X)| \geq f(X)$ for each $X \in 2^V$. Bernáth and Kobayashi [3] showed that the function f is skew supermodular (skew supermodularity

is defined in Section 2). For any skew supermodular set function h , Jain [9] proposed a seminal 2-approximation algorithm for computing a minimum-cost edge set F satisfying $|\delta_F(X)| \geq h(X)$, $X \in 2^V$. Although the node-connectivity constraints cannot be captured by set functions as the edge-connectivity constraints, they can be regarded as a request for covering a skew supermodular *biset* function, to which the 2-approximation algorithm is extended [6]. Therefore, the generalized terminal backup problem admits 2-approximation algorithms, regardless of the imposed connectivity constraints. One of our contributions is to improve these 2-approximations to 4/3-approximations.

Both of the above 2-approximation algorithms involve iterative rounding of the linear programming (LP) relaxations. Primarily, their performance analyses prove that the value of a variable in each extreme point solution of the LP relaxations is at least 1/2. Once this property of extreme point solutions is proven, the variables can be repeatedly rounded until a 2-approximate solution is obtained. Our 4/3-approximation algorithms are based on the same LP relaxations as the iterative rounding algorithms. We show that, in the generalized terminal backup problem, all variables in extreme point solutions of the relaxation take half-integral values. We also prove that the half-integral solution can be rounded into an integer solution with loss of factor at most 4/3.

It may be helpful for understanding our result to see the well-studied special case of $T = V$ and $u(e) = 1$ for each $e \in E$ (i.e., feasible solutions are simple r -edge covers). In this case, our LP relaxation minimizes $\sum_{e \in E} c(e)x(e)$ subject to $\sum_{e \in \delta(v)} x(e) \geq r(v)$ for each $v \in V$ and $0 \leq x(e) \leq 1$ for each $e \in E$, where $\delta(v)$ is the set of edges incident to the node v . It has been already known that an extreme point solution of this LP is half-integral, and the edges in $\{e \in E: x(e) = 1/2\}$ form odd cycles. The half-integral variables of the edges on an odd cycle can be rounded as follows. Suppose that edges e_1, \dots, e_k appears in the cycle in this order, where k is the cycle length (i.e., odd integer larger than one). For each $i, j \in \{1, \dots, k\}$, we define $x'_i(e_j) = 1$ if $j \geq i$ and $j \equiv i \pmod 2$, or if $j < i$ and $j \equiv i + 1 \pmod 2$, and $x'_i(e_j) = 0$ otherwise. Note that exactly $(k + 1)/2$ variables in $x'_1(e_1), \dots, x'_k(e_k)$ are equal to one, and the other $(k - 1)/2$ variables are equal to zero for each j . This means that

$$\sum_{i=1}^k \sum_{j=1}^k c(e_j)x'_i(e_j) = \sum_{j=1}^k c(e_j) \cdot \frac{k+1}{2} = (k+1) \sum_{j=1}^k c(e_j)x(e_j).$$

Let i^* minimize $\sum_{j=1}^k c(e_j)x'_{i^*}(e_j)$ in $i^* \in \{1, \dots, k\}$. Then, since $\sum_{j=1}^k c(e_j)x'_{i^*}(e_j) \leq \sum_{i=1}^k \sum_{j=1}^k c(e_j)x'_i(e_j)/k$, replacing $x(e_1), \dots, x(e_k)$ by $x'_{i^*}(e_1), \dots, x'_{i^*}(e_k)$ increases their costs by a factor at most $(k + 1)/k \leq 4/3$. We also observe that the feasibility of the solution is preserved even after the replacement. By applying this rounding for each odd cycle, the half-integral solution can be transformed into a 4/3-approximate integer solution.

Our result is obtained by extending the characterization of the edge structure whose corresponding variables are not integers, but the extension is not immediate. As in the above special case, those edges form cycles in the generalized terminal backup problem if the solution is a minimal feasible solution for the LP relaxation. However, the length of a cycle is not necessarily odd, and it is not clear how the half-integral solution should be rounded; In the above special case, we round up and down variables of edges on a cycle alternatively, but this obviously does not preserve the feasibility in the generalized terminal backup problem. The key ingredient in our result is to characterize the relationship between the cycles and the node sets or bisets corresponding to linearly independent tight constraints in the LP relaxation. We show that a cycle crosses maximal tight node set or bisets an odd number of times, which extends the property that the length of each cycle is odd in the special case.

Our rounding algorithm decides how to round a non-integer variable from the direction of the crossing between the corresponding edge and a tight node set or biset.

1.2 Minimum Cost Multiflow Problem

Multiflows are closely related to the generalized terminal backup problem. Among the many multiflow variants, we focus on the type sometimes called *free multiflows*. For $t, t' \in T$, $\mathcal{A}_{t,t'}$ denotes the set of paths that terminate at t and t' . Let \mathcal{A}_t denote $\bigcup_{t' \in T \setminus \{t\}} \mathcal{A}_{t,t'}$, and \mathcal{A} denote $\bigcup_{t \in T} \mathcal{A}_t$. $E(A)$ and $V(A)$ denote the sets of edges and nodes in $A \in \mathcal{A}$, respectively. We define a multiflow as a function $\psi: \mathcal{A} \rightarrow \mathbb{Q}_+$. In the edge-capacitated setting, an edge capacity $u(e) \in \mathbb{Z}_+$ is given, and we must satisfy $\sum\{\psi(A): A \in \mathcal{A}, e \in E(A)\} \leq u(e)$ for each $e \in E$. In the node-capacitated setting, a node capacity $u(v) \in \mathbb{Z}_+$ is given and $\sum\{\psi(A): A \in \mathcal{A}, v \in V(A)\} \leq u(v)$ is required for each $v \in V$. The multiflow ψ is called an *integral* multiflow if $\psi(A) \in \mathbb{Z}_+$ for each $A \in \mathcal{A}$, and is called a *half-integral* multiflow if $2\psi(A) \in \mathbb{Z}_+$ for each $A \in \mathcal{A}$. Let $c(A)$ denote $\sum_{e \in E(A)} c(e)$ for $A \in \mathcal{A}$. The cost of ψ is given by $\sum_{A \in \mathcal{A}} \psi(A)c(A)$.

In the edge-connectivity terminal backup, the connectivity requirement from a terminal t equates to requiring that a flow of amount $r(t)$ can be delivered from t to $T \setminus \{t\}$ in the graph (V, F) with unit edge-capacities if F is a feasible solution. This condition appears similar to the constraint that the graph (V, F) with unit edge-capacities admits a multiflow ψ such that $\sum_{A \in \mathcal{A}_t} \psi(A) \geq r(t)$. We note that (V, F) with unit edge-capacities admits a multiflow ψ if and only if the number of copies of $e \in E$ in F is at least $\sum_{A \in \mathcal{A}: e \in E(A)} \psi(A)$. These observations suggest a correspondence between the edge-connectivity terminal backup and the problem of finding a minimum cost multiflow ψ under the constraint that $\sum_{A \in \mathcal{A}_t} \psi(A) \geq r(t)$ for $t \in T$ in the edge-capacitated setting. We refer to such a multiflow computation as the *minimum cost multiflow problem* (in the edge-capacitated setting). The same correspondence exists between the node-connectivity terminal backup and the node-capacitated setting in the minimum cost multiflow problem.

However, the generalized terminal backup and the minimum cost multiflow problems are not equivalent. Especially, the minimum cost multiflow problem can be formulated in LP, whereas the generalized terminal backup problem is an integer programming problem. Even if multiflows are restricted to integral multiflows, the two problems are not equivalent. To observe this, let $G = (V, E)$ be a star with an odd number of leaves. We assume that T is the set of leaves, and each edge incurs one unit of cost. This star is a feasible solution to the terminal backup problem (i.e., $r(t) = 1$ for $t \in T$). In contrast, setting $r \equiv 1$ and $u \equiv 1$ admits no integral multiflow in the edge-capacitated setting, and no feasible (fractional) multiflows in the node-capacitated setting.

Nevertheless, similarities exist between terminal backups and multiflows. As mentioned above, we will show that an LP relaxation of the generalized terminal backup problem always admits a half-integral optimal solution. Similarly, half-integrality results are frequently reported for multiflows. Lovász [12] and Cherkassky [5] investigated $r \equiv 0$ in the edge-capacitated setting, and showed that a half-integral multiflow maximizes $\sum_{A \in \mathcal{A}} \psi(A)$ over all multiflows ψ . Using an identical objective function to ours, Karzanov [11, 10] sought to minimize the cost of multiflows. His feasible multiflow solutions are those attaining $\max \sum_{A \in \mathcal{A}} \psi(A)$ in the edge-capacitated setting with $r \equiv 0$, and he showed that the minimum cost is achieved by a half-integral multiflow. Babenko and Karzanov [2] and Hirai [7] extended Karzanov's result to node-cost minimization in the node-capacitated setting. In this scenario also, the optimal multiflow is half-integral.

In the present paper, we present a useful relationship between the generalized terminal

backup problem and the minimum cost multiflow problem in the edge-capacitated setting. We prove that the optimal solution of the LP used to approximate the edge-connectivity terminal backup is a half-integral multiflow, which also optimizes the minimum cost multiflow problem. Thereby, we can compute the minimum cost half-integral multiflow by solving the LP relaxation. This result is summarized in the following theorem.

► **Theorem 2.** *The minimum cost multiflow problem admits a half-integral optimal solution in the edge-capacitated setting, which can be computed in strongly polynomial time.*

In contrast, we find no useful relationship between the node-connectivity terminal backup and the node-capacitated setting of the minimum cost multiflow problem. We can only show that the LP relaxation of the node-connectivity terminal backup also has an optimal solution which is a half-integral multiflow in the edge-capacitated setting.

Despite its natural formulation, the minimum cost multiflow problem has not been previously investigated to our knowledge. We emphasize that Theorem 2 cannot be derived from previously known results on multiflows. The minimum cost multiflow problem may be solvable by reducing it to minimum cost maximum multiflow problems that (as mentioned above) admit polynomial-time algorithms. A naive reduction can be implemented as follows. Let ψ^* be a minimum cost multiflow that satisfies the flow demands from terminals, and let $\nu(t) = \sum_{A \in \mathcal{A}_t} \psi^*(A)$ for each $t \in T$. For each $t \in T$, we add a new node t' and connect t and t' by a new edge of capacity $\nu(t)$. The new terminal set T' is defined as $\{t' : t \in T\}$. Now the multiflow ψ^* can be extended to the multiflow of maximum flow value for the terminal set T' . Applying the algorithm in [11] to this new instance, we can solve the original problem. Moreover, if $\nu(t)$ is an integer for each $t \in T$, this reduction together with the half-integrality result in [10, 11] implies that an optimal multiflow in the minimum cost multiflow problem is half-integral. However, this naive reduction has two limitations. First, $\nu(t)$ is indeterminable without computing ψ^* . We only know that $\nu(t)$ cannot be smaller than $r(t)$. Second, we cannot ascertain that $\nu(t)$ is always an integer for each $t \in T$. Hence, this naive reduction seems to yield neither a polynomial-time algorithm nor the half-integrality of optimal multiflows claimed in Theorem 2.

Applying a structural result in [3] on the generalized terminal backup problem, it is easily shown that any integral solution to the edge-connectivity terminal backup provides a half-integral multiflow at the same cost. However, since the way to find an optimal solution for the edge-connectivity terminal backup is unknown, Theorem 2 is not derivable from this relationship. In proving the half-integrality of the LP relaxation required for Theorem 1, we immediately imply the quarter-integrality of a minimum cost multiflow (i.e., $4\psi(A) \in \mathbb{Z}_+$ for each $A \in \mathcal{A}$). The proof of Theorem 2 requires deeper investigation into the structure of half-integral LP solutions.

1.3 Structure of This Paper

In this article, due to the space limitation, we only sketch how our $4/3$ -approximation algorithm works in the case of $r \equiv 1$. We also omit proofs of several lemmas. We recommend referring to the full version of the present paper for the full description of our results. Section 2 introduces notations and essential preliminary facts. Section 3 presents required properties of extreme point solutions of an LP relaxation to the edge-connectivity terminal backup. Section 4 introduces our $4/3$ -approximation algorithm, restricted to the case of $r \equiv 1$. Section 5 concludes the paper.

2 Preliminaries

For an edge set F and a node set $X \in 2^V$, let $\delta_F(X)$ denote the set of edges in F with one end node in X and the other in $V \setminus X$. We identify a node $v \in V$ with the node set $\{v\}$. Thereby $\delta_F(v)$ denotes the set of edges incident to v in F . For simplicity, we write $\delta_E(X)$ as $\delta(X)$ when the edge set is unambiguously E . If an edge e is in $\delta(X)$, we say that e is *incident* to X .

We say that $X \in 2^V$ and $Y \in 2^V$ are *noncrossing* when $X \cap Y = \emptyset$, $X \subseteq Y$, or when $Y \subseteq X$. Otherwise, X and Y are called *crossing*. A family of node sets is called *laminar* if each pair of node sets in the family is noncrossing. The laminarity naturally defines a child-parent relationship among those in the family; if $X, Y \in \mathcal{L}$, and if $Y \in \mathcal{L}$ is the minimal node set such that $X \subseteq Y$ in \mathcal{L} , then Y is defined as the *parent* of X , and X is a *child* of Y . This child-parent relationship naturally leads to terminologies such as “ancestor” and “descendant.” For a node set Y in a laminar family \mathcal{L} and an edge set F , we let $F_{\mathcal{L}}^+(Y)$ and $F_{\mathcal{L}}^-(Y)$ respectively denote $\delta_F(Y) \setminus (\bigcup_{X \in \mathcal{X}} \delta_F(X))$ and $(\bigcup_{X \in \mathcal{X}} \delta_F(X)) \setminus \delta_F(Y)$, where \mathcal{X} denotes the set of children of Y in \mathcal{L} . If Y has no child, $F_{\mathcal{L}}^+(Y) = \delta_F(Y)$ and $F_{\mathcal{L}}^-(Y) = \emptyset$.

For $t \in T$, let $\mathcal{C}(t) = \{X \in 2^V : X \cap T = \{t\}\}$. We denote $\bigcup_{t \in T} \mathcal{C}(t)$ by \mathcal{C} . For a vector $x \in \mathbb{Q}_+^E$ and $E' \subseteq E$, let $x(E')$ represent $\sum_{e \in E'} x(e)$. Recall that in Section 1, we defined the set function f representing the edge-connectivity constraints by

$$f(X) = \begin{cases} r(t), & \text{if } t \in T, X \in \mathcal{C}(t), \\ 0, & \text{otherwise} \end{cases}$$

for each $X \in 2^V$.

Given a function $h: 2^V \rightarrow \mathbb{Q}_+$ and an edge-capacity function $u: E \rightarrow \mathbb{Z}_+$, we define $P(h, u)$ as the set of $x \in \mathbb{Q}_+^E$ such that

$$x(\delta(X)) \geq h(X) \quad \text{for } X \in 2^V \tag{1}$$

and $x(e) \leq u(e)$ for $e \in E$.

Let F be a multiset of edges in E , and χ_F denote the characteristic vector of F (i.e., $\chi_F \in \mathbb{Z}_+^E$ and F contains $\chi_F(e)$ copies of e for each $e \in E$). Note that $|\delta_F(X)| = \chi_F(\delta(X))$ for $X \in 2^V$. Hence, $\chi_F \in P(f, u)$ if and only if F is a feasible solution to the edge-connectivity terminal backup. These statements imply that the LP $\text{LP}(h, u) = \min \{\sum_{e \in E} c(e)x(e) : x \in P(h, u)\}$ relaxes the edge-connectivity terminal backups when $h = f$.

A biset function h is called (*positively*) *skew supermodular* when, for any $X \in 2^V$ with $h(X) > 0$ and $Y \in 2^V$ with $h(Y) > 0$, h satisfies

$$h(X) + h(Y) \leq h(X \cap Y) + h(X \cup Y) \tag{2}$$

or

$$h(X) + h(Y) \leq h(X \setminus Y) + h(Y \setminus X). \tag{3}$$

For any function $h: 2^V \rightarrow \mathbb{Q}_+$ and a vector $x: E \rightarrow \mathbb{Q}_+$, we let h_x denote the function such that $h_x(X) = h(X) - x(\delta(X))$ for each $X \in 2^V$. Bernáth and Kobayashi [3] reported that f_x is skew supermodular for any $x: E \rightarrow \mathbb{Q}_+$.

3 Structure of Extreme Point Solutions

In this section, we present the properties of the extreme points of $P(f, u)$. More precisely, we prove that each extreme point of $P(f, u)$ is half-integral, and that the edges whose

corresponding variables are half-integral are characteristically structured. Note that f is an integer-valued skew supermodular function, and $f(X) = 0$ for any $X \notin \mathcal{C}$.

3.1 Half-Integrality

In the following, we denote an integer-valued skew supermodular function by h , and an extreme point of $P(h, u)$ by x . Given an edge set F on V and a node set $X \in 2^V$, let $\eta_{F, X}$ denote the characteristic vector of $\delta_F(X)$, i.e., an $|F|$ -dimensional vector whose components are set to 1 if indexed by an edge in $\delta_F(X)$, and 0 otherwise. The following lemma has been previously proposed [9].

► **Lemma 3.** *Let $h: 2^V \rightarrow \mathbb{Q}_+$ be a skew supermodular function, and x be an extreme point of $P(h, u)$. Let $E_0 = \{e \in E: x(e) = 0\}$, $E_1 = \{e \in E: x(e) = u(e)\}$, and $F = E \setminus (E_0 \cup E_1)$. Let \mathcal{L} be an inclusion-wise maximal laminar subfamily of $\{X \in 2^V: x(\delta_F(X)) = h(X) - u(\delta_{E_1}(X)) > 0\}$ such that the vectors in $\{\eta_{F, X}: X \in \mathcal{L}\}$ are linearly independent. Then $|F| = |\mathcal{L}|$, and x is a unique vector that satisfies $x(\delta_F(X)) = h(X) - u(\delta_{E_1}(X)) > 0$ for each $X \in \mathcal{L}$, $x(e) = 0$ for each $e \in E_0$, and $x(e) = u(e)$ for each $e \in E_1$.*

We note that \mathcal{L} in Lemma 3 can be constructed in a greedy way; initialize \mathcal{L} to an empty set, and repeatedly add a biset X such that $x(\delta_F(X)) = h(X) - u(\delta_{E_1}(X)) > 0$ and $\eta_{F, X}$ is linearly independent of the characteristic vectors in the current \mathcal{L} . Hereafter, we assume that \mathcal{L} is constructed as claimed in Lemma 3. Similarly, E_0 , E_1 , and F are defined from x as in Lemma 3.

Let $\bar{x}: E \rightarrow \mathbb{Z}_+$, and define a function $h_{\bar{x}}(X)$ as $h(X) - \bar{x}(\delta(X))$ for $X \in 2^V$. Let $\mathbf{1}$ denote the $|E|$ -dimensional all-one vector. The following lemma relates only to the extreme points of $P(h_{\bar{x}}, \mathbf{1})$. In Corollary 5, we will show that this is sufficient for proving the half-integrality of $P(h, u)$. If $h(X) > 0$ holds only for $X \in \mathcal{C}$, we have $\mathcal{L} \subseteq \mathcal{C}$. In this case, no node set in \mathcal{L} has more than one child, and x is characterized as follows.

► **Lemma 4.** *Suppose that $h: 2^V \rightarrow \mathbb{Z}_+$ is an integer-valued skew supermodular function such that $h(X) > 0$ only for $X \in \mathcal{C}$. Let $\bar{x}: E \rightarrow \mathbb{Z}_+$, and let x be an extreme point of $P(h_{\bar{x}}, \mathbf{1})$. Let F denote $\{e \in E: 0 < x(e) < 1\}$. Then the following conditions hold:*

- (i) $|F_{\mathcal{L}}^+(X)| + |F_{\mathcal{L}}^-(X)| = 2$ for each $X \in \mathcal{L}$;
- (ii) If $e \in F$ is incident to a maximal node set in \mathcal{L} , then it is incident to exactly two maximal node sets in \mathcal{L} ;
- (iii) $x(e) = 1/2$ for each $e \in F$.

► **Corollary 5.** *Suppose that $h: 2^V \rightarrow \mathbb{Q}_+$ is a skew supermodular function such that $h(X) > 0$ only if $X \in \mathcal{C}$. Let $u: E \rightarrow \mathbb{Z}_+$. Given $x \in P(h, u)$, we define $\bar{x}: E \rightarrow \mathbb{Z}_+$ and $x': E \rightarrow \mathbb{Q}_+$ by $\bar{x}(e) = \lfloor x(e) \rfloor$ and $x'(e) = x(e) - \bar{x}(e)$, respectively for each $e \in E$. If x is an extreme point of $P(h, u)$, then x' is an extreme point of $P(h_{\bar{x}}, \mathbf{1})$. Moreover, $P(h, u)$ is half-integral if h is integer-valued.*

3.2 Path decompositions of extreme point solutions

In this section, we consider $x \in P(f, u)$. We denote $\{X \in \mathcal{L}: t \in X\}$ by $\mathcal{L}(t)$ for each $t \in T$. Let $t \in T$ with $\mathcal{L}(t) \neq \emptyset$, and let X be the maximal node set in $\mathcal{L}(t)$. We obtain a graph $G^s[X]$ from G by shrinking all the nodes in $V \setminus X$ into a single node s . Removing s from $G^s[X]$, we obtain another graph $G[X]$ (i.e., $G[X]$ is the subgraph of G induced by X). We suppose that each edge e in $G^s[X]$ or in $G[X]$ is capacitated by $x(e)$. Since all capacities are

half-integral, the maximum flow between s and t in $G^s[X]$ can be decomposed into a set of paths $R_1^t, \dots, R_{2r(t)}^t$ each of which accommodates a half unit of flow.

Let $X' \in \mathcal{L}(t)$. Each path between s and t passes through an edge in $\delta(X')$. Since $x(\delta(X')) = r(t)$, the edges in $\delta(X')$ are used to full capacity by the maximum flow, and each path R_i^t includes exactly one edge in $\delta(X')$.

Suppose that both R_i^t and R_j^t include a node $v \notin \{s, t\}$. Let e_i and e'_i be the edges incident to v on R_i^t , where e_i is near to s than e'_i . We define the edges e_j and e'_j incident to v on R_j^t , similarly. We assume that the following fact holds for any such paths R_i^t and R_j^t .

► **Assumption 1.** If $x(e_i)$ is half-integral and $x(e_j)$ is an integer, and if exactly one of $x(e'_i)$ and $x(e'_j)$ is half-integral, then $x(e'_i)$ is half-integral.

Indeed, if Assumption 1 does not hold, then exchanging the subpaths between v and t makes them satisfy it.

In the following discussion, we consider a maximum flow between a terminal t' and $T \setminus \{t'\}$ in G , where t' may equal t . In such a flow, each edge e is capacitated by $x(e)$. The flow quantity is at least $r(t')$ if and only if x satisfies (1) with $h = f$. Let \mathcal{S} be a path decomposition of the flow, in which each path in \mathcal{S} accommodates a half unit of flow. Let \mathcal{S}_t be the set of paths in \mathcal{S} that contain nodes in X (recall that X is the maximal node set in $\mathcal{L}(t)$). Without loss of generality, we can state the following fact.

► **Assumption 2.** Each path in \mathcal{S}_t ends at t . Moreover, $\{S' : S \in \mathcal{S}_t\} \subseteq \{R_1^t, \dots, R_{2r(t)}^t\}$, where S' is the subpath of S between t and the nearest node in $V \setminus X$.

If Assumption 2 is not satisfied by \mathcal{S} , we can modify the flow between t' and $T \setminus \{t'\}$ by replacing the subpaths of those in \mathcal{S}_t by appropriate paths in $R_1^t, \dots, R_{2r(t)}^t$, without decreasing the amount of flow.

We say that x is *minimal in $P(f, u)$* if $x \in P(f, u)$ and no $y \in P(f, u)$ exists such that $x \neq y$ and $x(e) \geq y(e)$ for any $e \in E$. Let edge e' be incident to a node in X . If x is minimal in $P(f, u)$, then $x(e') = |\{i = 1, \dots, 2r(t) : e' \in E(R_i^t)\}|/2$; Otherwise, as $x(e)$ is decreased, it would remain in $P(f, u)$.

► **Lemma 6.** Let x be an extreme minimal point in $P(f, u)$. Then $x(\delta(v))$ is an integer for each $v \in V$.

4 LP-rounding $4/3$ -Approximation Algorithm for Terminal Backup Problem

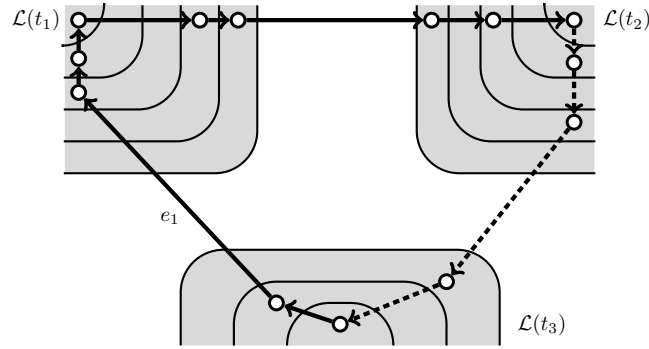
Our algorithm rounds a half-integral optimal solution to the LP relaxations into an integer solution. Let us assume that a minimal half-integral optimal solution x and a laminar family \mathcal{L} in Lemma 3 are given. In what follows, we explain how to round x .

Let F denote $\{e \in E : x(e) = 1/2\}$. We call the edges in F *half-integral edges*. $|\delta_F(v)|$ is even for each $v \in V$ because $x(\delta(v))$ is an integer by Lemma 6. Hence F can be decomposed into an edge-disjoint set of cycles. Let H be a cycle in the decomposition.

For each $e \in F$, \mathcal{L} contains a node set to which e is incident. Let \mathcal{L}' be the subfamily of \mathcal{L} that consists of the node sets to which edges in H are incident. Since $r \equiv 1$, exactly two edges in H are incident to each node set in \mathcal{L}' .

Let t_1, \dots, t_k be the terminals such that $\mathcal{L}(t_i) \cap \mathcal{L}' \neq \emptyset$ for each $i \in \{1, \dots, k\}$. We can prove the following lemma.

► **Lemma 7.** k is an odd integer larger than one.



■ **Figure 1** An example of a cycle of half-integral edges and the first assignment of labels to the edges. Edges drawn by solid and dashed lines are assigned “+” and “-,” respectively. The edges are oriented in the direction of traverse. The areas surrounded by thin solid lines represent the node sets in \mathcal{L} .

For each $i \in \{1, \dots, k\}$, let X_i denote the maximal node set in $\mathcal{L}(t_i) \cap \mathcal{L}'$, and let H_i be the subpath of H comprising of edges incident to node sets in $\mathcal{L}(t_i) \cap \mathcal{L}'$. Hence, if an edge are incident to both X_i and X_j , the edge is shared by H_i and H_j .

Let $e_1 = uv$ be an edge incident to X_1 , where we assume without loss of generality that $u \in X_1$ and $v \notin X_1$. Consider traversing $E(H)$, starting from e_1 in the direction from v to u . We say that t_i *appears* when we traverse an edge incident to two node sets $X_i \in \mathcal{L}(t_i)$ and $X_j \in \mathcal{L}(t_j)$ with $i \neq j$ in the direction from the end node in X_j to the one in X_i . Without loss of generality, we assume that the terminals appear in the increasing order of subscripts. Therefore, during the traverse of H , we first visit edges in H_1 , then those in H_2 , and so on. Suppose that $X \in \mathcal{L}(t_i)$ and $e \in \delta_H(X)$. We say that e is *outward* with respect to t_i if e is traversed from the end node in X to the other. Otherwise, e is called *inward*. This implies that, during the traverse of H_i , we first traverse edges inward with respect to t_i , and then those outward with respect to t_i .

We define k assignments of labels to the edges in H , where each edge is labeled by either “+” or “-.” Let us define the i -th assignment. If $e \in E(H_i)$, then e is labeled by “+.” If $e \in E(H_j)$ for some $j \neq i$, then its label is decided by the following rules.

- If $j - i$ is odd and e is outward with respect to t_j , e is labeled by “+.”
- If $j - i$ is odd and e is inward with respect to t_j , e is labeled by “-.”
- If $j - i$ is even and e is outward with respect to t_j , e is labeled by “-.”
- If $j - i$ is even and e is inward with respect to t_j , e is labeled by “+.”

Note that this assignment is consistent; if e is included in both H_j and H_{j+1} , then e is outward with respect to t_j and inward with respect to t_{j+1} , and hence e is assigned the same label from j and $j + 1$; e_1 is shared by H_1 and H_k , and similarly it is assigned the same label because k is odd. Figure 1 shows an example of the cycle H , and the first assignment of labels to the edges on H .

Our algorithm rounds $x(e)$ into 1 if e is labeled by “+,” and into 0 otherwise. Since we have k assignments of labels, we have k ways of rounding of edges in H . Our algorithm chooses the most cost-effective one among them.

Let us observe that this algorithm achieves $4/3$ -approximation. First, we prove that the above rounding increases the cost by a factor of at most $4/3$. Let x' be the vector obtained from x by the rounding.

► **Lemma 8.**

$$\sum_{e \in E} c(e)x'(e) \leq \frac{4}{3} \sum_{e \in E} c(e)x(e).$$

Proof. Let H be a cycle of half-integral edges. We show that

$$\sum_{e \in H} c(e)x'(e) \leq \frac{4}{3} \sum_{e \in H} c(e)x(e).$$

Applying this claim to all cycles in the decomposition of F , we can prove the lemma. We use the notations used in the definition of the rounding.

Let x_i denote the vector obtained by rounding $x(e)$, $e \in E(H)$ according to the i -th assignment of labels. We note that

$$\sum_{e \in H} c(e)x'(e) = \min_{1 \leq i \leq k} \sum_{e \in H} c(e)x_i(e) \leq \frac{1}{k} \sum_{i=1}^k \sum_{e \in H} c(e)x_i(e).$$

Recall that k is an odd number larger than one. In the k assignments, $e \in H$ is labeled “+” by the $(k+1)/2$ assignments. Thus,

$$\sum_{i=1}^k \sum_{e \in H} c(e)x_i(e) = \frac{k+1}{2} \sum_{e \in H} c(e).$$

Note that $\sum_{e \in H} c(e)x(e) = \sum_{e \in H} c(e)/2$. Therefore,

$$\frac{\sum_{e \in H} c(e)x'(e)}{\sum_{e \in H} c(e)x(e)} \leq \frac{k+1}{k} \leq \frac{4}{3},$$

where the last inequality follows from $k \geq 3$. ◀

Next, let us prove the feasibility of x' . For a path P and nodes u, v on P , we denote the subpath of P between u and v by $P[u, v]$.

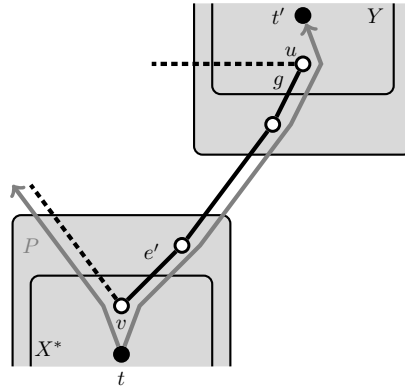
► **Lemma 9.** x' is a feasible solution to the terminal backup problem.

Proof. Obviously x' is an integer vector. Hence, to prove the feasibility of x' , the graph with edge-capacities x' admits a unit of flow from each terminal t to the other terminals. Since $x(\delta(X)) \geq 1$ for each $X \in \mathcal{C}(t)$, the graph capacitated by x admits such a flow. Hence we show that a flow for x' can be obtained by modifying the flow for x . In the following, we assume that x' is obtained by rounding variables corresponding to the half-integral edges in a cycle H . If required, the modification is repeated for each cycle of half-integral edges.

Recall the definitions of $R_1^t, \dots, R_{2r(t)}^t$ in Section 3.2. Since we are considering $r \equiv 1$, we have two paths R_1^t and R_2^t for each terminal t with $\mathcal{L}(t) \neq \emptyset$. We assume these paths satisfy Assumption 1. Fix a terminal t , and suppose that the flow from t to the other terminals with edge-capacities x delivers a half unit of flow along a path P , and another half unit along a path Q . We assume that $\mathcal{S} = \{P, Q\}$ satisfies Assumption 2.

If both P and Q contains no half-integral edge (with respect to x) labeled by “−,” the flow satisfies the capacity constraints defined from x' . Thus, let us consider the case where P includes a half-integral edge labeled by “−.” Let e be the one nearest to t among such edges, and let v be the end node of e near to t .

We first show that there exists $X^* \in \mathcal{L}(t)$ such that $e \in \delta(X^*)$ and $v \in X^*$. For arriving at a contradiction, suppose that such X^* does not exist. e is incident to at least one node



■ **Figure 2** The definitions in the proof of Lemma 9.

set in \mathcal{L} . In particular, Lemma 4(ii) implies that there exists a terminal $t' \in T$ and node set $X' \in \mathcal{L}(t')$ such that $e \in \delta(X')$ and $v \in X'$. However, this means that $t' \neq t$ and $P[t, v]$ enters X' when traversed from t to v . Assumption 2 indicates that the subpath of P between v and the end opposite to t is included by $R_1^{t'}$ or $R_2^{t'}$. Hence, the end of P opposite to t is t' , and P does not include e , which is a contradiction. Therefore, there exists $X \in \mathcal{L}(t)$ such that $e \in \delta(X)$ and $v \in X$.

This fact indicates that Q contains no “-”-labeled half-integral edge because of the following reason. Let P' be the maximal subpath of P that consists of edges incident to node sets in $\mathcal{L}(t)$. Since $\mathcal{L}(t) \neq \emptyset$, there exists R_1^t and R_2^t . By Assumption 2, P' is equal to R_1^t or R_2^t . Without loss of generality, let P' be equal to R_1^t . Then, Assumption 1 indicates that all “-”-labeled half-integral edges incident to node sets in $\mathcal{L}(t)$ is included in R_1^t . Since P and Q share no half-integral edges, Q does not include these edges in R_1^t . Hence, if Q contains a “-”-labeled half-integral edge, its both end node is included by some node sets in $\mathcal{L} \setminus \mathcal{L}(t)$. However, we can derive a contradiction similarly for the above claim with P .

Since $x(\delta(X^*)) = 1$, the other edge e' incident to v on H is also incident to X^* . By the label-assignment rules, e' is labeled by “+.” Let H' denote the subpath of H consisting of “+”-labeled edges and terminating at v . Let u be the other end node of H' , and let g be the edge incident to u on H' . By Lemma 4, there exists $Y \in \mathcal{L}$ with $g \in \delta(Y)$ and $u \in Y$. Y belongs to $\mathcal{L}(t')$ for some $t' \neq t$. g is included in a path $R_1^{t'}$ or $R_2^{t'}$. Without loss of generality, we suppose that $R_1^{t'}$ includes g' . We replace P by the concatenate of $P[t, v]$, H' , and $R_1^{t'}[u, t']$. See Figure 2 for illustration of this modification.

Let us observe that this modification preserves the capacity constraints. $P[t, v]$ was a part of P before the modification. The capacity of each edge on H' is increased by $1/2$ when x' replaces x . The capacity of each edge in $R_1^{t'}[u, t']$ is integer. Hence no capacity constraint is violated. ◀

5 Conclusion

We have presented $4/3$ -approximation algorithms for the generalized terminal backup problem. Our result also implies that the integrality gaps of the LP relaxations are at most $4/3$. These gaps are tight even in the edge cover problem (i.e., $T = V$ and $r \equiv 1$): Consider an instance in which G is a triangle with unit edge costs; The half-integral solution x with $x(e) = 1/2$ for all $e \in E$ is feasible to the LPs, and its cost is $3/2$; On the other hand, any integer solution

chooses at least two edges from the triangle; Since the costs of these integer solutions are at least 2, the integrality gap is not smaller than $4/3$ in this instance.

An obvious open problem is whether the generalized terminal backup problem admits polynomial-time exact algorithms or not. It seems hard to obtain such an algorithm by rounding solutions of the LP relaxations because of their integrality gaps. For the capacitated b -edge cover problem, an LP relaxation of integrality gap one is known [13]. For obtaining an LP-based polynomial-time algorithm for the generalized terminal backup problem, we have to extend this LP relaxation for the capacitated b -edge cover problem.

Another interesting approach is offered by combinatorial approximation algorithms because it is currently a major open problem to find a combinatorial constant-factor approximation algorithm for the survivable network design problem, for which the Jain's iterative rounding algorithm [9] achieves 2-approximation. The survivable network design problem involves more complicated connectivity constraints than the generalized terminal backup problem. Hence, study on combinatorial algorithms for the latter problem may give useful insights for the former problem. Recently, Hirai [8] showed that $\text{LP}(f, u)$ can be solved by a combinatorial algorithm. Indeed, he also showed that his algorithm can be used to implement our $4/3$ -approximation algorithm for the edge-connectivity terminal backup without generic LP solvers.

Many problems related to multiflows also remain open. We have shown that an LP solution provides a minimum cost half-integral multiflow that satisfies the flow demand from each terminal in the edge-capacitated setting. However, how the computation should proceed in the node-capacitated setting remains elusive. Computing a minimum cost integral multiflow under the same constraints is yet another problem worth investigating. We note that Burlet and Karzanov [4] solved a similar problem related to integral multiflows in the edge-capacitated setting. Their problem differs from ours in the fact that $\sum_{A \in \mathcal{A}_t} \psi(A)$ is required to match the specified value for each terminal t .

Acknowledgements This work was partially supported by JSPS KAKENHI Grant Number 25730008. The author thanks Hiroshi Hirai for sharing information on multiflows and his results in [8].

References

- 1 Elliot Anshelevich and Adriana Karagiozova. Terminal backup, 3D matching, and covering cubic graphs. *SIAM Journal on Computing*, 40(3):678–708, 2011.
- 2 Maxim A. Babenko and Alexander V. Karzanov. Min-cost multiflows in node-capacitated undirected networks. *Journal of Combinatorial Optimization*, 24(3):202–228, 2012.
- 3 Attila Bernáth and Yusuke Kobayashi. The generalized terminal backup problem. In *SODA*, pages 1678–1686, 2014.
- 4 Michel Burlet and Alexander V. Karzanov. Minimum weight (T, d) -joins and multi-joins. *Discrete Mathematics*, 181(1-3):65–76, 1998.
- 5 Boris V. Cherkassky. A solution of a problem on multicommodity flows in a network. *Ekonomika i Matematicheskie Metody*, 13(1):143–151, 1977.
- 6 Lisa Fleischer, Kamal Jain, and David P. Williamson. Iterative rounding 2-approximation algorithms for minimum-cost vertex connectivity problems. *Journal of Computer and System Sciences*, 72(5):838–867, 2006.
- 7 Hiroshi Hirai. Half-integrality of node-capacitated multiflows and tree-shaped facility locations on trees. *Mathematical Programming*, 137(1-2):503–530, 2013.

- 8 Hiroshi Hirai. L-extendable functions and a proximity scaling algorithm for minimum cost multiflow problem. *ArXiv e-prints*, November 2014.
- 9 Kamal Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.
- 10 Alexander V. Karzanov. A problem on maximum multiflow of minimum cost. *Combinatorial Methods for Flow Problems*, pages 138–156, 1979. in Russian.
- 11 Alexander V. Karzanov. Minimum cost multiflows in undirected networks. *Mathematical Programming*, 66(3):313–325, 1994.
- 12 László Lovász. On some connectivity properties of Eulerian graphs. *Acta Mathematica Hungarica*, 28(1):129–138, 1976.
- 13 Alexander Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer, 2003.

On Matrix Powering in Low Dimensions

Esther Galby¹, Joël Ouaknine², and James Worrell²

¹ École Normale Supérieure de Rennes, France

² Department of Computer Science, Oxford University, UK

Abstract

We investigate the *Matrix Powering Positivity Problem*, PosMatPow: given an $m \times m$ square integer matrix M , a linear function $f : \mathbb{Z}^{m \times m} \rightarrow \mathbb{Z}$ with integer coefficients, and a positive integer n (encoded in binary), determine whether $f(M^n) \geq 0$. We show that for fixed dimensions m of 2 and 3, this problem is decidable in polynomial time.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems

Keywords and phrases matrix powering, complexity, Baker’s theorem

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.329

1 Introduction

An important theme in theoretical computer science is the complexity of performing calculations on large (often exponential) but succinctly presented structures. Notable examples include the analysis of various problems on succinctly represented graphs [10, 20], as well as the study of PosSLP [2], the problem of determining whether an arithmetic circuit, with addition, multiplication, and subtraction gates, evaluates to a positive integer. Allender *et al.* show that a substantial fragment of modern numerical analysis reduces in polynomial time to PosSLP, as do several other well-known questions such as the Sum-of-Square-Roots Problem, which itself is instrumental, among others, in solving the Euclidean Travelling Salesman Problem. Very recently, an interesting ‘challenge’ (spiritually attributed to Dyson) was proposed by Lipton: find an efficient algorithm that, given an integer n , determines whether the reversal of 2^n as a decimal number is a power of 5 [13].¹

In all the above examples, the central issue is that the objects in question, while succinctly presented, are fundamentally of exponential size. In the case of PosSLP, for instance, it is trivial to construct an arithmetic circuit whose integer output is doubly exponential in the size of the circuit, i.e., requiring an exponential number of bits. In light of this observation, one might conjecture that the existence of polynomial-time algorithms for performing non-trivial calculations on such entities is generally doomed.

Perhaps surprisingly, polynomial-time algorithms *can* occasionally be found. For example, by exploiting deep results in analytic number theory, Hirvensalo *et al.* showed in [12] that the most significant digit in base 3 of expressions such as 2^n and the n th Fibonacci number could be computed in time polynomial in the size of n , something which at first sight is far from obvious.

In this paper, we are concerned with the *Matrix Powering Positivity Problem*, PosMatPow: given an $m \times m$ square integer matrix M , a linear function $f : \mathbb{Z}^{m \times m} \rightarrow \mathbb{Z}$ with integer coefficients, and a positive integer n , determine whether $f(M^n) \geq 0$. Note that

¹ Here by “efficient” one requires a running time polynomial in the size, or bit length, of the representation of n .



in general, the entries of M^n have exponentially many bits (in the size of n), even if M is encoded in unary, so a naive calculation based (for example) on iterated squaring would necessarily require exponential time.²

Problems involving powers of matrices appear in a wide range of contexts. For ‘small’ (i.e., unary-encoded) powers, the complexity of powering has been thoroughly investigated in [15], and shown to lie in TC^0 for any fixed dimension. The complexity of **PosMatPow** (for ‘large’—i.e., binary-encoded—powers) is instrumental in determining the overall complexity of the main algorithms presented in [18, 17] to decide positivity of linear recurrence sequences of low order. Allender *et al.* study the closely related problem of **BitMatPow** in [1] in which one seeks to determine the value of a specified bit in a large power of a given matrix. Already in dimension 2—and thus *a fortiori* in higher dimensions as well—Allender *et al.* provide some evidence that **BitMatPow** cannot be solved in polynomial time, although it is known to lie in the Counting Hierarchy CH.

Since the publication of Allender *et al.*’s seminal work [2], determining the complexity of **PosSLP** has become a problem of major importance; Etessami and Yannakakis, for example, show in [8] that the fundamental problem of finding mixed strategy profiles close to exact Nash equilibria in three-person games is **PosSLP**-hard. **PosSLP** lies in CH [2] but is not believed to belong to NP and much less to P. Unfortunately, no non-trivial lower bounds for it are known at present.

PosMatPow can be shown to reduce in polynomial time to both **PosSLP** and **BitMatPow** (increasing for the latter the dimension by 3). Thus in addition to upper complexity bounds, lower bounds for **PosMatPow** would be of significant interest.

The main result of this paper is that in dimensions 2 and 3, **PosMatPow** can be solved in polynomial time.³ This upper bound is achieved by attacking the problem via spectral techniques, and making use of sophisticated tools from algebraic number theory, transcendence theory, and numerical analysis. We leave as a challenging open problem the complexity of **PosMatPow** in higher dimensions.

2 Preliminaries

We review some of the mathematical apparatus used throughout this paper. Since our approach is predicated on spectral techniques, the efficient manipulation of algebraic numbers is of central importance. The reader may however wish to skip this section on a first reading and proceed directly to Sections 3 and 4 in which the main algorithms are presented.

2.1 Algebraic Numbers and Baker’s Theorem

For $p \in \mathbb{Z}[x]$ a univariate polynomial with integer coefficients, let us denote by $\|p\|$ the bit length of its representation as a list of coefficients encoded in binary. Note that the degree of p is at most $\|p\|$, and the *height* of p —i.e., the maximum magnitude of its coefficients—is at most $2^{\|p\|}$.

A complex number α is *algebraic* if it is the root of a univariate polynomial with integer coefficients. The *defining polynomial* of α , denoted p_α , is the unique polynomial of least

² Note that we are working in the standard bit-model of complexity theory, rather than the unit-cost arithmetic model in which **PosMatPow** (and **PosSLP**) would trivially be in polynomial time.

³ In dimension 3, this result requires that the base matrix M be encoded in unary. The exponent n and linear function f are, however, always encoded in binary.

degree, and whose coefficients do not have common factors, which vanishes at α . The *degree* and *height* of α are respectively those of p_α .

A standard representation⁴ for algebraic numbers is to encode α as a tuple comprising its defining polynomial together with rational approximations of its real and imaginary parts of sufficient precision to distinguish α from the other roots of p_α . More precisely, α can be represented by $(p_\alpha, a, b, r) \in \mathbb{Z}[x] \times \mathbb{Q}^3$ provided that α is the unique root of p_α inside the circle in \mathbb{C} of radius r centred at $a + bi$. A separation bound due to Mignotte [16] asserts that for roots $\alpha \neq \beta$ of a polynomial $p \in \mathbb{Z}[x]$, we have

$$|\alpha - \beta| > \frac{\sqrt{6}}{d^{(d+1)/2} H^{d-1}}, \quad (1)$$

where d and H are respectively the degree and height of p . Thus if r is required to be less than a quarter of the root-separation bound, the representation is well-defined and allows for equality checking. Given a polynomial $p \in \mathbb{Z}[x]$, it is well-known how to compute standard representations of each of its roots in time polynomial in $\|p\|$ [19, 7, 4]. Thus given α an algebraic number for which we have (or wish to compute) a standard representation, we write $\|\alpha\|$ to denote the bit length of this representation. From now on, when referring to computations on algebraic numbers, we always implicitly refer to their standard representations.

Given algebraic numbers α and β , one can test whether $\alpha = \beta$ as well as membership in \mathbb{R} in polynomial time. One can also compute $\alpha + \beta$, $\alpha\beta$, $1/\alpha$ (for non-zero α), $\bar{\alpha}$, $|\alpha|$, $\operatorname{Re}(\alpha)$, and $\operatorname{Im}(\alpha)$, all of which are algebraic, in polynomial time. Moreover, if $\alpha \in \mathbb{R}$, deciding whether $\alpha > 0$ can also be done in polynomial time. Efficient algorithms for all these tasks can be found in [7, 4].

We will also need the following result.

► **Proposition 1.** Given algebraic numbers α and β , together with an integer $n \geq 0$, one can decide whether $\alpha^n = \beta$ in time polynomial in both $\|\alpha\| + \|\beta\|$ and $\|n\| = \lceil \log_2 n \rceil$.

Proposition 1 can be proved directly using elementary algebraic number theory. Alternatively, it is an immediate consequence of the following lemma:

► **Lemma 1.** Let α and β be non-zero complex algebraic numbers, and consider the free abelian group L under addition given by $L = \{(u, v) \in \mathbb{Z}^2 : \alpha^u \beta^v = 1\}$. L has a basis whose vectors are polynomially bounded in $\|\alpha\| + \|\beta\|$. Moreover, such a basis can be computed in time polynomial in $\|\alpha\| + \|\beta\|$.

Note in the above that the bound is on the *magnitude* of the vectors in the basis (rather than the bit length of their representation), which follows from a deep result of Masser [14]. For a proof of Lem. 1, see also [11, 6].

Proposition 1 now easily follows: given α and β , compute a basis B for the corresponding free abelian group L , and decide whether $(n, -1) \in L = \operatorname{span}(B)$, which can be done in polynomial time. For example, if L has rank 2, i.e., $B = \{(u_1, v_1), (u_2, v_2)\}$ for some integers u_1, v_1, u_2 , and v_2 , the problem is equivalent to determining whether there exist integers x and y such that $x(u_1, v_1) + y(u_2, v_2) = (n, -1)$. Since the u_i 's and v_i 's have magnitude polynomial in $\|\alpha\| + \|\beta\|$, the size of this problem instance is logarithmic (hence *a fortiori* polynomial) in $\|\alpha\| + \|\beta\|$ and polynomial in $\|n\|$. Since solving linear equations over the integers can be carried out in polynomial time, the desired result follows.

⁴ Note that this representation is not unique.

We also record the following bounds, which are immediately derived from classical analytic results on polynomials (see, e.g., [21]). For α a non-zero algebraic number of height H , we have

$$\frac{1}{H+1} < \alpha < H+1. \tag{2}$$

If E and F are two fields such that $F \subseteq E$, we say that E is an *extension* of F and the **degree of E over F** , denoted $[E : F]$, is defined to be the dimension of E considered as a vector space over F . The degree is multiplicative: if E is an extension of F and F is itself an extension of L , then E is an extension of L of degree $[E : L] = [E : F][F : L]$.

A **number field** is an extension of \mathbb{Q} of finite degree. In particular, given any algebraic numbers $\alpha_1, \dots, \alpha_k$, $\mathbb{Q}(\alpha_1, \dots, \alpha_k)$ is the number field comprising all complex numbers that are equal to some polynomial in $\alpha_1, \dots, \alpha_k$ with rational coefficients.

Let $p \in \mathbb{Z}[x]$ be a quadratic polynomial with roots α and β . Then $[\mathbb{Q}(\alpha) : \mathbb{Q}] \leq 2$ and $\beta \in \mathbb{Q}(\alpha)$. On the other hand, if p is a cubic polynomial with roots α, β , and γ , then $[\mathbb{Q}(\alpha, \beta) : \mathbb{Q}] \leq 6$ and $\gamma \in \mathbb{Q}(\alpha, \beta)$. For K a number field with $\lambda, \bar{\lambda} \in K$, we have $[K(|\lambda|) : K] \leq 2$ since $|\lambda|^2 = \lambda\bar{\lambda}$. And also $[K(\nu) : \mathbb{Q}] \leq [K : \mathbb{Q}][\mathbb{Q}(\nu) : \mathbb{Q}]$ for any number field K and algebraic number ν .

Finally, we give a version of Baker’s deep theorem on linear forms in logarithms. The particular statement we have chosen is a sharp formulation due to Baker and Wüstholz [3]. In what follows, \log refers to the principal value of the complex logarithm function given by $\log z = \log |z| + i \arg z$, where $-\pi < \arg z \leq \pi$.

► **Theorem 2** (Baker and Wüstholz). *Let $\alpha_1, \dots, \alpha_m \in \mathbb{C}$ be algebraic numbers different from 0 or 1, and let $b_1, \dots, b_m \in \mathbb{Z}$ be integers. Write*

$$\Lambda = b_1 \log \alpha_1 + \dots + b_m \log \alpha_m.$$

Let $A_1, \dots, A_m, B \geq e$ be real numbers such that, for each $j \in \{1, \dots, m\}$, A_j is an upper bound for the height of α_j , and B is an upper bound for $|b_j|$. Let d be the degree of the number field $\mathbb{Q}(\alpha_1, \dots, \alpha_m)$ over \mathbb{Q} . Then if $\Lambda \neq 0$,

$$\log |\Lambda| > -(16md)^{2(m+2)} \log A_1 \dots \log A_m \log B.$$

2.2 Matrix Powers and Linear Recurrence Sequences

We recall some basic facts about linear algebra and linear recurrence sequences. An excellent reference on the latter is [9].

Let $M \in \mathbb{Z}^{m \times m}$ be a square integer matrix of dimension m . In this paper, we work with a *binary* encoding of M in two dimensions, and with a *unary* encoding of M in three dimensions. Both encodings are denoted $\|M\|$, relying on context for disambiguation.

In two dimensions (with binary encoding), we note that eigenvalues of M have degree at most 2 and height at most $2^{2\|M\|}$.

In three dimensions (with unary encoding), eigenvalues of M have degree at most 3 and height at most $\|M\|^3$.

Let $f : \mathbb{Z}^{m \times m} \rightarrow \mathbb{Z}$ be a linear function with integer coefficients: $f(x_1, \dots, x_{m^2}) = b_1 x_1 + \dots + b_{m^2} x_{m^2}$ for integers b_1, \dots, b_{m^2} . Since m -dimensional square integer matrices can be viewed as m^2 -tuples of integers, we shall assume a fixed order for entries and freely apply such functions to square matrices. We always encode f as a list of its coefficients in binary, and denote the size of this encoding by $\|f\|$.

Let $p(x) = x^m - a_1x^{m-1} - \dots - a_mx$ be the characteristic polynomial of M . For any $k \geq 0$, let $u_k = f(M^k)$. Then the sequence $\langle u_k \rangle_{k=0}^\infty$ is an integer linear recurrence sequence (LRS) obeying the recurrence

$$u_{k+m} = a_1u_{k+m-1} + \dots + a_mu_k. \quad (3)$$

Indeed, since $M^m - a_1M^{m-1} - \dots - a_mI = \mathbf{0}$ by the Cayley-Hamilton theorem, multiplying this equation by M^k , applying f on both sides and invoking linearity yields Eq. (3).

The characteristic polynomial of this LRS is p , hence the characteristic roots are the eigenvalues of M . Let us write

$$\text{spec}(M) = \{\rho_1, \dots, \rho_\ell, \lambda_1, \overline{\lambda_1}, \dots, \lambda_p, \overline{\lambda_p}\},$$

where each $\rho_i \in \mathbb{R}$ and each $\lambda_j \in \mathbb{C} \setminus \mathbb{R}$. There are now univariate polynomials A_1, \dots, A_ℓ and C_1, \dots, C_p such that, for all $n \geq 0$,

$$u_k = \sum_{i=1}^{\ell} A_i(k)\rho_i^k + \sum_{j=1}^p \left(C_j(k)\lambda_j^k + \overline{C_j(k)}\overline{\lambda_j}^k \right). \quad (4)$$

This expression is referred to as the *exponential polynomial* solution of $\langle u_k \rangle_{k=0}^\infty$. The polynomials A_i have real algebraic coefficients and the polynomials C_j have complex algebraic coefficients. The degree of each of these polynomials is at most one less than the multiplicity of the corresponding eigenvalue; thus in particular, these polynomials are identically constant when M has no repeated eigenvalue. For fixed m , all coefficients appearing in these polynomials can be computed in time polynomial in $\|\langle f, M \rangle\|$ (whether M is encoded in binary or unary),⁵ since they can be obtained by solving a system of linear equations involving the m constants u_0, \dots, u_{m-1} . As a result, these coefficients all belong to $\mathbb{Q}(\text{spec}(M))$, and their height (*qua* algebraic numbers) is bounded above by $2^{\mathcal{O}(\|\langle f, M \rangle\|)}$ (again regardless of whether M is encoded in binary or unary).

2.3 Approximation Algorithms for Transcendental Functions

Finally, we recall some classical numerical algorithms which are later invoked to efficiently compute approximations of transcendental functions applied to algebraic numbers.

Given a real number t and a positive integer m , we say that $q \in \mathbb{Q}$ is an *m -bit approximation of t* if $|t - q| < 2^{-m}$. We also sometimes refer to the calculation of such a q as “computing m bits of t ”, even though strictly speaking this form of words is not perfectly accurate.

► **Proposition 2.**

1. There exists an algorithm which takes as input a real algebraic number $\rho > 0$, together with a positive integer m , and returns an m -bit approximation of $\log \rho$ in time polynomial in both $\|\rho\|$ and m .
2. There exists an algorithm which takes as input two non-zero real algebraic numbers a and b , together with a positive integer m , and returns an m -bit approximation of $\arctan b/a$ in time polynomial in both $\|a\| + \|b\|$ and m .
3. There exists an algorithm which takes as input a positive integer m and returns an m -bit approximation of π in time polynomial in m .

⁵ We write $\|\langle f, M \rangle\|$ to denote the size of the joint encoding of f and M .

Proposition 2 follows from classical approximation results for transcendental functions due to Brent [5], together with the fact that we can compute approximations of algebraic numbers with polynomially many bits in polynomial time (see, e.g., [19]). Below we sketch the process for (1) of Prop. 2, relying on the following result.

► **Theorem 3** (Brent). *For any fixed real numbers $0 < a < b$, there exists an algorithm which, given an integer $p \geq 0$, evaluates $\log x$ in time $\mathcal{O}(p \log^2 p \log \log p)$, with relative error at most $\mathcal{O}(2^{-p})$, uniformly for all $x \in [a, b]$.*

Let ρ and m be as in Prop. 2 (1), and denote the height of ρ by H , recalling that $H \leq 2^{|\rho|}$. By Eq. (2), we have $\rho > 1/(H + 1)$, and for simplicity assume that $\rho < 1$; the alternative can be handled in a similar manner as what follows. We now aim to select a positive integer k with certain properties, to be listed in the remainder of this proof; we will then choose a specific value for k later on in such a way as to discharge all our assumptions.

The first requirement is that k be at most polynomial in $|\rho|$ and m . Next, compute $r \in \mathbb{Q}$ with $1/(2H) < r \leq \rho$ such that $\rho - r < 2^{-k}$; this can be achieved in time polynomial in k by computing polynomially many bits of ρ . Fix the interval $[a, b] = [2, 4]$ in Thm. 3 and find $j \in \mathbb{N}$ such that $2^j r \in [2, 4]$. Thanks to our lower bound on r such j is at most polynomial in $|\rho|$ and can be obtained in polynomial time.

We now invoke Thm. 3 to compute $u \in \mathbb{Q}$ such that $\frac{|u - \log 2^j r|}{\log 2^j r} < \frac{1}{2^{m+3}}$ in time polynomial in m . Since $\log 2^j r \leq \log 4 < 2$, we have $|v - \log r| < 2^{-m-2}$, where $v = u - j \log 2$. By invoking Thm. 3 once more, we can compute $v' \in \mathbb{Q}$ in time polynomial in m such that $|v' - v| < 2^{-m-2}$, whence $|v' - \log r| < 2^{-m-1}$.

Since the derivative of $\log x$ at point r is r^{-1} , we conclude that $|\log \rho - \log r| < r^{-1} 2^{-k}$. If we make the additional requirement on k that $r^{-1} 2^{-k} < 2^{-m-1}$, we can combine with our previous inequality to obtain $|v' - \log \rho| < 2^{-m}$, yielding an m -bit approximation of $\log \rho$ as required.

Finally, it remains to show that $k \in \mathbb{N}$ can be chosen so as to meet our various assumptions, a straightforward task which we leave to the reader.

3 The Two-Dimensional Matrix Powering Positivity Problem

The main result of this section is the following:

► **Theorem 4.** *In two dimensions, PosMatPow (with full binary encoding) is decidable in polynomial time.*

Proof. Consider an instance of the two-dimensional Matrix Powering Positivity Problem, comprising a linear function $f : \mathbb{Z}^4 \rightarrow \mathbb{Z}$, a 2×2 integer matrix M , and an integer $n \geq 0$. Assume that all this data is encoded in binary and denote by $|\langle f, M, n \rangle|$ the size of the instance. We wish to decide in polynomial time whether $f(M^n) \geq 0$. To this end, we consider the sequence $u_k = f(M^k)$ and study the exponential polynomial solution (Eq. 4) in which two cases arise: either (i) both eigenvalues of M are real (including the possibility of a single repeated real eigenvalue), or (ii) both eigenvalues are complex conjugates. In the latter, it is worth pointing out that the sign of the sequence will forever fluctuate.

In Case (i), let $\rho_1, \rho_2 \in \mathbb{R}$ be the eigenvalues of M . We distinguish two subcases depending on the multiplicity of ρ_1 . If ρ_1 is repeated (i.e., $\rho_1 = \rho_2$), then for all $k \geq 0$,

$$u_k = (ak + b)\rho_1^k$$

where a and b are two real algebraic constants; recall moreover from Sec. 2.2 that ρ_1 , a , and b can all be computed in polynomial time, and have representations of size linear in $|\langle f, M \rangle|$.

Assuming that $a \neq 0$ (the treatment being straightforward otherwise), it is easy to see that u_k has the same sign as

$$\begin{aligned} & b\rho_1^k, & \text{when } k < -b/a \\ & a\rho_1^k, & \text{when } k > -b/a. \end{aligned}$$

Note that for $-b/a \in \mathbb{N}$, $u_{-b/a} = 0$. It therefore remains to compare n to $-b/a$. Since arithmetic and inequality testing on algebraic numbers can be performed in polynomial time, the desired result follows.

Now assume that $\rho_1 \neq \rho_2$. Then we can compute two real algebraic constants a and b such that for all $k \geq 0$,

$$u_k = a\rho_1^k + b\rho_2^k.$$

If any of ρ_1 , ρ_2 , a , or b is zero, the solution is immediate. Thus assume otherwise and consider the sequence

$$\frac{u_k}{\rho_1^k} = a + b \left(\frac{\rho_2}{\rho_1} \right)^k.$$

We check whether u_n is zero for our given exponent n , or equivalently whether $(\rho_2/\rho_1)^n = -a/b$; by Prop. 1, this can be done in polynomial time. Otherwise, we have $u_n/\rho_1^n > 0$ iff

$$b \left(\frac{\rho_2}{\rho_1} \right)^n > -a. \tag{5}$$

Assume without loss of generality that the expressions on both sides of the inequality are positive (something which is readily checked). Then Eq. (5) holds iff

$$\log |b| + n \log |\rho_2| - n \log |\rho_1| > \log |a|.$$

In other words, the sign of the expression

$$\Lambda = \log |b| + n \log |\rho_2| - n \log |\rho_1| - \log |a|$$

determines that of u_n (modulo the sign of ρ_1^n). Note that $\rho_1, \rho_2, a, b \in \mathbb{Q}(\rho_1)$, so that the number field $\mathbb{Q}(|b|, |\rho_1|, |\rho_2|, |a|)$ has degree 2 over \mathbb{Q} . Moreover, we can easily compute an upper bound H on the heights of ρ_1 , ρ_2 , a , and b , such that $\log(H) = \mathcal{O}(|\langle f, M \rangle|)$. By Baker's theorem (Thm. 2), we then have

$$|\Lambda| > \exp \left(-(16 \cdot 4 \cdot 2)^{2(4+2)} (\log H)^4 \log n \right) = \frac{1}{n^{128^{12} (\log H)^4}} = \frac{1}{n^{|\langle f, M \rangle|^{\mathcal{O}(1)}}}.$$

Thus in order to determine the sign of Λ , it suffices to compute $|\langle f, M \rangle|^{\mathcal{O}(1)} \log_2 n = |\langle f, M, n \rangle|^{\mathcal{O}(1)}$ bits of Λ , i.e., a polynomial number of bits in the size of our problem instance $\langle f, M, n \rangle$. By Prop. 2, $|\langle f, M, n \rangle|^{\mathcal{O}(1)}$ -bit approximations of $\log |b|$, $\log |\rho_2|$, $\log |\rho_1|$, and $\log |a|$ can be obtained in polynomial time, whence the desired result follows.

We now turn to Case (ii), in which M has two complex eigenvalues λ and $\bar{\lambda}$. We have, for all $k \geq 0$, $u_k = c\lambda^k + \bar{c}\bar{\lambda}^k$, where c is a complex algebraic constant. Equivalently, letting $\theta = \arg \lambda$ and $\varphi = \arg c$, we can write

$$u_k = |c||\lambda|^k \cos(k\theta + \varphi). \tag{6}$$

Note that $\cos(n\theta + \varphi) = 0$ iff $(e^{i\theta})^n = e^{i(-\varphi \pm \pi/2)}$. Since $e^{i\theta}$ and $e^{i\varphi}$ are algebraic numbers with size linear in $\|\langle f, M \rangle\|$, by Prop. 1 the latter can be checked in time polynomial in $\|\langle f, M, n \rangle\|$.

Let us therefore assume that $u_n \neq 0$ for our given exponent n . We aim to bound (in absolute value) the expression $n\theta + \varphi$ away from $\pm\pi/2$ (modulo 2π). To this end, write

$$\Gamma = \arg e^{i(n\theta + \varphi)} = n\theta + \varphi - 2m\pi,$$

where m is the unique integer such that $-\pi < \Gamma \leq \pi$. The delicate situation is now if Γ is ‘close’ to $\pm\pi/2$. If that is not the case (for instance, if $\|\Gamma - \pi/2\| > 0.1$, say), then one can readily compute the sign of $\cos(n\theta + \varphi)$, and therefore that of u_n , in polynomial time. On the other hand, if Γ is close to $\pm\pi/2$ (for instance, if $\|\Gamma - \pi/2\| < 0.5$, say), then one can readily determine the sign of Γ . Assume that we are in the latter situation, and without loss of generality suppose that $\Gamma > 0$. Write

$$\Lambda = \frac{\pi}{2} - \Gamma = \frac{1}{i} \left(n \log \frac{\lambda}{|\lambda|} + \log \frac{c}{|c|} + (1 - 4m) \log i \right),$$

and let H be an upper bound for the heights of $\lambda/|\lambda|$ and $c/|c|$. Note that the degree of $\mathbb{Q}(\lambda, |\lambda|, c, |c|, i)$ over \mathbb{Q} is at most 16. Since $|1 - 4m| \leq 2n + 1$, it follows from Baker’s theorem that

$$|\Lambda| > \exp(-768^{10}(\log H)^2 \log(2n + 1)) = \frac{1}{(2n + 1)^{768^{10}(\log H)^2}} = \frac{1}{n^{\|\langle f, M \rangle\|^{\mathcal{O}(1)}}}, \tag{7}$$

since $\log(H) = \mathcal{O}(\|\langle f, M \rangle\|)$.

Thanks to Eq. (6) and the definition of Γ , the quantities u_n , $\cos(n\theta + \varphi)$, and $\cos(\Gamma)$ have the same sign. Since we are assuming that $0 < \Gamma \leq \pi$, it follows that $\cos(\Gamma)$ and Λ have the same sign as well. By Eq. (7), the sign of Λ (and therefore that of u_n) can be determined by computing $\|\langle f, M \rangle\|^{\mathcal{O}(1)} \log_2 n = \|\langle f, M, n \rangle\|^{\mathcal{O}(1)}$ bits of Λ , which can be done in polynomial time thanks to Prop. 2, by noting that for any algebraic $\alpha \in \mathbb{C} \setminus \{i, -i\}$ of modulus 1, $\log \alpha$ can be obtained by computing $\arctan(\text{Im}(\alpha)/\text{Re}(\alpha))$.

This concludes the proof of Thm. 4. ◀

4 The Three-Dimensional Matrix Powering Positivity Problem

We now move to three dimensions. We are given a linear function $f : \mathbb{Z}^9 \rightarrow \mathbb{Z}$, a 3×3 integer matrix M , and an integer $n \geq 0$. We assume that the base matrix M is encoded in unary, whereas the function f and exponent n are encoded in binary. Note in particular that this includes the important special case in which the base data $\langle f, M \rangle$ is fixed.

Our main result is as follows:

► **Theorem 5.** *In three dimensions, PosMatPow (with unary encoding of the base matrix and binary encoding of the linear function and of the exponent) is decidable in polynomial time.*

Proof. Let $\langle f, M, n \rangle$ be as above. We seek to determine whether $f(M^n) \geq 0$.

As before, write $u_k = f(M^k)$. Our strategy is to exhibit a bound N , of magnitude polynomial in $\|\langle f, M \rangle\|$, such that the sign of u_k is easily determined for $k \geq N$. Note on the other hand that, if $n < N$, then one can simply compute u_n outright in polynomial time.

We split our analysis into two main cases: (i) either M only has real eigenvalues, or (ii) two of M ’s eigenvalues are complex conjugates.

In Case (i), let ρ_1 be a real dominant eigenvalue of M . We focus on the hardest instance, in which ρ_1 has multiplicity 1; the other two alternatives are considerably simpler and can be handled similarly.

Let ρ_2 be a second real eigenvalue of M . Here, the critical case is when ρ_2 is repeated; the (easier) alternative can again be handled in similar fashion, and is therefore omitted.

We can thus write

$$u_k = a\rho_1^k + (bk + c)\rho_2^k, \tag{8}$$

where we assume that $a \neq 0$. Observe that since M is encoded in unary, one has an $\|M\|^3$ upper bound for the maximum height H of the eigenvalues ρ_1 and ρ_2 . Likewise, the real algebraic numbers $1/a$, b , and c all have representations of size linear in $\|\langle f, M \rangle\|$, and therefore have magnitude at most $2^{\mathcal{O}(\|\langle f, M \rangle\|)}$.

Note that if ρ_2 and ρ_1 have the same modulus, then $\rho_2 = -\rho_1$ and the treatment is straightforward; we therefore assume that $|\rho_2| < |\rho_1|$. Since both ρ_1 and ρ_2 have degree at most 3, Mignotte’s root-separation bound (Eq. 1) entails that $|\rho_2| = |\rho_1| - \delta$, where $\delta = \Omega(H^{-2})$.

Let $\gamma = \rho_2/\rho_1$. Equation (8) can be rewritten as

$$\frac{u_k}{\rho_1^k} = a + (bk + c)\gamma^k. \tag{9}$$

We also have

$$|\gamma| = \frac{|\rho_2|}{|\rho_1|} = \frac{|\rho_1| - \delta}{|\rho_1|} = 1 - \frac{\delta}{|\rho_1|}.$$

By Eq. (2), $|\rho_1| \leq H + 1$, from which we immediately conclude that $|\gamma| = 1 - \varepsilon$, where $1/\varepsilon = \mathcal{O}(H^3) = \mathcal{O}(\|M\|^9)$. We now aim to establish a bound N of magnitude polynomial in $\|\langle f, M \rangle\|$ such that, for $k \geq N$,

$$|a| > |(bk + c)\gamma^k|. \tag{10}$$

By Eq. (9), the sign of u_k is then automatically obtained for any k beyond N .

From the inequality $\log(1 - \varepsilon) < -\varepsilon$, we have $|\gamma| = 1 - \varepsilon < e^{-\varepsilon}$. In order for Eq. (10) to hold, it therefore suffices to have $|a| > |bk + c|e^{-k\varepsilon}$. Letting $x = k\varepsilon$, this translates to $e^x > |(bx/\varepsilon + c)/a|$. Thanks to our bounds on $|1/a|$, $|b|$, $|c|$, and $1/\varepsilon$, we can find $B = 2^{\mathcal{O}(\|\langle f, M \rangle\|)}$ such that $|(bx/\varepsilon + c)/a| \leq Bx$ for all $x \geq 1$. Now clearly the inequality $e^x > Bx$ holds provided that $x \geq 2 \log B$, or equivalently that $k \geq (2 \log B)/\varepsilon$. Letting $N = \lceil (2 \log B)/\varepsilon \rceil = \|\langle f, M \rangle\|^{\mathcal{O}(1)}$ and putting everything together, we see that Eq. (10) holds for $k \geq N$, as required.

We now turn to Case (ii), in which M has two complex conjugate eigenvalues λ and $\bar{\lambda}$, and one real eigenvalue ρ . For all $k \geq 0$, we have

$$u_k = a\rho^k + c\lambda^k + \bar{c}\bar{\lambda}^k, \tag{11}$$

for some algebraic constants $a \in \mathbb{R}$ and $c \in \mathbb{C}$. As before, ρ and λ have height bounded by $\|M\|^3$, whereas a and c have height bounded by $2^{\mathcal{O}(\|\langle f, M \rangle\|)}$. Moreover λ has degree at most 3 and $\rho, a, c \in \mathbb{Q}(\lambda, \bar{\lambda})$.

If $|\rho| > |\lambda|$, we can proceed straightforwardly through a growth argument akin to that invoked in Case (i) above, whereas if $|\rho| = |\lambda|$, the situation is very similar to Case (ii) of the two-dimensional instance of the problem, handled in the previous section, and can be dealt with in like fashion. We therefore focus on the situation in which $|\rho| < |\lambda|$.

Let $\theta = \arg \lambda$ and $\varphi = \arg c$. Equation (11) then becomes

$$u_k = a\rho^k + |c||\lambda|^k \cos(k\theta + \varphi).$$

Writing $\gamma = \rho/|\lambda|$, we have

$$\frac{u_k}{|c||\lambda|^k} = \frac{a}{|c|}\gamma^k + \cos(k\theta + \varphi), \tag{12}$$

and as before, $|\gamma| = 1 - \varepsilon$, where $1/\varepsilon = \mathcal{O}(\|M\|^9)$.

As in the two-dimensional case, we can check in polynomial time whether $\cos(n\theta + \varphi) = 0$, in which case the sign of u_n is readily determined. Otherwise, write $\Gamma = n\theta + \varphi - 2m\pi$, with $m \in \mathbb{Z}$ such that $-\pi < \Gamma \leq \pi$, and as before, without loss of generality, assume that Γ is ‘close’ to $\pi/2$, the other cases being either straightforward or handled similarly. Write $\Lambda = \pi/2 - \Gamma$, and note that

$$\cos(n\theta + \varphi) = \cos \Gamma = \sin \Lambda \quad \text{and} \quad |\sin \Lambda| > \frac{|\Lambda|}{2}. \tag{13}$$

We have

$$\Lambda = \frac{1}{i} \left(n \log \frac{\lambda}{|\lambda|} + \log \frac{c}{|c|} + (1 - 4m) \log i \right).$$

Since λ has degree at most 3, the degree of $\mathbb{Q}(\lambda, \bar{\lambda}, |\lambda|, c, |c|, i)$ over \mathbb{Q} is at most 48. Moreover, we can bound the height of $\lambda/|\lambda|$ and $c/|c|$ by some H with $\log H = \mathcal{O}(\|\langle f, M \rangle\|)$. Finally, we note that $|1 - 4m| \leq 2n + 1$. Applying Baker’s theorem, we get

$$|\Lambda| > \exp(-2304^{10}(\log H)^2 \log(2n + 1)) = \frac{1}{(2n + 1)^{2304^{10}(\log H)^2}} = \frac{1}{n^{\|\langle f, M \rangle\|^{\mathcal{O}(1)}}}. \tag{14}$$

It follows from Eqs. (13) and (14) that there is an absolute constant $T \in \mathbb{N}$ such that

$$|\cos(n\theta + \varphi)| > \frac{1}{n^{\|\langle f, M \rangle\|^T}}. \tag{15}$$

We now aim to establish a bound N of magnitude polynomial in $\|\langle f, M \rangle\|$ such that, if $n \geq N$, then

$$\left| \frac{a}{|c|}\gamma^n \right| < |\cos(n\theta + \varphi)|. \tag{16}$$

Thanks to Eq. (12), in that case the sign of u_n is the same as that of $\cos(n\theta + \varphi)$, and in turn the latter can be determined in polynomial time following the procedure outlined in Case (ii) of the two-dimensional instance of the problem, thanks to Eq. (14). On the other hand, if $n < N$, we simply note that u_n can then be computed outright in polynomial time.

By Eq. (15), and recalling that $|\gamma| = 1 - \varepsilon$, it is sufficient for Eq. (16) to hold to have

$$\frac{|a|}{|c|}(1 - \varepsilon)^n < \frac{1}{n^{\|\langle f, M \rangle\|^T}},$$

or equivalently (noting that $\log(1 - \varepsilon) < 0$),

$$n > -\frac{\|\langle f, M \rangle\|^T}{\log(1 - \varepsilon)} \log n - \frac{\log(|a|/|c|)}{\log(1 - \varepsilon)}.$$

Multiplying the above equation by 2 and writing $2n = n + n$, it is then sufficient for both

$$n > -2 \frac{\|\langle f, M \rangle\|^T}{\log(1 - \varepsilon)} \log n \quad \text{and} \quad (17)$$

$$n > -2 \frac{\log(|a|/|c|)}{\log(1 - \varepsilon)} \quad (18)$$

to hold.

For any $Q \geq 1$, one has $Q > 2 \log Q$, thus $Q^2 > Q \log(Q^2)$. In other words, $x > Q \log x$ for $x = Q^2$. But by comparing derivatives at the point $x = Q^2$, we see that the inequality $x > Q \log x$ holds for all $x \geq Q^2$. Writing

$$Q = -2 \frac{\|\langle f, M \rangle\|^T}{\log(1 - \varepsilon)},$$

we see that Eq. (17) holds provided $n \geq Q^2$. Since $1/\varepsilon = \mathcal{O}(\|M\|^9)$ and $|\log(1 - \varepsilon)| > \varepsilon$, we immediately have $Q^2 = \|\langle f, M \rangle\|^{\mathcal{O}(1)}$.

Next, let H be the maximum of the heights of a and c , noting that $\log H = \mathcal{O}(\|\langle f, M \rangle\|)$. By Eq. (2), $|a| < H + 1$ and $|c| > 1/(H + 1)$, whence

$$Q' = \left| -2 \frac{\log(|a|/|c|)}{\log(1 - \varepsilon)} \right| = \|\langle f, M \rangle\|^{\mathcal{O}(1)}.$$

It follows that by letting $N = \max\{\lceil Q^2 \rceil, \lceil Q' \rceil\}$, both Eqs. (17) and (18) hold provided that $n \geq N$, as required.

This concludes the proof of Thm. 5. ◀

5 Concluding Remarks

It is worth noting that our results can be extended in a fairly minor way, by considering matrices M and linear functions f with *rational* entries and coefficients: indeed, the rational formulation of PosMatPow reduces straightforwardly to its integer counterpart at the cost of a polynomial blowup in size.

Further extensions however appear elusive under the present framework. In the three-dimensional case, for instance, encoding the base matrix in binary would not yield a sufficiently large spectral gap (difference in magnitude between the largest and second-largest eigenvalues) for our present approach to go through; more specifically, the value of N required so that Eq. (10) hold would then potentially be exponential, thereby not leading to a polynomial-time algorithm. In four dimensions or higher, the situation worsens: we do not know how to produce a polynomial-time algorithm even for *fixed* base data M and f . A critical case is encountered when there are four or more dominant complex eigenvalues, ostensibly precluding the use of Baker's theorem.

The reader will have noticed the presence of various 'galactic' constants appearing in the analysis of our algorithms, and perhaps conclude that the approach we have laid out is unlikely to be feasible in practice. It is worth noting, however, that our analysis merely serves to establish (large) polynomial-time upper bounds, without any expectation that such bounds need be tight. On the contrary, we conjecture that the proposed approach, under careful implementation and engineering, would prove quite efficient in practice. Substantiating this empirically might however be expected to require non-trivial efforts.

References

- 1 E. Allender, N. Balaji, and S. Datta. Low-depth uniform threshold circuits and the bit-complexity of straight-line programs. *Elec. Coll. on Comput. Complex.*, 177(1), 2013.
- 2 E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5), 2009.
- 3 A. Baker and G. Wüstholz. Logarithmic forms and group varieties. *Jour. Reine Angew. Math.*, 442, 1993.
- 4 S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*. Springer, 2nd edition, 2006.
- 5 R. P. Brent. Fast multiple-precision evaluation of elementary functions. *J. ACM*, 23(2), 1976.
- 6 J.-Y. Cai, R. J. Lipton, and Y. Zalcstein. The complexity of the A B C problem. *SIAM J. Comput.*, 29(6), 2000.
- 7 H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag, 1993.
- 8 K. Etessami and M. Yannakakis. On the complexity of Nash equilibria and other fixed points (extended abstract). In *Proceedings of FOCS*. IEEE Computer Society, 2007.
- 9 G. Everest, A. van der Poorten, I. Shparlinski, and T. Ward. *Recurrence Sequences*. American Mathematical Society, 2003.
- 10 H. Galperin and A. Wigderson. Succinct representations of graphs. *Inf. Control*, 56(3), 1983.
- 11 G. Ge. *Algorithms Related to Multiplicative Representations of Algebraic Numbers*. PhD thesis, U.C. Berkeley, 1993.
- 12 M. Hirvensalo, J. Karhumäki, and A. Rabinovich. Computing partial information out of intractable: Powers of algebraic numbers as an example. *Jour. Number Theory*, 130, 2010.
- 13 R. J. Lipton. A challenge from Dyson. *Blog entry*, September 2014. <http://rjlipton.wordpress.com/2014/09/09/a-challenge-from-dyson/>.
- 14 D. W. Masser. Linear relations on algebraic groups. In *New Advances in Transcendence Theory*. Cambridge University Press, 1988.
- 15 C. Mereghetti and B. Palano. Threshold circuits for iterated matrix product and powering. *Theoret. Informatics Appl.*, 34(1), 2000.
- 16 M. Mignotte. Some useful bounds. In *Computer Algebra*, 1982.
- 17 J. Ouaknine and J. Worrell. On the positivity problem for simple linear recurrence sequences. In *Proceedings of ICALP*, number 8573 in Springer LNCS, 2014.
- 18 J. Ouaknine and J. Worrell. Positivity problems for low-order linear recurrence sequences. In *Proceedings of SODA*. SIAM, 2014.
- 19 V. Pan. Optimal and nearly optimal algorithms for approximating polynomial zeros. *Computers & Mathematics with Applications*, 31(12), 1996.
- 20 C. H. Papadimitriou and M. Yannakakis. A note on succinct representations of graphs. *Inf. Control*, 71(3), 1986.
- 21 Q. I. Rahman and G. Schmeisser. *Analytic Theory of Polynomials*. London Mathematical Society monographs. Oxford University Press, 2002.

The Complexity of Recognizing Unique Sink Orientations

Bernd Gärtner and Antonis Thomas

Department of Computer Science,
Institute of Theoretical Computer Science, ETH Zürich
8092 Zürich, Switzerland
{gaertner,athomas}@inf.ethz.ch

Abstract

Given a Boolean Circuit with n inputs and n outputs, we want to decide if it represents a Unique Sink Orientation (USO). USOs are useful combinatorial objects that serve as abstraction of many relevant optimization problems. We prove that recognizing a USO is **coNP**-complete. However, the situation appears to be more complicated for recognizing acyclic USOs. Firstly, we give a construction to prove that there exist cyclic USOs where the smallest cycle is of superpolynomial size. This implies that the straightforward representation of a cycle (i.e. by a list of vertices) does not make up for a **coNP** certificate. Inspired by this fact, we investigate the connection of recognizing an acyclic USO to **PSPACE** and we prove that the problem is **PSPACE**-complete.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics

Keywords and phrases complexity, recognizing, unique sink orientations, coNP, PSPACE

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.341

1 Introduction

Over the past 15 years, *unique sink orientations* (USO) have intensively been studied as simple and appealing combinatorial models for many concrete optimization problems. After their introduction by Stickney and Watson in the context of mathematical programming [16], USOs had been forgotten for more than 20 years, before Szabó and Welzl rediscovered them from a computational geometry angle [17]. Subsequently, the structural, algorithmic, and combinatorial aspects of USOs were investigated; new applications were found, in particular in the area of mathematical programming where the concept originally comes from. We refer the interested reader to Foniok et al. [4] and the references therein.

A USO is an orientation of the n -dimensional hypercube graph, with the property that every face of dimension $d \in \{0, 1, \dots, n\}$ induces a subgraph with a unique sink. In particular, there is a unique global sink, and the algorithmic problem is to find it.

In all known applications, the USO is given in *succinct representation*, i.e. there is an oracle that returns for a given vertex the orientations of the incident edges, and the question is how many oracle calls are necessary in order to find the global sink. The oracle itself can typically be implemented by a polynomial-time algorithm.

For a concrete such application, consider the problem of finding the smallest enclosing ball $B(P)$ of a set P of n affinely independent points in \mathbb{R}^{n-1} . Every subset $Q \subseteq P$ naturally corresponds to a vertex of the hypercube, and we have a directed edge from Q to $Q \cup \{p\}$, $p \notin Q$, if and only if p is outside of $b(Q)$, the smallest ball that has all points of Q on its boundary. This ball $b(Q)$ is easy to compute by solving a system of linear equations, so we have a polynomial-time oracle at our disposal. Moreover, the global sink S has the



© Bernd Gärtner and Antonis Thomas;
licensed under Creative Commons License CC-BY
32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).
Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 341–353
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



property that $b(S) = B(P)$, hence finding the global sink solves our geometric problem. The USO approach also works for the more general problem of finding the smallest enclosing ball of a set of *balls* [3], for general linear programs (LP) [8], and—this is the original application by Stickney and Watson— for P-matrix linear complementarity problems (PLCP) [16].

None of these problems have known *strongly* polynomial-time algorithms (i.e. polynomial in the real RAM model of computation); for PLCP, even weakly polynomial-time algorithms are not known. This would change if we could find the sink of an n -dimensional USO with a number of oracle calls that is polynomial in n .

Currently, we cannot, and it even seems somewhat stupid to further generalize problems that are already difficult. On the other hand, for some of the above concrete problems, the USO approach *does* yield the currently best known algorithms. Most notably, this is the case for a relevant class of LP in the RAM model [8], and for PLCP in general [17]. This clearly shows the usefulness of the USO abstraction, and the elegant combinatorial algorithms obtained in this abstraction [17].

Our contribution

In this paper, we study USO from a novel angle. While previous research mostly addresses the algorithmic problem of finding the global sink in a USO, we deal with the more fundamental problem of *recognizing* a USO, given in succinct representation. Concretely, we are interested in the computational complexity of deciding whether a succinct oracle indeed specifies a USO. In order to fit this problem into standard complexity theory, we assume that the oracle is implemented by a succinct Boolean circuit that in turn forms the input for the decision problem. Such a circuit has n input and n outputs, where n is the dimension of the USO; it is customary to use Boolean circuits in describing graphs succinctly (cf. [5]). By *succinct*, we mean that the size of the circuit is polynomial in n . We prove the following two main results.

1. It is **coNP**-complete to recognize USO, given in succinct Boolean circuit representation.
2. It is **PSPACE**-complete to recognize *acyclic* USO (AUSO), given in succinct Boolean circuit representation.

Here, an AUSO is a USO without any directed cycles. These results may come as a surprise, given that the *algorithmic* problem seems to be easier in the acyclic case: the best known (randomized) algorithm for finding the sink in an AUSO requires only a *subexponential* number of $\exp(2\sqrt{n})$ oracle calls [6]. For general USO, the best randomized bound is $O(1.438^n)$ [18].

Our results in particular show that there are simple certificates for non-USOs, but probably not for non-AUSOs. We explicitly show with a family of examples that the list of vertices on a directed cycle is not an efficient certificate for a non-AUSO, because such a list may have to be superpolynomially long. The construction works over an interesting and easy-to-analyze subclass of USOs (*flip matching orientations*) and is of independent interest.

The applications we advertise above reduce to digraphs that are guaranteed to be (A)USOs. Still, the complexity of recognizing an (A)USO from a succinct description is interesting from a theoretical viewpoint. In fact, similar theoretical results from the past include the recognizability of a P-Matrix, which is proved **coNP**-complete in [2]. Even though from applications (e.g. solving simple stochastic games [7]) we do get P-Matrices, the question of recognizability is still relevant.

The study of computational problems on graphs that are represented in an exponentially succinct way, through Boolean circuits, has been initiated by Galperin and Wigderson [5]. They proved that a number of trivial graph properties become **NP**-hard when the input of the graph is given in such a succinct way. Subsequently, Papadimitriou and Yannakakis

[13] proved that, under the same representation, problems that are **NP**-complete when the graph is given explicitly become **NEXP**-hard. Finally, in [1], Balcázar *et al.* prove that it is **PSPACE**-hard to decide several fundamental properties in succinct graphs, such as the existence of an Eulerian circuit and of a path connecting two given nodes.

The paper is organized as follows. Firstly, we introduce the concepts and the notation we use, in Section 2, together with three lemmas, from the work of Schurr and Szabó [15], that we use in our constructions. In Section 3 we prove **coNP**-completeness of the USO recognition problem. The **coNP** membership is implicit already in the work of Szabó and Welzl [17], and hardness will follow by a simple reduction from SAT. Section 4 shows that the canonical NO-certificate for the AUSO recognition problem—an explicit list of vertices on a directed cycle—cannot be used to establish **coNP** membership. To this end, we explicitly construct an n -dimensional USO with a unique directed cycle of length $\Omega(2^{n/3})$. Section 5 reveals the deeper reason for the failure of the cycle certificate, namely that the AUSO recognition problem is **PSPACE**-complete. For **PSPACE** membership, we use standard results from complexity theory and the theory of succinct graphs; our main contribution is **PSPACE** hardness, proved via a reduction from satisfiability of quantified Boolean formulas.

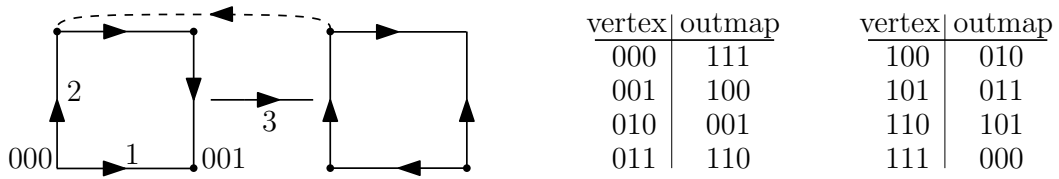
2 Preliminaries

We use the notation $[n] = \{1, \dots, n\}$, and $[i : j]$, $i < j$, for $[i : j] = \{i, \dots, j\}$. Let $Q_n = \{0, 1\}^n$, which we also interpret as the set of vertices of the n -dimensional hypercube. Let $\psi : Q_n \rightarrow Q_n$ be a Boolean function. Moreover, let C_ψ be a Boolean circuit with n inputs and n outputs that represents ψ . There is an explicit ordering on the coordinates and with x_i and $\psi(x)_i$ we denote the i th coordinate of the corresponding bitstring, where the first one is the **rightmost**. Here we use the term bitstring to refer to a ordered string of binary bits. When the subscript is a set, e.g. $x_{[k]}$ or $\psi(x)_{\{2,3\}}$, we mean the bitstring resulting from taking only the coordinates that appear in the subscript. Moreover, we use superscripts to differentiate different functions or different bitstrings, e.g. x^1, x^2 represent two different bitstrings and ψ^1, ψ^2 two different Boolean functions. With the notation \bar{x}_i we mean $1 - x_i$. Given two bitstrings x, y , with $x \cdot y$ we denote their concatenation. Given $x \in Q_n$ we define the neighborhood of x as $\mathcal{N}(x) = \{y \in Q_n \mid x \text{ and } y \text{ are at Hamming distance } 1\}$.

Let $I \in 2^{[n]}$ and $v \in Q_n$. A face of the hypercube, $F_{I,v}$, is defined as the set of vertices that are reached from v by flipping the coordinates defined by any subset of I , i.e. $F_{I,v} = \{u \in Q_n \mid u_i = v_i, \forall i \notin I\}$. The dimension of the face is $|I|$. We call edges the faces of dimension 1, e.g. $F_{\{i\},x}$, and vertices the faces of dimension 0.

We say that $\psi : Q_n \rightarrow Q_n$ represents an orientation when $\forall i \in [n]$ and $\forall x \in Q_n$ we have that $\psi(x)_i \neq \psi(x')_i$, where $F_{\{i\},x} = \{x, x'\}$, i.e. the orientation of every edge is consistent from both sides. Given an orientation ψ and a vertex $x \in Q_n$, we call $\psi(x)$ the *outmap* of x and we call ψ , simply, the outmap. With $G_\psi = (Q_n, E_n)$ we mean the digraph of the hypercube where the edges are oriented according to the outmap. That means that the edge on coordinate i is outgoing for vertex x if and only if $\psi(x)_i = 1$ (cf. Figure 1). Given an orientation ψ of Q_n , a vertex $x \in Q_n$ and an incident edge $F_{\{i\},x}$ we say that the edge is oriented backwards if $\psi(x)_i = x_i$ and is oriented forwards if $\psi(x)_i = \bar{x}_i$. For example, consider vertex 001 in Figure 1: the incident edges on coordinates 1 and 3 are forwards while the one on coordinate 2 is backwards. With $\psi_{\mathbb{F}}$ we denote the orientation such that $\psi_{\mathbb{F}}(x) = \bar{x}$, for all $x \in Q_n$. This orientation is called uniform forwards (all edges are forwards); similarly, the orientation defined by $\psi_{\mathbb{B}}(x) = x$, for all $x \in Q_n$, is called uniform backwards.

► **Definition 1.** A *unique sink orientation* (USO) is an orientation of the hypercube where



■ **Figure 1** An example of a cyclic USO (the vertices participating in the cycle are highlighted as discs). In the left part we give an illustration of the USO graph G_ψ (we explicitly indicate the vertices 000 and 001) and in the right part we give explicitly the Boolean function $\psi : Q_3 \rightarrow Q_3$. In this paper we draw USOs by depicting a number of faces (in this case the 2-dimensional faces) and show the orientation of the edges that connect those. The numbers show the ordering of the coordinates. An arc like the one with label 3 means that all edges on the 3rd coordinate are directed likewise. The dashed arc (also on coordinate 3) means that the specific edge (in this case $F_{\{3\},010}$) is reversed w.r.t. to the the orientation suggested by the (non-dashed) arc labeled 3.

the subgraph induced by every non-empty face has a unique sink.

The existence of a unique sink implies the analogous unique source [17]. As the whole hypercube is a face of itself that means that there is a unique sink (and source) for the whole hypercube, which we call global. We say that ψ or C_ψ represents a USO if the output corresponds to the outmap of a USO and thus G_ψ is a USO. The outmap of a USO is a bijection [17]. If, in addition G_ψ is acyclic, then we call it an acyclic USO (AUSO).

Finally, we give three lemmas from the work of Schurr and Szabó [15] that we use for our constructions. We rephrase the lemmas to use the notation used in the current paper. The first gives us tools to expand a USO to one with more coordinates. The interpretation we use in this paper is that we can take a k_1 -dimensional USO and embed in every vertex an k_2 -dimensional USO; the end-product is a $(k_1 + k_2)$ -dimensional USO, which is acyclic if all involved USOs are acyclic. Note that Lemma 2, as presented in [15], is slightly more general than here; we direct the reader to [15] for full generality.

► **Lemma 2** ([15], Lemma 3). *Let $k_1, k_2 \in \mathbb{N}$ and let $\psi : Q_{k_1} \rightarrow Q_{k_1}$ represent a USO. Let the Boolean functions $\psi^u : Q_{k_2} \rightarrow Q_{k_2}$, for each $u \in Q_{k_1}$, also represent USOs. Consider the Boolean function $\psi' : Q_n \rightarrow Q_n$, where $n = k_1 + k_2$. Let $x \in Q_n$; we define ψ' by*

$$\psi'(x) = \psi(x_{[k_2+1:n]}) \cdot \psi^{x_{[k_2+1:n]}}(x_{[k_2]}).$$

ψ' represents a USO. Furthermore, if ψ and all ψ_u are acyclic, then so is ψ' .

The second lemma says that we are allowed to orient the edge that connects two neighboring vertices x^1, x^2 any way we like and still have a USO, as long as the outmaps of the two vertices are exactly the same in every coordinate that is not the one of the incident edge.

► **Lemma 3** ([15], Corollary 6). *Let $\psi : Q_n \rightarrow Q_n$ represent a USO. Let $x^1, x^2 \in F_{\{i\},x^1} \subseteq Q_n$, such that $\psi(x^1)_{[1:n]\setminus\{i\}} = \psi(x^2)_{[1:n]\setminus\{i\}}$. Then, $\psi' : Q_n \rightarrow Q_n$ with $\psi'(x) = \psi(x)$, for all $x \in Q_n$ except $\psi'(x^1)_i = \overline{\psi(x^1)_i}$ and $\psi'(x^2)_i = \overline{\psi(x^2)_i}$ also represents a USO.*

The third gives lemma describe a constructive process to get an acyclic USO where we can choose which vertex is the global sink and which vertex is the global source.

► **Lemma 4** ([15], Corollary 4). *For any two distinct $x, y \in Q_n$, there exists a $\psi : Q_n \rightarrow Q_n$, with $\psi(x) = 0^n$ and $\psi(y) = 1^n$, such that ψ represents an acyclic USO.*

3 Recognizing USOs

In this section we prove that recognizing a USO is **coNP**-complete. The computational problem is USO-recognizability: We are given a Boolean circuit C_ψ such that $\psi : Q_n \rightarrow Q_n$ and the question is if ψ represents a USO. Note that a **coNP** upper bound for this problem is already known by [17]: A pair of vertices $x, y \in Q_n$ such that $\psi_i(x) = \psi_i(y), \forall i \in I$, where $I = \{i \in [n] \mid x_i \neq y_i\}$, constitutes a short NO certificate.

► **Theorem 5.** *USO-recognizability is **coNP**-complete.*

Proof. We describe a reduction from SAT to the complement of our problem. Let ϕ denote a SAT formula with n variables. By $\phi(x)$ we mean the evaluation of ϕ on $x \in \{0, 1\}^n$, which returns 0 for false and 1 for true. Based on ϕ we construct the Boolean circuit C_ψ , with $\psi : Q_{n+1} \rightarrow Q_{n+1}$. The function is such that on input $x \in Q_n$ we have $\psi(x \cdot 0) = x \cdot 0$ and $\psi(x \cdot 1) = x \cdot \overline{\phi(x)}$. It is easy to see that $x \in Q_n$ is satisfying for ϕ if and only if the pair $x \cdot 0, x \cdot 1$ violates the USO property. ◀

Note that the proof above really is about whether ψ represents a valid orientation. Furthermore, we observe that the hardness proof above also works for *completely unimodal numberings* (CUN). We define these, in the spirit of [19], as bijective functions of the form $\chi : Q_n \rightarrow [0 : 2^n - 1]$, such that every face F of the hypercube has a unique local minimum vertex $x_F \in F$ (which means that x_F attains the minimum value of χ over $\mathcal{N}(x_F) \cap F$). The search problem with CUNs is to find the vertex that attains the value 0. These numberings have been extensively studied, see e.g. [9, 19]. Of course, we can represent χ by a succinct Boolean circuit C_χ with n input and n output bits, such that the output is the binary representation of an integer number. Then, the computational problem of deciding if a given circuit represents a CUN can be proved **coNP**-hard by slightly modifying the reduction above. Moreover, CUNs have short NO certificates (i.e. two vertices that are both local minima of the same face) and thus recognizing if a given circuit represents a CUN is **coNP**-complete. Note that CUNs induce AUSOs by directing every edge from the larger to smaller values [19]. However, as we will see in Section 5, recognizing AUSOs is **PSPACE**-complete.

4 Long Cycles in USO

In this section, we present the construction of a cyclic USO that has a unique cycle of superpolynomial size (number of involved vertices). This demonstrates that we cannot expect a **coNP** upper bound for cyclicity in USOs by listing the set of vertices that participate in a cycle. This intuition is verified in the next section with Theorem 12, where we prove that it is actually **PSPACE**-hard to decide the cyclicity of a USO. At first, we introduce a special class of USOs.

► **Definition 6.** Consider the family of orientations that arises when we start with $G_{\psi_{\text{UF}}}$, choose a matching, and reverse the orientation of the edges of the matching. Call this *flip-matching* orientations (FMO).

Note that when we talk about FMOs in the construction below we mean the graph of the hypercube with the edges directed according to an FMO. Such orientations can be seen to be USOs, as a corollary of Lemma 3 [15]. This fact has also been shown by Matoušek, in [12], who used FMOs to provide $\binom{n}{e}^{2^n - 1}$ as a lower bound on the number of distinct USOs.

In the following, we explain some notation regarding cycles in orientations of the hypercube. Let $x \in Q_n$. With $|x|$ we denote the Hamming weight of x , i.e. the number of ones in the

bitstring x . Note that a forward (backward) edge increases (decreases) Hamming weight by 1. Let $\psi : Q_n \rightarrow Q_n$ be an orientation and consider $G_\psi = (Q_n, E_n)$. Let $c = \{v_1, \dots, v_k\} \subseteq Q_n$ be a k -cycle in G_ψ , that is a cycle over k vertices. Cycles are represented by the set of participating vertices, which we present in order of appearance; the last vertex in the sequence c has an outgoing edge to the first one.

Next, we observe that in an FMO every vertex that participates in a cycle must have an incident backward edge. Let $c \subseteq Q_n$ be a cycle in an FMO and let $v \in c$ be a vertex on the cycle. Assume that v has no backward edge attached. Let v' be the next vertex on c ; we have that $|v'| = |v| + 1$ because the edge $v \rightarrow v'$ is forwards. The vertices that follow v' on the cycle have Hamming weight at least $|v|$, because a lower Hamming weight would imply that there are two consecutive backward edges, which is not allowed by our graph being an FMO. Then we conclude that v is reached with a forward edge from a vertex of Hamming weight at least $|v|$, which is of course not possible. We have proved the following.

► **Lemma 7.** *Let $G = (Q_n, E_n)$ be an FMO. Let $c \subseteq Q_n$ be a cycle in G . Then, every vertex in c has an incident backward edge. It follows that edges on c alternate between forwards and backwards and that reversing a backward edge cannot create any new cycles.*

Following, we describe our lower bound construction. It is an inductive construction that builds an FMO of dimension n from an FMO of dimension $n - 3$. Note that in the resulting FMO we want exactly one cycle. For this we use Lemma 7 on the FMOs of dimension $n - 3$, in order to turn their unique cycles into paths and construct an FMO of dimension n that contains a unique cycle. The base cases are FMOs that contain a unique cycle of size $2n$ for $n = 3, 4, 5$; those are easy to construct, as an example see the 3-dimensional cyclic FMO in Figure 1 (at least 3 dimensions are needed for a USO to be cyclic and 6 is the smallest size for a cycle in a USO).

Let $G^{n-3} = (Q_{n-3}, E_{n-3})$ be the resulting graph after the l th induction step. Let $c = \{v_1, v_2, \dots, v_k\}$ be the unique cycle, of size $6 \leq |c| = k$, that is contained in G^{n-3} . We name the vertices v_1, \dots, v_k in order of appearance on the cycle. We assume that the first edge of the cycle $(v_1, v_2) \in E_{n-3}$ is forwards. In our construction we will use three variants of G^{n-3} w.r.t. Lemma 7 (thus turning the cycle into a path):

1. G_1^{n-3} is derived from G^{n-3} by reversing the backward edge $(v_k, v_1) \in E_{n-3}$;
2. G_2^{n-3} is derived from G^{n-3} by reversing the backward edge $(v_{k-2}, v_{k-1}) \in E_{n-3}$;
3. G_3^{n-3} is derived from G^{n-3} by reversing all the backward edges except $(v_{k-2}, v_{k-1}), (v_k, v_1) \in E_{n-3}$.

Note that the three graphs above are all FMOs. We obtain each of these graphs by reversing edges that were backwards in G^{n-3} to forwards. The fact that the edges described in the first two items are backwards can be seen by Lemma 7.

Now we describe how to proceed with the induction at the $l + 1$ th step and eventually construct G^n . Consider the set of faces $\mathcal{F} = \{F_{[n-3],x \cdot 0^{n-3}} \mid x \in Q_3\}$. These are the faces that appear as ellipsoids in Figure 2. We embed the orientation $G_{\psi|_{\mathcal{F}}}^{n-3}$ in all the faces of \mathcal{F} , with three exceptions: In face $F_{[n-3],110 \cdot 0^{n-3}}$ we embed G_1^{n-3} , in face $F_{[n-3],101 \cdot 0^{n-3}}$ we embed G_2^{n-3} and in face $F_{[n-3],011 \cdot 0^{n-3}}$ we embed G_3^{n-3} . The edges at the extra 3 coordinates follow the forward uniform orientation, except the following three edges that we orient backwards: $F_{\{n-2\},010 \cdot v_2}, F_{\{n-1\},100 \cdot v_k}, F_{\{n\},001 \cdot v_{k-2}}$. See Figure 2 for an illustration of the cycle in G^n .

► **Theorem 8.** *There exist cyclic n -dimensional FMOs that contain a unique cycle of size $\Omega(2^{\frac{n}{3}})$.*

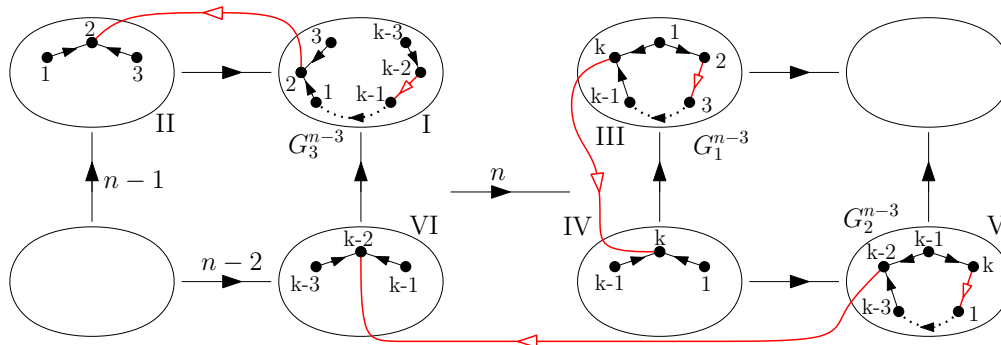


Figure 2 The ellipsoids represent the $(n - 3)$ -dimensional faces in \mathcal{F} . The black edges (with filled arrows) are forwards and the red edges (with non-filled arrows) are backwards. The dotted arcs represent a sequence of edges that starts with a forward one and ends with a backward one. Finally, we show only some of the vertices to illustrate the construction. A vertex with label i denotes v_i , the i th vertex on the cycle. The faces that contain the graphs $G_1^{n-3}, G_2^{n-3}, G_3^{n-3}$ are labeled. To see the cycle one can follow the Latin numbers I, \dots , VI.

Proof. Our construction, as we presented it above, satisfies the claimed theorem. Consider $G^n = (Q_n, E_n)$ and $c \subseteq Q_n$ the cycle in G^n . By induction, the faces where we embed the variants of G^{n-3} are FMOs. The other $(n - 3)$ -dimensional faces in \mathcal{F} contain the uniform orientation. The three new coordinates also obey the uniform orientation except the three edges that we reversed. All of the reversed edges are incident to vertices that do not have other backward edges incident. Thus, G^n is an FMO.

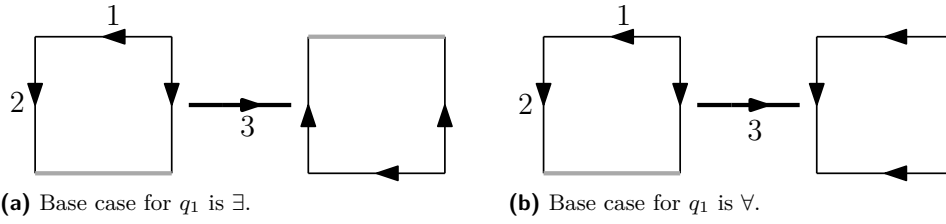
The existence of cycle c in G^n can be witnessed in Figure 2. Furthermore, there are no backward edges other than the ones that are incident to c . Thus, by Lemma 7, there is no other cycle and c is the unique cycle in G^n . Finally, let $C(n)$ denote the size of the cycle in our construction at dimension n . Then, we have the following recursive formula: $C(n) = 2C(n - 3) + 6 = \Omega(2^{\frac{n}{3}})$ ◀

5 Recognizing Acyclic USOs

We start the section with the formal definitions of the two computational problems of interest:

- *AUSO-Accessibility:* The input is a Boolean circuit C_ψ , such that $\psi : Q_m \rightarrow Q_m$, and two vertices $s, t \in Q_m$. The answer to an instance is YES if and only if ψ represents an acyclic USO such that there is a directed path from s to t in G_ψ .
- *USO-Cyclicity:* The input is a Boolean circuit C_ψ , such that $\psi : Q_m \rightarrow Q_m$. The answer to an instance is YES if and only if ψ represents a USO such that there is a directed cycle in G_ψ .

Both these problems can be seen to be in **PSPACE**. Firstly, as we argued in Section 3, we can check if ψ represents a USO in **coNP** (the relationship **coNP** \subseteq **PSPACE** is well-known). Then, the standard argument, that has been used to show that accessibility and cyclicity in directed graphs given by succinct representations, are in **PSPACE** (see e.g. [1, 13]) suffices in our case too. By this argument, we decide the existence of a cycle in the following way: we fix a vertex (non-deterministically) and pick the next vertex from the set of neighbors that can be accessed by an outgoing edge (also non-deterministically). If we reach the same vertex then we conclude that there is a directed cycle (formally here what we need to decide is the non-existence of a cycle; we are using the fact that all deterministic classes are closed



■ **Figure 3** An illustration of the two base cases of the inductive construction.

under complement). Similarly, to decide the existence of an $s - t$ path, we fix vertex s and perform the same process; if we reach t then we conclude that there is an $s - t$ path. These processes use only polynomial space (actually linear, only one vertex needs to be stored in memory) and they give non-deterministic **PSPACE** upper bounds, which is the same as deterministic **PSPACE** by Savitch's Theorem [14].

We are ready to present our first theorem which shows that it is **PSPACE**-hard, and thus by the above argument **PSPACE**-complete, to decide the problem **AUSO-Accessibility**.

► **Theorem 9.** *AUSO-Accessibility is PSPACE-complete.*

The proof is by reduction from the problem of deciding the satisfiability of a Quantified Boolean Formula (QBF) which is the standard **PSPACE**-complete problem. The input to the latter is a CNF formula Φ with n variables v_1, \dots, v_n and a set of n quantifiers q_1, \dots, q_n that can be either \exists or \forall . The construction is presented in an inductive fashion, where the induction is on the number of variables of the QBF formula. The base case is a 3-dimensional acyclic USO and then for each variable we add 3 coordinates when the next quantifier is existential and 4 coordinates when it is universal. All in all, the result of the construction is $\psi : Q_m \rightarrow Q_m$ which represents an acyclic USO and such that $m \leq 4(n - 1) + 3 = 4n - 1$. For this purpose, we describe the construction of G_ψ ; then, the question to be decided is if there exists a directed path from 0^m to 1^m .

We have a set of vertices, called *active* and denoted with $\mathcal{AV} \subset Q_m$. We call an edge $F_{\{i\},x}$ active when $F_{\{i\},x} \subset \mathcal{AV}$. We denote with gray color the active edges in the illustrations for the base case (cf. Figure 3) and the faces that contain active edges in the illustrations for the inductive steps (cf. Figure 4). With \mathcal{AV}^l we denote the set of active vertices after the l th inductive step. The size of \mathcal{AV} is 4 for the base case and it triples at each induction step ($|\mathcal{AV}^l| = 3|\mathcal{AV}^{l-1}|$). The orientations of the active edges depend on an evaluation of Φ for a given assignment that can be obtained by the coordinates of the active vertices. This process will be explained at a later step. We are ready now to describe our construction. The 3-dimensional base cases are presented in Figure 3.

Let $G^l = (Q_k, E_k)$, with $k < 4l$, be the graph after the l th induction step and let q_{l+1} be \exists . We introduce three extra coordinates. At coordinate $(k + 1)$ and $(k + 2)$ all edges are forwards. At coordinate $(k + 3)$ all edges are backwards except the edges $F_{\{k+3\},000 \cdot 1^k}$ and $F_{\{k+3\},010 \cdot 0^k}$ which are reversed. Then, we embed G^l in the faces $\mathcal{F}_0 = F_{[k],0^{k+3}}$, $\mathcal{F}'_0 = F_{[k],1 \cdot 0^{k+2}}$ and $\mathcal{F}_1 = F_{[k],111 \cdot 0^k}$. The rest of the faces in $\mathcal{F}^\exists = \{F_{[k-3],y \cdot 0^{k-3}} \mid y \in Q_3\}$ are all oriented according to $\psi_{\mathbb{U}\mathbb{B}}$ (backwards uniform) in the first k coordinates (cf. Figure 4a).

For the other case, let q_{l+1} be \forall . Introduce four extra coordinates. At coordinate $(k + 1)$ we have that the edges in face $F_{[k+2],0^{k+4}}$ are backwards and every other edge is forwards. At coordinate $(k + 2)$ all edges are backwards except edge $F_{\{k+2\},0000 \cdot 1^k}$ which is reversed. At coordinate $(k + 3)$ all edges are forwards. At coordinate $(k + 4)$ all edges are backwards except

the edge $F_{\{k+4\},0110\cdot 0^k}$ which is reversed. Then, we embed G^l in the faces $\mathcal{F}_0 = F_{[k],0^{k+4}}$, $\mathcal{F}'_0 = F_{[k],0010\cdot 0^k}$ and $\mathcal{F}_1 = F_{[k],1111\cdot 0^k}$. The rest of the faces in $\mathcal{F}^\forall = \{F_{[k-4],y\cdot 0^{k-4}} \mid y \in Q_4\}$ are all oriented according to $\psi_{\mathbb{U}\mathbb{B}}$ in the first k coordinates (cf. Figure 4b).

The graph $G^n = (Q_n, E_n)$ is the end product of our reduction (after the n th induction step). Note that G^n is not an FMO and neither will be the graph we construct in the proof of the next theorem. We still have to describe the orientation of the active edges in G^n . Let $v \in \mathcal{AV}$. The orientation of the active edge adjacent to v , say $e \in E_n$, is decided by the following simple algorithm:

- Let $x \in Q_n$ be the assignment for the variables of the input QBF which we build based on the coordinates of v . Initialize $j = 3$ and $x_1 = v_j$.
- For $i = 2$ to n repeat:
 - If q_i is \exists then set $j \leftarrow j + 3$ and $x_i \leftarrow v_{j-1}$.
 - If q_i is \forall then set $j \leftarrow j + 4$ and $x_i \leftarrow v_j$.
- If $\Phi(x) = 1$ then e is forwards, otherwise it is backwards.

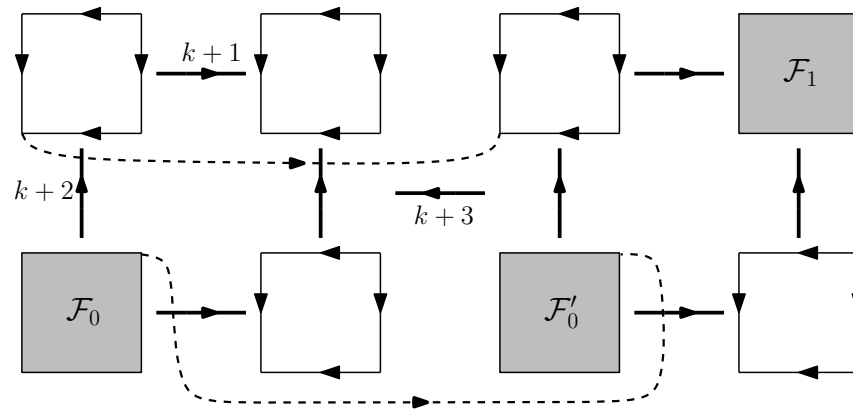
For example, consider the following simple QBF: $\Phi = (v_1 \vee v_2 \vee v_3)$, $q_1 = q_3 = \exists$ and $q_2 = \forall$. This gives rise to a USO over Q_{10} . We give the vertex $v = \mathbf{1001111000}$ as input to the algorithm above (the bold bits are the ones that the algorithm will extract). This is translated to the 3-length bitstring $x = 010$ which means that variable v_2 is set to true and the other two to false and thus $\Phi(x) = 1$ and the corresponding active edge is forwards.

► **Claim 10.** G^n is an acyclic USO.

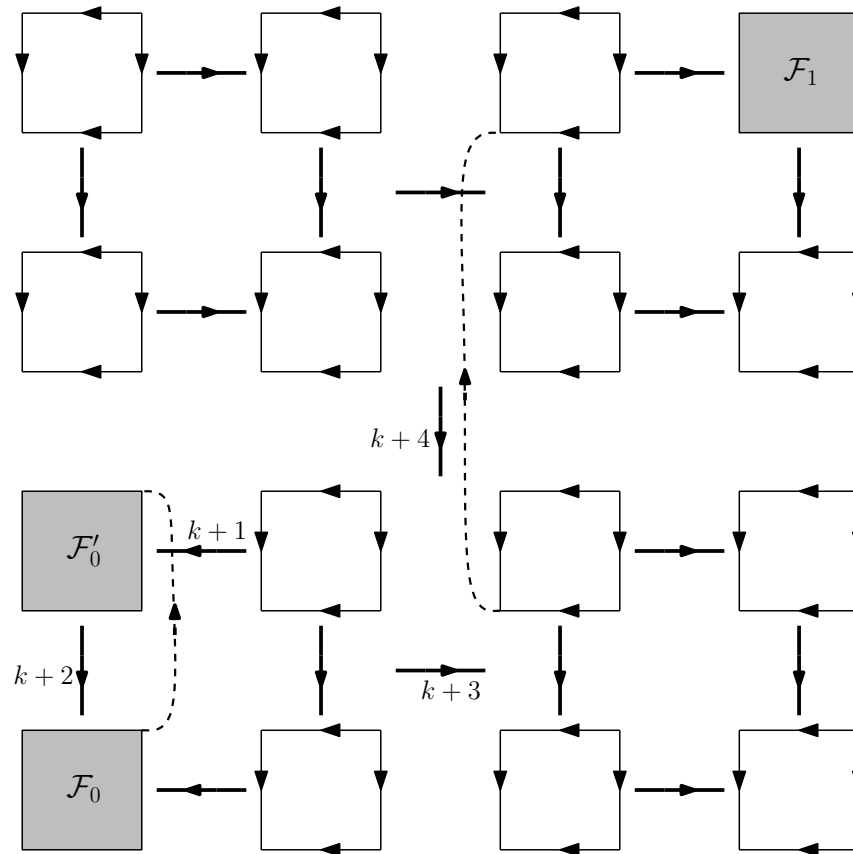
Proof. It can be seen by Lemma 3 that both base cases of the construction are 3-dimensional USOs, regardless of the orientation of the active edges. In addition, they are acyclic because at coordinate 3 every edge is forwards. Then, we argue that for every step of the induction the graph remains an acyclic USO. Consider the $l + 1$ th step of the induction and let q_{l+1} be \exists . Moreover, let $G^{l+1} = (Q_{k+3}, E_{k+3})$ and consider \mathcal{F}^\exists . It holds that every face in \mathcal{F}^\exists is an acyclic USO: for $\mathcal{F}_0, \mathcal{F}'_0, \mathcal{F}_1$ it holds by induction and in every other face we have the backwards uniform orientation.

We interpret the construction in two steps: First, the faces in \mathcal{F}^\exists are put on a 3-dimensional acyclic USO whose orientation is defined above by the orientation of the extra coordinates $(k + 1, k + 2, k + 3)$, before reversing the edges $F_{\{k+3\},000\cdot 1^k}$ and $F_{\{k+3\},010\cdot 0^k}$. The result is an acyclic USO by Lemma 2. In the next step we reverse the aforementioned edges. The result is a USO by Lemma 3. Moreover, reversing these edges does not create any cycles. The orientation in the face $F_{[k] \cup \{k+3\}, 0^{k+3}}$ remains acyclic after reversing $F_{\{k+3\},000\cdot 1^k}$ because the orientations in faces \mathcal{F}_0 and \mathcal{F}'_0 are identical (and a cycle in the former face would imply that the latter faces are cyclic; this is the reason we orient \mathcal{F}'_0 this way). A similar argument applies to reversing the edge $F_{\{k+3\},010\cdot 0^k}$ within the face $F_{[k] \cup \{k+3\}, 010\cdot 0^k}$. All the edges at coordinates $k + 1$ and $k + 2$ are forwards and thus a cycle can only involve the $k + 3$ th coordinate, which is not possible by the arguments above.

The situation is symmetrical when q^{l+1} is \forall and $G^{l+1} = (Q_{k+4}, E_{k+4})$. First, we argue about the set of faces \mathcal{F}^\forall . Then, why reversing the edge $F_{\{k+2\},0000\cdot 1^k}$ does not create any cycles within the face $F_{[k] \cup \{k+2\}, 0^{k+4}}$ and reversing the edge $F_{\{k+4\},0110\cdot 0^k}$ does not create any cycles within the face $F_{[k] \cup \{k+4\}, 0110\cdot 0^k}$. The argument is exactly the same as above. Remember that at the $k + 2$ th coordinate all edges are backwards and at the $k + 3$ th all edges are forwards. Then, we conclude that reversing the edge $F_{\{k+2\},0000\cdot 1^k}$ does not create any cycle in G^{l+1} , since all the edges at the $k + 1$ th and the $k + 4$ th coordinate that are incident to the face $F_{[k], 0^{k+4}}$ are backwards. Furthermore, we conclude that reversing the



(a) G^{l+1} with q_{l+1} is \exists . The k -dimensional faces in \mathcal{F}^\exists appear as 2-faces here.



(b) G^{l+1} with q_{l+1} is \forall . The k -dimensional faces in \mathcal{F}^\forall appear as 2-faces here.

■ **Figure 4** An illustration of the steps of the inductive construction. The active faces (faces that contain active edges) are filled with gray color. The reversed edges are depicted as dashed.

edge $F_{\{k+4\},0110\cdot 0^k}$ does not create any cycle in G^{l+1} , since in the face $F_{[k+4]\setminus\{k+3\},0100\cdot 0^k}$ all edges at the $k + 1$ th coordinate are forwards and at the $k + 2$ th are backwards. ◀

► **Claim 11.** 0^m is connected to 1^m in G^n if and only if the input QBF is satisfiable.

Proof. First, note that at the base case 0^3 is connected to 1^3 if and only if there is at least one forward active edge in the case q_1 is \exists and if and only if both active edges are forwards in the case q_1 is \forall .

Then, consider the $l + 1$ th step of the induction and let the quantifier q_{l+1} be \exists and $G^l = (Q_{k+3}, E_{k+3})$. There is a directed path from 0^{k+3} to 1^{k+3} if and only if at least one of the following is true: Either there is a directed path from 0^{k+3} to $000 \cdot 1^k$ or there is a directed path from $111 \cdot 0^k$ to 1^{k+3} . This is because the $k + 3$ th coordinate is directed backwards for all edges except the two we reversed during the construction ($F_{\{k+3\},000\cdot 1^k}$ and $F_{\{k+3\},010\cdot 0^k}$). If there is a directed path from 0^{k+3} to $000 \cdot 1^k$, then there is one from 0^{k+3} to 1^{k+3} through the edge $F_{\{k+3\},000\cdot 1^k}$. Otherwise, there is an edge from any vertex $x^1 \in \mathcal{F}_0$ to a vertex $x^2 \in F_{[k],010\cdot 0^k} \cap \mathcal{N}(x^1)$, from there to the vertex $010 \cdot 0^k$ and finally to $111 \cdot 0^k$ through edge $F_{\{k+3\},010\cdot 0^k}$. Thus, if there is a path from $111 \cdot 0^k$ to 1^{k+3} then there is a path from 0^{k+3} to 1^{k+3} .

Following, we consider the case that q_{l+1} is \forall and $G^l = (Q_{k+4}, E_{k+4})$. Then, there is a directed path from 0^{k+4} to 1^{k+4} if and only if there is a directed path from 0^{k+4} to $0000 \cdot 1^k$ and one from $1111 \cdot 0^k$ to 1^{k+4} . Note at the $k + 4$ th coordinate all edges are backwards, except $F_{\{k+4\},0110\cdot 0^k}$ and thus a path from 0^{k+4} to 1^{k+4} has to go through vertex $0110 \cdot 0^k$. The only way this is possible is if there is a path from 0^{k+4} to $0000 \cdot 1^k$ and from there through the edge $F_{\{k+2\},0000\cdot 1^k}$ to face \mathcal{F}'_0 and, finally, from there a path to face $F_{[k],0110\cdot 0^k}$. From the latter the vertex $1110 \cdot 0^k$ is accessible and finally the vertex $1111 \cdot 0^k$. A directed path from $1111 \cdot 0^k$ to 1^{k+4} completes the path from 0^{k+4} to 1^{k+4} .

Thus, we have shown the existence of which paths is mandatory, for the existence of a directed path from the all-zero to the all-one vertex in both cases. It remains to explain that these paths exist if and only if the input QBF is satisfiable.

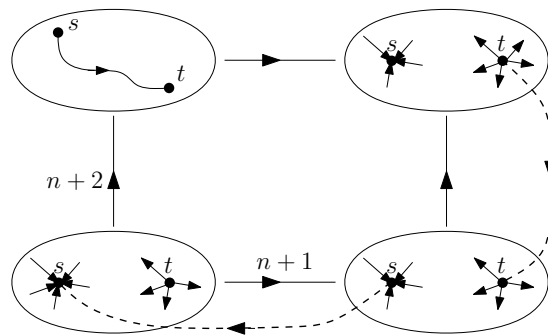
For the forward case of the claim assume that the input QBF is satisfiable. Then, there exists an assignment of the variables of Φ whose quantifier is existential such that, for any assignment of the rest of the variables, Φ is satisfiable. This means that for every step of the induction that corresponds to an existential quantifier there exists a directed path either from 0^{k+3} to $000 \cdot 1^k$ in \mathcal{F}_0 or from $111 \cdot 0^k$ to 1^{k+3} in \mathcal{F}_1 (since the corresponding active edges are forwards). For an inductive step that corresponds to a universal quantifier we have that there are directed paths both from 0^{k+4} to $0000 \cdot 1^k$ in \mathcal{F}_0 and from $1111 \cdot 0^k$ to 1^{k+4} in \mathcal{F}_1 . By the inductive construction this means there is a directed path in G^n from 0^m to 1^m .

Reversely, assume that there is a directed path from the vertex 0^m to the vertex 1^m in G^n . Again, by inductive reasoning. If q_{l+1} is \exists then there is a path in at least one of \mathcal{F}_0 and \mathcal{F}_1 ; this means that Φ is satisfiable for at least one of the two possible assignments. If q_{l+1} is \forall then there are both the paths in \mathcal{F}_0 and \mathcal{F}_1 ; this means that Φ is satisfiable for both possible assignments. ◀

In the next step we prove that USO-Cyclicity is **PSPACE**-hard based on the **PSPACE**-hardness of AUSO-Accessibility. This implies **PSPACE**-completeness by the arguments we gave in the beginning of this section.

► **Theorem 12.** *USO-Cyclicity is **PSPACE**-complete.*

Proof. The reduction is from QBF. In a first step, we reduce to an instance of AUSO-Accessibility as in the proof of Theorem 9. Then we have G_ψ which is an AUSO (we use this



■ **Figure 5** An illustration of the construction. The ellipsoids represent n -dimensional USOs. The face $F_{[n],10 \cdot 0^n}$ contains the orientation of the AUSO-Accessibility instance.

trick since the formal definition of AUSO-Accessibility does not guarantee that the input to the problem represents an AUSO). Based on G_ψ we define $G_{\psi'}$, where $\psi' : Q_{n+2} \rightarrow Q_{n+2}$. All the edges at coordinates $n+1$ and $n+2$ are forwards except $F_{\{n+1\},00 \cdot s}$ and $F_{\{n+2\},01 \cdot t}$ which are reversed. We have now defined the orientation of the edges at the two extra coordinates and we turn our attention to the first n ones. In face $F_{[n],10 \cdot 0^n}$ we embed G_ψ which is an AUSO. Let $G_{\psi''}$ be the AUSO graph that results by applying Lemma 4 with $\psi''(s) = 0^n$ and $\psi''(t) = 1^n$. We embed $G_{\psi''}$ in faces $F_{[n],00 \cdot 0^n}$, $F_{[n],01 \cdot 0^n}$ and $F_{[n],11 \cdot 0^n}$. It follows that $G_{\psi'}$ is a USO from the above argument and the fact that reversing the edges is safe by Lemma 3. An illustration of the construction can be found in Figure 5.

► **Claim 13.** There is a cycle in $G_{\psi'}$ if and only if there is a directed path from s to t in G_ψ .
 By construction, there is a path from vertex $10 \cdot t$ to $01 \cdot t$ through $11 \cdot t$. Note that since $11 \cdot t$ is the source of the face $F_{[n],11 \cdot 0^n}$ a path from any vertex of the face $F_{[n],10 \cdot 0^n}$ to $11 \cdot t$ has to go through vertex $10 \cdot t$. In $F_{[n],01 \cdot 0^n}$ there is path from $01 \cdot t$ (which is the source of the face) to $01 \cdot s$ (which is the sink of the face). From the latter there is a path to vertex $00 \cdot s$ (which is the sink of the face $F_{[n],00 \cdot 0^n}$ and thus no other vertex of the same face is accessible from it) and finally to $10 \cdot s$. In addition, note that the desired cycle is the only one that will use both coordinates $n+1$ and $n+2$ (if it exists). There is no other cycle that involves only one of the two extra coordinates. This is because the existence of such a cycle would imply that the orientations embedded in $F_{[n],00 \cdot 0^n}$, $F_{[n],01 \cdot 0^n}$ and $F_{[n],11 \cdot 0^n}$ are cyclic (we have also seen this argument in the proof of Claim 10). The claim follows. ◀

The reductions described in this section give as a result a directed graph. However, the graph of the hypercube is obviously of exponential size and we are interested in a Boolean circuit that succinctly describes it. As we have already argued, this is done by actually describing the outmap of the USO. The size of such a circuit depends only on n , the number of variables of the QBF. It is discussed in [1] that the techniques used by Ladner in [10] can be used to construct such a circuit in polynomial time. Nonetheless, in our case, and because the graph is very structured, it is not too hard to explicitly describe the construction of the actual circuits for Theorems 9 and 12. The description is a bit tedious and, due to the lack of space, we postpone it to the full version of the current paper. We remark that for the proof of Theorem 9 the circuit contains internally another circuit that, given an assignment x of the n variables of Φ , returns the evaluation $\Phi(x)$. The latter is used in the algorithm described in the proof of Theorem 9 to decide the orientation of the active edges. It is known that such evaluations can be performed in polynomial time (see e.g. [11]) and thus such a circuit is easy to obtain.

References

- 1 José L. Balcázar, Antoni Lozano, and Jacobo Torán. The complexity of algorithmic problems on succinct instances. In Ricardo Baeza-Yates and Udi Manber, editors, *Computer Science*, pages 351–377. Springer US, 1992.
- 2 Gregory E. Coxson. The P-matrix problem is co-NP-complete. *Mathematical Programming*, 64(1-3):173–178, 1994.
- 3 Kaspar Fischer and Bernd Gärtner. The smallest enclosing ball of balls: combinatorial structure and algorithms. *Internat. J. Comput. Geom. Appl.*, 14(4-5):341–378, 2004.
- 4 Jan Foniok, Bernd Gärtner, Lorenz Klaus, and Markus Sprecher. Counting Unique-Sink Orientations. *Discrete Applied Mathematics*, 163, Part 2:155–164, 2014.
- 5 Hana Galperin and Avi Wigderson. Succinct representations of graphs. *Information and Control*, 56(3):183–198, 1983.
- 6 Bernd Gärtner. The Random-Facet simplex algorithm on combinatorial cubes. *Random Structures & Algorithms*, 20(3):353–381, 2002.
- 7 Bernd Gärtner and Leo Rüst. Simple stochastic games and P-matrix generalized linear complementarity problems. In Maciej Liskiewicz and Rüdiger Reischuk, editors, *Proceedings of the 15th International Symposium on Fundamentals of Computation Theory (FCT’05)*, volume 3623 of *Lecture Notes in Computer Science*, pages 209–220. Springer, 2005.
- 8 Bernd Gärtner and Ingo Schurr. Linear Programming and Unique Sink Orientations. In *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 749–757, 2006.
- 9 Peter L. Hammer, Bruno Simeone, Thomas M. Liebling, and Dominique de Werra. From linear separability to unimodality: A hierarchy of pseudo-Boolean functions. *SIAM J. Discrete Math.*, 1(2):174–184, 1988.
- 10 Richard E. Ladner. The circuit value problem is log space complete for P. *SIGACT News*, 7(1):18–20, 1975.
- 11 Nancy Lynch. Log space recognition and translation of parenthesis languages. *J. ACM*, 24(4):583–590, 1977.
- 12 Jiří Matoušek. The number of Unique-Sink Orientations of the hypercube*. *Combinatorica*, 26(1):91–99, 2006.
- 13 Christos H. Papadimitriou and Mihalis Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71(3):181–185, 1986.
- 14 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.
- 15 Ingo Schurr and Tibor Szabó. Finding the sink takes some time: An almost quadratic lower bound for finding the sink of unique sink oriented cubes. *Discrete & Computational Geometry*, 31(4):627–642, 2004.
- 16 Alan Stickney and Layne Watson. Digraph models of Bard-type algorithms for the linear complementarity problem. *Math. Oper. Res.*, 3(4):322–333, 1978.
- 17 Tibor Szabó and Emo Welzl. Unique Sink Orientations of cubes. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS’01)*, pages 547–555, 2001.
- 18 Stefano Tessaro. Randomized algorithms to locate the sink in low dimensional Unique Sink Orientations of cubes. Semester thesis, Computer Science Department, ETH Zürich, 2004.
- 19 Kathy Williamson-Hoke. Completely unimodal numberings of a simple polytope. *Discrete Applied Mathematics*, 20(1):69–81, 1988.

New Geometric Representations and Domination Problems on Tolerance and Multitolerance Graphs*

Archontia C. Giannopoulou and George B. Mertzios

School of Engineering and Computing Sciences, Durham University, UK
archontia.giannopoulou@gmail.com, george.mertzios@durham.ac.uk

Abstract

Tolerance graphs model interval relations in such a way that intervals can tolerate a certain amount of overlap without being in conflict. In one of the most natural generalizations of tolerance graphs with direct applications in the comparison of DNA sequences from different organisms, namely *multitolerance* graphs, two tolerances are allowed for each interval – one from the left and one from the right side. Several efficient algorithms for optimization problems that are NP-hard in general graphs have been designed for tolerance and multitolerance graphs. In spite of this progress, the complexity status of some fundamental algorithmic problems on tolerance and multitolerance graphs, such as the *dominating set* problem, remained unresolved until now, three decades after the introduction of tolerance graphs. In this article we introduce two new geometric representations for tolerance and multitolerance graphs, given by points and line segments in the plane. Apart from being important on their own, these new representations prove to be a powerful tool for deriving both hardness results and polynomial time algorithms. Using them, we surprisingly prove that the dominating set problem can be solved in polynomial time on tolerance graphs and that it is APX-hard on multitolerance graphs, solving thus a longstanding open problem. This problem is the first one that has been discovered with a different complexity status in these two graph classes. Furthermore we present an algorithm that solves the independent dominating set problem on multitolerance graphs in polynomial time, thus demonstrating the potential of this new representation for further exploitation via sweep line algorithms.

1998 ACM Subject Classification G.2.2 Graph algorithms, F.2.2 Geometrical problems and computations: Computations on discrete structures

Keywords and phrases tolerance graphs, multitolerance graphs, geometric representation, dominating set problem, polynomial time algorithm, APX-hard

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.354

1 Introduction

A graph $G = (V, E)$ on n vertices is a *tolerance* graph if there exists a collection $I = \{I_v \mid v \in V\}$ of intervals on the real line and a set $t = \{t_v \mid v \in V\}$ of positive numbers (the tolerances), such that for any two vertices $u, v \in V$, $uv \in E$ if and only if $|I_u \cap I_v| \geq \min\{t_u, t_v\}$, where $|I|$ denotes the length of the interval I . The pair $\langle I, t \rangle$ is called a *tolerance representation* of G . If G has a tolerance representation $\langle I, t \rangle$, such that $t_v \leq |I_v|$ for every $v \in V$, then G is called a *bounded tolerance* graph.

If we replace in the above definition “min” by “max”, we obtain the class of *max-tolerance* graphs. Both tolerance and max-tolerance graphs have attracted many research

* Partially supported by the EPSRC Grant EP/K022660/1.

efforts [2, 4, 7, 9, 10, 12, 14–17] as they find numerous applications, especially in bioinformatics, among others [10, 12, 14]; for a more detailed account see the book on tolerance graphs [11]. One of their major applications is in the comparison of DNA sequences from different organisms or individuals by making use of a software tool like BLAST [1]. However, at some parts of the above genomic sequences in BLAST, we may want to be more tolerant than at other parts, since for example some of them may be biologically less significant or we have less confidence in the exact sequence due to sequencing errors in more error prone genomic regions. This concept leads naturally to the notion of *multitolerance* graphs which generalize tolerance graphs [11, 15, 19]. The main idea is to allow two different tolerances for each interval, one to each of its sides. Then, every interval tolerates in its interior part the intersection with other intervals by an amount that is a convex combination of these two border-tolerances.

Formally, let $I = [l, r]$ be an interval on the real line and $l_t, r_t \in I$ be two numbers between l and r , called *tolerant points*. For every $\lambda \in [0, 1]$, we define the interval $I_{l_t, r_t}(\lambda) = [l + (r_t - l)\lambda, l_t + (r - l_t)\lambda]$, which is the convex combination of $[l, l_t]$ and $[r_t, r]$. Furthermore, we define the set $\mathcal{I}(I, l_t, r_t) = \{I_{l_t, r_t}(\lambda) \mid \lambda \in [0, 1]\}$ of intervals. That is, $\mathcal{I}(I, l_t, r_t)$ is the set of all intervals that we obtain when we linearly transform $[l, l_t]$ into $[r_t, r]$. For an interval I , the *set of tolerance-intervals* τ of I is defined either as $\tau = \mathcal{I}(I, l_t, r_t)$ for some values $l_t, r_t \in I$ (the case of a *bounded* vertex), or as $\tau = \{\mathbb{R}\}$ (the case of an *unbounded* vertex). A graph $G = (V, E)$ is a *multitolerance* graph if there exists a collection $I = \{I_v \mid v \in V\}$ of intervals and a family $t = \{\tau_v \mid v \in V\}$ of sets of tolerance-intervals, such that: for any two vertices $u, v \in V$, $uv \in E$ if and only if $Q_u \subseteq I_v$ for some $Q_u \in \tau_u$, or $Q_v \subseteq I_u$ for some $Q_v \in \tau_v$. Then, the pair $\langle I, t \rangle$ is called a *multitolerance representation* of G . If G has a multitolerance representation with only bounded vertices, i.e., with $\tau_v \neq \{\mathbb{R}\}$ for every vertex v , then G is called a *bounded multitolerance* graph.

For several optimization problems that are NP-hard in general graphs, such as the coloring, clique, and independent set problems, efficient algorithms are known for tolerance and multitolerance graphs. However, only few of them have been derived using the (multi)tolerance representation (e.g. [10, 19]), while most of these algorithms appeared as a consequence of the containment of tolerance and multitolerance graphs to weakly chordal (and thus also to perfect) graphs [20]. To design efficient algorithms for (multi)tolerance graphs, it seems to be essential to assume that a suitable representation of the graph is given along with the input, as it has been recently proved that the recognition of tolerance graphs is NP-complete [17]. Recently two new geometric intersection models in the 3-dimensional space have been introduced for both tolerance graphs (the *parallelepiped* representation [16]) and multitolerance graphs (the *trapezopiped* representation [15]), which enabled the design of very efficient algorithms for such problems, in most cases with (optimal) $O(n \log n)$ running time [15, 16]. In spite of this, the complexity status of some algorithmic problems on tolerance and multitolerance graphs still remains open, three decades after the introduction of tolerance graphs in [8]. Arguably the two most famous and intriguing examples of such problems are the *minimum dominating set* problem and the *Hamilton cycle* problem (see e.g. [20, page 314]). Both these problems are known to be NP-complete on the greater class of weakly chordal graphs [3, 18] but solvable in polynomial time in the smaller classes of bounded tolerance and bounded multitolerance (i.e., trapezoid) graphs [6, 13]. The reason that these problems resisted solution attempts over the years seems to be that the existing representations for (multi)tolerance graphs do not provide enough insight to deal with these problems.

In this article we introduce a new geometric representation for multitolerance graphs, which we call the *shadow representation*, given by a set of line segments and points in the

plane. In the case of tolerance graphs, this representation takes a very special form, in which all line segments are horizontal, and therefore we call it the *horizontal shadow representation*. Note that both the shadow and the horizontal shadow representations are *not* intersection models for multitolerance graphs and for tolerance graphs, respectively, in the sense that two line segments may not intersect in the representation although the corresponding vertices are adjacent. However, the main advantage of these two new representations is that they provide substantially new insight for tolerance and multitolerance graphs and they can be used to interpret optimization problems (such as the dominating set problem and its variants) using computational geometry terms.

Apart from being important on their own, these new representations enable us to establish the complexity of the *minimum dominating set* problem on both tolerance and multitolerance graphs, thus solving a longstanding open problem. Given a horizontal shadow representation of a tolerance graph G , we present an algorithm that computes a minimum dominating set in polynomial time. On the other hand, using the shadow representation, we prove that the minimum dominating set problem is APX-hard on multitolerance graphs by providing a reduction from a special case of the set cover problem. That is, there exists no Polynomial Time Approximation Scheme (PTAS) for this problem unless $P=NP$. This is the first problem that has been discovered with a different complexity status in these two graph classes. Therefore, given the (seemingly) small difference between the definition of tolerance and multitolerance graphs, this dichotomy result appears to be surprising. Furthermore we present an easy algorithm that solves (using the shadow representation) the independent dominating set problem on multitolerance graphs in polynomial time. This algorithm demonstrates the potential of this new representation for further exploitation via sweep line algorithms. Due to lack of space, full proofs are given in a clearly marked appendix.

Throughout the article we consider simple undirected graphs with no loops or multiple edges. In an undirected graph G the edge between two vertices u and v is denoted by uv , and in this case u and v are said to be *adjacent* in G . We denote by $N(u) = \{v \in V : uv \in E\}$ the set of neighbors of a vertex u in G , and $N[u] = N(u) \cup \{u\}$. Given a graph $G = (V, E)$ and a subset $S \subseteq V$, $G[S]$ denotes the induced subgraph of G on the vertices in S . A subset $S \subseteq V$ is a *dominating set* of G if every vertex $v \in V \setminus S$ has at least one neighbor in S . A subset $S \subseteq V$ is an *independent set* of G if $G[S]$ has no edges. Furthermore $S \subseteq V$ is an *independent dominating set* of G if S is both an independent set and a dominating set of G . Note that any inclusion maximal independent set is also an independent dominating set. The *(independent) dominating set problem* is the problem of computing an (independent) dominating set of minimum size in a given graph G . Finally, given a set $X \subseteq \mathbb{R}^2$ of points in the plane, we denote by $H_{\text{convex}}(X)$ the *convex hull* defined by the points of X , and by $\bar{X} = \mathbb{R}^2 \setminus X$ the complement of X in \mathbb{R}^2 . For simplicity of the presentation we make the following notational convention throughout the paper: whenever we need to compute a set S with the smallest cardinality among a family \mathcal{S} of sets, we write $S = \min\{\mathcal{S}\}$.

2 Tolerance and Multitolerance Graphs

In this section we briefly revise the 3-dimensional intersection models for tolerance graphs [16] and multitolerance graphs [15] and some useful properties of these models that are needed for the remainder of the paper. Consider a multitolerance graph $G = (V, E)$ that is given along with a multitolerance representation R . Let V_B and V_U denote the set of bounded and unbounded vertices of G in this representation, respectively. Consider now two parallel lines L_1 and L_2 in the plane. For every vertex $v \in V = V_B \cup V_U$, we appropriately construct a

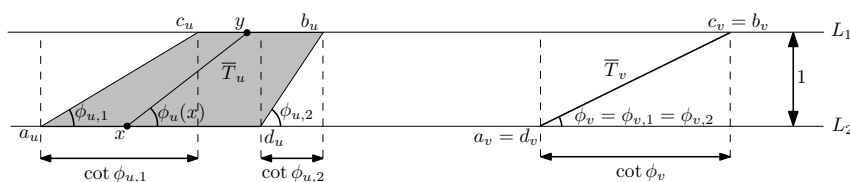


Figure 1 The trapezoid \bar{T}_u corresponds to the bounded vertex $u \in V_B$, while the line segment \bar{T}_v corresponds to the unbounded vertex $v \in V_U$.

trapezoid \bar{T}_v with its parallel lines on L_1 and L_2 , respectively (for details of this construction of the trapezoids we refer to [15]). According to this construction, for every unbounded vertex $v \in V_U$ the trapezoid \bar{T}_v is trivial, i.e., a line [15]. For every vertex $v \in V = V_B \cup V_U$ we denote by a_v, b_v, c_v, d_v the lower left, upper right, upper left, and lower right endpoints of the trapezoid \bar{T}_v , respectively. Note that for every unbounded vertex $v \in V_U$ we have $a_v = d_v$ and $c_v = b_v$, since \bar{T}_v is just a line segment. An example is depicted in Figure 1, where \bar{T}_u corresponds to a bounded vertex u and \bar{T}_v corresponds to an unbounded vertex v .

We now define the left and right angles of these trapezoids. For every angle ϕ , the values $\tan \phi$ and $\cot \phi = \frac{1}{\tan \phi}$ denote the tangent and the cotangent of ϕ , respectively. Furthermore, $\phi = \text{arccot } x$ is the angle ϕ , for which $\cot \phi = x$.

Definition 1 ([15]). For every vertex $v \in V = V_B \cup V_U$, the values $\phi_{v,1} = \text{arccot } (c_v - a_v)$ and $\phi_{v,2} = \text{arccot } (b_v - d_v)$ are the *left angle* and the *right angle* of \bar{T}_v , respectively. Moreover, for every unbounded vertex $v \in V_U$, $\phi_v = \phi_{v,1} = \phi_{v,2}$ is the *angle* of \bar{T}_v .

Note that without loss of generality we can assume that all endpoints and angles of the trapezoids are distinct, i.e., $\{a_u, b_u, c_u, d_u\} \cap \{a_v, b_v, c_v, d_v\} = \emptyset$ and $\{\phi_{u,1}, \phi_{u,2}\} \cap \{\phi_{v,1}, \phi_{v,2}\} = \emptyset$ for every $u, v \in V$ with $u \neq v$, as well as that $0 < \phi_{v,1}, \phi_{v,2} < \frac{\pi}{2}$ for all angles $\phi_{v,1}, \phi_{v,2}$ [15]. It is important to note here that this set of trapezoids $\{\bar{T}_v : v \in V = V_B \cup V_U\}$ is *not* an intersection model for the graph G , as two trapezoids \bar{T}_v, \bar{T}_w may have a non-empty intersection although $vw \notin E$. However the subset of trapezoids $\{\bar{T}_v : v \in V_B\}$ that corresponds to the *bounded* vertices is an intersection model of the induced subgraph $G[V_B]$, i.e., $uv \in E$ if and only if $\bar{T}_u \cap \bar{T}_v \neq \emptyset$ where $u, v \in V_B$.

In order to construct an intersection model for the whole graph G (i.e., including also the set V_U of the unbounded vertices), we exploit the third dimension as follows. Let $\Delta = \max\{b_v : v \in V\} - \min\{a_u : u \in V\}$ (where we consider the endpoints b_v and a_u as real numbers on the lines L_1 and L_2 , respectively). First, for every unbounded vertex $v \in V_U$ we construct the line segment $T_v = \{(x, y, z) : (x, y) \in \bar{T}_v, z = \Delta - \cot \phi_v\}$. For every bounded vertex $v \in V_B$, denote by $\bar{T}_{v,1}$ and $\bar{T}_{v,2}$ the left and the right line segment of the trapezoid \bar{T}_v , respectively. We construct two line segments $T_{v,1} = \{(x, y, z) : (x, y) \in \bar{T}_{v,1}, z = \Delta - \cot \phi_{v,1}\}$ and $T_{v,2} = \{(x, y, z) : (x, y) \in \bar{T}_{v,2}, z = \Delta - \cot \phi_{v,2}\}$. Then, for every $v \in V_B$, we construct the 3-dimensional object T_v as the convex hull $H_{\text{convex}}(\bar{T}_v, T_{v,1}, T_{v,2})$; this 3-dimensional object T_v is called the *trapezoepiped* of vertex $v \in V_B$. The resulting set $\{T_v : v \in V = V_B \cup V_U\}$ of objects in the 3-dimensional space is called the *trapezoepiped representation* of the multitolerance graph G [15]. This is an *intersection model* of G , i.e., two vertices v, w are adjacent if and only if $T_v \cap T_w \neq \emptyset$. For a proof of this fact and for more details about the trapezoepiped representation of multitolerance graphs we refer to [15].

An example of this construction is given in Figure 2. A multitolerance graph G with six vertices $\{v_1, v_2, \dots, v_6\}$ is depicted in Figure 2a, while the trapezoepiped representation of G is illustrated in Figure 2b. The set of bounded and unbounded vertices in this representation are $V_B = \{v_3, v_4, v_6\}$ and $V_U = \{v_1, v_2, v_5\}$, respectively.

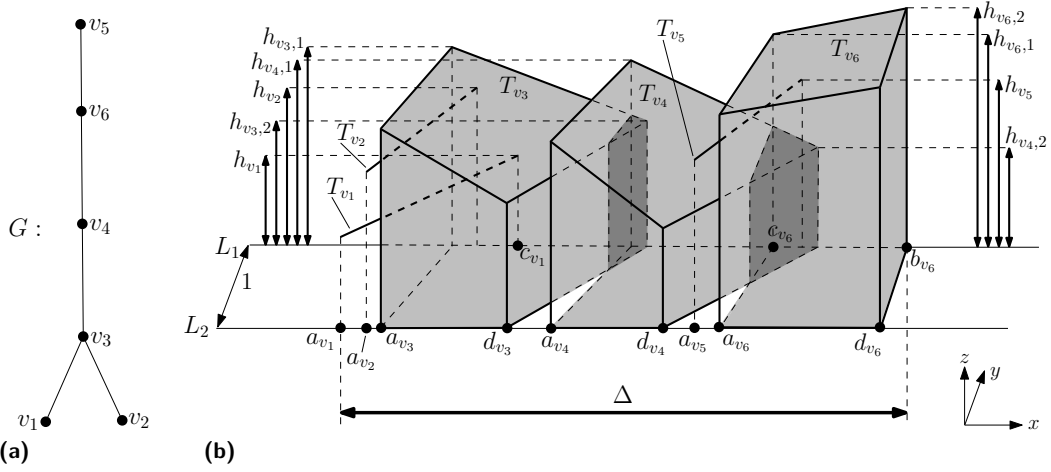


Figure 2 (a) A multitolerance graph G and (b) a trapezoped representation R of G . Here, $h_{v_i,j} = \Delta - \cot \phi_{v_i,j}$ for every bounded vertex $v_i \in V_B$ and $j \in \{1, 2\}$, while $h_{v_i} = \Delta - \cot \phi_{v_i}$ for every unbounded vertex $v_i \in V_U$.

► **Definition 2** ([15]). An unbounded vertex $v \in V_U$ is *inevitable* if replacing T_v by $H_{\text{convex}}(T_v, \bar{T}_v)$ creates a new edge uv in G ; then u is a *hovering vertex* of v and the set $H(v)$ of all hovering vertices of v is the *hovering set* of v . A trapezoped representation of a multitolerance graph G is called *canonical* if every unbounded vertex is inevitable.

In the example of Figure 2, v_2 and v_5 are inevitable unbounded vertices, v_1 and v_4 are hovering vertices of v_2 and v_5 , respectively, while v_1 is not an inevitable unbounded vertex. Therefore, this representation is not canonical for the graph G . However, if we replace T_{v_1} by $H_{\text{convex}}(T_{v_1}, a_{v_1}, c_{v_1})$, we get a canonical representation for G in which vertex v_1 is bounded.

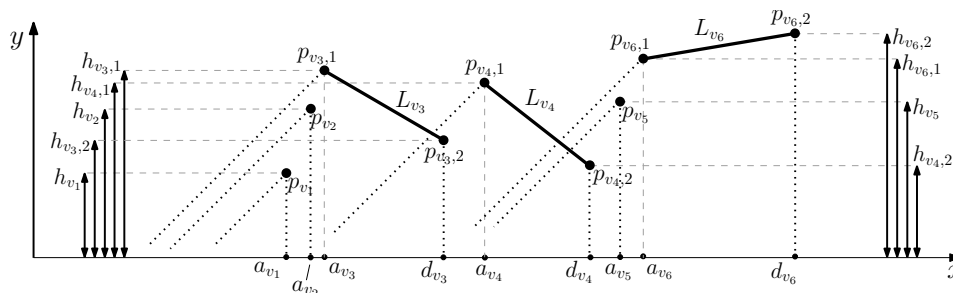
Let G be a multitolerance graph and R be a trapezoped representation of G , where $\phi_{u,1} = \phi_{u,2}$ for every bounded vertex $u \in V_B$. Then, for every $u \in V_B$, T_u becomes a *parallelepiped* and it can be proved that G is a *tolerance graph* [15]. This particular 3-dimensional intersection model for tolerance graphs is known as a *parallelepiped representation* [16].

3 The New Geometric Representations

In this section we introduce the *shadow representation* of multitolerance graphs, which is given by a set of line segments and points in the plane. Given a trapezoped representation of a multitolerance graph G with n vertices, we can compute a shadow representation of G in $O(n)$ time. Whenever G admits a parallelepiped representation (i.e., G is a tolerance graph) all line segments in the shadow representation of G become horizontal, and in this case we call it a *horizontal shadow representation*.

► **Definition 3** (shadow representation). Let $G = (V, E)$ be a multitolerance graph, R be a trapezoped representation of G , and V_B, V_U be the sets of bounded and unbounded vertices of G in R , respectively. We associate the vertices of G with the following points and line segments in the plane:

- for every $v \in V_B$, the points $p_{v,1} = (a_v, \Delta - \cot \phi_{v,1})$ and $p_{v,2} = (d_v, \Delta - \cot \phi_{v,2})$ and the line segment $L_v = (p_{v,1}, p_{v,2})$,
- for every $v \in V_U$, the point $p_v = (a_v, \Delta - \cot \phi_v)$.



■ **Figure 3** The shadow representation $(\mathcal{P}, \mathcal{L})$ of the multitolerance graph G of Figure 2. The unbounded vertices $V_U = \{v_1, v_2, v_5\}$ are associated with the points $\mathcal{P} = \{p_{v_1}, p_{v_2}, p_{v_5}\}$, while the bounded vertices $V_B = \{v_3, v_4, v_6\}$ are associated with the line segments $\mathcal{L} = \{L_{v_3}, L_{v_4}, L_{v_6}\}$, respectively.

The tuple $(\mathcal{P}, \mathcal{L})$, where $\mathcal{L} = \{L_v : v \in V_B\}$ and $\mathcal{P} = \{p_v : v \in V_U\}$, is the *shadow representation* of G . If $\phi_{v,1} = \phi_{v,2}$ for every $v \in V_B$, then $(\mathcal{P}, \mathcal{L})$ is the *horizontal shadow representation* of the tolerance graph G . Furthermore, the representation $(\mathcal{P}, \mathcal{L})$ is *canonical* if the initial trapezoepiped representation R is also canonical.

As an example we illustrate in Figure 3 the shadow representation $(\mathcal{P}, \mathcal{L})$ of the multitolerance graph G of Figure 2.

► **Definition 4** (shadow). For an arbitrary point $t = (t_x, t_y) \in \mathbb{R}^2$ the *shadow* of t is the region $S_t = \{(x, y) \in \mathbb{R}^2 : x \leq t_x, y - x \leq t_y - t_x\}$. Furthermore, for every line segment L_u , where $u \in V_B$, the *shadow* of L_u is the region $S_u = \bigcup_{t \in L_u} S_t$.

► **Definition 5** (reverse shadow). For an arbitrary point $t = (t_x, t_y) \in \mathbb{R}^2$ the *reverse shadow* of t is the region $F_t = \{(x, y) \in \mathbb{R}^2 : x \geq t_x, y - x \geq t_y - t_x\}$. Furthermore, for every line segment L_i , where $u \in V_B$, the *reverse shadow* of L_i is the region $F_i = \bigcup_{t \in L_i} F_t$.

► **Lemma 6.** Let G be a multitolerance graph and $(\mathcal{P}, \mathcal{L})$ be a shadow representation of G . Let $u \in V_B$ be a bounded vertex of G such that the corresponding line segment L_u is not trivial, i.e., L_u is not a single point. Then the angle of the line segment L_u with a horizontal line (i.e., parallel to the x -axis) is at most $\frac{\pi}{4}$ and at least $-\frac{\pi}{2}$.

Recall now that two unbounded vertices $u, v \in V_U$ are never adjacent. The connection between a multitolerance graph G and a shadow representation of it is given in Lemmas 7 and 8. Furthermore Lemma 9 describes how the hovering vertices of an unbounded vertex $v \in V_U$ (cf. Definition 2) can be seen in a shadow representation $(\mathcal{P}, \mathcal{L})$.

► **Lemma 7.** Let $(\mathcal{P}, \mathcal{L})$ be a shadow representation of a multitolerance graph G . Let $u, v \in V_B$ be two bounded vertices of G . Then $uv \in E$ if and only if $L_u \cap S_v \neq \emptyset$ or $L_v \cap S_u \neq \emptyset$.

► **Lemma 8.** Let $(\mathcal{P}, \mathcal{L})$ be a shadow representation of a multitolerance graph G . Let $v \in V_U$ and $u \in V_B$ be two vertices of G . Then $uv \in E$ if and only if $p_v \in S_u$.

► **Lemma 9.** Let $(\mathcal{P}, \mathcal{L})$ be a shadow representation of a multitolerance graph G . Let $v \in V_U$ be an unbounded vertex of G and $u \in V \setminus \{v\}$ be another arbitrary vertex. If $u \in V_B$ (resp. $u \in V_U$), then u is a hovering vertex of v if and only if $L_u \cap S_v \neq \emptyset$ (resp. $p_u \in S_v$).

In the example of Figure 3 the shadows of the points in \mathcal{P} and of the line segments in \mathcal{L} are shown with dotted lines. For instance, $p_{v_2} \in S_{v_3}$ and $p_{v_2} \notin S_{v_4}$, and thus the unbounded

vertex v_2 is adjacent to the bounded vertex v_3 but not to the bounded vertex v_4 . Furthermore $L_{v_3} \cap S_{v_4} \neq \emptyset$, and thus v_3 and v_4 are adjacent. On the other hand, $L_{v_3} \cap S_{v_6} = L_{v_6} \cap S_{v_3} = \emptyset$, and thus v_3 and v_6 are not adjacent. Finally $p_{v_1} \in S_{v_2}$ and $L_{v_4} \cap S_{v_5} \neq \emptyset$, and thus v_1 is a hovering vertex of v_2 and v_4 is a hovering vertex of v_5 . These facts can be also checked in the trapezoepiped representation of the same multitolerance graph G in Figure 2b.

4 Dominating Set is APX-hard on Multitolerance Graphs

In this section we prove that the dominating set problem on multitolerance graphs is APX-hard via an approximation-preserving reduction [21] from a special case of the set cover problem, namely SPECIAL 3-SET COVER [5].

► **Theorem 10.** DOMINATING SET is APX-hard on multitolerance graphs.

5 Bounded Dominating Set on Tolerance Graphs

In this section we use the *horizontal shadow representation* of tolerance graphs (cf. Section 3) to provide a polynomial time algorithm for a variation of the minimum dominating set problem on tolerance graphs, namely BOUNDED DOMINATING SET, formally defined below. This problem variation may be interesting on its own, but we use our algorithm for BOUNDED DOMINATING SET as a subroutine in our algorithm for the minimum dominating set problem on tolerance graphs, cf. Sections 6 and 7. Note that, given a horizontal shadow representation $(\mathcal{P}, \mathcal{L})$ of a tolerance graph $G = (V, E)$, the representation $(\mathcal{P}, \mathcal{L})$ defines a partition of the vertex set V into the set V_B of bounded vertices and the set V_U of unbounded vertices. We denote $\mathcal{P} = \{p_1, p_2, \dots, p_{|\mathcal{P}|}\}$ and $\mathcal{L} = \{L_1, L_2, \dots, L_{|\mathcal{L}|}\}$, where $|\mathcal{P}| + |\mathcal{L}| = |V_U| + |V_B| = |V|$.

With a slight abuse of notation, for any two elements $x_1, x_2 \in \mathcal{P} \cup \mathcal{L}$, we may say in the following that x_1 is adjacent with x_2 (or x_1 is a neighbor of x_2) if the vertices that correspond to x_1 and x_2 are adjacent in the graph G . Moreover, whenever $\mathcal{P}_1 \subseteq \mathcal{P}_2 \subseteq \mathcal{P}$ and $\mathcal{L}_1 \subseteq \mathcal{L}_2 \subseteq \mathcal{L}$, we may say that the set $\mathcal{P}_1 \cup \mathcal{L}_1$ *dominates* $\mathcal{P}_2 \cup \mathcal{L}_2$ if the vertices corresponding to $\mathcal{P}_1 \cup \mathcal{L}_1$ dominate the subgraph of G induced by the vertices corresponding to $\mathcal{P}_2 \cup \mathcal{L}_2$.

BOUNDED DOMINATING SET on Tolerance Graphs

Input: A horizontal shadow representation $(\mathcal{P}, \mathcal{L})$ of a tolerance graph G .

Output: A set $Z \subseteq \mathcal{L}$ of minimum size that dominates $(\mathcal{P}, \mathcal{L})$, or the announcement that \mathcal{L} does not dominate $(\mathcal{P}, \mathcal{L})$.

5.1 Notation and Terminology

For an arbitrary point $t = (t_x, t_y) \in \mathbb{R}^2$ we define two (infinite) lines passing through t :

- the vertical line $\Gamma_t^{\text{vert}} = \{(t_x, s) \in \mathbb{R}^2 : s \in \mathbb{R}\}$, i.e., the line that is parallel to the y -axis,
- the diagonal line $\Gamma_t^{\text{diag}} = \{(s, s + (t_y - t_x)) \in \mathbb{R}^2 : s \in \mathbb{R}\}$, i.e., the line that is parallel to the main diagonal $\{(s, s) \in \mathbb{R}^2 : s \in \mathbb{R}\}$.

For every point $t = (t_x, t_y) \in \mathbb{R}^2$, each of the lines $\Gamma_t^{\text{vert}}, \Gamma_t^{\text{diag}}$ separates \mathbb{R}^2 into two regions. With respect to the line Γ_t^{vert} we define the regions $\mathbb{R}_{\text{left}}^2(\Gamma_t^{\text{vert}}) = \{(x, y) \in \mathbb{R}^2 : x \leq t_x\}$ and $\mathbb{R}_{\text{right}}^2(\Gamma_t^{\text{vert}}) = \{(x, y) \in \mathbb{R}^2 : x \geq t_x\}$ of the points to the left and to the right of Γ_t^{vert} , respectively. With respect to the line Γ_t^{diag} , we define the regions $\mathbb{R}_{\text{left}}^2(\Gamma_t^{\text{diag}}) = \{(x, y) \in \mathbb{R}^2 : y - x \geq t_y - t_x\}$ and $\mathbb{R}_{\text{right}}^2(\Gamma_t^{\text{diag}}) = \{(x, y) \in \mathbb{R}^2 : y - x \leq t_y - t_x\}$ of the points to the left and to the right of Γ_t^{diag} , respectively.

Furthermore, for an arbitrary point $t = (t_x, t_y) \in \mathbb{R}^2$ we define the region A_t (resp. B_t) that contains all points that are both to the right (resp. to the left) of Γ_t^{vert} and to the right (resp. to the left) of Γ_t^{diag} . That is, $A_t = \mathbb{R}_{\text{right}}^2(\Gamma_t^{\text{vert}}) \cap \mathbb{R}_{\text{right}}^2(\Gamma_t^{\text{diag}})$ and $B_t = \mathbb{R}_{\text{left}}^2(\Gamma_t^{\text{vert}}) \cap \mathbb{R}_{\text{left}}^2(\Gamma_t^{\text{diag}})$.

Consider an arbitrary horizontal line segment $L_i \in \mathcal{L}$. We denote by l_i and r_i its left and its right endpoint, respectively; note that possibly $l_i = r_i$. Denote by $\mathcal{A} = \{l_i, r_i : 1 \leq i \leq |\mathcal{L}|\}$ the set of all endpoints of all line segments of \mathcal{L} . Furthermore denote by $\mathcal{B} = \{\Gamma_t^{\text{diag}} \cap \Gamma_{t'}^{\text{vert}} : t, t' \in \mathcal{A}\}$ the set of all intersection points of the vertical and the diagonal lines that pass from points of \mathcal{A} . Note that $\mathcal{A} \subseteq \mathcal{B}$.

Given a horizontal shadow representation $(\mathcal{P}, \mathcal{L})$ we always assume that the points $p_1, p_2, \dots, p_{|\mathcal{P}|}$ are ordered increasingly with respect to their x -coordinates. Similarly we assume that the horizontal line segments $L_1, L_2, \dots, L_{|\mathcal{L}|}$ are ordered increasingly with respect to the x -coordinates of their endpoint r_i .

► **Definition 11.** Let $1 \leq i, i' \leq |\mathcal{L}|$. The pair (i, i') is a *right-crossing pair* if $r_{i'} \in S_{r_i}$. Furthermore the pair (i, i') is a *left-crossing pair* if $l_i \in S_{l_{i'}}$. For every right-crossing pair (i, i') , we define $\mathcal{L}_{i, i'}^{\text{left}} = \{x \in \mathcal{P} \cup \mathcal{L} : x \subseteq B_t, \text{ where } t = \Gamma_{r_i}^{\text{vert}} \cap \Gamma_{r_{i'}}^{\text{diag}}\}$ and for every left-crossing pair (i, i') we define $\mathcal{L}_{i, i'}^{\text{right}} = \{x \in \mathcal{P} \cup \mathcal{L} : x \subseteq A_t, \text{ where } t = \Gamma_{l_i}^{\text{vert}} \cap \Gamma_{l_{i'}}^{\text{diag}}\}$

► **Definition 12.** Let $S \subseteq \mathcal{P} \cup \mathcal{L}$ be an arbitrary set. Let (i, i') be a right-crossing pair and (j, j') be a left-crossing pair. If $L_i, L_{i'} \in S$ and $S \subseteq \mathcal{L}_{i, i'}^{\text{left}}$, then (i, i') is the *end-pair* of the set S . If $L_j, L_{j'} \in S$ and $S \subseteq \mathcal{L}_{j, j'}^{\text{right}}$, then (j, j') is the *start-pair* of the set S .

5.2 The Algorithm

In this section we present our algorithm for BOUNDED DOMINATING SET on tolerance graphs, cf. Algorithm 1. Given a horizontal shadow representation $(\mathcal{P}, \mathcal{L})$ of a tolerance graph G , we first add two dummy line segments L_0 and $L_{|\mathcal{L}|+1}$ (with endpoints l_0, r_0 and $l_{|\mathcal{L}|+1}, r_{|\mathcal{L}|+1}$, respectively) such that all elements of $\mathcal{P} \cup \mathcal{L}$ are contained in A_{r_0} and in $B_{l_{|\mathcal{L}|+1}}$. Let $\mathcal{L}' = \mathcal{L} \cup \{L_0, L_{|\mathcal{L}|+1}\}$. Note that $(\mathcal{P}, \mathcal{L}')$ is a horizontal shadow representation of some tolerance graph G' , where the bounded vertices V'_B of G' correspond to the line segments of \mathcal{L}' and the unbounded vertices V'_U of G' correspond to the points of \mathcal{P} . Furthermore note that $V'_B = V_B \cup \{v_0, v_{|\mathcal{L}|+1}\}$ and $V'_U = V_U$, where v_0 and $v_{|\mathcal{L}|+1}$ are the (isolated) bounded vertices of G' that correspond to the line segments L_0 and $L_{|\mathcal{L}|+1}$, respectively.

For simplicity of the presentation, we refer in the following to the augmented set \mathcal{L}' of horizontal line segments by \mathcal{L} . In the remainder of this section we will write $\mathcal{L} = \{L_1, L_2, \dots, L_{|\mathcal{L}|}\}$ with the understanding that the first and the last line segments L_1 and $L_{|\mathcal{L}|}$ of \mathcal{L} are dummy. Furthermore, we will refer to the augmented tolerance graph G' by G .

For every pair of points $(a, b) \in \mathcal{A} \times \mathcal{B}$ such that $b \in \mathbb{R}_{\text{right}}^2(\Gamma_a^{\text{diag}})$, define $X(a, b)$ to be the set of all points of \mathcal{P} and all line segments of \mathcal{L} that are contained in the region $B_b \setminus \Gamma_b^{\text{vert}}$ and to the right of the line Γ_a^{diag} , i.e., $X(a, b) = \{x \in \mathcal{P} \cup \mathcal{L} : x \subseteq (B_b \setminus \Gamma_b^{\text{vert}}) \cap \mathbb{R}_{\text{right}}^2(\Gamma_a^{\text{diag}})\}$.

► **Definition 13.** Let $(a, b) \in \mathcal{A} \times \mathcal{B}$ be a pair of points such that $b \in \mathbb{R}_{\text{right}}^2(\Gamma_a^{\text{diag}})$. Furthermore let (i, i') be a right-crossing pair such that $b \in \mathbb{R}_{\text{left}}^2(\Gamma_{r_i}^{\text{vert}})$. Then $BD_{(\mathcal{P}, \mathcal{L})}(a, b, i, i')$ is a dominating set $Z \subseteq \mathcal{L}$ of $X(a, b)$ with the smallest size, in which (i, i') is its end-pair. If such a dominating set $Z \subseteq \mathcal{L}$ of $X(a, b)$ does not exist, we define $BD_{(\mathcal{P}, \mathcal{L})}(a, b, i, i') = \perp$.

Note that always $L_i \in BD_{(\mathcal{P}, \mathcal{L})}(a, b, i, i')$. However, since $b \in \mathbb{R}_{\text{left}}^2(\Gamma_{r_i}^{\text{vert}})$ in Definition 13, it follows that $L_i \not\subseteq B_b \setminus \Gamma_b^{\text{vert}}$, and thus $L_i \notin X(a, b)$. For simplicity of the presentation we may refer to the set $BD_{(\mathcal{P}, \mathcal{L})}(a, b, i, i')$ as $BD_G(a, b, i, i')$, where $(\mathcal{P}, \mathcal{L})$ is the horizontal shadow representation of the tolerance graph G , or just as $BD(a, b, i, i')$ whenever the horizontal shadow representation $(\mathcal{P}, \mathcal{L})$ is clear from the context.

Algorithm 1 BOUNDED DOMINATING SET on Tolerance Graphs

Input: A horizontal shadow representation $(\mathcal{P}, \mathcal{L})$, where $\mathcal{P} = \{p_1, p_2, \dots, p_{|\mathcal{P}|}\}$ and $\mathcal{L} = \{L_1, L_2, \dots, L_{|\mathcal{L}|}\}$

Output: A set $Z \subseteq \mathcal{L}$ of minimum size that dominates $(\mathcal{P}, \mathcal{L})$, or the announcement that \mathcal{L} does not dominate $(\mathcal{P}, \mathcal{L})$

- 1: Add two dummy line segments L_0 and $L_{|\mathcal{L}|+1}$ completely to the left and to the right of $\mathcal{P} \cup \mathcal{L}$, respectively
- 2: $\mathcal{L} \leftarrow \mathcal{L} \cup \{L_0, L_{|\mathcal{L}|+1}\}$; denote $\mathcal{L} = \{L_1, L_2, \dots, L_{|\mathcal{L}|}\}$, where now $L_1, L_{|\mathcal{L}|}$ are dummy
- 3: $\mathcal{A} \leftarrow \{l_i, r_i : 1 \leq i \leq |\mathcal{L}|\}$; $\mathcal{B} \leftarrow \{\Gamma_t^{\text{diag}} \cap \Gamma_{t'}^{\text{vert}} : t, t' \in \mathcal{A}\}$
- 4: **for** every pair of points $(a, b) \in \mathcal{A} \times \mathcal{B}$ such that $b \in \mathbb{R}_{\text{right}}^2(\Gamma_a^{\text{diag}})$ **do**
- 5: $X(a, b) \leftarrow \{x \in \mathcal{P} \cup \mathcal{L} : x \subseteq (B_b \setminus \Gamma_b^{\text{vert}}) \cap \mathbb{R}_{\text{right}}^2(\Gamma_a^{\text{diag}})\}$
- 6: **for** every $i, i' \in \{1, 2, \dots, |\mathcal{L}|\}$ **do**
- 7: **if** $r_{i'} \in S_{r_i}$ **then** $\{(i, i') \text{ is a right-crossing pair}\}$
- 8: **if** $\{L_i\} \cup \{L_{i'}\}$ dominates all elements of $X(a, b)$ **then** $BD_{(\mathcal{P}, \mathcal{L})}(a, b, i, i') \leftarrow \{L_i\} \cup \{L_{i'}\}$ {initialization}
- 9: $\mathcal{L}_{i, i'}^{\text{left}} \leftarrow \{L_k \subseteq B_t : t = \Gamma_{r_i}^{\text{vert}} \cap \Gamma_{r_{i'}}^{\text{diag}}\}$
- 10: **if** $\mathcal{L} \cap \mathcal{L}_{i, i'}^{\text{left}}$ does not dominate all elements of $X(a, b)$ **then** $BD_{(\mathcal{P}, \mathcal{L})}(a, b, i, i') \leftarrow \perp$ {initialization}
- 11: **else** $BD_{(\mathcal{P}, \mathcal{L})}(a, b, i, i') \leftarrow \mathcal{L} \cap \mathcal{L}_{i, i'}^{\text{left}}$ {initialization}
- 12: **for** every pair of points $(a, b) \in \mathcal{A} \times \mathcal{B}$ such that $b \in \mathbb{R}_{\text{right}}^2(\Gamma_a^{\text{diag}})$ **do**
- 13: **for** every $i, i' \in \{1, 2, \dots, |\mathcal{L}|\}$ **do**
- 14: **if** $r_{i'} \in S_{r_i}$ **then** $\{(i, i') \text{ is a right-crossing pair}\}$
- 15: Compute $Z_1 = \{L_i\} \cup \min_{c, j, j'} \{BD_{(\mathcal{P}, \mathcal{L})}(a, c, j, j')\}$ by Lemma 14
- 16: **if** $|Z_1| < |BD_{(\mathcal{P}, \mathcal{L})}(a, b, i, i')|$ **then** $BD_{(\mathcal{P}, \mathcal{L})}(a, b, i, i') \leftarrow Z_1$
- 17: Compute $Z_2 = \min_c \{BD_{(\mathcal{P}, \mathcal{L})}(a, c, i, i') \cup BD_{(\mathcal{P}, \mathcal{L})}(c, b, i, i')\}$ by Lemma 15
- 18: **if** $|Z_2| < |BD_{(\mathcal{P}, \mathcal{L})}(a, b, i, i')|$ **then** $BD_{(\mathcal{P}, \mathcal{L})}(a, b, i, i') \leftarrow Z_2$
- 19: **if** $BD_{(\mathcal{P}, \mathcal{L})}(l_1, r_{\mathcal{L}}, |\mathcal{L}|, |\mathcal{L}|) = \perp$ **then return** \mathcal{L} does not dominate $(\mathcal{P}, \mathcal{L})$
- 20: **else return** $BD_{(\mathcal{P}, \mathcal{L})}(l_1, r_{\mathcal{L}}, |\mathcal{L}|, |\mathcal{L}|) \setminus \{L_1, L_{|\mathcal{L}|}\}$

► **Lemma 14.** Let $(a, b) \in \mathcal{A} \times \mathcal{B}$ and let (i, i') be a right-crossing pair such that $BD(a, b, i, i') \neq \perp$. If $BD(a, b, i, i') \setminus L_i$ dominates all elements of $\{x \in X(a, b) : x \cap (S_i \cup F_i) \neq \emptyset\}$ then $BD(a, b, i, i') = \{L_i\} \cup \min_{c, j, j'} \{BD(a, c, j, j')\}$, where the minimum is taken over all c, j, j' such that:

- $c = \Gamma_{r_j}^{\text{vert}} \cap \Gamma_b^{\text{diag}}$ if $r_j \in \mathbb{R}_{\text{left}}^2(\Gamma_b^{\text{vert}})$, and $c = b$ otherwise,
- (j, j') is a right-crossing pair of $\mathcal{L}_{i, i'}^{\text{left}} \setminus \{L_i\}$, where $j' = i'$ whenever $i \neq i'$, and
- $\{L_j\} \cup \{L_{j'}\}$ dominates all elements of the set $X(a, b) \setminus X(a, c)$.

► **Lemma 15.** Let $(a, b) \in \mathcal{A} \times \mathcal{B}$ and let (i, i') be a right-crossing pair such that $BD(a, b, i, i') \neq \perp$. If $BD(a, b, i, i') \setminus L_i$ does not dominate all elements of $\{x \in X(a, b) : x \cap (S_i \cup F_i) \neq \emptyset\}$ then $BD(a, b, i, i') = \min_c \{BD(a, c, i, i') \cup BD(c, b, i, i')\}$ where the minimum is taken over all c such that:

- $c \in \mathcal{B} \cap \mathbb{R}_{\text{right}}^2(\Gamma_{l_i}^{\text{vert}}) \cap \mathbb{R}_{\text{right}}^2(\Gamma_a^{\text{diag}}) \cap (B_b \setminus \Gamma_b^{\text{vert}})$ and
- $\mathcal{P} \cap X(a, b) \cap F_c \cap F_i = \emptyset$.

► **Theorem 16.** Given a horizontal shadow representation $(\mathcal{P}, \mathcal{L})$ of a tolerance graph G , Algorithm 1 computes BOUNDED DOMINATING SET in polynomial time.

Algorithm 2 RESTRICTED BOUNDED DOMINATING SET on Tolerance Graphs

Input: A 6-tuple $\mathcal{I} = (\mathcal{P}, \mathcal{L}, j, j', i, i')$, where $(\mathcal{P}, \mathcal{L})$ is a horizontal shadow representation of a tolerance graph G , (j, j') is a left-crossing pair of $(\mathcal{P}, \mathcal{L})$, and (i, i') is a right-crossing pair of $(\mathcal{P}, \mathcal{L})$.

Output: A set $Z \subseteq \mathcal{L}$ of minimum size that dominates $(\mathcal{P}, \mathcal{L})$, where (j, j') is the start-pair and (i, i') is the end-pair of Z , or the announcement that $\mathcal{L} \cap \mathcal{L}_{j,j'}^{\text{right}} \cap \mathcal{L}_{i,i'}^{\text{left}}$ does not dominate $(\mathcal{P}, \mathcal{L})$.

- 1: **if** $(\mathcal{P}, \mathcal{L})$ contains a bad point $p \in \mathcal{P}$ or a bad line segment $L_k \in \mathcal{L}$ (cf. Definition 17) **then**
- 2: **return** $\mathcal{L} \cap \mathcal{L}_{j,j'}^{\text{right}} \cap \mathcal{L}_{i,i'}^{\text{left}}$ does not dominate $(\mathcal{P}, \mathcal{L})$
- 3: Compute the set $Z_1 \subseteq \mathcal{L}$ of all irrelevant line segments (cf. Definition 17)
- 4: $\mathcal{L} \leftarrow \mathcal{L} \setminus Z_1$; $r \leftarrow \Gamma_{r_i}^{\text{vert}} \cap \Gamma_{r_{i'}}^{\text{diag}}$
- 5: Compute the representation $(\widehat{\mathcal{P}}, \widehat{\mathcal{L}})$ by adding the elements $\{x_{k,1}, x_{k,2} : k \in \{j, j'\}\}$ to $(\mathcal{P}, \mathcal{L})$ (cf. Lemma 18)
- 6: **return** $BD_{(\widehat{\mathcal{P}}, \widehat{\mathcal{L}})}(l_{x_{j,1}}, r, i, i')$ {by calling Algorithm 1}

6 Restricted Bounded Dominating Set on Tolerance Graphs

In this section we use Algorithm 1 of Section 5 to provide a polynomial time algorithm (cf. Algorithm 2) for a slightly modified version of BOUNDED DOMINATING SET on tolerance graphs, which we call RESTRICTED BOUNDED DOMINATING SET, formally defined below.

RESTRICTED BOUNDED DOMINATING SET on Tolerance Graphs

Input: A 6-tuple $\mathcal{I} = (\mathcal{P}, \mathcal{L}, j, j', i, i')$, where $(\mathcal{P}, \mathcal{L})$ is a horizontal shadow representation of a tolerance graph G , (j, j') is a left-crossing pair of G , and (i, i') is a right-crossing pair of G .

Output: A set $Z \subseteq \mathcal{L}$ of minimum size that dominates $(\mathcal{P}, \mathcal{L})$, where (j, j') is the start-pair and (i, i') is the end-pair of Z , or the announcement that $\mathcal{L} \cap \mathcal{L}_{j,j'}^{\text{right}} \cap \mathcal{L}_{i,i'}^{\text{left}}$ does not dominate $(\mathcal{P}, \mathcal{L})$.

► **Definition 17.** Let $\mathcal{I} = (\mathcal{P}, \mathcal{L}, j, j', i, i')$ be an instance of RESTRICTED BOUNDED DOMINATING SET on tolerance graphs. Let $l = \Gamma_{l_j}^{\text{vert}} \cap \Gamma_{l_{j'}}^{\text{diag}}$ and $r = \Gamma_{r_i}^{\text{vert}} \cap \Gamma_{r_{i'}}^{\text{diag}}$. A point $p \in \mathcal{P}$ is a *bad point* if $p \in B_l$ or $p \in \mathbb{R}_{\text{right}}^2(\Gamma_r^{\text{vert}})$. A line segment $L_t \in \mathcal{L}$ is a *bad line segment* if $L_t \subseteq B_l$ or $L_t \subseteq A_r$. Furthermore a line segment $L_t \in \mathcal{L}$ is an *irrelevant line segment* if either $L_t \subseteq \overline{B_l} \cap \overline{A_r}$ and $L_t \notin \mathcal{L}_{j,j'}^{\text{right}} \cap \mathcal{L}_{i,i'}^{\text{left}}$, or L_t has an endpoint in $B_l \cup A_r$ and another point in $\overline{B_l} \cap \overline{A_r}$.

The next lemma will enable us to reduce RESTRICTED BOUNDED DOMINATING SET to BOUNDED DOMINATING SET on tolerance graphs.

► **Lemma 18.** Let $\mathcal{I} = (\mathcal{P}, \mathcal{L}, j, j', i, i')$ be an instance of RESTRICTED BOUNDED DOMINATING SET on tolerance graphs, which has no bad points $p \in \mathcal{P}$ and no bad or irrelevant line segments $L \in \mathcal{L}$. Then for every $k \in \{j, j'\}$ we can add two new elements $x_{k,1}, x_{k,2}$ to the set $\mathcal{P} \cup \mathcal{L}$ such that L_k is the only neighbor of $x_{k,1}$ and $x_{k,2}$, $k \in \{j, j'\}$.

► **Definition 19.** Let (j, j') be a left-crossing pair and (i, i') be a right-crossing pair in the horizontal shadow representation $(\mathcal{P}, \mathcal{L})$. Then $RD_{(\mathcal{P}, \mathcal{L})}(j, j', i, i')$ is a dominating set

$Z \subseteq \mathcal{L} \cap \mathcal{L}_{j,j'}^{\text{right}} \cap \mathcal{L}_{i,i'}^{\text{left}}$ of $(\mathcal{P}, \mathcal{L})$ with the smallest size, in which (j, j') and (i, i') are the start-pair and the end-pair, respectively. If such a dominating set Z does not exist, we define $RD_{(\mathcal{P}, \mathcal{L})}(j, j', i, i') = \perp$.

For simplicity of the presentation we may refer to the set $RD_{(\mathcal{P}, \mathcal{L})}(j, j', i, i')$ as $RD_G(j, j', i, i')$, where $(\mathcal{P}, \mathcal{L})$ is the horizontal shadow representation of the tolerance graph G .

► **Theorem 20.** *Given a 6-tuple $\mathcal{I} = (\mathcal{P}, \mathcal{L}, j, j', i, i')$, where $(\mathcal{P}, \mathcal{L})$ is a horizontal shadow representation of a tolerance graph G , (j, j') is a left-crossing pair and (i, i') is a right-crossing pair of $(\mathcal{P}, \mathcal{L})$, Algorithm 2 computes RESTRICTED BOUNDED DOMINATING SET in polynomial time.*

7 Dominating Set on Tolerance Graphs

In this section we present our main algorithm of the paper (cf. Algorithm 3) which computes in polynomial time a minimum dominating set of a tolerance graph G , given by a horizontal shadow representation $(\mathcal{P}, \mathcal{L})$. Algorithm 3 uses Algorithms 1 and 2 as subroutines (cf. Sections 5 and 6). Throughout this section we assume without loss of generality that the given tolerance graph G is connected and that G is given with a *canonical* horizontal shadow representation $(\mathcal{P}, \mathcal{L})$.

For every $p \in \mathcal{P}$ we denote by $N(p) = \{L_k \in \mathcal{L} : p \in S_k\}$ and $H(p) = \{x \in \mathcal{P} \cup \mathcal{L} : x \cap S_p \neq \emptyset\}$. Note that, due to Lemmas 8 and 9, $N(p)$ is the set of neighbors of p and $H(p)$ is the set of hovering vertices of p . Furthermore, for every $L_k \in \mathcal{L}$ we denote by $N(L_k) = \{p \in \mathcal{P} : p \in S_k\} \cup \{L_t \in \mathcal{L} : L_t \cap S_k \neq \emptyset \text{ or } L_k \cap S_t \neq \emptyset\}$. Note that, due to Lemmas 7 and 8, $N(L_k)$ is the set of neighbors of L_k .

Define now the subset $\mathcal{P}^* = \{p \in \mathcal{P} : \text{there exists no point } p' \in \mathcal{P} \text{ such that } p \in H(p')\}$. Note by the definition of the set \mathcal{P}^* that for every $p_1, p_2 \in \mathcal{P}^*$ we have $p_1 \notin S_{p_2} \cup F_{p_2}$.

Given a canonical horizontal shadow representation $(\mathcal{P}, \mathcal{L})$, where $\mathcal{P} = \{p_1, p_2, \dots, p_{|\mathcal{P}|}\}$ and $\mathcal{L} = \{L_1, L_2, \dots, L_{|\mathcal{L}|}\}$, we add two dummy line segments L_0 and $L_{|\mathcal{L}|+1}$ (with endpoints l_0, r_0 and $l_{|\mathcal{L}|+1}, r_{|\mathcal{L}|+1}$, respectively) such that all elements of $\mathcal{P} \cup \mathcal{L}$ are contained in A_{r_0} and in $B_{l_{|\mathcal{L}|+1}}$. Denote $\mathcal{L}' = \mathcal{L} \cup \{L_0, L_{|\mathcal{L}|+1}\}$. Furthermore we add one dummy point $p_{|\mathcal{P}|+1}$ such that all elements of $\mathcal{P} \cup \mathcal{L}'$ are contained in $B_{p_{|\mathcal{P}|+1}}$. Denote $\mathcal{P}' = \mathcal{P} \cup \{p_{|\mathcal{P}|+1}\}$. For simplicity of the presentation, we refer in the following to the augmented sets \mathcal{P}' and \mathcal{L}' of points and horizontal line segments by \mathcal{P} and \mathcal{L} , respectively. In the remainder of this section we will write $\mathcal{P} = \{p_1, p_2, \dots, p_{|\mathcal{P}|}\}$ and $\mathcal{L} = \{L_1, L_2, \dots, L_{|\mathcal{L}|}\}$ with the understanding that the last point $p_{|\mathcal{P}|}$ of \mathcal{P} , as well as the first and the last line segments L_1 and $L_{|\mathcal{L}|}$ of \mathcal{L} , are dummy. Note that the last point $p_{|\mathcal{P}|}$ (i.e., the new dummy point) belongs to the set \mathcal{P}^* .

For every $p_i, p_j \in \mathcal{P}^*$ with $i < j$, we define $G_j = \{x \in \mathcal{P} \cup \mathcal{L} : x \subseteq B_{p_j} \setminus \Gamma_{p_j}^{\text{vert}}\}$ and $G(i, j) = \{x \in G_j : x \subseteq A_{p_i}\}$. Note that $p_j \notin G_j$ and $p_j \notin G(i, j)$.

► **Definition 21.** Let $p_j \in \mathcal{P}^*$ and (i, i') be a right-crossing pair in G_j . Then $D(j, i, i')$ is a *minimum dominating set* of G_j whose end-pair is (i, i') . If there exists no dominating set Z of G_j whose end-pair is (i, i') , we define $D(j, i, i') = \perp$.

► **Lemma 22.** *Let G be a tolerance graph, $(\mathcal{P}, \mathcal{L})$ be a canonical representation of G , $p_j \in \mathcal{P}^*$, and a right-crossing pair (i, i') of G_j such that $D(j, i, i') \neq \perp$. Then*

$$D(j, i, i') = \min_{q', z, z', w, w'} \begin{cases} D(q, z, z') \cup \{p_k \in \mathcal{P}^* : q \leq k \leq q'\} \cup RD_{G(q', j)}(w, w', i, i') \\ BD_{G_j}(l_1, b, i, i'), \text{ where } b = \Gamma_{r_i}^{\text{vert}} \cap \Gamma_{r_{i'}}^{\text{diag}} \end{cases}$$

where the minimum is taken over all q', z, z', w, w' such that:

Algorithm 3 DOMINATING SET on Tolerance Graphs

Input: A canonical horizontal shadow representation $(\mathcal{P}, \mathcal{L})$, where $\mathcal{P} = \{p_1, p_2, \dots, p_{|\mathcal{P}|}\}$ and $\mathcal{L} = \{L_1, L_2, \dots, L_{|\mathcal{L}|}\}$.

Output: A set $D \subseteq \mathcal{L} \cup \mathcal{P}$ of minimum size that dominates $(\mathcal{P}, \mathcal{L})$.

- 1: Add two dummy line segments L_0 and $L_{|\mathcal{L}|+1}$ completely to the left and to the right of $\mathcal{P} \cup \mathcal{L}$, respectively
- 2: Add a dummy point $p_{|\mathcal{P}|+1}$ completely to the right of $L_{|\mathcal{L}|+1}$
- 3: $\mathcal{P} \leftarrow \mathcal{P} \cup \{p_{|\mathcal{P}|+1}\}$; $\mathcal{L} \leftarrow \mathcal{L} \cup \{L_0, L_{|\mathcal{L}|+1}\}$
- 4: Denote $\mathcal{P} = \{p_1, p_2, \dots, p_{|\mathcal{P}|}\}$ and $\mathcal{L} = \{L_1, L_2, \dots, L_{|\mathcal{L}|}\}$, where now $p_{|\mathcal{P}|}$, L_1 , and $L_{|\mathcal{L}|}$ are dummy
- 5: $\mathcal{P}^* \leftarrow \{p \in \mathcal{P} : \text{there exists no point } p' \in \mathcal{P} \text{ such that } p \in H(p')\}$
- 6: **for** every pair of points $(a, b) \in \mathcal{A} \times \mathcal{B}$ such that $b \in \mathbb{R}_{\text{right}}^2(\Gamma_a^{\text{diag}})$ **do**
- 7: $X(a, b) \leftarrow \{x \in \mathcal{P} \cup \mathcal{L} : x \subseteq (B_b \setminus \Gamma_b^{\text{vert}}) \cap \mathbb{R}_{\text{right}}^2(\Gamma_a^{\text{diag}})\}$
- 8: **for** every $p_j \in \mathcal{P}^*$ **do**
- 9: $G_j \leftarrow \{x \in \mathcal{P} \cup \mathcal{L} : x \subseteq B_{p_j} \setminus \Gamma_{p_j}^{\text{vert}}\}$
- 10: **for** every $i, i' \in \{1, 2, \dots, |\mathcal{L}|\}$ **do**
- 11: **if** $L_i, L_{i'} \in G_j$ and $r_{i'} \in S_{r_i}$ **then** $\{(i, i') \text{ is a right-crossing pair of } G_j\}$
- 12: **if** $X(r_{i'}, p_j)$ is not dominated by L_i and $L_{i'}$ **then** $D(j, i, i') \leftarrow \perp$
- 13: **if** there exists a point $p \in \mathcal{P} \cap G_j$ such that $p \in \mathbb{R}_{\text{right}}^2(\Gamma_{r_i}^{\text{vert}})$ **then** $D(j, i, i') \leftarrow \perp$
- 14: **if** $D(j, i, i') \neq \perp$ **then**
- 15: Compute $D(j, i, i')$ by Lemma 22 {by calling Algorithms 1 and 2}
- 16: **return** $D(|\mathcal{P}|, |\mathcal{L}|, |\mathcal{L}|) \setminus \{L_1, L_{|\mathcal{L}|}\}$

- $1 \leq q' < j$,
- $i, i' \notin N(p_{q'}) \cup H(p_{q'})$,
- (w, w') is a left-crossing pair of $G(q', j)$ such that $RD_{G(q', j)}(w, w', i, i') \neq \perp$,
- (z, z') is a right-crossing pair of $G_{q'}$,
- $q = \min\{1 \leq k \leq q' : p_k \in \mathcal{P}^*, p_k \in A_\omega, \text{ where } \omega = \Gamma_{r_z}^{\text{vert}} \cap \Gamma_{r_{z'}}^{\text{diag}}\}$,
- $D(q, z, z') \neq \perp$,
- $(H(p_q) \cup H(p_{q'})) \setminus \left(\bigcup_{q \leq k \leq q'} N(p_k)\right)$ are dominated by the line segments $L_z, L_{z'}, L_w, L_{w'}$,
- $G(q, q')$ is dominated by $\{p_k \in \mathcal{P}^* : q \leq k \leq q'\}$.

► **Theorem 23.** Given a canonical horizontal shadow representation $(\mathcal{P}, \mathcal{L})$ of a connected tolerance graph G , Algorithm 3 computes in polynomial time a minimum dominating set of G .

8 Independent Dominating Set on Multitolerance Graphs

In this section we provide a polynomial time sweep-line algorithm which, given a shadow representation $(\mathcal{P}, \mathcal{L})$ of a multitolerance graph G , computes in polynomial time a minimum independent dominating set of G .

► **Theorem 24.** Given a shadow representation $(\mathcal{P}, \mathcal{L})$ of a multitolerance graph G , we can compute a minimum independent dominating set in polynomial time.

Acknowledgements The second author wishes to thank Steven Chaplick for insightful initial discussions and for suggesting the paper [5].

References

- 1 S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215(3):403–410, 1990.
- 2 Kenneth P. Bogart, Peter C. Fishburn, Garth Isaak, and Larry Langley. Proper and unit tolerance graphs. *Discrete Applied Mathematics*, 60(1-3):99–117, 1995.
- 3 Kellogg S. Booth and J. Howard Johnson. Dominating sets in chordal graphs. *SIAM Journal on Computing*, 11(1):191–199, 1982.
- 4 Arthur H. Busch. A characterization of triangle-free tolerance graphs. *Discrete Applied Mathematics*, 154(3):471–477, 2006.
- 5 Timothy M. Chan and Elyot Grant. Exact algorithms and APX-hardness results for geometric packing and covering problems. *Computational Geometry*, 47(2):112–124, 2014.
- 6 Jitender S. Deogun and George Steiner. Polynomial algorithms for hamiltonian cycle in cocomparability graphs. *SIAM Journal on Computing*, 23(3):520–552, 1994.
- 7 Stefan Felsner. Tolerance graphs and orders. *Journal of Graph Theory*, 28(3):129–140, 1998.
- 8 M. C. Golumbic and C. L. Monma. A generalization of interval graphs with tolerances. In *Proceedings of the 13th Southeastern Conference on Combinatorics, Graph Theory and Computing, Congressus Numerantium 35*, pages 321–331, 1982.
- 9 M. C. Golumbic, C. L. Monma, and W. T. Trotter. Tolerance graphs. *Discrete Applied Mathematics*, 9(2):157–170, 1984.
- 10 M. C. Golumbic and A. Siani. Coloring algorithms for tolerance graphs: reasoning and scheduling with interval constraints. In *Proceedings of the Joint International Conferences on Artificial Intelligence, Automated Reasoning, and Symbolic Computation (AISC/Calculus)*, pages 196–207, 2002.
- 11 M. C. Golumbic and A. N. Trenk. *Tolerance Graphs*. Cambridge studies in advanced mathematics, 2004.
- 12 Michael Kaufmann, Jan Kratochvil, Katharina A. Lehmann, and Amarendran R. Subramanian. Max-tolerance graphs as intersection graphs: cliques, cycles, and recognition. In *Proceedings of the 17th annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 832–841, 2006.
- 13 Dieter Kratsch and Lorna Stewart. Domination on cocomparability graphs. *SIAM Journal on Discrete Mathematics*, 6(3):400–417, 1993.
- 14 Katharina Anna Lehmann, Michael Kaufmann, Stephan Steigele, and Kay Nieselt. On the maximal cliques in c -max-tolerance graphs and their application in clustering molecular sequences. *Algorithms for Molecular Biology*, 1, 2006.
- 15 George B. Mertzios. An intersection model for multitolerance graphs: Efficient algorithms and hierarchy. *Algorithmica*, 69(3):540–581, 2014.
- 16 George B. Mertzios, Ignasi Sau, and Shmuel Zaks. A new intersection model and improved algorithms for tolerance graphs. *SIAM Journal on Discrete Mathematics*, 23(4):1800–1813, 2009.
- 17 George B. Mertzios, Ignasi Sau, and Shmuel Zaks. The recognition of tolerance and bounded tolerance graphs. *SIAM Journal on Computing*, 40(5):1234–1257, 2011.
- 18 Haiko Müller. Hamiltonian circuits in chordal bipartite graphs. *Discrete Mathematics*, 156(1-3):291–298, 1996.
- 19 Andreas Parra. Triangulating multitolerance graphs. *Discrete Applied Mathematics*, 84(1-3):183–197, 1998.
- 20 Jeremy P. Spinrad. *Efficient graph representations*, volume 19 of *Fields Institute Monographs*. American Mathematical Society, 2003.
- 21 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, New York, NY, USA, 2011.

Comparing 1D and 2D Real Time on Cellular Automata

Anaël Grandjean and Victor Poupet

LIRMM, Université Montpellier 2
161 rue Ada, 34392 Montpellier, France

Abstract

We study the influence of the dimension of cellular automata (CA) for real time language recognition of one-dimensional languages with parallel input. Specifically, we focus on the question of determining whether every language that can be recognized in real time on a 2-dimensional CA working on the Moore neighborhood can also be recognized in real time by a 1-dimensional CA working on the standard two-way neighborhood. We show that 2-dimensional CA in real time can perform a linear number of simulations of a 1-dimensional real time CA. If the two classes are equal then the number of simulated instances can be polynomial.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Cellular automata, real time, language recognition

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.367

1 Introduction

Cellular automata (CA) were first introduced in the 1940s (published posthumously in 1966) by J. von Neumann and S. Ulam as a mathematical model to study self-replication [7]. Although initially studied as a dynamical system, A.R. Smith III proved that it was possible to embed Turing machines in their behavior [5] and were as such a convenient model for massively parallel computation.

Cellular automata are also well suited to work on various dimensions. The original CA by von Neumann is 2-dimensional, but the natural simulation of Turing machines is on one dimension. CA represent therefore a natural way to study how the dimension of the space affects the computing power of the machines [1, 6].

This article presents some results comparing the computational power of 1-dimensional and 2-dimensional CA on parallel input. The main open question in that respect is to determine whether or not all languages that can be recognized in real time on 2-dimensional CA can also be recognized in real time in 1 dimension [2].

The organization of the article is as follows. Section 2 recalls the basic definitions and concepts about cellular automata that are used throughout the article. Section 3 presents a construction on 1-dimensional CA that shows how it is possible to consider that a real time CA knows approximately where the middle (or any other fixed rational proportion) of the input word is from the start. This construction is used to prove the main theorem of Section 5. In Section 4 we present a classic technique on cellular automata that compresses the space-time diagram of a 1-dimensional CA. The main novelty here is that we perform the compression not on the middle of the input word but on an approximate position “near the center”. Section 5 presents and proves the main result of the article, which states in essence that 2-dimensional CA can simulate in real time a linear number of simulations of a 1-dimensional real time CA. Finally Section 6 discusses some consequences of the main theorems.



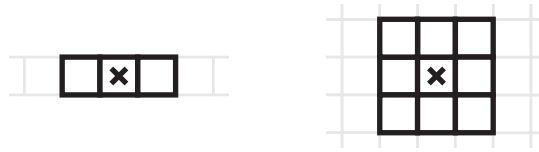
© Anaël Grandjean and Victor Poupet;
licensed under Creative Commons License CC-BY
32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).
Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 367–378



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



SYMPOSIUM
ON THEORETICAL
ASPECTS
OF COMPUTER
SCIENCE



■ **Figure 1** The standard 1-dimensional neighborhood (left) and the Moore 2-dimensional neighborhood (right).

2 Definitions

2.1 Cellular Automata

► **Definition 1** (Cellular Automaton). A *cellular automaton* (CA) is a quadruple $\mathcal{A} = (d, Q, \mathcal{N}, \delta)$ where:

- $d \in \mathbb{N}$ is the dimension of \mathcal{A} ;
- Q is a finite set whose elements are called *states*;
- $\mathcal{N} \subset \mathbb{Z}^d$ is a finite set called *neighborhood* of \mathcal{A} such that $0 \in \mathcal{N}$;
- $\delta : Q^{\mathcal{N}} \rightarrow Q$ is the local transition function of \mathcal{A} .

A *configuration* of the automaton is a mapping $\mathfrak{C} : \mathbb{Z}^d \rightarrow Q$. The elements of \mathbb{Z}^d are called *cells* and for a given cell $c \in \mathbb{Z}^d$, we say that $\mathfrak{C}(c)$ is the state of c in the configuration \mathfrak{C} . The set of all configurations over Q is denoted $\text{Conf}(Q)$. For a given configuration $\mathfrak{C} \in \text{Conf}(Q)$ and a cell $c \in \mathbb{Z}^d$, define the *neighborhood of c in \mathfrak{C}*

$$\mathcal{N}_{\mathfrak{C}}(c) = \begin{cases} \mathcal{N} & \rightarrow Q \\ n & \mapsto \mathfrak{C}(c+n) \end{cases}$$

From the local transition function δ , we define the *global transition function* $\Delta_{\mathcal{A}}$ of the automaton. The image of a configuration \mathfrak{C} by $\Delta_{\mathcal{A}}$ is obtained by replacing the state of each cell c by the image by δ of the neighborhood of c in \mathfrak{C} :

$$\Delta_{\mathcal{A}} : \begin{cases} \text{Conf}(Q) & \rightarrow \text{Conf}(Q) \\ \mathfrak{C} & \mapsto \begin{cases} \mathbb{Z}^d & \rightarrow Q \\ c & \mapsto \delta(\mathcal{N}_{\mathfrak{C}}(c)) \end{cases} \end{cases}$$

In this article, we will only consider 1-dimensional CA working on the standard neighborhood $\mathcal{N}_{\text{std}} = \{-1, 0, 1\}$ and 2-dimensional cellular automata working on the Moore neighborhood $\mathcal{N}_{\text{M}} = \{(x, y) \mid -1 \leq x, y \leq 1\}$ (see Figure 1).

2.2 Language Recognition

► **Definition 2** (Language Recognizer). Given a finite alphabet Σ and a language $L \subseteq \Sigma^*$, a d -dimensional CA \mathcal{A} with states Q is said to recognize L in time $f : \mathbb{N} \rightarrow \mathbb{N}$ with accepting states $Q_a \subseteq Q$ and quiescent state $q_0 \in Q$ if, $\Sigma \subseteq Q$ and for any word $w = u_0 u_1 \dots u_{n-1} \in \Sigma^*$, starting from the configuration

$$\begin{aligned} \mathbb{Z}^d & \rightarrow Q \\ (x, y_1, y_2, \dots, y_{d-1}) & \mapsto \begin{cases} u_x & \text{if } x \in \llbracket 0, n-1 \rrbracket \text{ and } \forall i, y_i = 0 \\ q_0 & \text{otherwise} \end{cases} \end{aligned}$$

the state of the origin at time $f(n)$ is in Q_a if and only if $w \in L$.

► **Definition 3** (Real and Linear Time). The *real time* function is the function $n \mapsto n - 1$. This time function corresponds to the minimal time necessary for information held on the last letter of the input word to reach the origin and hence affect the recognition of the word. The class of languages recognized in real time on 1 dimensional (resp. 2-dimensional) CA will be denoted $CA(n)$ (resp. $CA_2(n)$).

We will say that a language is recognized in *linear time* if it can be recognized in time $n \mapsto 2n$. The class of languages recognized in linear time on 1-dimensional (resp. 2 dimensional) CA will be denoted $CA(2n)$ (resp. $CA_2(2n)$).

Because there are linear acceleration theorems on 1-dimensional and 2-dimensional CA [4] (on the simple neighborhoods that we consider), any language recognized in time $n \mapsto kn$ for $k > 0$ is also recognized in time $n \mapsto 2n$, which explains the denomination of *linear time*. As for real time, it is a long open question to determine whether $CA(n) = CA(2n)$.

In this article, we investigate whether adding a dimension to the automaton increases linear and real time recognition power, namely if $CA(n) = CA_2(n)$ and $CA(2n) = CA_2(2n)$. These questions are long open problems (see problem 26 in [2]).

2.3 Tools

2.3.1 Space-Time Diagram

A space-time diagram is a 2-dimensional representation of the evolution of a 1-dimensional CA from a specific configuration. Each configuration in the evolution is represented by a line of the diagram, with time going from bottom to top. We do not usually consider space-time diagrams of 2-dimensional CA as these would be in 3 dimensions.

A specific point in space and time will be referred to as a *site* of the space-time diagram.

2.3.2 Layers

Given a CA $\mathcal{A} = (d, Q, \mathcal{N}, \delta)$, *adding a layer* to \mathcal{A} that performs a certain task consists in designing a specific CA working on a set of states Q' that performs the task and extending the set of states of \mathcal{A} to the product $Q \times Q'$. In doing so, the new product automaton can mimic the behavior of \mathcal{A} on its first coordinate and perform the new task on the second coordinate. From there, it is possible to modify the behavior of the automaton by having the two layers interact with each other.

As long as each layer requires only a finite number of states and there are only a finite number of layers, the total number of states of the resulting automaton remains finite.

3 Markers

In this section we investigate whether “marking” specific positions on the input word can help real time recognition of a language. The results in this section are a generalization of a technique used by O. Ibarra and T. Jiang in their proof that if $CA(n)$ is closed under reversal then $CA(n) = CA(2n)$ [3].

Given a finite alphabet Σ , we mark some letters of words of Σ^* by considering the extended alphabet $\Sigma \times \{0, 1\}$. We say that the word $(u_i, \delta_i)_{i \in \llbracket 0, n-1 \rrbracket} \in (\Sigma \times \{0, 1\})^*$ corresponds to the word $(u_i)_{i \in \llbracket 0, n-1 \rrbracket}$ where all the positions i such that $\delta_i = 1$ have been marked ($\delta_i = 0$ means that the letter has not been marked).

3.1 Exact and Fuzzy Marking

► **Definition 4** (Proportional Marking). Given $\alpha \in [0, 1[$ and a language L over an alphabet Σ , we define $L^{[\alpha]} \in (\Sigma \times \{0, 1\})^*$ as the language of words of L for which only the letter at position $\lfloor \alpha n \rfloor$ has been marked (n is the length of the word).

Formally, the word $(u_i, \delta_i)_{i \in \llbracket 0, n-1 \rrbracket} \in (\Sigma \times \{0, 1\})^*$ is in $L^{[\alpha]}$ if and only if $u_0 u_1 \dots u_{n-1} \in L$, $\delta_{\lfloor \alpha n \rfloor} = 1$ and for all other i , $\delta_i = 0$.

When working on real time CA algorithms it would sometimes be convenient to know where the middle of the word is, or some other specific ratio. Whether marking the letter of the input word corresponding to a fixed proportion of the word length can help recognize in real time languages that were not in $CA(n)$ is still an open question to our knowledge¹. Although an answer to this question would be very interesting, we can actually make many constructions with a weaker version that we can prove : instead of requiring a mark on the exact cell at position $\lfloor \alpha n \rfloor$, it is enough to have a mark on one of the cells between positions $\lfloor \alpha n \rfloor$ and $\lfloor \beta n \rfloor$ for $\alpha < \beta$.

► **Definition 5** (Fuzzy Marking). Given $\alpha, \beta \in [0, 1[$ with $\alpha \leq \beta$ and a language L over an alphabet Σ , we define $L^{[\alpha, \beta]} \in (\Sigma \times \{0, 1\})^*$ as the language of words of L for which exactly one letter between position $\lfloor \alpha n \rfloor$ and $\lfloor \beta n \rfloor$ has been marked (n is the length of the word).

Formally, the word $(u_i, \delta_i)_{i \in \llbracket 0, n-1 \rrbracket} \in (\Sigma \times \{0, 1\})^*$ is in $L^{[\alpha, \beta]}$ if and only if $u_0 u_1 \dots u_{n-1} \in L$, there is exactly one i_0 such that $\delta_{i_0} = 1$ and $i_0 \in \llbracket \lfloor \alpha n \rfloor, \lfloor \beta n \rfloor \rrbracket$.

The rest of this section will be devoted to the proof of the following theorem:

► **Theorem 6.** *For any language L and any $\alpha, \beta \in [0, 1]$ with $\alpha < \beta$,*

$$L^{[\alpha, \beta]} \in CA(n) \Rightarrow L \in CA(n)$$

First, notice that it is sufficient to prove the theorem for α and β rationals with $0 < \alpha < \beta < 1$. Indeed, for any $\alpha, \beta \in [0, 1]$ and $\alpha', \beta' \in \mathbb{Q}$ such that $\alpha \leq \alpha' < \beta' \leq \beta$ if $L^{[\alpha, \beta]} \in CA(n)$ then we can recognize $L^{[\alpha', \beta']}$ in real time by simulating the automaton \mathcal{A} that recognizes $L^{[\alpha, \beta]}$ in real time while simultaneously verifying that the marker is placed between $\lfloor \alpha' n \rfloor$ and $\lfloor \beta' n \rfloor$, which can be done in real time because α' and β' are both rationals. If the marker is in the right range then the input word is in $L^{[\alpha', \beta']}$ if and only if \mathcal{A} accepts it. If the marker is not in the right range, then the word is not accepted. From now on, we can therefore assume that $\alpha, \beta \in \mathbb{Q}$ and $0 < \alpha < \beta < 1$.

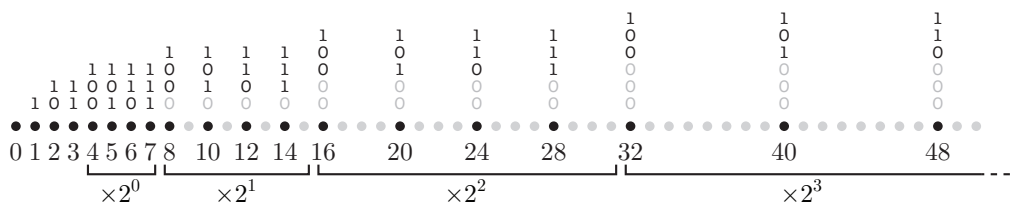
To prove the theorem, we assume that we have a CA \mathcal{A} that recognizes the language $L^{[\alpha, \beta]}$ in real time, for some $\alpha, \beta \in [0, 1]$, with $\alpha < \beta$. We will show how to make a CA \mathcal{A}' that recognizes L in real time.

The construction will be done in two steps. First we describe a CA that starts with a fixed set of positions marked (independently of the input length) and we show that with these markers we can recognize L in real time. Then we transform this CA into one that does not need the positions to be marked ahead of time.

3.2 Universal Markers

Let n_0 be the smallest integer such that $1 + \frac{1}{2^{n_0}} < \frac{\beta}{\alpha}$ and consider the set M of integers whose binary representation is such that all the digits 1 are on the $(n_0 + 1)$ most significant

¹ Note that if $CA(n) = CA(2n)$ then proportional marking with a rational ratio does not help real time recognition since the cell at position $\lfloor \alpha n \rfloor$ can be marked in n time steps for any rational α .



■ **Figure 2** The set M for $n_0 = 2$.

bits (see Figure 2):

$$M = \{x \times 2^k \mid x \in \llbracket 0, 2^{n_0+1} - 1 \rrbracket, k \in \mathbb{N}\} = \llbracket 0, 2^{n_0} - 1 \rrbracket \cup \{x \times 2^k \mid x \in \llbracket 2^{n_0}, 2^{n_0+1} - 1 \rrbracket, k \in \mathbb{N}\}$$

The set M contains an initial segment $\llbracket 0, 2^{n_0} \llbracket$ and copies of $\llbracket 2^{n_0}, 2^{n_0+1} \llbracket$ multiplied by the powers of 2 (indicated as bracketed “blocks” in Figure 2).

Let us now consider the ratio between consecutive elements of M . Denote by $(m_i)_{i \in \mathbb{N}}$ the elements of M in increasing order. For all $m_i \geq 2^{n_0}$, we have

$$1 + \frac{1}{2^{n_0+1} - 1} \leq \frac{m_{i+1}}{m_i} \leq 1 + \frac{1}{2^{n_0}} \tag{1}$$

The lower bound corresponds to the ratio between the last element of a block and the first of the next block (in the example with $n_0 = 2$, this ratio is $\frac{8}{7}$) and the upper bound corresponds to the ratio between the two first elements of a block (in the example it is $\frac{5}{4}$).

From the definition of n_0 , we have $\forall m_i \geq 2^{n_0}, \frac{m_{i+1}}{m_i} < \frac{\beta}{\alpha}$ and $\frac{m_{i+1}}{\beta} < \frac{m_i}{\alpha}$ so intervals $[\frac{m_i}{\beta}, \frac{m_i}{\alpha}]$ and $[\frac{m_{i+1}}{\beta}, \frac{m_{i+1}}{\alpha}]$ overlap, and hence $[2^{n_0}, +\infty[\subseteq \bigcup_{m \in M} [\frac{m}{\beta}, \frac{m}{\alpha}]$.

From this, we get $\forall x \geq 2^{n_0}, \exists m \in M, \lfloor \alpha x \rfloor \leq m \leq \lfloor \beta x \rfloor$.

Since M contains all the elements in the the missing initial segment $\llbracket 0, 2^{n_0} \rrbracket$, we have proved the following lemma:

► **Lemma 7.** $\forall n \in \mathbb{N}, \exists m \in M, \lfloor \alpha n \rfloor \leq m \leq \lfloor \beta n \rfloor$

We now know that for any input word w of length n , at least one of the elements of M lies between $\lfloor \alpha n \rfloor$ and $\lfloor \beta n \rfloor$ and can therefore be used by \mathcal{A} as a marker to know whether w is in L .

Moreover, from Equation (1), we have $\forall i, k \in \mathbb{N}$,

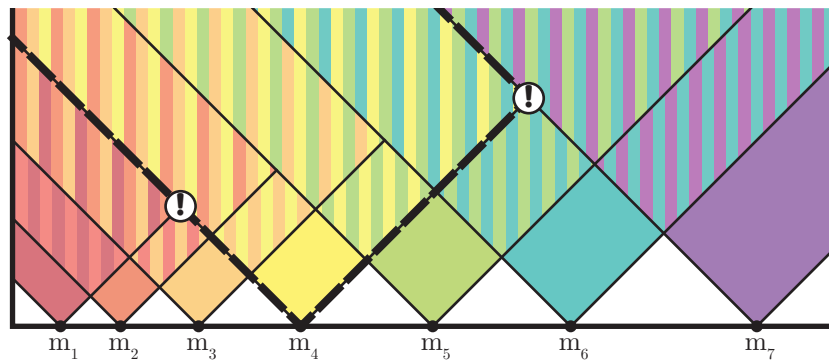
$$m_i \left(1 + \frac{1}{2^{n_0+1} - 1}\right)^k \leq m_{i+k} \tag{2}$$

Define k_0 as the smallest integer such that

$$\frac{1}{\alpha} \leq \left(1 + \frac{1}{2^{n_0+1} - 1}\right)^{k_0} \tag{3}$$

Equations (2) and (3) state that for any i , if $m_{i+k_0+1} \leq n$ then $m_{i+1} \leq \lfloor \alpha n \rfloor$ and thus $m_i < \lfloor \alpha n \rfloor$, which means that if a word is long enough to have a letter on m_{i+k_0+1} , then m_i is out of the range of valid markers. As a consequence, it is never necessary to consider more than $(k_0 + 1)$ elements of M at any given time.

We can now describe the first part of the construction. We assume that the automaton is given as input a word w of Σ^* on which all letters at indexes in M are marked. On such an



■ **Figure 3** Space-time diagram of the simulation. Each cone represents the area on which a simulation corresponding to an element of M is done. The cones extend to the left until the origin, but are interrupted to the right when there are too many simulations running at once.

input the automaton simulates the behavior of \mathcal{A} on w (as if no letter was marked), but each marked cell also starts a separate simulation of \mathcal{A} that considers that the letter is the only marked letter of the input. However, as previously observed, only the $(k_0 + 1)$ simulations corresponding to the largest elements of M are significant, all others correspond to markers at positions before the required range. This means that each cell only needs to simulate at most $(k_0 + 1)$ computations of \mathcal{A} and whenever a new computation should be taken into account, the one corresponding to the lowest element of m is discarded.

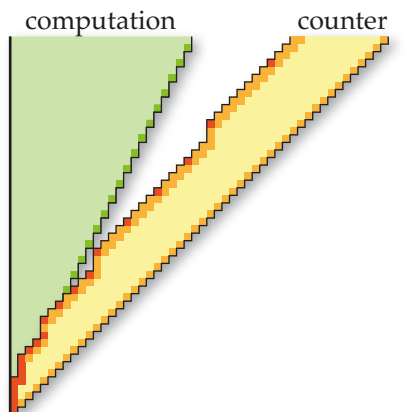
This behavior is illustrated by Figure 3. This figure represents a space-time diagram of the automaton. Each cone corresponds to a simulation of \mathcal{A} for which the marked cell is the origin of the cone. The figure corresponds to a case where $k_0 = 2$, meaning that at most 3 simulations are performed in parallel by each cell. The thick dashed line illustrates the area of the space-time diagram on which the simulation corresponding to the marker on m_4 is performed. Since this specific simulation starts on the cell m_4 , all space-time sites outside of the cone starting from that cell are not performing this specific simulation (they are however simulating the behavior of \mathcal{A} without any marker, which coincides with the behavior with a marker on m_4 on said sites out of the cone). As time passes, more and more cells are included in this cone and start performing this specific simulation. The two sites indicated by $\textcircled{!}$ correspond to events where a cell enters a fourth cone. Instead of starting a fourth simulation, it discards the simulation corresponding to the lowest m_i : on the left, the simulation for a marker at m_1 is discarded, on the right it's the simulation corresponding to the marker at m_4 that is discontinued.

In each simulation of \mathcal{A} , the automaton checks that the marker is located between $\lfloor \alpha n \rfloor$ and $\lfloor \beta n \rfloor$ by sending two signals from the marker towards the origin, one at speed α and the other at speed β (it is possible if α and β are rationals). If the marker is in the correct range, the signal moving at speed β will arrive before real time while the one moving at speed α will arrive after real time.

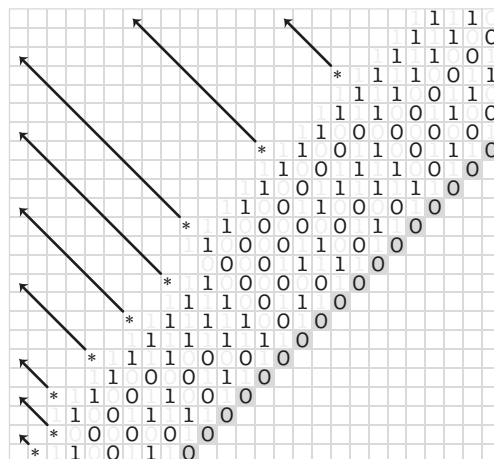
From Lemma 7, we know that there is a marker m in the correct range and since \mathcal{A} properly recognizes $L^{[\alpha, \beta]}$, the simulation of \mathcal{A} for the marker m will let the automaton know whether w is in L or not.

3.3 Construction of M

We now have to remove the requirement that the elements of M be marked on the input. To do this, we use a space-time compression technique: instead of starting the computation



■ **Figure 4** The diagonal counter is added on the compressed space-time diagram. Because the diagonal indexes are of logarithmic length, the counter and the main computation overlap only on a finite number of cells. Diagonals of indexes in M (here with $n_0 = 2$) are represented with a darker square.



■ **Figure 5** Detailed behavior of the diagonal counter. Least significant bits are on the bottom right. For better readability the digits of indexes on odd diagonals are represented in light grey. Indexes in M are marked with a (*) (here M is defined with $n_0 = 2$) and the signal sent towards the main computation is represented by an arrow.

immediately, the automaton moves the states from the input word towards the origin to group them three by three, and only then simulates the behavior of the original (uncompressed) CA. By performing such a compression, the initial configuration is mapped to the space-time line of slope 2, and the computation takes place in the cone between this line and the vertical axis (as illustrated by the green cone in Figure 4 in which each dark green cell on the right border holds 3 states from the initial configuration). Although the space-time diagram is strongly modified by the compression, the computation of the states on the origin cell does not suffer any slow down.

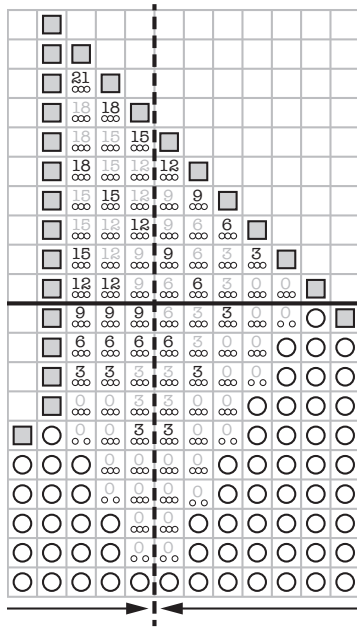
To mark the elements of M for the compressed computation, the CA builds a binary counter on the main diagonal of the space-time diagram to obtain the index of each transverse diagonal (see Figures 4 and 5).

Since it is easy to recognize binary representations of elements in M (all bits but the $(n_0 + 1)$ most significant must be 0), a signal can be sent along the diagonals whose index is in M so that the letters of the input word at positions in M can be marked before the compressed computation effectively starts.

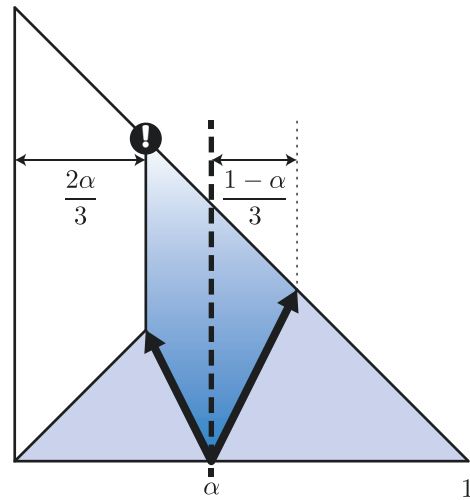
From an input word w in Σ^* , the automaton can therefore simulate the previously described automaton as if the positions in M were marked from the start, and determine in real time whether w is in L . This concludes the proof of Theorem 6.

4 Central Compression

In this section we describe a way to simulate the behavior of a 1-dimensional CA \mathcal{A} on an input $w \in \Sigma^*$ working in real time with another 1-dimensional CA \mathcal{A}' on input w with a marked position, by compressing the space-time diagram of \mathcal{A} .



■ **Figure 6** Simulation of a CA by compressing its space-time diagram by a factor 3 around a given mark (indicated by a thick dashed line). Each cell in the simulating area holds 3 states that correspond to states in the original space-time diagram. Numbers on the cells indicate which time step of the original automaton they are currently simulating.



■ **Figure 7** Diagram of the compression around a mark (thick dashed line) at position $[\alpha n]$. The thick arrows illustrate the sites where the letter of the input words are first taken into account. The site where the result of the simulated automaton is computed is indicated by the exclamation mark.

4.1 General Description

Assume that a special position has been marked on the input word of \mathcal{A}' . We want to group the states of the original simulated CA by groups of 3 around the mark as illustrated by Figure 6. To do so, the letters of the input word (represented as large circles in the figure) are shifted towards the marked position (indicated by a thick dashed line). Because the letters do not know in advance whether the mark is to their right or to their left, the AC uses two separate layers, one that shifts the letters to the right and the other to the left. Letters that move away from the mark will never be grouped and will not affect the simulation.

When letters reach the marked position, they stack on the corresponding cell. When a cell has 3 letters, it is considered full and its neighbors start gathering letters in turn. In Figure 6, grouped states are represented by small circles.

Once a cell is fully grouped, it watches its neighbors until it has enough information to simulate 3 steps of the original automaton at once on all its grouped states. This happens when its neighbors are fully grouped and their simulated time is at least equal to its own. During the compressed simulation, the difference between the simulated times of a cell and its neighbor is at most 3 (it can be -3, 0 or 3 since the simulation advances by 3 steps at a time). If a cell advances faster than its neighbor it has to memorize its previous state so that the neighbor can use it when doing its own transition. In Figure 6, the number in each cell represents the simulated time step: a cell numbered 3 for instance has to wait until both its

neighbors are labelled 3 or more before it can compute step 6 for all 3 of its grouped states. Cells represented with a grey square are cells that don't contain any significant information (they correspond to sites that are outside of the real time cone in the original space-time diagram) so their neighbors do not need to wait for their information.

If the mark around which the cells are grouped is in the first half of the input word (which is the case in Figure 6 as there are 9 letters left of the mark and 15 right) the simulation can take place properly as the left part can compute its states faster and have the information ready for the right part. The key point is that from the time when the rightmost cell is fully grouped (this time is indicated by a thick horizontal line in Figure 6) all the sites on the diagonal must be able to advance their computation by 3 steps. This guarantees that the leftmost cell has eventually computed as many steps of the original diagram as if it had started at the indicated time and advanced by 3 steps each time, which corresponds to the whole computation of the original diagram.

In Figure 6, the leftmost cell of the compressed area seems to be 2 steps behind real time at the end of the simulation (the initial configuration has 24 letters so the real time is 23). However, because the cell holds the states of the 3 leftmost cells of the original configuration at time 21, it has all the relevant information to determine the state of the origin at time 23.

There are of course many rounding problems when the number of letters left or right of the mark is not a multiple of 3. However these roundings cause at most a constant delay, which can be corrected by using a constant speed-up theorem [4].

4.2 Properties

By compressing the space-time diagram of the automaton around the marked position, we are able to perform the same computation with some significant differences.

First, the letters of the input word are not taken into account from the start of the computation, but rather in a sequential order. The time at which a given letter of the input word is effectively considered to determine the result of a transition in the simulation is proportional to its distance to the initial mark (see Figure 7 in which the sites where the letters of the input word are first taken into account are along the thick arrows).

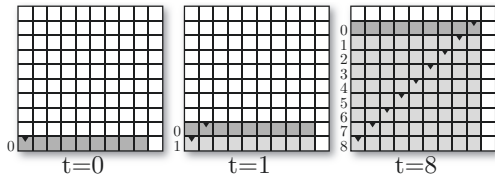
Second, the result of the computation is obtained significantly before real time, on a cell that it not the origin. The site where the result is obtained is represented by **!** in Figure 7.

Let us denote by α the proportion of the word at which the mark is set. When using the compression in a later section, we will need the free space left of the compressed area (which is of width $\frac{2\alpha}{3}$) to be larger than each of the sides of the compressed area. As said before, we need $\alpha \leq \frac{1}{2}$ for the simulation to work without delay, which means that the left side of the compressed area is smaller than the right side. Since the right side is of width $\frac{1-\alpha}{3}$, this means that we want $\frac{1-\alpha}{3} \leq \frac{2\alpha}{3}$, which amounts to $\frac{1}{3} \leq \alpha \leq \frac{1}{2}$.

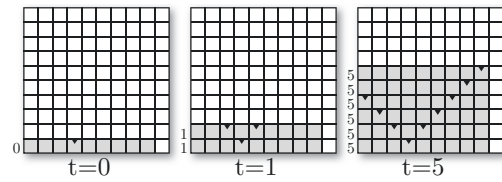
5 The Power of Space

In this section we compare the computational power of 2-dimensional CA working on the Moore neighborhood to that of 1-dimensional CA working on the standard neighborhood.

► **Definition 8.** Given a marked language $L \subseteq (\Sigma \times \{0,1\})^*$, we define $\tilde{L} \subseteq \Sigma^*$ as the language obtained by removing the marks of words in L (\tilde{L} can be seen as the result of the first projection map on L).



■ **Figure 8** Running n simulations of a 1-dimensional CA with a 2-dimensional CA.



■ **Figure 9** Real time parallel simulations of a 1DCA. At each time, a new line copies the current unmarked simulation of \mathcal{A} and continues from the same point (no delay). Each line has up to two special positions that are marked when the line starts simulating \mathcal{A} .

► **Theorem 9.** For any language $L \subseteq (\Sigma \times \{0, 1\})^*$ of words having at most one marked position, $L \in \text{CA}(2n) \Rightarrow \tilde{L} \in \text{CA}_2(2n)$.

Proof. Let $L \subseteq (\Sigma \times \{0, 1\})^*$ be a language in $\text{CA}(2n)$ of words having at most one marked position and \mathcal{A} be a 1-dimensional CA that recognizes L in time $n \mapsto 2n$. Let us describe a 2-dimensional CA \mathcal{A}' that recognizes \tilde{L} in linear time.

The input of \mathcal{A}' is an unmarked word $w \in \Sigma^*$ of length n . The idea is to use the second dimension of \mathcal{A}' to run n simulations of \mathcal{A} , one for each possible position of the mark as shown on Figure 8.

At time $t = 0$, the input is on the first line and a simulation of \mathcal{A} starts on that line with a mark on the leftmost cell of the word. At each subsequent time, the original input is copied on the next line (moving up), and a new simulation of \mathcal{A} is started on that line with the mark on the next position (moving right). At time $t = n$, the first line has simulated n steps of \mathcal{A} with a mark on the first cell, while the n -th line starts a new simulation with the mark on the last cell.

When a simulation finishes, the result is sent back towards the origin (on the first line). At time $t = 3n$ the simulation on line n is finished, and at time $t = 4n$ the results of all simulations are available on the origin (to be complete, the automaton must also run an extra simulation on the first line that simulates the behavior of \mathcal{A} on input w without any mark). The language \tilde{L} is therefore recognized in time $n \mapsto 4n$, and by using a linear acceleration we get $\tilde{L} \in \text{CA}_2(2n)$. ◀

► **Theorem 10.** For any language $L \subseteq (\Sigma \times \{0, 1\})^*$ of words having at most one marked position, $L \in \text{CA}(n) \Rightarrow \tilde{L} \in \text{CA}_2(n)$.

Proof. The basic idea is again to run parallel simulations of a 1-dimensional CA on the lines of the 2-dimensional CA, but because the automaton must work in real time it is not possible to waste a linear time starting the simulations, nor a linear time bringing back the results of the farthest simulation to the origin, which is why we use the compressed simulation presented in Section 4.

Consider a language $L \in (\Sigma \times \{0, 1\})^*$ such that all words of L have at most one marked position and a 1-dimensional CA \mathcal{A} that recognizes L in real time. We want to describe a 2-dimensional CA \mathcal{A}_2 that takes an unmarked word $w \in \Sigma^*$ of length n and decides in real time if by adding at most one mark to w we can obtain a word in L .

For now, let us assume that the input word w has a mark on a position between $\lfloor \frac{n}{3} \rfloor$ and $\lfloor \frac{n}{2} \rfloor$ so that we can run the compressed simulation easily. This mark will be referred to as the *compression mark*, it is different from the marks of L that we want to simulate on each line.

Instead of simulating \mathcal{A} , \mathcal{A}_2 will simulate an automaton \mathcal{A}_C that simulates \mathcal{A} with a central compression on the compression mark as described in Section 4.

The behavior of \mathcal{A}_2 is as follows:

- at time $t = 0$, the first line of \mathcal{A}_2 starts a simulation of \mathcal{A}_C as if no letter of the input was marked;
- at each time, this “unmarked” simulation is copied to the next line (moving up), each line continues the simulation from the step at which it is when copied (so that more and more lines are performing the same simulation of \mathcal{A}_C , without suffering any delay);
- meanwhile, each line has one or two *special positions*. The special position of the first line is the one where the compression mark is, and the special positions of line $(i + 1)$ are the one left of the leftmost special position of i , and the one right of the rightmost position of i (see Figure 9). Special positions on each line are marked when the line copies the simulation from the previous line.
- during the simulation of \mathcal{A}_C by a line, when one of the cells at a special position finishes grouping 3 letters of the input word, 3 new simulations of \mathcal{A}_C are started on this line, each considering that there was a mark on one of the letters of the input word that was grouped on the special position. Because each line has at most two special positions, at most 7 simulations of \mathcal{A}_C are run in parallel on each line (3 for each special position and the unmarked one).

This construction works because of the properties of the central compression discussed in Section 4.

First, the simulations of \mathcal{A}_C on each line can be performed properly without any delay because the time at which an input letter that eventually is grouped on a special position of the line becomes significant is after the activation of the line and the marking of its special positions. Therefore, all simulations of \mathcal{A}_C on all lines are synchronized, and the farther lines do not suffer any delay.

Second, the number of lines really used (on which significant simulations that correspond to a potential mark on an input letter) is equal to the length of the largest side of the compressed area (which is $\frac{1-\alpha}{3}$, see Figure 7). This length is less than the time remaining when the simulations of \mathcal{A}_C obtain their result ($\frac{2\alpha}{3}$), so it means that there is enough time to send back the result of the parallel simulations to the origin in real time.

The last detail is now to remove the requirement for the compression mark to be given as input. From Theorem 6, we know that if the computation can be performed in real time with a mark anywhere between positions $\lfloor \frac{n}{3} \rfloor$ and $\lfloor \frac{n}{2} \rfloor$ then it can be done in real time without previous marking. Technically, Theorem 6 only applies to 1-dimensional CA, but in this case the 2-dimensional CA performs 1-dimensional computations on each line almost independently so by having each line perform the construction from the proof of Theorem 6 we obtain the result for this specific 2-dimensional CA. ◀

6 Consequences

Let us now discuss some consequences of Theorem 10.

► **Corollary 11.** *The concatenation L_1L_2 of two 1-dimensional real time languages L_1 and L_2 is recognizable in real time by a 2-dimensional CA working on the Moore neighborhood.*

Proof. Given a word $w = uv$ with a mark between u and v , it is easy to check in real time if $u \in L_1$ and $v \in L_2$, so from Theorem 10 the unmarked language is in $CA_2(n)$. ◀

► **Corollary 12.** *If $CA(n) = CA_2(n)$, $CA(n)$ is closed under concatenation.*

Without the assumption that $CA(n) = CA_2(n)$, it is still unknown whether $CA(n)$ is closed under concatenation. Actually, it is also unknown whether $CA(2n)$ is closed under concatenation and even if the concatenation of two languages in $CA(n)$ is in $CA(2n)$.

► **Corollary 13.** *For any language L and any $\alpha \in \mathbb{Q} \cap [0, 1]$, $L^{[\alpha]} \in CA(n) \Rightarrow L \in CA_2(n)$.*

Proof. Given a word with one marked position, it is easy to check simultaneously in real time if the mark is at position $\lfloor \alpha n \rfloor$ and if the marked word is in $L^{[\alpha]}$. ◀

► **Corollary 14.** *If $CA(n) = CA_2(n)$, for any language L and any $\alpha \in \mathbb{Q} \cap [0, 1]$, $L^{[\alpha]} \in CA(n) \Rightarrow L \in CA(n)$.*

Under the assumption that $CA(n) = CA_2(n)$, Theorem 10 can also be strengthened:

► **Corollary 15.** *If $CA(n) = CA_2(n)$, for any $k \in \mathbb{N}$ and any language $L \subseteq (\Sigma \times \{0, 1\})^*$ of words having up to k marked positions, $L \in CA(n) \Rightarrow \tilde{L} \in CA(n)$.*

Proof. By induction on k . The case $k = 1$ is a direct consequence of Theorem 10 and the assumption that $CA(n) = CA_2(n)$.

If the corollary is true for up to k marks, and L is a language of words with up to $(k + 1)$ marks, let us consider the language $L' \subseteq (\Sigma \times \{0, 1\}^2)^*$ obtained from L by changing the first marked letter $(u_i, 1)$ to $(u_i, 0, 1)$, all other marked letters $(u_i, 1)$ into $(u_i, 1, 0)$ and all unmarked letters $(u_i, 0)$ into $(u_i, 0, 0)$ (effectively distinguishing the first mark from the others).

If $L \in CA(n)$ then $L' \in CA(n)$ since it's possible to simulate the recognition of L by considering that all marks are the same, while independently checking in real time that the distinguished mark is the first marked position. From Theorem 10 the language of words in L in which the first mark has been removed is therefore in $CA_2(n)$ and hence also in $CA(n)$. The words of this language have at most k marks so from the induction hypothesis, $L \in CA(n)$. ◀

References

- 1 Stephen N. Cole. Real-time computation by n -dimensional iterative arrays of finite-state machines. *IEEE Transactions on Computers*, C-18(4):349–365, 1969.
- 2 Marianne Delorme, Enrico Formenti, and Jacques Mazoyer. Open problems on cellular automata. Technical report, LIP - ENS Lyon, 2000.
- 3 Oscar H. Ibarra and Tao Jiang. Relating the power of cellular arrays to their closure properties. *Theoretical Computer Science*, 57(2-3):225–238, 1988.
- 4 Jacques Mazoyer and Nicolas Reimen. A linear speed-up theorem for cellular automata. *Theor. Comput. Sci.*, 101(1):59–98, 1992.
- 5 Alvy R. Smith III. Simple computation-universal cellular spaces. *J. ACM*, 18(3):339–353, 1971.
- 6 Véronique Terrier. Low complexity classes of multidimensional cellular automata. *Theor. Comput. Sci.*, 369(1-3):142–156, 2006.
- 7 John von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Urbana, IL, USA, 1966.

Tropical Effective Primary and Dual Nullstellensätze*

Dima Grigoriev¹ and Vladimir V. Podolskii^{2,3}

- 1 CNRS, Mathématiques, Université de Lille
France
Dmitry.Grigoryev@math.univ-lille1.fr
- 2 Steklov Mathematical Institute
Moscow, Russia
podolskii@mi.ras.ru
- 3 National Research University Higher School of Economics
Moscow, Russia

Abstract

Tropical algebra is an emerging field with a number of applications in various areas of mathematics. In many of these applications appeal to tropical polynomials allows to study properties of mathematical objects such as algebraic varieties and algebraic curves from the computational point of view. This makes it important to study both mathematical and computational aspects of tropical polynomials.

In this paper we prove tropical Nullstellensatz and moreover we show effective formulation of this theorem. Nullstellensatz is a next natural step in building algebraic theory of tropical polynomials and effective version is relevant for computational aspects of this field.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems, G.2.1 Combinatorics

Keywords and phrases tropical algebra, tropical geometry, Nullstellensatz

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.379

1 Introduction

A *min-plus* or *tropical semiring* is defined by the set \mathbb{K} , which can be \mathbb{R} , $\mathbb{R}_\infty = \mathbb{R} \cup \{+\infty\}$, \mathbb{Q} or $\mathbb{Q}_\infty = \mathbb{Q} \cup \{+\infty\}$ endowed with two operations *tropical addition* \oplus and *tropical multiplication* \odot defined in the following way:

$$x \oplus y = \min\{x, y\}, \quad x \odot y = x + y.$$

Tropical polynomials are a natural analog of classical polynomials. In classical terms it can be expressed in the form $f(\vec{x}) = \min_i M_i(\vec{x})$, where each $M_i(\vec{x})$ is a linear polynomial (a tropical monomial) in variables $\vec{x} = (x_1, \dots, x_n)$, and all coefficients of all M_i are nonnegative integers except a free coefficient which can be any element of \mathbb{K} .

The degree of a tropical monomial M is the sum of its coefficients (except the free coefficient) and the degree of a tropical polynomial f denoted by $\deg(f)$ is the maximal

* The first author is grateful to Max-Planck Institut für Mathematik, Bonn for its hospitality during the work on this paper.

The second author is partially supported by the Russian Foundation for Basic Research and the programme “Leading Scientific Schools”. Part of the work of the second author was done during the visit to Max-Planck Institut für Mathematik, Bonn.

degree of its monomials. A point $\vec{a} \in \mathbb{K}^n$ is a root of the polynomial f if the minimum $\min_i \{M_i(\vec{a})\}$ is either attained on at least two different monomials M_i or is infinite. We defer a more detailed definitions on the basics of min-plus algebra to Preliminaries.

Tropical polynomials have appeared in various areas of mathematics and found many applications (see, for example, [13, 20, 24, 21, 22, 12]). One of the most important advantage of tropical algebra is that it makes some properties of classical mathematical objects computationally accessible [26, 13, 20, 24]. One of the main goals of min-plus mathematics is to build a theory of tropical polynomials which would help to work with them and would possibly lead to new results in the related areas. Computational reasons, on the other hand, make it important to keep the theory maximally computationally efficient.

The best studied so far is the case of linear tropical polynomials and systems of linear tropical polynomials. For them the analog of the large part of the theory of classical linear polynomials was established. This includes studies of tropical analogs of the rank of a matrix and the independence of vectors [4, 15, 1], the analog of the determinant of a matrix and its properties [22], the analog of Gauss triangular form [8]. Also the solvability problem for tropical linear systems was studied from the complexity point of view. Interestingly, it turned out to be polynomially equivalent to a well known mean payoff games problem [10]. Thus, this problem lies in $\text{NP} \cap \text{coNP}$, but is not known to be in P .

For tropical polynomials of arbitrary degree less is known. In [23] the radical of a tropical ideal was explicitly described. In [26] it was shown that solvability problem for tropical polynomial systems is NP -complete.

Along with tropical polynomials there were also studied min-plus polynomials. Min-plus polynomial is an expression of the form $\min_i M_i(\vec{x}) = \min_j L_j(\vec{x})$, where M_i and L_j are tropical monomials. A point $\vec{a} \in \mathbb{K}^n$ is a root of the polynomial if $\min_i M_i(\vec{a}) = \min_j L_j(\vec{a})$.

Min-plus polynomials were studied mainly for its connections to dynamic programming (see [3, 16]). As in the case of tropical polynomials here the best studied case is the case of linear min-plus polynomials [3]. Also in [10] the connection of min-plus and tropical linear polynomials was established.

As for the min-plus polynomials of arbitrary degree much less is known. We are only aware of the result on the computational complexity of the system of min-plus polynomials: paper [11] shows that this problem is NP -complete.

Our results

The next natural step in developing of the theory of tropical polynomials would be an analog of classical Nullstellensatz, the theorem which for the classical polynomials constitutes one of the cornerstones of algebraic geometry. Concerning the tropical Nullstellensatz, the problem was already addressed in the paper [7]. In this paper there was established a general idea to approach this theorem in the tropical case through the dual formulation. Moreover, in [7] there was formulated a conjecture (which we restate below as Conjecture 3) capturing the formulation of the tropical dual Nullstellensatz and this conjecture was proven for the case of polynomials of 1 variable. Previously in [25] tropical dual Nullstellensatz was established for a pair of polynomials ($k = 2$) in 1 variable relying on the classical resultant and on the Kapranov's theorem [5, 25].

More specifically, in [7] there was considered a Macaulay matrix of the system of tropical polynomials $F = \{f_1, \dots, f_k\}$. This matrix can be easily constructed from F : we just consider all polynomials $f_i + M_j$ (in classical notation) of degree at most N , where N is a parameter and M_j is a tropical monomial. We put the coefficients of these polynomials in the rows of the matrix, where columns of the matrix correspond to monomials. Empty

entries of the matrix we fill with ∞ . The resulting matrix we denote by C_N . In [7] it was conjectured that the system of polynomials F has a solution iff the tropical linear system with the matrix C_N has a solution, and moreover N can be bounded by some function on n , k and the degree of polynomials in F (this refers to effectiveness).

In this paper we prove this conjecture. Moreover, we show an effective version of the theorem. That is, we pose bounds on N and provide examples showing that they are close to tight. These bounds are relevant for computational aspects of tropical polynomial systems. Surprisingly, it turns out that the cases of tropical semiring with and without ∞ differ dramatically. More specifically, in the case of tropical semirings $\mathbb{K} = \mathbb{R}$ and $\mathbb{K} = \mathbb{Q}$ we show that F has a solution iff the tropical linear system with the matrix C_N has a solution, where $N = (n + 2) \cdot k \cdot d$, d is the maximal degree of polynomials in F , k is the number of polynomials in F and n is the number of variables. For the case of tropical semirings $\mathbb{K} = \mathbb{R}_\infty$ and $\mathbb{K} = \mathbb{Q}_\infty$ we show a similar result, but with $N = (Cd)^{\min(n,k)}$ for some constant C . Thus for the case without ∞ the bound on N is polynomial in n, k, d and for the case with ∞ the bound on N is still polynomial in d , but is exponential in n and k . We give examples showing that our bounds on N are qualitatively optimal, that is the difference of the values of N in these cases is not an artifact of the proof, but is unavoidable. However, quantitatively there is a gap between upper and lower bounds, see Section 3 for details.

Regarding the substantial gap between the required degree in the finite and infinite cases we observe there is a similar situation for classical Nullstellensatz. Indeed, we show that in case of semiring \mathbb{R} the bound in a tropical effective Nullstellensatz depends on the sum of the degrees of the polynomials, while in case of larger semiring \mathbb{R}_∞ the bound depends on the product of the degrees (Theorems 4 and 9). We recall that for systems of classical polynomials over an algebraically closed field the bound on the effective Nullstellensatz depends on the sum of the degrees of polynomials in homogeneous (projective) case [18, 19] while the bound depends on the product of the degrees for arbitrary polynomials (affine case) [6, 17].

Next we show the primary version of tropical Nullstellensatz. We view Nullstellensatz as a duality¹ result for systems of polynomials: if there is *no* solution to the system of polynomials then some positive property holds (something *does* exist). In the classical case this positive property is the containment of 1 in the ideal generated by polynomials (over algebraically closed field). The naive analog does not hold for the tropical case. Indeed, for example, in the tropical ideal generated by the system of tropical polynomials $\{\min(x, 0), \min(x, 1)\}$ there are no polynomials with only one monomial and thus there is no polynomial 0. Basically, the point is that in the tropical semiring there is no subtraction, so in any algebraic combination of polynomials no monomials cancel out. To overcome this difficulty we introduce the notion of *nonsingular* tropical algebraic combination of tropical polynomials (see the definition in Preliminaries; here we only note that the property is simple and straightforward to check). For the primary tropical Nullstellensatz we show that there is no solution to tropical linear system F iff there is a nonsingular tropical algebraic combination of polynomials in F of degree at most N . We show this result for both cases of tropical semiring with and without ∞ and the value N in both cases corresponds to the size of Macaulay matrix in the tropical dual Nullstellensatz.

¹ To avoid a confusion we note that the word ‘dual’ is used in two different meanings. First, we use it in the term “dual Nullstellensatz” as opposed to standard version of Nullstellensatz. This means that dual Nullstellensatz is obtained from standard Nullstellensatz by (linear) duality. Second, we use the word ‘dual’ in term “duality result” to denote the general type of results. Since standard Nullstellensatz is a duality result itself, applying linear duality to it results in a non-duality result. Thus, dual Nullstellensatz is not a duality result.

To establish primary Nullstellensatz we need a duality for tropical linear systems. We show this duality result as a sidestep. However we note that this result is heavily based on already known results [2] and is a simple corollary of them.

We also prove similar results for the case of min-plus polynomials. As a sidestep of our analysis we show the close connection between tropical and min-plus systems of polynomials. We argue that these two models are very closely connected and that this connection can be used to establish new results in tropical algebra. The observation is that some results (like linear duality) are easier to obtain for min-plus polynomials and then translate to tropical polynomials, and some other results (like Nullstellensatz) on the other hand are easier to obtain for tropical polynomials and then translate to min-plus polynomials. In our opinion it is fruitful for further development of the theory to consider both models simultaneously.

Our techniques

We use the general approach of the paper [7] to Nullstellensatz through dual formulation.

To establish the dual Nullstellensatz we use methods of discrete geometry dealing with integer polyhedra. First we obtain dual Nullstellensatz for the case without ∞ . The case with ∞ requires much more additional work.

To obtain primary Nullstellensatz we apply the duality results for linear tropical polynomials. We note that these results rely on the completely different combinatorial techniques, namely on the connection to mean payoff games [2].

Other works on tropical Nullstellensatz

In paper [14] there was established Nullstellensatz for tropical semiring augmented with additional elements (called ghosts). This result is in the line with other results [24] trying to capture tropical mathematics by the means of the classical ones. However, tropical semiring augmented with ghosts constitutes (logically) a completely different model compared to usual tropical semiring. Thus our results are incomparable with the one of the paper [14].

We also note that the paper [23] (which has Nullstellensatz in the title) takes completely different view on Nullstellensatz. We consider Nullstellensatz as a result on the solvability of system of polynomials, and paper [23] views Nullstellensatz as a result on the structure of the radical of a tropical ideal. As it can be easily seen, for example, from our results during the translation from classical world to the tropical one, the connection between these two objects changes drastically (cf. with example $F = \{\min(x, 0), \min(x, 1)\}$ above). Thus our results are incomparable with the results of [23] as well.

The rest of the paper is organized as follows. In Section 2 we introduce main definitions. In Section 3 we present tropical and min-plus dual Nullstellensätze. In Section 4 we present tropical and min-plus primary Nullstellensätze. In Section 5 we present our results on tropical and min-plus linear duality. In Section 6 we present results on connection between tropical and min-plus polynomial systems.

Due to the space constraint in this extended abstract many proofs are omitted. They can be found in the full version of the paper [9].

2 Preliminaries

2.1 Min-plus algebra

Tropical and min-plus polynomials

A *min-plus* or *tropical semiring* is defined by the set \mathbb{K} , which can be \mathbb{R} , $\mathbb{R}_\infty = \mathbb{R} \cup \{+\infty\}$, \mathbb{Q} or $\mathbb{Q}_\infty = \mathbb{Q} \cup \{+\infty\}$ endowed with two operations, *tropical addition* \oplus and *tropical multiplication* \odot defined in the following way:

$$x \oplus y = \min\{x, y\}, \quad x \odot y = x + y.$$

Below we mainly consider $\mathbb{K} = \mathbb{R}$ and $\mathbb{K} = \mathbb{R}_\infty$. The proofs however literally translate to the cases of \mathbb{Q} and \mathbb{Q}_∞ .

The tropical (or min-plus) monomial in variables x_1, \dots, x_n is defined as

$$M = c \odot x_1^{\odot i_1} \odot \dots \odot x_n^{\odot i_n}, \tag{1}$$

where c is an element of the semiring \mathbb{K} and i_1, \dots, i_n are nonnegative integers. In usual notation the monomial is

$$M = c + i_1 x_1 + \dots + i_n x_n.$$

The degree of the monomial is defined as the sum $i_1 + \dots + i_n$. We denote $\vec{x} = (x_1, \dots, x_n)$ and for $I = (i_1, \dots, i_n)$ we introduce the notation

$$\vec{x}^I = x_1^{\odot i_1} \odot \dots \odot x_n^{\odot i_n}.$$

A tropical polynomial is the tropical sum of tropical monomials

$$f = \bigoplus_i M_i,$$

or in usual notation $f = \min_i M_i$. The degree of the tropical polynomial f denoted by $\deg(f)$ is the maximal degree of its monomials. A point $\vec{a} \in \mathbb{K}^n$ is a root of the polynomial f if the minimum $\min_i \{M_i(\vec{a})\}$ is either attained on at least two different monomials M_i or is infinite.

A min-plus polynomial is an expression of the form

$$\bigoplus_i M_i(\vec{x}) = \bigoplus_j L_j(\vec{x}),$$

where M_i, L_j are min-plus monomials. The degree of min-plus polynomial is the maximal degree among monomials M_i and L_j over all i, j . A point $\vec{a} \in \mathbb{K}^n$ is a root of this polynomial if the equality holds for $\vec{x} = \vec{a}$.

Linear polynomials

An important special case of tropical and min-plus polynomials are linear polynomials. They can be defined as general tropical polynomials of degree 1. However, it is convenient to denote by a linear polynomial an expression of the form

$$\min_{1 \leq j \leq n} \{a_j + x_j\}.$$

That is we assume that all variables are presented exactly once. The tropical linear system

$$\min_{1 \leq j \leq n} \{a_{ij} + x_j\}, \quad 1 \leq i \leq m, \tag{2}$$

can be naturally associated with its matrix $A \in \mathbb{K}^{m \times n}$. We will also use a matrix notation $A \odot \vec{x}$ for such system.

Analogously min-plus linear systems

$$\min_{1 \leq j \leq n} \{a_{ij} + x_j\} = \min_{1 \leq j \leq n} \{b_{ij} + x_j\}, \quad 1 \leq i \leq m,$$

can be associated with a pair of matrices A and B corresponding to the left-hand side and the right-hand side of an equation. We will also write min-plus linear system in a matrix form as $A \odot \vec{x} = B \odot \vec{x}$. It will be also convenient to consider min-plus linear systems of (componentwise) inequalities $A \odot \vec{x} \leq B \odot \vec{x}$. It is not hard to see that their expressive power is the same as of equations.

► **Lemma 1.** *Given a min-plus system of linear equations it is easy to construct an equivalent system of min-plus linear inequalities and visa versa.*

Proof. Indeed, each min-plus linear equation $L_1(\vec{x}) = L_2(\vec{x})$ is equivalent to the pair of min-plus inequalities $L_1(\vec{x}) \geq L_2(\vec{x})$ and $L_1(\vec{x}) \leq L_2(\vec{x})$. On the other hand min-plus linear inequality $L_1(\vec{x}) \leq L_2(\vec{x})$ is equivalent to the min-plus equation $L_1(\vec{x}) = \min(L_1(\vec{x}), L_2(\vec{x}))$. It is not hard to see that the last equation can be transformed to the form of min-plus linear equation. ◀

There is one more important convention we make concerning the case of tropical semiring with infinity. For two matrices $A, B \in \mathbb{R}_\infty^{n \times m}$ we say that the system $A \odot \vec{x} < B \odot \vec{x}$ has a solution if there is $\vec{x} \in \mathbb{R}_\infty^m$ such that for each row of the system if one of sides is finite, then strict inequality holds, but also the case where both sides are equal to ∞ is allowed (informally, we can say that $\infty < \infty$).

We also consider non-homogeneous tropical linear systems

$$\min_{1 \leq j \leq n} \{a_{ij} + x_j, a_i\}, \quad 1 \leq i \leq m. \quad (3)$$

This system can be naturally associated to the matrix $A \in \mathbb{K}^{m \times (n+1)}$ and written in the matrix form as $A \odot (\vec{x}, 0)$. Analogously, we can consider non-homogeneous min-plus linear systems $A \odot (\vec{x}, 0) \leq B \odot (\vec{x}, 0)$. We note that over $\vec{x} \in \mathbb{R}^n$ the tropical system $A \odot (\vec{x}, 0)$ is solvable iff homogeneous system $A \odot \vec{x}'$ is solvable, where $\vec{x}' = (\vec{x}, x_{n+1})$. Indeed, we can add the same number to all coordinates of the solution of the latter system to make $x_{n+1} = 0$. The same is true for min-plus case. But the same is not true over \mathbb{R}_∞ : homogeneous system always has a solution (just let $\vec{x} = (\infty, \dots, \infty)$), but non-homogeneous system does not always have a solution.

3 Tropical and Min-plus Dual Nullstellensatz

► **Definition 2.** For a given system of tropical polynomials $F = \{f_1, \dots, f_k\}$ in n variables we introduce its infinite Macaulay matrix C . The columns of C correspond to nonnegative integer vectors $I \in \mathbb{Z}_+^n$ and the rows of C correspond to the pairs (j, J) , where $1 \leq j \leq k$ and $J \in \mathbb{Z}_+^n$. For given I and (j, J) we let the entry $c_{(j, J), I}$ be equal to the coefficient of the monomial \vec{x}^I in the polynomial $\vec{x}^J \odot f_j$ (if there is no such monomial in the polynomial we assume that the entry is equal to $+\infty$). By C_N we denote the finite submatrix of the matrix C consisting of the columns I such that $i_1 + \dots + i_n \leq N$ and the rows which have all their finite entries in these columns. The tropical linear system associated with C_N will be of interest to us. Over \mathbb{R}_∞ we consider non-homogeneous system with the matrix C_N . The column corresponding to constant monomial is a non-homogeneous column.

For the system of min-plus polynomials $F = \{f_1 = g_1, \dots, f_k = g_k\}$ we analogously introduce the pair of matrices C and D corresponding to the left-hand sides and the right-hand sides of polynomials respectively. In the same way we introduce matrices C_N, D_N and the corresponding linear systems $C_N \odot \vec{y} = D_N \odot \vec{y}$. Analogously, for the case of \mathbb{R}_∞ we consider non-homogeneous systems.

In the paper [7] there were conjectured three forms of the tropical dual Nullstellensatz theorem. We state the most strong of them, effective Nullstellensatz theorem.

► **Conjecture 3** ([7]). *There is a function N of n and of $\deg(f_i)$ for $1 \leq i \leq k$ such that the system of polynomials F has a common tropical root iff the tropical linear system corresponding to the matrix C_N has a solution.*

Note that the classical analog of this statement is precisely the effective Nullstellensatz theorem in the dual form (see [7] for the detailed discussion).

In [7] the conjecture was proven for the case of $n = 1$. In this paper we prove the general case of the conjecture.

► **Theorem 4** (Tropical Dual Nullstellensatz). *Consider the system of tropical polynomials $F = \{f_1, \dots, f_k\}$ of n variables. Denote by d_i the degree of the polynomial f_i and let $d = \max_i d_i$.*

(i) *Over semiring \mathbb{R} the system F has a solution iff the Macaulay tropical linear system $C_N \odot \vec{y}$ for*

$$N = (n + 2)(d_1 + \dots + d_k)$$

has a solution.

(ii) *Over semiring \mathbb{R}_∞ the system F has a solution iff the non-homogeneous Macaulay tropical linear system $C_N \odot \vec{y}$ for*

$$N = \text{poly}(n, k) (2d)^{\min(n, k)}$$

has a solution.

Proof sketch. We describe the proof idea here. We concentrate on the case \mathbb{R} . For the case of \mathbb{R}_∞ much more additional work is required.

Throughout the proof we consider rows of the matrix C_N , solutions to $C_N \odot \vec{y}$, coefficients of tropical polynomials f_i . All of them can be viewed as vectors $\vec{a} = \{a_I\}_I$ which coordinates are labeled by $I \in D$ for some $D \subseteq \mathbb{Z}_+^n$. We further consider these vectors as the set of points $\{(I, a_I)\}_I$ in $(n + 1)$ -dimensional space. It is convenient to consider the first n dimension as horizontal and the last one as vertical.

We next consider, what does it mean for the vector \vec{a} to be a solution to one of the tropical linear equations $\vec{c} \odot \vec{y}$ of the system $C_N \odot \vec{y}$. By the definition this means that the value $c_I + a_I$ is minimized for at least two different I . It is not hard to see that equivalently this means that there is such $t \in \mathbb{R}$ that $\{-a_I + t\}_I$ lies below $\{c_I\}_I$ and has at least two common points with it. That is, we can adjust the set of points corresponding to $\{-a_I\}_I$ moving it along the vertical line in such a way that it lies below the set of points corresponding to the equation and has at least two common points with it. Thus we obtain geometrical interpretation of tropical solutions.

Next, we note that if we talk about solution to polynomial f , we can still consider the polynomial as a set of points $\{(I, f_I)\}_I$, where f_I is a coefficient of the monomial \vec{x}^I in f , but now the solution corresponds not to an arbitrary set of points $\{a_I\}_I$ but to a hyperplane.

Now it is not hard to capture the goal in geometric terms. We know that there is a solution to $C_N \odot \vec{y}$. We need to show that then there is a hyperplane solution to this system (or to system F , which is the same in the case of hyperplanes).

An interesting feature of our proof is how we obtain a hyperplane solution. The natural way would be to start with a non-hyperplane solution \vec{a} and somehow modify it to make it a hyperplane. This was an approach of paper [7] for the case $n = 1$. However, it is not clear how to do it for $n > 1$. Instead we actually find the hyperplane solution inside the system F .

For this for each polynomial f_j in F we consider the corresponding set of points $\{(I, f_{j,I})\}_I$, add to them all points (I, t) for all I and $t \geq f_{j,I}$ and consider the convex hull of this set of points. As a result we obtain a polytope P_j which is called extended Newton polytope. Note that P_j is infinite in the vertical direction.

Then we consider a new polytope P_0 . It can be expressed by the following formula:

$$P_0 = (n + 2) \cdot (P_1 + \dots + P_k),$$

where all operations are in the sense of the Minkowski sum. It turns out that the solution to the system F can be found in P_0 . Namely, one of the facets of P_0 is a solution.

To see the idea behind this let the bottom of P_0 be the set of lowest points of P_0 , first n coordinates of which are integer. Informally, the bottom of P_0 is a discrete version of the set of its non-vertical facets. Note that the bottom of P_0 can be considered as a vector $\{b_I\}_I$. First, it turns out (and it is not hard to show) that $\{b_I\}_I$ is a tropical linear combination of the rows of C_N . This means that a solution \vec{a} to $C_N \odot \vec{y}$ is also a solution to a tropical linear equation given by \vec{b} . Second observation is that it can be shown that for any i we can translate P_i to any place inside P_0 . Now we can consider the set of points $\{(I, b_I)\}_I$ and adjust the set $\{(I, a_I)\}_I$ in such a way that it is below $\{(I, b_I)\}_I$ and has at least two points in common with it. We consider one of these points and move P_i for arbitrary i to this point and inside of P_0 . Then P_i will lie above $\{(I, b_I)\}_I$. On the other hand it will correspond to one of the rows of C_N and thus \vec{a} will be a solution to it. Thus P_i and \vec{a} will have two points in common and they will also be common points of \vec{b} which lie between \vec{a} and P_i . Thus P_i will have two common points with \vec{b} , that is the bottom of P_0 is a solution to P_i . A more careful analysis along these lines shows that actually, one of the facets of P_0 is a solution to all polynomials in F . ◀

We show dual Nullstellensatz for min-plus case.

► **Theorem 5 (Min-Plus Dual Nullstellensatz).** *Consider the system of min-plus polynomials $F = \{f_1 = g_1, \dots, f_k = g_k\}$ of n variables. Denote by d_i the degree of the polynomial $f_i = g_i$ and let $d = \max_i d_i$.*

(i) *Over semiring \mathbb{R} the system F has a solution iff the Macaulay min-plus linear system $C_N \odot \vec{y} = D_N \odot \vec{y}$ for*

$$N = (n + 2) (d_1 + \dots + d_k)$$

has a solution.

(ii) *Over semiring \mathbb{R}_∞ the system F has a solution iff the non-homogeneous Macaulay min-plus linear system $C_N \odot \vec{y} = D_N \odot \vec{y}$ for*

$$N = \text{poly}(n, k) (2d)^{\min(n, k)}$$

has a solution.

The proof of this theorem is based on the application of the connection between tropical and min-plus polynomial systems, which we describe below, to tropical Dual Nullstellensatz.

We provide examples showing that our bounds on N are qualitatively tight. Namely for the semiring \mathbb{R} we consider the following family F of $(n + 1)$ tropical polynomials of degree d :

$$\begin{aligned} f_1 &= 0 \oplus 0 \odot x_1, \\ f_{i+1} &= 0 \odot x_i^{\odot d} \oplus 0 \odot x_{i+1}, \quad i \in [n - 1] \\ f_{n+1} &= 0 \oplus 1 \odot x_n. \end{aligned}$$

It is not hard to see that this system has no solutions. Indeed, if there is a solution, then from f_1 we can see that $x_1 = 0$, then from f_2 we can see that $x_2 = 0$ etc., from f_n we can see that $x_n = 0$. However from f_{n+1} we have that $x_n = -1$ which is a contradiction.

On the other hand, we show that the Macaulay tropical system with the matrix $C_{(d-1)(n-1)}$ corresponding to the system F has a solution.

For the semiring \mathbb{R}_∞ for any $d > 1$ we consider the following system F of tropical polynomials of variables x_1, \dots, x_n, y .

$$\begin{aligned} f_1 &= 0 \odot x_1 \odot y \oplus 0, \\ f_{i+1} &= 0 \odot x_i^{\odot d} \oplus 0 \odot x_{i+1}, \text{ for } i = 1, \dots, n - 1, \\ f_{n+1} &= 0 \odot x_{n-1}^{\odot d} \oplus 1 \odot x_n. \end{aligned}$$

This system clearly has no solutions. Indeed, we can consecutively show that all coordinates of a solution should be finite and then the polynomials f_n and f_{n+1} give a contradiction.

On the other hand, we show that non-homogeneous Macaulay system $C_{d^{n-1}-1} \odot \vec{y}$ has a solution.

Both of these examples translate to min-plus setting straightforwardly. The details of the proofs are omitted

We note that quantitatively there is a room for improvement between our lower and upper bounds on N . The gap is more substantial in the case of semiring \mathbb{R} . Assuming for the sake of simplicity that $n = k$ our upper bound gives approximately $N \leq dn^2$ and our lower bound gives $N \geq dn$. Thus we can formulate an open problem.

► **Open Problem 6.** *Close the gap between the upper and the lower bound on N in the tropical Nullstellensatz.*

4 Primary Tropical and Min-Plus Nullstellensatz

Next we establish Nullstellensatz in a more standard primary form.

We start with a more intuitive min-plus Nullstellensatz.

► **Theorem 7 (Min-Plus Primary Nullstellensatz).** *Consider the system of min-plus polynomials $F = \{f_1 = g_1, \dots, f_k = g_k\}$ of n variables. Denote by d_i the degree of the polynomial $f_i = g_i$ and let $d = \max_i d_i$.*

Over semiring \mathbb{R} the system F has no solution iff we can construct an algebraic min-plus combination $f = g$ of degree at most

$$N = (n + 2)(d_1 + \dots + d_k)$$

of them such that for each monomial $M = x_1^{\odot j_1} \odot \dots \odot x_n^{\odot j_n}$ its coefficient in f is greater than its coefficient in g . In algebraic combination $f = g$ we allow to use not only polynomials $f_i = g_i$, but also $g_i = f_i$.

Over semiring \mathbb{R}_∞ the system F has no solution iff we can construct an algebraic combination $f = g$ of degree at most

$$N = \text{poly}(n, k) (2d)^{\min(n, k)}$$

of them such that for each monomial $M = x_1^{\odot j_1} \odot \dots \odot x_n^{\odot j_n}$ its coefficient in f is greater than its coefficient in g and with additional property that the constant term in g is finite.

Proof. We present a proof for the case \mathbb{R} .

We will use the min-plus linear duality (Lemma 10 below) for the proof of this theorem.

By Theorem 1 the system of polynomials F has no solution over \mathbb{R} iff the corresponding Macaulay linear system

$$C_N \odot \vec{y} = D_N \odot \vec{y}$$

has no finite solution. By Lemma 1 this system is equivalent to the system of min-plus inequalities

$$\begin{pmatrix} C_N \\ D_N \end{pmatrix} \odot \vec{x} \leq \begin{pmatrix} D_N \\ C_N \end{pmatrix} \odot \vec{x}.$$

By Lemma 10 the fact that this system has no finite solution is equivalent to the fact that the dual system

$$\begin{pmatrix} D_N^T & C_N^T \end{pmatrix} \odot \begin{pmatrix} \vec{y} \\ \vec{z} \end{pmatrix} < \begin{pmatrix} C_N^T & D_N^T \end{pmatrix} \odot \begin{pmatrix} \vec{y} \\ \vec{z} \end{pmatrix}$$

has a solution in \mathbb{R}_∞^n (here we allow for both sides to be infinite in some rows; note that we have to use linear duality over \mathbb{R}_∞ since C_N and D_N might have infinite entries).

This system can be interpreted back in terms of polynomials. Indeed, note that now the columns of the matrices correspond to the equations of F multiplied by some \vec{x}^J and rows correspond to some monomials \vec{x}^I . Thus the solution to the system corresponds to the sum of equations of F multiplied by some monomials, such that each coefficient of the sum on the left side is smaller than the coefficient of the sum on the right side. The fact that we allow both sides to be infinite in some row corresponds to the fact that some monomials might be not presented in the sum. The fact that we allow infinite coordinates in the solution correspond to the fact that we do not have to use all polynomials of $\vec{x}^I f_j = \vec{x}^I g_j$ in algebraic combination. ◀

For the tropical case we will need the following definition.

► **Definition 8.** For the system of tropical polynomials f_1, \dots, f_k and tropical monomials M_1, \dots, M_m the algebraic combination

$$g = \bigoplus_{j=1}^m g_j,$$

where

$$g_j = M_j \odot f_{i_j},$$

is called nonsingular if the following two properties hold:

- for each monomial M of g there is a (unique) $1 \leq l(M) \leq m$ such that the coefficient of M at polynomial $g_{l(M)}$ is less than the coefficients of M at all other polynomials g_j for $j \neq l(M)$;
- for different M and M' we have $l(M) \neq l(M')$.

Now we can formulate tropical Nullstellensatz in a primary form.

► **Theorem 9** (Tropical Primary Nullstellensatz). *Consider the system of tropical polynomials $F = \{f_1 = g_1, \dots, f_k = g_k\}$ of n variables. Denote by d_i the degree of the polynomial f_i and let $d = \max_i d_i$.*

The system F has no solution over \mathbb{R} iff there is a nonsingular algebraic combination g for it of degree at most

$$N = (n + 2)(d_1 + \dots + d_k)$$

The system F has no solution over \mathbb{R}_∞ iff there is a nonsingular algebraic combination g for it of degree at most

$$N = \text{poly}(n, k) (2d)^{\min(n, k)}$$

and with finite constant monomial.

For the proofs of the last two theorems we apply min-plus and tropical linear duality (which we describe below) to min-plus and tropical dual Nullstellensätze respectively. The idea is simple. By dual Nullstellensatz we have that there is solution for system F iff there is a solution to Macaulay linear system. Applying linear duality to this system we get that there is a solution for F iff there is no solution to some new (tropical or min-plus) linear system. Finally, we interpret this solution back in terms of polynomials and obtain primary Nullstellensätze.

5 Linear Duality

We show the following simple formulation of min-plus duality.

► **Lemma 10.** *For two matrices $A, B \in \mathbb{R}^{n \times m}$ exactly one of the following is true.*

1. *There is a solution to $A \odot \vec{x} \leq B \odot \vec{x}$.*
2. *There is a solution to $B^T \odot \vec{y} < A^T \odot \vec{y}$.*

For two matrices $A, B \in \mathbb{R}_\infty^{n \times m}$ exactly one of the following is true.

1. *There is a solution $\vec{x} \neq (\infty, \dots, \infty)$ to $A \odot \vec{x} \leq B \odot \vec{x}$.*
2. *There is a finite solution to $B^T \odot \vec{y} < A^T \odot \vec{y}$.*

For two matrices $A, B \in \mathbb{R}_\infty^{n \times m}$ exactly one of the following is true.

1. *There is a finite solution to $A \odot \vec{x} \leq B \odot \vec{x}$.*
2. *There is a solution $\vec{y} \neq (\infty, \dots, \infty)$ to $B^T \odot \vec{y} < A^T \odot \vec{y}$.*

We show similar result for tropical duality.

► **Lemma 11.** *For a matrix $A \in \mathbb{R}^{n \times m}$ exactly one of the following is true.*

1. *There is a solution to $A \odot \vec{x}$.*
2. *There is \vec{z} such that in each row of $A^T \odot \vec{z}$ the minimum is attained only once and for each two rows the minimums are in different columns.*

For a matrix $A \in \mathbb{R}_\infty^{n \times m}$ exactly one of the following is true.

1. *There is a finite solution to $A \odot \vec{x}$.*
2. *There is \vec{z} such that in each row of $A^T \odot \vec{z}$ the minimum is attained only once or is equal to ∞ and for each two rows the (unique) minimums are in different columns.*

For a matrix $A \in \mathbb{R}_\infty^{n \times m}$ exactly one of the following is true.

1. *There is a solution to $A \odot \vec{x}$.*
2. *There is a finite \vec{z} such that in each row of $A^T \odot \vec{z}$ the minimum is attained only once and for each two rows the minimums are in different columns.*

The idea for the proof of min-plus linear duality is a connection to mean payoff games established in [2]. Once this connection is known the proof of min-plus linear duality is rather simple. However, we are not aware of a presentation of this (or similar) result in the literature, so due to the simplicity of the formulation and the fact that we use this result to obtain primary Nullstellensatz, we decide to include it in the paper.

For the proof of tropical linear duality we use connection between tropical and min-plus polynomials and deduce the tropical duality from the min-plus duality. However, we note that it also can be shown directly using the analysis of the paper [8].

6 Tropical vs. Min-plus

We also establish the connection between tropical and min-plus polynomial systems.

► **Lemma 12.** *For both \mathbb{R} and \mathbb{R}_∞ given a system of tropical polynomials we can construct a system of min-plus polynomials over the same set of variables and with the same set of solutions.*

In the other direction we do not have such a simple connection, but we can still prove the following lemma.

► **Lemma 13.** *For any system of min-plus polynomials F over n variables there is a system of tropical polynomials T over $2n$ variables and an injective linear transformation $H: \mathbb{R}_\infty^n \rightarrow \mathbb{R}_\infty^{2n}$ such that the image of the solutions of F coincides with the solution set of T . The same is true over semiring \mathbb{R} .*

The proofs of these lemmas follow the lines of the proof of the analogous statement for the case of linear polynomials in the paper [10].

References

- 1 Marianne Akian, Stephane Gaubert, and Alexander Guterman. Linear independence over tropical semirings and beyond. *Contemporary Mathematics*, 495:1–33, 2009.
- 2 Marianne Akian, Stephane Gaubert, and Alexander Guterman. Tropical polyhedra are equivalent to mean payoff games. *International Journal of Algebra and Computation*, 22(1), 2012.
- 3 Peter Butkovič. *Max-linear Systems: Theory and Algorithms*. Springer, 2010.
- 4 M. Develin, F. Santos, and B. Sturmfels. On the rank of a tropical matrix. *Combinatorial and computational geometry*, 52:213–242, 2005.
- 5 Manfred Einsiedler, Mikhail Kapranov, and Douglas Lind. Non-archimedean amoebas and tropical varieties. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 2006.601:139–157, 2007.
- 6 Marc Giusti, Joos Heintz, and Juan Sabia. On the efficiency of effective Nullstellensätze. *Computational complexity*, 3(1):56–95, 1993.
- 7 Dima Grigoriev. On a tropical dual Nullstellensatz. *Advances in Applied Mathematics*, 48(2):457 – 464, 2012.
- 8 Dima Grigoriev. Complexity of solving tropical linear systems. *Computational Complexity*, 22(1):71–88, 2013.
- 9 Dima Grigoriev and Vladimir V. Podolskii. Tropical effective primary and dual Nullstellensätze. *CoRR*, abs/1409.6215, 2014.
- 10 Dima Grigoriev and Vladimir V. Podolskii. Complexity of tropical and min-plus linear prevarieties. *Computational complexity, to appear*, pages 1–34, 2015.

- 11 Dima Grigoriev and Vladimir Shpilrain. Tropical cryptography. *Communications in Algebra*, 42(6):2624–2632, 2014.
- 12 Birkett Huber and Bernd Sturmfels. A polyhedral method for solving sparse polynomial systems. *Mathematics of Computation*, 64:1541–1555, 1995.
- 13 I. Itenberg, G. Mikhalkin, and E.I. Shustin. *Tropical Algebraic Geometry*. Oberwolfach Seminars. Birkhäuser, 2009.
- 14 Zur Izhakian. Tropical algebraic sets, ideals and an algebraic Nullstellensatz. *International Journal of Algebra and Computation*, 18(06):1067–1098, 2008.
- 15 Zur Izhakian and Louis Rowen. The tropical rank of a tropical matrix. *Communications in Algebra*, 37(11):3912–3927, 2009.
- 16 Stasys Jukna. Lower bounds for tropical circuits and dynamic programs. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:80, 2014.
- 17 János Kollár. Sharp effective Nullstellensatz. *J. Amer. Math. Soc.*, 1:963–975, 1988.
- 18 D. Lazard. Algèbre linéaire sur $K[X_1, \dots, X_n]$ et élimination. *Bull. Soc. Math. France*, 105(2):165–190, 1977.
- 19 D. Lazard. Resolution des systemes d’equations algebriques. *Theoret. Comput. Sci.*, 15(1):77–110, 1981.
- 20 Diane Maclagan and Bernd Sturmfels. *Introduction to Tropical Geometry*, volume 161 of *AMS Graduate Studies in Mathematics*. AMS, to appear, 2015.
- 21 Grigory Mikhalkin. Amoebas of algebraic varieties and tropical geometry. In Simon Donaldson, Yakov Eliashberg, and Mikhael Gromov, editors, *Different Faces of Geometry*, volume 3 of *International Mathematical Series*, pages 257–300. Springer US, 2004.
- 22 Jürgen Richter-Gebert, Bernd Sturmfels, and Thorsten Theobald. First steps in tropical geometry. *Idempotent Mathematics and Mathematical Physics, Contemporary Mathematics*, 377:289–317, 2003.
- 23 Eugeni Shustin and Zur Izhakian. A tropical Nullstellensatz. *Proceedings of the American Mathematical Society*, 135(12):3815–3821, 2007.
- 24 Bernd Sturmfels. *Solving Systems of Polynomial Equations*, volume 97 of *CBMS Regional Conference in Math*. American Mathematical Society, 2002.
- 25 Luis Felipe Tabera. Tropical resultants for curves and stable intersection. *Revista Matemática Iberoamericana*, 24(3):941–961, 04 2008.
- 26 Thorsten Theobald. On the frontiers of polynomial computations in tropical geometry. *J. Symb. Comput.*, 41(12):1360–1375, 2006.

Upper Tail Estimates with Combinatorial Proofs

Jan Hażła and Thomas Holenstein

ETH Zürich

Department of Computer Science, Zurich, Switzerland

{jan.hazla,thomas.holenstein}@inf.ethz.ch

Abstract

We study generalisations of a simple, combinatorial proof of a Chernoff bound similar to the one by Impagliazzo and Kabanets (RANDOM, 2010).

In particular, we prove a randomized version of the hitting property of expander random walks and use it to obtain an optimal expander random walk concentration bound settling a question asked by Impagliazzo and Kabanets.

Next, we obtain an upper tail bound for polynomials with input variables in $[0, 1]$ which are not necessarily independent, but obey a certain condition inspired by Impagliazzo and Kabanets. The resulting bound is applied by Holenstein and Sinha (FOCS, 2012) in the proof of a lower bound for the number of calls in a black-box construction of a pseudorandom generator from a one-way function.

We also show that the same technique yields the upper tail bound for the number of copies of a fixed graph in an Erdős–Rényi random graph, matching the one given by Janson, Oleszkiewicz, and Ruciński (Israel J. Math, 2002).

1998 ACM Subject Classification G.3 Probability and Statistics

Keywords and phrases concentration bounds, expander random walks, polynomial concentration

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.392

1 Introduction

Motivation and previous work

Concentration bounds are inequalities that estimate the probability of a random variable assuming a value that is far from its expectation. They have a multitude of applications all across the mathematics and theoretical computer science. See, e.g., textbooks [26, 25, 4, 10] for uses in complexity theory and randomised algorithms.

A typical setting is when this variable is a function $f(x)$ of n simpler random variables $x = (x_1, \dots, x_n)$ that possess a certain degree of independence and we try to bound said probability with a function decaying exponentially with n (or, maybe, n^ϵ for some $\epsilon > 0$).

The canonical examples are Chernoff-Hoeffding bounds [7, 13] for the sum of n independent random variables in $[0, 1]$ and Azuma’s inequality [5] for martingales.

The standard technique to prove Chernoff bounds is due to Bernstein [6]. The idea is to bound $E[e^{tf(x)}]$ for some appropriately chosen t , and then to apply Markov’s inequality.

Recently, Impagliazzo and Kabanets [16] gave a different, combinatorial proof of Chernoff bound, arguing that its simplicity and nature provide additional insight into understanding concentration. What is more, their proof is constructive in a certain sense (see [16] for details).

The proof given by Impagliazzo and Kabanets is related to previous published results: in [28], Schmidt, Siegel and Srinivasan give a Chernoff bound which is applicable in case the random variables $x = (x_1, \dots, x_n)$ are only m -wise independent for some large enough m . It

turns out that the expressions which appear in their computations have close counterparts in the proof in [16], but they still bound $E[e^{tf(x)}]$, and it seems to us that the approach in [16] makes the concepts clearer and the calculations shorter.

Another work related to [16] is due to Janson, Oleszkiewicz and Ruciński [17], who give an upper tail bound (i.e., a one-sided concentration bound) for the number of subgraphs in an Erdős-Rényi random graph $G_{n,p}$. The proof given in [17] bears much relationship to the proof given in [16]. We elaborate on that in Section 3.2.

Finally, there is a connection to an argument used by Rao to prove a concentration bound for parallel repetition of two-prover games [27]. As we will see, one of the ideas in the proof given in [16] is to consider a subset of the variables (x_1, \dots, x_n) . Rao also does this, with a somewhat different purpose.

Our contributions

In this paper we modify the proof of Impagliazzo and Kabanets and introduce a more general sufficient condition for concentration which we term *growth boundedness* (Section 3). Then, we show some applications of our framework.

First, we prove a randomized version of the hitting property of expander random walks (Theorem 4.1) and use it to obtain an optimal (up to a constant factor in the exponent) expander random walk concentration bound settling a question asked in [16] (Theorem 4.2).¹ We also show that our method is quite robust: with a little more effort one can improve the constant factor to the optimal one in case of large number of steps and small deviation (Theorem 4.3).

Second, we prove an upper tail bound for polynomials with input random variables in $[0, 1]$ (Theorem 5.2). Contrary to the previous work we are aware of, we do not assume that those variables are independent, but rather that they obey a condition similar to growth boundedness.

This bound is used in a proof of a lower bound for the complexity of a black-box construction of a pseudorandom generator from a one-way function [14]. Although [14] was published earlier, the proof of the bound is not contained there, but deferred to this paper instead. We outline how the bound was used in [14] in Section 5.1.

Notation

Throughout the paper we focus on the bounds of the form $\Pr[f(x) \geq \mu(1 + \epsilon)]$. We call such bounds “(multiplicative) upper tail bounds”.

Typically, we consider a probability distribution P_x over some vector of random variables $x = (x_1, \dots, x_n)$. We denote a random choice from P_x as $x \leftarrow P_x$. We try to explicitly indicate randomness whenever taking probability or expectation, i.e., we write $\Pr_{x \leftarrow P_x}[\dots]$ and so on. For a finite set A , let $a \leftarrow A$ be a shorthand for a uniform random choice of an element from A .

For a natural number n , let $[n] := \{1, \dots, n\}$. As usual, by $\binom{n}{k}$ we denote $\frac{\prod_{i=0}^{k-1} (n-i)}{k!}$ for $n \in \mathbb{R}$ and $k \in \mathbb{N}$. For $n \in \mathbb{N}$ and $0 \leq k \leq n$, we also identify $\binom{n}{k}$ with the set of subsets of $[n]$ of size k .

In particular, $(i_1, \dots, i_m) \leftarrow [n]^m$ denotes uniform choice of m elements from $[n]$ with repetition and $M \leftarrow \binom{n}{m}$ uniform choice of a subset of $[n]$ of size m .

¹ Of course the bound itself is not new. Impagliazzo and Kabanets asked if such a concentration bound can be obtained from the hitting property, i.e., using the technique from [16].

2 A Simple Proof of a Chernoff Bound

We start by presenting a short proof of a Chernoff bound in, arguably, the most basic setting.

► **Theorem 2.1.** *Let $x = (x_1, \dots, x_n)$ be i.i.d. over $\{0, 1\}^n$ with $\Pr[x_i = 1] = \frac{1}{2}$ and $\epsilon \in [0, \frac{1}{2}]$. Then,*

$$\Pr_{x \leftarrow P_x} \left[\sum_{i=1}^n x_i \geq \frac{n}{2}(1 + \epsilon) \right] \leq \exp \left(-\frac{\epsilon^2 n}{6} \right).$$

Proof. Let $m := \lceil \frac{\epsilon n}{3} \rceil$. We have

$$\begin{aligned} \mathbb{E}_{x \leftarrow P_x} \left[\left(\sum_{i=1}^n x_i \right)^m \right] &= n^m \Pr_{\substack{x \leftarrow P_x \\ (i_1, \dots, i_m) \leftarrow [n]^m}} [\forall j \in [m] : x_{i_j} = 1] \\ &= n^m \prod_{j=1}^m \Pr_{\substack{x \leftarrow P_x \\ (i_1, \dots, i_m) \leftarrow [n]^m}} [x_{i_j} = 1 \mid \forall k < j : x_{i_k} = 1] \\ &\leq n^m \left(\frac{\epsilon}{3} \cdot 1 + \left(1 - \frac{\epsilon}{3} \right) \cdot \frac{1}{2} \right)^m = \left(\frac{n}{2} \right)^m \left(1 + \frac{\epsilon}{3} \right)^m. \end{aligned}$$

Using Markov's inequality and $\frac{1+\epsilon/3}{1+\epsilon} \leq \exp(-\frac{\epsilon}{2})$ for $\epsilon \in [0, \frac{1}{2}]$,

$$\Pr \left[\left(\sum_{i=1}^n x_i \right)^m \geq \left(\frac{n}{2} \right)^m (1 + \epsilon)^m \right] \leq \left(\frac{1 + \frac{\epsilon}{3}}{1 + \epsilon} \right)^m \leq \exp \left(-\frac{\epsilon^2 n}{6} \right).$$

◀

The above is the simplest proof of the most basic Chernoff bound we know of, and we believe that it is worthwhile to state it explicitly. It can be obtained by adapting the proof given in [16] for the given setting, although a direct adaptation yields a slightly different (and probably a bit longer) argument. Alternatively, it can be seen as an instantiation of the proof given in [17] in case one is interested in counting the number of copies of K_2 (i.e., the number of edges) in a random graph $G_{n,p}$, after rather many simplifications that can be done for this very special case. Finally, it is a straightforward instantiation of our later proof given in Section 3.

3 Growth Boundedness

In this section we present the definition of growth-boundedness and prove that it implies concentration. In Section 3.1 we introduce growth boundedness without repetition: a variation of our concept that we use to prove the expander random walk bound.

► **Definition 3.1.** Let $\delta \geq 0$ and $m \in [n]$. A distribution P_x over $x = (x_1, \dots, x_n) \in \mathbb{R}_{\geq 0}^n$ with $\mu := \mathbb{E}_{\substack{x \leftarrow P_x \\ i \leftarrow [n]}} [x_i]$ is (δ, m) -growth bounded if

$$\mathbb{E}_{x \leftarrow P_x} \left[\left(\sum_{i=1}^n x_i \right)^m \right] \leq (\mu n)^m (1 + \delta)^m.$$

Equivalently, P_x is (δ, m) -growth bounded if and only if

$$\mathbb{E}_{\substack{x \leftarrow P_x \\ (i_1, \dots, i_m) \leftarrow [n]^m}} \left[\prod_{j=1}^m x_{i_j} \right] \leq \mu^m (1 + \delta)^m.$$

If random variables are over $\{0, 1\}$, this condition reduces to

$$\Pr_{\substack{x \leftarrow P_x \\ (i_1, \dots, i_m) \leftarrow [n]^m}} \left[\forall j \in [m] : x_{i_j} = 1 \right] \leq \mu^m (1 + \delta)^m .$$

We now state our main theorem:

► **Theorem 3.2.** *Let P_x be a distribution over $\mathbb{R}_{\geq 0}^n$, $\mu := \mathbb{E}_{\substack{x \leftarrow P_x \\ i \leftarrow [n]}} [x_i]$, $\mu > 0$, $\epsilon \geq 0$. If P_x is (δ, m) -growth bounded, then*

$$\Pr_{x \leftarrow P_x} \left[\sum_{i=1}^n x_i \geq \mu n (1 + \epsilon) \right] \leq \left(\frac{1 + \delta}{1 + \epsilon} \right)^m .$$

Proof. By Markov’s inequality and growth boundedness of P_x ,

$$\begin{aligned} \Pr_{x \leftarrow P_x} \left[\sum_{i=1}^n x_i \geq \mu n (1 + \epsilon) \right] &= \Pr_{x \leftarrow P_x} \left[\left(\sum_{i=1}^n x_i \right)^m \geq (\mu n)^m (1 + \epsilon)^m \right] \\ &\leq \left(\frac{1 + \delta}{1 + \epsilon} \right)^m . \end{aligned}$$

◀

There is an interesting connection between this proof (inspired by [17]) and the one used in [16], for details see Section 3.2.

We obtain more convenient bounds as a corollary:

► **Corollary 3.3.** *Let $\epsilon \geq 0$ and P_x be an $(\frac{\epsilon}{3}, m)$ -growth bounded distribution over $\mathbb{R}_{\geq 0}^n$ with $\mu := \mathbb{E}_{\substack{x \leftarrow P_x \\ i \leftarrow [n]}} [x_i]$, $\mu > 0$.*

1. *If $\epsilon \leq \frac{1}{2}$: $\Pr_{x \leftarrow P_x} \left[\sum_{i=1}^n x_i \geq \mu n (1 + \epsilon) \right] \leq \exp \left(-\frac{\epsilon m}{2} \right)$.*
2. *If $\epsilon \geq \frac{1}{2}$: $\Pr_{x \leftarrow P_x} \left[\sum_{i=1}^n x_i \geq \mu n (1 + \epsilon) \right] \leq \left(\frac{4}{5} \right)^m$.*
3. *If $\epsilon \geq 3$: $\Pr_{x \leftarrow P_x} \left[\sum_{i=1}^n x_i \geq \mu n (1 + \epsilon) \right] \leq 2^{-m}$.*

Proof. (1) follows because $\frac{1+\epsilon/3}{1+\epsilon} \leq \exp(-\frac{\epsilon}{2})$ for $\epsilon \in [0, \frac{1}{2}]$, (2) since $\frac{1+\epsilon/3}{1+\epsilon} \leq \frac{4}{5}$ for $\epsilon \geq \frac{1}{2}$ and (3) due to $\frac{1+\epsilon/3}{1+\epsilon} \leq \frac{1}{2}$ for $\epsilon \geq 3$. ◀

For example, suppose that x_1, \dots, x_n are independent over $\{0, 1\}^n$, $\Pr[x_i = 1] = \mu > 0$, and $\epsilon \in [0, \frac{1}{2}]$.

Using that for each M with $|M| \leq \frac{\epsilon \mu n}{3}$ we have

$$\Pr_{\substack{x \leftarrow P_x \\ i \leftarrow [n]}} [x_i = 1 \mid \forall j \in M : x_j = 1] = \left(\frac{|M|}{n} + \left(1 - \frac{|M|}{n} \right) \mu \right) \leq \frac{|M|}{n} + \mu \leq \mu \left(1 + \frac{\epsilon}{3} \right) ,$$

we can conclude that P_x is $(\frac{\epsilon}{3}, \lceil \frac{\epsilon \mu n}{3} \rceil)$ -growth bounded and

$$\Pr_{x \leftarrow P_x} \left[\sum_{i=1}^n x_i \geq \mu n (1 + \epsilon) \right] \leq \exp(-\epsilon^2 \mu n / 6) .$$

3.1 Growth boundedness without repetition

If one looks at the process in the growth boundedness definition as choosing a uniform m -tuple of indices (i_1, \dots, i_m) (with repetition), it is possible to make a similar argument for choosing a uniform set of indices of size m instead. In particular, we find it convenient in the proof of the expander random walk bound.

► **Definition 3.4.** Let $\delta \geq -1$ and $m \in [n]$. We say that a distribution P_x over $\{0, 1\}^n$ with $\mu := \Pr_{x \leftarrow P_x} [x_i = 1]$ is (δ, m) -growth bounded without repetition if

$$\Pr_{\substack{x \leftarrow P_x \\ M \leftarrow \binom{[n]}{m}}} [\forall i \in M : x_i = 1] \leq \mu^m (1 + \delta)^m.$$

► **Theorem 3.5.** Let P_x be a distribution over $\{0, 1\}^n$, $\mu := \Pr_{x \leftarrow P_x} [x_i = 1]$, $\mu > 0$, $\epsilon \geq 0$, $c \in [0, 1]$. If P_x is $(\delta, c\epsilon\mu n)$ -growth bounded without repetition then

$$\Pr_{x \leftarrow P_x} \left[\sum_{i=1}^n x_i \geq \mu n (1 + \epsilon) \right] \leq \left(\frac{1 + \delta}{1 + (1 - c)\epsilon} \right)^m,$$

where $m := c\epsilon\mu n$.

Proof. Set $q := \Pr[\sum_{i=1}^n x_i \geq \mu n (1 + \epsilon)]$ and compute:

$$\begin{aligned} \mu^m (1 + \delta)^m &\geq \Pr_{\substack{x \leftarrow P_x \\ M \leftarrow \binom{[n]}{m}}} [\forall i \in M : x_i = 1] \\ &\geq q \Pr_{\substack{x \leftarrow P_x \\ M \leftarrow \binom{[n]}{m}}} [\forall i \in M : x_i = 1 \mid \sum_{i=1}^n x_i \geq \mu n (1 + \epsilon)] \\ &\geq q \prod_{i=0}^{m-1} \frac{\mu n (1 + \epsilon) - i}{n - i} \\ &\geq q \mu^m (1 + (1 - c)\epsilon)^m. \end{aligned}$$

◀

► **Corollary 3.6.** Let $\epsilon \in [0, \frac{4}{5}]$ and P_x be a distribution over $\{0, 1\}^n$ that is $(\frac{\epsilon}{3}, m)$ -growth bounded without repetition for some $m \leq \frac{\epsilon\mu n}{6}$ with $\mu := \Pr_{x \leftarrow P_x} [x_i = 1]$, $\mu > 0$. Then,

$$\Pr_{x \leftarrow P_x} \left[\sum_{i=1}^n x_i \geq \mu n (1 + \epsilon) \right] \leq \exp \left(-\frac{\epsilon m}{3} \right).$$

Proof. Apply Theorem 3.5 and note that $\frac{1 + \epsilon/3}{1 + 5\epsilon/6} \leq \exp \left(-\frac{\epsilon}{3} \right)$ for $\epsilon \in [0, \frac{4}{5}]$. ◀

3.2 Connection of [16] and [17]

Recall the proof of Theorem 3.2. In the context of [16] and [17] we find it instructive to give an alternative proof, restricted to distributions over $\{0, 1\}^n$ (essentially the same as the proof of Theorem 3.5).

► **Theorem 3.7.** Let P_x be a distribution over $\{0, 1\}^n$, $\mu := \Pr_{x \leftarrow P_x} [x_i = 1]$, $\mu > 0$, $\epsilon \geq 0$. If P_x is (δ, m) -growth bounded, then

$$\Pr_{x \leftarrow P_x} \left[\sum_{i=1}^n x_i \geq \mu n (1 + \epsilon) \right] \leq \left(\frac{1 + \delta}{1 + \epsilon} \right)^m.$$

Proof. Set $q := \Pr[\sum_{i=1}^n x_i \geq \mu n(1 + \epsilon)]$, and see that²

$$\begin{aligned} \mu^m(1 + \delta)^m &\geq \Pr_{\substack{x \leftarrow \mathbb{P}_x \\ (i_1, \dots, i_m) \leftarrow [n]^m}} [\forall j \in [m] : x_{i_j} = 1] \\ &\geq q \Pr_{\substack{x \leftarrow \mathbb{P}_x \\ (i_1, \dots, i_m) \leftarrow [n]^m}} [\forall j \in [m] : x_{i_j} = 1 \mid \sum_{i=1}^n x_i \geq \mu n(1 + \epsilon)] \\ &\geq q \mu^m(1 + \epsilon)^m. \end{aligned}$$



The basic idea of the proof in [16] is to consider $\Pr_{x,M}[\forall i \in M : x_i = 1]$, where M is a subset of $[n]$ obtained by including each element in M independently with some probability q . Then, this is compared with $\Pr_{x,M}[\forall i \in M : x_i = 1 \mid \mathcal{E}]$, where \mathcal{E} is the event that $\sum_{i=1}^n x_i \geq \mu n(1 + \epsilon)$. In fact, we have

$$\Pr_x[\mathcal{E}] \leq \frac{\Pr_{x,M}[\forall i \in M : x_i = 1]}{\Pr_{x,M}[\forall i \in M : x_i = 1 \mid \mathcal{E}]}.$$

It is possible to show that for $m := \mathbb{E}[|M|] \ll n$ we have $\Pr_M[\forall i \in M : x_i = 1 \mid \mathcal{E}] \gtrsim \mu^m(1 + \epsilon)^m$. To see the intuition of this, simply note that this probability roughly equals the probability of only selecting red balls when one chooses with repetition m times out of n balls, at least $\mu n(1 + \epsilon)$ of which are red.³ Thus,

$$\Pr_x[\mathcal{E}] \lesssim \frac{\Pr_{x,M}[\forall i \in M : x_i = 1]}{\mu^m(1 + \epsilon)^m}. \tag{1}$$

Now note that this last argument only uses the probability over M , and so is independent of the distribution of x . Thus, for any distribution on which we can give a good upper bound on $\Pr_{x,M}[\forall i \in M : x_i = 1]$, the technique of [16] gives a concentration result.

The argument we use is very similar, but we pick M as an m -tuple whose elements are picked independently with repetition. However, then we also have

$$n^m \Pr_{x,M}[\forall i \in M : x_i = 1] = \mathbb{E}_{x,M}[(x_1 + \dots + x_n)^m].$$

By Markov’s inequality,

$$\Pr[\mathcal{E}] = \Pr[(x_1 + \dots + x_n)^m \geq (\mu n(1 + \epsilon))^m] \leq \frac{\Pr_{x,M}[\forall i \in M : x_i = 1]}{\mu^m(1 + \epsilon)^m},$$

which is almost the same as (1).

The view in (1) is the one adopted by [16]. Bounding the m -th moment and using Markov is the view adopted in [17]. The above argument shows that these views are closely related, and one can argue that the connection is given by growth boundedness.

4 Random Walks on Expanders

Overview and our results

For an introduction to expander graphs, see [15] or [30, Chapter 4]. In short, a λ -expander is a d -regular undirected graph G with the second largest (in terms of absolute value) eigenvalue of the transition matrix at most λ .

² Clearly $q = 0$ is not a problem.

³ The difference to the actual random experiment is that we do not keep each ball with probability m/n but instead choose exactly m times.

We consider a random walk on λ -expander starting in a uniform random vertex. It is a very useful fact in many applications that such a random walk behaves in certain respects very similarly to a random walk on the complete graph.

In particular, the so called hitting property [2, 20] states that the probability that an ℓ -step random walk on a λ -expander G stays completely inside a set $W \subseteq V := V(G)$ with $\mu := |W|/|V|$ is at most $(\mu + \lambda)^\ell$. A more general version [3] states that for each $M \subseteq [\ell]$ the probability that a random walk stays inside W in all steps from M is at most $(\mu + 2\lambda)^{|M|}$.

Our first result, which may be of independent interest, can be considered as a randomized version of the hitting property. Namely, we show that, given $\epsilon > 0$, for a relatively small random subset $M \subseteq [\ell]$ of size m the probability that a random walk on a λ -expander stays inside W in all steps from M is at most $(\mu(1 + \epsilon))^m$:

► **Theorem 4.1.** *Let G be a λ -expander with a distribution \mathbb{P}_r over V^ℓ representing an $(\ell - 1)$ -step random walk $r = (v_1, \dots, v_\ell)$ (with v_1 being a uniform starting vertex) and $W \subseteq V$ with $\mu := |W|/|V|$. Let $\epsilon \geq 0$ and $m \leq \min(1, \frac{1-\lambda}{\lambda} \frac{\epsilon\mu}{2})\ell$. Then,*

$$\Pr_{\substack{r \leftarrow \mathbb{P}_r \\ M \leftarrow \binom{[\ell]}{m}}} [\forall i \in M : v_i \in W] \leq (\mu(1 + \epsilon))^m .$$

Another important property of random walks on expander graphs is the Chernoff bound estimating the probability that the number of times a random walk visits W is far from its expectation. The first Chernoff bound for expander random walks was given by Gillman [11] and the problem was treated further in numerous works [21, 24, 1, 12, 32, 8].

Impagliazzo and Kabanets [16] apply their technique to obtain a bound for random walks on expander graphs, but in case of deviations smaller than λ they lose a factor of $\log(\frac{1}{\epsilon})$ in the exponent. They then ask if their technique can be modified to avoid this loss.

We answer this question affirmatively: using Theorem 4.1 we immediately obtain a bound that matches the known ones and does not suffer from the additional $\log(\frac{1}{\epsilon})$ factor while preserving the simplicity of the proof.

► **Theorem 4.2.** *Let the setting be as in Theorem 4.1 with $\mu > 0$. Define \mathbb{P}_x over $\{0, 1\}^\ell$ as $x_i = 1 \iff v_i \in W$ and let $\epsilon \in [0, \frac{4}{5}]$. Then,*

$$\Pr_{r \leftarrow \mathbb{P}_r} \left[\sum_{i=1}^{\ell} x_i \geq \mu\ell(1 + \epsilon) \right] \leq 2 \exp \left(- \frac{(1 - \lambda)\epsilon^2\mu\ell}{18} \right) .$$

Furthermore, we demonstrate robustness of our method by improving the exponent to $\frac{1-\lambda}{1+\lambda} \frac{\mu}{1-\mu} \frac{\epsilon^2\ell}{2} + o(\epsilon^2)\ell$, which is optimal for fixed λ, μ and $\epsilon \rightarrow 0_+$ and $\ell \rightarrow \infty$:

► **Theorem 4.3.** *Let the setting be as in Theorem 4.1 with $\mu \in (0, 1)$. Define \mathbb{P}_x over $\{0, 1\}^\ell$ as $x_i = 1 \iff v_i \in W$ and let $\epsilon \in [0, 1]$. Then, there exists c_μ that depends only on μ such that*

$$\Pr_{r \leftarrow \mathbb{P}_r} \left[\sum_{i=1}^{\ell} x_i \geq \mu\ell(1 + \epsilon) \right] \leq 2 \exp \left(- \frac{1 - \lambda}{1 + \lambda} \cdot \frac{\mu}{1 - \mu} \cdot \frac{\epsilon^2\ell}{2} + c_\mu \epsilon^3 \ln\left(\frac{1}{\epsilon}\right)\ell \right) .$$

For a proof of Theorem 4.3 see the full paper. In the following we prove Theorems 4.1 and 4.2.

Proofs

First, we need a coupling argument: let $m, \ell \in \mathbb{N}, m \leq \ell$ be given. We consider the distribution $D_{m,\ell}$ defined by the following process:

- Pick uniformly $M \leftarrow \binom{[\ell]}{m}$ and let $M := \{x_1, \dots, x_m\}$ with $x_1 < \dots < x_m$.
- Let $d_1 := x_1$ and $d_i := x_i - x_{i-1}$ for $i > 1$.

A bijection shows that $d = (d_1, \dots, d_m)$ is distributed uniformly on the $\binom{[\ell]}{m}$ m -tuples which satisfy $\sum_{i=1}^m d_i \leq \ell$ and $d_i > 0$. We now couple $D_{m,\ell}$ with independent random variables (see full paper for the proof):

► **Lemma 4.4.** *Let $0 < m \leq \ell$. There exists a distribution over $(d_1, \dots, d_m, e_1, \dots, e_m)$ such that:*

- $e_i \leq d_i$ for $1 \leq i \leq m$.
- (d_1, \dots, d_m) is distributed according to $D_{m,\ell}$.
- (e_1, \dots, e_m) are i.i.d. with e_i in \mathbb{N}_+ and $\Pr[e_i = k] \leq \frac{2m}{\ell}$ for every k .

Proof of Theorem 4.1. Pick $M \leftarrow \binom{[\ell]}{m}$ and let (d_1, \dots, d_m) be as in the definition of $D_{m,\ell}$.

► **Lemma 4.5.**

$$\Pr_{\substack{r \leftarrow P_r \\ M \leftarrow \binom{[\ell]}{m}}} [\forall i \in M : v_i \in W] \leq \mathbb{E}_{M \leftarrow \binom{[\ell]}{m}} \left[\prod_{i=1}^m (\mu + \lambda^{d_i}) \right].$$

Proof. Let $v := (\frac{1}{n}, \dots, \frac{1}{n})$ be the vector of the uniform distribution on V and let P_W be a diagonal $n \times n$ matrix with $(P_W)_{uu} = 1$ if $u \in W$ and $(P_W)_{uu} = 0$ otherwise. Note that $P_W^2 = P_W$.

Let A_G be the probability transition matrix of G . Let us denote the spectral norm of a matrix with $\|\cdot\|$. We bound the probability of a random walk staying in W on indices of M using a standard technique. In particular, we use (for the proof see [30, Claim 4.21]):

► **Claim 4.6.**

$$\|P_W A_G^k P_W\| \leq \mu + (1 - \mu)\lambda^k.$$

Fix M . First of all, by induction (and noting that $v A_G = v$):

$$\Pr_{r \leftarrow P_r} [\forall i \in M : v_i \in W] = |v P_W \prod_{i=2}^m A_G^{d_i} P_W|_1.$$

Estimate:

$$|v P_W \prod_{i=2}^m A_G^{d_i} P_W|_1 \leq \sqrt{\mu n} \cdot \|v P_W \prod_{i=2}^m A_G^{d_i} P_W\| \tag{2}$$

$$\leq \sqrt{\mu n} \cdot \|v P_W\| \prod_{i=2}^m \|P_W A_G^{d_i} P_W\| \tag{3}$$

$$= \mu \prod_{i=2}^m \|P_W A_G^{d_i} P_W\| \tag{4}$$

$$\leq \prod_{i=1}^m (\mu + \lambda^{d_i}), \tag{5}$$

where (2) is due to Cauchy-Schwarz inequality (note there are at most μn non-zero coordinates in the final vector), (3) follows from $\|AB\| \leq \|A\| \cdot \|B\|$, (4) from $\|v P_W\| = \sqrt{\frac{\mu}{n}}$ and (5) from Claim 4.6. ◀

The hope is that (d_1, \dots, d_m) behave “almost” like i.i.d. uniform random variables. This is indeed true, and by Corollary 4.4 we have (e_1, \dots, e_m) such that $e_i \leq d_i$ and e_i are i.i.d. with e_i in \mathbb{N}_+ and $\Pr[e_i = k] \leq \frac{2m}{\ell}$ for each k .

Putting this fact together with Lemma 4.5:

$$\begin{aligned} \Pr_{\substack{r \leftarrow P_r \\ M \leftarrow \binom{[\ell]}{m}}} [\forall i \in M : v_i \in W] &\leq \mathbb{E} \left[\prod_{i=1}^m (\mu + \lambda^{e_i}) \right] \\ &= \prod_{i=1}^m (\mu + \mathbb{E}[\lambda^{e_i}]) \\ &\leq \left(\mu + \frac{2m}{\ell} \cdot \frac{\lambda}{1-\lambda} \right)^m \leq \mu^m (1 + \epsilon)^m. \end{aligned}$$

◀

An immediate corollary of Theorem 4.1 is:

► **Corollary 4.7.** *Let the setting be as in Theorem 4.1. Define P_x over $\{0, 1\}^\ell$ as $x_i = 1 \iff v_i \in W$. Then, P_x is $(\epsilon, \min(\ell, \lfloor \frac{1-\lambda}{\lambda} \frac{\epsilon \mu \ell}{2} \rfloor))$ -growth bounded without repetition.*

Proof of Theorem 4.2. : Combine Corollary 4.7 with Corollary 3.6 (setting $m := \lfloor \frac{(1-\lambda)\epsilon\mu\ell}{6} \rfloor$).

◀

5 Polynomial Concentration

In certain applications it is desired to bound the concentration not only of the sum, but rather of a (low-degree) polynomial of some random variables.

In the case when (informally) the polynomial is such that the change in its value is bounded when the value of a single input variable is changed the Azuma’s inequality can be applied to bound concentration.

If this is not so, one can use techniques that were invented by Kim and Vu [22] and developed in a body of work that followed (in particular [31, 29]). In the special case of a multilinear low-degree polynomial $p(v)$ and an independent distribution of input variables P_v their concentration bound can be expressed, very roughly speaking, as a function of $\frac{\mu_0}{\mu'}$, where μ_0 is the expectation of $p(v)$ and $\mu' = \max_{K \neq \emptyset} \mathbb{E}[\partial_K p(v)]$.

We obtain a bound in similar spirit. It is not tight in general, but can be applied to arbitrary polynomials with positive coefficients over input random variables in $[0, 1]$ and is tight in the case of *elementary symmetric polynomials* $e_k(v) := \sum_{|S|=k} \prod_{i \in S} v_i$ (see the full paper for a proof).

Most importantly, as opposed to prior results, it does not require the input variables to be independent, but rather *almost independent* in a certain sense (for simplicity we limit ourselves to multilinear polynomials and inputs in $\{0, 1\}$, full treatment can be found in the full paper):

► **Definition 5.1.** Let P_v be a distribution over $\{0, 1\}^\ell$, $\delta \geq 0$ and $m \in [\ell]$. P_v is (δ, m) -almost independent if for each $M \subseteq [\ell]$ with $|M| \leq m$

$$\Pr_{v \leftarrow P_v} [\forall i \in M : v_i = 1] \leq (1 + \delta)^m \prod_{i \in M} \Pr_{v \leftarrow P_v} [v_i = 1].$$

Let us state our main theorem of this section.

Let P_v be a (δ, km) -almost independent distribution. Let $p(v)$ be a multilinear polynomial of degree k with positive coefficients. Our way to deal with dependencies in P_v is to state the bound in terms of P_v^* which is the distribution of independent variables with the same marginals as P_v (i.e., each v_i^* has the same distribution as v_i).

We express the concentration in terms of

$$\mu_i^* := \max_{\substack{K \subseteq [l] \\ |K|=i}} \mathbb{E}_{v \leftarrow P_v^*} [\partial_K p(v)].$$

Note that μ_0^* is the expectation of $p(v)$ under P_v^* .

► **Theorem 5.2.** *Let the setting be as above and $\epsilon > 0$. Then,*

$$\Pr_{v \leftarrow P_v} \left[p(v) \geq \mu_0^* (1 + \epsilon) \right] \leq \left(\frac{(1 + \delta)^k \left(1 + \frac{\sum_{i=1}^k \binom{km}{i} \mu_i^*}{\mu_0^*} \right)}{1 + \epsilon} \right)^m.$$

Proof outline. Write $p(v)$ as a sum of binary random variables (corresponding to the monomials) x_1, \dots, x_n . Due to Theorem 3.2 it is enough to show that (x_1, \dots, x_n) are (δ', m) -growth bounded, where $1 + \delta' = (1 + \delta)^k \left(1 + \frac{\sum_{i=1}^k \binom{km}{i} \mu_i^*}{\mu_0^*} \right)$.

Since P_v is (δ, km) -almost independent, this task can be further reduced to showing that if v is distributed according to P_v^* instead of P_v , then (x_1, \dots, x_n) are (δ'', m) -growth bounded, where $1 + \delta'' = \left(1 + \frac{\sum_{i=1}^k \binom{km}{i} \mu_i^*}{\mu_0^*} \right)$.

Fix $s < m$ and $(i_1, \dots, i_s) \in [n]^s$ and let M be the set of all indices j such that v_j influences at least one of x_{i_1}, \dots, x_{i_s} (note that $|M| \leq km$).

We write $p(v) = \sum_{K \subseteq M: |K| \leq k} p_K(v)$, where $p_K(v)$ consists of those monomials whose variables intersected with M are exactly K . Observe that

$$\mathbb{E}_{v \leftarrow P_v^*} \left[p_K(v) \mid \forall i \in M : v_i = 1 \right] \leq \mathbb{E}_{v \leftarrow P_v^*} \left[\partial_K p(v) \right].$$

To get growth boundedness for x_1, \dots, x_n we proceed by induction and bound

$$\begin{aligned} \Pr_{\substack{v \leftarrow P_v^* \\ i_{s+1} \leftarrow [n]}} \left[x_{i_{s+1}} = 1 \mid \forall j \in [s] : x_{i_j} = 1 \right] &= \frac{1}{n} \mathbb{E}_{v \leftarrow P_v^*} \left[p(v) \mid \forall i \in M : v_i = 1 \right] \\ &\leq \frac{1}{n} \sum_{K \subseteq M: |K| \leq k} \mathbb{E}_{v \leftarrow P_v^*} \left[\partial_K p(v) \right] \\ &\leq \frac{\mu_0^*}{n} \left(1 + \frac{\sum_{i=1}^k \binom{km}{i} \mu_i^*}{\mu_0^*} \right). \end{aligned}$$

◀

Let $\mu' := \max_{i \in [k]} \mu_i^*$. Since $\sum_{i=1}^k \binom{km}{i} \leq (km)^k$, we have:

► **Corollary 5.3.** *Let the setting be as in Theorem 5.2. Then,*

$$\Pr_{v \leftarrow P_v} \left[p(v) \geq \mu_0^* (1 + \epsilon) \right] \leq \left(\frac{(1 + \delta)^k \left(1 + \frac{(km)^k \mu'}{\mu_0^*} \right)}{1 + \epsilon} \right)^m.$$

5.1 An application in [14]

In [14] the authors prove a lower bound on the complexity of a black-box construction of a pseudorandom generator from a one-way function.

Part of their proof consists in using Theorem 5.2 to show a concentration bound for a certain polynomial. The proof of Theorem 5.2 is not included in [14], but deferred to this paper instead. Since the input variables of the polynomial are not independent, to the best of our knowledge no previous work is applicable to this case.⁴

The following random process is considered: pick a permutation $f : \{0, 1\}^n \rightarrow \{0, 1\}^n$ u.a.r. and consider the distribution P_g over 2^{2n} random variables $g := \{g_{x,y} : x, y \in \{0, 1\}^n\}$ defined as $g_{x,y} = 1$ if $f(x) = y$ and $g_{x,y} = 0$ otherwise.

The random variables in g are not independent, but it is easy to check that they are $(1, 2^{n-1})$ -almost independent. Also, the corresponding independent distribution P_g^* has expectation 2^{-n} for each $g_{x,y}$.

Fix $k \leq \frac{n}{100 \log n}$. [14] defines a certain multilinear polynomial $p(g)$ of degree at most k such that $\mu_0^* \leq 2^{n/15}$ and $\mu' \leq 2^{n/15}$ (we omit the details).

[14] needs to show that (for n big enough):

$$\Pr_{g \leftarrow P_g} [p(g) \geq 2^{n/10}] \leq 2^{-2^{n/100k}}.$$

To this end, calculate using Corollary 5.3 and setting $\delta := 1$, $\epsilon := 2^{9n/100}/\mu_0^*$ and $m := 2^{n/100k}$:

$$\begin{aligned} \Pr_{g \leftarrow P_g} [p(g) \geq \mu_0^* + 2^{9n/100}] &\leq \left(\frac{2^k \max \left(2, \frac{2^{k^k} 2^{n/100} \mu'}{\mu_0^*} \right)}{\frac{2^{9n/100}}{\mu_0^*}} \right)^{2^{n/100k}} \\ &\leq \left(\frac{2^{k+1} \max (\mu_0^*, k^k 2^{n/100} \mu')}{2^{9n/100}} \right)^{2^{n/100k}} \\ &\leq 2^{-2^{n/100k}}. \end{aligned}$$

5.2 Other applications

We note that despite the fact that the deviation for which we applied our theorem in Section 5.1 is big relative to the expectation, one can obtain meaningful bounds also for very small deviations.

This can be seen by taking a restricted version of Theorem 5.2:

► **Theorem 5.4.** *Let P_v be a distribution of independent variables (i.e., $P_v = P_v^*$) over $[0, 1]^\ell$. Let $p(v)$ be as in Theorem 5.2 and $\epsilon \in [0, \frac{1}{2}]$. Then:*

$$\Pr_{v \leftarrow P_v} [p(v) \geq \mu(1 + \epsilon)] \leq 2 \exp \left(- \frac{\epsilon}{6k} \left(\frac{\epsilon \mu}{\mu'} \right)^{1/k} \right).$$

Proof. Note that P_v are $(0, \ell)$ -almost independent. Take $m := \left\lfloor \frac{1}{k} \left(\frac{\epsilon \mu}{3 \mu'} \right)^{1/k} \right\rfloor$, obtain $(\frac{\epsilon}{3}, m)$ -growth boundedness as in Corollary 5.3 and apply Corollary 3.3.1. ◀

⁴ It was pointed out to us that a generalisation of the result of Latała and Łochowski [23] might be applicable (together with [9]). However, moment bound in [23] is optimal only up to a constant in the exponent that depends on the degree and the degree is non-constant in our setting.

For example, in a representative setting when Azuma-like methods fail: consider the polynomial that counts the triangles in Erdős–Rényi random graph $\mathbb{G}_{n,n^{-3/4}}$, i.e., $p(v) = \sum_{\{a,b,c\} \in \binom{[n]}{3}} v_{ab}v_{ac}v_{bc}$. We compute $\mu = \Theta(n^{3/4})$ and $\mu' = \Theta(1)$.

For $\epsilon \in [0, \frac{3}{16}]$ Theorem 5.4 gives:

$$\Pr_{v \leftarrow \mathbb{P}_v} \left[p(v) \geq \mu(1 + n^{-\epsilon}) \right] \leq \exp(-\Omega(n^{1/4-4\epsilon/3})).$$

This is comparable to the bound from [22] (which was the first paper to give a good bound in this setting). Better bounds are known, in particular we revisit the triangle counting in Section 6.

For some more discussion on the tightness of Theorem 5.2, see the full paper.

6 Counting Subgraphs in Random Graphs

In the proof of the polynomial concentration bound we consider values μ_i^* which are maxima of expectations of $\partial_K p(v)$ over sets K of size i . Each such value yields a contribution⁵ of $\binom{km}{i} \mu_i^*$ (proportional to the number of partial derivatives of this type in the subset of input variables of size km) and the “quality” of a concentration bound depends, roughly, on the maximum such contribution.

In principle, nothing prevents us from considering a different, possibly finer, division of partial derivatives into a constant number of classes, each with its own contribution.

In particular, it is an obvious fact that the number of occurrences of a fixed subgraph H in a random Erdős–Rényi graph (for some of the work on the problem see [18, 17, 19]) can be expressed in terms of a multilinear polynomial. In this setting we may divide the partial derivatives into classes corresponding to subgraphs of H . Interestingly, this yields an upper tail bound proof that is basically isomorphic to the famous one of Janson, Oleszkiewicz and Ruciński [17].

Our result holds in the setting of almost-independent distributions, readily applicable, for example, to $\mathbb{G}_{n,m}$ random graphs (of course the proof of [17] also generalises to those settings).

For details, see the full paper.

References

- 1 Carlos A. León and François Perron. Optimal Hoeffding bounds for discrete reversible Markov chains. *The Annals of Applied Probability*, 14(2):958–970, 05 2004.
- 2 Miklós Ajtai, János Komlós, and Endre Szemerédi. Deterministic simulation in LOGSPACE. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 132–140, New York, NY, USA, 1987. ACM.
- 3 Noga Alon, Uriel Feige, Avi Wigderson, and David Zuckerman. Derandomized graph products. *Computational Complexity*, 5(1):60–75, 1995.
- 4 Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.
- 5 Kazuoki Azuma. Weighted sums of certain dependent random variables. *Tôhoku Math. J. (2)*, 19:357–367, 1967.
- 6 Sergei N. Bernstein. On a modification of Chebyshev’s inequality and of the error formula of Laplace. *Ann. Sci. Inst. Sav. Ukraine, Sect. Math.*, 1, 1924.

⁵ Think of a constant k and a family of polynomials with m going to infinity.

- 7 Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):pp. 493–507, 1952.
- 8 Kai-Min Chung, Henry Lam, Zhenming Liu, and Michael Mitzenmacher. Chernoff-Hoeffding bounds for Markov chains: Generalized and simplified. In Christoph Dürr and Thomas Wilke, editors, *STACS*, volume 14 of *LIPICs*, pages 124–135. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.
- 9 Victor H. de la Peña and S. J. Montgomery-Smith. Bounds on the tail probability of U-statistics and quadratic forms. *Bulletin of the American Mathematical Society*, 31(2):223–227, 1994.
- 10 Devdatt Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.
- 11 David Gillman. A Chernoff bound for random walks on expander graphs. *SIAM J. Comput.*, 27(4):1203–1220, 1998.
- 12 Alexander Healy. Randomness-efficient sampling within NC^1 . *Computational Complexity*, 17(1):3–37, 2008.
- 13 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):pp. 13–30, 1963.
- 14 Thomas Holenstein and Makrand Sinha. Constructing a pseudorandom generator requires an almost linear number of calls. In *FOCS*, pages 698–707. IEEE Computer Society, 2012.
- 15 Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bulletin of the AMS*, 43(4):439–561, 2006.
- 16 Russell Impagliazzo and Valentine Kabanets. Constructive proofs of concentration bounds. In Maria J. Serna, Ronen Shaltiel, Klaus Jansen, and José D. P. Rolim, editors, *APPROX-RANDOM*, volume 6302 of *Lecture Notes in Computer Science*, pages 617–631. Springer, 2010.
- 17 Svante Janson, Krzysztof Oleszkiewicz, and Andrzej Ruciński. Upper tails for subgraph counts in random graphs. *Israel Journal of Mathematics*, 142(1):61–92, 2004.
- 18 Svante Janson and Andrzej Ruciński. The infamous upper tail. *Random Struct. Algorithms*, 20(3):317–342, 2002.
- 19 Svante Janson and Andrzej Ruciński. Upper tails for counting objects in randomly induced subhypergraphs and rooted random graphs. *Arkiv för matematik*, 49(1):79–96, 2011.
- 20 Nabil Kahalé. Eigenvalues and expansion of regular graphs. *J. ACM*, 42(5):1091–1106, September 1995.
- 21 Nabil Kahalé. Large deviation bounds for Markov chains. *Combinatorics, Probability & Computing*, 6(4):465–474, 1997.
- 22 Jeong Han Kim and Van H. Vu. Concentration of multivariate polynomials and its applications. *Combinatorica*, 20(3):417–434, 2000.
- 23 Rafał Łatała and Rafał Łochowski. Moment and tail estimates for multidimensional chaoses generated by positive random variables with logarithmically concave tails. *Progr. Probab.*, 56:77–92, 2003.
- 24 Pascal Lezaud. Chernoff-type bound for finite Markov chains. *Ann. Appl. Probab.*, 8(3):849–867, 1998.
- 25 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.
- 26 Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, Cambridge, New York, Melbourne, 1995. Réimpressions : 1997, 2000.
- 27 Anup Rao. Parallel repetition in projection games and a concentration bound. In *In Proc. 40th STOC*, pages 1–10. ACM, 2008.

- 28 Jeanette P. Schmidt, Alan Siegel, and Aravind Srinivasan. Chernoff-Hoeffding bounds for applications with limited independence. *SIAM J. Discret. Math.*, 8(2):223–250, May 1995.
- 29 Warren Schudy and Maxim Sviridenko. Concentration and moment inequalities for polynomials of independent random variables. In Yuval Rabani, editor, *SODA*, pages 437–446. SIAM, 2012.
- 30 Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012.
- 31 V. H. Vu. Concentration of non-Lipschitz functions and applications. *Random Struct. Algorithms*, 20(3):262–316, May 2002.
- 32 Roy Wagner. Tail estimates for sums of variables sampled by a random walk. *Combinatorics, Probability and Computing*, 17:307–316, 3 2008.

Minimum Cost Flows in Graphs with Unit Capacities

Andrew V. Goldberg^{*1}, Haim Kaplan², Sagi Hed², and Robert E. Tarjan^{*3,4}

- 1 Amazon.com Inc.
- 2 School of Computer Science, Tel Aviv University
- 3 Department of Computer Science, Princeton University
- 4 Intertrust Technologies, Sunnyvale CA

Abstract

We consider the minimum cost flow problem on graphs with unit capacities and its special cases. In previous studies, special purpose algorithms exploiting the fact that capacities are one have been developed. In contrast, for maximum flow with unit capacities, the best bounds are proven for slight modifications of classical blocking flow and push-relabel algorithms.

In this paper we show that the classical cost scaling algorithms of Goldberg and Tarjan (for general integer capacities) applied to a problem with unit capacities achieve or improve the best known bounds. For weighted bipartite matching we establish a bound of $O(\sqrt{r}m \log C)$ on a slight variation of this algorithm. Here r is the size of the smaller side of the bipartite graph, m is the number of edges, and C is the largest absolute value of an arc-cost. This simplifies a result of [Duan et al. 2011] and improves the bound, answering an open question of [Tarjan and Ramshaw 2012]. For graphs with unit vertex capacities we establish a novel $O(\sqrt{nm} \log(nC))$ bound. We also give the first cycle canceling algorithm for minimum cost flow with unit capacities. The algorithm naturally generalizes the single source shortest path algorithm of [Goldberg 1995].

1998 ACM Subject Classification E.1 Data Structures, F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases minimum cost flow, bipartite matching, unit capacity, cost scaling

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.406

1 Introduction

The input to the *minimum cost flow (MCF)* problem is a directed graph $G = (V, E)$ with integer arc costs and integer capacities. The goal is to find a circulation (flow) of minimum cost, where a circulation is a function on arcs that satisfies capacity constraints for all arcs and conservation constraints for all vertices. We denote $|V|$ by n , $|E|$ by m , the maximum capacity by U and the maximum cost value by C .¹ MCF has many applications (see e.g., [1]) and has been extensively studied.

An important subclass of MCF problems are the problems with unit capacities (*UMCF*). In particular, the *weighted bipartite matching (WBM)* problem is a special case of UMCF. The goal in WBM is to find a matching of maximum weight in a weighted bipartite graph. The following two problems are closely related to WBM and are also a special case of UMCF. The *assignment (AS)* problem is to find a perfect matching of maximum weight, and the

* Part of the work was done while the author was at Microsoft Research

¹ When writing $\log U$ and $\log C$, we assume that $U > 1$ and $C > 1$, respectively.

imperfect assignment (IAS) problem is to find a maximum weight matching of a given size F . The *single source shortest path (SSSP)* problem can be reduced to AS. Another special case is the UMCF problem on graphs with unit vertex capacities (or equivalently with an out-degree or an in-degree of one for every vertex).

The unit capacity maximum flow problem and the maximum bipartite matching problem can be reduced, respectively, to the UMCF and WBM problems with most arc costs zero and some arc costs plus or minus one. The best combinatorial bounds for these problems are $O(\min(m^{1/2}, n^{2/3})m)$ and $O(\sqrt{nm})$, respectively [13, 7]. Recently, Madry [15] proved an $\tilde{O}(m^{10/7})$ bound using interior point methods. Better combinatorial bounds are known for important special cases of the maximum bipartite matching problem when the flow value (which is the size of the matching) F is small and when the smaller of the two sizes of the input bipartite graph, r , is small. The bound for the former case is $O(\sqrt{F}m)$ [12] and for the latter case, $O(\sqrt{r}m)$ [2]. The best combinatorial bound for maximum flow with unit vertex capacities is $O(\sqrt{nm})$ [7].

Many of the results on unit-capacity maximum flow have been extended to UMCF. Gabow and Tarjan [8] showed an $O(\min(m^{1/2}, n^{2/3})m \log(nC))$ bound for UMCF and $O(\sqrt{nm} \log(nC))$ bound for WBM, AS, and IAS problems. Goldberg [9] gave an $O(\sqrt{nm} \log C)$ algorithm for SSSP, and Duan et al. [6] gave an algorithm for WBM with the same complexity. Ramshaw and Tarjan [16] developed an $O(\sqrt{r}m \log(rC))$ algorithm for WBM and an $O(\sqrt{F}m \log(FC))$ algorithm for IAS. However, these extensions are ad hoc: the algorithms are not MCF algorithms, and their connection to common MCF algorithms is not straightforward.

In this paper we develop a unified MCF framework for problems with unit and small integer capacities. The resulting algorithms are simple, intuitive, and match or improve the previous bounds. In essence we show that classical algorithms with slight modifications and a careful analysis give the same or better bounds for UMCF problems as special purpose algorithms.

We show, in fact, two unified frameworks for UMCF. The first, in Section 4, is a novel cycle canceling algorithm extending [9], which is the first cycle canceling algorithm that has good time bounds for UMCF problems. The second, in Section 3, is a pseudoflow framework which consists of the MCF cost-scaling algorithms of [11] with a fresh analysis for UMCF problems. Section 5 shows how to apply these frameworks to the special cases of UMCF. When applied to these special cases, the algorithms stay almost the same but require a more careful analysis.

Our paper unifies (or replaces) [11, 8, 16, 9] and the second part of [6] and obtains the following results:

- A novel $O(\sqrt{r}m \log(C))$ time bound for WBM, answering an open question of [16].²
- A novel $O(\sqrt{nm} \log(nC))$ time bound for UMCF with unit vertex capacities, which is the first extension of Even and Tarjan [7] from maximum flow to MCF.
- All the previous time bounds for UMCF, AS, IAS and SSSP. Namely $O(\min(m^{1/2}, n^{2/3})m \log(nC))$ for UMCF³, $O(\sqrt{nm} \log(nC))$ for AS, $O(\sqrt{F}m \log(FC))$ for IAS and $O(\sqrt{nm} \log C)$ for SSSP.

² Note that $r < n$, so this bound improves both the $O(\sqrt{nm} \log C)$ bound of Duan et al. [6] and the $O(\sqrt{r}m \log(rC))$ bound of Ramshaw and Tarjan [16].

³ And an $O(\min(n, \sqrt{mU}, n^{2/3}U^{1/3})m \min(U, \log n) \log(nC))$ bound for MCF.

2 Definitions and Notation

The input to the MCF problem is a directed graph $G = (V, E)$ where $|V| = n$ and $|E| = m$. Each arc $(v, w) \in E$ has an associated integer capacity $0 \leq u(v, w) \leq U$ and an associated integer cost $c(v, w)$ such that $|c(v, w)| \leq C$. For every input arc e , we add to E a *reverse arc* e^R with $u(e^R) = 0$ and $c(e^R) = -c(e)$. We also define $(e^R)^R = e$. To simplify notation, we usually assume that the graph has no parallel or anti-parallel arcs, so every arc is uniquely defined by its endpoints, and $(v, w)^R = (w, v)$.

A *circulation* f is a function $f : E \rightarrow R$ which is anti-symmetric: $\forall_{(v,w) \in E} f(v, w) = -f(w, v)$, satisfying the capacity constraints $\forall_{(v,w) \in E} f(v, w) \leq u(v, w)$ and the flow conservation constraints $\forall_{v \in V} \sum_{w \in V | (v,w) \in E} f(v, w) = 0$. For any flow function f we define $\text{cost}(f) = \frac{1}{2} \sum_{(v,w) \in E} f(v, w) \cdot c(v, w)$. A circulation is *optimal* if its cost is minimal. The *MCF problem* is to find an optimal circulation in G . The *MCF problem with vertex capacities of F* has the added constraint that $\forall_{v \in V} \sum_{(v,w) \in E, f(v,w) > 0} f(v, w) \leq F$.

A *pseudoflow* f is a function $f : E \rightarrow R$ satisfying $f(v, w) = -f(w, v)$ and $f(v, w) \leq u(v, w)$ for every arc $(v, w) \in E$. We define $E_f = \{(v, w) \in E \mid f(v, w) < u(v, w)\}$ as the set of *residual arcs*. For a vertex v we define $e_f(v) = \sum_{(w,v) \in E} f(w, v)$. We call a vertex v with $e_f(v) > 0$ an *excess* and a vertex v with $e_f(v) < 0$ a *deficit*. Note that a circulation is just a pseudoflow with no excesses or deficits.

Given a *potential function* $p : V \rightarrow R$ we call $c_p(v, w) = c(v, w) - p(v) + p(w)$ the *reduced cost* of the arc (v, w) . We say that a pseudoflow (or a circulation) f with potentials p is ϵ -*optimal* if for every arc $(v, w) \in E_f$ we have $c_p(v, w) \geq -\epsilon$. By linear programming duality, a 0-optimal circulation is optimal. We define $E_A = \{(v, w) \in E_f \mid c_p(v, w) < 0\}$ as the set of *admissible arcs*. We define $G_A = (V, E_A)$ as the *admissible subgraph*.

A graph G is *bipartite* if we can partition V into V_1 and V_2 such that $E \subseteq V_1 \times V_2$. A bipartite graph G is *balanced* if $|V_1| = |V_2|$. Given a bipartite graph, a *matching* is a set of edges that do not have common vertices. A *perfect matching* is a matching such that any vertex is adjacent to an edge in the matching. If the edges have weights, the weight of a matching is the sum of the edge weights. The WBM problem is to compute a matching with maximum weight among all matchings. The AS problem is to compute a perfect matching in a bipartite graph with maximum weight among all perfect matchings.⁴ The IAS problem is to compute a matching of size F in a bipartite graph with maximum weight among all such matchings. In these problems we use C as the maximum weight of an edge in the bipartite graph. The WBM, AS, and IAS problems can be formulated as UMCF problems (see Section 5).

A circulation is optimal if and only if there exists a potential function such that all residual arcs have non-negative reduced costs [14]. Another optimality condition is that a circulation is optimal if and only if the residual graph has no cycles of negative cost.

The cost-scaling method was introduced in [3, 17]. We use the more efficient variant due to Goldberg and Tarjan [11]. The cost-scaling method proceeds in iterations which we call *cost scales*; each one starts with a 2ϵ -optimal circulation and ends with an ϵ -optimal circulation. It starts with the zero circulation which is C -optimal. When $\epsilon < 1/n$, an ϵ -optimal circulation is optimal, so the method takes $O(\log(nC))$ iterations. For efficiency, we restrict ourselves to values of ϵ that are integral powers of two. Unless mentioned otherwise, we use $2^{\lceil \log C \rceil}$ as the initial value of ϵ . We assume that the reader is familiar with Dijkstra's algorithm [5] and its bucket-based implementation due to Dial [4].

⁴ Note that a perfect matching exists only in a balanced bipartite graph.

Algorithm 1 The cost scaling blocking flow algorithm of Goldberg and Tarjan [11].

<pre> function MCF(G) $\epsilon \leftarrow C, p \leftarrow 0, f \leftarrow 0$ while $\epsilon \geq \frac{1}{n}$ do $(\epsilon, f, p) \leftarrow \text{REFINE}(\epsilon, f, p)$ end while return f end function function RAISE-POTENTIALS(ϵ, f, p) while $\nexists E_A$-path from excess to deficit do $S \leftarrow \{v \in V \mid \exists E_A\text{-path from an excess to } v\}$ $\forall v \in S : p(v) \leftarrow p(v) + \epsilon$ end while end function </pre>	<pre> function REFINE(ϵ', f', p') $\epsilon \leftarrow \frac{\epsilon'}{2}, p \leftarrow p', f \leftarrow f'$ $\forall (v, w) \in E_A : f(v, w) \leftarrow u(v, w)$ while f is not a circulation do RAISE-POTENTIALS(ϵ, f, p) $f \leftarrow f + \text{a blocking flow in } G_A$ end while return (ϵ, f, p) end function </pre>
--	--

3 Pseudoflow Framework

Our pseudoflow framework for UMCF is based on the cost scaling algorithms of [11]. In this section we analyze the blocking flow algorithm of [11] for UMCF. One can get similar bounds for the push-relabel variant of [11]; we defer this result to the full version of the paper.

Algorithm 1 describes the blocking flow cost scaling algorithm of [11]. We say f is a *blocking flow* in G_A if f is a pseudoflow that: (i) Satisfies conservation constraints everywhere except at the excesses and deficits of G_A . (ii) Saturates at least one arc on every residual path from an excess to a deficit in G_A .

The algorithm starts a cost scale (the `refine` method in Algorithm 1) by saturating all admissible arcs, which turns the circulation into a pseudoflow. It then performs the following two steps until the pseudoflow becomes a circulation:

1. Iteratively increase by ϵ the potential of all vertices reachable from some excess in G_A , creating new admissible arcs. Stop when there is an admissible excess-to-deficit path.
2. Compute a blocking flow in G_A and add it to the current pseudoflow.

We implement step (1) (the `raise-potentials` method in Algorithm 1) with a modification that makes its running time linear, as described in Section 3.1. To analyze step (2), recall that Goldberg and Tarjan [11] show that G_A is acyclic throughout the execution of the algorithm. Since G_A also has unit capacities, we can compute a blocking flow in G_A in $O(m)$ time by a depth-first search in G_A .

Let f' and p' be the circulation and potentials at the start of the current cost scale (f' with p' is 2ϵ -optimal). Let f and p be the current pseudoflow and potentials maintained by the algorithm (f is ϵ -optimal w.r.t. p). Let $\lambda = \min(m^{1/2}, n^{2/3})$. Let $d(v) = \frac{p(v) - p'(v)}{\epsilon}$. Note that $d(v)$ must be integral. We divide every cost scale into two phases. The second phase begins when every excess v has $d(v) \geq \lambda$.

► **Lemma 1.** *The first phase of every cost scale runs in $O(m\lambda)$ time.*

Proof. After every blocking flow computation, for every excess v , $d(v)$ increases by at least one. It follows that for every excess v , $d(v) \geq \lambda$ after at most λ blocking flow computations. Since both step (1) and step (2) take $O(m)$ time, the lemma follows. ◀

► **Lemma 2.** *If k is the total excess at the start of the second phase, then the second phase takes $O(mk)$ time.*

Proof. Every blocking flow computation sends at least one unit of flow from some excess to some deficit. Since both step (1) and step (2) take $O(m)$ time, the lemma follows. ◀

Lemma 1 and 2 imply an $O(\lambda m)$ time bound on the Goldberg-Tarjan algorithm on UMCF, if we can show that when the second phase starts there is $O(\lambda)$ excess left. We introduce the following notation. We define $E^+ = \{(v, w) \in E \mid f'(v, w) > f(v, w)\}$ and let $G^+ = (V, E^+)$ be the subgraph of G induced by the arcs of E^+ . Note that if $(v, w) \in E^+$ then $(v, w) \in E_f$ and $(w, v) \in E_{f'}$. We say $Y \subseteq E$ is an *excess-deficit cut* if every path from an excess to a deficit contains an arc in Y .

Lemma 3 and 4 are the core of our analysis of the pseudoflow framework. They allow us to extend the analysis of maximum flow algorithms to MCF by applying the classical arguments to E^+ rather than to E_f . Lemma 4 is an extension of Lemma 5.7 of [11]. It essentially states that an excess-deficit cut in E^+ bounds the total amount of excess, just as a cut on E_A does. Lemma 5 will then conclude the analysis.

► **Lemma 3.** *If $(v, w) \in E^+$ then $d(v) \leq d(w) + 3$.*

Proof. Since $(w, v) \in E_{f'}$ then $c_{p'}(w, v) \geq -\epsilon' = -2\epsilon$ so we have $c_{p'}(v, w) \leq 2\epsilon$. Since $(v, w) \in E_f$ we maintain the invariant $c_p(v, w) \geq -\epsilon$. We subtract the inequalities and get $c_p(v, w) - c_{p'}(v, w) \geq -\epsilon - 2\epsilon$ and so $d(w)\epsilon - d(v)\epsilon \geq -3\epsilon$. It follows that $d(v) \leq d(w) + 3$. ◀

► **Lemma 4.** *Let f' and f be a circulation and pseudoflow in $G = (V, E)$. Let Y be an excess-deficit cut in G^+ . Then*

$$\sum_{v \in V, e_f(v) > 0} e_f(v) \leq \sum_{(v, w) \in Y} (f'(v, w) - f(v, w)) \leq \sum_{(v, w) \in Y} u_f(v, w).$$

Proof. The flow $f' - f$ is a feasible flow in G_f since for every arc $(v, w) \in E_f$ we have $f'(v, w) - f(v, w) \leq u(v, w) - f(v, w) = u_f(v, w)$. By definition we get that $f + (f' - f) = f'$. Since f' is a circulation, we get that $f' - f$ drains all the excess in f . ◀

► **Lemma 5.** *When the second phase starts there is $O(\lambda)$ excess left.*

Proof. During the second phase every excess v has $d(v) \geq \lambda$ and every deficit v has $d(v) = 0$.

Assume $\lambda = \sqrt{m}$. Consider the $\frac{1}{3}\lambda$ sets of arcs $A_i = \{(v, w) \in E^+ \mid d(v) \in \{3i, 3i - 1, 3i - 2\} \wedge d(w) \in \{3i - 3, 3i - 4, 3i - 5\}\}$ where $1 \leq i \leq \frac{\lambda}{3}$.⁵ By Lemma 3 any set A_i is an excess-deficit cut in G_f . By the pigeon hole principle there must be a set A_i such that $|A_i| \leq \frac{m}{\lambda/3} = 3\sqrt{m}$. So by Lemma 4 the remaining excess is at most $3\sqrt{m} = 3\lambda$.

Assume $\lambda = n^{2/3}$. We apply the pigeon hole principle to the sets of nodes $N_i = \{v \in V \mid d(v) \in \{6i, 6i - 1, 6i - 2, 6i - 3, 6i - 4, 6i - 5\}\}$ where $1 \leq i \leq \frac{\lambda}{6}$.⁶ There must be a set N_i such that $|N_i| \leq \frac{n}{\lambda/6} = 6n^{1/3}$. So there are at most $36n^{2/3}$ arcs in A_{2i} and therefore by Lemma 4 the remaining excess is at most $36n^{2/3} = 36\lambda$. ◀

Our analysis extends to networks with integer capacities bounded by U and produces a time bound of $O(\min\{n, m^{1/2}U^{1/2}, n^{2/3}U^{1/3}\}m \min\{\log n, U\} \log(nC))$. We defer an extended description to the full version of the paper.

3.1 Raise Potentials

We describe a faster implementation of **raise-potentials** from Algorithm 1 using Dial's shortest path algorithm. This kind of implementation is well known, see e.g. [8, 9, 16].

⁵ Note that $A_1 = \{(v, w) \in E^+ \mid d(v) \in \{3, 2, 1\} \wedge d(w) = 0\}$. We assume for simplicity of presentation that $\frac{\lambda}{3}$ is an integer.

⁶ Here we assume for simplicity that $\frac{\lambda}{6}$ is an integer.

Consider the length function $\ell(v, w) = \lfloor \frac{c_p(v, w)}{\epsilon} \rfloor + 1$ and the graph G_f with an additional source vertex r connected to every excess v with an arc (r, v) of length $\ell(r, v) = 0$. For every vertex v reachable from r , we find the distance $d_\ell(v)$ from r to v w.r.t. ℓ . Let u be the deficit with the shortest distance $d_\ell(u)$. For a vertex v such that $d_\ell(v) < d_\ell(u)$ we increase the potential $p(v)$ by $(d_\ell(u) - d_\ell(v))\epsilon$. Lemma 6 shows that this computes the same potentials as in the naive implementation of **raise-potentials** from Algorithm 1.

Lemma 7 shows that we need to compute distances only up to $3n$. This implies that Dial's algorithm runs in linear time and space. We terminate the computation once we reach a deficit; alternatively, we can add an arc (r, v) for every vertex v with $\ell(r, v) = 3n$ and run the algorithm to the end.

► **Lemma 6.** *Consider a run of the naive implementation of **raise-potentials** from Algorithm 1. Let S_i be the set S computed at the i -th iteration, and let $S_0 = \{v \in V : \exists u \text{ with } e_f(u) > 0 \text{ and } v \text{ reachable from } u \text{ in } G_A\}$. Then $S_i = \{v \in V \mid d_\ell(v) \leq i\}$.*

Proof. The proof is by induction on iterations of the naive implementation. For the basis, note that S_0 is the set of vertices reachable from excesses in the admissible graph. If (v, w) is admissible, then $\ell(v, w) = 0$, which implies the claim for S_0 . We assume the claim is true for S_0, \dots, S_{k-1} and prove it for S_k . Note that p and ℓ are the potentials and lengths before running **raise-potentials**.

First, we show that for every vertex v with $d_\ell(v) \leq k$ we have $v \in S_k$. By definition of the naive implementation $S_i \subseteq S_k$ for $i < k$ since all admissible arcs into vertices in S_i in iteration i remain admissible in subsequent iterations. Therefore by the induction hypothesis we need to consider only vertices v with $d_\ell(v) = k$. Let P be the shortest path from r to v (by ℓ distance). Going backwards from v towards r on P let (u, x) be the first arc we encounter with $d_\ell(u) < d_\ell(x) = k$ (possibly $x = v$). It follows that the arcs along P from x to v have zero length ℓ and are therefore admissible. By the induction hypothesis we have that $u \in S_{d_\ell(u)} \setminus S_{d_\ell(u)-1}$. So in the k 'th iteration the reduced cost of (u, x) is $c_p(u, x) - \epsilon \cdot (k - d_\ell(u)) = c_p(u, x) - \epsilon \cdot (d_\ell(x) - d_\ell(u)) = c_p(u, x) - \epsilon \cdot \lfloor \frac{c_p(u, x)}{\epsilon} \rfloor - \epsilon = (c_p(u, x) \bmod \epsilon) - \epsilon < 0$. So (u, x) is admissible in the k 'th iteration and since there is an admissible path from x to v the claim follows.

Finally, we show that for every vertex $v \in S_k$ we have $d_\ell(v) \leq k$. By the induction hypothesis we need to consider only vertices $v \in S_k \setminus S_{k-1}$. By definition of the naive implementation, in the k 'th iteration we found an admissible path P from some vertex in S_{k-1} to v . Let (u, x) be the last arc on P such that $u \in S_{k-1}$ (possibly $x = v$). By the induction hypothesis we get that $u \in S_{d_\ell(u)} \setminus S_{d_\ell(u)-1}$. So the reduced cost of (u, x) in the k 'th iteration is $c_p(u, x) - \epsilon \cdot (k - d_\ell(u)) < 0$. It follows that $\frac{c_p(u, x)}{\epsilon} - (k - d_\ell(u) - 1) < 1$ and so $\lfloor \frac{c_p(u, x)}{\epsilon} - (k - d_\ell(u) - 1) \rfloor \leq 0$. So by definition of shortest distances we get that $d_\ell(x) \leq d_\ell(u) + \ell(u, x) \leq k$. Since the subpath of P from x to v was always admissible we get that $d_\ell(v) \leq d_\ell(x) \leq k$. ◀

► **Lemma 7.** *For every $v \in V$ with $e_f(v) < 0$ we have $d_\ell(v) \leq 3n$.*

Proof. We assume for contradiction that $d_\ell(v) > 3n$. By Lemma 4 there must be an E^+ path from some excess w to v . We run the naive implementation of **raise-potentials** from Algorithm 1 for $d_\ell(v)$ iterations (we do not terminate if we find a deficit). By Lemma 6 after $d_\ell(v)$ iterations we have $d(w) \geq d_\ell(v) > 3n$ and $d(v) = 0$. Applying Lemma 3 telescopically along the E^+ path from w to v we get that $d(w) \leq d(v) + 3n = 3n$ which is a contradiction. ◀

Algorithm 2 A scaling iteration of the UMCF cycle canceling algorithm.

<pre> function REFINE(ϵ', f, p) $\epsilon \leftarrow \frac{1}{2}\epsilon'$ while $E_{bad} > 0$ do $k \leftarrow E_{bad}$ Cancel admissible cycles $S \leftarrow$ FIND-SET-OR-CHAIN() if S is a set then $\forall v \in S : p(v) \leftarrow p(v) + \epsilon$ else ELIMINATE-CHAIN(S) end if end while return (ϵ, f, p) end function function ELIMINATE-CHAIN($S = (v_0, \dots, v_{ S -1})$) $\forall t \in \{1, \dots, S -2\} : \text{PUSH}(v_t, v_{t+1})$ RAISE-POTENTIALS(ϵ, f, p) $\forall (v, w) \in E_A$-path from $v_{ S -1}$ to v_0 : PUSH(v, w) end function </pre>	<pre> function FIND-SET-OR-CHAIN() $\forall (v, w) \in E_{bad} : \ell'(v, w) = -1$ $\forall (v, w) \notin E_{bad} : \ell'(v, w) = 0$ $G' \leftarrow G_A$ with a new vertex r $\forall v \in V$: connect r to v with $\ell'(r, v) = 0$ Compute ℓ' distances from r in G' if $\exists v \in V$ $d_{\ell'}(r, v) \leq -\sqrt{k}$ then return E_A-path from r to v else $\forall_{i < \sqrt{k}} : A_i \leftarrow \{v \in V \mid d_{\ell'}(r, v) = -i\}$ $j \leftarrow \text{argmax}_i \{(v, w) \in E_{bad} \mid w \in A_i\}$ return $\cup_{j \leq i \leq \sqrt{k}} A_i$ end if end function function PUSH(v, w) $f(v, w) \leftarrow f(v, w) + 1, f(w, v) \leftarrow f(w, v) - 1$ end function </pre>
--	--

4

 Cycle Canceling Framework

We present a cycle canceling algorithm that solves the UMCF problem in $O(m^{3/2} \log(nC))$ time. Unlike the methods of the previous section, the cycle-canceling algorithm always maintains a feasible circulation, which is desirable in some contexts. On the other hand, we were unable to prove an $O(mn^{2/3} \log(nC))$ bound as we did in the preflow framework. Our algorithm is based on the shortest path algorithm of Goldberg [9]. The main difference is that the shortest path algorithm halts when it finds a negative cycle and the MCF algorithm cancels such cycles and proceeds.

We introduce more notation. We define $E_{bad} = \{(v, w) \in E \mid c_p(v, w) < -\epsilon\}$ as the set of *bad arcs*. Note that $E_{bad} \subseteq E_A$. For a length function ℓ and vertices $x, y \in V$ we define $d_\ell(x, y)$ to be the distance according to ℓ from x to y . Given $S \subset V$ and $T \subset E$, we say S is *T-closed* if for every arc $(v, w) \in T$, $v \in S$ implies $w \in S$.

We describe the cycle-canceling variant of **refine** in Algorithm 2. We use k to denote the number of bad arcs. We perform iterations of canceling admissible cycles, finding either a large E_A -closed set or a long E_A path and then using the set or path to raise potentials or cancel more cycles and reduce the number of bad arcs. In each iteration we reduce k by at least \sqrt{k} . The **eliminate-chain** method is the main place where our algorithm differs from Goldberg's shortest path algorithm. The latter stops when it finds a negative cycle, while we need to cancel such cycles.

We define an *iteration* of the algorithm as one iteration of the loop of the **refine** method. In every iteration we first cancel all admissible cycles in G . Goldberg and Tarjan [10] describe how to perform this in $O(m)$ time for unit capacity networks. Next we run the **find-set-or-chain** method. This method finds either a path in G_A with at least \sqrt{k} bad arcs or an E_A -closed set of vertices with at least \sqrt{k} incoming bad arcs.

We find the path or the set by creating a new graph G' from G_A with a length function ℓ' as defined in Algorithm 2. Since we canceled admissible cycles in G , the graph G' is acyclic and we find a topological ordering of vertices in G' in $O(m)$ time. Using this ordering, we compute the distance $d_{\ell'}(r, v)$ for every vertex v in $O(m)$ time. If some vertex v has $d_{\ell'}(r, v) \leq -\sqrt{k}$ then the path from r to v is the path we seek. Otherwise, by the pigeon hole principle, there is a set A_i of all the vertices with distance $-i$ that has at least \sqrt{k} incoming bad arcs. The set of all vertices with distance $-i$ or less is therefore an E_A -closed set of

vertices with at least \sqrt{k} incoming bad arcs.

If we find an E_A -closed set S , we increase the potentials of the vertices in S by ϵ . Since S is E_A -closed, raising the potential of S by ϵ does not create new bad arcs out of S . On the other hand each arc into S that was bad before the increase is no longer bad after the increase (f is 2ϵ -optimal so reduced costs are at least -2ϵ). Since S contained at least \sqrt{k} incoming bad arcs, we reduce the number of bad arcs by at least \sqrt{k} .

If we find a path, we run the `eliminate-chain` method. This method eliminates the bad arcs on the path by adjusting potentials and canceling cycles with bad arcs, without creating new bad arcs. The `eliminate-chain` method runs in linear time. See Section 4.1.

Each iteration in `refine` finds either a set or a chain and in either case eliminates at least \sqrt{k} bad arcs. So the number of iterations is $O(\sqrt{m})$. Each iteration takes $O(m)$ time. It follows that a cost scale takes $O(m^{3/2})$ time, yielding an $O(m^{3/2} \log(nC))$ UCMF algorithm.

4.1 Chain Elimination

The input to `eliminate-chain` is an admissible path $S = (v_0, v_1, \dots, v_{|S|-1})$. It is possible to implement `eliminate-chain` so that it cancels only negative cycles, but we describe a simpler implementation that may cancel cycles with positive cost, and a variant for which the circulation cost is monotonically decreasing.

The procedure starts by pushing a unit of flow from v_0 to $v_{|S|-1}$ along S . This saturates the bad arcs on S without creating new ones, and introduces a unit of deficit at v_0 and a unit of excess at $v_{|S|-1}$. We then run the `raise-potentials` method from Section 3.1 with a modified length function (see below). This increases the potentials and creates an admissible path, P , from $v_{|S|-1}$ to v_0 without introducing new bad arcs. To convert the pseudoflow into a circulation, we push a unit of flow on P . Since P is admissible, we do not create new bad arcs. Since bad arcs have reduced costs in the interval $[-2\epsilon, -\epsilon]$ the length ℓ of these arcs (defined in Section 3.1) is -1 . We re-define ℓ to be zero on bad arcs, that is $\ell(v, w) = \max(0, \lfloor \frac{c_p(v, w)}{\epsilon} \rfloor + 1)$. After `raise-potentials` updates p , the reduced costs of admissible arcs (and in particular of bad arcs) cannot decrease, so no new bad arcs are created.

Pushing a unit of flow on S and then returning it on P changes the flow on a set of cycles. Reduced costs of arcs on P may be positive w.r.t. the potentials at the beginning of `eliminate-chain`, and the cycles may have positive cost. We can modify the algorithm to make sure the total cost of these cycles is negative; then at least one of the cycles is negative. We use a different length function ℓ_m . The reverse arcs of arcs along S have reduced costs in $(0, 2\epsilon]$. For such arcs with reduced costs in $(0, \epsilon]$ we set $\ell_m = 0$ (instead of 1) and for such arcs with reduced costs in $(\epsilon, 2\epsilon]$ we $\ell_m = 1$ (instead of 2). For every other residual arc e we define $\ell_m(e) = \ell(e)$. Also, if the reversal of S become admissible, we always select it as the admissible path we push the flow back on; in this case `eliminate-chain` does not change f . Running `raise-potentials` with ℓ_m is equivalent to running the naive implementation from Algorithm 1 if we also treat reverse arcs of arcs along S with reduced costs in $(0, \epsilon]$ as admissible. One can prove this using the same arguments as in the proof of Lemma 6. It follows that the residual path we find from $v_{|S|-1}$ to v_0 may contain reverse arcs of arcs in S with reduced costs in $(0, \epsilon]$. Pushing the flow back on these arcs creates new admissible arcs but cannot create new bad arcs. Lemma 8 proves that with ℓ_m we get decreasing circulation costs.

► **Lemma 8.** *Let f' be the flow before `eliminate-chain` and f after. If $f' \neq f$ then $\text{cost}(f) < \text{cost}(f')$.*

Proof. Let P be the return path we select after **raise-potentials** and \bar{S} the reversal of S . P is a shortest path w.r.t. ℓ_m and \bar{S} is not, so we have $\sum_{(v,w) \in P} \ell_m(v,w) < \sum_{(v,w) \in \bar{S}} \ell_m(v,w)$ and so $\sum_{(v,w) \in P \setminus \bar{S}} \ell_m(v,w) < \sum_{(v,w) \in \bar{S} \setminus P} \ell_m(v,w)$. It follows that

$$\begin{aligned} \sum_{(v,w) \in P \setminus \bar{S}} \frac{c_p(v,w)}{\epsilon} &\leq \sum_{(v,w) \in P \setminus \bar{S}} \max(0, \lfloor \frac{c_p(v,w)}{\epsilon} \rfloor + 1) = \sum_{(v,w) \in P \setminus \bar{S}} \ell_m(v,w) \\ &< \sum_{(v,w) \in \bar{S} \setminus P} \ell_m(v,w) = \sum_{(v,w) \in \bar{S} \setminus P} (\lceil \frac{c_p(v,w)}{\epsilon} \rceil - 1) \leq \sum_{(v,w) \in \bar{S} \setminus P} \frac{c_p(v,w)}{\epsilon}. \end{aligned}$$

So we get that $\sum_{(v,w) \in P} c_p(v,w) < \sum_{(v,w) \in \bar{S}} c_p(v,w)$ and therefore $\sum_{(v,w) \in P \cup S} c_p(v,w) < 0$. ◀

5 Special Case Improvements and Generalizations

In this section we improve the bounds for our frameworks from Sections 3 and 4 for the special cases of SSSP, IAS, WBM and UMCF with unit vertex capacities.

First we review reductions from the WBM, AS and IAS problems to UMCF. The reduction from WBM is as follows. Given an instance $H = (V_1 \cup V_2, E_H)$ of WBM, we create a graph $G = (V, E)$. We add two new vertices s and t so that $V = V_1 \cup V_2 \cup \{s, t\}$. For each edge $(v, w) \in E_H$ we have an arc $(v, w) \in E$ whose cost in G is the negation of its cost in H (so that a maximum weight becomes a minimum cost). For every vertex $v \in V_1$ we add a new arc (s, v) with $c(s, v) = 0$. For every vertex $v \in V_2$ we add a new arc (v, t) with $c(v, t) = 0$. We also add the arc (t, s) with $c(t, s) = 0$. Let $r = \min\{|V_1|, |V_2|\}$. We set the capacities of all arcs to 1 except for the arc (t, s) , which gets a capacity of $u(t, s) = r$ (we can replace (t, s) by r parallel unit capacity arcs to get an UMCF instance). Clearly a minimum cost flow in G corresponds to a matching of maximum weight in H .

The reduction of AS to UMCF is the same as for WBM except that we set $c(t, s) = -nC$. Clearly a minimum cost flow in G corresponds to a perfect matching of maximum weight in H . The maximum absolute value of an arc cost in is nC rather than C . However we still state our bounds as a function of C defined as the largest absolute value of a weight in H . We reduce IAS to UMCF using the same reduction as for AS, except $u(t, s) = F$ and $c(t, s) = -FC$.

5.1 The Single Source Shortest Path Problem

We show how the cycle canceling algorithm of Section 4 can be slightly modified to match the $O(\sqrt{nm} \log C)$ time bound of [9]. In the modified algorithm we maintain f as the zero circulation except temporarily inside the **eliminate-chain** procedure. When the algorithm terminates, the reduced costs of all arcs are non-negative, so we can find all shortest paths from a source using Dijkstra's algorithm in $O(m + n \log n)$ time.

We run the $O(\log C)$ cost scales of the cycle canceling algorithm of Section 4 with the length function ℓ_m until $\epsilon = 1$. However, we terminate if we find an admissible cycle in **refine** or if the return path in **eliminate-chain** is not the reversal of S . In these cases there is a negative cycle in G_f . An analysis similar to the one of Section 4 shows that the running time of a cost scale is $O(\sqrt{nm})$ rather than $O(m^{3/2})$. This is done by defining a vertex as *bad* if it has a bad incoming arc and fixing at least \sqrt{k} bad vertices in each iteration of **refine** where k is the total number of bad vertices. The potential increase of an E_A -closed set fixes all the bad vertices in the set. Lemma 9 shows that running **eliminate-chain** with ℓ_m either fixes all the bad vertices along S or there is a negative cycle and we can terminate.

► **Lemma 9.** *If (x, v_i) is bad and $v_i \in S$ then there is a negative cycle or $d_{\ell_m}(x) > d_{\ell_m}(v_i)$*

Proof. Let S' be the part of the reverse path of S from $v_{|S|-1}$ to v_i . If $d_{\ell_m}(x) \leq d_{\ell_m}(v_i)$ then there is a path P from $v_{|S|-1}$ to x with $\sum_{(v,w) \in P} \ell(v,w) \leq \sum_{(v,w) \in S'} \ell(v,w)$. Using the same

arguments as in the proof of Lemma 8 we get that $\sum_{(v,w) \in P} c_p(v,w) - \sum_{(v,w) \in S'} c_p(v,w) \leq 0$. So the cycle of P then (x, v_i) and then the part of S from v_i to $v_{|S|-1}$ is a negative cycle. ◀

When $\epsilon = 1$ we run one last, slightly modified, cost scale. We change the definition of an *admissible arc* to be an arc with non-positive reduced cost. Since all potentials and costs at this point are integral and thereby multiples of ϵ , the reduced cost of an admissible arc is either -1 or 0 and it is ≥ 1 for all other arcs. A *bad arc* is an admissible arc of reduced cost -1 . Since zero reduced cost arcs are admissible, there are no zero reduced cost arcs outgoing from an E_A -closed set, and an increase of the potentials of the vertices in the set by 1 does not create new bad arcs. We define the length function ℓ in `eliminate-chain` accordingly: for every residual arc (v, w) we define $\ell(v, w) = \max(0, c_p(v, w))$. The arguments of Lemma 6 show that running `raise-potentials` with this length function does not create new bad arcs. The arguments of Lemma 8 show that if there isn't a negative cycle then we can push the flow back on the reversal of S .

It remains to show how to make the admissible network (by the new definition of admissibility) acyclic. We start every iteration of `refine` by contracting strongly connected components of G induced by zero reduced cost arcs. In the contracted graph we set the potentials of vertices to zero and the costs of arcs to be the reduced costs of the corresponding arcs before the contraction. This transformation takes linear time. At the end of the iteration we set the potential of every original vertex v to be its potential before the contraction plus the potential computed for the vertex into which v was contracted. See [9]. After the contraction G_A must be acyclic (otherwise there is a negative cycle and we terminate).

5.2 Imperfect Assignment Problem

We show an $O(\sqrt{Fm} \log(FC))$ time bound for IAS using either the pseudoflow or the cycle canceling framework. Since the cost of (t, s) is $-FC$, the zero circulation is not C -optimal with respect to the zero potential function. So instead of starting with the zero circulation we start with the circulation f consisting of a flow of value F from s to t together with a flow of F through the arc (t, s) , thereby saturating (t, s) (we assume that the value of the maximum flow from s to t is at least F as otherwise the problem is infeasible). This circulation is C -optimal with respect to the zero potential function.

The other modification we perform is a transformation of potentials before every cost scale as follows. For every vertex $v \in V_1$ we set $p(v) = p(v) + \epsilon$. For every vertex $v \in V_2$ we set $p(v) = p(v) + 2\epsilon$. We also set $p(t) = p(t) + 3\epsilon$. Lemma 10 specifies the outcome of the transformation.

► **Lemma 10.** *After applying the potential transformation we have $\sum_{(v,w) \in E_A} u_f(v, w) \leq 3F$.*

Proof. Consider an arc $(v, w) \in \{(s, t)\} \cup (\{s\} \times V_1) \cup (V_1 \times V_2) \cup (V_2 \times \{t\})$. Before the transformation $c_p(v, w) \geq -\epsilon$. After the transformation $c_p(v, w) \geq \epsilon - \epsilon = 0$. So $(v, w) \notin E_A$.

Let $R = E_f \cap ((V_1 \times \{s\}) \cup (V_2 \times V_1) \cup (\{t\} \times V_2))$ and consider an arc $(v, w) \in R$. Since $f(v, w) < u(v, w) = 0$ then $f(w, v) > 0$. Since f is a circulation and since all flow cycles of f contain (t, s) we get that $|R| \leq 3f(t, s)$. So $\sum_{(v,w) \in E_A} u_f(v, w) = |R| + u_f(t, s) \leq 3f(t, s) + u_f(t, s) = 3f(t, s) + u(t, s) - f(t, s) = 2f(t, s) + u(t, s) \leq 3F$. ◀

Lemma 10 implies that we start every cost scale in the cycle canceling framework with $O(F)$ bad arcs. An $O(\sqrt{Fm})$ time bound per cost scale follows using the same analysis as in Section 4. Lemma 11 implies an $\log(FC)$ bound on the number of cost scales:

► **Lemma 11.** *An ϵ -optimal circulation f with $\epsilon < \frac{1}{2F+4}$ is optimal.*

Proof. Consider a simple residual cycle Y . Pair up consecutive arcs that are not incident to s and t along Y . By the definition of G at least one arc in each pair is in $V_2 \times V_1$. It follows that $|Y| \leq 4 + 2|E_f \cap (V_2 \times V_1)| = 4 + 2|\{(w, v) \in V_1 \times V_2 \mid f(w, v) > 0\}| \leq 4 + 2F$, where the last inequality holds since by the conservation constraints in G we have that $|\{(w, v) \in V_1 \times V_2 \mid f(w, v) > 0\}| \leq u(t, s) \leq F$. It follows that $\text{cost}(Y) = \sum_{(v,w) \in Y} c(v, w) = \sum_{(v,w) \in Y} c_p(v, w) \geq \sum_{(v,w) \in Y} -\epsilon \geq -(4 + 2F)\epsilon > -1$. Since costs are integral it follows that $\text{cost}(Y) \geq 0$ so there are no negative residual cycles and therefore f is optimal. \blacktriangleleft

For the pseudoflow framework we use the same notation as in Section 3 except that we let $\lambda = \sqrt{F}$. We split a cost scale of the algorithm into two phases. The second phase starts when every excess v has $d(v) \geq \lambda$. Lemma 12 and the analysis from Section 3 yield the $O(\sqrt{F}m)$ time bound per cost scale for the pseudoflow framework.

► **Lemma 12.** *When the second phase starts there is $O(\sqrt{F})$ excess left.*

Proof. Consider the decomposition of the current pseudoflow f into simple cycles and paths from deficits to excesses. By the definition of G every simple flow cycle or path is of length at most 4. By Lemma 10 the total excess generated when we start a cost scale is at most $3F$ so there are at most $3F$ flow paths from deficits to excesses. Since $u(t, s) = F$ there are at most F flow cycles. It follows that $|\{(v, w) \in E \mid f(v, w) > 0\}| \leq 4 \cdot 3F + 4F = 16F$.

Let $R = E_f \cap ((V_1 \times \{s\}) \cup (V_2 \times V_1) \cup (\{t\} \times V_2) \cup \{(t, s)\})$. We have $\sum_{(v,w) \in R} u_f(v, w) \leq |\{(w, v) \in E \mid f(w, v) > 0\}| + F \leq 17F$. For $1 \leq i \leq \frac{\lambda}{12}$ let $A_i = \{(v, w) \in E_f \mid d(v) \in \{12i, 12i - 1, \dots, 12i - 11\}\}$.⁷ We apply the pigeon hole principle to the sets of arcs $A_i \cap R$. Since $\sum_{(v,w) \in R} u_f(v, w) \leq 17F$ there must be a set $A_i \cap R$ such that $\sum_{(v,w) \in A_i \cap R} u_f(v, w) \leq 17F / \frac{\lambda}{12} = 204\sqrt{F}$. We show that each $A_i \cap R$ is an excess-deficit cut in G^+ . A bound of $204\sqrt{F}$ on the remaining excess follows using Lemma 4.

Consider an excess-deficit E^+ residual path P . We show that it has an arc in each set $A_i \cap R$. Clearly every simple residual path in G_f , and P in particular, must have at least one arc in R out of every 4 consecutive arcs. Furthermore, since an excess has $d(v) = \lambda$, a deficit has $d(v) = 0$, and by Lemma 3, P must contain 4 consecutive arcs in each A_i . It follows that one of these 4 arcs is in R and therefore in $A_i \cap R$. \blacktriangleleft

5.3 Weighted Bipartite Matching

In this section we show the $O(\sqrt{r}m \log(C))$ time bound for the WBM problem. Recall that r is the number of vertices in the smaller “side” of the bipartite graph. We use the same ideas as Duan et al. [6], but within our cost-scaling frameworks, obtaining a simpler algorithm with the improved time bound. First we show how to obtain an $O(\sqrt{r}m \log(rC))$ time bound using either framework. Then we show how to improve the bound to $O(\sqrt{r}m \log C)$ by introducing a preprocessing stage based on the pseudoflow framework and a postprocessing stage based on the cycle canceling framework.

Before every cost scale we perform the same potential transformation as in Section 5.2. A lemma analogous to Lemma 10 shows that following this transformation $\sum_{(v,w) \in E_A} u_f(v, w) \leq 3r$. This gives a bound of $O(\sqrt{r}m)$ time for a cost scale in the cycle canceling framework. For the pseudoflow framework we define $\lambda = \sqrt{r}$ and split a cost scale into two phases as in Section 5.2. Lemma 13 implies the $O(\sqrt{r}m)$ time bound for a cost scale in the pseudoflow framework. For both frameworks, a lemma analogous to Lemma 11 implies $O(\log rC)$ cost scales yielding the $O(\sqrt{r}m \log(rC))$ bound.

⁷ As in the previous section, to simplify the presentation we assume that $\frac{\lambda}{12}$ is an integer.

► **Lemma 13.** *When the second phase starts there is $O(\sqrt{r})$ excess left.*

Proof. Assume without loss of generality that $r = |V_1| \leq |V_2|$. For $1 \leq i \leq \frac{\lambda}{12}$ let $K_i = \{v \in V \mid d(v) \in \{12i, 12i - 1, \dots, 12i - 10, 12i - 11\}\}$ and let $N_i = K_i \cap V_1$.⁸ By the pigeon hole principle there must be a set N_j such that $|N_j| \leq \frac{r}{\frac{12}{\lambda}} = 12\sqrt{r}$. We argue in the next paragraph that the set of E^+ arcs outgoing from every N_i is an excess-deficit cut in G^+ . Since each vertex in V_1 has only one outgoing arc that may be in E^+ , the size of the cut corresponding to N_j is at most $12\sqrt{r}$ and the lemma follows by Lemma 4.

Consider a simple path P of E^+ arcs from an excess to a deficit. By Lemma 3, P must have at least 4 consecutive vertices in K_i . Since P is simple, one of these vertices v has $v \in V_1$ and so we get $v \in N_i$. The single outgoing E^+ arc of v must be on P . ◀

Next we improve the running time to $O(\sqrt{r}m \log C)$. The improved algorithm has three stages. First we find an $\frac{C}{\sqrt{r}}$ -optimal circulation in $O(\sqrt{r}m)$ time. Then we run cost scales from either framework to get an $\frac{1}{\sqrt{r}}$ -optimal circulation in $O(\sqrt{r}m \log C)$ time. Finally we convert the circulation to an optimal one in $O(\sqrt{r}m)$ time.

For the first stage we set $\epsilon = \frac{C}{\sqrt{r}}$. We set $p(v) = -C$ for every vertex $v \in \{s\} \cup V_1$ and $p(v) = 0$ for every vertex $v \in V_2 \cup \{t\}$. We initialize f with the zero circulation. Finally we call the **refine** method of the blocking flow algorithm from Section 3. Lemma 14 shows that this call to **refine** performs at most $O(\sqrt{r})$ blocking flow computations. It was first proved for a balanced bipartite graph by Duan et al. [6]. It follows that this stage runs in $O(\sqrt{r}m)$ time.

► **Lemma 14.** *The first stage performs at most $\sqrt{r} + 2$ blocking flow computations.*

Proof. Consider the initial potentials. Arcs outgoing from s or incoming into t have reduced cost 0. Every arc $(v, w) \in V_1 \times V_2$ has $c_p(v, w) \geq -C + C + 0 \geq 0$. The arc (t, s) has $c_p(t, s) = -C$. It follows that $E_A = \{(t, s)\}$. So after saturating admissible arcs we have $e_f(s) > 0$, $e_f(t) < 0$ and $E^+ = \{(s, t)\}$. Since t is the only deficit we have $p(t) = 0$. After y blocking flows we have that $p(s) = -C + y\frac{C}{\sqrt{r}}$. It follows that $c_p(s, t) = -p(s)$ is nonnegative for $y \leq \sqrt{r}$ and is negative for $y = \sqrt{r} + 1$. So (s, t) is admissible after $\sqrt{r} + 1$ blocking flows and the $(\sqrt{r} + 2)$ 'th blocking flow drains all remaining excess from s through the arc (s, t) . ◀

For the finish-up stage we run the last cost scale of the SSSP algorithm from Section 5.1, except that whenever we encounter an admissible (negative) cycle we do not terminate. Instead we cancel the cycle and start a new iteration of **refine**. Consider a negative cycle. By integrality, its cost is -1 or less, so by ϵ -optimality (recall that $\epsilon \leq 1/\sqrt{r}$ at this point) the cycle must contain at least \sqrt{r} arcs with negative reduced cost. This observation implies that canceling a negative cycle saturates at least \sqrt{r} bad arcs. It follows that the number of time we restart an iteration of **refine** is $O(\sqrt{r})$.

5.4 Minimum Cost Flow with Unit Vertex Capacities

We show an $O(\sqrt{n}m \log(nC))$ time bound on the MCF problem with unit vertex capacities using our pseudoflow framework. We consider the formulation of the problem on networks where every vertex has an out-degree of 1 or an in-degree of 1. The time bound is proved in the same way as in Section 3 except we let $\lambda = \sqrt{n}$ and we let Lemma 15 replace Lemma 5.

⁸ To simplify the presentation we assume that $\lambda/12$ is an integer.

► **Lemma 15.** *When the second phase starts there is $O(\sqrt{n})$ excess left.*

Proof. During the second phase every excess v has $d(v) \geq \lambda$ and every deficit v has $d(v) = 0$. Consider the $\frac{1}{3}\lambda$ sets of vertices $V_i = \{v \in V \mid d(v) \in \{3i, 3i - 1, 3i - 2\}\}$ where $1 \leq i \leq \frac{\lambda}{3}$. Also consider $A_i = \{(v, w) \in E^+ \mid (v \in V_i \text{ and has out-degree } 1) \text{ or } (w \in V_i \text{ and has in-degree } 1)\}$. By Lemma 3 for any set V_i , every E^+ excess to deficit path must contain a vertex in V_i . It follows that every A_i is an excess-deficit cut in G_f . By the pigeon hole principle there must be a set V_i such that $|V_i| \leq \frac{n}{\lambda/3} = 3\sqrt{n} = 3\lambda$ and therefore $|A_i| \leq 3\sqrt{n}$. So by Lemma 4 the remaining excess is at most 3λ . ◀

Acknowledgements We thank an anonymous reviewer for a simplification of the algorithm of Section 4.

References

- 1 R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.
- 2 R. K. Ahuja, J. B. Orlin, C. Stein, and R. E. Tarjan. Improved Algorithms for Bipartite Network Flow. *SIAM J. Comput.*, 23:906–933, 1994.
- 3 R. G. Bland and D. L. Jensen. On the Computational Behavior of a Polynomial-Time Network Flow Algorithm. *Math. Prog.*, 54:1–41, 1992.
- 4 R. B. Dial. Algorithm 360: Shortest Path Forest with Topological Ordering. *Comm. ACM*, 12:632–633, 1969.
- 5 E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numer. Math.*, 1:269–271, 1959.
- 6 Ran Duan, Seth Pettie, and Hsin-Hao Su. Scaling algorithms for approximate and exact maximum weight matching. *CoRR*, abs/1112.0790, 2011.
- 7 S. Even and R. E. Tarjan. Network Flow and Testing Graph Connectivity. *SIAM J. Comput.*, 4:507–518, 1975.
- 8 H. N. Gabow and R. E. Tarjan. Faster Scaling Algorithms for Network Problems. *SIAM J. Comput.*, 18:1013–1036, 1989.
- 9 A. V. Goldberg. Scaling Algorithms for the Shortest Paths Problem. *SIAM J. Comput.*, 24:494–504, 1995.
- 10 A. V. Goldberg and R. E. Tarjan. Finding Minimum-Cost Circulations by Canceling Negative Cycles. *J. Assoc. Comput. Mach.*, 36:873–886, 1989.
- 11 A. V. Goldberg and R. E. Tarjan. Finding Minimum-Cost Circulations by Successive Approximation. *Math. of Oper. Res.*, 15:430–466, 1990.
- 12 J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ Algorithm for Maximum Matching in Bipartite Graphs. *SIAM J. Comput.*, 2:225–231, 1973.
- 13 A. V. Karzanov. O nakhozhdanii maksimal'nogo potoka v setyakh spetsial'nogo vida i nekotorykh prilozheniyakh. In *Matematicheskie Voprosy Upravleniya Proizvodstvom*, volume 5. Moscow State University Press, Moscow, 1973. In Russian; title translation: On Finding Maximum Flows in Networks with Special Structure and Some Applications.
- 14 Jr. L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton Univ. Press, Princeton, NJ, 1962.
- 15 A. Madry. "navigating central path with electrical flows: From flows to matchings, and back". In *FOCS*, pages 253–262, 2013.
- 16 L. Ramshaw and R. Endre Tarjan. "a weight-scaling algorithm for min-cost imperfect matchings in bipartite graphs". In *FOCS*, pages 581–590, 2012.

- 17 H. Röck. Scaling Techniques for Minimal Cost Network Flows. In U. Pape, editor, *Discrete Structures and Algorithms*, pages 181–191. Carl Hansen, Munich, 1980.

Inductive Inference and Reverse Mathematics*

Rupert Hölzl¹, Sanjay Jain², and Frank Stephan³

- 1 Department of Mathematics, National University of Singapore, S17, 10 Lower Kent Ridge Road, Singapore 119076, Republic of Singapore, r@hoelzl.fr
- 2 Department of Computer Science, National University of Singapore, COM2, 15 Computing Drive, Singapore 117417, Republic of Singapore, sanjay@comp.nus.edu.sg
- 3 Department of Mathematics and Department of Computer Science, National University of Singapore, S17, 10 Lower Kent Ridge Road, Singapore 119076, Republic of Singapore, fstephan@comp.nus.edu.sg

Abstract

The present work investigates inductive inference from the perspective of reverse mathematics. Reverse mathematics is a framework which relates the proof strength of theorems and axioms throughout many areas of mathematics in an interdisciplinary way. The present work looks at basic notions of learnability including Angluin’s tell-tale condition and its variants for learning in the limit and for conservative learning. Furthermore, the more general criterion of partial learning is investigated. These notions are studied in the reverse mathematics context for uniformly and weakly represented families of languages. The results are stated in terms of axioms referring to domination and induction strength.

1998 ACM Subject Classification F.4.1 Mathematical Logic.

Keywords and phrases reverse mathematics, recursion theory, inductive inference, learning from positive data

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.420

1 Introduction

It is standard practice in mathematics to use known theorems to prove others. In these cases it can often be observed that some theorem T seems to be “stronger” than another theorem U in the sense that T allows proving U easily, but not vice versa. In the 1970s, Friedman [11] proposed a framework that formalises this intuition and allows gauging the different strengths of theorems that can be found in classical mathematics.

The general idea is to assume only a subset of the axioms of second order arithmetic, which by itself is too weak to prove the theorems in question, and then to analyse whether one theorem implies the other over this weak base system. Of course, if we want to *exactly* determine the strength of a mathematical theorem T with regards to logical implication, then we need to look in both directions: which theorems are implied by T and which imply T ? As all of mathematics is ultimately founded on axioms, it is a natural next step to extend this study to the relation between axioms and theorems, and to wonder what *axioms* are exactly equivalent to a given theorem T , that is, imply T and are implied by T .

This “inverted” approach – where one uses theorems to prove axioms instead of the other way around – explains the name of this field of study: reverse mathematics. The subject has

* R. Hölzl was fully and S. Jain and F. Stephan partially supported by NUS/MOE grant R146-000-184-112 (MOE2013-T2-1-062); furthermore, S. Jain is partially supported by NUS grant C252-000-087-001.

developed well since its inception, in particular thanks to many substantial contributions made by Simpson and his students [19]. The methodology of reverse mathematics has been applied to many fields of classical mathematics, for example, to group theory, to vector algebra, to analysis, and – especially in recent years – to combinatorics, including Ramsey theory and related fields. We refer to the books of Hirschfeldt [14] and Simpson [19] which are convenient resources for the topic and give many references.

In the practice of reverse mathematics we will look at proper subsets of the axioms of second order arithmetic and will investigate the properties of possible models of these axiom sets. Such a model will be of the form $(M, +, \cdot, <, 0, 1, \mathcal{S})$, where M is a (not necessarily standard) model of the natural numbers and \mathcal{S} is a class of subsets of M . The minimal axiom system over which we will work is called RCA_0 . Informally speaking, the axioms of this system guarantee that \mathcal{S} contains at least all recursive sets and is closed under join and Turing reduction. Furthermore, the axioms ensure that the system satisfies Σ_1 -induction with parameters from \mathcal{S} ; in particular, even in nonstandard models of RCA_0 all numbers of the form $\max_{i < n} f(i)$ exist for all functions $f \in \mathcal{S}$.

More precisely, RCA_0 postulates that $(M, +, \cdot, <, 0, 1)$ behaves sufficiently similar to the natural numbers, in the following sense, and that \mathcal{S} satisfies the following closure properties:

- The ordering $<$ is linear, transitive and antireflexive and has 0 as the least element;
- The successor mapping $x \mapsto x + 1$ satisfies that $x < x + 1$ and $x < y \Leftrightarrow x + 1 < y + 1$ as well as that 0 is the only number x which is not equal to $y + 1$ for some other number y ;
- The addition $+$ is inductively defined from the successor by $x + 0 = x$ and $x + (y + 1) = (x + y) + 1$;
- The ordering $<$ is definable from $+$ by $x < y \Leftrightarrow \exists z [x + z + 1 = y]$;
- The multiplication is inductively defined from the addition by $x \cdot 0 = 0$ and $x \cdot (y + 1) = (x \cdot y) + x$;
- The second order model satisfies Σ_1 -induction, that is, if $I \subseteq M$ is defined by a Σ_1 -formula using parameters from \mathcal{S} and satisfies for all e the implication $[\forall d < e (d \in I)] \Rightarrow e \in I$ then I is equal to M ;
- The set \mathcal{S} contains \emptyset and all sets which are recursive in the model $(M, +, \cdot, <, 0, 1)$;
- The second order model is a Turing ideal, that is, if $I, J \in \mathcal{S}$ then $I \oplus J = \{i + i : i \in I\} \cup \{j + j + 1 : j \in J\}$ is also a member of \mathcal{S} and, furthermore, if $I \in \mathcal{S}$ and J can be obtained from I by both a Σ_1 -definition and a Π_1 -definition then $J \in \mathcal{S}$.

Note that the last statement ensures that \mathcal{S} is closed under join and Turing reducibility.

The model which contains *exactly* the recursive sets is called the minimal model of RCA_0 . Of course there are many models of RCA_0 that are much richer than the minimal model. In particular if M is the standard model of natural numbers, then there is also the model where \mathcal{S} is the power set of M ; for nonstandard models, the power-set of M cannot be a model as it fails the induction axiom. There also exist many intermediate models between those two extremes. When M is the standard model of the natural numbers, then $(M, +, \cdot, <, 0, 1, \mathcal{S})$ is called an ω -model and due to their well-behavedness (compared to nonstandard models), they are better understood than nonstandard models in reverse mathematics. However, various complicated results in reverse mathematics were only obtained through the use of nonstandard models [8, 9].

As we will show in this article, many results in inductive inference relate to the following three axioms from reverse mathematics:

- The axiom DOM which says that for every weakly represented family of functions in \mathcal{S} (defined below) there exists a function in \mathcal{S} growing faster than all members of the family;
- The axiom ACA_0 which says that the class \mathcal{S} is closed under Turing jump;

- The axiom $\text{I}\Sigma_2$ which postulates that every Σ_2 set I definable using parameters from \mathcal{S} satisfies the induction axiom: if $\forall e \llbracket \forall d < e [d \in I] \Rightarrow e \in I \rrbracket$ then $I = M$.

Note that $\text{I}\Sigma_2$ is satisfied for all standard models of the natural numbers; however, if M is a nonstandard model, assuming $\text{I}\Sigma_2$ is a nontrivial constraint. By identifying a function with its graph we can also informally talk about the functions from M to M that exist in the model (that is, whose graphs are in \mathcal{S}).

In an informal way, we will often think of the sets in \mathcal{S} as being the recursive sets, even for sets that are not recursive in the classical sense of recursion theory. This seemingly strange fact can be understood as follows: Often in the reverse mathematics context we wonder whether a certain object exists in a model, or what additional axiom – say, for example, comprehension for Σ_2^0 formulas – is needed to ensure its existence. We are then allowed to apply these additional axioms relative to any object X already existing in the model, no matter if X is recursive in the classical sense. That is, as soon as we know that X is guaranteed to exist in \mathcal{S} , we are allowed to take advantage of it, so for our purposes it is as good as recursive.

In this article we propose to apply the methodology of reverse mathematics to the field of inductive inference. We would like to point out that articles by de Brecht and Yamamoto [5] and by Hayashi [13] pursue the same idea, but in ways that differ from our approach and from each other. We proceed with defining central notions and analysing basic results of the field of inductive inference [1, 2, 3, 6, 7, 12, 15, 17, 21]. We will in particular study Angluin’s tell-tale condition for learnability and related results. In this context the notion of finiteness of a set is of high importance as Angluin’s tell-tale sets are finite. We point out that in the reverse mathematics setting some care is required with regard to this, as the universe M of the model \mathcal{S} may be nonstandard. We therefore fix the term “finite” for a subset of M to mean that the subset of M has an upper bound and “infinite” to mean that no such bound exists. Furthermore, it should be noted that “finite sets” are always considered to be “finite sets contained in \mathcal{S} ” and that they are precisely those sets E for which there is a member $e \in M$ with $e = \sum_{d \in E} 2^d$.

Furthermore, we use Cantor’s pairing function $\langle x, y \rangle = (x + y) \cdot (x + y + 1)/2 + y$ and extend it appropriately to triples and quadruples and so on. Now we code a family of sets $\{A_e\}_{e \in M}$ using a single set A by defining $x \in A_e \Leftrightarrow \langle e, x \rangle \in A$. Such families of sets are called uniformly represented families. Similarly one can define a uniformly represented family of functions by $F_e(x) = F(\langle e, x \rangle)$ using one representation function. This notion was generalised to the notion of weakly represented families of functions as follows [18]. Assume that a representation set $A \in \mathcal{S}$ satisfies the following conditions on quadruples:

- For all e, x, y, z, y', z' : If $\langle e, x, y, z \rangle, \langle e, x, y', z' \rangle \in A$ then $y = y'$ and $z = z'$;
- If $\langle e, x, y, z \rangle \in A$ and $x' < x$ then there exist y' and z' such that $\langle e, x', y', z' \rangle \in A$ and $\langle e, x', y', z' \rangle < \langle e, x, y, z \rangle$.

The intention behind the second condition is to ensure that the coding quadruples for each function appear in the family in order ascending in the function argument x , even if the function is not monotone; for this purpose we use the fourth component of the quadruples as padding parameter. The first condition ensures that for each e, x there is at most one code defining $F_e(x)$. An index is invalid if there is some x where $F_e(x)$ is not defined. Hence the set $D = \{e: \forall x \exists y, z [\langle e, x, y, z \rangle \in A]\}$ is the index set of functions in the weakly represented family and for $e \in D$, $F_e(x)$ is the unique y such that $\langle e, x, y, z \rangle \in A$ for some z . The family $\{F_e\}_{e \in D}$ is called the weakly represented family defined by A and every function F_e , $e \in D$, is a function in the given second order model $(M, +, \cdot, <, 0, 1, \mathcal{S})$. Furthermore, in the case that all functions F_e in the family are $\{0, 1\}$ -valued, they can also be viewed as the

characteristic functions of a weakly represented family of sets and might be denoted with A_e rather than F_e . Note that Dzhafarov and Mummert [10] have considered the more general concept of *enumerated families* of sets.

In some of the proofs in this article we will make statements of the form “ $A_e(x)$ can be retrieved from A in time less than s .” By this we mean that the code number $c = \langle e, x, A_e(x), z \rangle \in A$ is bounded by s . The intuition of time is explained by the fact that $A_e(x)$ or the padding parameter z could be very large, so that it will depend on c how far we need to search in A to determine the function value $A_e(x)$ for a given x .

In the reverse mathematics setting, we will of course only work with representation sets A as above that exist in the given model of second order arithmetic \mathcal{S} . In the case of uniformly represented families then also their index set D as above must exist in \mathcal{S} . But note the important fact that for families that are only weakly represented, this will typically not be the case, that is, D is usually not required to exist in \mathcal{S} , only A always exists in \mathcal{S} . For this reason, we need to be careful in this article when working with families of functions, because a learner (which has to be a function in \mathcal{S} from finite sequences of elements of $M \cup \{\#\}$ to M) may conjecture members of M that are not members of D , as at the time of the conjecture it cannot know whether a particular member of M is a valid index or not. Note that weakly represented families can be much more general than uniformly represented families; for example, for a fixed member $A \in \mathcal{S}$, the family of all A -recursive functions is weakly representable but in general not uniformly representable.

We now turn to the more formal notations from learning theory. Note that the families defined above correspond to the classes of possible learning targets in learning theory. The general scenario is that one possible learning target is presented to the learner in an infinite sequence of data and the learner has to identify which of the possible targets the data is from. Such a data presentation is called a text. We define the notion of a text in a way that is compatible with reverse mathematics, that is, in such a way that when M is equal to the standard natural numbers the definition coincides with the traditional one, but in the case of nonstandard models they may differ.

► **Definition 1.** A text for a set $A \in \mathcal{S}$ is a function $T: M \rightarrow M \cup \{\#\}$ in \mathcal{S} such that $\{T(n) : n \in M \wedge T(n) \neq \#\} = A$. We call $\#$ the pause symbol. Without loss of generality we assume that $T(x) \in \{0, 1, \dots, x\} \cup \{\#\}$.

The pause symbol “ $\#$ ” is a padding symbol that carries no information and is useful to give a text for empty set. Again as usual we will write M^* for the set of finite sequences over $M \cup \{\#\}$. These can be thought of as the prefixes of texts. Take note that the word “finite” needs to be understood in the reverse mathematics sense discussed above. M^* can be represented by some canonical indexing, where each finite sequence σ is represented by the canonical index of the set $\{\langle x, 0 \rangle : \sigma(x) = \#\} \cup \{\langle x, y + 1 \rangle : \sigma(x) = y\}$. One can prove by induction over a text that such canonical indices exist for every prefix of a text.

► **Definition 2** (Angluin [2]; Gold [12]; Osherson, Stob and Weinstein [17]). Let $\{A_e\}_{e \in D}$ be a uniformly or weakly represented family and let $\{B_e\}_{e \in E}$ be a hypothesis space such that $\{A_e\}_{e \in D} \subseteq \{B_e\}_{e \in E}$. A learner is a function $L: M^* \rightarrow M$, where the elements of M^* are represented by canonical indices.

A learner L learns in the limit a family $\{A_e\}_{e \in D}$ if for every $e \in D$ and every text T for A_e the learner outputs a sequence of hypotheses $e_n = L(T(0) \dots T(n))$ such that, for some n , for all $m \geq n$, each hypothesis e_m is equal to e_{m+1} and $e_m \in E$ and $B_{e_m} = A_e$.

A conservative learner never makes an unjustified mind change. So if $n < m$ and $e_n \neq e_m$ then either $e_n \notin E$ or there exists $k \leq m$ with $T(k) \in M - B_{e_n}$. Conservative learning then requires learning in the limit by a conservative learner.

A learner partially learns the family $\{A_e\}_{e \in D}$ if for every $e \in D$ and every text T for A_e the learner outputs a sequence of hypotheses as above such that there is exactly one d with $\forall m \exists n > m [d = e_n]$ and, furthermore, this d satisfies $d \in E$ and $B_d = A_e$.

Note that often $B_e = A_e$ for all e and $E = D$, that is, the original family is used as hypothesis space. The intuition for learning in the limit is that when a learner learns a family, its output should converge to an index of the member of the family that the given text corresponds to. If invalid texts are presented to the learner, it may output numbers that are not actually in the set D of valid indices. Partial learning is a more general learning notion which, in the classical setting, allows learning the family of all r.e. sets.

Due to space constraints the proofs of most of the results in this article have been omitted.

2 Angluin's Condition

Angluin [2] gave a fundamental condition for the learnability of so-called indexed families of sets. These are families of sets such that there exists a computable two-place function F which on input (e, x) outputs 1 if $x \in L_e$ and 0 if $x \notin L_e$. As F works for all e , the closest equivalent to indexed families in the area of reverse mathematics are uniformly represented families. Angluin's condition (also called Angluin's tell-tale condition/criterion) says that one can learn an indexed family from positive data in the limit if and only if one can enumerate for each member A_e of the family a finite tell-tale subset B_e of A_e such that there is no other member A_d of the family with $B_e \subseteq A_d \subset A_e$. In reverse mathematics, it is difficult to handle finite sets, therefore one mostly represents them by canonical indices. However, for the tell-tale sets it is sufficient to consider bounds (called tell-tale bounds) b_e for each A_e such that there is no A_d with $A_e \cap \{0, 1, \dots, b_e\} \subseteq A_d \subset A_e$.

► **Definition 3.** Let a weakly represented family $\{A_e\}_{e \in D}$ with index set D be given:

1. The family satisfies Angluin's condition in general iff for each $e \in D$ there is a bound b_e such that there is no $d \in D$ with $A_e \cap \{0, 1, \dots, b_e\} \subseteq A_d \subset A_e$;
2. The family satisfies Angluin's condition in the limit iff there is a two-place function $g \in \mathcal{S}$ such that for every $e \in D$ the values $g(\langle e, 0 \rangle), g(\langle e, 1 \rangle), \dots$ approximate from below a bound b_e such that there is no $d \in D$ with $A_e \cap \{0, 1, \dots, b_e\} \subseteq A_d \subset A_e$;
3. The family satisfies Angluin's condition effectively iff there is a function $g \in \mathcal{S}$ such that for all $e \in D$ we have that there is no $d \in D$ with $A_e \cap \{0, 1, \dots, g(e)\} \subseteq A_d \subset A_e$.

To avoid confusion we point out the informal use of the word "effectively" in the third item, which needs to be understood as " $g \in \mathcal{S}$."

Blum and Blum [3] established the existence of so-called locking-sequences; that is, whenever a learner learns a language X there is a finite sequence of elements in X such that, after having processed this sequence, the learner conjectures a hypothesis which will not be changed on any subsequent data drawn from $X \cup \{\#\}$. Blum and Blum's proof can easily be modified to carry over to the reverse mathematics setting; it then proves the following statement.

► **Theorem 4.** RCA_0 proves the following: Suppose a weakly represented family $\{A_e\}_{e \in D}$ and a learner L are given such that for every $e \in D$ and every text T for A_e , L converges on T to an index $d \in D$ with $A_d = A_e$. Then, there is a procedure which for every index $e \in M$ converges in the limit to a finite sequence (represented by a code); in the case that $e \in D$, this sequence is a locking sequence for A_e .

This existence of locking-sequences then shows that every weakly represented family which is learnable in the limit must satisfy Angluin's tell-tale condition with a general bound: the bound is simply the largest element contained in the locking sequence. Section 3 will address the question of which of the above variants of Angluin's tell-tale condition is sufficient for learning all weakly represented families satisfying it. The axiom DOM will be identified as necessary and sufficient for this.

In Section 4 we will then follow Angluin's approach more closely and investigate uniformly represented families which, as mentioned before, are the closest equivalent in reverse mathematics to the indexed families that Angluin studied. The difference is that Angluin's families are *actually* uniformly recursive, while our uniformly represented families are only uniformly recursive relative to the parameter A representing them. This corresponds to the paradigm described above that in the reverse mathematics context often all sets in \mathcal{S} are treated *as if* they were recursive. As we will show, for uniformly represented families, the degree of effectiveness of the bound in Angluin's condition is crucial. Section 5 then looks at sufficient criteria for learning from the classical theory and shows that in reverse mathematics they work for uniformly represented families as well. However, for weakly represented families we will again require the axiom DOM. In Section 6 we will study partial learning.

3 Learnability of Weakly Represented Families

As mentioned above, the counterpart of the indexed families studied by Angluin are the uniformly represented families in reverse mathematics. So it is not surprising that to prove similar results for families that are represented in a less accessible way, such as weakly represented families, we will need an additional assumption on the second order model. This assumption is the axiom DOM which will turn out to be equivalent to saying that every weakly represented family satisfying Angluin's condition is learnable in the limit. The axiom DOM says that every weakly represented family of functions is dominated by a single function in \mathcal{S} . Note that Adleman and Blum [1] showed that one can learn all classes of graphs of recursive functions (which all satisfy Angluin's condition) iff one has access to a dominating function as an oracle. The axiom DOM now enforces that for every weakly represented family of functions there is such a dominating function in \mathcal{S} ; this function can therefore be used by the learner (which also has to be an object in \mathcal{S}).

► **Theorem 5.** *Over RCA_0 , the following conditions are equivalent:*

1. *The axiom DOM holds, that is, for every weakly represented family $\{F_e\}_{e \in D}$ of functions there is a function $f \in \mathcal{S}$ dominating this family in the sense that $\forall e \in D \exists x \forall y > x [F_e(y) < f(y)]$;*
2. *The index set of every weakly represented family can be approximated in the limit;*
3. *Every weakly represented family satisfying Angluin's condition effectively can be learnt in the limit;*
4. *Every weakly represented family satisfying Angluin's condition in the limit can be learnt in the limit;*
5. *Every weakly represented family satisfying Angluin's condition generally can be learnt in the limit.*

Proof. $1 \Rightarrow 2$: Let $\{F_e\}_{e \in D}$ be a weakly represented family with representation set A . Then the set of the functions G_e for $e \in M$ which assign to x the minimum tuple (if it exists) $\langle e, x, y, z \rangle \in A$ also forms a weakly represented family, and G_e is total iff $e \in D$. Thus the index set of this weakly represented family is also D .

By assumption there is a function f dominating all $\{G_e\}_{e \in D}$. Now it holds that $e \in D$ if and only if, for almost all numbers x , there are pairwise distinct elements of the form $\langle e, 0, y_0, z_0 \rangle, \dots, \langle e, x, y_x, z_x \rangle \in A \cap \{0, 1, \dots, f(x)\}$. This is because on one hand, if $e \in D$, the existence of these elements below $f(x)$ follows from the fact that f dominates G_e . On the other hand, if $e \notin D$, then there exists an x such that A does not contain *any* element of the form $\langle e, x, \cdot, \cdot \rangle$, so in particular there is no sequence as above.

Now one defines a function g by letting $g(e, x) = 1$ iff there are elements of the form $\langle e, 0, y_0, z_0 \rangle, \dots, \langle e, x, y_x, z_x \rangle \in A \cap \{0, 1, \dots, f(x)\}$, and $g(e, x) = 0$ otherwise. Then we have that $g(e, x)$ converges to 1 exactly when $e \in D$ and $g(e, x)$ converges to 0 exactly when $e \notin D$, so g is as needed.

$2 \Rightarrow 1$: Assume that a weakly represented family $\{F_e\}_{e \in D}$ has an index set D which is approximated by g in the limit and has the representation set A . Then one can construct the following function f :

$$f(x) = \min\{t: \forall e \leq x [\exists u, y, z \leq t (g(e, u+x) = 0 \vee \langle e, x, y, z \rangle \in A)]\}.$$

This function f is total, as for all indices e either a stage $u+x$ is found with $g(e, u+x) = 0$ or some value $\langle e, x, y, z \rangle$ is retrieved from A .

The minimum is taken over only finitely many conditions (in the square brackets) and for every condition individually the minimal t can be computed from e and x (using the same parameter set in the second order model as for the computation of A). Therefore, using Σ_1 -induction, $f(x)$ exists as the maximum over the t 's that are minimal for the individual conditions (for each $e \leq x$). Note that the “+” in the definition of f ensures that wrong behaviour of g during the first finitely many approximation stages is ignored in the limit.

The function f dominates each function F_e with $e \in D$, as for that function there is a large enough $x \geq e$ with $g(e, u+x) = 1$ for all u and therefore $f(x') \geq F_e(x')$ for all $x' \geq x$.

1 and $2 \Rightarrow 5$: Let $\{A_e\}_{e \in D}$ be a weakly represented family satisfying Angluin's tell-tale condition generally. Furthermore, by the second condition there is a function $g \in \mathcal{S}$ such that, if $e \in D$ then $\lim_x g(e, x) = 1$ else $\lim_x g(e, x) = 0$. Now define for each e and bound b a function $G_{e,b}$ such that $G_{e,b}(x)$ is the first $t \geq x$ found such that for each $d \leq x$ at least one of the following three conditions applies:

- We have $g(d, u+x) = 0$ or $g(e, u+x) = 0$ for some $u \leq t$;
- There is a number $x' \leq t$ such that $A_d(x')$ and $A_e(x')$ can be retrieved from the representation set within time t and either $x' \in A_d - A_e$ or $x' \in A_e - A_d \wedge x' \leq b$;
- The values of A_d and A_e up to x have been retrieved from the representation set within time t and $A_d(x') = A_e(x')$ for all $x' \leq x$.

These three conditions search for either e not being a valid index, or d not being a valid index, or x' witnessing that A_d is not a subset of A_e , or x' being an element of the tell-tale set of A_e that is not in A_d , or A_d being equal to A_e up to x . Note that the function $G_{e,b}$ is total for those b which are valid bounds for F_e ; thus the index set of the family $\{G_{e,b}\}_{(e,b) \in D'}$ is the set of all (e, b) such that either $e \notin D$ or b is a valid general bound for Angluin's condition with respect to A_e . Now there is a function f dominating all the $G_{e,b}$ in the weakly represented family. Note that whenever $G_{e,b}$ is in this family then so is $G_{e,b+1}$ and $G_{e,b}(x) \geq G_{e,b+1}(x)$ for all x .

Without loss of generality one can assume that any number x does not appear in the text earlier than at stage x – this is achieved by inserting pause symbols into the text at all places where needed. Let $(e_0, b_0), (e_1, b_1), \dots$ be a sequence of pairs in which each pair of index and bound appears infinitely often. The learner has the initial counter value 0, the initial hypothesis e_0 and initial bound b_0 . Assume that after processing s items, the learner

has the counter n , the previous hypothesis e_n and the bound b_n . To determine whether an update to these parameters is needed, the learner now checks whether they satisfy all of the following conditions:

- We have $g(e_n, u + s) = 1$ for all $u \leq f(s)$ and all values $A_{e_n}(x)$ for $x \leq s$ can be retrieved from the representation set within time $f(s)$;
- It holds that $G_{e_n, b_n}(s)$ is defined within $f(s)$ steps;
- All data x with $x \leq b_n \wedge x \in A_{e_n}$ have been observed in the text so far;
- No datum x with $x \notin A_{e_n}$ has been observed in the text so far.

If (e_n, b_n) satisfies all these conditions then the learner keeps the counter n , hypothesis e_n and the bound b_n , else the learner changes the counter to $n + 1$, the hypothesis to e_{n+1} and the bound to b_{n+1} . Assume that the learner converges to an incorrect hypothesis e_n or a hypothesis with an incorrect bound b_n , then one of the following happens at some future stage s eventually:

- It holds that $g(e_n, u + s) = 0$ for some $u \leq f(s)$ (in the case that e_n is not a valid index);
- $G_{e_n, b_n}(s)$ is not defined (in the case that the bound b_n is invalid and that there is a $d \leq s$ inside D discovered with $A_e \cap \{0, 1, \dots, b_n\} \subseteq A_d \subset A_e$);
- Not all data in $A_e \cap \{0, 1, \dots, b_n\}$ have shown up in the text or some datum outside A_e has shown up in the text (in the case that the index and the bound are valid but that the hypothesis is not the correct one).

All these conditions imply that the hypothesis will be updated to e_{n+1} (and the bound to b_{n+1}) in contradiction to the assumption. The next possibility is that the learner would infinitely often have a counter value n such that (e_n, b_n) is some fixed correct pair (e, b) . As the function f dominates $G_{e, b}$, it holds for all sufficiently large s where the current (e_n, b_n) is equal to (e, b) that all four conditions from the above update test are satisfied and that therefore the current (e_n, b_n) will be kept and n will not be incremented. So the learner indeed converges to the correct hypothesis e . As the function h from s to the n currently processed is increasing and grows each step at most by one and is a member of \mathcal{S} , this function h is either eventually constant or has range M ; hence the above two cases (converging to a wrong hypothesis or taking one correct hypothesis infinitely often) are exhaustive and the learner is correct.

$5 \Rightarrow 4 \Rightarrow 3$. This follows from the definition.

$3 \Rightarrow 2$. Let $\{F_e\}_{e \in D}$ be any weakly represented family of functions (as represented by the set $F \in \mathcal{S}$). Now define a new weakly represented family $A_{\langle e, s \rangle}$ of sets such that $A_{\langle e, 0 \rangle} = \{\langle e, x \rangle : x \in M\}$ in case that $e \in D$ and let $\langle e, 0 \rangle$ be an invalid index in case that $e \notin D$. Let $A_{\langle e, s+1 \rangle} = \{\langle e, x \rangle : x \leq s\}$ in case that $s = \max\{\langle e, u, y, z \rangle \in F : u, y, z \in M\}$ and let $\langle e, s+1 \rangle$ be an invalid index otherwise. Note that for each e , there is a unique s such that $\langle e, s \rangle$ is a valid index: we denote the corresponding unique $A_{\langle e, s \rangle}$ as A_e .

Assume now that this weakly represented family is learnable in the limit. Then, uniformly in e , there is a text T_e which contains all the pairs $\langle e, x \rangle$ such that for some $\langle e, u, y, z \rangle \geq x$, $\langle e, u, y, z \rangle \in F$. This text T_e is a text for A_e . The learner converges on T_e to some index d in the limit. By simulating the learner one can make a function g such that

- $\lim_{t \rightarrow \infty} g(e, t)$ converges to 0 in the case that the learner converges on the text T_e to an index d for a set which does not contain $\langle e, x \rangle$ for some $x \in M$,
- $\lim_{t \rightarrow \infty} g(e, t)$ converges to 1 in the case that the learner converges on the text T_e to an index d for a set containing $\langle e, x \rangle$ for each $x \in M$.

The first case occurs iff $A_e = A_{\langle e, s+1 \rangle}$ for some s and the second case occurs iff $A_e = A_{\langle e, 0 \rangle}$. Here the first case coincides with $e \notin D$ and the second with $e \in D$. Thus g is correct. ◀

One might ask whether the necessity of DOM in this context is due to the difficulty of finding indices in weakly represented families rather than the difficulty of learning the languages. Therefore one might be inclined to choose a more comprehensive but somehow easier hypothesis space. However, in the proof of Theorem 5 ($3 \Rightarrow 2$) we only check whether the learner converges to an index of a set not containing some pair $\langle e, x \rangle$. As this is a property of the set, and not of its index, the choice of hypothesis space is not crucial for the proof.

Raghavan, Stephan and Zhang [18] investigate the strength of DOM. They show that under RCA_0 and $\mathcal{I}\Sigma_2$, DOM implies COH but not vice versa. This result is the counterpart to the recursion-theoretic result that every high Turing degree contains a cohesive set. Furthermore, for ω -models, there are also connections to set-theoretically motivated axioms. For example, DOM is true iff MAD is false [18]. Here MAD is the statement that there exists a *maximal almost disjoint family*, that is, a weakly represented family of sets $\{A_e\}_{e \in D}$ such that (i) for all $d, e \in D$ with $d \neq e$, $A_d \cap A_e$ is finite, and (ii) for every infinite $B \in \mathcal{S}$ there is an e such that $B \cap A_e$ is infinite. It is also known that DOM does not imply WKL_0 , the statement that every infinite binary tree in \mathcal{S} has an infinite branch in \mathcal{S} .

4 Uniformly Represented Families

We now show that Angluin's classical theorem also applies for uniformly represented families in the framework of reverse mathematics.

► **Theorem 6.** *Over RCA_0 , a uniformly represented family is learnable in the limit if and only if it satisfies Angluin's condition in the limit.*

One might ask when a learner exists in the case of general bounds in place of limit bounds.

► **Theorem 7.** *Over RCA_0 , DOM holds iff every uniformly represented family satisfying Angluin's condition with a general bound is learnable in the limit.*

Angluin [2] introduced the notion of conservative learning by requiring that a conservative learner only makes a mind change (that is, updates its hypothesis) if some datum observed so far is not contained in the previously conjectured set. Conservative learners do, therefore, never overgeneralise the language to be learnt. Thus before a conservative learner conjectures some language X it needs to ensure that there is no proper subset of X in the family being learnt that could explain the data observed so far. This requirement enforces the effective version of Angluin's condition and may require that the learner use a different hypothesis space than the family to be learnt. Such a hypothesis space is itself a family which needs to contain all sets from the family to be learnt but possibly also other sets.

► **Theorem 8.** *Over RCA_0 , a uniformly represented family $\{C_e\}_{e \in M}$ is conservatively learnable using some hypothesis space $\{A_e\}_{e \in M}$ if and only if $\{C_e\}_{e \in M}$ is contained in some uniformly represented family $\{B_e\}_{e \in M}$ (possibly different from $\{A_e\}_{e \in M}$) which satisfies Angluin's condition effectively.*

One might also ask in which cases every uniformly represented family satisfying Angluin's bound only in general is conservatively learnable. By Theorem 8 this only happens when for every uniformly represented family it is equivalent whether it satisfies Angluin's bound in general or effectively. This then allows coding the halting problem into such a family, and one obtains the following corollary. Finally, over ACA_0 , the index set D of any weakly represented family is in \mathcal{S} ; so the result carries over to weakly represented families.

► **Corollary 9.** *Over RCA_0 , the following statements are equivalent:*

1. ACA_0 (that is, every set arithmetically definable from parameters in \mathcal{S} is also in \mathcal{S} and, in particular, \mathcal{S} is closed under the Turing jump);
2. Every uniformly represented family satisfying Angluin's tell-tale condition with a general bound is conservatively learnable;
3. Every weakly represented family satisfying Angluin's tell-tale condition with a general bound is conservatively learnable.

5 Sufficient Criteria

Angluin [2] looked at sufficient criteria for learning. In the reverse mathematics setting, all these criteria can be proven to be sufficient over RCA_0 for uniformly represented families; for weakly represented families, the additional axiom DOM is again needed and sufficient to build the learners. The first of these criteria considered is finite thickness.

► **Theorem 10.** *Say that a family $\{A_e\}_{e \in D}$ has finite thickness if and only if every $x \in M$ is contained in only finitely many A_e , that is, for every $x \in M$ there is a bound b such that for all $e > b$, either $e \notin D$ or $x \notin A_e$ or $A_e = A_d$ for some $d \leq b$.*

1. Over RCA_0 , every uniformly represented family which has finite thickness is learnable in the limit.
2. Over RCA_0 , DOM is equivalent to the statement that every weakly represented family which has finite thickness is learnable in the limit.

The property of finite thickness has been strengthened to finite elasticity [20]. Finite elasticity mainly says that one cannot construct a text which in each step makes a concept inconsistent that was consistent before. Abstracting from the requirement that this happens in every step, one can also formulate this the other way round: A family has finite elasticity if and only if for every text there is a prefix of the text such that every concept inconsistent with the full text is also inconsistent with this prefix.

► **Theorem 11.** *Say that a family $\{A_e\}_{e \in D}$ has finite elasticity if and only if for every $T: M \rightarrow M \cup \{\#\}$ in \mathcal{S} there is a prefix $\sigma \preceq T$ such that for all $e \in D$, $\text{range}(\sigma) \subseteq A_e \Rightarrow \text{range}(T) \subseteq A_e$.*

1. Over RCA_0 , every uniformly represented family which has finite elasticity is learnable in the limit.
2. Over RCA_0 , DOM is equivalent to the statement that every weakly represented family which has finite elasticity is learnable in the limit.

Note that finite elasticity is only a sufficient criterion. For example the learnable class of all sets of the form $\{0, 1, \dots, e\}$ with $e \in M$ does not have finite elasticity.

Kobayashi [4, 16] considered another sufficient learnability criterion which is a further strengthening of the property of finite elasticity: A class is learnable if for every language A_e there is a finite subset E such that $E \subseteq A_d \Rightarrow A_e \subseteq A_d$ for all other languages A_d in the class. This learnability condition was proven in the context of indexed families and holds without any effectivity requirement on finding this finite subset. One can carry it over to uniformly represented and weakly represented families as follows.

► **Theorem 12.** *Say that a family $\{A_e\}_{e \in D}$ admits characteristic subsets if and only if for all $e \in D$ exists $b \in M$ such that for all $d \in D$ we have $A_e \cap \{0, 1, \dots, b\} \subseteq A_d \Rightarrow A_e \subseteq A_d$.*

1. Over RCA_0 , every uniformly represented family which admits characteristic subsets is learnable in the limit.

2. Over RCA_0 , DOM is equivalent to the statement that every weakly represented family which admits characteristic subsets is learnable in the limit.

Note that admitting characteristic subsets is a stronger property than Angluin's tell-tale criterion, as the former condition enforces $A_e \cap \{0, 1, \dots, b\} \subseteq A_d \Rightarrow A_e \subseteq A_d$ while Angluin's tell-tale criterion merely enforces $A_e \cap \{0, 1, \dots, b\} \subseteq A_d \Rightarrow A_d \not\subseteq A_e$.

6 Partial Learning

Osherson, Stob and Weinstein [17] introduced the notion of partial learning where to be successful a learner is required to output one correct hypothesis infinitely often and all other hypotheses at most finitely often. This fundamental concept allows to learn all classes of r.e. languages, provided that the hypothesis space permits padding. Our proofs of the corresponding results in reverse mathematics depend on the axiom $I\Sigma_2$ which, for example, proves that every set in a weakly represented family has a least index. It is unknown whether this is an inherent requirement for obtaining the statements, or one more involved arguments could dispense with.

- **Theorem 13.** *Over RCA_0 , a weakly represented family $\{A_d\}_{d \in D}$ is partially learnable if and only if there is a further weakly represented family $\{B_e\}_{e \in E}$ such that*
- *for all $d \in D$ there is exactly one $e \in E$ with $B_e = A_d$ and*
 - *all $e \in E$ are in D and satisfy $A_e = B_e$.*

That is, $\{B_e\}_{e \in E}$ is a trimmed version of $\{A_d\}_{d \in D}$ containing exactly one index for each set.

- **Theorem 14.** *Over RCA_0 , every uniformly represented family is partially learnable.*

- **Theorem 15.** *Over RCA_0 and $I\Sigma_2$, for every weakly represented family $\{A_e\}_{e \in D}$, there is a partial learner using the weakly represented family $\{B_{\langle e, b \rangle}\}_{e \in D, b \in M}$ with $B_{\langle e, b \rangle} = A_e$ for all $e \in D$, $b \in M$ as hypothesis space.*

Proof. Let a weakly represented family $\{A_e\}_{e \in D}$ be given, let A be its representation set and let $X \in \mathcal{S}$. Now consider the Σ_2 index set

$$I = \{e : \exists x \forall y, z [(e, x, y, z) \notin A \vee y \neq X(x)]\}$$

consisting of the e 's which are not indices of X in $\{A_e\}_{e \in D}$. In the case that X does not have a minimal index, the index set I satisfies for all e the property $(\forall d < e [d \in I] \Rightarrow e \in I)$ and then X does not have any index in the weakly represented family. Given the minimal index e of a member of the family, one can define for $d < e$ the uniform Σ_2 singletons

$$U_d = \{\min\{x : A_d(x) \text{ is not defined or } A_d(x) \neq A_e(x)\}\}.$$

Let b_e be the least upper bound on all numbers appearing in some U_d , with $d < e$. Now one defines the partial learner as follows: A hypothesis $\langle e, b \rangle$ is output at least n times if and only if there is $s \geq n$ such that the following conditions are satisfied:

- $A_e(0), A_e(1), \dots, A_e(n)$ can be retrieved from A in time s ;
- There is no $d < e$ such that for all $x \leq b$ the descriptions of $A_d(x)$ and $A_e(x)$ can be retrieved from A in time s and such that $A_d(x) = A_e(x)$;
- For all $b' < b$ there is a $d < e$ such that for all $x \leq b'$ the descriptions of $A_d(x)$ and $A_e(x)$ can be retrieved from A in time s and such that $A_d(x) = A_e(x)$.

One can verify that on a text for a member X of the weakly represented family, exactly one pair $\langle e, b \rangle$ is output infinitely often and this is given by the least index e of X and the least bound b such that all $d < e$ satisfy that either $A_d(b)$ is not defined or there is $x \leq b$ with $A_e(x) \neq A_d(x)$. Thus the family is partially learnt by the given learner. ◀

The previous result establishes that, over RCA_0 and $\text{I}\Sigma_2$, every member of a weakly represented family has a least index. This assumption is an essential ingredient of the learning algorithm and is equivalent to $\text{I}\Sigma_2$ over RCA_0 .

► **Proposition 16.** *Over RCA_0 , the axiom $\text{I}\Sigma_2$ is equivalent to the statement that in every weakly represented family, all its members have a minimal index.*

Proof. The sufficiency of $\text{I}\Sigma_2$ was already shown in Theorem 15. For the necessity assume that $\text{I}\Sigma_2$ is not satisfied. Then there is a Σ_2 set I which is a proper subset of M and satisfies for all e that $[(\forall d < e: d \in I) \Rightarrow e \in I]$. As the set is Σ_2 , there is a ternary $\{0, 1\}$ -valued function $g \in \mathcal{S}$ such that $e \in I \Leftrightarrow \exists n \in M \forall m \in M [g(e, n, m) = 1]$. Now define a weakly represented family such that every member of it is equal to M and its description A contains, for each e , inductively the pairs $\langle e, n, 1, z_n \rangle$ with $z_0 = 0$ and $z_{n+1} = z_n + m$ for the least m such that $g(e, n, m) = 0$. Now consider an arbitrary e .

- If there is a least n such that z_{n+1} is not defined then $g(e, n, m) = 1$ for all m and $e \in I$.
- If there is no least n with the property that z_{n+1} is not defined then consider the Σ_1 set $J = \{n: z_{n+1} \text{ is defined}\}$ and use Σ_1 -induction to show that $J = M$. It follows that all z_n are defined and thus for all n exists an $m = z_{n+1} - z_n$ with $g(e, n, m) = 0$. Hence $e \notin I$.

It follows that the complement of I is the index set of the so constructed weakly represented family and this index set does not contain a minimal element by the choice of I . However, the index set contains only indices of the unique member M of the family, contradiction. ◀

► **Theorem 17.** *Over RCA_0 , $\text{I}\Sigma_2$ and DOM , every weakly uniform family can be partially learnt using the family itself as hypothesis space.*

Proof. Given a weakly represented family $\{A_e\}_{e \in D}$ with representation set A , one can consider the weakly represented family of functions $\{G_e\}_{e \in D}$ such that for each $e \in D$ and $x \in M$, $G_e(x)$ is the unique tuple of the form $\langle e, x, y, z \rangle \in A$ defining $A_e(x)$. The family of these G_e is dominated by some function $f \in \mathcal{S}$. Now, the learner outputs an index e at least n times iff there is an $m \geq n$ and an $s \geq f(m)$ such that the following conditions are met:

1. $A_e(0), A_e(1), \dots, A_e(m)$ can be retrieved within time $f(m)$ from A ;
2. There is no $d < e$ such that $A_d(0), A_d(1), \dots, A_d(m)$ can be retrieved within time $f(m)$ from A and such that $A_d(0) = A_e(0), A_d(1) = A_e(1), \dots, A_d(m) = A_e(m)$;
3. All numbers $x \leq m$ with $A_e(x) = 1$ have occurred within the first s members of the text;
4. No number $x \leq m$ with $A_e(x) = 0$ has occurred within the first s members of the text.

The assumptions are sufficient to prove that this partial learner indeed succeeds to partially learn the languages in the family; note that the first two conditions together with f being a dominating function enforce that only minimal indices – whose existence is ensured by $\text{I}\Sigma_2$ – are output infinitely often and that the last two conditions enforce that a minimal index is output infinitely often iff it is correct. ◀

Acknowledgments. The authors would like to thank Chong Chi Tat and Yang Yue for interesting discussions on aspects of reverse mathematics related to this paper.

References

- 1 Lenny Adleman and Manuel Blum. Inductive inference and unsolvability. *The Journal of Symbolic Logic*, 56:891–900, 1991.
- 2 Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.
- 3 Lenore Blum and Manuel Blum. Towards a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.
- 4 Matthew de Brecht, Masanori Kobayashi, Hiroo Tokunaga, and Akihiro Yamamoto. Inferability of closed set systems from positive data. In *New Frontiers in Artificial Intelligence, JSAI 2006. Conference and Workshops, Tokyo, Japan, 5–9 June 2006, Revised Selected Papers*, volume 4384 of *Lecture Notes in Computer Science*, pages 265–275. Springer, 2007.
- 5 Matthew de Brecht and Akihiro Yamamoto. Mind change complexity of inferring unbounded unions of restricted pattern languages from positive data. *Theoretical Computer Science*, 411(7–9):976–985, 2010.
- 6 Matthew de Brecht and Akihiro Yamamoto. Topological properties of concept spaces (full version). *Information and Computation*, 208(4):327–340, 2010.
- 7 John Case and Chris Lynes. Machine inductive inference and language identification. In *Proceedings of the Ninth International Colloquium on Automata, Languages and Programming*, volume 140 of *Lecture Notes in Computer Science*, pages 107–115. Springer, 1982.
- 8 Chi Tat Chong, Theodore A. Slaman, and Yue Yang. Π_1^1 -conservation of combinatorial principles weaker than Ramsey’s theorem for pairs. *Advances in Mathematics*, 230(3):1060–1077, 2012.
- 9 Chi Tat Chong, Theodore A. Slaman, and Yue Yang. The metamathematics of stable Ramsey’s theorem for pairs. *The Journal of the American Mathematical Society*, 27:863–892, 2014.
- 10 Damir D. Dzharafarov and Carl Mummert. On the strength of the finite intersection principle. *Israel Journal of Mathematics*, 196:345–361, 2013.
- 11 Harvey Friedman. Some systems of second order arithmetic and their use. In *Proceedings of the International Congress of Mathematicians, Vancouver, 1974*, volume 1, pages 235–242, 1974.
- 12 E. Mark Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
- 13 Susumu Hayashi. Mathematics based on incremental learning — excluded middle and inductive inference. *Theoretical Computer Science*, 350:125–139, 2006.
- 14 Dennis Hirschfeldt. *Slicing the Truth. On the Computable and Reverse Mathematics of Combinatorial Principles*, volume 28 of *Lecture Notes Series, Institute for Mathematical Sciences, National University of Singapore*. World Scientific, 2014.
- 15 Sanjay Jain, Daniel Osherson, James Royer, and Arun Sharma. *Systems That Learn: An Introduction to Learning Theory, Second Edition*. The MIT Press, Cambridge, Massachusetts, 1999.
- 16 Satoshi Kobayashi. Approximate identification, finite elasticity and lattice structure of hypothesis space. Technical Report CSIM 96–04, Department of Computer Science and Information Mathematics, University of Electro-Communications, 1996.
- 17 Daniel Osherson, Michael Stob, and Scott Weinstein. *Systems That Learn, An Introduction to Learning Theory for Cognitive and Computer Scientists*. The MIT Press, Cambridge, Massachusetts, 1986.
- 18 Dilip Raghavan, Frank Stephan, and Jing Zhang. Weakly represented families in reverse mathematics. Manuscript, 2014.
- 19 Stephen G. Simpson. *Subsystems of Second Order Arithmetic*. Cambridge University Press, 2009.

- 20 Keith Wright. Identification of unions of languages drawn from an identifiable class. In *Proceedings of the Second Annual Workshop on Computational Learning Theory*, pages 328–333. Morgan Kaufmann, 1989.
- 21 Thomas Zeugmann, Steffen Lange, and Shyam Kapur. Characterizations of monotonic and dual monotonic language learning. *Information and Computation*, 120(2):155–173, 1995.

Dynamic Planar Embeddings of Dynamic Graphs

Jacob Holm and Eva Rotenberg

DIKU, Dept. of Computer Science, University of Copenhagen, Denmark
jh@poplar.dk, roden@di.ku.dk

Abstract

We present an algorithm to support the dynamic embedding in the plane of a dynamic graph. An edge can be inserted across a face between two vertices on the boundary (we call such a vertex pair linkable), and edges can be deleted. The planar embedding can also be changed locally by flipping components that are connected to the rest of the graph by at most two vertices. Given vertices u, v , $\text{linkable}(u, v)$ decides whether u and v are linkable, and if so, returns a list of suggestions for the placement of (u, v) in the embedding. For non-linkable vertices u, v , we define a new query, $\text{one-flip-linkable}(u, v)$ providing a suggestion for a flip that will make them linkable if one exists. We will support all updates and queries in $O(\log^2 n)$ time. Our time bounds match those of Italiano et al. for a static (flipless) embedding of a dynamic graph. Our new algorithm is simpler, exploiting that the complement of a spanning tree of a connected plane graph is a spanning tree of the dual graph. The primal and dual trees are interpreted as having the same Euler tour, and a main idea of the new algorithm is an elegant interaction between top trees over the two trees via their common Euler tour.

1998 ACM Subject Classification E.1 Data Structures, F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases dynamic graphs, planar embeddings, data structures

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.434

1 Introduction

We present a data structure for supporting and maintaining a dynamic planar embedding of a dynamic graph. In this article, a *dynamic graph* is a graph where edges can be removed or inserted, and vertices can be cut or joined, but where an edge (u, v) can only be added if it does not violate planarity. More precisely, the edges around each vertex are ordered cyclically by the embedding, similar to the edge-list representation of the graph. A *corner* (of a face) is the gap between consecutive edges incident to some vertex. Given two corners c_u and c_v of the same face f , incident to the vertices u and v respectively, the operation $\text{insert}(c_u, c_v)$ inserts an edge between u and v in the dynamic graph, and embeds it across f via the specified corners. We provide an operation $\text{linkable}(u, v)$ that returns such a pair of corners c_u and c_v if they exist. If there are more options, we can list them in constant time per option after the first. A vertex may be cut through two corners, and linkable vertices may be joined by corners incident to the same face, if they are connected, or incident to any face otherwise. That is, joining vertices corresponds to linking them across a face with some edge e , and then contracting e .

It may often be relevant to change the embedding, e.g. in order to be able to insert an edge. In a *dynamic embedding*, the user is allowed to change the embedding by what we call flips, that is, to turn part of the graph upside down in the embedding. Of course, the relevance of this depends on what we want to describe with a dynamic plane graph. If the application is to describe roads on the ground, flipping orientation would not make much



© Jacob Holm and Eva Rotenberg;

licensed under Creative Commons License CC-BY

32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).

Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 434–446

Leibniz International Proceedings in Informatics



LIPIcs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



sense. But if we have the application of graph drawing or chip design in mind, flips are indeed relevant. In the case of chip design, a layer of a chip is a planar embedded circuit, which can be thought of as a planar embedded graph. An operation similar to flip is also supported by most drawing software.

Given two vertices u, v , we may ask whether they can be linked after modifying the embedding with only one flip. We introduce a new operation, the $\text{one-flip-linkable}(u, v)$ query, which answers that question, and returns the vertices and corners describing the flip if it exists.

Our data structure is an extension to a well-known duality-based dynamic representation of a planar embedded graph known as a tree-cotree decomposition [4]. We maintain top-trees [1] both for the primary and dual spanning trees. We use the fact that they share a common (extended) Euler tour - in a new way - to coordinate the updates and enable queries that either tree could not answer by itself. All updates and queries in the combined structure are supported in $O(\log^2 n)$, plus, in case of $\text{linkable}(u, v)$, the length of the returned list.

1.1 Dynamic Decision Support Systems

An interesting and related problem is that of dynamic planarity testing of graphs. That is, we have a planar graph, we insert some edge, is the graph still planar, that is, does there still exist an embedding of it in the plane?

The problem of dynamic planarity testing appears technically harder than our problem, and in its basic form it is only relevant when the user is completely indifferent to the actual embedding of the graph. What we provide here falls more in the category of a decision support system for the common situation where the desired solution is not completely captured by the simple mathematical objective, in this case planarity. We are supporting the user in finding a good embedding, e.g., telling what are the options for inserting an edge (the linkable query), but leave the actual choice to the user. We also support the users in changing their mind about the embedding, e.g. by flipping components, so as to make edge insertions possible. Using the one-flip-linkable query we can even suggest a flip that would make a desired edge insertion possible if one exists.

1.2 Previous work

Dynamic graphs have been studied for several decades. Usually, a fully dynamic graph is a graph that may be updated by the deletion or insertion of an edge, while decremental or incremental refers to graphs where edges may only be deleted or inserted, respectively. A dynamic graph can also be one where vertices may be deleted along with all their incident edges, or some combination of edge- and vertex updates [14].

Hopcroft and Tarjan [9] were the first to solve planarity testing of static graphs in linear time. Incremental planarity testing was solved by La Poutre [12], who improved on work by Di Battista, Tamassia, and Westbrook [2, 3, 15], to obtain a total running time of $O(\alpha(q, n))$ where q is the number of operations, and where α is the inverse-Ackermann function. Galil, Italiano, and Sarnak [8] made a data structure for fully dynamic planarity testing with $O(n^{2/3})$ worst case time per update, which was improved to $O(n^{1/2})$ by Eppstein et al. [5]. For maintaining embeddings of planar graphs, Italiano, La Poutre, and Rauch [10] present a data structure for maintaining a planar embedding of a dynamic graph, with time $O(\log^2 n)$ for update and for linkable-query, where insert only works when compatible with the embedding. The dynamic tree-cotree decomposition was first introduced by Eppstein et al. [6] who used it to maintain the MST of a planar embedded dynamic graph subject to a sequence of

change-weight($e, \Delta x$) operations in $O(\log n)$ time per update. Eppstein [4] presents a data structure for maintaining the MST of a dynamic graph, which handles updates in $O(\log n)$ if the graph remains plane. More precisely the user specifies a combinatorial embedding in terms a cyclic ordering of the edges around each vertex. Planarity of the user specified embedding is checked using Euler's formula. Like our algorithm, Eppstein's supports flips. The fundamental difference is that Eppstein does not offer any support for keeping the embedding planar, e.g., to answer linkable(u, v), the user would in principle have to try all n^2 possible corner pairs c_u and c_v incident to u and v , and ask if insert(c_u, c_v) violates planarity.

As far as we know, the one-flip-linkable query has not been studied before. Technically it is the most challenging operation supported in this paper.

The highest lower bound for the problem of planarity testing is Pătraşcu's $\Omega(\log n)$ lower bound for fully dynamic planarity testing [13]. From the reduction it is clear that this lower bound holds as well for maintaining an embedding of a planar graph as we do in this article.

2 Maintaining a dynamic embedding

In this section we present a high-level overview of a data structure to maintain a dynamic embedding of a planar graph. In the following, unless otherwise stated, we will assume $G = (V, E)$ is a planar graph with a given combinatorial embedding and that $G^* = (F, E^*)$ is its dual.

Our primary goal is to be able to answer linkable(u, v), where u and v are vertices: Determine if an edge between u and v can be added to G without violating planarity and without changing the embedding. If it can be inserted, return the list of pairs of *corners* (see Definition 3 below). Each corner-pair, (c_u, c_v) , describes a unique place where such an edge may be inserted. If no such pair exists, return the empty list.

The data structure must allow efficient updates such as insert, remove, cut, join, and flip. We defer the exact definitions of these operations to Section 2.4.

As in most other dynamic graph algorithms we will be using a spanning tree as the main data structure, and note:

► **Observation 1.** *If $E_T \subseteq E$ induces a spanning tree T in G , then $(E \setminus E_T)^*$ induces a spanning tree \bar{T}^* in G^* called the co-tree of T .*

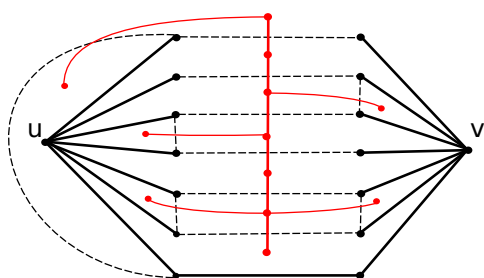
► **Observation 2.** *If u and v are vertices, T is spanning tree, and e is any edge on the T -path between u and v , then any face containing both u and v lies on the cycle induced by e^* in the co-tree \bar{T}^* .*

Thus the main idea is to search a path in the co-tree. This is complicated by the fact (see Figure 1) that the set of faces that are adjacent to u and/or to v need not be contiguous in \bar{T}^* , so it is possible for the cycle to change arbitrarily between faces that are adjacent to any combination of neither, one, or both of u, v .

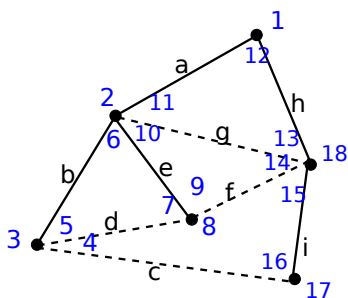
Our linkable query will consist of two phases. A marking phase, in which we "activate" (mark) all corners incident to each of the two vertices we want to link (see Section 2.2), and a searching phase, in which we search for faces incident to "active" (marked) corners at both vertices (see Section 2.3). But first, we define corners.

2.1 Corners and the extended Euler tour

The concept of a corner in an embedded graph turns out to be very important for our data structure. Intuitively it is simply a place in the edge list of a vertex where you might insert a new edge, but as we shall see it has many other uses.



■ **Figure 1** The co-tree path may switch arbitrarily between faces that are adjacent to any combination of neither, one, or both of u, v .



■ **Figure 2** This graph has extended Euler Tour $1a2b3c4d5b6e7d8f9e\dots18h$, or, to write only the edges: **abcdbedfegahgfcih**. Edges not in the spanning tree are drawn with dotted lines.

► **Definition 3.** If G is a non-trivial connected, combinatorially embedded graph, a *corner* in G is a 4-tuple (f, v, e_1, e_2) where f is a face, v is a vertex, and e_1, e_2 are edges (not necessarily distinct) that are consecutive in the edge lists of both v and f . If $G = (\{v\}, \emptyset)$ and $G^* = (\{f\}, \emptyset)$, there is only one corner, namely the tuple (f, v) .

Note that faces and vertices appear symmetrically in the definition. Thus, there is a one-to-one correspondence between the corners of G and the corners of G^* . This is important because it lets us work with corners in G and G^* interchangeably. Even more interesting is:

► **Observation 4.** Given a spanning tree T of G , there is a natural extension of the concept of an Euler tour of T into an extended Euler tour $EET(T)$ as a cyclic arrangement that contains each edge in G exactly twice and each corner in G exactly once (see Figure 2). Furthermore, the corresponding tour $EET(\bar{T}^*)$ in G^* defines exactly the opposite cyclical arrangement of the corresponding edges and corners in G^* .

Thus, segments of the extended Euler tour translate directly between T and \bar{T}^* . By *segment*, we mean any contiguous sub-list of the cycle.

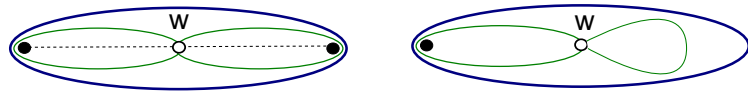
The high level algorithm (to be explained in detail later) is now to build a structure consisting of an arbitrary spanning tree T and its co-tree \bar{T}^* such that we can

1. Find an edge e on the T -path between u and v . This is easy.
2. Mark all corners incident to u and v in T . This is complicated by the existence of vertices of high degree, so a lazy marking scheme is needed. However, it is easier than marking them in \bar{T}^* directly, since each vertex has a unique place in T and no place in \bar{T}^* .
3. Transfer those marks from T to \bar{T}^* using Observation 4. We can do this as long as the lazy marking scheme works in terms of segments of the extended Euler tour.
4. Search the cycle induced by e^* in \bar{T}^* for faces that are incident to a marked corner on both sides of the path.

2.2 Marking scheme

We need to be able to mark all corners incident to the two query vertices u and v , and we need to do it in a way that operates on segments of the extended Euler tour. To this end

■ **Figure 3** The vertex w is a boundary vertex of the green clusters, but not of their blue parent clusters.



we will maintain a top tree over T (see [1]).

Given a tree T , a top tree for T is a binary tree. At the lowest level, its leaves are the edges of T . Its internal nodes, called *clusters*, are sub-trees of T with at most two boundary vertices. At the highest level its root is a single cluster containing all of T . A non-boundary vertex of the subset $S \subset V$ may not be adjacent to a vertex of $V \setminus S$. A cluster with two boundary vertices is called a *path cluster*, and other clusters are called *leaf clusters*. Any internal node is formed by the merged union of its (at most two) children. All operations on the top tree are implemented by first breaking down part of the old tree from the top with $O(\log n)$ calls to a *split* operation, end then building the new tree with $O(\log n)$ calls to a *merge* operation.

► **Observation 5.** *We can maintain a top tree over T such that each cluster consists of edges from at most two segments of $EET(T)$, using $O(\log n)$ calls to merge and split per update. Path clusters will have edges from two segments, and leaf clusters, one segment.*

The operation *expose* on the top tree takes one or two vertices and make them boundary of the level root cluster. Modifying this only slightly, one may even *expose* corners, giving complete control over which $EET(T)$ segment is available for information or modification. We may even expose any constant number of vertices or corners, but then at the highest level of the top tree, in stead of only T , we may have some constant number of clusters.

► **Observation 6.** *Whenever a merge of two clusters in the top tree causes a vertex w to stop being a boundary vertex (see Figure 3), all corners incident to w are contained in one or two $EET(T)$ segments. These segments will be sub-segments of the (one or two) segments corresponding to the parent cluster C (blue in Figure 3), and will not contain any corners incident to the (one or two) boundary vertices of C .*

Now suppose we associate a (lazy) *deactivation count* with each corner that is 0 before we start building the top tree. Define the merge operation on the top tree such that whenever a merge discards a boundary vertex we *deactivate* all corners on the at most two segments of $EET(T)$ mentioned in Observation 6 by increasing that count (and define the split operation on the top tree to reactivate them as necessary). When the top tree is complete, the corners that are still *active* (have deactivation count 0) are exactly those incident to the boundary vertices of the root of the top tree. These boundary vertices are controlled by the *expose* operation on the top tree and changing the boundary vertices require only $O(\log n)$ merges and splits, so we have now argued the following

► **Lemma 7.** *We can mark/unmark all corners incident to vertices u and v by increasing and decreasing the deactivation counts on $O(\log n)$ segments of the extended Euler tour.*

What we really want is to be able to search for the marked corners in \overline{T}^* , so instead of storing the counts (even lazily) in the top tree over T , we will store them in a top tree over \overline{T}^* . Again, each cluster in this top tree covers one or two segments of the extended Euler tour. For each segment S we keep track of the minimum deactivation count $c_{\min}(S)$, and a $\delta(S)$ that needs to be applied to all corners in the segment. To update the deactivation counts of an arbitrary segment S , all we need to do is modify the $O(\log n)$ clusters that are affected, which can be done in $O(\log n)$ time, leading to

► **Lemma 8.** *We can maintain a top tree over \overline{T}^* that has c_{\min} and δ values for each EET segment in each cluster in $O(\log^2 n)$ time per change to the marked u and v vertices.*

► **Observation 9.** *This is enough for, given a face f and a vertex u , checking whether f is incident to u .*

2.3 Linkable query

Unfortunately, the c_{\min} and δ values discussed in Section 2.2 are not quite enough to let us find the corners we are looking for. We can use it to ask what marked corners a given face is incident to, but we do not have enough to find *pairs* of marked corners on opposite sides of the same face on the co-tree path.

As noted in Observation 2 all candidates to a common face for two given vertices u and v , must lie on some path in the dual tree. And a path which is easily found! Since the dual of a primal tree edge induces a cycle that separates u and v , we may use the path between the dual endpoints f, g of any edge on the primal tree path between u and v . Furthermore, once we expose the path (f, g) in the dual tree, if $f \neq g$, it will have two EET-segments: the minimum deactivation count of one EET-segment is 0 iff any non-endpoint faces are incident to v , the other iff any are incident to u . Checking the endpoint faces can be done (cf. Observation 9), but to find non-endpoint faces we need more structure.

To just output *one* common face, our solution is for each path cluster in the top tree over the co-tree to keep track of a *single* internal face f_{\min} on the cluster path that is incident to minimally deactivated corners on either side of the cluster path if such a face exists.

► **Lemma 10.** *We can maintain a top tree over \overline{T}^* that has c_{\min} and δ values for each EET-segment in each cluster and f_{\min} values for each path cluster in $O(\log^2 n)$ time per change to the marked u and v vertices.*

Proof. Each merge only has to check the at most two f_{\min} values at the children and may discard or keep them based solely on the c_{\min} and δ values available. ◀

► **Lemma 11.** *We can support each $\text{linkable}(u, v)$ in $O(\log^2 n)$ time per operation.*

Proof. If u and v are not in the same connected component we pick any corners c_u and c_v adjacent to u and v and return them. Otherwise we use $\text{expose}(u, v)$ on the top tree over T to activate all corners adjacent to u and v and to find an edge e on the T -path from u to v (e.g. the first edge on the path). Let f, g be the endpoints of e^* , and call $\text{expose}(f, g)$ on the top tree over \overline{T}^* . Let h be the f_{\min} value of the resulting root. We can now test each of f, g, h using the c_{\min} values to find the desired corners if they exist. ◀

► **Lemma 12.** *If there are more valid answers to $\text{linkable}(u, v)$ we can find k of them in $O(\log^2 n + k)$ time.*

Proof. For each leaf cluster and for each side of each path cluster we can maintain the list of minimally deactivated corners adjacent to each boundary vertex. Then, instead of maintaining a single face f_{\min} for each path cluster, we can maintain a linked list of all relevant faces in the same time. And for each side of each face in the list we can point to a list of minimally deactivated corners that are adjacent to that side. For leaf-clusters, we point to a linked list of minimally deactivated corners incident to the boundary vertex. Upon the merge of clusters, face-lists and corner-lists may be linked together, and the point of concatenation is stored in the resulting merged cluster in case of a future split. Note that each face occurs in exactly one face-list.

As before, to perform $\text{linkable}(u, v)$, expose u, v in the primal tree. Let e_0 be an edge on the tree-path between u and v , and expose the endpoints of e_0^* in the dual top tree. Now, the maintained face-list in the root of the dual top tree contains all faces incident to u, v , except maybe the endpoints of e_0^* , which can be handled separately, as before. The total time is therefore $O(\log^2 n)$ for the necessary expose operations, and then $O(1)$ for each reply. ◀

► **Observation 13.** *If we separately maintain a version of this data structure for the dual graph, then for faces f, g , $\text{linkable}(f, g)$ in that structure lets us find vertices that are incident to both f and g .*

2.4 Updates

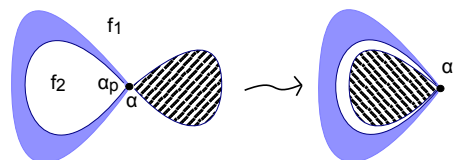
In addition to the query, our data structure supports the following set of update operations:

- $\text{insert}(c_u, c_v)$ where c_u and c_v are corners that are either in different connected components, or incident to the same face. Adds a new edge to the graph, inserting it between the edges of c_u at one end and between the edges of c_v at the other. Returns the new edge.
- $\text{remove}(e)$. Removes the edge e from the graph. Returns the two corners that could be used to insert the edge again.
- $\text{join}(c_u, c_v)$ where c_u and c_v are corners that are either in separate components of the graph or in the same face. Combines the vertices u and v into a single new vertex w and returns the two new corners c_w and c'_w that may be used to split it again using $\text{cut}(c_w, c'_w)$.
- $\text{cut}(c_w, c'_w)$ where c_w and c'_w are corners sharing a vertex w . Splits the vertex into two new vertices u and v and returns corners c_u and c_v that might be used to join them again using $\text{join}(c_u, c_v)$.
- $\text{flip}(C)$ where C is any connected component of the graph. Reverses the order of the edges at each vertex/face cycle of the component.

When calling $\text{remove}(e)$ on a non-bridge tree-edge e , we need to search for a replacement edge. Luckily, e^* induces a cycle in the dual tree, and any other edge on that cycle is a candidate for a replacement edge. If we like, we can augment the dual top tree so we can find the minimal-weight replacement edge, simply let each path cluster remember the cheapest edge on the tree-path, and expose the endpoints of e^* . If we want to keep T as a minimum spanning tree, we also need to check at each insert and join that we remove the maximum-weight edge on the induced cycle from the spanning tree.

In general, when we need to update both the top trees over T and \bar{T}^* we must be careful that we first do the splits needed in the top tree over T to make each unchanged sub-tree into a (partial) top tree by itself, then update the top tree over \bar{T}^* and finally do the remaining splits and merges to rebuild the top tree over T . This is necessary because the merge and split we use for T depend on T and \bar{T}^* having related extended Euler tours.

Any change to the graph, especially to the spanning tree, implies a change to the extended Euler Tour. Furthermore, any deletion or insertion of an edge implies a merge or split in the dual tree. E.g. if an edge is inserted across a face, that face is split in two. As a more complex example, if the non-bridge tree-edge $e = (u, v)$ is deleted, the replacement edge is removed from the dual tree, and the endpoints of e^* are merged.



■ **Figure 4** An articulation flip at the vertex α .



Figure 5 A separation flip at a separation pair (blue). The flip makes vertex u linkable with vertex v .

Finally, for flip to work we have to use a version of top trees that is not tied to a specific clockwise orientation of the vertices. The version in [1] that is based on a reduction to Frederickson’s topology trees [7] works fine for this purpose.

► **Definition 14** (Articulation flip). Having vertex split and vertex join functions, we may perform an *articulation-flip* — a flip in an articulation point: Given a vertex α incident to the face f_1 in two corners, c and c' , we may cut through c, c' , obtaining two graphs G_1, G_2 , having split α in vertices $\alpha_1 \in G_1, \alpha_2 \in G_2$, and having introduced new corners c_1, c_2 where we cut. Now, given a corner α_p incident to α_1 and incident to some face f_2 , we may join α_1 with α_2 by the corners α_2, α_p , with or without having flipped the orientation of G_2 .

► **Definition 15** (Separation flip). Similarly, given a separation pair α, β , incident to the faces f, g with corners c_1, \dots, c_4 , we may split through those corners, obtaining two graphs. We may then flip the orientation of one of them, and rejoin. We call this a *separation-flip*.

3 One-flip linkable query

Given vertices u, v , we have already presented a data structure to find a common face for u, v . Given they do not share a common face, we will determine if an articulation flip exists such that an edge between them can be inserted, and given no such articulation-flip exists, we will determine if a separation-flip that makes the edge insertion (u, v) possible exists.

Let f_1 and f_2 be faces in G , and let S be a subgraph of G . We say that S separates f_1 and f_2 if f_1 and f_2 are not connected in $G^* \setminus (E[S])^*$. Here, $E[X]$ denotes the set of edges of the subgraph X , $E[f]$ the edges incident to the face f , and $V[f]$ the incident vertices.

► **Observation 16.** Given a cycle C that is induced in $T \cup \{e\}$ by some edge e and given any two faces f_1, f_2 not separated by C , any face f such that $C \cup E[f]$ separates f_1 and f_2 lies on the path $f_1 \cdots f_2$ in \bar{T}^* .

Let f_1 and f_2 be faces of G , and let $S = V[f_1] \cap V[f_2]$ be the set of vertices they have in common. Let C denote the set of corners between vertices in S and faces in $\{f_1, f_2\}$. The sub-graphs obtained by cutting G through all the corners of C are called *flip-components* of G w.r.t. f_1 and f_2 . Flip-components which are only incident to one vertex of S can be flipped with an articulation-flip, and flip-components incident to two vertices can be flipped with a separation-flip. (See Figure 6.)

► **Observation 17.** Note that the perimeter of a flip component always consists of the union of a path along the face of f_u with a path along the face of f_v . One of these paths is trivial (equal to a point) exactly when u, v are linkable via an articulation-flip.

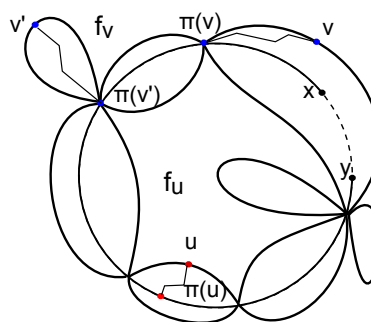


Figure 6 The faces f_u and f_v have five common vertices, and there are eight flip-components with respect to them.

Given vertices u, v in G , that are connected and not incident to a common face, we wish to find faces f_u and f_v such that u and v are in different flip-components w.r.t. f_u and f_v .

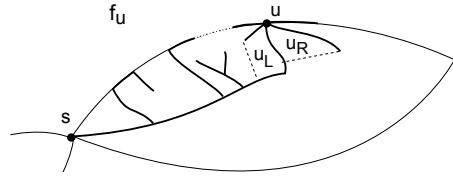
3.1 Finding one face

Let u and v be given vertices, and assume there exists faces f_u and f_v such that $u \in V[f_u] \setminus V[f_v]$, $v \in V[f_v] \setminus V[f_u]$, and u and v are in different flip-components w.r.t. f_u and f_v .

Let u_L, u_R be the left and right faces adjacent to the first edge on the path from u to v . Similarly let v_L, v_R be the left and right faces adjacent to the first edge on the path from v to u .

► **Lemma 18.** *Face f_u is on the \overline{T}^* -path $u_L \cdots u_R$ and face f_v is on the \overline{T}^* -path $v_L \cdots v_R$.*

Proof. For symmetry reasons, we need only be concerned with the case f_u . The perimeter of a flip-component consists of edges incident to f_u and edges incident to f_v (see Observation 17). Furthermore, in order for u, v to be linkable via a flip, u needs to lie on the perimeter of its flip-component. We also know that the tree-path from v to u must go through a point s in S which lies on the boundary of u 's flip-component. Thus, there must exist a path p in G from $\pi \in S$ to u , consisting only of edges incident to f_u . Note that $u \notin S$ since u, v were not already linkable. If the first edge e_u on the tree path from u to v is not already incident to f_u , then the union of p and the tree must contain an induced cycle containing e_u , separating u_L from u_R , induced by an edge e_i incident to f_u . (See Figure 7.) But then, the co-tree path from u_L to u_R goes through e_i^* , which means it goes through f_u . ◀



► **Figure 7** The co-tree path from u_L to u_R goes through f_u . The proof uses that the tree-path from u to v goes through some $s \in S$ on the boundary of u 's flip-component.

► **Lemma 19.** *If there exists an induced cycle C separating f_u from f_v such that $u \notin V[C]$ and $v \in V[C]$, then $f_u = \text{meet}(u_L, u_R, f)$ where $f \in \{v_L, v_R\}$ is the face that is on the same side of C as u . Here, $\text{meet}(a, b, c)$ denotes a 's projection to the path $b \cdots c$.*

Proof. By Lemma 18 f_u is on the path $u_L \cdots u_R$. And since $C \cup E[f_u]$ separates f from u_L and u_R it is on the paths $u_L \cdots f$ and $u_R \cdots f$ by Observation 16. ◀

► **Lemma 20.** *If there exists an induced cycle C separating f_u from f_v such that $u \notin V[C]$ and $v \notin V[C]$, then either $f_u = \text{meet}(u_L, u_R, v_L) = \text{meet}(u_L, u_R, v_R)$ or $f_v = \text{meet}(v_L, v_R, u_L) = \text{meet}(v_L, v_R, u_R)$.*

Proof. Let e be the edge in $C \setminus T$, and let e_u, e_v be the faces adjacent to e that are on the same side of C as f_u and f_v respectively. Then e is on all 4 paths in \overline{T}^* with u_L or u_R at one end and v_L or v_R at the other. At least one of u, v is in a different flip-component from e , so we can assume without loss of generality that u is. By Lemma 18 f_u is on the path $u_L \cdots u_R$. And since $C \cup f_u$ separates u_L and u_R from e_u , f_u is on both the paths $u_L \cdots e_u$ and $u_R \cdots e_u$ by Observation 16. Thus $f_u = \text{meet}(u_L, u_R, e_u) = \text{meet}(u_L, u_R, v_L) = \text{meet}(u_L, u_R, v_R)$. ◀

► **Lemma 21.** *If an induced cycle C separates f_u from f_v such that $u \in V[C]$ and $v \in V[C]$, then either $f_u = \text{meet}(u_L, v_L, v_R) = \text{meet}(u_L, u_R, v_R)$ or $f_u = \text{meet}(u_R, v_L, v_R) = \text{meet}(u_L, u_R, v_L)$ or $f_v = \text{meet}(v_L, u_L, u_R) = \text{meet}(v_L, v_R, u_R)$ or $f_v = \text{meet}(v_R, u_L, u_R) = \text{meet}(v_L, v_R, u_L)$.*

Proof. Let e be the edge in $C \setminus T$, and let e_u, e_v be the faces adjacent to e that are on the same side of C as f_u and f_v respectively. Then e is on all 4 paths in \overline{T}^* with u_L or u_R at one

end and v_L or u_R at the other. Assume that u_L and v_R are on the side of C containing f_u and u_R and v_L are on the side of C containing f_v . At least one of u, v is in a different flip-component from e , so assume that v is. By lemma 18 f_u is on the path $u_L \cdots e_u \subset u_L \cdots u_R$. And since $C \cup f_u$ separates u_L and e_u from v_R it is on both the paths $u_L \cdots v_R$ and $e_u \cdots v_R$ by Observation 16. Thus $f_u = \text{meet}(u_L, e_u, v_R) = \text{meet}(u_L, v_L, v_R) = \text{meet}(u_L, u_R, v_R)$. The remaining cases are symmetric. ◀

► **Theorem 22.** *If f_u, f_v exist, either $f_u \in \{\text{meet}(u_L, u_R, v_L), \text{meet}(u_L, u_R, v_R)\}$ or $f_v \in \{\text{meet}(u_L, v_L, v_R), \text{meet}(u_R, v_L, v_R)\}$.*

Proof. If they exist there is at least one induced cycle C separating them. This cycle must have the properties of at least one of Lemmas 19, 20, or 21. ◀

By computing the at most two different meet values and checking which ones (if any) contain u or v we therefore get at most two candidates and are guaranteed that at least one of them is in $\{f_u, f_v\}$ if they exist.

► **Lemma 23.** *Given a top tree over a tree T , with vertices $a, b, c \in T$, we can find $\text{meet}(a, b, c)$ in logarithmic time.*

Proof. Split all clusters containing a, b , or c as a non-boundary vertex. There are only $O(\log n)$ of those. After these split-operations, we have a tree with $O(\log n)$ vertices. Use this tree to find $\text{meet}(a, b, c)$ in linear time. ◀

3.2 Finding the other face

► **Lemma 24.** *Let u, v , and f_u be given. Then the first edge e_L on $f_u \cdots v_L$ or the first edge e_R on $f_u \cdots v_R$ induces a cycle $C(e_R)$ or $C(e_L)$ in T that separates f_u from f_v .*

Proof. By lemma 18, f_v is on $v_L \cdots v_R$ in \overline{T}^* , so the first edge on $f_u \cdots f_v$ is also the first edge on either $f_u \cdots v_L$ or $f_u \cdots v_R$. ◀

Thus given the correct f_u we can find at most two candidates for an edge e that induces a cycle $C(e)$ in T that separates f_u from f_v , and be guaranteed that one of them is correct.

► **Observation 25.** *For each vertex, v , we may consider the projection $\pi(v)$ of v onto the cycle C . For each flip-component, X , we may consider the projection $\pi(X) = \{\pi(v) \mid v \in X\}$. If X is an articulation-flip component, the projection $\pi(X)$ is a single point in $S = V[f_u] \cap V[f_v]$. If X is a separation-flip component, its projection is a segment of the cycle, $\pi_1 \cdots \pi_2$, between the separation pair $(\pi_1, \pi_2) \subset C(e)$ where $\pi_1, \pi_2 \in S$.*

3.2.1 Finding an articulation-flip

Let (x, y) be any edge inducing a cycle C in $T \cup \{(x, y)\}$ that separates f_u from f_v , let $\pi(u) = \text{meet}_T(u, x, y)$ be the projection of u on C .

Now the articulation-flip cases are not necessarily symmetrical. First we present how to detect an articulation-flip, given u, v , and f_u , if f_v plays the role of f_2 (see Definition 14).

If the flip-component containing v is an articulation-flip component, then $\pi(v)$ is an articulation point incident to both f_u and f_v , but the opposite is not necessarily the case. Assume $\pi(v)$ is incident to both f_u and f_v and let c_u denote a corner between $\pi(v)$ and f_u .

Note that if $\pi(v)$ is an articulation point with corners c_1, c_2 both incident to f_v , then f_v is an articulation point in the dual graph with corners c_1, c_2 both incident to $\pi(v)$. Removing

■ **Figure 8** If f_v is an articulation point, so is $\pi(v)$. But then the co-tree path from u_L to v_L must go through f_v . Left: Primal graph. Right: Dual graph.



f_v from the dual graph would split its component into several components, and clearly, aside from f_v , only faces in *one* of these components may contain faces incident to v . Any path in the co-tree starting and ending in different components w.r.t the split will have the property that the first face incident to v on that path is f_v . (See Figure 8.)

Now, in the case $f_u = f_1$ and $f_v = f_2$, to find the corner of $\pi(v)$ incident to f_u , we can simply use `query($\pi(v)$, u)` from before, which will return a corner of f_u incident to $\pi(v)$. To find the two corners of f_v : With the dual structure (see Observation 13) we may mark the face f_v , and expose the vertices u, v . Now, $\pi(v)$ has a unique place in the face-list of some cluster — if and only if that place is in the root cluster, and $c_{\min} = 0$ for both segments of that cluster, f_v plays the role of f_2 . That is, iff $\pi(v)$ has a corner incident to f_v to one side, and a corner incident to f_v to the other side. In affirmative case, $\pi(v)$ appears with at least one corner to either vertex list; those corners can now be used as cutting-corners for the articulation-flip.

If instead f_v played the role of f_1 , a similar procedure is done with $\pi(u)$.

► **Theorem 26.** *Given u, v are not already linkable, we can determine whether u, v are linkable via an articulation-flip in time $O(\log^2 n)$.*

3.2.2 Finding a separation-flip

Assume v, u are not linkable via an articulation-flip, determine if they are linkable via a separation-flip.

► **Lemma 27.** *Let (x, y) be any edge inducing a cycle C in $T \cup \{(x, y)\}$ that separates f_u from f_v , let $\pi(u) = \text{meet}_T(u, x, y)$ be the projection of u on C . Let e_1, e_2 be the edges incident to $\pi(u)$ on C . Then at least one of e_1, e_2 is in the same flip-component as u w.r.t f_u and f_v .*

Proof. This follows from Observation 25: If $\pi(u)$ is an endpoint of an arc $f_1 \cdots f_2$, then only one of the edges is in the same flip-component. If the projection is not an endpoint, then both of the edges are in the same flip-component. ◀

► **Lemma 28.** *Let C be any induced cycle separating f_u from f_v , let e_u be an edge on C in the same flip-component as u , let f_1 be the face adjacent to e_u that is separated from f_u by C , and let $f_2 \in \{v_L, v_R\}$ be a face on the same side of C as f_1 . Then f_v is the first face on $f_1 \cdots f_2$ that contains v .*

Proof. $C \cup E[f_v]$ separates f_1 and f_2 , so by Observation 16 f_v is on the $f_1 \cdots f_2$ path. It must be the first face on that path that contains v because for any face f after that, $C \cup V[f]$ does not separate f_1 and f_2 , since it can only touch the part of C between $u' = \text{meet}(u, x, y)$ and $v' = \text{meet}(v, x, y)$ where (x, y) is the edge inducing C . ◀

3.3 Finding the separation pair and corners

Assume u, v are not linkable and not linkable via an articulation-flip.

► **Lemma 29.** *Given u, v, f_u , and f_v , let $(x, y)^*$ be any edge on $f_u \cdots f_v$ inducing a separating cycle C . If $\pi(u) = \pi(v) = \alpha$, then α is one of the separation points if it is adjacent to both f_u and f_v , and otherwise no separation pair for u, v exists. The other separation point, β , is then the first vertex $\neq \alpha$ adjacent to both f_u and f_v on either $\alpha \cdots x$ or $\alpha \cdots y$. If instead $\pi(u) \neq \pi(v)$, then α, β are amongst the first two vertices adjacent to both f_u and f_v either on $\pi(u) \cdots x$ and $v \cdots x$, or on $u \cdots y$ and $v \cdots y$.*

Proof. If the projection of u equals the projection of v , but u and v are in different flip-components, then the next point incident to both f_u and f_v along the cycle to either side will be the one we are looking for. However, (x, y) may be internal in the flip component containing u or that containing v , and thus one of the searches may return the empty list. But then the other will return the desired pair of vertices.

If the projections are different, and do not themselves form the desired pair (α, β) , then we may assume without loss of generality that $\pi(v)$ does not belong to the flip-component containing u . Let X_v, X_u denote the flip-components containing u and v , respectively. If (x, y) is in X_v , such that no edge on $\pi(v) \cdots x$ is incident to both f_u and f_v , then the first vertex on $\pi(v) \cdots y$ incident to f_u and f_v is α . Recall (Observation 25) that $\pi(X_u)$ is an arc $\pi_1 \cdots \pi_2 \subset C$, and suppose without loss of generality π_1 is on the path $u \cdots v$. If $\pi(u) = \pi_1$, β is the second vertex on the path u to y incident to both f_u and f_v , as $\pi(u)$ itself is the first. Otherwise, the first such vertex on the path is β . If, on the other hand, (x, y) did not belong to X_v , let x be the vertex of x, y with the property that the path $u \cdots x$ goes through $\pi(u)$. Then the first vertices on the paths to x which are incident to f_u and f_v both, will be the desired separation pair. ◀

► **Lemma 30.** *In the scenario above, we may find the first two vertices on the path incident to both faces in time $O(\log^2 n)$.*

Proof. We use the dual structure (see Observation 13) to search for vertices incident to f_u and f_v . Now since the path $\pi(u) \cdots x$ is a sub-path of the cycle C induced by (x, y) which separates f_u from f_v , all corners incident to f_v will be on one side, and all corners incident to f_u will be on the other side of the path, or at the endpoints. Thus, we expose f_u and f_v in the dual structure, which takes time $O(\log^2 n)$. Now expose $\pi(u), x$ in the primal tree. Since this path is part of the separating cycle, if $c_{\min} = 0$ for both segments, then the maintained vertex-list will contain exactly those vertices incident to both faces, and a corner list for each of them. We now deal separately with the endpoints exactly as with linkable, by exposing the endpoint faces one by one in the dual structure, and noting whether $c_{\min} = 0$ and in that case, the corner list, for each endpoint. ◀

We conclude with the following theorem.

► **Theorem 31.** *We can maintain an embedding of a dynamic graph under insert, remove, split, join, and flip, together with queries that*

1. *Answer whether an edge can be inserted between given endpoints with no other changes to embedding, and if so, where.*
2. *Answer whether there exists a flip that would change the answer for query 1 from “no” to “yes”, and if so, what flip.*

The worst case time per operation is $O(\log^2 n)$.

Acknowledgments We would like to thank Christian Wulff-Nilsen and Mikkel Thorup for many helpful and interesting discussions and ideas.

References

- 1 Stephen Alstrup, Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Transactions on Algorithms*, 1(2):243–264, 2005.
- 2 Giuseppe Di Battista and Roberto Tamassia. Incremental planarity testing. In *FoCS, 1989*, pages 436–441. IEEE, 1989.
- 3 Giuseppe Di Battista and Roberto Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25:956–997, 1996.
- 4 David Eppstein. Dynamic generators of topologically embedded graphs. *SODA '03*, pages 599–608, 2003.
- 5 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification: I. planarity testing and minimum spanning trees. *J. CSS*, 52(1):3 – 27, 1996.
- 6 David Eppstein, Giuseppe F. Italiano, Roberto Tamassia, Robert E. Tarjan, Jeffery R. Westbrook, and Moti Yung. Maintenance of a minimum spanning forest in a dynamic planar graph. *J. Algorithms*, 13(1):33–54, March 1992. Special issue for 1st SODA.
- 7 Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985.
- 8 Zvi Galil, Giuseppe F. Italiano, and Neil Sarnak. Fully dynamic planarity testing with applications. *J. ACM*, 46:28–91, 1999.
- 9 John Hopcroft and Robert E. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, October 1974.
- 10 Giuseppe F. Italiano, Johannes A. La Poutré, and Monika H. Rauch. Fully dynamic planarity testing in planar embedded graphs (extended abstract). *ESA '93, Proceedings*, pages 212–223, 1993.
- 11 David R. Karger. Random sampling in cut, flow, and network design problems. *Mathematics of Operations Research*, pages 648–657, 1994.
- 12 Johannes A. La Poutré. Alpha-algorithms for incremental planarity testing (preliminary version). In *STOC '94*, pages 706–715. ACM, 1994.
- 13 Mihai Pătraşcu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM Journal on Computing*, 35(4):932–963, 2006. See also STOC'04, SODA'04.
- 14 Mihai Pătraşcu and Mikkel Thorup. Planning for fast connectivity updates. In *FOCS '07*, pages 263–271, 2007.
- 15 Jeffery Westbrook. Fast incremental planarity testing. In W. Kuich, editor, *ALP*, volume 623, pages 342–353. Springer Berlin Heidelberg, 1992.

On the Information Carried by Programs about the Objects They Compute*

Mathieu Hoyrup¹ and Cristóbal Rojas²

- 1 Inria Nancy Grand Est
615 rue du jardin botanique, 54600 Villers-lès-Nancy, France
hoyrup@inria.fr
- 2 Departamento de Matemáticas, Universidad Andres Bello
República 220, Santiago, Chile.
crojas@mat-unab.cl

Abstract

In computability theory and computable analysis, finite programs can compute infinite objects. Presenting a computable object via any program for it, provides at least as much information as presenting the object itself, written on an infinite tape. What additional information do programs provide? We characterize this additional information to be any upper bound on the Kolmogorov complexity of the object. Hence we identify the exact relationship between Markov-computability and Type-2-computability. We then use this relationship to obtain several results characterizing the computational and topological structure of Markov-semidecidable sets.

1998 ACM Subject Classification F.1.1 Models of Computation/Computability theory

Keywords and phrases Markov-computable, representation, Kolmogorov complexity, Ershov topology

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.447

1 Introduction

We assume that the reader is familiar with Turing machines and basic computability theory over the natural numbers. To define computability over infinite objects, one still uses Turing Machines but has to set up a way for them to access such objects. In any case, the input of the machine is a finite or infinite sequence of symbols written on the input tape and one has to choose a suitable way to describe infinite objects by such symbolic sequences. We now briefly describe the two main approaches that have been developed.

The first one was introduced and studied by Turing [20], Grzegorzczuk [5], Lacombe [9] and later Kreitz and Weihrauch [21] and is nowadays known as Type-2-computability. In this model, the description itself is completely written on the input tape of the machine. At any time, the machine can read a finite portion of this description. We will call this the *Type-2-model*. The second approach, promoted by the Russian school led by Markov [10, 8], gives an alternative. In this model one restricts the action of the machine to operate on computable (infinite) objects only, in the sense that they have computable descriptions. Instead of having access to the description itself as in the Type-2-model, the machine here has access to a *program* computing a description. We will call this the *Markov-model*.

* This work was partially supported by Inria program “Chercheur invité”. CR was partially supported by FONDECYT project 11110226 and BASAL PFB-03 CMM, Universidad de Chile.



■ **Table 1** Some celebrated results comparing Markov-computability to Type-2-computability.

Objects	Decidability	Semidecidability
Partial computable functions	Markov \equiv Type-2 <i>Rice</i>	Markov \equiv Type-2 <i>Rice-Shapiro</i>
Total computable functions	Markov \equiv Type-2 <i>Kreisel et al/Ceitin</i>	Markov $>$ Type-2 <i>Friedberg</i>

These two approaches provide a priori different computability notions, and their comparison has been an important subject of study [13, 11, 18, 7, 1, 4, 12, 6, 19].

It is clear that the Markov-model is at least as powerful as the Type-2-model, so the question is: does it allow to compute strictly more than the Type-2-model? The answer depends on the objects that we consider, and the algorithmic tasks we want to perform on them. The computational power of these models can therefore be classified according to these parameters. Table 1 summarizes the most celebrated results in this direction. The computable objects considered are the partial computable functions and the total computable functions. The algorithmic tasks considered are decidability and semidecidability of properties about these objects.

Kreisel-Lacombe-Shoenfield/Ceitin's Theorem [7, 1] for instance, states that over total computable functions, Markov-decidability is equivalent to Type-2-decidability¹. This means that the machine trying to decide a property, when provided with a program p for a function f , cannot do better than just running p to evaluate f . The machine gains no additional information about f from p . We note that Ceitin's version of this result shows that over the real line, Markov-computable functions and Type-2-computable functions coincide.

On the other hand, Friedberg [4] exhibited properties about total computable functions that are Markov-semidecidable but not Type-2-semidecidable. So that for semidecidability, a program p for a function f *does* give some additional information that can be exploited by the machine. The main question we raise in this paper is the following:

Can we characterize the additional useful information contained in a program computing an object, as compared to having the object itself ?

To get some intuition, consider the following fundamental difference between the two models. In the Type-2-model, at any given time only a finite portion of the description of x is provided, which corresponds to a finite approximation of x . Clearly, this approximation is also good for infinitely many other objects – all the ones that are “close enough” to x . In particular, x is never completely specified. In the Markov-model on the other hand, the program provided to the machine completely specifies x from the beginning of the calculation! This increases the predictive power of M , which might therefore be able to perform stronger calculations. The point is to understand in which situations this fact can be exploited. A trivial example is obtained when one considers the relativized setting: every function is Markov-computable relative to an appropriate (powerful enough) oracle. Whereas whatever oracle A we consider, Type-2-computable functions relative to A must always be continuous.

This observation takes us to another interesting point that separates the Markov-model from the Type-2-model, namely their topological structure. It is well known that Type-2-computability and topology are closely related: e.g. the Type-2-computable functions are

¹ In its original form, this theorem is stated for functionals.

exactly the effectively continuous ones, and the Type-2-semidecidable properties exactly correspond to the effectively open sets. The connection between Markov-computability and topology, on the other hand, appears to be much less clear. In particular, Friedberg's construction provides a Markov-semidecidable set which is not open (for the standard topology restricted to computable elements).

An obvious solution to relate Markov-computability to topology is to consider precisely the topology generated by all the Markov-semidecidable sets – the so called Ershov's topology. The question then becomes:

How do Markov-semidecidable sets look like? can we characterize Ershov's open sets ?

In the present paper we make use of Kolmogorov Complexity to provide a fairly complete answer to these and other questions in different settings. Our main result is a characterization of the additional information provided by a program, when the class of objects considered are the computable points of an effective topological space. It can be informally stated as follows (see Section 3):

► **Theorem A.** *Over effective topological spaces, a program computing x provides as much information as a description of x itself together with **any upper bound on the Kolmogorov complexity of x .***

Here, the Kolmogorov complexity $K(x)$ of a computable infinite object x is to be understood as the size of the shortest program computing a description of x (Kolmogorov complexity of infinite objects was first defined by Schnorr [15]). Obviously, any program for x trivially provides, in addition to a description, an upper bound on its Kolmogorov complexity. Theorem A says that this bound is all the exploitable additional information it provides.

Thus, we have a third model to deal with computable infinite objects. In this model, input x is presented to the machine as a pair (d, k) , where d is a description for x and k a bound on the Kolmogorov complexity of x . We shall call this the **K-model**. In these terms, a particular case of Theorem A can be stated as follows: if \mathcal{X}, \mathcal{Y} are *effective topological spaces* (not necessarily metric) and X_c, Y_c are the corresponding sets of computable points, then a function $f : X_c \rightarrow Y_c$ is Markov-computable if and only if it is K-computable.

A simple observation shows that one can not in general compute a program for x from a K-description of x , meaning that the two notions are not fully equivalent. Despite this fact, Theorem A is valid in great generality: it holds for decidability, semidecidability and also higher in the hierarchy. In proving this we make a fundamental use of the Recursion Theorem. Interestingly, although the Recursion Theorem does not relativize (a well known fact), Theorem A does in many cases.

The K-model also sheds light into the structure of the open sets of Ershov's topology, providing a nice characterization in terms of Kolmogorov complexity, at least in the particular case of the extended natural numbers $\overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$.

► **Theorem B.** *On the extended natural numbers, the Ershov topology is generated by the sets $\{n\}$ and $\{x \in \overline{\mathbb{N}} : K(x) < h(x)\}$ for some computable order h .*

With the same techniques, we are able to prove several other related results that are interesting on their own. For example, we show that there is no effective enumeration of the Markov-semidecidable sets of $\overline{\mathbb{N}}$ and that there is a Markov-semidecidable subset of $\{0, 1\}^{\overline{\mathbb{N}}}$ that is not Σ_2^0 .

Finally, in the search of the limitations of our techniques, we turn our attention to more general spaces and analyze functions with values on topological spaces that have an

■ **Table 2** Some results comparing Markov-computability, K-computability, and Type-2-computability. $\mathbb{S} = \{\perp, \top\}$ is the Sierpiński space whose topology is generated by $\{\top\}$.

Space \mathcal{X}	Semidecidable	\emptyset' -Semidecidable	$F: \mathcal{X} \rightarrow \mathcal{O}(\mathbb{B})$
\mathbb{S}	Markov \equiv K \equiv Type-2	Markov $>$ K \equiv Type-2	Markov $>$ K \equiv Type-2
Partial functions	Markov \equiv K \equiv Type-2	Markov $>$ K \equiv Type-2	Markov $>$ K $>$ Type-2
Total functions	Markov \equiv K $>$ Type-2	Markov \equiv K $>$ Type-2	Markov ? K $>$ Type-2

admissible representation but are not countably-based. In particular, when this is the space of open subsets of Baire space $\mathcal{O}(\mathbb{B})$, we show that Markov-computability can be strictly stronger than K-computability:

► **Theorem C.** *For functions from the partial computable functions with values on $\mathcal{O}(\mathbb{B})$ one has that:*

$$\text{Markov-computability} > \text{K-computability} > \text{Type-2-computability}.$$

One of the main question that remains open is whether the first strict inequality in Theorem C holds if we replace the *partial* computable functions by the *total* ones. The situation is summarized in Table 2.

The paper is organized as follows. We start by providing the basic notions and definitions in Section 2. In Section 3 we introduce the K-model and present our main results. Section 4 contains several results that shed light on the structure of Markov-semidecidable sets and in Section 5 we present the announced negative results. Finally, Section 6 contains a list of related problems for possible future work.

2 Background

2.1 Notations and basic definitions.

We assume the reader is familiar with computability theory. Let $\{\varphi_e\}_{e \in \mathbb{N}}$ be an effective enumeration of the set of computable partial functions. We denote by $P_c(\mathbb{N})$ the collection of c.e. subsets of \mathbb{N} and $W_e = \text{dom}(\varphi_e)$ the induced effective enumerations of its elements. If $A \in P_c(\mathbb{N})$, an *index* of A is a number e such that $W_e = A$. If A is a c.e. set, implicitly given by an index, $A[s]$ is the finite subset of A enumerated by stage s , so that $A[s] \subseteq A[s + 1]$ and $A = \bigcup_s A[s]$. We use the notation $A[\text{at } s] = A[s] \setminus A[s - 1]$ if $s \geq 1$ and $A[\text{at } 0] = A[0]$. If F is a finite subset of \mathbb{N} then $[F]$ is the collection of supersets of F . $\mathbb{B} = \mathbb{N}^{\mathbb{N}}$ will denote Baire space.

2.2 Effective topological spaces.

An *effective topological space* is a tuple $(\mathcal{X}, \tau, \mathcal{B})$ where (\mathcal{X}, τ) is a non-empty topological space, $\mathcal{B} = \{\mathcal{B}_i\}_{i \in \mathbb{N}}$ is numbered basis such that there exists a computable function $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ satisfying $\mathcal{B}_i \cap \mathcal{B}_j = \bigcup_{k \in W_{f(i,j)}} \mathcal{B}_k$.

Given an effective topological space \mathcal{X} , the standard representation is defined as a surjective map $\rho : \text{dom}(\rho) \subseteq \mathbb{B} \rightarrow \mathcal{X}$ satisfying $\rho(f) = x$ whenever $\{f(n) : n \in \mathbb{N}\} = \{i : x \in \mathcal{B}_i\}$. We will call any $f \in \rho^{-1}(x)$, a **Type-2-name** of x . An element x is **computable** if it has a computable Type-2-name. We denote by \mathbf{X}_c the set of computable points. The countable set X_c has a canonical numbering ν defined by $\nu(e) = x$ if φ_e is a name of x . We

will call such an e a *Markov-name* of x . To facilitate the reading of the paper, we will use the font A, N, U when working on the space X_c , and the fonts $\mathcal{A}, \mathcal{N}, \mathcal{U}$ when working on \mathcal{X} .

2.2.1 Type-2-computability and Markov-computability

Let $(\mathcal{X}, \tau, \mathcal{B})$ and $(\mathcal{Y}, \tau', \mathcal{B}')$ be effective topological spaces. In what follows R stands for both *Type-2* and *Markov*. A set $A \subseteq X_c$ is *R-semidecidable* if there is a Turing machine M which, when provided with an R -name of x , halts if and only if $x \in A$. A function $f : X_c \rightarrow Y_c$ is *R-computable* if there is a Turing machine M which, when provided with an R -name of x , writes an R -name for $f(x)$ on its one-way output tape. It is not hard to see that a function $f : X_c \rightarrow Y_c$ is R -computable if and only if the sets $f^{-1}(\mathcal{B}'_i)$ are uniformly R -semidecidable.

► **Remark.** It is worth noting that for a function $f : X_c \rightarrow Y_c$, being Markov-computable is equivalent to having a Machine M which, provided with a Markov-name of x , outputs a *Type-2-name* of $f(x)$. Indeed, combining the program for x with the program for M gives a program for $f(x)$. We also note that a function $f : X_c \rightarrow Y_c$ which is Type-2-computable does not necessarily extend to a Type-2-computable function $\bar{f} : \mathcal{X} \rightarrow \mathcal{Y}$.

A numbering η of X_c is *admissible* if it is equivalent to the canonical numbering ν in the sense that there exists partial computable functions f and g such that $\nu = \eta \circ f$ on $\text{dom}(\nu)$ and $\eta = \nu \circ g$ on $\text{dom}(\eta)$. The Markov-computability notions do not depend on the choice of the admissible numbering. We will often use the admissible numbering η of X_c defined by $\eta(e) = x$ whenever $W_e = \{i \in \mathbb{N} : x \in \mathcal{B}_i\}$.

Type-2-computability and topology are closely related. A set $U \subseteq \mathcal{X}$ is an *effective open set* if there exists $e \in \mathbb{N}$ such that $U = \bigcup_{i \in W_e} \mathcal{B}_i$. If $A = U \cap X_c$, we will then say that A is effectively open *in* X_c . The connection is established by the following result (see [21]).

► **Theorem 1.** *A set $A \subseteq X_c$ is Type-2-semidecidable if and only if it is effectively open in X_c . Therefore, a function $f : X_c \rightarrow Y_c$ is Type-2-computable if and only if it is effectively continuous, i.e. the sets $f^{-1}(\mathcal{B}'_i)$ are uniformly effectively open in X_c .*

As mentioned in the introduction, in order to have an analogous result for Markov-computability, we have to use Ershov's topology on X_c , which may be different from the topology of \mathcal{X} restricted to X_c .

► **Example 2.** Let $\mathbb{B} = \mathbb{N}^{\mathbb{N}}$ be the Baire space. For each finite sequence u , let $[u]$ be the set of infinite extensions of u , called a cylinder. We endow \mathbb{B} with the topology generated by the cylinders, which is an effective topology. The standard numbering φ_e of partial computable functions, restricted to the indices of total functions is an admissible numbering of \mathbb{B}_c .

► **Example 3.** Let $\mathcal{P}(\mathbb{N})$ be space of subsets of \mathbb{N} . For each finite set $F \subseteq \mathbb{N}$, let $[F]$ be the set of supersets of F . We endow $\mathcal{P}(\mathbb{N})$ with the Scott topology, generated by the sets $[F]$, which is an effective topology. The standard numbering $W_e = \text{dom}(\varphi_e)$ of c.e. sets is an admissible numbering of $\mathcal{P}(\mathbb{N})_c$.

3 Main results

In this section, $(\mathcal{X}, \tau, \mathcal{B})$ is always an effective topological space and X_c is the subset of computable elements. We start by explaining the main idea behind our results. Let $x \in X_c$ be a fixed element. From a machine Type-2-semideciding a set A containing x , one can

compute a neighborhood \mathcal{N} of x such that for every element $y \in X_c$ the following implication holds:

$$y \in \mathcal{N} \implies y \in A. \quad (1)$$

Now assume that A has the weaker property of being Markov-semidecidable, and still contains x . From a machine Markov-semideciding A one cannot in general compute such a neighborhood, which may not exist as shown by Friedberg's example. However, from the Markov-name of any other element $y \in X_c$ one can still compute a neighborhood \mathcal{N}_y of x such that implication (1) holds. Further, as a finite intersection of neighborhoods is still a neighborhood, one can compute a neighborhood \mathcal{N} satisfying implication (1) for all y in a given finite set. Using this argument we can show that the problem $x \in A$ can be Type-2-semidecided *as soon as we know, in addition, a finite list of programs containing at least one for x* . This additional information is equivalent to having any upper bound on the Kolmogorov complexity of x , which leads us to the notion of K-computability that we now introduce.

3.1 K-computability

► **Definition 4.** The *Kolmogorov complexity* $K(x)$ of a computable element $x \in X_c$ is the length of a shortest program computing a Type-2-name of x .

In this paper, whether we use prefix-free, monotone or plain machines will not make any difference so we do not need to specify the definition any further.

► **Definition 5.** A *K-name* of a computable element $x \in X_c$ consists of a pair (k, f) where $k \geq K(x)$ and f is a Type-2-name of x .

► **Remark.** Note that k is only an upper bound on the Kolmogorov complexity of x and not necessarily of f , which may even be non computable. Note also that knowing any such k is effectively equivalent to knowing any upper bound on a Markov-name of x . This is what we will rather use in our proofs.

The *K-computability* notions are defined in the same way as in the previous section. We will denote by $X_c(k)$ the set of computable elements whose Kolmogorov complexity is at most k . Note that $X_c = \bigcup_k X_c(k)$ and that K-computability is the same as Type-2-computability on $X_c(k)$, uniformly in k . In particular, a set $A \subseteq X_c$ is K-semidecidable iff there exists uniformly effective open sets \mathcal{U}_k such that $A \cap X_c(k) = \mathcal{U}_k \cap X_c(k)$.

Thus, for each notion of computability we have so far three versions, depending on the way the objects are represented.

It is clear that one can compute K-names from Markov-names. An important first observation is the fact that the converse does not necessarily hold. In other words, the representations underlying Markov-computability and K-computability are not equivalent.

► **Proposition 6.** *In general, it is not possible to compute Markov-names from K-names.*

One can show that on Cantor space, Markov-names are limit-computable (can be *learned*) from K-names: given x and $k \geq K(x)$, one can compute a sequence of natural numbers converging to an index of x (this problem was investigated in the context of inductive inference [3]). One can moreover show that relative to the halting set, Markov-names are uniformly computable from K-names. A c.e. set, however, cannot be learned. Actually one can prove a stronger statement.

► **Proposition 7.** *There is no Turing functional Φ that, given an index e and a Type-2-name of a set W which is either \mathbb{N} or W_e , computes a sequence of numbers converging to an index of W .*

The rest of this section is devoted to show that, despite the facts above, the notions of Markov-computability and K-computability are indeed equivalent to a large extent.

3.2 Equivalence between Markov-computability and K-computability

We will use the Recursion Theorem. See [14].

► **Theorem 8** (Recursion Theorem). *For every computable total function f , there exists e such that $\varphi_e = \varphi_{f(e)}$. Moreover, e can be computed from an index of f .*

The following Lemma contains the main technical arguments.

► **Lemma 9.** *Let A be a c.e. subset of \mathbb{N} . There exist uniformly effective Scott open sets $\mathcal{U}_k \subseteq \mathcal{P}(\mathbb{N})$, such that for every c.e. set E the following hold:*

- (i) *if all the indices of E belong to A then $E \in \mathcal{U}_k$ for every k ,*
- (ii) *if no index of E belongs to A then $E \notin \mathcal{U}_k$ for every $k \geq K(E)$.*

The argument is uniform: the open sets \mathcal{U}_k are effective, uniformly in a c.e. index of A .

Proof. Using the Recursion theorem, there is a computable function $e(a, b)$ such that for all $a, b \in \mathbb{N}$,

$$W_{e(a,b)} = \begin{cases} W_a & \text{if } e(a, b) \notin A, \\ W_a[t] \cup W_b & \text{if } e(a, b) \in A[\text{at } t]. \end{cases}$$

Let $k \in \mathbb{N}$. We define an effective open set \mathcal{U}_k . Compute b_k such that every element whose complexity is less than k has an index less than b_k . If a is such that for all $b \leq b_k$, $e(a, b) \in A$ then let t be minimal such that $e(a, b) \in A[t]$ for all $b \leq b_k$, enumerate $[W_a[t]]$ into \mathcal{U}_k .

We now check the two announced conditions. *i)* Let $E \subseteq \mathbb{N}$ be a c.e. set. Assume that every index of E belongs to A and let a be an index of E . For all b , $e(a, b) \in A$ (otherwise $e(a, b)$ is an index of $W_a = E$ but $e(a, b) \notin A$, contradiction), so \mathcal{U}_k contains $[W_a[t]]$ for some t , which contains E . *ii)* Assume that $K(E) \leq k$, that no index of E belongs to A and that $E \in \mathcal{U}_k$. Let $b \leq b_k$ be an index of E . As $E \in \mathcal{U}_k$, E belongs to some $[W_a[t]]$ enumerated into \mathcal{U}_k (here a is not the same as above and is not assumed to be an index of E). As $e(a, b) \in A$, $W_{e(a,b)} = W_a[t'] \cup W_b$ for some $t' \leq t$. As $W_a[t'] \subseteq W_a[t] \subseteq W_b$, $e(a, b)$ is an index of E that belongs to A , contradicting the assumption. ◀

We now state the main explicit versions of Theorems A and B.

► **Theorem 10.** *Let \mathcal{X} be an effective topological space. A set $A \subseteq X_c$ is Markov-semidecidable iff it is K-semidecidable. The equivalence is uniform.*

Proof. Every effective topological space is Type-2-computably homeomorphic to a subspace of $\mathcal{P}(\mathbb{N})$: to $x \in \mathcal{X}$, associate $\{i \in \mathbb{N} : x \in \mathcal{B}_i\}$ where \mathcal{B}_i is enumeration of the basis of \mathcal{X} . Hence we can assume that \mathcal{X} is a subspace of $\mathcal{P}(\mathbb{N})$. Let $I \subseteq \mathbb{N}$ be a c.e. set such that for all $e \in \mathbb{N}$ for which $W_e \in X_c$, it holds $W_e \in A \iff e \in I$. Each c.e. set $E \in X_c$ either has all its indices in I or has no index in I , so the effective open sets \mathcal{U}_k provided by Lemma 9 coincide with A on the set of elements of X_c whose complexity is at most k . Now, a machine K-semideciding A works as follows: given a Type-2-name of $E \in X_c$ and $k \geq K(E)$, it tests whether $E \in \mathcal{U}_k$ and halts in this case only. ◀

► **Corollary 11.** *Let \mathcal{X}, \mathcal{Y} be effective topological spaces. A function $f : X_c \rightarrow Y_c$ is Markov-computable iff f is K-computable. The equivalence is uniform.*

Proof. Let B_i be the numbered basis of \mathcal{Y} . f is Markov-computable iff the sets $f^{-1}(B_i)$ are uniformly Markov-semidecidable iff these sets are K-semidecidable (Theorem 10) iff f is K-computable. ◀

We now show that the argument in the proof of Lemma 9 can be extended from semi-decidability to weaker classes of properties, showing that for most algorithmic tasks, the additional information given by programs is indeed just an upper bound on the Kolmogorov complexity.

Hierarchies. Let \mathcal{X} be an effective topological space. We consider the finite levels of the effective Borel hierarchy, defined as follows. The class Σ_1^0 consists of the effective open sets. The class Σ_{n+1}^0 consists of the effective unions of differences of Σ_n^0 -sets. The classes Π_n^0 consists of complements of Σ_n^0 -sets. The class Δ_n^0 is the intersection of Σ_n^0 and Π_n^0 . Inside the class Δ_2^0 we consider the finite levels of the effective difference hierarchy. For $n \in \mathbb{N}$, the class \mathcal{D}_n consists of the differences of n effective open sets, i.e. the sets $(\mathcal{U}_0 \setminus \mathcal{U}_1) \cup \dots (\mathcal{U}_{n-2} \setminus \mathcal{U}_{n-1})$ if n is even and the sets $(\mathcal{U}_0 \setminus \mathcal{U}_1) \cup \dots \mathcal{U}_{n-1}$ if n is odd. In the case $\mathcal{X} = \mathbb{N}$ with the discrete topology, the effective Borel hierarchy is exactly the arithmetical hierarchy, the class \mathcal{D}_n of effective difference hierarchy is exactly the class of n -c.e. sets.

► **Theorem 12.** *A set $A \subseteq X_c$ is Markov- n -c.e. iff it is K- n -c.e. More precisely, the set of indices of elements of A is n -c.e. on the set of indices of X_c iff there exist uniformly effective open sets $\mathcal{U}_k^1, \dots, \mathcal{U}_k^n$ such that $A \cap X_c(k) = \mathcal{D}_n(\mathcal{U}_k^1, \dots, \mathcal{U}_k^n) \cap X_c(k)$.*

It is known from [17] that there exists a Markov-2-c.e. subset of $\mathcal{P}(\mathbb{N})$ that is not even Π_2^0 . Hence Markov-2-c.e. sets are not the differences of Markov-semidecidable sets.

In the following theorem, we need to assume an additional property on the space \mathcal{X} . Namely, that the domain of the standard representation on \mathcal{X} is a Π_2^0 set. This is the case for example for the so called *quasi-Polish spaces* (see [2]).

► **Theorem 13.** *A set $A \subseteq X_c$ is Markov- Σ_2^0 iff it is K- Σ_2^0 . More precisely, the set of indices of elements of A is Σ_2^0 on the set of indices of X_c iff there exist uniformly effective open sets $\mathcal{U}_k^n, \mathcal{V}_k^n$ such that $A \cap X_c(k) = \bigcup_n (\mathcal{U}_k^n \setminus \mathcal{V}_k^n) \cap X_c(k)$.*

► **Remark.** In case \mathcal{X} is a Polish space, the sets \mathcal{V}_k are not needed and therefore the last part of the statement reads $A \cap X_c(k) = \bigcup_n \mathcal{U}_k^n \cap X_c(k)$.

4 Structure of Markov-semidecidable sets

Here we provide several results that shed light on the computational and topological structure of Markov-semidecidable sets. Our first result shows that Markov-semidecidable sets share some of the nice properties of Type-2-semidecidable sets.

► **Proposition 14.** *Assume that \mathcal{X} contains a dense computable sequence. Given a Markov-semidecidable set A , it is semi-decidable whether A is non-empty. If A is non-empty, one can compute a sequence of points $\{x_i\} \subseteq A$ which is dense in A .*

Proof. Using the Recursion theorem, there is a computable function $e(a)$ such that $x_{e(a)} = x_a$ if $e(a) \notin A$, or $x_{e(a)}$ is some point from the dense sequence in some neighborhood of x_a if $e(a) \in A$ at time t . A is non-empty iff there is a such that $e(a) \in A$. When A is non-empty,

one can compute an element in A : look for a such that $e(a) \in A$, $x_{e(a)}$ is such a point. To get a computable dense sequence, apply this argument in parallel to the intersection of A with each basic open set \mathcal{B}_i . ◀

The following result provides an upper bound on the effective Borel complexity of Markov-semidecidable sets.

► **Proposition 15.** *Let $A \subseteq X_c$ be Markov-semidecidable. There exist uniformly effective open sets $\mathcal{U}_k \subseteq \mathcal{X}$ such that $A = \bigcap_k \mathcal{U}_k \cap X_c$.*

Proof. Let \mathcal{U}_k be the effective open sets from the proof of Theorem 10 and define $\mathcal{S} = \bigcap_k \mathcal{U}_k$. We already know that $A \subseteq \mathcal{U}_k$ for all k . If $x \in X_c \cap \mathcal{S}$ then let $k \geq K(x)$. Since $x \in X_c(k) \cap \mathcal{U}_k = X_c(k) \cap A$, we conclude that $x \in A$. ◀

The result above is actually tight, as the following theorem shows. For a finite string u , let us define the monotone complexity $Km(u)$ of u as the length of a shortest program computing a (finite or infinite) binary sequence extending u . The program writes its output on a one-way output tape and may never halt. Again the precise definition of $Km(u)$ (Levin or Schnorr monotone or process complexity) does not make any difference for our purposes. The only important property is that for a computable sequence x , $Km(x|_n) \leq Km(x)$ for all n . For the sake of completeness, let us recall original Friedberg’s example. We present it in a way that is more convenient for our purposes.

► **Theorem 16 (Friedberg).** *On the Cantor space, the set*

$$\mathcal{A} = \{0^\omega\} \cup \bigcup_{n: Km(0^n 1) < \log(n) - 1} [0^n 1].$$

is Markov-semidecidable but not open. Hence the Ershov topology is strictly stronger than the Cantor topology.

Proof. We show that \mathcal{A} is K-semidecidable. Given an infinite binary sequence x (a Type-2-description) and a bound k on $K(x)$, we only need to read the first $e = 2^{k+2}$ bits of x . If we see only zeros, we accept. Otherwise one gets $0^n 1 \dots$ for some $n < e$, then test whether $Km(0^n 1) < \log(n) - 1$. ◀

► **Remark.** Friedberg’s example happens to be Σ_2^0 . It is an effective open set appended with a limit point. We strengthen Friedberg’s example by constructing a Markov-semidecidable set which is far from being open.

► **Theorem 17.** *There is a Markov-semidecidable subset of $\{0, 1\}_c^\mathbb{N}$ that is not Σ_2^0 . It is a non-empty closed subset of $\{0, 1\}_c^\mathbb{N}$ with empty interior, defined by*

$$A = \{x \in \{0, 1\}_c^\mathbb{N} : \forall n, Km(x|_n) < n/2 + c\} \quad \text{for some sufficiently large } c \in \mathbb{N}.$$

For the following results, we restrict our attention to the space $\mathcal{X} = \overline{\mathbb{N}} = \mathbb{N} \cup \{\infty\}$ whose topology is generated by the singletons $\{n\}$ and the semi-lines $[n, \infty]$, for $n \in \mathbb{N}$. Note that every point in this space is computable, so that $\mathcal{X} = X_c$.

Friedberg’s example translated to this space reads $\{x \in \overline{\mathbb{N}} : K(x) < \log(x) - 1\}$, which inspires the following definition.

► **Definition 18.** We define the **Friedberg sets** of $\overline{\mathbb{N}}$ to be the ones of the form $\{x \in \overline{\mathbb{N}} : K(x) < h(x)\}$, where $h : \mathbb{N} \rightarrow \mathbb{N}$ is any computable order, namely, any non decreasing unbounded computable function.

Note that a computable order can always be extended to a computable function $h : \overline{\mathbb{N}} \rightarrow \overline{\mathbb{N}}$, with $h(\infty) = \infty$.

Friedberg sets are Markov-semidecidable just like the original set. The next two results show that, unlike Cantor space, the only Markov-semidecidable sets over $\overline{\mathbb{N}}$ which are not Type-2-semidecidable are essentially the Friedberg sets.

► **Proposition 19.** *If $A \subseteq \overline{\mathbb{N}}$ is Markov-semidecidable and contains ∞ then there is a computable order h such that A contains a Friedberg set.*

Proof. Since A is K-semidecidable, for each k one can compute $p(k)$ such that $[p(k), \infty] \cap \{x : K(x) \leq k\} \subseteq A$. One can assume that $p(k)$ is increasing. Let $h(n) = \min\{i : p(i) > n\}$. If $n \notin A$ then $p(K(n)) > n$ (just take $k = K(n)$), so one has $h(n) \leq K(n)$. ◀

► **Remark.** Observe that $K(x)$ here coincides with the usual notion of Kolmogorov complexity of natural numbers.

Proposition 19 provides a nice characterization of the Ershov's open sets.

► **Corollary 20.** *The Ershov topology is generated by the singletons $\{n\}$ and the Friedberg sets.*

► **Remark.** Note that the sets $[n, \infty]$ can be expressed as the Friedberg sets $\{x \in \overline{\mathbb{N}} : K(x) < h(x)\}$ where $h(x) = 0$ for $x < n$ and $h(x) = c(x + 1)$ for $x \geq n$, where c is such that $K(n) \leq c(n + 1)$ for all $n \in \mathbb{N}$.

Whether or not one can find such a characterization on other spaces such as the Cantor space is an interesting question.

We end this section by observing that, unlike Type-2-semidecidable sets, Markov-semidecidable sets cannot be effectively enumerated.

► **Proposition 21.** *There is no effective enumeration of the Markov-semidecidable subsets of $\overline{\mathbb{N}}$.*

5 When Markov beats Kolmogorov

In this section we explore the limits of our methods. We first look at the relativized case, and show that there are simple cases that separate Markov-computability from K-computability. However, we also show that, interestingly, the equivalence persists if the space has a Polish structure.

5.1 Relativization

Let $\mathbb{S} = \{\perp, \top\}$ be the Sierpiński space with topology given by $\{\emptyset, \{\top\}, \{\perp, \top\}\}$. Note that as \mathbb{S} is finite, K-computability is trivially equivalent to Type-2-computability simply because all the elements share a common upper bound on their Kolmogorov complexities, which therefore provides no interesting information. Relativizing w.r.t. the Halting set, we can then separate Markov-decidability from Type-2-decidability, and therefore from K-decidability.

► **Remark 22.** *The set $\{\perp\} \subseteq \mathbb{S}$ is Markov-decidable relative to the Halting set but is not Type-2-decidable relative to any oracle.*

Proof. It is not decidable relative to any oracle simply because it is not clopen. ◀

Similarly, over $\mathcal{P}(\mathbb{N})$, \emptyset'' separates K-semidecidability from Type-2-semidecidability (without oracle, the two notions coincide with Markov-semidecidability by Rice-Shapiro theorem).

► **Proposition 23.** *The set $\{\mathbb{N}\} \subseteq \mathcal{P}(\mathbb{N})$ is K-semidecidable relative to \emptyset'' but is not Type-2-semidecidable relative to any oracle.*

However, metric spaces behave differently. Although stated on Cantor space, the next result extends to any computable metric space [21].

► **Proposition 24.** *Let $O \subseteq \mathbb{N}$. A subset of $\{0, 1\}_c^{\mathbb{N}}$ is Markov-semidecidable relative to O if and only if it is K-semidecidable relative to O .*

Proof. There are two cases, depending on whether O computes the halting set or not.

If O computes the halting set then by the remark following Proposition 6, Markov-names can be uniformly computed from K-names relative to O , which gives the result. This part only works in the case of the Cantor space.

If O does not compute the halting set then we show that Lemma 9 and Theorem 10 still hold relative to O on any effective topological space. We show an alternative proof of Lemma 9 avoiding the Recursion theorem. There is a computable function $e(a, b, c)$ such that

$$W_{e(a,b,c)} = \begin{cases} W_a & \text{if } \varphi_c(c) \text{ does not halt,} \\ W_a[t] \cup W_b & \text{if } \varphi_c(c) \text{ halts in time } t. \end{cases}$$

Let $A \subseteq \mathbb{N}$ be c.e. relative to O . Given $k \in \mathbb{N}$ we define an effective open set \mathcal{U}_k . Compute b_k such that every element whose complexity is less than k has an index less than b_k . If a is such that for all $b \leq b_k$, there exists $c = c(a, b)$ such that $e(a, b, c) \in A$ and $\varphi_c(c)$ halts then let t be the minimal halting time of $\varphi_c(c)$ for $c = c(a, b)$ with $b \leq b_k$.

By the same argument as in the proof of Lemma 9, if no index of E belongs to A then $E \notin \mathcal{U}_k$ for all $k \geq K(E)$. If all the indices of $E \subseteq \mathbb{N}$ belong to A but $E \notin \mathcal{U}_k$ for some k then let a be an index of E . For each b, c , if $\varphi_c(c)$ does not halt then $W_{e(a,b,c)} = W_a = E$ so $e(a, b, c) \in A$. As $E \notin \mathcal{U}_k$, there is $b \leq b_k$ such that for all c , if $\varphi_c(c)$ halts then $e(a, b, c) \notin A$. As a result, for all c , $\varphi_c(c)$ halts iff $e(a, b, c) \notin A$, so the complement of the halting set is many-one reducible to A so it is Turing reducible to O , a contradiction. ◀

5.2 Functions to non-effective topological spaces

In this section we provide results that strictly separate our three notions of computability: Markov-computability, K-computability and Type-2-computability. The idea of the constructions is to build uniform versions of the examples given in Remark 22 and Proposition 23. For this, one can imagine a function with two arguments, where the second argument $f \in \mathbb{B}$ is always provided to the machine by a Type-2-name and plays the role of the oracle. The only difficulty is then to make the computation work in a uniform way in the oracle parameter. In order to get a well defined function w.r.t. our models, we express it as a function of the first argument only, but with values on $\mathcal{O}(\mathbb{B})$, which is the set of open subsets of the Baire space endowed with the topology generated by the following sets: given a compact set $K \subseteq \mathbb{B}$, the class of open subsets of \mathbb{B} containing K is open. This topology is not countably-based, and hence it is not an effective topology. However it does have an admissible representation [16].

We now present the details of the simplest case, a uniform version of Remark 22. This result contrasts with Corollary 11.

► **Theorem 25.** *There exists a Markov-computable function $F : \mathbb{S} \rightarrow \mathcal{O}(\mathbb{B})$ that is not K-computable.*

Proof. We use the admissible numbering $\nu_{\mathbb{S}}$ of \mathbb{S} defined by $\nu_{\mathbb{S}}(e) = \top$ if $\varphi_e(e) \downarrow$, $\nu_{\mathbb{S}}(e) = \perp$ otherwise. We define two effective open sets U_{\perp}, U_{\top} and define $F(\perp) = U_{\perp}$ and $F(\top) = U_{\top}$. First, let $U_{\perp} = \mathbb{B}$. Let $T : \mathbb{N} \rightarrow \mathbb{N}$ be defined as follows: $T(n)$ is the halting time of $\varphi_n(n)$ if it halts, $T(n) = 0$ otherwise. The open set $U_{\top} := \mathbb{B} \setminus \{T\}$ happens to be effective. First the function F is not Lacombe computable because it is not continuous: indeed, F is not monotonic as $\perp \leq \top$ but $U_{\perp} = \mathbb{B}$ is not contained in $U_{\top} \subsetneq \mathbb{B}$. As \mathbb{S} is finite, F is not K-computable neither. However F is Markov-computable. Given an index e of $s \in \mathbb{S}$, enumerate U_{\top} and enumerate the set of functions f such that $\varphi_e(e)$ does not halt in exactly $f(e)$ steps. The latter set of functions is effectively open, uniformly in e . If $\varphi_e(e) \uparrow$ then the whole space \mathbb{B} is enumerated. If $\varphi_e(e) \downarrow$ then nothing more than U_{\top} is enumerated. Intuitively, given e and f , from T one can decide whether $\varphi_e(e)$ halts, i.e. whether $\nu_{\mathbb{S}}(e) = \perp$. ◀

A similar construction, based on Proposition 23, yields a function $F : \mathcal{P}(\mathbb{N}) \rightarrow \mathcal{O}(\mathbb{B})$ which is K-computable but not Type-2-computable by replacing the function T from Theorem 25 by a function T' computing \emptyset'' and such that $\mathbb{B} \setminus \{T'\}$ is effectively open.

Combining all these results, and using that fact that Theorem 25 can clearly also be realized using $\mathcal{P}(\mathbb{N})$ in place of \mathbb{S} , we obtain our announced Theorem C.

► **Theorem 26.** *For functions from $\mathcal{P}(\mathbb{N})$ with values on $\mathcal{O}(\mathbb{B})$ one has that:*

Markov-computability $>$ K-computability $>$ Type-2-computability.

While Type-2-computable functions are always Scott continuous (i.e. monotone and compact), one can show that K-computable functions are always monotone but not necessarily compact. Markov-computable functions may even not be monotone.

Let us now briefly discuss whether Theorem 26 holds for functions from the Cantor space to $\mathcal{O}(\mathbb{B})$. Friedberg's example of a Markov (hence K)-semidecidable set that is not Type-2-semidecidable directly implies the second inequality. However the idea behind the proof of the first inequality cannot be applied on Cantor space. Indeed, using Proposition 24 one can show that the analog of the function of Theorem 25 is actually K-computable.

► **Proposition 27.** *The function $G : \{0, 1\}^{\mathbb{N}} \rightarrow \mathcal{O}(\mathbb{B})$ mapping 0^{ω} to \mathbb{B} and any other sequence to $\mathbb{B} \setminus \{T\}$ is K-computable.*

Proof. Given x, k and f , apply the algorithm given by Proposition 24 to semi-decide, if $f = T$, whether $x = 0^{\omega}$. In parallel, semidecide whether $f \neq T$. ◀

We leave the following question open: is there a Markov-computable function from the Cantor space to $\mathcal{O}(\mathbb{B})$ that is not K-computable?

6 Future work

We list a few problems for future work.

- Find a characterization of the Ershov topology on other spaces than $\overline{\mathbb{N}}$, like the Cantor space.
- Determine for which levels of the effective difference hierarchy the Markov-model and the K-model are equivalent. We know from Theorem 12 that the equivalence holds for the *finite* levels. What about the level ω ?

- All our results hold when the space \mathcal{X} is an effective topological space. However the three models also make sense on any represented space. It seems like an interesting research program to study the extent to which our results are valid in this case.
- Compare the effective Borel hierarchy induced by the Markov-semidecidable sets, the hierarchy induced by the arithmetical hierarchy on the indices and the effective Borel hierarchy induced by the standard topology.

References

- 1 G. S. Ceitin. Algorithmic operators in constructive metric spaces. *Trudy Matematiki Instituta Steklov*, 67:295–361, 1962. English translation: *American Mathematical Society Translations*, series 2, 64:1-80, 1967.
- 2 Matthew de Brecht. Quasi-polish spaces. *Ann. Pure Appl. Logic*, 164(3):356–381, 2013.
- 3 Rusins Freivalds and Rolf Wiehagen. Inductive inference with additional information. *Journal of Information Processing and Cybernetics*, 15:179–185, 1979.
- 4 Richard M. Friedberg. Un contre-exemple relatif aux fonctionnelles récursives. *Comptes Rendus de l'Académie des Sciences*, 247:852–854, 1958.
- 5 Andrzej Grzegorzcyk. On the definitions of computable real continuous functions. *Fundamenta Mathematicae*, 44:61–71, 1957.
- 6 Peter Hertling. Computable real functions: Type 1 computability versus Type 2 computability. In *CCA*, 1996.
- 7 G. Kreisel, D. Lacombe, and J.R. Schoenfield. Fonctionnelles récursivement définissables et fonctionnelles récursives. *Comptes Rendus de l'Académie des Sciences*, 245:399–402, 1957.
- 8 Boris A. Kushner. The constructive mathematics of A. A. Markov. *Amer. Math. Monthly*, 113(6):559–566, 2006.
- 9 Daniel Lacombe. Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables réelles I-III. *Comptes Rendus Académie des Sciences Paris*, 240,241:2478–2480,13–14,151–153, 1955.
- 10 A. A. Markov. On the continuity of constructive functions (russian). *Uspekhi Mat. Nauk*, 9:226–230, 1954.
- 11 J. Myhill and J. C. Shepherdson. Effective operations on partial recursive functions. *Mathematical Logic Quarterly*, 1(4):310–317, 1955.
- 12 Marian B. Pour-El. A comparison of five “computable” operators. *Mathematical Logic Quarterly*, 6(15-22):325–340, 1960.
- 13 H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 74(2):pp. 358–366, 1953.
- 14 Hartley Jr. Rogers. *Theory of Recursive Functions and Effective Computability*. MIT Press, Cambridge, MA, USA, 1987.
- 15 C.P. Schnorr. Optimal enumerations and optimal gödel numberings. *Mathematical systems theory*, 8(2):182–191, 1974.
- 16 Matthias Schröder. Extended admissibility. *Theoretical Computer Science*, 284(2):519–538, 2002.
- 17 Victor L. Selivanov. Index sets in the hyperarithmetical hierarchy. *Siberian Mathematical Journal*, 25:474–488, 1984.
- 18 N. Shapiro. Degrees of computability. *Transactions of the American Mathematical Society*, 82:281–299, 1956.
- 19 Dieter Spreen. Representations versus numberings: on the relationship of two computability notions. *Theoretical Computer Science*, 262(1):473–499, 2001.
- 20 Alan Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2, 42:230–265, 1936.
- 21 Klaus Weihrauch. *Computable Analysis*. Springer, Berlin, 2000.

Communication Complexity of Approximate Matching in Distributed Graphs

Zengfeng Huang¹, Božidar Radunović², Milan Vojnović³, and Qin Zhang⁴

- 1 MADALGO, Aarhus University, Denmark
huangzf@cse.ust.hk
- 2,3 Microsoft Research, Cambridge, UK
{bozidar,milanv}@microsoft.com
- 4 Indiana University Bloomington, USA
qzhangcs@indiana.edu

Abstract

In this paper we consider the communication complexity of approximation algorithms for maximum matching in a graph in the message-passing model of distributed computation. The input graph consists of n vertices and edges partitioned over a set of k sites. The output is an α -approximate maximum matching in the input graph which has to be reported by one of the sites. We show a lower bound on the communication complexity of $\Omega(\alpha^2 kn)$ and show that it is tight up to poly-logarithmic factors. This lower bound also applies to other combinatorial problems on graphs in the message-passing computation model, including max-flow and graph sparsification.

1998 ACM Subject Classification F.2.3 Tradeoffs between Complexity Measures

Keywords and phrases approximate maximum matching, distributed computation, communication complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.460

1 Introduction

Massive data volumes require scaling out computations using distributed clusters of machines which are nowadays commonly deployed in data centres. The data is typically stored distributively across different machines (we refer to as sites) which are interconnected with a communication network. It is desired to process such distributed data with a limited communication among sites which avoids the communication network becoming a bottleneck. A particular interest has been devoted to data in the form of a graph that arises in many applications including online services, online social networks, biological and other networks. There has been a surge of interest in distributed iterative computations using graph input data and resolving queries in distributed graph databases. In practice, the size of a graph can be as large as in the order of a billion of vertices and a trillion of edges, e.g. semantic web knowledge graphs and online social networks [12]. An important research direction is to design efficient algorithms for processing of large-scale graphs in distributed systems which has been one of the focuses of the theoretical computer science community, e.g. [25, 23, 5, 4].

In this paper we consider the problem of approximate computation of a maximum matching in a graph that is stored edge-partitioned across different sites. There are several performance measures of interest in computations over distributed data including the communication complexity in terms of the number of bits or messages, the time complexity in terms of the number of time units or rounds, and the storage complexity in terms of the number of bits.



© Zengfeng Huang, Božidar Radunović,
Milan Vojnović, and Qin Zhang;
licensed under Creative Commons License CC-BY

32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).

Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 460–473



Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



In this paper we focus on the performance measure of the communication complexity in the number of bits required to approximately compute a maximum matching in a graph. Our main result is a tight lower bound on the communication complexity for computing an approximate maximum matching in a graph.

We consider the distributed computation model known in the literature as the *message-passing* model, see, e.g., [27, 10]. A message-passing model consists of k sites, P^1, \dots, P^k . Each site P^i holds a piece of input data x^i and the sites want to jointly compute a given function $f(x^1, \dots, x^k)$. The sites are allowed to have point-to-point communications between each other. At the end of the computation, at least one site should return the answer. Our goal is to minimize the total communication cost between the sites. For technical convenience, we introduce another special party called the *coordinator*. The coordinator does not have any input. We require that all sites can only talk with the coordinator, and at the end of the computation, the coordinator should output the answer. We call this model *the coordinator model*. Note that we have essentially replaced the clique communication topology with the star topology, which increases the total communication cost only by a factor of 2, which does not affect the order of the asymptotic communication complexity.

1.1 Our Results and Techniques

We study the approximate maximum matching problem in the message-passing model which we refer to as Distributed Matching Reporting (DMR). Given a set of $k > 1$ sites and an input graph $G = (V, E)$ with $|V| = n$ vertices and the set of edges $E = E^1 \cup E^2 \cup \dots \cup E^k$ such that the set of edges E^i is assigned to site P^i , at the end of the computation, the coordinator is required to report an α -approximation of the maximum matching in graph G . In this paper show the following main theorem.

► **Theorem 1.** *Any approximation algorithm for computing an α -approximation for DMR in the message-passing model with error probability $1/4$ has the communication complexity of $\Omega(\alpha^2 kn)$ bits, under assumption that $k \leq n$. This communication complexity holds for bipartite graphs.*

It is noteworthy that a simple greedy algorithm solves DMR for $\alpha = 1/2$ with the communication cost of $O(kn \log n)$ bits. This greedy algorithm is based on computing a maximal matching by using a straightforward sequential procedure which we define as follows. Let $G(E')$ be the graph induced by a subset of edges $E' \subseteq E$. Site P^1 computes a maximal matching M^1 in $G(E^1)$, and sends it to P^2 via the coordinator. Site P^2 then computes a maximal matching M^2 in $G(E^1 \cap E^2)$ by greedily adding edges in E^2 to M^1 , and then sends M^2 to site P^3 . This procedure continues until site is reached P^k , which after computing M^k sends it to the coordinator. The matching M_k is a maximal matching in the graph G , hence it is a $1/2$ -approximation of a maximum matching in G . The communication cost of this protocol is $O(kn \log n)$ bits because the size of each M^i is at most n . This shows that our lower bound is tight up to a $\log n$ factor. In Section 3.4, we show that our lower bound is also tight with respect to the approximation factor α for any $\alpha \leq 1/2$ up to a $\log n$ factor. It was showed by Woodruff and Zhang [30] that many statistical estimation problems and combinatorial graph problems require $\Omega(kn)$ bits of communication to obtain an *exact* solution. Our lower bound shows that for DMR even computing a constant approximation requires this amount of communication.

Our lower bound is also of wider applicability to other combinatorial problems on graphs. Since a bipartite maximum matching problem can be found by solving a *max-flow* problem, our lower bound also holds for approximate computation of a max-flow problem. Our lower

bound also implies a lower bound for *graph sparsification* problem, see the definition of graph sparsification, e.g., in [5]. This is because in our lower bound construction (see Section 3), the bipartite graph under consideration contains many cuts of size 1 which have to be included in a sparsifier. By our construction these edges form a good approximate maximum matching. In Ahn, Guha, and McGregor [5], it is shown that there is a sketch-based $O(1)$ -approximate graph sparsification algorithm with the sketch size of $\tilde{O}(n)$, which directly translates to an approximation algorithm of $\tilde{O}(kn)$ communication in our model. Thus, our lower bound is tight up to a polylogarithmic factor.

We briefly discuss the main ideas and techniques of our proof of the lower bound for DMR. As a hard instance, we use a bipartite graph $G = (U, V, E)$ with $|U| = |V| = n/2$. Each site P^i holds a set of $q = n/(2k)$ vertices which is a partition of the set of left vertices U . The neighbors of each vertex in U is determined by a two-party set-disjointness instance (DISJ, defined formally in Section 3.2). In total there are $q \times k = n/2$ DISJ instances, and we want to perform a direct-sum type of argument on these $n/2$ DISJ instances. We show that due to symmetry, the answer of DISJ can be recovered from a reported matching, and then we use information complexity to establish the direct-sum theorem. For this purpose, we also need to give a new definition of the information cost of a protocol in the message-passing model. We believe that our techniques could prove useful in establishing communication complexity for other graph problems in the message-passing model. One reason is that for many graph problems whose solution certificates "span" the whole graph (e.g., connected components, vertex cover, dominating set, etc), it is natural that hard instances would be like for the matching problem, i.e., each of the k sites holds roughly n/k vertices and the neighborhood of each vertex defines an independent instance of a two-party communication problem.

1.2 Related Work

The approximate maximum matching problem has been studied extensively in the literature in various settings. In this section we only review the results obtained in some most related models, namely the streaming computation model [6], the MapReduce model [19, 14], and the traditional distributed model of computation (which is different from ours, see discussions below). In the streaming computation model, the maximum matching problem was presented as one of the open problems by McGregor [1] and a number of results have been established, e.g., by McGregor [26], Epstein et al. [13], Ahn and Guha [2, 3], Ahn, Guha and McGregor [4], Zelke [32], Konard, Magniez and Mathieu [21], Kapralov [17], Kapralov, Khanna and Sudan [18]. Much of the previous work was devoted to the semi-streaming model that allows for $\tilde{O}(n)$ space, and these algorithms can be directly used to obtain an $\tilde{O}(kn)$ communication cost for $O(1)$ -approximate matching in the message-passing model. The maximum matching problem was also studied in the MapReduce model, e.g., by Lattanzi et al. [23]. Under certain assumptions, they obtain a $1/2$ -approximation algorithm in $O(1)$ rounds and $\tilde{O}(m)$ communication bits where m is the number of edges in the graph. In the context of traditional distributed computation models, Lotker et al [25, 24] considered the problem of approximate solving of maximum matching problem in a synchronous distributed computation model. In this computation model, each vertex is associated with a processor and edges represent bidirectional communication. The time is assumed to progress over synchronous rounds where in each round each processor may send messages to its neighbors, which are then received and processed in the same round by their recipients. This computation model is different from the message-passing computation model considered in this paper. In their model the input graph and the communication topology are the same while in the

message-passing model considered here the communication topology is essentially a complete graph which is different from the input graph and in general sites are not vertices in the topology graph. Lotker et al. [24] (built on Wattenhofer and Wattenhofer [28], Lotker et al. [25]) showed existence of $(1 - \epsilon)$ -approximation algorithms for the maximum matching problem with $O(\log n)$ rounds. This implies the communication cost of $\tilde{O}(m)$ bits.

The message-passing computation model has recently attracted quite some attention by the research community, e.g. Phillips, Verbin and Zhang [27], Woodruff and Zhang [29], Braverman et al [10], Woodruff and Zhang [30], Klauck et al [20], and Woodruff and Zhang [31]. A wide set of statistical and graph problems has been shown to be hard in the sense of requiring $\Omega(kn)$ bits of communication, including the graph-connectivity problem [27, 30], exact computation of the number of distinct elements [30], k -party set-disjointness [10], and some were even showed to be hard for random order inputs [20]. A similar but different input distribution from ours was used in [10] to show an $\Omega(kn)$ communication lower bound for the k -party set-disjointness problem. The work presented in this paper was obtained independently and concurrently with [10] with the first version of the paper made online as a technical report [15] in April 2013. Similar distributions were also used previously in [27, 29] which appears to be natural because of the nature of the message-passing model. There may exist a reduction between the k -party set-disjointness studied in [10] and DMR but this is not clear unless one would establish a rigorous proof of this claim. Our proof is different from that in [10]: we use a reduction of the k -party DMR problem to a 2-party set-disjointness problem using symmetrisation, while [10] use a coordinative-wise direct-sum theorem to reduce the k -party set-disjointness problem to a k -party 1-bit problem.

2 Preliminaries

Conventions. Let $[n] = \{1, 2, \dots, n\}$. All logarithms are with base of 2. We use capital letters X, Y, \dots to denote random variables or sets, and the lower case letters x, y, \dots to denote specific values of random variables X, Y, \dots . We write $x \sim \mu$ to mean that x is chosen randomly according to the distribution μ . We often refer to a player as a *site* which is suitable in the coordinator model under consideration.

Information Theory. For two random variables X and Y , we use $H(X)$ to denote the Shannon entropy of the random variable X , and $H(X|Y)$ to denote the conditional entropy of X given Y . Let $I(X; Y) = H(X) - H(X|Y)$ denote the mutual information between X and Y , and $I(X; Y|Z)$ be the conditional mutual information given Z . We know that $I(X; Y) \geq 0$ for any X, Y . We will need the following inequalities from the information theory.

Data processing inequality: If random variables X and Z are conditionally independent given Y , then $I(X; Y | Z) \leq I(X; Y)$ and $I(X; Z) \leq I(X; Y)$.

Super-additivity of mutual information: If X^1, \dots, X^t are independent, then $I(X^1, \dots, X^t; Y) \geq \sum_{i=1}^t I(X^i; Y)$.

Sub-additivity of mutual information: If X^1, \dots, X^t are conditional independent given Y , then $I(X^1, \dots, X^t; Y) \leq \sum_{i=1}^t I(X^i; Y)$.

Communication Complexity. In the two party communication complexity, we have two players Alice and Bob. Alice is given $x \in \mathcal{X}$ and Bob is given $y \in \mathcal{Y}$, and they want to jointly compute some function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{Z}$, by exchanging messages according to a randomized protocol Π . We use Π_{xy} to denote the random transcript (i.e., the concatenation

of messages) when Alice and Bob run Π on the input (x, y) , and $\Pi(x, y)$ to denote the output of the protocol. When the input (x, y) is clear from the context, we will simply use Π to denote the transcript. We say Π is a δ -error protocol if for all (x, y) , the probability that $\Pi(x, y) \neq f(x, y)$ is no larger than δ , where the probability is over the randomness used in Π . Let $|\Pi_{xy}|$ be the length of the transcript. The communication cost of Π is $\max_{x,y} |\Pi_{xy}|$. The δ -error randomized communication complexity of f , denoted by $R_\delta(f)$, is the minimal cost of any δ -error protocol for f . The multi-party NIH communication complexity model is a natural generalization of the two-party model, where we have k parties and each has a piece of input, and they want to compute some function together by exchanging messages. For more information about the communication complexity we refer readers to [22].

Information Complexity. The communication complexity measures the number of bits needed to be exchanged by multiple players in order to compute some function together, while the information complexity studies the amount of information of the inputs that must be revealed by the protocol. It was extensively studied in the last decade, e.g., [11, 7, 8, 29, 9]. There are several definitions of information complexity. In this paper, we will follow the definition used in [7]. In the two-party case, let μ be a distribution on $\mathcal{X} \times \mathcal{Y}$, we define the information cost of Π measured under μ as $IC_\mu(\Pi) = I(XY; \Pi | R)$, where $(X, Y) \sim \mu$ and R is the public randomness used in Π . For any function f , we define the information complexity of f parameterized by μ and δ as $IC_{\mu,\delta}(f) = \min_{\delta\text{-error } \Pi} IC_\mu(\Pi)$.

Information Complexity in the Coordinator Model. We can indeed extend the above definition of information complexity to k -party coordinator model. That is, let X^i be the input of i -th player with $(X^1, \dots, X^k) \sim \mu$ and Π be the whole transcript, then we could define $IC_\mu(\Pi) = I(X^1, \dots, X^k; \Pi | R)$. However, such a definition does not fully explore the point-to-point communication feature of the coordinator model. Indeed, the lower bound we can prove using such a definition is at most what we can prove under the blackboard model and our problem admits a simple algorithm with communication $O(n \log n + k)$ in the blackboard model. In this paper we give a new definition of information complexity for the coordinator model, which allows us to prove higher lower bounds compared with the simple generalization. Let Π^i be the transcript between i -th player and the coordinator, thus $\Pi = \Pi^1 \circ \Pi^2 \circ \dots \circ \Pi^k$. We define the information cost of a problem f with respect to input distribution μ and error parameter δ ($0 \leq \delta \leq 1$) in the coordinator model as $IC_{\mu,\delta}(f) = \min_{\delta\text{-error } \Pi} \sum_{i=1}^k I(X^1, \dots, X^k; \Pi^i)$.

► **Theorem 2.** $R_\delta(f) \geq IC_{\mu,\delta}(f)$ for any distribution μ .

Proof. For any protocol Π , the expected size of its transcript is (we abuse the notation by using Π also for the transcript) $\mathbb{E}[|\Pi|] = \sum_{i=1}^k \mathbb{E}[|\Pi^i|] \geq \sum_{i=1}^k H(\Pi^i) \geq IC_{\mu,\delta}(\Pi)$. The theorem then follows since the worst-case cost is at least the average. ◀

► **Lemma 3.** If Y is independent of the random coins used by the protocol Π , then $IC_{\mu,\delta}(f) \geq \min_{\Pi} \sum_{i=1}^k I(X^i, Y; \Pi^i)$.

Proof. It follows directly from the data processing inequality, since Π and Y are conditionally independent given X^1, \dots, X^k . ◀

3 The Complexity of DMR

In this section we first prove the lower bound in Theorem 1 and then establish its tightness.

An outline of the proof of the lower bound is given as follows. The lower bound is established by constructing a hard distribution on the set of bipartite graphs $G = (U, V, E)$ with $|U| = |V| = n/2$. For the purpose of this outline, we consider the special case in which the number of sites is such that $k = n/2$. Each site is assigned one node in U together with all its adjacent edges. A natural idea to approximately compute a maximum matching in a graph is to randomly sample a few edges from each site, and hope that we can find a good matching using these edges. To rule out such strategies, we create many *noisy* edges: we randomly pick a small set of nodes $V_0 \subset V$ of size roughly $\alpha n/10$ and connect each node in U to each node in V_0 randomly with a constant probability. There are $\Theta(\alpha n^2)$ such edges and the size of the matching formed by these edges is at most $\alpha n/10 \approx \alpha/2 \cdot \text{OPT}$ where OPT is the size of the optimal solution. We next create a set of *important* edges between U and $V \setminus V_0$ such that each node in U is adjacent to at most one random node in $V \setminus V_0$. These edges are important in the sense that although there are only $\Theta(|U|) = \Theta(n)$ of such edges, the size of the matching they can form is $\Theta(\text{OPT})$. Therefore, to compute a matching of size at least $\alpha \cdot \text{OPT}$, it is necessary to find and include $\Theta(\alpha \cdot \text{OPT}) = \Theta(\alpha n)$ important edges. We then show that finding an important edge is in some sense equivalent to solving a set-disjointness (DISJ) instance, and thus we have to solve many DISJ instances. The concrete implementation of this intuition is via an embedding argument. In the general case, we create $n/(2k)$ copies of the random bipartite graph, each of size $2k$. Each site gets $n/(2k)$ nodes. We then prove a direct-sum theorem using information complexity.

The lower bound is established by characterizing the information cost of the DISJ problem for specific input distributions. Before doing this we first characterize the information complexity of a primitive problem AND. We next reduce DISJ to DMR and prove an information cost lower bound for DMR.

3.1 The AND Problem

In the AND problem, Alice and Bob hold bits x and y , respectively, and they want to compute $\text{AND}(x, y) = x \wedge y$. Let A be Alice's input and B be Bob's input. We define two input distributions ν_1 and μ_1 for (A, B) as follows. Let $p = c \cdot \alpha \in (0, 1/2]$, where c is a constant to be chosen later.

ν_1 : Choose a random bit $W \in \{0, 1\}$ such that $\Pr[W = 0] = p$ and $\Pr[W = 1] = 1 - p$. If $W = 0$, we set $B = 0$, and $A = 0$ or 1 with equal probability. If $W = 1$, we set $A = 0$, and set $B = 1$ with probability $1 - p$ and $B = 0$ with probability p . Thus, we have

$$(A, B) = \begin{cases} (0, 0) & \text{with probability } 3p/2 - p^2, \\ (0, 1) & \text{with probability } 1 - 2p + p^2, \\ (1, 0) & \text{with probability } p/2. \end{cases}$$

W here serves as an auxiliary random variable to break the dependence between A and B , since ν_1 is not a product distribution. The use of W will be clear in the reduction. Let τ be the distribution of W . Note that τ partitions ν_1 , i.e., given τ , ν_1 is a product distribution.

μ_1 : Choose W according to τ , and then choose (A, B) according to ν_1 given W . Next, we reset A to be 0 or 1 with equal probability. Let δ_1 be the probability that $(A, B) = (1, 1)$ under distribution μ_1 . We have $\delta_1 = (1 - 2p + p^2)/2$.

For $p = 1/2$, it is proved in [7] that if a private coin protocol Π has worst case error $1/2 - \beta$, then $I(A, B; \Pi | W) \geq \Omega(\beta^2)$, where the information cost is measured with respect to ν_1 . Here we extend this to any $p \leq 1/2$ and distributional error. We say a protocol has a one-sided error δ for AND under a distribution if it is always correct when $\text{AND}(x, y) = 0$, and is correct with probability at least $1 - \delta$ when $\text{AND}(x, y) = 1$.

► **Theorem 4.** *Let Π be the transcript of any public coin protocol for AND on input distribution μ_1 with error probability $\delta_1 - \beta$ for a $\beta \in (0, \delta_1)$. We have $I(A, B; \Pi | W, R) = \Omega(\beta^2 p / \delta_1^2)$, where the information is measured when $W \sim \tau$, $(A, B) \sim \nu_1$, and R is the public randomness. If Π has a one-side error $1 - \beta$, then $I(A, B; \Pi | W, R) = \Omega(\beta p)$.*

Proof. Our proof follows [7]. To handle a general $p \leq 1/2$, we explore the convexity of mutual information. To extend the result to distributional error, we give a more careful analysis and show that the information cost is high as long as the average error is small. The proof is somewhat technical and is deferred to the full version of the paper. ◀

3.2 The DISJ Problem

In the DISJ problem, Alice holds $s = \{s_1, \dots, s_k\} \in \{0, 1\}^k$ and Bob holds $t = \{t_1, \dots, t_k\} \in \{0, 1\}^k$, and they want to compute $\text{DISJ}(s, t) = \bigvee_{\ell=1}^k \text{AND}(s_\ell, t_\ell)$. Let $S = \{S_1, \dots, S_k\}$ be Alice's input and $T = \{T_1, \dots, T_k\}$ be Bob's input. We define two input distributions ν_k and μ_k for (S, T) as follows.

ν_k : Choose $W = \{W_1, \dots, W_k\} \sim \tau^k$, and then choose $(S_\ell, T_\ell) \sim \nu_1$ given W_ℓ , for each $1 \leq \ell \leq k$. For notation convenience, let $\nu_{k|w^*}$ be the distribution of S conditioned on $W = w$, and let $\nu_{k|w^*}$ be the distribution of T conditioned on $W = w$.

μ_k : Choose $W = \{W_1, \dots, W_k\} \sim \tau^k$, and then choose $(S_\ell, T_\ell) \sim \nu_1$ given W_ℓ , for each $1 \leq \ell \leq k$. Next, we pick a special coordinate D uniformly at random from $\{1, \dots, k\}$, and reset S_D to be 0 or 1 with equal probability. Note that $(S_D, T_D) \sim \mu_1$, and the probability that $\text{DISJ}(S, T) = 1$ is also δ_1 . For notation convenience, let $\mu_{k|S=s}$ be the distribution of T conditioned on $S = s$, and let $\mu_{k|T=t}$ be the distribution of S conditioned on $T = t$.

We define the one-sided error for DISJ similarly: A protocol has a one-sided error δ for DISJ if it is always correct when $\text{DISJ}(x, y) = 0$, and is correct with probability at least $1 - \delta$ when $\text{DISJ}(x, y) = 1$.

► **Theorem 5.** *Let Π be the transcript of any public coin protocol for DISJ on input distribution μ_k with error probability $\delta_1 - \gamma$ for a $\gamma \in (0, \delta_1)$. We have $I(S, T; \Pi | W, R) = \Omega(\gamma^2 p k / \delta_1^2)$, where the information is measured when $W \sim \tau^k$, $(S, T) \sim \mu_k$, and R is the public randomness used by the protocol. If Π has a one-sided error $1 - \gamma$, then $I(S, T; \Pi | W, R) = \Omega(\gamma p k)$.*

Proof. The proof is deferred to the full version of the paper. ◀

3.3 Proof of the Main Theorem

To give a proof for Theorem 1, we first reduce DISJ to DMR. Before going to the detailed reduction, we provide an overview of the hard input distribution that we construct for DMR. The whole graph is a random bipartite graph consisting of $q = n/(2k)$ i.i.d. random bipartite graphs G^1, \dots, G^q , where $G^j = (U^j, V^j, E^j)$ with $U^j = \{u^{j,1}, \dots, u^{j,k}\}$ and $V^j = \{v^{j,1}, \dots, v^{j,k}\}$. The set of neighbors of each vertex $u^{j,i} \in U^j$, for $i \in [k]$, is determined by a k -bit random vector $X^{j,i}$, that is, $(u^{j,i}, v^{j,\ell}) \in E^j$ if $X_\ell^{j,i} = 1$. The k (k -bit) random vectors $\{X^{j,1}, \dots, X^{j,k}\}$ are chosen as follows: we first choose $(X^{j,1}, Y^j) \sim \mu_k$, and then independently choose for each $i \in \{2, \dots, k\}$, a k -bit vector $X^{j,i}$ according to the conditional distribution $\mu_{k|T=Y^j}$. Finally, the input for the i -th site is simply vertices $\{u^{1,i}, \dots, u^{q,i}\}$ and all their incident edges, which is actually determined by $X^i = \{X^{1,i}, \dots, X^{q,i}\}$. Note that $Y = \{Y^1, \dots, Y^q\}$ is *not* part of the input for DMR; it is used to construct $X^{j,i}$ ($i \in [k], j \in [q]$).

Input Reduction. Let $s \in \{0, 1\}^k$ be Alice's input and $t \in \{0, 1\}^k$ be Bob's input for DISJ. Alice and Bob construct an input $\{X^1, \dots, X^k\}$ for DMR, where $X^i = \{X^{1,i}, \dots, X^{q,i}\}$ with $X^{j,i} \in \{0, 1\}^k$ ($j \in [q]$) is the input for site i .

1. Alice and Bob use public coins to sample an index I uniformly at random from $\{1, \dots, k\}$. Alice constructs the input X^I for the I -th site, and Bob constructs the inputs $X^1, \dots, X^{I-1}, X^{I+1}, \dots, X^k$ for the other $k - 1$ sites.
2. Alice and Bob use public coins to sample an index J uniformly at random from $\{1, \dots, q\}$.
3. Alice sets $X^{J,I} = s$, and Bob sets $Y^J = t$. For each $i \in [k] \wedge i \neq I$, Bob privately samples $X^{J,i}$ according to $\mu_k|_{T=t}$. This finishes the construction of G^J .
4. For each $j \in [q] \wedge j \neq J$, they construct G^j as follows,
 - (a) Alice and Bob first use public coins to sample $W^j = \{W_1^j, \dots, W_k^j\} \sim \tau^k$ (see the definition of τ in Section 3.1).
 - (b) Alice and Bob privately sample $X^{j,I}$ and Y^j according to conditional distributions $\nu_k|_{*W^j}$ and $\nu_k|_{W^j*}$, respectively. Bob also privately samples $X^{j,1}, \dots, X^{j,I-1}, X^{j,I+1}, \dots, X^{j,k}$ independently according to the conditional distribution $\nu_k|_{T=Y^j}$.
 - (c) Alice privately samples $D^{j,I}$ uniformly at random from $\{1, \dots, k\}$, and resets $X_{D^{j,I}}^{j,I}$ to be 0 or 1 with equal probability. This makes $\{X^{j,I}, Y^j\} \sim \mu_k$. Bob does the same for all $i \in [k] \wedge i \neq I$. That is, for each $i \in [k] \wedge i \neq I$, he privately samples $D^{j,i}$ uniformly at random from $\{1, \dots, k\}$, and resets $X_{D^{j,i}}^{j,i}$ to be 0 or 1 with equal probability.

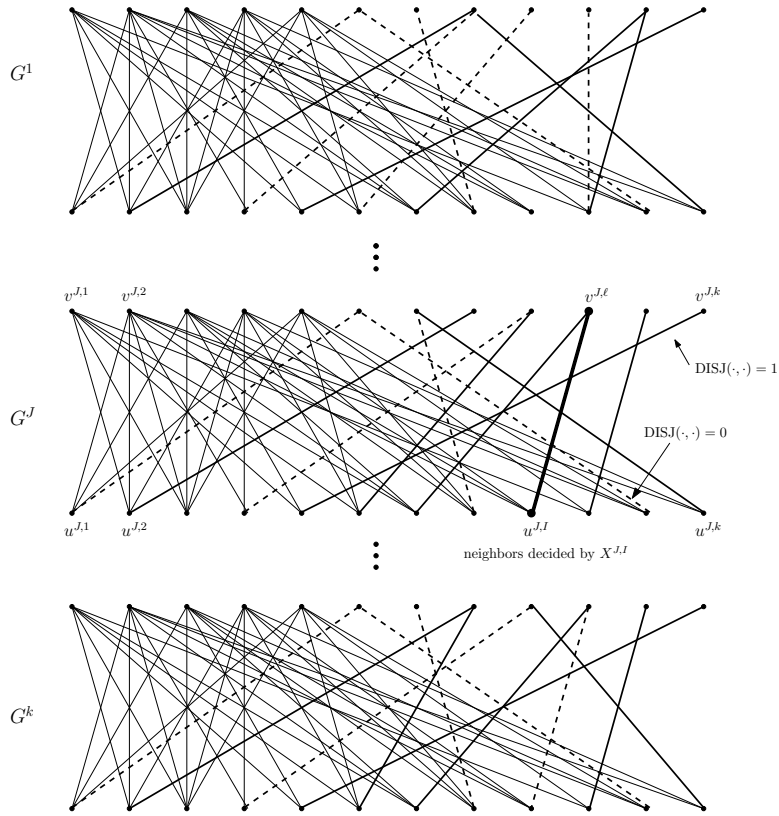
Note that the I -th site's input X^I is determined by the public coins, Alice's input s and her private coins. And the remaining $k - 1$ sites' inputs $\{X^1, \dots, X^{I-1}, X^{I+1}, \dots, X^k\}$ are determined by the public coins, Bob's input t and his private coins. Let ϕ denote the distribution of $\{X^1, \dots, X^k\}$ when (s, t) is chosen according to the distribution μ_k . We have included Figure 1 for the illustration purpose.

In this reduction, in each bipartite graph G^j , we carefully embed k instances of DISJ in random positions, and the output of a DISJ instance determines whether a specific edge in the graph exists or not. In the whole graph, we embed a total of $k \times q = n/2$ DISJ instances. The input of one such DISJ instance is just the original input of Alice and Bob, and the other $(n/2 - 1)$ instances are sampled by Alice and Bob using public and private random coins. Such a symmetric construction can be used to argue that if the original DISJ instance is solved, then with a good probability, at least $\Omega(n)$ of embedded DISJ instances are solved. We will see the proof that the original DISJ instance can be solved by solving DMR also relies on the symmetric property.

Let $p = \alpha/20 \leq 1/20$, where recall that p is a parameter in distribution μ_k and α is the approximation parameter. Now, given a protocol \mathcal{P}' for DMR that achieves an α -approximation and error probability $1/4$ with respect to ϕ , we construct a protocol \mathcal{P} for DISJ with one-sided error probability $1 - \alpha/10$ with respect to μ_k , as follows.

Protocol \mathcal{P}

1. Given an input $(S, T) \sim \mu_k$, Alice and Bob construct an input $\{X^1, \dots, X^k\} \sim \phi$ for DMR as described by the input reduction above. Let $Y = \{Y^1, \dots, Y^q\}$ be the set sampled during the construction of $\{X^1, \dots, X^k\}$. Let I, J be the two indices sampled by Alice and Bob during the reduction.
2. Alice plays the I -th site, and Bob plays the other $k - 1$ sites and the coordinator. They run \mathcal{P}' for DMR. Any communication between the I -th site and the other $k - 1$ sites and the coordinator will be exchanged between Alice and Bob. For any communication between the other $k - 1$ sites and the coordinator, Bob just simulates it without any actual communication. At the end the coordinator (that is, Bob) gets a matching M .



■ **Figure 1** Each edge corresponds to a DISJ instance where a solid edge indicates an instance with output 1 and a dashed edge indicates an instance with output 0. Solid thick edges are the important edges. A good approximate matching has to output many important edges, thus a solid thick edge needs to be in the output matching with sufficiently large probability. The thick edge $(u^{J,I}, v^{J,L})$ corresponds to $\text{DISJ}(X^{J,I}, Y^J) = \text{DISJ}(s, t)$, that is to the original 2-party disjointness problem embedded by Alice and Bob. If $\text{DISJ}(s, t) = 1$, then $(u^{J,I}, v^{J,L})$ is a solid edge and needs to be included in the output matching with a sufficiently large probability.

3. Bob outputs 1 if and only if there exists an edge $(u^{J,I}, v^{J,L})$ in the matching M for some $\ell \in [k]$, such that $Y_\ell^J \equiv T_\ell = 1$, and 0 otherwise.

Correctness. First, suppose $\text{DISJ}(S, T) = 0$, i.e., $S_\ell \wedge T_\ell = 0$ for all $\ell \in [k]$. Then, for each $\ell \in [k]$, we must have either $Y_\ell^J \equiv T_\ell = 0$ or $X_\ell^{J,I} \equiv S_\ell = 0$, but $X_\ell^{J,I} = 0$ means no edge between $u^{J,I}$ and $v^{J,\ell}$. Thus \mathcal{P} will always answer correctly when $\text{DISJ}(S, T) = 0$, i.e., it has a one-sided error.

Now suppose that $S_\ell = T_\ell = 1$ for a certain $\ell \in [k]$ (note that there is at most one such ℓ according to our construction), which we denoted by L . The output of \mathcal{P} is correct if $(u^{J,I}, v^{J,L}) \in M$. In the rest of the analysis we estimate the probability that this event happens.

For each $G^j = \{U^j, V^j\}$ ($j \in [q]$), let $U_1^j = \{u^{j,i} \mid \text{DISJ}(X^{j,i}, Y^j) = 1\}$ and $U_0^j = U^j \setminus U_1^j$. Let $V_1^j = \{v^{j,\ell} \mid Y_\ell^j = 1\}$ and $V_0^j = V^j \setminus V_1^j$. Let $U_0 = \cup_{j=1}^q U_0^j$, $U_1 = \cup_{j=1}^q U_1^j$, $V_0 = \cup_{j=1}^q V_0^j$ and $V_1 = \cup_{j=1}^q V_1^j$. Intuitively, edges between $U_0 \cup U_1$ and V_0 can be seen as *noisy* edges, since the total number of such edges is large but the maximum matching they can form is small (at most $|V_0| \leq 2pn$ according to Lemma 6, see below). On the contrary, we say the

edges between U_1 and V_1 the *important* edges, since the maximum matching they can form is large, though the total number of such edges is small. Note that there is no edge between U_0 and V_1 . Therefore, to find a good matching we must choose many edges from the important edges. A key feature here is that all important edges are *symmetric*, that is, each important edge is equally likely to be the edge $(u^{J,I}, v^{J,L})$. Thus with a good probability $(u^{J,I}, v^{J,L})$ will be included in the matching returned by \mathcal{P}' . Using this we can answer whether $X^{J,I}$ ($= S$) and Y^J ($= T$) intersect or not, thus solving the original DISJ problem.

We first estimate the size of the maximum matching in graph $G = \{G^1, \dots, G^q\}$. Recall we set $p = \alpha/20 \leq 1/20$ and $\delta_1 = (1 - 2p + p^2)/2$, thus $9/20 < \delta_1 < 1/2$.

► **Lemma 6.** *With probability 0.99, the following events happen.*

1. $|V_0| \leq 2pn$. In this case the size of the maximum matching formed by edges between V_0 and $U_0 \cup U_1$ is no more than $2pn$.
2. The maximum matching of the graph G is at least $0.2n$.

Proof. The first item follows simply by a Chernoff bound. Note that each vertex in $\bigcup_{j \in [q]} V^j$ is included in V_0 independently with probability $(2p - p^2)$, and $\mathbb{E}[|V_0|] = (2p - p^2)n/2$, therefore $\Pr[|V_0| \geq 2pn] \leq \Pr[|V_0| - \mathbb{E}[|V_0|] \geq pn] \leq e^{-\Omega(p^2n)}$.

For the second item, we first consider the size of the matching in G^j for a fixed $j \in [q]$, that is, a matching between vertices in U^j and V^j . For each $i \in [k]$, let L^i be the coordinate ℓ where $X_\ell^{j,i} = Y_\ell^j = 1$ if such an ℓ exists (note that by our construction at most one such coordinate exists), and NULL otherwise.

We use a greedy algorithm to construct a matching between U^j and V^j . For i from 1 to k , we connect $u^{j,i}$ to v^{j,L^i} if L^i is not NULL and v^{j,L^i} is not connected by any $u^{j,i'}$ ($i' < i$). At the end, the size of the matching is essentially the number of distinct elements in $\{L^1, \dots, L^k\}$, which we denote by R . We have the following claim.

► **Claim 1.** It holds $R \geq 0.25k$ with probability $1 - O(1/k)$.

Proof. The proof is similar to Lemma 4 in [29]. By our construction, we have $\mathbb{E}[|U_1^j|] = \delta_1 k$ and $\mathbb{E}[|V_1^j|] = (1 - 2p + p^2)k$. Similar to the first item we have that with probability $(1 - e^{-\Omega(k)})$, $|V_1^j| \geq 0.9 \cdot \mathbb{E}[|V_1^j|] = 0.9 \cdot (1 - 2p + p^2)k \geq 0.8k$ (recall $p \leq 1/20$) and $|U_1^j| \geq 0.9 \cdot \mathbb{E}[|U_1^j|] \geq 0.4k$. Therefore with probability $(1 - e^{-\Omega(k)})$, R must be at least the value R' of the following bin-ball game: We throw each of $0.4k$ balls to one of the $0.8k$ bins uniformly at random, and then count the number of non-empty bins at the end of the process. By Fact 1 and Lemma 1 in [16], we have $\mathbb{E}[R'] = (1 - \lambda) \cdot 0.4k$ for some $\lambda \in [0, 1/4]$ and $\text{Var}[R'] < 4(0.4k)^2 / (0.8k) = 0.8k$. Thus by Chebyshev's Inequality we have

$$\Pr[R' < \mathbb{E}[R'] - 0.05k] \leq \frac{\text{Var}[R']}{(0.05k)^2} < 320/k.$$

Thus with probability $1 - O(1/k)$, we have $R \geq R' \geq 0.25k$. ◀

Therefore, for each $j \in [q]$, with probability $1 - O(1/k)$, we can find a matching in G^j of size at least $0.25k$. If $q = n/(2k) = o(k)$, then by a simple union bound it holds that with probability at least 0.99, the size of the maximum matching in $G = \{G^1, \dots, G^q\}$ is at least $0.25n$. Otherwise, since G^1, \dots, G^q are constructed independently, by another application of Chernoff bound, we have that with probability $1 - e^{-\Omega(q)} \geq 0.99$, the size of the maximum matching in $G = \{G^1, \dots, G^q\}$ is at least $0.2n$. ◀

Now let us make our intuition above more precise. First, if \mathcal{P}' is an α -approximation protocol with error probability $1/4$, then by Lemma 6 we have that with probability at

least $3/4 - 0.01 \geq 2/3$, \mathcal{P}' will output a matching M containing at least $(\alpha \cdot 0.2n - 2pn)$ important edges. We know that there are at most $n/2$ important edges and the edge $(u^{J,I}, v^{J,L})$ is one of them. We say (i, j, ℓ) is important for G , if $(u^{j,i}, v^{j,\ell})$ is an important edge in G . Since our construction is totally symmetric, for any G in the support, we have $\Pr[I = i, J = j, L = \ell \mid G] = \Pr[I = i', J = j', L = \ell' \mid G]$, for any (i, j, ℓ) and (i', j', ℓ') which are important in G . In other words, given an input G , the protocol can not distinguish between any two important edges. Then we can apply the principle of deferred decisions to decide the value (I, J) after the matching has already been computed, i.e., the probability $(u^{J,I}, v^{J,L}) \in M$ is at least $2/3 \cdot \frac{\alpha \cdot 0.2n - 2pn}{n/2} \geq \alpha/10$. Recall that we have chosen $p = \alpha/20$. To sum up, protocol \mathcal{P} solves DISJ correctly with one-sided error at most $1 - \alpha/10$.

Information Cost. Now we analyze the information cost of DMR. Let $\Pi = \Pi^1 \circ \Pi^2 \circ \dots \circ \Pi^k$ be the best protocol for DMR with respect to input distribution ϕ and one-sided error probability $1 - \alpha/10$. By Lemma 3, we have $IC_{\phi, \delta}(\text{DMR}) \geq \sum_{i=1}^k I(X^i, Y; \Pi^i)$. Let $W^{-J} = \{W^1, \dots, W^q\} \setminus W^J$, and $W = W^J W^{-J}$. Recall that in our input reduction I, J, W^{-J} are public coins used by Alice and Bob.

$$\begin{aligned}
2/n \cdot IC_{\phi, \delta}(\text{DMR}) &\geq 1/(qk) \cdot \sum_{i=1}^k I(X^i, Y; \Pi^i) \\
&\geq 1/(qk) \cdot \sum_{i=1}^k I(X^i, Y; \Pi^i \mid W) \quad (\text{data processing inequality}) \\
&\geq 1/(qk) \cdot \sum_{i=1}^k \sum_{j=1}^q I(X^{j,i}, Y^j; \Pi^i \mid W^{-j}, W^j) \quad (\text{super-additivity}) \quad (1) \\
&= 1/(qk) \cdot \sum_{i=1}^k \sum_{j=1}^q I(S, T; \Pi^i \mid I = i, J = j, W^{-j}, W_{S,T}) \quad (2) \\
&= I(S, T; \Pi^I \mid I, J, W^{-J}, W_{S,T}) \\
&\geq I(S, T; \Pi^* \mid W_{S,T}, R) \quad (3) \\
&= \Omega(\alpha^2 k), \quad (4)
\end{aligned}$$

where

1. $W_{S,T} \sim \tau^k$ is the random variable used to sample (S, T) from μ_k . Eq. (2) holds because the distribution of W^j is the same as that of $W_{S,T}$, and the conditional distribution of $(X^{j,i}, Y^j, \Pi^i \mid W^{-j}, W^j)$ is the same as $(S, T, \Pi^i \mid I = i, J = j, W^{-j}, W_{S,T})$.
2. In Eq. (3), Π^* is the best protocol for DISJ with one-sided error probability at most $1 - \alpha/10$ and R is the public randomness used in Π^* . The information is measured according to μ_k .
3. Eq. (4) holds by Theorem 5. Recall that we have set $p = \alpha/20$.

Therefore, we have $R_{1/4}(\text{DMR}) \geq IC_{\phi, 1/4}(\text{DMR}) \geq \Omega(\alpha^2 kn)$, proving our Theorem 1.

3.4 Tightness of the Lower Bound

In this section we present an α -approximation algorithm with an upper bound on the communication complexity which matches the lower bound for $\alpha \leq 1/2$ up to polylogarithmic factors.

The algorithm consists of two steps. In the first step, each site computes a local maximum matching and sends its size to the coordinator. The coordinator compares these sizes, and

then sends a message to the site that has the largest local maximum matching. This site then sends the local maximum matching to the coordinator. We can assume that the size of this matching is not larger than αn , as otherwise, the local matching of that site can be declared to be the output of the algorithm, since it is already an α -approximation. Note that the communication cost of this step is at most $O((k + \alpha n) \log n)$ bits. In the second step, the coordinator picks each site randomly with probability $\alpha' = 8\alpha$, and computes a maximal matching among the sites picked using the straightforward algorithm that we described in the introduction. The communication cost of this step is at most $O((k + \alpha^2 kn) \log n)$ bits in expectation. We next show correctness of the algorithm.

Let X_i be a random variable indicating the event that the i -th site is picked in the second step, and we have $\mathbb{E}[X_i] = \alpha'$ and $\text{Var}[X_i] = \alpha'(1 - \alpha')$. Let M be the global maximum matching and $m = |M|$. We use m_i to denote the number of edges in M which belong to the i -th site, thus $\sum_i m_i = m$ (recall that we assume edge partitioning where edges are partitioned disjointly across the set of k sites). For the same reason as in the first step, we can again assume that $m_i \leq \alpha m$ for all $i \in [k]$, since otherwise, we will already get an α -approximation. Let Y be the size of the maximal matching that is obtained in the second step. Recall that a maximal matching is at least $1/2$ of a maximum matching, thus we have $Y \geq \frac{1}{2} \cdot \sum_{i=1}^k m_i X_i$. Let $Y' = \sum_{i=1}^k m_i X_i$. So we have $\mathbb{E}[Y'] = \alpha' m$ and $\text{Var}[Y'] = \alpha'(1 - \alpha') \sum_{i=1}^k m_i^2 \leq \alpha' \cdot \alpha m^2 = 8\alpha^2 m^2$. The inequality holds since we assume that $m_i \leq \alpha m$ for all $i \in [k]$. Now, we can apply Chebyshev's inequality to bound the error probability. We have $\Pr[|Y' - \alpha' m| \geq 6\alpha m] \leq 8/36 < 1/4$. Therefore, with probability at least $3/4$, it holds $Y \geq 1/2 \cdot Y' \geq 1/2 \cdot 2\alpha m = \alpha m$.

► **Theorem 7.** *For every given $\alpha \leq 1/2$, there exists a randomized algorithm that computes an α -approximation of the maximum matching in a graph with probability at least $3/4$ at the communication cost of $O((k + \alpha^2 nk + \alpha n) \log n)$ bits.*

Note that $\Omega(\alpha n)$ is a trivial lower bound, simply because the size of the output could be as large as $\Omega(\alpha n)$. Obviously, $\Omega(k)$ is a lower bound, since the coordinator has to talk to each of the sites at least once. Thus, together with the lower bound $\Omega(\alpha^2 kn)$ in Theorem 1, the upper bound above is tight up to a $\log n$ factor.

4 Concluding Remarks

In this paper we showed a tight lower bound on the communication complexity for the approximate maximum matching problem in the message-passing model. An interesting open problem is the complexity of the counting version of the problem, i.e., the communication complexity if we only want to compute an approximation of the *size* of a maximum matching in a graph. Note that our proof of the lower bound relies on the fact that the algorithm has to return a certificate of the matching. Hence, in order to prove a lower bound for the counting version of the problem one may need to use new ideas and it is also possible that a better upper bound exists. In a recent work [18], the counting version of the matching problem was studied in the random-order streaming model. They proposed an algorithm that uses one pass and polylog space, which computes a polylog approximation of the size of the maximum matching. A general interesting direction for future research is to investigate the communication complexity for other combinatorial problems on graphs, for example, connected components, minimum spanning tree, vertex cover and dominating set. The techniques used for approximate maximum matching problem in the present paper could be of use here.

Acknowledgements The authors would like to thank Ke Yi for useful discussions.

References

- 1 Question 16: Graph matchings (Andrew McGregor) in open problems in data streams and related topics IITK workshop on algorithms for data streams, 2006. <http://www.cse.iitk.ac.in/users/sganguly/data-stream-probs.pdf>.
- 2 Kook Jin Ahn and Sudipto Guha. Laminar families and metric embeddings: Non-bipartite maximum matching problem in the semi-streaming model. *CoRR*, abs/1104.4058, 2011.
- 3 Kook Jin Ahn and Sudipto Guha. Linear programming in the semi-streaming model with application to the maximum matching problem. In *Proceedings of the 38th international conference on Automata, languages and programming - Volume Part II*, ICALP'11, pages 526–538, Berlin, Heidelberg, 2011. Springer-Verlag.
- 4 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 459–467. SIAM, 2012.
- 5 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proceedings of the 31st symposium on Principles of Database Systems*, PODS '12, pages 5–14, New York, NY, USA, 2012. ACM.
- 6 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.
- 7 Z. Bar-Yossef, T. S. Jayram, R. Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *J. Comput. Syst. Sci.*, 68:702–732, June 2004.
- 8 B. Barak, M. Braverman, X. Chen, and A. Rao. How to compress interactive communication. In *Proceedings of the 42nd ACM symposium on Theory of computing*, pages 67–76. ACM, 2010.
- 9 M. Braverman. Interactive information complexity. In *Proceedings of the 44th symposium on Theory of Computing*, pages 505–524. ACM, 2012.
- 10 Mark Braverman, Faith Ellen, Rotem Oshman, Toniann Pitassi, and Vinod Vaikuntanathan. A tight bound for set disjointness in the message-passing model. In *FOCS*, pages 668–677, 2013.
- 11 Amit Chakrabarti, Yaoyun Shi, Anthony Wirth, and Andrew Yao. Informational complexity and the direct sum problem for simultaneous message complexity. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 270–278, 2001.
- 12 Jack Clark. Facebook rides Unicorn to graph search nirvana. The Register, http://www.theregister.co.uk/2013/03/07/facebook_unicorn_helps_graph_search, January 2013.
- 13 L. Epstein, A. Levin, J. Mestre, and D. Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM Journal on Discrete Mathematics*, 25(3):1251–1265, 2011.
- 14 Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. pages 374–383, 2011.
- 15 Zengfeng Huang, Bozidar Radunovic, Milan Vojnovic, and Qin Zhang. Communication complexity of approximate maximum matching in distributed graph data. no. MSR-TR-2013-35, <http://research.microsoft.com/apps/pubs/default.aspx?id=188946>, 2013.
- 16 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proc. ACM Symposium on Principles of Database Systems*, 2010.

- 17 Michael Kapralov. Improved lower bounds for matchings in the streaming model. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '13*, 2013.
- 18 Michael Kapralov, Sanjeev Khanna, and Madhu Sudan. Approximating matching size from random streams. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 734–751, 2014.
- 19 Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. pages 938–948, 2010.
- 20 Hartmut Klauck, Danupon Nanongkai, Gopal Pandurangan, and Peter Robinson. The distributed complexity of large-scale graph processing. *CoRR*, abs/1311.6209, 2013.
- 21 Christian Konrad, Frédéric Magniez, and Claire Mathieu. Maximum matching in semi-streaming with few passes. In *APPROX-RANDOM*, pages 231–242, 2012.
- 22 E. Kushilevitz and N. Nisan. Communication complexity. 1997.
- 23 Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. Filtering: a method for solving graph problems in mapreduce. In *Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures, SPAA '11*, pages 85–94, New York, NY, USA, 2011. ACM.
- 24 Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. Improved distributed approximate matching. In *SPAA*, pages 129–136, 2008.
- 25 Zvi Lotker, Boaz Patt-Shamir, and Adi Rosen. Distributed approximate matching. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing, PODC '07*, pages 167–174, New York, NY, USA, 2007. ACM.
- 26 Andrew McGregor. Finding graph matchings in data streams. In *Proceedings of the 8th international workshop on Approximation, Randomization and Combinatorial Optimization Problems, and Proceedings of the 9th international conference on Randomization and Computation: algorithms and techniques, APPROX'05/RANDOM'05*, pages 170–181, Berlin, Heidelberg, 2005. Springer-Verlag.
- 27 Jeff M. Phillips, Elad Verbin, and Qin Zhang. Lower bounds for number-in-hand multiparty communication complexity, made easy. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '12*, pages 486–501. SIAM, 2012.
- 28 Mirjam Wattenhofer and Roger Wattenhofer. Distributed weighted matching. In *Distributed Computing, 18th International Conference, DISC 2004, Amsterdam, The Netherlands, October 4-7, 2004, Proceedings*, pages 335–348, 2004.
- 29 David P. Woodruff and Qin Zhang. Tight bounds for distributed functional monitoring. In *Proceedings of the 44th symposium on Theory of Computing, STOC '12*, pages 941–960, New York, NY, USA, 2012. ACM.
- 30 David P. Woodruff and Qin Zhang. When distributed computation does not help. *CoRR*, abs/1304.4636, 2013.
- 31 David P. Woodruff and Qin Zhang. An optimal lower bound for distinct elements in the message passing model. In *SODA*, pages 718–733, 2014.
- 32 Mariano Zelke. Weighted matching in the semi-streaming model. *Algorithmica*, 62(1-2):1–20, February 2012.

Stochastic Scheduling of Heavy-tailed Jobs

Sungjin Im^{*1}, Benjamin Moseley², and Kirk Pruhs^{†3}

- 1 Electrical Engineering and Computer Science
University of California, Merced
sim3@ucmerced.edu
- 2 Computer Science and Engineering
Washington University in St. Louis
bmoseley@wustl.edu
- 3 Computer Science
University of Pittsburgh
kirk@cs.pitt.edu

Abstract

We revisit the classical stochastic scheduling problem of nonpreemptively scheduling n jobs so as to minimize total completion time on m identical machines, $P \parallel \mathbb{E} \sum C_j$ in the standard 3-field scheduling notation. Previously it was only known how to obtain reasonable approximation if jobs sizes have low variability. However, distributions commonly arising in practice have high variability, and the upper bounds on the approximation ratio for the previous algorithms for such distributions can be even inverse-polynomial in the maximum possible job size. We start by showing that the natural list scheduling algorithm Shortest Expected Processing Time (SEPT) has a bad approximation ratio for high variability jobs. We observe that a simple randomized rounding of a natural linear programming relaxation is a $(1 + \epsilon)$ -machine $O(1)$ -approximation assuming the number of machines is at least logarithmic in the number of jobs. Turning to the case of a modest number of machines, we develop a list scheduling algorithm that is $O(\log^2 n + m \log n)$ -approximate. Our results together imply a $(1 + \epsilon)$ -machine $O(\log^2 n)$ -approximation for an arbitrary number of machines. Intuitively our list scheduling algorithm finds an ordering that not only takes the expected size of a job into account, but also takes into account the probability that job will be big.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, Sequencing and scheduling

Keywords and phrases stochastic scheduling, completion time, approximation

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.474

1 Introduction

Scheduling jobs on identical machines with the objective of minimizing total completion time is a well studied class of scheduling problems as well as being one of the most basic multiple machines scheduling settings. Basic versions of these problems and their variants are reasonably well understood in both the online and offline settings. For example, if the processing times of the jobs are available to the scheduler, then list scheduling the jobs in increasing order of their sizes yields an optimal nonpreemptive schedule [4]. See [16]

* Supported in part by NSF grant CCF-1409130.

† Supported in part by NSF grants CCF-1115575, CNS-1253218, CCF-1421508, and an IBM Faculty Award.

for various approximation results. Unfortunately, in many systems, the scheduler may not know a priori the exact processing times of jobs. But often the scheduler may know a priori, from past invocations of the job, stochastic information about a job's size. Thus there is significant research on such stochastic scheduling problems (see for example [9]).

We revisit the classical stochastic scheduling problem of *non-preemptively* scheduling a collection \mathcal{J} of n jobs so as to minimize the total expected completion time on m identical machines, $P \parallel \mathbb{E} \sum C_j$ in the standard 3-field scheduling notation. The algorithm and optimal solution are assumed to be nonanticipatory. In the stochastic setting, a nonanticipatory scheduler only knows a priori the probability distribution on the size P_j of each job j . If a machine is not running a job at a particular time, the nonanticipatory scheduler may optionally assign a job to that machine at that time; if it assigns a job, then the job must be run to completion, which is when the nonanticipatory scheduler finally learns the realized size of the job. Note that the scheduler can be dynamic in the sense that it can make scheduling decisions at time t based on the revealed processing times of jobs scheduled up to time t . If a job j is started at time S_j and is still being scheduled at time t then the scheduler only knows the job's size is at least $t - S_j$ and if the job j completes by time t , the scheduler knows its realized size.

In contrast to the worst-case offline and online settings, this problem is not well understood in the stochastic setting for general distributions. The most natural nonanticipatory algorithm is Shortest Expected Processing Time (SEPT), which always assigns a job with the minimum expected size to a machine that is free. SEPT is known to be an optimal nonanticipatory algorithm on a single machine [10]. As is standard in stochastic settings, approximation is relative to the optimal nonanticipatory algorithm. For identical machines, SEPT is optimal if job sizes are exponentially distributed or are stochastically comparable in pairs [15, 14]. However, for general distributions, even the question of whether SEPT has a reasonable approximation ratio independent of the variability of the jobs' processing times for identical machines was open.

We start by resolving this open question by showing that the approximation ratio of SEPT is $\Omega(n^{1/4})$. For full details see Section 2. It is useful to consider this instance as it also represents the types of instances that are hard to design and analyze algorithms for. In this instance most of the jobs are small with high probability. The remaining jobs are either of size zero, or are (very) big. The expected number of jobs that become big is approximately m . If the high variability jobs are run first (as SEPT does given the proper settings of the parameters) then there can be a large difference in the cost depending on whether m jobs become big, clogging up the machines for a long period of time, or whether only $m - 1$ jobs become big, leaving one machine to finish off the small jobs. This instance demonstrates that the problem is delicate/fragile in the sense that small changes in the input, or changes in the random realizations of sizes for a small number of jobs, can have a large impact on the objective.

This demonstrates that the main difficulty in these types of stochastic scheduling problems is handling high variability jobs. A standard research approach for problems, that are difficult when some parameter is large, is to seek algorithms that perform reasonably when this parameter is small. To the best of our knowledge, this is the approach taken by all previous research for these types of stochastic scheduling problems. For example, [8, 13] give algorithms for our problem $P \parallel \mathbb{E} \sum C_j$, and generalizations thereof, and show that they have approximation ratios that are roughly linear in the squared coefficient of variation, which is the variance divided by the square of the expectation. So these algorithms would provide reasonable approximation if the sizes of the jobs did not have high variability.

Unfortunately, the distributions that most commonly arise in practice, Zipf distributions with small α parameters, have high variability [3, 5, 1]. In a Zipf distribution the probability that a job has size s is proportional to $1/s^\alpha$. Zipf distributions with $\alpha \in (1, 2)$ have a squared coefficient of variation which is $\tilde{\Omega}(P^{\alpha-1})$ where P is the maximum possible job size. For example, for the common case that $\alpha \approx 2$, this gives approximation ratio approximately $O(P)$, which is quite weak as it is achieved by every algorithm that doesn't unnecessarily idle a processor.

The starting point for our research was to investigate whether one can obtain reasonable approximation when jobs may have high variability. It was clear a priori that success would require the development of new analysis techniques. The approaches in the stochastic scheduling literature all require low variability. The approaches in the deterministic scheduling literature are all essentially based on volume arguments, in which one bounds the total volume of work that will be processed by the algorithm before a specific job completes. Unfortunately, on instances such as the lower bound instance for SEPT, similar volume based arguments again only lead to approximate ratios depending on the squared coefficient of variation [8, 13]. Further the most natural candidate algorithms, like SEPT, all perform badly.

1.1 Our Results and Contributions

Our main result is a polynomial-time $(1 + \epsilon)$ -machine $O(\frac{1}{\epsilon^2} \log^2 n)$ -approximation algorithm for any number of machines. Our analysis is broken into two parts depending on the number of machines as compared to the number of jobs. The most challenging case is when the number of machines is small. In Section 4 we develop a novel polynomial-time list scheduling algorithm LS. Here list scheduling means that the algorithm initially computes an ordered list of the jobs, and whenever a machine becomes free, the algorithm assigns the next job on the list to the available machine. In Section 5 we show the approximation ratio for LS is $O(\log^2 n + m \log n)$. When m is $O(\log n)$ this implies an $O(\log^2 n)$ -approximation *without* resource augmentation. Adopting the viewpoint of the approximation algorithms community that a poly-log approximation is reasonable, albeit at the top end of reasonable, then LS is reasonable for a smallish/poly-log number of machines. Our main result is based on two, related, insights:

- We can construct a lower bound for optimum using two characteristics of the jobs:
 - The expected size of a job (so the same characteristic that SEPT uses), and
 - the probability that a job becomes big.
- We can construct a list using these same two parameters so that the list scheduling algorithm LS will never be too far off from our lower bound.

In most stochastic approximation literature, finding a good lower bound for the adaptive adversary is crucial for the analysis, and we believe our lower bound is worth further investigation for other stochastic scheduling problems. As mentioned before, prior to our work, the best known approximation ratio for arbitrary job size distributions was the maximum possible job size, even when there are only two machines. A more detailed overview can be found in Section 3.

We then consider the case that the number of machines is not small relative to n . In our lower bound instance for SEPT it is the case that the objective could be significantly affected by a small change in the amount of resources available. This is a clear signal that a resource augmentation analysis might be useful. If the number of machines is large, then allowing the algorithm some modest resource/machine augmentation seems reasonable.

We show in Section 6 that a simple randomized rounding of a natural linear programming relaxation is a $(1 + \epsilon)$ -machine $O(1)$ -approximation provided that $m \geq \Omega((1/\epsilon^2) \log n)$; here the algorithm is allowed to use $(1 + \epsilon)m$ machines and is compared against the optimal adaptive algorithm that can only use m machines. The analysis follows by showing, using standard concentration arguments, that if there are at least this many machines, then with high probability it is never that case that all machines are “clogged up”.

Finally, we show in Section 2 that the approximation ratio of SEPT is $\Omega(n^{1/4})$.

1.2 Other Related Work

Most of the related results in the literature also hold for the more general problem where jobs have weights and the objective is the weighted sum of completion times. For a single machine, the algorithm WSEPT (running jobs with high weight to expected size ratio) is 2-approximate, and this approximation ratio is best possible [10]. Turning back to identical machines, it is known that WSEPT is asymptotically optimal; that is, the approximation tends to one as the number of jobs tends to infinity [14, 15]. [8] gives a list scheduling policy based on a linear programming relaxation where the approximation is linear in the squared coefficient of variation. It is known that this is the best approximation ratio possible if jobs must be irrevocably assigned to machines a priori [13]. This approach was extended to an online setting in [11], to allow the possibility of precedence constraints in [12], and to allow the possibility of related machines in [13]. The fact that the approximation results also hold for weighted completion time is in part explained by the fact that they are based on linear programming formulations, which easily incorporate weights. [7] gives a combinatorial algorithm for the setting that job arrive online in a list, and must be assigned to machines as they arrive, and again show an approximation ratio that is linear in the square coefficient of variation. For deterministic sizes, there is a polynomial time approximation scheme [2].

2 Lower Bound for SEPT

► **Theorem 1.** *The algorithm SEPT rule has an approximation ratio $\Omega(n^{1/4})$.*

Proof. We first describe the example. There are m machines where m is greater than a sufficiently large constant such that $\exp(-m/16) < 1/m^8$. There are two types of jobs.

- Type-1: There are $2m^2$ jobs, and each job has size 1 with probability $1/m$, and 0 otherwise.
- Type-2: There are $m^4/4$ jobs, and each job has size m^2 with probability $1/m^3$, and 0 otherwise.

Note that all jobs have the same expected size. Therefore, SEPT rule can schedule jobs in arbitrarily order. Suppose it first schedules Type-1 jobs. By applying a standard Chernoff bound (for example, Theorem 20 with $\mu = 2m$ and $\delta = 1/2$), with a probability of at least $1 - \exp(-m/4) \geq 1 - 1/m^8$, at least m jobs will have size 1, thereby delaying all Type-2 jobs after time 1. Hence the total completion time of SEPT will be $\Omega(m^4)$ in expectation.

In contrast, suppose the adversary schedule Type-2 jobs first. The adversary can learn at an infinitesimally small time, say $1/m^4$, the empty machines. Let \mathcal{E} denote the event that the number of such machines is at least $m/2$. Observe that $\Pr[\mathcal{E}] \geq 1 - \exp(-m/16) \geq 1 - 1/m^8$ by Theorem 20 with $\mu = m/4$ and $\delta = 1$. Since the total job size is at most $O(m^6)$ in all cases, and the total number of jobs is $O(m^4)$, the expected total completion time of the adversary in the event of $\neg\mathcal{E}$ is at most $O(m^6 \cdot m^4 \cdot \frac{1}{m^8}) = O(m^2)$. Now consider the case that the event \mathcal{E} occurs. The adversary distributes Type-1 jobs evenly on the empty machines it

discovered at an infinitesimally small time. Since there are at least $m/2$ empty machines, and there are $2m^2$ Type-1 jobs to schedule, no machine is assigned more than $O(m)$ Type-1 jobs. Hence, even in the worst case where all Type-1 jobs have size 1, every Type-1 job is completed by time $O(m)$. In sum, we have shown the adversary's total completion time is $O(m^3)$ in expectation. Since $n = \Theta(m^4)$, the gap follows. ◀

3 Intuitive Overview of the Design and Analysis of the Algorithm LS

We now give an informal overview of the intuition behind the intertwined design and analysis of the algorithm LS (occasionally oversimplifying some issues).

The initial starting point is the way in which we estimate the total completion time. Let τ_k be the time that the $\frac{n}{2^k}$ th to last job completes. Let G_k be those jobs that complete between τ_{k-1} and τ_k . By rounding down the completion times in G_k to τ_{k-1} we obtain an estimate $\sum_k \tau_k n/2^k$ of the total completion time that is accurate within a constant factor. To see this note that the decrease in total completion time for the $n/2^k$ jobs in G_k can be charged to a $[\tau_{k-1}, \tau_k]$ portion of the completion times of the $n/2^k$ jobs that complete after τ_k . It will be convenient to consider job starting times, instead of job completion times. This is, without any real loss of generality, as the sum of the starting times differs from the sum of the completion times by only the sum of the processing times, and the expected sum of the processing times is the same for all algorithms. Then intuitively our algorithm needs to determine the jobs in G_k , which roughly one would expect should be the $n/2^k$ jobs in positions $[n - n/2^{k-1}, n - n/2^k]$ in the algorithm's list, so that these jobs are all likely to start by a deadline τ_k that is as early as possible. Let us for the moment assume that the algorithm knows the "correct" value of the deadline τ_k . The algorithm must then solve the following informal subproblem:

Key Subproblem (k, τ): Given a cardinality k and a deadline τ , which set $E_{k,\tau}$ of $n/2^k$ jobs should be excluded so as to maximize the probability that the remaining set $A_{k,\tau}$ of $n - n/2^k$ jobs can all start by time τ ?

We give an algorithm SPLIT for selecting $E_{k,\tau}$. We then show that if SPLIT isn't likely to start all jobs in $A_{k,\tau}$ by deadline τ_k then the optimal adaptive algorithm likely has a comparable number of jobs unfinished by deadline τ/Δ . Here Δ is a parameter that we will eventually set to $m + \log n$. This relaxed deadline contributes a Δ factor to the approximation ratio. It will be convenient to call a job small if it has size at most τ/Δ , and call a job big otherwise.

Our algorithm SPLIT certainly should exclude those jobs with high expected processing time. Here all expected processing times will be conditioned on the fact that the job is small, since all big jobs are equally bad for the optimal adaptive algorithm. Our lower bound for SEPT suggests that we should also exclude those jobs that are most likely to be big as these jobs are the ones most likely to clog up the machines. The algorithm SPLIT splits the $n/2^k$ exclusions in $E_{k,\tau}$ equally between the $n/2^{k+1}$ jobs with the highest expected processing times, the $n/2^{k+1}$ jobs with highest probability of being big. The algorithm SPLIT then list schedules the jobs in $A_{k,\tau}$ in an arbitrary order.

The key part of our analysis of SPLIT, and of almost all algorithm analyses of such stochastic problems, is lower bounding optimal. Here we are able to lower bound optimal using the same exact two job characteristics, the probability that a job is big and the expected size, that SPLIT uses. The analysis is split into two cases. The first case is when the aggregate expected size jobs in $A_{k,\tau}$ is at least $\tau/2$. In this case, there is sufficient probability mass on small sizes (this is where we need that Δ is sufficiently large) so that

a standard lower tail bound can be used to show with high probability the aggregate size of the small jobs is close to expectation. The result is then established using the fact that the aggregate sizes divided by m is a lower bound for optimal. The second case is when the aggregate expected size of jobs in $A_{k,\tau}$ is at most $\tau/2$. Then with high probability the small jobs from $A_{k,\tau}$ don't have sufficient aggregate size to keep one machine busy until time τ . Thus if SPLIT has all machines busy at time τ , then SPLIT must have seen at least m big jobs. But as SPLIT excluded the jobs most likely to be big, the optimal algorithm also likely saw m big jobs, and thus still have all machines busy at time τ/Δ .

It is natural to try to extend the algorithm SPLIT to create a list for LS by first picking in arbitrary order the jobs in $\mathcal{J} - E_{1,\tau_1}$, which are intuitively the $n/2$ jobs most likely startable by τ_1 , then picking in arbitrary order the jobs in $\mathcal{J} - E_{1,\tau_1} - E_{2,\tau_2}$, which are intuitively the $n/4$ jobs in the set of $3n/4$ jobs most likely startable by τ_2 that were not previously picked, and on the phase k , picking in arbitrary order the jobs in $\mathcal{J} - \cup_{i \leq k} E_{i,\tau_i} = \cap_{i \leq k} A_{i,\tau_i}$. There are two difficulties with this approach. We end up surmounting both difficulties in a similar fashion.

The first difficulty is that we do not know a priori the "right" values for the τ_k 's. Using standard transformations we can without loss of generality assume that the range of possible times is polynomially bounded, and that we can restrict our attention to τ_k being one of the logarithmically many times that are an integer power of two. We then modify our solution $E_{k,\tau}$ to the subproblems (k, τ) by excluding $n/(2^k \log n)$ jobs, instead of $n/2^k$. Again the exclusions in $E_{k,\tau}$ are split equally between jobs that have the largest expected sizes, and those that are most likely to be big. Let the excluded set $E_k = \cup_i E_{k,2^i}$ be the union of the excluded sets for various possible values of τ_k . We could then construct our list by first picking in arbitrary order the jobs in $\mathcal{J} - E_1$, then picking in arbitrary order the jobs in $\mathcal{J} - E_1 - E_2$, and on the phase k , picking in arbitrary order the jobs in $\mathcal{J} - \cup_{i \leq k} E_i$. Because E is the union of essentially all possible E_{k,τ_k} 's, we know that we are excluding the excluded jobs from the subproblem corresponding to the "right" τ_k . The redefinition of the $E_{k,\tau}$'s costs a log factor in our approximation ratio.

The remaining problem with this ordering is the possibility that the excluded sets may not be consistent. For example, a job j such that $j \notin E_1$ and $j \in E_2$ is an inconsistency as it is not possible to schedule j after τ_2 and before τ_1 . To solve this we let the excluded set $E'_k = \cup_{i \geq k} E_i$ be the union of the previously defined excluded sets for later times. The algorithm LS then constructs its list by first picking in arbitrary order the jobs in $\mathcal{J} - E'_1$, then picking in arbitrary order the jobs in $\mathcal{J} - E'_1 - E'_2$, and on phase k , picking in arbitrary order the jobs in $\mathcal{J} - \cup_{i \leq k} E'_i$. Because a job is excluded in E'_k if it is in any later excluded set E_i , we know that there will be no inconsistencies. This also guarantees the hereditary condition that $E'_{k+1} \subseteq E'_k$, which means that earlier scheduled jobs are not in later excluded sets. Because the size of the sets E_k are geometrically decreasing, these additional exclusions don't change the size of the excluded sets by more than a constant factor.

4 Algorithm LS

In this section we more formally describe the list scheduling algorithm LS, and introduce some notation that will be needed in the analysis. To aid in our later analysis, we describe the algorithm in terms of the complements of the excluded sets discussed in the last section. Recall $A_{k,\ell}$ is the complement of the excluded set $E_{k,\ell}$, and taking the complement of the union is equivalent to taking the intersection of the complements.

We assume that the number of machines $m \geq 2$. We show in Lemma 2 that we can assume

without loss of generality that all possible job sizes are in the range $[1, n^{10}]$. Intuitively, if a job is sufficiently small, then it can change the total completion time objective by very little. The upper bound then follows by noting that at least two jobs have to become big to clog up the machines for a long period of time, and the probability that this happens is quite small. The proof of Lemma 2 can be found in Section 5.1. Let Δ be the smallest integer greater than $1000 \max\{m, \log n\}$ that is a power of two.

► **Lemma 2.** *Suppose we have a nonanticipatory fixed-priority algorithm that is α -approximate for the simplified instances of n jobs where every job j is instantiated to a size between 1 and n^{10} , i.e. $1 \leq P_j \leq n^{10}$ for all jobs j . Then one can get a $O(\alpha)$ -approximation for an arbitrary instance consisting of n jobs.*

We now introduce our algorithm. For intuition guiding the development of the algorithm, we refer the reader to the overview of the algorithm given in Section 3.

Algorithm LS ($k \in [\log n]$, and $\ell \in [12 \log n]$)

1. For each pair of k, ℓ , compute $A_{k,\ell}$ as follows. Let $\tau := 2^\ell$. Let $A_{k,\ell}$ be the intersection of the following two sets $A_{k,\ell}^h$ and $A_{k,\ell}^v$:
 - Let $A_{k,\ell}^h$ be the $n - \frac{n}{24 \cdot 2^k \cdot \log n}$ jobs with the smallest $h_{j,\ell}$ values where $h_{j,\ell} := \Pr[P_j \geq 2^\ell / \Delta]$.
 - Let $A_{k,\ell}^v$ be the $n - \frac{n}{24 \cdot 2^k \cdot \log n}$ jobs with the smallest $v_{j,\ell}$ values where $v_{j,\ell} := \sum_{s < 2^\ell / \Delta} s \cdot \Pr[P_j = s]$.
2. Define $A_k := \bigcap_{\ell \in [12 \log n]} A_{k,\ell}$.
3. Define $A'_k := \bigcap_{k \leq k' \leq \log n} A_{k'}$.
4. Consider k in increasing order. For each k , schedule jobs in $A'_k \setminus A'_{k-1}$ in an arbitrary but fixed order assigning jobs to any available machine.

The following Lemmas are immediate from the algorithm's description, and will be useful for our analysis.

► **Lemma 3.** *It holds that*

- For all k, ℓ , $n - \frac{n}{12 \cdot 2^k \cdot \log n} \leq |A_{k,\ell}| \leq n - \frac{n}{24 \cdot 2^k \cdot \log n}$.
- For all k , $n - \frac{n}{2^k} \leq |A_k| \leq n - \frac{n}{24 \cdot 2^k \cdot \log n}$.
- For all k , $n - \frac{n}{2^{k-1}} \leq |A'_k| \leq n - \frac{n}{24 \cdot 2^k \cdot \log n}$.
- For all k , $A'_k \subseteq A'_{k+1}$.

5 Analysis

This section is devoted to proving Theorem 4.

► **Theorem 4.** *The algorithm LS is $O(\log^2 n + m \log n)$ -approximate for scheduling n stochastic jobs non-preemptively on m identical machines with the objective of minimizing the total completion time in expectation.*

Our analysis is based on Lemma 8 that states how the solution to the subproblem parameterized by k, ℓ can be charged to the adversary's cost. That is, we will show if the algorithm cannot start all jobs in A_k , which is a subset of $A_{k,\ell}$, by a deadline $\tau = 2^\ell$, thereby delaying $n - |A_k|$ jobs after τ , then the adversary is more likely to delay a comparable number of jobs after time τ / Δ .

To formally state Lemma 8, we need to introduce some notation. Let $L(J')$ denote the *earliest* time when a machine becomes available after starting *all* jobs in J' ; here the associated algorithm is implicitly given. Note that $L(J')$ depends on the realized processing time of jobs. Let A_k^* denote the $n - \frac{n}{24 \cdot 2^k \log n}$ jobs the adversary starts the earliest; recall that $|A_{k,\ell}^h| = |A_{k,\ell}^v| = n - \frac{n}{24 \cdot 2^k \log n}$, and $|A_{k,\ell}| \geq n - \frac{n}{12 \cdot 2^k \log n}$. Note that A_k^* could be stochastic while $A_{k,\ell}$ is deterministic.

The quantity $L^W(J')$ is defined similar to $L(J)$, but for the the worst *anticipatory* list scheduling algorithm W . So W knows the jobs sizes and it maximizes the earliest time when a machine becomes available after starting all the jobs in J' . Obviously LS performs better than the worst anticipatory algorithm. Lemma 5, Lemma 6, and Lemma 7 state straightforward properties of these times. The proof of Theorem 4 follows by application of Lemma 8, Lemma 5, Lemma 6 and basic algebra. The cornerstone of the analysis is the proof of Lemma 8, which we postpone until the end of the section.

► **Lemma 5.** *The function $L^W(\cdot)$ is monotone, i.e., for any realization of job sizes and any two sets of jobs, $J' \subseteq J$, we have $L^W(J') \leq L^W(J)$.*

► **Lemma 6.** *For all k and ℓ and for any realization of job sizes, $L(A'_k) \leq L^W(A'_k) \leq L^W(A_k) \leq L^W(A_{k,\ell})$.*

Proof. This follows from the the fact that $A'_k \subseteq A_k \subseteq A_{k,\ell}$, and that LS is a list scheduling algorithm. ◀

► **Lemma 7.** *For all k , $L(A_k^*) \leq L(A_{k+1}^*)$.*

Proof. By definition of A_k^* , we know that $A_k^* \subseteq A_{k+1}^*$. Then, the lemma is immediate since the earliest time when a machine becomes available after the optimal scheduler starts all jobs in A_k^* can be only smaller than the analogously defined time after the same optimal scheduler starts all jobs in A_{k+1}^* . ◀

We now formally state our key lemma.

► **Lemma 8.** *For all k , we have $\Pr[L^W(A_k) \geq 2^\ell] \leq \Pr[L(A_k^*) \geq 2^\ell / \Delta] + O(\frac{1}{n^{12}})$.*

Before proving Lemma 8, we show how it implies Theorem 4.

Proof of Theorem 4. Since all jobs have sizes at most n^{10} , the maximum total completion time can be at most n^{12} . Hence we will proceed with our analysis ignoring the small additive term in the right-hand-side of Lemma 8 since it will add only 1 to the total completion time in expectation, and all jobs have sizes at least 1. Note that it suffices to bound $\mathbb{E} \sum_j S_j$ where S_j is j 's starting time. This is because the algorithm's cost is $\sum_j S_j$ plus $\sum_j P_j$, and $\mathbb{E} \sum_j P_j$ is a clear lower bound to the adversary as we can observe in Lemma 15. We let $\mathbf{1}[\mathcal{E}]$ be an indicator variable that is 1 if the event \mathcal{E} occurs and 0 otherwise, for some event \mathcal{E} .

$$\begin{aligned}
\sum_j S_j &\leq \sum_j \sum_{\ell \geq 0} 2^{\ell+1} \cdot \mathbf{1}[S_j \geq 2^\ell] \leq \sum_j \sum_{\ell \geq \log \Delta} 2^{\ell+1} \cdot \mathbf{1}[S_j \geq 2^\ell] + O(\Delta n) \\
&= \sum_{k \geq 1} \sum_{j \in A'_k \setminus A'_{k-1}} \sum_{\ell \geq \log \Delta} 2^{\ell+1} \cdot \mathbf{1}[S_j \geq 2^\ell] + O(\Delta n) \\
&\leq \sum_{k \geq 1} |A'_k \setminus A'_{k-1}| \cdot \sum_{\ell \geq \log \Delta} 2^{\ell+1} \cdot \mathbf{1}[L(A'_k) \geq 2^\ell] + O(\Delta n) \\
&\leq \sum_{k \geq 1} O\left(\frac{n}{2^k}\right) \cdot \sum_{\ell \geq \log \Delta} 2^\ell \cdot \mathbf{1}[L(A'_k) \geq 2^\ell] + O(\Delta n) \quad [\text{Lemma 3}]
\end{aligned}$$

The first inequality follows since for some ℓ , $2^\ell \leq S_j < 2^{\ell+1}$. By taking the expectation on both sides, we have

$$\begin{aligned} \mathbb{E} \sum_j S_j &\leq \sum_{k \geq 1} O\left(\frac{n}{2^k}\right) \sum_{\ell \geq \log \Delta} 2^\ell \cdot \Pr[L(A'_k) \geq 2^\ell] + O(\Delta n) \\ &\leq \sum_{k \geq 1} O\left(\frac{n}{2^k}\right) \sum_{\ell \geq \log \Delta} 2^\ell \cdot \Pr[L^W(A_k) \geq 2^\ell] + O(\Delta n) \\ &\leq \sum_{k \geq 1} O\left(\frac{n}{2^k}\right) \sum_{\ell \geq \log \Delta} 2^\ell \cdot \Pr[L(A_k^*) \geq 2^\ell / \Delta] + O(\Delta n) \\ &\leq \sum_{k \geq 1} O\left(\frac{n}{2^k}\right) \sum_{\ell \geq 1} \Delta \cdot 2^\ell \cdot \Pr[L(A_k^*) \geq 2^\ell] + O(\Delta n) \\ &\leq 2 \sum_{k \geq 2} O\left(\frac{n}{2^k}\right) \sum_{\ell \geq 1} \Delta \cdot 2^\ell \cdot \Pr[L(A_k^*) \geq 2^\ell] + O(\Delta n) \end{aligned}$$

The second and third inequalities are due to Lemma 6 and Lemma 8, respectively. In the last inequality, we used the fact $L(A_2^*) \geq L(A_1^*)$, which follows from Lemma 7. We can charge $O(\Delta n)$ to the optimal cost since the nonanticipatory optimal solution must have total expected completion time at least $\sum_j \mathbb{E}P_j \geq n$, and our goal is to show a $O(\Delta \log n)$ -approximation.

To upper bound the remaining terms, we lower bound OPT as follows.

$$\text{OPT} \geq \sum_{k \geq 2} |A_{k+1}^* \setminus A_k^*| \cdot L(A_k^*) \geq \Theta(1) \cdot \sum_{k \geq 2} O\left(\frac{n}{2^k \log n}\right) \sum_{\ell} 2^\ell \cdot \mathbf{1}[L(A_k^*) \geq 2^\ell]$$

The first inequality follows since no job in $A_{k+1}^* \setminus A_k^*$ starts before time $L(A_k^*)$. By taking the expectation on this equation and combining it with the above equation, we conclude that our algorithm is $O(\Delta \log n)$ -approximation, deriving Theorem 4. \blacktriangleleft

We are now ready to prove the key lemma.

Proof of Lemma 8. We will be concerned with the probability that the optimal adaptive algorithm cannot start $n' := n - \frac{n}{24 \cdot 2^k \log n}$ jobs before time τ/Δ . We will say that a job j is big if its realized size $P_j \geq \tau/\Delta$, and say the job is small otherwise. We consider two cases depending on the volume of small jobs. Fix k and ℓ . For notational simplicity, let $\tau := 2^\ell$.

Case A: $\sum_{j \in A_k} v_j \geq \tau/2$: We first show in Lemma 9 and Lemma 10 that the aggregate size of the first n' small jobs that the optimal adaptive algorithms starts is likely at least $\tau/4$. Lemma 11 then shows that this is sufficient volume so that the optimal adaptive algorithm cannot start n' jobs before time τ/Δ .

To make this formal, we define a sequence of random variables $\{X_q\}$ where X_q refers to the "small" size of the q th earliest job that is started by the optimal adaptive algorithm – X_q is set to P_j if $P_j < \tau/\Delta$, otherwise 0. Also let Y_q be the 0-1 random variable that becomes 1 if the q th earliest job that the adversary starts becomes large, otherwise 0.

► **Lemma 9.** $\sum_{q \in [n']} \mathbb{E}[X_q] \geq \tau/2$.

Proof. This observation immediately follows from the fact that $A_k \subseteq A_{k,\ell} \subseteq A_{k,\ell}^v$, and $A_{k,\ell}^v$ consists of n' jobs with the smallest $v_{j,\ell} := \sum_{s < \tau/\Delta} s \cdot \Pr[P_j = s]$ values. \blacktriangleleft

► **Lemma 10.** $\Pr[\sum_{q \in [n']} X_q \leq \tau/4] \leq \frac{1}{n^{1/2}}$.

Proof. To apply Theorem 20, we scale down X_q by τ/Δ . Recall that $X_q \leq \tau/\Delta$ and $\Delta \geq 1000 \max\{m, \log n\}$. By using Theorem 20 with $\mu \geq \frac{\tau}{2}/\frac{\tau}{\Delta} \geq \frac{\Delta}{2}$ and $\epsilon \geq 1/2$, we derive that the probability is at most $\exp(-\epsilon^2\mu/2) \leq \exp(-\Delta/16) \leq 1/n^{12}$. ◀

► **Lemma 11.** *If $\sum_{q \in [n']}$ $X_q \geq \tau/4$, then $L(A_k^*) \geq \tau/\Delta$.*

Proof. For the sake of contradiction, suppose that $L(A_k^*) \leq \tau/\Delta$. Since each small job has size at most τ/Δ , a machine can be busy until time $2\tau/\Delta$ due to small jobs that are started by time τ/Δ . Hence it must be the case that $\sum_{q \in [n']}$ $X_q \leq m \cdot 2\tau/\Delta < \tau/4$, which is a contradiction. ◀

Case B: $\sum_{j \in A_k} v_j \leq \tau/2$: We show in Lemma 12 that the algorithm W likely didn't start enough small jobs to even fill up one machine until time τ . Lemma 13 then shows that it must be the case that the algorithm W then must have started m big jobs before time τ . Lemma 14 then shows that the optimal adaptive algorithm must have started m big jobs before time τ/Δ and before starting n' jobs. Thus the optimal adaptive likely cannot finish n' jobs before time τ/Δ .

To make this formal, let X'_j denote job j 's small size. That is, X'_j is set to P_j if $P_j \leq \tau/\Delta$, otherwise 0. Let Y'_j be the 0-1 random variable that becomes 1 if job j becomes large, otherwise 0. The difference between X_q and X'_j (likewise between Y_q and Y'_j) is that X'_j is concerned with the size of a fixed job j while X_q is concerned with the size of the q_{th} earliest job the adversary starts – the q_{th} job can change since the adversary is not necessarily a fixed-priority scheduler. By applying a concentration inequality, we can show,

► **Lemma 12.** $\Pr[\sum_{j \in A_k} X'_j \geq \tau] \leq 1/n^{12}$.

Proof. We scale down X_q by τ/Δ . By applying Theorem 20 with $\mu \leq \frac{\tau}{2}/\frac{\tau}{\Delta} \leq \frac{\Delta}{2}$ and $\epsilon = \frac{\tau/\Delta}{\mu} - 1 = \frac{\Delta}{\mu} - 1 \geq \frac{\Delta}{2\mu}$, we upper bound the probability by $\exp\left(-\frac{(\Delta/(2\mu))^2\mu}{2(1+(\frac{\Delta}{2\mu}-1)/3)}\right) \leq \exp\left(-\frac{(\Delta/(2\mu))^2\mu}{\Delta/\mu}\right) = \exp(-\Delta/4) \leq 1/n^{12}$. ◀

► **Lemma 13.** *If $\sum_{j \in A_k} X'_j < \tau$ and $L^W(A_k) \geq \tau$, then there must be at least m jobs in A_k with realized sizes at least τ/Δ .*

Proof. For the sake of contradiction, suppose there are less than m big jobs. Then there must exist a machine that is busy until time τ scheduling small jobs, which is a contradiction to the condition $\sum_{j \in A_k} X'_j < \tau$. ◀

► **Lemma 14.** $\Pr[\sum_{q \in [n']}$ $Y_q \geq m] \geq \Pr[\sum_{j \in A_{k,\ell}}$ $Y'_j \geq m] \geq \Pr[\sum_{j \in A_k}$ $Y'_j \geq m]$.

Proof. Notice that A_k is a subset of $A_{k,\ell}^h$ with $\ell = \log_2(\tau/\Delta)$. Since $A_{k,\ell}^h$ consists of n' jobs with the smallest $h_{j,\ell} := \Pr[P_j \geq 2^{\ell} = \tau/\Delta]$ values, the probability that the adversary finds at least m big jobs while scheduling the first n' jobs it starts must be as large as the probability that our algorithm finds m big jobs while scheduling jobs in A_k . ◀

This concludes the proof of Lemma 8. ◀

5.1 Proof of the Simplifying Assumption (Lemma 2)

In this section we prove Lemma 2. Due to the space constraints, we defer the proof of Lemma 16, 17, and 18 to the full version of this paper.

We begin with the following simple lower bound on the adversary.

► **Lemma 15.** $\mathbb{E} \text{OPT} \geq \mathbb{E} \sum_j P_j$.

Motivated by this lower bound, from now on we assume w.l.o.g. that $\mathbb{E} \sum_j P_j = 1$ by scaling jobs sizes uniformly.

We now show that one can assume that every job is instantiated to a size at least $1/n^2$. Let \mathcal{I}^1 be the instance obtained from the original instance $\mathcal{I}^0 := \mathcal{I}$ by replacing P_j with $P_j + 1/n^2$. We show that the optimal completion time can only double in the transition from \mathcal{I}^0 to \mathcal{I}^1 . Let $\text{OPT}(\mathcal{I})$ denote the adversary or its objective on instance \mathcal{I} .

► **Lemma 16.** $\mathbb{E} \text{OPT}(\mathcal{I}^1) \leq 2 \cdot \mathbb{E} \text{OPT}(\mathcal{I}^0)$.

► **Lemma 17.** *Given an algorithm A^1 for \mathcal{I}^1 , one can derive an algorithm A^0 for \mathcal{I}^0 with the same expected total completion time or smaller.*

Hence assuming all jobs have sizes at least $1/n^2$ only loses factor 2 in the approximation ratio. Now we argue that if a job is instantiated to have a very large size, we can ignore such a bad case since it contributes to the algorithm's cost very little. To simplify our argument, we will assume that our algorithm is the *worst anticipatory fixed-priority* algorithm. That is, the worst algorithm does the following: it observes each job's realized size, and finds the worst ordering between jobs in J such that assigning each job to the earliest available machine according to the ordering maximizes the total completion time. If we can show that the event that there is a job that has a huge size can contribute to the expected total completion time by only a fraction of $\mathbb{E} \sum_j P_j = 1$, then we will be able to ignore such an event. Let **BAD** denote the worst algorithm we will consider, or its total completion time depending on the context. In the following, $\text{BAD}(w)$ denote **BAD**'s objective when an outcome (realization of job sizes) w occurs. Intuitively, **BAD** can have a huge total completion time only when at least two jobs are realized to have huge sizes, which can happen with a very small probability. This is where we use the fact $m \geq 2$.

► **Lemma 18.** *Let \mathcal{E} be the event that $\max_j P_j \geq n^8$. Then $\sum_{w \in \mathcal{E}} \text{BAD}(w) \Pr[w] \leq O(1) \mathbb{E} \text{OPT}$.*

6 LP-based Algorithm with Machine Augmentation

► **Theorem 19.** *Suppose $m \geq \frac{36}{\epsilon^2} \log n$. Then there is a polynomial time $O(1)$ -approximation that schedules n stochastic jobs non-preemptively on $(1 + \epsilon)m$ identical machines, when compared against the adversary using m machines, with the goal of minimizing the total completion time in expectation.*

Proof. Let $x_{i,\tau}$ be the probability that the adversary schedules job i at time τ . Let $q_{i,d}$ denote the probability that job i has size no smaller than d . The following LP relaxation is due to [13].

$$\min \sum_{i,\tau} \tau \cdot x_{i,\tau} \quad \text{s.t.} \quad \sum_{\tau \geq 0} x_{i,\tau} \geq 1 \quad \forall i; \quad \sum_{i,\tau \leq t} q_{i,t-\tau} \cdot x_{i,\tau} \leq m \quad \forall t \geq 0; \quad x_{i,\tau} \geq 0 \quad \forall i, \tau \geq 0$$

The objective is the total expected starting time of all jobs. The first constraints say that each job must be scheduled. The second constraints ensure that at any time at most

m machines are used. These are valid constraints due to the nonanticipatory nature of the adversary: job i 's size is realized independent of when it is started.

We now show a simple algorithm using $m' = (1 + \epsilon)m$ machines. Since $\{x_{i,\tau}\}_\tau$ is a distribution over job i 's starting times, we naturally set i 's starting time S_i to τ with probability $x_{i,\tau}$. We order jobs in increasing order of S_i , and schedule job i on any available machine at time t – we will show that this is always possible with a high probability. If not possible, we switch to an arbitrary fixed-priority algorithm.

We first claim that we only need to consider times $1 \leq t \leq n^5$ in the LP. Unfortunately, we cannot use Lemma 2 here since this LP-based algorithm is not a fixed-priority algorithm. However, we can use most of the simplifying argument in Section 5.1 with small tweak. We can show that one can assume without loss of generality that all jobs have sizes at least 1 and $n \leq \mathbb{E} \sum_j P_j \leq n^2$. Then, we know that any non-idle algorithm has total completion time at most $n^2 \mathbb{E} \max_j P_j \leq n^4$ in expectation. This implies no optimal LP solution schedules a job by more than $1/n$ after time n^5 , i.e. $\sum_{t' \geq n^5} x_{i,t'} \leq 1/n$ for all i . This is because the second constraint of the LP is trivially satisfied for any optimal solution for all times after n^5 . Hence we only need to consider times $1 \leq t \leq n^5$. This proves the LP has a size polynomial in n .

We will show that for each $1 \leq t \leq n^5$, the number of jobs whose intervals $(S_i, S_i + P_i)$ intersect time t is at least m' with a probability of at most $1/n^{23}$; let B_t refer to the bad event with respect to time t . To show $\Pr[B_t] \leq 1/n^{23}$ fix a time $t \in [0, n^5]$. Observe that the probability that job i 's interval intersects time t is $\sum_{i,\tau \leq t} q_{i,t-\tau} \cdot x_{i,\tau}$, and let X_i is the 0-1 random variable that becomes 1 if such an event happens. By the second constraints of the LP, we have $\mathbb{E} \sum_i X_i \leq m$. By the applying Bernstein inequalities (Theorem 21) with $\Delta = m' - m$, $b = 1$, and $V \leq m$, we can upper bound the probability by $\exp(-\frac{\Delta^2}{2V+2b\Delta/3}) \leq \exp(-\frac{\epsilon^2 m}{2+2\epsilon/3}) \leq \frac{1}{n^{12}}$. when $m \geq \frac{36}{\epsilon^2} \log n$ and $\epsilon \leq 1$.

Now consider a fixed job i . The probability job i can be started at time S_i as suggested by the LP is at least $1 - 1/n^{18}$ via a simple union bound over all times between 1 and n^5 . If it is the case, we can charge i 's starting time to the LP cost. Otherwise, we can still charge i 's expected starting time when it starts before time n^{10} to $\mathbb{E}P_i \geq 1$. Now let $\mathcal{E}(q)$ denote the event that job i starts between time n^q and n^{q+1} . Note that for event $\mathcal{E}(q)$ to happen, there must be at least m jobs that have size at least n^{q-2} blocking all m machines. Hence $\Pr[\mathcal{E}(q)] \leq \binom{n}{m} \cdot (\frac{1}{n^{q-4}})^m \leq \frac{1}{n^{m(q-5)}}$; here Markov inequality was used with $\mathbb{E}P_i \leq n^2$. Hence the expected starting time of job i when it is at least n^{10} is at most $\sum_{q \geq 10} n^{q+1} \cdot \Pr[\mathcal{E}(q)] \leq o(1)$ when $m \geq 3$. Again, we can charge this to $\mathbb{E}P_i \geq 1$. ◀

7 Concentration Inequalities

► **Theorem 20** ([6]). *Let the random variables X_1, X_2, \dots, X_n be independent, with $0 \leq X_i \leq 1$ for each i . Let $S_n = \sum X_i$, let $\mu = \mathbb{E}(S_n)$. Then, any $\delta > 0$, $\Pr[S_n \geq (1 + \delta)\mu] \leq \exp(-\frac{\delta^2 \mu}{2(1+\delta/3)})$ and $\Pr[S_n \leq (1 - \delta)\mu] \leq \exp(-\frac{1}{2}\delta^2 \mu)$.*

► **Theorem 21** ([6]). *Let X_1, X_2, \dots, X_n be n independent random variables such that for all $i \in [n]$, $X_i \leq b$. Let $Y = \sum_{i=1}^n X_i$, $\mu := \mathbb{E}[Y]$, and $V := \text{Var}[Y]$. Then it follows that*

$$\Pr[Y - \mu \geq \Delta] \leq \exp(-\Delta^2 / (2V(1 + (b\Delta/3V)))).$$

Acknowledgements We thank Marc Uetz for bringing this problem to our attention, for helpful discussions through the research process, and for his assistance during the writing process.

References

- 1 L. A. Adamic and B. A. Huberman. Zipf's law and the internet. *Glottometrics*, 3:143–150, 2002.
- 2 Foto N. Afrati, Evripidis Bampis, Chandra Chekuri, David R. Karger, Claire Kenyon, Sanjeev Khanna, Ioannis Milis, Maurice Queyranne, Martin Skutella, Clifford Stein, and Maxim Sviridenko. Approximation schemes for minimizing average weighted completion time with release dates. In *FOCS*, pages 32–44, 1999.
- 3 David Easley and Jon Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, New York, NY, USA, 2010.
- 4 W. Horn. Minimizing average flowtime with parallel machines. *Operations Research*, 21:846–847, 2006.
- 5 Blachander Krishnamurthy and Jennifer Wexford. *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.
- 6 Colin McDiarmid. Concentration. In Michel Habib, Colin McDiarmid, Jorge Ramirez-Alfonsin, and Bruce Reed, editors, *Probabilistic Methods for Algorithmic Discrete Mathematics*, volume 16 of *Algorithms and Combinatorics*, pages 195–248. Springer Berlin Heidelberg, 1998.
- 7 N. Megow, M. Uetz, and T. Vredeveld. Models and algorithms for stochastic online scheduling. *Mathematics of Operations Research*, 31(3):513–525, 2006.
- 8 R. H. Möhring, A. S. Schulz, and M. Uetz. Approximation in stochastic scheduling: The power of LP-based priority policies. *Journal of the ACM*, 46:924–942, 1999.
- 9 Michael Pinedo. *Scheduling Theory, Algorithms, and Systems*. Springer, 2008.
- 10 M. H. Rothkopf. Scheduling with random service times. *Management Science*, 12:703–713, 1966.
- 11 A. S. Schulz. Stochastic online scheduling revisited. In B. Yang, D.-Z. Du, and C. Wang, editors, *Combinatorial Optimization and Applications*, volume 5165 of *Lecture Notes in Computer Science*, pages 448–457. Springer, 2008.
- 12 M. Skutella and M. Uetz. Stochastic machine scheduling with precedence constraints. *SIAM Journal on Computing*, 34:788–802, 2005.
- 13 Martin Skutella, Maxim Sviridenko, and Marc Uetz. Stochastic scheduling on unrelated machines. In *STACS*, pages 639–650, 2014.
- 14 Gideon Weiss. Approximation results in parallel machines stochastic scheduling. *Annals of Operations Research*, 26:195–242, 1990.
- 15 Gideon Weiss. Turnpike optimality of Smith's rule in parallel machines stochastic scheduling. *Mathematics of Operations Research*, 17:255–270, 1992.
- 16 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

On Finding the Adams Consensus Tree

Jesper Jansson¹, Zhaoxian Li², and Wing-Kin Sung^{2,3}

- 1 Laboratory of Mathematical Bioinformatics, Institute for Chemical Research, Kyoto University, Gokasho, Uji, Kyoto 611-0011, Japan
E-mail: jj@kuicr.kyoto-u.ac.jp
Funded by The Hakubi Project and KAKENHI grant number 26330014.
- 2 School of Computing, National University of Singapore, 13 Computing Drive, Singapore 117417
E-mail: lizhaoxianfagg@gmail.com, ksung@comp.nus.edu.sg
- 3 Genome Institute of Singapore, 60 Biopolis Street, Genome, Singapore 138672

Abstract

This paper presents a fast algorithm for finding the Adams consensus tree of a set of conflicting phylogenetic trees with identical leaf labels, for the first time improving the time complexity of a widely used algorithm invented by Adams in 1972 [1]. Our algorithm applies the centroid path decomposition technique [9] in a new way to traverse the input trees' centroid paths in unison, and runs in $O(kn \log n)$ time, where k is the number of input trees and n is the size of the leaf label set. (In comparison, the old algorithm from 1972 has a worst-case running time of $O(kn^2)$.) For the special case of $k = 2$, an even faster algorithm running in $O(n \cdot \frac{\log n}{\log \log n})$ time is provided, which relies on an extension of the wavelet tree-based technique by Bose *et al.* [6] for orthogonal range counting on a grid. Our extended wavelet tree data structure also supports truncated range maximum queries efficiently and may be of independent interest to algorithm designers.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory, J.3 Life and Medical Sciences

Keywords and phrases phylogenetic tree, Adams consensus, centroid path, wavelet tree

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.487

1 Introduction

Scientists use *phylogenetic trees* to describe treelike evolutionary history [10, 17, 20, 22]. A *consensus tree* is a phylogenetic tree that reconciles two or more given phylogenetic trees with identical leaf labels but different branching patterns, e.g., obtained from alternative data sets or obtained by resampling during phylogenetic reconstruction or phylogenetic analysis.

The concept of a consensus tree was introduced by Adams in 1972 [1], and the tree constructed by the algorithm in [1] is nowadays referred to as the *Adams consensus tree*. Since conflicting branching information can be resolved in various ways, a number of alternative definitions of consensus trees have been proposed and analyzed in the literature since then; see, e.g., the surveys in [8], Chapter 30 in [10], or Chapter 8.4 in [22]. However, the Adams consensus tree was the only existing consensus tree of any kind for several years and thus gained popularity among the research community early on. It has been implemented in classic phylogenetics software packages such as PAUP* [23] and COMPONENT [18]. Over the decades, many articles in biology have utilized the Adams consensus tree to reach their conclusions; some examples of highly cited ones include [15, 19, 24].

Apart from its historical significance, two useful features of the Adams consensus tree are that it preserves the nesting information common to all the input trees [2] and that it



© Jesper Jansson, Zhaoxian Li, and Wing-Kin Sung;
licensed under Creative Commons License CC-BY
32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).
Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 487–499



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



SYMPOSIUM
ON THEORETICAL
ASPECTS
OF COMPUTER
SCIENCE

does not introduce any new rooted triplet information [8]. Another feature of the Adams consensus tree is its robustness; adding extra copies of any of the input trees will not affect the output [10], whereas the structure of the so-called *majority rule consensus tree* [16] or the *frequency difference consensus tree* [12] may change completely. In addition, the Adams consensus tree is insensitive to the order in which the input trees are provided [1], as opposed to the *greedy consensus tree* [8, 11]. Finally, it may be much more informative than the *strict consensus tree* [21] and the *loose consensus tree* [7] in cases where a few leaves are in the wrong positions in some of the input trees due to noisy data (for an example, refer to Figure 1 in reference [2]).

The original algorithm of [1] for building the Adams consensus tree has a worst-case running time of $O(kn^2)$, where k is the number of input trees and n is the size of the leaf label set [20]. Despite its practical usefulness, its running time has not been improved in the last forty years. The purpose of this paper is to achieve a better time complexity. The algorithm of [1] is reviewed in Section 1.2, and Section 2 shows that its *expected* running time is in fact $o(kn^2)$ for trees generated by some realistic models of evolution. Next, Section 3 gives an improved algorithm whose worst-case running time is $O(kn \log n)$, based on a new way of applying the centroid path decomposition technique [9]. Finally, Section 4 presents an even faster method for the case $k = 2$ with a worst-case running time of $O(n \cdot \frac{\log n}{\log \log n})$, using an extension of the wavelet tree of Bose *et al.* [6] (described in Section 4.2).

1.1 Definitions and notation

We will use the following definitions. A *phylogenetic tree* is a rooted, unordered, leaf-labeled tree such that all leaves have different labels and every internal node has at least two children. Below, phylogenetic trees are called “trees” for short, and every leaf in a tree is identified with its label. All edges in a tree are assumed to be directed from the root to the leaves.

Let T be a tree. The set of all nodes in T and the set of all leaves in T are denoted by $V(T)$ and $\Lambda(T)$, respectively. For any $u, v \in V(T)$, u is called a *descendant of v* and v is called an *ancestor of u* if there exists a (possibly empty) directed path in T from v to u ; if this path is nonempty then we write $u \prec v$ and call u a *proper descendant of v* and v a *proper ancestor of u* . For any $u \in V(T)$, T^u is the subtree of T rooted at u , i.e., the subgraph of T induced by the node u and all of its proper descendants in T . For any $u \in V(T)$, let $Child^T(u)$ be the set of all children of u in T . The *depth* of any $u \in V(T)$, denoted by $depth^T(u)$, is the number of edges on the unique path from the root of T to u . For any nonempty $X \subseteq V(T)$, $lca^T(X)$ is the lowest common ancestor in T of the nodes in X .

For any nonempty $B \subseteq \Lambda(T)$, define the *restriction of T to B* , denoted by $T|B$, as the tree T' with leaf label set B and node set $\{lca^T(\{u, v\}) : u, v \in B\}$ that preserves the ancestor relations from T , i.e., that satisfies $lca^T(B') = lca^{T'}(B')$ for all nonempty $B' \subseteq B$.

Next, let $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$ be any set of trees satisfying $\Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k) = L$ for some leaf label set L . The *Adams consensus tree of \mathcal{S}* [1, 2] is the unique tree T with $\Lambda(T) = L$ for which the following two properties hold:

- For any $A, B \subseteq L$, if $lca^{T_j}(A) \prec lca^{T_j}(B)$ in every $T_j \in \mathcal{S}$ then $lca^T(A) \prec lca^T(B)$.
- For any $u, v \in V(T)$, if $u \prec v$ in T then $lca^{T_j}(\Lambda(T^u)) \prec lca^{T_j}(\Lambda(T^v))$ in every $T_j \in \mathcal{S}$.

See Figure 1 for an example. Importantly, it was proved in [2] that these two properties are satisfied by the output of the algorithm in [1] (reviewed in Section 1.2 below). This means that to prove the correctness of a new algorithm for building the Adams consensus tree, one just needs to show that its output is equal to the output of the algorithm in [1].

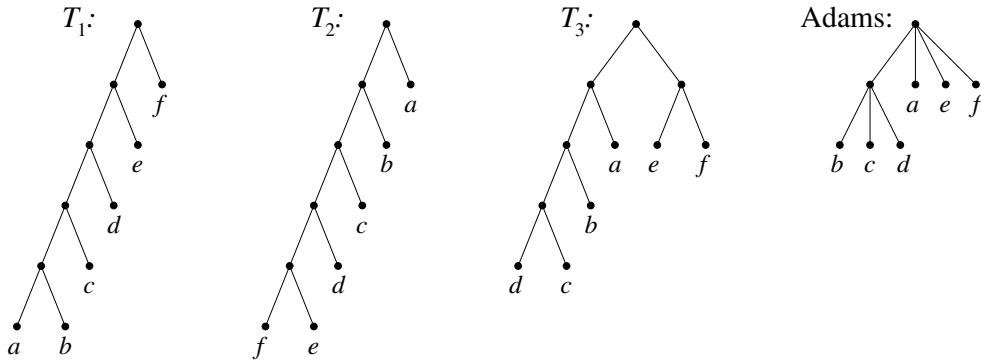


Figure 1 An example. Let $\mathcal{S} = \{T_1, T_2, T_3\}$ as above with $\Lambda(T_1) = \Lambda(T_2) = \Lambda(T_3) = \{a, b, c, d, e, f\}$. The Adams consensus tree of \mathcal{S} is shown on the right. Also note that in this particular example, the Adams consensus tree of \mathcal{S} does not equal the Adams consensus tree of $\{A, T_3\}$, where A is the Adams consensus tree of $\{T_1, T_2\}$.

For any input set \mathcal{S} of trees with identical leaf label sets, we write $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$ and define $L = \Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k)$. To express the time complexity of any algorithm computing the Adams consensus tree of \mathcal{S} , we define $k = |\mathcal{S}|$ and $n = |L|$.

1.2 Previous work

The Adams consensus tree can be computed by the algorithm from [1], which we will now describe. From here on, it will be referred to as `Old_Adams_consensus`. The pseudocode is given in Algorithm 1.

For any tree T , define $\pi(T) = \{\Lambda(T^c) : c \in \text{Child}^T(r), \text{ where } r \text{ is the root of } T\}$. Observe that $\pi(T)$ is a partition of $\Lambda(T)$. Next, for any set of trees $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$ with $\Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k) = L$ for a leaf label set L , define $\pi(\mathcal{S})$ to be the partition of L in which, for every part $B \in \pi(\mathcal{S})$, it holds that $B = \cap_{j=1}^k \Lambda(T_j^{c_j})$ for some child c_j of the root of T_j for each $j \in \{1, 2, \dots, k\}$. Thus, $\pi(\mathcal{S})$ is the product of the partitions $\pi(T_1), \pi(T_2), \dots, \pi(T_k)$. As an example, in Figure 1, we have $\pi(T_1) = \{\{a, b, c, d, e\}, \{f\}\}$, $\pi(T_2) = \{\{a\}, \{b, c, d, e, f\}\}$, $\pi(T_3) = \{\{a, b, c, d\}, \{e, f\}\}$, and $\pi(\mathcal{S}) = \{\{a\}, \{b, c, d\}, \{e\}, \{f\}\}$.

To compute $\pi(\mathcal{S})$, one can apply Procedure `Compute_partition` in Algorithm 2. It encodes each $\ell \in L$ by a vector of length k whose j th entry $m_j(\ell)$ (for $j \in \{1, 2, \dots, k\}$) indicates which child of the root of T_j is an ancestor of ℓ . In this way, any two leaf labels

Algorithm 1 Algorithm `Old_Adams_consensus`, adapted from [1].

Algorithm `Old_Adams_consensus`

Input: A set $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$ of trees with $\Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k)$.

Output: The Adams consensus tree of \mathcal{S} .

- 1: **if** T_1 has only one leaf **then** let $T := T_1$; /* Base case of the recursion */
 - 2: **else** /* General case of the recursion */
 - 3: $\pi := \text{Compute_partition}(\mathcal{S})$;
 - 4: **for** every $B \in \pi$ **do** $T_B := \text{Old_Adams_consensus}(\{T_1|B, T_2|B, \dots, T_k|B\})$;
 - 5: Create a tree T whose root is the parent of the root of T_B for every $B \in \pi$;
 - 6: **end if**
 - 7: **return** T ;
-

Algorithm 2 Procedure `Compute_partition`.

 Procedure `Compute_partition`
Input: A set $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$ of trees with $\Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k) = L'$.

Output: A list of all parts in the partition $\pi(\mathcal{S})$ of L' .

- 1: Fix an arbitrary left-to-right ordering of the children of the root of every $T_j \in \mathcal{S}$ and denote the i th child (according to this ordering) of the root of T_j by c_j^i ;
 - 2: **for** every $\ell \in L'$ **do** compute the vector $(m_1(\ell), m_2(\ell), \dots, m_k(\ell))$, where for $j \in \{1, 2, \dots, k\}$, $m_j(\ell) = i$ if and only if ℓ is a descendant of c_j^i in T_j ;
 - 3: Put the vectors $(m_1(\ell), m_2(\ell), \dots, m_k(\ell))$ for all $\ell \in L'$ in a list \mathcal{W} and sort \mathcal{W} ;
 - 4: Do a single scan of \mathcal{W} to identify the parts in $\pi(\mathcal{S})$ and **return** them;
-

in L belong to the same part in $\pi(\mathcal{S})$ if and only if their vectors are identical. By sorting the list \mathcal{W} of all vectors and scanning \mathcal{W} to find vectors that are identical, the parts in $\pi(\mathcal{S})$ are obtained.

`Old_Adams_consensus` first computes $\pi(\mathcal{S})$. It then recursively constructs the Adams consensus tree of $\{T_1|B, T_2|B, \dots, T_k|B\}$ for each B in $\pi(\mathcal{S})$ and attaches all of them to a newly created common root node. By Theorem 3 in [2], this yields the Adams consensus tree of \mathcal{S} . According to [20], the time complexity of `Old_Adams_consensus` is $O(kn^2)$.

2 Preliminaries

This section reanalyzes the time complexity of `Old_Adams_consensus`. For any $B \subseteq L$, say that B is a *relevant block* if at any point of the algorithm's execution, Step 4 makes a recursive call with $\{T_1|B, T_2|B, \dots, T_k|B\}$ as the argument. Define $\mathcal{B} = \{B : B \text{ is a relevant block}\}$. For every $\ell \in L$, define $\mathcal{B}(\ell) = \{B \in \mathcal{B} : \ell \in B\}$.

► **Lemma 1.** *For every $\ell \in L$, it holds that $|\mathcal{B}(\ell)| \leq \min_{j=1}^k \text{depth}^{T_j}(\ell)$.*

Proof. Step 3 of `Old_Adams_consensus` initially generates a partition π_1 of L , and there exists exactly one relevant block B_1 in π_1 such that $\ell \in B_1$. Then, during the recursive call `Old_Adams_consensus` ($\{T_1|B_1, T_2|B_1, \dots, T_k|B_1\}$), a partition π_2 of B_1 is generated in the same way, and there exists exactly one relevant block B_2 in π_2 such that $\ell \in B_2$. This process is repeated until a relevant block of the form $B_m = \{\ell\}$ is reached and the recursion stops. At any recursion level i , when `Old_Adams_consensus` ($\{T_1|B_i, T_2|B_i, \dots, T_k|B_i\}$) makes a call to `Old_Adams_consensus` ($\{T_1|B_{i+1}, T_2|B_{i+1}, \dots, T_k|B_{i+1}\}$), it always holds that $\text{depth}^{T_j|B_{i+1}}(\ell) \leq \text{depth}^{T_j|B_i}(\ell) - 1$ for all trees $T_j \in \mathcal{S}$. Hence, the number of recursive calls that involve ℓ is upper-bounded by $\min_{j=1}^k \text{depth}^{T_j}(\ell)$. ◀

► **Theorem 2.** *`Old_Adams_consensus` runs in $O(k \cdot \sum_{\ell \in L} \min_{j=1}^k \text{depth}^{T_j}(\ell))$ time.*

Proof. We first explain how to implement the procedure `Compute_partition` to run in $O(k|L'|)$ time, where L' is the leaf label set of its input \mathcal{S} . In Step 2, use the *level ancestor* data structure from [4] as follows: Spend $O(|L'|)$ time to preprocess each $T_j \in \mathcal{S}$ so that the ancestor of any $\ell \in L'$ at depth 1 in T_j can be retrieved in $O(1)$ time. This preprocessing takes $O(k|L'|)$ time, and finding the vectors $(m_1(\ell), m_2(\ell), \dots, m_k(\ell))$ for all $\ell \in L'$ subsequently takes a total of $O(k|L'|)$ time. In Step 3, sort the list \mathcal{W} in $O(k|L'|)$ time by radix sort.

Next, we consider `Old_Adams_consensus`. Before running the algorithm, use the method in Section 8 of [9] to preprocess each $T_j \in \mathcal{S}$ in $O(n)$ time so that $T_j|B$ for any $B \subseteq L$ can be constructed in $O(|B|)$ time. This takes $O(kn)$ time in total. It follows from the

definition of $T_j|B$ in Section 1.1 that for any $A \subsetneq B$, $(T_j|B)|A = T_j|A$ holds, so the same preprocessing works for all recursion levels and does not need to be repeated during recursive calls. Excluding the time required by its recursive calls, the running time of `Old_Adams_consensus` ($\{T_1|B, T_2|B, \dots, T_k|B\}$) then becomes $O(k|B|)$ for each $B \in \mathcal{B}$. In total, the running time of `Old_Adams_consensus`(\mathcal{S}) is $O(kn + \sum_{B \in \mathcal{B}} k|B|) = O(k \cdot \sum_{B \in \mathcal{B}} |B|) = O(k \cdot \sum_{\ell \in L} |\mathcal{B}(\ell)|)$. By Lemma 1, $\sum_{\ell \in L} |\mathcal{B}(\ell)| \leq \sum_{\ell \in L} \min_{j=1}^k \text{depth}^{T_j}(\ell)$. The theorem follows. \blacktriangleleft

Since $|L| = n$ and $\text{depth}^{T_j}(\ell) < n$ for all $\ell \in L$ and $T_j \in \mathcal{S}$, Theorem 2 implies that the worst-case running time of `Old_Adams_consensus` is $O(kn^2)$, as already mentioned in [20]. However, if the average leaf depth is small then the running time will be better. According to Theorem 2, we obtain:

► **Corollary 3.** *If \mathcal{S} is a set of trees with expected average leaf depth α then the expected running time of `Old_Adams_consensus` is $O(kn\alpha)$.*

For example, the expected average leaf depth in a random binary phylogenetic tree with n leaves generated in the Yule-Harding model [5, 14, 20], the uniform model [5, 20], and the activity model [14] (with the activity parameter p set to $\frac{1}{2}$) is $O(\log n)$ [5, 14], $O(n^{1/2})$ [5], and $O(n^{1/2})$ [14], respectively. In these cases, the expected running time of `Old_Adams_consensus` will be $O(kn \log n)$, $O(kn^{1.5})$, and $O(kn^{1.5})$.

3 New algorithm for k input trees

This section gives a more efficient solution for computing the Adams consensus tree of k input trees. The algorithm is called `New_Adams_consensus_k` and its worst-case running time is $O(kn \log n)$.

The main idea is to use the centroid path decomposition technique [9] in a new manner to avoid making recursive calls to “large” subproblems, and treat them iteratively instead. Essentially, by utilizing Lemma 4 below, the algorithm implicitly computes $\pi(\mathcal{S})$ in such a way that the Adams consensus tree can be constructed recursively for all parts in $\pi(\mathcal{S})$, *except for one*. To handle the remaining part, its corresponding Adams consensus tree is constructed iteratively by going down the centroid paths in all the trees in unison and applying Lemma 4 at each level. (As a side note, this kind of “synchronized centroid path traversal” appears to be a novel way of applying the centroid path decomposition technique.) Finally, the Adams consensus tree of \mathcal{S} is assembled by attaching the root of each tree constructed for the parts in $\pi(\mathcal{S})$ to a new root node.

The details of the algorithm are described below, and the pseudocode is listed in Algorithm 3.

Some additional definitions are needed. Recall from [9] that a *centroid path* in a tree T is a path in T of the form $P = \langle p_\alpha, p_{\alpha-1}, \dots, p_1 \rangle$, where the node p_{w-1} for every $w \in \{2, \dots, \alpha\}$ is any child of p_w with the maximum number of leaf descendants, and p_1 is a leaf. Let P be a centroid path in a tree T . For any $u \in V(T)$ such that u does not belong to P but the parent of u does, the subtree T^u is called a *side tree* of P . For any side tree τ of a centroid path starting at the root of a tree T , the property $|\Lambda(\tau)| \leq |\Lambda(T)|/2$ holds.

A *delete* operation on any non-root, internal node u in a tree is the operation of letting all of u ’s children become children of the parent of u , and then removing u and the edge between u and its parent. A *fan tree* is a tree in which either all the leaves are children of the root, or there is just a single leaf.

Algorithm 3 Algorithm `New_Adams_consensus_k`.Algorithm `New_Adams_consensus_k`**Input:** A set $\mathcal{S} = \{T_1, T_2, \dots, T_k\}$ of trees with $\Lambda(T_1) = \Lambda(T_2) = \dots = \Lambda(T_k) = L$.**Output:** The Adams consensus tree of \mathcal{S} .

```

1: if  $T_1$  has only one leaf then
2:    $T := T_1$ ;
3: else
4:   for  $j := 1$  to  $k$  do
5:     Let  $P_j$  be a centroid path in  $T_j$  starting at the root, construct the tree  $T'_j$  based
       on  $P_j$ , and preprocess  $T_j$ ;
6:   end for
7:    $h := 0$ ;
8:   repeat
9:      $h := h + 1$ ;
10:     $X_h := \{x \in L : \text{for some } j \in \{1, 2, \dots, k\}, x \text{ belongs to a fan tree attached to the}$ 
        $\text{root of } T'_j\}$ ;
11:     $\pi_{X_h} := \text{Compute\_restricted\_partition}(\{T'_1, T'_2, \dots, T'_k\}; X_h)$ ;
12:    for  $j := 1$  to  $k$  do  $T'_j := T'_j | (\Lambda(T'_j) \setminus X_h)$ ;
13:  until  $\Lambda(T'_1) = \emptyset$ ;
14:  for  $j := 1$  to  $k$  do
15:    for  $w := 1$  to  $h$  do construct  $T_i|B$  for all  $B \in \pi_{X_w}$ ;
16:  end for
17:  for  $w := h$  downto  $1$  do
18:    for  $B \in \pi_{X_w}$  do  $T_B := \text{New\_Adams\_consensus\_k}(\{T_1|B, T_2|B, \dots, T_k|B\})$ ;
19:    Create a tree  $Q_w$  whose root is the parent of the root of every  $T_B$ ,  $B \in \pi_{X_w}$ ;
20:    if  $w < h$  then attach the root of  $Q_{w+1}$  as a child of the root of  $Q_w$ ;
21:  end for
22:   $T := Q_1$ ;
23: end if
24: return  $T$ ;

```

For each $j \in \{1, 2, \dots, k\}$, let P_j be a centroid path in T_j that starts at the root of T_j . Let T'_j be the tree obtained by taking a copy of T_j and doing a delete operation on every non-root, internal node whose parent does not belong to P_j ; note that by performing all delete operations in top-down order, T'_j can be constructed in $O(n)$ time. Thus, T'_j consists of the centroid path P_j with a collection of fan trees attached to it, and each such fan tree's leaf label set is equal to the leaf label set of one of the side trees of P_j . The T'_j -tree is a useful summary of T_j that enables us to quickly retrieve the leaf label set of any side tree in T_j or to check which side tree in T_j that a specified leaf belongs to in $O(1)$ time.

As in `Old_Adams_consensus` above, `New_Adams_consensus_k` needs to compute the partition $\pi(\mathcal{S})$ of L to determine the branching structure at the top level of the Adams consensus tree. However, for efficiency reasons, it does not compute $\pi(\mathcal{S})$ directly. Instead, it computes a *restricted partition*, defined as follows: For any $X \subseteq L$, let $\pi(\mathcal{S}; X) = \{B \cap X : B \in \pi(\mathcal{S}) \text{ and } |B \cap X| \geq 1\}$. In other words, $\pi(\mathcal{S}; X)$ is the partition $\pi(\mathcal{S})$ restricted to elements in X . Note that $\pi(\mathcal{S}; X)$ may not be a true partition of X as it can be a singleton. To continue the example from Figure 1 in Section 1.2 where we had $\pi(\mathcal{S}) = \{\{a\}, \{b, c, d\}, \{e\}, \{f\}\}$, if $X = \{a, b, c\}$ then $\pi(\mathcal{S}; X) = \{\{a\}, \{b, c\}\}$ and if $X = \{b, c\}$ then $\pi(\mathcal{S}; X) = \{\{b, c\}\}$.

► **Lemma 4.** *Let $X = \{x \in L : \text{for some } j \in \{1, 2, \dots, k\}, x \text{ belongs to a fan tree attached to the root of } T'_j\}$. If $X \neq L$ then $\pi(\{T_1, T_2, \dots, T_k\}) = \pi(\{T'_1, T'_2, \dots, T'_k\}; X) \cup \{L \setminus X\}$, and if $X = L$ then $\pi(\{T_1, T_2, \dots, T_k\}) = \pi(\{T'_1, T'_2, \dots, T'_k\}; X)$.*

Proof. X is also equal to $\{x \in L : \text{for some } j \in \{1, 2, \dots, k\}, x \text{ belongs to a side tree of } P_j \text{ attached to the root of } T_j\}$. Consider any $B \in \pi(\{T_1, T_2, \dots, T_k\})$. If B contains at least one element from X then $B \subseteq X$, and consequently $B \cap X = B$ and $B \in \pi(\{T_1, T_2, \dots, T_k\}; X)$. On the other hand, if B contains no elements from X then B must be equal to $L \setminus X$. Therefore, $\pi(\{T_1, T_2, \dots, T_k\}) \subseteq \pi(\{T_1, T_2, \dots, T_k\}; X) \cup \{L \setminus X\}$ when $X \neq L$, and $\pi(\{T_1, T_2, \dots, T_k\}) \subseteq \pi(\{T_1, T_2, \dots, T_k\}; X)$ when $X = L$.

Next, consider any $B \in \pi(\{T_1, T_2, \dots, T_k\}; X)$. By definition, $B \in \pi(\{T_1, T_2, \dots, T_k\})$. Also, if $X \neq L$ then $L \setminus X$ is nonempty and consists of all leaves that are descendants of the child of the root of T_j that lies on P_j for every $j \in \{1, 2, \dots, k\}$; since all these leaves belong to the same part in $\pi(T_j)$ for each $j \in \{1, 2, \dots, k\}$, we have $L \setminus X \in \pi(\{T_1, T_2, \dots, T_k\})$. Thus, $\pi(\{T_1, T_2, \dots, T_k\}; X) \cup \{L \setminus X\} \subseteq \pi(\{T_1, T_2, \dots, T_k\})$ when $X \neq L$, and $\pi(\{T_1, T_2, \dots, T_k\}; X) \subseteq \pi(\{T_1, T_2, \dots, T_k\})$ when $X = L$.

Finally, $\pi(\{T'_1, T'_2, \dots, T'_k\}; X) = \pi(\{T_1, T_2, \dots, T_k\}; X)$ by the construction of the T'_j -trees. The lemma follows. ◀

We now describe `New_Adams_consensus_k`.

First, for each $j \in \{1, 2, \dots, k\}$, Steps 4–6 build P_j and T'_j and preprocess T_j in $O(n)$ time as in Section 8 of [9] so that for any specified partition π of L , the set of all trees of the form $T_j|B_i$ with $B_i \in \pi$ can be constructed in $O(n)$ total time later on. The algorithm then enters a **repeat**-loop (Steps 8–13) that computes and stores the restricted partition $\pi(\{T'_1, T'_2, \dots, T'_k\}; X_1)$, where X_1 is the subset X of $\Lambda(T'_1)$ ($= \Lambda(T'_2) = \dots = \Lambda(T'_k)$) defined in Lemma 4. By Lemma 4, the parts in $\pi(\{T'_1, T'_2, \dots, T'_k\}; X_1)$ along with $\Lambda(T'_1) \setminus X_1$ yield the partition at the top level of the Adams consensus tree. After that, the leaves belonging to X_1 are removed from all the T'_j -trees. The process is repeated until the T'_j -trees are empty, and each subsequent iteration of the **repeat**-loop mimics the computations at one recursion level in `Old_Adams_consensus` that determine how to further partition the leaves in the set $\Lambda(T'_1) \setminus X_1$. Next, the algorithm constructs $T_j|B$ for every part B previously computed by the **repeat**-loop for all $j \in \{1, 2, \dots, k\}$ (Steps 14–16). Then, the Adams consensus tree Q_w at each level w is built by recursively computing the Adams consensus tree T_B for every part B in $\pi(\{T'_1, T'_2, \dots, T'_k\}; X_w)$ at this level (Step 18), combining the obtained solutions (Step 19), and attaching the Adams consensus tree Q_{w+1} for the part corresponding to $L \setminus X_w$ in Lemma 4 (Step 20). Lastly, the tree Q_1 obtained at the topmost level is returned (Step 24). The correctness follows from Lemma 4 and the correctness of `Old_Adams_consensus`.

The time complexity is given by the next theorem:

► **Theorem 5.** `New_Adams_consensus_k` runs in $O(kn \log n)$ time.

Proof. Denote the time complexity of `New_Adams_consensus_k`($\{T_1|L', T_2|L', \dots, T_k|L'\}$) for any $L' \subseteq L$ by $t(L')$.

We derive a recurrence for $t(L')$ in the following way. Steps 4–6 build P_j and T'_j and preprocess T_j in $O(|L'|)$ time for each $j \in \{1, 2, \dots, k\}$, i.e., in $O(k|L'|)$ time in total. Iteration h of the **repeat**-loop computes a set X_h in Step 10, which takes $O(k|X_h|)$ time by using the T'_j -trees, and the restricted partition $\pi_{X_h} = \pi(\{T'_1, T'_2, \dots, T'_k\}; X_h)$ of X_h in Step 11, which also takes $O(k|X_h|)$ time by using the technique from Procedure `Compute_partition` in Algorithm 2 and the first part of the proof of Theorem 2. To implement Step 12 in $O(k|X_h|)$ time, update each T'_j -tree directly by removing all leaves that belong to X_h as well as any previously internal node that turns into a leaf as a result and contracting any outgoing edge from

a node of degree 1. Constructing all the trees $T_j|B$ in Steps 14–16 takes a total of $O(k|L'|)$ time with the technique from Section 8 of [9]. Finally, for each $w \in \{1, 2, \dots, h\}$, the recursive calls in Step 18 take $\sum_{B \in \pi_{X_w}} t(B)$ time and building Q_w in Steps 19 and 20 takes $O(|X_w|)$ time. In total, the time complexity is $t(L') = O(k|L'|) + \sum_{w=1}^h (O(k|X_w|) + \sum_{B \in \pi_{X_w}} t(B))$.

To solve the recurrence, we use the fact that $\bigcup_{w=1}^h \pi_{X_w}$ is a partition of L' . Write $\pi_{L'} = \bigcup_{w=1}^h \pi_{X_w}$. Then $t(L') = O(k|L'|) + \sum_{B \in \pi_{L'}} t(B)$. Since every part $B \in \pi_{L'}$ is of size at most $|L'|/2$ according to the definition of a side tree of a centroid path, the problem size is reduced by (at least) half for each successive recursive call. Thus, there are $O(\log |L'|)$ recursion levels. The total size of all subproblems in each recursive level is $O(|L'|)$, so each recursion level takes $O(k|L'|)$ time. This gives $t(L') = O(k|L'| \log |L'|)$. ◀

4 New algorithm for two input trees

Here, we present an even faster algorithm for the case $k = 2$. The algorithm is named `New_Adams_consensus_2` and has a worst-case running time of $O(n \cdot \frac{\log n}{\log \log n})$.

4.1 Outline of the algorithm

Consider any recursive call of the form `Old_Adams_consensus` ($\{T_1|B, T_2|B\}$) for some $B \subseteq L$ in the algorithm in Section 1.2. To obtain the partition of the leaves in B , the algorithm will spend $\Omega(|B|)$ time using the procedure `Compute_partition`. A faster method for doing the partitioning is needed to improve the overall running time. First, we observe that by the definition of the algorithm, B always satisfies $B = \Lambda(T_1^u) \cap \Lambda(T_2^v)$ for some pair of nodes $u \in V(T_1)$, $v \in V(T_2)$. This means that successive recursive calls to the algorithm can be specified by pairs of vertices from T_1 and T_2 . Secondly, we observe that the algorithm needs to proceed recursively from (u, v) only to those (u', v') , where $u' \in \text{Child}^{T_1}(u)$ and $v' \in \text{Child}^{T_2}(v)$, for which $|\Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'})| > 0$. Based on these observations, define $Z_{u,v} = \{(u', v') : u' \in \text{Child}^{T_1}(u), v' \in \text{Child}^{T_2}(v), |\Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'})| > 0\}$. We have:

► **Lemma 6.** *Suppose $u \in V(T_1)$ and $v \in V(T_2)$ are given. Let $B = \Lambda(T_1^u) \cap \Lambda(T_2^v)$, $\gamma = \text{lca}^{T_1}(B)$, and $\delta = \text{lca}^{T_2}(B)$. If $|B| > 1$ then $\pi(\{T_1|B, T_2|B\}) = \pi(\{T_1^u|B, T_2^v|B\}) = \pi(\{T_1^\gamma|B, T_2^\delta|B\}) = \{\Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'}) : (u', v') \in Z_{\gamma,\delta}\}$.*

Proof. By definition, $\pi(\{T_1^\gamma|B, T_2^\delta|B\})$ is equal to $\{\Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'}) : u' \in \text{Child}^{T_1}(\gamma), v' \in \text{Child}^{T_2}(\delta), \Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'}) \neq \emptyset\} = \{\Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'}) : (u', v') \in Z_{\gamma,\delta}\}$. ◀

Algorithm `New_Adams_consensus_2` (summarized in Algorithm 4) uses this lemma to compute the Adams consensus tree of $\{T_1^u|B, T_2^v|B\}$ for any two specified nodes $u \in V(T_1)$, $v \in V(T_2)$, where $B = \Lambda(T_1^u) \cap \Lambda(T_2^v)$. (Selecting u and v to be the roots of T_1 and T_2 thus yields the Adams consensus tree of T_1 and T_2 .)

The algorithm works as follows. If $|B| = 1$ then the answer is just the common leaf in $\Lambda(T_1^u) \cap \Lambda(T_2^v)$. Otherwise, the algorithm computes $\gamma = \text{lca}^{T_1}(B)$ and $\delta = \text{lca}^{T_2}(B)$, calls a procedure `Compute_Z` (to be described in Section 4.3) to construct $Z_{\gamma,\delta}$, and then, for every $(u', v') \in Z_{\gamma,\delta}$, computes its corresponding Adams consensus tree $T_{u',v'}$ recursively. The Adams consensus tree of $\{T_1^u|B, T_2^v|B\}$ is obtained by attaching the computed $T_{u',v'}$ -trees to a newly created common root node. Lemma 6 implies that this gives the same output as `Old_Adams_consensus`, so the correctness is guaranteed by the correctness of the latter.

Algorithm 4 Algorithm New_Adams_consensus_2.

Algorithm New_Adams_consensus_2

Input: $u \in V(T_1)$, $v \in V(T_2)$, where T_1, T_2 are two given trees with $\Lambda(T_1) = \Lambda(T_2)$.

Output: The Adams consensus tree of $\{T_1^u|B, T_2^v|B\}$, where $B = \Lambda(T_1^u) \cap \Lambda(T_2^v)$.

```

1: Compute  $c := |\Lambda(T_1^u) \cap \Lambda(T_2^v)|$ ;
2: if  $c = 1$  then
3:   Let  $T$  be a tree consisting of the only common leaf in  $\Lambda(T_1^u) \cap \Lambda(T_2^v)$ ;
4: else
5:   Find the leftmost leaf  $a$  and the rightmost leaf  $a'$  in  $T_1|(\Lambda(T_1^u) \cap \Lambda(T_2^v))$ , and the
   leftmost leaf  $b$  and the rightmost leaf  $b'$  in  $T_2|(\Lambda(T_1^u) \cap \Lambda(T_2^v))$ ;
6:    $\gamma := lca^{T_1}(a, a')$ ;  $\delta = lca^{T_2}(b, b')$ ;
7:    $Z := \text{Compute\_Z}(\gamma, \delta)$ ;
8:   Let  $T$  be a tree consisting of a single root node  $r$ ;
9:   for every  $(u', v') \in Z$  do
10:     $T_{u',v'} := \text{New\_Adams\_consensus\_2}(u', v')$ ;
11:    Attach  $T_{u',v'}$  as a child of  $r$ ;
12:   end for
13: end if
14: return  $T$ ;
```

4.2 Auxiliary data structure for orthogonal range counting on a grid

The time complexity of `New_Adams_consensus_2` is analyzed in Section 4.3 below. It relies on an efficient data structure for orthogonal range counting on a grid, developed in this subsection and summarized in Lemma 8. This data structure is an extension of the wavelet tree-based data structure used by Bose *et al.* [6] for supporting orthogonal range counting queries on a grid. Our extension consists of also supporting *truncated range maximum* (or *minimum*) queries efficiently, where the objective is to report the point with the maximum (or minimum) x-coordinate inside any query rectangle $[1..\ell] \times [s..s']$, if any. Furthermore, we bound the time needed to *construct* the data structure since this is crucial in our application.

Firstly, for smaller grids, we have:

► **Lemma 7.** *Let $N = \{(1, N[1]), \dots, (n, N[n])\}$ be a set of points on an $n \times t$ grid, where $t = O(\log^\epsilon n)$ for any constant ϵ with $0 < \epsilon < 1/2$, such that every column contains exactly one point. We can build a data structure in $O(n)$ time after which: (i) counting the number of points inside any query rectangle $[1..\ell] \times [s..s']$ takes $O(1)$ time; and (ii) reporting the point with the maximum (or minimum) x-coordinate inside any query rectangle $[1..\ell] \times [s..s']$ takes $O(1)$ time.*

Proof. Omitted from the conference version of the paper due to space constraints. ◀

For larger grids, we apply Lemma 7 to obtain:

► **Lemma 8.** *Let $N = \{(1, N[1]), \dots, (n, N[n])\}$ be a set of points on an $n \times n$ grid such that every column contains exactly one point and every row contains exactly one point. We can build a data structure $D(N)$ in $O(n \cdot \frac{\log n}{\log \log n})$ time after which: (i) counting the number of points inside any query rectangle $[x..x'] \times [y..y']$ takes $O(\frac{\log n}{\log \log n})$ time; and (ii) reporting the point with the maximum (or minimum) x-coordinate inside any query rectangle $[x..x'] \times [y..y']$ takes $O(\frac{\log n}{\log \log n})$ time.*

Proof. The basic data structure is the same as in the proof of Lemma 6 in [6], namely a t -ary wavelet tree. Here, we select $t = \log^\epsilon n$ for any $0 < \epsilon < 0.5$.

On the top level, we project the n points into the 2d-space $[1..n] \times [1..t]$ by converting each point $(i, N[i])$ to $(i, N_1[i])$, where $N_1[i] = \lfloor N[i]/(n/t) \rfloor$. We use Lemma 7 to maintain a range query data structure for $\{(i, N_1[i]) : i = 1, \dots, n\}$, and also build a rank data structure that lets us compute $\text{rank}_j(i)$ in $N_1[1..n]$ (here, $\text{rank}_j(i)$ is the number of occurrences of j in $N_1[1..i]$). This data structure can be built in $O(n)$ time. To be precise, we store $\text{rank}_j(i)$ for every i which is a multiple of $\log^2 n$, requiring $O(\frac{nt \log n}{\log^2 n}) = O(n)$ bits space. We also store $\text{rank}_j(i) - \text{rank}_j(\log^2 n \lfloor \frac{i}{\log^2 n} \rfloor)$ for every i which is a multiple of $\frac{\log n}{\log \log n}$, requiring $O(t \frac{n \log \log n}{\log n} \log \log nt) = O(n)$ bits. We precompute a table $\text{occtable}(x_1, \dots, x_\ell, j)$ that stores the number of occurrences of j in x_1, \dots, x_ℓ , where $1 \leq x_i \leq t$, $1 \leq j \leq t$, $\ell \leq \frac{\log n}{\log \log n}$. This table has $o(n)$ entries and can be computed in $o(n)$ time. By taking $x = \lfloor \frac{i}{\log^2 n} \rfloor \log^2 n$ and $y = \lfloor \frac{i \log \log n}{\log n} \rfloor \frac{\log n}{\log \log n}$, we have $\text{rank}_j(i) = \text{rank}_j(x) + (\text{rank}_j(y) - \text{rank}_j(x)) + \text{occtable}(N[i - y + 1], \dots, N[i], j)$, which can be computed in constant time.

On the second level, based on $N_1[]$, we partition the n points into t point sets $N_{2,1}, \dots, N_{2,t}$. The set $N_{2,j}$ contains all the points where $N_1[i] = j$. Let $n_{2,j} = |N_{2,j}|$. Every point $(i, N[i])$ in $N_{2,j}$ is projected into the 2d-space $[1..n_{2,j}] \times [1..t]$. Suppose the rank of i is r among all the x-coordinates of the points in $N_{2,j}$. Then, $(i, N[i])$ is converted to $(r, N_{2,j}[r])$ where $N_{2,j}[r] = \lfloor (N[i] - (n/t)j)/(n/t^2) \rfloor$. We use Lemma 7 again to maintain a range query data structure for these $n_{2,j}$ points. We also build a rank data structure for $N_{2,j}[1..n_{2,j}]$ in $O(n_{2,j})$ time. We continue the process recursively and build the above data structures on each level. Since there are $\log_t n$ levels, the entire data structure can be constructed in $O(n \log_t n)$ time.

Next, given any query rectangle $[\ell_1.. \ell_2] \times [s_1..s_2]$, we proceed in a similar manner as in [6]. Let $z_1 = \lceil s_1/(n/t) \rceil$ and $z_2 = \lfloor s_2/(n/t) \rfloor$. The query is partitioned into: (1) $[\ell_1.. \ell_2] \times [s_1..(n/t)z_1]$; (2) $[\ell_1.. \ell_2] \times [(n/t)z_1 + 1..(n/t)z_2]$; and (3) $[\ell_1.. \ell_2] \times [(n/t)z_2 + 1..s_2]$.

Query (2) is equivalent to the query $[\ell_1.. \ell_2] \times [z_1 + 1..z_2]$ among the points in $\{(i, N_1[i]) : i = 1, \dots, n\}$, and can be solved in $O(1)$ time according to Lemma 7. Let $x_1 = \text{rank}_{z_1-1}(\ell_1)$ and $x_2 = \text{rank}_{z_1-1}(\ell_2)$, and denote $y_1 = s_1 - (n/t)(z_1 - 1)$ and $y_2 = s_2 - (n/t)(z_2 - 1)$. Query (1) is equivalent to the query $[x_1..x_2] \times [y_1..y_2]$ among the points in N_{2,z_1-1} . We handle this query recursively. Query (3) is handled in the same way. As there are $\log_t n$ levels and each level takes $O(1)$ time, the query is answered in $O(\log_t n) = O(\frac{\log n}{\log \log n})$ time. ◀

4.3 Time complexity

This subsection explains how to implement `New_Adams_consensus_2`. Do the following preprocessing:

- Fix an arbitrary left-to-right ordering of the children at every node in T_1 . For $i \in \{1, 2, \dots, n\}$, let $L_1(i)$ be the i th leaf in T_1 in the resulting left-to-right ordering. (Thus, $(L_1(1), L_1(2), \dots, L_1(n))$ is a permutation of L .) Define $L_2(i)$ for $i \in \{1, 2, \dots, n\}$ analogously using T_2 . Let $N = \{(L_1^{-1}(\ell), L_2^{-1}(\ell)) : \ell \in L\}$ and build the data structure $D(N)$ from Lemma 8.
- For $j \in \{1, 2\}$, preprocess T_j in $O(n)$ time so that any $\text{lca}^{T_j}(B)$ -query can be answered in $O(|B|)$ time [3, 13].
- As in the proof of Theorem 2 in Section 2 above, preprocess T_j for $j \in \{1, 2\}$ with the level ancestor data structure of [4] in $O(n)$ time so that the ancestor of any $\ell \in L$ at depth 1 in T_j can be returned in $O(1)$ time. Also preprocess T_j for $j \in \{1, 2\}$ in $O(n)$ time as in Section 8 of [9] so that $T_j|B$ for any $B \subseteq L$ can be constructed in $O(|B|)$ time.

The preprocessing takes $O(n \cdot \frac{\log n}{\log \log n})$ time in total.

Next, for any pair of siblings $u, u' \in V(T_j)$, $j \in \{1, 2\}$, let $T_j^{u..u'}$ denote the set of all rooted subtrees of the form T_j^x , where x belongs to the interval of siblings $[u, \dots, u']$ in T_j , and define $\Lambda(T_j^{u..u'}) = \bigcup_{x \in [u, \dots, u']} \Lambda(T_j^x)$.

► **Lemma 9.** *Given the data structure $D(N)$ in Lemma 8, for any siblings u and u' in T_1 and any siblings v and v' in T_2 , the value of $|\Lambda(T_1^{u..u'}) \cap \Lambda(T_2^{v..v'})|$ can be found in $O(\frac{\log n}{\log \log n})$ time. Furthermore, the leftmost and rightmost leaves in T_1 (or T_2) among all leaves in $\Lambda(T_1^{u..u'}) \cap \Lambda(T_2^{v..v'})$ can be reported in $O(\frac{\log n}{\log \log n})$ time.*

Proof. Let l_u be the leftmost leaf in T_1^u and $r_{u'}$ the rightmost leaf in $T_1^{u'}$. Then each $\ell \in \Lambda(T_1^{u..u'})$ satisfies $L_1^{-1}(l_u) \leq L_1^{-1}(\ell) \leq L_1^{-1}(r_{u'})$. Similarly, each $\ell \in \Lambda(T_2^{v..v'})$ satisfies $L_2^{-1}(l_v) \leq L_2^{-1}(\ell) \leq L_2^{-1}(r_{v'})$, where l_v is the leftmost leaf in T_2^v and $r_{v'}$ the rightmost leaf in $T_2^{v'}$. Hence, any $\ell \in L$ belongs to $\Lambda(T_1^{u..u'}) \cap \Lambda(T_2^{v..v'})$ if and only if the point $(L_1^{-1}(\ell), L_2^{-1}(\ell))$ lies in the rectangle defined by $[L_1^{-1}(l_u)..L_1^{-1}(r_{u'})] \times [L_2^{-1}(l_v)..L_2^{-1}(r_{v'})]$ on the grid represented by $D(N)$. By Lemma 8, the lemma follows. ◀

Lemma 9 allows the $Z_{u,v}$ -sets to be computed quickly by the procedure `Compute_Z` shown in Algorithm 5. More precisely:

► **Lemma 10.** *Given the data structure $D(N)$ in Lemma 8, the procedure `Compute_Z` can compute the set $Z_{u,v}$ for any $u \in V(T_1)$ and $v \in V(T_2)$ in $O(|Z_{u,v}| \cdot \frac{\log n}{\log \log n})$ time.*

Proof. Let u_1, \dots, u_α be the ordered list of children of u and v_1, \dots, v_β the ordered list of children of v . The procedure identifies the pairs $(u_p, v_q) \in Z_{u,v}$ in increasing order of u_p and then in increasing order of v_q . In the outer loop, each child u_p of u satisfying $\Lambda(T_1^{u_p}) \cap \Lambda(T_2^v) \neq \emptyset$ is identified from left to right by using Lemma 9 in Step 4 and the level

Algorithm 5 Procedure `Compute_Z`.

Procedure `Compute_Z`

Input: $u \in V(T_1)$, $v \in V(T_2)$, where T_1, T_2 are two given trees with $\Lambda(T_1) = \Lambda(T_2)$.

Output: $Z_{u,v} = \{(u', v') : u' \in \text{Child}^{T_1}(u), v' \in \text{Child}^{T_2}(v), |\Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'})| > 0\}$.

- 1: Let $u_1..u_\alpha$ and $v_1..v_\beta$ be the ordered lists of children of u and v , respectively;
 - 2: $Z := \emptyset$; $i := 1$;
 - 3: **while** $i \leq \alpha$ **do**
 - 4: Find the leftmost leaf a in T_1 such that $a \in \Lambda(T_1^{u_i..u_\alpha}) \cap \Lambda(T_2^v)$;
 - 5: If no such a exists, **break**;
 - 6: Identify the $u_p \in \text{Child}^{T_1}(u)$ such that $a \in \Lambda(T_1^{u_p})$;
 - 7: $j := 1$;
 - 8: **while** $j \leq \beta$ **do**
 - 9: Find the leftmost leaf b in T_2 such that $b \in \Lambda(T_1^{u_p}) \cap \Lambda(T_2^{v_j..v_\beta})$;
 - 10: If no such b exists, **break**;
 - 11: Identify the $v_q \in \text{Child}^{T_2}(v)$ such that $b \in \Lambda(T_2^{v_q})$;
 - 12: Let $Z := Z \cup \{(u_p, v_q)\}$ and $j := q + 1$;
 - 13: **end while**
 - 14: Let $i := p + 1$;
 - 15: **end while**
 - 16: **return** Z ;
-

ancestor data structure in Step 6. Each u_p is thus identified in $O(\frac{\log n}{\log \log n})$ time. Then, for each such u_p , the inner loop similarly finds every child v_q of v with $\Lambda(T_1^{u_p}) \cap \Lambda(T_2^{v_q}) \neq \emptyset$ from left to right, using $O(\frac{\log n}{\log \log n})$ time per v_q . In total, the procedure spends $O(|Z_{u,v}| \cdot \frac{\log n}{\log \log n})$ time to compute $Z_{u,v}$. ◀

Finally, we are ready to analyze the running time of `New_Adams_consensus_2`. For any $u' \in V(T_1)$, $v' \in V(T_2)$, denote $S_{u',v'} = \Lambda(T_1^{u'}) \cap \Lambda(T_2^{v'})$.

► **Theorem 11.** *Given the data structure $D(N)$ in Lemma 8, `New_Adams_consensus_2`(u, v) for any $u \in V(T_1)$ and $v \in V(T_2)$ runs in $O(|S_{u,v}| \cdot \frac{\log n}{\log \log n})$ time.*

Proof. Let $T(u, v)$ be the running time of `New_Adams_consensus_2`(u, v), including the time required to compute γ , δ , and $Z_{\gamma,\delta}$ and the recursive calls `New_Adams_consensus_2`(u', v') for all $(u', v') \in Z_{\gamma,\delta}$. By using Lemma 9, Steps 1 and 5 can be carried out in $O(\frac{\log n}{\log \log n})$ time. Step 6 takes $O(1)$ time because of the preprocessing and Step 7 takes $O(|Z_{\gamma,\delta}| \cdot \frac{\log n}{\log \log n})$ time according to Lemma 10. We therefore have $T(u, v) = \sum_{(u',v') \in Z_{\gamma,\delta}} T(u', v') + O(|Z_{\gamma,\delta}| \cdot \frac{\log n}{\log \log n})$. Observe that in the base case, i.e., where $|S_{u,v}| = 1$, it holds that $T(u, v) = O(\frac{\log n}{\log \log n})$.

We apply the recursion-tree method to solve the recurrence for $T(u, v)$. The root of the recursion tree for $T(u, v)$ represents the top level of recursion, and its cost is $O(|Z_{\gamma,\delta}| \cdot \frac{\log n}{\log \log n})$. There are $|Z_{\gamma,\delta}|$ subtrees attached to the root, each of which corresponds to a recursion tree for one $T(u', v')$ where $(u', v') \in Z_{\gamma,\delta}$. The leaves of the recursion tree represent the base cases of the recursion, i.e., those $T(x, y)$ satisfying $|S_{x,y}| = 1$, and they each have cost $O(\frac{\log n}{\log \log n})$. It follows that the recursion tree for $T(u, v)$ has exactly $|S_{u,v}|$ leaves and no nodes with degree 1. Now, the value of $T(u, v)$ is equal to the sum of the costs taken over all nodes in the recursion tree. Clearly, the total contribution of the leaves is $O(|S_{u,v}| \cdot \frac{\log n}{\log \log n})$. We rewrite the cost of each internal node in the recursion tree as $O(d \cdot \frac{\log n}{\log \log n})$, where d is the degree of that node. Since the sum of the degrees of all internal nodes in a tree without any nodes of degree 1 is less than twice the number of leaves, the contribution of the internal nodes is also $O(|S_{u,v}| \cdot \frac{\log n}{\log \log n})$. The total running time is $T(u, v) = O(|S_{u,v}| \cdot \frac{\log n}{\log \log n})$. ◀

Recall that $D(N)$ is constructed during the preprocessing phase using $O(n \cdot \frac{\log n}{\log \log n})$ time. Theorem 11 implies that `New_Adams_consensus_2`(r_1, r_2), where r_i is the root of T_i for $i \in \{1, 2\}$, computes the Adams consensus tree of $\{T_1, T_2\}$ in $O(n \cdot \frac{\log n}{\log \log n})$ time.

Acknowledgments: The authors would like to thank the anonymous reviewers for their suggestions.

References

- 1 E. N. Adams III. Consensus techniques and the comparison of taxonomic trees. *Systematic Zoology*, 21(4):390–397, 1972.
- 2 E. N. Adams III. N-trees as nestings: Complexity, similarity, and consensus. *Journal of Classification*, 3(2):299–317, 1986.
- 3 M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics (LATIN 2000)*, volume 1776 of *LNCS*, pages 88–94. Springer-Verlag, 2000.
- 4 M. A. Bender and M. Farach-Colton. The Level Ancestor Problem simplified. *Theoretical Computer Science*, 321(1):5–12, 2004.

- 5 M. G. B. Blum, O. François, and S. Janson. The mean, variance and limiting distribution of two statistics sensitive to phylogenetic tree balance. *The Annals of Applied Probability*, 16(4):2195–2214, 2006.
- 6 P. Bose, M. He, A. Maheshwari, and P. Morin. Succinct orthogonal range search structures on a grid with applications to text indexing. In *Proceedings of the 11th International Symposium on Algorithms and Data Structures (WADS 2009)*, volume 5664 of *LNCS*, pages 98–109. Springer-Verlag, 2009.
- 7 K. Bremer. Combinable component consensus. *Cladistics*, 6(4):369–372, 1990.
- 8 D. Bryant. A classification of consensus methods for phylogenetics. In M. F. Janowitz, F.-J. Lapointe, F. R. McMorris, B. Mirkin, and F. S. Roberts, editors, *Bioconsensus*, volume 61 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 163–184. American Mathematical Society, 2003.
- 9 R. Cole, M. Farach-Colton, R. Hariharan, T. Przytycka, and M. Thorup. An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees. *SIAM Journal on Computing*, 30(5):1385–1404, 2000.
- 10 J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Inc., Sunderland, Massachusetts, 2004.
- 11 J. Felsenstein. PHYLIP, version 3.6. Software package, Department of Genome Sciences, University of Washington, Seattle, U.S.A., 2005.
- 12 P. A. Goloboff, J. S. Farris, M. Källersjö, B. Oxelman, M. J. Ramírez, and C. A. Szumik. Improvements to resampling measures of group support. *Cladistics*, 19(4):324–332, 2003.
- 13 D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.
- 14 E. Hernández-García, M. Tuğrul, E. Alejandro Herrada, V. M. Eguíluz, and K. Klemm. Simple models for scaling in phylogenetic trees. *International Journal of Bifurcation and Chaos*, 20(3):805–811, 2010.
- 15 Z.-X. Luo, Q. Ji, J. R. Wible, and C.-X. Yuan. An early Cretaceous tribosphenic mammal and metatherian evolution. *Science*, 302(5652):1934–1940, 2003.
- 16 T. Margush and F. R. McMorris. Consensus n -Trees. *Bulletin of Mathematical Biology*, 43(2):239–244, 1981.
- 17 L. Nakhleh, T. Warnow, D. Ringe, and S. N. Evans. A comparison of phylogenetic reconstruction methods on an Indo-European dataset. *Transactions of the Philological Society*, 103(2):171–192, 2005.
- 18 R. Page. COMPONENT, version 2.0. Software package, University of Glasgow, U.K., 1993.
- 19 E. R. Seiffert. Revised age estimates for the later Paleogene mammal faunas of Egypt and Oman. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 103(13):5000–5005, 2006.
- 20 C. Semple and M. Steel. *Phylogenetics*, volume 24 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, 2003.
- 21 R. R. Sokal and F. J. Rohlf. Taxonomic congruence in the Leptopodomorpha re-examined. *Systematic Zoology*, 30(3):309–325, 1981.
- 22 W.-K. Sung. *Algorithms in Bioinformatics: A Practical Introduction*. Chapman & Hall/CRC, 2010.
- 23 D. L. Swofford. PAUP*, version 4.0. Software package, Sinauer Associates, Inc., Sunderland, Massachusetts, 2003.
- 24 X. Xu, J. M. Clark, C. A. Forster, M. A. Norell, G. M. Erickson, D. A. Eberth, C. Jia, and Q. Zhao. A basal tyrannosauroid dinosaur from the Late Jurassic of China. *Nature*, 439(7077):715–718, 2006.

Flip Distance Is in FPT Time $\mathcal{O}(n + k \cdot c^k)$

Iyad Kanj¹ and Ge Xia²

- 1 School of Computing, DePaul University
Chicago, USA
ikanj@cs.depaul.edu
- 2 Department of Computer Science
Lafayette College
Easton, USA, xiag@lafayette.edu

Abstract

Let \mathcal{T} be a triangulation of a set \mathcal{P} of n points in the plane, and let e be an edge shared by two triangles in \mathcal{T} such that the quadrilateral Q formed by these two triangles is convex. A *flip* of e is the operation of replacing e by the other diagonal of Q to obtain a new triangulation of \mathcal{P} from \mathcal{T} . The *flip distance* between two triangulations of \mathcal{P} is the minimum number of flips needed to transform one triangulation into the other. The FLIP DISTANCE problem asks if the flip distance between two given triangulations of \mathcal{P} is k , for some given $k \in \mathbb{N}$. It is a fundamental and a challenging problem.

In this paper we present an algorithm for the FLIP DISTANCE problem that runs in time $\mathcal{O}(n + k \cdot c^k)$, for a constant $c \leq 2 \cdot 14^{11}$, which implies that the problem is fixed-parameter tractable. The NP-hardness reduction for the FLIP DISTANCE problem given by Lubiw and Pathak can be used to show that, unless the exponential-time hypothesis (ETH) fails, the FLIP DISTANCE problem cannot be solved in time $\mathcal{O}^*(2^{o(k)})$. Therefore, one cannot expect an asymptotic improvement in the exponent of the running time of our algorithm.

1998 ACM Subject Classification F.2.2 Geometrical Problems and Computations, G.2.1 Combinatorial Algorithms

Keywords and phrases triangulations, flip distance, parameterized algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.500

1 Introduction

Let \mathcal{P} be a set of n points in the plane. A *triangulation* of \mathcal{P} is a partitioning of the convex hull of \mathcal{P} into triangles such that the set of vertices of the triangles in the triangulation is \mathcal{P} . Note that the convex hull of \mathcal{P} may contain points of \mathcal{P} in its interior.

A *flip* to an (interior) edge e in a triangulation of \mathcal{P} is the operation of replacing e by the other diagonal of the quadrilateral formed by the two triangles that share e , provided that this quadrilateral is convex; otherwise, flipping e is not permissible. The *flip distance* between two triangulations $\mathcal{T}_{initial}$ and \mathcal{T}_{final} of \mathcal{P} is the length of a shortest sequence of flips that transforms $\mathcal{T}_{initial}$ into \mathcal{T}_{final} . This distance is always well-defined and is $O(|\mathcal{P}|^2)$ (e.g., see [8]). The FLIP DISTANCE problem is: Given two triangulation $\mathcal{T}_{initial}$ and \mathcal{T}_{final} of \mathcal{P} , and $k \in \mathbb{N}$, decide if the flip distance between $\mathcal{T}_{initial}$ and \mathcal{T}_{final} is k .

Triangulations are a very important subject of study in computational geometry, and they have applications in computer graphics, visualization, and geometric design (see [17, 19, 20, 24], to name a few). Flips in triangulations and the FLIP DISTANCE problem have received a large share of attention (see [3] for a review). The FLIP DISTANCE problem is a very fundamental and challenging problem, and different aspects of this problem have

been studied, including the combinatorial, geometrical, topological, and computational aspects [1, 2, 3, 4, 7, 8, 10, 18, 21, 22]. We can define the triangulations graph of \mathcal{P} , whose vertex-set is the set of all triangulations of \mathcal{P} , and in which two triangulations/vertices are adjacent if and only if their distance is 1. It is well-known that the triangulations graph has diameter $O(n^2)$ [8], and hence, we can transform any triangulation into another by a sequence of $O(n^2)$ flips. Moreover, it is known that the number of vertices in the triangulations graph is $\Omega(2.33^n)$ [1]. Therefore, solving the FLIP DISTANCE problem by finding a shortest path between the two triangulations in the triangulations graph is not feasible. A very similar problem to the FLIP DISTANCE was studied by Wagner [23] in 1936, who considered triangulated planar graphs instead.

The complexity of the FLIP DISTANCE problem was resolved very recently (2012) by Lubiw and Pathak [10, 11] who showed the problem to be NP-complete. Simultaneously, and independently, the problem was shown to be APX-hard by Pilz [18]. Very recently, Aichholzer et al. [2] showed the problem to be NP-complete for triangulations of a simple polygon. Resolving the complexity of the problem for the special case when \mathcal{P} is in a convex position (*i.e.*, triangulations of a convex polygon) remains a longstanding open problem for at least 25 years (see [22]); this problem is equivalent to the problem of computing the rotation distance between two rooted binary trees [4, 22]. Cleary and St. John [4] showed that this special case (convex polygon) is fixed-parameter tractable (FPT): They gave a kernel of size $5k$ for the problem and presented an $\mathcal{O}^*((5k)^k)$ -time FPT algorithm based on this kernel (the $\mathcal{O}^*(\cdot)$ notation suppresses polynomial factors in the input size). The upper bound on the kernel size for the convex case was subsequently improved to $2k$ by Lucas [12], who also gave an $\mathcal{O}^*(k^k)$ -time FPT algorithm for this case. The kernelization approaches used in [4, 12] for the convex case are not applicable to the general case. In particular, the reduction rules used in [4, 12] to obtain a kernel for the convex case, and hence the FPT algorithms based on these kernels, do not generalize to the problem under consideration in this paper.

In this paper we present an $\mathcal{O}(n + k \cdot c^k)$ -time algorithm ($c \leq 2 \cdot 14^{11}$) for the FLIP DISTANCE problem for triangulations of an arbitrary point-set in the plane, which shows that the problem is FPT. Our result is a significant improvement over the $\mathcal{O}^*(k^k)$ -time algorithm by Lucas [12] for the simpler convex case. The NP-hardness reduction by Lubiw and Pathak can be used to show that, unless the exponential-time hypothesis (ETH) fails [9], the FLIP DISTANCE problem cannot be solved in time $\mathcal{O}^*(2^{o(k)})$. Therefore, one should not expect an asymptotic improvement in the exponent of the running time of the presented algorithm. While it is not very difficult to show that the FLIP DISTANCE problem is FPT based on some of the structural results in this paper, obtaining an $\mathcal{O}^*(c^k)$ -time algorithm, for some constant c , is quite involved, and requires a deep understanding of the structure of the problem.

Our approach is as follows. For any solution to a given instance of the problem, we can define a directed acyclic graph (DAG), whose nodes are the flips in the solution, that captures the dependency relation among the flips. We show that any topological sorting of this DAG corresponds to a valid solution of the instance. The difficult part is how, without knowing the DAG, to navigate the triangulation and perform the flips in an order that corresponds to a topological sorting of the DAG. We present a *very simple* nondeterministic algorithm that performs a sequence of “flip/move”-type local actions in a triangulation, where each local action has constant-many choices. The key is to show that there exists such a sequence of actions that corresponds to a topological sorting of the DAG associated with a solution to the instance, and that the length of this sequence is linear in the number of nodes in the DAG. This will us to simulate the nondeterministic algorithm by an $\mathcal{O}^*(c^k)$ -time deterministic algorithm. To achieve the above goal, we develop structural results that reveal some of the structural intricacies of this fundamental and challenging problem.

Even though the triangulations considered in the paper are triangulations of a point-set in the plane, the presented algorithm works as well for triangulations of any polygonal region (even with points in its interior). Moreover, using a reduction in [11] from the FLIP DISTANCE of triangulations of a polygonal region with holes to the FLIP DISTANCE of triangulations of a polygonal region with points in its interior, the algorithm presented in this paper can solve the (more general) FLIP DISTANCE problem of triangulations of a polygonal region with (possible) holes within the same time upper bound.

2 Preliminaries

Let \mathcal{P} be a set of n points in the plane, and let \mathcal{T} be a triangulation of \mathcal{P} . Let e be an interior (non-boundary) edge in \mathcal{T} . The *quadrilateral associated with e* in \mathcal{T} is defined to be the quadrilateral formed by the two adjacent triangles in \mathcal{T} that share e as an edge. Let e be an edge in \mathcal{T} such that the quadrilateral Q in \mathcal{T} associated with e is convex. A *flip f* with underlying edge e is an operation performed to e in triangulation \mathcal{T} that removes e and replaces it with the other diagonal of Q , thus obtaining a new triangulation of \mathcal{P} from \mathcal{T} . We use the notation $\varepsilon(f)$ to denote the underlying edge e of a flip f in \mathcal{T} , and the notation $\varphi(f)$ to denote the new diagonal/edge resulting from flip f . Note that $\varphi(f)$ is not in \mathcal{T} . We say that a flip to an edge e is *admissible* in triangulation \mathcal{T} if e is in \mathcal{T} and the quadrilateral associated with e is convex. We say that two distinct edges e and e' in \mathcal{T} *share a triangle* if e and e' appear in the same triangle in \mathcal{T} . We say that two distinct edges e and e' between points in \mathcal{P} *cross* if e and e' intersect in their interior.

Let \mathcal{T} be a triangulation. A sequence of flips $F = \langle f_1, \dots, f_r \rangle$ is *valid* with respect to \mathcal{T} if there exist triangulations $\mathcal{T}_0, \dots, \mathcal{T}_r$ such that $\mathcal{T}_0 = \mathcal{T}$, f_i is admissible in \mathcal{T}_{i-1} , and performing flip f_i in \mathcal{T}_{i-1} results in triangulation \mathcal{T}_i , for $i = 1, \dots, r$. In this case we say that \mathcal{T}_r is the *outcome* of applying F to \mathcal{T} and that F *transforms* \mathcal{T} into \mathcal{T}_r , and we write $\mathcal{T} \xrightarrow{F} \mathcal{T}_r$. The *length* of F , denoted $|F|$, is the number of flips in it. Many flips in a sequence F may have the same underlying edge, but all those flips are distinct flips. For two flips f_i and f_h of F such that $i < h$, a flip f_p in F is said to be *between* f_i and f_h if $i < p < h$.

For two triangulations $\mathcal{T}_{initial}$ and \mathcal{T}_{final} of \mathcal{P} , the *flip distance* between $\mathcal{T}_{initial}$ and \mathcal{T}_{final} is the smallest $d \in \mathbb{N}$ such that there is a sequence F of length d satisfying that $\mathcal{T}_{initial} \xrightarrow{F} \mathcal{T}_{final}$. The FLIP DISTANCE problem is defined as follows:

FLIP DISTANCE

Given: Two triangulation $\mathcal{T}_{initial}$ and \mathcal{T}_{final} of \mathcal{P} .

Parameter: k .

Question: Is the flip distance between $\mathcal{T}_{initial}$ and \mathcal{T}_{final} equal to k ?

Let G be a graph. $V(G)$ and $E(G)$ denote the vertex-set and the edge-set of G , respectively, and $|G|$ denotes the *size* of G , which is $|V(G)| + |E(G)|$. For a directed graph G , a *weakly connected component* of G is a (maximal) connected component of the underlying undirected graph of G ; for simplicity, we will use the term *component* of a directed graph G to refer to a weakly connected component of G . Otherwise, we assume familiarity with basic graph theory, and refer to [5] for more information.

A *parameterized problem* is a set of instances of the form (x, k) , where x is the input instance and $k \in \mathbb{N}$ is the *parameter*. A parameterized problem is *fixed-parameter tractable*, shortly FPT, if there is an algorithm that solves the problem in time $f(k)|x|^c$, where f is a

computable function and $c > 0$ is a constant. We refer to [6, 16] for more information about parameterized complexity.

3 Structural results

Let \mathcal{T} be a triangulation and let $F = \langle f_1, \dots, f_r \rangle$ be a valid sequence of flips with respect to \mathcal{T} . We denote by \mathcal{T}_i , for $i = 1, \dots, r$, the triangulation that is the outcome of applying the (valid) subsequence of flips $\langle f_1, \dots, f_i \rangle$ to \mathcal{T} .

► **Definition 1.** Let f_i and f_j be two flips in F such that $1 \leq i < j \leq r$. Flip f_j is said to be *adjacent* to flip f_i , denoted $f_i \rightarrow f_j$, if:

- (1) either $\varphi(f_i) = \varepsilon(f_j)$ or $\varphi(f_i)$ and $\varepsilon(f_j)$ share a triangle in triangulation \mathcal{T}_{j-1} ; and
- (2) $\varphi(f_i)$ is not flipped between f_i and f_j , that is, there does not exist a flip f_p in F , where $i < p < j$, such that $\varepsilon(f_p) = \varphi(f_i)$.

The above adjacency relation defined on the flips in F can be naturally represented by a directed acyclic graph (DAG), denoted \mathcal{D}_F , where the nodes of \mathcal{D}_F are the flips in F , and its arcs represent the (directed) adjacencies in F . Note that by definition, if $f_i \rightarrow f_j$ then $i < j$. For simplicity, we will label the nodes in \mathcal{D}_F with the labels of their corresponding flips in F .

► **Lemma 2.** *Every node in \mathcal{D}_F has indegree at most 5. Therefore, $|E(\mathcal{D}_F)| \leq 5 \cdot |V(\mathcal{D}_F)|$ and $|\mathcal{D}_F| \leq 6 \cdot |V(\mathcal{D}_F)|$.*

► **Lemma 3.** *Let \mathcal{T}_0 be a triangulation and let $F = \langle f_1, \dots, f_r \rangle$ be a sequence of flips such that $\mathcal{T}_0 \xrightarrow{F} \mathcal{T}_r$. Let $\pi(F)$ be a permutation of the flips in F such that $\pi(F)$ is a topological sorting of \mathcal{D}_F . Then $\pi(F)$ is a valid sequence of flips such that $\mathcal{T}_0 \xrightarrow{\pi(F)} \mathcal{T}_r$.*

Proof. Proceed by induction on $|F|$. If $|F| \leq 1$, then the statement obviously holds true. Suppose that the statement is true for any F such that $|F| < r$, where $r > 1$, and consider a sequence F such that $|F| = r$.

Let f_s be the last flip in $\pi(F)$. Since $\pi(F)$ is a topological sorting of \mathcal{D}_F , f_s must be a sink in \mathcal{D}_F . It follows that no flip after f_s in F is adjacent to f_s in \mathcal{D}_F . Let Q be the quadrilateral associated with $\varphi(f_s)$ in triangulation \mathcal{T}_s . Then no flips after f_s in F has its underlying edge in Q (i.e., as a boundary edge of Q or as a diagonal of Q), which means that the two adjacent triangles forming Q in \mathcal{T}_s remain unchanged throughout the flips after f_s in F . Therefore, we can safely move the flip f_s to the end of the sequence F without affecting the other flips in F nor the validity of F . Let this new sequence be F' ; then it follows from the previous argument that $\mathcal{T}_0 \xrightarrow{F'} \mathcal{T}_r$. Since f_s appears at the end of F' , $F' - f_s$ is a valid sequence with respect to \mathcal{T}_0 that transforms \mathcal{T}_0 into some triangulation \mathcal{T} such that $\mathcal{T} \xrightarrow{f_s} \mathcal{T}_r$. Note that since f_s is a sink in \mathcal{D}_F , $\pi(F) - f_s$ is a permutation of the flips in $F' - f_s$ that is a topological sorting of $\mathcal{D}_F - f_s$. By the inductive hypothesis, $\pi(F) - \{f_s\}$ transforms \mathcal{T}_0 into \mathcal{T} . Since $\mathcal{T} \xrightarrow{f_s} \mathcal{T}_r$, appending f_s to the end of $\pi(F) - \{f_s\}$ results in $\pi(F)$ such that $\mathcal{T}_0 \xrightarrow{\pi(F)} \mathcal{T}_r$. This completes the inductive proof. ◀

► **Corollary 4.** *Let \mathcal{T}_0 be a triangulation and let $F = \langle f_1, \dots, f_r \rangle$ be a sequence of flips such that $\mathcal{T}_0 \xrightarrow{F} \mathcal{T}_r$. For any given ordering (C_1, \dots, C_ℓ) of the components in \mathcal{D}_F , there is a permutation $\pi(F)$ of the flips in F such that $\mathcal{T}_0 \xrightarrow{\pi(F)} \mathcal{T}_r$, and such that for any two flips $f_i \in C_t$ and $f_j \in C_s$, where $1 \leq t < s \leq \ell$, f_i appears before f_j in $\pi(F)$. That is, all the flips in the same component appear as a consecutive block in $\pi(F)$, and the order of the blocks in $\pi(F)$ is the same as the given order of their corresponding components.*

► **Definition 5.** Let $(\mathcal{T}_{initial}, \mathcal{T}_{final}, k)$ be an instance of FLIP DISTANCE. An edge in $\mathcal{T}_{initial}$ that is not in \mathcal{T}_{final} is called a *changed edge*. If a sequence F is a solution to the instance $(\mathcal{T}_{initial}, \mathcal{T}_{final}, k)$, we call a component in \mathcal{D}_F *essential* if the component contains a flip f such that $\varepsilon(f)$ is a changed edge, otherwise, the component is called *nonessential*.

► **Lemma 6.** Let $(\mathcal{T}_{initial}, \mathcal{T}_{final}, k)$ be an instance of FLIP DISTANCE, and suppose that F is a solution to the instance. Then every component of \mathcal{D}_F is essential.

Proof. Suppose, to get a contradiction, that \mathcal{D}_F contains a nonessential component C . Let F_C be the subsequence of F consisting of the flips that are in C . We will show that $F - F_C$ is a solution to the instance $(\mathcal{T}_{initial}, \mathcal{T}_{final}, k)$. This will contradict the minimality of F because the number of flips in F is the flip distance between $\mathcal{T}_{initial}$ and \mathcal{T}_{final} .

By Corollary 4, we can assume that all the flips in F_C appear consecutively (*i.e.*, as a single block) at the end of F . Let \mathcal{T}' be the outcome of applying $F - F_C$ to $\mathcal{T}_{initial}$. It suffices to show that $\mathcal{T}' = \mathcal{T}_{final}$. Suppose that this is not the case. Since the number of edges in \mathcal{T}' and \mathcal{T}_{final} is the same, there must exist an edge $e \in \mathcal{T}'$ such that $e \notin \mathcal{T}_{final}$. Therefore, C must contain a flip f such that $\varepsilon(f) = e$; assume that f is the first such flip in C . Since C is nonessential, $e \notin \mathcal{T}_{initial}$, otherwise e would be a changed edge. Therefore, there must exist a flip f' in $F - F_C$ such that $\varphi(f') = e$; we can assume that f' is the last such flip in $F - F_C$. By the definition of adjacency in \mathcal{D}_F , there is an arc from node f' in $\mathcal{D}_F - C$ to node f in C , contradicting the assumption that C is a component of \mathcal{D}_F . ◀

Let $(\mathcal{T}_{initial}, \mathcal{T}_{final}, k)$ be an instance of FLIP DISTANCE, and suppose that F is a solution for $(\mathcal{T}_{initial}, \mathcal{T}_{final}, k)$. By Lemma 6, \mathcal{D}_F does not contain nonessential components, and by Corollary 4, we can assume that all the flips in the same component of \mathcal{D}_F appear as a consecutive block in F . We shall call such a solution F satisfying the above properties a *normalized* solution. Suppose that $F = \langle f_1, \dots, f_k \rangle$ is a normalized solution to an instance $(\mathcal{T}_{initial}, \mathcal{T}_{final}, k)$ of FLIP DISTANCE, and let C be a component of \mathcal{D}_F . The following lemma provides several sufficient conditions for a directed path to exist between two flips in C :

► **Lemma 7.** Let f_i and f_h , where $i < h$, be two flips in C . If one of the following conditions is true, then there is a directed path from f_i to f_h in C :

- (1) $\varphi(f_h)$ crosses $\varepsilon(f_i)$.
- (2) $\varphi(f_h) = \varepsilon(f_i)$.
- (3) $\varepsilon(f_i) = \varepsilon(f_h)$.
- (4) $\varphi(f_i) = \varepsilon(f_h)$, or $\varphi(f_i)$ and $\varepsilon(f_h)$ share a triangle T in \mathcal{T}_j , for some j satisfying $i \leq j < h$.

4 The algorithm

Using the structural results in Section 3, it is not difficult to obtain an FPT algorithm for FLIP DISTANCE that runs in $\mathcal{O}^*(c^{k^2})$ time, for some constant c . For instance, starting from an edge in the current triangulation (which corresponds to a flip in the DAG representing the remaining solution), we can grow a BFS-like tree of size c^k searching for the next edge to flip (corresponding to a source node in the DAG), and flip this edge. Repeating this process k times gives an $\mathcal{O}^*(c^{k^2})$ -time algorithm for the problem. Our goal, however, is to obtain an $\mathcal{O}^*(c^k)$ -time algorithm for the problem, for some constant c . Achieving this goal turns out to be quite challenging, and requires a deep understanding of the structure of the problem. We did so by analyzing the relation between the DAG associated with a solution to a problem instance and the changing structure of the underlying triangulations.

4.1 Overview of the algorithm

In this subsection we give an intuitive description of how our algorithm works. Let $(\mathcal{T}_{initial}, \mathcal{T}_{final}, k)$ be an instance of FLIP DISTANCE. In order to solve the instance, by Lemma 3, it suffices for the algorithm to perform a sequence of k flips that is a topological sorting of the DAG \mathcal{D}_F associated with a normalized solution F to the instance. Needless to say, the difficulty is that we do not know F , nor do we know \mathcal{D}_F . By Lemma 6, each component of \mathcal{D}_F is essential, and hence, must contain a changed edge. The algorithm starts by picking a changed edge e in $\mathcal{T}_{initial}$. There must exist a flip f in \mathcal{D}_F such that $e = \varepsilon(f)$; let us refer to the component of \mathcal{D}_F containing f by C . We explain next how the algorithm, starting at e in $\mathcal{T}_{initial}$, performs a sequence of flips that is a topological sorting of C ; this can be easily extended to a sequence of flips that is a topological sorting of \mathcal{D}_F .

Clearly, the algorithm cannot start by performing f because other flips may precede f in the solution. Instead, the algorithm searches for an edge $\varepsilon(f_s)$ in $\mathcal{T}_{initial}$ that is the underlying edge of a source node f_s in C , and flips $\varepsilon(f_s)$. Now we explain how the algorithm searches for $\varepsilon(f_s)$ in $\mathcal{T}_{initial}$ without having access to C . The algorithm starts at edge e in $\mathcal{T}_{initial}$ and nondeterministically “takes a walk” in which each step/action consists of moving to an edge that shares a triangle with the edge that the algorithm is currently at; the number of such local actions is the length of the walk. We show (Lemma 10) that there exists a source node f_s in C such that, starting at the changed edge e , the algorithm can walk in the *current triangulation* $\mathcal{T}_{initial}$ from e to $\varepsilon(f_s)$, and that the length of this walk is at most the length of the path from f_s to f in C . Suppose that the algorithm nondeterministically guessed the right walk, and walked to $\varepsilon(f_s)$ in $\mathcal{T}_{initial}$. The algorithm then flips $\varepsilon(f_s)$, thus performing flip f_s in C , to obtain a new triangulation $\mathcal{T}_{current}$, and stays at the edge $\varphi(f_s)$ in $\mathcal{T}_{current}$. To continue the sequence of flips that corresponds to a topological sorting of C , the algorithm should flip next an edge in $\mathcal{T}_{current}$ that corresponds to a source node in the resulting DAG $C_{current} = C - f_s$. Hence, the algorithm needs to walk from $\varphi(f_s)$ in $\mathcal{T}_{current}$ to a source node in $C_{current}$, and to flip the edge corresponding to that source node in $\mathcal{T}_{current}$, and so on and so forth. To show how to perform this desired sequence of nondeterministic actions so that total number of actions remains linear in k , we define a spanning subgraph J_C of the underlying graph of C . We then show that there exists a sequence of local actions by the algorithm, in which the edge-flips is a topological sorting of C , that simulates a recursive traversal of J_C . This mapping of the actions of the algorithm to a specific traversal of J_C will allow us to “charge” the actions of the algorithm to the nodes and edges of J_C , thus obtaining the desired linear upper bound on the number of actions of the algorithm in terms of the size of J_C , and hence the size of C .

4.2 The nondeterministic actions of the algorithm

The algorithm is a nondeterministic algorithm that starts from a changed edge in a triangulation $\mathcal{T}_{initial}$ and performs a sequence of actions. The algorithm is equipped with a stack. Each action σ of the algorithm acts on some edge e in a triangulation that we refer to as the *current triangulation* (before σ), denoted $\mathcal{T}_{current}$. Initially $\mathcal{T}_{current} = \mathcal{T}_{initial}$, and $\mathcal{T}_{current}$ before action σ is the triangulation resulting from applying the sequence of actions preceding σ to $\mathcal{T}_{initial}$. Each action σ of the algorithm is of the following possible types:

- (i) Move to one of the (at most 4) edges that share a triangle with e in $\mathcal{T}_{current}$.
- (ii) Flip e , and move to one of the 4 edges that shared a triangle with e in $\mathcal{T}_{current}$.
- (iii) Flip e , push the edge created by the flip into the stack, and move to one of the 4 edges that shared a triangle with e in $\mathcal{T}_{current}$.

- (iv) Flip e , jump to the edge on the top of the stack.
- (v) Flip e , jump to the edge on the top of the stack, and pop the stack.

A *walk* starting from an edge e in a triangulation is a sequence of actions all of which are of type (i). Since there are 4 choices for each action of types (i)-(iii) and 1 choice for each action of types (iv)-(v), we have:

► **Proposition 8.** The number of choices for any action by the algorithm is at most 14.

4.3 The sequence of actions on a component of \mathcal{D}_F

Let $F = \langle f_1, \dots, f_k \rangle$ be a normalized solution to an instance of FLIP DISTANCE. Let C be a component of \mathcal{D}_F . By Corollary 4, we can assume that all the flips in C appear at the beginning of F , that is, form a prefix of F ; let $F_C = \langle f_1, \dots, f_t \rangle$ be the prefix of F corresponding to the flips in C . This subsection is dedicated to proving the following theorem:

► **Theorem 9.** *Let C be a component of \mathcal{D}_F . There is a sequence of actions for the nondeterministic algorithm of length at most $11|V(C)|$ that, starting from a changed edge $\varepsilon(f_h)$ for some $f_h \in C$, performs the flips in C in a topologically-sorted order.*

To prove the above theorem, we define a spanning subgraph J_C of the underlying graph of C recursively. We then exhibit a sequence of actions of the algorithm that can be depicted by a recursive traversal of J_C . By that we mean that the actions performed by the algorithm in the triangulations correspond to a traversal of the edges and nodes of J_C , and such that the sequence of edge-flips performed by the algorithm is a topological sorting of C . We initialize J_C to be empty, and we start the recursive definition of J_C at a node in C that corresponds to a changed edge in the current triangulation. We will then add edges and nodes to J_C , and recurse on the connected components of the graph resulting from C after a source node in C has been removed. Since during the recursion nodes and edges get removed from C , the resulting graph of C may consist of several connected components that we will refer to as *chunks*, in order to distinguish them from the components of \mathcal{D}_F . Assume that the current triangulation is \mathcal{T} when we are recursing on a chunk H to define its spanning subgraph J_H . The recursive call starts at a node f_h in H that we call the *entry point* of H . At the top level of the recursion, C is the only chunk (in the recursive definition), and the entry point of $H = C$ is a node in C corresponding to a changed edge in the current triangulation. We will define in Lemma 10 a directed path $B = \langle b_1 = f_s, \dots, b_\ell = f_h \rangle$ in H from a source node f_s in H to the entry point f_h of H . With the path B , we correspond a walk W , defined in Lemma 10, that the algorithm performs in the current triangulation from $\varepsilon(f_h)$ to $\varepsilon(f_s)$. We add B to J_H , we add the edges between f_s and the entry point of each chunk in $H - f_s$ to J_H , and we recurse on the chunks of $H - f_s$ to complete the recursive definition of J_H . The corresponding actions of the algorithm (with the recursive definition of J_H) consist of performing the walk W , flipping $\varepsilon(f_s)$, and recursively performing the sequence of actions corresponding to the traversals of the chunks in $H - f_s$. Note that to flip a single edge, the algorithm takes a walk in the current triangulation to a source node in C . Therefore, if we are not careful in how we do the traversal of C , the length of all these walks could be quadratic in k . To ensure that when the algorithm is done performing the sequence of actions in a chunk it can go back to continue with the other chunks, the algorithm uses a stack to store the edge $\varphi(f_s)$, resulting from flipping the “connecting node” f_s of all these chunks, so that the algorithm, after performing all the flips in a chunk of $H - f_s$, can go back by a single action to $\varphi(f_s)$. We start with the following lemma:

► **Lemma 10.** *Let f_h be a node in a chunk H such that $\varepsilon(f_h)$ is an edge in the current triangulation \mathcal{T} . There is a walk W in \mathcal{T} from $\varepsilon(f_h)$ to an edge $\varepsilon(f_s)$ in \mathcal{T} , where f_s is a source node of H , such that there is a directed path B from f_s to f_h in H that we refer to as the backbone of H . Moreover, the length of the walk W is at most the length of B .*

Proof. If f_h is a source in H then $f_h = f_s$, the path B consists of f_s , and the length of the walk W is 0. The statement is trivially true in this case. Now assume that f_h is not a source node in H .

Since $\varepsilon(f_h)$ is an edge in \mathcal{T} , let Q be the quadrilateral associated with $\varepsilon(f_h)$ in \mathcal{T} . Since f_h is not a source in H and F is a minimal solution, one of the edges on the boundary of Q must be flipped before f_h ; let f_p be the first such flip in H . Since $\varepsilon(f_p)$ and $\varepsilon(f_h)$ share a triangle in \mathcal{T}_{p-1} , $\varphi(f_p)$ and $\varepsilon(f_h)$ share a triangle in \mathcal{T}_p , and by Lemma 7, there is a directed path from f_p to f_h in the component C of \mathcal{D}_F . Since the nodes removed from C during the recursive definition are always source nodes in their current chunks, there is a directed path from f_p to f_h in the current chunk H . The edges $\varepsilon(f_h)$ and $\varepsilon(f_p)$ share a triangle in \mathcal{T} , and hence, in one action (of type (i)) the algorithm can go from $\varepsilon(f_p)$ to $\varepsilon(f_h)$ in \mathcal{T} .

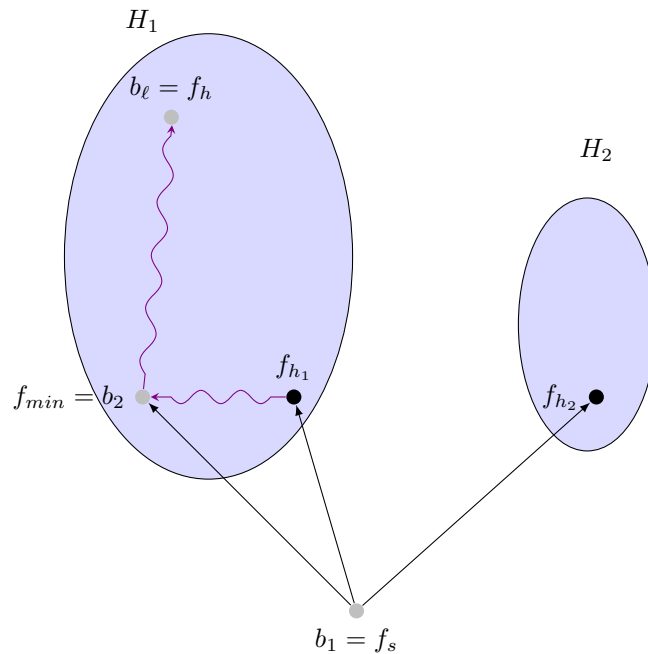
If f_p is a source node in H , then we are done; otherwise, applying the above argument to f_p , we can find a flip f_q such that $\varepsilon(f_p)$ and $\varepsilon(f_q)$ share a triangle in \mathcal{T} and there is a directed path from f_q to f_p in H . We can repeat this process until we reach a source node f_s in H . Going from $\varepsilon(f_h)$ to $\varepsilon(f_s)$ in \mathcal{T} involves only actions of type (i), and hence, defines a walk W from $\varepsilon(f_h)$ to $\varepsilon(f_s)$ in \mathcal{T} . The length of W is at most the total number of flips in a directed path B from f_s to f_h in H , which is composed of the directed paths defined in the process described above (from f_p to f_h , f_q to f_p , and so on). ◀

We now formally give the recursive definition of J_C , described for a chunk H with entry point f_h of a graph resulting from C during the recursion. Recall that at the top level of the recursion $H = C$, and f_h is a node in C corresponding to changed edge in the starting triangulation.

► **Definition 11.** Let H be a chunk with entry point f_h . The subgraph J_H of H is defined recursively as follows.

- (1) Let $B = \langle f_s = b_1, \dots, b_\ell = f_h \rangle$, where f_s is a source node in H (possibly $f_h = f_s$), be the backbone of H defined in Lemma 10.
- (2) Remove f_s from H and let H_1, \dots, H_x be the chunks of $H^- = H - f_s$; define f_s to be the *connecting point* to each of the chunks H_1, \dots, H_x .
- (3) For each chunk H_p , $p = 1, \dots, x$, if H_p contains nodes from previously-defined backbones during the recursive definition, then let f_{min} be the node in H_p of minimum index (with respect to F) that belongs to a previously-defined backbone; define the *entry point* of H_p to be the node f_{h_p} in H_p that is adjacent to f_s and that has a path to f_{min} in H_p , and in case more than one neighbor of f_s satisfies this property pick the neighbor with the minimum index with respect to F (we will prove in Lemma 12 that the node f_{h_p} is well defined). Otherwise (H_p does not contain nodes from previously-defined backbones), define the entry point of H_p to be the flip f_{h_p} in H_p with the minimum index h_p (with respect to F) that is adjacent to f_s . (See Figure 1 for illustration.)
- (4) Define the subgraph J_p of H_p with entry point f_{h_p} recursively, for $p = 1, \dots, x$.
- (5) Define J_H to be the union of the edges in B , the edges in J_p and the edges between f_s and each entry point of H_p , for $p = 1, \dots, x$.

Let J_C be the subgraph of the underlying graph of C resulting from applying the above recursive definition to C starting at a flip corresponding to a changed edge in C . We have:



■ **Figure 1** Illustration of the definition of entry points, where backbone nodes are colored gray. The entry point of H_1 is the node f_{h_1} in H_1 adjacent to f_s that has a path to the backbone node f_{min} with the minimum index (node b_2 in this case). The entry point of H_2 , which does not contain backbone nodes in this case, is the node f_{h_2} of minimum index (h_2) that is adjacent to f_s .

► **Lemma 12.** *All the backbones, defined during the recursive definition of J_C , that exist in the same chunk are edge disjoint, and belong to a single (simple) path in the chunk; on this path the (remaining) nodes from each backbone appear consecutively. Moreover, the entry node of a chunk, defined in step 3 of Definition 11, is well-defined.*

Proof. The proof is by induction on the number of recursive steps (depth of the recursion) taken to form a chunk. The statement is clearly true at the top level of the recursion where the only chunk is C , whose entry point is defined to be a flip corresponding to a changed edge, and there is only one defined backbone. Suppose now that chunk H_p resulted from a chunk H in one recursive step, and that the statement is true for H (inductive hypothesis).

H_p was obtained by removing a source node f_s from H , which is a backbone node. By the inductive hypothesis, all the backbones in H are edge-disjoint and belong to a path P in H . Since f_s is a source node in H , f_s must be the tail of P . If H_p does not contain any previously-defined backbone nodes, then the entry point f_{h_p} of H_p is defined to be the node in H_p with the minimum index that is adjacent to f_s , and in this case f_{h_p} is well-defined. Moreover, there is only one backbone in H_p . Therefore, the statement of the lemma is true in this case. Suppose now that H_p contains at least one node from a previously-defined backbone. Because the underlying graph of H_p is connected and $P^- = P - f_s$ is a path, it follows that H_p contains P^- . Let b be the node adjacent to f_s on P^- , *i.e.*, the tail of P^- . By the inductive hypothesis, P^- contains all the previously-defined backbone nodes in H_p , and in particular, P^- contains the node f_{min} in H_p of minimum index (the minimum index is with respect to F) that belongs to a previously-defined backbone. Since b is adjacent to f_s , it follows from the preceding that node f_{h_p} is well-defined because b satisfies that it is

adjacent to f_s and there is a path from b to the backbone node in H with the minimum index, namely f_{min} (possibly b itself). Now let B_{H_p} be the backbone of H_p . Since B_{H_p} is a path whose head is f_{h_p} , and since — by the choice of f_{h_p} — there is a path from f_{h_p} to f_{min} , the indices of the nodes in B_{H_p} are not larger than the index of f_{min} , which is the backbone node on P^- of the minimum index. Therefore, the set of edges in B_{H_p} is disjoint from the set of backbone edges on P^- (and hence, from the set of backbone edges in H_p) that belong to previously-defined backbones. Since B_{H_p} is a path in H_p , and since there is a path from f_{h_p} to f_{min} in H_p , all the backbone edges in H_p form a path in which all the (remaining) nodes of each backbone appear consecutively. This completes the inductive proof. ◀

► **Corollary 13.** *All the backbones defined in the recursive definition of J_C are edge-disjoint.*

Proof. It suffices to show that when a backbone B of a chunk H is defined during the recursive definition of J_C , the edges of B are different from the edges of all previously-defined backbones. Clearly, the edges of B are different from those of the backbones in chunks other than H , and from the edges of previously-defined backbones that have been previously removed during the recursive definition of J_C . By Lemma 12, the edges of B are different from those of the backbones other than B that (may) exist in H . The statement follows. ◀

► **Lemma 14.** *Let C be a component of $\mathcal{D}_{\mathcal{F}}$. The subgraph J_C formed by applying Definition 11 to C is a spanning subgraph of the underlying graph of C .*

Proof. The statement follows from the connectedness of C and Definition 11 by a simple inductive argument: J_C contains a source node f_s of C and an edge from f_s to each chunk in $C - f_s$. ◀

We define next a sequence of actions that the algorithm performs starting at a changed edge (corresponding to a node in C) in the current triangulation and that corresponds to a traversal of J_C . Let f_i, f_h be the connecting and the entry points to a chunk $H \neq C$, respectively. At the top level of the recursion, where $H = C$ is a component of $\mathcal{D}_{\mathcal{F}}$, define f_h to be a flip in C whose underlying edge $\varepsilon(f_h)$ is a changed edge (f_i need not be defined).

► **Definition 15.** Let H be a chunk with entry point f_h . The sequence of actions of the nondeterministic algorithm on H is defined as follows.

- (a) The nondeterministic algorithm performs the walk W from $\varepsilon(f_h)$ to $\varepsilon(f_s)$ (in the current triangulation \mathcal{T}) defined in Lemma 10 that corresponds to the backbone $B = \langle f_s = b_1, \dots, b_\ell = f_h \rangle$ of H .
- (b) The nondeterministic algorithm flips the edge $\varepsilon(f_s)$.
- (c) The algorithm nondeterministically pushes $\varphi(f_s)$ into the stack if there is more than one chunk in $H^- = H - f_s$, and moves to the entry point of the first chunk in H^- .
- (d) The nondeterministic algorithm recursively performs the sequence of actions on each chunk of H^- , nondeterministically moving to the edge $\varphi(f_s)$ on the top of the stack when performing the last action in each chunk, and following that with a move (if needed) to the underlying edge of the entry point of a new chunk, which shares a triangle with $\varphi(f_s)$ (or is identical to it) by Lemma 16 below.
- (e) The algorithm nondeterministically moves to the top of the stack and pops the stack after performing the last action in the last chunk of H^- (in case there is more than one).

► **Lemma 16.** *Let f_i, f_h be the connecting and entry points to a chunk $H \neq C$, respectively. Suppose that the current triangulation is \mathcal{T} when the sequence of actions of the algorithm defined in Definition 15 is applied on H . Then either $\varphi(f_i) = \varepsilon(f_h)$, or $\varphi(f_i)$ and $\varepsilon(f_h)$ share a triangle in \mathcal{T} .*

Proof (Theorem 9). It is clear that the order of the flips performed by the algorithm in the sequence of actions described in Definition 15 corresponds to a topological sorting of C because every flip corresponds to the removal of a source node from a DAG resulting from C in the recursive definition of J_C , and because J_C is a spanning subgraph of C by Lemma 14. Therefore, it suffices to show that this sequence has length at most $11|V(C)|$. To do so, we charge the actions of the algorithms to the nodes and edges of J_C .

When invoked on a chunk H , the algorithm starts at an entry node f_h of H ; initially $H = C$ and f_h is a node in C whose underlying edge is a changed edge in the current triangulation \mathcal{T} . In Lemma 10 we showed that there is a path $B = \langle f_s = b_1, \dots, b_\ell = f_h \rangle$ from a source node f_s in H to f_h that corresponds to a walk by the algorithm from edge $\varepsilon(f_h)$ to $\varepsilon(f_s)$ in \mathcal{T} ; moreover, the length of this walk is at most the length of B . The algorithm can perform this walk using actions of type (i) (as defined in Subsection 4.2), and the number of such actions is at most the length of B . When the algorithm is at edge $\varepsilon(f_s)$ in \mathcal{T} , it flips edge $\varepsilon(f_s)$, which is one action either of type (ii) or (iii). Next, the algorithm recurses on each chunk H_p of $H - f_s$, starting at the entry point f_{h_p} of H_p . In Lemma 16, we showed that the edges $\varphi(f_s)$ and $\varepsilon(f_{h_p})$ are either identical, or they share a triangle in the current triangulation when the algorithm is recursively called on H_p . Hence, in at most one action the algorithm can move from $\varphi(f_s)$ to $\varepsilon(f_{h_p})$. If there is more than one chunk in $H - f_s$ (the algorithm nondeterministically decides), the algorithm pushes $\varphi(f_s)$ into the stack after flipping f_s . In case there is only one chunk left, the algorithm also pops the stack after jumping to the top of the stack. It is not difficult to see that each of the steps corresponds to one action of the algorithm from types (i)-(v).

To prove that the length of the sequence of actions is at most $11|V(C)|$, we charge these actions to the nodes and edges of J_C . The sequence of actions can be classified into two categories: actions with flips and actions without flips. The number of actions with flips is at most the number of nodes in J_C , which is $|V(C)|$. Note that actions that involve moving to the top of the stack, or popping the stack, or both, are combined with flips, and hence have been accounted for. The actions without flips are all of type (i), and can be further divided into two groups: (I) those done in a walk W corresponding to a backbone B of a chunk H , and (II) those done when the algorithm moves from an underlying edge $\varphi(f_s)$ (on the top of the stack) of a source node f_s in a chunk H to an edge whose corresponding node is an entry point of a chunk resulting from removing f_s from H . The number of actions in group (I) is at most $|E(C)|$; this is because, by Corollary 13, the edges of different backbones are distinct, and hence the total number of such edges (and hence actions in group (I)) is at most $|E(J_C)| \leq |E(C)|$. To bound the number of actions in group (II), observe that each such action corresponds to an edge in C from f_s to the entry point of a chunk resulting from removing f_s from H . Since f_s is removed from J_C upon making the recursive calls to the resulting chunks, we can charge each such action in a one-to-one fashion to edges of $E(J_C)$. Therefore, the number of actions in group (II) is at most $|E(J_C)| \leq |E(C)|$. Therefore, the total number of actions of type (i) is at most $2|E(J_C)| \leq 2|E(C)|$. It follows that the total number of actions performed by the algorithm when applied to C is at most $|V(C)| + 2|E(J_C)| \leq 11|V(C)|$ (by Lemma 2). ◀

4.4 Putting all together: the whole algorithm

Let F be a normalized solution to the instance $(\mathcal{T}_{initial}, \mathcal{T}_{final}, k)$. Order the changed edges arbitrarily, and denote this ordering by \mathcal{O} . The algorithm starts by guessing the number of components t , where $t \leq k$, in \mathcal{D}_F . The algorithm then guesses the number of flips

k_1, \dots, k_t in the components C_1, \dots, C_t , respectively, of \mathcal{D}_F satisfying $k_1, \dots, k_t \geq 1$ and $k_1 + k_2 + \dots + k_t = k$. Fix such a guess (k_1, \dots, k_t) .

The algorithm performs t iterations: $\ell = 1, \dots, t$. We define $\mathcal{T}_{initial}^\ell$, $\ell = 1, \dots, t$, to be the triangulation that resulted from $\mathcal{T}_{initial}$ after the flips in the first ℓ iterations are performed. We define $\mathcal{T}_{initial}^0 = \mathcal{T}_{initial}$. For each $\ell = 1, \dots, t$, do the following: Pick the next edge $e \in \mathcal{O}$. If e is not a changed edge anymore with respect to $\mathcal{T}_{initial}^{\ell-1}$ and \mathcal{T}_{final} , then skip to the next edge in \mathcal{O} . Otherwise (e is in $\mathcal{T}_{initial}^{\ell-1}$ but not in \mathcal{T}_{final}), perform a sequence of actions starting from e in $\mathcal{T}_{initial}^{\ell-1}$ until either the number of flips performed is k_ℓ , or the number of actions performed reaches $11k_\ell$. Let F_ℓ be the sequence of flips performed in the current iteration, and note that $\mathcal{T}_{initial}^{\ell-1} \xrightarrow{F_\ell} \mathcal{T}_{initial}^\ell$. After the last iteration $\ell = t$, if $\mathcal{T}_{initial}^t = \mathcal{T}_{final}$ then accept; otherwise reject.

► **Theorem 17.** *Let $(\mathcal{T}_{initial}, \mathcal{T}_{final}, k)$ be an instance of FLIP DISTANCE. The above non-deterministic algorithm decides $(\mathcal{T}_{initial}, \mathcal{T}_{final}, k)$ correctly, and it can be simulated by a deterministic algorithm that runs in time $\mathcal{O}(n + k \cdot c^k)$.*

Proof. It is easy to see that the correctness of the algorithm follows from the following: (1) there is a guess for the algorithm of the correct number of components t , and of (k_1, \dots, k_t) such that k_i is the exact number of flips in C_i , $i = 1, \dots, t$; and (2) by Theorem 9, there is a nondeterministic sequence of actions by the algorithm of length at most $11k_i$ that, starting from a changed edge in C_i , performs the k_i flips in C_i in a topologically-sorted order.

We only need to analyze the deterministic running time needed to simulate the non-deterministic algorithm. The initial processing of the triangulations to find the changed edges takes $\mathcal{O}(n)$ time. The total number of sequences (k_1, \dots, k_t) , for $t = 1, \dots, k$, satisfying $k_1 + \dots + k_t = k$ and $k_1, \dots, k_t \geq 1$, is known as the *composition number* of (integer) k , and is equal to 2^{k-1} . For each such sequence (k_1, \dots, k_t) , we iterate through the numbers k_1, \dots, k_t in the sequence. For a number k_i , $1 \leq i \leq t$, by Theorem 9, we need to try every sequence of at most $11k_i$ actions, and in which each action is one of 14 choices (by Proposition 8). Therefore, the number of such sequences is at most 14^{11k_i} . It follows that the total number of sequences that the algorithm needs to enumerate to find a witness to the solution (if it exists) is at most: $\sum_{t=1}^k \sum_{(k_1, \dots, k_t)} (14^{11k_1} \times \dots \times 14^{11k_t}) = \mathcal{O}(2^{k-1} 14^{11k}) = \mathcal{O}(c^k)$, where $c \leq 2 \cdot 14^{11}$. Since each sequence of actions can be carried out in time $\mathcal{O}(k)$, and the resulting triangulation at the end of the sequence can be compared to \mathcal{T}_{final} in $\mathcal{O}(k)$ time as well, the running time for each enumerated sequence is $\mathcal{O}(k)$. It follows from the above that the running of the deterministic algorithm is $\mathcal{O}(n + k \cdot c^k)$. Finally we point out that the algorithm needs to decide whether k is the flip distance between $\mathcal{T}_{initial}$ and \mathcal{T}_{final} , which means that no sequence of flips of length smaller than k exists that transforms $\mathcal{T}_{initial}$ to \mathcal{T}_{final} . This can be decided by invoking the algorithm on each of the instances $(\mathcal{T}_{initial}, \mathcal{T}_{final}, k')$, for $k' = 0, \dots, k$. The running time remains $\mathcal{O}(n + k \cdot c^k)$ because $\sum_{k'=0}^k \mathcal{O}(k' \cdot c^{k'}) = \mathcal{O}(k \cdot c^k)$. ◀

5 Concluding remarks

Improving the upper bound $2 \cdot 14^{11}$ on the constant c in the running time of our algorithm to a small value is an important open problem. Another important open problem is investigating the kernelization of FLIP DISTANCE. One can obtain an exponential-size kernel based on the results in this paper, but the question of whether there is a polynomial-size kernel is important and challenging. Recall that a kernel of size $2k$ was given by Lucas [12] for the convex case. Finally, we note that FLIP DISTANCE falls broadly into the category of reconfiguration problems, for which several parameterized complexity results appeared very recently (see [13, 14, 15]).

References

- 1 O. Aichholzer, F. Hurtado, and M. Noy. A lower bound on the number of triangulations of planar point sets. *Computational Geometry*, 29(2):135–145, 2004.
- 2 O. Aichholzer, W. Mulzer, and A. Pilz. Flip distance between triangulations of a simple polygon is NP-complete. In *Proceedings of ESA*, volume 8125 of *Lecture Notes in Computer Science*, pages 13–24. Springer, 2013.
- 3 P. Bose and F. Hurtado. Flips in planar graphs. *Computational Geometry*, 42(1):60–80, 2009.
- 4 S. Cleary and K. St. John. Rotation distance is fixed-parameter tractable. *Information Processing Letters*, 109(16):918–922, 2009.
- 5 R. Diestel. *Graph Theory*. Springer, Berlin, 4th edition, 2010.
- 6 R. Downey and M. Fellows. *Parameterized Complexity*. Springer, New York, 1999.
- 7 S. Hanke, T. Ottmann, and S. Schuierer. The edge-flipping distance of triangulations. *Journal of Universal Computer Science*, 2(8):570–579, 1996.
- 8 F. Hurtado, M. Noy, and J. Urrutia. Flipping edges in triangulations. *Discrete & Computational Geometry*, 22(3):333–346, 1999.
- 9 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.
- 10 A. Lubiw and V. Pathak. Flip distance between two triangulations of a point set is NP-complete. In *Proceedings of CCCG*, pages 119–124, 2012.
- 11 A. Lubiw and V. Pathak. Flip distance between two triangulations of a point set is NP-complete. arXiv.org e-Print archive, paper cs.CG/1205.2425, May 2012.
- 12 J. Lucas. An improved kernel size for rotation distance in binary trees. *Information Processing Letters*, 110(12):481–484, 2010.
- 13 A. Mouawad, N. Nishimura, and V. Raman. Vertex cover reconfiguration and beyond. In *Proceedings of ISAAC*, volume 8889 of *Lecture Notes in Computer Science*, pages 452–463. Springer, 2014.
- 14 A. Mouawad, N. Nishimura, V. Raman, N. Simjour, and A. Suzuki. On the parameterized complexity of reconfiguration problems. In *Proceedings of IPEC*, volume 8246 of *Lecture Notes in Computer Science*, pages 281–294. Springer, 2013.
- 15 A. Mouawad, N. Nishimura, V. Raman, and M. Wrochna. Reconfiguration over tree decompositions. In *Proceedings of IPEC*, volume 8894 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2014.
- 16 R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, USA, 2006.
- 17 A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, New York, NY, USA, 1992.
- 18 A. Pilz. Flip distance between triangulations of a planar point set is APX-hard. *Computational Geometry*, 47(5):589–604, 2014.
- 19 A. Saalfeld. Joint triangulations and triangulation maps. In *Proceedings of SoCG*, pages 195–204. ACM, 1987.
- 20 L. Schumaker. Triangulations in CAGD. *IEEE Computer Graphics and Applications*, 13(1):47–52, 1993.
- 21 R. Sibson. Locally equiangular triangulations. *The Computer Journal*, 21(3):243–245, 1978.
- 22 D. Sleator, R. Tarjan, and W. Thurston. Rotation distance, triangulations, and hyperbolic geometry. In *Proceedings of STOC*, pages 122–135. ACM, 1986.
- 23 K. Wagner. Bemerkungen zum Vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936.
- 24 D. Watson and G. Philip. Systematic triangulations. *Computer Vision, Graphics, and Image Processing*, 22(2):310, 1983.

New Pairwise Spanners

Telikepalli Kavitha

Tata Institute of Fundamental Research, India

kavitha@tcs.tifr.res.in

Abstract

Let $G = (V, E)$ be an undirected unweighted graph on n vertices. A subgraph H of G is called an (all-pairs) purely additive spanner with stretch β if for every $(u, v) \in V \times V$, $\text{dist}_H(u, v) \leq \text{dist}_G(u, v) + \beta$. The problem of computing sparse spanners with small stretch β is well-studied. Here we consider the following relaxation: we are given $\mathcal{P} \subseteq V \times V$ and we seek a sparse subgraph H where $\text{dist}_H(u, v) \leq \text{dist}_G(u, v) + \beta$ for each $(u, v) \in \mathcal{P}$. Such a subgraph is called a pairwise spanner with additive stretch β and our goal is to construct such subgraphs that are sparser than all-pairs spanners with the same stretch. We show sparse pairwise spanners with additive stretch 4 and with additive stretch 6. We also consider the following special cases: $\mathcal{P} = S \times V$ and $\mathcal{P} = S \times T$, where $S \subseteq V$ and $T \subseteq V$, and show sparser pairwise spanners for these cases.

1998 ACM Subject Classification G.2.2 Graph Algorithms

Keywords and phrases undirected graphs, spanners, approximate distances, additive stretch

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.513

1 Introduction

Let $G = (V, E)$ be an undirected unweighted graph on n vertices. A subgraph H of G is called a spanner with multiplicative stretch α and additive stretch β if for every pair $(u, v) \in V \times V$, we have $\delta_H(u, v) \leq \alpha \cdot \delta_G(u, v) + \beta$, where $\delta_G(u, v)$ (similarly, $\delta_H(u, v)$) is the u - v distance in G (resp., H). The objective in spanner problems is to construct a subgraph H that is as sparse as possible, however for each pair of vertices (u, v) , the u - v distance in H is close to the u - v distance in G , i.e., the stretch is small.

When the multiplicative stretch α is 1 and the additive stretch β is $O(1)$, the spanner H is said to be *purely additive*. We currently know purely additive spanners only for additive stretches 2, 4, and 6. The additive stretch 2 spanner has size $O(n^{3/2})$ [1], the additive stretch 4 spanner has size $\tilde{O}(n^{7/5})$ [10], and the additive stretch 6 spanner has size $O(n^{4/3})$ [6]. In this paper we consider the problem of obtaining sparser subgraphs for the following relaxed problem: distances for all pairs in $V \times V$ need not be well-approximated in the subgraph – here we have a subset $\mathcal{P} \subseteq V \times V$ of *critical* pairs and we seek a subgraph H where $\delta_H(u, v) \leq \delta_G(u, v) + O(1)$ for every $(u, v) \in \mathcal{P}$.

Our goal is to find a subgraph H that is sparser (for a large range of values of $|\mathcal{P}|$) than the all-pairs spanner with the same additive stretch. This problem was first studied by Coppersmith and Elkin [12] who sought subgraphs where distances for pairs in \mathcal{P} were exactly preserved. They called such subgraphs *pairwise preservers* and showed such subgraphs of size $O(\min\{n\sqrt{|\mathcal{P}|}, n + |\mathcal{P}|\sqrt{n}\})$ for any $\mathcal{P} \subseteq V \times V$. They left it as an open question to study the approximate variants of pairwise preservers.

This question was studied by Cygan et al. [15] who called such subgraphs *pairwise spanners* or \mathcal{P} -spanners. A trade-off between the additive stretch and size of a \mathcal{P} -spanner was shown in [15], where the least stretch \mathcal{P} -spanner had additive stretch 4 with size $O(n^{4/3}|\mathcal{P}|^{1/6})$ and the sparsest \mathcal{P} -spanner had size $\tilde{O}(n|\mathcal{P}|^{1/4})$ with additive stretch 4 log n .

Subsequently, a pairwise spanner of size $\tilde{O}(n|\mathcal{P}|^{1/3})$ and additive stretch 2 was shown in [20]. We show the following results here.

► **Theorem 1.** *Let $G = (V, E)$ be an undirected unweighted graph. Then for any $\mathcal{P} \subseteq V \times V$, the following subgraphs can be constructed in polynomial time (where $|V| = n$):*

1. a \mathcal{P} -spanner of size $\tilde{O}(n \cdot |\mathcal{P}|^{2/7})$ and additive stretch 4,
2. a \mathcal{P} -spanner of size $O(n \cdot |\mathcal{P}|^{1/4})$ and additive stretch 6.

Comparing our pairwise spanners with pairwise preservers and all-pairs spanners: the pairwise spanner in Theorem 1.1 is sparser than the $\tilde{O}(n^{7/5})$ -sized all-pairs spanner with additive stretch 4 when $|\mathcal{P}|$ is $o(n^{7/5})$ and it is sparser than the $O(n + |\mathcal{P}|\sqrt{n})$ -sized pairwise preserver when $|\mathcal{P}|$ is $\omega(n^{7/10})$. The pairwise spanner in Theorem 1.2 is sparser than the above pairwise preserver when $|\mathcal{P}|$ is $\omega(n^{2/3})$ and it is sparser than the $O(n^{4/3})$ -sized all-pairs spanner with additive stretch 6 when $|\mathcal{P}|$ is $o(n^{4/3})$.

A natural setting for $\mathcal{P} \subseteq V \times V$ is when the set \mathcal{P} of relevant pairs is $S \times T$, where S is one set of endpoints or *sources* and T is another set of endpoints or *destinations* and we seek a subgraph where s - t distances are well-approximated for every $(s, t) \in S \times T$. The case where $S = T$ was considered earlier and an $(S \times S)$ -spanner of size $O(n\sqrt{|S|})$ with additive stretch 2 is known [15, 25]. Here we seek $(S \times T)$ -spanners that are sparser than an $(S \cup T) \times (S \cup T)$ -spanner or an $(S \times V)$ -spanner (for instance, when $|S| \ll |T| \ll n$). To the best of our knowledge, $(S \times T)$ -spanners are being studied for the first time here and we will refer to them as *ST-spanners*. We show the following result on *ST-spanners*.

► **Theorem 2.** *For any subsets S and T of V , an ST -spanner of size $O(n \cdot (|S||T|)^{1/4})$ and additive stretch 4 can be constructed in polynomial time.*

Thus the above result combines the size of the pairwise spanner in Theorem 1.2 with the stretch of the one in Theorem 1.1, in other words, we get a \mathcal{P} -spanner of size $O(n|\mathcal{P}|^{1/4})$ with additive stretch 4 when $\mathcal{P} = S \times T$. Our next theorem shows that Theorem 1 can be further improved for *sourcewise spanners*, i.e., when $\mathcal{P} = S \times V$. Sourcewise spanners form a natural and interesting class of general \mathcal{P} -spanners and *ST-spanners*.

► **Theorem 3.** *The following subgraphs can be constructed in polynomial time for any $S \subseteq V$:*

1. an $(S \times V)$ -spanner of size $\tilde{O}(n \cdot (n|S|)^{2/9})$ and additive stretch 4,
2. an $(S \times V)$ -spanner of size $O(n \cdot (n|S|)^{1/5})$ and additive stretch 6.

Sourcewise spanners of size $\tilde{O}(n \cdot (n|S|)^{1/4})$ and additive stretch 2 were shown in [20]. Trade-off results between the size and additive stretch of sourcewise spanners were shown in [15, 21]. We show a size vs additive stretch trade-off for *ST-spanners* in Theorem 4.

► **Theorem 4.** *For any integer $k \geq 1$ and $S, T \subseteq V$, an ST -spanner with additive stretch $2k$ and size $\tilde{O}(n \cdot (|S|^\gamma |T|)^{1/(2\gamma+1)})$, where $\gamma = k + 1$, can be constructed in polynomial time.*

By setting T to be the set of *cluster centers* (defined in Section 2) in Theorem 4, we get for $k \geq 2$, sourcewise spanners with additive stretch $2k$ and size $\tilde{O}(n^{1+1/r} |S|^{k/r})$, where $r = 2k + 2$. This matches the current best trade-off for sourcewise spanners (from [21]). An advantage with our construction is that it is deterministic, hence the bound is on the worst case size of the sourcewise spanner constructed here while the construction in [21] was randomized, hence the bound there was on the expected size of the sourcewise spanner.

An all-pairs purely additive spanner of size $O(n^{1+\frac{2\delta}{2\delta+1}})$ can be obtained from a \mathcal{P} -spanner with additive stretch $O(1)$ and size $O(n \cdot |\mathcal{P}|^\delta)$ by taking $\mathcal{P} = T \times T$, where $T = \{\text{cluster centers}\}$; thus if $\delta < 1/4$, we get an all-pairs purely additive spanner of size $O(n^{1+\epsilon})$, where $\epsilon < 1/3$. No such purely additive spanners are currently known.

1.1 Background and Related Results

Graph spanners were introduced by Peleg and Schaffer [23] in 1989. The problem of efficiently constructing sparse spanners with a small multiplicative stretch is well-understood: Althöfer et al. [2] in 1993 showed that in any weighted graph G on n vertices and for any integer $k \geq 1$, there is a spanner of size $O(n^{1+1/k})$ with multiplicative stretch $2k - 1$. More efficient constructions of spanners in weighted graphs can be found in [8, 27, 26]. There are several applications in graph/network algorithms that use spanners: for instance, approximate shortest paths [3, 11, 17], approximate distance oracles [28, 7, 5], labeling schemes [22, 19], network design [24], routing [4, 13, 14].

For unweighted graphs, we seek spanners with purely additive stretch – the first such spanner was by Aingworth et al. [1] (with followup work in [16, 18]) which showed a spanner with additive stretch 2 and size $O(n^{3/2})$. The additive stretch 6 spanner is due to Baswana et al. [6] and the additive stretch 4 spanner is due to Chechik [10]. The current best trade-off between sparsity and additive stretch is from [10]: for any $\delta \in [3/17, 1/3)$, there is a subgraph of size $\tilde{O}(n^{1+\delta})$ and additive stretch $\tilde{O}(n^{(1-3\delta)/2})$.

Bollobás et al. [9] were the first to study a variant of pairwise preservers – they studied D -preservers where the problem was to compute a sparse pairwise preserver for the set $\mathcal{P} = \{(u, v) : \delta_G(u, v) \geq D\}$. As mentioned earlier, Coppersmith and Elkin [12] studied the general problem of pairwise preservers for any $\mathcal{P} \subseteq V \times V$.

The study of pairwise spanners was initiated by Cygan et al. [15] who showed the following trade-off for pairwise spanners – for any $\mathcal{P} \subseteq V \times V$ and integer $k \geq 1$: additive stretch $4k$ and size $O(n^{1+1/r} \cdot (k|\mathcal{P}|)^{k/2r})$, where $r = 2k + 1$ and the following trade-off between size and additive stretch for sourcewise spanners – for any subset S and any integer $k \geq 1$: additive stretch $2k$ and size $O(n^{1+1/r}(k|S|)^{k/r})$, where $r = 2k + 1$. Parter [21] showed sparse *multiplicative* sourcewise spanners and a lower bound of $\Omega(n|S|^{1/k}/k)$ on the size of a sourcewise spanner with additive stretch $2(k - 1)$, for any integer $k \geq 1$.

1.2 Techniques

All our algorithms start with the clustering step where vertices are grouped into *clusters* and at the end of this step, we are left with a post-clustering subgraph that contains all edges of G , except some inter-cluster edges. In each of our algorithms, the rest of the algorithm has to decide which of these missing inter-cluster edges should get added to the post-clustering subgraph. We use the steps of path-buying and path-hitting to make these decisions.

The *path-buying* step was introduced in [6] and has been used in several spanner algorithms [15, 20, 21]; here each shortest path is “evaluated” and if it is affordable, the path is bought, i.e., its missing edges are added to the current subgraph. Otherwise it will have to be the case that the path is already well-approximated in the current subgraph. We use path-buying with appropriate value functions in our algorithms for pairwise/sourcewise spanner with additive stretch 6 and *ST*-spanner with additive stretch 4.

We use *path-hitting* in our other algorithms. Path-hitting was first seen in the near-quadratic time algorithm in [29] for an $\tilde{O}(n^{4/3})$ -sized spanner with additive stretch 6. This technique aims at hitting the neighborhood of each shortest path: this was done in [29] by randomly sampling vertices and appropriate near-shortest paths between the sampled vertices and other vertices were added to form the spanner. In our path-hitting subroutines here, we select a small number of clusters so that certain critical subpaths of all our relevant shortest paths are hit, i.e., for each such subpath, there will be some cluster among our selected ones that intersects it. Finally, our subgraph will contain shortest paths between

appropriate pairs of selected clusters.

Section 2 describes the clustering step and other preliminaries. Theorems 1 and 2 are shown in Section 3. Section 4 has our results on sourcewise spanners and Section 5 has the trade-off result for ST -spanners. Due to space constraints, the proofs of some lemmas are omitted; they will be included in the full version of the paper.

2 Preliminaries

A *clustering* of a graph $G = (V, E)$ is a collection $\{C_1, \dots, C_r\}$ where each C_i is a subset of vertices and $C_i \cap C_j = \emptyset$ for $i \neq j$. Note that we do not require $\cup_i C_i$ to be equal to V ; the vertices in $V - \cup_i C_i$ will be called *unclustered*. Associated with each cluster C_i is a vertex called its *center*, denoted by $\text{center}(C_i)$, with the following property: every vertex in C_i is a neighbor of $\text{center}(C_i)$. Note that $\text{center}(C_i) \notin C_i$.

Given a graph G and an integer h , where $1 \leq h \leq n$, the following simple procedure constructs a clustering $\mathcal{C}_h = \{C_1, \dots, C_r\}$ and returns $\langle \mathcal{C}_h, U \rangle$, where $U = V - \cup_i C_i$ is the set of unclustered vertices.

– Initially all vertices are unclustered and \mathcal{C}_h is empty. While there is a vertex v with at least h unclustered neighbors, we form a new cluster C by picking any h of these neighbors of v and these vertices are now clustered; v becomes $\text{center}(C)$ and C gets added to \mathcal{C}_h . When no vertex has h or more unclustered neighbors, this procedure terminates and returns $\langle \mathcal{C}_h, U \rangle$, where U is the set of unclustered vertices.

It is easy to see that every pair of clusters is disjoint and each cluster C is a collection of h vertices. So $|\mathcal{C}_h|$, the number of clusters, is at most n/h . Associated with \mathcal{C}_h is a post-clustering subgraph G_h that consists of all the edges incident to unclustered vertices, all *intra-cluster* edges (i.e., edges (u, v) where both u and v belong to the same cluster), as well as the edges (v, c) , where v is a clustered vertex and c is the center of v 's cluster. It can be shown that G_h has $O(nh)$ edges. We define the *cost* of a path below.

► **Definition 5.** For any path ρ in G , let $\text{cost}(\rho)$ denote the number of edges of ρ that are missing in the post-clustering subgraph G_h .

The following lemma from [15] gives a lower bound on the number of distinct clusters that are incident on a shortest path ρ whose cost is t or more. We say a cluster C and a shortest path ρ intersect each other if C and ρ have at least one vertex in common.

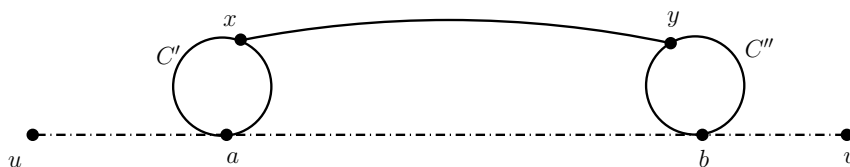
► **Lemma 6** (from [15]). *Let ρ be a shortest path in G with $\text{cost}(\rho) \geq t$. Then there are at least $t/2$ distinct clusters of \mathcal{C}_h that intersect ρ .*

Another subroutine that our algorithms will use is that of efficiently computing a small-sized “hitting set” $\mathcal{A} \subseteq \mathcal{C}_h$ for a set Q of paths. We define such a set \mathcal{A} below.

► **Definition 7.** Let Q be a set of paths. Then $\mathcal{A} \subseteq \mathcal{C}_h$ is a *hitting set* for Q if for every path $q \in Q$, there is at least one cluster in \mathcal{A} that intersects q .

When each $q \in Q$ has several clusters incident on it, a small-sized hitting set can be computed easily by the greedy algorithm. We know from Lemma 6 that if a shortest path has many missing edges in G_h , then it has several distinct clusters incident on it. Lemma 8 uses this property to obtain a small-sized hitting set for such a set Q of paths.

► **Lemma 8.** *Let Q be a set of shortest paths such that each $q \in Q$ has at least λ missing edges in G_h . Then the greedy algorithm finds a hitting set $\mathcal{A} \subseteq \mathcal{C}_h$ of size $O(n/(h\lambda) \cdot \log |Q|)$ for Q .*



■ **Figure 1** If $\delta_H(a, b) > \delta_G(a, b) + 4$ before Step 5 of our algorithm, then we would have bought a path $SP(x, y)$ of length at most $\delta_G(a, b)$ between C' and C'' in Step 5. The u - v path $u-a-x-y-b-v$ has length at most $\delta_G(u, v) + 4$.

It will be convenient to assume that there is a *unique* shortest path between any pair of vertices. So for every pair of vertices u and v , we will choose one u - v shortest path in G as *the* u - v shortest path (denoted by $SP(u, v)$) and we will ensure that every subpath of a chosen shortest path is again a chosen shortest path. Let $\mathcal{F} = \{\rho_1, \dots, \rho_{\binom{n}{2}}\}$ be the set of all these chosen shortest paths in G .

Let $Q \subseteq \mathcal{F}$ be the set of shortest paths p such that $\text{cost}(p) \geq (n \log n)/h^2$. Lemma 8 tells us that Q has a hitting set \mathcal{X}_h of size $O(h)$. Consider the step of augmenting the post-clustering subgraph G_h with the edges of BFS trees rooted at the centers of clusters in \mathcal{X}_h . As there are only $O(h)$ such trees, the total number of edges added by this augmentation is $O(nh)$. The following lemma from [20] is a useful consequence of this augmentation.

► **Lemma 9.** *Let u and v be a pair of vertices such that $\text{cost}(SP(u, v)) \geq (n \log n)/h^2$. Then the augmented post-clustering graph has a u - v path of length at most $\delta_G(u, v) + 2$.*

3 Pairwise Spanners

In this section we prove Theorems 1 and 2 stated in Section 1. We first describe the construction of a \mathcal{P} -spanner of size $\tilde{O}(n|\mathcal{P}|^{2/7})$ with additive stretch 4. The input is an undirected unweighted graph $G = (V, E)$ along with $\mathcal{P} \subseteq V \times V$.

Our algorithm starts by running the clustering step with an appropriate parameter h and augments the post-clustering graph G_h by adding BFS trees rooted at the centers of clusters in \mathcal{X}_h , where \mathcal{X}_h is an $O(h)$ -sized hitting set for $Q = \{\rho \in \mathcal{F} : \text{cost}(\rho) \geq (n \log n)/h^2\}$. Call the resulting graph H .

We are ready to buy 2ℓ new edges for each $(u, v) \in \mathcal{P}$, for an appropriate parameter ℓ . If $p = SP(u, v)$ is *expensive*, i.e., $\text{cost}(p) > 2\ell$, then we buy only a prefix p' and a suffix p'' of p such that $\text{cost}(p') = \text{cost}(p'') = \ell$. The main idea here is *path-hitting* – we find small-sized sets \mathcal{A} and \mathcal{B} of clusters so that there is a cluster $C' \in \mathcal{A}$ that intersects p' and there is a cluster in $C'' \in \mathcal{B}$ that intersects p'' .

For each pair of clusters $(C_0, C_1) \in \mathcal{A} \times \mathcal{B}$, we add to H at most one shortest path over all pairs of vertices in $C_0 \times C_1$. Then we have an approximate shortest path between u and v of the form $u - C' \rightsquigarrow C'' - v$, where $C' \in \mathcal{A}$ and $C'' \in \mathcal{B}$, and we will show that this resulting u - v path has additive stretch 4 (see Figure 1).

Our algorithm is presented below, let $h = \lceil |\mathcal{P}|^{2/7} \log^{3/7} n \rceil$ and $\ell = \lceil (n \log^{3/2} n)/h^{5/2} \rceil$.

1. Run the clustering step with the above h and let G_h be the post-clustering subgraph. Augment G_h with $O(h)$ BFS trees as described above. Let H be the resulting graph.
2. For each $(u, v) \in \mathcal{P}$ do: (let $\rho = SP(u, v)$)
 - (a) if $\text{cost}(\rho) \leq 2\ell$ then add all the missing edges of ρ to H .

- (b) else add to H all the missing edges of $\text{prefix}(\rho)$ and $\text{suffix}(\rho)$, where $\text{prefix}(\rho)$ (similarly, $\text{suffix}(\rho)$) is the minimal prefix (resp., suffix) of ρ that has ℓ edges missing in G_h .
3. For each $p \in \mathcal{F}$ such that $\text{cost}(p) > 2\ell$ do: *{recall that \mathcal{F} is the set of all shortest paths}*
 - let $Q_0 = \{\text{prefix}(p) : p \in \mathcal{F}\}$ and $Q_1 = \{\text{suffix}(p) : p \in \mathcal{F}\}$.
 4. Determine \mathcal{A} and \mathcal{B} greedily, where $\mathcal{A} =$ hitting set for Q_0 and $\mathcal{B} =$ hitting set for Q_1 .
 5. For each $C_0 \in \mathcal{A}$ and $C_1 \in \mathcal{B}$ do:
 - if there is a pair $(x, y) \in C_0 \times C_1$ such that $\delta_H(x, y) > \delta_G(x, y) + 4$, then find such a pair $(x', y') \in C_0 \times C_1$ with least $\delta_G(x', y')$ and add to H all the missing edges in $SP(x', y')$.

Lemma 10 shows that H is a \mathcal{P} -spanner with additive stretch 4 and Lemma 11 bounds the size of the subgraph H . Thus Theorem 1.1 follows.

► **Lemma 10.** *Let H be the subgraph computed by the algorithm above. Then $\delta_H(u, v) \leq \delta_G(u, v) + 4$ for each $(u, v) \in \mathcal{P}$.*

Proof. Consider any pair $(u, v) \in \mathcal{P}$ and let $\rho = SP(u, v)$. If $\text{cost}(\rho) \leq 2\ell$ then by Step 2(a), the entire path ρ is present in H and so $\delta_H(u, v) = \delta_G(u, v)$ in this case. If $\text{cost}(\rho) > 2\ell$ then there is a cluster $C' \in \mathcal{A}$ incident on the prefix ρ' of ρ and a cluster $C'' \in \mathcal{B}$ incident on the suffix ρ'' of ρ .

Let a be the first vertex of C' incident on ρ and b be the last vertex of C'' incident on ρ (see Figure 1). If $\delta_H(a, b) > \delta_G(a, b) + 4$ before Step 5 of our algorithm, then we would have bought a path of length at most $\delta_G(a, b)$ between C' and C'' in Step 5. Thus after Step 5, we have $\delta_H(a, b) \leq \text{diameter}_H(C') + \delta_H(C', C'') + \text{diameter}_H(C'') \leq \delta_G(a, b) + 4$.

In Step 2(b) of our algorithm, we would have added all edges in the u - a subpath and the b - v subpath of ρ , so $\delta_H(u, a) = \delta_G(u, a)$ and $\delta_H(b, v) = \delta_G(b, v)$. Thus at the end of the algorithm, we have $\delta_H(u, v) \leq \delta_H(u, a) + \delta_H(a, b) + \delta_H(b, v) \leq \delta_G(u, v) + 4$. ◀

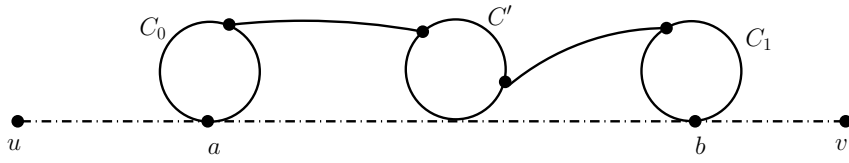
► **Lemma 11.** *The size of the final subgraph H is $O(nh)$, where $h = \lceil |\mathcal{P}|^{2/7} \log^{3/7} n \rceil$.*

Proof. Initially the size of H is the size of the post-clustering graph G_h , which is $O(nh)$. The size of H remains $O(nh)$ after the augmentation with $O(h)$ BFS trees. We buy at most 2ℓ edges per pair in \mathcal{P} in Step 2, so this adds to $2\ell|\mathcal{P}|$. For each $(C_0, C_1) \in \mathcal{A} \times \mathcal{B}$, we buy at most one shortest path – also such a shortest path has at most $(n \log n)/h^2$ missing edges in G_h ; otherwise it would already have been approximated within an additive stretch of 2 in H by some BFS tree (by Lemma 9). So we buy at most $(n \log n)/h^2$ edges per $(C_0, C_1) \in \mathcal{A} \times \mathcal{B}$.

It follows from Lemma 8 that $|\mathcal{A}|$ and $|\mathcal{B}|$ are at most $O(n \log n / (h\ell))$. Thus the total number of edges bought in Step 5 is $O((n \log n)/h^2 \cdot (n \log n / (h\ell))^2)$. This is $O(nh)$ since $\ell = \lceil (n \log^{3/2} n) / h^{5/2} \rceil$. Thus the total size of H is $O(nh + \ell|\mathcal{P}|)$. Substituting the values of ℓ and h , it follows that the size of H is $O(n|\mathcal{P}|^{2/7} \log^{3/7} n)$. ◀

A pairwise spanner with additive stretch 6. We now show a simple algorithm to construct a \mathcal{P} -spanner of size $O(n|\mathcal{P}|^{1/4})$ and additive stretch 6. The main step here is *path-buying* – for each shortest path ρ whose endpoints are in \mathcal{P} , if ρ is “affordable”, then we *buy* ρ , i.e., add to H all the missing edges of ρ .

Definition 12 captures the value of a path. For any pair of clusters C_1, C_2 in a subgraph H , let $\delta_H(C_1, C_2) = \min\{\delta_H(a, b) : a \in C_1, b \in C_2\}$. Similarly, for any pair of clusters C_1, C_2 incident on a path p , let $\delta_p(C_1, C_2)$ be the minimum value of $\delta_p(x, y)$, where $x \in C_1$ and $y \in C_2$ lie on path p .



■ **Figure 2** We have $\delta_H(C_0, C') \leq \delta_\rho(C_0, C')$ and $\delta_H(C', C_1) \leq \delta_\rho(C', C_1)$. Since the diameter of each cluster is 2, this implies $\delta_H(a, b) \leq \delta_G(a, b) + 6$.

► **Definition 12.** For any subgraph H of G and path p in G , let $\text{value}_H(p)$ be the number of pairs of clusters (C_1, C_2) incident on p such that $\delta_p(C_1, C_2) < \delta_H(C_1, C_2)$.

Thus $\text{value}_H(p)$ is the number of pairs of clusters incident on p whose distance in H would improve upon adding p to H . We also extend the **cost** function used in our earlier algorithm to $\text{cost}_H(\cdot)$: for any path p , let $\text{cost}_H(p)$ be the number of edges of p that are missing in H . Our algorithm is described below. Let $h = \lceil |\mathcal{P}|^{1/4} \rceil$ and $\ell = \lceil n/h^3 \rceil$.

1. Run the clustering step with parameter h . Let H be the post-clustering subgraph G_h .
2. For each $(u, v) \in \mathcal{P}$ do: (let $\rho = SP(u, v)$)
 - (a) if $\text{cost}_H(\rho) \leq \max\{4\ell, 8\text{value}_H(\rho)/\ell\}$ then buy ρ .
 - (b) else buy $\text{prefix}(\rho)$ and $\text{suffix}(\rho)$.
 [prefix(ρ) is the minimal prefix of ρ such that $\text{cost}_H(\text{prefix}(\rho)) = \ell$ and similarly, suffix(ρ) is the minimal suffix of ρ such that $\text{cost}_H(\text{suffix}(\rho)) = \ell$.]

► **Lemma 13.** For every pair $(u, v) \in \mathcal{P}$, we have $\delta_H(u, v) \leq \delta_G(u, v) + 6$.

Proof. Let $\rho \in \mathcal{F}$ be the u - v shortest path, where $(u, v) \in \mathcal{P}$. This path ρ gets considered in Step 2 of our algorithm. If we buy ρ in Step 2(a), then $\delta_H(u, v) = \delta_G(u, v)$. Hence assume ρ was not bought, let $\text{cost}_H(\rho) = t$ when ρ gets considered in Step 2. That is, at the time of processing ρ in Step 2, there are t edges of ρ missing in the current subgraph H .

We know that $t > 4\ell$, otherwise ρ would have been bought in Step 2(a). In Step 2(b), the prefix and suffix of ρ with ℓ edges missing in H , i.e., $\rho' = \text{prefix}(\rho)$ and $\rho'' = \text{suffix}(\rho)$, are bought. Let $q = \rho - (\rho' \cup \rho'')$, this is the *middle portion* of ρ consisting of $t - 2\ell$ missing edges. It follows from Lemma 6 that there are at least $\ell/2$ clusters incident on ρ' and on ρ'' .

Let \mathcal{A}_0 (similarly, \mathcal{A}_1) be the set of clusters incident on ρ' (resp., ρ''). The subpath q has at least $(t - 2\ell)/2$ incident clusters (call this set of clusters \mathcal{B}). We will show Claim 1.

► **Claim 1.** There are clusters $C_0 \in \mathcal{A}_0$, $C' \in \mathcal{B}$, and $C_1 \in \mathcal{A}_1$ such that $\delta_H(C_0, C') \leq \delta_\rho(C_0, C')$ and $\delta_H(C', C_1) \leq \delta_\rho(C', C_1)$.

We will now assume the above claim and finish the proof of Lemma 13. Then we will prove Claim 1. This claim guarantees that the graph H contains a path $C_0 \rightsquigarrow C' \rightsquigarrow C_1$ such that the $C_0 \rightsquigarrow C'$ subpath has length at most $\delta_\rho(C_0, C')$ and the $C' \rightsquigarrow C_1$ subpath has length at most $\delta_\rho(C', C_1)$ (see Figure 2). Thus there are vertices $a \in \rho' \cap C_0$ and $b \in \rho'' \cap C_1$ such that $\delta_H(a, b) \leq \delta_G(a, b) + 6$ (via the $C_0 \rightsquigarrow C' \rightsquigarrow C_1$ path in H and $\text{diameter}_H(C) \leq 2$ for all $C \in \mathcal{C}_h$). We buy all the missing edges in ρ' and in ρ'' in Step 2(b), hence we have $\delta_H(u, a) = \delta_G(u, a)$ and $\delta_H(b, v) = \delta_G(b, v)$. Since $\delta_H(u, v) \leq \delta_H(u, a) + \delta_H(a, b) + \delta_H(b, v)$, we get $\delta_H(u, v) \leq \delta_G(u, v) + 6$. ◀

Proof of Claim 1. Suppose there are no such clusters C_0, C' , and C_1 . Then for each cluster $C' \in \mathcal{B}$, either $\delta_\rho(C_0, C') < \delta_H(C_0, C')$ for every $C_0 \in \mathcal{A}_0$ or $\delta_\rho(C', C_1) < \delta_H(C', C_1)$

for every $C_1 \in \mathcal{A}_1$. Since $|\mathcal{A}_0| \geq \ell/2$, $|\mathcal{A}_1| \geq \ell/2$, and $|\mathcal{B}| \geq (t - 2\ell)/2$, this means that

$$\text{value}_H(\rho) \geq \frac{\ell}{2} \cdot \frac{t - 2\ell}{2} > \frac{\ell}{2} \cdot \frac{t}{4} = \frac{\ell \cdot t}{8}, \quad (1)$$

where the second inequality follows from the fact that $t > 4\ell$, so $\ell < t/4$, thus $t - 2\ell > t/2$. Since we did not buy ρ in Step 2(a), it must be the case that $t > 8\text{value}_H(\rho)/\ell$, i.e., $\text{value}_H(\rho) < \ell \cdot t/8$. This contradicts Inequality (1). \blacktriangleleft

► **Lemma 14.** *The size of the final subgraph H is $O(n|\mathcal{P}|^{1/4})$.*

Proof. Initially the size of H is $O(nh)$ which is $O(n|\mathcal{P}|^{1/4})$. The total number of edges added in Step 2(b) is at most $2\ell|\mathcal{P}|$ since we buy at most 2ℓ edges per element in \mathcal{P} . Since $\ell = \lceil n/h^3 \rceil$ where $h = \lceil |\mathcal{P}|^{1/4} \rceil$, it follows that $O(\ell|\mathcal{P}|)$ is $O(n|\mathcal{P}|^{1/4})$. The total number of edges added in Step 2(a) is $\sum_{\rho} \text{cost}(\rho)$ where the sum is over all the paths ρ that got bought in this step during the entire algorithm.

Let us evaluate $\sum_{\rho} \text{cost}(\rho)$ for the paths ρ bought in Step 2(a) – this is at most $4\ell|\mathcal{P}| + \sum_{\rho} 8\text{value}_H(\rho)/\ell$ since we buy ρ only when $\text{cost}_H(\rho) \leq \max\{4\ell, 8\text{value}_H(\rho)/\ell\}$. Let us bound $\sum_{\rho} \text{value}_H(\rho)/\ell$ where the sum is over all the paths bought.

We say a pair of clusters $(C_1, C_2) \in \mathcal{C}_h \times \mathcal{C}_h$ supports ρ if (C_1, C_2) contributes positively to $\text{value}_H(\rho)$. Consider all the shortest paths that were supported by a fixed pair (C_1, C_2) . We claim at most 5 of them could have been bought in our algorithm. This is because once a shortest path ρ supported by (C_1, C_2) gets bought, we have $\delta_H(C_1, C_2) \leq \delta_G(C_1, C_2) + 4$ since this path ρ is $SP(x, y)$ for some $(x, y) \in C_1 \times C_2$. Thereafter, every time a path supported by (C_1, C_2) gets bought, $\delta_H(C_1, C_2)$ (strictly) decreases. Thus

$$\sum_{\rho} \frac{\text{value}_H(\rho)}{\ell} = \frac{1}{\ell} \sum_{(C_1, C_2)} \text{number of paths supported by } (C_1, C_2) \text{ that got bought} \leq \frac{5|\mathcal{C}_h|^2}{\ell},$$

where the middle sum is over all pairs of clusters $(C_1, C_2) \in \mathcal{C}_h \times \mathcal{C}_h$. Substituting $|\mathcal{C}_h| \leq n/h$ and $\ell = \lceil n/h^3 \rceil$, the right side above is bounded by $O(nh)$. Hence the size of H is $O(n|\mathcal{P}|^{1/4})$. \blacktriangleleft

Theorem 1.2 follows from Lemmas 13 and 14. This finishes the proof of Theorem 1 stated in Section 1.

An ST -spanner with additive stretch 4. The input here is $G = (V, E)$ along with $S \subseteq V$ and $T \subseteq V$. We assume without loss of generality that $|S| \leq |T|$. Our algorithm here again uses *path-buying*, however with a new value function that is defined below.

► **Definition 15.** For any subgraph H of G and any path p in G with one endpoint in S (call this vertex s) and the other endpoint in T (call this vertex t), define $\text{value}_H(p)$ as follows:

$$\text{value}_H(p) = \ell \cdot |\{C_0 : \delta_p(s, C_0) < \delta_H(s, C_0)\}| + 2 \cdot |\{(C_1, C_2) : \delta_p(C_1, C_2) < \delta_H(C_1, C_2)\}|,$$

where all the clusters C_0, C_1, C_2 above have to be incident on p and the value $\ell = \lceil n/h^3 \rceil$ (the parameter h will be set to $\lceil (|S||T|)^{1/4} \rceil$).

In other words, $\text{value}_H(p) = \ell\alpha_1 + 2\alpha_2$, where α_1 is the number of clusters incident on p whose distance to s via p is better than the current distance to s in H and α_2 is the number of pairs of clusters incident on p whose distance in p is better than their current distance in H . Our algorithm is described below. Let $h = \lceil (|S||T|)^{1/4} \rceil$.

1. Run the clustering step with parameter h . Let H be the post-clustering subgraph G_h .
2. For each $(s, t) \in S \times T$ do: (let $p = SP(s, t)$)
 - (a) If $\text{cost}_H(p) \leq \max\{2\ell, 4\text{value}_H(p)/\ell\}$ then add to H all the missing edges of p .
 - (b) Else buy the minimal suffix p' of p such that $\text{cost}_H(p') = \ell$, i.e., add to H the last ℓ missing edges of p .

Lemma 16 states the correctness of the above algorithm. Its proof (which is omitted here) is similar to the proofs of Lemmas 13 and 14. Theorem 2 stated in Section 1 thus follows.

► **Lemma 16.** *The size of the final subgraph H is $O(n \cdot (|S| |T|)^{1/4})$. For every pair $(s, t) \in S \times T$, we have $\delta_H(s, t) \leq \delta_G(s, t) + 4$.*

4 Sourcewise spanners

The input here is $G = (V, E)$ along with a subset of V , which is the set of *sources*. In this section we will prove Theorem 3. We first show Theorem 3.2 whose proof follows quite easily from our ST -spanner with additive stretch 4 (shown in the previous section). Let $S' \subseteq V$ be the set of sources here – we will be using the symbols S and T for the 2 sets in the ST -spanner algorithm, so we use the symbol S' to refer to the set of sources here. Broadly speaking, we take the sets S and T in the ST -spanner to be the set S' and the set of all cluster centers to get a sourcewise spanner.

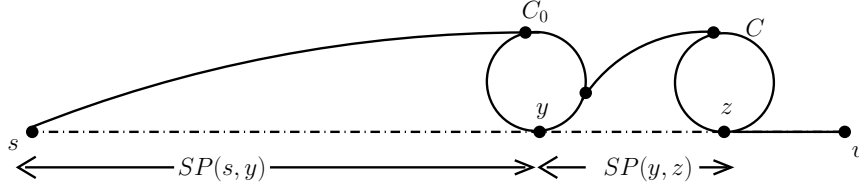
However we need to be careful about which of S, T becomes S' and which becomes the set of cluster centers: recall that our algorithm for ST -spanners assumed $|S| \leq |T|$. So we assign the sets S and T as follows: if $|S'| \leq n^{2/3}$ then assign $S = S'$ else assign $T = S'$. More precisely, our algorithm does the following:

1. Run the clustering step with parameter $h = \lceil (n|S'|)^{1/5} \rceil$. Let H be the post-clustering subgraph G_h . Let $T' = \{\text{cluster centers}\}$.
2. If $|S'| \leq n^{2/3}$ then set $S = S'$ and $T = T'$; else set $S = T'$ and $T = S'$.
3. Run Step 2 of our ST -spanner algorithm with additive stretch 4 (the parameter $\ell = \lceil n/h^3 \rceil$); return the ST -spanner H .

It can be shown (proof omitted here) that the subgraph returned by the ST -spanner algorithm is an $(S' \times V)$ -spanner with additive stretch 6 and that this subgraph has size $O(n \cdot (n|S'|)^{1/5})$. Thus Theorem 3.2 stated in Section 1 follows.

We now prove Theorem 3.1 by describing an algorithm to construct a sparse $(S \times V)$ -spanner with additive stretch 4, where $S \subseteq V$ is the set of sources. We run the clustering step with $h = \lceil (|S| n \log^2 n)^{2/9} \rceil$ and initialize the subgraph H to the post-clustering subgraph G_h . Then we augment H with BFS trees rooted at $O(h)$ selected cluster centers so that every path $p \in \mathcal{F}$ with $\text{cost}(p) \geq (n \log n)/h^2$ has some adjacent cluster center that has been selected. For every $s \in S$ and cluster C , we pick the shortest path between s and its closest vertex $x \in C$ (in case of ties, x is chosen arbitrarily) and call it $SP(s, C)$ henceforth.

Let $\mathcal{R} \subseteq \mathcal{F}$ be the collection of paths $SP(s, C)$ where $s \in S$ and $C \in \mathcal{C}_h$. Corresponding to each $\rho \in \mathcal{R}$, we are ready to buy λ new edges, where $\lambda^2 = (n \log n)^2/h^5$. So if $\text{cost}(\rho) \leq \lambda$, then we buy ρ . For any $\rho \in \mathcal{R}$, if $\text{cost}(\rho) > \lambda$, then let ρ_0 to be the minimal suffix of ρ that has $\lfloor \lambda \rfloor$ missing edges in the post-clustering subgraph G_h ; let Q_0 be the set containing all such suffixes ρ_0 . We now use *path-hitting* to determine $\mathcal{A}_0 \subseteq \mathcal{C}_h$ so that for each $\rho_0 \in Q_0$ there is at least one cluster $C_0 \in \mathcal{A}_0$ that intersects ρ_0 .



■ **Figure 3** There are at most λ^2 missing edges in the path $SP(s, y)$ and at most λ missing edges in the path $SP(y, z)$. We will buy a path $s \rightsquigarrow C_0 \rightsquigarrow C$, where C is the last cluster on $SP(s, v)$.

1. For each $s \in S$ and $C_0 \in \mathcal{A}_0$: if there exists any $x \in C_0$ such that $\text{cost}(SP(s, x)) \leq \lambda^2$ then buy $SP(s, x')$ where $x' \in C_0$ is the closest vertex to s that satisfies $\text{cost}(SP(s, x')) \leq \lambda^2$ and also run the following step.

1.1. For each $C \in \mathcal{C}_h$ such that C_0 intersects $SP(s, u)$ for some $u \in C$ do:

- if there is some path $SP(a, b)$ where $(a, b) \in C_0 \times C$ with $\text{cost}(SP(a, b)) \leq \lambda$ and buying this path improves $\delta_H(s, C)$, then buy $SP(a, b)$.

We now deal with $\rho \in \mathcal{R}$ such that $\text{cost}(\rho) > \lambda^2$. For any such ρ , define ρ_1 be the minimal prefix of ρ with $\lfloor \lambda^2 \rfloor$ missing edges in the current H ; let Q_1 be the set containing all such prefixes ρ_1 . We again use path-hitting to determine $\mathcal{A}_1 \subseteq \mathcal{C}_h$ so that for each $\rho_1 \in Q_1$ there is at least one cluster $C_1 \in \mathcal{A}_1$ that intersects ρ_1 . We run the following two steps now.

2. For each $(s, C_1) \in S \times \mathcal{A}_1$: if there exists any $w \in C_1$ such that $\text{cost}(SP(s, w)) \leq \lambda^2$ then buy $SP(s, w')$ where $w' \in C_1$ is the closest vertex to s that satisfies $\text{cost}(SP(s, w')) \leq \lambda^2$.
3. For each $(C_1, C) \in \mathcal{A}_1 \times \mathcal{C}_h$: if there is a pair $(a, b) \in C_1 \times C$ such that $\delta_H(a, b) > \delta_G(a, b) + 2$ and buying $SP(a, b)$ improves $\delta_H(C_1, C)$, then buy $SP(a, b)$.

► **Lemma 17.** For every pair $(s, v) \in S \times V$, we have $\delta_H(s, v) \leq \delta_G(s, v) + 4$.

Proof. Consider any pair $(s, v) \in S \times V$ and let $p = SP(s, v)$. We can assume that $\text{cost}(p) < (n \log n)/h^2$, otherwise $\delta_H(s, v) \leq \delta_G(s, v) + 2$ due to adding certain BFS trees to H .

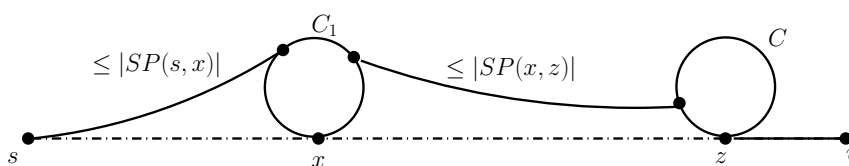
Let z be the last clustered vertex on p , i.e., z is the clustered vertex that is closest to v on this path p . Let C be the cluster containing z . Consider $\rho = SP(s, C)$. If $\text{cost}(\rho) \leq \lambda$, then we would have bought ρ , which means $\delta_H(s, z) \leq \delta_G(s, z) + 2$, thus $\delta_H(s, v) \leq \delta_G(s, v) + 2$. So let us assume that $\text{cost}(\rho) > \lambda$. We have two cases here:

Case(1): Suppose $\lambda < \text{cost}(\rho) \leq \lambda^2$. There is a cluster $C_0 \in \mathcal{A}_0$ incident on the suffix ρ_0 of ρ . In other words, the path ρ restricted to the subpath $C_0 \rightsquigarrow C$ has at most λ missing edges. Let y be the first vertex of C_0 on ρ . Since $\text{cost}(\rho) \leq \lambda^2$, we have $\text{cost}(SP(s, y)) \leq \lambda^2$. So in Step (1) we either buy all the missing edges of $SP(s, y)$ or we already have a path $s \rightsquigarrow C_0$ in H of length at most $\delta_G(s, y)$. Thus we have $\delta_H(s, C_0) \leq \delta_G(s, y)$.

Now consider the y - z subpath of ρ (see Figure 3). We know that there are at most λ edges of $SP(y, z)$ that are missing in the subgraph H . Buying these λ edges causes $\delta_H(s, C) \leq \delta_H(s, C_0) + 2 + |SP(y, z)| \leq \delta_G(s, z) + 2$.

So either $\delta_H(s, C)$ is already at most $\delta_G(s, z) + 2$ or we buy the missing edges of $SP(y, z)$ in Step (1.1) and make $\delta_H(s, C) \leq \delta_G(s, z) + 2$. Thus after this step, we have $\delta_H(s, v) \leq \delta_G(s, v) + 4$ using the path $s \rightsquigarrow C_0 \rightsquigarrow C \rightsquigarrow v$.

Case(2): We are left with the case when $\text{cost}(\rho) > \lambda^2$. In this case we would have a cluster $C_1 \in \mathcal{A}_1$ incident on the prefix ρ_1 of ρ with at most λ^2 missing edges. Let x be the first vertex of C_1 on ρ (see Figure 4). While considering the pair (s, C_1) , we would have either bought all the missing edges of $SP(s, x)$ in Step (2) (as there are at most λ^2 missing edges here) or we already have a path $s \rightsquigarrow C_1$ of length at most $\delta_G(s, x)$.



■ **Figure 4** H has an $s \rightsquigarrow C_1$ path of length $|SP(s, C_1)|$. There is also a $C_1 \rightsquigarrow C$ path of length $|SP(x, z)|$.

Consider the pair $(C_1, C) \in \mathcal{A}_1 \times \mathcal{C}_h$. Step (3) for the pair $(x, z) \in (C_1, C)$ ensures that either $\delta_H(x, z) \leq \delta_G(x, z) + 2$ or there is already a $C_1 \rightsquigarrow C$ path of length at most $|SP(x, z)|$. In either case there is a path $C_1 \rightsquigarrow z$ of length at most $\delta_G(x, z) + 2$. Since $\delta_H(s, C_1) \leq \delta_G(s, x)$ and $\delta_H(C_1, z) \leq \delta_G(x, z) + 2$, it follows that $\delta_H(s, v) \leq \delta_G(s, v) + 4$. ◀

Theorem 3.1 stated in Section 1 follows from Lemma 17 and Lemma 18 stated below.

▶ **Lemma 18.** *The size of the final subgraph H is $O(nh)$, where $h = \lceil (|S|n \log^2 n)^{2/9} \rceil$.*

5 ST-spanners: A trade-off result

In this section we present our algorithm to construct a sparse ST -spanner with additive stretch $2k$, for any integer $k \geq 1$. We first describe the algorithm and show that for any $(s, t) \in S \times T$, the final subgraph H has an s - t path of length at most $|SP(s, t)| + 2k$.

Initialization. We run the clustering step with an appropriate parameter h and the subgraph H is initialized to the post-clustering subgraph G_h . Then we augment H using $O(h)$ BFS trees so that for each shortest path $p \in \mathcal{F}$ with $\text{cost}(p) \geq (n \log n)/h^2$, the subgraph H has a path of length at most $|p| + 2$ between p 's endpoints.

Set the parameters $\ell = nh/(k|S||T|)$ and $\alpha = (|S||T|k \log n/h^3)^{1/k}$. Note that these parameters have been set so that $\alpha^k \ell = (n \log n)/h^2$. For $(s, t) \in S \times T$, if $\text{cost}(SP(s, t)) \leq \ell$, then we buy $SP(s, t)$. So we now have to deal with approximating shortest paths $p \in \mathcal{F}$ with endpoints in $S \times T$ that satisfy $\ell < \text{cost}(p) < \alpha^k \ell$.

Divide each such path p into *critical* subpaths as follows. Let $\alpha^{r-1} \ell < \text{cost}(p) \leq \alpha^r \ell$ for some $r \in \{1, \dots, k\}$. Then for each $j \in \{0, 1, \dots, (r-1)\}$:

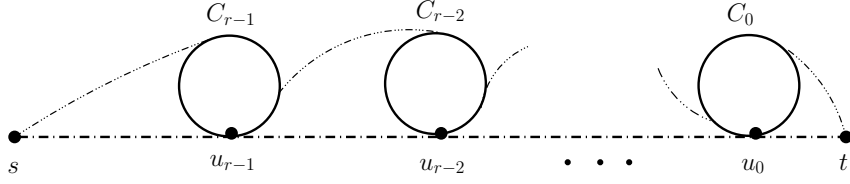
- let $p_j =$ minimal suffix of p with $\lfloor \alpha^j \ell \rfloor$ edges missing in the post-clustering graph G_h .

Thus p can be split as $p' \parallel p_{r-1}$ for some prefix p' . Split p_{r-1} into $q_{r-1} \parallel q_{r-2} \parallel \dots \parallel q_0$ as follows: let $q_0 = p_0$ and for $1 \leq j \leq r-1$, let q_j be the prefix of p_j obtained by removing p_{j-1} (a suffix of p_j) from p_j . For each j , let Q_j be the set of all q_j 's. So each $p = SP(s, t)$ with $\text{cost}(p) \geq \alpha^{r-1} \ell$ has a subpath $q_j \in Q_j$, for $j = 0, \dots, r-1$.

Determine hitting sets $\mathcal{A}_0, \dots, \mathcal{A}_{k-1}$ for these sets Q_0, \dots, Q_{k-1} , respectively. In other words, \mathcal{A}_j is a set of clusters such that for every $q_j \in Q_j$, there is at least one cluster in \mathcal{A}_j that intersects q_j .

1. For each $s \in S$ and cluster $C_{j-1} \in \mathcal{A}_{j-1}$ (where $1 \leq j \leq k$) do
 - for every vertex x in C_{j-1} : if $\text{cost}(SP(s, x)) \leq \alpha^j \ell$ and buying $SP(s, x)$ improves $\delta_H(s, C_{j-1})$, then buy $SP(s, x)$.

In order to see why this step is useful, consider $p = SP(s, t)$ such that $\alpha^{r-1} \ell < \text{cost}(p) \leq \alpha^r \ell$ for some $1 \leq r \leq k$. As described above, we can write p as $p' \parallel q_{r-1} \parallel q_{r-2} \parallel \dots \parallel q_0$. In each hitting set \mathcal{A}_j (for $0 \leq j \leq r-1$), we would have at least one cluster that intersects q_j : let C_j be such a cluster. For each j , let u_j be the first vertex of cluster C_j on the path p , i.e., while traversing p from s to t , the first vertex of C_j that we encounter is u_j (see Figure 5).



■ **Figure 5** The subgraph H has a path of length at most $|SP(s, u_{r-1})|$ between s and C_{r-1} .

Consider the $s - u_{r-1}$ subpath of p . The shortest path $SP(s, u_{r-1})$, being a subpath of p , has cost at most $\alpha^r \ell$. Hence while processing the pair (s, C_{r-1}) when we consider $SP(s, u_{r-1})$ in Step (1) above, we either buy $SP(s, u_{r-1})$ or we already have $\delta_H(s, C_{r-1}) \leq \delta_G(s, u_{r-1})$. In other words, after Step (1) we have $\delta_H(s, C_{r-1}) \leq \delta_G(s, u_{r-1})$. We would similarly like to have $\delta_H(C_{i-1}, C_{r-2}) \leq |SP(u_{r-1}, u_{r-2})|$ and make $\delta_H(s, C_{r-2}) \leq \delta_G(s, u_{r-2}) + 2$ and so on. In order to accomplish this, we do as follows.

2. For each $s \in S$ and for $j = k - 1$ down to 1 do
 - for every $(C_j, C_{j-1}) \in \mathcal{A}_j \times \mathcal{A}_{j-1}$ so that there is some $v \in C_{j-1}$ with $SP(s, v) \cap C_j \neq \emptyset$
 - for each $(x, y) \in C_j \times C_{j-1}$: if $\text{cost}(SP(x, y)) \leq 2\alpha^j \ell$ and buying $SP(x, y)$ improves $\delta_H(s, C_{j-1})$, then buy $SP(x, y)$.

In the above step, corresponding to $j = r - 1$ and the vertex $s \in S$, we consider the pair (C_{r-1}, C_{r-2}) on the path p shown in Figure 5. Since there is a vertex $u_{r-2} \in C_{r-2}$ such that $SP(s, u_{r-2})$ intersects C_{r-1} , we run the innermost for loop of Step (2) for (C_{r-1}, C_{r-2}) . Also $\text{cost}(SP(u_{r-1}, u_{r-2})) \leq \text{cost}(q_{r-1}) + \text{cost}(q_{r-2}) \leq 2\alpha^{r-1} \ell$.

So either we already have $\delta_H(s, C_{r-2}) \leq \delta_G(s, u_{r-2}) + 2$ or we buy $SP(u_{r-1}, u_{r-2})$ and make $\delta_H(s, C_{r-2}) \leq \delta_H(s, C_{r-1}) + 2 + |SP(u_{r-1}, u_{r-2})|$, which is at most $\delta_G(s, u_{r-2}) + 2$ since $\delta_H(s, C_{r-1}) \leq \delta_G(s, u_{r-1})$ by the end of Step (1).

It is easy to see that for any $1 \leq i \leq r - 1$, if $\delta_H(s, C_i) \leq \delta_G(s, u_i) + 2(r - i - 1)$ when the index $j = i + 1$, then when the index $j = i$ and the pair (C_i, C_{i-1}) on path p gets considered, then $\delta_H(s, C_{i-1})$ becomes at most $\delta_H(s, C_i) + 2 + |SP(u_i, u_{i-1})|$ which is at most $\delta_G(s, u_{i-1}) + 2(r - i)$.

Thus at the end of Step (2), we have $\delta_H(s, C_0) \leq \delta_G(s, u_0) + 2(r - 1)$.

3. Finally for each $(s, t) \in S \times T$ and $C_0 \in \mathcal{A}_0$ such that C_0 intersects $SP(s, t)$ do
 - for each $w \in C_0$: if $\text{cost}(SP(w, t)) \leq \ell$ and buying $SP(w, t)$ improves $\delta_H(s, t)$, then buy $SP(w, t)$.

Thus in Step (3), when we consider the cluster C_0 and the pair (s, t) , we either buy $SP(u_0, t)$ which ensures $\delta_H(s, t) \leq \delta_H(s, C_0) + 2 + |SP(u_0, t)| \leq \delta_G(s, t) + 2r$ or we already have $\delta_H(s, t) \leq \delta_G(s, t) + 2r$. Since $r \leq k$, it follows that H has an additive stretch of $2k$ for all distances in $S \times T$. This finishes the description of our algorithm and its correctness. Lemma 19 (proof omitted here) bounds the size of H .

► **Lemma 19.** *The final subgraph H has $O(nh)$ edges, where $h = \lceil k\sqrt{\log n} \cdot (|S|^{k+1} |T|)^{\frac{1}{2k+3}} \rceil$.*

We can now conclude Theorem 4 stated in Section 1, i.e., for any integer $k \geq 1$ and $S, T \subseteq V$, an ST -spanner with additive stretch $2k$ and size $\tilde{O}(n \cdot (|S|^\gamma |T|)^{1/(2\gamma+1)})$, where $\gamma = k + 1$, can be constructed in polynomial time.

By running the clustering step with $h = \lceil k\sqrt{\log n} \cdot (|S|^{k+1} n)^{1/(2k+4)} \rceil$ and taking $T = \{\text{cluster centers}\}$, Theorem 4 gives us an ST -spanner H of size $O(nh)$ and additive stretch $2k$. Since T is the set of cluster centers, it is easy to see that H is an $(S \times V)$ -spanner with additive stretch $2k + 2$.

► **Corollary 20.** For any integer $k \geq 1$ and subset $S \subseteq V$, an $(S \times V)$ -spanner with additive stretch $2t$ and size $\tilde{O}(n^{1+1/(2t+2)} \cdot |S|^{t/(2t+2)})$, where $t = k + 1$, can be constructed in polynomial time.

Acknowledgments. Thanks to the reviewers for their helpful comments.

References

- 1 D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999.
- 2 I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete and Computational Geometry*, 9:81–100, 1993.
- 3 B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Near-linear time construction of sparse neighborhood covers. *SIAM Journal on Computing*, 28(1):263–277, 1998.
- 4 B. Awerbuch and D. Peleg. Routing with polynomial communication-space trade-off. *SIAM Journal on Computing*, 5(2):151–162, 1992.
- 5 S. Baswana and T. Kavitha. Faster algorithms for all-pairs approximate shortest paths in undirected graphs. *SIAM Journal on Computing*, 39(7):2865–2896, 2010.
- 6 S. Baswana, T. Kavitha, K. Mehlhorn, and S. Pettie. Additive spanners and (α, β) -spanners. *ACM Transactions on Algorithms*, 7(1), 2010.
- 7 S. Baswana and S. Sen. Approximate distance oracles for unweighted graphs in $o(n^2 \log n)$ time. *ACM Transactions on Algorithms*, 2(4):557–577, 2006.
- 8 S. Baswana and S. Sen. A simple linear time algorithm for computing a $(2k - 1)$ -spanner of $o(n^{1+1/k})$ size in weighted graphs. *Random Structures and Algorithms*, 30(4):532–563, 2007.
- 9 B. Bollobás, D. Coppersmith, and M. Elkin. Sparse distance preservers and additive spanners. *SIAM Journal on Discrete Math.*, 19(4):1029–1055, 2005.
- 10 S. Chechik. New additive spanners. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 498–512, 2013.
- 11 E. Cohen. Fast algorithms for constructing t -spanners and paths of stretch t . In *Proceedings of the 34th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 648–658, 1993.
- 12 D. Coppersmith and M. Elkin. Sparse source-wise and pair-wise distance preservers. In *Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 660–669, 2005.
- 13 L. J. Cowen. Compact routing with minimum stretch. *Journal of Algorithms*, 28:170–183, 2001.
- 14 L. J. Cowen and C. G. Wagner. Compact roundtrip routing in directed networks. *Journal of Algorithms*, 50(1):79–95, 2004.
- 15 M. Cygan, F. Grandoni, and T. Kavitha. On pairwise spanners. In *Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 209–220, 2013.
- 16 D. Dor, S. Halperin, and U. Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2004.
- 17 M. Elkin. Computing almost shortest paths. *ACM Transactions on Algorithms*, 1(2):283–323, 2005.
- 18 M. Elkin and D. Peleg. $(1 + \epsilon, \beta)$ -spanner construction for general graphs. *SIAM Journal on Computing*, 33(3):608–631, 2004.
- 19 C. Gavoille, D. Peleg, S. Perennes, and R. Raz. Distance labeling in graphs. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 210–219, 2001.

- 20 T. Kavitha and N. M. Varma. Small stretch pairwise spanners. In *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 601–612, 2013.
- 21 M. Parter. Bypassing Erdős’ girth conjecture: Hybrid spanners and sourcewise spanners. In *Proceedings of the 41st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 608–619, 2014.
- 22 D. Peleg. Proximity-preserving labeling schemes. *Journal of Graph Theory*, 33(3):167–176, 2000.
- 23 D. Peleg and A. A. Schaffer. Graph spanners. *Journal of Graph Theory*, 13:99–116, 1989.
- 24 D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. *SIAM Journal on Computing*, 18:740–747, 1989.
- 25 S. Pettie. Low distortion spanners. *ACM Transactions on Algorithms*, 6(1), 2009.
- 26 L. Roditty, M. Thorup, and U. Zwick. Deterministic constructions of approximate distance oracles and spanners. In *Proceedings of the 32nd Int. Colloq. on Automata, Languages, and Programming (ICALP)*, pages 261–272, 2005.
- 27 L. Roditty and U. Zwick. On dynamic shortest paths problems. In *Proceedings of the 12th Annual European Symposium on Algorithms (ESA)*, pages 580–591, 2004.
- 28 M. Thorup and U. Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, 2005.
- 29 D. P. Woodruff. Additive spanners in nearly quadratic time. In *Proceedings of the 37th Int. Colloq. on Automata, Languages, and Programming (ICALP)*, pages 463–474, 2010.

Multi- k -ic Depth Three Circuit Lower Bound

Neeraj Kayal¹ and Chandan Saha²

1 Microsoft Research India

neeraka@microsoft.com

2 Indian Institute of Science

chandan@csa.iisc.ernet.in

Abstract

In a multi- k -ic depth three circuit every variable appears in at most k of the linear polynomials in every product gate of the circuit. This model is a natural generalization of multilinear depth three circuits that allows the formal degree of the circuit to exceed the number of underlying variables (as the formal degree of a multi- k -ic depth three circuit can be kn where n is the number of variables). The problem of proving lower bounds for depth three circuits with high formal degree has gained in importance following a work by Gupta, Kamath, Kayal and Saptharishi [7] on depth reduction to high formal degree depth three circuits. In this work, we show an exponential lower bound for multi- k -ic depth three circuits for any arbitrary constant k .

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems, I.1.1 Expressions and Their Representation

Keywords and phrases arithmetic circuits, multilinear circuits, depth three circuits, lower bound, individual degree

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.527

1 Introduction

The recent years have witnessed some promising progress in arithmetic circuit lower bounds. A line of research attempts to better understand the prospect of proving super-polynomial arithmetic circuit lower bound by proving strong lower bounds for small depth circuits - thanks to the beautiful depth reduction results in these works [20, 2, 13, 19]. A work by Gupta, Kamath, Kayal and Saptharishi [7] showed that in order to separate VP from VNP, it is sufficient to prove a *strong-enough* lower bound for depth three circuits. The formal degree¹ of a depth three circuit can be much larger than the degree of the polynomial that it computes. This fact is exhibited in [7]: quite interestingly, there is a depth three circuit with formal degree $n^{O(\sqrt{n})}$ (and also size $n^{O(\sqrt{n})}$) that computes Det_n , the determinant of an $n \times n$ symbolic matrix. Note that in this case the formal degree $n^{O(\sqrt{n})}$ is also much higher than the number of variables n^2 . It follows from [7] that if we are able to show an $n^{\omega(\sqrt{n})}$ size lower bound for depth three circuits of formal degree $n^{O(\sqrt{n})}$ computing the Perm_n (the permanent of an $n \times n$ symbolic matrix) then we would end up separating the circuit complexity of the determinant and the permanent polynomials (also proving $\text{VP} \neq \text{VNP}$).

¹ formal degree of a circuit C is the formal degree of its output gate. Formal degree of a $+$ gate is the maximum of the formal degrees of its children, whereas formal degree of a \times gate is the sum of the formal degrees of its children.

1.1 Motivation and our result

The issue of large formal degree of a circuit, compared to the actual degree and the number of variables of the polynomial being computed, poses a challenge to the existing lower bound techniques in particular the complexity measures that have been used successfully to prove lower bounds for certain interesting models of circuits having low formal degree. The partial derivatives measure, the shifted partials and the closely related projected shifted partials, and the evaluation dimension are examples of such effective measures.

The partial derivatives measure was introduced and used by Nisan and Wigderson in an influential work [15] to prove an exponential lower bound for homogeneous² depth three circuits with formal degree less than the number of variables. A lower bound for depth three circuits with large formal degree will trivially imply a lower bound for homogeneous depth three circuits with large formal degree. This prompts us to pose the following problem,

► **Problem 1.** Over fields of characteristic zero, prove a super polynomial lower bound for homogeneous depth three circuits with formal degree $D = k \cdot n$, where k is an arbitrary constant and n is the number of variables.³

In other words, can we prove a lower bound even if we allow the degree of the polynomial (being computed) to equal the formal degree of the depth three circuit that is only modestly higher than the number of variables? We do not know if the partial derivatives measure, or in fact any of the known measures and techniques, can be used to solve this problem. But, doing so might offer some insight into depth three circuits with large formal degree. We note that solving Problem 1 would automatically take us to the realm of *non-multilinear* polynomials.

Building on the partial derivatives measure, Kayal [9] has introduced the shifted partials measure which has been used subsequently to prove an exponential lower bound for homogeneous depth four circuits⁴ [10, 14] (albeit, using a variant of the shifted partials measure called the projected shifted partials)⁵. A recent work by Kayal and Saha [11] uses the projected shifted partials measure to prove an exponential lower bound for depth three circuits with arbitrarily large formal degree but with somewhat low bottom fanin. It is not clear to us if the projected shifted partials can be used to solve Problem 1.

The evaluation dimension measure (defined later) has been used by Raz and Yehudayoff [18] to prove an exponential lower bound for multilinear⁶ depth three circuits⁷. More precisely, they have shown a size lower bound of $2^{\Omega(d)}$ for any multilinear depth three circuit computing Det_d . Note that the formal degree of a multilinear depth three circuit is less or equal to the number of variables of the polynomial it computes. In the context of studying

² a circuit is homogeneous if every gate of the circuit computes a homogeneous polynomial (meaning, all monomials have the same degree)

³ Over any fixed finite field, a solution to this problem already follows from the works of [6] and [5].

⁴ with formal degree less than the number of variables

⁵ [10] builds upon the works of [8] and [12].

⁶ every variable occurs in at most one of the linear polynomials in every product gate of a multilinear depth three circuit

⁷ in fact, their result is more general and applies to constant depth multilinear circuits. Also, their result builds on an earlier work by Raz [16] who showed a quasi-polynomial lower bound for general multilinear formulas. Both [18] and [16] use the rank of a partial derivatives matrix as a measure which can be shown to be the same as the evaluation dimension - a concept used in [3].

depth three circuits with large formal degree, a natural generalization of multilinear depth three circuits is the model of multi- k -ic depth three circuits (defined below) that allows the formal degree of the circuit to be higher than the number of variables.

► **Definition 1.** A depth three circuit is *multi- k -ic* if every variable appears in at most k of the linear polynomials in every product gate of the circuit.

For example, the expression $(x_1 + 2x_2)(4x_1 - x_3) + x_2^2 + (x_3 - x_2)(x_1 + x_2)$ is a multi-2-ic⁸ depth three circuit. The formal degree of a multi- k -ic depth three circuit can be as high as $k \cdot n$, where n is the number of variables. A question, related to Problem 1, is the following: even if we allow the degree of the polynomial computed to equal the formal degree of the multi- k -ic circuit that computes it, can we prove a lower bound for this model?

► **Problem 2.** Prove an exponential lower bound for multi- k -ic depth three circuits for any arbitrary constant k .

Could the evaluation dimension be useful in solving this problem⁹? In this work, we answer this question in the affirmative.

► **Theorem 2.** *Let k be any arbitrary constant. There is a family of n -variate, degree $k \cdot n$ polynomials $\{f_n\}$ in VNP such that any multi- k -ic depth three circuit computing f_n must have size $2^{\Omega(n/2^{25k})}$.*

We will prove the above theorem in the rest of this article, but leave Problem 1 open. (We have not tried to optimize the constant 2^{25} in the above theorem.)

2 The measure - evaluation dimension

Let $f(x_1, \dots, x_n)$ be a polynomial in $\mathbb{F}[x_1, \dots, x_n]$, and $S = \{x_{i_1}, \dots, x_{i_m}\}$ be a subset of the variables $\mathbf{x} = \{x_1, \dots, x_n\}$. For a point $\mathbf{a} = (a_1, \dots, a_m) \in \mathbb{F}^m$, let $f_{S=\mathbf{a}} \in \mathbb{F}[\mathbf{x} \setminus S]$ denote the polynomial f evaluated at $x_{i_j} = a_j$ for every $j \in [m]$. Let $\text{eval}_S(f)$ be the \mathbb{F} -linear space spanned by the polynomials $\{f_{S=\mathbf{a}}\}_{\mathbf{a} \in \mathbb{F}^m}$, i.e.

$$\text{eval}_S(f) = \mathbb{F}\text{-span}(\{f_{S=\mathbf{a}} : \mathbf{a} \in \mathbb{F}^m\})$$

► **Definition 3.** Evaluation dimension of a polynomial f with respect to a subset of variables S is defined as the dimension of the vector space $\text{eval}_S(f)$. It is denoted by $\text{evalDim}_S(f)$.

Let us state a couple of useful properties of the evaluation dimension.

► **Lemma 4.** *Let f and g be two polynomials in $\mathbb{F}[\mathbf{x}]$ and $S \subseteq \mathbf{x}$. Then*

1. (subadditivity) $\text{evalDim}_S(f + g) \leq \text{evalDim}_S(f) + \text{evalDim}_S(g)$
2. (submultiplicativity) $\text{evalDim}_S(f \cdot g) \leq \text{evalDim}_S(f) \cdot \text{evalDim}_S(g)$

⁸ ‘multiquadratic’ sounds better here

⁹ The works of Grenet, Koiran, Portier, and Strozecki [4] and of Agrawal, Saha, Saptharishi and Saxena [1] proved lower bounds for certain models of depth four circuits with high formal degree, using properties of the real- τ -conjecture and the Jacobian respectively. The top fanin of such depth four circuits is essentially low or can be assumed to be low without loss of generality - a feature that is crucially used in their proofs. We do not know if their techniques can be used to solve Problem 2.

Proof. The subadditivity property follows from the observation that every polynomial in the space $\text{eval}_S(f + g)$ is a sum of a polynomial in $\text{eval}_S(f)$ and a polynomial in $\text{eval}_S(g)$. Now suppose the polynomials f_1, \dots, f_p form a basis of the space $\text{eval}_S(f)$ and similarly g_1, \dots, g_q form a basis of $\text{eval}_S(g)$. Then every polynomial in the space $\text{eval}_S(f \cdot g)$ can be expressed as an \mathbb{F} -linear combination of polynomials $f_i g_j$ with $i \in [p]$ and $j \in [q]$. This shows the submultiplicativity property. \blacktriangleleft

3 An explicit polynomial with high evaluation dimension

Let g be a polynomial in $4n$ variables $\mathbf{u} = \{u_1, \dots, u_{2n}\}$ and $\mathbf{x} = \{x_1, \dots, x_{2n}\}$, and $k \in \mathbb{Z}^+$ be an arbitrary positive integer. To every set $A \subseteq [2n]$, associate a set B_A in the following manner:

- If $|A| \geq n$ then B_A is a fixed subset of A of size exactly equal to $\bar{A} = [2n] \setminus A$.
- If $|A| < n$ then B_A is a fixed subset of \bar{A} of size exactly equal to A .

One way of fixing B_A is to take lexicographically the smallest subset. For a set $A \subseteq [2n]$ and $\mathbf{e} = \{e_1, \dots, e_{|A|}\} \in \mathbb{Z}^{|A|}$, let $x_A^{\mathbf{e}} \stackrel{\text{def}}{=} \prod_{i \in A} x_i^{e_i}$ and $u_A \stackrel{\text{def}}{=} \prod_{i \in A} u_i$. Define the polynomial $f_A(\mathbf{x})$ as follows.

$$f_A(\mathbf{x}) = \begin{cases} \sum_{\mathbf{e} \in \{0, \dots, k\}^{|\bar{A}|}} x_{B_A}^{\mathbf{e}} \cdot x_A^{\mathbf{e}} & \text{if } |A| \geq n \\ \sum_{\mathbf{e} \in \{0, \dots, k\}^{|A|}} x_A^{\mathbf{e}} \cdot x_{B_A}^{\mathbf{e}} & \text{if } |A| < n \end{cases}$$

Define g as,

$$g = \sum_{A \subseteq [2n]} u_A \cdot f_A(\mathbf{x}). \quad (1)$$

Polynomial g satisfies the following property.

► **Lemma 5.** *For every $A \subseteq [2n]$, there is an assignment of the \mathbf{u} variables to field constants such that $\text{evalDim}_{\mathbf{x}_A}(g)$, where $\mathbf{x}_A = \{x_i : i \in A\}$, (after setting the \mathbf{u} variables) is $(k + 1)^{\min(|A|, |\bar{A}|)}$.*

Proof. Let $A \subseteq [2n]$. Consider this assignment of the \mathbf{u} variables: $u_i = 1$ if $i \in A$ and zero otherwise. Denote the polynomial g under this assignment by $g_{\mathbf{u}_A=1}$, which equals $f_A(\mathbf{x})$. Hence,

$$\text{evalDim}_{\mathbf{x}_A}(g_{\mathbf{u}_A=1}) = \text{evalDim}_{\mathbf{x}_A}(f_A)$$

Now, it is not difficult to see that the evaluation dimension of f_A with respect to \mathbf{x}_A equals $(k + 1)^{|A|}$ (respectively, $(k + 1)^{|\bar{A}|}$) if $|A| < n$ (respectively, $|A| \geq n$). \blacktriangleleft

We also note that g defines a polynomial family in VNP. The construction of g is inspired by a similar construction in an earlier work of Raz [17].

Picking a random \mathbf{x}_A . Suppose we form a set A by picking every $i \in [2n]$ independently at random with probability $\frac{1}{2}$. By Chernoff bound, $|A| \in [(1 - \delta)n, (1 + \delta)n]$ with probability at least $1 - e^{-n\delta^2/3}$ for any $\delta > 0$. We will study the evaluation dimension of g and the multi- k -ic depth three circuit that computes it with respect to such a random $\mathbf{x}_A = \{x_i : i \in A\}$ after assigning field values to the \mathbf{u} -variables. The parameter δ will be a fixed function of k (to be specified later in Section 6).

► **Corollary 6.** *By Lemma 5, if A is chosen randomly (as described above) then $\text{evalDim}_{\mathbf{x}_A}(g_{\mathbf{u}_A=1})$ is at least $(k+1)^{(1-\delta)n}$ with probability higher than $1 - e^{-n\delta^2/3}$, for any $\delta > 0$.*

The above corollary provides a lower bound on the evaluation dimension of g . We will now show an upper bound on the evaluation dimension of a multi- k -ic depth three circuit with respect to a random \mathbf{x}_A . This, together with Corollary 6, will give us the relevant lower bound as outlined below. In the rest of this article whenever we write A is ‘random’ we mean A is formed by picking every $i \in [2n]$ independently at random with probability $\frac{1}{2}$.

4 Proof outline

Let $C = \sum_{i=1}^s T^{(i)}$ be a multi- k -ic depth three circuit computing g (as defined in Equation 1), where every $T^{(i)}$ is a product of linear polynomials. We will refer to $T^{(i)}$ as a *product term* (or simply a *term*) of C . Since C is multi- k -ic, every variable appears in at most k linear polynomials in every $T^{(i)}$. Let $A \subseteq [2n]$ be a random set and $\mathbf{x}_A = \{x_i : i \in A\}$ be the corresponding subset of \mathbf{x} . For any polynomial $h(\mathbf{x}, \mathbf{u})$, denote by $h_{\mathbf{u}_A=1}$ the polynomial h with $u_i = 1$ if $i \in A$ and $u_i = 0$ if $i \notin A$. Note that $h_{\mathbf{u}_A=1}$ is a polynomial in only the \mathbf{x} -variables.

$$\begin{aligned} g &= C = \sum_{i=1}^s T^{(i)} \\ \Rightarrow g_{\mathbf{u}_A=1} &= C_{\mathbf{u}_A=1} = \sum_{i=1}^s T_{\mathbf{u}_A=1}^{(i)} \\ \Rightarrow \text{evalDim}_{\mathbf{x}_A}(g_{\mathbf{u}_A=1}) &\leq \sum_{i=1}^s \text{evalDim}_{\mathbf{x}_A}(T_{\mathbf{u}_A=1}^{(i)}), \end{aligned}$$

where the last inequality follows from the subadditive property of the evaluation dimension (Lemma 4). Now, suppose we are able to show that $\text{evalDim}_{\mathbf{x}_A}(T_{\mathbf{u}_A=1}^{(i)})$ is upper bounded by a quantity $U(k, n, \delta)$ for every $i \in [s]$ with high probability over the random choice of A . Then by applying union bound,

$$\text{evalDim}_{\mathbf{x}_A}(g_{\mathbf{u}_A=1}) \leq s \cdot U(k, n, \delta),$$

also with high probability. In other words, by the above observation and Corollary 6, there exists a choice of A such that

$$\begin{aligned} (k+1)^{(1-\delta)n} &\leq \text{evalDim}_{\mathbf{x}_A}(g_{\mathbf{u}_A=1}) \leq s \cdot U(k, n, \delta) \\ \Rightarrow s &\geq \frac{(k+1)^{(1-\delta)n}}{U(k, n, \delta)}. \end{aligned}$$

This will give us a lower bound on the top fanin of C . We are now left with the task of finding a suitable expression for $U(k, n, \delta)$, which we do in the following section.

5 Evaluation dimension of a term of a multi- k -ic depth-3 circuit

Notations

Let us focus on a product term $T^{(i)} = T$ (say). Let $T = \prod_{j=1}^d \ell_j$, where ℓ_j is a linear polynomial and c be a positive integer constant (to be fixed later in Section 6). Split the linear polynomials in T into three parts:

$$\begin{aligned}
P^{(1)} &:= \prod_{j \in [d]} \ell_j \text{ such that } \ell_j \text{ has exactly one or no } \mathbf{x}\text{-variables} \\
P^{(2)} &:= \prod_{j \in [d]} \ell_j \text{ such that the number of } \mathbf{x}\text{-variables in } \ell_j \text{ is between two and } ck \\
P^{(3)} &:= \prod_{j \in [d]} \ell_j \text{ such that } \ell_j \text{ has greater than } ck \text{ } \mathbf{x}\text{-variables}
\end{aligned}$$

Also let,

$$m_i := \text{the number of linear polynomials in } T \text{ with exactly } i \text{ } \mathbf{x}\text{-variables.}$$

Naturally, $T = P^{(1)} \cdot P^{(2)} \cdot P^{(3)}$. Also, the number of linear polynomials in $P^{(1)}$ is $m_0 + m_1$, the number of linear polynomials in $P^{(2)}$ equals $\sum_{i=2}^{ck} m_i$, and the number of linear polynomials in $P^{(3)}$ equals $\sum_{i > ck} m_i$.

► **Claim 7.** For any $A \subseteq [2n]$, $\text{evalDim}_{\mathbf{x}_A}(T_{\mathbf{u}_A=1}) \leq \text{evalDim}_{\mathbf{x}_A}(P_{\mathbf{u}_A=1}^{(2)}) \cdot \text{evalDim}_{\mathbf{x}_A}(P_{\mathbf{u}_A=1}^{(3)})$

Proof. By the submultiplicativity property of evaluation dimension (Lemma 4),

$$\text{evalDim}_{\mathbf{x}_A}(T_{\mathbf{u}_A=1}) \leq \text{evalDim}_{\mathbf{x}_A}(P_{\mathbf{u}_A=1}^{(1)}) \cdot \text{evalDim}_{\mathbf{x}_A}(P_{\mathbf{u}_A=1}^{(2)}) \cdot \text{evalDim}_{\mathbf{x}_A}(P_{\mathbf{u}_A=1}^{(3)}).$$

However, it is easy to see that $\text{evalDim}_{\mathbf{x}_A}(P_{\mathbf{u}_A=1}^{(1)}) = 1$. ◀

We will upper bound the evaluation dimension of $P_{\mathbf{u}_A=1}^{(3)}$ with respect to \mathbf{x}_A for any A , and the evaluation dimension of $P_{\mathbf{u}_A=1}^{(2)}$ with respect to \mathbf{x}_A for a random A . Let r_2 be the number of occurrences of the \mathbf{x} -variables among the linear polynomials in $P^{(2)}$ and r_3 be the number of occurrences of the \mathbf{x} -variables in $P^{(3)}$. Since every variable occurs in at most k linear polynomials in T and there are $2n$ \mathbf{x} -variables,

$$r_2 + r_3 \leq 2kn \tag{2}$$

5.1 Evaluation dimension of $P^{(3)}$

► **Lemma 8.** For any $A \subseteq [2n]$, $\text{evalDim}_{\mathbf{x}_A}(P_{\mathbf{u}_A=1}^{(3)}) \leq 2^{\frac{r_3}{ck}}$.

Proof. The evaluation dimension of $P_{\mathbf{u}_A=1}^{(3)}$ with respect to the \mathbf{x}_A -variables cannot exceed 2^b , where b is the number of linear polynomials in $P^{(3)}$. Observe that the degree of $P^{(3)}$ with respect to the \mathbf{x} -variables is less than $\frac{r_3}{ck}$, as every linear polynomial in the product $P^{(3)}$ has more than ck \mathbf{x} -variables. ◀

5.2 Evaluation dimension of $P^{(2)}$

Coloring of linear polynomials. Every linear polynomial in the product $P^{(2)} = P$ (say) has more than one and less than or equal to ck \mathbf{x} -variables. We *color* the linear polynomials in P in such a way that no two linear polynomials with the same color have a common \mathbf{x} -variable. This coloring can be done greedily using at most $(k-1)ck + 1 \leq ck^2$ colors. Let the number of colors used be q ; we will identify these colors with $\{1, \dots, q\}$. Now we can split the product P into at most $q \leq ck^2$ parts (one per color), say $Q^{(1)}, \dots, Q^{(q)}$, such that

every $Q^{(j)}$ is a product of linear polynomials in P that are colored j . This also implies that $Q^{(j)}$ is *multilinear* in the \mathbf{x} -variables. Naturally,

$$P = \prod_{j=1}^q Q^{(j)}.$$

To understand the evaluation dimension of P , we will focus on the polynomials $Q^{(j)}$.

5.2.1 Some more notations and bounds

Let $m_{i,j}$ be the number of linear polynomials in $Q^{(j)}$ with exactly i many \mathbf{x} -variables. Hence, $m_i = \sum_{j \in [q]} m_{i,j}$ for every integer $i \in [2, ck]$. Let A be a random subset of $[2n]$ (in the sense described in Section 3). Let $r_{i,j}$ be the number of linear polynomials in $Q^{(j)}$ with strictly more than i \mathbf{x} -variables and exactly i \mathbf{x}_A -variables. Note that only such linear polynomials with at least one \mathbf{x}_A -variable, but not all \mathbf{x} -variables are \mathbf{x}_A -variables, contribute to the evaluation dimension of P with respect to \mathbf{x}_A . We will refer to such linear polynomials as *partially touched* (by A) linear polynomials. The expected value of $r_{i,j}$ over the random choice of A is

$$\begin{aligned} \mathcal{E}[r_{i,j}] &= \sum_{\ell=i+1}^{ck} \binom{\ell}{i} \cdot \frac{1}{2^\ell} \cdot m_{\ell,j} \\ &\geq \frac{i+1}{2^{ck}} \cdot \sum_{\ell=i+1}^{ck} m_{\ell,j} \\ &\geq \frac{1}{2^{ck-1}} \cdot \sum_{\ell=i+1}^{ck} m_{\ell,j} \quad (\text{as } i \geq 1) \end{aligned} \tag{3}$$

The above expression for the expectation can be derived from the observation that a linear polynomial with ℓ \mathbf{x} -variables ($\ell > i$) has exactly i \mathbf{x}_A -variables with probability $\binom{\ell}{i} \cdot \frac{1}{2^\ell}$. We will see how $r_{i,j}$ contributes to the evaluation dimension of P later. But, first, in order to get a handle on the value of $r_{i,j}$ we would like to argue that it is close to its expected value with high probability. Since $Q^{(j)}$ is multilinear, if $\mathcal{E}[r_{i,j}]$ is sufficiently large, we can apply Chernoff bound on $r_{i,j}$ and show that $(1 - \delta)\mathcal{E}[r_{i,j}] \leq r_{i,j} \leq (1 + \delta)\mathcal{E}[r_{i,j}]$ with high probability. By Equation 3, expectation of $r_{i,j}$ is large if $\sum_{\ell=i+1}^{ck} m_{\ell,j}$ is large. This motivates us to split $Q^{(j)}$ further depending on the value of $\sum_{\ell=i+1}^{ck} m_{\ell,j}$.

5.2.2 Splitting $Q^{(j)}$ further

Let τ_j be the maximum number less than ck such that

$$\sum_{\ell=\tau_j+1}^{ck} m_{\ell,j} \geq \frac{n}{ck^2 \cdot \Delta}, \tag{4}$$

where $\Delta = \Delta(k)$ is a sufficiently large constant, dependent on k , to be fixed later in Section 6. Let $Q^{(j)}$ be the product of those linear polynomials in $Q^{(j)}$ that contribute to $r_{i,j}$ for $i > \tau_j$, and $\tilde{Q}^{(j)}$ the product of those linear polynomials in $Q^{(j)}$ that contribute to $r_{i,j}$ for $i \in [1, \tau_j]$. By Equation 4,

$$\sum_{i=\tau_j+1}^{ck-1} r_{i,j} \leq \sum_{i=\tau_j+2}^{ck} m_{i,j} < \frac{n}{ck^2 \cdot \Delta} \tag{5}$$

Let $P' = \prod_{j=1}^q Q^{(j)}$ and $\tilde{P} = \prod_{j=1}^q \tilde{Q}^{(j)}$. Then,

$$\text{evalDim}_{\mathbf{x}_A}(P_{\mathbf{u}_A=1}) \leq \text{evalDim}_{\mathbf{x}_A}(\tilde{P}_{\mathbf{u}_A=1}) \cdot \text{evalDim}_{\mathbf{x}_A}(P'_{\mathbf{u}_A=1}),$$

as a linear polynomial contributes to the evaluation dimension of P only if it is partially touched (by A). By Equation 5, the number of linear polynomials in P' is upper bounded by

$$\sum_{j=1}^q \sum_{i=\tau_j+1}^{ck-1} r_{i,j} \leq \frac{n}{ck^2 \cdot \Delta} \cdot q \leq \frac{n}{\Delta} \quad (\text{as } q \leq ck^2)$$

Hence,

$$\text{evalDim}_{\mathbf{x}_A}(P'_{\mathbf{u}_A=1}) \leq 2^{\frac{n}{\Delta}}, \quad (6)$$

as the evaluation dimension cannot exceed 2^b , where b is the number of linear polynomials in P' . By choosing a large enough Δ in the analysis later, we will ensure that $\text{evalDim}_{\mathbf{x}_A}(P'_{\mathbf{u}_A=1})$ is negligible compared to other relevant terms.

5.2.3 Computing evaluation dimension of \tilde{P}

Since $\tilde{Q}^{(j)}$ is a product of those linear polynomials that contribute to $r_{i,j}$ for $i \in [1, \tau_j]$, by Equations 3 and 4,

$$\mathcal{E}[r_{i,j}] \geq \frac{1}{2^{ck-1}} \cdot \frac{n}{ck^2 \cdot \Delta},$$

for every $i \in [1, \tau_j]$. For any fixed $j \in [q]$, $Q^{(j)}$ is multilinear. Hence, by applying Chernoff bound,

$$\Pr\{|r_{i,j} - \mathcal{E}[r_{i,j}]| > \delta \cdot \mathcal{E}[r_{i,j}]\} \leq e^{-\frac{\delta^2 \cdot \mathcal{E}[r_{i,j}]}{3}} \leq e^{-\frac{\delta^2 \cdot n}{3 \cdot 2^{ck-1} ck^2 \Delta}}$$

By union bound, $\Pr\{|r_{i,j} - \mathcal{E}[r_{i,j}]| > \delta \cdot \mathcal{E}[r_{i,j}]\}$ for any $j \in [q]$ and $i \in [1, \tau_j]$, is bounded by,

$$\varepsilon_1 := ck^2 \cdot ck \cdot e^{-\frac{\delta^2 \cdot n}{3 \cdot 2^{ck-1} ck^2 \Delta}} \quad (7)$$

As n is much larger compared to the constants k, c, δ, Δ , the above ‘error probability’ ε_1 is negligible. Hence, with probability at least $1 - \varepsilon_1$,

$$(1 - \delta) \cdot \mathcal{E}[r_{i,j}] \leq r_{i,j} \leq (1 + \delta) \cdot \mathcal{E}[r_{i,j}] \quad (8)$$

for every $j \in [q], i \in [1, \tau_j]$.

Let r_i be the number of linear polynomials in \tilde{P} with more than i \mathbf{x} -variables and exactly i \mathbf{x}_A -variables. Then,

$$\begin{aligned} r_i &= \sum_{j \in [q]: i \in [1, \tau_j]} r_{i,j} \\ \mathcal{E}[r_i] &= \sum_{j \in [q]: i \in [1, \tau_j]} \mathcal{E}[r_{i,j}] \end{aligned}$$

The notation $j \in [q] : i \in [1, \tau_j]$ means the sum is over those $j \in [q]$ for which $i \in [1, \tau_j]$. By Equation 8,

$$(1 - \delta)\mathcal{E}[r_i] \leq r_i \leq (1 + \delta)\mathcal{E}[r_i]$$

with probability at least $1 - \varepsilon_1$. This implies

$$\begin{aligned}
 r_i &\leq (1 + \delta) \cdot \sum_{j \in [q]: i \in [1, \tau_j]} \mathcal{E}[r_{i,j}] \\
 &= (1 + \delta) \cdot \sum_{j \in [q]: i \in [1, \tau_j]} \sum_{\ell=i+1}^{ck} \binom{\ell}{i} \cdot \frac{1}{2^\ell} \cdot m_{\ell,j} \quad (\text{by Equation 3}) \\
 &= (1 + \delta) \cdot \sum_{\ell=i+1}^{ck} \binom{\ell}{i} \cdot \frac{1}{2^\ell} \cdot \sum_{j \in [q]: i \in [1, \tau_j]} m_{\ell,j} \\
 &\leq (1 + \delta) \cdot \sum_{\ell=i+1}^{ck} \binom{\ell}{i} \cdot \frac{1}{2^\ell} \cdot \sum_{j \in [q]} m_{\ell,j} \\
 &= (1 + \delta) \cdot \sum_{\ell=i+1}^{ck} \binom{\ell}{i} \cdot \frac{1}{2^\ell} \cdot m_\ell.
 \end{aligned}$$

Let e_x be the number of occurrences of a variable $x \in \mathbf{x}_A$ in the linear polynomials in \tilde{P} . Then, by the above equation, with probability at least $1 - \varepsilon_1$,

$$\begin{aligned}
 \sum_{x \in \mathbf{x}_A} e_x &= \sum_{i=1}^{ck-1} i \cdot r_i \\
 &\leq (1 + \delta) \cdot \sum_{i=1}^{ck-1} i \cdot \sum_{\ell=i+1}^{ck} \binom{\ell}{i} \cdot \frac{1}{2^\ell} \cdot m_\ell \\
 &= (1 + \delta) \cdot \sum_{i=1}^{ck-1} \sum_{\ell=i+1}^{ck} \binom{\ell-1}{i-1} \cdot \frac{1}{2^\ell} \cdot \ell \cdot m_\ell \\
 &\leq (1 + \delta) \cdot \sum_{\ell=2}^{ck} \sum_{i=1}^{\ell-1} \binom{\ell-1}{i-1} \cdot \frac{1}{2^\ell} \cdot \ell \cdot m_\ell \\
 &= (1 + \delta) \cdot \sum_{\ell=2}^{ck} (2^{\ell-1} - 1) \cdot \frac{1}{2^\ell} \cdot \ell \cdot m_\ell \\
 &= (1 + \delta) \cdot \sum_{\ell=2}^{ck} \left(1 - \frac{1}{2^{\ell-1}}\right) \cdot \frac{1}{2} \cdot \ell \cdot m_\ell \\
 &\leq (1 + \delta) \cdot \left(1 - \frac{1}{2^{ck-1}}\right) \cdot \frac{1}{2} \cdot \sum_{\ell=2}^{ck} \ell \cdot m_\ell
 \end{aligned}$$

Observe that $\sum_{\ell=2}^{ck} \ell \cdot m_\ell$ is the number of occurrences of the \mathbf{x} -variables in P . Hence, $\sum_{\ell=2}^{ck} \ell \cdot m_\ell = r_2$ and so with probability at least $1 - \varepsilon_1$,

$$\sum_{x \in \mathbf{x}_A} e_x \leq (1 + \delta) \cdot \left(1 - \frac{1}{2^{ck-1}}\right) \cdot \frac{r_2}{2} \tag{9}$$

Let $\varepsilon_0 = e^{-\frac{\delta^2 n}{3}}$.

► **Lemma 9.** *With probability at least $1 - (\varepsilon_0 + \varepsilon_1)$ over the random choice of A ,*

$$\text{evalDim}_{\mathbf{x}_A}(\tilde{P}_{\mathbf{u}_A=\mathbf{1}}) \leq \left[\left(1 - \frac{1}{2^{ck-1}}\right) \cdot \frac{r_2}{2n} + 1 \right]^{(1+\delta) \cdot n}$$

Proof. Since A is chosen randomly by picking every $i \in [2n]$ independently at random with probability $\frac{1}{2}$, $|\mathbf{x}_A| \leq (1+\delta) \cdot n$ with probability at least $1 - \varepsilon_0$. The evaluation dimension of \tilde{P} with respect to \mathbf{x}_A cannot exceed the number of distinct \mathbf{x}_A -monomials in \tilde{P} with coefficients from $\mathbb{F}[\mathbf{x} \setminus \mathbf{x}_A]$. The number of such monomials is upper bounded by $\prod_{x \in \mathbf{x}_A} (e_x + 1)$. By AM-GM inequality,

$$\begin{aligned} \prod_{x \in \mathbf{x}_A} (e_x + 1) &\leq \left[\frac{\sum_{x \in \mathbf{x}_A} (e_x + 1)}{|\mathbf{x}_A|} \right]^{|\mathbf{x}_A|} \\ &\leq \left[\frac{(1+\delta) \left(1 - \frac{1}{2^{ck-1}}\right) \cdot \frac{r_2}{2}}{|\mathbf{x}_A|} + 1 \right]^{|\mathbf{x}_A|}, \quad (\text{by Equation 9}) \end{aligned}$$

with probability at least $1 - \varepsilon_1$. Hence, with probability at least $1 - (\varepsilon_0 + \varepsilon_1)$,

$$\prod_{x \in \mathbf{x}_A} (e_x + 1) \leq \left[\left(1 - \frac{1}{2^{ck-1}}\right) \cdot \frac{r_2}{2n} + 1 \right]^{(1+\delta)n},$$

as the above expression increases with $|\mathbf{x}_A|$ and $|\mathbf{x}_A| \in [(1-\delta)n, (1+\delta)n]$ with probability at least $1 - \varepsilon_0$. \blacktriangleleft

► **Corollary 10.** *With probability at least $1 - (\varepsilon_0 + \varepsilon_1)$ over the random choice of A ,*

$$\text{evalDim}_{\mathbf{x}_A}(P_{\mathbf{u}_A=1}^{(2)}) \leq \left[\left(1 - \frac{1}{2^{ck-1}}\right) \cdot \frac{r_2}{2n} + 1 \right]^{(1+\delta)n} \cdot 2^{\frac{n}{\Delta}}$$

Proof. Follows from the above lemma and Equation 6. \blacktriangleleft

5.3 Evaluation dimension of a term

Let T be a product term in a multi- k -ic depth three circuit.

► **Lemma 11.** *With probability at least $1 - (\varepsilon_0 + \varepsilon_1)$ over the random choice of A ,*

$$\text{evalDim}_{\mathbf{x}_A}(T_{\mathbf{u}_A=1}) \leq \left[\left(1 - \frac{1}{2^{2ck}}\right) (k+1) \right]^n \cdot (k+1)^{\delta n},$$

if $c \geq 3, k \geq 4$ and $\Delta = 2^{2ck}$.

Proof. By Claim 7, Lemma 8 and Corollary 10,

$$\text{evalDim}_{\mathbf{x}_A}(T_{\mathbf{u}_A=1}) \leq \left[\left(1 - \frac{1}{2^{ck-1}}\right) \cdot \frac{r_2}{2n} + 1 \right]^{(1+\delta)n} \cdot 2^{\frac{n}{\Delta}} \cdot 2^{\frac{r_3}{ck}}$$

Recall from Equation 2, $r_2 + r_3 \leq 2kn$. Let $r_2 \leq \alpha \cdot 2kn$ and $r_3 \leq (1-\alpha) \cdot 2kn$ where $0 \leq \alpha \leq 1$. Then,

$$\text{evalDim}_{\mathbf{x}_A}(T_{\mathbf{u}_A=1}) \leq \left[\left(1 - \frac{1}{2^{ck-1}}\right) \cdot \alpha k + 1 \right]^n \cdot 2^{\frac{n}{\Delta}} \cdot 2^{\frac{2(1-\alpha)n}{c}} \cdot \left[\left(1 - \frac{1}{2^{ck-1}}\right) k + 1 \right]^{\delta n}$$

Since $2^{\frac{1}{y}} \leq 1 + \frac{1}{y}$ for every $y \geq 1$,

$$\left[\left(1 - \frac{1}{2^{ck-1}}\right) \cdot \alpha k + 1 \right] \cdot 2^{\frac{1}{\Delta}} \cdot 2^{\frac{2(1-\alpha)}{c}} \leq \left[\left(1 - \frac{1}{2^{ck-1}}\right) \alpha k + 1 \right] \cdot \left(1 + \frac{1}{\Delta}\right) \cdot \left(1 + \frac{2(1-\alpha)}{c}\right),$$

as $\Delta \geq 1$ and $c \geq 3$. The quantity $\left[\left(1 - \frac{1}{2^{ck-1}}\right) \alpha k + 1 \right] \cdot \left(1 + \frac{2(1-\alpha)}{c}\right)$ when treated as a function of $\alpha \in [0, 1]$ is maximized at $\alpha = 1$, assuming $c \geq 3, k \geq 4$. Therefore,

$$\begin{aligned} \left[\left(1 - \frac{1}{2^{ck-1}}\right) \cdot \alpha k + 1 \right] \cdot 2^{\frac{1}{\Delta}} \cdot 2^{\frac{2(1-\alpha)}{c}} &\leq \left[\left(1 - \frac{1}{2^{ck-1}}\right) k + 1 \right] \cdot \left(1 + \frac{1}{\Delta}\right) \\ &\leq \left(1 - \frac{1}{2^{2ck}}\right) \cdot (k + 1) \quad (\text{as } \Delta = 2^{2ck}) \end{aligned}$$

This proves the lemma as $\left[\left(1 - \frac{1}{2^{ck-1}}\right)k + 1\right]^{\delta n} \leq (k + 1)^{\delta n}$. ◀

6 Proof of Theorem 2

Following the setting of parameters in Lemma 11, let $c = 3, \Delta = 2^{6k}$ and without loss of generality $k \geq 4$. Also, let

$$\delta = \frac{\ln\left(1 + \frac{1}{2^{2ck+1}}\right)}{2 \cdot \ln(k + 1)} = \frac{\ln\left(1 + \frac{1}{2^{6k+1}}\right)}{2 \cdot \ln(k + 1)},$$

and denote the upper bound in Lemma 11 by $U(k, n, \delta)$.

► **Lemma 12.** *If $g(\mathbf{x}, \mathbf{u})$, as defined in Equation 1, is computed by a multi- k -ic depth three circuit C then the top fanin s of C is at least $2^{\Omega\left(\frac{n}{2^{25k}}\right)}$.*

Proof. By union bound, with probability at least $1 - (\varepsilon_0 + s \cdot \varepsilon_1)$ over the random choice of A , the evaluation dimension of every term in C is upper bounded by $U(k, n, \delta)$. By Equation 7,

$$\varepsilon_1 := ck^2 \cdot ck \cdot e^{-\frac{\delta^2 \cdot n}{3 \cdot 2^{ck-1} \cdot ck^2 \Delta}}.$$

So, if $s \leq e^{\frac{\delta^2 n}{9 \cdot 2^{3k} \cdot k^2 \cdot \Delta}}$ then there exists an A such that evaluation dimension of every term of C is upper bounded by $U(k, n, \delta)$. Otherwise,

$$s > e^{\frac{\delta^2 n}{9 \cdot 2^{3k} \cdot k^2 \cdot \Delta}} = 2^{\Omega\left(\frac{n}{2^{25k}}\right)}$$

and we already have the lower bound. If evaluation dimension of every term is upper bounded by $U(k, n, \delta)$ then following the discussion in Section 4,

$$\begin{aligned} s &\geq \frac{(k + 1)^{(1-\delta)n}}{U(k, n, \delta)} \\ &= \left(1 - \frac{1}{2^{2ck}}\right)^{-n} \cdot (k + 1)^{-2\delta n} = 2^{\Omega\left(\frac{n}{2^{6k}}\right)}, \end{aligned}$$

after plugging in the value of δ from above. ◀

The proof of Theorem 2 is immediate from the above lemma.

7 Discussion

In order to gain a better understanding of the strengths and limitations of the existing complexity measures, like partial derivatives, (projected) shifted partials, evaluation dimension etc., it is perhaps worth exploring some natural and interesting models of circuits for which we still do not know of any super-polynomial lower bound. Such a model of circuits emerging

from our work is *multi- k -ic* formulas: Let x be a variable and g be a gate. The formal degree of x at g , denoted $\deg_x(g)$, is defined as follows. If g is a \times -gate with children g_1 and g_2 then $\deg_x(g) = \deg_x(g_1) + \deg_x(g_2)$. If g is a $+$ -gate with children g_1 and g_2 then $\deg_x(g) = \max\{\deg_x(g_1), \deg_x(g_2)\}$. If g is an input gate labelled with x then $\deg_x(g) = 1$, otherwise $\deg_x(g) = 0$. A formula is multi- k -ic if for every variable x and every gate g , the formal degree of x at g is bounded by k .

- Can we prove super-polynomial lower bounds for constant depth multi- k -ic formulas?
- Can we prove super-polynomial lower bounds for multi- k -ic formulas?

References

- 1 Manindra Agrawal, Chandan Saha, Ramprasad Saptharishi, and Nitin Saxena. Jacobian hits circuits: hitting-sets, lower bounds for depth- d occur- k formulas & depth-3 transcendence degree- k circuits. In *STOC*, pages 599–614, 2012.
- 2 Manindra Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *FOCS*, pages 67–75, 2008.
- 3 Michael A. Forbes and Amir Shpilka. Quasipolynomial-Time Identity Testing of Non-commutative and Read-Once Oblivious Algebraic Branching Programs. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 243–252, 2013.
- 4 Bruno Grenet, Pascal Koiran, Natacha Portier, and Yann Strozecki. The Limited Power of Powering: Polynomial Identity Testing and a Depth-four Lower Bound for the Permanent. In *Proceedings of the 30th Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 127–139, 2011.
- 5 Dima Grigoriev and Marek Karpinski. An exponential lower bound for depth 3 arithmetic circuits. In *STOC*, pages 577–582, 1998.
- 6 Dima Grigoriev and Alexander A. Razborov. Exponential complexity lower bounds for depth 3 arithmetic circuits in algebras of functions over finite fields. In *FOCS*, pages 269–278, 1998.
- 7 Ankit Gupta, Pritish Kamath, Neeraj Kayal, and Ramprasad Saptharishi. Arithmetic circuits: A chasm at depth three. In *Foundations of Computer Science (FOCS)*, pages 578–587, 2013.
- 8 Ankit Gupta, Neeraj Kayal, Pritish Kamath, and Ramprasad Saptharishi. Approaching the chasm at depth four. In *Conference on Computational Complexity (CCC)*, 2013.
- 9 Neeraj Kayal. An exponential lower bound for the sum of powers of bounded degree polynomials. Technical report, Electronic Colloquium on Computational Complexity (ECCC), 2012.
- 10 Neeraj Kayal, Nutan Limaye, Chandan Saha, and Srikanth Srinivasan. An Exponential Lower Bound for Homogeneous Depth Four Arithmetic Formulas. In *Foundations of Computer Science (FOCS)*, pages 61–70, 2014.
- 11 Neeraj Kayal and Chandan Saha. Lower Bounds for Depth Three Arithmetic Circuits with small bottom fanin. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:89, 2014.
- 12 Neeraj Kayal, Chandan Saha, and Ramprasad Saptharishi. A super-polynomial lower bound for regular arithmetic formulas. In *STOC*, pages 146–153, 2014.
- 13 Pascal Koiran. Arithmetic circuits: The chasm at depth four gets wider. *Theor. Comput. Sci.*, 448:56–65, 2012.
- 14 Mrinal Kumar and Shubhangi Saraf. On the power of homogeneous depth 4 arithmetic circuits. In *Foundations of Computer Science (FOCS)*, pages 363–373, 2014.

- 15 Noam Nisan and Avi Wigderson. Lower bounds on arithmetic circuits via partial derivatives. *Computational Complexity*, 6(3):217–234, 1997.
- 16 Ran Raz. Multi-linear formulas for permanent and determinant are of super-polynomial size. *J. ACM*, 56(2), 2009.
- 17 Ran Raz. Elusive functions and lower bounds for arithmetic circuits. *Theory of Computing*, 6(1):135–177, 2010.
- 18 Ran Raz and Amir Yehudayoff. Lower Bounds and Separations for Constant Depth Multi-linear Circuits. *Computational Complexity*, 18(2):171–207, 2009.
- 19 Sébastien Tavenas. Improved bounds for reduction to depth 4 and depth 3. In *MFCS*, pages 813–824, 2013.
- 20 L.G. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM Journal on Computing*, 12(4):641–644, 1983.

Automorphism Groups of Geometrically Represented Graphs*

Pavel Klavík and Peter Zeman

Computer Science Institute, Charles University in Prague,
Address, Czech Republic
{klavik,zeman}@iuuk.mff.cuni.cz.

Abstract

Interval graphs are intersection graphs of closed intervals and *circle graphs* are intersection graphs of chords of a circle. We study automorphism groups of these graphs. We show that interval graphs have the same automorphism groups as trees, and circle graphs have the same as pseudo-forests, which are graphs with at most one cycle in every connected component.

Our technique determines automorphism groups for classes with a strong structure of all geometric representations, and it can be applied to other graph classes. Our results imply polynomial-time algorithms for computing automorphism groups in term of group products.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases automorphism group, geometric intersection graph, interval graph, circle graph

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.540

1 Introduction

The study of symmetries of geometrical objects is an ancient topic in mathematics and its precise formulation led to group theory. Symmetries play an important role in many distinct areas. In 1846, Galois used symmetries of the roots of a polynomial in order to characterize polynomials which are solvable by radicals. Some big objects are highly symmetrical, for instance the well-known Rubik's Cube has 43, 252, 003, 274, 489, 856, 000 symmetries. They can be understood using group theory and used for working with the Rubik's Cube (designing algorithms for solving it, etc.). Symmetries have important applications in differential equations, physics, chemistry, crystallography, etc.

Automorphism Groups of Graphs. The symmetries of a graph X are described by its automorphism group $\text{Aut}(X)$. Every automorphism is a permutation of the vertices which preserves adjacencies and non-adjacencies. Frucht [9] proved that every finite group is isomorphic to the automorphism group of some graph X . General mathematical structures can be encoded by graphs [18] while preserving automorphism groups.

Most graphs are asymmetric, i.e., have only the trivial automorphism [14]. However, many combinatorial and graph theory results rely on highly symmetrical graphs. Automorphism groups are important for studying large objects, since these symmetries allow one to simplify and understand the objects. This algebraic approach is together with the recursion and counting arguments the only technique known for working with big objects.

*Supported by CE-ITI (P202/12/G061 of GAČR) and Charles University as GAUK 196213. Many omitted details and proofs are in the full version: [arXiv:1407.2136](https://arxiv.org/abs/1407.2136).

Highly symmetrical large graphs with nice properties are often constructed algebraically from small graphs. For instance, Hoffman-Singleton graph is a 7-regular graph of diameter 2 with 50 vertices [19]. It has 252000 automorphisms and can be constructed from 25 “copies” of a small multigraph with 2 vertices and 7 edges [26]. Similar constructions are used in designing large computer networks [7, 34]. For instance the well-studied degree-diameter problem asks, given integers d and k , to find a maximal graph X with diameter d and degree k . Such graphs are desirable networks having small degrees and short distances. Currently, the best constructions are highly symmetrical graphs made using groups [27].

For a class \mathcal{C} of graphs, let $\text{Aut}(\mathcal{C})$ denote its automorphism groups, i.e., $\text{Aut}(\mathcal{C}) = \{\text{Aut}(X) : X \in \mathcal{C}\}$. We say that a class \mathcal{C} of graphs is *universal* if every finite group is isomorphic to some group in $\text{Aut}(\mathcal{C})$, and *non-universal* otherwise.

The oldest non-trivial result concerning automorphism groups of restricted graph classes is for trees (TREE) by Jordan [21] from 1869. He proved that $\text{Aut}(\text{TREE})$ contains precisely those groups that can be obtained from the trivial group by a sequence of two operations: the direct product and the wreath product with a symmetric group. The direct product constructs the automorphisms that act independently on non-isomorphic subtrees and the wreath product constructs the automorphisms that permute isomorphic subtrees.

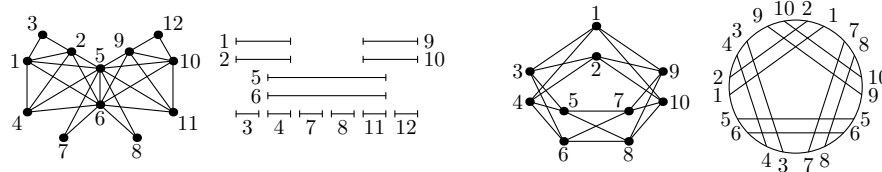
Graph Isomorphism Problem. This famous problem asks whether two input graphs X and Y are the same up to a relabeling. This problem is obviously in NP, and not known to be polynomially-solvable or NP-complete. Aside integer factorization, this is a prime candidate for an intermediate problem with the complexity between P and NP-complete. It belongs to the low hierarchy of NP [30], which implies that it is unlikely NP-complete. (Unless the polynomial-time hierarchy collapses to its second level.) The graph isomorphism problem is known to be polynomially solvable for the classes of graphs with bounded degree [24] and with excluded topological subgraphs [16].

The graph isomorphism problem is closely related to computing generators of an automorphism group. Assuming X and Y are connected, we can test $X \cong Y$ by computing generators of $\text{Aut}(X \dot{\cup} Y)$ and checking whether there exists a generator which swaps X and Y . For the converse relation, Mathon [25] proved that generators of the automorphism group can be computed using $\mathcal{O}(n^4)$ instances of graph isomorphism. Compared to graph isomorphism, automorphism groups of restricted graph classes are much less understood.

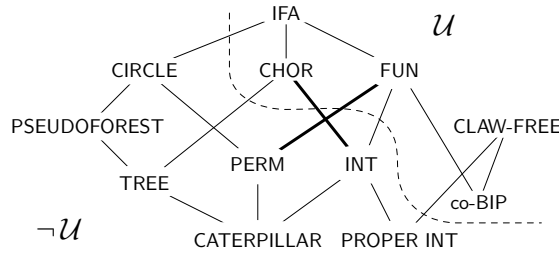
Geometric Representations. In this paper, we study automorphism groups of geometrically represented graphs. The main question is how the geometry influences their automorphism groups. For instance, the geometry of a sphere translates to 3-connected planar graphs which have unique embeddings [32]. Thus, their automorphism groups are so called spherical groups which are automorphism groups of tilings of a sphere. For general planar graphs, the automorphism groups are more complex and they were described by Babai [1] using semidirect products of spherical and symmetric groups; see also [8].

We focus on intersection representations. An *intersection representation* \mathcal{R} of a graph X is a collection $\{R_v : v \in V(X)\}$ such that $uv \in E(X)$ if and only if $R_u \cap R_v \neq \emptyset$; the intersections encode the edges. To get nice graph classes, one typically restricts the sets R_v to particular classes of geometrical objects; for an overview, see the classical books [15, 31]. We show that a well-understood structure of all intersection representations allows one to determine the automorphism group. In particular, we study interval graphs and circle graphs, and our technique can be also applied to other graph classes.

To obtain an *interval representation* of a graph, we restrict the sets R_v to closed intervals of the real line. In a *circle representation*, the sets R_v are chords of a circle. A graph is an



■ **Figure 1** On the left, an interval graph and one of its interval representations. On the right, a circle graph and one of its circle representations.



■ **Figure 2** The inclusions between considered graph classes. We denote universal classes by \mathcal{U} , and non-universal by $\neg\mathcal{U}$. The bold edges are two infinite hierarchies, discussed in Section 6.

interval (resp. *circle*) *graph* if it has an interval (resp. circle) representation; see Fig. 1 for examples. We denote these classes by INT and CIRCLE, respectively.

Related Graph Classes. Figure 2 depicts graph classes important for this paper. *Caterpillar graphs* (CATERPILLAR) are trees with every leaf attached to a central path. They form the intersection of trees and interval graphs. *Chordal graphs* (CHOR) are intersection graphs of subtrees of trees. They contain no induced cycles of length four or more and naturally generalize interval graphs. Chordal graphs have universal automorphism groups [23].

Pseudoforests (PSEUDOFORREST) are graphs for which every connected component is a *pseudotree*, where pseudotree is a connected graph with at most one cycle. Each pseudoforest is a circle graph. The automorphism groups of pseudoforests can be constructed from the automorphism groups of trees by semidirect products with cyclic and dihedral groups, which constructs the automorphisms rotating/reflecting unique cycles.

Function graphs (FUN) are intersection graphs of continuous functions $f : [0, 1] \rightarrow \mathbb{R}$. Equivalently, function graphs are *co-comparability graphs* which means their complements can be transitively oriented. Every interval graph is a co-comparability graph since disjoint pairs of intervals can be oriented from left to right. *Permutation graphs* (PERM) are function graphs which can be represented by linear functions.

Claw-free graphs (CLAW-FREE) are graphs with no induced $K_{1,3}$. Roberts proved [28] that $\text{CLAW-FREE} \cap \text{INT}$ is equal to the class of *proper interval graphs* (PROPER INT) which are interval graphs with representations in which no interval properly contains another. The complements of bipartite graphs (co-BIP) are universal. They are claw-free and contained in function graphs since each bipartite graph is transitively orientable.

Interval filament graphs (IFA) are intersection graphs of the following sets. For every R_u , we choose an interval $[a, b]$ and R_u is a continuous function $[a, b] \rightarrow \mathbb{R}$ such that $R_u(a) = R_u(b) = 0$ and $R_u(x) > 0$ for $x \in (a, b)$. They generalize circle, chordal, and function graphs.

► **Theorem 1.**

- (i) $\text{Aut}(\text{INT}) = \text{Aut}(\text{TREE})$,
- (ii) $\text{Aut}(\text{connected PROPER INT}) = \text{Aut}(\text{CATERPILLAR})$,
- (iii) $\text{Aut}(\text{CIRCLE}) = \text{Aut}(\text{PSEUDOFORREST})$.

Concerning (i), this equality is not well known. It was stated by Hanlon [17] without a proof in the conclusion of his paper from 1982 on enumeration of interval graphs. Our structural analysis is based on PQ-trees [2] which combinatorially describe all interval representations of an interval graph. It explains this equality and further solves an open problem of Hanlon: for a given interval graph, to construct a tree with the same automorphism group. Without PQ-trees, this equality is surprising since these classes are very different. Caterpillar graphs which form their intersection have very limited groups and we characterize them in Lemma 5. The result (ii) easily follows from the known properties of proper interval graphs and our structural understanding of $\text{Aut}(\text{INT})$.

Using PQ-trees, Colbourn and Booth [4] give a linear-time algorithm to compute permutation generators of the automorphism group of an interval graph. In comparison, our description allows to construct an algorithm which outputs the automorphism group in the form of group products which reveals its structure.

Concerning (iii), we are not aware of any results on automorphism groups of circle graphs. One inclusion is trivial since $\text{PSEUDOFORREST} \subsetneq \text{CIRCLE}$. The other one is based on split-trees which describe all representations of circle graphs. The semidirect product with a cyclic or a dihedral group corresponds to the rotations/reflections of the central vertex of a split-tree. Geometrically, it corresponds to the rotations/reflections of the entire symmetric representation. Our approach is similar to the algorithm for circle graph isomorphism [20].

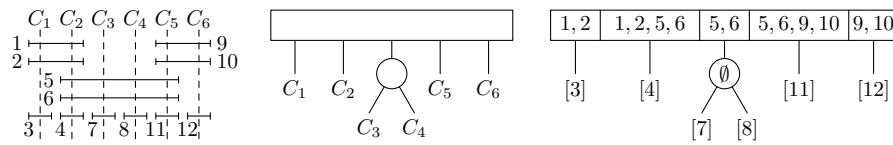
Structure. We describe the automorphism groups of interval graphs in Section 2 and of circle graphs in Section 3. In Section 4, we interpret our results in terms of actions of automorphism groups on sets of all representations. We explain our general technique for determining the automorphism group from the geometric structure of all representations. Further, we relate it to well-known results of map theory. Our results are constructive and lead to polynomial-time algorithms computing automorphism groups of interval and circle graphs; see Section 5. We conclude with several open problems.

Preliminaries. We use X and Y for graphs, M , T and S for trees and G , H and others for groups. The vertices and edges of X are $V(X)$ and $E(X)$. The set of all maximal cliques is denoted by $\mathcal{C}(X)$. A permutation π of $V(G)$ is an *automorphism* if $uv \in E(G) \iff \pi(u)\pi(v) \in E(G)$. We use \mathbb{S}_n , \mathbb{D}_n and \mathbb{Z}_n for the *symmetric*, *dihedral* and *cyclic groups*.

We quickly define semidirect and wreath products; see [3, 29] for details. Given two groups N and H , and a group homomorphism $\varphi: H \rightarrow \text{Aut}(N)$, we can construct a new group $N \rtimes_{\varphi} H$ as the Cartesian product $N \times H$ with the operation defined as $(n_1, h_1) \cdot (n_2, h_2) = (n_1 \cdot \varphi(h_1)(n_2), h_1 \cdot h_2)$. The group $N \rtimes_{\varphi} H$ is called the *semidirect product of N and H with respect to the homomorphism φ* . The *wreath product* $G \wr \mathbb{S}_n$ is a shorthand for $G^n \rtimes_{\psi} \mathbb{S}_n$ where ψ is defined naturally by $\psi(\pi) = (g_1, \dots, g_n) \mapsto (g_{\pi(1)}, \dots, g_{\pi(n)})$.

2 Automorphism Groups of Interval Graphs

In this section, we prove Theorem 1(i) and (ii). We introduce PQ-trees which describe all interval representations. Using them, we derive a characterization of $\text{Aut}(\text{INT})$ which we prove to be equivalent to Jordan's characterization of $\text{Aut}(\text{TREE})$. We solve the open



■ **Figure 3** An ordering of the maximal cliques, and the corresponding PQ-tree and MPQ-tree. The P-nodes are denoted by circles, the Q-nodes by rectangles.

problem of Hanlon [17] by constructing for a given interval graph a tree with the same automorphism group, and we also show the converse construction.

PQ-trees. Booth and Lueker [2] invented a data structure called *PQ-tree* to solve the long-standing open problem of recognizing interval graphs in linear time. It is based on the following characterization of interval graphs.

► **Lemma 2** (Fulkerson and Gross [10]). *A graph X is an interval graph if and only if there exists an ordering of the maximal cliques such that for every $x \in V(X)$ the maximal cliques containing x appear consecutively.*

PQ-trees are rooted trees with two types of inner nodes: *P-nodes* and *Q-nodes*. The leaves correspond one-to-one to the maximal cliques of X . For every inner node, the order of its children is fixed. The order of the leaves from left to right is called a *frontier*. See Fig. 3.

There are two *equivalence transformations*: (i) an arbitrary permutation of the children of a P-node, and (ii) a reversal of the order of the children of a Q-node. Two PQ-trees are *equivalent* if we can get one from the other by a sequence of equivalence transformations. Booth and Lueker [2] proved that for every interval graph there exists a unique PQ-tree representing all possible orderings of the maximal cliques as frontiers of its equivalent trees. In other words, this PQ-tree encodes all interval representations.

Every automorphism $\alpha \in \text{Aut}(X)$ induces some permutation of the maximal cliques $\mathcal{C}(X)$. However, multiple automorphisms can reorder $\mathcal{C}(X)$ in the same way. Two vertices are called *twin vertices* if they belong to the same maximal cliques. Two automorphisms of X can permute the maximal cliques the same but permute the twin vertices differently. PQ-trees describe the structure of the maximal cliques of an interval graph, but to determine $\text{Aut}(X)$ we need some additional information about the twin vertices.

MPQ-trees. A *modified PQ-tree* is created from a PQ-tree by adding information about the vertices. They were described by Korte and Möhring [22] to simplify linear-time recognition of interval graphs. An equivalent idea was already used by Coulborn and Booth [4] for computing automorphism groups of interval graphs.

Suppose that T is a PQ-tree corresponding to an interval graph X . In the MPQ-tree M , we assign sets, called *sections*, to the nodes of T ; see Fig. 3. The leaves and P-nodes have each assigned only one section, while Q-nodes have one section for every child. We assign these sections in the following way:

- For every leaf L , the section $\text{sec}(L)$ contains those vertices that are only in the maximal clique represented by L , and no other maximal cliques.
- For every P-node P , the section $\text{sec}(P)$ contains those vertices that are in all maximal cliques of the subtree of P , and no other maximal cliques.
- For every Q-node Q and its children T_1, \dots, T_n , the section $\text{sec}_i(Q)$ contains those vertices that are in the maximal cliques represented by the leaves of the subtree of T_i and also



■ **Figure 4** (a) Construction of the operation (d) from Lemma 4. (b) Trees attached to a path by their roots. Since the automorphism group is not isomorphic to $(\text{Aut}(T_1) \times \text{Aut}(T_2) \times \text{Aut}(T_3)) \rtimes_{\varphi} \mathbb{Z}_2$, we fix it by subdividing v_1v_2 and v_2v_3 .

some other T_j , but not in any other maximal clique outside the subtree of Q . We put $\text{sec}(Q) = \text{sec}_1(Q) \cup \dots \cup \text{sec}_n(Q)$.

Two vertices are in the same sections of an MPQ-tree if and only if they are twin vertices.

Automorphisms of PQ and MPQ-trees. Let T be a PQ-tree corresponding to an interval graph X . A sequence ε of equivalence transformations is an *automorphism of T* if there exists $\alpha \in \text{Aut}(X)$ such that α reorders the maximal cliques $\mathcal{C}(X)$ in the same way as ε . We get a group homomorphism $\phi : \text{Aut}(X) \rightarrow \text{Aut}(T)$ where $\phi(\alpha)$ is the unique automorphism of T permuting $\mathcal{C}(X)$ the same as α . By the first isomorphism theorem, we have that $\text{Aut}(T)$ is isomorphic to a subgroup of $\text{Aut}(X)$.

Let M be the MPQ-tree with its nodes N_1, \dots, N_k . An *automorphism of a node N* is a permutation of the vertices inside the sections of N . For a P-node, $\text{Aut}(N)$ is isomorphic to \mathbb{S}_n . For a Q-node, it is a direct product of symmetric groups. An *automorphism of M* is a $(k + 1)$ -tuple $(\nu_{N_1}, \dots, \nu_{N_k}, \varepsilon)$ where ν_{N_i} is an automorphism of the node N_i and ε is an automorphism of the underlying PQ-tree T . Each automorphism of N uniquely corresponds to an automorphism α of X , so $\text{Aut}(M) \cong \text{Aut}(X)$.

Automorphism Groups of Interval Graphs. To get $\text{Aut}(X)$, we just need to determine $\text{Aut}(M)$. We also make use of the following result due to Jordan:

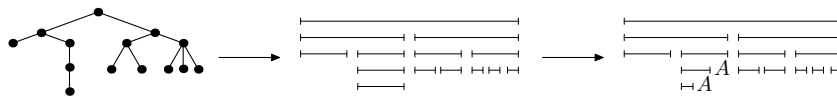
► **Theorem 3** (Jordan [21]). *If X_1, \dots, X_n are pairwise non-isomorphic connected graphs and X is the disjoint union of k_i copies of X_i , then $\text{Aut}(X) \cong \text{Aut}(X_1) \wr_{k_1} \times \dots \times \text{Aut}(X_n) \wr_{k_n}$.*

► **Lemma 4.** *A group $G \in \text{Aut}(\text{INT})$ if and only if $G \in \mathcal{I}$, where the class \mathcal{I} is defined inductively as follows:*

- (a) $\{1\} \in \mathcal{I}$.
- (b) If $G_1, G_2 \in \mathcal{I}$, then $G_1 \times G_2 \in \mathcal{I}$.
- (c) If $G \in \mathcal{I}$ and $n \geq 2$, then $G \wr \mathbb{S}_n \in \mathcal{I}$.
- (d) If $G_1, G_2, G_3 \in \mathcal{I}$ and $G_1 \cong G_3$, then $(G_1 \times G_2 \times G_3) \rtimes_{\varphi} \mathbb{Z}_2 \in \mathcal{I}$, where $\varphi : \mathbb{Z}_2 \rightarrow \text{Aut}(G_1 \times G_2 \times G_3)$ is the homomorphism defined as $\varphi(0) = \text{id}$ and $\varphi(1) = (g_1, g_2, g_3) \mapsto (g_3, g_2, g_1)$.

Proof (Sketch). We first prove that $\mathcal{I} \subseteq \text{Aut}(\text{INT})$. Clearly $\{1\} \in \text{Aut}(\text{INT})$. It remains to show that the class $\text{Aut}(\text{INT})$ is closed under (b), (c) and (d). For (b), we can show this by attaching two interval graphs X_1 and X_2 on an asymmetric interval graph. Clearly, the resulting graph represents the direct product of $\text{Aut}(X_1)$ and $\text{Aut}(X_2)$. For (c), let $G \in \text{Aut}(\text{INT})$ and $n \geq 2$. There exists an interval graph Y such that $\text{Aut}(Y) \cong G$. We construct X as the disjoint union of n copies of Y . By Theorem 3, it follows that $\text{Aut}(X) \cong G \wr \mathbb{S}_n$. For (d), we construct an interval graph X by attaching X_1, X_2 and X_3 to a path as in Fig. 4a, where $\text{Aut}(X_i) = G_i$ and $X_1 \cong X_3$. Then $\text{Aut}(X) \cong (G_1 \times G_2 \times G_3) \rtimes_{\varphi} \mathbb{Z}_2$.

For the converse, we show that $\text{Aut}(M) \in \mathcal{I}$. We have three cases for the root of M . For a P-node, $\text{Aut}(M)$ is determined by the automorphism groups of its subtrees using



■ **Figure 5** First, we place the intervals according to the structure of the tree. We get $\text{Aut}(X) \cong \mathbb{S}_3 \times \mathbb{S}_2 \times \mathbb{S}_3$, but $\text{Aut}(T) \cong \mathbb{S}_2 \times \mathbb{S}_3$. We fix this by adding copies of an asymmetric path A which has the trivial automorphism group.

Theorem 3, so the operations (b) and (c) are sufficient. For an asymmetric Q-node, $\text{Aut}(M)$ is the direct product of the automorphism groups of its subtrees. For a symmetric Q-node, we apply the operation (d) where G_1 corresponds to the automorphisms of the left part of the Q-node, G_2 to the middle part and G_3 to the right part. The semidirect product with \mathbb{Z}_2 corresponds to reversing the Q-node. ◀

This lemma connects $\text{Aut}(\text{INT})$ and the geometrical structure of an interval representation. The operation (b) applies to non-isomorphic independent parts of the representation, (c) to isomorphic parts which can be arbitrary permuted, and (d) to parts which can only be reflected vertically.

Proof of Theorem 1(i). It easily follows from Lemma 4 that $\text{Aut}(\text{INT}) = \text{Aut}(\text{TREE})$. We show that (d) can be expressed using (b) and (c). Assuming $G_1 \cong G_3$, we get

$$(G_1 \times G_2 \times G_3) \rtimes_{\varphi} \mathbb{Z}_2 \cong (G_1 \times G_3) \rtimes_{\varphi} \mathbb{Z}_2 \times G_2 \cong G_1 \wr \mathbb{Z}_2 \times G_2.$$

An alternative proof shows that the automorphism groups of trees are closed under (d). Suppose that $G_1, G_2, G_3 \in \text{Aut}(\text{TREE})$ and $G_1 \cong G_3$. Then there exist trees T_1, T_2 and T_3 such that $\text{Aut}(T_i) \cong G_i$ and $T_1 \cong T_3$. We construct a tree T by attaching T_1, T_2 , and T_3 to a path by the roots, as shown in Fig. 4b. ◀

From Interval Graphs to Trees. We solve the open problem of Hanlon [17]. For an interval graph X , we construct a tree T such that $\text{Aut}(X) \cong \text{Aut}(T)$. Consider the MPQ-tree M for X . We know that $\text{Aut}(M) \cong \text{Aut}(X)$ and we just need to encode the structure of M into T . We do this inductively.

Suppose a P-node P is in the root. Then its subtrees can be encoded by trees and we just attach them to a common root. Further, if $\text{sec}(P)$ is non-empty, we attach a star with $|\text{sec}(P)|$ leaves to the root. As before, we possibly need to modify this by subdivision, and we get $\text{Aut}(T) \cong \text{Aut}(M)$.

Let a Q-node Q be in the root. If Q is asymmetric, we attach the trees corresponding to the subtrees of Q and stars corresponding to the vertices of equal sections of Q to an asymmetric path. If Q is symmetric, then $\text{Aut}(M) \cong (G_1 \times G_2 \times G_3) \rtimes \mathbb{Z}_2$ and we just attach trees T_1, T_2 and T_3 to a path as in Fig. 4b. In both cases, $\text{Aut}(T) \cong \text{Aut}(M)$.

From Trees to Interval Graphs. For a rooted tree T , we construct an interval graph X such that $\text{Aut}(T) \cong \text{Aut}(X)$ as follows. We place the intervals by copying the structure of T , as shown in Fig. 5. Each interval is contained exactly in the intervals of its ancestors. If T contains a vertex with only one child, then $\text{Aut}(T) < \text{Aut}(X)$. This can be fixed by adding asymmetric paths, as in Fig. 5.

Automorphism Groups of Proper Interval Graphs. As an application of the previously derived characterization of $\text{Aut}(\text{INT})$, we show that the automorphism groups of connected proper interval graphs are the same as the automorphism groups of caterpillars. First, we derive a characterization of $\text{Aut}(\text{CATERPILLAR})$.

► **Lemma 5.** *Let X be a caterpillar graph and let P be the central path.*

- (i) *If no automorphism swaps the path P , then the group $\text{Aut}(X)$ is isomorphic to a direct product of symmetric groups.*
- (ii) *If there exists an automorphism of X that swaps the path P , then*

$$\text{Aut}(X) \cong (G_1 \times G_2 \times G_3) \rtimes_{\varphi} \mathbb{Z}_2,$$

where G_2 is isomorphic to \mathbb{S}_k , $G_1 \cong G_3$ are isomorphic to a direct product of symmetric groups, and φ is the homomorphism defined as $\varphi(0) = \text{id}$ and $\varphi(1) = (g_1, g_2, g_3) \mapsto (g_3, g_2, g_1)$.

Proof (Sketch). The root of an MPQ-tree M representing a caterpillar graph X is a Q-node. All twin classes are trivial, since X is a tree. Each child of the root is either a P-node, or a leaf. All children of every P-node are leaves. If there exist an automorphism that swaps the central path P , then the root is symmetric, otherwise it is asymmetric. We can determine $\text{Aut}(M)$ similarly as in the proof of Lemma 4. ◀

Proof of Theorem 1(ii). According to Corneil [5], the MPQ-tree representing a connected proper interval graph contains only one Q-node with the maximal cliques attached to it. It is possible that the sections of this Q-node are nontrivial. This equality of automorphism groups follows by Lemma 5 and the proof of Lemma 4. ◀

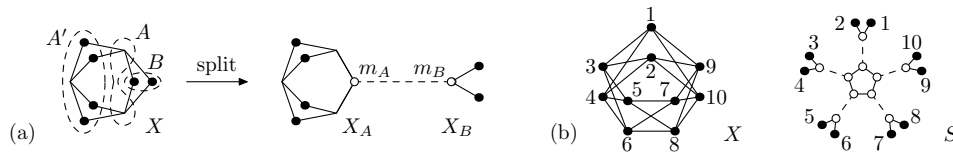
3 Automorphism Groups of Circle Graphs

In this section, we prove Theorem 1(iii). We start by introducing *split decomposition* (used for recognizing circle graphs) which is described by a *split-tree*. Similarly as in Section 2, we show for a split-tree S that $\text{Aut}(S) \cong \text{Aut}(X)$. From now on, we focus on connected circle graphs and we want to establish that their automorphism groups are the same as the automorphism groups of pseudotrees (PSEUDOTREE).

Split Decomposition. A *split* of X is a partition of the set $V(X)$ into four parts A , B , A' and B' such that:

- For every $a \in A$ and every $b \in B$, we have $ab \in E(X)$.
- There is no edge between A' and $B \cup B'$, and between B' and $A \cup A'$.
- Both sides have at least two vertices: $|A \cup A'| \geq 2$ and $|B \cup B'| \geq 2$.

The split decomposition takes *any* split of X , and replaces X by graphs X_A and X_B . The graph X_A is induced by $A \cup A' \cup \{m_A\}$, where m_A is a *marker vertex* adjacent exactly to the vertices in A . The graph X_B is defined similarly for B , B' and m_B ; see Fig. 6a. The decomposition is then applied recursively on X_A and X_B . Graphs containing no splits are called *prime graphs*. According to [11], every prime circle graph has a unique circle representation up to rotations and reflections. It is standard to stop the split decomposition also on *degenerate graphs* which are K_n and $K_{1,n}$ (which clearly are circle graphs). The reason is that these graphs have many splits but are very simple. The fundamental property is that a graph X is a circle graph if and only if X_A and X_B are circle graphs.



■ **Figure 6** (a) An example of a split. The marker vertices are depicted in white. (b) The split-tree S with dashed tree-edges. We have $\text{Aut}(S) \cong \mathbb{Z}_2^5 \rtimes \mathbb{D}_5$.

Split-tree. We encode the steps of the split decomposition by a tree structure. If X contains a split (A, B, A', B') , then we replace X by the graphs X_A and X_B , and connect the marker vertices m_A and m_B by a *tree-edge*. We repeat this recursively on X_A and X_B . The resulting graph is called a *split-tree*, since tree-edges connect prime and degenerate graphs in a tree pattern; see Fig. 6b. Each prime or degenerate graph is a *node* of the split-tree.

In [12], split-trees are defined in terms of graph-labeled trees. However, our definition is more suitable for working with automorphism groups. Cunningham [6] proved that the split-tree S for a graph X is uniquely determined. Clearly, a graph is a circle graph if and only if each node of its split-tree is a circle graph. The following lemma says that the split-tree S captures the adjacencies in X ; we omit the proof.

► **Lemma 6.** *The vertices $x, y \in V(X)$ are adjacent if and only if there exists an alternating path $x, m_1, m_2, \dots, m_k, y$ in the split-tree S such that each m_i is a marker vertex, each $m_{2i-1}m_{2i}$ is a tree-edge and the remaining edges belong to $E(X)$.*

Automorphisms of a Split-tree. The split-tree S is a labeled graph where some vertices are labeled as marker vertices and some edges are labeled as tree-edges. An automorphism of S is required to preserve these labels, so it maps marker vertices only to marker vertices and tree-edges only to tree-edges. We show that the automorphism group of S is isomorphic to $\text{Aut}(X)$.

► **Lemma 7.** *Let S be a split-tree representing X . Then $\text{Aut}(S) \cong \text{Aut}(X)$.*

Proof. First, we show that each $\sigma \in \text{Aut}(S)$ induces a unique automorphism α of X . We define $\alpha = \sigma \upharpoonright_{V(X)}$. By Lemma 6, two vertices $x, y \in V(X)$ are adjacent if and only if there exists an alternating path in S connecting them. Since σ is an automorphism, the existence of this alternating path is preserved between x and y and between $\sigma(x)$ and $\sigma(y)$. Therefore $xy \in E(X) \iff \alpha(x)\alpha(y) \in E(X)$.

For the converse, we show that $\alpha \in \text{Aut}(X)$ induces a unique automorphism $\sigma \in \text{Aut}(S)$. On the non-marker vertices, σ is determined. On the marker vertices, we define σ recursively. Let (A, B, A', B') be a split in X . This split is mapped by α to another split (C, D, C', D') , i.e., $\alpha(A) = C$, $\alpha(A') = C'$, $\alpha(B) = D$, and $\alpha(B') = D'$. By applying the split decomposition to the first split, we get the graphs X_A and X_B with the marker vertices $m_A \in V(X_A)$ and $m_B \in V(X_B)$. Similarly, for the second split we get X_C, X_D with $m_C \in V(X_C)$ and $m_D \in V(X_D)$. Since α is an automorphism, we have that $X_A \cong X_C$ and $X_B \cong X_D$. It follows that the unique split-trees of X_A and X_C are isomorphic, and similarly for X_B and X_D . Therefore, we define $\sigma(m_A) = m_C$ and $\sigma(m_B) = m_D$, and we finish the rest recursively. ◀

► **Lemma 8.** *A connected circle graph X has $\text{Aut}(X) \in \text{Aut}(\text{PSEUDOTREE})$.*

Proof (Sketch). We begin by proving the following characterization:

$$\text{Aut}(\text{PSEUDOTREE}) = \bigcup_{n \geq 1} \text{Aut}(\text{TREE}) \rtimes \mathbb{D}_n \cup \text{Aut}(\text{TREE}) \rtimes \mathbb{Z}_n.$$

Suppose a pseudotree Y contains a cycle, otherwise $\text{Aut}(Y) \in \text{Aut}(\text{TREE}) \rtimes \mathbb{Z}_1$. Then $\text{Aut}(Y)$ preserves the cycle. The subgroup of $\text{Aut}(Y)$ fixing the cycle belongs to $\text{Aut}(\text{FOREST}) = \text{Aut}(\text{TREE})$, and $\text{Aut}(Y)$ acts on the cycle as a dihedral or cyclic group. This can be described by a semidirect product, and so $\text{Aut}(Y) \in \text{Aut}(\text{TREE}) \rtimes \mathbb{D}_n$ or $\text{Aut}(\text{TREE}) \rtimes \mathbb{Z}_n$.

Let X be a connected circle graph and S a split-tree for X . By Lemma 7 we have that $\text{Aut}(X) \cong \text{Aut}(S)$. Since X is a circle graph, each node of S is a prime or degenerate graph. The automorphism group of a degenerate graph is isomorphic to \mathbb{S}_n . According to [11], each circle graph that is prime has a unique circle representation, up to rotations and reflections. It follows that the automorphism group of a prime circle graph is a subgroup of \mathbb{D}_n .

The split tree S consists of prime and degenerate graphs connected by tree-edges. The center of the split-tree is a node or a tree-edge. In the latter case, we subdivide the tree-edge by creating two new marker vertices and connecting them by a normal edge. So, we assume that the center is a node C . Every automorphism of S maps C to C .

We root S by C . Let $N \neq C$ be a node of S . If N is a degenerate graph, then we can arbitrarily permute its isomorphic children. If N is a prime graph, then we can only reverse the order of its children. This is because the vertex of N which is connected by a tree-edge with the parent of N has to be fixed. The subgroup of $\text{Aut}(S)$ that fixes C is in $\text{Aut}(\text{FOREST}) = \text{Aut}(\text{TREE})$, similarly as for interval graphs.

If the center C is a degenerate graph, then $\text{Aut}(S) \in \text{Aut}(\text{TREE})$ since it closed under (b) and (c) of Lemma 4. Otherwise, C is a prime graph and $\text{Aut}(S)$ acts on C as a subgroup of a dihedral group. Therefore, $\text{Aut}(S) \in \text{Aut}(\text{PSEUDOTREE})$. ◀

The above lemma geometrically describes automorphisms of circle graphs. The center C corresponds to the essential geometrical structure of X , and it can be rotated and possibly reflected. The remainder of X is attached to C via the structure of S , so it is less free. We note that the automorphism groups $\text{Aut}(\text{PSEUDOFORREST})$ can be constructed from $\text{Aut}(\text{PSEUDOTREE})$ by Theorem 3.

We are ready to prove that $\text{Aut}(\text{CIRCLE}) = \text{Aut}(\text{PSEUDOFORREST})$:

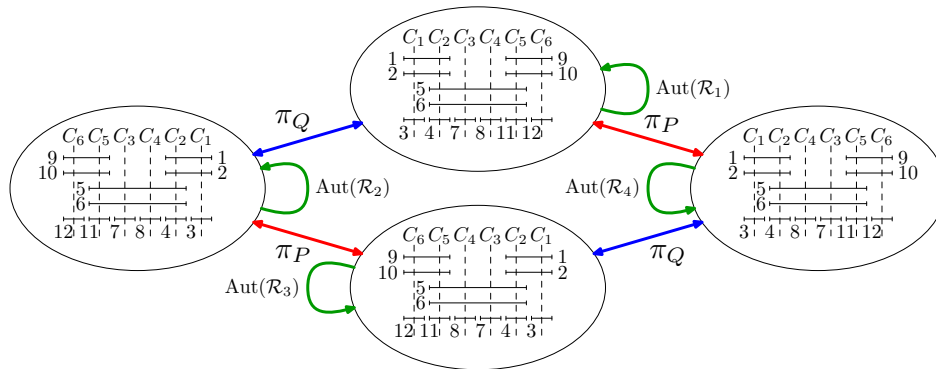
Proof of Theorem 1(iii). Each connected circle graph X has $\text{Aut}(X) \in \text{Aut}(\text{PSEUDOTREE})$ according to Lemma 8. Since every pseudotree is a connected circle graph, these two classes have the same automorphism groups. Circle graphs and pseudotrees are closed under disjoint unions, hence the equality follows. ◀

4 Automorphism Groups Acting on Intersection Representations

We denote by \mathfrak{Rep} the set of all intersection representations of a graph X . Every automorphism $\pi \in \text{Aut}(X)$ creates from $\mathcal{R} \in \mathfrak{Rep}$ another representation \mathcal{R}' such that $R'_{\pi(u)} = R_u$; so π swaps the labels of the sets of \mathcal{R} . We denote \mathcal{R}' as $\pi(\mathcal{R})$, and $\text{Aut}(X)$ acts on \mathfrak{Rep} .

The general set \mathfrak{Rep} is too large. Therefore it is more convenient to define a suitable equivalence relation \sim . We factorize \mathfrak{Rep} by \sim and we work with \mathfrak{Rep}/\sim , which contains exactly one representation from every equivalence class. It is reasonable to assume that \sim is a congruence with respect to the action of $\text{Aut}(X)$, which means that for every $\mathcal{R} \sim \mathcal{R}'$ and $\pi \in \text{Aut}(X)$, we have $\pi(\mathcal{R}) \sim \pi(\mathcal{R}')$. We consider the induced action of $\text{Aut}(X)$ on \mathfrak{Rep}/\sim .

We assume that stabilizer of $\mathcal{R} \in \mathfrak{Rep}/\sim$ is a normal subgroup $\text{Aut}(\mathcal{R})$ of $\text{Aut}(X)$ which describes automorphisms inside this representation. The quotient $\text{Aut}(X)/\text{Aut}(\mathcal{R})$ describes all morphisms which change one representation in the orbit of \mathcal{R} into another one. Our strategy for understanding $\text{Aut}(X)$ is by decomposing it geometrically into $\text{Aut}(\mathcal{R})$,



■ **Figure 7** An interval graph with four non-equivalent representations. Its MPQ-tree M , depicted in Fig. 3, has one Q-node and one P-node. The graph has three classes of twin vertices of size two, so $\text{Aut}(\mathcal{R}) \cong \mathbb{S}_2^3$. The quotient group $\text{Aut}(T)$ is generated by two automorphism: π_Q corresponding to flipping the Q-node, and π_P corresponding to permuting the P-node. We have $\text{Aut}(T) \cong \mathbb{Z}_2^2$.

which is mostly very simple, and $\text{Aut}(X)/\text{Aut}(\mathcal{R})$, for which we need to understand the structure of all representations.

This approach is inspired by well-known results in map theory. A *map* \mathcal{M} is a 2-cell embedding of a graph; i.e, aside vertices and edges, it prescribes a rotation scheme for the edges incident with each vertex. One defines $\text{Aut}(\mathcal{M})$ as the subgroup of $\text{Aut}(X)$ which preserves/reflects the rotational schemes. Unlike $\text{Aut}(X)$, we know that $\text{Aut}(\mathcal{M})$ is always small and can be easily determined in polynomial time. But the quotient $\text{Aut}(X)/\text{Aut}(\mathcal{M})$ describes morphisms between different maps and can be very complicated.

Interval Graphs. For an interval graph X , the set \mathfrak{Rep} consists of all assignments of closed intervals which define X . It is natural to consider two interval representations equivalent if one can be transformed into the other by continuous shifting of the endpoints of the intervals while preserving the correctness of the representation. Then each representation of \mathfrak{Rep}/\sim corresponds to a different ordering of the maximal cliques from left to right. Figure 7 depicts an interval graph with four different non-equivalent representations in \mathfrak{Rep}/\sim .

We interpret our results of Section 2 in terms of the action of $\text{Aut}(X)$ on \mathfrak{Rep} . We proved that $\text{Aut}(X) \cong \text{Aut}(M)$ where M is the MPQ-tree. If an automorphism is in the stabilizer, then it fixes the ordering of the maximal cliques and it can only permute twin vertices. Therefore $\text{Aut}(\mathcal{R})$ is a product of symmetric groups, one for each equivalence class of twin vertices. In the description using MPQ-trees, each equivalence class corresponds to a set of vertices which are contained in the same sections. Every stabilizer is the same and every orbit of the action of $\text{Aut}(X)$ is isomorphic. Different orderings of the maximal cliques correspond to different reorderings of the PQ-tree. The defined $\text{Aut}(T)$ describes morphisms of representations belonging to one orbit of the action of $\text{Aut}(X)$, so these representations are the same up to the labeling of the intervals. It is the quotient group $\text{Aut}(M)/\text{Aut}(\mathcal{R})$ which is isomorphic to $\text{Aut}(X)/\text{Aut}(\mathcal{R})$.

Circle Graphs. For a circle graph X , the set \mathfrak{Rep} consists of all assignments of chords of a circle which define X . Two representations are considered equivalent if one can be transformed into other by (i) continuous shifting of chords while preserving the representation and (ii) swapping two chords with the same neighbors such that there is no other endpoint in between them. We call two vertices x and y *semi-twin vertices* if $N(x) = N(y)$. They

again form equivalence classes, and two representation are equivalent if they have the same circular ordering of chords up to permuting semi-twin vertices.

We interpret the results of Section 3 in terms of the action of $\text{Aut}(X)$. It follows that $\text{Aut}(\mathcal{R})$ is a direct product of symmetric groups, corresponding to permuting semi-twin vertices. It consists of all automorphisms which fix marker vertices of the split tree S . The quotient $\text{Aut}(X)/\text{Aut}(\mathcal{R})$ describes all structural transformations of the split tree. For the central node C , rotation/reflection is possible, so we get a subgroup of \mathbb{D}_n . If $N \neq C$ is a prime graph, we can only apply the geometric reflection with the axis perpendicular to the chord of the marker vertex, so their symmetries are trivial or \mathbb{Z}_2 . For a degenerate graph $N \neq C$, one can arbitrary permute isomorphic subtrees, so it is a direct product of wreath products with symmetric groups. So $\text{Aut}(X)/\text{Aut}(\mathcal{R}) \in \text{Aut}(\text{PSEUDOTREE})$.

5 Algorithms for Computing Automorphism Groups

We have described the structure of automorphism groups of interval and circle graphs. In this section, we briefly explain algorithmic implications of our results which allow to compute automorphism groups in terms of basic groups \mathbb{Z}_n , \mathbb{D}_n and \mathbb{S}_n , and their group products. This description is much better than just outputting permutations generating $\text{Aut}(X)$. Many tools of the computational group theory are devoted to getting better understanding of an unknown group, described by generators (permutations, matrices) or relators (presentations). Our description gives this structural understanding of $\text{Aut}(X)$ for free.

For interval graphs, a linear-time algorithm follows from the standard tools and techniques. The MPQ-tree M is computed in time $\mathcal{O}(n + m)$. We can compute $\text{Aut}(T)$ in a similar manner as the automorphism group of a rooted tree. Therefore, we get a recursive description in terms of group products, and we can describe their generators.

For circle graphs, our description easily leads to a polynomial-time algorithm, by computing the split tree and understanding its symmetries. The best algorithm for computing split-trees runs in almost linear time [13]. With a careful implementation and checking all details, one can likely match this time for computing $\text{Aut}(X)$ using our results.

6 Open Problems

We conclude this paper with several open problems concerning automorphism groups of other intersection-defined classes of graphs; for an overview see [15, 31].

We do not describe $\text{Aut}(\text{PERM})$. But our results and the inclusions $\text{CATERPILLAR} \subsetneq \text{PERM} \subsetneq \text{CIRCLE}$ imply that they are non-universal, between $\text{Aut}(\text{CATERPILLAR})$ and $\text{Aut}(\text{CIRCLE})$. We believe that our techniques can be applied.

► Problem 1. What is $\text{Aut}(\text{PERM})$?

Circular-arc graphs (CIRCULAR-ARC) are intersection graphs of circular arcs and they naturally generalize interval graphs. Surprisingly, this class is very complex and more different from interval graphs than it seems. The paper of Hsu [20] relates circular-arc graphs to circle graphs. It easily follows that $\text{Aut}(\text{CIRCULAR-ARC}) \supseteq \text{Aut}(\text{PSEUDOTREE})$.

► Problem 2. What is $\text{Aut}(\text{CIRCULAR-ARC})$? Is it equal to $\text{Aut}(\text{PSEUDOTREE})$?

Figure 2 depicts two infinite hierarchies of graph classes, one between INT and CHOR , and the other one between PERM and FUN . In both cases, the bottom graph class has non-universal automorphism groups and the top one has universal automorphism groups.

Let Y be any fixed graph. The class Y -GRAPH consists of all intersections graphs of connected subgraphs of a subdivision of Y . Observe that K_2 -GRAPH = INT and

$$\bigcup_{T \in \text{TREE}} T\text{-GRAPH} = \text{CHOR}.$$

The infinite hierarchy between INT and CHOR is formed by T -GRAPH for which $\text{INT} \subseteq T\text{-GRAPH} \subsetneq \text{CHOR}$. If Y contains a cycle, then Y -GRAPH is no longer contained in CHOR. The simplest of these classes are circular-arc graphs which are equal to K_3 -GRAPH.

► **Conjecture 1.** For every fixed graph Y , the class Y -GRAPH is non-universal.

The hierarchy between PERM and FUN is defined using the Dushnik-Miller dimension of partially ordered sets. Every poset is equal to the intersection of some linear orderings, and this dimension is the least number of these linear orderings. The complement of every function graph can be transitively oriented, and its dimension is the least dimension of all its transitive orientations. We denote the class of all function graphs of the dimension at most k by k -DIM. It follows that 1-DIM are all complete graphs, 2-DIM = PERM, and

$$\bigcup_{k \in \mathbb{N}} k\text{-DIM} = \text{FUN}.$$

We note that recognition of k -DIM is NP-complete for $k > 2$ [33].

► **Problem 3.** What are $\text{Aut}(k\text{-DIM})$? Are they non-universal for every $k \in \mathbb{N}$?

References

- 1 L. Babai. Automorphism groups of planar graphs II. In *Infinite and finite sets (Proc. Conf. Keszthely, Hungary)*, 1973.
- 2 K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and planarity using PQ-tree algorithms. *J. Comput. System Sci.*, 13:335–379, 1976.
- 3 N. Carter. *Visual group theory*. MAA, 2009.
- 4 C. J. Colbourn and K. S. Booth. Linear time automorphism algorithms for trees, interval graphs, and planar graphs. *SIAM J. Comput.*, 10(1):203–225, 1981.
- 5 Derek G Corneil, Hiryoung Kim, Sridhar Natarajan, Stephan Olariu, and Alan P Sprague. Simple linear time recognition of unit interval graphs. *Information Processing Letters*, 55(2):99–104, 1995.
- 6 W.H. Cunningham. Decomposition of directed graphs. *SIAM Journal on Algebraic Discrete Methods*, 3:214–228, 1982.
- 7 P. Erdős, S. Fajtlowicz, and A. J. Hoffman. Maximum degree in graphs of diameter 2. *Networks*, 10(1):87–90, 1980.
- 8 J. Fiala, P. Klavík, J. Kratochvíl, and R. Nedela. Algorithmic aspects of regular graphs covers with applications to planar graphs. In *Lecture Notes in Computer Science, Automata, Languages, and Programming ICALP 2014*, volume 8572, pages 489–501, 2014.
- 9 R. Frucht. Herstellung von graphen mit vorgegebener abstrakter gruppe. *Compositio Mathematica*, 6:239–250, 1939.
- 10 D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pac. J. Math.*, 15:835–855, 1965.
- 11 C. P. Gabor, K. J. Supowit, and W.-L. Hsu. Recognizing circle graphs in polynomial time. *Journal of the ACM (JACM)*, 36(3):435–473, 1989.
- 12 E. Gioan and C. Paul. Split decomposition and graph-labelled trees: Characterizations and fully dynamic algorithms for totally decomposable graphs. *Discrete Appl. Math.*, 160(6):708–733, 2012.

- 13 E. Gioan, C. Paul, M. Tedder, and D. Corneil. Practical and efficient circle graph recognition. *Algorithmica*, pages 1–30, 2013.
- 14 C. D. Godsil and G. Royle. *Algebraic graph theory*, volume 207. Springer New York, 2001.
- 15 M. C. Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57. Elsevier, 2004.
- 16 M. Grohe and D. Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing*, STOC '12, pages 173–192, 2012.
- 17 P. Hanlon. Counting interval graphs. *Transactions of the American Mathematical Society*, 272(2):383–426, 1982.
- 18 Z. Hedrlín, A. Pultr, et al. On full embeddings of categories of algebras. *Illinois Journal of Mathematics*, 10(3):392–406, 1966.
- 19 A. J. Hoffman and R. R. Singleton. On moore graphs with diameters 2 and 3. *IBM Journal of Research and Development*, 4(5):497–504, 1960.
- 20 W. L. Hsu. $\mathcal{O}(M \cdot N)$ algorithms for the recognition and isomorphism problems on circular-arc graphs. *SIAM Journal on Computing*, 24(3):411–439, 1995.
- 21 C. Jordan. Sur les assemblages de lignes. *Journal für die reine und angewandte Mathematik*, 70:185–190, 1869.
- 22 N. Korte and R. H. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM J. Comput.*, 18(1):68–81, 1989.
- 23 G. S. Lueker and K. S. Booth. A linear time algorithm for deciding interval graph isomorphism. *Journal of the ACM (JACM)*, 26(2):183–195, 1979.
- 24 E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42–65, 1982.
- 25 R. Mathon. A note on the graph isomorphism counting problem. *Information Processing Letters*, 8(3):131–136, 1979.
- 26 B. D. McKay, M. Miller, and J. Širáň. A note on large graphs of diameter two and given maximum degree. *J. Combin. Theory Ser. B*, 74(1):110–118, 1998.
- 27 M. Miller and J. Širáň. Moore graphs and beyond: A survey of the degree/diameter problem. *Electronic Journal of Combinatorics*, 61:1–63, 2005.
- 28 Fred S Roberts. Indifference graphs. *Proof techniques in graph theory*, 139:146, 1969.
- 29 J. J. Rotman. *An introduction to the theory of groups*, volume 148. Springer, 1995.
- 30 U. Schöningh. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988.
- 31 J. P. Spinrad. *Efficient Graph Representations.: The Fields Institute for Research in Mathematical Sciences.*, volume 19. American Mathematical Soc., 2003.
- 32 H. Whitney. Nonseparable and planar graphs. *Trans. Amer. Math. Soc.*, 34:339–362, 1932.
- 33 Mihalis Yannakakis. The complexity of the partial order dimension problem. *SIAM Journal on Algebraic Discrete Methods*, 3(3):351–358, 1982.
- 34 S. Zhou. A class of arc-transitive Cayley graphs as models for interconnection networks. *SIAM Journal on Discrete Mathematics*, 23(2):694–714, 2009.

Correlation Clustering and Two-edge-connected Augmentation for Planar Graphs*

Philip N. Klein^{†‡1}, Claire Mathieu^{‡2}, and Hang Zhou^{‡3}

- 1 Brown University, United States
klein@brown.edu
- 2 CNRS, École Normale Supérieure, France
cmathieu@di.ens.fr
- 3 École Normale Supérieure, France
hangzhou@di.ens.fr

Abstract

In *correlation clustering*, the input is a graph with edge-weights, where every edge is labelled either $+$ or $-$ according to similarity of its endpoints. The goal is to produce a partition of the vertices that disagrees with the edge labels as little as possible.

In *two-edge-connected augmentation*, the input is a graph with edge-weights and a subset R of edges of the graph. The goal is to produce a minimum weight subset S of edges of the graph, such that for every edge in R , its endpoints are two-edge-connected in $R \cup S$.

For *planar graphs*, we prove that correlation clustering reduces to two-edge-connected augmentation, and that both problems have a polynomial-time approximation scheme.

1998 ACM Subject Classification G.2.2 Graph Theory, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases correlation clustering, two-edge-connected augmentation, polynomial-time approximation scheme, planar graphs

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.554

1 Introduction

1.1 Correlation Clustering

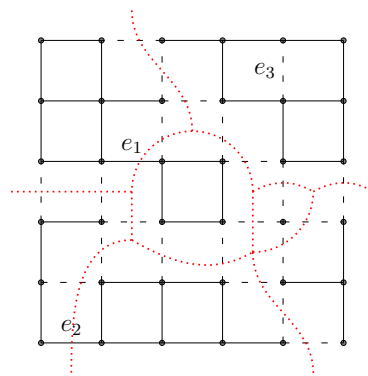
Correlation clustering takes as input a graph whose edges are labelled either $\langle + \rangle$ or $\langle - \rangle$. A $\langle + \rangle$ edge represents evidence that its endpoints belong in the same cluster, and a $\langle - \rangle$ edge represents evidence that its endpoints belong in different clusters. Each edge has a non-negative *weight* reflecting the strength of the evidence. The goal is to find a clustering minimizing the total weight of edges inconsistent with that evidence. This formulation, previously from computational biology [10], was introduced by Bansal, Blum, and Chawla [8]. They suggested as an application the clustering of documents into topics.

In this paper, we study the case when the graph is planar. The motivation comes from *image segmentation*. The goal is to partition the image into regions representing different image components. An image is represented by a grid of pixels. For each pair of neighboring pixels, comparing the pixels' values yields an assessment of how likely the pixels are to belong

* The full version of the paper is available on the authors' websites.

† Research funded by NSF Grants CCF-0964037 and CCF-1409520.

‡ Some of this research was done during a semester program at ICERM, the NSF-supported Institute for Computational and Experimental Research in Mathematics at Brown University.



■ **Figure 1** In this unweighted grid, every solid (resp. dashed) edge represents a pair of similar (resp. dissimilar) pixels. Dotted lines indicate an optimal partition with inconsistent edges e_1 , e_2 , e_3 .

to the same region. There can be spurious assessments. So global optimization is needed to find a good segmentation. See Figure 1. When an image is large, it is common for a visual task to first coalesce coherent uniform neighborhoods of pixels into superpixels, using preprocessing based on local properties such as brightness, color, and texture, see [3, 30]. We then extract a local similarity measure on pairs of adjacent superpixels, and the goal is to find a good segmentation of the superpixel graph under that measure. To achieve this, researchers used correlation clustering as the formulation [4, 5, 6, 26, 36]. They gave experimental results based on techniques such as integer linear programming or linear programming relaxation.

Note that the superpixel graph is planar. However, correlation clustering is NP-hard for planar graphs [7]. Prior to this work, the best result with theoretical guarantee was a constant-factor approximation for minor-excluded graphs by Demaine, Emanuel, Fiat, and Immorlica [17]. In this paper, we give a *polynomial-time approximation scheme* (PTAS).

► **Theorem 1.** *For any $\epsilon > 0$, there is a polynomial-time $(1 + \epsilon)$ -approximation algorithm for correlation clustering in weighted planar graphs.*

Related work

Why do we restrict ourselves to planar graphs? Because the general (weighted) problem is APX-hard [8]. Charikar, Guruswami, and Wirth [16] and independently Demaine, Emanuel, Fiat, and Immorlica [17] gave logarithmic-factor approximation algorithms. There have been improved approximation algorithms when the graph is complete [1, 8, 16]; or when, for each edge, the agreement weight and disagreement weight of that edge sum to one [1, 8]; or when, in addition, the weights satisfy the triangle inequality [22]. When the number of clusters is limited to a constant, Giotis and Guruswami [23] gave a PTAS. The problem was also studied in a planted model [31] and from the viewpoint of fixed-parameter tractability [15].

We discussed the problem of minimizing weight of disagreement; maximizing weight of agreement is equivalent at optimality but easier to approximate [8, 16, 35]. Researchers have also considered other objective functions [2].

1.2 Two-edge-connected Augmentation

In the field of telecommunications, an important task is to ensure that the network is resilient against single-link failures [34]. The *two-edge-connected augmentation* problem takes as input a graph G with non-negative edge-weights and a subset R of edges of the graph. The goal is

to find a minimum-weight subset S of edges of the graph such that for every edge $uv \in R$, u and v are two-edge-connected in the subgraph $R \cup S$. We give a PTAS for this problem when the graph is planar.

► **Theorem 2.** *For any $\epsilon > 0$, there is a polynomial-time $(1 + \epsilon)$ -approximation algorithm for two-edge-connected augmentation in weighted planar graphs.*

Related work

A closely related problem is *two-edge-connected spanning subgraph*, for which a constant-factor approximation algorithm was known [25]. When the graph is planar, Berger and Grigni [11] gave a PTAS. One might think that this would lead to a PTAS for our problem, but it is not the case because the weight of a two-edge-connected augmentation can be much smaller than the minimum weight of a two-edge-connected spanning subgraph. For the *Steiner-type generalization* of the two-edge-connected subgraph problem, there was a constant-factor approximation algorithms [28]. When the graph is planar, Borradaile and Klein [13] gave a PTAS.¹

There is a variety of other related work, see [29] for a survey. Some studied the special case when the weights are all one [20, 21, 32], or when, in addition, the graph is complete [19]. There was a 2-approximation algorithm for the related problem of augmenting a connected subgraph to achieve two-edge-connectivity among a pre-specified set of terminal vertices [33]. Edge-connectivity augmentation problems were subsumed by the work of Jain [24] on survivable network design.

2 Techniques and Notations

Our techniques for proving Theorem 1 and Theorem 2 include *planar duality*, *prize-collecting clustering*, *brick decomposition*, *sphere-cut decomposition*, and *dynamic programming*.

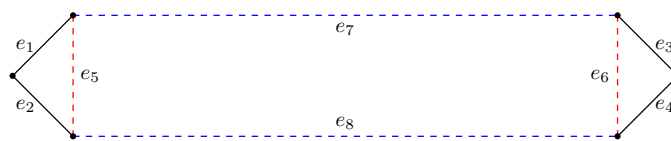
Throughout the paper, we allow graphs to have parallel edges. For a graph G , we note $V[G]$ as its vertex set and $E[G]$ as its edge set. For a subset $H \subseteq E[G]$, we identify H with the subgraph induced by edges from H . The weight of H is defined by $\sum_{e \in H} \text{weight}(e)$. The *boundary* $\partial(H)$ is the set of vertices u that are incident to some edge of H and to some edge of $E[G] \setminus H$. Similarly, for a subset $U \subseteq V[G]$, its *boundary* $\partial(U)$ is the set of edges uv such that $u \in U$ and $v \in V[G] \setminus U$. A *plane graph* is a planar graph together with a planar embedding. We use the phrases *plane graph* and *planar graph* interchangeably. We use $OPT(G, R)$ to denote the weight of the optimal two-edge-connected augmentation for (G, R) . The parameters G and R are omitted when they are clear from the context.

3 Theorem 2 Implies Theorem 1

We address correlation clustering and two-edge-connected augmentation in one paper because of the reduction in Theorem 3, which shows that Theorem 2 implies Theorem 1.

► **Theorem 3.** *There is an approximation-preserving reduction from correlation clustering in a weighted planar graph to two-edge-connected augmentation in a weighted planar graph.*

¹ In their problem, a solution is allowed to include multiple copies of edges of the input graph.



■ **Figure 2** In the example, $R = \{e_1, e_2, e_3, e_4\}$. The optimal two-edge-connected augmentation consists of edges e_5 and e_6 . However, any Steiner tree connecting the edges of R must include one of the edges e_7 and e_8 , whose weight may be much higher than weight $(\{e_5, e_6\})$.

► **Remark.** In practice, we may use an approximation algorithm for two-edge-connected augmentation that is different from the algorithm in Theorem 2, and then from the reduction (Theorem 3), we obtain an algorithm for planar correlation clustering with the same approximation factor.

► **Lemma 4 (Bridge-Deletion Lemma).** *Let G be a plane graph. Let R be a subset of $E[G]$. Let S be a minimal two-edge-connected augmentation for (G, R) . Then every connected component in the subgraph $R \cup S$ is two-edge-connected.*

Proof of Theorem 3. Given a correlation-clustering instance G_0 with $\langle - \rangle$ edges, construct an instance (G, R) of two-edge-connected augmentation as follows: To obtain the graph G , start with the planar dual of the G_0 , and add duplicates of the duals of the $\langle - \rangle$ edges. The weights are preserved. Define R to be the original (non-duplicate) duals of the $\langle - \rangle$ edges. Let S be a minimal two-edge-connected augmentation. By the Bridge-Deletion Lemma, every connected component in $R \cup S$ is two-edge-connected. Define the clusters of G_0 to be the connected components when edges dual to $R \cup S$ are removed. ◀

4 Reduction to Instance with a Connected Skeleton

Without loss of generality, we assume for the rest of the paper that the edges of R have weight zero.

To prove Theorem 2, we focus on a related version (Theorem 5), where we are given in addition a connected subgraph T that contains every edge of R . We defer the proof of Theorem 5 to later sections.

► **Theorem 5 (Augmentation Theorem).** *Let G be a plane graph with edge-weights. Let R be a subset of $E[G]$. Let T be a connected subgraph of G that contains every edge of R . For every $\epsilon > 0$, there is a polynomial-time algorithm $\text{AUGMENT-CONNECTED}(G, R, T, \epsilon)$ that computes a two-edge-connected augmentation S for (G, R) such that $\text{weight}(S) \leq (1 + \epsilon)\text{OPT}(G, R) + \epsilon^2 \cdot \text{weight}(T)$.*

In the rest of this section, we prove Theorem 2 using the Augmentation Theorem. One might consider connecting all edges of R with a Steiner tree T , and then applying the Augmentation Theorem. However, OPT could be much smaller than the minimum weight of a Steiner tree when the solution is not connected (see Figure 2). In that case, the upper bound given by the Augmentation Theorem would not imply an approximation scheme.

Fortunately, there is an algorithmic tool, called *prize-collecting clustering*, due to Bateni, Hajiaghayi, and Marx [9], that addresses exactly this kind of obstacle. They used it in addressing the Steiner forest problem. They started with a 2-approximate solution, and used prize-collecting clustering to decompose the instance into subinstances. We use the same approach for two-edge-connected augmentation, see Algorithm 1.

Algorithm 1 REDUCE-TO-CONNECTED**Input:** a weighted planar graph G and a subset R of edges, $\epsilon > 0$ **Output:** connected subgraphs T_1, \dots, T_k

- 1: $Y \leftarrow$ two-edge-connected augmentation with weight at most $2 \cdot OPT$
- 2: $(U_1, \dots, U_\ell) \leftarrow$ two-edge-connected components of $R \cup Y$
- 3: Contract each component U_i to build a new graph \hat{G}
- 4: For every $v \in \hat{G}$, let ϕ_v be ϵ^{-1} times the weight of the component corresponding to v
- 5: Do prize-collecting clustering on \hat{G} and ϕ , obtaining a forest F
- 6: Return the connected components T_1, \dots, T_k of the subgraph $F \cup R \cup Y$ of G

Algorithm 2 AUGMENT**Input:** a planar graph G , a subset of edges R , and $\epsilon > 0$ **Output:** two-edge-connected augmentation S for (G, R)

- 1: $(T_1, \dots, T_k) \leftarrow$ REDUCE-TO-CONNECTED($G, R, \epsilon/7$) ▷ Theorem 7
- 2: **for** $i \leftarrow 1$ to k **do**
- 3: $S_i \leftarrow$ AUGMENT-CONNECTED($G, R \cap T_i, T_i, \epsilon/7$) ▷ Theorem 5
- 4: **return** $(\bigcup_i S_i) \setminus R$

Line 1 computes a 2-approximate solution using Jain's algorithm [24] (which solves a much more general problem).

► **Lemma 6** (corollary from [24]). *There is an algorithm that computes in polynomial time a two-edge-connected augmentation Y for (G, R) such that $\text{weight}(Y) \leq 2 \cdot OPT$ and that every connected component in $R \cup Y$ is two-edge-connected.*

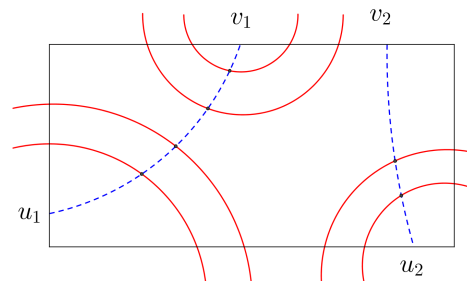
Line 5 uses prize-collecting clustering, which receives a graph with vertex-potentials ϕ_v and returns a forest F of edges of weight at most $2 \sum_v \phi_v$. Since the sum of vertex-potentials is at most $2\epsilon^{-1} \cdot OPT$, the weight of F is at most $4\epsilon^{-1} \cdot OPT$. Using essentially the same arguments as in [9], we obtain the following.

► **Theorem 7** (variant of Theorem 1.3 in [9]). *Let G be a plane graph with edge-weights. Let R be a subset of $E[G]$. For fixed ϵ , Algorithm 1 computes in polynomial time a set of connected subgraphs T_1, \dots, T_k with the following properties:*

- $\bigcup_i T_i$ contains every edge of R .
- $\sum_i \text{weight}(T_i) \leq (4/\epsilon + 2)OPT(G, R)$.
- $\sum_i OPT(G_i, R_i) \leq (1 + \epsilon)OPT(G, R \cap T_i)$

Proof of Theorem 2. The top-level algorithm of Theorem 2 is given in Algorithm 2. By the Augmentation Theorem (Theorem 5) and Property 1 of Theorem 7, the output is a two-edge-connected augmentation for (G, R) .

For each i , the weight of S_i is at most $(1 + \epsilon/7)OPT(G, R \cap T_i) + (\epsilon/7)^2 \cdot \text{weight}(T_i)$. Summing over i and combining Properties 2 and 3 of Theorem 7, we infer that the weight of the output solution is at most $(1 + \epsilon)OPT(G, R)$. ◀



■ **Figure 3** The rectangle is a brick. The solid curves are parts of a near-optimal solution. The dashed curves illustrate the u_1 -to- v_1 path and the u_2 -to- v_2 path inside the brick.

5 Techniques for Proving the Augmentation Theorem

5.1 New Use of Brick Decomposition

For non-local problems in weighted planar graphs in which the weight of the optimal solution can be much smaller than the weight of the graph, the *brick decomposition* technique of [14] has proved to be quite versatile: a planar embedded subgraph M (called the *mortar graph*) is selected, and the *bricks* are the subgraphs of G embedded in the faces of M (see Section 6.1). The key is the following properties of M .

Property 1: M has weight $O(OPT)$;

Property 2: There exists a near-optimal solution that crosses the boundary of each brick only a constant number of times.

Both properties are achievable for problems such as *Steiner tree* [14], *Steiner forest* [9], *TSP* [12], and *two-edge-connected survivability* [13] for the variant in which the solution is allowed to include multiple copies of edges of the input graph.

The main obstacle in applying this approach to *two-edge-connected augmentation* is that Property 2 seems unachievable using the known brick-decomposition construction. We therefore use the mortar graph in a new way. We take additional care in the construction of the mortar graph because of the edges of R . As a consequence, instead of Property 2, we can show that, after a transformation² of the instance, we have:

Property 2' (Structure Theorem): There exists a near-optimal solution such that, for any brick and any two vertices u, v on the boundary of the brick, there exists a u -to- v Jordan curve inside the brick that **intersects the near-optimal solution at only a constant number of points**.³ See Figure 3.

Property 2' is proved by *reducing nesting* and *adding boundary cycles*. See Section 6.

5.2 Outline of Algorithm AUGMENT-CONNECTED

We use ideas from [14] which we now summarize:

1. Build a mortar graph of G based on the connected skeleton T .
2. Do Breadth-First Search (BFS) on the dual of the mortar graph, and select a mod- k residue j such that edges whose levels are congruent to j have total weight at most $1/k$ times the weight of the mortar graph.

² The transformation is to add artificial copies of the brick boundaries. See Figure 4 in Section 6.

³ The constant depends on ϵ .

3. Commit to including these edges in the ultimate solution; this decomposes the graph into subinstances each consisting of at most k levels of bricks.
4. A planar graph consisting of only k BFS levels has branchwidth $2k$, i.e., can be recursively decomposed into clusters of edges such that each cluster is bounded by at most $2k$ vertices. However, here we must diverge. Note that the branch decomposition obtained above has a special form: it is a *sphere-cut decomposition*, which means that each cluster of edges is precisely the set of edges enclosed by a Jordan curve J that intersects no edges (and intersects a constant number of vertices) of the mortar graph. This is where Property 2' comes in: each segment of J traversing a brick can be replaced with a curve that intersects a constant number of points of the near-optimal solution. This yields a new Jordan curve J' that passes through a constant number of points of the near-optimal solution. Such structure enables us to design a dynamic program (DP), given in Section 7.

For each cluster of the sphere-cut decomposition, the DP enumerates all possibilities of the intersection points of the unknown near-optimal solution with the partially unknown Jordan curve J' . The DP also enumerates all possibilities of the connected structure of the part of the solution inside J' . See Section 7.2. Note that there may be some edges of the graph that are in the parent cluster but not in the child clusters (Figure 8), so the DP must do a bit of extra work to go from tables for the children to the table for the parent. See Section 7.3.

6 Structure Theorem

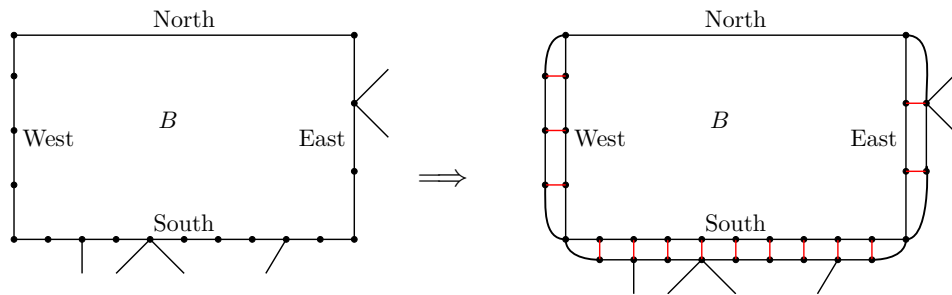
The Structure Theorem (Theorem 11) is the key to the polynomial-time performance of the dynamic program (Section 7). Before stating the theorem, we recall the definition and properties of *brick decomposition* from [14] in Section 6.1, and we illustrate the transformation of *doubling brick boundaries* in Section 6.2.

6.1 Mortar Graph and Brick Decomposition

► **Definition 8** (Mortar Graph and Bricks, slight adaptation from [14]). Let G be a plane graph with edge-weights. Let R be a subset of $E[G]$. Let M be a subgraph of G . For each face F of M , we define a *brick* B as the planar subgraph of G embedded inside the face, including the boundary edges of F . We denote the *interior* of B as the brick without the boundary edges of F . We call M a *mortar graph* of G if the boundary of every brick B , in counter-clockwise order, is the concatenation of four paths North_B , South_B , East_B , West_B (the subscript B is omitted when it is clear from the context), such that:

1. No edge of R is in the interior of B , or on South_B , East_B , or West_B .
2. South_B is a shortest path in B , and every proper subpath of North_B is an almost shortest path in B , i.e., its weight is at most $(1 + \epsilon)$ times the weight of the shortest path between its endpoints in B ;
3. There exists an integer $k = O(1/\epsilon^4)$ and vertices s_0, s_1, \dots, s_k ordered from west to east along South_B such that, for any vertex x on the segment $[s_i, s_{i+1})$ of South_B , the weight of the segment between x and s_i along South_B is less than ϵ times the weight of the shortest path between x and North_B in B .

► **Lemma 9** (Brick-Decomposition Lemma, slight adaptation from [14]). *Let G be a planar graph with edge-weights. Let R be a subset of $E[G]$. Let T be a connected subgraph of G that contains every edge of R . There is a polynomial-time algorithm that computes a mortar graph M of G such that:*



■ **Figure 4** Doubling the South, East, and West boundaries of the brick B . The new edges between vertices and their copies have weight 0.

1. $\text{weight}(M) = O(\text{weight}(T) / \epsilon)$;
2. $\sum_{\text{brick } B} \text{weight}(\text{East}_B \cup \text{West}_B) = O(\epsilon^2 \cdot \text{weight}(T))$.

6.2 Doubling Brick Boundaries

The proof of the Structure Theorem applies to a modified version of the graph in which artificial copies of the South, East, and West brick boundaries are added (Figure 4), and zero-weight edges are added between corresponding vertices. We call this *doubling* these boundaries. Note that no edges of R are duplicated (according to Property 1 of Definition 8). Let H be the resulting graph.

► **Lemma 10** (Boundary-Doubling Lemma). *A two-edge-connected augmentation for (G, R) can be transformed into a two-edge-connected augmentation for (H, R) in linear time without increasing the weight, and vice versa.*

As a consequence, it suffices to find a near-optimal solution for (H, R) .

6.3 Theorem Statement

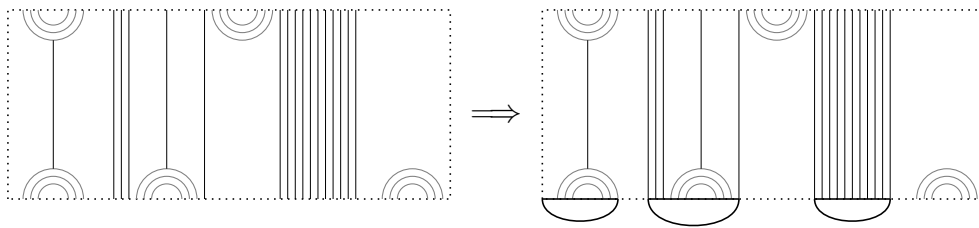
► **Theorem 11** (Structure Theorem). *Let G be a plane graph with edge-weights. Let R be a subset of $E[G]$. Let M be the mortar graph of G . Let H be the graph obtained from G by doubling the South, East, and West boundaries of every brick.*

For any two-edge-connected augmentation S_0 for (H, R) , there exists a two-edge-connected augmentation S for (H, R) such that:

- $\text{weight}(S) \leq (1 + \epsilon)\text{weight}(S_0) + 3 \sum_{\text{brick } B} \text{weight}(\text{East}_B \cup \text{West}_B)$;
- *For any brick and any two vertices u, v on the boundary of the brick, there exists a u -to- v Jordan curve inside the brick that has $O(1/\epsilon^4)$ crossings with S , all occurring at vertices.*

6.4 Proof Sketch

The proof of the Structure Theorem consists in modifying the initial solution so that any pair of vertices on the boundary of a brick can be connected by a curve that has few crossings with the modified solution. Figure 5 shows the kind of curve we use. It starts at a given vertex u on the brick boundary, traverses nested paths to reach the South boundary, then bypasses South-to-North paths using cycles formed by the duplicated edges of the South boundary, and finally again traverses nested paths to reach the given vertex v on the brick boundary. In order to have a small number of crossings, we must ensure that the number of



■ **Figure 7** Adding South cycles (the bold cycles in the right figure) into the solution. The number of South cycles needed is $O(1/\epsilon^4)$ due to Property 3 in the definition of bricks.

7.1 Sphere-Cut Decomposition

Our DP is based on a special kind of branch-decomposition of plane graphs, called a *sphere-cut decomposition* (see [18]): A *noose* of a plane graph is a Jordan curve that intersects only vertices of the graph and not edges. A *sphere-cut decomposition of width w* is a family of non-crossing nooses each intersecting at most w vertices; the nooses form a binary tree by the enclosure relation, each leaf noose encloses exactly one edge, and each edge is enclosed by a leaf noose. For each noose in the sphere-cut decomposition, we refer to the set of edges enclosed as a *cluster*.

► **Lemma 13** (trivial adaptation from [27]). *Let G be a plane graph whose dual graph has diameter k . Then G has a sphere-cut decomposition of width $2k$, and it can be computed in linear time.*

7.2 Specification of DP Table

In this section, we define the index of the DP table and the value at an index.

By Lemma 13, M has a sphere-cut decomposition \mathcal{SC} of width $O(1/\epsilon^3)$. The first index of the DP table is a cluster E of \mathcal{SC} .

Let S_0 be the optimal two-edge-connected augmentation for (H, R) , and let S be the solution obtained in the Structure Theorem (Theorem 11). By the Bridge-Deletion Lemma (Lemma 4), we can modify S so that every connected component in $R \cup S$ is two-edge-connected, without increasing the weight of S . For every cluster E of \mathcal{SC} , let J_E be the noose enclosing E and of minimum number of crossings with $R \cup S$ (all occurring at vertices), breaking ties by choosing the minimally enclosing one.⁴ It is easy to show that the family of nooses $\{J_E\}_{E \in \mathcal{SC}}$ is non-crossing.

► **Lemma 14.** *For every cluster E of \mathcal{SC} , J_E intersects $O(1/\epsilon^7)$ vertices of $R \cup S$.*

Proof. Since \mathcal{SC} has width $O(1/\epsilon^3)$, there is a noose enclosing E that has $O(1/\epsilon^3)$ intersections with M . From one intersection to the next, it goes across a single brick, and by the Structure Theorem (Theorem 11), the part inside this brick can be chosen so as to have $O(1/\epsilon^4)$ intersections with S . This results in a noose enclosing E that has $O(1/\epsilon^7)$ intersections with $R \cup S$. ◀

Let $Q^* \subseteq V[H]$ denote the (unknown) set of $O(1/\epsilon^7)$ intersection vertices of J_E with $S \cup R$. The second index of the DP table is a subset $Q \subseteq V[H]$ of size $O(1/\epsilon^7)$.

⁴ Since the noose is a geometric object, it is not uniquely defined, but a discrete formulation can be given using the *face-vertex incidence graph*.

Next, we encode the connectivity structure of the part of $R \cup S$ inside J_E . Let R_E (resp. Γ^*) denote the set of edges of R (resp. S) that are inside J_E . Define a forest F_0^* from $R_E \cup \Gamma^*$ by contracting every two-edge-connected component into a node. A node of F_0^* is called *internal* if its corresponding two-edge-connected component in $R_E \cup \Gamma^*$ does not contain any node from Q^* , i.e., the component is strictly inside J_E . We then define a forest F^* from F_0^* by splicing internal nodes of degree 2 and removing internal nodes that are singletons. By the construction, F^* has at most $|Q^*|$ non-internal nodes, and it does not contain internal nodes of degree 0, 1, or 2. So F^* has at most $2|Q^*| - 2$ nodes. The third index of the DP table is a forest F of at most $2|Q| - 2$ nodes. Moreover, there is a map ψ^* giving the natural many-to-one map from Q^* to nodes of F^* . The fourth index of the DP table is a map ψ from Q to $V[F]$. To summarize:

► **Definition 15** (DP index). An *index of the DP table*, also called a *DP index*, contains the following:

- E : a cluster of the sphere-cut decomposition \mathcal{SC}
- Q : a subset of $V[H]$ of size $O(1/\epsilon^7)$
- F : a forest of size at most $2|Q| - 2$
- ψ : a map from Q to $V[F]$, such that every node of degree 0, 1, or 2 in the forest F belongs to the image of ψ .

In addition, the triple (Q, F, ψ) as defined above is called a *partial DP index*.⁵

A set of edges Γ is *consistent* with a DP index (E, Q, F, ψ) if applying the previous construction to $R_E \cup \Gamma$ leads to the connectivity structure described by (F, ψ) . For every DP index (E, Q, F, ψ) , define its *value* $DP(E, Q, F, \psi)$ as the minimum weight among a collection of Γ 's, such that:

- *Correctness*: Every Γ in this collection is consistent with (E, Q, F, ψ) ;
- *Optimality*: If $(Q, F, \psi) = (Q^*, F^*, \psi^*)$, then Γ^* is in this collection.

In order to prove the Dynamic-Programming Theorem (Theorem 12), we only need to find a polynomial-time algorithm to fill in the DP table and to output the value $DP(M, \emptyset, \emptyset, \emptyset^0)$.⁶

7.3 Hole Region between Parent and Children

Let E be a cluster of \mathcal{SC} and let E_1 and E_2 be its child clusters. Let $Q^*, Q_1^*, Q_2^* \subseteq V[H]$ be the sets of intersections of $R \cup S$ with J_E, J_{E_1}, J_{E_2} . The *hole region* is the area inside J_E but outside J_{E_1} and J_{E_2} in the plane.⁷ See Figure 8. We remark that the hole region cannot contain edges from R .

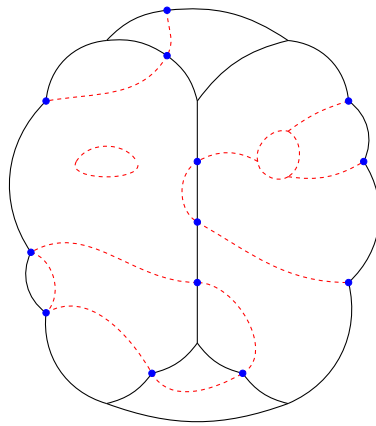
Let $\hat{\Gamma}^*$ denote the set of edges of S in the hole region. Let \hat{Q}^* denote the set of intersections of S with the boundary of the hole region. We have $\hat{Q}^* \subseteq Q^* \cup Q_1^* \cup Q_2^*$, thus $|\hat{Q}^*| = O(1/\epsilon^7)$. From $\hat{\Gamma}^*$ and \hat{Q}^* , we encode the connectivity structure of the part of S in the hole region as a forest \hat{F}^* of at most $2|\hat{Q}^*| - 2$ nodes and a map $\hat{\psi}^* : \hat{Q}^* \rightarrow V[\hat{F}^*]$. This is similar to the encoding in Section 7.2.

We use a side table T for the computation at hole regions. The table is indexed by a partial DP index $(\hat{Q}, \hat{F}, \hat{\psi})$. The *value* $T(\hat{Q}, \hat{F}, \hat{\psi})$ is defined as the minimum weight of any $\hat{\Gamma}$ that is consistent with $(\hat{Q}, \hat{F}, \hat{\psi})$ and contains no cycles.

⁵ Note that the description of Q, F, ψ is independent of E .

⁶ The DP outputs the *value* of a solution, not the solution itself; but it is easy to enrich the DP in the standard manner so that it also outputs the solution achieving the value.

⁷ Note that J_E, J_{E_1} , and J_{E_2} are non-crossing.



■ **Figure 8** J_E is the outermost boundary. It encloses 4 areas that are separated by the solid curves. J_{E_1} (resp. J_{E_2}) is the boundary of the left (resp. right) area. The *hole region* contains the top and bottom areas. The dashed paths represent $R \cup S$ inside J_E . The points represent vertices from $Q^* \cup Q_1^* \cup Q_2^*$.

7.4 Implementation of DP Table

First, the algorithm fills in the side table T during the preprocessing. Notice that any $\hat{\Gamma} \subseteq E[H]$ that is consistent with $(\hat{Q}, \hat{F}, \hat{\psi})$ and contains no cycles is such that, every node a in \hat{F} corresponds to a vertex u_a in the graph H , and every edge ab in \hat{F} corresponds to a path between u_a and u_b in $\hat{\Gamma}$. Therefore, to compute the value $T(\hat{Q}, \hat{F}, \hat{\psi})$, the algorithm enumerates, for every $a \in \hat{F}$, the vertex u_a among $V[H]$. For every $ab \in \hat{F}$, it then computes the shortest path between u_a and u_b in H . The union of all these shortest paths defines the current $\hat{\Gamma}$. The value $T(\hat{Q}, \hat{F}, \hat{\psi})$ is the minimum weight of all $\hat{\Gamma}$'s during the enumeration. The overall running time of the preprocessing is thus polynomial.

Next, the algorithm fills in the DP table in the order of the index E from bottom up in SC . Consider a DP index (E, Q, F, ψ) . Let E_1 and E_2 be the child clusters of E . The algorithm enumerates every combination of (E_1, Q_1, F_1, ψ_1) , (E_2, Q_2, F_2, ψ_2) , and $(\hat{Q}, \hat{F}, \hat{\psi})$ that are *compatible* with (E, Q, F, ψ) , and the current weight is the sum of the three entries. $DP(E, Q, F, \psi)$ is assigned with the minimum weight during the enumeration.

Acknowledgements We would like to thank Howard J. Karloff for helping make the connection between correlation clustering and two-edge-connected augmentation in planar graphs; Nabil Mustafa for numerous discussions; and Grigory Yaroslavtsev.

References

- 1 Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *Journal of the ACM*, 55(5), 2008.
- 2 Nir Ailon and Edo Liberty. Correlation clustering revisited: The true cost of error minimization problems. In *International Colloquium on Automata, Languages and Programming*, pages 24–36. Springer, 2009.
- 3 Sharon Alpert, Meirav Galun, Ronen Basri, and Achi Brandt. Image segmentation by probabilistic bottom-up aggregation and cue integration. In *Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

- 4 Amir Alush and Jacob Goldberger. Ensemble segmentation using efficient integer linear programming. *Pattern Analysis and Machine Intelligence*, 34(10):1966–1977, 2012.
- 5 Amir Alush and Jacob Goldberger. Break and conquer: Efficient correlation clustering for image segmentation. In *Similarity-Based Pattern Recognition*, volume 7953, pages 134–147. Springer, 2013.
- 6 Bjoern Andres, Jörg H. Kappes, Thorsten Beier, Ullrich Kothe, and Fred A. Hamprecht. Probabilistic image segmentation with closedness constraints. In *International Conference on Computer Vision*, pages 2611–2618. IEEE, 2011.
- 7 Yoram Bachrach, Pushmeet Kohli, Vladimir Kolmogorov, and Morteza Zadimoghaddam. Optimal coalition structure generation in cooperative graph games. In *Conference on Artificial Intelligence*, 2013.
- 8 Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.
- 9 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, and Dániel Marx. Approximation schemes for Steiner forest on planar graphs and graphs of bounded treewidth. *Journal of the ACM*, 58(5):21, 2011.
- 10 Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3/4):281–297, 1999.
- 11 A. Berger and M. Grigni. Minimum weight 2-edge-connected spanning subgraphs in planar graphs. In *International Colloquium on Automata, Languages and Programming*, volume 4596, pages 90–101, 2007.
- 12 Glencora Borradaile, Erik D. Demaine, and Siamak Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. *Algorithmica*, 68(2):287–311, 2014.
- 13 Glencora Borradaile and Philip N. Klein. The two-edge connectivity survivable network problem in planar graphs. *International Colloquium on Automata, Languages and Programming*, pages 485–501, 2008.
- 14 Glencora Borradaile, Philip N. Klein, and Claire Mathieu. An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. *ACM Transactions on Algorithms*, 5(3):31, 2009.
- 15 Sebastian Böcker and Jan Baumbach. Cluster editing. In Paola Bonizzoni, Vasco Brattka, and Benedikt Löwe, editors, *The Nature of Computation. Logic, Algorithms, Applications*, volume 7921, pages 33–44. Springer, 2013.
- 16 Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360–383, 2005.
- 17 Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immerlica. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2):172–187, 2006.
- 18 Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. *Algorithmica*, 58(3):790–810, 2010.
- 19 Kapali P. Eswaran and R. Endre Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, 1976.
- 20 Guy Even, Jon Feldman, Guy Kortsarz, and Zeev Nutov. A 1.8 approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. *ACM Transactions on Algorithms*, 5(2):21:1–21:17, 2009.
- 21 Guy Even, Guy Kortsarz, and Zeev Nutov. A 1.5-approximation algorithm for augmenting edge-connectivity of a graph from 1 to 2. *Information Processing Letters*, 111(6):296–300, 2011.
- 22 Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data*, 1(1), 2007.

- 23 Ioannis Giotis and Venkatesan Guruswami. Correlation clustering with a fixed number of clusters. In *Symposium on Discrete algorithm*, pages 1167–1176. ACM, 2006.
- 24 Kamal Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39–60, 2001.
- 25 Samir Khuller and Uzi Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM*, 41(2):214–235, 1994.
- 26 Sungwoong Kim, Sebastian Nowozin, Pushmeet Kohli, and Chang Dong Yoo. Higher-order correlation clustering for image segmentation. In *Advances in Neural Information Processing Systems*, pages 1530–1538, 2011.
- 27 Philip N. Klein and Shay Mozes. Optimization algorithms for planar graphs. In preparation, manuscript at <http://planarity.org>.
- 28 Philip N. Klein and R. Ravi. When cycles collapse: A general approximation technique for constrained two-connectivity problems. In *Integer Programming and Combinatorial Optimization*, pages 39–55, 1993.
- 29 Guy Kortsarz and Zeev Nutov. Approximating minimum cost connectivity problems. *Approximation Algorithms and Metaheuristics*, 2007.
- 30 David R. Martin, Charless C. Fowlkes, and Jitendra Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *Pattern Analysis and Machine Intelligence*, 26(5):530–549, 2004.
- 31 Claire Mathieu and Warren Schudy. Correlation clustering with noisy input. In *Symposium on Discrete Algorithms*, pages 712–728, 2010.
- 32 Hiroshi Nagamochi. An approximation for finding a smallest 2-edge-connected subgraph containing a specified spanning tree. *Discrete Applied Mathematics*, 126(1):83 – 113, 2003.
- 33 R Ravi. Approximation algorithms for Steiner augmentations for two-connectivity. Technical Report CS-92-21, Department of Computer Science, Brown University, 1992.
- 34 Mauricio Resende and Panos Pardalos. *Handbook of optimization in telecommunications*. Springer, 2008.
- 35 Chaitanya Swamy. Correlation clustering: maximizing agreements via semidefinite programming. In *Symposium on Discrete Algorithms*, pages 526–527. SIAM, 2004.
- 36 Julian Yarkony, Alexander Ihler, and Charless C Fowlkes. Fast planar correlation clustering for image segmentation. In *European Conference on Computer Vision*, volume 7577, pages 568–581. Springer, 2012.

Extended Formulation Lower Bounds via Hypergraph Coloring?*

Stavros G. Kolliopoulos¹ and Yannis Moysoglou²

1 Department of Informatics and Telecommunications, National and Kapodistrian University of Athens
Panepistimiopolis Ilissia, Athens 157 84, Greece
sgk@di.uoa.gr

2 Department of Informatics and Telecommunications, National and Kapodistrian University of Athens
Panepistimiopolis Ilissia, Athens 157 84, Greece
gmoys@di.uoa.gr

Abstract

Exploring the power of linear programming for combinatorial optimization problems has been recently receiving renewed attention after a series of breakthrough impossibility results. From an algorithmic perspective, the related questions concern whether there are compact formulations even for problems that are known to admit polynomial-time algorithms.

We propose a framework for proving lower bounds on the size of extended formulations. We do so by introducing a specific type of extended relaxations that we call *product relaxations* and is motivated by the study of the Sherali-Adams (SA) hierarchy. Then we show that for every approximate extended formulation of a polytope P , there is a product relaxation that has the same size and is at least as strong. We provide a methodology for proving lower bounds on the size of approximate product relaxations by lower bounding the chromatic number of an underlying hypergraph, whose vertices correspond to gap-inducing vectors.

We extend the definition of product relaxations and our methodology to mixed integer sets. However in this case we are able to show that *mixed product relaxations* are at least as powerful as a special family of extended formulations. As an application of our method we show an exponential lower bound on the size of approximate mixed product relaxations for the metric capacitated facility location problem (CFL), a problem which seems to be intractable for linear programming as far as constant-gap compact formulations are concerned. Our lower bound implies an unbounded integrality gap for CFL at $\Theta(N)$ levels of the *universal* SA hierarchy which is *independent of the starting relaxation*; we only require that the starting relaxation has size $2^{o(N)}$, where N is the number of facilities in the instance. This proof yields the first such tradeoff for an SA procedure that is independent of the initial relaxation.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.1.6 Optimization

Keywords and phrases linear programming, extended formulations, inapproximability, facility location

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.568

* This research has been co-financed by the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program “Education and Lifelong Learning” of the National Strategic Reference Framework (NSRF) - Research Funding Program: “Thalis. Investing in knowledge society through the European Social Fund”.

1 Introduction

In the past few years there has been an increasing interest in exposing the limitations of compact LP formulations for combinatorial optimization problems. The goal is to show a lower bound on the size of *extended formulations (EFs)* for a particular problem. Extended formulations add extra variables to the natural problem space; the increase in dimension may yield a smaller number of facets. The minimum size over all extended formulations is the *extension complexity* of the corresponding polytope. A superpolynomial lower bound on the extension complexity is of intrinsic interest in polyhedral combinatorics and implies that there is no polynomial-time algorithm relying purely on the solution of a compact linear program. It does not however rule out efficient LP-based algorithms that combine algorithmic steps of arbitrary type, such as preprocessing, primal-dual, etc., with linear programming.

In the seminal paper of Yannakakis [21] the problem of lower bounding the size of extended formulations was considered for the first time: exponential lower bounds were proved for symmetric extended formulations of the matching and TSP polytopes. Yannakakis [21] identified also a crucial combinatorial parameter, the nonnegative rank of the slack matrix of the underlying polytope P , and he showed that it equals the extension complexity of P . A strong connection of the extension complexity of a polytope to communication complexity was made in [21], by showing that the nonnegative rank of the slack matrix is at least the size of its minimum rectangle cover. That connection has been exploited in several results on the extension complexity of polytopes.

Fiorini et al. [13] lifted the symmetry condition on the result of [21] regarding the TSP polytope, thus answering a long-standing open problem. The result was obtained by showing that the correlation polytope has exponential extension complexity which in turn was shown using communication complexity tools. Recently, Rothvoß [19] removed the symmetry condition for the matching polytope as well, answering the second long-standing open question of [21]. This was done by a breakthrough in bounding a refined version of the rectangle covering number.

A more general question is that of the size of approximate extended formulations. This problem was first considered in [7] where the methodology of [13] was extended to approximate formulations and an exponential bound for the linear encoding of the $n^{1/2-\epsilon}$ -approximate clique problem was given. Subsequently, Braverman and Moitra [10] extended the former bound to $n^{1-\epsilon}$ -approximate formulations of the clique, following a new, information theoretic, approach. Braun and Pokutta in [8] further strengthened the lower bounds by introducing the notion of common information. Very recently, Braun and Pokutta [9] extended the result of [19] to approximate formulations of the matching polytope by combining ideas of the latter with the notion of common information.

In [11] it was proved that in terms of approximating maximum constraint satisfaction problems, LPs of size $O(n^k)$ are exactly as powerful as $O(k)$ -level relaxations in the Sherali-Adams hierarchy. Their proof differs from previous work in showing that polynomials of low degree can approximate the functional version of the factorization theorem of [21].

The *metric capacitated facility location* problem (CFL) is a well-studied problem for which, while constant-factor approximations are known [6, 2], no efficient LP relaxation with constant integrality gap is known. An instance I of CFL is defined as follows. We are given a set F of facilities and a set C of clients, with each facility i having a capacity U_i and each client j having a demand $d_j > 0$. We may open facility i by paying its opening cost f_i and we may assign demand from client j to facility i by paying the connection cost c_{ij} per unit of demand. The latter costs satisfy the following variant of the triangle inequality:

$c_{ij} \leq c_{ij'} + c_{i'j} + c_{i'j}$ for any $i, i' \in F$ and $j, j' \in C$. We are asked to open a subset $F' \subseteq F$ of the facilities and assign all the client demand to the open facilities while respecting the capacities. The goal is to minimize the total opening and connection cost. The question whether an efficient relaxation exists for CFL is among the most important open problems in approximation algorithms [20]. In a recent breakthrough, the first $O(1)$ -factor LP-based algorithm for CFL was given in [4]. The proposed relaxation is, however, exponential in size and, according to the authors of [4], not known to be separable in polynomial time. To our knowledge, there has not been a compact EF for approximate CFL achieving even an $o(|F|)$ gap.

In previous work [17, 18], we proved among other results for CFL, unbounded integrality gaps for the Sherali-Adams hierarchy when starting from the natural LP relaxation. We also disqualified, with respect to obtaining a $O(1)$ gap, valid inequalities from the literature such as the flow-cover inequalities and their generalizations.

1.1 Our contribution

In this paper we propose a new approach for proving lower bounds on the size of approximate extended formulations. Our contribution is summarized by the following.

First we introduce a family of extended relaxations of a given polytope which we call *product relaxations*. The product relaxations are inspired by the study of the Sherali-Adams hierarchy. Given a polytope $K \subseteq [0, 1]^d$ that corresponds to a linear relaxation of the problem at hand, the Sherali-Adams relaxation $SA^t(K)$ at level t is produced by a lift-and-project method, where initially every constraint in the description of K is multiplied with all t -subsets of variables and their complements. The resulting products of variables are then linearized, i.e., each replaced by a single variable, and finally one projects back to the original variable space of K . The variable space of the product relaxations is exactly the space of the final d -level Sherali-Adams relaxation, after linearization and before projection. The variables have the intuitive meaning of corresponding to products over sets of variables from the original space – the "intuitive meaning" of a variable is made precise through the notion of the *section* f of an extended relaxation $Q(x, y)$ of a polytope $P(x)$. This is a function f that maps an integer point $x \in P(x)$ to a vector of values $(x, y) = f(x)$ such that $f(x) \in Q(x, y)$. (See Section 2 for the necessary definitions).

We prove in Theorem 3 that for any ρ -approximate extended formulation of a 0-1 polytope there is a product relaxation of the same size that is at least as strong. The proof is short and accessible. Theorem 3 reduces lower bounding the size of an extended formulation, which uses some unknown space and encoding, of a polytope P , to lower bounding the size of product relaxations of P . In the product space we have the concrete advantage of knowing the section of the target relaxation. We extend the definition of product relaxations and our methodology to mixed integer sets. However in this case we are able to show that *mixed product relaxations* are at least as powerful as a special family of extended formulations (cf. Theorem 5).

We note that our approach does not rely on the notion of the slack matrix introduced by Yannakakis [21]. It differs from that of [11] in which the slack functions of the factorization theorem [21] were shown to be approximable, for max CSPs, by low-degree polynomials and thus SA gaps are transferred to general linear programs.

Then we use a methodology for proving lower bounds for relaxations for which the section is known and in particular for product relaxations. Similar arguments have been used in the context of bounding the number of facets of specific polyhedra, but prior to our work, they seemed inapplicable for lower bounding the size of arbitrary EFs which lift the polytope

in arbitrary variable spaces. The method is the following: first define a set of vectors in the space of the relaxation such that for each one of those vectors there is an admissible objective function witnessing an integrality gap of ρ . We call that set of vectors the *core*. Then show that, for any partition of the core into fewer than κ parts, there must be some part containing a set of conflicting vectors. A set of infeasible vectors is *conflicting* if its convex hull has nonempty intersection with the convex hull of $\{f(x) \mid x \in P(x) \cap \{0, 1\}^n\}$, which is always included in the feasible region of a product relaxation – here $f(x)$ is the section we associate with product relaxations. Thus, we get that at least κ inequalities are needed to separate the members of the core from the feasible region and so κ is a lower bound on the size of any ρ -approximate product relaxation. By considering the hypergraph whose set of vertices corresponds to the aforementioned set of vectors and whose set of hyperedges corresponds to the sets of conflicting vectors, the chromatic number of the hypergraph is a lower bound on the size of every ρ -approximate extended formulation (cf. Theorem 8). Moreover, there is always a core such that the chromatic number of the resulting, possibly infinite, hypergraph equals the extension complexity of the polytope at hand. Thus the characterization of extension complexity in Theorem 8 can be seen as an alternative to the nonnegative rank of the slack matrix. The conflicting vectors are fractional solutions, which are hard to separate from the integer solutions. The method comes closer to standard LP/SDP integrality gap arguments than the existing combinatorial approaches for lower bounding extension complexity.

When arguing about the polyhedral complexity of a specific polytope, i.e., the minimum size of its formulation in the original variable space, the above method can always be simplified to finding a set of gap-inducing vectors with the property that (almost) any pair of them are conflicting. The underlying hypergraph reduces then to a simple graph that is very dense, almost a clique, and thus has high chromatic number. We used this idea in a preliminary version of this work [16] to derive exponential bounds on the polyhedral complexity of approximate metric capacitated facility location, where only the classic variables are used (cf. Corollary 14). A similar idea was independently used by Kaibel and Weltge in [15] to derive lower bounds on the number of facets of a polyhedron which contains a given integer set X and whose set of integer points is $\text{conv}(X) \cap \mathbb{Z}^d$.

We exhibit a concrete application of our methodology by proving in Theorem 12 an exponential lower bound on the size of any $O(N)$ -approximate mixed product relaxation for the CFL polytope, where N is the number of facilities in the instance. This result can be shown to imply (cf. Theorem 13) that the $\Omega(N)$ -level SA relaxation for CFL, which is obtained from any starting LP of size $2^{o(N)}$ defined on the classic set of variables, has unbounded gap $\Omega(N)$. Note, that it is well-known that by lifting only the facility variables, at N levels the integer polytope is obtained for CFL [5]. This settles the open question of [3] whether there are LP relaxations upon which the application of lift-and-project methods captures the strength of preprocessing steps for CFL. Our result establishes for the first time such a tradeoff for a *universal* SA procedure that is independent of the starting relaxation K . The proof follows the methodology outlined above and is different from the standard arguments that apply only to the SA lifting of a specific LP. Our earlier SA construction in [18] applied the local-global method [12] that constructs an appropriate distribution of solutions for each explicit constraint of the starting LP.

We leave as an open problem the extension of the equivalence between product and extended relaxations from 0-1 programs to mixed integer sets. We also believe that it would be of interest to revisit known extension-complexity lower bounds using our method, so as to obtain simpler proofs.

2 Preliminaries

$X \subseteq \mathbb{R}^d$, is a *mixed integer set* if there is $p \in \{1, \dots, d-1\}$ such that $d = n + p$ and $X \subseteq \{0, 1\}^n \times [0, 1]^p$. A *valid relaxation* of the mixed integer set X is any polyhedron P such that $\text{conv}(X) \subseteq P$. Given a valid relaxation P of X , such that $\text{conv}(X) \cap (\{0, 1\}^n \times [0, 1]^p) = P \cap (\{0, 1\}^n \times [0, 1]^p)$, the *level k Sherali-Adams (SA) procedure*, $k \geq 1$, is as follows [1]. Let P be defined by the linear constraints $Ax - b \leq 0$. For every constraint $\pi(x) \leq 0$ of P , for every set of variables $U \subseteq \{x_i \mid i = 1, \dots, n\}$ such that $|U| \leq k$, and for every $W \subseteq U$, consider the *lifted* valid constraint: $\pi(x) \prod_{x_i \in U-W} x_i \prod_{x_i \in W} (1 - x_i) \leq 0$. Linearize the system obtained this way by replacing (i) x_i^2 with x_i for all i (ii) $\prod_{x_i \in I \subseteq [n]} x_i$ with x_I and (iii) $x_k \prod_{x_i \in I \subseteq [n]} x_i$, where $k \in \{n+1, \dots, d\}$ with v_{Ik} . $\text{SA}^k(P)$ is the projection of the resulting linear system onto the original variables $\{x_1, \dots, x_d\}$. We call $\text{SA}^k(P)$ the relaxation *obtained from P at level k* of the SA hierarchy. It is well-known that $\text{SA}^n(P) = \text{conv}(X)$ (see, e.g., [5]). If X is a 0-1 set, i.e., $X \subseteq \{0, 1\}^d$, the above definitions hold mutatis mutandis and $\text{SA}^d(P) = \text{conv}(X)$.

Given a polyhedron $K(x, y) = \{(x, y) \in \mathbb{R}^d \times \mathbb{R}^{d_y} \mid Ax + By \leq b\}$ the *projection to the x -space* is defined as $\{x \in \mathbb{R}^d \mid \exists y \in \mathbb{R}^{d_y} : Ax + By \leq b\}$, denoted as $\text{proj}_x(K(x, y))$. An *extended formulation (relaxation)* of a polyhedron $P(x) \subseteq \mathbb{R}^d$ is a linear system $K(x, y) = \{(x, y) \in \mathbb{R}^d \times \mathbb{R}^{d_y} \mid Ax + By \leq b\}$ such that $\text{proj}_x(K(x, y)) = P(x)$ ($\text{proj}_x(K(x, y)) \supseteq P(x)$). The *size* of a polyhedron $P(x)$ is the minimum number of inequalities in its halfspace description. The *extension complexity* of $P(x)$ is the minimum size of an extended formulation of $P(x)$.

We define now ρ -approximate formulations as in [7]. Given a combinatorial optimization problem T , a *linear encoding* of T is a pair (L, O) where $L \subseteq \{0, 1\}^*$ is the set of *feasible solutions* to the problem and $O \subset \mathbb{R}^*$ is the set of *admissible objective functions*. An instance of the linear encoding is a pair (d, w) where d is a positive integer defining the dimension of the instance and $w \subseteq O \cap \mathbb{R}^d$ is the set of admissible cost functions for instances of dimension d . Solving the instance (d, w) means finding $x \in L \cap \{0, 1\}^d$ such that $w^T x$ is either maximum or minimum, according to the type of problem T . Let $P = \text{conv}(\{x \in \{0, 1\}^d \mid x \in L\})$ be the corresponding 0-1 polytope of dimension d . Given a linear encoding (L, O) of a maximization problem, the corresponding polytope P , and $\rho \geq 1$, a ρ -*approximate extended formulation* of P is an extended relaxation $Ax + By \leq b$ of P with $x \in \mathbb{R}^d, y \in \mathbb{R}^{d_y}$ such that

$$\begin{aligned} \max\{w^T x \mid Ax + By \leq b\} &\geq \max\{w^T x \mid x \in P\} && \text{for all } w \in \mathbb{R}^d \text{ and} \\ \max\{w^T x \mid Ax + By \leq b\} &\leq \rho \max\{w^T x \mid x \in P\} && \text{for all } w \in O \cap \mathbb{R}^d. \end{aligned}$$

For a minimization problem, we require

$$\begin{aligned} \min\{w^T x \mid Ax + By \leq b\} &\leq \min\{w^T x \mid x \in P\} && \text{for all } w \in \mathbb{R}^d \text{ and} \\ \min\{w^T x \mid Ax + By \leq b\} &\geq \rho^{-1} \min\{w^T x \mid x \in P\} && \text{for all } w \in O \cap \mathbb{R}^d. \end{aligned}$$

The ρ -*approximate extension complexity* of 0-1 integer polytope $P(x) \subseteq [0, 1]^d$ is the minimum size of a ρ -approximate extended formulation of P . Given an extended formulation $Q(x, y)$ of $P(x)$, a *section* of Q is defined as a vector-valued boolean function $g(x): \{0, 1\}^d \rightarrow \mathbb{R}^{d+d_y}$ such that for $x \in P(x) \cap \{0, 1\}^d$, $g(x)$ belongs to $Q(x, y)$ and $\text{proj}_x(g(x)) = x$. Intuitively, the section extends the encoding of solutions to the auxiliary variables y . Clearly, if a particular extended formulation Q has been specified a priori, different such functions can be defined by filling in the last d_y coordinates of $g(x_o)$ with a value from $\{y \in \mathbb{R}^{d_y} \mid Ax_o + By \leq b\}$.

► **Definition 1.** Given a 0-1 integer polytope $P(x) \subseteq [0, 1]^d$, a *product relaxation* $D(z)$ of $P(x)$ is an extended relaxation $D(z)$ of $P(x)$, where $z \in \mathbb{R}^{2^d-1}$ and for every nonempty

subset $\mathcal{E} \subseteq \{x_1, x_2, \dots, x_d\}$ of the original variables, we have a variable $z_{\mathcal{E}}$, (where $z_{\{x_i\}}$ denotes x_i , $i = 1, \dots, d$), and there is a section $f(x)$ of D s.t. the corresponding coordinate of f at \mathcal{E} is $f_{\mathcal{E}}(x) = \prod_{x_i \in \mathcal{E}} x_i$. We refer to this function f as *the product section*.

Let f denote the product section. Define the *canonical product relaxation* of P as $\hat{D} = \text{conv}\{f(x) \mid x \in P(x) \cap \{0, 1\}^{d_x}\}$. The polytope \hat{D} corresponds to the “tightest” possible product relaxation.

For a mixed integer set $M(x, w) \subseteq \{0, 1\}^{d_x} \times \mathbb{R}^{d_w}$ the corresponding mixed integer polytope $P(x, w)$ is $\text{conv}(M(x, w))$. In case one starts from a mixed integer polytope, the additional z variables of the product relaxation correspond to sets that contain at most one fractional variable. Including only one fractional variable in each product, mimics the variable space of the final-level SA relaxation.

► **Definition 2.** Let $P(x, w) \subseteq [0, 1]^{d_x} \times \mathbb{R}^{d_w}$ be a mixed integer polytope. A *mixed product relaxation* $D(z)$ of $P(x, w)$ is an extended relaxation $D(z)$ of $P(x, w)$, where $z \in \mathbb{R}^{(d_w+1)2^{d_x}-1}$, with $z_{\{w_j\}} = w_j$, $j = 1, \dots, d_w$, and

(i) for every set $\emptyset \neq \mathcal{E} \subseteq \{x_1, x_2, \dots, x_{d_x}\}$ we define $d_w + 1$ variables: one that we denote $z_{\mathcal{E}}$ and, for each fractional variable w_j , $j = 1, \dots, d_w$, one that we denote $z_{\mathcal{E}w_j}$. Moreover $z_{\{x_i\}}$ denotes x_i , $i = 1, \dots, d_x$.

(ii) there is a section $f(x, w)$ of D s.t. the corresponding coordinates of f are $f_{\mathcal{E}}(x, w) = (\prod_{x_i \in \mathcal{E}} x_i)$ and, for each variable w_j , $j = 1, \dots, d_w$, $f_{\mathcal{E}w_j}(x, w) = (\prod_{x_i \in \mathcal{E}} x_i) \cdot w_j$. We refer to this function f as *the mixed product section*.

The *canonical product relaxation* of $P(x, w)$ is similarly defined as $\hat{D} = \text{conv}\{f(x, w) \mid (x, w) \in P(x, w) \cap (\{0, 1\}^{d_x} \times \mathbb{R}^{d_w})\}$.

Note that the lifted polytope produced by the d -level (d_x -level) Sherali-Adams procedure applied on some specific linear relaxation of the 0-1 polytope $P(x)$ (mixed integer $P(x, w)$), after linearization and before projection to the original variables, is a (mixed) product relaxation.

3 The expressive power of product relaxations

In this section we show the following. For every 0-1 polytope $P(x)$ and every (approximate) extended formulation $Q(x, y) = \{(x, y) \in \mathbb{R}^{d_x} \times \mathbb{R}^{d_y} \mid Ax + By \leq b\}$ of $P(x)$ there is a product relaxation $T[Q(x, y)]$ whose size is at most that of $Q(x, y)$ and is at least as strong.

A *substitution* T is a linear map of the form $y = Tz$ where T is a $d_y \times (2^{d_x} - 1)$ matrix and z is a $2^{d_x} - 1$ dimensional vector having a coordinate $z_{\mathcal{E}}$ for each nonempty set \mathcal{E} of the form $\{x_i \mid i \in S \subseteq \{1, \dots, d_x\}\}$. For any substitution T , the *translation* of $Q(x, y)$, denoted $T[Q(x, y)]$, the formulation resulting by substituting $T_{(i)}z$, for y_i , $i = 1, \dots, d_y$. Here $T_{(i)}$ denotes the i th row of T . If in addition $T[Q(x, y)]$ is a product relaxation of $P(x)$ we say that it is a *translation of Q to product relaxations* (recall that the original variables x_i coincide with the variables $z_{\{x_i\}}$). Observe that the number of inequalities of $T[Q(x, y)]$ is the same as in $Q(x, y)$. The translation may heighten exponentially the dimension, but since our methodology will give lower bounds on the size of the product relaxations those bounds apply to the size of $Q(x, y)$ as well.

► **Theorem 3.** Given a 0-1 polytope $P(x) \subseteq [0, 1]^{d_x}$, for every polytope $Q(x, y)$ such that $P(x) \subseteq \text{proj}_x(Q(x, y))$ there is a translation $T[Q(x, y)]$ to product relaxations such that $P(x) \subseteq \text{proj}_x(T[Q(x, y)]) \subseteq \text{proj}_x(Q(x, y))$.

Proof. We shall give a substitution T for the variables $y \in \mathbb{R}^{d_y}$ of $Q(x, y)$ so that the theorem holds. Let $g(x)$ be a section of $Q(x, y)$ (recall that a section associates every feasible 0-1 vector x of $P(x)$ to a specific y such that $(x, y) \in Q(x, y)$).

Observe that the coordinates of the product section plus the constant 1 correspond exactly to the monomials of the Fourier basis. We denote by $(p, 1) \in \mathbb{R}^{n+1}$ the vector resulting from $p \in \mathbb{R}^n$ by appending the scalar 1 as an extra coordinate. By basic functional analysis (see, e.g., [14]), there is a $d_y \times 2^{d_x}$ matrix A such that

$$g(x) = A \cdot (f(x), 1) \tag{1}$$

We define the substitution T by linearizing the above equation; we replace the sections g and f with the corresponding variable vectors y and z (recall z is the product vector) to obtain:

$$y = A \cdot (z, 1).$$

Obviously $\text{proj}_x(T[Q(x, y)]) \subseteq \text{proj}_x(Q(x, y))$: from any feasible solution (x_0, z_0) of $T[Q(x, y)]$ we can derive a feasible solution (x_0, y_0) of $Q(x, y)$ by setting y_0 equal to Az_0 .

We will now show that $P(x) \subseteq \text{proj}_x(T[Q(x, y)])$. It suffices to show that for every $x' \in P(x) \cap \{0, 1\}^{d_x}$ the vector $f(x')$ is feasible for $T[Q(x, y)]$ as required by the definition of product relaxations. Observe that by letting the z vector take the values $f(x')$, by (1) we get that the quantities involved in the inequalities of $T[Q(x, y)]$ are the exact same quantities involved in the corresponding inequalities of $Q(x, y)$ for $g(x')$. But by the definition of section, $g(x')$ is feasible for $Q(x, y)$ and thus $f(x')$ is feasible for $T[Q(x, y)]$. ◀

► **Corollary 4.** *A lower bound b on the size of any product relaxation D which is a ρ -approximate extended formulation of the 0-1 polytope $P(x)$, for $\rho \geq 1$, implies a lower bound b on the size of any ρ -approximate extended formulation $Q(x, y)$ of $P(x)$.*

Let $P(x, w)$ be a mixed integer polytope. The notion of the section of P for some extended relaxation $Q(x, w, y)$ of P is more challenging. Intuitively, the solutions are characterized by two parts – a boolean part of the 0 – 1 assignments on the integer variables x and a "linear" part of the real variables w in the following sense: once the boolean part (the "hard" one) is fixed, the linear part can be obtained as the feasible region of a (usually small) system of inequalities, possibly empty.

Motivated by the above we define the following type of sections for an extended formulation $Q(x, w, y)$ of a mixed-integer polytope. A *mixed-linear section* of EF Q is a section g for which at variable y_i the value $g_i(x', w)$ for a given integer vector x' is an affine function on w denoted $g_i^{x'}(w)$. If there is such a mixed-linear section for $Q(x, w, z)$, we say that Q is an *extended formulation with a mixed linear section*. An example of EFs with a mixed linear section are formulations arising from the SA procedure where y is the vector of the new variables corresponding to the linearized products. The following theorem can be proved similarly to Theorem 3.

► **Theorem 5.** *Given a mixed integer polytope $P(x, w) \subseteq [0, 1]^{d_x} \times \mathbb{R}^{d_w}$, for every ρ -approximate, $\rho \geq 1$, extended formulation $Q(x, w, y)$ with a mixed linear section, there is a translation $T[Q(x, w, y)]$ to mixed product relaxations such that*

$$P(x, w) \subseteq \text{proj}_{x,w}(T[Q(x, w, y)]) \subseteq \text{proj}_{x,w}(Q(x, w, y)).$$

Proof. Let the dimension of $P(x, w)$ be $d = d_x + d_w$. We shall give for the variables $y \in \mathbb{R}^{d_y}$ of $Q(x, w, y)$ a substitution T so that the theorem holds.

Consider a variable y_i and the corresponding coordinate of the mixed linear section, $g_i^{x'}(w) = \sum_j b_j^{x'} w_j + c_{x'}$ for each $x' \in \{0, 1\}^{d_x}$ and $i = 1, \dots, d_y$.

First, we will prove a helpful claim which states a fact from elementary Fourier analysis in our setting. For $x, s \in \text{proj}_x (P(x, w) \cap (\{0, 1\}^{d_x} \times \mathbb{R}^{d_w}))$, define the boolean indicator operator $\chi_s(x)$ to be 1 when $s = x$ and 0 otherwise. First, we will show that this operator can be expressed as a linear combination of the product sections constrained to monomials with only boolean variables. In other words, we determine coefficients $a_{\mathcal{E}}^s$, $\mathcal{E} \subseteq \{x_1, \dots, x_{d_x}\}$, such that $\chi_s(x) = \sum_{\mathcal{E}} a_{\mathcal{E}}^s f_{\mathcal{E}}(x)$. The translation of the indicator operator $i_s(x)$ of an integer solution s is a linear expression of the form $T_{i_s} = \sum_{\mathcal{E}} a_{\mathcal{E}}^s P[\mathcal{E}](x)$. We shall iteratively generate the coefficients $a_{\mathcal{E}}^s$. The only nonzero coefficients will be those corresponding to sets of variables that are supersets of the set of variables being 1 in s – let that set be \mathcal{E}_1^s . We give the construction iteratively starting from $|\mathcal{E}_1^s|$ to d_x , defining in step k the coefficients of such sets of size k .

In the first iteration simply set $a_{\mathcal{E}_1^s} = 1$. At step $k > |\mathcal{E}_1^s|$, for each set \mathcal{E}' of size k that is a superset of \mathcal{E}_1^s , set $a_{\mathcal{E}'}^s = -\sum_{\mathcal{E} \subset \mathcal{E}'} a_{\mathcal{E}}^s$. This concludes the definition of the coefficients.

► **Claim 3.1.** For each integer solution $s' \in \text{proj}_x (P(x, w) \cap (\{0, 1\}^{d_x} \times \mathbb{R}^{d_w}))$, $\chi_s(s') = \sum_{\mathcal{E}} a_{\mathcal{E}}^s f_{\mathcal{E}}(s')$.

Proof of the claim. By overloading the notation, we denote by s both the integer solution and the support of that integer solution, that is the set $\{x_i \mid s_i = 1\}$. If $s' \supseteq s$ then the nonzero terms of the sum $\sum_{\mathcal{E}} a_{\mathcal{E}}^s f_{\mathcal{E}}(s')$ are exactly those that correspond to sets \mathcal{E} such that $s \subseteq \mathcal{E} \subseteq s'$. We have that $\sum_{\mathcal{E}} a_{\mathcal{E}}^s f_{\mathcal{E}}(s') = \sum_{\mathcal{E} \subseteq s'} a_{\mathcal{E}}^s$ which, by the construction of the coefficients, is 1 if $s = s'$ and 0 if $s' \supset s$, as required. Otherwise, if $s - s' \neq \emptyset$, then all the $f_{\mathcal{E}}(s')$ with nonzero coefficients are 0, so $\sum_{\mathcal{E}} a_{\mathcal{E}}^s f_{\mathcal{E}}(s') = 0$.

By Claim 3.1 we have that for an integer vector $s \in \{0, 1\}^{d_x}$ the indicator operator $\chi_s(x)$ is equal to $\sum_{\mathcal{E} \subseteq \{x_1, \dots, x_{d_x}\}} a_{\mathcal{E}}^s f_{\mathcal{E}}(x)$. For each set of integer variables \mathcal{E} and each fractional variable w_j let $z_{\mathcal{E}w_j}$ denote the corresponding mixed product variable and $f_{\mathcal{E}w_j}(x, w)$ the corresponding coordinate of the mixed product section. It is now easy to show the following.

► **Claim 3.2.** For each mixed integer solution (x', w') , and for $i = 1, \dots, d_y$, $g_i^{x'}(w') = \sum_j \sum_{\mathcal{E}} a_{\mathcal{E}}^s b_j^{x'} f_{\mathcal{E}w_j}(x', w') + \sum_{\mathcal{E}} a_{\mathcal{E}}^s c_{x'} f_{\mathcal{E}}(x')$.

To conclude the definition of T , set

$$y^i = \sum_{x'} \sum_j \sum_{\mathcal{E}} a_{\mathcal{E}}^s b_j^{x'} z_{\mathcal{E}w_j} + \sum_{x'} \sum_{\mathcal{E}} a_{\mathcal{E}}^s c_{x'} z_{\mathcal{E}}, \quad i = 1, \dots, d_y.$$

which implies

$$y^i = \sum_{x'} \sum_{\mathcal{E}} a_{\mathcal{E}}^s \left(\sum_j b_j^{x'} z_{\mathcal{E}w_j} + c_{x'} z_{\mathcal{E}} \right), \quad i = 1, \dots, d_y$$

By Claim 3.2, using arguments similar to the ones in the proof of Theorem 3, it follows that $P(x, w) \subseteq \text{proj}_{x,w} (T[Q(x, w, y)]) \subseteq \text{proj}_{x,w} (Q(x, w, y))$. ◀

► **Corollary 6.** A lower bound b on the size of any mixed product relaxation D which is a ρ -approximate extended formulation of the 0-1 mixed integer polytope $P(x, w)$ implies a lower bound b on the size of any ρ -approximate extended formulation $Q(x, w, y)$ of $P(x, w)$ with a mixed linear section.

4 A method for lower bounding the size of LPs with known sections

Here we present a methodology to lower bound the size of relaxations that achieve a desired integrality gap. For simplicity we do not deal in this section with mixed integer sets.

Our method can be summarized as follows. Let $G(z) \subseteq [0, 1]^d$ be a 0-1 polytope. We design a family \mathcal{I} of instances parameterized by the dimension d . For each instance $I \in \mathcal{I}$ of dimension d we define a set of points $\mathcal{C}_I \subseteq [0, 1]^d \setminus G(z)$ which we call the *core of I with respect to G* . Note that the points of the core must be infeasible for G . To prove a lower bound $r(n)$ on the size of G it suffices to show that at least that many inequalities are needed to separate \mathcal{C}_I from G . Additionally, for a minimization problem with O being the set of admissible objective functions, if for some $z \in \mathcal{C}_I$ there is an admissible cost function w_z such that $w_z^T z < \rho^{-1} \text{Opt}_{I, w_z}$, $0 < \rho \leq 1$, where Opt_{I, w_z} is the cost of the optimal integer solution with respect to w_z , we call z ρ -gap inducing wrt O . If we design the core so that all its members are ρ -gap inducing, the lower bound will hold for ρ -approximate formulations.

To define constructively the core for a specific family of extended formulations of a polytope P the sections of the variables z must be known. This requirement is fulfilled by the product relaxations we will focus on. By Theorem 3 above, proving a lower bound on the size for an arbitrary extended relaxation $Q(x, y)$ of a polytope $P(x)$ can be reduced to a proof of the same bound on the size of a corresponding product relaxation $D(z)$. The following meta-theorem shows that such a proof can always be obtained by proving the existence of a suitable core for the product relaxation. Recall the definition of the “tightest” product relaxation of $P(x)$, \hat{D} , in Section 2. We say that a set of vectors $s \subseteq [0, 1]^d \setminus \hat{D}$ is *conflicting* if $\text{conv}(s) \cap \hat{D} \neq \emptyset$. Any single valid inequality of \hat{D} cannot separate all points of a conflicting set. Given a set $O_d \subseteq \mathbb{R}^d$ of admissible objective functions associated with a 0-1 polytope $P(x) \subseteq [0, 1]^d$, we define $\hat{O}_d \subseteq \mathbb{R}^{2d-1}$, to contain the vectors in O_d extended with zeroes in the coordinates corresponding to the non-singleton product variables.

► **Theorem 7.** *Given a 0-1 polytope $P(x) \subseteq [0, 1]^d$, and an associated set of admissible objective functions $O_d \subseteq \mathbb{R}^d$, the ρ -approximate extension complexity, $\rho \geq 1$, of $P(x)$ is at least $r(n)$, iff there exists a family of instances $\mathcal{I}(n)$ and, for every $I \in \mathcal{I}$, a core \mathcal{C}_I wrt \hat{D} , which consists of ρ -gap inducing vectors wrt \hat{O}_d , with the following property: for any partition of \mathcal{C}_I into less than $r(n)$ parts there must be a part containing a set of conflicting vectors.*

Proof. Assume first that the ρ -approximate extension complexity is at least $r(n)$. Define \mathcal{C}_I to be the set of all ρ -gap inducing product vectors. If we can partition \mathcal{C}_I into less than $r(n)$ parts so that there is no conflicting subset s in any part, then we can define an inequality for each part of the partition that separates the vectors of at least that part from \hat{D} . But we know that less than $r(n)$ inequalities cannot separate all the ρ -gap inducing product vectors. Thus we have that for any decomposition of those vectors into less than $r(n)$ parts there must be a part containing a set of conflicting vectors.

Conversely, assume we can find a core \mathcal{C}_I wrt \hat{D} consisting of ρ -gap inducing vectors such that for any partition of \mathcal{C}_I into less than $r(n)$ sets there must be a part containing a set of conflicting vectors. Then the size of \hat{D} is at least $r(n)$. If not, there is a decomposition into less than $r(n)$ parts where each part consists of the core members separated by each inequality – in case a member is separated by more than one inequality, we arbitrarily include it into just one of the resulting parts. Observe that \mathcal{C}_I is not only a core wrt \hat{D} but also is a core wrt any ρ -approximate product relaxation of P . By Theorem 3, the lower bound $r(n)$ applies to the size of any ρ -approximate extended formulation of P . ◀

Let $\mathcal{H}(\mathcal{C}_I)$ be the, possibly infinite, hypergraph with vertices the members of \mathcal{C}_I and

hyperedges the conflicting subsets of \mathcal{C}_I . Theorem 7 can be restated more conveniently:

► **Theorem 8.** *Given a 0-1 polytope $P(x) \subseteq [0, 1]^d$, and an associated set of admissible objective functions $O_d \subseteq \mathbb{R}^d$, the ρ -approximate extension complexity, $\rho \geq 1$, of $P(x)$ is at least $r(n)$, iff there exists a family of instances $\mathcal{I}(n)$ and, for every $I \in \mathcal{I}$, a core \mathcal{C}_I wrt \hat{D} , which consists of ρ -gap inducing vectors wrt \tilde{O}_d , such that $\mathcal{H}(\mathcal{C}_I)$ has chromatic number $r(n)$.*

Theorem 7 suggests that the best possible lower bound on the extension complexity can always be achieved by proving the existence of an appropriate core in the product space. In the applications in this paper we implement a version of the method that imposes stronger requirements on the decomposition, namely the constructed hypergraph will be a clique.

5 Lower bounds for approximate mixed product relaxations for CFL

For CFL, the linear encoding $\mathcal{N}_{\text{CFL}} = (L, O)$ is defined as follows. For a CFL instance, given the number n of facilities, the number m of clients, the capacities $K \in \mathbb{R}_+^n$ and the demands $D \in \mathbb{R}_+^m$, we use the classic variables $y_i, i = 1, \dots, n, x_{ij}, i = 1, \dots, n, j = 1, \dots, m$ with the usual meaning of facility opening and client assignment respectively. The set of feasible solutions (y, x) is defined in the obvious manner. Thus for dimension $d = n + nm, L \cap \{0, 1\}^d$ is completely determined by the quadruple (n, m, K, D) . The set of admissible objective functions $O \cap \mathbb{R}^{n+nm}$ is the set of pairs (\mathbf{f}, \mathbf{c}) where $\mathbf{f} \in \mathbb{R}_+^n$ are the facility opening costs and $\mathbf{c} = [c_{ij}] \in \mathbb{R}_+^{nm}$ are connection costs that satisfy $c_{ij} \leq c_{i'j} + c_{ij'} + c_{ij'}$.

The capacitated facility location problem with general capacities and demands is a mixed integer optimization problem where the facilities are opened integrally but the clients are allowed to be assigned fractionally to the set of opened facilities. In this section, we show an exponential lower bound on the size of any mixed product relaxation of the CFL polytope.

In our proof we will consider a parameterized instance $I = I(3n, m, U, d)$ with uniform capacities U and uniform unit demands $d = 1$, where $3n$ is the number of facilities, and m the number of clients. Furthermore we will have that the number of clients is $m = n^4 + 1$ and the capacities and demands are such that $(n^4 + 1) - nU = 2^{-n^2}$. Observe that $n^3 < U < (n^3 + 1)$. In order to define the core \mathcal{C}_I of the instance I we first describe a random experiment based on whose outcome we will later define the members of the core. Given disjoint sets $k, l \subseteq F$ of size n each, the random experiment defines a distribution $\mathcal{D}_{k,l}$ over mixed integer vectors in the classic encoding. These vectors correspond in general to pseudo-solutions. The following experiment defines the distribution $\mathcal{D}_{k,l}$. The quantities \bar{x}_{ij} are defined in Lemma 9 below.

RANDOM EXPERIMENT

Facilities in k are always opened.

Case 1. With probability $1 - \frac{20}{n^2(1+1/n)}$ all facilities in $F - l$ are opened and those of l are closed. Distribute evenly the client demand to facilities in k . Note that this outcome of the experiment does not respect the capacities.

Case 2. Otherwise, i.e., with probability $\frac{20}{n^2(1+1/n)}$, pick at random a subset q of the facilities in $F - k$ with at least one facility from l and open them. Assign randomly demand to each facility i in $q \cap l$ so that i takes $\frac{\sum_j \bar{x}_{ij}}{10/n^2}$ units and the rest of the demand is equally distributed to the facilities in k .

► **Lemma 9.** *The expected vector (\bar{y}, \bar{x}) wrt $\mathcal{D}_{k,l}$ is the following: $\bar{y}_i = 1$ for $i \in k, \bar{y}_i = 1 - \frac{10}{n^2(1+1/n)}$ for $i \in F - k - l, \bar{y}_i = \frac{20(2^{n-1})}{n^2(1+1/n)(2^n-1)}$ for $i \in l$. For all $j \in C, \bar{x}_{ij} = \frac{1-n^{-2}}{|k|}$ for $i \in k, \bar{x}_{ij} = 0$ for $i \in F - \{k \cup l\}, \bar{x}_{ij} = \frac{n^{-2}}{|l|}$ for $i \in l$.*

The distribution $\mathcal{D}_{k,l}$ will be subsequently used to define the members of the core \mathcal{C}_I . Let \mathcal{E} be a subset of integer variables in the original space, i.e., $\mathcal{E} \subseteq \{y_1, \dots, y_{3n}\}$. We denote by $E_{\mathcal{D}_{k,l}}[\mathcal{E}]$ the expectation of the event where all the variables in \mathcal{E} have value 1, i.e., the expectation of the product $\prod_{y_{i_k} \in \mathcal{E}} y_{i_k}$. Similarly, we denote $E_{\mathcal{D}_{k,l}}[\mathcal{E}x_{ij}]$ the expectation of the product $(\prod_{y_{i_k} \in \mathcal{E}} y_{i_k}) \cdot x_{ij}$. Let $\chi(\text{case1}), \chi(\text{case2})$ be the 0-1 random variables that indicate whether Case 1 and Case 2 occur, respectively. We denote by $E_{\mathcal{D}_{k,l}}[\mathcal{E} \cap \text{case1}]$ the expectation of the product $(\prod_{y_{i_k} \in \mathcal{E}} y_{i_k}) \cdot \chi(\text{case1})$ and by $E_{\mathcal{D}_{k,l}}[\mathcal{E}x_{ij} \cap \text{case1}]$ the expectation of the product $(\prod_{y_{i_k} \in \mathcal{E}} y_{i_k}) \cdot x_{ij} \cdot \chi(\text{case1})$. Similarly for Case 2. Intuitively, $E_{\mathcal{D}_{k,l}}[\mathcal{E}x_{ij} \cap \text{case1}]$ is the "mass" that $\mathcal{D}_{k,l}$ assigns to x_{ij} over all outcomes of case 1 where the variables of \mathcal{E} have value 1.

To simplify notation, we use $z(i)$ instead of z_i to refer to a coordinate of vector z indexed by i . From now on, P denotes the CFL polytope and \hat{D} its canonical product relaxation.

► **Definition 10.** Fix a set $k \subset F$ of size n . The core \mathcal{C}_I of the instance $I(3n, n^4 + 1, U, 1)$ wrt \hat{D} is the following set of product vectors: $\forall l \subset F$ with $|l| = n$ and $k \cap l = \emptyset$ and for every set \mathcal{E} of integer variables and for every fractional variable x_{ij} we define $z_{k,l}(\mathcal{E}) = E_{\mathcal{D}_{k,l}}[\mathcal{E}]$ and $z_{k,l}(\mathcal{E}x_{ij}) = E_{\mathcal{D}_{k,l}}[\mathcal{E}x_{ij}]$.

Now we are ready to state the key Lemma 11 from which our main theorem will be derived. The proof of the lemma is quite technical and is deferred to the full version. A sketch is presented in Section 6.

► **Lemma 11.** For any two $z_{k,l}, z_{k,l'} \in \mathcal{C}_I$ such that $l - l' \neq \emptyset$ there is some $z \in \text{conv}(z_{k,l}, z_{k,l'})$ which is feasible for \hat{D} .

► **Theorem 12.** Given the family of CFL instances $I(3n, n^4 + 1, U, 1)$, each member of \mathcal{C}_I is $\Omega(n)$ -gap inducing and $\chi(\mathcal{H}(\mathcal{C}_I)) = 2^{\Omega(n)}$. Therefore, there is a constant $c > 0$, s.t. any cN -approximate EF for CFL with a mixed linear section has size $2^{\Omega(N)}$, where N is the number of facilities.

Proof. Since we proved in Lemma 11 that any two members of the core \mathcal{C}_I form a conflicting set, $\mathcal{H}(\mathcal{C}_I)$ is a clique and thus its chromatic number is $|\mathcal{C}_I| = \binom{2n}{n} = 2^{\Theta(n)}$. For each member of the core $z_{k,l}$ there is an admissible cost function $w_{k,l}$ inducing $\Theta(n)$ gap: facilities in l have unit opening costs and every other facility has 0 opening cost. The facilities in $k \cup l$ and all the clients are co-located, and the rest of the facilities are co-located at distance 2^{n^2} from the former. Observe that each feasible mixed integer solution has a cost of at least 1 since either some facility in l must be opened integrally or at least 2^{-n^2} client demand has to be assigned to some facility in $F - k - l$. On the other hand the cost of $z_{k,l}$ wrt $w_{k,l}$ is $\Theta(n^{-1})$ since the (y, x) projection of $z_{k,l}$ is the expected vector (\bar{y}, \bar{x}) of $\mathcal{D}_{k,l}$. ◀

For every instance I of CFL it is easy to see that there is an exact mixed product relaxation of size $2^N p$ where p is polynomial in the size of the instance. The idea is to define a formulation for each choice of the opened facilities and then take the convex hull of those polytopes.

► **Observation 5.1.** There is an exact mixed product relaxation of the CFL polytope of size $2^N p$, where $p = \Theta(mN)$, N and m being the number of facilities and clients respectively.

► **Theorem 13.** Let P be any linear relaxation of the CFL polytope for the family of instances $I(3n, n^4 + 1, U, 1)$ that uses the encoding \mathcal{N}_{CFL} and has size $2^{o(n)}$. There is a constant $c > 0$, such that for all $t \leq cn$, the integrality gap of $\text{SA}^t(P)$ is $\Omega(n)$.

Proof. Observe that for every level of SA there is a suitable projection of \mathcal{C}_I that yields a legal core with respect to the product variables used in that level. Therefore, the lower bound on the size implied by Theorem 12 holds at all levels. The number of the inequalities of the t -level SA relaxation after the lifting and linearization stages, and before projection, obtained from any starting relaxation P of size r is less than $r \binom{n}{t} 2^t$. By choosing $t \leq cn$, with c sufficiently small, we obtain that $r \binom{n}{t} 2^t \leq r 2^{\delta n}$ for a small $\delta > 0$. By Theorem 12 we get that for this value of t , the integrality gap on the given family of instances is $\Omega(n)$. This is asymptotically tight since SA is known to produce an exact formulation after $3n$ levels \blacktriangleleft

We obtain as a direct consequence a lower bound on the size of formulations that use only the classic variables y_i, x_{ij} .

► **Corollary 14.** *Let P be any linear relaxation of the CFL polytope that uses the encoding \mathcal{N}_{CFL} and has integrality gap $o(N)$, N being the number of facilities. Then P has size $2^{\Omega(N)}$.*

6 Proof sketch for Lemma 11

In the first part of the proof we will show that by exchanging some measure of some components of the two product vectors $z_{k,l}, z_{k,l'}$ of the core, we can construct two new product vectors $z_{k,l}^*, z_{k,l'}^*$ each of which is feasible for \hat{D} . Consider the two sets of facilities $l - l'$ and $l' - l$. Clearly $|l - l'| = |l' - l| > 0$, since $l \neq l'$ and $|l| = |l'| = n$. We construct a product vector $z_{k,l}^*$ based on $z_{k,l}$ and making some alterations and, symmetrically, a product vector $z_{k,l'}^*$ based on $z_{k,l'}$. We give below the construction of $z_{k,l}^*$.

Construction of $z_{k,l}^*$

For any set \mathcal{E} containing only facilities from $F - l'$ with at least one from $l - l'$: $z_{k,l}^*(\mathcal{E}) = z_{k,l}(\mathcal{E}) + E_{\mathcal{D}_{k,l'}}[\mathcal{E} \cap \text{case1}]$ (Similarly, for any i, j , $z_{k,l}^*(\mathcal{E}x_{ij}) = z_{k,l}(\mathcal{E}x_{ij}) + E_{\mathcal{D}_{k,l'}}[\mathcal{E}x_{ij} \cap \text{case1}]$). In the case set \mathcal{E} contains only facilities from $F - l$ with at least one from $l' - l$ we have $z_{k,l}^*(\mathcal{E}) = z_{k,l}(\mathcal{E}) - E_{\mathcal{D}_{k,l}}[\mathcal{E} \cap \text{case1}]$. (Similarly, for any i, j , $z_{k,l}^*(\mathcal{E}x_{ij}) = z_{k,l}(\mathcal{E}x_{ij}) - E_{\mathcal{D}_{k,l}}[\mathcal{E}x_{ij} \cap \text{case1}]$). In any other case and for any i, j let $z_{k,l}^*(\mathcal{E}) = z_{k,l}(\mathcal{E})$ and $z_{k,l}^*(\mathcal{E}x_{ij}) = z_{k,l}(\mathcal{E}x_{ij})$.

Next we show, and this is by far the most complicated part of the proof, that the constructed $z_{k,l}^*$ and $z_{k,l'}^*$ are indeed the expected vectors of distributions $\mathcal{D}_{k,l}^*$ and $\mathcal{D}_{k,l'}^*$, respectively, over feasible mixed integer product solutions. In the last step of the proof we show the following, which is an easy consequence of the construction of $z_{k,l}^*$ and $z_{k,l'}^*$.

► **Claim 6.1.** $1/2(z_{k,l}^* + z_{k,l'}^*) \in \text{conv}(z_{k,l}, z_{k,l'})$.

7 Discussion

In the proof of our result for CFL we provided a core whose underlying hypergraph is actually a graph and moreover a clique. For other problems, especially for 0-1 polytopes, we believe that the power of general hypergraphs needs to be exploited, if one wishes to derive a tight bound on the extension complexity. Observe that our methodology requires only the existence of a suitable core, and thus, one could possibly employ probabilistic arguments to prove the existence of suitable hypergraphs of high chromatic number.

In the case of mixed integer polytopes, we believe that the mixed product relaxations can be shown to be strong enough to simulate any extended formulation, as is the case for product relaxations and 0-1 polytopes.

Acknowledgements We thank the anonymous reviewers of an earlier version for valuable comments.

References

- 1 Warren P. Adams and Hanif D. Sherali. Linearization strategies for a class of zero-one mixed integer programming problems. *Oper. Res.*, 38(2):217–226, April 1990.
- 2 Ankit Aggarwal, Anand Louis, Manisha Bansal, Naveen Garg, Neelima Gupta, Shubham Gupta, and Surabhi Jain. A 3-approximation algorithm for the facility location problem with uniform capacities. *Math. Program.*, 141(1-2):527–547, 2013.
- 3 Hyung-Chan An, Aditya Bhaskara, and Ola Svensson. Centrality of trees for capacitated k-center. *CoRR*, abs/1304.2983, 2013.
- 4 Hyung-Chan An, Mohit Singh, and Ola Svensson. LP-based algorithms for capacitated facility location. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 256–265. IEEE Computer Society, 2014.
- 5 Egon Balas, Sebastián Ceria, and Gérard Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Math. Program.*, 58(3):295–324, February 1993.
- 6 Manisha Bansal, Naveen Garg, and Neelima Gupta. A 5-approximation for capacitated facility location. In Leah Epstein and Paolo Ferragina, editors, *Algorithms – ESA 2012*, volume 7501 of *Lecture Notes in Computer Science*, pages 133–144. Springer Berlin Heidelberg, 2012.
- 7 Gábor Braun, Samuel Fiorini, Sebastian Pokutta, and David Steurer. Approximation limits of linear programs (beyond hierarchies). In *FOCS*, pages 480–489, 2012.
- 8 Gábor Braun and Sebastian Pokutta. Common information and unique disjointness. In *FOCS*, pages 688–697. IEEE Computer Society, 2013.
- 9 Gábor Braun and Sebastian Pokutta. The matching polytope does not admit fully-polynomial size relaxation schemes. *CoRR*, abs/1403.6710, 2014.
- 10 Mark Braverman and Ankur Moitra. An information complexity approach to extended formulations. In *Proc. STOC*, pages 161–170, 2013.
- 11 Siu On Chan, James R. Lee, Prasad Raghavendra, and David Steurer. Approximate constraint satisfaction requires large LP relaxations. In *Proc. FOCS*, pages 350–359, 2013.
- 12 Wenceslas Fernandez de la Vega and Claire Kenyon-Mathieu. Linear programming relaxations of maxcut. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 53–61, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.
- 13 Samuel Fiorini, Serge Massar, Sebastian Pokutta, Hans Raj Tiwary, and Ronald de Wolf. Linear vs. semidefinite extended formulations: exponential separation and strong lower bounds. In *Proceedings of the 44th Symposium on Theory of Computing*, STOC '12, pages 95–106, New York, NY, USA, 2012. ACM.
- 14 Stasys Jukna. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012.
- 15 Volker Kaibel and Stefan Weltge. Lower bounds on the sizes of integer programs without additional variables. In Jon Lee and Jens Vygen, editors, *IPCO*, volume 8494 of *Lecture Notes in Computer Science*, pages 321–332. Springer, 2014.
- 16 Stavros G. Kolliopoulos and Yannis Moysoglou. Exponential lower bounds on the size of approximate formulations in the natural encoding for capacitated facility location. *CoRR*, abs/1312.1819, 2013.
- 17 Stavros G. Kolliopoulos and Yannis Moysoglou. Tight bounds on the Lovász-Schrijver rank for approximate capacitated facility location. *Manuscript*, 2013.
- 18 Stavros G. Kolliopoulos and Yannis Moysoglou. Sherali-Adams gaps, flow-cover inequalities and generalized configurations for capacity-constrained facility location. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, pages 297–312, 2014.

- 19 Thomas Rothvoß. The matching polytope has exponential extension complexity. In *Proc. STOC*, pages 263–272, 2014.
- 20 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, New York, NY, USA, 1st edition, 2011.
- 21 Mihalis Yannakakis. Expressing combinatorial optimization problems by linear programs. *Journal of Computer and System Sciences*, 43(3):441 – 466, 1991.

Lempel-Ziv Factorization May Be Harder Than Computing All Runs

Dmitry Kosolobov

Ural Federal University
Ekaterinburg, Russia
dkosolobov@mail.ru

Abstract

The complexity of computing the Lempel-Ziv decomposition and the set of all runs (= maximal repetitions) is studied in the decision tree model of computation over ordered alphabet. It is known that both these problems can be solved by RAM algorithms in $O(n \log \sigma)$ time, where n is the length of the input string and σ is the number of distinct letters in it. We prove an $\Omega(n \log \sigma)$ lower bound on the number of comparisons required to construct the Lempel-Ziv decomposition and thereby conclude that a popular technique of computation of runs using the Lempel-Ziv decomposition cannot achieve an $o(n \log \sigma)$ time bound. In contrast with this, we exhibit an $O(n)$ decision tree algorithm finding all runs in a string. Therefore, in the decision tree model the runs problem is easier than the Lempel-Ziv decomposition. Thus we support the conjecture that there is a linear RAM algorithm finding all runs.

1998 ACM Subject Classification F.2.2 Pattern Matching

Keywords and phrases Lempel-Ziv factorization, runs, repetitions, decision tree, lower bounds

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.582

1 Introduction

String repetitions called runs and the Lempel-Ziv decomposition are structures that are of a great importance for data compression and play a significant role in stringology. Recall that a run of a string is a nonextendable (with the same minimal period) substring whose minimal period is at most half of its length. The definition of the Lempel-Ziv decomposition is given below. In the decision tree model, a widely used model to obtain lower bounds on the time complexity of various algorithms, we consider algorithms finding these structures. We prove that any algorithm finding the Lempel-Ziv decomposition on a general ordered alphabet must perform $\Omega(n \log \sigma)^1$ comparisons in the worst case, where n denotes the length of input string and σ denotes the number of distinct letters in it. Since until recently, the only known efficient way to find all runs of a string was to use the Lempel-Ziv decomposition, one might expect that there is a nontrivial lower bound in the decision tree model on the number of comparisons in algorithms finding all runs. These expectations were also supported by the existence of such a bound in the case of unordered alphabet. In this paper we obtain a somewhat surprising fact: in the decision tree model with an ordered alphabet, there exists a linear algorithm finding all runs. This can be interpreted as one cannot have lower bounds on the decision tree model for algorithms finding runs (a similar result for another problem is provided in [22] for example) but on the other hand, this result supports the conjecture that there is a linear RAM algorithm finding all runs.

¹ Throughout the paper, \log denotes the logarithm with the base 2.

The Lempel-Ziv decomposition [16] is a basic technique for data compression and plays an important role in stringology. It has several modifications used in various compression schemes. The decomposition considered in this paper is used in LZ77-based compression methods. All known efficient algorithms for computation of the Lempel-Ziv decomposition on a general ordered alphabet work in $O(n \log \sigma)$ time (see [6, 20, 10]), though all these algorithms are time and space consuming in practice. However for the case of polynomially bounded integer alphabet, there are efficient linear algorithms [1, 5, 7], space efficient online algorithms [19, 21, 23], and space efficient algorithms in external memory [12].

Repetitions of strings are fundamental objects in both stringology and combinatorics on words. The notion of run, introduced by Main in [17], allows to grasp the whole periodic structure of a given string in a relatively simple form. In the case of unordered alphabet, there are some limitations on the efficiency of algorithms finding periodicities; in particular, it is known [18] that any algorithm that decides whether an input string over a general unordered alphabet has at least one run, requires $\Omega(n \log n)$ comparisons in the worst case. In [14], Kolpakov and Kucherov proved that any string of length n contains $O(n)$ runs and proposed a RAM algorithm finding all runs in linear time provided the Lempel-Ziv decomposition is given. Thereafter much work has been done on the analysis of runs (e.g. see [8, 9, 15, 2]) but until the recent paper [3], all efficient algorithms finding all runs of a string on a general ordered alphabet used the Lempel-Ziv decomposition as a basis. Bannai et al. [3] use a different method based on Lyndon decomposition but unfortunately, their algorithm spends $O(n \log \sigma)$ time too. Clearly, due to the found lower bound, our linear algorithm finding all runs doesn't use the Lempel-Ziv decomposition yet our approach differs from that of [3].

The paper is organized as follows. Section 2 contains some basic definitions used throughout the paper. In Section 3 we give a lower bound on the number of comparisons required to construct the Lempel-Ziv decomposition. In Section 4 we present additional definitions and combinatorial facts that are necessary for Section 5, where we describe our linear decision tree algorithm finding all runs.

2 Preliminaries

A *string of length n* over the alphabet Σ is a map $\{1, 2, \dots, n\} \mapsto \Sigma$, where n is referred to as the length of w , denoted by $|w|$. We write $w[i]$ for the i th letter of w and $w[i..j]$ for $w[i]w[i+1] \dots w[j]$. Let $w[i..j]$ be the empty string for any $i > j$. A string u is a *substring* (or a *factor*) of w if $u = w[i..j]$ for some i and j . The pair (i, j) is not necessarily unique; we say that i specifies an *occurrence* of u in w . A string can have many occurrences in another string. An integer p is a *period* of w if $0 < p < |w|$ and $w[i] = w[i+p]$ for $i = 1, \dots, |w|-p$. For any integers i, j , the set $\{k \in \mathbb{Z}: i \leq k \leq j\}$ (possibly empty) is denoted by $\overline{i, j}$.

The only computational model that is used in this paper is the *decision tree* model. Informally, a decision tree processes input strings of given *fixed* length and each path starting at the root of the tree represents the sequence of pairwise comparisons made between various letters in the string. The computation follows an appropriate path from the root to a leaf; each leaf represents a particular answer to the studied problem.

More formally, a decision tree processing strings of length n is a rooted directed ternary tree in which each interior vertex is labeled with an ordered pair (i, j) of integers, $1 \leq i, j \leq n$, and edges are labeled with the symbols “<”, “=”, “>” (see Fig. 1). The *height* of a decision tree is the number of edges in the longest path from the root to a leaf of the tree. Consider a path p connecting the root of a fixed decision tree to some vertex v . Let t be a string of length n . Suppose that p satisfies the following condition: it contains a vertex labeled with a

some letter of s . Otherwise, we can replace a_{i_j} with the letter $a_{i_{j-1}}$ or $a_{i_{j+1}}$ thus changing the Lempel-Ziv decomposition of the whole string; the details are provided below. Obviously, $|s| + |t| = n$ and there are $(\sigma/2 - 1)^k$ possible strings t of the form (1). Let us take a decision tree which computes the Lempel-Ziv decomposition for the strings of length n . It suffices to prove that each leaf of this tree is reachable by at most one string st with t of the form (1). Indeed, such decision tree has at least $(\sigma/2 - 1)^k$ leaves and the height of the tree is at least $\log_3((\sigma/2 - 1)^k) = k \log_3(\sigma/2 - 1) = \Omega(n \log \sigma)$.

Suppose to the contrary that some leaf of the decision tree is reachable by two distinct strings $r = st$ and $r' = st'$ such that t and t' are of the form (1); then for some $l \in \overline{1, n}$, $r'[l] \neq r[l]$. Obviously $l = |s| + 2l'$ for some $l' \in \overline{1, k}$ and therefore $r[l] = a_p$ for some even $p \in \overline{2, \sigma-2}$. Suppose $r'[l] < r[l]$. Let $l_1 < \dots < l_m$ be the set of all integers $l' > |s|$ such that for any string t_0 of the form (1), if the string $r_0 = st_0$ reaches the same leaf as the string r , then $r_0[l'] = r_0[l]$. Consider a string r'' that differs from r only in the letters $r''[l_1], \dots, r''[l_m]$ and put $r''[l_1] = \dots = r''[l_m] = a_{p-1}$. Let us first prove that the string r'' reaches the same leaf as r . Consider a vertex of the path connecting the root and the leaf reachable by r . Let the vertex be labeled with a pair (i, j) . We have to prove that the comparison of $r''[i]$ and $r''[j]$ leads to the same result as the comparison of $r[i]$ and $r[j]$. The following cases are possible:

1. $i, j \neq l_q$ for all $q \in \overline{1, m}$; then $r[i] = r''[i]$ and $r[j] = r''[j]$;
2. $i = l_q$ for some $q \in \overline{1, m}$ and $r[i] < r[j]$; then since $r''[l_q] = a_{p-1} < a_p = r[l_q] = r[i]$ and $r[j] = r''[j]$, we obtain $r''[i] < r''[j]$;
3. $i = l_q$ for some $q \in \overline{1, m}$ and $r[i] > r[j]$; then we have $j \neq p/2$ because $r[p/2] = r'[p/2] = a_{p-1} > r'[i]$ while $r'[i] > r'[j]$, and thus since $r[i] = a_p > r[j]$, we see that $a_{p-1} = r''[i] > r[j] = r''[j]$;
4. $i = l_q$ for some $q \in \overline{1, m}$ and $r[i] = r[j]$; then, by definition of the set $\{l_1, \dots, l_m\}$, $j = l_{q'}$ for some $q' \in \overline{1, m}$ and $r''[i] = r''[j] = a_{p-1}$;
5. $j = l_q$ for some $q \in \overline{1, m}$; this case is symmetric to the above cases.

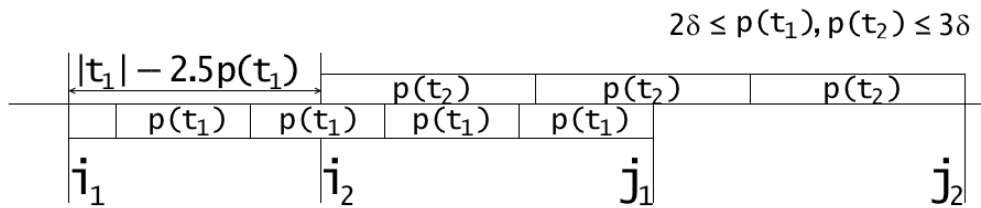
Thus r'' reaches the same leaf as r . But the strings r and r'' have the different Lempel-Ziv decompositions: the Lempel-Ziv decomposition of r'' has one letter factor a_{p-1} at position l_1 while r does not since $r[l_1-1..l_1+1] = a_\sigma a_p a_\sigma$ is a substring of $s = r[1..|s|]$. This contradicts to the fact that the analyzed tree computes the Lempel-Ziv decomposition. ◀

4 Runs

In this section we consider some combinatorial facts that will be useful in our main algorithm described in the following section.

The *exponent* of a string t is the number $|t|/p$, where p is the minimal period of t . A *run* of a string t is a substring $t[i..j]$ of exponent at least 2 and such that both substrings $t[i-1..j]$ and $t[i..j+1]$, if defined, have strictly greater minimal periods than $t[i..j]$. A run whose exponent is greater than or equal to 3 is called a *cubic run*. For a fixed $d \geq 1$, a *d-short run* of a string t is a substring $t[i..j]$ which can be represented as xyx for nonempty strings x and y such that $0 < |y| \leq d$, $|xy|$ is the minimal period of $t[i..j]$, and both substrings $t[i-1..j]$ and $t[i..j+1]$, if defined, have strictly greater minimal periods.

▶ **Example 2.** The string $t = aabaabab$ has four runs $t[1..2] = aa$, $t[4..5] = aa$, $t[1..7] = aabaaba$, $t[5..8] = abab$ and one 1-short run $t[2..4] = aba$. The sum of exponents of all runs is equal to $2 + 2 + \frac{7}{3} + 2 \approx 8.33$.



■ **Figure 2** Overlapping cubic runs t_1 and t_2 such that $2\delta \leq p(t_1) < p(t_2) \leq 3\delta$.

As it was proved in [14], the number of all runs is linear in the length of string. We use a stronger version of this fact.

► **Lemma 3** (see [3, Theorem 9]). *The number of all runs in any string of length n is less than n .*

The following lemma is a straightforward corollary of [15, Lemma 1].

► **Lemma 4** (see [15]). *For a fixed $d \geq 1$, any string of length n contains $O(n)$ d -short runs.*

We also need a classical property of periodic strings.

► **Lemma 5** (see [11]). *Suppose a string w has periods p and q such that $p+q-\text{gcd}(p, q) \leq |w|$; then $\text{gcd}(p, q)$ is a period of w .*

► **Lemma 6.** *Let t_1 and t_2 be substrings with the periods p_1 and p_2 respectively. Suppose t_1 and t_2 have a common substring of the length $p_1 + p_2 - \text{gcd}(p_1, p_2)$ or greater; then t_1 and t_2 have the period $\text{gcd}(p_1, p_2)$.*

Proof. It is immediate from Lemma 5. ◀

Unfortunately, in a string of length n the sum of exponents of runs with the minimal period p or greater is not equal to $O(\frac{n}{p})$ as the following example from [13] shows: $(01)^k(10)^k$. Indeed, for any $p < 2k$, the string $(01)^k(10)^k$ contains at least $k - \lfloor p/2 \rfloor$ runs with the shortest period p or greater: $1(01)^i(10)^{i-1}$ for $i \in \overline{\lfloor p/2 \rfloor, k-1}$. However, it turns out that this property holds for cubic runs.

► **Lemma 7.** *For any $p \geq 2$ and any string t of length n , the sum of exponents of all cubic runs in t with the minimal period p or greater is less than $\frac{12n}{p}$.*

Proof. Consider a string t of length n . For any string u , $e(u)$ denotes the exponent of u and $p(u)$ denotes the minimal period of u . Denote by \mathcal{R} the set of all cubic runs of t . Let $t_1 = t[i_1..j_1]$ and $t_2 = t[i_2..j_2]$ be distinct cubic runs such that $i_1 \leq i_2$. It follows from Lemma 6 that t_1 and t_2 cannot have a common substring of length $p(t_1) + p(t_2)$ or longer. Therefore if $p(t_1)$ and $p(t_2)$ are sufficiently close, then positions i_1 and i_2 cannot be close.

Let δ be a positive integer. Suppose $2\delta \leq p(t_1), p(t_2) \leq 3\delta$; then either $j_1 < i_2$ or $j_1 - i_2 < p(t_1) + p(t_2) \leq 2.5p(t_1)$. The latter easily implies $i_2 - i_1 > \delta$ and therefore $\rho = |\{u \in \mathcal{R} : 2\delta \leq p(u) \leq 3\delta\}| < \frac{n}{\delta}$. Moreover, we have $i_2 - i_1 \geq |t_1| - 2.5p(t_1) = (e(t_1) - 2.5)p(t_1) \geq (e(t_1) - 2.5)2\delta$ (see Fig. 2). Hence $\sum_{u \in \mathcal{R}, 2\delta \leq p(u) \leq 3\delta} (e(u) - 2.5)2\delta \leq n$ and

then
$$\sum_{u \in \mathcal{R}, 2\delta \leq p(u) \leq 3\delta} e(u) \leq \frac{n}{2\delta} + 2.5\rho < \frac{3n}{\delta}.$$

Now it follows that if we have a sequence $\{\delta_i\}$ such that the union of the segments $[2\delta_i, 3\delta_i]$ covers the segment $[p, n]$, then the sum of exponents of all cubic runs with the minimal period p or greater is less than $\sum_i \frac{3n}{\delta_i}$. Denote $\delta_i = (\frac{3}{2})^i$ and $k = \lfloor \log_{\frac{3}{2}} \frac{n}{p} \rfloor$. Evidently $\delta_k = (\frac{3}{2})^k \leq \frac{n}{p}$ and the union of the segments $\{[2\delta_i, 3\delta_i]\}_{i=k}^{\infty}$ covers $[p, n]$. Finally, we obtain

$$\sum_{u \in \mathcal{R}, p(u) \geq p} e(u) < \sum_{i=k}^{\infty} \frac{3n}{\delta_i} = \sum_{i=k}^{\infty} 3n \left(\frac{2}{3}\right)^i = 3n \frac{(2/3)^k}{1/3} \leq 9n \frac{4}{3^p} = \frac{12n}{p}. \quad \blacktriangleleft$$

5 Linear Decision Tree Algorithm Finding All Runs

We say that a decision tree processing strings of length n *finds all runs with a given property* P if for each distinct strings t_1 and t_2 such that $|t_1| = |t_2| = n$ and t_1 and t_2 reach the same leaf of the tree, the substring $t_1[i..j]$ is a run satisfying P iff $t_2[i..j]$ is a run satisfying P for all $i, j \in \overline{1, n}$.

We say that two decision trees processing strings of length n are equivalent if for each reachable leaf a of the first tree, there is a leaf b of the second tree such that for any string t of length n , t reaches a iff t reaches b . The *basic height* of a decision tree is the minimal number k such that each path connecting the root and a leaf of the tree has at most k edges labeled with the symbols “<” and “>”.

For a given positive integer p , we say that a run r of a string is *p-periodic* if $2p \leq |r|$ and p is a (not necessarily minimal) period of r . We say that a run is a *p-run* if it is q -periodic for some q which is a multiple of p . Note that any run is 1-run.

► **Example 8.** Let us describe a “naive” decision tree finding all p -runs in strings of length n . Denote by t the input string. Our tree simply compares $t[i]$ and $t[j]$ for all $i, j \in \overline{1, n}$ such that $|i - j|$ is a multiple of p . The tree has the height $\sum_{i=1}^{\lfloor n/p \rfloor} (n - ip) = O(n^2/p)$ and the same basic height.

Note that a decision tree algorithm finding runs doesn’t report runs in the same way as RAM algorithms do. The algorithm only collects sufficient information to conclude where the runs are; once its knowledge of the structure of the input string becomes sufficient to find all runs without further comparisons of symbols, the algorithm stops and doesn’t care about the processing of obtained information. To simplify the construction of an efficient decision tree, we use the following lemma that enables us to estimate only the basic height of our tree.

► **Lemma 9.** *Suppose a decision tree processing strings of length n has basic height k . Then it is equivalent to a decision tree of height $\leq k + n$.*

Proof. To construct the required decision tree of height $\leq k + n$, we modify the given decision tree of basic height k . First, we remove all unreachable vertices of this tree. After this, we contract each non-branching path into a single edge, removing all intermediate vertices and their outgoing edges. Indeed, the result of a comparison corresponding to such an intermediate vertex is determined by the previous comparisons. So, it is straightforward that the result tree is equivalent to the original tree. Now it suffices to prove that there are at most $n-1$ edges labeled with the symbol “=” along any path connecting the root and some leaf.

Observe that if we perform $n-1$ comparisons on n elements and each comparison yields an equality, then either all elements are equal or the result of at least one comparison can be deduced by transitivity from other comparisons. Suppose a path connecting the root and some leaf has at least n edges labeled with the symbol “=”. By the above observation, the

path contains an edge labeled with “=” leaving a vertex labeled with (i, j) such that the equality of the i th and the j th letters of the input string follows by transitivity from the comparisons made earlier along this path. Then this vertex has only one reachable child. But this is impossible because all such vertices of the original tree were removed during the contraction step. This contradiction finishes the proof. ◀

► **Lemma 10.** *For any integers n and p , there is a decision tree that finds all p -periodic runs in strings of length n and has basic height at most $2\lceil n/p \rceil$.*

Proof. Denote by t the input string. The algorithm is as follows (note that the resulting decision tree contains only comparisons of letters of t):

1. assign $i \leftarrow 1$;
2. if $t[i] \neq t[i+p]$, then assign $i \leftarrow i + p$, $h \leftarrow \min\{i, n - p\}$ and for $i' = h-1, h-2, \dots$, compare $t[i']$ and $t[i'+p]$ until $t[i'] \neq t[i'+p]$;
3. increment i and if $i \leq n - p$, jump to line 2.

Obviously, the algorithm performs at most $2\lceil n/p \rceil$ symbol comparisons yielding inequalities. Let us prove that the algorithm finds all p -periodic runs.

Let $t[j..k]$ be a p -periodic run. For the sake of simplicity, suppose $1 < j < k < n$. To discover this run, one must compare $t[l]$ and $t[l+p]$ for each $l \in \overline{j-1, k-p+1}$. Let us show that the algorithm performs all these comparisons. Suppose, to the contrary, for some $l \in \overline{j-1, k-p+1}$, the algorithm doesn't compare $t[l]$ and $t[l+p]$. Then for some i_0 such that $i_0 < l < i_0 + p$, the algorithm detects that $t[i_0] \neq t[i_0+p]$ and “jumps” over l by assigning $i = i_0 + p$ at line 2. Obviously $i_0 < j$. Then $h = \min\{i_0 + p, n - p\} < k$ and hence for each $i' = h-1, h-2, \dots, j-1$, the algorithm compares $t[i']$ and $t[i'+p]$. Since $j - 1 \leq l < i_0 + p$, $t[l]$ and $t[l+p]$ are compared, contradicting to our assumption. ◀

► **Theorem 11.** *There is a constant c such that for any integer n , there exists a decision tree of height at most cn that finds all runs in strings of length n .*

Proof. By Lemma 9, it is sufficient to build a decision tree with linear basic height. So, below we count only the comparisons yielding inequalities and refer to them as *inequality comparisons*. In fact we prove the following more general fact: for a given string t of length n and a positive integer p , we find all p -runs performing $O(n/p)$ inequality comparisons. To find all runs of a string, we simply put $p = 1$.

Let us outline the plot of the proof. Firstly, we briefly describe all steps of our decision tree algorithm finding all p -runs. Secondly, we discuss each of these steps: its correctness and the number of inequality comparisons performed; and this is the largest part of the proof. Finally, we estimate the overall number of inequality comparisons; the main difficulty of the estimation is in the recursive nature of our algorithm.

The algorithm consists of five steps. Each step finds p -runs of t with a given property. Let us choose a positive integer constant $d \geq 2$ (the exact value is defined below.) The algorithm is roughly as follows:

1. find in a straightforward manner all p -runs having periods $\leq dp$;
2. using the information from step 1, build a new string t' of length n/p such that periodic factors of t and t' are strongly related to each other;
3. find p -runs of t related to periodic factors of t' with exponents less than 3;
4. find p -runs of t related to periodic factors of t' with periods less than d ;
5. find p -runs of t related to other periodic factors of t' by calling steps 1–5 recursively for some substrings of t .

Step 1. Initially, we split the string t into n/p contiguous blocks of length p (if n is not a multiple of p , we pad t on the right to the required length with a special symbol which is less than all other symbols.) For each $i \in \overline{1, n/p}$ and $j \in \overline{1, d}$, we denote by $m_{i,j}$ the minimal $k \in \overline{1, p}$ such that $t[(i-1)p+k] \neq t[(i-1)p+k+jp]$ and we put $m_{i,j} = -1$ if $ip + jp > n$ or there is no such k . To compute $m_{i,j}$, we simply compare $t[(i-1)p+k]$ and $t[(i-1)p+k+jp]$ for $k = 1, 2, \dots, p$ until $t[(i-1)p+k] \neq t[(i-1)p+k+jp]$.

► **Example 12.** Let $t = bbba \cdot aada \cdot aaaa \cdot aaaa \cdot aada \cdot aaaa \cdot aaab \cdot bbbb \cdot bbbb$, $p = 4$, $d = 2$. The following table contains $m_{i,j}$ for $j = 1, 2$:

i	1	2	3	4	5	6	7	8	9
$t[(i-1)p+1..ip]$	bbba	aada	aaaa	aaaa	aada	aaaa	aaab	bbbb	bbbb
$m_{i,1}, m_{i,2}$	1, 1	3, 3	-1, 3	3, -1	3, 3	4, 1	1, 1	-1, -1	-1, -1

To compute a particular value of $m_{i,j}$, one needs at most one inequality comparison (zero inequality comparisons if the computed value is -1 .) Further, for each $i \in \overline{1, n/p}$ and $j \in \overline{1, d}$, we compare $t[ip-k]$ and $t[ip-k+jp]$ (if defined) for $k = 0, 1, \dots, p-1$ until $t[ip-k] \neq t[ip-k+jp]$; similar to the above computation of $m_{i,j}$, this procedure performs at most one inequality comparison for any given i and j . Hence, the total number of inequality comparisons is at most $2dn/p$. Once these comparisons are made, all pq -periodic runs in the input string are determined for all $q \in \overline{1, d}$.

Step 2. Now we build an auxiliary structure induced by $m_{i,j}$ on the string t . In this step, no comparisons are performed; we just establish some combinatorial properties required for further steps. We make use of the function:

$$\text{sgn}(a, b) = \begin{cases} -1, & a < b, \\ 0, & a = b, \\ 1, & a > b. \end{cases}$$

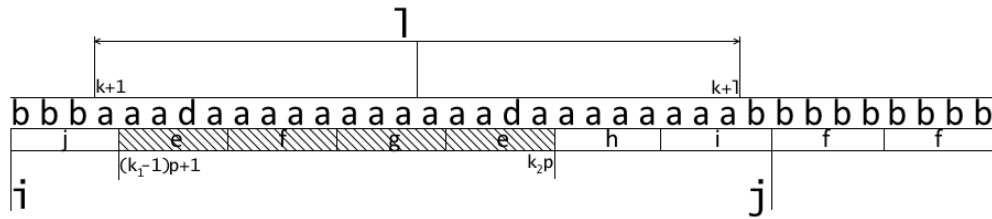
We create a new string t' of length n/p . The alphabet of this string can be taken arbitrarily, we just describe which letters of t' coincide and which do not. For each $i_1, i_2 \in \overline{1, n/p}$, $t'[i_1] = t'[i_2]$ iff for each $j \in \overline{1, d-1}$, either $m_{i_1,j} = m_{i_2,j} = -1$ or the following conditions hold simultaneously:

$$\begin{aligned} & m_{i_1,j} \neq -1, m_{i_2,j} \neq -1, \\ & m_{i_1,j} = m_{i_2,j}, \\ & \text{sgn}(t[(i_1-1)p+m_{i_1,j}], t[(i_1-1)p+m_{i_1,j}+jp]) = \text{sgn}(t[(i_2-1)p+m_{i_2,j}], t[(i_2-1)p+m_{i_2,j}+jp]). \end{aligned}$$

Note that the status of each of these conditions is known from step 1. Also note that the values $m_{i,d}$ are not used in the definition of t' ; we computed them only to find all dp -periodic p -runs.

► **Example 12 (continued).** Denote $s_i = \text{sgn}(t[(i-1)p+m_{i,1}], t[(i-1)p+m_{i,1}+p])$. Let $\{e, f, g, h, i, j\}$ be a new alphabet for the string t' . The following table contains $m_{i,1}$, s_i , and $t'[i]$:

i	1	2	3	4	5	6	7	8	9
$t[(i-1)p+1..ip]$	bbba	aada	aaaa	aaaa	aada	aaaa	aaab	bbbb	bbbb
$m_{i,1}$	1	3	-1	3	3	4	1	-1	-1
s_i	1	1	-	-1	1	-1	-1	-	-
$t'[i]$	j	e	f	g	e	h	i	f	f



■ **Figure 3** A p -run corresponding to d -short run $t'[k_1..k_2] = efge$, where $k_1 = 2, k_2 = 5, p = 4, d = 2, q = 3, k = 3, l = 2pq = 24, i = (k_1 - 2)p + 1 = 1, j = (k_2 + d)p = 28$.

If t contains two identical sequences of d blocks each, i.e., $t[(i_1 - 1)p + 1..(i_1 - 1 + d)p] = t[(i_2 - 1)p + 1..(i_2 - 1 + d)p]$ for some i_1, i_2 , then $m_{i_1, j} = m_{i_2, j}$ for each $j \in \overline{1, d - 1}$ and hence $t'[i_1] = t'[i_2]$. This is why $t'[2] = t'[5]$ in Example 12. On the other hand, equal letters in t' may correspond to different sequences of blocks in t , like the letters $t'[3] = t'[8]$ in Example 12. The latter property makes the subsequent argument more involved but allows us to keep the number of inequality comparisons linear. Let us point out the relations between periodic factors of t and t' .

Let for some $q > d, t[k + 1..k + l]$ be a pq -periodic p -run, i.e., $t[k + 1..k + l]$ is a p -run that is not found in step 1. Denote $k' = \lceil k/p \rceil$. Since $t[k + 1..k + l]$ is pq -periodic, t' has some periodicity in the corresponding substring, namely, $u = t'[k' + 1..k' + \lfloor l/p \rfloor - d]$ has the period q (see example below). Let $t'[k_1..k_2]$ be the largest substring of t' containing u and having the period q . Since $2q \leq \lfloor l/p \rfloor = |u| + d, t'[k_1..k_2]$ is either a d -short run with the minimal period q or a run whose minimal period divides q .

► **Example 12** (continued). Consider Fig. 3. Let $k = 3, l = 24$. The string $t[k + 1..k + l] = a \cdot aada \cdot aaaa \cdot aaaa \cdot aada \cdot aaaa \cdot aaaa \cdot aaa$ is a p -run with the minimal period $pq = 12$ (here $q = 3 > 2 = d$). Denote $k' = \lceil k/p \rceil = 1, k_1 = 2$, and $k_2 = 5$. The string $t'[k' + 1..k' + \lfloor l/p \rfloor - d] = t'[k_1..k_2] = t'[2..5] = efge$ is a d -short run of t' with the minimal period $q = 3$.

Conversely, given a run or d -short run $t'[k_1..k_2]$ with the minimal period q , we say that a p -run $t[k + 1..k + l]$ corresponds to $t'[k_1..k_2]$ (or $t[k + 1..k + l]$ is a p -run corresponding to $t'[k_1..k_2]$) if $t[k + 1..k + l]$ is, for some integer r, rpq -periodic and $t'[k' + 1..k' + \lfloor l/p \rfloor - d]$, where $k' = \lceil k/p \rceil$, is a substring of $t'[k_1..k_2]$ (see Fig. 3 and Example 12).

The above observation shows that each p -run of t that is not found in step 1 corresponds to some run or d -short run of t' . Let us describe the substring that must contain all p -runs of t corresponding to a given run or d -short run $t'[k_1..k_2]$. Denote $i = (k_1 - 2)p + 1$ and $j = (k_2 + d)p$. Now it is easy to see that if $t[k + 1..k + l]$ is a p -run corresponding to $t'[k_1..k_2]$, then $t[k + 1..k + l]$ is a substring of $t[i..j]$.

► **Example 12** (continued). For $k = 3$ and $l = 24$, the string $t[k + 1..k + l] = a \cdot aada \cdot aaaa \cdot aaaa \cdot aada \cdot aaaa \cdot aaaa \cdot aaa$ is a p -run corresponding to $t'[k_1..k_2] = efge$, where $k_1 = 2, k_2 = 5$. Indeed, the string $t'[k' + 1..k' + \lfloor l/p \rfloor - d] = t'[2..5]$, for $k' = \lceil k/p \rceil = 1$, is a substring of $t'[k_1..k_2]$. Denote $i = (k_1 - 2)p + 1 = 1, j = (k_2 + d)p = 28$. Observe that $t[k + 1..k + l] = t[4..27]$ is a substring of $t[i..j] = t[1..28]$.

It is possible that there is another p -run of t corresponding to the string $t'[k_1..k_2]$. Consider the following example.

► **Example 13**. Let $t = fabc ded abcd eda aif j f f a a i f j f f, p = 2, d = 2$. Denote $s_i = \text{sgn}(t[(i - 1)p + m_{i,1}], t[(i - 1)p + m_{i,1} + p])$. Let $\{w, x, y, z\}$ be a new alphabet for the string

t' . The following table contains $m_{i,1}$, s_i , and t' :

i	1	2	3	4	5	6	7	8	9	10	11	12	13
$t[(i-1)p+1..ip]$	fa	bc	de	da	bc	de	da	ai	fj	fa	ai	fj	ff
$m_{i,1}$	1	1	2	1	1	2	1	1	2	1	1	2	-1
s_i	1	-1	1	1	-1	1	1	-1	1	1	-1	1	-
$t'[i]$	x	y	z	x	y	z	x	y	z	x	y	z	w

Note that p -runs $t[2..13] = abcded \cdot abcded$ and $t[14..25] = aai f j f \cdot aai f j f$ correspond to the same p -run of t' , namely, $t'[1..12] = xyz \cdot xyz \cdot xyz \cdot xyz$.

Thus to find for all $q > d$ all pq -periodic p -runs of t , we must process all runs and d -short runs of t' .

Step 3. Consider a noncubic run $t'[k_1..k_2]$. Let q be its minimal period. Denote $i = (k_1 - 2)p + 1$ and $j = (k_2 + d)p$. The above analysis shows that any p -run of t corresponding to $t'[k_1..k_2]$ is a p' -periodic run of $t[i..j]$ for some $p' = pq, 2pq, \dots, lpq$, where $l = \lfloor (j - i + 1)/(2pq) \rfloor$. Since $(k_2 - k_1 + 1)/q < 3$, we have $l = \lfloor (k_2 - k_1 + 2)/(2q) + d/(2q) \rfloor = O(d)$. Hence to find all p -runs of $t[i..j]$, it suffices to find for each $p' = pq, 2pq, \dots, lpq$ all p' -periodic runs of $t[i..j]$ using Lemma 10. Thus the processing performs $O(l(j - i + 1)/pq) = O(d^2) = O(1)$ inequality comparisons. Analogously we process d -short runs of t' . Therefore, by Lemmas 3 and 4, only $O(|t'|) = O(n/p)$ inequality comparisons are required to process all d -short runs and noncubic runs of t' .

Now it suffices to find all p -runs of t corresponding to cubic runs of t' .

Step 4. Let $t'[k_1..k_2]$ be a cubic run with the minimal period q . In this step we consider the case $q < d$. It turns out that such small-periodic substrings of t' correspond to substrings in t that are either periodic and discovered at step 1, or aperiodic. Therefore this step does not include any comparisons. The precise explanation follows.

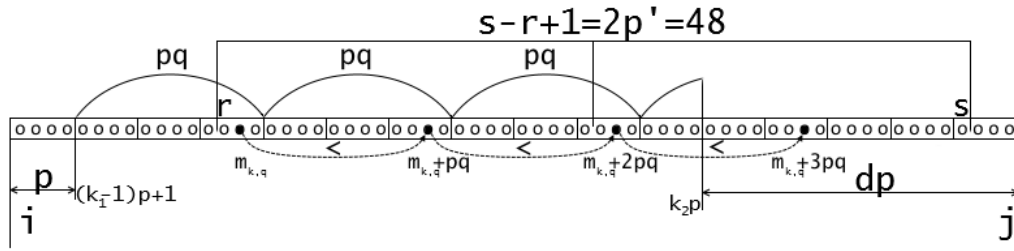
Suppose that $m_{k,q} = -1$ for all $k \in \overline{k_1, k_1+q-1}$. Then $m_{k,q} = -1$ for all $k = k_1, \dots, k_2$ by periodicity of $t'[k_1..k_2]$. Therefore by the definition of $m_{k,q}$, we have $t[k] = t[k+pq]$ for all $k \in \overline{(k_1-1)p+1, k_2p}$. Hence the substring $t[(k_1-1)p+1..k_2p+pq]$ has the period pq . Now it follows from Lemma 6 that any p -run of t corresponding to $t'[k_1..k_2]$ is pq -periodic and therefore was found in step 1 because $pq < dp$.

Suppose that $m_{k,q} \neq -1$ for some $k \in \overline{k_1, k_1+q-1}$. Denote $s = (k - 1)p + m_{k,q}$, $l = \lfloor (k_2p - s)/pq \rfloor + 1$. Let $r \in \overline{1, l}$. Since $t'[k] = t'[k+rq]$, we have $m_{k,q} = m_{k+rq,q}$ and $\text{sgn}(t[s], t[s+pq]) = \text{sgn}(t[s+rpq], t[s+(r+1)pq])$ (see Fig. 4). Therefore, one of the following sequences of inequalities holds:

$$\begin{aligned} t[s] < t[s+pq] < t[s+2pq] < \dots < t[s+lpq], \\ t[s] > t[s+pq] > t[s+2pq] > \dots > t[s+lpq]. \end{aligned} \tag{2}$$

Let p' be a multiple of pq such that $p' > dp$. Now it suffices to show that due to the found ‘‘aperiodic chain’’, there are no p' -periodic p -runs of t corresponding to $t'[k_1..k_2]$.

Suppose, to the contrary, $t[r..s]$ is a p' -periodic p -run corresponding to $t'[k_1..k_2]$ (see Fig. 4). Denote $r' = \lceil (r - 1)/p \rceil$ and $l' = \lfloor (s - r + 1)/p \rfloor$. By the definition of corresponding p -runs, $u = t'[r'+1..r'+l'-d]$ is a substring of $t'[k_1..k_2]$. Since $s - r + 1 \geq 2p'$ and $p' > dp$, we have $|u| = l' - d \geq 2p'/p - d > p'/p$. Therefore, $r \leq r'p + m_{r'+1,q} < r'p + m_{r'+1,q} + p' \leq s$ and the inequalities (2) imply $t[r'p+m_{r'+1,q}] \neq t[r'p+m_{r'+1,q} + p']$, a contradiction.



■ **Figure 4** A cubic run of t' with the shortest period $q = 3 < d = 5$, where $p = 4$, $k_1 = 2$, $k_2 = 11$, $k = 4$, $m_{k,q} = 3$, $l = 3$, $p' = 2pq = 24$.

Step 5. Let $t'[k_1..k_2]$ be a cubic run with the minimal period q such that $q \geq d$. Denote $i = (k_1 - 2)p + 1$ and $j = (k_2 + d)p$. To find all p -runs corresponding to the run $t'[k_1..k_2]$, we make a recursive call executing steps 1–5 again with new parameters $n = j - i + 1$, $p = pq$, and $t = t[i..j]$.

After the analysis of all cubic runs of t' , all p -runs of t are found and the algorithm stops. Now it suffices to estimate the number of inequality comparisons performed during any run of the described algorithm.

Time analysis. As shown above, steps 1–4 require $O(n/p)$ inequality comparisons. Let $t'[i_1..j_1], \dots, t'[i_k..j_k]$ be the set of all cubic runs of t' with the minimal period d or greater. For $l \in \overline{1, k}$, denote by q_l the minimal period of $t'[i_l..j_l]$ and denote $n_l = j_l - i_l + 1$. Let $T(n, p)$ be the number of inequality comparisons required by the algorithm to find all p -runs in a string of length n . Then $T(n, p)$ can be computed by the following formula:

$$T(n, p) = O(n/p) + T((n_1 + d + 1)p, pq_1) + \dots + T((n_k + d + 1)p, pq_k) .$$

For $l \in \overline{1, k}$, the number n_l/q_l is, by definition, the exponent of $t'[i_l..j_l]$. It follows from Lemma 7 that the sum of exponents of all cubic runs of t' with the shortest period d or larger is less than $\frac{12n}{dp}$. Note that for any $l \in \overline{1, k}$, $n_l \geq 3q_l \geq 3d$ and therefore $n_l + d + 1 < 2n_l$. Thus assuming $d = 48$, we obtain $\frac{(n_1+d+1)p}{pq_1} + \dots + \frac{(n_k+d+1)p}{pq_k} < \frac{2n_1}{q_1} + \dots + \frac{2n_k}{q_k} \leq \frac{24n}{dp} = \frac{n}{2p}$. Finally, we have $T(n, p) = O(\frac{n}{2^0 p} + \frac{n}{2^1 p} + \frac{n}{2^2 p} + \dots) = O(n/p)$. The reference to Lemma 9 ends the proof. ◀

6 Conclusion

Lemma 9 which expresses a non-constructive property is the bottleneck for the conversion of our decision tree algorithm into a RAM algorithm. So, it remains an open problem whether there exists a linear RAM algorithm finding all runs in a string over a general ordered alphabet. Moreover, it is unknown if there is a linear RAM algorithm that decides whether a given string has runs (this problem was posed in [4, Chapter 4]). However, it is still possible that there are nontrivial lower bounds in some more sophisticated models that are strongly related to RAM model.

Acknowledgement The author would like to thank Arseny M. Shur for many valuable comments and the help in the preparation of this paper. Also the author wishes to acknowledge anonymous referees for detailed and helpful comments.

References

- 1 M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms*, 2(1):53–86, 2004.
- 2 G. Badkobeh, M. Crochemore, and C. Toopsuwan. Computing the maximal-exponent repeats of an overlap-free string in linear time. In *String Processing and Information Retrieval*, pages 61–72. Springer, 2012.
- 3 H. Bannai, T. I, S. Inenaga, Y. Nakashima, M. Takeda, and K. Tsuruta. The “runs” theorem. *arXiv preprint arXiv:1406.0263v4*, 2014.
- 4 D. Breslauer. *Efficient string algorithmics*. PhD thesis, Columbia University, 1992.
- 5 G. Chen, S. J. Puglisi, and W. F. Smyth. Lempel–ziv factorization using less time & space. *Mathematics in Computer Science*, 1(4):605–623, 2008.
- 6 M. Crochemore. Transducers and repetitions. *Theoretical Computer Science*, 45:63–86, 1986.
- 7 M. Crochemore, L. Ilie, and W. F. Smyth. A simple algorithm for computing the lempel-ziv factorization. In *Data Compression Conference (DCC’08)*, pages 482–488. IEEE, 2008.
- 8 M. Crochemore, L. Ilie, and L. Tinta. The “runs” conjecture. *Theoretical Computer Science*, 412(27):2931–2941, 2011.
- 9 M. Crochemore, M. Kubica, J. Radoszewski, W. Rytter, and T. Waleń. On the maximal sum of exponents of runs in a string. *Journal of Discrete Algorithms*, 14:29–36, 2012.
- 10 E. R. Fiala and D. H. Greene. Data compression with finite windows. *Communications of the ACM*, 32(4):490–505, 1989.
- 11 N. J. Fine and H. S. Wilf. Uniqueness theorems for periodic functions. *Proceedings of the American Mathematical Society*, 16(1):109–114, 1965.
- 12 J. Karkkainen, D. Kempa, and S. J. Puglisi. Lempel-ziv parsing in external memory. In *Data Compression Conference (DCC’14)*, pages 153–162. IEEE, 2014.
- 13 R. Kolpakov. On primary and secondary repetitions in words. *Theoretical Computer Science*, 418:71–81, 2012.
- 14 R. Kolpakov and G. Kucherov. Finding maximal repetitions in a word in linear time. In *40th Annual Symposium on Foundations of Computer Science*, pages 596–604. IEEE, 1999.
- 15 R. Kolpakov, M. Podolskiy, M. Posypkin, and N. Khrapov. Searching of gapped repeats and subrepetitions in a word. In *Combinatorial Pattern Matching*, pages 212–221. Springer, 2014.
- 16 A. Lempel and J. Ziv. On the complexity of finite sequences. *Information Theory, IEEE Transactions on*, 22(1):75–81, 1976.
- 17 M. G. Main. Detecting leftmost maximal periodicities. *Discrete Applied Mathematics*, 25(1):145–153, 1989.
- 18 M. G. Main and R. J. Lorentz. Linear time recognition of squarefree strings. In *Combinatorial Algorithms on Words*, pages 271–278. Springer, 1985.
- 19 D. Okanohara and K. Sadakane. An online algorithm for finding the longest previous factors. In *Algorithms-ESA 2008*, pages 696–707. Springer, 2008.
- 20 M. Rodeh, V. R. Pratt, and S. Even. Linear algorithm for data compression via string matching. *Journal of the ACM (JACM)*, 28(1):16–24, 1981.
- 21 T. Starikovskaya. Computing lempel-ziv factorization online. In *Mathematical Foundations of Computer Science 2012*, pages 789–799. Springer, 2012.
- 22 J. D. Ullman, A. V. Aho, and D. S. Hirschberg. Bounds on the complexity of the longest common subsequence problem. *Journal of the ACM (JACM)*, 23(1):1–12, 1976.
- 23 J. Yamamoto, H. Bannai, S. Inenaga, and M. Takeda. Faster compact on-line lempel-ziv factorization. *arXiv preprint arXiv:1305.6095v1*, 2013.

Visibly Counter Languages and Constant Depth Circuits

Andreas Krebs, Klaus-Jörn Lange, and Michael Ludwig

WSI – University of Tübingen
Sand 13, 72076 Tübingen, Germany
{krebs,lange,ludwig}@informatik.uni-tuebingen.de

Abstract

We examine visibly counter languages, which are languages recognized by visibly counter automata (a.k.a. input driven counter automata). We are able to effectively characterize the visibly counter languages in AC^0 and show that they are contained in $FO[+]$.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes, F.4.1 Mathematical Logic, F.4.3 Formal Languages

Keywords and phrases visibly counter automata, constant depth circuits, AC^0 , $FO[+]$

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.594

1 Introduction

One important topic of complexity theory is the characterization of regular languages contained in constant depth complexity classes [7, 3, 5, 4]. In [4] Barrington et al. showed that the regular sets in AC^0 are exactly the languages definable by first order logic using regular predicates.

We extend this approach to certain non-regular languages. The *visibly pushdown languages* (VPL) are a sub-class of the context-free languages containing the regular sets which exhibits many of the decidability and closure properties of the regular languages. Their essential feature is that the use of the pushdown store is purely input-driven: The input alphabet is partitioned into call, return and internal letters. On a call letter, the pushdown automaton (PDA) must push a symbol onto its stack, on a return letter it must pop a symbol, and on an internal letter it cannot access the stack at all. This is a severe restriction, however it allows visibly pushdown automata (VPA) to accept non-regular languages, the simplest example being $a^n b^n$. At the same time, VPA are less powerful than general PDA: They even cannot check if a string has an equal number of a 's and b 's. In fact, due to the visible nature of the stack, membership testing for VPA might be easier than for general PDA. It is known to be in NC^1 [8] and hence it is NC^1 -complete. On the other hand the membership problem for the context-free languages is complete for SAC^1 [20].

Visibly counter automata (VCA) [2] were introduced by Bárány, Löding, and Serre as a restricted model of visibly pushdown automata as they were of use to decide a certain sub-class membership problem of VPL. They still contain all regular sets.

In this paper, we show that all visible one-counter languages in AC^0 are definable by first order logic using addition as a numerical predicate. Our techniques allow us to decide whether a visible one-counter language is in fact a member of AC^0 .

Examples of visible counter languages are:

- The set $\{a^n b^n \mid n \geq 0\}^*$, the Kleene-closure of $\{a^n b^n \mid n \geq 0\}$, is in AC^0 .
- The one-sided Dyck language of a single pair of parentheses. This language is not in AC^0 .



© Andreas Krebs, Klaus-Jörn Lange, and Michael Ludwig;
licensed under Creative Commons License CC-BY
32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).
Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 594–607
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



- The set $\{\{a, aba\}^n b^n \mid n \geq 0\}$ is as hard as the set EQUALITY of all strings in $\{0, 1\}^*$ in which the numbers of ones coincides with those of zeros and is thus not in AC^0 .

Another interest stems from descriptive complexity [13]. Here we use the model of predicate logic for language recognition where variables are associated with word positions. An interesting question is whether one can extend the conjecture of Straubing to a family of non-regular languages. It states that for any set of quantifiers Q the intersection with regular languages relates to the use of regular predicates: $Q[arb] \cap \text{REG} = Q[\text{Reg}]$. This question was examined in detail in [15]. Here we show $\text{FO}[arb] \cap \text{VCL} \subseteq \text{FO}[+]$, which shows that only the addition predicate is needed.

The rough idea of our proof is to exhibit two decidable properties of a visible counter automaton \mathcal{A} which together characterize the property of the accepted language $L(\mathcal{A})$:

- The first property concerns the "height behavior" of words. E.g. the Dyck set contains all possible height progressions and is not in AC^0 . However the language $L = \{a^n b^n \mid n \geq 0\}$ has a very simple height behavior. The language L^* is more complicated in this respect but is still in AC^0 . We introduce the notion of *simple height behavior* of a language to capture this. If $L(\mathcal{A})$ has simple height behavior, then a matching predicate is definable in $\text{FO}[+]$. If not, then $L(\mathcal{A})$ is not in AC^0 .
- The second property is a modification of quasi-aperiodicity (see [4, 19]) of regular languages fit for our needs. If $L(\mathcal{A})$ does not have the property we can reduce a language outside AC^0 to it. Otherwise we get a certain $\text{FO}[\text{Reg}]$ formula.

If $L(\mathcal{A})$ has the two properties then by using the matching predicate and the $\text{FO}[\text{Reg}]$ formula, we can build an $\text{FO}[+]$ formula for $L(\mathcal{A})$.

Due to the space constraints we omit some of the proofs. We thank the anonymous referees for their helpful comments.

2 Preliminaries

By \mathbb{Z} we denote the integers, by \mathbb{N} the non-negative integers and by \mathbb{Q} the rational numbers. An *alphabet* is a finite set Σ and ϵ is the empty word. A *language* is a subset of Σ^* . For a word w , $|w|$ is the length of the word and $|w|_M$ for $M \subseteq \Sigma$ is the number of letters in w which belong to M . If not locally defined otherwise, w_i is the letter in position i in w . For a language $L \subseteq \Sigma^*$, the set $F(L) \subseteq \Sigma^*$ is the set of all *factors* of words in L . For every language $L \subseteq \Sigma^*$ we define a congruence relation, the *syntactic congruence* of L : For $x, y \in \Sigma^*$ it is $x \sim_L y$ iff for all $u, v \in \Sigma^*$ we have $uxv \in L \Leftrightarrow uyv \in L$. The *syntactic monoid* is $\text{syn}(L)$, the set of equivalence classes under \sim_L with the induced multiplication. The *syntactic morphism* of L is $\eta_L: \Sigma^* \rightarrow \text{syn}(L)$.

We use circuits as a model of computation. Important complexity classes include:

- AC^0 - polynomial-size circuits of constant depth with Boolean gates of arbitrary fan-in.
- ACC_k^0 - AC^0 circuits plus modulo- k -gates. ACC^0 is the union of ACC_k^0 for all k .
- TC^0 - polynomial-size circuits of constant depth with threshold gates of arbitrary fan-in.
- NC^1 - polynomial-size circuits of logarithmic depth with bounded fan-in.

Circuits have a certain input length. However it is desirable to be able to treat arbitrary long inputs. This is achieved with families of circuits which contain one circuit for each input length. If for some $n \in \mathbb{N}$ the circuit with input length n is computable in some complexity bound, we speak of uniformity. One prominent example is so called DLOGTIME-uniformity. Consult e.g. [21] for further references on circuit complexity.

We also use the model of first-order predicate logic over finite words. Variables range over word positions and so the numerical predicates are sets of word positions. E.g. $<$ is a

predicate of arity two with obvious semantic. We allow for existential and all quantification: \exists, \forall . We write $\text{FO}[\langle] for the set of languages we get of first-order formulas with \langle predicate. The $+$ predicate has arity three: $(i, j, k) \in +$ iff $i + j = k$.$

The class (of non-uniform) AC^0 coincides with first order logic with arbitrary numerical predicates, which we denote by $\text{FO}[arb]$ [12, 10]. For the interplay of circuits and logic see [19]. A prominent theorem is the equivalence of star-free languages, $\text{FO}[\langle]$ languages and languages with aperiodic syntactic monoid [18, 16]. A monoid M is aperiodic if for all $m \in M$ it holds that $m^i = m^{i+1}$ for some i or equivalently if no subset of M is a non-trivial group. For us a related notion is also important: The intersection of AC^0 and the regular languages is captured exactly by the set of languages which have a quasi-aperiodic syntactic morphism η_L . It is quasi-aperiodic if for all $t > 0$, $\eta_L(\Sigma^t)$ does not contain a non-trivial group. Languages with quasi-aperiodic syntactic morphisms are exactly the ones in $\text{FO}[Reg]$, that is first order logic using the regular predicates, which are the order predicate and the modulo predicates [4].

We will use the following languages:

- EQUALITY = $\{w \in \{0, 1\}^* : |w|_0 = |w|_1\}$ which is TC^0 -hard.
- $\text{MOD}_k = \{w \in \{0, 1\}^* : |w|_0 \equiv 0 \pmod{k}\}$ which is ACC_k^0 -hard.

Neither EQUALITY nor PARITY = MOD_2 is in AC^0 [9, 11].

Mehlhorn [17] and independently also Alur and Madhusudan [1] introduced *input-driven* or *visibly pushdown automata*. Here, the input symbol determines the stack operation, i.e. if a symbol is pushed or popped. This leads to a partition of Σ into call, return and internal letters: $\Sigma = \Sigma_{\text{call}} \cup \Sigma_{\text{ret}} \cup \Sigma_{\text{int}}$. Then $\hat{\Sigma} = (\Sigma_{\text{call}}, \Sigma_{\text{ret}}, \Sigma_{\text{int}})$ is a visibly alphabet. In the rest of the paper we always assume that there is a visibly alphabet for Σ .

We define a function $\Delta : \Sigma^* \rightarrow \mathbb{Z}$ which gives us the *height* of a word by $\Delta(w) = |w|_{\Sigma_{\text{call}}} - |w|_{\Sigma_{\text{ret}}}$. Each word w over a visibly alphabet can be assigned its *height profile* w^Δ , which is a map $\{0, \dots, |w|\} \rightarrow \mathbb{Z}$ with $w^\Delta(i) = \Delta(w_1 \cdots w_i)$. A word w is *well-matched* if w^Δ maps into \mathbb{N} and $\Delta(w) = 0$. Two positions i, j of a word are *matched* if $w_i \in \Sigma_{\text{call}}$, $w_j \in \Sigma_{\text{ret}}$, and $w_{i+1} \dots w_{j-1}$ is well-matched. In a well-matched word, every position i has a matching position j , unless w_i is an internal letter. Thus, positions with letters in Σ_{int} are always unmatched. We say what a word w has a *non-negative height profile* if $\Delta(w_1 \cdots w_i) \geq 0$ for all $i \in \{0, \dots, |w|\}$.

Bárány, Löding, and Serre [2] introduced the notion of *visibly counter automata* (VCA). Since every VPA can be determinized and this is also true for visibly counter automata, we restrict ourselves to deterministic automata:

► **Definition 1** (*m-VCA*). An *m-VCA* \mathcal{A} over $\hat{\Sigma} = (\Sigma_{\text{call}}, \Sigma_{\text{ret}}, \Sigma_{\text{int}})$ is a tuple: $\mathcal{A} = (Q, q_0, F, \hat{\Sigma}, \delta_0, \dots, \delta_m)$, where $m \geq 0$ is the threshold, Q is the set of states, q_0 the initial state, F the set of final states and $\delta_i: Q \times \Sigma \rightarrow Q$ is the transition functions.

A configuration is an element of $Q \times \mathbb{N}$. Note that *m-VCA*s, similar to VPAs, can only recognize words where the height profile is non-negative. All other words are rejected. An *m-VCA* \mathcal{A} performs the following transition when a letter $\sigma \in \Sigma$ is read: $(q, k) \xrightarrow{\sigma} (\delta_{\min(m, k)}(q, \sigma), k + \Delta(\sigma))$. Then $w \in L(\mathcal{A})$ iff $(q_0, 0) \xrightarrow{w} (f, h)$ for $f \in F$ and $h \geq 0$. Note that we slightly modified the semantics compared to [2], as they required $h = 0$ for a word to be accepted. Our version is more general since we can accept languages which contain words w with $\Delta(w) > 0$. Bárány et al. needed that their 0-VCA only accepts languages of well-matched words. With our definition we would need an 2-VCA so simulate this.

We say that a word w loops through q if $(q, h) \xrightarrow{w} (q, h + \Delta(w))$ and $\Delta(w_1 \cdots w_i) + h > m$ for all positions i .

► **Definition 2** (VCL). The class of the visibly counter languages (VCL) contains the languages recognized by an m -VCA for some m .

3 Properties of Visibly Counter Languages

We will present two properties which a VCL language L must fulfill to be in AC^0 . The first property concerns the behavior of the height profiles of the words in L . Intuitively, we need to be able to compute most of the height profile in AC^0 . The second property assumes that the height profile is known and is about computing the states which the automaton will pass on its run on the input. This is closely related to the property a regular language must have to be in AC^0 .

In the following we will decompose the automaton so that we can handle the two properties independently. After that we treat the two properties and show that L is not in AC^0 , if one of the properties is violated. On the other hand we are able to build an $FO[+]$ formula in case L has these two properties.

3.1 Decomposition of the Automaton

We will split the computation of the automaton in two steps. The first part is the computation of the height profile. The second one can be seen as the regular part of the language. Formally, we will extend the alphabet to include the stack-height information up to a threshold. We will consider a new language of words over the extended alphabet. This language will be regular since the information for the decision which δ_i to use is already coded into the input.

Similar to a regular transducer we define a transduction that appends the height profile to a given word. In the following we fix a visibly alphabet $\hat{\Sigma}$ of Σ .

► **Definition 3** (Height transduction). We let $\tau_m : \Sigma^* \rightarrow \Sigma_m^*$ where $\Sigma_m = \Sigma \times \{0, \dots, m\}$. With τ_m we assign to each position in the word its height up to the threshold m .

$$\begin{aligned} \tau_m(w) = \tau_m(w_1 w_2 \cdots w_n) = \\ (w_1, \Delta_m(\epsilon))(w_2, \Delta_m(w_1)) \cdots (w_i, \Delta_m(w_1 \cdots w_{i-1})) \cdots (w_n, \Delta_m(w_1 \cdots w_{n-1})) \end{aligned}$$

where $\Delta_m(w) = \min(\Delta(w), m)$. The transduction τ_m is only defined on words with a non-negative height profile. We call a word in Σ_m^* *valid* if it is in $F(\tau_m(\Sigma^*))$. We also say that i is the *label* of the letter $(a, i) \in \Sigma_m$.

► **Example 4.** If a is a push letter, and b a pop letter, then $\tau_2(aba) = (a, 0)(a, 1)(b, 2)(a, 1)$.

► **Definition 5.** For an m -VCA $\mathcal{A} = (Q, q_0, F, \hat{\Sigma}, \delta_0, \dots, \delta_m)$ we define $R_{\mathcal{A}} = L(M)$ where M is a finite automaton $M = (Q, q_0, F, \Sigma_m, \delta)$, with $\delta(q, (a, i)) = \delta_i(q, a)$.

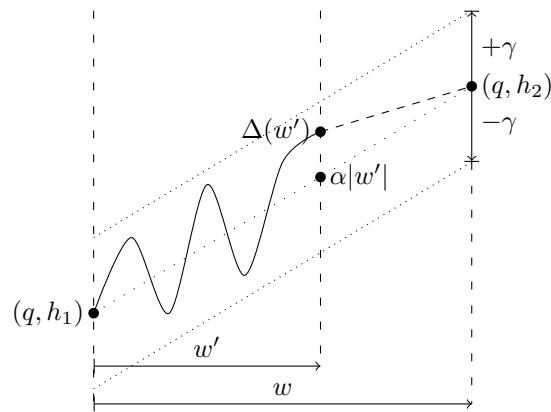
The following statement is obvious:

► **Lemma 6.** If \mathcal{A} is an m -VCA, then $w \in L(\mathcal{A})$ if and only if $\tau_m(w) \in R_{\mathcal{A}}$.

3.2 Height Computation

In this section we investigate in which cases the transduction τ_m is expressible as an $FO[+]$ formula.

For this we need to be able to count the number of call letters minus the number of return letters in the prefix, which is in general TC^0 -hard. Yet, if all the states that are important to the height computation i.e. can occur in loops that have a “fixed slope”, then the computation will be in AC^0 . We fix some m -VCA \mathcal{A} .



■ **Figure 1** Visualization of a state q having fixed slope where α is the actual slope and γ is the corridor. If w' is a prefix of w then $\Delta(w')$ has to stay in the corridor.

► **Definition 7** (fixed slope). We say that a state q has a *fixed slope* if there are numbers $\alpha \in \mathbb{Q}$ and $\gamma \in \mathbb{N}$ so that if for all words $w \in \Sigma^*$ with $(q, h_1) \xrightarrow{w} (q, h_2)$ and $h_1 + \Delta(w') \geq m$ for all prefixes w' of w it holds that:

■ $h_2 = \alpha|w| + h_1$

■ $\alpha|w'| - \gamma \leq \Delta(w') \leq \alpha|w'| + \gamma$ for all prefixes w' of w

We call α the *slope* and γ the *corridor* of q .

Figure 1 shows the concept of this definition.

As we will see, we can think of states with a fixed slope as of those which do not pose a problem when computing the stack height in $\text{FO}[+]$. However there can be states without a fixed slope, which do not make the language too hard for $\text{FO}[+]$, since it is possible that the recognition of a word does not depend on its height profile any more if \mathcal{A} has visited such a state. This happens if from this point the height of the word can never reach height levels below m any more. The next definition captures this idea by some kind of reachability property. Figure 2 visualizes the idea.

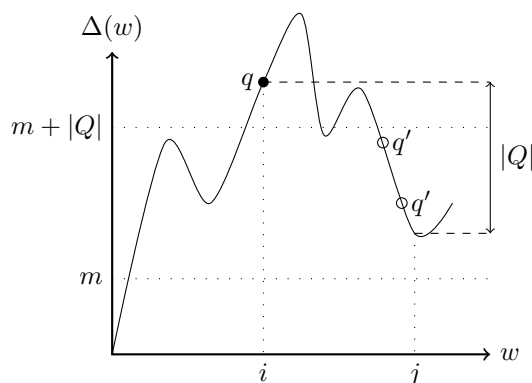
► **Definition 8** (active). A state q is *active* if there is a word $w \in L(\mathcal{A})$ with positions i and j , $i < j$, such that after reading $w_1 \cdots w_i$, \mathcal{A} is in q , $\Delta(w_1 \cdots w_i) > m + |Q|$ and $\Delta(w_1 \cdots w_i) - \Delta(w_1 \cdots w_j) > |Q|$.

Before we prove that for a VCL language L every active state needs to have a fixed slope for L to be in AC^0 , we give an example of a typical case of a hard language.

► **Example 9.** Consider the language $L = \{(a|aba)^n b^n \mid n \in \mathbb{N}\}$. This language is clearly in VCL but there is no m -VCA for L where every active state has a fixed slope. In fact L is not in AC^0 because of this. We can reduce the TC^0 -hard language $\text{EQUALITY} \subseteq \{0, 1\}^*$ to L . Let ϕ and ψ be morphisms with $\phi(0) = aaa$, $\phi(1) = aba$ and $\psi(0) = \psi(1) = bb$. The reduction is $f(w) = \phi(w)\psi(w)$ which is in AC^0 . As one can see, the number of push letters a and pop letters b is in balance iff there are as many 0's as 1's: $|f(w)|_a = 3|w|_0 + 2|w|_1 = |f(w)|_b = |w|_1 + 2|w|_0$. This is equivalent to $|w|_0 = |w|_1$.

In the following lemma and its proof, we generalize the idea of the previous example.

► **Lemma 10.** If a language $L \in \text{VCL}$ is recognized by an m -VCA which has an active state without a fixed slope, then L is not in AC^0 .



■ **Figure 2** The state q is an active state since there is a word $w \in L$ such that q occurs at position i with a height above $m + |Q|$ and there is a word position j with a height difference of $|Q|$ compared to i . By this we know that there is a down loop - in this case through state q' . This is used in lemma 10.

Proof. Let \mathcal{A} be an m -VCA with $L = L(\mathcal{A})$ having a state q which is active but does not have a fixed slope. This implies that there are words besides the empty word forming a loop through q . In fact, there must be two words u and v which loop through q with $|u| = |v|$ and $\Delta(u) > \Delta(v)$. Also there must exist a word $w \in L$ with certain properties: intuitively w has a “high” position, and thus there must be loops going up to and down from that position and one loop goes through q . In the following we assume q to be responsible for an up loop. The case where q is responsible for the down loop can be treated similarly. To be precise, w has the following properties:

- Using position i from the definition of *active*, in the run q appears in position i and w has a height above $m + |Q|$ in i .
- Because of the existence of position j with smaller height (a difference of at least $|Q| + 1$), there must be a down loop. Let q' be a state looped when going down.
- We can partition w into $w = \alpha\beta\gamma$ with $\alpha = w_1 \cdots w_i$ and q' is reached after $\alpha\beta$ the first time, i.e. never in between α and $\alpha\beta$.
- There is a word $x \in \Sigma^*$ looping through q' . We assume that $-\Delta(x) > \Delta(\alpha\beta)$, which is equivalent to $\Delta(\alpha\beta x) < 0$.

It is important to note that between α and $\alpha\beta$ the height never falls below m .

In the following we want to reduce the language $\text{EQUALITY} \subseteq \{0, 1\}^*$, which is in TC^0 but not in AC^0 , to L by using the following pumping approach:

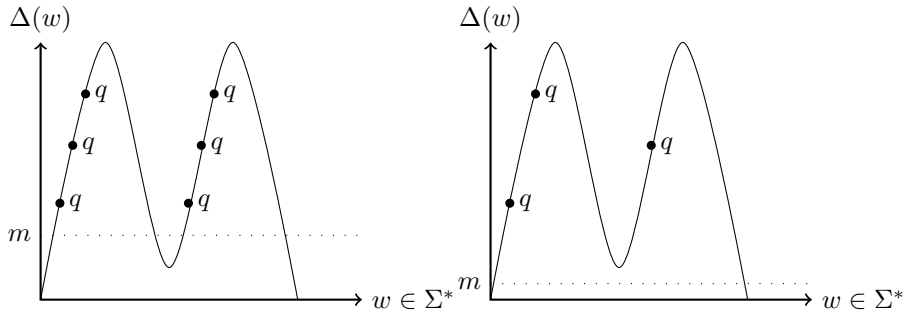
$$\alpha u^{-k\Delta(x)} \beta x^{k\Delta(u)} \gamma \in L$$

for all $k \geq 0$. This is true, since $\Delta(u^{-k\Delta(x)}) = -\Delta(x^{k\Delta(u)})$

We define the following words: $u' = u^{-2\Delta(x)}$, $v' = v^{-2\Delta(x)}$, and $x' = x^{\Delta(u)+\Delta(v)}$.

The key property is $\Delta(u'u') > \Delta(u'v') = -\Delta(x'x') > \Delta(v'v')$ and $|u'| = |v'|$. We define the morphisms $\phi, \psi: \{0, 1\}^* \rightarrow \Sigma^*$ with $\phi(0) = u'$, $\phi(1) = v'$ and $\psi(0) = \psi(1) = x'$ and the map $f: w \mapsto \alpha\phi(w)\beta\psi(w)\gamma$. Since for all morphisms we used it holds that for two words of same length, the images have the same length, f is computable in AC^0 .

For $w \in \{0, 1\}^*$, let \bar{w} be the word, where 0 and 1 are switched, e.g. $w = 0100$, then $\bar{w} = 1011$. We now have $w \in \text{EQUALITY} \Leftrightarrow f(w) \in L \wedge f(\bar{w}) \in L$. This is true since if w has more 0's than 1's (or vice versa) then either $\Delta(f(w))$ or $\Delta(f(\bar{w}))$ is negative (which is



■ **Figure 3** The left example shows an active state which might have a fixed slope (at least the pictured situation is no counter example). Through q we get an up loop and after the word has reached a height level below m , q can be reached again. In the right example however we see that q is active and does not have a fixed slope.

ensured by the condition $-\Delta(x) \geq \Delta(\alpha\beta)$ we had on x) and such words cannot be accepted by some VCA. So f is in fact a reduction and hence $L \notin AC^0$. ◀

In the proof of the previous lemma, we saw that we get a property of the accepted language. If we have two automata for some language and one of them has an active state without a fixed slope and the other one does not then we get a contradiction using a pumping argument.

► **Corollary 11.** *If for some m -VCA \mathcal{A} every active state has a fixed slope then in all VCA for $L(\mathcal{A})$ the active states have fixed slopes.*

This corollary justifies to formulate a property of languages:

► **Definition 12** (simple height behavior). If in some VCA all active states have a fixed slope, we say that the recognized language has simple height behavior.

Figure 3 shows situations being relevant for this property.

We now assume $L(\mathcal{A})$ has simple height behavior. In this case we can compute a sufficient approximation of the matching predicate in $FO[+]$ and in turn use this predicate to define a stack height predicate.

We would like to define the matching predicate in $FO[+]$ that is true for all words w with two positions i, j that are matching positions, i.e. $\Sigma_{\text{call}}(w_i)$ and $\Sigma_{\text{ret}}(w_j)$ and $w_{i+1} \dots w_{j-1}$ is well-matched. Even if a language L is in $FO[+]$, the matching predicate is not necessarily in $FO[+]$. Hence we only approximate the matching predicate from below, i.e. we only have false negatives and recognize all matching pairs of positions that are needed later.

First, we need to define some helper predicates that allow us to verify that the height profile of some factor $w_{i+1} \dots w_j$ has a slope α and stays within a corridor $\pm\gamma$ around this slope and the height profile is above some minimal value h_l .

► **Definition 13.** For every $\alpha \in \mathbb{Q}$ and $\gamma \in \mathbb{N}$ we define a 5-ary predicate $B_{\alpha,\gamma}(x, y, s, t, l)$ such that: $w_{x=i, y=j, s=h_s, t=h_t, l=h_l} \models B_{\alpha,\gamma}(x, y, s, t, l)$ iff

- $\Delta(w_{i+1} \dots w_j) = h_t - h_s$,
- for all $i < k \leq j$ we have $\Delta(w_{i+1} \dots w_k) > h_l - h_s$,
- for all $i < k \leq j$ we have $\alpha|w_{i+1} \dots w_k| - \gamma \leq \Delta(w_{i+1} \dots w_k) \leq \alpha|w_{i+1} \dots w_k| + \gamma$, and
- $\Delta(w_{i+1} \dots w_j) = \alpha|w_{i+1} \dots w_j|$.

► **Lemma 14.** For any $\alpha \in \mathbb{Q}$ and $\gamma \in \mathbb{N}$, the predicate $B_{\alpha,\gamma}(x,y,s,t,l)$ can be defined in $\text{FO}[+]$.

► **Lemma 15.** Given an m -VCA \mathcal{A} , such that $L = L(\mathcal{A})$ has simple height behavior, we can define a binary predicate M in $\text{FO}[+]$ such that for every $w \in \Sigma^*$ and positions i, j of w :

- $w_{x=i,y=j} \models M(x,y)$ implies that the position i matches the position j in w .
- If $w \in L$ and there is a $k > i$ with $\Delta(w_1 \dots w_k) \leq m$ and the position i matches the position j then $w_{x=i,y=j} \models M(x,y)$.

To prove this, we will first define such a predicate for positions i, j that both have stack height larger than $m + |Q|$ and then define it inductively for smaller stack heights.

Fix a word w and i, j . The question is, how to verify that i and j are matching positions. To do so, we need to verify that w_i is a push letter, w_j is a pop letter and the word $z = w_{i+1} \dots w_{j-1}$ is well-matched.

For intuition we first consider a simple case. Assume that $z \in (\Sigma_{\text{call}}^* \Sigma_{\text{ret}}^*)^k$, then we could guess the $2k - 1$ positions x_1, \dots, x_{2k-1} where we switch between push and pop letters. We would verify that we push more on the stack than pop for every prefix of z , i.e. $x_1 - (x_2 - x_1) \geq 0$, $x_1 - (x_2 - x_1) + (x_3 - x_2) - (x_4 - x_3) \geq 0, \dots$ Finally we need to test if the sum of the length of the intervals with push letters is equal to the sum of the length of the intervals with pop letters.

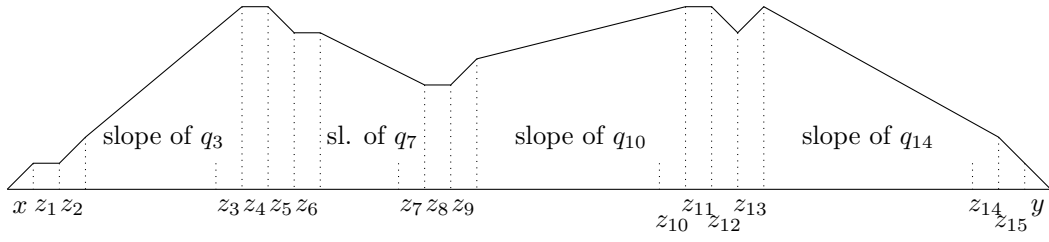
Unfortunately we cannot assume that there is a constant k such that all words z are of this form. But we have a similar form for each factor z where we need to test if it is a well-matched word if the whole word w belongs to L . Assume here that w is well-matched. Since $w \in L$ there is an accepting run for w , hence every state occurring in an interval of height at least $m + |Q|$ is an active state, and every active state has a fixed slope. Let q be an active state that appears more than once in the run of w inside of z . Let k, l be the first and last position inside z where the state q occurs. Then the height difference $\Delta(z_k \dots z_l)$ is $\alpha_q \cdot (l - k)$, where α_q is the slope of the state q . Since there are only finitely many states, we can split z into a fixed number of intervals such that in each interval the stack height is “nearly linear” increasing or decreasing or being constant. If we cannot find such a fixed number of intervals then w cannot be in L . The following lemma will formalize this idea.

► **Lemma 16.** Given a language $L \subseteq \Sigma^*$ in VCL with simple height behavior, we can define a binary predicate $M_{>m+|Q|}$ in $\text{FO}[+]$ such that for every $w \in \Sigma^*$ and positions i, j of w :

- $w_{x=i,y=j} \models M_{>m+|Q|}(x,y)$ implies the position i matches the position j in w .
- If $w \in L$ and $\Delta(w_1 \dots w_i) > m + |Q|$ and there is a $k > i$ with $\Delta(w_1 \dots w_k) \leq m$ and the position i matches the position j then $w_{x=i,y=j} \models M_{>m+|Q|}(x,y)$.

Proof. We will first give the intuition on how to define the predicate $M_{>m+|Q|}$. Then we will show if $M_{>m+|Q|}$ is true that the positions i, j are actually matching positions, and finally that for $w \in L$ and i, j matching positions with stack height at least m , the predicate is true.

Let L be accepted by some m -VCA \mathcal{A} . Following the idea above our formula will need to guess at most $n = |Q| + 1$ points z_0, \dots, z_n and the “slope” between these points represented by a state q_1, \dots, q_n . Finally we guess the stack-height h_0, \dots, h_n at the points z_0, \dots, z_n relative to $\Delta(w_1 \dots w_{i-1})$.



■ **Figure 4** Example for positions of z_1, \dots, z_{15} fitting to the input word.

$$\begin{aligned}
 M_{>m+|Q|}(x, y) = & x < y \wedge \Sigma_{\text{call}}(x) \wedge \Sigma_{\text{ret}}(y) \wedge \\
 & \bigvee_{(q_1, \dots, q_n) \in Q^n} \exists z_0 \dots \exists z_n \exists h_1 \dots \exists h_n \\
 & z_0 = x \wedge z_n = y - 1 \wedge h_0 = h_n \\
 & \bigwedge_{i=0}^{n-1} z_i \leq z_{i+1} \wedge A_{q_{i+1}}(z_i, z_{i+1}, h_i, h_{i+1}, h_0)
 \end{aligned}$$

The formulas A are defined below.

The idea is that the formula $A_{q_{i+1}}$ needs to verify that the guess was correct in the sense that the slope in interval $z_i + 1 \dots z_{i+1}$ is equal to the slope of q_{i+1} having a stack height difference of $h_{i+1} - h_i$. Note that we do not have to guess the state of the accepting run, but only *some* state with the same slope. In the case of an interval length 0 or 1 the formula $A_{q_{i+1}}$ will ignore the state q_{i+1} and directly check if the height difference is zero respectively corresponds to the single letter. See figure 4 as a sketch.

Finally we define the formula $A_{q_{i+1}}(z_i, z_{i+1}, h_i, h_{i+1}, h_0)$:

- If $z_i = z_{i+1} \wedge h_i = h_{i+1} \wedge h_i > h_0$ then the formula is true.
- If $z_i + 1 = z_{i+1}$ the formula is true if $h_i = h_{i+1} - 1 \wedge h_i > h_0$ (resp. $h_i = h_{i+1} \wedge h_i > h_0$ or $h_i = h_{i+1} + 1 \wedge h_{i+1} > h_0$) and $\Sigma_{\text{call}}(z_{i+1})$ (resp. $\Sigma_{\text{int}}(z_{i+1})$ or $\Sigma_{\text{ret}}(z_{i+1})$).
- In the case that $z_i + 1 < z_{i+1}$ we use the predicate $B_{\alpha, \gamma}(z_i, z_{i+1}, h_i, h_{i+1}, h_0)$ where α is the slope and γ the corridor of q .
- Otherwise the predicate is false.

Finally, we need to verify that with our definition of $M_{>m+|Q|}$ we satisfy the conditions of the lemma. The first condition is certainly true as guessing and verifying the stack height always is correct if all A predicates are true. For the second condition we need to show that it satisfies to guess n “turning points”. We only consider the case $w \in L$, hence there is an accepting run of w and the sequence of states within the positions of x and y since $w \in L$ and the height profile will be below m at some point in the suffix all states of the accepting run are active states and hence have fixed slope. If the distance of x and y is less than n we could simply guess all states in this sequence. But their distance might be larger, hence we compress this sequence. For a state with a fixed slope the whole interval between the first and last occurrence should have a fixed slope and hence can be recognized by a single A predicate. So in the compressed sequence states with a fixed slope will occur only once. Hence it satisfies to guess n “turning points”. ◀

At this point we have defined the predicate $M_{>m+|Q|}$. We can now define predicates M_k for height k under the assumption we have defined M_{k+1} already. This way we inductively get M_0 .

► **Example 17.** Consider $L = \{a(a^n b^n)^* b \mid n \in \mathbb{N}\}$. If $w \in L$ then the first and the last letter of w match but the number of intervals can be arbitrary large. There is a 2 – VCA for L but no 1 – VCA. This reflects in the matching predicates.

Proof of Lemma 15. By the previous lemma we have a predicate $M_{>_{m+|Q|}}$. Fix a word w . Any two positions i, j are matching positions if and only if $w_i \in \Sigma_{\text{call}}, w_j \in \Sigma_{\text{ret}}$, and every push letter w_s with $i < s < j$ is matched to a pop letter w_t with $i < s < t < j$. Note that if i, j are at stack height h then s, t will be always at stack height $> h$, hence we can test if i, j are matched testing matching in between only for words of larger stack height.

$$\begin{aligned} M_k(x, y) &= M_{k+1}(x, y) \\ &\vee x < y \wedge \Sigma_{\text{call}}(x) \wedge \Sigma_{\text{ret}}(y) \\ &\wedge \forall z(x < z < y \wedge \neg \Sigma_{\text{int}}(z)) \Rightarrow (\exists z' x < z' < y \wedge (M_{k+1}(z, z') \vee M_{k+1}(z', z))) \end{aligned}$$

Note that the first line in the definition of M_k , ensures that the power to recognize a matching increases from M_{k+1} to M_k . This way M_0 will be true for all matchings which can occur in a word in L . Hence $M(x, y) = M_0(x, y)$. ◀

A position is of stack height 0 if all call-positions in the prefix have matching return-positions in the prefix. Similar a position is of stack height i if there are i call-positions in the prefix with push letters and all positions are matched by positions in the same interval generated by those i positions.

► **Lemma 18.** For every constant $0 \leq j < m$, we can define a monadic predicate $H_k(x)$ in $\text{FO}[+]$ such that:

- $w_{x=i} \models H_j(x)$ then $\Delta(w_1 \dots w_{i-1}) = k$ for arbitrary $w \in \Sigma^*$.
- $w_{x=i} \models H_j(x)$ iff $\Delta(w_1 \dots w_{i-1}) = k$ for all $w \in L$.

Proof. A position i has stack height k iff all but k call letters in the prefix $w_1 \dots w_{i-1}$ match. It is obvious that this can be defined in $\text{FO}[+]$ using our matching predicates.

The matching predicates might have false negatives resulting in false negatives of H_k . But in the case of $w \in L$ and the case that the height at position i is $k < m$ the matching predicate is exact on the prefix $w_1 \dots w_{i-1}$ and hence the height is correctly presented by H_k . ◀

We let $H_{\geq m} = \neg \bigvee_{k=0}^{m-1} H_k$ be the negation of these predicates. Hence for $w \notin L$ the predicate might have false-positives, i.e., the predicate might suggest a stack-height greater or equal to m while in fact it is less than m .

3.3 The Regular Part

In this section we will show a second property which in addition to the property of the previous section - simple height behavior - is sufficient to characterize the visibly counter languages in AC^0 . This second property concerns $R_{\mathcal{A}}$. If $R_{\mathcal{A}}$ is in $\text{FO}[Reg]$ and if L has simple height behavior, then we can build an $\text{FO}[+]$ formula for L . Unfortunately there are cases where $R_{\mathcal{A}}$ is not in $\text{FO}[Reg]$, but still L is in $\text{FO}[+]$. The problem here is that there can be words which are witness for $\eta_{R_{\mathcal{A}}}$ not being quasi-aperiodic, but which are not images of τ_m ; then we cannot deduct that L is not in AC^0 .

First we introduce a normal-form on visibly counter automata which concerns loops. In the following definition we call a state q *dead* if there is no $w = w_1 w_2 \in L$, so that $(q_0, 0) \xrightarrow{w_1} (q, h) \xrightarrow{w_2} (q', h')$ with $h \geq m$.

► **Definition 19.** An m -VCA \mathcal{A} is called *loop-normal* if for all $x, y \in \Sigma^*$ with $\Delta(xy_1 \cdots y_k) \geq m$ for $0 \leq k \leq |y|$ and $(q_0, 0) \xrightarrow{x} (q, h_1) \xrightarrow{y} (q, h_2)$, $q \in Q$ then either q is a dead state or there is $z \in \Sigma^*$ with $xyz \in L(\mathcal{A})$ and one of the following is true, depending on $\Delta(y)$:

- If $\Delta(y) > 0$, then there is a partition of z into $z = z_1 z_3$ and a word $z_2 \in \Sigma^*$ so that for all $i \geq 0$ we have that $xyy^i z_1 z_2^i z_3 \in L(\mathcal{A})$.
- If $\Delta(y) < 0$, then there is a partition of x into $x = x_1 x_3$ and a word $x_2 \in \Sigma^*$ so that for all $i \geq 0$ we have that $x_1 x_2^i x_3 y y^i z \in L(\mathcal{A})$.
- If $\Delta(y) = 0$, then $xy^i z \in L$ for all $i \geq 0$.

We also require that $\delta_i = \delta_m$ for $m - |Q| < i < m$.

The idea of this definition is, that if a prefix reaches a state in \mathcal{A} that can be completed to a word in L then no matter how many loops through this state are appended, the word can still be completed to a word in L .

► **Lemma 20.** *For every m -VCA \mathcal{A} recognizing a language L there is a loop-normal m' -VCA \mathcal{A}' recognizing L .*

In this proof \mathcal{A} is equipped with a modulo $|Q|!$ counter coded into the states. This way the looping word y (let us say $\Delta(y) > 0$ here) has a height which is a multiple of $|Q|!$. Using this, one can find the corresponding down looping word z'_2 . Then $\Delta(y)$ is a multiple of $\Delta(z'_2)$ and so one can construct z_2 from z'_2 with $\Delta(y) = -\Delta(z_2)$.

The intersection of the regular languages and AC^0 is characterized by the quasi-aperiodicity of the syntactic morphism. A regular language R is in AC^0 iff $\eta_R(\Sigma^t)$ has only trivial groups for all t . If $R \notin AC^0$ then there exist words of equal length spanning a group. In this case we can use those words to build an AC^0 reduction from an ACC_k^0 -hard language to R . The same we want to do with $R_{\mathcal{A}}$. Unfortunately there can be words of equal length spanning a group in the syntactic monoid of $R_{\mathcal{A}}$ but still $L = L(\mathcal{A})$ is in AC^0 . The reason is that actually we are only interested in $R_{\mathcal{A}} \cap \tau_m(\Sigma^*)$, i.e. in a restricted set of inputs. This intersection however is not regular any more.

In the following we use $\tau_m(\Sigma^*)$ which is the set of *restricted inputs* we are interested in. It contains prefixes of labeled well-matched words. The set $F(\tau_m(\Sigma^*))$ is the set of factors of words in $\tau_m(\Sigma^*)$. Keep in mind that τ_m is not defined for inputs with negative height-profile, e.g. $\tau_m(ba)$ is undefined if a is a push and b a pop letter.

► **Lemma 21.** *If \mathcal{A} is a loop-normal m -VCA then if there is a number $t > 0$ and a set $G \subseteq \Sigma_m^t$ with $G^* \subseteq F(\tau_m(\Sigma^*))$ so that the set $\eta_{R_{\mathcal{A}}}(G)$ contains a non-trivial group, then $L \notin AC^0$.*

This is proved by reducing MOD_k for some k to L which is possible if the property is met. If so, the words generating a group can be appended after each other so that they still are a valid input.

► **Lemma 22.** *If for all $t > 0$ and for all G with $G \subseteq \Sigma_m^t$ and $G^* \subseteq F(\tau_m(\Sigma^*))$ the set $\eta_{R_{\mathcal{A}}}(G)$ does not contain a non-trivial group, then there is an FO[Reg] formula ϕ with*

$$L(\phi) \cap \tau_m(\Sigma^*) = R_{\mathcal{A}} \cap \tau_m(\Sigma^*).$$

The proof is based on the proof in [19] which constructs an FO[Reg] formula for quasi-aperiodic languages. We have a weaker property than quasi-aperiodicity, so we have to treat groups which might occur. We can show that if our weaker property is met, occurring groups can be eliminated without changing the language under the restricted inputs.

4 Results

If we combine our statements from the previous section, we get the following results.

- **Theorem 23.** *For a loop-normal m -VCA \mathcal{A} , $L = L(\mathcal{A})$ is in AC^0 if and only if*
- *$L(\mathcal{A})$ has simple height behavior and*
 - *for all $t > 0$ and for all $G \subseteq \Sigma_m^t$ with $G^* \subseteq F(\tau_m(\Sigma^*))$ the set $\eta_{R_{\mathcal{A}}}(G)$ does not contain a non-trivial group.*

Proof. We already proved the direction from left to right with lemmas 10 and 21.

If we have \mathcal{A} where all active states have a fixed slope and the formula ϕ from lemma 22, we can build an $FO[+]$ formula for L : Begin with the $FO[Reg]$ formula ϕ . This formula operates on the alphabet Σ_m and uses letter predicates $Q_{(a,k)}x$. We replace them by $(Q_a(x) \wedge H_k(x))$ if $k < m$ and by $(Q_a(x) \wedge H_{\geq m}(x))$ if $k = m$. The resulting formula is ϕ' and operates over Σ .

If a word w is in L then $w \models \phi'$. The only thing we have to take care of are false positives in the $H_{\geq m}$ predicate which we mentioned earlier in the paper. A false positive here can only occur if there is a non-active state q without fixed slope. This state loops up and never comes down to m again (otherwise it would be active). So if we have a word which visits q but reaches a height smaller than m after q , it cannot be in L . But then there is a word $w' = w_1xw_2$, where w_1 brings \mathcal{A} in the state q , x loops through q , $\Delta(x) > m$ and $w = w_1w_2$. On w' , $H_{\geq m}$ will not have a false positive, since after q the word is always above m . Also w' cannot be in L and if $w' \notin L$ then also $w \notin L$. Hence if $w \notin L$ then $w \not\models \phi'$. ◀

In the proof of the previous theorem we constructed an $FO[+]$ formula for every VCL in AC^0 . We can state our main result in different ways:

- **Corollary 24.** *The following statements are true:*
- $VCL \cap FO[arb] \subseteq FO[+]$.
 - $VCL \cap AC^0 \subseteq FO[+]$.
 - $VCL \cap AC^0 \subseteq DLOGTIME - uniformAC^0$.

It is easy to verify that all the properties of the previous lemma are decidable. Hence we have an effective characterization.

- **Corollary 25.** *Given some visibly counter automaton \mathcal{A} , it is decidable whether $L(\mathcal{A})$ lies in AC^0 , resp. in $FO[+]$.*

Proof. Given \mathcal{A} , we have to check for all states of \mathcal{A} if they are active and if they have a fixed slope. If there is an active state without fixed slope then $L(\mathcal{A}) \notin AC^0$.

For deciding if a state $q \in Q$ has fixed slope, make a list of all words looping through q up to length $|Q|$. Then calculate the slope of all words. They are all equal iff q has a fixed slope.

For deciding if a state q is active, check if there is a word x with $\Delta(x) > m + |Q|$ which brings \mathcal{A} in state q . This is as hard as the membership problem for VCL. If such an x exists then check for all words $y \in \Sigma^*$ up to length $\Delta(x)|Q|$ if $xy \in L(\mathcal{A})$ and if there is a prefix y' of y with $\Delta(x) - \Delta(y') > |Q|$. If such a y exists then q is active.

Finally we have to decide our modified quasi-aperiodicity property. First of all, t can be bounded by a constant relative to the syntactic monoid of $R_{\mathcal{A}}$, where \mathcal{A} is a loop-deterministic automaton for L . Then there are only finitely many sets G to consider. The requirement of $G^* \subseteq F(\tau_m(\Sigma^*))$ is equivalent to $GG \subseteq F(\tau_m(\Sigma^*))$ which then is also decidable. ◀

5 Discussion

Algebraic methods usable for languages up to now mainly pertain to finite monoids, i.e. to regular languages. We see our results as a step towards the further application of algebraic methods in the non-regular case. A natural continuation of this line of research would be an algebraic theory for visible pushdown languages and their subclasses. A promising approach to go here might be the use of forest algebras [6].

The characterization of the regular languages in AC^0 as the class $FO[Reg]$ used the notion of *quasi-aperiodic* regular sets which is of an algebraic nature. Our result is oriented in this direction, but is not as algebraic. It still leaves open to characterize exactly the set of all visible counter languages contained in AC^0 in terms of logic.

In [14] the notion of dense completeness has been introduced. A family of formal languages \mathcal{F} is said to be densely complete in a complexity class \mathcal{C} if both $\mathcal{F} \subset \mathcal{C}$ and for each $C \in \mathcal{C}$ there exists a $F \in \mathcal{F}$ so that $C \leq F$ and $F \leq C$, i.e.: F and C have the same complexity. While the context-free languages turn out to be densely complete in the class SAC^1 , the regular languages are not densely complete in the class NC^1 . As a consequence of our result we are able to show unconditionally that the visible one-counter languages, which are contained in NC^1 , are not densely complete in NC^1 . Up to now, dense families of formal languages are known for the non-deterministic classes, $NSPACE(\log n)$, SAC^1 , and NP , only.

In our work we explored the intersection of a formal language class and a circuit-based complexity class. Aside from the pair AC^0 and VCL , there are some other combinations worth being investigated using our methods.

References

- 1 Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *STOC*, pages 202–211. ACM, 2004.
- 2 Vince Bárány, Christof Löding, and Olivier Serre. Regularity problems for visibly pushdown languages. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23–25, 2006, Proceedings*, volume 3884 of *Lecture Notes in Computer Science*, pages 420–431. Springer, 2006.
- 3 David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1 . *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.
- 4 David A. Mix Barrington, Kevin J. Compton, Howard Straubing, and Denis Thérien. Regular languages in NC^1 . *J. Comput. Syst. Sci.*, 44(3):478–499, 1992.
- 5 David A. Mix Barrington and Denis Thérien. Finite monoids and the fine structure of NC^1 . *J. ACM*, 35(4):941–952, 1988.
- 6 Mikolaj Bojańczyk and Igor Walukiewicz. Forest algebras, 2007.
- 7 Ashok K. Chandra, Steven Fortune, and Richard J. Lipton. Unbounded fan-in circuits and associative functions. *J. Comput. Syst. Sci.*, 30(2):222–234, 1985.
- 8 Patrick W. Dymond. Input-driven languages are in $\log n$ depth. *Inf. Process. Lett.*, 26(5):247–250, 1988.
- 9 Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, circuits, and the polynomial-time hierarchy. In *FOCS*, pages 260–270, 1981.
- 10 Yuri Gurevich and Harry R. Lewis. A logic for constant-depth circuits. *Information and Control*, 61(1):65–74, 1984.
- 11 Johan Håstad. Almost optimal lower bounds for small depth circuits. In *STOC*, pages 6–20. ACM, 1986.

- 12 Neil Immerman. Languages that capture complexity classes. *SIAM J. Comput.*, 16(4):760–778, 1987.
- 13 Neil Immerman. *Descriptive Complexity*. Springer, New York, 1999.
- 14 Andreas Krebs and Klaus-Jörn Lange. Dense completeness. In Hsu-Chun Yen and Oscar H. Ibarra, editors, *Developments in Language Theory - 16th International Conference, DLT 2012, Taipei, Taiwan, August 14-17, 2012. Proceedings*, volume 7410 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 2012.
- 15 Pierre McKenzie, Michael Thomas, and Heribert Vollmer. Extensional uniformity for boolean circuits. *SIAM J. Comput.*, 39(7):3186–3206, 2010.
- 16 Robert McNaughton and Seymour Papert. *Counter-free automata. With an appendix by William Henneman*. Research Monograph No.65. Cambridge, Massachusetts, and London, England: The M. I. T. Press. XIX, 163 p., 1971.
- 17 Kurt Mehlhorn. Pebbling mountain ranges and its application to defl-recognition. In Jaco de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming*, volume 85 of *Lecture Notes in Computer Science*, pages 422–435. Springer Berlin Heidelberg, 1980.
- 18 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.
- 19 Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhäuser, Boston, 1994.
- 20 H. Venkateswaran. Properties that characterize LOGCFL. *J. Comput. Syst. Sci.*, 43(2):380–404, 1991.
- 21 Heribert Vollmer. *Introduction to circuit complexity - a uniform approach*. Texts in theoretical computer science. Springer, 1999.

Optimal Decremental Connectivity in Planar Graphs*

Jakub Łącki¹ and Piotr Sankowski²

- 1 University of Warsaw
Warsaw, Poland
j.lacki@mimuw.edu.pl
- 2 University of Warsaw
Warsaw, Poland
sank@mimuw.edu.pl

Abstract

We show an algorithm for dynamic maintenance of connectivity information in an undirected planar graph subject to edge deletions. Our algorithm may answer connectivity queries of the form ‘Are vertices u and v connected with a path?’ in constant time. The queries can be intermixed with any sequence of edge deletions, and the algorithm handles all updates in $O(n)$ time. This results improves over previously known $O(n \log n)$ time algorithm.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.2 Graph Theory

Keywords and phrases decremental connectivity, planar graphs, dynamic connectivity, algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.608

1 Introduction

The *dynamic graph connectivity* problem consists in maintaining connectivity information about an undirected graph, which is undergoing modifications. Typically, the modifications are additions or removals of edges or vertices. In this paper we focus on the problems in which each modification adds or removes a single edge. These problems have three variants: in the *incremental* version, edges can only be added to the graph, in the *decremental* one the edges may only be removed, whereas in the *fully dynamic* version both edge insertions and deletions are allowed. Graph updates are intermixed with a set of connectivity queries of the form ‘Are vertices u and w in the same connected component?’

We consider the decremental connectivity problem for planar graphs, and show an algorithm that may answer connectivity queries in constant time and process any sequence of edge deletions in $O(n)$ time. The previously known best running time of $O(n \log n)$ was obtained by using the fully dynamic algorithm. We assume word-RAM model with standard operations.

* Jakub Łącki is a recipient of the Google Europe Fellowship in Graph Algorithms, and this research is supported in part by this Google Fellowship. Piotr Sankowski is partially supported by ERC grant PAAI no. 259515, NCN grant "Efficient planar graph algorithms" and the Foundation for Polish Science.

1.1 Prior work

It is easy to see that incremental graph connectivity can be solved using an algorithm for the union-find problem. It follows from the result of Tarjan [16] that a sequence of t edge insertions and t queries can be handled in $O(t\alpha(t))$ time, where $\alpha(t)$ is the extremely slowly growing inverse Ackermann function.

There has been a long line of research considering the fully dynamic connectivity in general graphs [6, 3, 8, 10, 19, 11, 21]. The best currently known algorithms have polylogarithmic update and query time. Thorup [19] has shown a randomized Monte Carlo algorithm with $O(\log n(\log \log n)^3)$ amortized update and $O(\log n/\log \log \log n)$ query time.¹ An algorithm by Wulff-Nilsen [21] handles updates in slightly worse $O(\log^2 n/\log \log n)$ amortized time, but it is deterministic and answers queries in $O(\log n/\log \log n)$ time. The best algorithm with worst-case update guarantee is a randomized algorithm by Kapron, King and Mountjoy [11], which processes updates in $O(\log^5 n)$ time and answers queries in $O(\log n/\log \log n)$ time. However, if we require a deterministic algorithm with worst-case running time guarantee, nothing better than a $O(\sqrt{n})$ time algorithm is known [6, 3, 2].

For the decremental variant, Thorup [18] has shown a randomized algorithm, which can process any sequence of edge deletions in $O(m \log(n^2/m) + n(\log n)^3(\log \log n)^2)$ time and answers queries in constant time. Here, m is the initial number of edges in the graph. If $m = \Theta(n^2)$, the update time is $O(m)$, whereas for $m = \Omega(n(\log n \log \log n)^2)$ it is $O(m \log n)$.

The picture is much simpler in case of planar graphs. Eppstein et. al [5] gave a fully dynamic algorithm which handles updates and queries in $O(\log n)$ amortized time, but requires that the graph embedding remains fixed. For the general case (i.e., when the embedding may change) Eppstein et. al [4] gave an algorithm with $O(\log^2 n)$ worst-case update time and $O(\log n)$ query time.

In planar graphs, the best known solution for the incremental connectivity problem is the union-find algorithm. However, for the special case when the final resulting planar graph is given upfront, and the edge insertions and queries are given later in a dynamic fashion Gustedt [7] has shown an $O(n)$ time algorithm. On the other hand, for the decremental problem nothing better than a direct application of the fully dynamic algorithm is known. This is different from both general graphs and trees, where the decremental connectivity problems have better solutions than what could be achieved by a simple application of their fully dynamic counterparts. In case of general graphs, the best total update time was $O(m \log n)$ [18] (except for very sparse graphs, including planar graphs), compared to $O(m \log n(\log \log n)^3)$ time for the fully dynamic variant. For trees, only $O(n)$ time is necessary to perform all updates in the decremental scenario [1], while in the fully dynamic case one can use dynamic trees and obtain $O(\log n)$ worst case update time.

There has also been some progress in obtaining lower bounds for dynamic connectivity problems. Tarjan and La Poutré [17, 15] have shown that incremental connectivity requires $\Omega(\alpha(n))$ time per operation on a pointer machine. Henzinger and Fredman [9] considered the fully dynamic problem and RAM model and obtained a lower bound of $\Omega(\log n/\log \log n)$, which also works for plane graphs. This was improved by Demaine and Pătraşcu [14] to a lower bound of $\Omega(\log n)$ in cell-probe model. The lower bound holds also for plane graphs.

¹ Throughout the paper we use n and m to denote, respectively, the number of vertices and the number of edges in the graph.

1.2 Our results

We show an algorithm for the decremental connectivity problem in planar graphs, which processes any sequence of edge deletions in $O(n)$ time and answers queries in constant time. This improves over the previous bound of $O(n \log n)$, which can be obtained by applying the fully dynamic algorithm by Eppstein [5], and matches the running time of decremental connectivity on trees [1].

In fact, we present a $O(n)$ time reduction from the decremental connectivity problem to a collection of incremental problems in graphs of total size $O(n)$. These incremental problems have a specific structure: the set of allowed union operations forms a planar graph and is given in advance. As shown by Gustedt [7], such a problem can be solved in linear time. Our result shows that in terms of total update time, the decremental connectivity problem in planar graphs is definitely not harder than the incremental one. It should be noted that the union-find algorithm can process any sequence of k query or update operations in $O(k\alpha(n))$ time, while in our algorithm we are only able to bound the time to process any sequence of edge deletions.

Moreover, since fully dynamic connectivity has a lower bound of $\Omega(\log n)$ (even in plane graphs) shown by Demaine and Pătraşcu [14], our results imply that in planar graphs decremental connectivity is strictly easier than the fully dynamic one. We suspect that the same holds for general graphs, and we conjecture that it is possible to break the $\Omega(\log n)$ bound for a single operation of a decremental connectivity algorithm, or the $\Omega(m \log n)$ bound for processing a sequence of m edge deletions.

Our algorithm, unlike the majority of algorithms for maintaining connectivity, does not maintain the spanning tree of the current graph. As a result, it does not have to search for a replacement edge when an edge from the spanning tree is deleted. Our approach is based on a novel and very simple approach for detecting bridges, which alone gives $O(n \log n)$ total update time. We use the fact that a deletion of edge uw in the graph causes some connected component to split if both sides of uw belong to the same face. This condition can in turn be verified by solving an incremental connectivity problem in the dual graph. When we detect a deletion that splits a connected component, we start two parallel DFS searches from u and w to identify the *smaller* of the two new components. Once the first search finishes, the other one is stopped. A simple argument shows that this algorithm runs in $O(n \log n)$ time.

We then show that the DFS searches can be speeded up using an r -division, that is a decomposition of a planar graph into subgraphs of size at most $r = \log^2 n$. This gives an algorithm running in $O(n \log \log n)$ time. For further illustration of this idea we show how to apply it twice in order to obtain an $O(n \log \log \log n)$ time algorithm. Then, we observe that the $O(n \log \log \log n)$ time algorithm reduces the problem of maintaining connectivity in the input graph to maintaining connectivity in a number of graphs of size at most $O(\log^2 \log n)$. The number of such graphs is so small that we can simply precompute the answers for all of them and use these precomputed answers to obtain the main result of the paper. The preprocessing of all graphs of bounded size is again an idea that, to the best of our knowledge, has never been previously used for designing dynamic graph algorithms.

1.3 Organization of the paper

In Section 2 we introduce notation and recall some of the concepts that we later use. The following sections describe our algorithm. We start with the description of the simple $O(n \log n)$ time algorithm in Section 3, and then in every section we show an improvement in the running time.

In Section 4 we show how to use r -division to get an $O(n \log \log n)$ algorithm. Section 5, shows how to improve the reduction, so that it can be used more than once, which results in an $O(n \log \log \log n)$ time algorithm. Finally, in Section 6 we show how to solve the decremental connectivity in optimal time for graphs of size $O(\log^2 \log n)$, after initial preprocessing. This, combined with the reduction applied twice, gives the main result of the paper.

2 Preliminaries

Let $G = (V, E)$ be an undirected, unweighted planar graph, and $n = |V|$. By $V(G)$, $E(G)$ and $F(G)$ we denote the sets of vertices, edges and faces of G . The Euler's formula states that $|V(G)| - |E(G)| + |F(G)| = |CC(G)| + 1$, where $CC(G)$ is the set of connected components of G . The *dual graph* G^* is constructed from G by embedding a single vertex in every face of G and connecting the vertices in adjacent faces of G . Note that if two faces f_1, f_2 share more than one edge, G^* has multiple edges between f_1 and f_2 .

In the paper we deal with algorithms that maintain the connectivity information about a graph G subject to edge deletions. By the total running time we denote the total time of handling deletions of all edges from the graph.

The identifier of a connected component (henceforth denoted *cc-identifier*) is a value assigned to a vertex $v \in V$, which uniquely identifies the connected component of G , i.e., two vertices have the same cc-identifier if and only if they belong to the same connected component. The cc-identifiers change as the edges are deleted, and they may not be preserved after edge deletion. An algorithm maintains cc-identifiers *explicitly* if after every deletion it returns the list of changes to the cc-identifiers. We assume that cc-identifiers are integers that require $\log n + O(1)$ bits.² Note that an algorithm which maintains cc-identifiers explicitly can be simply turned into an algorithm with constant query time. In order to answer a query regarding two vertices, it suffices to compare the cc-identifiers of the two vertices. By definition, the vertices are in the same connected component if and only if their cc-identifiers are equal.

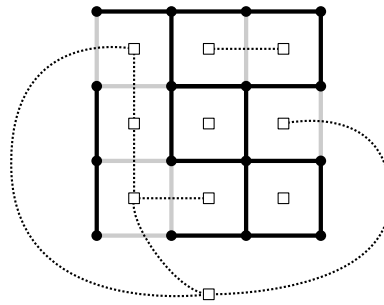
Let us now recall the notion of an r -division. A *region* R is an edge-induced subgraph of G . A *boundary vertex* of a region R is a vertex $v \in V(R)$ that is adjacent to an edge $e \notin E(R)$. We denote the set of boundary vertices of a region R by $\partial(R)$. An r -division \mathcal{P} of G is a partition of G into $O(n/r)$ edge-disjoint regions (which might share vertices), such that each region contains at most r vertices and $O(\sqrt{r})$ boundary vertices. The set of boundary vertices of a division \mathcal{P} , denoted $\partial(\mathcal{P})$ is the union of the sets $\partial(R)$ over all regions R of \mathcal{P} . Note that $|\partial(\mathcal{P})| = O(n/\sqrt{r})$.

► **Lemma 1** ([13, 20]). *Let $G = (V, E)$ be an n -vertex biconnected triangulated planar graph and $1 \leq r \leq n$. An r -division of G can be constructed in $O(n)$ time.*

Let G be a planar graph. In the preprocessing phase of our algorithms, we build an r -division of G . This r -division will be updated in a natural way, as edges are deleted from G . Namely, when an edge is deleted from the graph, we update its r -division by deleting the corresponding edge. However, if we strictly follow the definition, what we obtain may no longer be an r -division.

For that reason, we loosen the definition of an r -division, so that it includes the divisions obtained by deleting edges. Consider an r -division \mathcal{P} built for a graph G . Moreover, let G'

² Throughout this paper, $\log n$ denotes binary logarithm.



■ **Figure 1** The graphs from the proof of Lemma 3. Edges of G are drawn with solid black lines, whereas the gray lines depict edges that have been deleted from G . The small squares are vertices of D_G , and the dotted lines are edges of D_G .

be a graph obtained from G by deleting edges, and let \mathcal{P}' be the r -division \mathcal{P} updated in the following way. Let R be a region of \mathcal{P} . Then, we define the graph R' in \mathcal{P} obtained by removing edges from R to be a region of \mathcal{P}' , although it may no longer be an edge-induced subgraph of G' , e.g., it may contain isolated vertices. Similarly, we define the set of boundary vertices of \mathcal{P}' to be the set of boundary vertices of \mathcal{P} . Again, according to this definition, a boundary vertex v of \mathcal{P}' may be incident to edges of a single region (because the edges incident to v that belonged to other regions have been deleted). In the following, we say that \mathcal{P}' is an r -division of G' .

Since Lemma 1 requires the graph to be biconnected and triangulated, in order to obtain an r -division for a graph which does not have these properties, we first add edges to G to make it biconnected and triangulated, then compute the r -division of G , and then delete the added edges both from G and its division.

Without loss of generality, we can assume that each vertex $v \in V$ has degree at most 3. This can be assured by triangulating the dual graph in the very beginning. In particular, this assures that each vertex belongs to a constant number of regions in an r -division.

3 $O(n \log n)$ Time Algorithm

Let G be a planar graph subject to edge deletions. We call an edge deletion *critical* if and only if it increases the number of components of G , i.e., the deleted edge is a bridge in G . We first show a dynamic algorithm that for every edge deletion decides, whether it is critical. It is based on a simple relation between the graph G and its dual.

► **Lemma 2.** *Let G be a planar graph subject to edge deletions. There exists an algorithm that for each edge deletion decides whether it is critical. It runs in $O(n)$ total time.*

Proof. The intuition behind the proof is as follows. We maintain the number of faces in G . In order to do that, when an edge e is deleted, we simply merge faces on both sides of e (if they are different from each other). This can be implemented using union-find data structure on the vertices of the dual graph.

More formally, we build and maintain a graph D_G . Initially, this is a graph consisting of vertices of G^* (faces of G). When an edge is deleted from G , we add its dual edge to D_G (see Fig. 1). Clearly, the connected components of D_G are exactly the faces of G . Since edges are only added to D_G , we can easily maintain the number of connected components in D_G with a union-find data structure.

This allows us to detect critical deletions in G . After every edge deletion, we know the number of edges and vertices of G . Moreover, we know that the number of faces of G is equal to the number of connected components of D_G , which we also maintain. As a result, by Euler's formula, we get the number of connected components of G , so in particular we may check if the deletion caused the number of connected components to increase. The algorithm executes $O(n)$ find and union operations on the union-find data structure.

In addition to that, the sequence of union operations has a certain structure. Let G_1 be the initial version of the graph G (before any edge deletion). Observe that each union operation takes as arguments the endpoints of an edge of G_1^* . The variant of the union-find problem, in which the set of allowed union operations forms a planar graph given during initialization, was considered by Gustedt [7]. He showed that for this special case of the union-find problem there exists an algorithm that may execute any sequence of $O(n)$ operations in $O(n)$ time (for an n -vertex planar graph). Thus, we infer that our algorithm runs in $O(n)$ time. ◀

We can now use Lemma 2 to show a simple decremental connectivity algorithm that runs in $O(n \log n)$ total time.

► **Lemma 3.** *Let G be a planar graph subject to edge deletions. There exists a decremental connectivity algorithm that for every vertex of G maintains its cc-identifier explicitly. It runs in $O(n \log n)$ total time.*

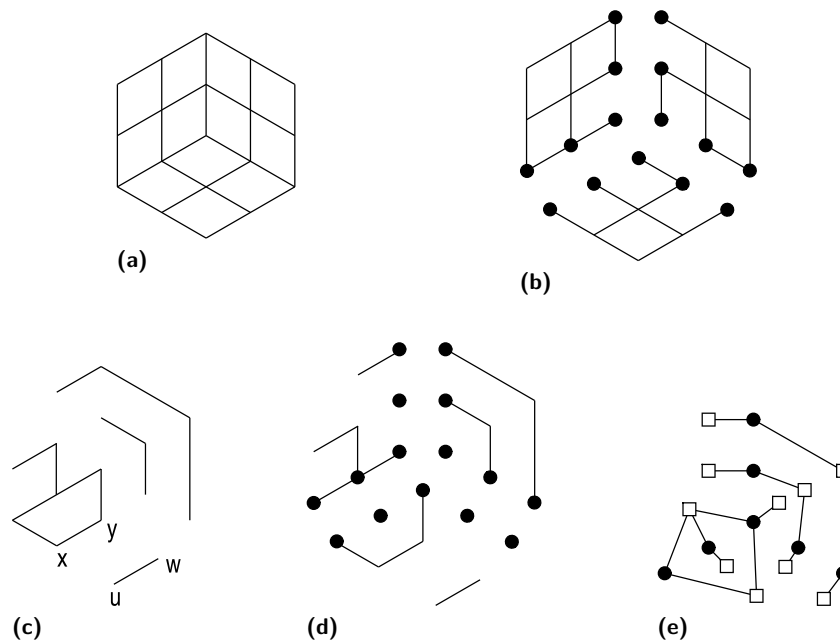
Proof. We use Lemma 2 to detect critical deletions. When an edge uw is deleted, and the deletion is not critical, nothing has to be done. Otherwise, after a critical deletion, some connected component C breaks into two components C_u and C_w ($u \in C_u$, $w \in C_w$) and we start two parallel depth-first searches from u and w . We stop both searches once the first of them finishes. W.l.o.g. assume that it is the search started from u . Thus, we know that the size of C_u is at most half of the size of C .³ We can now iterate through all vertices of C_u and change their cc-identifiers to a new unique number. All these steps require $O(|C_u|)$ time. The running time of the algorithm is proportional to the total number of changes of the cc-identifiers. Since every vertex changes its identifier only when the size of its connected component halves, we infer that the total running time is $O(n \log n)$. ◀

4 $O(n \log \log n)$ Time Algorithm

In order to speed up the $O(n \log n)$ algorithm, we need to speed up the linear depth-first searches that are run after a critical edge deletion. We build an r -division \mathcal{P} of G for $r = \log^2 n$ and use a separate decremental connectivity algorithm to maintain the connectivity information inside each region. On top of that, we maintain a *skeleton graph* that represents connectivity information between the set of boundary vertices (and possibly some other vertices that we consider important). Loosely speaking, since the number of boundary vertices is $O(n/\log n)$ we can pay a cost of $O(\log n)$ for maintaining the cc-identifier for each of them.

► **Definition 4.** Consider an r -division \mathcal{P} of a planar graph $G = (V, E)$ and a set V_s (called a *skeleton set*), such that $\partial(\mathcal{P}) \subseteq V_s \subseteq V$. The *skeleton graph* for \mathcal{P} and V_s is a graph over the skeleton set V_s and some additional auxiliary vertices. Consider a region R of \mathcal{P} . Group

³ Since the graph has constant degree, we may assure that both searches are synchronized in terms of the number of visited vertices.



■ **Figure 2** Panels 2a and 2b show a sample graph G and its r -division into three regions (boundary vertices are marked with small circles). In panel 2c there is graph G' obtained from G by a sequence of edge deletions. Panel 2d shows its r -division obtained from the r -division of G (again, boundary vertices are marked with small circles). Finally, panel 2e contains the skeleton graph of G' . Auxiliary vertices are marked with squares.

vertices of $V_s \cap V(R)$ into sets V_1, \dots, V_k , such that two vertices belong to the same set if and only if there is a path in R that connects them. For each set V_i add a new auxiliary vertex w_i and add an edge $w_i x$ for every $x \in V_i$.

For illustration, see Fig. 2.

► **Lemma 5.** *The skeleton graph has $O(|V_s|)$ vertices and edges.*

Proof. For each region R , we add at most one vertex and edge per each vertex of $V_s \cap V(R)$. Since each vertex belongs to a constant number of regions, we get the desired bound. ◀

► **Lemma 6.** *If $u, w \in V_s$, then u and w are connected in the skeleton graph if and only if they are connected in G .*

Proof. Consider a region R of the r -division. From the construction it follows that two vertices of $V_s \cap V(R)$ are connected in G with a path inside R iff they are connected in the part of the skeleton graph built for this region.

(\implies) Follows directly from the above observation.

(\impliedby) Consider a path P in G between u and w . Break this path into subpaths at each element of V_s . Since $\partial(\mathcal{P}) \subseteq V_s \subseteq V$, each resulting subpath is fully contained in one region of the r -division. Clearly, from the property given at the beginning of the proof, for each subpath there exists a corresponding path in the skeleton graph. ◀

In our algorithm we will update the skeleton graph of G , as edges are deleted. Similarly to the $O(n \log n)$ algorithm, we need a way of detecting whether an edge deletion in G increases the number of connected components in the skeleton graph.

► **Lemma 7.** *Let G be a dynamic planar graph, subject to edge deletions. Assume that we maintain its skeleton graph G_s computed for an r -division \mathcal{P} and a skeleton set V_s . An edge deletion in G causes an increase in the number of connected components in G_s if and only if the deletion is critical in G and there exists a region of \mathcal{P} , in which the deletion disconnects some two vertices of V_s .*

Before we proceed with the proof, let us note that all its conditions are necessary. In particular, a critical deletion in G may not disconnect some two vertices of a skeleton set in a region (e.g. edge uw in Fig. 2c, whose deletion does not affect the skeleton graph at all). It may also happen that the deletion is not critical in G , but inside some region it disconnects some two vertices of V_s (e.g. edge xy in Fig. 2c).

Proof. By Lemma 6, two vertices of V_s are connected in G iff they are connected in G_s .

(\implies) If two vertices of V_s become disconnected in G_s , they also become disconnected in G , so the edge deletion is critical. The deletion has to disconnect some two vertices in a region, because otherwise the graph G_s would not change at all.

(\impliedby) Assume that the deletion disconnected vertices $u, w \in V_s$ in a region R . Thus, the deleted edge was on some path from u to w . Since the edge deletion is critical in G , the deleted edge was a bridge in G . After the deletion there is no path from u to w in G and consequently also in G_s . ◀

Before we proceed with the algorithm, we show how to extend an algorithm maintaining cc-identifiers with two useful operations.

► **Lemma 8.** *Let $G = (V, E)$ be a planar graph and let $X \subseteq V$. Assume there exists a decremental connectivity algorithm that maintains cc-identifiers of a set $X \subseteq V$ explicitly and processes updates in $\Omega(n)$ total time. Then, we can extend the algorithm, so that:*

- *after every edge deletion, if the deletion disconnects some two vertices of X , it reports a pair of vertices that become disconnected,*
- *given a cc-identifier, it returns a vertex $v \in X$ with the same cc-identifier (or reports that such a vertex does not exist).*

The extended algorithm has the same asymptotic running time.

Proof. Since each cc-identifier can be encoded in $\log n + O(1)$ bits, there are $O(n)$ possible cc-identifiers. Thus, for each possible cc-identifier c , we maintain a list L_c of vertices of X , whose cc-identifier is c . Note that maintaining these lists takes time that is linear in the number of changes of cc-identifiers. Moreover, we need $O(n)$ time to initialize the lists L_c .

Observe that the lists allow us to find a vertex of X of given cc-identifier in constant time, so the second claim follows. To show the first claim, consider a case when after an edge deletion some (but not all) elements from a list L_c are removed. All these elements have to be added to a single list $L_{c'}$ and $L_{c'}$ must have been empty before the new elements were added. This means that the number of distinct cc-identifiers have increased, and some elements of X became disconnected. We can now take any $u \in L_c$ and $w \in L_{c'}$ and report that u and w became disconnected. ◀

We are ready to show the main building block of our $O(n \log \log n)$ algorithm.

► **Lemma 9.** *Let G be a planar graph. Assume there exists a decremental connectivity algorithm that runs in $f(n)$ time and maintains cc-identifiers explicitly. Then, there exists a decremental connectivity algorithm that runs in $O(n + n \cdot f(\log^2 n) / \log^2 n)$ time and answers queries in $O(1)$ time.*

Proof. We build an r -division \mathcal{P} of G for $r = \log^2 n$. By Lemma 1, this takes $O(n)$ time. For each region R of the division, we run the assumed decremental algorithm to handle edge deletions. We use A_R to denote the algorithm run for region R . A_R maintains cc-identifiers of $V(R)$ explicitly. We call these cc-identifiers *local* cc-identifiers. We also extend each A_R according to Lemma 8, taking $X = \partial(\mathcal{P}) \cap V(R)$. Moreover, we use Lemma 2 to detect critical deletions in G .

We build the skeleton graph G_s for G , r -division \mathcal{P} and a skeleton set $V_s = \partial(\mathcal{P})$. We maintain G_s , as edges are deleted, that is the deletions in G are reflected in G_s . This can be done using the algorithms A_R . By Lemma 8, A_R can report that some two vertices of V_s become disconnected inside R . This means that G_s needs to be updated. Observe that the part of G_s inside a region R can be implicitly represented as a partition of $V_s \cap V(R)$, where two vertices belong to the same element of the partition, if they are connected in R . Thus, if a deletion causes t local cc-identifiers to change, we may update G_s in $O(t)$ time. As a result, the time for updating G_s is linear in the number of local cc-identifiers that are changed.

For every vertex of G_s , we maintain its cc-identifier (called a *global* cc-identifier). Once G_s is updated after an edge deletion, we use Lemma 7 to check whether the number of connected components of G_s increased. According to the Lemma, it suffices to check whether the deletion is critical in G (this is reported by the algorithm of Lemma 2), and whether some two elements of the skeleton set became disconnected within some region (using Lemma 8).

When we detect that the number of connected components of the skeleton graph G_s has increased, similarly to the $O(n \log n)$ algorithm, we run two parallel DFS searches to identify the smaller of the two new connected components, and update the global cc-identifiers.

In order to answer a query regarding two vertices u and w , we perform two checks. First, if the vertices belong to the same region, we check whether there exists a path connecting them that does not contain any boundary vertices. This can be done by querying algorithm A_R for the appropriate region.

Then, we check whether there is a path from u to w that contains some boundary vertex. For each of the two vertices, we find two arbitrary boundary vertices b_u and b_w that u and w are connected to (using Lemma 8). Then, we check whether b_u and b_w have the same global cc-identifier.

Let us now analyze the running time. The algorithm of Lemma 2 requires $O(n)$ time. The algorithms A_R take $O(n \cdot f(r)/r) = O(n \cdot f(\log^2 n)/\log^2 n)$ time. Lastly, we bound the running time of the DFS searches performed to update the global cc-identifiers. We use an argument similar to the one in the proof of Lemma 3. The skeleton graph has $O(n/\log n)$ vertices, and each global cc-identifier can change at most $O(\log(n/\log n)) = O(\log n)$ times. Hence, the DFS searches require $O((n/\log n) \log n) = O(n)$ time. The lemma follows. ◀

By applying Lemma 3 to Lemma 9, we obtain the following.

► **Lemma 10.** *There exists a decremental connectivity algorithm for planar graphs that runs in $O(n \log \log n)$ total time.*

Proof. The total update time of the algorithm of Lemma 3 is $f(n) = O(n \log n)$. Thus, the running time is $O(n + n \cdot f(\log^2 n)/\log^2 n) = O(n + n \log^2 n \log \log n / \log^2 n) = O(n \log \log n)$. ◀

5 $O(n \log \log \log n)$ Time Algorithm

In order to obtain a faster algorithm, we would like to use Lemma 9 multiple times, starting from the $O(n \log n)$ algorithm, and each time applying the Lemma to the algorithm obtained in the previous step. This, however, cannot be done directly. While the Lemma requires an algorithm that maintains all cc-identifiers explicitly, it does not produce an algorithm with this property. We deal with this problem in this section.

Observe that in the proof of Lemma 9 we only needed the assumed decremental algorithm to maintain the cc-identifiers of the vertices of the skeleton set. This fact can be exploited in the following way. We show that if we have an algorithm that maintains cc-identifiers of some vertices, we may construct another (possibly faster) algorithm with the same property.

► **Lemma 11.** *Assume there exists a decremental connectivity algorithm for planar graphs that, given a graph $G = (V, E)$ and a set $V_e \subseteq V$ (called an explicit set):*

- maintains cc-identifiers of the vertices of V_e explicitly,
 - processes updates in $f(n) + O(|V_e| \log n)$ time,
 - may return the cc-identifier of any vertex in $g(n)$ time,
- where $f(n)$ and $g(n)$ are nondecreasing functions.

Then, there exists a decremental connectivity algorithm for planar graphs, which, given a graph $G = (V, E)$ and a set $V_e \subseteq V$:

- maintains cc-identifiers of the vertices of V_e explicitly,
- processes updates in $O(n + |V_e| \log n + n \cdot f(\log^2 n) / \log^2 n)$ time,
- may return the cc-identifier of any vertex in $g(\log^2 n) + O(1)$ time.

Proof. We build an r -division \mathcal{P} of G for $r = \log^2 n$. By Lemma 1, this takes $O(n)$ time. We also build a skeleton graph G_s , by taking a skeleton set $V_s := V_e \cup \partial(\mathcal{P})$. Hence, $|V_s| = |V_e| + n / \log n$.

For each region R of \mathcal{P} , we run a copy A_R of the assumed decremental connectivity algorithm, extended according to Lemma 8. Observe that in the proof of Lemma 9, we only need A_R to explicitly maintain cc-identifiers of $V_s \cap V(R)$. Thus, the set of explicit vertices for algorithm A_R is $V_s \cap V(R)$. Hence, A_R maintains local cc-identifiers of these vertices.

We maintain the graph G_s and its global cc-identifiers in the same way as in the proof of Lemma 9. The only difference is that now the skeleton set V_s is bigger. Let us bound the running time. First, algorithm A_R uses $f(\log^2 n) + O(|V_s \cap V(R)| \log n)$ time. Summing it over all regions, we obtain

$$\begin{aligned} \sum_{R \in \mathcal{P}} f(\log^2 n) + O(|V_s \cap V(R)| \log n) &= O(n \cdot f(\log^2 n) / \log^2 n + |V_s| \log n) \\ &= (n \cdot f(\log^2 n) / \log^2 n + |V_e| \log n + n / \log n \cdot \log n) \\ &= (n \cdot f(\log^2 n) / \log^2 n + |V_e| \log n + n). \end{aligned}$$

Note that we use the fact that each vertex is contained in a constant number of regions. The the running time of depth-first searches used to update the global cc-identifiers is

$$O(|V_s| \log n) = O(n / \log n \cdot \log n + |V_e| \log n) = O(n + |V_e| \log n).$$

Thus, the total update time is $O(n + |V_e| \log n + n \cdot f(\log^2 n) / \log^2 n)$.

Since the cc-identifiers of vertices of G_s are maintained explicitly, in particular we explicitly maintain the cc-identifiers of vertices of V_e . It remains to describe the process of

computing the global cc-identifier of an arbitrary vertex $v \in V$. Assume that v belongs to a region R (in case v is a boundary vertex, we may use an arbitrary region containing it). We first query A_R to obtain the local cc-identifier of v . We use Lemma 8 to check whether there exists a vertex b_v in $V_s \cap V(R)$ that has the same local cc-identifier as v . If this is the case, since b_v belongs to the skeleton set, we return its global cc-identifier (maintained explicitly). Otherwise, we return a new cc-identifier by encoding as an integer a pair consisting of the identifier of the region containing v (this requires $\log O(n/\log^2 n) = \log n + O(1) - 2 \log \log n$ bits) and the local cc-identifier of v (which requires $\log \log^2 n + O(1) = 2 \log \log n + O(1)$ bits). Overall, the resulting cc-identifier requires $\log n + O(1)$ bits. Thus, obtaining a cc-identifier of an arbitrary vertex requires $g(\log^2 n) + O(1)$ time. ◀

The main advantage of Lemma 11 over Lemma 9 is that we may apply Lemma 11 recursively to obtain better algorithms. We can view applying Lemma 11 as reducing connectivity in a graph of size n to connectivity in a collection of graphs of size $\log^2 n$. If we apply Lemma 11 to itself, we obtain the following.

► **Lemma 12.** *Assume there exists a decremental connectivity algorithm for planar graphs that, given a graph $G = (V, E)$ and a set $V_e \subseteq V$ (called an explicit set):*

- maintains cc-identifiers of the vertices of V_e explicitly,
 - processes updates in $f(n) + O(|V_e| \log n)$ time,
 - may return the cc-identifier of any vertex in $g(n)$ time,
- where $f(n)$ and $g(n)$ are nondecreasing functions.

Then, there exists a decremental connectivity algorithm for planar graphs, which, given a graph $G = (V, E)$ and a set $V_e \subseteq V$:

- maintains cc-identifiers of the vertices of V_e explicitly,
- processes updates in $O(n + |V_e| \log n + n \cdot f(\log^2 \log^2 n) / \log^2 \log^2 n)$ time,
- may return the cc-identifier of any vertex in $g(\log^2 \log^2 n) + O(1)$ time.

Proof. We apply Lemma 11 to the assumed algorithm and obtain an algorithm with total update time $f_1(n) + O(|V_e| \log n)$, where $f_1(n) = O(n + n \cdot f(\log^2 n) / \log^2 n)$ and query time $g_1(n) = g(\log^2 n) + O(1)$. Then, we apply the Lemma again to the new algorithm and get a new algorithm, whose total update time is

$$\begin{aligned} & O(n + |V_e| \log n + n \cdot f_1(\log^2 n) / \log^2 n) = \\ & = O(n + |V_e| \log n + n(\log^2 n + \log^2 n \cdot f(\log^2 \log^2 n) / \log^2 \log^2 n) / \log^2 n) \\ & = O(n + |V_e| \log n + n \cdot f(\log^2 \log^2 n) / \log^2 \log^2 n). \end{aligned}$$

It answers queries in $g(\log^2 \log^2 n) + O(1)$ time. ◀

We may now apply Lemma 12 to the simple $O(n \log n)$ algorithm (see Lemma 3) to obtain the following.

► **Lemma 13.** *There exists a decremental connectivity algorithm, which processes any sequence of updates in $O(n \log \log \log n)$ time.*

Proof. The simple algorithm processes updates in $f(n) = O(n \log n)$ time. Thus, we have $f(\log^2 \log^2 n) = O((\log^2 \log^2 n) \log(\log^2 \log^2 n)) = O((\log^2 \log^2 n) \log \log \log n)$, so the total update time is $O(n \log \log \log n)$. Since $g(n) = O(1)$, the query time is constant. ◀

6 $O(n)$ Time Algorithm

In this section we finally show an algorithm that runs in $O(n)$ time. Observe that in Lemma 12, we run the assumed decremental algorithm on graphs of size $\log^2 \log^2 n$. However, the number of all such graphs is so small, that we may precompute all necessary connectivity information for all of them.

► **Lemma 14.** *Let w be the word size and $\log n \leq w$. After preprocessing in $o(n)$ time, we may repeatedly initialize and run algorithms for decremental maintenance of connected components in graphs of size $t = O(\log^2 \log n)$. These algorithms may be given a set of vertices V_e , and maintain the cc-identifiers of vertices of V_e explicitly. An algorithm for a graph of size t runs in $O(t + |V_e| \log t)$ time and may return the cc-identifier of every vertex in $O(1)$ time.*

Proof. We will call the set V_e the *explicit set*. The state of the algorithm is uniquely described by the current set of edges in the graph and the explicit set. There are $2^{t(t-1)/2}$ labeled undirected graphs on t vertices (including non-planar graphs) and $O(2^t)$ possible explicit sets. Thus, there are $O(2^{t^2})$ possible states, which, for $t = O(\log^2 \log n)$ gives $2^{O(\log^4 \log n)} = 2^{o(\log n)} = o(n)$. In particular, each state can be encoded as a binary string of length $O(\log^4 \log n)$ which fits in a single machine word.

For each state, we precompute cc-identifiers. Moreover, for each pair of state and an edge to be deleted, we compute the changes to the cc-identifiers of vertices in the explicit set. Observe that if the edge deletion is critical, we simply need to compute the set of vertices in the smaller out of the two connected components that are created and store the intersection of this set and V_e . These vertices should be assigned new, unique cc-identifiers.

We encode the graph by a binary word of length $O(\log^4 \log n)$, where each bit represents an edge between some pair of vertices. Thus, when an edge is deleted, we may compute the new state of the algorithm in constant time by switching off a single bit. For any planar graph and any sequence of deletions, the number of changes of cc-identifiers of vertices of V_e is $O(|V_e| \log n)$ (using the analysis similar to the one from the proof of Lemma 3). The query time is constant, since the cc-identifiers are maintained explicitly. For each of the $2^{O(\log^4 \log n)}$ states, we require $O(\log^4 \log n)$ preprocessing time. Thus, the preprocessing time is $o(n)$. ◀

We may now apply Lemma 12 to the algorithm of Lemma 14 to obtain the main result of this paper.

► **Theorem 15.** *There exists a decremental connectivity algorithm for planar graphs that supports updates in $O(n)$ total time and answers queries in constant time.*

7 Conclusion and Open Problems

We have shown a reduction from the decremental connectivity problem in planar graphs to incremental connectivity. As a result, we obtain an algorithm for decremental connectivity that processes all updates in optimal $O(n)$ time and answers queries in constant time. This shows that the total time complexity of the decremental problem is not $\Omega(n \log n)$, which seemed to be a natural bound. In other words we have shown that a lower bound of $\Omega(n \log n)$, that would be an analogous to the lower bound in [14], cannot hold for decremental algorithms in planar graphs. We actually conjecture that even for general graphs with $O(n)$ edges there exists an $o(n \log n)$ time decremental algorithm.

An interesting question would be to study the worst-case time complexity of decremental connectivity in planar graphs, which has not been fully understood yet. And, contrary to the incremental problem, no nontrivial lower bounds are known.

References

- 1 Stephen Alstrup, Jens P. Secher, and Maz Spork. Optimal on-line decremental connectivity in trees. *Inf. Process. Lett.*, 64(4):161–164, 1997.
- 2 David Eppstein, Zvi Galil, and Giuseppe F. Italiano. *Improved sparsification*. Information and Computer Science, University of California, Irvine, 1993.
- 3 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Amnon Nissenzweig. Sparsification - a technique for speeding up dynamic graph algorithms. *J. ACM*, 44:669–696, 1997.
- 4 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification: I. Planarity testing and minimum spanning trees. *J. Comput. Syst. Sci.*, 52(1):3–27, 1996.
- 5 David Eppstein, Giuseppe F. Italiano, Roberto Tamassia, Robert Endre Tarjan, Jeffery Westbrook, and Moti Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algorithms*, 13(1):33–54, 1992.
- 6 Greg N. Frederickson. Data structures for on-line updating of minimum spanning trees, with applications. *SIAM J. Comput.*, 14(4):781–798, 1985.
- 7 Jens Gustedt. Efficient union-find for planar graphs and other sparse graph classes. *Theoretical Computer Science*, 203(1):123 – 141, 1998.
- 8 Monika R. Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46(4):502–516, July 1999.
- 9 Monika Rauch Henzinger and Michael L. Fredman. Lower bounds for fully dynamic connectivity problems in graphs. *Algorithmica*, 22(3):351–362, 1998.
- 10 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.
- 11 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '13, pages 1131–1142. SIAM, 2013.
- 12 Sanjeev Khanna, editor. *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*. SIAM, 2013.
- 13 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 505–514. ACM, 2013.
- 14 Mihai Pătraşcu and Erik D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM J. Comput.*, 35(4):932–963, 2006.
- 15 Johannes A. La Poutré. Lower bounds for the union-find and the split-find problem on pointer machines. *J. Comput. Syst. Sci.*, 52(1):87–99, 1996.
- 16 Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, 1975.
- 17 Robert Endre Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *J. Comput. Syst. Sci.*, 18(2):110–127, 1979.
- 18 Mikkel Thorup. Decremental dynamic connectivity. *J. Algorithms*, 33(2):229–243, 1999.
- 19 Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In F. Frances Yao and Eugene M. Luks, editors, *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 343–350. ACM, 2000.

- 20 Freek van Walderveen, Norbert Zeh, and Lars Arge. Multiway simple cycle separators and I/O-efficient algorithms for planar graphs. In Khanna [12], pages 901–918.
- 21 Christian Wulff-Nilsen. Faster deterministic fully-dynamic graph connectivity. In Khanna [12], pages 1757–1769.

Testing Small Set Expansion in General Graphs*

Angsheng Li¹ and Pan Peng^{1,2}

1 State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences

angsheng@iso.ac.cn

2 Department of Computer Science, Technische Universität Dortmund

pan.peng@tu-dortmund.de

Abstract

We consider the problem of testing small set expansion for general graphs. A graph G is a (k, ϕ) -expander if every subset of volume at most k has conductance at least ϕ . Small set expansion has recently received significant attention due to its close connection to the unique games conjecture, the local graph partitioning algorithms and locally testable codes.

We give testers with two-sided error and one-sided error in the *adjacency list* model that allows degree and neighbor queries to the oracle of the input graph. The testers take as input an n -vertex graph G , a volume bound k , an expansion bound ϕ and a distance parameter $\varepsilon > 0$. For the two-sided error tester, with probability at least $2/3$, it accepts the graph if it is a (k, ϕ) -expander and rejects the graph if it is ε -far from any (k^*, ϕ^*) -expander, where $k^* = \Theta(k\varepsilon)$ and $\phi^* = \Theta(\frac{\phi^4}{\min\{\log(4m/k), \log n\} \cdot (\ln k)})$. The query complexity and running time of the tester are $\tilde{O}(\sqrt{m}\phi^{-4}\varepsilon^{-2})$, where m is the number of edges of the graph. For the one-sided error tester, it accepts every (k, ϕ) -expander, and with probability at least $2/3$, rejects every graph that is ε -far from (k^*, ϕ^*) -expander, where $k^* = O(k^{1-\xi})$ and $\phi^* = O(\xi\phi^2)$ for any $0 < \xi < 1$. The query complexity and running time of this tester are $\tilde{O}(\sqrt{\frac{n}{\varepsilon^3}} + \frac{k}{\varepsilon\phi^4})$.

We also give a two-sided error tester in the *rotation map* model that allows (*neighbor, index*) queries and degree queries. This tester has asymptotically almost the same query complexity and running time as the two-sided error tester in the adjacency list model, but has a better performance: it can distinguish any (k, ϕ) -expander from graphs that are ε -far from (k^*, ϕ^*) -expanders, where $k^* = \Theta(k\varepsilon)$ and $\phi^* = \Theta(\frac{\phi^2}{\min\{\log(4m/k), \log n\} \cdot (\ln k)})$.

In our analysis, we introduce a new graph product called *non-uniform replacement product* that transforms a general graph into a bounded degree graph, and approximately preserves the expansion profile as well as the corresponding spectral property.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases graph property testing, small set expansion, random walks, spectral graph theory

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.622

1 Introduction

Graph property testing is an effective algorithmic paradigm to deal with real-world networks, the scale of which has become so large that it is even impractical to read the whole input. In

* Both authors are partially supported by the Grand Project “Network Algorithms and Digital Information” of the Institute of software, Chinese Academy of Sciences and by a National Basic Research Program (973) entitled computational Theory on Big Data of Cyberspace, grant No. 2014CB340302. The second author acknowledges the support of ERC grant No. 307696.

the setting of testing a graph property P , we are given as input a graph G and we want to design an algorithm (called *tester*) to distinguish the case that G has property P from the case that G is “far from” the property P with high success probability (say $2/3$). Here, the notion of being “far from” is parameterized by a distance parameter ε . In most situations, a graph G is said to be ε -far from property P if one has to modify at least an ε fraction of the representation (or edges) of G to obtain a graph G' with property P . We assume the input graph G can be accessed through an oracle \mathcal{O}_G and the goal is to design property testers that make as few queries as possible to \mathcal{O}_G .

Since the seminal work of Goldreich and Ron [17], many testers have been developed for different graph properties, such as k -colorability, bipartiteness, acyclicity, triangle-freeness and many others. Most of these testers apply only to the adjacency matrix model or the adjacency list model, depending on the types of queries the testers are allowed to ask the oracle. The former model is most suitable for *dense* graphs and general characterizations on the testability of a property in this model has been given (e.g., [2]). The latter model is most suitable for *sparse* graphs, and several property testers using the techniques of local search or random walks are known, while it is not well understood what properties are testable in constant time in this model. Even less is known about testers, testability results or even models for general graphs (see recent surveys [36, 14]).

In this paper, we focus on property testers for general graphs. We will consider the *adjacency list* model that allows *degree* queries and *neighbor* queries to the oracle of the graph [31]. For the degree query, when specified a vertex v , the oracle returns the degree of v ; for the neighbor query, when specified a vertex v and an index i ; the oracle returns the i th neighbor of v . The adjacency list model also applies to the bounded degree graphs with an additional restriction that a fixed upper-bound was assumed on the degrees [17]. We will also consider a new model which we call *rotation map* model that allows degree queries and (*neighbor, index*) queries to the oracle [24]. For the (*neighbor, index*) query, when specified a vertex v and an index i , the oracle returns a pair (u, j) such that u is the i th neighbor of v and j is the index of u as a neighbor of v . Note that the rotation map model is at least as strong as the adjacency list model.

We study the problem of testing *small set expansion* for general graphs. Given a graph $G = (V, E)$ with n vertices and m edges, and a set $S \subseteq V$, let the *volume* of S be the sum of degree of vertices in S , that is, $\text{vol}(S) := \sum_{v \in S} \deg_G(v)$, where $\deg_G(v)$ denotes the degree of vertex v . Define the *conductance* of S as $\phi(S) := \frac{e(S, V \setminus S)}{\text{vol}(S)}$, where $e(S, V \setminus S)$ is the number of edges leaving S ; and define the *k -expansion profile* of G as $\phi(k) := \min_{S: \text{vol}(S) \leq k} \phi(S)$. A graph G is called a (k, ϕ) -expander if $\phi(k) \geq \phi$, that is, all the subsets in G with volume at most k have conductance at least ϕ . We will refer to small set expander as (k, ϕ) -expander and refer to small set expansion as $\phi(k)$.

Besides of the relation to the mixing time of random walks [26], small set expansion has been of much interest recently for its close connection to the unique games conjecture [32, 7], the design of local graph partitioning algorithms in massive graphs [39, 5, 6, 30, 22], and locally testable codes that are testable with linear number of queries [8]. Approximation algorithms and spectral characterizations for the small set expansion problem have been studied [7, 25, 23, 22, 30, 29]. It is natural to ask if one can efficiently (in sublinear time) test if a graph is a small set expander.

1.1 Our results

We give testers for small set expansion in the adjacency list model as well as the rotation map model for general graphs. We use the common definition of distance between graphs.

More precisely, a graph G with m edges is said to be ε -far from a (k, ϕ) -expander if one has to modify at least εm edges of G so that it becomes a (k, ϕ) -expander. We will assume throughout the paper that $m = \Omega(n)$ (and a brief discussion is given in Section 2), while the algorithm is not given as input the number of edges m .

1.1.1 Testers in adjacency list model

Our first result is a property tester for small set expansion with two-sided error in the adjacency list model.

► **Theorem 1.** *Given degree and neighbor query access to an n -vertex graph, a volume bound k , a distance parameter ε and a conductance bound ϕ , there exists an algorithm that with probability at least $2/3$, accepts any graph that is a (k, ϕ) -expander, and rejects any graph that is ε -far from any (k^*, ϕ^*) -expander, where $k^* = \Theta(k\varepsilon)$ and $\phi^* = \Theta(\frac{\phi^4}{\min\{\log(4m/k), \log n\} \cdot (\ln k)})$, where m is the number of edges of G . The query complexity and running time of the algorithm are $\tilde{O}(\sqrt{m}\phi^{-4}\varepsilon^{-2})$.*

Note that the running time of the tester matches the best known algorithms for testing the conductance of G which corresponds to the case $k = m$ (see further discussions below).

As a byproduct of our analysis for the above two-sided error tester, we obtain a one-sided error tester (that accepts every (k, ϕ) -expander) by invoking a local algorithm for finding small sparse cuts. We show the following result.

► **Theorem 2.** *Given degree and neighbor query access to an n -vertex graph, a volume bound k , a conductance bound ϕ , and a distance parameter ε , there exists an algorithm that always accepts any graph that is a (k, ϕ) -expander, and with probability at least $2/3$ rejects any graph that is ε -far from any (k^*, ϕ^*) -expander, where $k^* = O(k^{1-\xi})$ and $\phi^* = O(\xi\phi^2)$ for any $0 < \xi < 1$. Furthermore, whenever it rejects a graph, it provides a certificate that the graph is not a (k, ϕ) -expander in the form of a set of volume at most k and expansion at most ϕ . The query complexity and running time of the algorithm are $\tilde{O}(\sqrt{\frac{n}{\varepsilon^3}} + \frac{k}{\varepsilon\phi^4})$.*

Note that ξ is not necessarily a constant, and the running time of the above algorithm is sublinear in m for $k = O(\frac{m}{\log^{\Omega(1)} n})$ and constant ϕ .

1.1.2 Tester in rotation map model

We also give a two-sided error tester in the rotation map model. Note that the gap of the conductance value in completeness and soundness here is smaller than the corresponding gap in the tester in adjacency list model.

► **Theorem 3.** *Given degree and (neighbor, index) query access to an n -vertex graph, a volume bound k , a distance parameter ε and a conductance bound ϕ , there exists an algorithm that with probability at least $2/3$, accepts any graph that is a (k, ϕ) -expander, and rejects any graph that is ε -far from any (k^*, ϕ^*) -expander, where $k^* = \Theta(k\varepsilon)$ and $\phi^* = \Theta(\frac{\phi^2}{\min\{\log(4m/k), \log n\} \cdot (\ln k)})$, where m is the number of edges of G . The query complexity and running time of the algorithm are $\tilde{O}(\sqrt{m}\phi^{-2}\varepsilon^{-2})$.*

1.1.3 Graph transformation

The analysis of the above two-sided error tester involves analyzing random walks on a bounded degree graph by the spectral property of small set expander and a new graph

product which we call *non-uniform replacement product* that transforms every graph (with possible multiple edges and self-loops) into a bounded degree graph, and in the process, the expansion profile of the resulting graph does not differ by much from that of the original graph. This transformation may be of independent interest, and we present the formal result below. Let \mathcal{L}_G be the normalized Laplacian matrix of a graph G and let $\lambda_i(G)$ denote the i th smallest eigenvalues of \mathcal{L}_G .

► **Theorem 4.** *Let $\phi < 1$ and $k \leq m$. For any graph $G = (V, E)$ with n vertices and m edges, there exists a 16-regular graph G' with $\Theta(m)$ vertices such that*

1. *If $S \subseteq V(G)$ is a subset in G with $\phi_G(S) \leq \phi$, then there exists a set $S' \subseteq V(G')$, such that $|S'| = \Theta(\text{vol}_G(S))$ and $\phi_{G'}(S') \leq \phi/16$;*
2. *If for any set $S \subseteq V(G)$ with $\text{vol}_G(S) \leq k$, $\phi_G(S) \geq \phi$, then*
 - (a) *for any $S' \subseteq V(G')$ with $|S'| \leq \Theta(k)$, $\phi_{G'}(S') = \Omega(\phi^2)$.*
 - (b) *for any $\alpha > 0$, it holds that $\lambda_{\frac{(1+\alpha)2m}{k}}(G') = \Omega(\alpha^6 \phi^2 (\log \frac{2m}{k})^{-1})$, and $\lambda_{(\frac{2m}{k})^{1+\alpha}}(G') = \Omega(\alpha \phi^2 \log_n \frac{2m}{k})$. Furthermore, if $k = m$, then $\lambda_2(G') = \Omega(\phi^2)$.*

Note that by recent spectral characterization of small set expansion of G and the preconditions of the Item 2 of Theorem 4, we have $\lambda_{\frac{(1+\alpha)2m}{k}}(G) = \Omega(\alpha^6 \phi^2 (\log \frac{2m}{k})^{-1})$, $\lambda_{(\frac{2m}{k})^{1+\alpha}}(G) = \Omega(\alpha \phi^2 \log_n (2m/k))$, and if $k = m$, $\lambda_2(G) = \Omega(\phi^2)$ (see Section 2.2). Also we stress that Item 2b above is not a direct consequence of Item 2a and inequalities in Section 2.2, and its proof involves a more refined spectral analysis. The main point from G to G' is that the property of small set expansion is well preserved and the maximum degree is also greatly reduced, which is comparable to work on constructions from high degree expanders to constant degree expanders (see eg., [33, 4]).

1.2 Other related work

There is an interesting line of research on testing the special case of the (k, ϕ) -expander for $k = m$, which is often abbreviated as ϕ -*expander*. The corresponding quantity $\phi(m)$ is often called the *expansion (or conductance)* of G [19]. Goldreich and Ron [16] have proposed an expansion tester for bounded degree graphs in the adjacency list model. The tester (with different setting parameters) has later been analyzed by Czumaj and Sohler [11], Nachmias and Shapira [28], and Kale and Seshadhri [20], and it is proven that the tester can distinguish d -regular ϕ -expanders from graphs that are ε -far from any d -regular $\Omega(\eta\phi^2)$ -expanders for any $\eta > 0$. The query complexity and running time of the tester are $O(\frac{n^{0.5+\eta}}{\phi^2} (\varepsilon^{-1} \log n)^{O(1)})$, which is almost optimal by a lower bound of $\Omega(\sqrt{n})$ given by Goldreich and Ron [17]. Li, Pan and Peng [24] give an expansion tester in the rotation map model with query complexity and running time $\tilde{O}(\frac{m^{1/2+\eta}}{\phi^2} (\varepsilon^{-1} \log n)^{O(1)})$ for general graphs that matches the best known tester for bounded degree graphs. We remark that when $k = m$, our two-sided tester in the rotation map model can be also guaranteed to test the conductance $\phi(m)$ of G with the same running time and approximation performance. In [24], a product called *non-uniform zig-zag product* was proposed to transfer an arbitrary graph into a bounded degree graph. However, the analysis there is more involved and does not seem to generalize to the k -expansion profile for any $k \leq m$ as considered here. Our analysis here is both simple and applicable to the broader case.

The techniques of random walks have also been used to test bipartiteness under different models [15, 21, 10]. In particular, Kaufman et al. extend the bipartiteness tester in bounded degree graphs to general graphs [21] and they also used the idea of replacing high degree vertices by expander graphs. Furthermore, we will also use their techniques for emulating

random walks (by performing queries to the oracle of the original graph) and sampling vertices almost uniformly in the transformed graph. However, the transformed graph in [21] may still have large maximum degree (that may be twice the average degree of the original graph), which is not applicable to our case. Ben-Eliezer et al. studied the strength of different query types in the context of property testing in general graphs [9]. The analysis for the expansion of the replacement product (and the zig-zag product) of two regular graphs are introduced in [35, 33, 37, 34].

1.3 Organization of the paper

The rest of the paper is organized as follows. In Section 2 we give some basic definitions and introduce the tools for our analysis. Then we introduce the non-uniform replacement product and show its property in Section 3. In Section 4, we give all our testers and prove the performance of these testers. Finally, we give a short conclusion in Section 5. All missing proofs can be found in the full version of the paper¹.

2 Preliminaries

Let $G = (V, E)$ be an undirected and simple graph with $|V| = n$ and $|E| = m$. Let $\deg_G(v)$ denote the degree of a vertex v . As mentioned in the introduction, we consider the *adjacency list* model and the *rotation map* model. In the adjacency list model, the graph is represented by its adjacency list, which is also accessible through an oracle access \mathcal{O}_G , and the algorithm is allowed to perform degree and neighbor queries to \mathcal{O}_G . In the rotation map model, the graph is represented by its rotation map that for each vertex u and an index $i \leq \deg_G(u)$, in the (u, i) th location of the representation the pair (v, j) is stored such that v is the i th neighbor of u and u is the j th neighbor of v . We are given an oracle access \mathcal{O}_G to the rotation map of G and allowed to perform degree queries and (neighbor, index) queries to \mathcal{O}_G . We remark that the rotation map model is at least as strong as the adjacency list model. For a graph with maximum degree bounded by d , we assume that d is a constant independent of n .

For a vertex subset $S \subseteq V$, let $e_G(S, V \setminus S)$ be the number of edges leaving S . Let $\text{vol}_G(S) := \sum_{v \in S} \deg_G(v)$ and $\phi_G(S) := e_G(S, \bar{S}) / \text{vol}_G(S)$ be the *volume* and the *conductance* of S in G , respectively. Note that $\text{vol}_G(G) := \text{vol}_G(V) = 2|E|$. In the following, when it is clear from context, we will omit the subscript G . Define the *k-expansion profile* of G as $\phi(k) := \min_{S: \text{vol}(S) \leq k} \phi(S)$. In particular, $\phi(m)$ is often referred to the *conductance (or expansion) of G* and we let $\phi(G) := \phi(m)$. A graph is called a ϕ -expander if $\phi(G) \geq \phi$.

► **Definition 5.** A graph G is a (k, ϕ) -expander if $\phi(k) \geq \phi$. Equivalently, G is a (k, ϕ) -expander if for every $S \subseteq V$ with volume $\text{vol}(S) \leq k$ has conductance $\phi(S) \geq \phi$.

We have the following definition of graphs that are ε -far from (k, ϕ) -expanders.

► **Definition 6.** A graph G is ε -far from any (k, ϕ) -expander if one has to modify at least εm edges of G to obtain a (k, ϕ) -expander.

As mentioned before, we will assume that $m = \Omega(n)$, as otherwise, there exists $n - o(n)$ isolated vertices in G , and the graph cannot be a (k, ϕ) -expander even for constant k and any $\phi > 0$. Furthermore, since we will only sample a constant number of vertices (as we do

¹ Full version available at <http://arxiv.org/abs/1209.5052>

in all our testers), then with high probability, the sampled vertices are all isolated, and in this case, we can safely reject the graph.

We will use bold letters to denote row vectors. For any vector $\mathbf{p} \in \mathbb{R}^V$, let $\mathbf{p}(S) := \sum_{v \in S} \mathbf{p}(v)$ and let $\|\mathbf{p}\|_1 = \sum_{v \in V} |\mathbf{p}(v)|$, $\|\mathbf{p}\|_2 = \sqrt{\sum_{v \in V} \mathbf{p}(v)^2}$ denote the l_1, l_2 -norm of \mathbf{p} , respectively. Let $\text{supp}(\mathbf{p})$ be the support of \mathbf{p} . Let $\mathbf{1}_S$ be the characteristic vector of S , that is, $\mathbf{1}_S(v) = 1$ if $v \in S$ and $\mathbf{1}_S(v) = 0$ otherwise. Let $\mathbf{1}_v := \mathbf{1}_{\{v\}}$.

2.1 Lazy random walks

We now introduce some tools that will be used in the design and analysis of our algorithms. The following also applies to graphs with possible multiple edges and/or self-loops. First, we define the *lazy random walks* on G . In a lazy random walk, if we are currently at vertex v , then in the next step, we choose a random neighbor u with probability $1/2 \deg_G(v)$ and move to u . With the remaining probability $1/2$, we stay at v .

For a given graph G , let A denote its adjacency matrix and let D denote the diagonal matrix such that $D_{u,u} = \deg(u)$ for any u . Let I denote the identity matrix. Then $W := (I + D^{-1}A)/2$ is the probability transition matrix of the lazy random walk of G . Note that if \mathbf{p}_0 is a probability distribution on V , then $\mathbf{p}_0 W^t$ denotes distribution of the endpoint of a length t lazy random walk with initial distribution \mathbf{p}_0 . In particular, we let $\mathbf{p}_v^t = \mathbf{1}_v W^t$ be the probability distribution of the endpoint of a walk of length t starting from vertex v . Furthermore, we let $\|\mathbf{p}_v^t\|_2^2$ denote the *collision probability* of such a walk.

For any lazy random walk matrix $W = \frac{I+D^{-1}A}{2}$, it is well known that all its eigenvalues are real (see eg. [30]). Furthermore, if we let $\eta_1(W) \geq \dots \geq \eta_n(W)$ denote the eigenvalues of W , then $0 \leq \eta_i(W) \leq 1$ for any $i \leq n$.

2.2 Spectral characterization of expansion profile

For a graph G , let $\mathcal{L} := I - D^{-1/2}AD^{-1/2}$ be the normalized Laplacian matrix of G . Let $0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n \leq 2$ be eigenvalues of \mathcal{L} . It is straightforward to verify that $\eta_i = 1 - \frac{\lambda_i}{2}$ for any $1 \leq i \leq n$, where η_i is the i th largest eigenvalue of the lazy random walk matrix W of G . We have the following lemmas relating the expansion profile and the eigenvalues of \mathcal{L} .

► **Lemma 7** (Cheeger inequality, [3, 1, 38]). *For every graph G , we have $\frac{\lambda_2}{2} \leq \phi(G) \leq \sqrt{2\lambda_2}$.*

► **Lemma 8** ([23, 25]). *For every graph G , $h \in \mathbb{N}$ and any $\alpha > 0$, we have $\phi(\frac{(1+\alpha)2m}{h}) \leq O(\frac{1}{\alpha^3} \sqrt{\lambda_h \log h})$.*

► **Lemma 9** ([40, 30, 29]). *For every graph G , $h \in \mathbb{N}$ and any $\alpha > 0$, we have $\phi(\frac{2m}{h^{1-\alpha}}) \leq O(\sqrt{(\lambda_h/\alpha) \log_h n})$.*

We remark that in some of references (eg. [23]), the k -expansion profile is defined to be the minimum conductance over all possible subsets of *size* at most k , rather than the *volume* measurement as defined here. However, their proofs imply that Lemma 8 and 9 also hold for our case.

2.3 A local algorithm for finding small sparse sets

We will need the following local algorithm for finding small sparse set to give a one-sided error tester in general graphs as well as to analyze the soundness of our testers. Here, the local algorithm takes as input a vertex v and only explores a small set of the vertices and

edges that are “close” to v , if the volume k of the target set is small. It only needs to perform degree queries and neighbor queries to the oracle of the input graph.

LocalSS(G, v, T, δ)

1. Let $\mathbf{q}_0 = \mathbf{1}_v$. For each time $0 \leq t \leq T$:
 - a. Define $\tilde{\mathbf{p}}_t$ such that $\tilde{\mathbf{p}}_t(u) = \mathbf{q}_t(u)$ if $\mathbf{q}_t(u) \geq \delta \deg(v)$ and $\tilde{\mathbf{p}}_t(u) = 0$ if $\mathbf{q}_t(u) < \delta \deg(v)$. Compute $\mathbf{q}_{t+1} := \tilde{\mathbf{p}}_t W$.
 - b. Let $s_t = |\text{supp}(\tilde{\mathbf{p}}_t)|$. Order the vertices in $\text{supp}(\tilde{\mathbf{p}}_t)$ so that $\frac{\tilde{\mathbf{p}}_t(v_1)}{\deg(v_1)} \geq \frac{\tilde{\mathbf{p}}_t(v_2)}{\deg(v_2)} \geq \dots \geq \frac{\tilde{\mathbf{p}}_t(v_{s_t})}{\deg(v_{s_t})}$.
 - c. For each $1 \leq i \leq s_t$, let $S_{i,t}$ be the first i vertices in this ordering.
2. Output the subgraph X with the smallest conductance among all the sets $\{S_{i,t}\}_{0 \leq t \leq T, 1 \leq i \leq s_t}$.

The performance of the above algorithm is guaranteed in the following lemma, which follows by combining Proposition 8 in [30] and Theorem 2 [22]. (More specifically, the first part of the lemma is Proposition 8 in [30] and the “Furthermore” part of the lemma follows from the proof of Theorem 2 [22]. See also the paragraph “Independent Work” in [22])

► **Lemma 10.** *Let $G = (V, E)$ and $t \geq 1$. If $S \subseteq V$ satisfies that $\phi(S) \leq \psi$, then there exists a subset $\hat{S} \subseteq S$ such that $\text{vol}(\hat{S}) \geq \text{vol}(S)/2$, and for any $v \in \hat{S}$, we have $\mathbf{p}_v^t(S) \geq c_1(1 - \frac{3\psi}{2})^t$ for some constant $c_1 > 0$. Furthermore, if $\text{vol}(S) \leq k$, then the algorithm **LocalSS**, with parameters $G, v, T = O(\frac{\zeta \log k}{\psi}), \delta = O(\frac{k^{-1-\zeta}}{T})$ for any $\zeta > 0$, will find a set X such that $\text{vol}(X) \leq O(k^{1+\zeta})$ and $\phi(X) \leq O(\sqrt{\psi/\zeta})$. The algorithm can be implemented in time $\tilde{O}(k^{1+2\zeta}\psi^{-2})$.*

3 Non-uniform replacement product

In this section, we give the definition of non-uniform replacement product and also show its property, which will be used in our testers for general graphs. Let $G = (V, E)$ be a graph with possible multiple edges or self-loops and with minimum degree $\delta \geq d$. Let $\mathcal{H} = \{H_u\}_{u \in V}$ be a family of $|V|$ graphs. The graph family \mathcal{H} is called a *proper d -regular graph family* of G if for each $u \in V$, H_u is a d -regular graph (with possible parallel edges or self-loops) with vertex set $[\deg_G(u)] := \{1, \dots, \deg_G(u)\}$. For any graph G and its proper d -regular graph family \mathcal{H} , the *non-uniform replacement product* of G and \mathcal{H} , denoted by $G \circledast \mathcal{H}$, is defined as follows.

1. For each vertex u in $V(G)$, the graph $G \circledast \mathcal{H}$ contains a copy of a H_u .
2. For any edge $(u, v) \in E(G)$, for each $i \in [\deg_G(u)]$, we specify a unique *but arbitrary* index $j \in [\deg_G(v)]$, and place d parallel edges between the i th vertex in H_u and the j th vertex in H_v .

Now that $G \circledast \mathcal{H}$ is a $2d$ -regular graph with $2|E|$ vertices. We will use (u, i) to index the vertices in $G \circledast \mathcal{H}$. We have the following lemma that formally characterize the intuition that if all the graphs in \mathcal{H} are expanders, that is, for any $H \in \mathcal{H}$, $\phi(H)$ is larger than some universal constant, then the expansion profile of G' will not differ by too much from the expansion profile of G .

► **Lemma 11.** *Let $G = (V, E)$ be a graph with minimum degree $\delta(G) \geq d$. Let \mathcal{H} be a proper d -regular graph family of G , and let $G' = G \circledast \mathcal{H}$. We have that*

- If $S \subseteq V(G)$ is a subset with $\phi(S) \leq \phi$ then the set $S' := \{(u, i) \in V(G') \mid u \in S, 1 \leq i \leq \deg_G(u)\} \in V(G')$ satisfies that $|S'| = \text{vol}(S)$ and $\phi_{G'}(S') \leq \phi/2$.
- If for any set $S \subseteq V(G)$ with $\text{vol}(S) \leq k$, $\phi(S) \geq \phi$ and for any u , the conductance of H_u satisfies $\phi(H_u) \geq \delta$, then for any set $S' \subseteq V(G')$ with $|S'| \leq \Theta(k)$, $\phi_{G'}(S') = \Omega(\delta\phi^2)$.

When the rotation map of the graph G is explicitly given, we define the *non-uniform replacement product with rotation map* of G and \mathcal{H} , denoted as $G^{(r)} \textcircled{R} \mathcal{H}$, as follows.

1. For each vertex u in $V(G)$, the graph $G^{(r)} \textcircled{R} \mathcal{H}$ contains a copy of a H_u .
2. For any edge $(u, v) \in E(G)$ such that v is the i th neighbor of u and u is the j th neighbor of v , we place d parallel edges between the i th vertex in H_u and the j th vertex in H_v .

Note that the above replacement product with rotation map is a special case of the (general) replacement product defined before. Thus, it not only satisfies the combinatorial property of expansion profile given in Lemma 11, but also satisfies the following nice spectral properties.

► **Lemma 12.** *Let $G = (V, E)$ be a graph with minimum degree $\delta(G) \geq d$. Let \mathcal{H} be a proper d -regular graph family of G , and let $G' = G^{(r)} \textcircled{R} \mathcal{H}$ be the replacement product with rotation map of G and \mathcal{H} . We have that*

- G' satisfies the two properties in Lemma 11.
- If for any set $S \subseteq V(G)$ with $\text{vol}(S) \leq k$, $\phi(S) \geq \phi$ and for any u , $\eta_2(W_{H_u}) \leq 1 - \delta$ for some $\delta > 0$, then for any $\alpha > 0$, $\eta_{\frac{(1+\alpha)2m}{k}}(W_{G'}) \leq 1 - \Omega(\delta^2 \alpha^6 \phi^2 (\log \frac{2m}{k})^{-1})$, $\eta_{(2m/k)^{1+\alpha}}(W_{G'}) \leq 1 - \Omega(\alpha \delta^2 \phi^2 \log_n(2m/k))$. Furthermore, when $k = m$, we have $\eta_2(W_{G'}) \leq 1 - \Omega(\delta^2 \eta^2)$.

Theorem 4 can be proved directly once we have Lemma 12.

Proof of Theorem 4. For any graph $G = (V, E)$, we first turn it into a graph $G_{\geq 8}$ with minimum degree 8 by adding an appropriate number of self-loops to vertices with degree smaller than 8. Note that this only changes the conductance of a set by a factor of 8. Now we let \mathcal{H} be a proper 8-regular graph family for $G_{\geq 8}$ such that for any $u \in V$, H_u is a Margulis expander with $\deg_{G_{\geq 8}}(u)$ vertices [27, 13]. Therefore, each H_u is an expander such that $\phi(H_u)$ and $1 - \eta_2(W_{H_u})$ are larger than some universal constants. Then we let $G' = G_{\geq 8}^{(r)} \textcircled{R} \mathcal{H}$, $d = 8$ and specify δ to be a constant in Lemma 12. By definition, G' is a 16-regular graph. Finally, the theorem follows by Lemma 12 and the fact that $\eta_i = 1 - \frac{\lambda_i}{2}$. ◀

4 Testers for small set expansion

In this section, we give all our testing algorithms for small set expansion. We first show a property of graphs that are far from small set expander in Section 4.1, which will be useful for all our testers. Then in Section 4.2, we give a two-sided error tester in bounded degree model, which illustrates basic ideas underlying our algorithms. Finally, we give testers in adjacency list model and in the rotation map for general graphs in Section 4.3, 4.4, respectively.

4.1 A property of graphs that are far from small set expander

The following lemma shows that if a general graph G is far from (k, ϕ) -expander, then there exist disjoint subsets such that each of them is of small size and small conductance, and the total volume of these sets are large. This lemma will be useful for the analysis of all the testers.

► **Lemma 13.** *Let c_2 be some constant and let $\phi^* \leq \frac{1}{20c_2}$. If a graph G is ε -far from (k^*, ϕ^*) -expander, then there exist disjoint subsets $S_1, \dots, S_q \subseteq V$ such that $\text{vol}(S_1 \cup \dots \cup S_q) \geq \frac{\varepsilon m}{15}$, and for each $i \leq q$, $\text{vol}(S_i) \leq 2k^*$, $\phi(S_i) < 11c_2\phi^*$.*

4.2 A tester for bounded degree graphs

Now we give a two-sided error tester for bounded degree graphs. This tester is very intuitive and simple: we sample a small number of vertices, and for each sampled vertex v , we perform independently a number of random walks from v and calculate the number of collisions Z_v between the endpoints of these random walks. We accept the graph if and only if Z_v is small for every sampled vertex v . We remark that this idea originates from the tester for expansion for bounded degree graphs [16, 11, 20, 28]. The main difference between our small set expansion tester and the previous expansion testers is the choice of parameters.

Given a d -bounded degree graph G , we define the following d -regularized random walk on G : at each vertex v , with probability $\text{deg}_G(v)/2d$, we jump to a randomly chosen neighbor of v , and with the remaining probability $1 - \frac{\text{deg}_G(v)}{2d}$, we stay at v . This random walk is equivalent to the lazy random walk on the virtually constructed d -regular graph G_{reg} that is obtained by adding an appropriate number of self-loops on each vertex in G . Note that to perform such a random walk, we only need to perform neighbor queries to the oracle of G . Our tester for bounded degree graphs is as follows.

SSETester2-Bound(G, s, r, ℓ, σ)

1. Repeat s times:
 - a. Select a vertex v uniformly at random from V .
 - b. Perform r independent d -regularized random walks of length ℓ starting from v .
 - c. Let Z_v be the number of pairwise collisions among the endpoints of these r random walks.
 - d. If $Z_v > \sigma$ then abort and output **reject**.
2. Output **accept**.

We can show that by choosing appropriate parameters, the above algorithm is a property tester for small set expansion for bounded degree graphs. We have the following theorem.

► **Theorem 14.** *Given neighbor query access to a d -bound-degree graph G , a volume bound k , a distance parameter ε and a conductance bound ϕ , then the algorithm **SSETester2-Bound** with parameters $s = \Theta(1/\varepsilon)$, $r = \Theta(\sqrt{n}/\varepsilon)$, $\ell = \Theta(\frac{(\ln k) \cdot \log(2nd/k)}{\phi^2})$ and $\sigma = \binom{r}{2} \frac{60}{k\varepsilon}$, accepts any (k, ϕ) -expander graph G with degree bounded by d and rejects any graph that is ε -far from (k^*, ϕ^*) -expander with degree bounded by d , where $k^* = \Theta(k\varepsilon/d)$, $\phi^* = \Theta(\frac{\phi^2}{(\ln k) \cdot \log(2nd/k)})$, with probability at least $2/3$. The query complexity and running time are $\tilde{O}(\sqrt{n}\phi^{-2}\varepsilon^{-2})$.*

4.3 Testers in the adjacency list model for general graphs

In this section, we give testers for small set expansion for general graphs in the adjacency list model.

4.3.1 A two-sided error tester

To give a two-sided error tester for general graphs, we first note that the tester for bounded degree graphs given in Section 4.2 does not apply to general graphs, which may have an

arbitrary large degree. For example, in a star graph the collision probability of a lazy random walk will be very large on the “central” vertex, however, the conductance of star graph is large and it is thus a small set expander. This implies that we cannot directly apply our tester for bounded degree graphs to general graphs.

In the following, we show that we can use the non-uniform replacement product (without rotation map) defined in Section 3 to first turn our input graph G into a bounded degree graph G' , and then we perform independent random walks on the newly transformed graphs G' to determine whether to accept or reject the input graph G . We should keep in mind that we are only given degree and neighbor query access to G rather than G' .

We first define G' . To do so, we first specify a proper d -regular graph family \mathcal{H} for G . We will let $d = 8$, and first turn G into a graph $G_{\geq 8}$ with minimum degree 8 by adding an appropriate number of self-loops on vertices with degree smaller than 8. Note that this modification only changes the conductance of a set by a factor of 8. Now we let \mathcal{H} be the graph family that for any $u \in G$, H_u is a Margulis expander with $\deg_{G_{\geq 8}}(u)$ vertices. We stress that such expanders are explicitly constructible [27, 13]. Furthermore, given any vertex $i \in H_u$, we can determine the neighborhood of i in constant time. Now we define $G' = G_{\geq 8} \circledast \mathcal{H}$.

By definition of G' , we can specify a vertex (u, i) to connect to a vertex in $\cup_{v:(v,u) \in E} H_v$ in an *arbitrary* manner. This important property allows us to construct G' when we go along and emulate random walks in G' very efficiently by performing degree and neighbor queries to G . We stress here that if the non-uniform replacement product with rotation map of G is used (see Section 4.4), then the neighbor of (u, i) in the final graph is fixed, and we do not know how to efficiently emulate the corresponding (lazy) random walks by only using degree and neighbor queries to G .

Now we briefly introduce a process for emulating random walks on G' . The argument is very similar to the analogous case given in Section 4.2 in [21]. We give a brief description here. To emulate random walks on G' , if we are currently at a vertex (u, i) , then with probability $1/2$, we stay at (u, i) ; with probability $1/4$, we jump to a randomly chosen neighbor (u, j) in H_u , which can be done in constant time since H_u is explicitly constructible; with the remaining probability $1/4$, we need to jump to the outside of H_u . Now if we have already specified its neighbor outside of H_u , say (v, j) , then we directly jump to (v, j) . Otherwise, we have to specify the outside neighbor of (u, i) first. The specification can be done by recording a set $A(u)$ of neighbors that has already been specified to some vertex in H_u and then either sampling new neighbors or attaching unspecified vertices arbitrarily according to $A(u)$. The amortized number of required degree and neighbor queries to G is $O(\log^2 n)$. We refer to [21] for more details.

There is one more issue that we should take care of: how to sample vertices (almost) uniformly at random from G' . This issue is almost equivalent to sampling edges almost uniformly from G , and has also been analyzed in [21]. In particular, Kaufman et al. have proved the following lemma.

► **Lemma 15** ([21]). *Let $\mu > 0$. There exists a procedure `Sample-Edges-Almost-Uniformly-in-G` that performs $O(\sqrt{n/\mu} \log m)$ degree and neighbor queries and for all but $(\mu/4)m$ of edges e in G , the probability that the procedure outputs e is at least $1/(64m)$. In particular, the output edge e is in the form of (v, i) for $1 \leq i \leq \deg(v)$.*

By setting $\mu = \varepsilon/c_3$ in the above lemma, for a sufficiently large constant c_3 , we will directly invoke `Sample-Edges-Almost-Uniformly-in-G` to sample a vertex (v, i) in G' .

Finally, to specify the number of random walks r , to be $O(\sqrt{m})$, we should have an estimate

of m or the average degree d_{avg} of G . This can be achieved by Feige’s algorithm [12, 18], which gives a constant factor estimate of d_{avg} by performing $O(\sqrt{n})$ queries to G .

Now we give a description of our two-sided error tester.

```

SSETester2-List( $G, s, r, \ell, \sigma$ )
1. Repeat  $s$  times:
  a. Sample an edge  $(v, i)$  by calling the procedure Sample-Edges-Almost-Uniformly-in- $G$  with  $\mu = \varepsilon/c_3$ , where  $c_3$  is a sufficiently large constant.
  b. Perform  $r$  independent lazy random walks in  $G_{\geq 8} \textcircled{R} \mathcal{H}$  of length  $\ell$  starting from  $v$  by the above emulation process.
  c. Let  $Z_v$  be the number of pairwise collisions among the endpoints of these  $r$  random walks.
  d. If  $Z_v > \sigma$  then abort and output reject.
2. Output accept.
    
```

By setting $s = \Theta(1/\varepsilon)$, $r = \Theta(\sqrt{m}/\varepsilon)$, $\ell = \Theta(\frac{\min\{\log(4m/k), \log n\} \cdot (\ln k)}{\phi^4})$ and $\sigma = \binom{r}{2} \frac{60}{k\varepsilon}$ in the algorithm **SSETester2-List**, we can prove Theorem 1 using similar analysis to the proof of Theorem 14.

4.3.2 A one-sided error tester

Now we present our property testing algorithm **SSETester1-List** with one-sided error for small set expansion. This tester invokes a local algorithm **LocalSS** introduced in Section 2.3 and applies to the adjacency list model.

```

SSETester1-List( $G, s, T, \delta$ )
1. Repeat  $s$  times:
  a. Sample an edge  $(v, i)$  by calling the procedure Sample-Edges-Almost-Uniformly-in- $G$  with  $\mu = \varepsilon/c_3$ , where  $c_3$  is a sufficiently large constant.
  b. If LocalSS( $G, v, T, \delta$ ) finds a set  $X$  with volume at most  $k$  and conductance at most  $\phi$ , then abort and output reject.
2. Output accept.
    
```

For any $0 < \xi < 1$, we set parameters $s = \Theta(1/\varepsilon)$, $T = O(\frac{\log k}{\phi^2})$, and $\delta = O(\frac{k^{-1+\xi/2}}{T})$ in the above algorithm, which will then be used to prove Theorem 2.

4.4 A tester in the rotation map model for general graphs

In this section, we give a tester in the rotation map model, in which we assume that the rotation map of G is explicitly given, that is, when specified a vertex v and an index i , the oracle returns a pair (u, j) such that u is the i th neighbor of v and j is the index of u as a neighbor of v . We use the non-uniform replacement product with rotation map to transform G into a 16-regular graph G' . To perform this transformation, we also need first to turn G into a graph $G_{\geq 8}$ with minimum degree 8, and specify \mathcal{H} to be a proper 8-regular Margulis expanders, and then let $G' = G_{\geq 8}^{(r)} \textcircled{R} \mathcal{H}$. Now the tester first samples a number of vertices almost uniformly in G' and then performs independent random walks on G' to decide whether to accept G or not, as we did before.

Our tester in rotation map model is almost the same as the two-sided tester in adjacency model in Section 4.3.1, and with information of the rotation map of G , we are actually able to give a better tester by using the spectral property of G' given in Lemma 12 (see Theorem 3).

However, as we mentioned before, since now we cannot specify the neighbor of a vertex (u, i) in an arbitrary manner, we do not know how to emulate random walks efficiently by only performing degree and neighbor queries to G . That is why we introduced (neighbor, index) query and the rotation map model.

Here we emulate random walks on G' by performing degree and (neighbor, index) queries to G : if we are currently at a vertex (u, i) , then with probability $1/2$, we stay at (u, i) ; with probability $1/4$, we jump to a randomly chosen neighbor (u, j) in H_u ; with the remaining probability $1/4$, we jump to vertex (v, j) such that v is the i th neighbor of u and u is the j th neighbor of v in G . Note that only in the last case, we need to perform (neighbor, index) queries to the oracle of G .

SSETester2-Map (G, s, r, ℓ, σ)

1. Repeat s times:

- a. Sample an edge (v, i) by calling the procedure **Sample-Edges-Almost-Uniformly-in- G** with $\mu = \varepsilon/c_3$, where c_3 is a sufficiently large constant.
- b. Perform r independent lazy random walks in $G_{\geq 8}^{(r)} \circledast H$ of length ℓ starting from v by using rotation map of G .
- c. Let Z_v be the number of pairwise collisions among the endpoints of these r random walks.
- d. If $Z_v > \sigma$ then abort and output **reject**.

2. Output **accept**.

Theorem 3 can now be proven by using the above algorithm **SSETester2-Map** with parameters $s = \Theta(1/\varepsilon)$, $r = \Theta(\sqrt{m}/\varepsilon)$, $\ell = \Theta(\frac{\min\{\log n, \log(4m/k)\} \cdot (\ln k)}{\phi^2})$ and $\sigma = \binom{r}{2} \frac{60}{k\varepsilon}$.

5 Conclusions

We give property testers for small set expansion in general graphs, including a two-sided error tester and a one-sided error tester in adjacency list model, and a two-sided error tester in rotation map model in which the algorithm can perform (neighbor, index) queries as well as degree queries. Our analysis for two-sided error testers uses a non-uniform replacement product to transform an arbitrary graph into a bounded degree graph that well preserves expansion profile.

It is unclear if the rotation map model is strictly stronger than the adjacency list model. In particular, we do not know if the newly introduced (neighbor, index) query is necessary for us to obtain a tester with at most quadratic loss in the conductance parameter. It will be interesting to give a two-sided error tester in the adjacency list model that distinguishes (k, ϕ) -expanders from graphs that are ε -far from any $(\Theta(k\varepsilon), \tilde{\Theta}(\phi^2))$ -expander, as we obtained in the rotation map model. It is also left open if the query complexity and/or running time of the two-sided testers could be improved to $\tilde{O}(\sqrt{n}(\phi^{-1}\varepsilon^{-1})^{O(1)})$, without dependency on the number of edges m .

Acknowledgements We would like to thank anonymous referees of RANDOM 2014 for their very detailed and helpful comments to an earlier version of this paper.

References

- 1 N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.
- 2 N. Alon, E. Fischer, I. Newman, and A. Shapira. A combinatorial characterization of the testable graph properties: it’s all about regularity. *SIAM Journal on Computing*, 39(1):143–167, 2009.
- 3 N. Alon and V. Milman. λ_1 , isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory, Series B*, 38(1):73–88, 1985.
- 4 N. Alon, O. Schwartz, and A. Shapira. An elementary construction of constant-degree expanders. *Comb. Probab. Comput.*, 17(3):319–327, May 2008.
- 5 R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In *Symposium on Foundations of Computer Science*, 2006.
- 6 R. Andersen and Y. Peres. Finding sparse cuts locally using evolving sets. In *Symposium on Theory of Computing*, STOC ’09, 2009.
- 7 S. Arora, B. Barak, and D. Steurer. Subexponential algorithms for unique games and related problems. In *Foundations of Computer Science (FOCS)*, pages 563–572, 2010.
- 8 B. Barak, P. Gopalan, J. Hastad, R. Meka, P. Raghavendra, and D. Steurer. Making the long code shorter. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 370–379. IEEE, 2012.
- 9 I. Ben-Eliezer, T. Kaufman, M. Krivelevich, and D. Ron. Comparing the strength of query types in property testing: the case of testing k -colorability. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1213–1222, 2008.
- 10 A. Czumaj, M. Monemizadeh, K. Onak, and C. Sohler. Planar graphs: Random walks and bipartiteness testing. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 423–432. IEEE, 2011.
- 11 A. Czumaj and C. Sohler. Testing expansion in bounded-degree graphs. *Combinatorics, Probability and Computing*, 19(5-6):693–709, 2010.
- 12 U. Feige. On sums of independent random variables with unbounded variance and estimating the average degree in a graph. *SIAM Journal on Computing*, 35(4):964–984, 2006.
- 13 O. Gabber and Z. Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, 1981.
- 14 O. Goldreich. *Property testing: current research and surveys*, volume 6390. Springer-Verlag New York Inc, 2010.
- 15 O. Goldreich and D. Ron. A sublinear bipartiteness tester for bounded degree graphs. *Combinatorica*, 19(3):335–373, 1999.
- 16 O. Goldreich and D. Ron. On testing expansion in bounded-degree graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(20), 2000.
- 17 O. Goldreich and D. Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.
- 18 O. Goldreich and D. Ron. Approximating average parameters of graphs. *Random Structures & Algorithms*, 32(4):473–493, 2008.
- 19 S. Hoory, N. Linial, and A. Wigderson. Expander graphs and their applications. *Bulletin of the American Mathematical Society*, 43(4):439–561, 2006.
- 20 S. Kale and C. Seshadhri. An expansion tester for bounded degree graphs. *SIAM J. Comput.*, 40(3):709–720, 2011.
- 21 T. Kaufman, M. Krivelevich, and D. Ron. Tight bounds for testing bipartiteness in general graphs. *SIAM Journal on computing*, 33(6):1441–1483, 2004.
- 22 T. C. Kwok and L. C. Lau. Finding small sparse cuts by random walk. In *APPROX-RANDOM*, pages 615–626, 2012.
- 23 J.R. Lee, S. Oveis Gharan, and L. Trevisan. Multi-way spectral partitioning and higher-order cheeger inequalities. *ACM symposium on Theory of computing*, 2012.

- 24 A. Li, Y. Pan, and P. Peng. Testing conductance in general graphs. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 18, page 101, 2011.
- 25 A. Louis, P. Raghavendra, P. Tetali, and S. Vempala. Many sparse cuts via higher eigenvalues. *ACM symposium on Theory of computing*, 2012.
- 26 L. Lovász and R. Kannan. Faster mixing via average conductance. In *ACM symposium on Theory of computing*, pages 282–287, 1999.
- 27 G. A. Margulis. Explicit constructions of expanders. *Problemy Peredachi Informatsii*, 9:71–80 (in Russian), 1973.
- 28 A. Nachmias and A. Shapira. Testing the expansion of a graph. *Information and Computation*, 208(4):309–314, 2010.
- 29 R. O’Donnell and D. Witmer. Improved small-set expansion from higher eigenvalues. *Arxiv preprint arXiv:1204.4688*, 2012.
- 30 S. Oveis Gharan and L. Trevisan. Approximating the expansion profile and almost optimal local graph clustering. In *Foundations of Computer Science*, 2012.
- 31 M. Parnas and D. Ron. Testing the diameter of graphs. *Random Structures & Algorithms*, 20(2):165–183, 2002.
- 32 P. Raghavendra and D. Steurer. Graph expansion and the unique games conjecture. In *ACM symposium on Theory of computing*, pages 755–764, 2010.
- 33 O. Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55:17:1–17:23, 2008.
- 34 O. Reingold, L. Trevisan, and S. Vadhan. Pseudorandom walks on regular digraphs and the rl vs. l problem. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 457–466. ACM, 2006.
- 35 O. Reingold, S. Vadhan, and A. Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders. *Annals of Mathematics*, 155:157–187, 2002.
- 36 D. Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2010.
- 37 E. Rozenman and S. Vadhan. Derandomized squaring of graphs. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 436–447. Springer, 2005.
- 38 A. Sinclair and M. Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82(1):93–133, 1989.
- 39 D.A. Spielman and S.H. Teng. A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. *arXiv:0809.3232*, 2008.
- 40 D. Steurer. On the complexity of unique games and graph expansion. *PhD diss., Princeton University*, 2010.

Paid Exchanges are Worth the Price

Alejandro López-Ortiz¹, Marc P. Renault^{*†2}, and Adi Rosén³

1 University of Waterloo, Canada
alopez-o@uwaterloo.ca

2 Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France
marc.renault@lip6.fr

3 CNRS and Université Paris Diderot, France
adiro@liafa.univ-paris-diderot.fr

Abstract

We consider the list update problem as defined in the seminal work on competitive analysis by Sleator and Tarjan [12]. In this problem, a sequence of requests, consisting of items to access in a linked list, is given. After an item is accessed it can be moved to any position forward in the list at no cost (free exchange), and, at any time, any two adjacent items can be swapped at a cost of 1 (paid exchange). The cost to access an item is its current position in the list. The goal is to dynamically rearrange the list so as to minimize the total cost (accrued from accesses and exchanges) over the request sequence.

We show a lower bound of 12/11 on the worst-case ratio between the performance of an (offline) optimal algorithm that can only perform free exchanges and that of an (offline) optimal algorithm that can perform both paid and free exchanges. This answers an outstanding question that has been open since 1996 [10].

1998 ACM Subject Classification F.2.2 Analysis of Algorithms and Problem Complexity (sequencing and scheduling)

Keywords and phrases list update problem, online computation, online algorithms, competitive analysis, lower bounds

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.636

1 Introduction

The *list update problem* consists of a linked list of ℓ items and a finite request sequence. Each request is to access an item of the list. Each item access begins at the head of the list and follows the list item by item until the requested item is reached. The cost to access the i -th item in the list is thus i . Then, the requested item can be moved forward in the list at no cost and such a move is called a *free exchange*. At any time, two adjacent items may be swapped at a cost of 1 and such swaps are called *paid exchanges*. The goal is to dynamically rearrange the list over the request sequence so as to minimize the total cost of accesses and paid exchanges over the request sequence.

The list update problem (also called the list access problem) was one of the two problems studied in the seminal work on competitive analysis of Sleator and Tarjan [12] (the other being the paging problem). It is a fundamental problem in the area of algorithms that has

* Research supported in part by ANR project NeTOC.

† Work performed while the author was at LIAFA, Université Paris Diderot – Paris 7, Sorbonne Paris-Cité, France.

been intensely studied, particularly, due to its importance for compression algorithms [5]. For a recent survey on the list update problem, see [8].

In [12], Sleator and Tarjan present a 2-competitive online deterministic algorithm called MOVE TO FRONT (MTF) that Irani showed later to be an optimal online deterministic algorithm [7]. As its name implies, MTF moves every requested item to the front, using a free exchange. Also, in [7], Irani presented the first online randomized algorithm for the list update problem; it has a competitive ratio of $15/8$. Reingold and Westbrook presented the first barely random online algorithm called BIT that has a competitive ratio of $7/4$ [11]. The best known randomized online algorithm, COMB, of Albers et al. [2] has a competitive ratio of 1.6 and only uses free exchanges. The COMB algorithm randomly uses the barely random online algorithm BIT with a probability of $4/5$ and the non-parameterized, deterministic online algorithm TIMESTAMP [1] with a probability of $1/5$. The currently best randomized online lower bound is 1.50115 [4]. It should be noted that all the best known online algorithms use only free exchanges [8].

The offline problem is known to NP-hard [3]. It is not known if this holds if only free exchanges are permitted. In [10], an algorithm that computes the optimal schedule that uses only paid exchanges is shown to have a running time of $O(2^\ell(\ell-1)n)$, where ℓ is the length of the list and n is the number of requests.¹ Based on the work of [10], an alternative algorithm that computes the optimal schedule, with a running time of $O(2^\ell \ell! f(\ell) + n + \ell n)$, where $f(\ell) \leq \ell! 3^{\ell!}$, is presented in [6].

Free vs. Paid Exchanges

In [12], Sleator and Tarjan claim that an algorithm that uses paid exchanges and free exchanges can be converted to an algorithm that uses only free exchanges without increasing the cost. This claim turns out not to be true as Reingold and Westbrook gave the counterexample of the request sequence $\langle 3, 2, 2, 3 \rangle$ for a list of length 3 with a starting configuration of 1, 2, 3 [10]. An optimal algorithm serves this sequence at a cost of 8 by moving item 1 to the back of list with paid exchanges at a cost of 2, and then serving the sequence at a cost of 6. From an enumeration of all possible schedules that use only free moves, it can be seen that an algorithm using only free exchanges serves this sequence for a cost of at least 9, implying that, in the worst case, there is at least an additive constant in the difference between the performance of an optimal algorithm that uses only free exchanges and an unrestricted optimal algorithm. Further, Reingold and Westbrook show that the opposite is true: they show that an algorithm can replace the free exchanges by paid exchanges without increasing the cost [10]. They also show that the permitted paid exchanges can be further restricted, without increasing the cost, to allow only "subset transfers" (see Definition 2 below).

The competitive ratio of 1.6 for the COMB algorithm [1] (as described above) implies an upper bound of 1.6 on the worst case ratio between the cost of an optimal algorithm restricted to free exchanges and the cost of an unrestricted optimal algorithm, over all finite request sequences.

Our Contribution

We compare the cost of an optimal algorithm that can only perform free exchanges, denoted by OPT_FREE, and an optimal algorithm that can use both paid and free exchanges, denoted

¹ As we indicate later, one can assume without loss of generality that the optimal schedule uses only paid exchanges.

by OPT. We show that there is a multiplicative gap of at least $12/11$ on the worst-case ratio, over all possible finite request sequences, between the performance of OPT_FREE and OPT. Until now, it was not known if there is such a gap in an asymptotic sense. We answer this question in the affirmative, thus solving a question that has been open for almost 20 years since Reingold and Westbrook [10] gave the counterexample to the claim of Sleator and Tarjan.

As all online algorithms with currently best known competitive ratios use only free exchanges [8], our result suggests that, in order to achieve better upper bounds, it may be useful to consider online algorithms that make use of paid exchanges.

2 Preliminaries

The *list update problem* consists of a linked list of ℓ items and a finite request sequence of accesses. Each request is to access an item of the list. Each item access begins at the head of the list and there is a cost of 1 to the algorithm for each item accessed until the requested item is found. That is, the cost to access the i -th item in the list is i . Then, the requested item can be moved forward to any position in the list at no cost and such a move is called a *free exchange*. At any time, two adjacent items may be swapped at a cost of 1 and these swaps are called *paid exchanges*. The goal is to dynamically rearrange the list over the request sequence so as to minimize the total cost of accesses and paid exchanges over the request sequence.

Note that, in the offline version of the list update problem (as defined above), the input is still a request sequence that must be served in order. The difference between the offline and online versions is that the offline algorithm has knowledge of the entire request sequence whereas, in the online version, a request is not revealed until all prior requests in the sequence have been served.

For an algorithm ALG and a request sequence σ , we denote the cost to ALG to serve σ by $\text{ALG}(\sigma)$.

We will use OPT to denote an unrestricted optimal (offline) algorithm, and we will use OPT_FREE to denote an optimal (offline) algorithm restricted to using only free exchanges. For the request sequences, we will denote multiple requests in a row to the same item by using exponents, e.g. x^k means that x is requested k times in a row.

In [9, 10], Reingold and Westbrook consider the offline version of the list update problem and show several properties of an offline optimum that uses both paid and free exchanges such as the following lemma.

► **Lemma 1.** [9][Cor. 3.2] *If an item x is requested 3 or more times consecutively, then an optimal offline algorithm must move it to the front before the second access.*

In [9, 10], Reingold and Westbrook also define the notion of a subset transfer and show that there exists an optimal algorithm that only performs such moves.

► **Definition 2 (Subset Transfer).** Let x be a requested item. A *subset transfer* is a move, performed just before x is accessed, of a subset of the items ahead of x in the list to the position immediately after x such that the relative order of the items in the subset is maintained.

► **Theorem 3.** [10][Thm. 2] *There is an optimal offline algorithm that does only subset transfers.*

Using Lemma 1 and Theorem 3, we get the following theorem that states that, for any sequence consisting of at least 3 consecutive requests to every item, MTF is OPT_FREE.

► **Theorem 4.** *Let $\sigma = \langle x_1^{k_1}, \dots, x_j^{k_j} \rangle$, where, for all i , $k_i \geq 3$ and, for $i < j$, $x_i \neq x_{i+1}$. For any initial list configuration, there exists an OPT_FREE that moves each x_i , $1 \leq i \leq j$, to the front of the list immediately after the first access to x_i of $x_i^{k_i}$ in σ .*

Proof. By Lemma 1, an (unrestricted) optimal algorithm must move each x_i , $1 \leq i \leq j$, of σ to the front before the second request to that item. Furthermore, by Theorem 3, there exists such optimal algorithm that only performs subset transfers; denote this optimal algorithm by OPT. Observe that if OPT does not move x_i to the front immediately before the first request to x_i , but does move x_i to the front immediately before the second request to x_i , then it cannot be optimal, since smaller cost could be achieved by moving x_i to the front immediately before the first request to x_i . We conclude that OPT is an optimal, subset-transfer-only, algorithm, that moves each x_i , $1 \leq i \leq j$, of σ to the front immediately before the first request to x_i . Observe now that since OPT is a subset-transfer-only algorithm, then OPT does not perform any other rearrangements in the list while processing σ .

The action by OPT of moving x_i to the front by subset transfer immediately before the first request to x_i , and then accessing x_i k_i times, can be accomplished for the same cost by an algorithm restricted to free exchanges. This is done by first accessing x_i (on the first request to x_i), then moving x_i to the front by a free exchange, and then accessing x_i for the remaining $k_i - 1$ times. It follows that there exists an algorithm restricted to free moves, that on σ moves every x_i , $1 \leq i \leq j$, to the front immediately after the first request to x_i , and its cost is equal to the cost of the optimal unrestricted algorithm for σ . This algorithm must therefore be OPT_FREE for σ . ◀

Informally, the next theorem shows that, on a series of sequential requests, it is not to the advantage of ALG_FREE to delay moving the requested item forward. That is, for an arbitrary algorithm that only performs free exchanges, denoted by ALG_FREE, and, for a sequence of consecutive requests to an item x , such that β is the position closest to the head of the list to which x is moved by the end of these consecutive requests, if ALG_FREE would move x to β immediately after the first request, it would not increase its cost. This holds for both offline and online algorithms, but online algorithms generally are not able take advantage of this fact given that they do not in general know the subsequent requests.

► **Theorem 5.** *Let $\sigma = \langle \sigma_1, \nu, \sigma_2 \rangle$, where ν is at least two consecutive requests to the same item x . Let β be the position of x immediately after ν for an arbitrary algorithm ALG_FREE. There exists an algorithm ALG_FREE' that moves x to β immediately after the first request of ν such that $\text{ALG_FREE}'(\sigma) \leq \text{ALG_FREE}(\sigma)$, and ALG_FREE' serves σ_1 and σ_2 exactly as ALG_FREE.*

Proof. The algorithm ALG_FREE' is defined to serve σ_1 in the same manner as ALG_FREE, to then move x to position β immediately after the first request of ν , and to serve σ_2 in the same manner as ALG_FREE. Note that the list configurations of ALG_FREE' and ALG_FREE match prior to and after serving ν . Therefore, the cost to both algorithms is the same for σ_1 and σ_2 .

Since ALG_FREE uses only free moves, i.e. moves of items towards the head of the list, it follows that the cost of ALG_FREE' for all requests in ν is no more than the cost of ALG_FREE for those requests. Therefore, $\text{ALG_FREE}'(\sigma) \leq \text{ALG_FREE}(\sigma)$. ◀

3 Lower Bound for OPT_FREE

In this section, we give a lower bound for the free move optimal offline algorithm as compared to the unrestricted optimal offline algorithm. That is, we are comparing the power of paid exchanges and free exchanges versus only free exchanges. We show that, for the case of a list of length at least 3, the ratio between the performance of OPT_FREE and that of OPT is at least $12/11 > 1.09$ in the worst case. More formally, we show that there exists an infinite family of finite request sequences σ_r , $r > 0$, such that the cost of an offline algorithm that can use paid exchanges, PAID, increases with r , and such that $\frac{\text{OPT_FREE}(\sigma_r)}{\text{OPT}(\sigma_r)} \geq \frac{\text{OPT_FREE}(\sigma_r)}{\text{PAID}(\sigma_r)} \geq 12/11$. This implies that, for any $\varepsilon > 0$ and any additive constant η that does not depend on the request sequence, there does not exist a free exchange algorithm, ALG_FREE, such that $\text{ALG_FREE}(\sigma) \leq (\frac{12}{11} - \varepsilon) \text{OPT}(\sigma) + \eta$ for all σ .

To prove the claim, we use a list of length 3 and we begin by defining a request sequence $R(L)$. For a given initial list configuration L , we define the request sequence $R(L)$ and a deterministic offline algorithm PAID that uses paid exchanges. By relabelling the list of PAID after having served $R(L)$ to match L , we can define an arbitrarily long request sequence σ_r consisting of repeated requests to $R(L)$ based on a relabelling of the list state of PAID after each $R(L)$. Our result applies to a list of length at least 3: If the list has a length greater than 3, we can ignore all but 3 items. Hence, without loss of generality, we only consider lists of length 3.

Line of Proof

As indicated above, our proof uses arbitrarily long request sequences, σ_r , $r \geq 1$ that are built by a repeated concatenation of r short request sequence $R(L)$, defined using a relabelling of the list state of PAID after each $R(L)$. We first prove two claims related to a single short request sequence $R(L)$. Namely, that PAID serves $R(L)$ starting with list configuration L at cost of 11; and that any OPT_FREE that serves $R(L)$ starting with list configuration L has cost at least 12. This however only repeats the claim of Reingold and Westbrook as to the existence of a request sequence with an additive difference between the optimal performance with free exchanges only and the optimal performance with both free and paid exchanges. We then concatenate these short request sequences to create a long request sequence. Observe that a multiplicative gap does not follow from such a concatenation. Indeed, an optimal algorithm that uses only free exchanges could potentially pay more than 12 for a given request sequence $R(L)$, reach a different list configuration, and then be able to serve the next $R(L)$ with cost less than 12, thus paying in total no more than 23 for the two sequences (or have such a phenomenon over a sequence of more than two sequences $R(L)$). To overcome this difficulty we prove that, for the long sequences that we consider, σ_r , any OPT_FREE must reach the same configuration as PAID does at the end of each $R(L)$. We can then conclude that for σ_r the cost of PAID is $11r$ and the cost of any OPT_FREE is at least $12r$.

Offline Paid Exchange Algorithm

For a list of length 3 with a starting list configuration $L = y, x_1, x_2$, we define the request sequence $R(L) = \langle x_2, x_1^3, x_2^3 \rangle$.

Let PAID be an unrestricted offline algorithm for $R(L)$ defined as follows. Before the first request of $R(L)$, using two paid exchanges, x_1 and x_2 are moved to the front of the list. Then, immediately before the second request to any x_i , $1 \leq i \leq 2$, PAID moves x_i to the front.

Immediately from the definition of PAID, we have the following facts.

► **Fact 6.** *Given a starting list configuration of $L = y, x_1, x_2$, after serving $R(L)$, the list configuration of PAID is x_2, x_1, y .*

► **Fact 7.** *Given a starting list configuration of $L = y, x_1, x_2$, the cost of PAID to serve $R(L)$ is 11.*

Proof. The cost to bring x_1, x_2 to the front by paid exchanges is 2 and the list configuration is now x_1, x_2, y . The cost of the first access to x_2 is 2, the cost to the next three requests of x_1 is 3. The second access to x_2 costs 2 and then x_2 is brought to the front and the remaining two accesses cost 2. Overall, the cost to PAID is 11. ◀

Arbitrarily Long Request Sequences

For an initial list configuration of $L = y, x_1, x_2$, from Fact 6, the configuration of the list of PAID after serving $R(L)$ is x_2, x_1, y . Therefore, after serving $R(L)$, with a relabelling of the list of PAID to that of L , $R(L)$ can subsequently be requested again, and this can be repeated to create arbitrarily long request sequences. That is, if $L' = x_2, x_1, y$ (as is the list configuration PAID after serving $R(L)$ for $L = y, x_1, x_2$), then $R(L') = \langle y, x_1^3, y^3 \rangle$.

Let $\sigma_r = \langle R_1(L_1), R_2(L_2), \dots, R_r(L_r) \rangle$ such that $R_j(L_j)$ is based on L_j , where L_j is the configuration of the list of PAID after serving R_1, \dots, R_{j-1} for $1 < j \leq r$ and $L_1 = L$ is the initial configuration of the list. We will use the term round to signify a subsequence $R(L)$ in σ_r .

Optimal (Offline) Free Exchange Algorithm

Let MTF be the algorithm that moves every requested item to the front. Immediately from the definition of MTF, we have the following fact.

► **Fact 8.** *Given a starting list configuration of $L = y, x_1, x_2$, after serving $R(L)$, the list configuration of MTF is x_2, x_1, y .*

Note that, when starting from the same initial list configuration and serving $R(L)$, the list configuration of MTF is exactly that of PAID after serving $R(L)$.

For an initial configuration $L = y, x_1, x_2$ and $R(L) = \langle x_2, x_1^3, x_2^3 \rangle$, the following lemma shows that MTF is an optimal free move algorithm for $R(L)$.

► **Lemma 9.** *For an initial list configuration $L = y, x_1, x_2$, $\text{MTF}(R(L)) = 12$ and $\text{OPT_FREE}(R(L)) = 12$.*

Proof. Irrespective of the specific free exchange algorithm, the access cost for the first request is 3 and there are 3 possible list configurations after the access. They are y, x_1, x_2 ; y, x_2, x_1 ; and x_2, y, x_1 (this last configuration corresponds to that of MTF). By Theorem 4, applied to the suffix of $R(L)$, after serving the first request of $R(L)$, $\langle x_1^3, x_2^3 \rangle$, every OPT_FREE moves x_1 and x_2 to the front of the list on the next request to each item. Table 1 summarizes the costs of the 3 possible ways to serve $R(L)$, as a function of the list configuration after the first request. The actions of MTF on $R(L)$ correspond to the x_2, y, x_1 column which is a minimum. ◀

We note that our proof will go through also if instead of using MTF we would use the algorithm that results in the list configuration as defined in the first configuration in the table.

■ **Table 1** For an initial list configuration of $L = y, x_1, x_2$, this table summarizes the potential optimal free exchange algorithms for $R(L) = \langle x_2, x_1^3, x_2^3 \rangle$. From Theorem 4, we know that after the first request every OPT_FREE moves all the items to the front of the list for the remaining requests. Therefore, the only variable is the configuration of the list immediately after the first request. Columns 3 – 5 represent the three possible list configurations. Column 1 is the index in $R(L)$ of the request listed in column 2. From the table, the first and third list configurations are optimal, and MTF corresponds to the third list configuration.

Request		List Configuration		
		y, x_1, x_2	y, x_2, x_1	x_2, y, x_1
1	x_2	3	3	3
2	x_1	2	3	3
3	x_1^2	2	2	2
6	x_2	3	3	2
7	x_2^2	2	2	2
Total:		12	13	12

The Last Round of σ_r

In the following lemma, we show that any OPT_FREE moves any item x to the front of the list immediately after the first access of three consecutive requests to x in $R_r(L_r)$ of σ_r , i.e. in the last round of σ_r .

► **Lemma 10.** For $\sigma_r = \langle R_1(L_1), \dots, R_r(L_r) \rangle$, every OPT_FREE moves any item x to the front of the list immediately after the first access of three consecutive requests to x in $R_r(L_r)$, where $L_1 = y, x_1, x_2$ and $L_j, 1 < j \leq r$, is the list configuration of PAID after serving $\langle R_1(L_1), \dots, R_{j-1}(L_{j-1}) \rangle$.

Proof. Let $L_r = y, x_1, x_2$ and let A be an arbitrary OPT_FREE algorithm. By way of contradiction, assume that A does not move some $x_i \in R_r(L_r)$ to the front immediately after the first request to $x_i^3 \in R_r(L_r)$.

Let $\sigma' = \langle R_1(L_1), \dots, R_{r-1}(L_{r-1}), x_2 \rangle$ and $\sigma'' = \langle x_1^3, x_2^3 \rangle$ (note that $\sigma_r = \langle \sigma', \sigma'' \rangle$). Define \hat{A} to be a free move algorithm that serves σ' exactly as A and moves all $x_j \in \sigma''$ to the front immediately after the first request to each item in σ'' .

Since A and \hat{A} serve σ' in the same manner, $\hat{A}(\sigma') = A(\sigma')$ and the list configurations of A and \hat{A} are the same immediately after σ' . From Theorem 4, \hat{A} is OPT_FREE over the remainder of the sequence. But, given the list configuration of both A and \hat{A} after serving σ' , starting with that list configuration, $\hat{A}(\sigma'') = \text{OPT_FREE}(\sigma'') < A(\sigma'')$. Therefore, $\hat{A}(\sigma_r) = A(\sigma') + \text{OPT_FREE}(\sigma'') < A(\sigma_r)$ which contradicts the fact that A is an optimal free exchange algorithm. ◀

The Rest of σ_r

In the next lemma, we show that the property proved in Lemma 10 for the last round of σ_r can be proved for all of σ_r . Namely, we show that for σ_r there exists an OPT_FREE that moves any item x to the front after the first access of any three consecutive requests to the same item. We note that Theorem 4 holds only for the specific type of sequence defined in the statement of that theorem, and that the property proved in the next lemma does not hold in general for an arbitrary sequence. For example, it can be verified that the sequence $\langle 5, 5, 5, 4, 3, 2, 1, 4, 3, 2, 1, 4, 3, 2, 1 \rangle$ (starting with list configuration 1, 2, 3, 4, 5) can be served

by OPT_FREE at cost of 44, while if ALG_FREE moves item 5 to the front immediately after the first request to item 5, then the cost of ALG_FREE is at least 45.

► **Lemma 11.** *For $\sigma_r = \langle R_1(L_1), \dots, R_r(L_r) \rangle$, there exists an OPT_FREE that moves any item x to the front of the list immediately after the first access of three consecutive requests to x , where $L_1 = y, x_1, x_2$ and $L_j, 1 < j \leq r$, is the list configuration of PAID after serving $\langle R_1(L_1), \dots, R_{j-1}(L_{j-1}) \rangle$.*

Proof. In this proof, for σ_r , we consider an arbitrary OPT_FREE algorithm A and show that, if the property does not hold for A , then there exists another OPT_FREE algorithm A' that does move every item x to the front of the list immediately after the first access of three consecutive requests to x , and such that $A'(\sigma_r) \leq A(\sigma_r)$. This will be done by defining a sequence of algorithms A_q , starting with $A_0 = A$, and by reverse induction on i and j over all the $x_i^3 \in R_j(L_j) \in \sigma_r$. That is, we consider the rounds from $R_r(L_r)$ to $R_1(L_1)$ and the consecutive three requests in each round from the last consecutive three requests to the first. For each x^3 , if A_q does not move x to the front immediately after the first request, we define A_{q+1} , based on A_q , such that the desired property holds for x^3 and all subsequent consecutive three requests, and we show that the cost does not increase.

In the proof, we use the following notations. Let x^3 be three consecutive requests in $R_j(L_j)$ for which A_q does not have the desired property. We will denote all the requests in σ_r before x^3 by σ_1 . The requests after x^3 will be denoted by σ_2 . Note that σ_2 could be an empty sequence. For the analysis, we will often (Case 2 and Case 3 below) further partition σ_2 into disjoint subsequences $\langle \sigma_3, \dots, \sigma_p \rangle$ such that $\sigma_r = \langle \sigma_1, x^3, \sigma_3, \dots, \sigma_p \rangle$. At a risk of a slight abuse of notation, we will denote the cost of a subsequence of an arbitrary σ_r to an algorithm, ALG, that serves all of σ_r , as $\text{ALG}(r_i, \dots, r_j) = \text{ALG}(r_1, \dots, r_j) - \text{ALG}(r_1, \dots, r_{i-1})$, where the prefix and the suffix are understood implicitly. That is, $\text{ALG}(r_i, \dots, r_j)$ is the cost accrued by ALG over the requests r_i, \dots, r_j of σ_r given that ALG has served the prefix r_1, \dots, r_{i-1} and will serve the remaining requests. Therefore, we have that $\text{ALG}(\sigma_r) = \text{ALG}(\sigma_1) + \text{ALG}(x^3) + \text{ALG}(\sigma_3) + \dots + \text{ALG}(\sigma_p)$. Further note that by Theorem 5, we can assume without loss of generality that A_q does not move x further ahead in the list on the second or third requests of x^3 .

Definition of \hat{A}_q

We first define an algorithm \hat{A}_q that we use extensively in the proof. For $\sigma = \langle \sigma_1, x_i^3, \sigma_2 \rangle$ and algorithm A_q as defined previously, let \hat{A}_q be an algorithm that serves σ_1 in the same manner as A_q and then moves x_i from position $\alpha > 1$ to the front of the list at the first request of x_i . Immediately after serving x_i , the configuration of the list of A_q is some B, x_i, C and the configuration of the list of \hat{A}_q is x_i, B, C , where B is the set of items ahead of x_i in the configuration of A_q at this point and C is the set of items behind x_i in the configuration of A_q . As long as the list configurations of A_q and \hat{A}_q differ, for each $x_j \in \sigma_2$, if A_q moves x_j to the front, \hat{A}_q moves x_j to front. Otherwise, \hat{A}_q does not move x_j forward at all. Once the list configurations of A_q and \hat{A}_q match, \hat{A}_q will serve the remaining requests exactly as A_q . Note that it is possible that the list configuration of \hat{A}_q may never match that of A_q (see Case 1 below).

From the definition of \hat{A}_q , and the fact that the list has length of 3, we have the following useful properties.

$$\hat{A}_q(\sigma_1) = A_q(\sigma_1) , \quad (1) \quad |C| = 2 - |B| , \quad (3)$$

$$|B| \geq 1 , \quad (2) \quad \beta = |B| + 1 , \quad (4)$$

where β is the position to which x_i is moved by A_q . Further, given that A_q moves x_i from α to β , $1 < \beta \leq \alpha$, and \hat{A}_q moves x_i from α to the front of the list, we have the following properties.

$$A_q(x_i^3) = \alpha + 2\beta, \tag{5}$$

$$\hat{A}_q(x_i^3) = \alpha + 2 \tag{6}$$

$$= A_q(x_i^3) - 2\beta + 2, \tag{7}$$

where (7) follows by replacing α in (6) by the value of α in (5).

We now turn to the inductive proof. For a list of length 3, there are two alternating list configurations for PAID (i.e. values for L_j) before each $R_j(L_j)$: y, x_1, x_2 and x_2, x_1, y . Therefore, x_1 is requested in every $R_j(L_j)$, and x_2 and y are requested in alternating $R_j(L_j)$'s.

For $x_i \in R_j(L_j)$, which is the last point in σ_r for which A_q does not move x_i to the front immediately after the first request of three consecutive requests, we can distinguish between three cases: (1) x_i is never requested again in σ_r ; (2) x_i is requested again in $R_{j+1}(L_{j+1})$, i.e. in the next round; (3) x_i is requested again in $R_{j+2}(L_{j+2})$, i.e. in the round after the next round. Note that this partitioning is exhaustive.

At each inductive step such that A_q does not have the desired property, we define an algorithm A_{q+1} based on A_q , for $q \geq 0$, and show that $A_{q+1}(\sigma_r) \leq A_q(\sigma_r)$. This is done by case analysis over the three cases defined above.

Case 1: $x_i \in R_j(L_j)$ is never requested again in σ_r .

Recall that $\sigma_r = \langle \sigma_1, x_i^3, \sigma_2 \rangle$. When $j = r$, this case follows immediately from Lemma 10 by defining A_{q+1} to be the algorithm defined in the proof of Lemma 10.

When $j < r$, we define A_{q+1} to be \hat{A}_q as defined above. For a list of length 3, this only occurs when $j = r - 1$ and $i = 2$, where $L_{r-1} = y, x_1, x_2$ and, hence, $R_{r-1}(L_{r-1}) = \langle x_2, x_1^3, x_2^3 \rangle$. For $L_{r-1} = y, x_1, x_2$, $R_r(L_r) = \langle y, x_1^3, y^3 \rangle$ and x_2 is not requested.

Denote $\sigma_1 = \langle R_1(L_1), \dots, R_{r-2}(L_{r-2}), x_2, x_1^3 \rangle$,

$\sigma_2 = \langle y, x_1^3, y^3 \rangle$.

Cost for $\sigma_2 = \langle y, x_1^3, y^3 \rangle$. By the induction hypothesis we know that A_q moves x_1 and y to the front of the list on the first request to x_1 and on the second request to y . It follows that the configurations of the lists of \hat{A}_q and A_q will match before the the third request to y is processed.

If y is in B , then the total cost to access y for \hat{A}_q over σ_2 is at most 2 more than that of A_q over σ_2 . This follows from the fact that there are two requests to y before A_q must move y to the front, according to the induction hypothesis and, if y is in B , then \hat{A}_q has x_1 in front of y , whereas A_q does not.

If y is in C , the total cost to access y for \hat{A}_q over σ_2 is at most 1 more than that of A_q over σ_2 . This can occur if, on the first access to y in σ_2 , A_q were to move y between x_1 and x_2 in its list. Then, on the second access, y is one item closer to the front in the list of A_q as compared to the list of \hat{A}_q .

By the induction hypothesis, A_q must move x_1 to the front on the first request to x_1 in σ_2 . Therefore, if x_1 is in B , the first access costs 1 more to \hat{A}_q as compared to A_q and, if x_1 is in C , the cost for the first access is the same for both \hat{A}_q and A_q .

This gives that for σ_2 ,

$$\begin{aligned} \hat{A}_q(\sigma_2) &\leq A_q(\sigma_2) + 2|B| + |C| - 1 \\ &= A_q(\sigma_2) + |B| + 1, \end{aligned} \tag{8}$$

where the last line follows by applying (3).

Using (1), we get

$$\begin{aligned} \hat{A}_q(\sigma_r) &= A_q(\sigma_1) + \hat{A}_q(x_i^3, \sigma_2) \\ &\leq A_q(\sigma_r) - 2\beta + |B| + 3, \text{ using (7) and (8),} \\ &= A_q(\sigma_r) - |B| + 1, \text{ using (4),} \\ &\leq A_q(\sigma_r), \text{ by (2).} \end{aligned}$$

Case 2: $x_i \in R_j(L_j)$ and $x_i \in R_{j+1}(L_{j+1})$, i.e. x_i is requested in the next round.

Recall that $\sigma_r = \langle \sigma_1, x_i^3, \sigma_2 \rangle$. For $L_j = y, x_1, x_2$, $R_j(L_j) = \langle x_2, x_1^3, x_2^3 \rangle$ and $R_{j+1}(L_{j+1}) = \langle y, x_1^3, y^3 \rangle$. For this case, we define A_{q+1} to be \hat{A}_q as defined above.

Define $\sigma_1 = \langle R_1(L_1), \dots, R_{j-1}(L_{j-1}), x_2 \rangle$,

$$\sigma_3 = \langle x_2^3, y, x_1^3 \rangle,$$

$$\sigma_4 = \langle y^3, R_{j+2}(L_{j+2}), \dots, R_r(L_r) \rangle.$$

Note that $\sigma_2 = \langle \sigma_3, \sigma_4 \rangle = \langle x_2^3, R_{j+1}(L_{j+1}), \dots, R_r(L_r) \rangle$.

Cost for $\sigma_3 = \langle x_2^3, y, x_1^3 \rangle$. After serving x_1^3 , the configuration of the list of \hat{A}_q is x_1, B, C and the list of A_q is B, x_1, C . By the induction hypothesis, A_q will move x_2 to the front on the first request to x_2 in σ_3 . This request and the request to y will each cost 1 more to \hat{A}_q than to A_q if they are in B . If they are in C , there is no additional cost to \hat{A}_q as compared to A_q . Finally, on the first request to x_1 in σ_3 , x_1 is no further from the front in \hat{A}_q than it is in A_q . Then, by the induction hypothesis, A_q moves x_1 to the front for the remaining requests to x_1 in σ_3 as does \hat{A}_q . Therefore,

$$\hat{A}_q(\sigma_3) \leq A_q(\sigma_3) + |B|. \tag{9}$$

List Configuration after σ_3 . By the induction hypothesis, A_q will move x_2 and x_1 to the front of the list immediately after the first access to each one in σ_3 . Consider the state of the lists of A_q and \hat{A}_q immediately after serving σ_3 , depending on whether or not A_q moves y to the front. If A_q does not move y to the front of the list, the configuration of its list will be x_1, x_2, y , and, by the definition of \hat{A}_q , its list configuration will also be x_1, x_2, y . If A_q does move y to the front of the list, the configuration of its list will be x_1, y, x_2 , and, by the definition of \hat{A}_q , its list configuration will also be x_1, y, x_2 .

Cost for $\sigma_4 = \langle y^3, R_{j+2}, \dots, R_r \rangle$. After serving σ_3 , the configurations of the lists of \hat{A}_q and of A_q are the same. Therefore,

$$\hat{A}_q(\sigma_4) = A_q(\sigma_4). \tag{10}$$

Summing (1), (7), (9), and (10), we get that the cost for \hat{A}_q over σ_r is

$$\begin{aligned} \hat{A}_q(\sigma_r) &\leq A_q(\sigma_r) - 2\beta + 2 + |B| \\ &= A_q(\sigma_r) - |B|, \text{ using (4),} \\ &< A_q(\sigma_r), \text{ by (2).} \end{aligned}$$

Case 3: $x_i \in R_j(L_j)$ and $x_i \in R_{j+2}(L_{j+2})$, i.e. x_i is requested in the round after next.

Recall that $\sigma_r = \langle \sigma_1, x_i^3, \sigma_2 \rangle$. We define A_{q+1} to be \hat{A}_q . For $L_j = y, x_1, x_2$, $R_j(L_j) = \langle x_2, x_1^3, x_2^3 \rangle$, $R_{j+1}(L_{j+1}) = \langle y, x_1^3, y^3 \rangle$, and $R_{j+2}(L_{j+2}) = \langle x_2, x_1^3, x_2^3 \rangle$.

Define $\sigma_1 = \langle R_1(L_1), \dots, R_{j-1}(L_{j-1}), x_2, x_1^3 \rangle$,

$\sigma_3 = \langle y, x_1^3, y^3 \rangle$, and

$\sigma_4 = \langle R_{j+2}(L_{j+2}), \dots, R_r(L_r) \rangle$.

Note that $\sigma_2 = \langle \sigma_3, \sigma_4 \rangle$.

Cost for $\sigma_3 = \langle y, x_1^3, y^3 \rangle$. After serving σ_1, x_2^3 , the configuration of the list of \hat{A}_q is x_2, B, C and the configuration of the list of A_q is B, x_2, C . By the induction hypothesis, A_q will move y to the front of the list on the second request to y in σ_3 and x_1 to the front of the list on the first request to x_1 in σ_3 . This is exactly the same scenario as σ_2 for Case 1. Similarly to (8) for Case 1 we have,

$$\hat{A}_q(\sigma_3) \leq A_q(\sigma_3) + |B| + 1. \quad (11)$$

List Configuration after σ_3 . Since both y and x_1 are moved to the front of the list in σ_3 by both A_q and \hat{A}_q , the configuration of the lists of both A_q and \hat{A}_q is y, x_1, x_2 after σ_3 .

Cost for $\sigma_4 = \langle R_{j+2}(L_{j+2}), \dots, R_r(L_r) \rangle$. After serving σ_3 , the configurations of the lists of \hat{A}_q and of A_q are the same. Therefore,

$$\hat{A}_q(\sigma_4) = A_q(\sigma_4). \quad (12)$$

Summing (1), (7), (11), and (12), we get that the cost for \hat{A}_q over σ_r is

$$\begin{aligned} \hat{A}_q(\sigma_r) &\leq A_q(\sigma_r) - 2\beta + |B| + 3 \\ &= A_q(\sigma_r) - |B| + 1, \text{ using (4),} \\ &\leq A_q(\sigma_r), \text{ by (2).} \end{aligned}$$

To conclude, for each of the three cases possible at each inductive step, we have shown that there exists an algorithm with the desired property. Overall, we have shown that $A'(\sigma_r) = A_q(\sigma_r) \leq \dots \leq A_0(\sigma_r) = A(\sigma_r)$ which concludes the proof. \blacktriangleleft

For σ_r as defined above, Lemma 11 shows that there exists an `OPT_FREE` that moves x to the front when x is requested at least three times in a row. Let `OPT_FREE*` be such an `OPT_FREE`. It follows that the list configuration of `OPT_FREE*` after each $R_j(L_j) \in \sigma_r$ is the same as that of `PAID`. For an initial list configuration of $L = y, x_1, x_2$, Lemma 9 shows that the algorithm `MTF` is an optimal free move algorithm for $R(L)$. Since the list configuration of `MTF` after serving $R_1(L_1), \dots, R_j(L_j)$, $1 \leq j \leq r$, is the same as `OPT_FREE*`, combined with the previous fact, this implies that `MTF` is an optimal free exchange algorithm for σ_r . Hence, `MTF` serves all $R_{j+1}(L_{j+1})$ at a cost no more than that of `OPT_FREE*`. This is formally stated in the following lemma.

► Lemma 12. For $\sigma_r = \langle R_1(L_1), \dots, R_r(L_r) \rangle$, $\text{MTF}(\sigma_r) = \text{OPT_FREE}(\sigma_r)$, where $L_1 = y, x_1, x_2$ and L_j , $1 < j \leq r$, is the list configuration of `PAID` after serving $\langle R_1(L_1), \dots, R_{j-1}(L_{j-1}) \rangle$.

Proof. By Lemma 11, there exists an `OPT_FREE` that will have the same configuration as `PAID` and `MTF` immediately before $R_j(L_j)$, $1 \leq j \leq r$. Let `OPT_FREE*` be such an `OPT_FREE`. Since the list configuration of `MTF` and `OPT_FREE*` match prior to serving every $R_j(L_j)$, Lemma 9 implies that $\text{MTF}(\sigma_r) = \text{OPT_FREE}(\sigma_r)$. ◀

Using the fact that, for any $r > 0$, `MTF` is an optimal free exchange algorithm for σ_r , we can, in the following lemma and theorem, give a lower bound on the worst-case ratio between `OPT_FREE`(σ) and `OPT`(σ) by analysing the ratio between `MTF`(σ_r) and `PAID`(σ_r) for $\sigma_r = \langle R_1(L_1), \dots, R_r(L_r) \rangle$, where $L_1 = y, x_1, x_2$ and L_j , $1 < j \leq r$, is the list configuration of `PAID` after serving $\langle R_1(L_1), \dots, R_{j-1}(L_{j-1}) \rangle$.

▶ **Lemma 13.** For $r > 0$, $\frac{\text{OPT_FREE}(\sigma_r)}{\text{OPT}(\sigma_r)} \geq \frac{\text{OPT_FREE}(\sigma_r)}{\text{PAID}(\sigma_r)} = \frac{12}{11} > 1.09$.

Proof. Let $\sigma = \langle R_1(L_1), \dots, R_r(L_r) \rangle$, where $L_1 = y, x_1, x_2$ and L_j , $1 < j \leq r$, is the list configuration of `PAID` after serving $\langle R_1(L_1), \dots, R_{j-1}(L_{j-1}) \rangle$.

From Lemma 12 and Lemma 9, `MTF` is an optimal free move algorithm for σ_r with a cost of $12r$ and, from Fact 7, the cost of `PAID` for σ_r is $11r$. Therefore, $\frac{\text{OPT_FREE}(\sigma_r)}{\text{OPT}(\sigma_r)} \geq \frac{\text{OPT_FREE}(\sigma_r)}{\text{PAID}(\sigma_r)} = \frac{12}{11}$. ◀

▶ **Theorem 14.** Let `ALG_FREE` be a free move algorithm such that $\text{ALG_FREE}(\sigma) \leq \alpha \cdot \text{OPT}(\sigma) + \eta$. For any η not dependent on σ , $\alpha \geq 12/11$.

Proof. Towards a contradiction, assume $\alpha = \frac{12}{11} - \varepsilon$ for some $\varepsilon > 0$. Hence, $\text{ALG_FREE}(\sigma_r) \leq (\frac{12}{11} - \varepsilon) \text{OPT}(\sigma_r) + \eta$. Solving for ε and σ_r such that $\text{ALG_FREE}(\sigma_r) > \eta$,

$$\varepsilon \leq \frac{12}{11} - \frac{\text{ALG_FREE}(\sigma_r) - \eta}{\text{OPT}(\sigma_r)} \leq \frac{12}{11} - \frac{\text{ALG_FREE}(\sigma_r) - \eta}{\text{PAID}(\sigma_r)} \leq \frac{\eta}{\text{PAID}(\sigma_r)} \tag{13}$$

by the fact that $\text{PAID}(\sigma_r) \geq \text{OPT}(\sigma_r)$ and Lemma 13. Since η does not depend on r , and $\text{ALG_FREE}(\sigma_r)$ and $\text{PAID}(\sigma_r)$ both increase with r , we have a contradiction by choosing a sufficiently large r such that $\text{ALG_FREE}(\sigma_r) > \eta$ and (13) no longer holds. ◀

4 Conclusions

We showed that the difference in the performance between an offline optimal algorithm restricted to free exchanges and an unrestricted offline optimal algorithm is at least a multiplicative factor of $12/11$, answering a question that has been open since 1996 [10].

Based on computer simulations, we believe that it should be possible to generalize the construction presented here and, based on this generalization, improve the lower bound to $3 - \sqrt{3}$.

Further, it would be interesting to consider upper bounds, in particular, an (offline) algorithm restricted to free exchanges that improves upon the 1.6 upper bound that follows from the randomized online algorithm `COMB` [2].

We note that the currently best known online algorithms use only free exchanges (cf. [8]). Our results bring up the possibility that improving the currently best randomized competitive ratio for the list update problem might necessitate introducing paid exchanges into the algorithm. The same might apply also to offline approximation algorithms.

Acknowledgements The authors would like to thank Amos Fiat, Rob van Stee, and Uri Zwick for useful discussions. We also wish to thank the anonymous reviewer who pointed out a minor change allowing an improvement of the lower bound from $13/12$ to $12/11$.

References

- 1 Susanne Albers. Improved randomized on-line algorithms for the list update problem. In Kenneth L. Clarkson, editor, *SODA*, pages 412–419. ACM/SIAM, 1995.
- 2 Susanne Albers, Bernhard von Stengel, and Ralph Werchner. A combined bit and timestamp algorithm for the list update problem. *Inf. Process. Lett.*, 56(3):135–139, 1995.
- 3 Christoph Ambühl. Offline list update is np-hard. In Mike Paterson, editor, *ESA*, volume 1879 of *Lecture Notes in Computer Science*, pages 42–51. Springer, 2000.
- 4 Christoph Ambühl. *On the List Update Problem*. PhD thesis, ETH Zürich, 2002.
- 5 Jon Louis Bentley, Daniel Dominic Sleator, Robert Endre Tarjan, and Victor K. Wei. A locally adaptive data compression scheme. *Commun. ACM*, 29(4):320–330, 1986.
- 6 Torben Hagerup. Online and offline access to short lists. In Ludek Kucera and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2007, 32nd International Symposium, MFCS 2007, Český Krumlov, Czech Republic, August 26-31, 2007, Proceedings*, volume 4708 of *Lecture Notes in Computer Science*, pages 691–702. Springer, 2007.
- 7 Sandy Irani. Two results on the list update problem. *Inf. Process. Lett.*, 38(6):301–306, 1991.
- 8 Shahin Kamali and Alejandro López-Ortiz. A survey of algorithms and models for list update. In Andrej Brodnik, Alejandro López-Ortiz, Venkatesh Raman, and Alfredo Viola, editors, *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, volume 8066 of *Lecture Notes in Computer Science*, pages 251–266. Springer, 2013.
- 9 Nick Reingold and Jeffery Westbrook. Off-line algorithms for the list update problem. Technical Report YALEU/DCS/TR-805, Yale University, 1990. <http://cpsc.yale.edu/sites/default/files/files/tr805.pdf>.
- 10 Nick Reingold and Jeffery Westbrook. Off-line algorithms for the list update problem. *Inf. Process. Lett.*, 60(2):75–80, 1996.
- 11 Nick Reingold, Jeffery Westbrook, and Daniel Dominic Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994.
- 12 Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.

Undecidability in Binary Tag Systems and the Post Correspondence Problem for Five Pairs of Words

Turlough Neary

Institute of Neuroinformatics, University of Zürich and ETH Zürich, Switzerland
tneary@ini.phys.ethz.ch

Abstract

Since Cocke and Minsky proved 2-tag systems universal, they have been extensively used to prove the universality of numerous computational models. Unfortunately, all known algorithms give universal 2-tag systems that have a large number of symbols. In this work, tag systems with only 2 symbols (the minimum possible) are proved universal via an intricate construction showing that they simulate cyclic tag systems. We immediately find applications of our result. We reduce the halting problem for binary tag systems to the Post correspondence problem for 5 pairs of words. This improves on 7 pairs, the previous bound for undecidability in this problem. Following our result, only the cases for 3 and 4 pairs of words remains open, as the problem is known to be decidable for 2 pairs. In a further application, we apply the reductions of Vesa Halava and others to show that the matrix mortality problem is undecidable for sets with six 3×3 matrices and for sets with two 18×18 matrices. The previous bounds for the undecidability in this problem was seven 3×3 matrices and two 21×21 matrices.

1998 ACM Subject Classification F.1.2 [Theory of Computation]: Computation by Abstract Devices—Modes of Computation

Keywords and phrases tag system, Post correspondence problem, undecidability

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.649

1 Introduction

Introduced by Post [17], tag systems have been used to prove Turing universality in numerous computational models, including some of the simplest known universal systems [6, 10, 11, 14, 19, 20, 21]. Many universality results rely either on direct simulation of tag systems or on a chain of simulations the leads back to tag systems. Such relationships between models means that improvements in one model often has applications to many others. The results in [23] are a case in point, where an exponential improvement in the time efficiency of tag systems had the domino effect of showing that many of the simplest known models of computation [6, 10, 11, 14, 19, 20, 21] are in fact polynomial time simulators of Turing machines. Despite being central to the search for simple universal systems for 50 years, tag systems have not been the subject of simplification since the sixties.

In 1961, Minsky [13] solved Post’s longstanding open problem by showing that tag systems, with deletion number 6, are universal. Soon after, Cocke and Minsky [5] proved that tag systems with deletion number 2 (2-tag systems) are universal. Later, Hao Wang [22] showed that 2-tag systems with even shorter instructions were universal. The systems of both Wang, and Cocke and Minsky use large alphabets and so have a large number of rules. Here we show that tag systems with only 2 symbols, and thus only 2 rules, are universal. Surprisingly, one of our two rules is trivial. We find immediate applications of our result. Using Cook’s [6] reduction of tag systems to cyclic tag systems, it is a straightforward matter

to give a binary cyclic tag system program that is universal and contains only two 1 symbols. We also use our binary tag system construction to improve the bound for the number of pairs of words for which the Post correspondence problem [18] is undecidable, and the bounds for the simplest sets of matrices for which the mortality problem [16] is undecidable.

The search for the minimum number of word pairs for which the Post correspondence problem is undecidable began in the 1980s [4]. The best result until now was found by Matiyasevich and Sénizergues, whose impressive 3-rule semi-Thue system [12], along with a reduction due to Claus [4], showed that the problem is undecidable for 7 pairs of words. Improving on this undecidability bound of 7 pairs of words seemed like a challenging problem. In fact, Blondel and Tsitsiklis [2] stated in their survey “The decidability of the intermediate cases ($3 \leq n \leq 6$) is unknown but is likely to be difficult to settle”. We give the first improvement on the bound of Matiyasevich and Sénizergues in 17 years: We reduce the halting problem for our binary tag system to the Post correspondence problem for 5 pairs of words. This leaves open only the cases for 3 and 4 pairs of words, as the problem is known to be decidable for 2 pairs [7].

A number of authors [1, 3, 8, 9, 16], have used undecidability bounds for the Post correspondence problem to find simple matrix sets for which the mortality problem is undecidable. The matrix mortality problem is, given a set of $d \times d$ integer matrices, decide if the zero matrix can be expressed as a product of matrices from the set. Halava et al. [9] proved the mortality problem undecidable for sets with seven 3×3 matrices, and using a reduction due to Cassaigne and Karhumäki [3] they also showed the problem undecidable for sets with two 21×21 matrices. Applying the reductions used in [3, 8] to our new bound, we find that the matrix mortality problem is undecidable for sets with six 3×3 matrices and for sets with two 18×18 matrices.

In the sequel while simulating cyclic tag systems our binary tag system produces garbage that grows exponentially during the simulation, and this results in an exponential time simulation overhead.

2 Preliminaries

We write $c_1 \vdash c_2$ if a configuration c_2 is obtained from c_1 via a single computation step. We let $c_1 \vdash^t c_2$ denote a sequence of t computation steps. The length of a word w is denoted $|w|$, and ϵ denotes the empty word. We let $\langle v \rangle$ denote the encoding of v , where v is a symbol or a word. We use the binary modulo operation $a = m \bmod n$, where $a = m - ny$, $0 \leq a < n$, and a, m, n , and y are integers.

2.1 Tag Systems

► **Definition 1.** A tag system consists of a finite alphabet of symbols Σ , a finite set of rules $R : \Sigma \rightarrow \Sigma^*$ and a deletion number $\beta \in \mathbb{N}$, $\beta \geq 1$.

The tag systems we consider are deterministic. The computation of a tag system acts on a word $w = w_0 w_1 \dots w_{|w|-1}$ (here $w_i \in \Sigma$) which we call the *dataword*. The entire configuration is given by w . In a computation step, the symbols $w_0 w_1 \dots w_{\beta-1}$ are deleted and we apply the rule for w_β , i.e. a rule of the form $w_\beta \rightarrow w_{0,1} w_{0,2} \dots w_{0,e}$, by appending the word $w_{0,1} w_{0,2} \dots w_{0,e}$ (here $w_{0,j} \in \Sigma$). A dataword (configuration) w' is obtained from w via a single computation step as follows:

$$w_0 w_1 \dots w_\beta \dots w_{|w|-1} \vdash w_\beta \dots w_{|w|-1} w_{0,1} w_{0,2} \dots w_{0,e}$$

where $w_0 \rightarrow w_{0,1}w_{0,2}\dots w_{0,e} \in R$. A tag system halts if $|w| < \beta$. We use the term *round* to describe the $\lfloor \frac{|w|}{\beta} \rfloor$ or $\lceil \frac{|w|}{\beta} \rceil$ computation steps that traverse a word w exactly once. We say a symbol w_0 is *read* if and only if at the start of a computation step it is the leftmost symbol (i.e. the rule $w_0 \rightarrow w_{0,0}w_{0,1}\dots w_{0,e}$ is applied), and we say a word $w = w_0w_1\dots w_{|w|-1}$ is *entered with shift* $z < \beta$ if w_z is the leftmost symbol that is read in w . We let $w_{[z]}^w$ denote the word obtained by removing the leftmost z symbols of w (i.e. $w_{[z]}^w = w_z\dots w_{|w|-1}$) and let $\overline{w}_{[z]}$ denote the sequence of symbols read during a single round on $w_{[z]}^w$. So $\overline{w}_{[z]} = w_zw_{z+\beta}w_{z+2\beta}w_{z+3\beta}, \dots, w_{z+l\beta}$ where $z+l\beta < |w|$. If $z < \beta$ then $\overline{w}_{[z]}$ is read when w is entered with shift z and we call $\overline{w}_{[z]}$ track z of w . A word w has a *shift change* of $0 \leq s < \beta$ if $|w| = y\beta - s$ where $y \in \mathbb{N}$ and $y > 0$. The proof of Lemma 2 is left to the reader.

► **Lemma 2 (shift change).** *Given a tag system T with deletion number β and the word $rv \in \Sigma^*$, where the word r has a shift change of s and $|v| \geq \beta$, after one round of T on r entered with shift z the word v is entered with shift $(z + s) \bmod \beta$.*

2.2 Cyclic Tag Systems

Cyclic tag systems were introduced and proved universal by Cook [6].

► **Definition 3.** A cyclic tag system $\mathcal{C} = \alpha_0, \dots, \alpha_{p-1}$ is a list of words $\alpha_i \in \{0, 1\}^*$ called *appendants*.

A *configuration* of a cyclic tag system consists of (i) a *marker* that points to a single appendant α_m in \mathcal{C} , and (ii) a word $w = w_1\dots w_{|w|} \in \{0, 1\}^*$. We call w the *dataword*. Intuitively the list \mathcal{C} is a program with the marker pointing to instruction α_m . In the initial configuration the marker points to appendant α_0 and w is the binary input word.

► **Definition 4.** A computation step is deterministic and acts on a configuration in one of two ways:

- If $w_1 = 0$ then w_1 is deleted and the marker moves to appendant $\alpha_{(m+1) \bmod p}$.
- If $w_1 = 1$ then w_1 is deleted, the word α_m is appended onto the right end of w , and the marker moves to appendant $\alpha_{(m+1) \bmod p}$.

A cyclic tag system completes its computation if (i) the dataword is the empty word or (ii) it enters a repeating sequence of configurations. As an example we give first 5 steps of the cyclic tag system $\mathcal{C} = 001, 01, 11$ on the input word 101. In each configuration \mathcal{C} is given on the left with the marked appendant highlighted in bold font.

$001, 01, 11 \quad 101 \quad \vdash \quad 001, \mathbf{01}, 11 \quad 01001 \quad \vdash \quad 001, 01, \mathbf{11} \quad 1001$
 $\vdash \quad \mathbf{001}, 01, 11 \quad 00111 \quad \vdash \quad 001, \mathbf{01}, 11 \quad 0111 \quad \vdash \quad 001, 01, \mathbf{11} \quad 111 \quad \vdash \quad \dots$

3 The Halting Problem for Binary Tag Systems

► **Definition 5 (Halting problem for binary tag systems).** Given a 2-symbol tag system \mathcal{T} with deletion number β and a dataword w , does \mathcal{T} produce a sequence of computation steps of the form $w \vdash^* w'$ where $|w'| < \beta$?

► **Theorem 6.** *The halting problem for binary tag systems is undecidable.*

The proof of Theorem 6 proceeds in two stages. First we construct a non-halting binary tag system $\mathcal{T}_{\mathcal{C}}$ that simulates an arbitrary cyclic tag system \mathcal{C} (Sections 3.1 and 3.2). Following this in Lemma 9 we show how to modify our construction so that when simulating the cyclic tag system in [15] our system halts if and only if it is simulating a halting Turing machine.

■ **Table 1** Table defining u . In the middle column is the sequence of symbols (track) read in u when the object in the left column is entered with shift $(\beta - 4m) \bmod \beta$, where $\beta = 4p$ is the deletion number, α_m is a cyclic tag system appendant, and $\langle \sigma_i \rangle'$ is a binary word where $\langle \sigma_i \rangle' = bbcb$ if $\sigma_i = 0$, and $\langle \sigma_i \rangle' = bbbcb$ if $\sigma_i = 1$. Also $\langle \sigma_1 \rangle'$ is the word $\langle \sigma_1 \rangle'$ with its leftmost symbol removed.
[1]

Object	Track read in u	Values for m and α_m
u	$\overline{u}_{[(\beta-4m) \bmod \beta]} = c^s$	$0 \leq m < p$
$\langle \epsilon \rangle = bubb$	$\overline{u}_{[\beta-1]} = cbbb(bcbbb)^{p-1}c^{s-5p+1}$ $\overline{u}_{[\beta-4m-1]} = c^s$	$m = 0$ $0 < m < p$
$\langle 0 \rangle = bbubb$	$\overline{u}_{[\beta-2]} = cbbb(bcbbb)^{p-1}c^{s-5p+1}$ $\overline{u}_{[\beta-4m-2]} = c^s$	$m = 0$ $0 < m < p$
$\langle 1 \rangle = bbub$	$\overline{u}_{[\beta-3]} = cbbb(bcbbb)^{p-1}c^{s-5p+1}$ $\overline{u}_{[\beta-3]} = \langle \sigma_1 \rangle' \langle \sigma_2 \rangle' \dots \langle \sigma_v \rangle' c^{s-5v+1}$ $\overline{u}_{[\beta-4m-3]} = \langle \sigma_1 \rangle' \langle \sigma_2 \rangle' \dots \langle \sigma_v \rangle' c^{s-5v}$	$\alpha_0 = \epsilon, \quad m = 0$ $\alpha_0 = \sigma_1 \sigma_2 \dots \sigma_v$ for $v > 0, \quad m = 0$ $\alpha_m = \sigma_1 \sigma_2 \dots \sigma_v$ for $0 < m < p$

3.1 Binary Tag System \mathcal{T}_C and Its Encoding

Here we give a binary tag system \mathcal{T}_C that simulates the computation of an arbitrary $\mathcal{C} = \alpha_0, \dots, \alpha_{p-1}$. The deletion number of \mathcal{T}_C is $\beta = 4p$, its alphabet is $\{b, c\}$, and its rules are of the form $b \rightarrow b$ and $c \rightarrow u$, where $u \in \{b, c\}^*$. The binary word u encodes the entire program of \mathcal{C} and is defined by Table 1, where $|u| = \beta s$, $s \geq 5(\max(p, r))$ and r is the length of the longest appendant in \mathcal{C} . See Section 3.3.1 for an example of how Table 1 is used to give u .

The cyclic tag system symbols 0 and 1 are encoded as the binary words $\langle 0 \rangle = bbubb$ and $\langle 1 \rangle = bbub$ respectively. We refer to $\langle 0 \rangle$ and $\langle 1 \rangle$ as objects.

► **Definition 7** (Input to \mathcal{T}_C). An arbitrary input dataword $w_1 w_2 \dots w_n \in \{0, 1\}^*$ to a cyclic tag system is encoded as the \mathcal{T}_C input dataword $\langle w_1 \rangle \langle w_2 \rangle \dots \langle w_n \rangle$.

During the simulation we make use of an extra *garbage* object: the binary word $\langle \epsilon \rangle = bubb$. The cyclic tag system configuration $(\alpha_0, \alpha_1 \dots \alpha_{m-1} \mathbf{\alpha}_m \alpha_{m+1} \dots \alpha_{p-1} \quad w_1 w_2 \dots w_l)$ is encoded as

$$\overline{\langle w_1 \rangle}_{[(\beta-4m) \bmod \beta]} \{ \langle \epsilon \rangle^p, u \}^* \langle w_2 \rangle \{ \langle \epsilon \rangle^p, u \}^* \langle w_3 \rangle \dots \{ \langle \epsilon \rangle^p, u \}^* \langle w_l \rangle \{ \langle \epsilon \rangle^p, u \}^* \quad (1)$$

where $\overline{\langle w_1 \rangle}_{[(\beta-4m) \bmod \beta]}$ denotes the word given by an object $\langle w_1 \rangle \in \{\langle 0 \rangle, \langle 1 \rangle\}$ with its leftmost $[(\beta - 4m) \bmod \beta]$ symbols deleted. This implies that $\langle w_1 \rangle$ is entered with the shift $[(\beta - 4m) \bmod \beta]$ and this shift value records that the currently marked cyclic tag system appendant is α_m . If a u subword in the dataword of \mathcal{T}_C does not form part of one of the three objects $\langle 0 \rangle$, $\langle 1 \rangle$ and $\langle \epsilon \rangle$ we will refer to this u subword as a garbage object. So words of the form $\{ \langle \epsilon \rangle, u \}^*$ in Equation (1) consist only of garbage objects.

3.2 The Simulation Algorithm

The sequence of symbols that is read in an object is determined by the shift value with which it is entered (see Section 2.1). So in the simulation the shift value is used for algorithm

$$\begin{array}{ll}
 \text{(i)} & \langle 1 \rangle_{[\beta-4m]} \langle a_2 \rangle \langle a_3 \rangle \dots \langle a_l \rangle \quad \vdash^s \quad \langle a_2 \rangle_{[(\beta-4(m+1)) \bmod \beta]} \langle a_3 \rangle \dots \langle a_l \rangle \langle \sigma_1 \rangle \langle \sigma_2 \rangle \dots \langle \sigma_v \rangle u^{s-5v} \\
 \text{(ii)} & \langle 0 \rangle_{[\beta-4m]} \langle a_2 \rangle \langle a_3 \rangle \dots \langle a_l \rangle \quad \vdash^s \quad \langle a_2 \rangle_{[(\beta-4(m+1)) \bmod \beta]} \langle a_3 \rangle \dots \langle a_l \rangle u^s \\
 \text{(iii)} & \langle \epsilon \rangle_{[\beta-4m]} \langle a_2 \rangle \langle a_3 \rangle \dots \langle a_l \rangle \quad \vdash^s \quad \langle a_2 \rangle_{[(\beta-4(m+1)) \bmod \beta]} \langle a_3 \rangle \dots \langle a_l \rangle u^s \\
 \text{(iv)} & u_{[(\beta-4m) \bmod \beta]} \langle a_2 \rangle \langle a_3 \rangle \dots \langle a_l \rangle \quad \vdash^s \quad \langle a_2 \rangle_{[(\beta-4m) \bmod \beta]} \langle a_3 \rangle \dots \langle a_l \rangle u^s
 \end{array}$$

■ **Figure 1** Objects $\langle 1 \rangle$, $\langle 0 \rangle$, $\langle \epsilon \rangle$ and u being read when entered with shift $[\beta - 4m]$. Above \vdash^s denotes the s computation steps that read each object, $a_i \in \{u, \langle 0 \rangle, \langle 1 \rangle, \langle \epsilon \rangle\}$, $\beta = 4p$ is the deletion number, and p is the length of \mathcal{C} 's program. In (i), (ii) and (iii) $0 < m < p$ and in (iv) $0 \leq m < p$. On the left is the dataword before the object is read and on the right is the dataword after the object has been read. After $\langle 1 \rangle$, $\langle 0 \rangle$ or $\langle \epsilon \rangle$ is read the adjacent object $\langle a_2 \rangle$ is entered with shift $[\beta - 4(m + 1)]$, and in (iv) after u is read $\langle a_2 \rangle$ is entered with shift $[(\beta - 4m) \bmod \beta]$. Reading the objects $\langle 0 \rangle$, $\langle \epsilon \rangle$ or u appends u^s , and reading the object $\langle 1 \rangle$ appends the encoding of cyclic tag system appendant $\alpha_m = \sigma_1 \sigma_2 \dots \sigma_v$.

control flow. Figures 1 and 2 give a high level view of reading each of the four objects that appear in Equation (1) and this view includes the shift change that occurs from reading each object. Objects $\langle 1 \rangle$, $\langle 0 \rangle$ and $\langle \epsilon \rangle$ have length $|\langle 1 \rangle| = |\langle 0 \rangle| = |\langle \epsilon \rangle| = \beta s + 4$ and so have a shift change of $\beta - 4$ (see end of Section 2.1). So when these objects are entered with shift $[(\beta - 4m) \bmod \beta]$ the adjacent object $\langle a_2 \rangle$ is entered with shift $[(\beta - 4(m + 1)) \bmod \beta]$ (as shown in Figures 1 and 2). When reading $\langle 1 \rangle$ or $\langle 0 \rangle$ this shift change of $\beta - 4$ simulates the appendant marker moving from appendant α_m to $\alpha_{(m+1) \bmod p}$. Recall that $\beta = 4p$ and so reading p of the $\langle 0 \rangle$ and $\langle 1 \rangle$ objects has a shift change of 0 (since $0 = p(\beta - 4) \bmod \beta$). This shift change of 0 means that the encoding of the marked appendant returns to its original value after reading p objects, correctly simulating that the appendant marker beginning at α_m has moved through the entire length p circular program of \mathcal{C}' and returned to its original position of marking α_m . The garbage objects $\langle \epsilon \rangle$ and u in Equation (1) appear only in garbage words of the form $\{\langle \epsilon \rangle^p, u\}^*$ which have a shift change of 0 (since $|\langle \epsilon \rangle^p| = (1 + ps)\beta$ and $|u| = \beta s$) and so have no effect on the shift value when entering subsequent objects (see Lemma 2). In Figures 1 and 2 we see that reading an $\langle 1 \rangle$ simulates reading a 1 by appending the encoding of the marked appendant $\alpha_m = \sigma_1 \sigma_2 \dots \sigma_v$ and reading an $\langle 0 \rangle$ simulates reading a 0 by appending a garbage word from $\{\langle \epsilon \rangle^p, u\}^*$ to simulate that nothing is appended. In Figures 1 and 2 when the garbage objects $\langle \epsilon \rangle$ and u are read they append garbage words from $\{\langle \epsilon \rangle^p, u\}^*$ that have no effect on the simulation.

To help the reader, in Section 3.3 we define u for a specific example and use this value to give an example of our algorithm simulating a cyclic tag system computation step. Here we explain how Table 1 defines u so that when each object is read it appends the appendants shown in Figures 1 and 2. Recall that a track $\bar{w}_{[z]}$ is the sequence of symbols read in a word w when it is entered with shift z (see Section 2.1). Table 1 defines u by giving (in the middle column) each possible track in u (for detailed example see Section 3.3.1). Each u track is read when the object in the left column of Table 1 is entered with shift $[(\beta - 4m) \bmod \beta]$. To see this note that the number of b symbols that proceed the u word in each object causes a shift change which in turn causes the u track given in middle column of Table 1 to be read. For example the word bbb at the left end of $\langle 1 \rangle = bbbub$ has a shift change value of $\beta - 3$ (see Section 2.1) and thus when $\langle 1 \rangle$ is entered with shift $[(\beta - 4m) \bmod \beta]$ we enter u with shift $[\beta - 4m - 3]$. So when $m > 0$ and we read an $\langle 1 \rangle$, we see from row 8 of Table 1 that track $\bar{u}_{[\beta-4m-3]} = \langle \sigma_1 \rangle' \langle \sigma_2 \rangle' \dots \langle \sigma_v \rangle' c^{s-5v}$ is read. Applying the rules

$$\begin{array}{lll}
\text{(i)} & \langle 1 \rangle_{[0]} \langle a_2 \rangle \langle a_3 \rangle \dots \langle a_l \rangle & \vdash^{s+1} \quad \langle a_2 \rangle_{[\beta-4]} \langle a_3 \rangle \dots \langle a_l \rangle \langle \sigma_1 \rangle \langle \sigma_2 \rangle \dots \langle \sigma_v \rangle u^{s-5v+1} \\
\text{(ii)} & \langle 0 \rangle_{[0]} \langle a_2 \rangle \langle a_3 \rangle \dots \langle a_l \rangle & \vdash^{s+1} \quad \langle a_2 \rangle_{[\beta-4]} \langle a_3 \rangle \dots \langle a_l \rangle \langle \epsilon \rangle^p u^{s-5p+1} \\
\text{(iii)} & \langle \epsilon \rangle_{[0]} \langle a_2 \rangle \langle a_3 \rangle \dots \langle a_l \rangle & \vdash^{s+1} \quad \langle a_2 \rangle_{[\beta-4]} \langle a_3 \rangle \dots \langle a_l \rangle \langle \epsilon \rangle^p u^{s-5p+1}
\end{array}$$

■ **Figure 2** Objects $\langle 1 \rangle$, $\langle 0 \rangle$ and $\langle \epsilon \rangle$ being read when entered with shift 0. Above \vdash^{s+1} denotes the $s+1$ computation steps that read each object when entered with shift 0. Here $a_i \in \{u, \langle 0 \rangle, \langle 1 \rangle, \langle \epsilon \rangle\}$, $\beta = 4p$ is the deletion number, and p is the length of \mathcal{C} 's program. On the left is the dataword before the object is read and on the right is the dataword after the object has been read. After $\langle 1 \rangle$, $\langle 0 \rangle$ or $\langle \epsilon \rangle$ is read the adjacent object a_2 is entered with shift $[\beta - 4]$. Reading the objects $\langle 0 \rangle$ or $\langle \epsilon \rangle$ appends $\langle \epsilon \rangle^p u^{s-5p+1}$, and reading the object $\langle 1 \rangle$ appends the encoding of cyclic tag system appendant $\alpha_0 = \sigma_1 \sigma_2 \dots \sigma_v$.

$b \rightarrow b$ and $c \rightarrow u$ when reading this track appends the appendant $\langle \sigma_1 \rangle \langle \sigma_2 \rangle \dots \langle \sigma_v \rangle c^{s-5v}$ as shown in Figure 1 (i). For $m > 0$ the entire sequence of symbol read in $\langle 1 \rangle$, $\langle 0 \rangle$, and $\langle \epsilon \rangle$ are respectively given by the u tracks in rows 3, 5 and 8 of Table 1. One can see that applying the rules $b \rightarrow b$ and $c \rightarrow u$ to these tracks appends the correct appendant for each object read in Figure 1. For $m = 0$ we have a special case where to get the entire sequence of symbols read in each object we prepend an extra b to the u tracks given in rows 2, 4, 6 and 7 of Table 1 and now applying the rules $b \rightarrow b$ and $c \rightarrow u$ to this sequence appends the appendants shown in Figure 2. For example, to give the sequence read in $\langle 1 \rangle$ when $m = 0$, we prepend b to track $\bar{u}_{[\beta-3]} = \langle \sigma_1 \rangle' \langle \sigma_2 \rangle' \dots \langle \sigma_v \rangle' c^{s-5v+1}$ (row 7 of Table 1) to give the sequence $b \langle \sigma_1 \rangle' \langle \sigma_2 \rangle' \dots \langle \sigma_v \rangle' c^{s-5v+1} = \langle \sigma_1 \rangle' \langle \sigma_2 \rangle' \dots \langle \sigma_v \rangle' c^{s-5v+1}$ which appends $\langle \sigma_1 \rangle \langle \sigma_2 \rangle \dots \langle \sigma_v \rangle c^{s-5v+1}$ as shown in Figure 2 (i). Finally, in row 1 of Table 1 are the u tracks that give the sequence of symbols read in u garbage objects for all values of m .

3.2.1 Tag system $\mathcal{T}_{\mathcal{C}}$ Simulating an Arbitrary Computation Step of \mathcal{C}

Equation (2) gives an arbitrary computation step of cyclic tag system \mathcal{C} , where $w' = \sigma_1 \dots \sigma_v$ if $w_1 = 1$ and $\alpha_m = \sigma_1 \sigma_2 \dots \sigma_v$, and $w' = \epsilon$ if $w_1 = 0$.

$$\alpha_0, \dots, \alpha_m \alpha_{m+1} \dots \alpha_{p-1} \quad w_1 w_2 \dots w_l \quad \vdash \quad \alpha_0, \dots, \alpha_m \alpha_{m+1} \dots \alpha_{p-1} \quad w_2 \dots w_l w' \quad (2)$$

Lemma 8 essentially states that $\mathcal{T}_{\mathcal{C}}$ simulates the arbitrary computation step given in Equation (2). In Lemma 8, Equations (3) and (4) respectively encode the left and right configurations in Equation (2) (see Equation (1)).

► **Lemma 8** ($\mathcal{T}_{\mathcal{C}}$ simulates an arbitrary computation step of \mathcal{C}). *Given a dataword of the form*

$$\langle w_1 \rangle_{[(\beta-4m) \bmod \beta]} \{ \langle \epsilon \rangle^p, u \}^* \langle w_2 \rangle \{ \langle \epsilon \rangle^p, u \}^* \langle w_3 \rangle \dots \{ \langle \epsilon \rangle^p, u \}^* \langle w_l \rangle \{ \langle \epsilon \rangle^p, u \}^* \quad (3)$$

where $w_i \in \{0, 1\}$, $\mathcal{T}_{\mathcal{C}}$ reads the word $\langle w_1 \rangle_{[(\beta-4m) \bmod \beta]} \{ \langle \epsilon \rangle^p, u \}^*$ to give a dataword of the form

$$\langle w_2 \rangle_{[(\beta-4(m+1)) \bmod \beta]} \{ \langle \epsilon \rangle^p, u \}^* \langle w_3 \rangle \dots \{ \langle \epsilon \rangle^p, u \}^* \langle w_l \rangle \{ \langle \epsilon \rangle^p, u \}^* \langle w' \rangle \{ \langle \epsilon \rangle^p, u \}^* \quad (4)$$

where $\langle w' \rangle = \langle \sigma_1 \rangle \langle \sigma_2 \rangle \dots \langle \sigma_v \rangle$ if $w_1 = 1$ and $\alpha_m = \sigma_1 \sigma_2 \dots \sigma_v$, and $\langle w' \rangle = \epsilon$ if $w_1 = 0$.

Proof. We use Figures 1 and 2 to verify that given Configuration (3), \mathcal{T}_C produces Configuration (4). Following this we discuss the correctness of Figures 1 and 2.

From (i) and (ii) of Figures 1 and 2 there are 4 possible cases for reading $\langle w_1 \rangle$, where each case is determined by the value of $\langle w_1 \rangle \in \{ \langle 1 \rangle, \langle 0 \rangle \}$ and the shift ($[0]$ or $[\beta - 4m]$ for $m > 0$) with which it is entered. The technique for verifying that \mathcal{T}_C produces Configuration (4) from Configuration (3) is similar for all 4 cases and so we will only go through one case. We choose the case where $\langle w_1 \rangle = \langle 1 \rangle$ and is entered with shift $[\beta - 4m]$ for $m > 0$. From Figure 1 (i) when we read $\langle w_1 \rangle = \langle 1 \rangle$ with shift $[\beta - 4m]$ in Configuration (3) we get

$$\{\langle \epsilon \rangle^p, u\}^*_{[(\beta-4(m+1)) \bmod \beta]} \langle w_2 \rangle \{\langle \epsilon \rangle^p, u\}^* \langle w_3 \rangle \dots \{\langle \epsilon \rangle^p, u\}^* \langle w_l \rangle \{\langle \epsilon \rangle^p, u\}^* \langle \sigma_1 \rangle \langle \sigma_2 \rangle \dots \langle \sigma_v \rangle u^{s-5v} \quad (5)$$

In Configuration (5) the word $\{\langle \epsilon \rangle^p, u\}^*$ is entered with shift $[(\beta - 4(m + 1)) \bmod \beta]$. Each $\langle \epsilon \rangle$ has a shift change of $\beta - 4$ and each u has a shift change of 0, and so as we read the word $\{\langle \epsilon \rangle^p, u\}^*$ the $\langle \epsilon \rangle$ and u objects are entered with shifts of the form $[(\beta - 4m') \bmod \beta]$ where $0 \leq m' < p$. So the cases for reading the objects in the word $\{\langle \epsilon \rangle^p, u\}^*$ are given by (iii) in Figures 1 and 2 and (iv) in Figure 1. Thus when we read the word $\{\langle \epsilon \rangle^p, u\}^*$ at the left end of Configuration (5) we append a word of the form $\{\langle \epsilon \rangle^p, u\}^*$ as shown in Configuration (6). Recall that words of the form $\{\langle \epsilon \rangle^p, u\}^*$ have a shift change of $[0]$ and so when we enter $\{\langle \epsilon \rangle^p, u\}^*$ with shift $[(\beta - 4(m + 1)) \bmod \beta]$ as shown in Configuration (5) we also enter the adjacent object $\langle w_2 \rangle$ with shift $[(\beta - 4(m + 1)) \bmod \beta]$ as shown in Configuration (6).

$$\langle w_2 \rangle_{[(\beta-4(m+1)) \bmod \beta]} \{\langle \epsilon \rangle^p, u\}^* \langle w_3 \rangle \dots \{\langle \epsilon \rangle^p, u\}^* \langle w_l \rangle \{\langle \epsilon \rangle^p, u\}^* \langle \sigma_1 \rangle \dots \langle \sigma_v \rangle u^{s-5v} \{\langle \epsilon \rangle^p, u\}^* \quad (6)$$

Configuration (6) is of the form of Configuration (4) for the case where $\langle w_1 \rangle = \langle 1 \rangle$ and is entered with shift $[(\beta - 4m) \bmod \beta]$ for $m > 0$. So for this case we have shown using Figures 1 and 2 that given Configuration (3) \mathcal{T}_C produces Configuration (4). The correctness of Figures 1 and 2 can be verified by showing that on each line of these figures, reading the leftmost object in the configuration on the left produces the configuration on the right. This is achieved by showing that when we enter the adjacent object $\langle a_2 \rangle$ with the correct shift and also that the correct word gets appended at the end of the dataword. The shift value with which $\langle a_2 \rangle$ is entered follows immediately from Lemma 2 and the length of the object being read. We direct the reader to paragraph 2 of Section 3.2 where we explain how Table 1 defines u so that when each object is read it appends the appendants shown in Figures 1 and 2. ◀

Our first main Theorem (Theorem 6) follows from Lemma 9. For tag system \mathcal{T}'_C in Lemma 9 we restrict the type of input allowed as this lets us simulate \mathcal{T}'_C in Theorem 11 when proving the Post correspondence problem undecidable for 5 pairs of words.

▶ **Lemma 9.** *Let \mathcal{T}'_C be an arbitrary binary tag system with deletion number β , alphabet $\{b, c\}$ and rules of the form $b \rightarrow b$ and $c \rightarrow u_0 u_1 \dots u_l b$ ($u_i \in \{b, c\}$). The halting problem is undecidable for tag systems of the form of \mathcal{T}'_C when given $u_{\beta-1} u_\beta \dots u_l b$ as input.*

Proof. In [15] the cyclic tag system \mathcal{C} simulates the computation of an arbitrary Turing machine and appends an appendant (which we will call α_h) encoding the Turing machine halt state if and only if the simulated Turing machine halts. So given the cyclic tag system \mathcal{C} , its input $w_1 w_2 \dots w_n$, and an appendant α_h , we construct a binary tag system \mathcal{T}'_C that takes $u_{\beta-1} u_\beta \dots u_l b$ as input and simulates \mathcal{C} on input w halting if and only if \mathcal{C} appends α_h . It then follows that halting problem is undecidable for tag systems of the form of \mathcal{T}'_C when given $u_{\beta-1} u_\beta \dots u_l b$ as input.

■ **Table 2** Table defining u for tag system in Lemma 9. In the middle column is the sequence of symbols (or track) read in u when the object in the left column is entered with shift $[(\beta - 10m + 1) \bmod \beta]$, where $\beta = 10p$ is the deletion number, α_m is a cyclic tag system appendant, $\langle \epsilon \rangle' = b^4 cb^6$, and $\langle \sigma_i \rangle' = b^6 cb^4$ if $\sigma_i = 0$, and $\langle \sigma_i \rangle' = b^8 cb^2$ if $\sigma_i = 1$. Also $\langle \epsilon \rangle'$ and $\langle \sigma_1 \rangle'$ are the words $\langle \epsilon \rangle'$ and $\langle \sigma_1 \rangle'$ with their leftmost symbol removed.

Object	Track read in u	Values for m and α_m
input track	$\bar{u}_{[\beta-1]} = b^{\beta-2} \langle w_1 \rangle' \dots \langle w_n \rangle' u^{s-11(n+p)-\beta+2} (\langle \epsilon \rangle')^p$	
u	$\bar{u}_{[(\beta-10m+1) \bmod \beta]} = c^s$	$0 \leq m < p$
$\langle \epsilon \rangle = b^4 ub^6$	$\bar{u}_{[\beta-3]} = \langle \epsilon \rangle' (\langle \epsilon \rangle')^{p-1} c^{s-11p+1}$ $\bar{u}_{[\beta-10m-3]} = (\langle \epsilon \rangle')^p c^{s-11p}$	$m = 0$ $0 < m < p$
$\langle 0 \rangle = b^6 ub^4$	$\bar{u}_{[\beta-5]} = \langle \epsilon \rangle' (\langle \epsilon \rangle')^{p-1} c^{s-11p+1}$ $\bar{u}_{[\beta-10m-5]} = (\langle \epsilon \rangle')^p c^{s-11p}$	$m = 0$ $0 < m < p$
$\langle 1 \rangle = b^8 ub^2$	$\bar{u}_{[\beta-7]} = \langle \sigma_1 \rangle' \langle \sigma_2 \rangle' \dots \langle \sigma_v \rangle' c^{s-11v+1}$ $\bar{u}_{[\beta-10m-7]} = \langle \sigma_1 \rangle' \langle \sigma_2 \rangle' \dots \langle \sigma_v \rangle' c^{s-11v}$ $\bar{u}_{[\beta-10h-7]} = bu^{s-1}$	$\alpha_0 = \sigma_1 \dots \sigma_v, \quad m = 0$ $\alpha_m = \sigma_1 \dots \sigma_v, \quad 0 < m < p, \quad m \neq h$
halting tracks	$\bar{u}_{[2i]} = b^s$	$i = \{0, 1, 2, 3, \dots, \frac{\beta-2}{2}\}$

We obtain \mathcal{T}'_C by modifying tag system \mathcal{T}_C from Section 3.1. In \mathcal{T}'_C , u is defined by Table 2 and we have $\langle 0 \rangle = b^6 ub^4$, $\langle \epsilon \rangle = b^4 ub^6$, $\langle 1 \rangle = b^8 ub^2$. The deletion number is now $\beta = 10p$, and $|u| = \beta s$ with $s = 11(\max(p+n+\beta-2, r))$. On input $u_{\beta-1} u_\beta \dots u_l b$ tag system \mathcal{T}'_C reads track $\bar{u}_{[\beta-1]}$ and from row 1 of Table 2 this appends $b^{\beta-2} \langle w_1 \rangle' \langle w_2 \rangle' \dots \langle w_n \rangle' u^{s-11(n+p)-\beta+2} (\langle \epsilon \rangle')^p$. So \mathcal{T}'_C begins its computation by appending the encoding of the input to \mathcal{C} . Reading $u_{\beta-1} u_\beta \dots u_l b$, which has a shift change of $\beta - 1$, causes us to enter the encoded dataword with a shift of $\beta - 1$ and this means that we can enter u garbage objects with shift $\beta - 1$ and read the track that appends the encoded input. To avoid this we append the word $b^{\beta-2}$ to the left of $\langle w_1 \rangle'$ and since $b^{\beta-2}$ has a shift change of 2 the encoded input is entered with shift [1] (instead of $[\beta - 1]$). After reading this $b^{\beta-2}$ \mathcal{T}'_C begins the simulation of \mathcal{C} on input w , making use of the same algorithm as \mathcal{T}_C . Since we enter the encoding of the input word w with shift [1] and objects $\langle 0 \rangle = b^6 ub^4$ and $\langle 1 \rangle = b^8 ub^2$ now have a shift change of $\beta - 10$, \mathcal{T}'_C enters objects with shifts of the form $[(\beta - 10m + 1) \bmod \beta]$. This means that when reading garbage object u and objects $\langle \epsilon \rangle$, $\langle 0 \rangle$, and $\langle 1 \rangle$, we enter u with shifts of $[(\beta - 10m + 1) \bmod \beta]$, $[(\beta - 10m - 3) \bmod \beta]$, $[(\beta - 10m - 5) \bmod \beta]$ and $[(\beta - 10m - 7) \bmod \beta]$ respectively. This gives the shift values for the tracks in Table 2. By comparing the tracks in Tables 1 and 2 we can see that the when objects in \mathcal{T}'_C are read they append similar sequences of objects to those in \mathcal{T}_C . So the computation of \mathcal{T}'_C proceeds in the same manner as the computation of \mathcal{T}_C . However, if \mathcal{C} appends α_h then we enter $\langle 1 \rangle$ with shift $[\beta - 6h]$ and track $\bar{u}_{[\beta-6h-4]} = bc^{s-1}$ is read appending the word bu^{s-1} . When bu^{s-1} is read during the next traversal of the dataword the single b in this word causes a shift change of $\beta - 1$ which means that all u subwords in the dataword of \mathcal{T}'_C will now be entered with an even valued shift. From row 10 of Table 1 all even valued tracks in u append only b symbols and so after one further traversal the dataword consists entirely of b symbols. After this the rule $b \rightarrow b$, which appends one b and deletes β symbols, is repeated until the number of symbols is $< \beta$ and the computation halts. ◀

■ **Table 3** Table defining u for the cyclic tag system $\mathcal{C} = 10, 0$. In the middle column is the sequence of symbols (track) read in u when the object in the left column is entered with shift $(8 - 4m) \bmod 8$, where α_m is a cyclic tag system appendant, $\langle 1 \rangle' = bbbcb$, $\langle 0 \rangle' = bcbb$ and $\langle 0 \rangle'_{[1]} = cbb$.

Object	Track read in u	Values for m and α_m
u	$\overline{u}_{[(8-4m) \bmod 8]} = c^{10}$	$0 \leq m < 2$
$\langle \epsilon \rangle = bubb$	$\overline{u}_{[7]} = cbbb(bcbbb)c$	$m = 0$
	$\overline{u}_{[3]} = c^{10}$	$m = 1$
$\langle 0 \rangle = bbubb$	$\overline{u}_{[6]} = cbbb(bcbbb)c$	$m = 0$
	$\overline{u}_{[2]} = c^{10}$	$m = 1$
$\langle 1 \rangle = bbub$	$\overline{u}_{[5]} = \langle 0 \rangle'_{[1]} c^6$	$\alpha_0 = 0$
	$\overline{u}_{[1]} = \langle 0 \rangle' \langle 1 \rangle'$	$\alpha_1 = 01$

3.3 Example Simulation for $\mathcal{T}_{\mathcal{C}}$

3.3.1 Using Table 1 to define u

To help explain how Table 1 is used to define u , we take the example of defining u for the cyclic tag system program $\mathcal{C} = 0, 01$. The value for u is given in Equation (7), where to improve readability, we have split u into two equal length subwords u' and u'' with a space between every fourth symbol.

$$\begin{aligned}
 u &= u'u'' & (7) \\
 u' &= cbcc\ cbcc\ cbcc\ cccb\ cccc\ cbbb\ cbcc\ cbbb\ cbcc\ cccb \\
 u'' &= cbcc\ cccc\ cbcc\ cccb\ cbcc\ cccb\ cccc\ cccb\ cbcc\ cccc
 \end{aligned}$$

From Section 3.1 when $\mathcal{C} = 0, 01$ we have $p = 2$, $s \geq 10$, $\beta = 8$, $|u| = 80$, $\alpha_0 = 0$ and $\alpha_1 = 01$. By substituting these values into Table 1 we get Table 3 which defines u for our tag system that simulates $\mathcal{C} = 0, 01$. Table 3 defines the word u as a series of tracks. Recall from Section 2.1 that a track $\overline{w}_{[z]} = w_z w_{z+\beta} w_{z+2\beta}, \dots, w_{z+l\beta}$ in a word w is the sequence of symbol read in that word when it is entered with shift z . Here $\beta = 8$ and so for $m = 0$, row 1 of Table 3 defines track $\overline{u}_{[0]} = u_0 u_8 u_{16} \dots u_{72} = c^{10}$, which is shown in bold below.

$$\begin{aligned}
 u' &= \mathbf{cbcc}\ cbcc\ \mathbf{cbcc}\ cccb\ \mathbf{cccc}\ cbbb\ \mathbf{cbcc}\ cbbb\ \mathbf{cbcc}\ cccb \\
 u'' &= \mathbf{cbcc}\ cccc\ \mathbf{cbcc}\ cccb\ \mathbf{cbcc}\ cccb\ \mathbf{cccc}\ cccb\ \mathbf{cbcc}\ cccc
 \end{aligned}$$

Taking row 2 of Table 3 gives $\overline{u}_{[7]} = u_7 u_{15} u_{23} \dots u_{79} = cbbb(bcbbb)c$ which again is given in bold immediately below.

$$\begin{aligned}
 u' &= cbcc\ cbcc\ \mathbf{cbcc}\ cccb\ cccc\ cbbb\ cbcc\ cbbb\ \mathbf{cbcc}\ cccb \\
 u'' &= cbcc\ cccc\ \mathbf{cbcc}\ cccb\ cbcc\ cccb\ cccc\ cccb\ \mathbf{cbcc}\ cccc
 \end{aligned}$$

Rows 3 to 7 of Table 3 give the tracks that complete the definition of u for our tag system that simulates the cyclic tag system $\mathcal{C} = 10, 0$.

3.3.2 Simulating a Computation Step with \mathcal{T}_C .

In this section we give the low level details of our simulation algorithm for \mathcal{T}_C by simulating the first computation step $(\mathbf{0}, 01 \ 11 \vdash \ 0, \mathbf{01} \ 10)$ of cyclic tag system $\mathcal{C} = 0, 01$ on the input dataword 11. The input dataword 11 to \mathcal{C} is encoded via Definition 7 as the tag system dataword $\langle 1 \rangle \langle 1 \rangle$ and using $\langle 1 \rangle = bbub$ and Equation (7) this can be rewritten as

$$\overbrace{bbb \ cbcc \ cbcc \ cbcc \ cccb \ cccc \ cbbb \ cbcc \ cbbb \ cbcc \ cccb}^{\langle 1 \rangle} u'' b \langle 1 \rangle \quad (8)$$

u'

Because the dataword is quite long we only give the left end of the dataword as b and c symbols and use higher level objects on the right. In configuration (8) the leftmost symbol is b and so we apply the rule $b \rightarrow b$ by appending b and deleting the leftmost 8 symbols from the dataword to give

$$\vdash \quad \underbrace{bcc \ cbcc \ cccb \ cccc \ cbbb \ cbcc \ cbbb \ cbcc \ cccb}_{u'_{[5]}} u'' b \langle 1 \rangle b \quad (9)$$

Above u is entered with shift 5 and so we begin reading track $\bar{u}_{[5]}$ from Table 3. We apply the rules $b \rightarrow b$ and $c \rightarrow u$ of \mathcal{T}_C to give the next four computation steps

$$\begin{aligned} \vdash & \quad cbb \ cccc \ cbbb \ cbcc \ cbbb \ cbcc \ cccb \ u'' b \langle 1 \rangle bb \\ \vdash & \quad bbb \ cbcc \ cbbb \ cbcc \ cccb \ u'' b \langle 1 \rangle bbu \\ \vdash & \quad bbb \ cbcc \ cccb \ u'' b \langle 1 \rangle bbub \\ \vdash & \quad cbb \ u'' b \langle 1 \rangle \underbrace{bbubb}_{\langle 0 \rangle} \end{aligned}$$

During the first 5 computation steps the word $\langle 0 \rangle = bbubb$ was appended to the dataword. Below we give a rewritten form of the configuration immediately above where $bbubb$ is replaced with $\langle 0 \rangle$ and u'' is replaced with its value from Equation (7). We also give the next 5 computation steps

$$\begin{aligned} & \quad cbb \ \underbrace{cbcc \ cccc \ cbcc \ cccb \ cbcc \ cccb \ cccc \ cccb \ cbcc \ cccc}_u b \langle 1 \rangle \langle 0 \rangle \\ \vdash^5 & \quad ccc b \langle 1 \rangle \langle 0 \rangle u^5 \end{aligned}$$

Below we have rewritten the configuration given immediately above and given the last configuration in this simulated computation step.

$$ccc b \ \overbrace{bbb \ cbcc \ cbcc \ cbcc \ cccb \ cccc \ cbbb \ cbcc \ cbbb \ cbcc \ cccb}^{\langle 1 \rangle} u'' b \langle 0 \rangle u^5 \quad (10)$$

u'

$$\vdash \quad \underbrace{bcc \ cbcc \ cbcc \ cccb \ cccc \ cbbb \ cbcc \ cbbb \ cbcc \ cccb}_{u'_{[4]}} u'' b \langle 0 \rangle u^6 \quad (11)$$

$u'_{[1]}$

In Equation (11) above we have finished reading the leftmost $\langle 1 \rangle$ in the dataword and completed our simulation of the computation step $(\mathbf{0}, 01 \quad 11 \vdash \mathbf{0}, \mathbf{01} \quad 10)$. The word $\langle 0 \rangle u^6$ was appended simulating appendant $\alpha_0 = 0$ from the program $\mathcal{C} = 0, 01$ was appended. From Section 3.1 the u subwords in u^6 are considered garbage objects and these u subwords have no effect on the computation (see first paragraph of Section 3.2). Also, in Equation (11) we see that the next $\langle 1 \rangle$ is entered with shift 4 which encodes that second appendant $\alpha_1 = 01$ in the program $\mathcal{C} = 0, 01$ is now marked. To see this recall that entering an object with a shift of $(\beta - 4m) \bmod \beta$ encodes that appendant α_m is marked and since $\beta = 8$ we get a shift of 4 when $m = 1$ meaning α_1 is marked. The dataword in Equation (11) is of the form given in Equation (1) and is ready to begin simulation of the next computations step.

4 The Post Correspondence Problem for 5 Pairs of Words

In Theorem 11 we reduce the halting problem for the binary tag system given in Lemma 9 to the Post correspondence problem for 5 pairs of words.

► **Definition 10** (Post correspondence problem). Given a set of pairs of words $\{(r_i, v_i) \mid r_i, v_i \in \Sigma^*, i \in \{0, 1, 2, \dots, n\}\}$ where Σ is a finite alphabet, determine whether or not there exists a non-empty sequence $i_1, i_2, \dots, i_l, \in \{0, 1, 2, \dots, n\}$ such that $r_{i_1} r_{i_2} \dots r_{i_l} = v_{i_1} v_{i_2} \dots v_{i_l}$.

► **Theorem 11.** *The Post correspondence problem is undecidable for 5 pairs of words.*

Proof. We reduce the halting problem for the binary tag systems $\mathcal{T}'_{\mathcal{C}}$ in Lemma 9 to the Post correspondence problem instance given by the 5 pairs of words

$$\mathcal{P} = \{(1, 1\langle u_0 \rangle \langle u_1 \rangle \dots \langle u_l \rangle 10), (10^\beta 1, 110), (10^\beta 1, 0), (1, 0), (10^\beta 1111, 1111)\}$$

where ϵ is the empty word and $u_i \in \{b, c\}$. The symbols b and c in $\mathcal{T}'_{\mathcal{C}}$ are encoded as $\langle b \rangle = 10^\beta 1$ and $\langle c \rangle = 1$ respectively, where β is the deletion number of $\mathcal{T}'_{\mathcal{C}}$. Let $r = r_{i_1} r_{i_2} \dots r_{i_l}$ and $v = v_{i_1} v_{i_2} \dots v_{i_l}$, where each $(r_i, v_i) \in \mathcal{P}$ and r is a prefix of v . We will call the pair (r, v) a configuration of \mathcal{P} . An arbitrary dataword $x_0 x_1 \dots x_q b \in \{b, c\}^* b$ is encoded by a \mathcal{P} configuration of the form

$$(r, v) = (r, r\langle x_0 \rangle \langle x_1 \rangle \dots \langle x_q \rangle 10^\beta) \tag{12}$$

In each configuration (r, v) , the unmatched part of v (i.e. $\langle x_0 \rangle \langle x_1 \rangle \dots \langle x_q \rangle 10^\beta$) encodes the current dataword of $\mathcal{T}'_{\mathcal{C}}$.

Starting from the pair $(1, 1\langle u_0 \rangle \langle u_1 \rangle \dots \langle u_l \rangle 10)$, if $u_0 = c$ we add the pair $(1, 0)$ and this matches $\langle c \rangle = 1$ simulating the deletion of u_0 . If, on the other hand, $u_0 = b$ we add the pair $(10^\beta 1, 0)$ and this matches $\langle b \rangle = 10^\beta 1$ simulating the deletion of u_0 . So after matching $\langle u_0 \rangle$ we have $(1\langle u_0 \rangle, 1\langle u_0 \rangle \langle u_1 \rangle \dots \langle u_l \rangle 100)$. We match $\beta - 1$ encoded $\mathcal{T}'_{\mathcal{C}}$ symbols in this way to give $(r, v) = (1\langle u_0 \rangle \dots \langle u_{\beta-2} \rangle, 1\langle u_1 \rangle \langle u_2 \rangle \dots \langle u_l \rangle 10^\beta)$. The configuration is now of the form given in Equation (12) and the unmatched sequence $\langle u_{\beta-1} \rangle \dots \langle u_l \rangle 10^\beta$ in v encodes the input dataword to $\mathcal{T}'_{\mathcal{C}}$ in Lemma 9.

A step of $\mathcal{T}'_{\mathcal{C}}$ on an arbitrary dataword $x_0 x_1 \dots x_q b$ is of one the two forms:

$$cx_1 \dots x_q b \vdash x_{\beta-1} \dots x_q b u_1 \dots u_l b \tag{13}$$

$$bx_1 \dots x_q b \vdash x_{\beta-1} \dots x_q b b \tag{14}$$

These computation steps are simulated as follows: In Equation (12), if $x_0 = c$ then $\langle x_0 \rangle = 1$ and we add the pair $(1, 1\langle u_0 \rangle \langle u_1 \rangle \dots \langle u_l \rangle 10)$ to simulate the $\mathcal{T}'_{\mathcal{C}}$ rule $c \rightarrow u_0 \dots u_l b$, and

this gives $(r1, r1\langle x_1 \rangle \dots \langle x_q \rangle 10^\beta 1 \langle u_0 \rangle \langle u_1 \rangle \dots \langle u_l \rangle 10)$. In Equation (12), if $x_0 = b$ then $\langle x_0 \rangle = 10^\beta 1$ and we add the pair $(10^\beta 1, 110)$ to simulate the \mathcal{T}'_C rule $b \rightarrow b$, giving $(r10^\beta 1, r10^\beta 1 \langle x_1 \rangle \dots \langle x_q \rangle 10^\beta 110)$. In both cases we simulate the deletion of a further $\beta - 1$ tag system symbols as we did in the previous paragraph to complete the simulated computation step. So if $x_0 = c$ this gives $(r1\langle x_1 \rangle \dots \langle x_{\beta-1} \rangle, r1\langle x_1 \rangle \dots \langle x_q \rangle 10^\beta 1 \langle u_0 \rangle \langle u_1 \rangle \dots \langle u_l \rangle 10^\beta)$, with the unmatched part in this pair encoding the new dataword on the right of Equation (13). Or, if $x_0 = b$ we get $(r10^\beta 1 \langle x_1 \rangle \dots \langle x_{\beta-1} \rangle, r10^\beta 1 \langle x_1 \rangle \dots \langle x_q \rangle 10^\beta 110^\beta)$, with the unmatched part in this pair encoding the new dataword on the right of Equation (14). The simulated computation step for both cases is now complete.

Now we show that \mathcal{P} simulates \mathcal{T}'_C halting with a matching pair of words. Recall that $|u| = s\beta$ where we can choose s to be any natural number greater than some constant (see Section 3.1). We choose s to be of the form $s = x(\beta - 1) + 1$ and so the input dataword $u_{\beta-1} \dots u_l b$ to \mathcal{T}'_C has length $x\beta(\beta - 1) + 1$ and the rules either increase its length by $x\beta(\beta - 1)$ (rule $c \rightarrow u_0 \dots u_l b$) or decrease it by $\beta - 1$ (rule $b \rightarrow b$), which means all datawords of \mathcal{T}'_C have length $y(\beta - 1) + 1$, where $y \in \mathbb{N}$. From Lemma 9, \mathcal{T}'_C halts when the length of its dataword (which has the form b^*) is less than the deletion number β . So, when \mathcal{T}'_C halts we have $y(\beta - 1) + 1 < \beta$ which means the dataword is a single b . From Equation (12), this is encoded as the configuration $(r, v) = (r, r10^\beta)$. By appending the pair $(10^\beta 1111, 1111)$ to (r, v) , we get the matching pair $(r10^\beta 1111, r10^\beta 1111)$ when \mathcal{T}'_C halts.

To complete our proof we show that choices that do not follow the simulation as described above leads to a mismatch. We must have $(1, 1\langle u_0 \rangle \langle u_1 \rangle \dots \langle u_l \rangle 10)$ as the leftmost pair as putting any other pair from \mathcal{P} on the left will not give a match. Now recall that the pair $(1, 1\langle u_0 \rangle \langle u_1 \rangle \dots \langle u_l \rangle 10)$ encodes the initial configuration (or input) for \mathcal{T}'_C . So now we show that given the encoding of an arbitrary configuration any choice that does not follow the simulation leads to a mismatch. From Equation (12) an arbitrary configuration has the form $(r, r\langle x_0 \rangle \langle x_1 \rangle \dots \langle x_q \rangle 10^\beta)$. If $x_0 = b$ then $\langle x_0 \rangle = 10^\beta 1$ and we cannot choose either of the pairs $(1, 1\langle u_0 \rangle \langle u_1 \rangle \dots \langle u_l \rangle 10)$ or $(1, 0)$ as they will allow no further matches, the pair $(10^\beta 1111, 1111)$ cannot be chosen as four consecutive 1s do not appear in the encoding (this is because in u we cannot have 2 encoded c symbols ($\langle c \rangle = 1$) next to each other). The pair $(10^\beta 1, 0)$ appends a 0 onto the right sequence to give $10^{\beta+1}$ which cannot be matched as it is only possible to match 0 sequences of the form $10^\beta 1$. Similar arguments are used for the case of $x_0 = c$ and so we do not repeat them. After we have matched $\langle x_0 \rangle$ our simulation algorithm requires that we simulate the deletion of a further $\beta - 1$ symbols. If we deviate from the simulation either we find almost immediately that no more matches are possible or we end up with a sequence of zeros that does not have the form $10^\beta 1$ and so cannot be matched. After simulating the deletion of $\beta - 1$ symbols we have completed the simulation of an arbitrary computation step and arrived at an encoded configuration of the form given by Equation (12). So it is not possible to find a match in \mathcal{P} if we do not follow the simulation described above. Therefore, \mathcal{P} has a matching sequence if and only if \mathcal{T}'_C halts. ◀

By applying the reductions in [8] and [3] to \mathcal{P} in Theorem 11 we get Corollary 12.

► **Corollary 12.** *The matrix mortality problem is undecidable for sets with six 3×3 matrices and for sets with two 18×18 matrices.*

Acknowledgements: This work is supported by Swiss National Science Foundation grant number 200021-141029. I would like to thank Matthew Cook, Damien Woods, Vesa Halava, and Mika Hirvensalo for their comments and discussions.

References

- 1 Vincent D. Blondel and John N. Tsitsiklis. When is a pair of matrices mortal? *Information Processing Letters*, 63(5):283–286, 1997.
- 2 Vincent D. Blondel and John N. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000.
- 3 Julien Cassaigne and Juhani Karhumäki. Examples of undecidable problems for 2-generator matrix semigroup. *TCS*, 204(1-2):29–34, 1998.
- 4 Volker Claus. Some remarks on PCP(k) and related problems. *Bull. EATCS*, 12:54–61, 1980.
- 5 John Cocke and Marvin Minsky. Universality of tag systems with $P = 2$. *Journal of the ACM*, 11(1):15–20, 1964.
- 6 Matthew Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1):1–40, 2004.
- 7 Andrzej Ehrenfeucht, Juhani Karhumäki, and Grzegorz Rozenberg. The (generalized) Post correspondence problem with lists consisting of two words is decidable. *TCS*, 21(2):119–144, 1982.
- 8 Vesa Halava and Tero Harju. Mortality in matrix semigroups. *American Mathematical Monthly*, 108(7):649–653, 2001.
- 9 Vesa Halava, Tero Harju, and Mika Hirvensalo. Undecidability bounds for integer matrices using Claus instances. *IJFCS*, 18(5):931–948, 2007.
- 10 Tero Harju and Maurice Margenstern. Splicing systems for universal Turing machines. In *DNA 10*, volume 3384 of *LNCS*, pages 149–158. Springer, 2005.
- 11 Kristian Lindgren and Mats G. Nordahl. Universal computation in simple one-dimensional cellular automata. *Complex Systems*, 4(3):299–318, 1990.
- 12 Yuri Matiyasevich and Géraud Sénizergues. Decision problems for semi-Thue systems with a few rules. *TCS*, 330(1):145–169. (An earlier version appeared in “11th Annual IEEE Symposium on Logic in Computer Science, LICS 1996”), 2005.
- 13 Marvin Minsky. Recursive unsolvability of Post’s problem of “tag” and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, 1961.
- 14 Marvin Minsky. Size and structure of universal Turing machines using tag systems. In *Recursive Function Theory: Proceedings, Symposium in Pure Mathematics*, volume 5, pages 229–238, Providence, 1962. AMS.
- 15 Turlough Neary and Damien Woods. P-completeness of cellular automaton Rule 110. In *ICALP 2006, Part I*, volume 4051 of *LNCS*, pages 132–143. Springer, 2006.
- 16 Michael S. Paterson. Unsolvability in 3×3 matrices. *Studies in Applied Mathematics*, 49(1):105–107, 1970.
- 17 Emil L. Post. Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65(2):197–215, 1943.
- 18 Emil L. Post. A variant of a recursively unsolvable problem. *Bulletin of The American Mathematical Society*, 52:264–268, 1946.
- 19 Raphael M. Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones Mathematicae*, 12(3):177–209, 1971.
- 20 Yurii Rogozhin. Small universal Turing machines. *TCS*, 168(2):215–240, 1996.
- 21 Paul Rothemund. A DNA and restriction enzyme implementation of Turing machines. In *DNA Based Computers*, volume 27 of *DIMACS*, pages 75–119. AMS, 1996.
- 22 Hao Wang. Tag systems and lag systems. *Mathematical Annals*, 152:65–74, 1963.
- 23 Damien Woods and Turlough Neary. On the time complexity of 2-tag systems and small universal Turing machines. In *47th Annual IEEE Symposium on Foundations of Computer Science*, pages 439–448, 2006.

Separation and the Successor Relation*

Thomas Place and Marc Zeitoun

LaBRI, Bordeaux University, France, firstname.lastname@labri.fr.

Abstract

We investigate two problems for a class \mathcal{C} of regular word languages. The \mathcal{C} -membership problem asks for an algorithm to decide whether an input language belongs to \mathcal{C} . The \mathcal{C} -separation problem asks for an algorithm that, given as input two regular languages, decides whether there exists a third language in \mathcal{C} containing the first language, while being disjoint from the second. These problems are considered as means to obtain a deep understanding of the class \mathcal{C} .

It is usual for such classes to be defined by logical formalisms. Logics are often built on top of each other, by adding new predicates. A natural construction is to enrich a logic with the successor relation. In this paper, we obtain new and simple proofs of two transfer results: we show that for suitable logically defined classes, the membership, resp. the separation problem for a class enriched with the successor relation reduces to the same problem for the original class.

Our reductions work both for languages of finite words and infinite words. The proofs are mostly self-contained, and only require a basic background on regular languages. This paper therefore gives simple proofs of results that were considered as difficult, such as the decidability of the membership problem for the levels 1, $3/2$, 2 and $5/2$ of the dot-depth hierarchy.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases separation problem, regular word languages, logics, decidable characterizations, semidirect product

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.662

1 Introduction

A central problem in the theory of formal languages is to characterize and understand the expressive power of high level specification formalisms. Monadic second order logic (MSO) is such a formalism, which is both expressive and robust. For several classes of structures, such as words or trees, it has the same expressive power as finite automata and defines the class of regular languages. In this paper, we investigate fragments of MSO over words. In this context, understanding the expressive power of a fragment is associated to two decision problems: the *membership problem* and the *separation problem*.

For a fixed logical fragment \mathcal{F} , the \mathcal{F} -*membership problem* asks for a decision procedure that tests whether some input regular language can be expressed by a formula from \mathcal{F} . To obtain such an algorithm, one has to consider and understand *all* properties that can be expressed within \mathcal{F} , which requires a deep understanding of the fragment \mathcal{F} . On the other hand, the \mathcal{F} -*separation problem* is more general. It asks for a decision procedure that tests whether given *two* input regular languages, there exists a third one in \mathcal{F} containing the first language while being disjoint from the second one.

Since regular languages are closed under complement, membership reduces to separation: a language is in \mathcal{F} if and only if it can be separated from its complement. Usually, the

* Supported by ANR 2010 BLAN 0202 01 FREC

separation problem is more difficult than the membership problem but also more rewarding with respect to the knowledge gained on the investigated fragment \mathcal{F} .

These two problems have been considered and solved for many natural fragments of monadic second order logic. Among these, the most prominent one is first-order logic, $\text{FO}(<)$, equipped with a predicate $<$ for the linear ordering. The solution to the membership problem, known as the McNaughton-Papert-Schützenberger Theorem [20, 10], has been revisited until recently [5]. The theorem states that a regular language is definable in $\text{FO}(<)$ if and only if its *syntactic semigroup* is aperiodic. The syntactic semigroup is a finite algebraic object that can be computed from any regular language. Since aperiodicity can be defined as an equation that needs to be satisfied by all of its elements, this yields decidability of $\text{FO}(<)$ -definability. This result now serves as a template, which is commonly followed in this line of research.

The separation problem has also been successfully solved for first-order logic [7]. Actually, the problem was first addressed in a purely algebraic framework, and was later identified as equivalent to our separation problem [2]. As for membership, this problem is still revisited today and a new self-contained and combinatorial proof was obtained in [18].

Motivation. We are interested in natural fragments of $\text{FO}(<)$ obtained by restricting either the number of variables or the number of quantifier alternations allowed in formulas. Such restrictions in general give rise to several variants of the same fragment. Indeed, in most cases, the drop in expressive power forbids the use of natural relations that could be defined from the linear order in $\text{FO}(<)$. The main example considered in this paper is $+1$, the *successor relation*, together with predicates *min* and *max* for the first and last positions in a word. This means that one can define two distinct variants of the same fragment depending on whether we decide to explicitly add these predicates in the signature or not. An example is the fragment Σ_n , which consists of first-order formulas whose prenex normal form has at most $(n - 1)$ quantifier alternations and starts with an existential block. Since defining $+1$ requires an additional quantifier alternation, $\Sigma_n(<, +1, \text{min}, \text{max})$ has indeed stronger expressiveness than $\Sigma_n(<)$. The motivation of this paper is to obtain decidability results for such enriched fragments.

State of the Art. Even when the weak fragment is known to have decidable membership, proving that the enriched one has the same property can be nontrivial. Examples include the membership proofs of $\mathcal{B}\Sigma_1(<, +1, \text{min}, \text{max})$ (Boolean combinations of $\Sigma_1(<, +1, \text{min}, \text{max})$ formulas) and $\Sigma_2(<, +1)$, which require difficult and intricate combinatorial arguments [8, 6, 9] or a wealth of algebraic machinery [12, 13]. Another issue is that most proofs directly deal with the enriched fragment. Given the jungle of such logical fragments, it is desirable to avoid such an approach, treating each variant of the same fragment independently. Instead, a satisfying approach is to first obtain a solution of the membership and separation problems for the less expressive variant and then to lift it to other variants via a generic transfer result.

This approach has first been investigated by Straubing for the membership problem [22] in an algebraic framework, and later adapted to be able to treat classes not closed under complement [13]. Transferring the logical problem to this algebraic framework requires preliminary steps, still *specific* to the investigated class, to prove that:

1. A language is definable in the fragment if and only if its syntactic semigroup belongs to a specific algebraic variety \mathcal{V} (e.g., the variety of aperiodic monoids for $\text{FO}(<)$), and
2. Membership to \mathcal{V} is decidable.

Next, though this is not immediate, for most fragments of $\text{FO}(<)$, it has been proved that

3. When the weaker variant corresponds to a variety \mathcal{V} , the variant with successor corresponds to the variety $\mathcal{V} * \mathcal{D}$, built generically from \mathcal{V} .

Hence, Straubing's approach was to prove that

4. the operator $\mathcal{V} \mapsto \mathcal{V} * \mathcal{D}$ preserves decidability.

Unfortunately, this is not true in general [3]. Actually, while decidability is preserved for all known logical fragments, there is no generic result that captures them all. In particular, for the less expressive fragments, one has to use completely *ad hoc* proofs. In the separation setting, things behave well: it has been shown that decidability of separation is preserved by the operation $\mathcal{V} \mapsto \mathcal{V} * \mathcal{D}$ [21]. While interesting when already starting from algebra, this approach has several downsides:

- Dealing with algebra hides the logical intuitions, while our primary goal is to understand the expressiveness of logics.
- Going from logic to algebra requires to be acquainted with new notions and vocabulary, as well as involved theoretical tools. Proofs are also often nontrivial and require a deep understanding of complex objects, which may be scattered in the bibliography.
- Despite step 4, which is generic to some extent, arguments specific to the investigated class are pushed to steps 1–3, and they are often nontrivial.

Contributions. We give a new proof that decidability of separation can be transferred from a weak to an enriched fragment. We present the result in two different forms.

The first one is non-algebraic: we work directly with the logical fragments, without using varieties. The transfer result is generic and its proof mostly is: the only specific argument is an Ehrenfeucht-Fraïssé game that can be adapted to all natural fragments with minimal difficulty (we prove it in the long version of this paper for all considered fragments, see [19]). The benefits of this new proof are that:

1. It is self-contained and much simpler than previous ones. It only relies on two basic well-known notions: recognizability by semigroups and Ehrenfeucht-Fraïssé games.
2. It works with classes that are not closed under complement, contrary to [21]. This allows us to capture the Σ and Π levels in the quantifier alternation hierarchy of first-order logic.
3. Under an additional hypothesis on the logical fragment, which is met for most fragments we investigate and easy to check, the decidability result of the separation problem also extends to the membership problem.
4. The proof adapts smoothly to infinite words using the notion of ω -semigroups, as shown in the long version of this paper [19].

The second form of our result is algebraic and generic. We prove that $\mathcal{V} \mapsto \mathcal{V} * \mathcal{D}$ preserves the decidability of separation for varieties, hence giving an elementary proof of a result of [21]. Even in this algebraic form, we completely bypass involved constructions or notions, such as pointlike sets for categories developed in [21], thus making the proof accessible.

As corollaries, since $\mathcal{B}\Sigma_1(<)$ and $\Sigma_2(<)$ both enjoy decidable separation [4, 16, 17], we obtain that this is also the case for the fragments $\mathcal{B}\Sigma_1(<, +1, \text{min}, \text{max})$ and $\Sigma_2(<, +1)$, known as levels 1 and 3/2 of the dot-depth hierarchy. These new results strengthen the previous ones [8, 6] that showed decidability of membership and were considered as difficult. We actually obtain that separation for $\Sigma_n(<, +1, \text{min}, \text{max})$ reduces to separation for $\Sigma_n(<)$. Since we also transfer decidability of the membership problem, and since the fragments $\mathcal{B}\Sigma_2(<)$ of Boolean combinations of $\Sigma_2(<)$ formulas and $\Sigma_3(<)$ have decidable membership [17] we

deduce that the same holds for $\mathcal{B}\Sigma_2(<, +1)$ and $\Sigma_3(<, +1)$, known as levels 2 and 5/2 of the dot-depth hierarchy.

Organization of the Paper. In Section 2, we set up the notation and we present the separation problem and the logics we deal with. Section 3 is devoted to our main tool: languages of well-formed words. In Section 4, we use it to prove our transfer result for all fragments from the logical perspective, and in Section 5, we show that decidability of the separation problem for the variety \mathbb{V} entails the same for $\mathbb{V} * \mathbb{D}$.

2 Preliminaries

In this section, we provide preliminary definitions on regular languages defined by logical fragments and on separation. We also present our main contribution.

Words, Languages. We fix a finite alphabet A . Let A^+ be the set of all nonempty finite words and let A^* be the set of all finite words over A . If u, v are words, we denote by $u \cdot v$ or by uv the word obtained by concatenating u and v . For convenience, we only consider, without loss of generality, languages that do not contain the empty word. That is, a language is a subset of A^+ . We work with regular languages, that is, languages definable by finite automata.

Separation. Given three languages K, L, L' , we say that K *separates* L from L' if

$$L \subseteq K \text{ and } K \cap L' = \emptyset.$$

If \mathcal{C} is a class of languages, we say that L is \mathcal{C} -*separable* from L' if there exists $K \in \mathcal{C}$ that separates L from L' . Note that if \mathcal{C} is closed under complement, L is \mathcal{C} -separable from L' if and only if L' is \mathcal{C} -separable from L . However, this is not true for a class \mathcal{C} not closed under complement, such as the classes $\Sigma_n(<)$ of the quantifier alternation hierarchy, which we shall consider.

Given a class \mathcal{C} , the \mathcal{C} -*separation problem* asks for an algorithm which, given as input two regular languages L, L' , decides whether L is \mathcal{C} -separable from L' . The \mathcal{C} -*membership problem*, which asks whether an input regular language belongs to \mathcal{C} , reduces to the \mathcal{C} -separation problem, as a regular language belongs to \mathcal{C} iff it is \mathcal{C} -separable from its complement.

Logics. We investigate several fragments of first-order logic on finite words. We view a finite word as a logical structure made of a sequence of positions labeled over A . We work with first-order logic $\text{FO}(<)$ using a unary predicate P_a for each $a \in A$, which selects positions labeled with an a , as well as binary predicates '=' for equality and '<' for the linear order. Such a formula defines the regular language of all words that satisfy it. We will freely use the name of a logical fragment of $\text{FO}(<)$ to denote the class of languages definable in this fragment. Observe that $\text{FO}(<)$ is powerful enough to express the following logical relations:

- First position, $\text{min}(x)$: $\forall y \neg(y < x)$.
- Last position, $\text{max}(x)$: $\forall y \neg(x < y)$.
- Successor, $y = x + 1$: $x < y \wedge \neg(\exists z x < z \wedge z < y)$.

However, for most fragments of $\text{FO}(<)$ this is not the case. For example, in the two-variables restriction $\text{FO}^2(<)$ of $\text{FO}(<)$, it is not possible to express successor, as it requires quantifying over a third variable. For these fragments \mathcal{F} , adding the predicates min, max

■ **Table 1** Logical fragments to which the technique applies.

Weak variant	FO(=)	FO ² (<)	$\Sigma_n(<)$	$\mathcal{B}\Sigma_n(<)$
Strong variant	FO(=, +1)	FO ² (<, +1)	$\Sigma_n(<, +1, \min, \max)$	$\mathcal{B}\Sigma_n(<, +1, \min, \max)$

and +1 yields a strictly more powerful logic \mathcal{F}^+ . Our goal is to prove a transfer result for such fragments: given a fragment, if the separation problem is decidable for the weak variant \mathcal{F} , then it is decidable as well for the strong variant \mathcal{F}^+ obtained by enriching \mathcal{F} with the above relations. The technique is *generic*, meaning that it is not bound to a particular logic. In particular, our transfer result applies to the following well-known logical fragments:

- FO(=), the restriction of FO(<) in which the linear order cannot be used, and only equality between two positions can be tested. The enriched fragment FO(=, +1) (*min* and *max* can be eliminated from the formulas) defines locally threshold testable languages [24].
- All levels in the quantifier alternation hierarchy of first-order logic. A first-order formula is $\Sigma_n(<)$ (resp. $\Pi_n(<)$) if its prenex normal form contains at most $(n - 1)$ quantifier alternations and starts with an \exists (resp. a \forall) quantifier block. Finally, a $\mathcal{B}\Sigma_n(<)$ formula is a boolean combination of $\Sigma_n(<)$ and $\Pi_n(<)$ formulas.
Since for all fragments above $\Sigma_2(<)$, a formula involving *min* and *max* can be expressed without these predicates in the same logic, we shall denote the enriched fragments by $\Sigma_1(<, +1, \min, \max)$, $\mathcal{B}\Sigma_1(<, +1, \min, \max)$, and then by $\Sigma_2(<, +1)$, $\mathcal{B}\Sigma_2(<, +1)$, ...
- FO²(<), the restriction of FO(<) using only two reusable variables. The corresponding enriched fragment is FO²(<, +1), since *min* and *max* can again be eliminated from the formulas.

Table 1 summarizes all fragments the technique applies to. We prove the following theorem.

► **Theorem 1.** *Let \mathcal{F} and \mathcal{F}^+ be respectively the weak and strong variants of one of the logical fragments in Table 1. Then \mathcal{F}^+ -separability can be effectively reduced to \mathcal{F} -separability.*

As explained in the introduction, we prove this theorem in two flavors: the first one, Theorem 4, is purely logical. It is self-contained and elementary, but is not entirely generic. The other one, Theorem 15, is purely algebraic and generic: the transfer works from an algebraic class (for which only fairly general restrictions are assumed) to an enriched one. Yet, it relies on already established results to be instantiated on the fragments of Table 1.

All these logical fragments have a rich history and have been extensively studied in the literature. In particular, the separation problem is known to be decidable for the following fragments: FO(=), FO²(<), $\Sigma_1(<)$, $\mathcal{B}\Sigma_1(<)$, $\Sigma_2(<)$ [4, 16, 17]. This means that, from our results, we obtain decidability of separation for FO(=, +1), FO²(<, +1), $\Sigma_1(<, +1, \min, \max)$, $\mathcal{B}\Sigma_1(<, +1, \min, \max)$ and $\Sigma_2(<, +1)$. Note that for FO(=, +1), FO²(<, +1) and $\mathcal{B}\Sigma_1(<, +1, \min, \max)$, the results could already be obtained as corollaries of algebraic theorems of Steinberg [21] and Almeida [2]. As explained in the introduction, an issue with this approach is that the proof of Steinberg's result relies on deep algebraic arguments and is not tailored to separation (the connection with separation is made by Almeida [2]). For $\Sigma_1(<, +1, \min, \max)$ and $\Sigma_2(<, +1)$, the result is new, as Steinberg's result does not apply to classes of languages that are not closed under complement.

3 Tools: Semigroups and Well-Formed Words

In this section, we define the main tools used in the paper. First, we recall the well-known semigroup based definition of regular languages: a language is regular if and only if it can be recognized by a finite semigroup. Our second tool, *well-formed words*, is specific to our problem and plays a key role in our transfer result.

3.1 Semigroups and Monoids

We work with the algebraic representation of regular languages in terms of semigroups. A *semigroup* is a set S equipped with an associative product, written $s \cdot t$ or st . A *monoid* is a semigroup S having a neutral element 1_S , *i.e.*, such that $s \cdot 1_S = 1_S \cdot s = s$ for all $s \in S$. If S is a semigroup, then S^1 denotes the monoid $S \cup \{1_S\}$ where $1_S \notin S$ is a new element, acting as neutral element. Note that we add such a new identity even if S is already a monoid.

An element $e \in S$ is *idempotent* if $e \cdot e = e$. We denote by $E(S)$ the set of idempotents of S . Given a *finite* semigroup S , it is folklore and easy to see that there is an integer $\omega(S)$ (denoted by ω when S is understood) such that for all s of S , s^ω is idempotent: $s^\omega = s^\omega s^\omega$.

Note that A^+ and A^* equipped with concatenation are respectively a semigroup and a monoid called the *free semigroup over A* and the *free monoid over A* . Let $L \subseteq A^+$ be a language and S be a semigroup (resp. a monoid). We say that L is *recognized by S* if there exist a morphism $\alpha : A^+ \rightarrow S$ (resp. $\alpha : A^* \rightarrow S$) and a set $F \subseteq S$ such that $L = \alpha^{-1}(F)$.

Semigroups and Separation. The separation problem takes as input two regular languages L, L' . It is convenient to work with a single object recognizing both of them, rather than having to deal with two. Let S, S' be semigroups recognizing L, L' together with the associated morphisms α, α' , respectively. Clearly, L and L' are both recognized by $S \times S'$ with the morphism $\alpha \times \alpha' : A^+ \rightarrow S \times S'$ mapping w to $(\alpha(w), \alpha'(w))$. From now on, we work with such a single semigroup recognizing both languages. Replacing $S \times S'$ with its image under $\alpha \times \alpha'$, one can also assume that this morphism is surjective. To sum up, we assume from now on, w.l.o.g., that L and L' are recognized by a single surjective morphism.

3.2 Well-Formed Words

In this section, we define our main tool for this paper. Assume that \mathcal{F} is the weak variant of one of the logical fragments of Table 1 and let \mathcal{F}^+ be the corresponding enriched variant. To any semigroup morphism $\alpha : A^+ \rightarrow S$ into a finite semigroup S , we associate a new alphabet \mathbb{A}_α called the alphabet of *well-formed words*. The main intuition behind this notion is that the \mathcal{F}^+ -separation problem for any two regular languages recognized by α can be reduced to the \mathcal{F} -separation problem for two regular languages over \mathbb{A}_α .

The alphabet \mathbb{A}_α , called *alphabet of well-formed words of α* , is defined from $\alpha : A^+ \rightarrow S$ by:

$$\mathbb{A}_\alpha = (E(S) \times S \times E(S)) \cup (S \times E(S)) \cup (E(S) \times S) \cup S.$$

We will not be interested in all words of \mathbb{A}_α^+ , but only in those that are well-formed. A word $w \in \mathbb{A}_\alpha^+$ is said to be *well-formed* if one of the following two properties holds:

- w is a single letter $s \in S$,
- w has length ≥ 2 and is of the form

$$(s_0, f_0) \cdot (e_1, s_1, f_1) \cdots (e_n, s_n, f_n) \cdot (e_{n+1}, s_{n+1}) \in (S \times E(S)) \cdot (E(S) \times S \times E(S))^* \cdot (E(S) \times S)$$

with $f_i = e_{i+1}$ for all $0 \leq i \leq n$.

► **Fact 2.** *The set of well-formed words of \mathbb{A}_α^+ is a regular language.*

We now define a morphism $\beta : \mathbb{A}_\alpha^+ \rightarrow S$ as follows. If $s \in S$, we set $\beta(s) = s$, if $(e, s) \in E(S) \times S$, we set $\beta((e, s)) = es$, if $(s, e) \in S \times E(S)$, we set $\beta((s, e)) = se$ and if $(e, s, f) \in E(S) \times S \times E(S)$, we set $\beta((e, s, f)) = esf$.

Associated Language of Well-formed Words. To any language $L \subseteq A^+$ that is recognized by α , one associates a language of well-formed words $\mathbb{L} \subseteq \mathbb{A}_\alpha^+$:

$$\mathbb{L} = \{w \in \mathbb{A}_\alpha^+ \mid w \text{ is well-formed and } \beta(w) \in \alpha(L)\}.$$

By definition, the language $\mathbb{L} \subseteq \mathbb{A}_\alpha^+$ is the intersection of the language of well-formed words with $\beta^{-1}(\alpha(L))$. Therefore, it is immediate by Fact 2 that it is regular, more precisely:

► **Fact 3.** *Let $L \subseteq A^+$ be recognized by α . Then, the associated language of well-formed words $\mathbb{L} \subseteq \mathbb{A}_\alpha^+$ is a regular language that one can effectively compute from a recognizer of L .*

4 Logical Approach

In this section, we prove Theorem 1 from a logical perspective. We begin with presenting our *separation* theorem, which will entail the *membership* theorem as a simple consequence.

► **Theorem 4.** *Let \mathcal{F} and \mathcal{F}^+ be respectively the weak and strong variants of one of the logical fragments in Table 1.*

Let L, L' be two languages recognized by a morphism $\alpha : A^+ \rightarrow S$ into a finite semigroup S . Let $\mathbb{L}, \mathbb{L}' \subseteq \mathbb{A}_\alpha^+$ be the languages of well-formed words associated with L, L' , respectively. Then L is \mathcal{F}^+ -separable from L' iff \mathbb{L} is \mathcal{F} -separable from \mathbb{L}' .

Theorem 4 reduces \mathcal{F}^+ -separation to \mathcal{F} -separation. The latter was already known to be decidable for several weak variants in Table 1, namely for $\text{FO}(=)$ [15], $\text{FO}^2(<)$ [16], $\Sigma_1(<)$ [4], $\mathcal{B}\Sigma_1(<)$ [4, 16] and $\Sigma_2(<)$ [17]. Hence, we get the following corollary.

► **Corollary 5.** *Let L, L' be regular languages. Then the following problems are decidable:*

- *whether L is $\text{FO}(=, +1)$ -separable from L' .*
- *whether L is $\text{FO}^2(<, +1)$ -separable from L' .*
- *whether L is $\Sigma_1(<, +1, \min, \max)$ -separable from L' .*
- *whether L is $\mathcal{B}\Sigma_1(<, +1, \min, \max)$ -separable from L' .*
- *whether L is $\Sigma_2(<, +1)$ -separable from L' .*

Notice that since the membership problem reduces to the separation problem, this also gives a new proof that all these fragments have a decidable membership problem. This is of particular interest for $\text{FO}^2(<, +1)$, $\mathcal{B}\Sigma_1(<, +1, \min, \max)$ and $\Sigma_2(<, +1)$ for which the previous proofs, which can be found in, or derived from [22, 1, 14], [8], and [6, 13, 12] respectively, are known to be quite involved. It turns out that for $\Sigma_2(<, +1)$, we can do even better and entirely avoid separation. Indeed, when \mathcal{F} is expressive enough, Theorem 4 can be used to prove a similar theorem for the membership problem.

► **Theorem 6.** *Let \mathcal{F} and \mathcal{F}^+ be respectively the weak and strong variants of one of the logical fragments in Table 1. Moreover, assume that for any alphabet of well-formed words, the set of well-formed words over this alphabet is definable in \mathcal{F} .*

Let L be a language recognized by a morphism $\alpha : A^+ \rightarrow S$ into a finite semigroup S . Let $\mathbb{L} \subseteq \mathbb{A}_\alpha^+$ be the language of well-formed words associated with L . Then L is definable in \mathcal{F}^+ iff \mathbb{L} is definable in \mathcal{F} .

Proof. Set $K = A^+ \setminus L$ and let \mathbb{K} be the associated language of well-formed words. Observe that by definition, $\mathbb{K} \cup \mathbb{L}$ is the set of all well-formed words.

If \mathbb{L} is definable in \mathcal{F} , then \mathbb{L} is \mathcal{F} -separable from \mathbb{K} , hence by Theorem 4, L is \mathcal{F}^+ -separable from K , and so L is definable in \mathcal{F}^+ . Conversely, if L is definable in \mathcal{F}^+ , then L is \mathcal{F}^+ -separable from K and by Theorem 4, \mathbb{L} is \mathcal{F} -separable from \mathbb{K} . Since $\mathbb{K} \cup \mathbb{L}$ is the set of all well-formed words, \mathbb{L} is the intersection of the separator with the set of all well-formed words, which by hypothesis is also definable in \mathcal{F} . Therefore, \mathbb{L} is definable in \mathcal{F} . ◀

Observe that being well-formed can be expressed in $\Pi_2(<)$: essentially, a word is well-formed if for all pairs of positions, either there is a third one in-between, or the labels of the two positions are “compatible”. Hence, among the fragments of Table 1, Theorem 6 applies to all fragments including and above $\Pi_2(<)$ in the quantifier alternation hierarchy. While such a transfer result was previously known [22, 13], the presentation and the proof are new. In particular, since membership is known to be decidable for $\Pi_2(<)$ [12], $\mathcal{B}\Sigma_2(<)$ [17] and $\Sigma_3(<)$ [17], we obtain new and simpler proofs of the following results.

► **Corollary 7.** *Given a regular language L , one can decide whether*

- *L is definable by a $\Sigma_2(<, +1)$ (resp. by a $\Pi_2(<, +1)$) formula.*
- *L is definable by a $\mathcal{B}\Sigma_2(<, +1)$ formula.*
- *L is definable by a $\Sigma_3(<, +1)$ (resp. by a $\Pi_3(<, +1)$) formula.*

It remains to prove Theorem 4. We devote the rest of the section to this proof. An important remark is that the proof of the right to left direction is constructive: we start with an \mathcal{F} formula that separates \mathbb{L} from \mathbb{L}' and use it to construct an \mathcal{F}^+ formula that separates L from L' . Note that the argument is generic for all fragments we consider.

On the other hand, the other direction, namely Proposition 9 below, requires a specific argument tailored to each fragment, which is a straightforward but tedious Ehrenfeucht-Fraïssé argument. Due to lack of space, we provide proofs of this proposition for each fragment in the long version [19] of this paper.

4.1 From \mathcal{F}^+ -separation to \mathcal{F} -separation

We prove that if L is \mathcal{F}^+ -separable from L' , then \mathbb{L} is \mathcal{F} -separable from \mathbb{L}' . We actually prove the contrapositive: if \mathbb{L} is *not* \mathcal{F} -separable from \mathbb{L}' , then L is *not* \mathcal{F}^+ -separable from L' . We rely on a construction which, to any well-formed word $\mathbf{u} \in \mathbb{A}_\alpha^+$ and any integer $i > 0$, associates a canonical word $\lceil \mathbf{u} \rceil_i \in A^+$.

Canonical Word Associated to a Well-formed Word. To any $s \in S$, we associate an arbitrarily chosen nonempty word $\lceil s \rceil \in A^+$ such that $\alpha(\lceil s \rceil) = s$ (which is possible since α has been chosen surjective). Let $i > 0$. From a well-formed word $\mathbf{u} \in \mathbb{A}_\alpha^+$, we build a word $\lceil \mathbf{u} \rceil_i \in A^+$ as follows. If $\mathbf{u} = s \in S$, then $\lceil \mathbf{u} \rceil_i = \lceil s \rceil$ for all i . Otherwise, we have by definition

$$\mathbf{u} = (s_0, e_1)(e_1, s_1, e_2) \cdots (e_{n-1} s_{n-1} e_n)(e_n, s_n).$$

For a natural $i > 0$, we set

$$\lceil \mathbf{u} \rceil_i = \lceil s_0 \rceil \lceil e_1 \rceil^i \lceil s_1 \rceil \lceil e_2 \rceil^i \cdots \lceil e_{n-1} \rceil^i \lceil s_{n-1} \rceil \lceil e_n \rceil^i \lceil s_n \rceil.$$

Recall that β is the morphism $\beta : \mathbb{A}_\alpha^+ \rightarrow S$ mapping \mathbf{u} to $s_0 e_1 s_1 \cdots s_{n-1} e_n s_n$. Since $e_j \in E(S)$ for all j , it is immediate that $\alpha(\lceil \mathbf{u} \rceil_i) = \beta(\mathbf{u})$, hence we get the following fact:

► **Fact 8.** For every $i > 0$ and every well-formed word $u \in \mathbb{A}_\alpha^+$, we have $u \in \mathbb{L}$ (resp. $u \in \mathbb{L}'$) if and only if $\lceil u \rceil_i \in L$ (resp. $u \in L'$).

We now proceed with the proof. We use the classical preorders associated to fragments of first-order logic. The (*quantifier*) *rank* of a first-order formula φ is the largest number of quantifiers along a branch in the parse tree of φ . Given $u, v \in A^+$, we write $u \preceq_k^{+1} v$ if any \mathcal{F}^+ formula of rank k that is satisfied by u is satisfied by v as well. Similarly, for $u, v \in \mathbb{A}_\alpha^+$, we write $u \preceq_k v$ if any \mathcal{F} formula of rank k that is satisfied by u is satisfied by v as well. One can verify that \preceq_k and \preceq_k^{+1} are preorders, as well as the following standard fact:

$$\begin{aligned} L \subset A^+ \text{ is definable by an } \mathcal{F}^+ \text{ formula of rank } k \text{ iff } L &= \{u' \mid \exists u \in L \text{ st. } u \preceq_k^{+1} u'\} \\ \mathbb{L} \subset \mathbb{A}_\alpha^+ \text{ is definable by an } \mathcal{F} \text{ formula of rank } k \text{ iff } \mathbb{L} &= \{u' \mid \exists u \in \mathbb{L} \text{ st. } u \preceq_k u'\}. \end{aligned} \quad (1)$$

Note that when \mathcal{F} and \mathcal{F}^+ are closed under complement, then \preceq_k and \preceq_k^{+1} are actually equivalence relations. We can now state the main proposition of this direction.

► **Proposition 9.** For any $k \in \mathbb{N}$, there exist $\ell \in \mathbb{N}$ and $i \in \mathbb{N}$ such that for any well-formed words $u, u' \in \mathbb{A}_\alpha^+$ satisfying $u \preceq_\ell u'$, we have $\lceil u \rceil_i \preceq_k^{+1} \lceil u' \rceil_i$.

For all fragments of Table 1, Proposition 9 is proved using classical Ehrenfeucht-Fraïssé arguments. While each proof is specific, the underlying ideas are similar. We present these proofs in the long version of this paper [19]. We finish the subsection by explaining how Proposition 9 can be used to terminate the proof of the first direction of Theorem 4.

We argue by contrapositive: assume that \mathbb{L} is *not* \mathcal{F} -separable from \mathbb{L}' . By definition this means that no language definable in \mathcal{F} separates \mathbb{L} from \mathbb{L}' . In particular, for any ℓ , the language

$$\{u' \mid \exists u \in \mathbb{L} \text{ st. } u \preceq_\ell u'\},$$

which is definable in \mathcal{F} by (1), cannot be a separator. Note that this language contains \mathbb{L} . Hence, for all $\ell \in \mathbb{N}$, there exist $u \in \mathbb{L}$ and $u' \in \mathbb{L}'$ such that $u \preceq_\ell u'$. We deduce from Proposition 9 and Fact 8 that for all $k \in \mathbb{N}$, there exist $u \in L$ and $u' \in L'$ such that $u \preceq_k^{+1} u'$. It follows, again by (1), that L is *not* \mathcal{F}^+ -separable from L' , which terminates the proof.

4.2 From \mathcal{F} -separation to \mathcal{F}^+ -separation

We now prove that if \mathbb{L} is \mathcal{F} -separable from \mathbb{L}' , then L is \mathcal{F}^+ -separable from L' . We do so by building an \mathcal{F}^+ -definable separator. This proof is this time entirely generic. We rely on a construction that is dual to the one used previously: to any word $w \in A^+$, we associate a canonical well-formed word $\lfloor w \rfloor \in \mathbb{A}_\alpha^+$.

Canonical Well-formed Word Associated to a Word. To any word w of A^+ , we associate a canonical well-formed word $\lfloor w \rfloor \in \mathbb{A}_\alpha^+$ such that $\alpha(w) = \beta(\lfloor w \rfloor)$. This construction is adapted from [14] and is originally inspired by [22].

Fix an arbitrary order on the set $E(S)$. For a position x of w , let $u_x \in A^+$ be the infix of w obtained by keeping only positions $x - (|S| - 1)$ to x . If position $x - (|S| - 1)$ does not exist, u_x is just the prefix of w ending at x . A position x is said *distinguished* if there exists an idempotent $e \in E(S)$ such that $\alpha(u_x) \cdot e = \alpha(u_x)$. Additionally, we always define the rightmost position as distinguished, even if it does not satisfy the property. Set $x_1 < \dots < x_{n+1}$ as the distinguished positions in w , so that x_{n+1} is the rightmost position. Let $e_1, \dots, e_n \in E(S)$ be such that for all $1 \leq i \leq n - 1$, e_i is the smallest idempotent such that $\alpha(u_{x_i}) \cdot e_i = \alpha(u_{x_i})$.

If $n = 0$, *i.e.*, if the only distinguished position is the rightmost one, set $\lfloor w \rfloor = \alpha(w) \in \mathbb{A}_\alpha$. Otherwise, we define $\lfloor w \rfloor \in \mathbb{A}_\alpha^+$ as the word:

$$\lfloor w \rfloor = (\alpha(w_0), e_1) \cdot (e_1, \alpha(w_1), e_2) \cdots (e_{n-1}, \alpha(w_{n-1}), e_n) \cdot (e_n, \alpha(w_n)) \quad (2)$$

where w_0 is the prefix of w ending at position x_1 , for all $1 \leq i \leq n - 1$, w_i is the infix of w obtained by keeping positions $x_i + 1$ to x_{i+1} , and w_n is the suffix of w starting at position $x_n + 1$. Note that by construction, $\lfloor w \rfloor$ is well-formed.

The next statement follows from the definition of β , and from the fact that by definition of the words w_i and of the chosen idempotents, we have $\alpha(w_0 \cdots w_i) e_{i+1} = \alpha(w_0 \cdots w_i)$.

► **Fact 10.** *For all $w \in A^+$, we have $\alpha(w) = \beta(\lfloor w \rfloor)$. Therefore, $w \in L$ iff $\lfloor w \rfloor \in \mathbb{L}$ and $w \in L'$ iff $\lfloor w \rfloor \in \mathbb{L}'$.*

To any distinguished position x_i in w , we now associate the position $\lfloor x \rfloor = i$ in $\lfloor w \rfloor$. Our main motivation for using this construction is its local canonicity, which is stated in the following lemma.

► **Lemma 11.** *Let $w \in A^+$. Then we have the following properties:*

- (a) *whether a position x is distinguished in w , and if so the label of position $\lfloor x \rfloor$ in $\lfloor w \rfloor$ only depends on the infix of w of length $2|S|$ ending at position x . That is, if the infixes of length $2|S|$ ending at x and y are equal, then x is distinguished iff so is y , and in that case, the labels of $\lfloor x \rfloor$ and $\lfloor y \rfloor$ in $\lfloor w \rfloor$ are equal.*
- (b) *the label of the last position of $\lfloor w \rfloor$ only depends on the suffix of length $2|S|$ of w .*

Proof. It is immediate that whether x is distinguished and if so the associated idempotent only depends on the infix u_x of length at most $|S|$ ending at x . Therefore, to prove (a), it suffices to show that all infixes w_i used in (2) are of size at most $|S|$, or in other words, that among $|S| + 1$ consecutive positions, at least one is distinguished. So let us consider an infix $a_1 \cdots a_{|S|+1}$ of w of length $|S| + 1$. It is immediate from the pigeonhole principle that there exist $i < j$ such that $\alpha(a_1 \cdots a_i) = \alpha(a_1 \cdots a_j) = \alpha(a_1 \cdots a_i) \cdot (\alpha(a_{i+1} \cdots a_j))^\omega$. Hence, the position corresponding to a_i is distinguished. The proof of the second assertion is similar. ◀

L is \mathcal{F}^+ -separable from L' . We can now construct our separator. The construction follows from the next proposition.

► **Proposition 12.** *Let $\mathbb{K} \subseteq \mathbb{A}_\alpha^+$ that can be defined using an \mathcal{F} formula φ . Then there exists an \mathcal{F}^+ formula Ψ over alphabet A such that for every word $w \in A^+$:*

$$w \models \Psi \text{ if and only if } \lfloor w \rfloor \models \varphi.$$

Proof. Proposition 12 follows from the following simple consequence of Lemma 11.

► **Claim 13.** *For any $\mathfrak{a} \in \mathbb{A}_\alpha$ there exists a formula $\gamma_{\mathfrak{a}}(x)$ of \mathcal{F}^+ with a free variable x , such that for any $w \in A^+$ and any position x of w , we have $w \models \gamma_{\mathfrak{a}}(x)$ iff x is distinguished and $\lfloor x \rfloor$ has label \mathfrak{a} in $\lfloor w \rfloor$.*

This claim holds since by Lemma 11, formula $\gamma_{\mathfrak{a}}(x)$ only needs to explore the neighborhood of size $2|S|$ of x , which is trivially possible for all fragments \mathcal{F}^+ we consider. To conclude the proof of Proposition 12, it suffices to define Ψ as the formula constructed from φ by restricting all quantifiers to positions that are distinguished and to replace all tests $P_{\mathfrak{a}}(x)$ by $\gamma_{\mathfrak{a}}(x)$. ◀

We can now finish the proof of Theorem 4. Assume that \mathbb{L} is \mathcal{F} -separable from \mathbb{L}' and let φ be an \mathcal{F} formula defining a separator. We denote by Ψ the \mathcal{F}^+ formula obtained from φ as defined in Proposition 12. We prove that Ψ defines a language separating L from L' .

We first prove that $L \subseteq \{w \mid w \models \Psi\}$. Assume that $w \in L$. Then by Fact 10, we have $\lfloor w \rfloor \in \mathbb{L}$. Hence, $\lfloor w \rfloor \models \varphi$ and so $w \models \Psi$ by definition of Ψ . The proof that $L' \subseteq \{w \mid w \not\models \Psi\}$ is identical: if $w \in L'$, we have $\lfloor w \rfloor \in \mathbb{L}'$ by Fact 10. Hence, $\lfloor w \rfloor \not\models \varphi$ and $w \not\models \Psi$ by definition of Ψ . \blacktriangleleft

5 Algebraic Approach

We now present an algebraic version of Theorem 4: the operator $\mathbb{V} \mapsto \mathbb{V} * \mathbb{D}$ preserves decidability of separation.

We would like to emphasize again that the ideas behind this theorem are essentially the same as for Theorem 4. In particular, proofs presented in the long version of this paper [19] only rely on elementary notions, thus bypassing complex constructions usually used to prove this kind of result, even if the statement itself requires some additional algebraic vocabulary.

The section is organized in three parts.

- We first briefly recall how classes of languages corresponding to our logical fragments are given an algebraic definition: for each fragment, an associated class of finite semigroups (or monoids) \mathbb{V} , a *variety*, has already been characterized, such that the class of languages definable in the fragment is exactly the class of languages that are recognized by a semigroup (or monoid) of \mathbb{V} .
- In the second part, we define what “adding the successor relation” means in this context. Given a variety \mathbb{V} , this generally corresponds to considering a new variety built on top of \mathbb{V} via an operation called the *semidirect product*. This new variety is denoted $\mathbb{V} * \mathbb{D}$.
- Finally, in the last part, we state our main theorem: for any variety \mathbb{V} , separability for the variety $\mathbb{V} * \mathbb{D}$ reduces to separability for the variety \mathbb{V} .

5.1 Varieties

A *variety of semigroups* (resp. *monoids*) is a class of finite semigroups (resp. monoids) closed under three natural operations: finite direct product, subsemigroup (or submonoid), and homomorphic image. A variety \mathbb{V} defines a class of languages, also noted \mathbb{V} , namely the class of all of languages recognized by semigroups (resp. monoids) in \mathbb{V} . There is an issue however: all classes of languages defined in this way have to be closed under complement, since the set of languages recognized by any semigroup is closed under complement. This prevents us from capturing logical fragments that are not closed under complement, such as $\Sigma_2(<)$. This problem has been solved in [11] with the notions of *ordered semigroups and monoids*. Intuitively, such a semigroup is parametrized by a partial order and the set of languages it recognizes is then restricted with respect to this partial order. These classical constructions will be recalled in the long version of this paper [19], as well as varieties corresponding to all fragments we deal with.

All logical fragments presented in Section 2 correspond to varieties that have been fully identified. For each fragment, its non-enriched variant corresponds to a variety \mathbb{V} of (ordered) monoids and its enriched version to the variety of (ordered) semigroups $\mathbb{V} * \mathbb{D}$ built from \mathbb{V} . For example, the fragment $\text{FO}^2(<)$ corresponds to the variety of monoids DA and the fragment $\text{FO}^2(<, +1)$ to the variety of semigroups $\text{DA} * \mathbb{D}$ [23] (see the long version [19] for a bibliography with all correspondences).

5.2 Semidirect Product

The Variety D. The variety \mathbf{D} consists of all finite ordered semigroups S such that for all $s \in S$ and all $e \in E(S)$, we have $se = e$. From a language perspective, a language L is recognized by a semigroup in \mathbf{D} iff there exists $k \in \mathbb{N}$ such that membership of a word w to L only depends on the suffix of length k of w .

Semidirect Product. Let M be an ordered monoid and let T be an ordered semigroup. A *semidirect product* of M and T is an operation that is parametrized by an *action* of T on M and outputs a new ordered semigroup, whose base set is $M \times T$. Therefore, one can obtain different semidirect products out of the same M and T , depending on the chosen action (we recall the construction in the long version [19]). One can next lift this product at the level of varieties.

We are interested in the semidirect products of the form $\mathbf{V} * \mathbf{D}$, the variety of ordered semigroups generated by all semidirect products of an ordered monoid of \mathbf{V} by an ordered semigroup of \mathbf{D} . The reason why we introduce such semidirect products is the following theorem, which gathers several nontrivial results from the literature. The reader is referred to the long version of this paper [19] for details.

► **Theorem 14.** *Let \mathbf{V} be a variety corresponding to a fragment \mathcal{F} from the ones presented in Table 1. Then, the variety corresponding to the fragment \mathcal{F}^+ is $\mathbf{V} * \mathbf{D}$.*

5.3 Main Theorem

We have now the machinery needed to state our main theorem. For any variety of ordered monoids \mathbf{V} , we reduce $(\mathbf{V} * \mathbf{D})$ -separability to \mathbf{V} -separability.

► **Theorem 15.** *Let \mathbf{V} be a non-trivial variety of ordered monoids. Let L and L' be two languages both recognized by the same morphism $\alpha : A^+ \rightarrow S$ into a finite semigroup S . Set $\mathbb{L}, \mathbb{L}' \subseteq \mathbb{A}_\alpha^+$ as the languages of well-formed words associated to L, L' , respectively. Then, L is $(\mathbf{V} * \mathbf{D})$ -separable from L' if and only if \mathbb{L} is \mathbf{V} -separable from \mathbb{L}' .*

The proof of Theorem 15 is presented in the full version of this paper [19]. As it was the case for Theorem 4, the proof is both elementary and constructive: if there exists a separator for \mathbb{L} and \mathbb{L}' in \mathbf{V} , we use it to construct a separator for L and L' in $\mathbf{V} * \mathbf{D}$.

In view of Theorem 14, Theorem 15 applies to all fragments we introduced. This means that Theorem 4 can be given an alternate indirect proof within this algebraic framework by combining Theorem 15 and Theorem 14. Hence, this also yields another proof of Corollary 5.

6 Conclusion

We proved that separation is decidable over finite words for the following logical fragments: $\text{FO}(=, +1)$, $\Sigma_1(<, +1, \min, \max)$, $\mathcal{B}\Sigma_1(<, +1, \min, \max)$, $\Sigma_2(<, +1)$ and $\text{FO}^2(<, +1)$. To achieve this, we presented a simple reduction to the same problem for the weaker fragments $\text{FO}(=)$, $\Sigma_1(<)$, $\mathcal{B}\Sigma_1(<)$, $\Sigma_2(<)$ and $\text{FO}^2(<)$.

The reduction itself is entirely generic to all fragments and its proof is elementary, and also mostly generic. In particular, the technique can be used to prove that the reduction works for other natural fragments of first-order logic. An interesting example to which these results apply is the quantifier alternation hierarchy within $\text{FO}^2(<)$ (known as the Trotter-Weil hierarchy, and which is decidable [25]). However, the separation problem for

classes in this hierarchy has yet to be investigated. We also obtained direct proofs that membership is decidable for $\mathcal{B}\Sigma_2(<, +1)$ and $\Sigma_3(<, +1)$.

Finally, we presented an algebraic formulation of this reduction, which recovers a previously known result by Steinberg [21], while having a much simpler proof. One can expect extending these results to other fragments, such as enrichment with modulo predicates. Another advantage of this technique is that it can be extended in a straightforward way to the same logical fragments over words of infinite length. This yields identical transfer results. We leave the presentation of these results for further work.

References

- 1 Jorge Almeida. A syntactical proof of locality of DA. *International Journal on Algebra and Computation*, 6:165–177, 1996.
- 2 Jorge Almeida. Some algorithmic problems for pseudovarieties. *Publicationes Mathematicae Debrecen*, 54:531–552, 1999. Proc. of Automata and Formal Languages, VIII.
- 3 Karl Auinger. On the decidability of membership in the global of a monoid pseudovariety. *International Journal on Algebra and Computation*, 20(2):181–188, 2010.
- 4 Wojciech Czerwiński, Wim Martens, and Tomáš Masopust. Efficient separability of regular languages by subsequences and suffixes. In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming, ICALP'13*, volume 7966 of *Lecture Notes in Computer Science*, pages 150–161. Springer, 2013.
- 5 Volker Diekert and Paul Gastin. First-order definable languages. In *Logic and Automata: History and Perspectives*, volume 2, pages 261–306. Amsterdam University Press, 2008.
- 6 Christian Glaßer and Heinz Schmitz. Languages of dot-depth $3/2$. *Theory of Computing Systems*, 42(2):256–286, 2008.
- 7 Karsten Henckell. Pointlike sets: the finest aperiodic cover of a finite semigroup. *Journal of Pure and Applied Algebra*, 55(1-2):85–126, 1988.
- 8 Robert Knast. A semigroup characterization of dot-depth one languages. *Rairo Informatique Théorique et Applications*, 17(4):321–330, 1983.
- 9 Manfred Kufleitner and Alexander Lauser. Around dot-depth 1. *International Journal of Foundations of Computer Science*, 23(6):1323–1340, 2012.
- 10 Robert McNaughton and Seymour Papert. *Counter-Free Automata*. MIT Press, 1971.
- 11 Jean-Éric Pin. A variety theorem without complementation. *Russian Mathematics, (Izvestija vuzov. Matematika)*, 39:80–90, 1995.
- 12 Jean-Éric Pin and Pascal Weil. Polynomial closure and unambiguous product. *Theory of Computing Systems*, 30(4):383–422, 1997.
- 13 Jean-Éric Pin and Pascal Weil. The wreath product principle for ordered semigroups. *Communications in Algebra*, 30:5677–5713, 2002.
- 14 Thomas Place and Luc Segoufin. Decidable characterization of $\text{FO}^2(<, +1)$ and locality of DA. Unpublished, to appear, 2014.
- 15 Thomas Place, Larijn van Rooijen, and Marc Zeitoun. Separating regular languages by locally testable and locally threshold testable languages. In *Proceedings of the 34th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'13*, volume 24 of *LIPICs*, pages 363–375. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.
- 16 Thomas Place, Larijn van Rooijen, and Marc Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *Proceedings of the 28th MFCS'13*, volume 8087 of *Lecture Notes in Computer Science*, pages 729–740. Springer, 2013.
- 17 Thomas Place and Marc Zeitoun. Going higher in the first-order quantifier alternation hierarchy on words. In *Proceedings of the 41th International Colloquium on Automata,*

- Languages, and Programming, ICALP'14*, volume 8573 of *Lecture Notes in Computer Science*, pages 342–353, 2014. <http://arxiv.org/pdf/1404.6832v1>.
- 18 Thomas Place and Marc Zeitoun. Separating regular languages with first-order logic. In *Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL'14) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS'14)*, 2014.
 - 19 Thomas Place and Marc Zeitoun. A transfer theorem for the separation problem. *CoRR*, abs/1501.00569, 2015. <http://arxiv.org/abs/1501.00569>.
 - 20 Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.
 - 21 Benjamin Steinberg. A delay theorem for pointlikes. *Semigroup Forum*, 63(3):281–304, 2001.
 - 22 Howard Straubing. Finite semigroup varieties of the form $V * D$. *Journal of Pure and Applied Algebra*, 36:53–94, 1985.
 - 23 Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing, STOC'98*, pages 234–240. ACM, 1998.
 - 24 Wolfgang Thomas. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25(3):360–376, 1982.
 - 25 Manfred Kufleitner Pascal Weil. On logical hierarchies within FO^2 -definable languages. *Logical Methods in Computer Science*, 8(3), 2012.

Computing 2-Walks in Polynomial Time

Andreas Schmid¹ and Jens M. Schmidt²

1 Max Planck Institute for Informatics, Saarbrücken, Germany

2 TU Ilmenau, Ilmenau, Germany

Abstract

A *2-walk* of a graph is a walk visiting every vertex at least once and at most twice. By generalizing decompositions of Tutte and Thomassen, Gao, Richter and Yu proved that every 3-connected planar graph contains a closed 2-walk such that all vertices visited twice are contained in 3-separators. This seminal result generalizes Tutte's theorem that every 4-connected planar graph is Hamiltonian as well as Barnette's theorem that every 3-connected planar graph has a spanning tree with maximum degree at most 3. The algorithmic challenge of finding such a closed 2-walk is to overcome big overlapping subgraphs in the decomposition, which are also inherent in Tutte's and Thomassen's decompositions.

We solve this problem by extending the decomposition of Gao, Richter and Yu in such a way that all pieces, in which the graph is decomposed into, are edge-disjoint. This implies the first polynomial-time algorithm that computes the closed 2-walk mentioned above.

1998 ACM Subject Classification G.2.2 Graph algorithms

Keywords and phrases algorithms and data structures, 2-walks, 3-connected planar graphs, Tutte paths, 3-trees

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.676

1 Introduction

Among the most fundamental problems in graph theory is the question whether a graph is *Hamiltonian*, i.e., contains a cycle of length $n := |V|$. Whitney [17] proved that every 4-connected maximal planar graph is Hamiltonian. Tutte extended this result by showing that actually every 4-connected planar graph is Hamiltonian [16]. Thomassen [15] simplified Tutte's result and proved the generalization that every 4-connected planar graph contains a path of length $n - 1$ between any given two vertices. There are numerous examples proving that 3-connected planar graphs are not necessarily Hamiltonian; in fact, even deciding whether a 3-connected 3-regular planar graph is Hamiltonian is NP-hard [10]. However, one may ask how “close” 3-connected planar graphs are to Hamiltonicity. To this end, let a *k-walk* be a walk that visits every vertex at least once and at most k times (edges may be visited multiple times). A walk is *closed* if it has the same start- and endvertex. Thus, a closed 1-walk is a Hamiltonian cycle.

Jackson and Wormald conjectured in [13] that every 3-connected planar graph contains a closed 2-walk. In a seminal result [7], Gao and Richter proved this conjecture in 1994 in the affirmative. One year later, Gao, Richter and Yu [8] published a refined decomposition that gives the existence of a very special closed 2-walk, namely one in which every vertex visited twice is contained in a 3-separator. This decomposition is involved and its presentation in [8] very densely written; in addition, it contains a flaw, which was fixed in the erratum [9]. However, as an immediate consequence, this special closed 2-walk forms a Hamiltonian cycle if the graph is 4-connected and, hence, generalizes Tutte's theorem to 3-connected planar graphs. It also generalizes Thomassen's result for 4-connected planar graphs. One of the

remarkable aspects of the result from Gao, Richter and Yu is that it generalizes yet another research direction. Barnette [2] proved that every 3-connected planar graph contains a *3-tree*, i.e., a spanning tree with maximum degree at most 3. A 3-tree can be computed in linear-time due to a Ph.D.-thesis by Strothmann [14]. Recently, Biedl showed that 3-trees (and in fact, more special variants of them) can also be computed by canonical orderings [3]. Interestingly, a 3-tree can be directly obtained from a closed 2-walk in linear time, as shown in [13, Lemma 2.2(ii)].

So far, 2-walks form the most general existence result in the above line of research. We are interested in the computational complexity of finding the above special closed 2-walk [8, 9]. Although the existence proof is over 20 years old, it is not even known whether such a 2-walk can be computed in polynomial time (neither for [7] nor for [8, 9]).

Much more is known for its preceding variants: Inspired by Tutte's classic result, Gouyou-Beauchamps [11] showed that a Hamiltonian cycle in a 4-connected planar graph can be computed in polynomial time. The crux of this approach lies in the fact that the subgraphs arising from Tutte's decomposition may overlap in an unbounded number of vertices and edges. This made it very difficult to bound the running time spent in the recursion tree reflecting the decomposition.

Asano, Kikuchi and Saito showed that a Hamiltonian cycle can be computed in linear-time when the 4-connected planar input graph is additionally maximal planar [1]. Thomassen claimed that one could also derive a polynomial-time algorithm from his more general existence proof in [15]. In [4] it was shown that this statement was too optimistic, as the subgraphs arising from his decomposition may again overlap in big parts. Chiba and Nishizeki [5] showed that this problem can be avoided for 4-connected planar input graphs and gave a linear-time algorithm to compute a Hamiltonian cycle for these graphs. However, the general problem of overlapping subgraphs in 3-connected graphs has not been resolved. Even the decomposition in [8, 9] bears the same obstruction that made previous algorithmic results difficult, namely big overlapping subgraphs.

As main result, we propose how to overcome this problem by extending the decomposition of Gao, Richter and Yu such that all arising subgraphs will be edge-disjoint. This leads to the first polynomial-time algorithm that computes the special closed 2-walk of [8, 9], generalizing the previous results. The result is stated for the class of circuit graphs, which contain all 3-connected graphs. We aim for a detailed and self-contained description of this decomposition.

► **Theorem 1.** *Let G be a circuit graph with external face boundary C and let x, y be vertices of C . A closed 2-walk of G can be computed in polynomial time such that x and y are visited exactly once and every vertex visited twice is contained in either a 2-separator or an internal 3-separator of G .*

2 Preliminaries

We assume familiarity with standard graph theoretic notations as in [6]. A *k-separator* of a graph $G = (V, E)$ is a set of k vertices whose deletion leaves a disconnected graph. Let $n := |V|$ and $m := |E|$. A graph G is *k-connected* if $n > k$ and G contains no $(k-1)$ -separator. A set of paths intersecting pairwise at most at their endvertices are called *independent*. For a path P and two vertices $x, y \in P$, let the subpath from x to y in P be xPy .

A central concept for the decomposition is the notion of *H-bridges*: For a subgraph H of G , an *H-bridge* of G is a component K of $G - V(H)$ together with all edges joining vertices of K with vertices of H and the endvertices of these edges. Although standard notation

allows an H -bridge to be a single edge, we excluded this case from the definition, as such bridges will not play any role for 2-walks. A vertex in an H -bridge L is an *attachment* of L if it is also in H , and it is an *internal* vertex of L otherwise.

A *plane* graph is a planar embedding of a graph. For two vertices x, y of a cycle C in a plane graph, let xCy be the clockwise path from x to y in C . For a cycle C in a plane graph G , let the subgraph of G *inside* C be the subgraph induced by $E(C)$ and all edges intersecting the open disc-homeomorph of the plane interior of C . A subgraph inside a cycle of a 3-connected plane graph G is not necessarily 3-connected; however, its only 2-separators must have both vertices on the external face. Since we will often use induction on such subgraphs when describing the decomposition, we will deal with circuit graphs instead of 3-connected plane graphs. A *circuit graph* (G, C) is a 2-connected plane graph G with external face boundary C such that the following property is satisfied:

► **Definition 2** (3-Paths Property). For every vertex v in $G \setminus C$, G contains three independent paths from v to distinct vertices in C .

Clearly, circuit graphs generalize 3-connected plane graphs. In the following, we will give several lemmas about circuit graphs that will be used throughout the paper.

► **Lemma 3.** Let $\{u, v\}$ be a 2-separator of a circuit graph (G, C) . Every component of $G \setminus \{u, v\}$ contains a vertex of C .

Proof. Assume to the contrary that $G \setminus C$ has a component K with $V(K) \cap V(C) = \emptyset$. Since K does not contain a vertex of C , each path from a vertex $w \in V(K)$ to C contains u or v . Thus, there are no three independent paths from w to C , contradicting the 3-Paths Property. ◀

► **Lemma 4.** Let $\{u, v\}$ be a 2-separator of a circuit graph (G, C) . Then u and v are contained in C and $G \setminus \{u, v\}$ has exactly two components.

Proof. First assume that u or v , say u , is not contained in C . As $\{u, v\}$ is a 2-separator of G , $G \setminus \{u, v\}$ has at least two components. Since $u \notin V(C)$, one component of $G \setminus \{u, v\}$ must contain all remaining vertices of C . This contradicts Lemma 3. For the second claim, observe that $G \setminus \{u, v\}$ has at most two components that contain vertices of C , as $C \setminus \{u, v\}$ is the union of at most two paths. Thus, a third component would contradict Lemma 3. ◀

Next, we state several lemmas how a circuit graph can be decomposed into smaller circuit graphs.

► **Lemma 5** ([7]). Let $\{u, v\}$ be a 2-separator of a circuit graph (G, C) . For each $\{u, v\}$ -bridge H of G (recall that $H \neq uv$), $H \cup uv$ is a circuit graph.

► **Lemma 6** ([7]). Let C' be any cycle in a circuit graph (G, C) and let H be the subgraph inside C' . Then (H, C') is a circuit graph.

A *block* is a maximal connected subgraph that does not contain a 1-separator. Every block is either 2-connected or has at most two vertices. It is well-known that the blocks of a graph partition its edge-set. A graph G is called a *chain of blocks* if it consists of blocks B_1, B_2, \dots, B_k such that $V(B_i) \cap V(B_{i+1})$, $1 \leq i < k$, are pairwise distinct 1-separators of G and G contains no other 1-separator. Thus, a chain of blocks is a graph, whose block-cut tree [12] is a path. A key idea in the decomposition is that deleting a vertex of the external face boundary of a circuit graph results in a plane chain of blocks. Every such block will again be a circuit graph due to Lemma 6.

► **Lemma 7** ([7]). *Let (G, C) be a circuit graph and let $v \in V(C)$. Then $G \setminus v$ is a plane chain of blocks B_1, B_2, \dots, B_k and, if $k > 1$, one of the neighbours of v in C is in $B_1 \setminus B_2$ and the other is in $B_k \setminus B_{k-1}$.*

3 From Tutte Paths to 2-Walks

We recapitulate the fundamental steps of Gao, Richter and Yu [8, 9] for proving the existence of a closed 2-walk. A crucial role is played by the notion of a Tutte path. A *Tutte path* (*Tutte cycle*) of a circuit graph (G, C) is a path (*cycle*) T for which every T -bridge has exactly 2 attachments if it contains an edge of C and otherwise exactly 3 attachments. A Tutte path from x to y through u has startvertex x , endvertex y and contains u ; we will sometimes say that u is the *intermediate vertex* of T . Tutte paths can be used to construct a closed 2-walk if the attachments of its T -bridges are sufficiently disjoint. In [8, 9], the existence of a Tutte path T with T -bridges B_1, B_2, \dots, B_k was proven for which a set $S = \{s_1, s_2, \dots, s_k\}$ of vertices exists such that s_i is an attachment of B_i for each i . The set S is called *system of distinct representatives* (*SDR*) of the T -bridges. The next results give the existence of such Tutte paths and cycles; Theorem 8 is slightly weaker than the one in [8, 9] (in which $y \in V(G)$), but sufficient for our needs.

► **Theorem 8** ([8, 9]). *Let (G, C) be a circuit graph, let $x, u, y \in V(C)$ with $x \neq y$ and let $a \in \{x, u\}$. Then there is a Tutte path P of G from x to y through u and an SDR S of the P -bridges such that $a \notin S$.*

According to Lemma 7, $G \setminus x$ is a plane chain of blocks. By computing a Tutte path for every such block and extending the union of these Tutte paths to x (using the two incident edges in C), we immediately obtain a Tutte cycle of G . Note that the time for computing this Tutte cycle is dominated by the computation of the Tutte paths (see Lines 2–4 of Algorithm 1).

► **Corollary 9** ([8, 9]). *Let (G, C) be a circuit graph and let $x, y \in V(C)$. Then there is a Tutte cycle T of G and an SDR S of the T -bridges in G with $x, y \in V(T)$ and $x, y \notin S$.*

Proving the existence of an SDR as in Corollary 9 is the crucial new insight of Gao, Richter and Yu's paper [8, 9]. It implies the existence of a closed 2-walk. The idea is to use the vertices of the SDR S as branch vertices, at which the walk deviates from T into 2-walks of the T -bridges, which exist by induction. The constructed closed 2-walk will therefore have special properties for the vertices visited twice. Let an *internal 3-separator* S of a circuit graph (G, C) be a 3-separator such that $G - S$ contains a component disjoint from C .

► **Theorem 10** ([8, 9]). *Let (G, C) be a circuit graph and let $x, y \in V(C)$. Then there is a closed 2-walk W in G visiting x and y exactly once such that every vertex visited twice is contained in either a 2-separator or an internal 3-separator of G .*

We are interested in the computational complexity of finding the 2-walk of Theorem 10 when an efficient subroutine for computing Tutte paths is known.

Algorithm 1 gives a high-level description of the steps taken for the proof of Theorem 10. For all steps except the computation of Tutte paths in Line 3 and the computation of suitable circuit subgraphs for the recursion on L in Line 6, the corresponding existence proofs give immediately linear-time algorithms. It can be readily shown that the computation of Line 6 exceeds the time spent for computing a Tutte path by at most a factor m ; hence, we can reduce to computing Tutte paths.

Algorithm 1

```

1: procedure 2-WALK( $(G, C)$ ,  $x, y \in V(C)$ )
2:   for every block  $B$  of the plane chain of blocks  $G \setminus x$  do
3:     Compute a Tutte path  $P_B$  of  $B$  ▷ crucial
4:   Join the union of all computed Tutte paths to  $x$  and obtain a Tutte cycle  $T$  of  $G$ 
5:   for every  $T$ -bridge  $L$  do
6:     Recurse on  $L$  to compute a 2-walk  $W_L$  ▷ polynomially dependent on Line 3
7:   Output the union of  $T$  and all  $W_L$ 

```

However, it is not even clear whether a Tutte path itself can be computed in polynomial time, as its existence proof uses a decomposition into circuit subgraphs that may overlap in large parts. We will show that a Tutte path can be computed in polynomial time; this implies our main Theorem 1.

4

 Finding Tutte Paths

We will prove Theorem 8 by extending the decomposition of Gao, Richter and Yu. The extended decomposition will only branch into edge-disjoint circuit graphs and thus turn out to be algorithmically accessible. In the following sections, we will first review some steps given in [8, 9] needed to set up the decomposition, then explain how we can avoid overlapping subgraphs, and finally give the details of the extended decomposition.

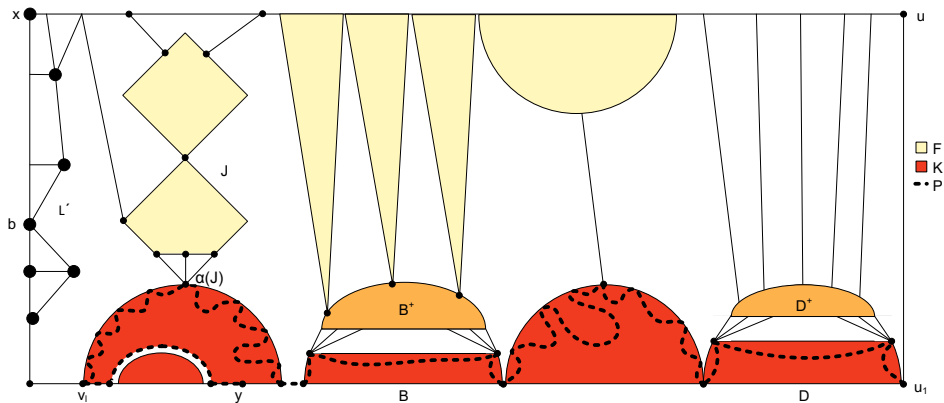
4.1 Setting up the Decomposition

We review the initial steps taken for the original decomposition in [8, 9]. Let (G, C) be a circuit graph, let $x, u, y \in V(C)$ with $x \neq y$ and let $a \in \{x, u\}$. We want to find a Tutte path from x to y through u . The vertex a acts as a place-holder that allows us to prevent x or u to be in the SDR S ; this will be useful for the induction.

We first eliminate some symmetric cases. If $u = x$, we can choose any other vertex $v \in V(C) \setminus x$ and assign $u = v$. The same holds if $u = y$ and $a \neq u$. If $a = u = y$, we interchange the roles of x and y and proceed as above. Thus we can assume that $u \notin \{x, y\}$. We will need y to be in uCx in a later step. Therefore if $y \in xCu$, we flip the current embedding of G such that in the new embedding $y \in uCx$.

The proof of Theorem 8 proceeds by induction on the number of edges in G . If $|E(G)| = 3$, G is a triangle. In that case, the Tutte path we are looking for is xuy , the corresponding SDR S is empty and there are clearly no overlapping subgraphs. For the induction step, let u_1 the neighbour of u that is not in xCu . In the special case that $u_1 = y$, we define $K := u_1$. Otherwise, we define K as the *minimal connected union* of blocks of $G \setminus xCu$ that contains u_1 and y , where minimality is with respect to the number of blocks (see Figure 1). As argued before, the blocks of $G \setminus xCu$ form a tree; by minimality, K will be a plane chain of blocks. Let B_1, \dots, B_l be the blocks of K such that $u_1 \in B_1$ and $y \in B_l$ and let C_i be the external face boundary of B_i . We number the 1-separators in K from v_1 to v_{l-1} , i.e., the blocks B_i and B_{i+1} intersect exactly in v_i . In addition, we set $v_0 := u_1$ and define v_l as the vertex in B_l nearest to x in u_1Cx .

For each $(K \cup xCu)$ -bridge L , L intersects K in at most one vertex, as otherwise a block of K would not be maximal. We call this vertex, if it exists, $\alpha(L)$. Note that the edge uu_1 is not a $(K \cup xCu)$ -bridge by definition. It is however possible that there is a $(K \cup xCu)$ -bridge



■ **Figure 1** A circuit graph (G, C) , in which the plane chain of blocks K is depicted in dark grey (red) and F is the subgraph induced by xCu and the vertices of light grey (yellow) subgraphs. Here, F and K overlap in the grey (orange) subgraphs B^+ and D^+ . The part P' from u_1 to y of the desired Tutte path of G can be computed by induction on the blocks of K .

that contains v_1Cx . If so, we denote this special bridge by L' (otherwise, v_1Cx is just an edge). The bridge L' is special among the $(K \cup xCu)$ -bridges, as it is the only one that may have exactly two attachments; all other bridges have at least three attachments by the 3-Path Property.

4.2 Avoiding Overlapping Subgraphs

In the proof of Theorem 8 in [8, 9], the authors define a second connected subgraph F that overlaps with K and then recurse on both subgraphs separately by constructing Tutte paths of every block of these subgraphs (see Figure 1). The recursively constructed Tutte paths of F (giving a path from x to u) and in K (giving a path from u_1 to y) are then concatenated with uu_1 to get the desired Tutte path of G . The overlapping parts of F and K may therefore receive multiple recursive calls, which prevents to bound the running time of this decomposition.

However, the description of F in [8, 9] suggests that an overlapping subgraph in this decomposition consists always of the inner vertex set of some bridge of the Tutte path computed for K . In the following, we will compute a Tutte path from u_1 to y , but instead of doing this in K , we will do this in a slightly modified subgraph $\eta(K)$. This augmentation will allow us to identify and exclude possible overlapping subgraphs in advance. We first state some results about bridges of Tutte paths T . For the next observation, recall that T -bridges are not single edges.

► **Observation 11.** *Let (G, C) be a circuit graph and let T be a Tutte path of G . Then the attachments of any T -Bridge with two attachments form a 2-separator in G .*

According to Lemma 3, both vertices of a 2-separator in a circuit graph lie on the external face boundary. The following lemma strengthens this statement for the 2-separators that are attachments of T -bridges.

► **Lemma 12.** *Let (G, C) be a circuit graph with a Tutte path T from $x \in V(C)$ to $y \in V(C)$. Then every T -Bridge with two attachments has either both attachments on xCy or both on yCx .*

Proof. Assume otherwise. Let J be a T -bridge with two attachments $\{c, d\}$, $c \in xCy \setminus \{x, y\}$ and $d \in yCx \setminus \{x, y\}$. By Observation 11, $\{c, d\}$ is a 2-separator in G . Thus, $G \setminus \{c, d\}$ contains exactly two components X and Y with $x \in X$ and $y \in Y$ that cover $C \setminus \{c, d\}$, according to Lemma 4. Due to Lemma 3, X and Y must contain at least one vertex of C each. It follows that the inner vertex set of J is either X or Y . In both cases, J contains an edge of T , which contradicts that J is a T -bridge. \blacktriangleleft

We explain the idea for our decomposition; the precise decomposition will be given in the next section. Let T be a Tutte path from u_1 to y of K and consider any T -bridge J . In the decomposition of [8, 9], by planarity, J can only intersect an overlapping part if it intersects the upper external face boundary of K . Then J has exactly two attachments c and d , according to the definition of a Tutte path and the fact that J contains a boundary edge of some block of K . By Observation 11 and Lemma 4, c and d must be as well on the boundary of K . In fact, c and d are on the upper boundary of K by Lemma 12. In summary, the only parts of K that would have possibly overlapped in the original decomposition are the T -bridges with exactly two attachments on the upper boundary of K (see also Figure 1).

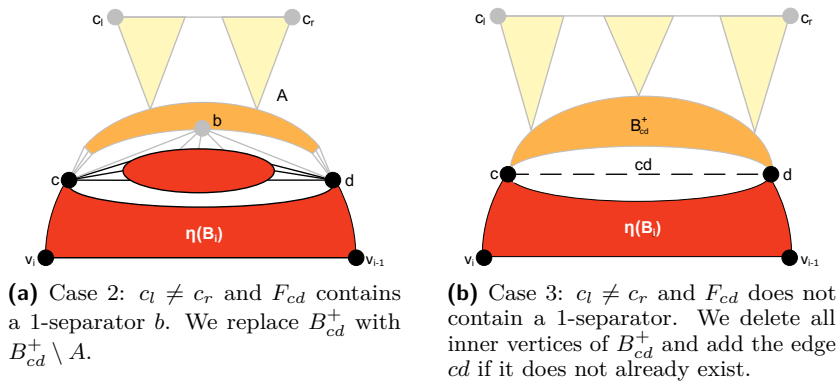
Thus, if we find all 2-separators in $v_i C_i v_{i-1}$ for a block B_i of K before we actually compute a Tutte path of this block, we have identified all subgraphs of this block which would have possibly overlapped in the original decomposition. Let $\{c, d\}$ be a 2-separator of a block B_i such that c and d is in $v_i C_i v_{i-1}$. Let further B_{cd}^+ be the $\{c, d\}$ -bridge in B_i that contains the path $cC_i d$ (see Figure 1). We call a 2-separator $\{c, d\}$ in $v_i C_i v_{i-1}$ *maximal* in $v_i C_i v_{i-1}$ if there is no other 2-separator $\{c', d'\}$ in $v_i C_i v_{i-1}$ with c and d in $c' C_i d'$. A block B_i may contain several maximal 2-separators; however, they must be consecutive on $v_i C_i v_{i-1}$. For the computation of a Tutte path of B_i , we will first find all maximal 2-separators in C_i . The next smaller 2-separators inside them will only be computed if necessary.

Let $\{c, d\}$ be a 2-separator of B_i with c and d in $v_i C_i v_{i-1}$ and let v be an inner vertex of B_{cd}^+ . Then c_l and c_r are defined as the vertices in $x C u$ closest to x and u , respectively, that are reachable from v by a path not containing any vertex of $\{c, d\} \cup V(C)$ as inner vertex (possibly $c_l = c_r$). Figure 2 shows two examples where $c_l \neq c_r$. For a 2-separator $\{c, d\}$ of B_i with c and d in $v_i C_i v_{i-1}$, let F_{cd}' be the $\{c, d, c_l, c_r\}$ -bridge that contains B_{cd}^+ and let $F_{cd} := F_{cd}' \setminus \{c, d\}$. The subgraph F_{cd} contains the overlapping parts of K of the original decomposition as discussed above.

In order to modify K to $\eta(K)$, we iterate through all maximal 2-separators $\{c, d\}$ of every block of K and “cut off” some B_{cd}^+ in a predefined way. This will allow us to compute Tutte paths for every block of $\eta(K)$ and iteratively detour these Tutte paths to the subgraphs B_{cd}^+ if necessary. For some B_{cd}^+ , we will add a special edge to $\eta(K)$ whose containment in the previously computed Tutte path will decide whether such a detour is needed. The exact definition of $\eta(K)$ is dependent on the existence of a 1-separator in F_{cd} . For the relevant case $c_l \neq c_r$, we will prove that a vertex b is a 1-separator of F_{cd} if and only if $\{b, c, d\}$ is a 3-separator of G . If such a 1-separator b exists, we will show that b can actually be chosen in such a way that the subgraph of F_{cd} “above” b is a block; such a vertex will additionally be unique.

► Lemma 13. *Let $c_l \neq c_r$. A vertex $b \in F_{cd}$ is a 1-separator of F_{cd} if and only if $\{b, c, d\}$ is a 3-separator of G . No 1-separator of F_{cd} is contained in $c_l C c_r$.*

Lemma 13 implies that there is a block of F_{cd} that contains $c_l C c_r$. We call this block A . Note that there may be many 1-separators in F_{cd} . However, there is exactly one such 1-separator that is contained in A .



■ **Figure 2** The two cases of modifying K to $\eta(K)$. In both cases, the remaining part of B_{cd}^+ is the dark grey (red) subgraph, i.e., the grey (orange) part of B_{cd}^+ is deleted.

► **Lemma 14.** *Let $c_l \neq c_r$ and let F_{cd} contain a 1-separator. Then F_{cd} contains a unique 1-separator b such that $b \in A$.*

We are now ready to define $\eta(K)$.

► **Definition 15.** Let $\eta(K)$ be the graph obtained from K by performing the following for every maximal 2-separator $\{c, d\} \neq \{v_i, v_{i-1}\}$ of every block B_i of K .

Case 1: $c_l = c_r$

Do nothing.

Case 2: $c_l \neq c_r$ and F_{cd} contains a 1-separator (see Figure 2(a))

Replace B_{cd}^+ with $B_{cd}^+ \setminus A$.

Case 3: $c_l \neq c_r$ and F_{cd} contains no 1-separator (see Figure 2(b))

Delete all inner vertices of B_{cd}^+ and add the edge cd if cd does not already exist.

For a block B_i of K , let $\eta(B_i)$ be the corresponding block of $\eta(K)$. Let $\eta(C_i)$ be the external boundary of $\eta(B_i)$. Note that $\eta(K)$ is no longer a plane chain of blocks of $G \setminus xCu$, as the modified blocks $\eta(B_i)$ are not maximal any more in G . However, every $\eta(B_i)$ that is not just an edge is still a circuit graph, as shown next.

► **Lemma 16.** *Every $\eta(B_i)$ that is not an edge is a circuit graph.*

In the following, whenever dealing with a maximal 2-separator $\{c, d\}$ of K , the variables $F_{cd}, F'_{cd}, c_l, c_r, B_i, A$ will always refer to the previously defined objects and b will refer to the unique 1-separator of F_{cd} defined in Lemma 14.

4.3 Extending the Decomposition

We extend the decomposition described so far. First, we find a preliminary Tutte path P of $\eta(K)$, which will eventually be modified to a Tutte path of G in Section 4.3.2. As a speciality in advance, there are two kinds of $(K \cup xCu)$ -bridges, for which the extension of P into these bridges is not hard to show; these are the *isolated* $(K \cup xCu)$ -bridges, which have all their attachments on xCu and the special bridge L' . Here, we will assume that G contains neither isolated bridges nor L' .

For a $(K \cup xCu)$ -bridge L , let $C(L)$ be the shortest path in v_lCu that contains all attachments of L in v_lCu . When considering such L , the endpoints of $C(L)$ closest to v_l and u in v_lCu are called c_l and c_r , respectively ($c_l = c_r$ is possible).

4.3.1 Finding a Tutte Path of $\eta(K)$

We continue the decomposition of a circuit graph (G, C) of Section 4.1 by computing a Tutte path $P_{\eta(K)}$ of $\eta(K)$ from u_1 to y and an SDR of the $P_{\eta(K)}$ -bridges. For each block $\eta(B_i)$ of $\eta(K)$, we compute $P_{\eta(B_i)}$ and an SDR $S_{\eta(B_i)}$ of the $P_{\eta(B_i)}$ -bridges as follows.

If $\eta(B_i)$ is just an edge $v_{i-1}v_i$, set $P_{\eta(B_i)} := v_{i-1}v_i$ and $S_{\eta(B_i)} := \emptyset$. Otherwise, if $i < l$, compute by induction a Tutte path $P_{\eta(B_i)}$ of $\eta(B_i)$ from v_{i-1} to v_i and a SDR $S_{\eta(B_i)}$ of all $P_{\eta(B_i)}$ -bridges such that $v_i \notin S_{\eta(B_i)}$ (as intermediate vertex, an arbitrary vertex in $V(C_i) \setminus \{v_{i-1}, v_i\}$ can be chosen). If $i = l$, compute a Tutte path $P_{\eta(B_l)}$ of $\eta(B_l)$ from v_{l-1} to y through v_l and an SDR $S_{\eta(B_l)}$ of all $P_{\eta(B_l)}$ -bridges. We apply the induction on $\eta(B_l)$ such that $v_l \notin S_{\eta(B_l)}$. Then $P_{\eta(K)} = \cup_{i=1}^l P_{\eta(B_i)}$ is the desired Tutte path of $\eta(K)$ from u_1 to y .

Every $P_{\eta(B_i)}$ -bridge with three attachments in $\eta(B_i)$ is also a $P_{\eta(B_i)}$ -bridge with three attachments in G . Every internal vertex of such a $P_{\eta(B_i)}$ -bridge has the same neighbourhood in $\eta(B_i)$ as in G . Therefore, each such bridge preserves its number of attachments in G . The same argument holds for the $P_{\eta(B_i)}$ -bridges in $\eta(B_i)$ that have exactly two attachments and contain an edge of C . In fact, these two observations do not only hold for $P_{\eta(B_i)}$, but for any Tutte path P_H of some circuit graph $H \subset G$. We will therefore only discuss P_H -bridges in the remainder of the paper that have exactly two attachments in H and do not contain any edge of C . We will show that these bridges have exactly three attachments in G .

In order to find the desired Tutte path P of (G, C) , we initially set $P := xCu_1 \cup P_{\eta(K)}$ and then modify P step by step such that the final path P is a Tutte path, does not contain any edge cd that was added in Case 3, and admits an SDR S of all P -bridges. We will decompose G into smaller circuit graphs on which we apply induction. These graphs will pairwise intersect in at most one vertex, i.e., they are *edge-disjoint*. By carefully choosing a when applying the induction, we will avoid that the vertex in the intersection is a representative in both graphs. The modification of P starts by handling the $(K \cup xCu)$ -bridges that have an attachment on K , but are not contained in any F_{cd} . We next show useful details of these bridges.

► **Lemma 17.** *Let L be any $(K \cup xCu)$ -bridge for which $\alpha(L)$ exists and which is not contained in some F_{cd} . Then $\alpha(L) \in \eta(K)$ and $\alpha(L) \in P_{\eta(B_i)}$.*

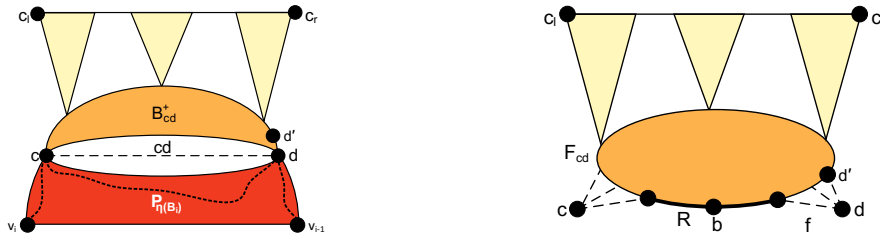
4.3.2 Finding a Tutte Path of G

Algorithm 2: *FindTuttePath* $((G, C), x, u, y, P, S)$

Input: $(G, C), x, u, y, P, S$, where P is the preliminary Tutte path from x to y of Section 4.3.1 and S the corresponding SDR

Output: A Tutte path of (G, C) stored in P and an SDR S of the P -bridges in G stored in S

1. For every $(K \cup xCu)$ -bridge L in G with $\alpha(L) \in \eta(K)$:
 - According to Lemma 17, $\alpha(L) \in P_{\eta(B_i)}$ for some B_i .
 - Let $J = (L \cup C(L)) \setminus \alpha(L)$.
 - J is 2-connected: L has an inner vertex by definition of bridge and thus at least two attachments on C by the 3-Paths Property. Hence, $|V(J)| \geq 3$. Starting with $C(L)$ and adding the two paths to $C(L)$ from every remaining vertex in J due to the 3-Paths Property gives an *open ear decomposition* [18]. Thus, J is 2-connected.
 - It follows that the boundary of J is a cycle and J is a circuit graph.
- a. Compute a Tutte path P_J from c_l to c_r and an SDR S_J of all P_J -bridges by induction such that depending on a , either c_l or c_r is not in S_J . If $a = x$, apply the induction



(a) A maximal 2-separator $\{c, d\}$ of B_i such that $c_l \neq c_r$ and F_{cd} contains no 1-separator. In this case, cd is not contained in $P_{\eta(B_i)}$.

(b) The subgraph F_{cd} (not containing dashed edges). We compute a Tutte path $P_{F_{cd}}$ of F_{cd} from c_l to c_r through $b \in R$ (the fat line depicts the path R).

■ **Figure 3** Step 4(a) of *FindTuttePath*.

such that $c_l \notin S_J$. Otherwise, if $a = u$, apply the induction such that $c_r \notin S_J$.

b. Set $P := P \setminus c_l C c_r \cup P_J$ and $S := S \cup S_J$.

- By the 3-Paths Property, every P_J -bridge in J that has exactly two attachments and does not contain an edge of C must contain a vertex that in G is a neighbour of $\alpha(L)$. Each such P_J -bridge will therefore become a P -bridge with exactly three attachments in G .

2. For every maximal 2-separator $\{c, d\}$ of K satisfying Case 1 of Definition 15:

- Let J be any $P_{\eta(B_i)}$ -bridge in $\eta(B_i)$ that contains an edge of $c\eta(C_i)d$. We show that every such J becomes a P -bridge in G with exactly three attachments. By the 3-Path Property, there is a path from every inner vertex of J to some vertex in C that does neither contain c nor d . In this case the only possible such vertex is $c_l = c_r$. Thus, J is a P -bridge in G with exactly three attachments, one of which is c_l .

3. For every maximal 2-separator $\{c, d\}$ of K satisfying Case 2 of Definition 15:

a. Compute a Tutte path P_A of the block A of F_{cd} from c_l to c_r through b and an SDR S_A of all P_A -bridges. Apply the induction such that $a \notin S_A$, analogously to Step 1(a).

b. Set $P := P \setminus c_l C c_r \cup P_A$ and $S := S \cup S_A$.

- Let H be the $\{b, c, d\}$ -bridge in G that does not contain $c_l C c_r$, according to Lemma 13.

- Consider any P_A -bridge J with exactly two attachments in A that does not contain an edge of C . By the 3-Paths Property, J must contain an inner vertex that has a neighbour in $G \setminus A$. Since b is a 1-separator of F_{cd} in A and $b \in P_A$, the set of all such neighbours is either $\{c\}$, $\{d\}$ or $\{c, d\}$. We will show that the last case is not possible; hence, every such P_A -bridge will become a P -bridge with exactly three attachments in G . As P_A is a Tutte path and J has only two attachments, J contains an edge of the external boundary of A . By planarity and the existence of (the connected) $\{b, c, d\}$ -bridge H in G , J cannot be adjacent to both, c and d .

- In the case that $P_{\eta(B_i)}$ contains an edge of H , there may exist $P_{\eta(B_i)}$ -bridges $J \subseteq H \setminus b$ with two attachments having both attachments in $c\eta(C_i)d$. We show that every such J becomes a P -bridge in G with exactly three attachments. By the 3-Path Property, there is a path from every inner vertex of J to some vertex in C that does neither contain c nor d . As $J \subset H$, this path contains b . Thus, J is a P -bridge in G with exactly three attachments, one of which is b .

4. For every maximal 2-separator $\{c, d\}$ of K satisfying Case 3 of Definition 15:

a. If $cd \notin P_{\eta(B_i)}$ (see Figure 3):

- Let f be the face in B_i that contains cd and an inner vertex of B_{cd}^+ .
 - Let R be the path obtained from the boundary of B_{cd}^+ in f by deleting c and d .
 - i. Choose an arbitrary vertex b in R .
 - ii. Compute a Tutte path $P_{F_{cd}}$ of F_{cd} from c_l to c_r through b by induction on F_{cd} and an SDR $S_{F_{cd}}$ of all $P_{F_{cd}}$ -bridges. Apply the induction such that $a \notin S_{F_{cd}}$, analogously to Step 1(a).
 - iii. Set $P := P \setminus c_l C c_r \cup P_{F_{cd}}$ and $S := S \cup S_{F_{cd}}$.
 - Consider any $P_{F_{cd}}$ -bridge J with exactly two attachments in F_{cd} that does not contain an edge of C . By the 3-Paths Property, the inner vertex set of J is neighboured to either $\{c\}$, $\{d\}$ or $\{c, d\}$. We show that the last case is not possible, which proves that every such $P_{F_{cd}}$ -bridge becomes a P -bridge in G with exactly three attachments. By the choice of R , the only vertex that may be adjacent to c and d is b (in that case, $R = \{b\}$). However, b is not a neighbour of an inner vertex of J , as $b \in P_{F_{cd}}$. This proves the claim.
- b. If $cd \in P_{\eta(B_i)}$:
- Recall that cd was possibly added during the construction of $\eta(K)$ and may therefore not be in G . We aim to replace cd in $P_{\eta(B_i)}$ with a Tutte path of B_{cd}^+ from c to d .
 - According to Lemma 5, $B_{cd}^+ \cup cd$ is a circuit graph.
 - Let d' be the neighbour of d on the boundary of $B_{cd}^+ \cup cd$ that is different from c .
 - Let $K' := (B_{cd}^+ \cup cd) \setminus d$. According to Lemma 7, K' is a plane chain of blocks $B'_1, B'_2, \dots, B'_{\nu'}$, such that $d' \in B'_1$ and $c \in B'_{\nu'}$. Note that K' is a subgraph of G , as it does not contain cd .
 - By planarity, every $K \cup xCu$ -bridge L in G that is contained in F_{cd} has its attachment $\alpha(L)$ (if exists) on the upper boundary of K' , while every neighbour of d is on the lower boundary of K' .
 - We will replace $cd \in P_{\eta(B_i)}$ with the union of the edge dd' and a Tutte path of $\eta(K')$ from d' to c ; the Tutte path is constructed in the very same way as we did for K , i.e., by first computing $\eta(K')$, then Tutte paths of the blocks of $\eta(K')$ and then branching into the different steps of *FindTuttePath*. This will iterate on the maximal 2-separators of K' , which are the sets of next smaller 2-separators of K . Note that $\eta(K)$ and $\eta(K')$ are edge-disjoint.
 - Technically, $\eta()$ is defined on a given circuit graph. We face this problem by constructing the following artificial circuit graph G' , which allows for a proper definition of $\eta(K')$.
 - Let G' be the union of $K' \cup c_l C c_r$, all $K \cup xCu$ -bridges that are contained in F_{cd} , and the new edges cc_l and $c_r d'$. Clearly, G' is a circuit graph (G', C') . Let $x' := c_l$, $u' := c_r$, $u'_1 := d'$ and $y' := c$.
 - Then K' is consistent to our previous definition, i.e., the *minimal connected union* of blocks of $G' \setminus x' C' u'$ that contains y' and u'_1 , and $\eta(K')$ is well-defined in dependence of G' and $\{x', u', y'\}$.
 - i. Compute $\eta(K')$ from K' .
 - ii. For each block $\eta(B'_i)$ of $\eta(K')$, compute a Tutte path $P_{\eta(B'_i)}$ and an SDR $S_{\eta(B'_i)}$ of the $P_{\eta(B'_i)}$ -bridges in $\eta(B'_i)$ by induction, as described in Section 4.3.1.
 - iii. Set $P' := c_l P c_r \cup P_{\eta(B'_1)} \cup \dots \cup P_{\eta(B'_{\nu'})} \cup c_r d'$.
 - iv. Set $S' := S_{\eta(B'_1)} \cup \dots \cup S_{\eta(B'_{\nu'})}$.
 - v. Apply *FindTuttePath* $((G', C'), x', u', y', P', S')$.
 - vi. Set $P := P \setminus c_l C c_r \setminus cd \cup x P c_l \cup c_l P' c_r \cup c_r P d \cup dd' \cup d' P' c \cup c P y$.
 - vii. Set $S := S \cup S'$.

- By construction, (G', C') does neither contain an L' -bridge nor an isolated bridge; moreover, P' is exactly the preliminary Tutte path of (G', C') computed in Section 4.3.1. Thus, $\text{FindTuttePath}((G', C'), x', u', y', P')$ outputs a Tutte path of (G', C') and stores it in P' . The above construction of P then forwards the changes that were made for P' to P .
- Since P' is a Tutte path of (G', C') and by the 3-Paths Property, the only P' -bridges with two attachments that do not contain an edge of C must have an inner vertex that is a neighbour of d . As $d \in P$, such P' -bridges will become P -bridges with exactly three attachments in G .

4.4 Polynomial Time Bound for Computing Tutte Paths

It remains to show that Algorithm 2 runs in polynomial time. Clearly, all recursive calls are made on pairwise edge-disjoint circuit subgraphs; it is also easy to see that every single step of Algorithm 2 can be computed in polynomial time $O(m^k)$. It thus suffices to show that the number of recursion calls is polynomial in m and that we did not add too many new edges for the recursive calls.

Let $T(m)$ be the running time of Algorithm 2 on G having m edges. If there are j recursive calls made for the circuit graph G , let G_i be the circuit graph of the i th such call and let $m_i := |E(G_i)|$ for all $1 \leq i \leq j$. If we would not add any new edge during Algorithm 2, $T(m) = O(m^k) + \sum_{i=1}^j T(m_i)$. Let w be the neighbor of v_l in $v_l Cx$. As all G_i are edge-disjoint and do not contain the edges uu_1 and $v_l w$, we have $\sum_{i=1}^j m_i \leq m - 2$. Solving the recurrence gives then $T(m) \in O(m^{k+1})$.

However, we may have added an edge cd during the construction of $\eta(K)$ whenever we were in Case 3 of Definition 15. In each such case, the only recursive call made for G in which cd takes part is the one, say G_1 , that computes the Tutte path of $\eta(B_i)$ (see Section 4.3.1). In G_1 and for each such cd , the edge dd' (see Figure 3(a)) is not contained, which restores validity of the above argument.

The most crucial open question that we want to investigate in the future is how the given polynomial running time for computing a special closed 2-walk can be improved to a low order polynomial.

References

- 1 T. Asano, S. Kikuchi, and N. Saito. A linear algorithm for finding Hamiltonian cycles in 4-connected maximal planar graphs. *Discrete Applied Mathematics*, 7(1):1–15, 1984.
- 2 D. Barnette. Trees in polyhedral graphs. *Canadian Journal of Mathematics*, 18:731–736, 1966.
- 3 T. Biedl. Trees and co-trees with bounded degrees in planar 3-connected graphs. In *14th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT'14)*, pages 62–73, 2014.
- 4 N. Chiba and T. Nishizeki. A theorem on paths in planar graphs. *Journal of graph theory*, 10(4):449–450, 1986.
- 5 N. Chiba and T. Nishizeki. The Hamiltonian cycle problem is linear-time solvable for 4-connected planar graphs. *Journal of Algorithms*, 10(2):187–211, 1989.
- 6 R. Diestel. *Graph Theory*. Springer, fourth edition, 2010.
- 7 Z. Gao and R. B. Richter. 2-Walks in circuit graphs. *Journal of Combinatorial Theory, Series B*, 62(2):259–267, 1994.
- 8 Z. Gao, R. B. Richter, and X. Yu. 2-Walks in 3-connected planar graphs. *Australasian Journal of Combinatorics*, 11:117–122, 1995.

- 9 Z. Gao, R. B. Richter, and X. Yu. Erratum to: 2-Walks in 3-connected planar graphs. *Australasian Journal of Combinatorics*, 36:315–316, 2006.
- 10 M. R. Garey, D. S. Johnson, and R. E. Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM J. Comput.*, 5(4):704–714, 1976.
- 11 D. Gouyou-Beauchamps. The Hamiltonian circuit problem is polynomial for 4-connected planar graphs. *SIAM Journal on Computing*, 11(3):529–539, 1982.
- 12 F. Harary and G. Prins. The block-cutpoint-tree of a graph. *Publ. Math. Debrecen*, 13:103–107, 1966.
- 13 B. Jackson and N. C. Wormald. k -Walks of graphs. *Australasian Journal of Combinatorics*, 2:135–146, 1990.
- 14 W.-B. Strothmann. *Bounded degree spanning trees*. PhD thesis, FB Mathematik/Informatik und Heinz Nixdorf Institut, Universität-Gesamthochschule Paderborn, 1997.
- 15 C. Thomassen. A theorem on paths in planar graphs. *Journal of Graph Theory*, 7(2):169–176, 1983.
- 16 W. T. Tutte. A theorem on planar graphs. *Transactions of the American Mathematical Society*, 82:99–116, 1956.
- 17 H. Whitney. A theorem on graphs. *Annals of Mathematics*, 32(2):378–390, 1931.
- 18 H. Whitney. Non-separable and planar graphs. *Transactions of the American Mathematical Society*, 34(1):339–362, 1932.

Towards an Isomorphism Dichotomy for Hereditary Graph Classes

Pascal Schweitzer

RWTH Aachen University
Ahornstraße 55, 52074 Aachen, Germany
schweitzer@informatik.rwth-aachen.de

Abstract

In this paper we resolve the complexity of the isomorphism problem on all but finitely many of the graph classes characterized by two forbidden induced subgraphs. To this end we develop new techniques applicable for the structural and algorithmic analysis of graphs. First, we develop a methodology to show isomorphism completeness of the isomorphism problem on graph classes by providing a general framework unifying various reduction techniques. Second, we generalize the concept of the modular decomposition to colored graphs, allowing for non-standard decompositions. We show that, given a suitable decomposition functor, the graph isomorphism problem reduces to checking isomorphism of colored prime graphs. Third, we extend the techniques of bounded color valence and hypergraph isomorphism on hypergraphs of bounded color class size as follows. We say a colored graph has generalized color valence at most k if, after removing all vertices in color classes of size at most k , for each color class C every vertex has at most k neighbors in C or at most k non-neighbors in C . We show that isomorphism of graphs of bounded generalized color valence can be solved in polynomial time.

1998 ACM Subject Classification G.2.2 Graph Theory, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases graph isomorphism, modular decomposition, bounded color valence, reductions, forbidden induced subgraphs

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.689

1 Introduction

Given two graphs G_1 and G_2 , the graph isomorphism problem asks whether there exists a bijection from the vertices of G_1 to the vertices of G_2 that preserves adjacency and non-adjacency. In this paper we continue the systematic investigation of the complexity of graph isomorphism on hereditary graph classes with a focus on classes characterized by finitely many forbidden induced subgraphs as initiated in [21] (see also [22]).

Given a set of finite graphs H_1, \dots, H_t we define (H_1, \dots, H_t) -free to be the class of all graphs that do not contain any H_i as an induced subgraph. In the light of the unknown complexity status of the graph isomorphism problem, the goal in this context is typically to classify the complexity for the various graph classes into being polynomial time solvable or isomorphism complete (i.e., polynomially equivalent to graph isomorphism). Recently, in [26] it is shown that, assuming that graph isomorphism is not polynomial time solvable in general, there exist graph classes closed under taking (not necessarily induced) subgraphs which are of intermediate complexity. Trivially, this implies the same conditional existence of hereditary graph classes (i.e., graph classes characterized by forbidden induced subgraphs) of intermediate complexity. However, the construction in [26] intrinsically requires the use of infinitely many forbidden subgraphs. In contrast to this, in [26] it is also shown that there are



© Pascal Schweitzer;

licensed under Creative Commons License CC-BY

32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).

Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 689–702

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



SYMPOSIUM
ON THEORETICAL
ASPECTS
OF COMPUTER
SCIENCE

no intermediate graph classes characterized by finitely many forbidden subgraphs. However, this statement does not carry over to forbidden induced subgraphs and the question of the existence of intermediate graph classes characterized by finitely many forbidden induced subgraphs remains open. A more precise statement of the dichotomy result in [26] is that a graph class characterized by finitely many forbidden subgraphs has a polynomial time solvable graph isomorphism problem if one of the forbidden graphs is a union of subdivided stars, and is graph isomorphism complete otherwise. These graphs, the forests of subdivided stars, also play a central role in the complexity of hereditary graph classes.

With respect to classes defined by forbidden induced subgraphs, there is a dichotomy for the isomorphism problem on H_1 -free graphs into polynomially solvable and isomorphism complete cases. In [21] the complexity of the isomorphism problem on (H_1, H_2) -free graphs is determined for various pairs (H_1, H_2) and the results also follow the “polynomially solvable versus isomorphism complete” dichotomy. The crucial cases that were not resolved are those where either H_1 or H_2 is a complete graph. More specifically, except for finitely many cases, the unresolved cases were shown to be polynomially equivalent to a class where one of the graphs is a complete graph. In the light of all of these results we conjecture the following:

► **Conjecture 1.** *If $\mathcal{C} = (H_1, \dots, H_t)$ -free is a graph class defined by the finite set of forbidden induced subgraphs H_1, \dots, H_t then graph isomorphism of graphs in \mathcal{C} is polynomial time solvable or isomorphism complete.*

In this paper we continue the investigation of the complexity of the isomorphism problem on the graph classes focusing on the case of two forbidden subgraphs, where one of the graphs is complete. As mentioned above these are the crucial cases that were not resolved. For the resolution of those classes, techniques beyond those that were developed in [21] are required. These techniques are presented in this paper leading to the following theorem:

► **Theorem 2.** *On all but finitely many classes of the form (H_1, H_2) -free the graph isomorphism problem is polynomial time solvable or isomorphism complete.*

Contribution. In order to prove the theorem, three new techniques for the structural and algorithmic analysis of graphs are developed in this paper.

Firstly, we develop a methodology to show isomorphism completeness of the isomorphism problem on graph classes by providing a unifying framework for various reductions typically used for that purpose. The advantage of this framework is that it allows a streamlined abstract way to argue why some class is isomorphism complete, which boils down to an algorithmically checkable argument.

Secondly, we generalize the modular decomposition to colored graphs and define the concept of a colored modular decomposition with respect to a decomposition functor. This not only allows us to show that the graph isomorphism problem reduces to colored isomorphism of prime graphs but it also allows us to decompose graphs that are prime with respect to the classic modular decomposition. To show this reduction we also describe how to remodel an algorithm that has access to an oracle producing a complete invariant for a graph class into an algorithm that only has access to an isomorphism test of the graph class.

Thirdly, we extend the techniques of bounded color valence and hypergraph isomorphism on hypergraphs of bounded color class size as follows. We say a colored graph has generalized color valence at most k if, after removing all vertices in color classes of size at most k , for each color class C every vertex has at most k neighbors in C or at most k non-neighbors in C . We show that isomorphism of graphs of bounded generalized color valence can be solved in polynomial time. This generalization allows us to perform isomorphism tests for graphs

whose automorphism group cannot be forced into having bounded size composition factors even when using finitely many individualization steps. Since for such graphs alternating groups of unbounded size can appear among the composition factors, it seems that the standard group theoretic techniques cannot be directly applied. This shows that indeed new techniques were required to solve the particular cases.

We apply the three mentioned techniques to resolve the complexity of isomorphism on all but finitely many of the graph classes characterized by two forbidden induced subgraphs. For the resolved classes we either provide a reduction from the general problem or a polynomial time algorithm. The applications of the techniques include for example showing that bipartite graphs that are free from a fixed forbidden double star refine into graphs of bounded generalized color valence. This can be used to show that isomorphism of graphs of bounded clique number with a fixed forbidden double star can be solved in polynomial time. We in turn apply this in conjunction with the colored modular decomposition technique to solve isomorphism for graphs of bounded clique number which do not contain P_5 (a path of length 4) as an induced subgraph. We apply the general reductions to show that classes of graphs without cliques of size 4 and certain unions of paths are extensive enough to have an isomorphism problem that is isomorphism complete. Furthermore, we apply the modular decomposition techniques in conjunction with the bounded generalized color valence to analyze the structure of various graph classes of bounded clique number with certain forbidden forests of subdivided stars.

Related work. We refer the reader to [3, 18, 28] for an introduction to the diverse complexity-theoretic results related to the isomorphism problem. There are numerous results known on the complexity of graph isomorphism of hereditary graph classes. A collection showing the problem for many classes to be equivalent to the general problem is given by Booth and Colbourn [5]. In that paper, the complexity of classes characterized by one forbidden subgraph H is shown to depend on whether the graph H is an induced subgraph of P_4 (the path on 4 vertices). The systematic study of classes characterized by two forbidden subgraphs was initiated in [21].

With regard to algorithms, a recent very general result, implying many results devised earlier on more special graph classes, is a theorem of Grohe and Marx [13] that shows that isomorphism of graph classes defined by a forbidden topological minor can be solved in polynomial time.

There are several applications of modules or some form of modular decomposition in the context of graph isomorphism. For example, Goldberg's plain exponential algorithm [11] uses the concept of sections, which can be seen as colored modules. Furthermore, Junttila and Kaski [17] define non-uniform components within the individualization-refinement approach that also constitute colored modules. As described below, Rao [27] also exploits the classic modular decomposition to devise an isomorphism algorithm for gem and co-gem free graphs (i.e., $(P_4 \cup K_1, \overline{P_4 \cup K_1})$ -free graphs). His technique can be seen as a special case of the techniques using colored modular decompositions described in this paper. Other hereditary graph classes, for which isomorphism algorithms make use of modular decomposition, are for example subclasses of circular-arc graphs (see [7]).

In the early stages of the discovery of the group theoretic technique for the isomorphism problem, Luks [24] applied the technique to show that isomorphism of graphs of bounded maximum degree can be solved in polynomial time. Babai [2] applied the notion of bounded color valence in his algorithm for the general isomorphism problem. Miller (see [25]) applied this technique in a series of papers to perform isomorphism tests of k -separable and k -contactable graphs as well as isomorphism of hypergraphs of bounded color class size (see

also [1], [4] and [30]). In our generalization in this paper, only the subgraph induced by the color classes that are not of bounded size is required to exhibit bounded color valence.

There have been several studies aiming at dichotomy results for computational problems on graph classes characterized by forbidden induced subgraphs. For example, this has been done for the computation of the chromatic number [20], dominating sets [23], coloring [9], and list coloring [12] for two forbidden subgraphs.

Moreover there are numerous results analyzing whether the clique width of a graph class characterized by forbidden subgraphs is bounded (see [8] for an extensive list of references).

Structure of the paper. We mainly apply the three techniques developed in this paper to classes characterized by two forbidden induced subgraphs. However, the intention behind their presentation is to allow them to be applicable in a broader sense to practical and theoretical algorithms for isomorphism of general graphs or maybe for the classes of bounded clique width. The paper first presents the techniques and then hints at how to apply them to various classes of two forbidden subgraphs. The full version [29] provides details and proofs that have mostly been omitted.

In the first part of the paper we provide preliminaries such as introducing notation and recalling basic tools (Section 2). We then devise a methodology to prove isomorphism completeness results (Section 3) and briefly describe how to simulate a complete invariant given only an isomorphism algorithm (Section 4). After this we turn to techniques for isomorphism testing using modular decompositions (Section 5) and devise a polynomial time algorithm for graphs of bounded generalized color valence (Section 6).

The reduction techniques can be applied to show the isomorphism completeness of various graph classes characterized by forbidden induced subgraphs. The algorithmic techniques can be applied to graph classes with forbidden double stars and graph classes of graphs without induced paths of length 4. The techniques can also be applied to analyze specific triangle-free graphs and specific graphs of bounded clique number. (For details see [29].) We conclude by showing that together with the theorems in [21] this resolves the complexity of all but finitely many graph classes defined by two forbidden induced subgraphs (Section 7).

2 Preliminaries

In this paper all graphs are finite, simple, undirected graphs. For a graph G , by $V(G)$ and $E(G)$ we denote the vertex set and the edge set, respectively. By $N_G(S) = N(S)$ we denote the neighborhood of a set S , i.e., the vertices in $V(G) \setminus S$ adjacent to some vertex in S . By id we always denote the identity map. The bipartite complement of a bipartite graph G with bipartition classes A and B is obtained by replacing $E(G)$ with $\{\{a, b\} \mid a \in A, b \in B\} \setminus E(G)$.

We write $H \leq G$ if the graph G contains a graph H as an induced subgraph. A graph G is H -free if $H \not\leq G$. It is (H_1, \dots, H_k) -free, if it is H_i -free for all i . A graph class \mathcal{C} is H -free (respectively (H_1, \dots, H_k) -free) if this is true for all $G \in \mathcal{C}$. A graph class \mathcal{C} is *hereditary* if it is closed under taking induced subgraphs. The class (H_1, \dots, H_k) -free is the class of all (H_1, \dots, H_k) -free graphs. Note that each class (H_1, \dots, H_k) -free is hereditary. We say a graph G *contains* a graph H (as an induced subgraph) if an induced subgraph of G is isomorphic to H .

By I_t , K_t , P_t , and C_t we denote the independent set, the clique, the path, and the cycle on t vertices, respectively. The *clique number* of a graph G is the largest integer t such that G contains K_t . By $H \dot{\cup} H'$ we denote the disjoint union of H and H' ; we use tH for the disjoint union of t copies of the graph H . By \overline{G} we denote the (edge) complement of G . The graph $\overline{K_2 \dot{\cup} I_2}$, i.e., the graph obtained from K_4 by deleting an edge, is called the *diamond*.

A *star* is a graph isomorphic to the complete bipartite graph $K_{1,t}$ for some positive integer t . The *subdivision* of an edge is the replacement of the edge with a path of length two. A *subdivided star* is a possibly repeated subdivision of a star. If a subdivided star has a vertex of degree at least 3 then this vertex is unique and called the *center*. For non-negative integers a_0, \dots, a_t with $a_t > 0$, we define the graph $H(a_t, \dots, a_1, a_0)$ to be the disjoint union of an independent set of size a_0 with the following subdivided star H . The star H is the subdivided star that for $i \in \{1, \dots, t\}$ has exactly a_i leaves at distance i from the center and no other leaves. (If $\sum_{i=1}^t (a_i) < 3$ the center of H is defined so that the graph H is a path whose two leaves have suitable distances from that center.) In [21] it is shown that isomorphism of (H_1, \dots, H_t) -free graphs is isomorphism complete unless one of the forbidden graphs is a forest of subdivided stars.

In this paper a *colored graph* is a vertex colored graph whose coloring does not need to be proper (i.e., adjacent vertices can have the same color). Isomorphisms between colored graphs are required to respect the colors. The *naive vertex refinement* algorithm, or 1-dimensional Weisfeiler-Lehman algorithm, is a standard technique of repeatedly recoloring the vertices, refining the partition induced by the colors, by using the multiplicity of colors appearing among the neighbors of a vertex (see for example [28]). It has the property that after the refinement, the number of neighbors a vertex v has in a color class C only depends on the colors of v and C . Such a coloring is called *stable*. A graph has *color valence at most k* if for every vertex v and every color class C there are at most k neighbors of v in C or there are at most k non-neighbors of v in C .

3 Reductions

In this section we develop a systematic approach to proving isomorphism invariant reductions. This provides general means to construct isomorphism complete graph classes. Standard reductions like subdividing, taking complements, and adding isolated vertices fall into this framework. Likewise, most reductions performed in [21] and also various reductions in [5] fall into this framework.

► **Definition 3.** Let J be a finite set and $L: J \times J \rightarrow \{A, N\}$ be a labeling assigning every ordered pair of vertices the label A for adjacent or N for non-adjacent. Moreover let $L_N: J \times J \rightarrow \mathbb{N} \cup \{\infty\}$ be a labeling assigning every ordered pair of vertices a non-negative integer or infinity.

A graph G belongs to *the class encoded by the labeled graph (J, L, L_N)* if there exists a map $\phi: V(G) \rightarrow J$ such that the following hold:

1. If $v \in V(G)$ and $j \in J$ such that $L(\phi(v), j) = A$ and $L_N(\phi(v), j) \neq \infty$ then there are at most $L_N(\phi(v), j)$ vertices $v' \in V(G) \setminus \{v\}$ that are non-adjacent to v such that $\phi(v') = j$.
2. If $v \in V(G)$ and $j \in J$ such that $L(\phi(v), j) = N$ and $L_N(\phi(v), j) \neq \infty$ then there are at most $L_N(\phi(v), j)$ vertices $v' \in V(G) \setminus \{v\}$ that are adjacent to v such that $\phi(v') = j$.

In this definition, the triple (J, L, L_N) should be thought of as a generalized graph. The graph class encoded by (J, L, L_N) then contains graphs that can be obtained by replacing the elements of J with sets of vertices. In some contexts, this is referred to as blowing-up the elements of J . Adjacency of the new vertices is essentially governed by the adjacency of the original graph. However, the values of L_N control the number of exceptions to this rule that are allowed per vertex.

The definition captures various constructions used in graph theory. A first class of examples is formed by the complete multipartite graphs, which turn out to be graphs modeled

with the function L satisfying $L(x, y) = N$ if and only if $x = y$ and the function L_N being the constant function evaluating to 0. A second, more general class of examples modeled by the definition is the graphs of bounded color valence, which are graphs that frequently appear in the context of graph isomorphism. Such graphs are obtained whenever L_N is a bounded function. The coloring corresponds to the map ϕ . A third example is the class of graphs that map homomorphically onto a finite graph H . This class is obtained by letting (J, L) model the graph H (i.e., $V(H) = J$ and $L(x, y) = A$ if and only if $\{x, y\}$ is an edge of H) and setting $L_N(x, y) = 0$ if $L(x, y) = N$ and $L_N(x, y) = \infty$ otherwise.

Of interest to us is the complexity of the isomorphism problem of graph classes encoded by a triple (J, L, L_N) . It turns out that when L_N is a bounded function then isomorphism can be solved in polynomial time.

► **Theorem 4.** *Let (J, L, L_N) be a triple that encodes a graph class. If all values of L_N are finite then isomorphism of graphs encoded by (J, L, L_N) can be solved in polynomial time.*

The theorem implicitly shows that isomorphism of graphs that have bounded color valence for some coloring that uses a bounded number of color classes can be solved in polynomial time, even if the color classes are not given.

► **Corollary 5.** *For every positive integer c , graph isomorphism of graphs whose vertices can be partitioned into c color classes such that the graph has color valence at most c can be solved in polynomial time.*

While encodings can be used to show polynomial time solvability of certain graph classes, they can also be used to show hardness results as follows.

► **Definition 6.** An encoding (J, L, L_N) is a *simple path encoding* in case L is symmetric (i.e., if $L(j, j') = L(j', j)$ holds) and there is a sequence of vertices (p_1, \dots, p_t) of length at least 2 in J such that $L_N(p_1, p_2) = \infty$, $L_N(p_t, p_{t-1}) \geq 2$ and for all $k \in \{1, \dots, t-1\}$ we have $L_N(p_k, p_{k+1}) \geq 1$ and $L_N(p_{k+1}, p_k) \geq 1$.

Intuitively, a simple path encoding allows enough freedom to encode bipartite graphs with one bipartition class having vertices of degree two. We can formally prove this statement in the form of a reduction.

► **Theorem 7.** *A class of graphs encoded by a simple path encoding is isomorphism complete.*

The theorem can be applied to show that various classes are isomorphism complete.

► **Theorem 8.** *The classes $(2K_2 \dot{\cup} K_1, K_4)$ -free, $(P_6, P_4 \dot{\cup} P_2, K_4)$ -free, $(H(1, 0, 3, 0), K_4)$ -free, bipartite $(2P_3 \dot{\cup} K_1)$ -free, $(H(1, 0, 2, 0), K_5)$ -free are isomorphism complete.*

The theorem in turn implies that various classes (H_1, H_2) -free that are superclasses of one of these classes are isomorphism complete (such as $(K_3, 2P_3 \dot{\cup} K_1)$ -free).

4 Isomorphism, Invariants and Canonical Labeling

A *complete graph invariant* for a graph class \mathcal{C} is a map $\text{Inv}: \mathcal{C} \rightarrow \mathcal{D}$ into some class \mathcal{D} such that for graphs G_1 and G_2 in \mathcal{C} we have $\text{Inv}(G_1) = \text{Inv}(G_2)$ if and only if G_1 and G_2 are isomorphic. A *canonical labeling* is a map that assigns to every graph G a graph $C(G)$ with vertex set $V(C(G)) = \{1, \dots, |G|\}$ and an isomorphism $\phi: G \rightarrow C(G)$ such that the map assigning $C(G)$ to G is a complete invariant. There are relatively general techniques with which one can turn a complete invariant into a canonical labeling algorithm [14, 15, 19].

In this paper, we are mainly interested in isomorphism algorithms, as opposed to canonical labeling algorithms or complete invariants. We will therefore require a tool to simulate an invariant within one execution of our algorithm, given only an algorithm that performs isomorphism checks. Our simulated invariant will not be consistent across different calls of the same algorithm.

► **Theorem 9.** *Let A be a polynomial time algorithm with access to a complete invariant \mathcal{O} for a graph class \mathcal{C} given as oracle. Suppose the outputs of A are independent of the choice of the invariant \mathcal{O} . If isomorphism of graphs in \mathcal{C} can be solved in polynomial time then there is a polynomial-time algorithm B whose outputs coincide with those of A , which does not require access to an oracle.*

We use the theorem to replace an invariant with a isomorphism algorithm in the next section when dealing with modular decompositions (more precisely to prove Theorem 15).

5 Colored modular decomposition

In this section we are concerned with modular decompositions and their application to the isomorphism problem. We will work with colored graphs since this is convenient in the graph isomorphism context. However, we will also generalize the concept of a module to that of a colored module, since this is required by our later applications. Since we do not require previous knowledge about the uncolored modular decomposition, we will not define it. We refer the reader to the survey by Habib and Paul [16] for more information on the uncolored decomposition and its algorithmic applications. For the colors, we will assume that there is a linear order on the colors. In algorithmic applications such a linear order can always be obtained by comparing the bit-strings corresponding to the colors lexicographically.

► **Definition 10.** Let G be a colored graph. A *colored module* is a subset M of $V(G)$ such that for all $v \in V(G) \setminus M$, if $x, x' \in M$ are of the same color then either v is adjacent to both vertices x and x' or to neither x nor x' .

A module M is *non-trivial* if it contains at least two vertices that cannot be distinguished by vertices outside of the module. That is, M is non-trivial if there are $x, y \in M$ such that for all $v \in V(G) \setminus M$ the vertex v is adjacent to x if and only if it is adjacent to y . Note that every module that contains two vertices of the same color is non-trivial.

► **Definition 11.** A map assigning to every graph G a subset of the vertices $M(G)$ is said to be *isomorphism invariant* if for every isomorphism $\phi: G \rightarrow G'$ we have $\phi(M(G)) = M(G')$. A map that assigns to every graph G a partition of a subset of the vertices $M(G) = \{M_1, \dots, M_k\}$ is said to be *isomorphism invariant* if for every isomorphism $\phi: G \rightarrow G'$ we have $M(G') = \{\phi(M_1), \dots, \phi(M_k)\}$.

► **Definition 12.** A *decomposition functor* is a map assigning to every graph G a partition of a subset of the vertices into modules that is isomorphism invariant.

A graph G is *prime* with respect to a decomposition functor Mod if $\text{Mod}(G)$ does not contain non-trivial modules. While there is a standard decomposition functor for the uncolored case, for the colored case it is in general not clear whether we can find a useful functor to decompose the graphs, and we have to find such a functor for a given graph class first, in order to decompose the graphs. See [29] for an example illustrating that this allows us to decompose graphs that are prime with respect to classic modular decomposition.

In the remainder of this section, we argue for certain decomposition functors that graph isomorphism for a hereditary graph class can be solved in polynomial time if the isomorphism problem for colored prime graphs in the class can be solved in polynomial time. To facilitate the proof we can assume that we are given a complete invariant for the prime graphs in the hereditary graph class and then apply Theorem 9.

Our next goal is to define the concept of the quotient graph. In the uncolored case, the quotient graph is obtained by replacing each module with a single vertex whose adjacency to the rest of the graph is the same as that of every vertex of the module. However, for the colored case, the adjacency to the rest of the graph depends on the color of the vertex in the module. This means that for every adjacency type we need to retain a vertex that has the same adjacency type with respect to vertices outside the module. (For a vertex v , the adjacency type with respect to vertices outside the module is the set of vertices outside the module adjacent to v .)

A *replacement operator* is an isomorphism invariant map that assigns every non-trivial module M in a decomposition of a graph G an induced subgraph of M in which the vertices are possibly recolored. We require that for every adjacency type of vertices in M at least one vertex of M is maintained. Let $\text{Inv}(M)$ be a complete graph invariant. Given a family of modules $\{M_1, \dots, M_k\}$ that partitions the graph, the *quotient graph* is obtained by simultaneously replacing all modules using the replacement operator (i.e., removing from the module M_i all vertices not in the induced subgraph assigned to M_i by the replacement operator) and then recoloring every vertex v as the triple $(\chi(v), L, \text{Inv}(M_v))$, where $\chi(v)$ is the color of v after the replacement, L is a list of the colors of vertices with the same adjacency type as v , and $\text{Inv}(M_v)$ is the invariant of the module containing v . We say that the decomposition functor is *simple* with respect to a replacement operator if for every complete invariant every quotient graph is prime. Intuitively, this means that the decomposition functor provides us with maximal modules.

► **Definition 13.** Given a decomposition functor, we say a replacement operator is *reversible* if the following holds: two graphs G_1 and G_2 are isomorphic if and only if their colored quotient graphs with respect to the decomposition functor and the replacement operator are isomorphic.

Note that reversibility does not depend on the complete invariant that is used for the recoloring of the quotient graph.

We remark that for uncolored graphs the definitions of module, primality and the quotient graph coincide with the usual definition from the literature (see [16]). In that context, the decomposition functor is typically chosen to partition the graph into components, components of the complement graph, or maximal modules. The replacement operator simply replaces the entire module by one vertex. However, for the applications we have in mind, we require the more general concept of colored modules. Certain conditions immediately imply that a replacement operator is reversible.

► **Lemma 14.** *A replacement operator is reversible if 1.) there is only one trivial module, 2.) all replacements contain only one vertex, or 3.) non-trivial modules induce connected graphs, but the non-trivial modules are pairwise non-adjacent.*

A reversible simple decomposition functor can be used to test isomorphism by considering only isomorphisms between quotient graphs and modules. Iterating this yields an isomorphism test for decomposable graphs if one has access to a complete invariant for prime graphs. Using Theorem 9 we can replace the requirement for a complete invariant by an isomorphism algorithm.

► **Theorem 15.** *Let \mathcal{C} be a hereditary graph class and Mod a simple polynomial-time computable decomposition functor with polynomial-time computable, reversible replacement operator R for colored graphs in \mathcal{C} . If the isomorphism problem for colored prime graphs in \mathcal{C} can be solved in polynomial time then the isomorphism problem of all graphs in \mathcal{C} can be solved in polynomial time.*

The theorem in particular applies to the standard uncolored modular decomposition. This decomposition is associated with a simple decomposition functor with polynomial-time computable, reversible replacement. In his diploma thesis, Fuhlbrück [10] also describes a reduction for the standard uncolored decomposition functor. Already when trying to solve the isomorphism problem for uncolored graphs in a graph class \mathcal{C} with the described method, the fact that the quotient graph needs to be colored implies that we require an isomorphism algorithm for colored prime graphs. In [27], Rao describes a special case of the theorem for the uncolored modular decomposition, essentially considering hereditary graph classes in which every prime graph is of bounded size. Moreover, in the isomorphism context, the bi-join decomposition, also described in [27], can also be treated by using colored modules.

In our applications of Theorem 15 we also use another unrelated technique of dealing with small color classes that we describe next.

6 Bounded generalized color valence

In this section we show that isomorphism of graphs of bounded generalized color valence can be solved in polynomial time. However, the proofs of the lemmas and theorems within the section require familiarity with the computational group theoretic methods that have been developed within the context of the isomorphism problem. For example, when computing automorphism groups, cosets, and sets of isomorphisms between two combinatorial objects, we use a succinct representation by generators and a representative. We also use the set stabilizer theorem [24] for groups with composition factors of bounded size (see also [1, 4, 25]), which implies that for a permutation group with composition factors of bounded size acting faithfully on a set, we can compute the stabilizer of any given subset in polynomial time. For a good overview over the various computational group theoretic techniques, we refer the reader to [30].

Before we solve the isomorphism problem of graphs of bounded generalized color valence, let us consider some examples. Hypergraphs on sets of bounded color class size can directly be encoded as graphs by adding a vertex for every hyperedge that is adjacent to the elements of the hyperedge. Thus, a polynomial time algorithm for bounded generalized color valence also gives rise to a polynomial time algorithm for hypergraphs of bounded color class size. However, there are examples that are not captured by this. Consider a graph of maximum degree at most c consisting of a large number of components such that there is only a small number of isomorphism types among these components. Now add an arbitrary finite number of new vertices colored with new colors such that the added vertices form color classes of size at most c . The new vertices are connected via edges in an arbitrary way to the original vertices. The resulting graph has bounded generalized color valence. However, its automorphism group may contain composition factors of arbitrarily large size. The isomorphism algorithm for graphs of bounded degree exploits the fact that at least for components it is possible, by individualizing one vertex, to obtain a group with bounded composition factor. In the example graphs just described, it is not clear how the standard group theoretic arguments can be applied.

Nevertheless, our goal is to prove that graph isomorphism of graphs with bounded generalized color valence can be solved in polynomial time. Note that the classes of bounded generalized color valence are closed under refinement operations such as individualization and naive vertex refinement.

To solve the isomorphism problem, we need to deal with the small color classes. If there are only a bounded number of them, we can apply individualization to all vertices in small color classes. However, the number of small color classes can grow linearly in the number of vertices. To remedy this problem, we exploit the existence of certain colored modules and use group theoretic techniques.

We first define a decomposition functor that works well with graphs of bounded generalized color valence that have been refined with naive vertex refinement. Given a subset S of vertices, we say a vertex $v' \notin S$ contained in the color class C' is of degree dependence at most d with respect to S if there is a vertex v in S such that $v' \in N(v) \wedge |(N(v) \cap C')| \leq d$ or $v' \notin N(v) \wedge |C' \setminus (N(v))| \leq d$. Intuitively, this definition says that there is a vertex $v \in S$ such that individualization of this vertex followed by refinement with respect to adjacency towards v produces a set of size at most d within which v' can be found.

► **Definition 16.** A non-empty subset S is a d -degree dependence module, if no vertex outside S has degree dependence at most d with respect to S .

► **Lemma 17.** Let G be a stable graph of color valence at most c . If G does not contain color classes of size at most $2c$ then the minimal c -degree dependence modules partition G . Moreover, the map assigning every such graph the family of minimal c -degree dependence modules is a polynomial-time computable decomposition functor.

In Lemma 17, using naive vertex refinement is essential. Indeed, consider a wheel, i.e., a cycle with an added center adjacent to every other vertex. In this graph, there is only one non-trivial 3-degree dependence module, namely the set that contains only the center of the wheel. Thus the set of 3-degree dependence modules does not partition the graph.

By defining a faithful group action of the automorphism group of a graph on the c -degree dependence modules it is possible to gradually compute the automorphism group, yielding also an isomorphism test for such graphs.

► **Theorem 18.** Graph isomorphism for colored graphs of generalized color valence at most c can be solved in polynomial time.

We can combine Theorem 18 and Theorem 15 to show that several graph classes have a polynomial-time solvable isomorphism problem. These classes include various subcases for which the complexity of the isomorphism problem was also previously unresolved.

► **Theorem 19.** The isomorphism problem for the classes $(H(1, b, 0), K_s)$ -free, (P_5, K_t) -free, $(H(1, 0, b, 1), K_3)$ -free and $(K_{1,s} \cup K_{1,s}, K_t)$ -free can be solved in polynomial time.

7 Comprehensiveness of the case distinction

We now argue that the theorems developed throughout this paper together with the theorems from [21] resolve, except for finitely many cases, the complexity of graph isomorphism for (H_1, H_2) -free graphs. In fact, said theorems also either provide a polynomial time algorithm or a reduction from the general isomorphism problem to the respective classes.

Proof of Theorem 2. Let (H_1, H_2) -free be the graph class for which we want to determine the complexity. By the results in [21], we may assume that one of the graphs, H_2 say, is a

complete graph. Since graph isomorphism for K_2 -free graphs is polynomial-time solvable, we further assume that H_2 has at least 3 vertices. By [21, Lemma 2] we may consequently assume that H_1 is a forest of subdivided stars, since the problem is graph isomorphism complete otherwise. If H_1 contains 3 non-trivial components, then H_1 contains $3K_2$ and the problem is isomorphism complete [21, Lemma 5]. In the following, we can thus assume that H_1 has at most 2 non-trivial components. If H_1 has no vertex of degree 3 then if H_1 is sufficiently large, it is either of the form $P_i \dot{\cup} I_t$ with $i \leq 3$ or it contains the graph $2K_2 \dot{\cup} I_2$. In the former case, graph isomorphism is polynomial-time solvable by Theorem 19. In the latter case, the isomorphism problem is graph isomorphism complete [21, Lemma 5].

(*The case $H_2 = K_3$*). Suppose H_2 is the graph K_3 . Since H_2 is fixed, we can assume that H_1 is sufficiently large and we can thus assume by the observation above that H_1 contains a vertex of degree 3. If H_1 does not contain a P_4 then H_1 is a union of at most two stars plus isolated vertices. If there is at most one star, the problem is polynomial-time solvable by Theorem 19 (or by [21, Theorem 4]). Assuming there are two stars, if there is more than one isolated vertex then H_1 contains $2K_2 \dot{\cup} 2K_1$ and the problem is isomorphism complete by [21, Lemma 5]. If neither of the stars is only a single edge then if there exists an isolated vertex in H_1 the problem is isomorphism complete by Theorem 8 and if there is no isolated vertex in H_1 it is polynomial-time solvable by Theorem 19. Finally, if one of the stars is only an edge then by Theorem 19 the problem is polynomial-time solvable.

If H_1 contains a P_4 and there are two non-adjacent vertices not in the same connected component as the P_4 then H_1 contains $P_4 \dot{\cup} 2K_1$ and the problem is graph isomorphism complete [21, Lemma 5]. The assumption that this is not the case implies that the vertex of degree 3 is in the same component as the P_4 . Since isomorphism of $(2K_2 \dot{\cup} 2K_1)$ -free triangle-free graphs is isomorphism complete [21, Lemma 5], by assuming that H_1 is sufficiently large, we may further assume there is at most one additional vertex not in the connected component of the P_4 . If there is only one vertex of degree 1 non-adjacent to the vertex h of degree at least 3, and additionally this vertex has distance 2 from h , then H_1 is an induced subgraph of $H(1, 0, b, 1)$ for some positive integer b . This implies that the problem is polynomial-time solvable by Theorem 19. If H_1 contains two leaves of distance at least 3 from the center, then if H_1 is sufficiently large it also contains $2K_2 \dot{\cup} 2K_1$. If H_1 contains a leaf of distance at least 4 then H_1 contains $P_4 \dot{\cup} 2K_2$. In both cases the problem is graph isomorphism complete by [21, Lemma 5].

(*The case $H_2 = K_n$ with $n > 3$*). Suppose now H_2 is the graph K_n for some $n \geq 4$. Suppose first, H_1 contains two non-trivial components. If it contains an isolated vertex, then it contains $2K_2 \dot{\cup} K_1$ and the problem is isomorphism complete by Theorem 8. If one of the components contains P_4 and the graph is not connected, then the graph contains $P_4 \dot{\cup} K_1$ and the problem is isomorphism complete by [21, Theorem 3]. Otherwise, the two components form a double star and the problem is polynomial-time solvable by Theorem 19.

Thus we may assume now that there is only one non-trivial component. If there is no P_4 in H_1 , the problem is solvable by [21, Theorem 4]. Otherwise, by [21, Theorem 3], we can assume that H_1 is connected. If H_1 is isomorphic to P_5 , the problem is solvable by Theorem 19. If H_1 is isomorphic to P_6 the problem is isomorphism complete by Theorem 8. We can thus assume that H_1 contains a vertex of degree at least 3, which we call the center. If there is only one leaf not adjacent to the center and this leaf has distance 2 from the center, then the problem is polynomial-time solvable by Theorem 19. If there are two leaves not adjacent to the center then H_1 contains $P_4 \dot{\cup} K_1$ and the problem is isomorphism complete by [21, Theorem 3]. If there is a leaf of distance at least 3 from the center and the center has degree 4 then the problem is graph isomorphism complete by Theorem 8.

We can thus assume that the center has degree 3. Under these conditions, if $H_2 = K_4$ and H_1 is sufficiently large, then H_1 contains P_6 and the problem is isomorphism complete by Theorem 8. In the remaining cases $H_2 = K_n$ with $n \geq 5$ and H_1 has a leaf of distance at least 2 from the center. In this case the problem is isomorphism complete by Theorem 8. ◀

8 Conclusion

There is an intricate relationship between the boundedness of the clique width on a graph class and polynomial-time solvability of the isomorphism problem. While there are classes of unbounded clique width for which the isomorphism problem is solvable in polynomial time (for example the graphs of bounded degree or the graphs characterized by a forbidden star and a forbidden clique (see [21])), there are no classes known to be isomorphism complete and known to have bounded clique width. It is conceivable, but not known to be true, that isomorphism on graphs of bounded clique width is solvable in polynomial time. In fact, it seems that typically an isomorphism reduction to a graph class \mathcal{C} yields a proof for the unboundedness of the clique width. It is easy to see that every class of graphs encoded by a simple path encoding has unbounded clique width. The reason is that if G' is the graph produced from a graph G by the reduction detailed in the proof of Theorem 7 then G can be obtained from G' by a finite application of edge complementations between two sets, followed by an unbounded number of local complementations and vertex removals. This shows that the clique width of G is bounded by a function of the clique width of G' . Thus, if G has large clique width then G' has large clique width.

► **Corollary 20.** *The graphs encoded by a simple path encoding have unbounded clique width.*

This implies that many classes (H_1, H_2) -free have unbounded clique width. In particular the class $(P_4 \cup K_1, K_4)$ -free has unbounded clique width, which was previously an unknown case (see [8]). Conversely, when a graph class is shown to be polynomial-time solvable, often this is due to some structural insight that actually amounts to showing that the clique width of the class is bounded. On another note it appears that the solution of the generalized color valence problem presented in this paper, more precisely a generalization of the technique, is a first step towards showing polynomial-time solvability of graphs of bounded clique width. More concretely, the generalization applies to graphs with fixed rank decomposition.

As mentioned in the introduction, for various other computational problems, classification results have been considered for classes characterized by two forbidden graphs. For these problems, there had been extensive prior work and algorithmic techniques were already available. For the isomorphism problem, such techniques were lacking and the intention of this paper is to develop them. The fact that for new classes the complexity of isomorphism can be determined using these techniques, while other methods seem to fail, shows that these techniques provide something conceptually different.

Concerning a list of open cases that still remain for the classes characterized by two forbidden subgraphs, one has to analyze precisely to which classes the techniques can be applied. For example, Brandstädt and Kratsch [6] show that cycles of length 5 in $(P_5, \overline{P_4 \cup K_1})$ -free graphs are disjoint. By coloring vertices inside a 5-cycle depending on whether they have a neighbor outside the cycle, we obtain a decomposition functor. Furthermore, there is a description of the relevant prime graphs in [6] which allows us to apply the modular decomposition technique. This is just an example and a comprehensive description of the various cases that can actually be handled with the techniques remains as future work.

Acknowledgments I thank Matasha Mazis for inspiring comments and suggestions.

References

- 1 Vikraman Arvind, Bireswar Das, Johannes Köbler, and Seinosuke Toda. Colored hypergraph isomorphism is fixed parameter tractable. In *FSTTCS*, pages 327–337, 2010.
- 2 László Babai. Moderately exponential bound for graph isomorphism. In *FCT*, pages 34–50, 1981.
- 3 László Babai. *Handbook of Combinatorics (vol. 2)*, chapter Automorphism groups, isomorphism, reconstruction, pages 1447–1540. MIT Press, 1995.
- 4 László Babai and Eugene M. Luks. Canonical labeling of graphs. In *STOC*, pages 171–183, 1983.
- 5 Kellogg S. Booth and C. J. Colbourn. Problems polynomially equivalent to graph isomorphism. Technical Report CS-77-04, Comp. Sci. Dep., Univ. Waterloo, 1979.
- 6 Andreas Brandstädt and Dieter Kratsch. On the structure of (P_5, gem) -free graphs. *Discrete Applied Mathematics*, 145(2):155–166, 2005.
- 7 Andrew Curtis, Min Lin, Ross McConnell, Yahav Nussbaum, Francisco Soullignac, Jeremy Spinrad, and Jayme Szwarcfiter. Isomorphism of graph classes related to the circular-ones property. *Discrete Mathematics and Theoretical Computer Science*, 15(1):157–182, 2013.
- 8 Konrad Dabrowski and Daniël Paulusma. Clique-width of graph classes defined by two forbidden induced subgraphs. *CoRR*, abs/1405.7092, 2014.
- 9 Konrad K. Dabrowski, Petr A. Golovach, and Daniel Paulusma. Colouring of graphs with ramsey-type forbidden subgraphs. *Theoretical Computer Science*, 522(0):34–43, 2014.
- 10 Frank Fuhlbrück. Fixed-parameter tractability of the graph isomorphism and canonization problems. Diploma thesis, Humboldt-Universität zu Berlin, 2013.
- 11 Mark K. Goldberg. A nonfactorial algorithm for testing isomorphism of two graphs. *Discrete Applied Mathematics*, 6(3):229–236, 1983.
- 12 Petr A. Golovach and Daniël Paulusma. List coloring in the absence of two subgraphs. *Discrete Applied Mathematics*, 166:123–130, 2014.
- 13 Martin Grohe and Dániel Marx. Structure theorem and isomorphism test for graphs with excluded topological subgraphs. In *STOC*, pages 173–192, 2012.
- 14 Yuri Gurevich. From invariants to canonization. *Bulletin of the EATCS*, 63, 1997.
- 15 Yuri Gurevich. From invariants to canonization. In *Current Trends in Theoretical Computer Science*, pages 327–331. World Scientific, 2001.
- 16 Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.
- 17 Tommi A. Junttila and Petteri Kaski. Conflict propagation and component recursion for canonical labeling. In *TAPAS*, pages 151–162, 2011.
- 18 Johannes Köbler, Uwe Schöning, and Jacobo Torán. *The graph isomorphism problem: its structural complexity*. Birkhäuser Verlag, Basel, Switzerland, 1993.
- 19 Johannes Köbler and Oleg Verbitsky. From invariants to canonization in parallel. In *CSR*, pages 216–227, 2008.
- 20 Daniel Král, Jan Kratochvíl, Zsolt Tuza, and Gerhard J. Woeginger. Complexity of coloring graphs without forbidden induced subgraphs. In *WG*, pages 254–262, 2001.
- 21 Stefan Kratsch and Pascal Schweitzer. Graph isomorphism for graph classes characterized by two forbidden induced subgraphs. In *WG*, pages 34–45, 2012.
- 22 Stefan Kratsch and Pascal Schweitzer. Graph isomorphism for graph classes characterized by two forbidden induced subgraphs. *CoRR*, abs/1208.0142, 2012.
- 23 Vadim V. Lozin. A decidability result for the dominating set problem. *Theoretical Computer Science*, 411(44–46):4023–4027, 2010.
- 24 Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25(1):42–65, 1982.

- 25 Gary L. Miller. Isomorphism testing and canonical forms for k -contractable graphs (a generalization of bounded valence and bounded genus). In *FCT*, pages 310–327, 1983.
- 26 Yota Otachi and Pascal Schweitzer. Isomorphism on subgraph-closed graph classes: A complexity dichotomy and intermediate graph classes. In *ISAAC*, pages 111–118, 2013.
- 27 Michaël Rao. Decomposition of (gem,co-gem)-free graphs. Unpublished, available at <http://www.labri.fr/perso/rao/publi/decompgemcogem.ps>, 2007.
- 28 Pascal Schweitzer. *Problems of unknown complexity: Graph isomorphism and Ramsey theoretic numbers*. PhD thesis, Universität des Saarlandes, Germany, 2009.
- 29 Pascal Schweitzer. Towards an isomorphism dichotomy for hereditary graph classes. *CoRR*, abs/1411.1977, 2014.
- 30 Ákos Seress. *Permutation Group Algorithms*. Cambridge Tracts in Mathematics. Cambridge University Press, 2003.

Existential Second-order Logic over Graphs: A Complete Complexity-theoretic Classification

Till Tantau

Institute of Theoretical Computer Science
Universität zu Lübeck, Germany
tantau@tcs.uni-luebeck.de

Abstract

Descriptive complexity theory aims at inferring a problem's computational complexity from the syntactic complexity of its description. A cornerstone of this theory is Fagin's Theorem, by which a property is expressible in *existential second-order logic* (ESO logic) if, and only if, it is in NP. A natural question, from the theory's point of view, is which syntactic fragments of ESO logic also still characterize NP. Research on this question has culminated in a dichotomy result by Gottlob, Kolaitis, and Schwentick: for each possible quantifier prefix of an ESO formula, the resulting prefix class over graphs either contains an NP-complete problem or is contained in P. However, the exact complexity of the prefix classes inside P remained elusive. In the present paper, we clear up the picture by showing that for each prefix class of ESO logic, its reduction closure under first-order reductions is either FO, L, NL, or NP. For undirected self-loop-free graphs two containment results are especially challenging to prove: containment in L for the prefix $\exists R_1 \cdots \exists R_n \forall x \exists y$ and containment in FO for the prefix $\exists M \forall x \exists y$ for monadic M . The complex argument by Gottlob et al. concerning polynomial time needs to be carefully reexamined and either combined with the logspace version of Courcelle's Theorem or directly improved to first-order computations. A different challenge is posed by formulas with the prefix $\exists M \forall x \forall y$, which we show to express special constraint satisfaction problems that lie in L.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.4.1 Mathematical Logic

Keywords and phrases existential second-order logic, descriptive complexity, logarithmic space

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.703

1 Introduction

Fagin's Theorem [9] establishes a tight connection between complexity theory and finite model theory: A language lies in NP if, and only if, it is the set of all finite models (coded appropriately as words) of some formula in *existential second-order logic* (ESO logic). This machine-independent characterization of a major complexity class sparked the research area of descriptive complexity theory, which strives to characterize the computational complexity of languages by the syntactic structure of the formulas that can be used to describe them. Nowadays, syntactic logical characterizations have been found for all major complexity classes, see [13] for an overview, although some syntactic extras (like numerical predicates) are often needed for technical reasons.

When looking at subclasses of NP like P, NL, L, or NC^1 , one might hope that syntactic restrictions of ESO logic can be used to characterize them; and the most natural way of restricting ESO formulas is to limit the number and types of quantifiers used. All ESO formulas can be rewritten in prenex normal form as $\exists R_1 \cdots \exists R_r \forall x_1 \exists x_2 \cdots \forall x_{n-1} \exists x_n \psi$,



© Till Tantau;

licensed under Creative Commons License CC-BY

32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).

Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 703–715

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



SYMPOSIUM
ON THEORETICAL
ASPECTS
OF COMPUTER
SCIENCE

where the R_i are second-order variables, the x_i are first-order variables, and ψ is quantifier-free. Formulas like $\phi_{3\text{-colorable}} = \exists R \exists G \exists B \forall x \forall y ((R(x) \vee G(x) \vee B(x)) \wedge (E(x, y) \rightarrow \neg(R(x) \wedge R(y)) \wedge \neg(G(x) \wedge G(y)) \wedge \neg(B(x) \wedge B(y))))$, which describes the NP-complete problem 3-COLORABLE, show that we do not need the full power of ESO logic to capture NP-complete problems: the prefix $\exists R \exists G \exists B \forall x \forall y$ suffices. However, do formulas of the form, say, $\exists R \forall x \exists y \psi$ also capture all of NP; or do they characterize exactly, say, P? This question lies at the heart of a detailed study by Gottlob, Kolaitis, and Schwentick [11] entitled *Existential Second-Order Logic Over Graphs: Charting the Tractability Frontier*, where the following dichotomy is shown: For each possible syntactic restriction of the quantifier block of ESO formulas, the resulting *prefix class* either contains an NP-complete problem or is contained in P. For instance, it is shown there that all graph problems expressible by formulas of the form $\exists R \forall x \forall y \psi$ lie in P, while some problems expressible by formulas of the form $\exists R \forall x \forall y \forall z \psi$ are NP-complete. The dichotomy does not, however, settle the question of whether all of P – or at least some interesting subclass thereof like logarithmic space (L) or nondeterministic logarithmic space (NL) – is described by one of the logical fragments.

1.1 Contributions of This Paper

One cannot really hope to show that the prefix class of, say, the quantifier prefix $\exists R \forall x \forall y$ is exactly P, since $P \neq \text{NP}$ would follow: This syntactically severely restricted prefix class can be shown [6, Proposition 10.6] to be contained in $\text{NTIME}(n^k)$ for some constant k and is thus provably different from NP by the time hierarchy theorem. The best one can try to prove are statements like “this prefix class is contained in P and contains a problem complete for P” or, phrased more succinctly, “the reduction closure of this prefix class is P.” Our main result, Theorem 1.1, consists of such statements: *For each possible ESO prefix class, its reduction closure under first-order reductions is either FO, L, NL, or NP.* In particular, no prefix class yields P as its reduction closure (unless, of course, $P = \text{NP}$ or $\text{NL} = P$).

It makes a difference which vocabulary we are allowed to use in our formulas and which logical structures we are interested in: Results depend on whether we consider arbitrary graphs, undirected graphs, undirected graphs without self-loops, or just strings. (In this paper, all considered graphs are finite.) The case of strings has been addressed and settled in [6]. In the present paper we consider the same three cases as in [11]: In our vocabulary, we always have just a single binary relational symbol (E), so all models of formulas are graphs. We then differentiate between directed graphs, undirected graphs, and undirected graphs without self-loops (which we call *basic graphs* for brevity). Note that allowing self-loops, whose presence at a vertex x can be tested with the formula $E(x, x)$, is equivalent to considering basic graphs together with an additional monadic input predicate.

To describe the syntactic fragments of ESO logic easily and succinctly, we use the notation of [11]: The uppercase letter E denotes the presence of an existential second-order quantifier, an optional index as in E_2 denotes the arity of the quantifier, and the lowercase letters a and e denote universal and existential first-order quantifiers, respectively. The prefix type of the formula $\phi_{3\text{-colorable}}$ mentioned earlier is $EEEaa$ (or even $E_1E_1E_1aa$ since the predicates are monadic) and we say that $\phi_{3\text{-colorable}}$ has prefix type $EEEaa$ (and also $E_1E_1E_1aa$). We will use regular expressions over the alphabet $\{a, e, E, E_1, E_2, E_3, \dots\}$ to denote patterns of prefix types such as E^*aa for “any number of existential second-order quantifiers followed by exactly two universal first-order quantifiers.” To define the three kinds of prefix classes that we are interested in, for a formula ϕ let $\text{MODELS}_{\text{directed}}(\phi) = \{G \mid G \text{ is a directed graph and } G \models \phi\}$, $\text{MODELS}_{\text{undirected}}(\phi) = \{G \mid G \text{ is an undirected graph and } G \models \phi\}$, and $\text{MODELS}_{\text{basic}}(\phi) = \{G \mid G \text{ is a basic graph and } G \models \phi\}$. For instance,

$\text{MODELS}_{\text{basic}}(\phi_{3\text{-colorable}}) = 3\text{-COLORABLE}$ (ignoring coding issues). Next, for a prefix type pattern P , let $\text{FD}_{\text{directed}}(P) = \{\text{MODELS}_{\text{directed}}(\phi) \mid \phi \text{ has a prefix type in } P\}$ and define $\text{FD}_{\text{undirected}}(P)$ and $\text{FD}_{\text{basic}}(P)$ similarly for undirected and basic graphs. “FD” stands for “Fagin-definable” and Fagin’s Theorem can be stated succinctly as $\text{FD}_{\text{strings}}(E^*(ae)^*) = \text{NP}$.

As stated earlier, in the context of syntactic fragments of ESO logic it makes sense to consider reduction closures of prefix classes rather than the prefix classes themselves. It will not matter much which particular kind of reductions we use, as long as they are weak enough. All our reductions will be *first-order reductions* [13], which are first-order queries with access to the bit predicate or, equivalently, functions computable by a logarithmic-time-uniform constant-depth circuit family.¹ Let us write $A \leq_{\text{fo}} B$ if A can be reduced to B using first-order reductions. Let us write $\overline{\text{FD}}_{\text{directed}}(P) = \{A \mid A \leq_{\text{fo}} B \in \text{FD}_{\text{directed}}(P)\}$ for the reduction closure of $\text{FD}_{\text{directed}}(P)$ and define $\overline{\text{FD}}_{\text{undirected}}(P)$ and $\overline{\text{FD}}_{\text{basic}}(P)$ similarly.

► **Theorem 1.1 (Main Result).** *The following table completely classifies all prefix classes of ESO logic over basic graphs (upper part) and undirected and directed graphs (lower part):²*

<i>If P is at least one of ...</i>	<i>and at most one of ..., then</i>	
–	$(ae)^*, E^*e^*a, E_1ae$	$\overline{\text{FD}}_{\text{basic}}(P) = \text{FO}$
E_1E_1ae, E_2ae	E^*ae	$\overline{\text{FD}}_{\text{basic}}(P) = \text{L}$
E_1aa	Eaa	$\overline{\text{FD}}_{\text{basic}}(P) = \text{L}$
E_1eaa	E_1e^*aa	$\overline{\text{FD}}_{\text{basic}}(P) = \text{NL}$
$E_1aaa, E_1E_1aa, E_2eaa, E_1eae,$ E_1aee, E_1aea, E_1aae	$E^*(ae)^*$	$\overline{\text{FD}}_{\text{basic}}(P) = \text{NP}$
–	$(ae)^*, E^*e^*a$	$\overline{\text{FD}}_{\text{undirected}}(P) = \overline{\text{FD}}_{\text{directed}}(P) = \text{FO}$
E_1aa	E_1e^*aa, Eaa	$\overline{\text{FD}}_{\text{undirected}}(P) = \overline{\text{FD}}_{\text{directed}}(P) = \text{NL}$
$E_1aaa, E_1E_1aa, E_2eaa, E_1ae$	$E^*(ae)^*$	$\overline{\text{FD}}_{\text{undirected}}(P) = \overline{\text{FD}}_{\text{directed}}(P) = \text{NP}$

Note that we always have $\overline{\text{FD}}_{\text{undirected}}(P) = \overline{\text{FD}}_{\text{directed}}(P)$, which is not trivial, especially for the prefix E_1aa : On undirected graphs, using only two universally quantified variables, it seems difficult to express “non-symmetric” properties, suggesting $\text{FD}_{\text{undirected}}(E_1aa) \subseteq \text{L}$. However, using a gadget construction, we will show that $\text{FD}_{\text{undirected}}(E_1aa)$ contains an NL-complete problem.

As an application of the theorem, let us use it to prove $\text{EVEN-CYCLE} \in \text{L}$, which is the problem of detecting the presence of a cycle³ of even length in basic graphs B . The complexity of this problem has been researched for a long time, see [12] for a discussion and variants. The idea is to consider the following ESO formulas:

$$\phi_m = \exists C_1 \cdots \exists C_m \forall x \exists y \left(E(x, y) \wedge \bigvee_{i=1}^m (C_i(x) \wedge C_{(i \bmod m)+1}(y) \wedge \bigwedge_{j \neq i} \neg C_j(x)) \right). \quad (1)$$

They “describe” the following situation: The basic graph can be colored with m different colors so that each vertex x is connected to a “next” vertex y with the “next” color (with color C_1 following C_m). For $m > 2$, it is not hard to see that $B \models \phi_m$ if, and only if, every connected component of B contains a cycle whose length is a multiple of m . Since ϕ_m has quantifier prefix E^*ae and the graphs are basic, the second row concerning basic

¹ As a technicality, since we use first-order reductions with access to the bit predicate, by FO we refer to “first-order logic with access to the bit predicate,” which is the same as logarithmic-time-uniform AC^0 .

² The “interesting” prefixes, where the complexity classes differ between the two parts, are highlighted.

³ A cycle in an undirected graph must, of course, have length at least 3 and consist of distinct vertices.

graphs in Theorem 1.1 tells us that $B \models \phi_m$ can be decided in logarithmic space. The following algorithm now shows $\text{EVEN-CYCLE} \in \text{L}$: In a basic input graph B , replace all edges by length-2 paths, then test whether $C \models \phi_4$ holds for some connected component C of B .

1.2 Technical Contributions

The proofs of the statements $\text{FD}_{\text{basic}}(E^*ae) \subseteq \text{L}$ and $\text{FD}_{\text{basic}}(E_1ae) \subseteq \text{FO}$ require a sophisticated technical machinery. In both cases, our proofs follow the ideas of a 35-page proof of $\text{FD}_{\text{basic}}(E^*ae) \subseteq \text{P}$ in [11]. The central observation concerning the first statement is that the *algorithmically* most challenging part in the proof of [11] is the application of Courcelle's Theorem [5] to graphs of bounded tree width. It has been shown in [8] that there is a logspace version of Courcelle's Theorem, which will allow us to lower the complexity from P to L when the input graphs have bounded tree width. For graphs of unbounded tree width, we will explain how the other polynomial time procedures from the proof of [11] can be reimplemented in logarithmic space.

To prove $\text{FD}_{\text{basic}}(E_1ae) \subseteq \text{FO}$, we need to lower the complexity of the involved algorithms further. The idea is to again follow the ideas from [11] for E^*ae . When there is just a single monadic predicate, certain algorithmic aspects of the proof can be simplified so severely that they can actually be expressed in first-order logic. Note, however, that already a second monadic predicate or a single binary predicate makes the complexity jump up to L, that is, $\overline{\text{FD}}_{\text{basic}}(E_1E_1ae) = \overline{\text{FD}}_{\text{basic}}(E_2ae) = \text{L}$.

Concerning the remaining claims from Theorem 1.1 that are not already proved in [11], two cases are noteworthy: Proving that $\text{FD}_{\text{basic}}(E_1eaa)$ contains an NL-complete problem turns out to require a nontrivial gadget construction. Proving $\text{FD}_{\text{basic}}(E_1aa) \subseteq \text{L}$ requires a reformulation of the problems in $\text{FD}_{\text{basic}}(E_1aa)$ as special constraint satisfaction problems and showing that these lie in L.

1.3 Related Work

The study of the expressive power of syntactic fragments of logics dates back decades; the decidability of prefix classes of first-order logic, for instance, has been solved completely in a long sequence of papers, see [2] for an overview. Interestingly, the first-order Ackermann prefix class ae plays a key role in that context and both E_1ae and E^*ae turn out to be the most complicated cases in the context of the present paper, too. The expressive power of monadic second-order logic (MSO logic) has also received a lot of attention, for instance in [3, 5, 7], but emphasis has been on restricted structures rather than on syntactic fragments.

Concerning syntactic fragments of ESO logic, the two papers most closely related to the present paper are [6] by Eiter, Gottlob, and Gurevich and [11] by Gottlob, Kolaitis, and Schwentick. In the first paper, a similar kind of classification is presented as in the present paper, only over *strings* rather than *graphs*. It is shown there that for all prefix patterns P the class $\text{FD}_{\text{strings}}(P)$ is either equal to NP; is not equal to NP but contains an NP-complete problem; is equal to REG; or is a subclass of FO. Interestingly, two classes of special interest are $\text{FD}_{\text{strings}}(E_1^*ae)$ and $\text{FD}_{\text{strings}}(E_1^*aa)$, both of which are the minimal classes equal to the regular languages (by the results of Büchi [3]). In comparison, by the results of the present paper $\overline{\text{FD}}_{\text{basic}}(E_1^*ae) = \overline{\text{FD}}_{\text{basic}}(E_1E_1ae) = \text{L}$, while $\overline{\text{FD}}_{\text{basic}}(E_1ae) = \text{FO}$, and $\overline{\text{FD}}_{\text{basic}}(E_1^*aa) = \overline{\text{FD}}_{\text{basic}}(E_1E_1aa) = \text{NP}$, while $\overline{\text{FD}}_{\text{basic}}(E_1aa) = \text{L}$.

The present paper builds on the paper [11] by Gottlob, Kolaitis, and Schwentick, which contains many of the upper and lower bounds from Theorem 1.1 for the class NP as well as most of the combinatorial and graph-theoretic arguments needed to prove $\text{FD}_{\text{basic}}(E^*ae) \subseteq \text{L}$

and $\text{FD}_{\text{basic}}(E_1ae) \subseteq \text{FO}$. The paper misses, however, the finer classification provided in our Theorem 1.1 and Remark 5.1 of [11] expresses the unclear status of the exact complexity of $\text{FD}_{\text{basic}}(E^*ae)$ at the time of writing, which hinges on a problem called $\text{SATU}(P)$: “Note also that for each P , $\text{SATU}(P)$ is probably not a PTIME-complete set. [...] This is due to the check for bounded treewidth, which is in LOGCFL (cf. Wanke [1994]) but not known to be in NL.” The complexity of the check for bounded tree width was settled only later, namely in a paper by Elberfeld, Jakoby, and the author [8], and shown to lie in L. This does not mean, however, that the proof of [11] immediately yields $\text{FD}_{\text{basic}}(E^*ae) \subseteq \text{L}$ since the application of Courcelle’s Theorem is but one of several subprocedures in the proof and since a generalization of tree width rather than normal tree width is used.

1.4 Organization of This Paper

To prove Theorem 1.1, we need to prove the lower bounds implicit in the first column of the theorem’s table and the upper bounds implicit in the second column. The lower bounds are proved in Section 2 by presenting reductions from complete problems for L, NL, or NP. The upper bounds are proved in Section 3, where we prove, in order, $\text{FD}_{\text{basic}}(Eaa) \subseteq \text{L}$, $\text{FD}_{\text{basic}}(E^*ae) \subseteq \text{L}$, and $\text{FD}_{\text{basic}}(E_1ae) \subseteq \text{FO}$ using arguments drawn from different areas.

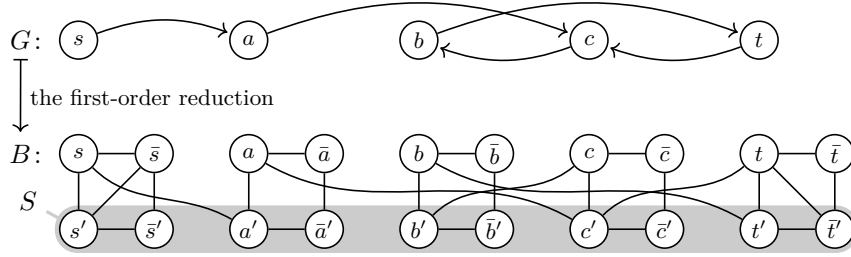
Only the proof *ideas* are given in this conference paper, please see the technical report version for full proofs [16].

2 Lower Bounds: Hardness for L and NL

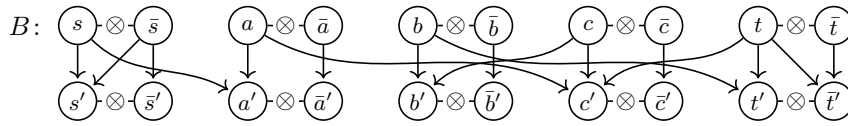
For each of the prefix patterns listed in the first column of the table in Theorem 1.1 we now show that their prefix classes contain problems that are hard for L, NL, or NP. The problems from which we reduce are listed in Table 1. As can be seen, we only need to prove new results for a minority of the classes since the NP cases have already been settled in [11].

■ **Table 1** The lower bounds in Theorem 1.1 are proved by showing that the problems in this table, which are complete for the classes in the claims, are either expressible in the fragment or are at least reducible to a problem expressible in the fragment. The problem UNREACH asks whether there is *no* path from s to t in a directed graph. The problems A_2 and A_3 are explained below.

<i>Claim</i>	<i>Hard problem</i>	<i>Proved where</i>
<i>Lower bounds for basic graphs</i>		
$\overline{\text{FD}}_{\text{basic}}(E_1E_1ae) \supseteq \text{L}$	A_3	Lemma 2.1
$\overline{\text{FD}}_{\text{basic}}(E_2ae) \supseteq \text{L}$	A_2	Lemma 2.2
$\overline{\text{FD}}_{\text{basic}}(E_1aa) \supseteq \text{L}$	2-COLORABLE	[11, Remark 3.1]
$\overline{\text{FD}}_{\text{basic}}(E_1eaa) \supseteq \text{NL}$	UNREACH	Lemma 2.3
$\overline{\text{FD}}_{\text{basic}}(E_1aaa) \supseteq \text{NP}$	POSITIVE-ONE-IN-THREE-3SAT	[11, Theorem 2.2]
$\overline{\text{FD}}_{\text{basic}}(E_1E_1aa) \supseteq \text{NP}$	3-COLORABLE	[11, Theorem 2.3]
$\overline{\text{FD}}_{\text{basic}}(E_2eaa) \supseteq \text{NP}$	3-COLORABLE	[11, Theorem 2.4]
$\overline{\text{FD}}_{\text{basic}}(E_1eae) \supseteq \text{NP}$	3SAT	[11, Theorem 2.5]
$\overline{\text{FD}}_{\text{basic}}(E_1aee) \supseteq \text{NP}$	NOT-ALL-EQUAL-3SAT	[11, Theorem 2.6]
$\overline{\text{FD}}_{\text{basic}}(E_1aea) \supseteq \text{NP}$	POSITIVE-ONE-IN-THREE-3SAT	[11, Theorem 2.7]
$\overline{\text{FD}}_{\text{basic}}(E_1aae) \supseteq \text{NP}$	POSITIVE-ONE-IN-THREE-3SAT	[11, Theorem 2.8]
<i>Remaining lower bounds for undirected and, thereby, also for directed graphs</i>		
$\overline{\text{FD}}_{\text{undirected}}(E_1aa) \supseteq \text{NL}$	UNREACH	Lemma 2.3
$\overline{\text{FD}}_{\text{undirected}}(E_1ae) \supseteq \text{NP}$	3SAT	[11, Theorem 2.1]



■ **Figure 1** Example of the reduction from Lemma 2.3. The directed graph G on top is reduced to the basic graph at the bottom. The edges from the “squares” result from the first rule given in the full proof in the full paper, the curved edges result from the second rule, and the two diagonal edges result from the last rule.



■ **Figure 2** Visualization of the requirements concerning which vertices may lie in M imposed by the formula ψ : For edges with label \otimes exactly one end must lie in M and for directed edges, if the tail of the edge lies in M , the head must also lie in M .

The two special languages A_2 and A_3 in the table are defined as follows: For $m \geq 2$ let $A_m = \{G \mid G \text{ is an undirected graph in which each connected component contains a cycle whose length is a multiple of } m\}$. These languages are all hard for L: In [4, page 388, remarks for problem UFA] it is shown that the reachability problem for graphs consisting of just two undirected trees is complete for L. Since L is trivially closed under complement, testing whether there is *no* path from a vertex u to a vertex v in a graph consisting of two trees is also complete for L, which in turn is the same as asking whether u and v lie in different trees. To reduce this question to A_m , attach cycles of length $2m$ to both u and v . Then all (namely both) components of the resulting graph contain a cycle whose length is a multiple of m if, and only if, u and v lie in different components. (Using a cycle length of $2m$ rather than m ensures that also for $m = 2$ we attach a proper cycle.)

► **Lemma 2.1.** $A_3 \in \text{FD}_{\text{basic}}(E_1 E_1 a e)$.

Proof idea. Use ϕ_3 from equation (1), but get rid of one of the second-order quantifiers. ◀

► **Lemma 2.2.** $A_2 \in \text{FD}_{\text{basic}}(E_2 a e)$.

Proof idea. Use $\exists F \forall x \exists y (E(x, y) \wedge F(x, y) \wedge \neg F(y, x) \wedge (F(x, x) \leftrightarrow \neg F(y, y)))$. ◀

► **Lemma 2.3.** UNREACH reduces to a problem in $\text{FD}_{\text{basic}}(E_1 e a a)$ and also to a problem in $\text{FD}_{\text{undirected}}(E_1 a a)$.

Proof idea. Undirected graphs are essentially the same as basic graphs with an extra monadic relation S^1 that is part of the input. Similarly, a single existential first-order quantifier such as the one in $E_1 e a a$ allows us to pick a vertex and then single out the set of vertices connected to it. Thus, essentially, it suffices to show that UNREACH reduces to $\text{MODELS}_{\text{basic}}(\exists M \forall x \forall y \psi)$ where ψ is a formula over the vocabulary (E^2, S^1) .

The reduction works as shown in Figure 1: Each vertex of the original directed graph gets replaced by four vertices that are connected in a square. Two of them are in the set S ,

■ **Table 2** The upper bounds from Theorem 1.1 and where they are proved. Missing upper bounds for basic and undirected graphs follow from the bounds for directed graphs on the right.

<i>Claims for basic graphs</i>		<i>Proved where</i>	<i>Claims for directed graphs</i>		<i>Proved where</i>
$\text{FD}_{\text{basic}}(E_1ae)$	$\subseteq \text{FO}$	Section 3.3	$\text{FD}_{\text{directed}}((ae)^*)$	$\subseteq \text{FO}$	trivial
$\text{FD}_{\text{basic}}(E^*ae)$	$\subseteq \text{L}$	Section 3.2	$\text{FD}_{\text{directed}}(E^*e^*a)$	$\subseteq \text{FO}$	[11, Theorem 3.1]
$\text{FD}_{\text{basic}}(Eaa)$	$\subseteq \text{L}$	Section 3.1	$\text{FD}_{\text{directed}}(E_1e^*aa)$	$\subseteq \text{NL}$	[11, Theorem 3.2]
			$\text{FD}_{\text{directed}}(Eaa)$	$\subseteq \text{NL}$	[11, Theorem 3.4]
			$\text{FD}_{\text{directed}}(E^*(ae)^*)$	$\subseteq \text{NP}$	Fagin's Theorem

the others are not. Directed edges in the original graph get replaced by undirected edges between one of the four vertices of the tail vertex and one of the four vertices of the head vertex. Additionally, there are edges inside the square of the source and of the target.

The formula ψ expresses that edges inside S and edges outside S correspond to an exclusive or with respect to membership in M , edges between vertices in S and outside S correspond to an implication: If the vertex outside S is in M , so must the vertex inside S . Figure 2 visualizes this situation. One then shows the following: There is some M that makes ϕ true if, and only if, there can be *no* path from s to t in G since we must have $s \in M$, $t \notin M$, and together with s the set M must contain all vertices reachable from s . ◀

3 Upper Bounds: Containment in FO and L

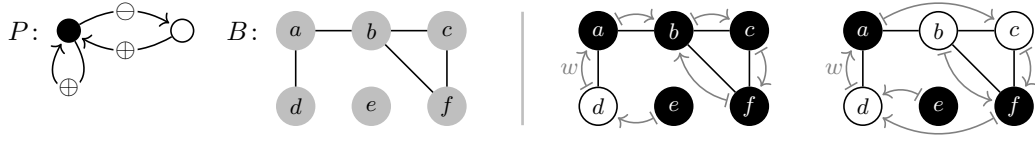
The second column of the table in Theorem 1.1 lists upper bounds that we address in the present section. Table 2 shows the order in which we tackle them.

3.1 Eaa Over Basic Graphs: Reformulation as Constraint Satisfaction

Our first upper bound, $\text{FD}_{\text{basic}}(Eaa) \subseteq \text{L}$, is proved in two steps: First, we reformulate the problems in $\text{FD}_{\text{basic}}(Eaa)$ as special constraint satisfaction problems (CSPs) in Lemma 3.1. Second, we show that these CSPs lie in L in Lemma 3.2.

It will not be necessary to formally introduce the whole theory of constraint satisfaction problems since we will only encounter one very specialized form. Furthermore, our CSPs do not quite fit into the standard framework and major results on CSPs like Schaefer's Theorem [15] or the refined version thereof [1] do not settle the complexity of these special CSPs. Nevertheless, we will need some basic terminology: In a binary CSP, we are given a universe U and a set of *constraints*, each of which picks a number of elements from U and specifies one or more possibilities concerning which of these elements may lie in a *solution* $X \subseteq U$. A *constraint language* specifies the types of constraints that we are allowed to use. For instance the constraint language for 3SAT specifies that constraints (which are clauses) must rule out one of the eight possibilities concerning which of the elements (which are the variables) are in X (are set to *true*). We need to deviate from this framework in one important way: we require that there is a constraint for *every* pair of distinct elements of U , not just for some of them. Unfortunately, this deviation inhibits our applying the classification of the complexity of CSPs from [1]; more precisely, the smallest standard CSP classes that are able to express the special CSPs we are interested in are known to contain NL-complete languages – while we wish to prove containment in L.

For sets $C, D \subseteq \{0, 1, 2\}$ we define a $\{C, D\}$ -*constraint satisfaction problem* P on a universe U to be a mapping that maps each size-2 subset $\{x, y\} \subseteq U$ to either C or D . A *solution* for P is a subset $X \subseteq U$ such that for all size-2 subsets $\{x, y\} \subseteq U$ we have



■ **Figure 3** Example of a pattern graph $P = (C, A^\oplus, A^\ominus)$ with two “colors” black and white (so $C = \{\text{black}, \text{white}\}$, $A^\oplus = \{(\text{black}, \text{black}), (\text{white}, \text{black})\}$, and $A^\ominus = \{(\text{black}, \text{white})\}$) and an uncolored (“gray”) example graph B . We have $B \in \text{SATURATION}(P)$ as shown by two examples of legal colorings of B together with witness functions w (in gray).

$\{|x, y\} \cap X\} \in P(\{x, y\})$. In other words, P fixes for every pair of two vertices x or y one of two possible constraints concerning *how many* elements of $\{x, y\}$ may lie in X . Let $\text{CSP}\{C, D\} = \{P \mid P \text{ is a } \{C, D\}\text{-CSP that has a solution}\}$. As an example, $\text{CSP}\{\{1\}, \{0, 1, 2\}\}$ is essentially the same as the problem $2\text{-COLORABLE} = \text{BIPARTITE}$ since a $\{1\}$ -constraint enforces that exactly one of two vertices must lie in X (and, hence, corresponds to an edge), while a $\{0, 1, 2\}$ -constraint has no effect (and, hence, corresponds to no edge being present). In Lemma 3.2 we show that all $\text{CSP}\{C, D\}$ lie in L , which is fortunate since we reduce to them:

► **Lemma 3.1.** *For every Eaa-formula ϕ there are sets $C, D \subseteq \{0, 1, 2\}$ such that the set $\text{MODELS}_{\text{basic}}(\phi)$ reduces to $\text{CSP}\{C, D\}$.*

Proof idea. By [11, Lemma 3.3] we may assume that ϕ has the form $\exists M \forall x \forall y \psi$ with a *monadic* quantifier M . Rewrite ψ as $x \neq y \rightarrow ((E(x, y) \rightarrow \gamma) \wedge (\neg E(x, y) \rightarrow \delta))$ where γ and δ only contain $M(x)$ and $M(y)$ as atomic formulas. Since the graphs are basic and x and y are interchangeable, γ and δ can only make claims concerning $|\{x, y\} \cap M|$. Use C to encode the claim made by γ and D to encode δ . ◀

► **Lemma 3.2.** *Let $C, D \subseteq \{0, 1, 2\}$. Then $\text{CSP}\{C, D\} \in L$.*

Proof idea. Argue for each choice of C and D how we can check in logarithmic space whether a $\{C, D\}$ -CSP P has a solution $X \subseteq U$. Most cases are quite trivial; the only interesting ones are $\text{CSP}\{\{1\}, \{0, 1, 2\}\}$, which we already saw to be essentially the same as $2\text{-COLORABLE} = \text{BIPARTITE} \in L$, and $\text{CSP}\{\{0, 1\}, \{1, 2\}\}$, which is essentially the same as SPLIT-GRAPH and hence lies even in FO by a characterization of [10]. ◀

3.2 E^*ae Over Basic Graphs: From P to L

Our objective is to show $\text{FD}_{\text{basic}}(E^*ae) \subseteq L$ in this section. More precisely, we only need to show $\text{FD}_{\text{basic}}(E_1^*ae) \subseteq L$ since [11, Theorem 4.1] states $\text{FD}_{\text{basic}}(E^*ae) = \text{FD}_{\text{basic}}(E_1^*ae)$.

A proof of the weaker claim $\text{FD}_{\text{basic}}(E_1^*ae) \subseteq P$ is spread over the 35 pages of Sections 4, 5, and 6 of the paper [11] and consists of two kinds of arguments: graph-theoretic ones and algorithmic ones. Since the graph-theoretic arguments are independent of complexity-theoretic considerations, our main job is to show how the algorithms described by Gottlob et al. can be implemented in logarithmic space rather than polynomial time.

Similarly to the switch from model checking problems to graphs problems in the previous section, we also wish to reformulate the model checking problems $\text{MODELS}_{\text{basic}}(\phi)$ for E_1^*ae -formulas ϕ in a graph-theoretic manner. Gottlob et al. introduce the notion of *pattern graphs* for this: A *pattern graph* $P = (C, A^\oplus, A^\ominus)$ consists of a set of *colors* C , a set $A^\oplus \subseteq C \times C$ of \oplus -arcs, and a set $A^\ominus \subseteq C \times C$ of \ominus -arcs (A^\oplus and A^\ominus need not be disjoint). Given a basic graph $B = (V, E)$, a *coloring of G with respect to P* is a function $c: V \rightarrow C$. A mapping $w: V \rightarrow V$ is called a *witness function for a coloring c* if for all $x \in V$ we have

(1) $x \neq w(x)$, (2) if $\{x, w(x)\} \in E$, then $(c(x), c(w(x))) \in A^\oplus$, and (3) if $\{x, w(x)\} \notin E$, then $(c(x), c(w(x))) \in A^\ominus$.⁴ If there exists a coloring together with a witness function for B with respect to P , we say that B can be saturated by P and the saturation problem $\text{SATURATION}(P)$ is the set of all basic graphs that can be saturated by P , see Figure 3 for an example.

The intuition behind these definitions is that a witness function tells us for each x in $\forall x$ which y in $\exists y$ we must pick to make a formula ϕ of the form $\exists M_1 \cdots \exists M_n \forall x \exists y \psi$ true. The pattern graph encodes the restrictions imposed by ψ and the monadic predicates M_i :

► **Fact 3.3** ([11, Theorem 4.6]). For every formula $\phi = \exists M_1 \cdots \exists M_n \forall x \exists y \psi$, where the M_i are monadic and ψ is quantifier-free, there is a pattern graph P with 2^n vertices such that $\text{MODELS}_{\text{basic}}(\phi) = \text{SATURATION}(P)$.

Thus, it remains to show $\text{SATURATION}(P) \in \text{L}$ for all pattern graphs P . Towards this aim, for a fixed pattern graph P we devise logspace algorithms that work for larger and larger classes of basic graphs B , ending with the class of all basic graphs.

Graphs of Bounded Tree Width and Special Graphs We start by considering only graphs of *bounded tree width*, an important class of graphs introduced by Robertson and Seymour in [14]: A *tree decomposition* of a graph B is a tree T together with a mapping that assigns subsets of B 's vertices (called *bags*) to the nodes of T . The bags must have two properties: First, for every edge $\{x, y\}$ of B there must be some bag that contains both x and y . Second, the nodes of T whose bags contain a given vertex x must be connected in T . The *width* of a decomposition is the size of its largest bag (minus 1 for technical reasons). The *tree width* of B is the minimal width of any tree decomposition for it. A class of graphs has *bounded tree width* if there is a constant c such that all graphs in the class have tree width at most c . From an algorithmic point of view, many problems that can be solved efficiently on trees can also be solved efficiently on graphs of bounded tree width. Courcelle's Theorem turns this into a precise statement:

► **Fact 3.4** (Courcelle's Theorem, [5]). For every MSO-formula ϕ and $t \geq 1$ we have

$$\text{MODELS}_{\text{basic}}(\phi) \cap \{G \mid G \text{ has tree width at most } t\} \in \text{LINTIME}.$$

Gottlob et al. apply this theorem to show that when the input graphs B have bounded tree width, we can decide whether $B \in \text{SATURATION}(P)$ holds in polynomial time: the property $B \in \text{SATURATION}(P)$ is easily described in MSO logic. We can lower the complexity from “polynomial time” to “logarithmic space” by using the following logarithmic space version of Courcelle's Theorem:

► **Fact 3.5** (Logspace Version of Fact 3.4, [8]). For every MSO-formula ϕ and $t \geq 1$ we have

$$\text{MODELS}_{\text{basic}}(\phi) \cap \{G \mid G \text{ has tree width at most } t\} \in \text{L}.$$

In their graph-theoretic arguments, Gottlob et al. encounter not only graphs of bounded tree width, but also graphs that they call (k, t) -*special* and which are defined as follows: For a basic graph $B = (V, E)$ let us call two vertices u and v *equivalent* if for all $x \in V \setminus \{u, v\}$ we have $\{u, x\} \in E$ if, and only if, $\{v, x\} \in E$. Observe that this defines an easy-to-check

⁴ Using $\{u, v\}$ to indicate an undirected edge between u and v in a basic graph and, in not-so-slight abuse of notation, even writing $\{u, v\} \in E$, helps in distinguishing these edges from the directed edges in the pattern graph. Formally, we mean of course $(u, v) \in E$ and $(v, u) \in E$; and $E \subseteq V \times V$ holds.

equivalence relation on the vertices of B and that each equivalence class is either a clique or an independent set of B . A graph is (k, t) -special if we can remove (up to) k equivalence classes A_1, \dots, A_k from the graph such that the remaining graph has tree width at most t .

The intuition behind (k, t) -special graphs is that equivalent vertices are “more or less indistinguishable” and, thus, for a large enough equivalence class removing some vertices does not change whether the graph can be saturated or not. Formally, let B be (k, t) -special and let A_1, \dots, A_k be to-be-removed equivalence classes. We obtain an s -shrink of B by repeatedly removing vertices from those A_i that have more than s vertices until all of them have at most s vertices. The proof of Lemma 6.4 in [11] implies the following two facts:

► **Fact 3.6.** For every k, t , and pattern graph P there is an s such for every s -shrink B' of a (k, t) -special graph B we have $B \in \text{SATURATION}(P)$ if, and only if, $B' \in \text{SATURATION}(P)$.

► **Fact 3.7.** An s -shrink of a (k, t) -special graph has tree width at most $t + sk$.

In Lemmas 6.3 and 6.4 of [11], Gottlob et al. present polynomial-time algorithms for testing whether a graph is (k, t) -special and for computing an s -shrink when the test is positive. The following lemma shows that we can reimplement these algorithms in a space-efficient manner (which the original algorithms are not):

► **Lemma 3.8.** For every s, k , and t , there is a logspace computable function that maps every (k, t) -special graph B to an s -shrink of B (and all other graphs to “not (k, t) -special”).

Proof idea. Find a tuple (v_1, \dots, v_k) of vertices such that removing all vertices equivalent to some v_i leaves behind a graph of tree width at most t . Then for each v_i leave only the lexicographically first s vertices equivalent to v_i in the graph. ◀

The following lemma sums up the bottom line of the above discussion:

► **Lemma 3.9.** For every pattern graph P and all k and t we have

$$\text{SATURATION}(P) \cap \{B \mid B \text{ is } (k, t)\text{-special}\} \in \text{L}.$$

Proof idea. To decide $\text{SATURATION}(P)$ on (k, t) -special graphs B , compute a shrink B' , which has bounded tree width, and apply the logspace version of Courcelle’s Theorem. ◀

Graphs With Self-Saturating Mixed Cycles We extend the class of graphs that our logspace machines can handle to graphs that are not necessarily (k, t) -special, but at least contain a *mixed self-saturating cycle*. A *self-saturating cycle* of a basic graph $B = (V, E)$ with respect to a pattern graph $P = (C, A^\oplus, A^\ominus)$ is a sequence $(v_1, v_2, \dots, v_{n+1})$ of vertices in V for $n \geq 2$ where the v_i for $i \in \{1, \dots, n\}$ are all different, $v_{n+1} = v_1$, and we can assign colors $c: \{v_1, \dots, v_n\} \rightarrow C$ such that for all $i \in \{1, \dots, n\}$ we have: if $\{v_i, v_{i+1}\} \in E$, then $(c(v_i), c(v_{i+1})) \in A^\oplus$; and if $\{v_i, v_{i+1}\} \notin E$, then $(c(v_i), c(v_{i+1})) \in A^\ominus$. In other words, B restricted to $\{v_1, \dots, v_n\}$ can be saturated with the “natural” witness function that “moves along” the cycle. The following is an easy observation concerning self-saturating cycles:

► **Lemma 3.10.** For every $B \in \text{SATURATION}(P)$ there is a self-saturating cycle in B for P .

Proof idea. Just “follow the witness function” until it runs into a cycle. ◀

A self-saturating cycle is *mixed* if for some $i, j \in \{1, \dots, n\}$ we have $\{v_i, v_{i+1}\} \in E$ and $\{v_j, v_{j+1}\} \notin E$, otherwise the cycle is called *pure*. In Figure 3, (b, c, f, b) is a pure self-saturating cycle and (a, c, f, d, a) is a mixed self-saturating cycle as proved by the two example colorings. Two facts concerning mixed self-saturating cycles will be important:

► **Fact 3.11** ([11, Lemma 6.5]). For every pattern graph P there is a constant d such that every basic graph that has a mixed self-saturating cycle with respect to P also has such a cycle of length at most d .

► **Fact 3.12** ([11, Section 6.3]). For each pattern graph P there exist k and t such that $B \in \text{SATURATION}(P)$ holds for all graphs B that contain a mixed self-saturating cycle but are not (k, t) -special.

► **Lemma 3.13.** *For every pattern graph P , we have*

$$\text{SATURATION}(P) \cap \{B \mid B \text{ contains a mixed self-saturating cycle}\} \in \text{L}.$$

Proof idea. Fact 3.11 gives a logspace procedure for detecting mixed self-saturating cycles. Combine it with Fact 3.12 and Lemma 3.9. ◀

Arbitrary Basic Graphs The last step is to extend our algorithm to graphs that do not contain mixed self-saturating cycles (and are not (k, t) -special, but this will no longer be important). Clearly, by considering the union of the languages from Lemma 3.13 above and Lemma 3.14 below, we see that $\text{SATURATION}(P) \in \text{L}$ holds for all pattern graphs P .

► **Lemma 3.14.** *For every pattern graph P , we have*

$$\text{SATURATION}(P) \cap \{B \mid B \text{ contains no mixed self-saturating cycle}\} \in \text{L}.$$

Proof idea. Theorem 5.17 of [11] provides a polynomial-time algorithm for deciding $B \in \text{SATURATION}(P)$ when there are no mixed self-saturating cycles in B . The algorithmically relevant operations in the proof are (1) computing complement graphs (exchanging edges and non-edges), (2) computing connected components, and (3) applying Courcelle’s Theorem to these components. Clearly, all three operations can also be implemented in logarithmic space using Reingold’s Theorem and the logspace version of Courcelle’s Theorem. ◀

3.3 E_{1ae} Over Basic Graphs: From L to FO

Our final task for this paper is showing $\text{FD}_{\text{basic}}(E_{1ae}) \subseteq \text{FO}$.⁵ By Fact 3.3, it suffices to show $\text{SATURATION}(P) \in \text{FO}$ for all pattern graphs with *two* colors (denoted “white” and “black” in the following) and this will be our objective in this section.⁶

In the previous section we proved $\text{SATURATION}(P) \in \text{L}$ for all pattern graphs by developing logspace algorithms that worked for larger and larger classes of graphs. However, this approach is bound to fail for the class FO since properties like “the graph is a tree” (let alone “the graph is (k, t) -special”) are not expressible in first-order logic. Instead, in this section we show $\text{SATURATION}(P) \in \text{FO}$ directly for each possible pattern graph with two colors.

The simplest case arises when $P = (C, A^\oplus, A^\ominus)$ is acyclic (meaning that the directed graph $(C, A^\oplus \cup A^\ominus)$ is acyclic): Lemma 3.10 shows that we then have $\text{SATURATION}(P) = \emptyset$ since self-saturating cycles cannot exist for such P . Thus, we only need to consider pattern graphs P with cycles (self-loops are also cycles, here). Since P only has two colors, there are only few ways in which such cycles may arise. The more cycles there are, the easier it will be to color the graph, so we first handle the case that there are cycles both in A^\oplus and A^\ominus , then that there is a cycle in A^\oplus or in A^\ominus , and finally that there is only a cycle in $A^\oplus \cup A^\ominus$.

⁵ In contrast, Lemmas 2.1 and 2.2 show that if we have *two* monadic quantifiers or one *binary* quantifier, the prefix class contains an L-complete problem.



⁶ In contrast, using three colors we can describe L-complete problems: $\text{SATURATION}(P) = A_3$ where P contains a \oplus -labeled 3-cycle and A_3 is the L-complete language from Table 1.

► **Lemma 3.15.** *Let $P = (\{\text{black}, \text{white}\}, A^\oplus, A^\ominus)$ contain cycles both in A^\oplus and A^\ominus . Then $\text{SATURATION}(P)$ contains all graphs with at least two vertices (and is hence in FO).*

Proof idea. The interesting case is a \oplus -cycle involving both colors. In each connected component, choose a vertex and color the vertices black or white depending on whether they have odd or even distance from the chosen vertex. The witness of a vertex is a vertex nearer to the chosen vertex (except for the chosen vertex, whose witness is any neighbor). ◀

► **Lemma 3.16.** *Let $P = (\{\text{black}, \text{white}\}, A^\oplus, A^\ominus)$ contain a cycle in A^\oplus or in A^\ominus . Then $\text{SATURATION}(P) \in \text{FO}$.*

Proof idea. The most interesting case is exactly the pattern graph shown in Figure 3. We distinguish the cases that the input graph B consists of a matching plus some isolated vertices or has a connected component of size at least three. If the matching is a single edge, $B \notin \text{SATURATION}(P)$; otherwise, $B \in \text{SATURATION}(P)$ since one can devise similar methods as in the previous lemma for coloring the graph and constructing a witness function. ◀

We are left with the case that the set $A^\oplus \cup A^\ominus$ contains a cycle, but neither A^\oplus nor A^\ominus does. This is only possible when P is either  or . For this special kind of cycle, there is an analogue of Fact 3.12 that does not refer to (k, t) -special graphs:

► **Fact 3.17** ([11, Lemma 6.7]). For every pattern graph P , we have $B \in \text{SATURATION}(P)$ for all B that contain a self-saturating cycle for P on which \oplus - and \ominus -arcs alternate.

► **Lemma 3.18.** *Let $P = (\{\text{black}, \text{white}\}, A^\oplus, A^\ominus)$ contain a cycle in $A^\oplus \cup A^\ominus$, but none in A^\oplus nor in A^\ominus . Then $\text{SATURATION}(P) \in \text{FO}$.*

Proof idea. Use Fact 3.11 to detect a mixed self-saturating cycle using d existential first-order quantifiers. The existence of such a cycle in B is a necessary condition for $B \in \text{SATURATION}(P)$ by Lemma 3.10 and also a sufficient condition by Fact 3.17. ◀

4 Conclusion

In the present paper we have completely classified the first-order reduction closures of prefix classes of ESO logic over directed, undirected, and basic graphs: each one of them is equal to one of the standard classes FO, L, NL, or NP. It turned out that the prefix classes for directed and undirected graphs are always the same, but often differ from the prefix classes for basic graphs. Especially interesting prefixes that mark the border between one complexity class and the next are E_1ae , E^*ae , and Eaa .

A natural question that arises is: Can we find a prefix class whose reduction closure is P? By the results of the present paper, this cannot be an ESO prefix class, unless unlikely collapses occur. However, what about prefix classes of general second-order logic? We may similarly ask whether any class other than L, NL, and the classes of the polynomial hierarchy can be characterized by a prefix class of second-order logic.

Together with the results from [6], we now have a fairly complete picture of the complexity of all ESO prefix classes over directed graphs, undirected graphs, basic graphs, and strings. Concerning arbitrary logical structures, Gottlob et al. [11] already point out that their P-NP-dichotomy for directed graphs generalizes to the collection of all finite structures over any relational vocabulary that contains a relation symbol of arity at least two; and it is not hard to see that our Theorem 1.1 also generalizes in this way (a closer look at the FO and NL upper bounds in [11] shows that they hold for arbitrary structures). The complexity of prefix classes over other special structures is, however, still open, including those of trees, infinite words, and bipartite graphs.

References

- 1 Eric Allender, Michael Bauland, Neil Immerman, Henning Schnoor, and Heribert Vollmer. The complexity of satisfiability problems: Refining Schaefer's theorem. *Journal of Computer and System Sciences*, 75(4):245–254, 2009.
- 2 Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Springer-Verlag, Berlin, 1997.
- 3 Julius R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.
- 4 Stephen A. Cook and Pierre McKenzie. Problems complete for deterministic logarithmic space. *Journal of Algorithms*, 8(5):385–394, 1987.
- 5 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.
- 6 Thomas Eiter, Georg Gottlob, and Yuri Gurevich. Existential second order logic over strings. *Journal of the ACM*, 47(1):77–131, 2000.
- 7 Michael Elberfeld, Martin Grohe, and Till Tantau. Where first-order and monadic second-order logic coincide. In *Proceedings of the 27th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2012)*, pages 265–274. IEEE Computer Society, 2012.
- 8 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS 2010)*, pages 143–152, 2010.
- 9 Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. *Complexity of Computation*, 7:43–74, 1974.
- 10 Stéphane Földes and Peter L. Hammer. Split graphs. In *Proceedings of the Eighth South-eastern Conference on Combinatorics, Graph Theory and Computing*, Congressus Numerantium XIX, pages 311–315. Louisiana State University, Baton Rouge, Louisiana, 1977.
- 11 Georg Gottlob, Phokion G. Kolaitis, and Thomas Schwentick. Existential second-order logic over graphs: Charting the tractability frontier. *Journal of the ACM*, 51(2):312–362, 2004.
- 12 Edith Hemaspaandra, Holger Spakowski, and Mayur Thakur. Complexity of cycle length modularity problems in graphs. In *Proceedings of the 6th Latin American Symposium on Theoretical Informatics (LATIN 2004)*, volume 2976 of *Lecture Notes in Computer Science*, pages 509–518. Springer, 2004.
- 13 Neil Immerman. *Descriptive Complexity Theory*. Springer-Verlag, New York, 1998.
- 14 Neil Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986.
- 15 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Symposium on Theory of Computing (STOC 1978)*, pages 216–226. ACM Press, 1978.
- 16 Till Tantau. Existential second-order logic over graphs: A complete complexity-theoretic classification. Technical Report arxiv:1412.6396 [cs.LO], ArXiv e-prints, 2014.

The Returning Secretary*

Shai Vardi

Blavatnik School of Computer Science, Tel Aviv University.
Tel Aviv, Israel
shaivar1@post.tau.ac.il

Abstract

In the online random-arrival model, an algorithm receives a sequence of n requests that arrive in a random order. The algorithm is expected to make an irrevocable decision with regard to each request based only on the observed history. We consider the following natural extension of this model: each request arrives k times, and the arrival order is a random permutation of the kn arrivals; the algorithm is expected to make a decision regarding each request only upon its last arrival. We focus primarily on the case when $k = 2$, which can also be interpreted as each request arriving at, and departing from the system, at a random time.

We examine the secretary problem: the problem of selecting the best secretary when the secretaries are presented online according to a random permutation. We show that when each secretary arrives twice, we can achieve a competitive ratio of $0.767974\dots$ (compared to $1/e$ in the classical secretary problem), and that it is optimal. We also show that without any knowledge about the number of secretaries or their arrival times, we can still hire the best secretary with probability at least $2/3$, in contrast to the impossibility of achieving a constant success probability in the classical setting.

We extend our results to the matroid secretary problem, introduced by Babaioff et al. [3], and show a simple algorithm that achieves a 2-approximation to the maximal weighted basis in the new model (for $k = 2$). We show that this approximation factor can be improved in special cases of the matroid secretary problem; in particular, we give a $16/9$ -competitive algorithm for the returning edge-weighted bipartite matching problem.

1998 ACM Subject Classification F.1.2 Online computation

Keywords and phrases online algorithms, secretary problem, matroid secretary

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.716

1 Introduction

The *secretary problem* [23, 10] is the following: n random items are presented to an observer in random order, with each of the $n!$ permutations being equally likely. There is complete preference order over the items, which the observer is able to query, for the items he¹ has seen so far. As each item is presented, the observer must either accept it, at which point the process ends, or reject it, and then it is lost forever. The goal of the observer is to maximize the probability that he chooses the “best” item (i.e., the one ranked first in the preference order). This problem models many scenarios; one such scenario is the one for which the problem is named: n secretaries arrive one at a time, and an interviewer must make an irrevocable decision whether to accept or reject each secretary upon arrival. Another is the

* Supported in part by the Google Europe Fellowship in Game Theory.

¹ We use male pronouns throughout this paper for simplicity. No assumption on the genders of actual agents is intended.

house-selling problem, in which buyers arrive and bid for the house, and the seller would like to accept the highest offer. An alternative way of modeling this problem is the following. Each secretary is allocated, independently and uniformly at random, a real number $r \in [0, 1]$, which represents his arrival time. As before, the interviewer sees the secretaries in the order of arrival and must make an irrevocable decision before seeing the next secretary. It is easy to see that the two models are essentially equivalent (assuming n is known, see e.g., [7]) - the arrival times define a permutation over the secretaries, with each permutation being equally likely. The optimal solution for the classical secretary problem is well known - wait until approximately n/e secretaries have passed² (alternatively until time $t = 1/e$), and thereafter, accept a secretary if and only if he is the best out of all secretaries observed so far (e.g., [15, 7]). This gives a probability of success of at least $1/e$.

Consider the following generalization of the secretary problem: Assume that each secretary arrives k times, and the interviewer has to make a decision upon each secretary's last arrival. We model this as follows: Allocate each secretary k numbers, independently and uniformly at random from $[0, 1]$, which represent his k arrival times. (Equivalently, we may consider only the order of arrivals; in this case each of the $(kn)!$ permutations over the arrival events is equally likely.) A decision whether to accept or reject a secretary must be made between his first and last arrival. We call this problem the $(k-1)$ -returning secretary problem. The secretary problem is a classical example of the random-arrival online model (e.g., [4, 24]), and our model immediately applies to this more general framework, capturing several natural variations thereof, for example:

1. Requests may not require (or expect) an immediate answer and will therefore visit the system several times to query it.
2. When requests arrive, the system gives them either a rejection or an acceptance, or an invitation to return at some later time. It turns out that in many cases, very few requests actually need to return; in the secretary problem, for example, a straightforward analysis shows that the optimal algorithm will only ask $O(\log n)$ secretaries to return.³
3. Requests may enter the system and leave at some later time. The time the request stays in the system can vary from "until just before the next item arrives", in which case no information is gained, to "until the end", in which case the problem reduces to an offline one. Clearly we would like something in between. When $k = 2$, the second random variable allocated to the query can be interpreted as the time that the query leaves the system, giving a natural formulation of this property in the spirit of the random-arrival online model.

1.1 Our Results

When each secretary returns once (i.e., $k = 2$), we show that the optimal solution has a similar flavor to that of the classical secretary problem - wait until some fraction of the secretaries have passed (ignoring how many times each secretary has arrived), and thereafter hire the best secretary (out of those we have seen so far), upon his second arrival. To tightly bound the probability of success (for large n), we examine the case when each of the $2n$ arrival times is selected uniformly at random from $[0, 1]$. We use this model to show that the success probability tends to $0.767974\dots$ as n grows. In the classical secretary problem,

² The exact number for each n can be computed by dynamic programming, see e.g., [15].

³ The algorithm will only ask the i^{th} secretary that arrives to return if he is the best out of all the secretaries it has seen thus far. The probability of this is $1/i$. Summing over all secretaries gives the bound.

it is essential to know the number of secretaries arriving in order to achieve a constant success probability. We consider the case when n is not known in advance (and there is no extra knowledge, such as arrival time distribution), and show that by choosing the best secretary we have seen once he returns, with no waiting period, we can still obtain a success probability of at least $2/3$. We also consider cases when $k > 2$: we show that for $k = 3$, we can achieve a success probability of at least 0.9 , even without knowledge of n , and show that setting $k = \Theta(\log n)$ guarantees success with arbitrarily high probability ($1 - \frac{1}{n^\alpha}$ for any α).

We extend our results to the *matroid secretary problem*, introduced by Babaioff et al., [3], which is an adaptation of the classical secretary problem to the domain of *weighted matroids*. A weighted matroid is a pair $\mathcal{M} = (E, \mathcal{I})$ of elements E and independent sets \mathcal{I} , and a weight function $w : E \rightarrow \mathbb{R}$, which obeys the properties of heredity and exchange (see Section 2 for a formal definition). In the matroid secretary problem, the elements of a weighted matroid are presented in random order to the online algorithm. The algorithm maintains a set S of selected elements; when an element e arrives, the algorithm must decide whether to add it to S , under the restriction that $S \cup \{e\}$ is an independent set of the matroid. The algorithm's goal is to maximize the sum of the weights of the items in S . It is currently unknown whether there exists an algorithm that can find a set whose expected weight is a constant fraction of the optimal offline solution. The best result to date is an $O(\log \log \rho)$ -competitive algorithm,⁴ where ρ is the rank of the matroid, due to Lachish [22]. We show that in the returning online model, there is an algorithm which is 2-competitive in expectation (independent of the rank). We also show that for bipartite edge-weighted matching, and hence for transversal matroids⁵ in general, this result can be improved, and show a $16/9$ -competitive algorithm.

1.2 Related Work

The origin of the secretary problem is still being debated: the problem first appeared in print in 1960 [13]; its solution is often credited to Lindley [23] or Dynkin [10]. Hundreds of papers have been published on the secretary problem and variations thereof; for a review, see [12]; for a historical discussion, see [11]. Kleinberg [19] introduced a version of the secretary problem in which we are allowed to choose k elements, with the goal of maximizing their sum. He gave a $1 - O(\sqrt{1/k})$ -competitive algorithm, and showed that this setting applies to strategy-proof online auction mechanisms.

The *matroid secretary problem* was introduced by Babaioff et al. [3]. They gave an $O(\log \rho)$ -competitive algorithm for general matroids, where ρ is the rank of the matroid, and several constant-competitive algorithms for special cases of the matroid secretary problem. Lachish [22] gave an $O(\log \log \rho)$ algorithm for the matroid secretary problem. There have been several improvements on special cases since then. Babaioff et al., [2] gave algorithms for the discounted and weighted secretary problems; Korula and Pál [20] showed that *graphic matroids*⁶ admit $2e$ -competitive algorithms; Kesselheim et al. [18] gave a $1/e$ -competitive algorithm for the secretary problem on transversal matroids and showed that this is optimal.

⁴ An online algorithm whose output is within a factor c of the optimal offline output is said to be c -competitive; see Section 2 for a formal definition.

⁵ Transversal matroids (see Section 4 for a definition) are a special case of bipartite edge-weighted matching.

⁶ In a *graphic matroid* $G = (V, E)$, the elements are the edges of the graph G and a set is independent if it does not contain a cycle.

Soto [28] gave a $2e^2/(e-1)$ -competitive algorithm when the adversary can choose the set of weights of the elements, but the weights are assigned at random to the elements, (and the elements are presented in a random order). Gharan and Vondrák [14] showed that once the weights are random, the ordering can be made adversarial, and that this setting still admits $O(1)$ -competitive algorithms. There have been other interesting results in this field; for a recent survey, see [9].

1.3 Comparison with Related Models

There are several other papers which consider online models with arrival and departure times. Due to the surge in interest in algorithmic game theory over the past 15 years, and the economic implications of the topic, it is unsurprising that many of these papers are economically motivated. Hajiaghayi et al. [17], consider the case of an auction in which an auctioneer has k goods to sell and the buyers arrive and depart dynamically. They notice and make use of the connection to the secretary problem to design strategy-proof mechanisms: they design an e -competitive (w.r.t. efficiency) strategy-proof mechanism for the case $k = 1$, which corresponds to the secretary problem, and extend the results to obtain $O(1)$ -competitive mechanisms for $k > 1$. Hajiaghayi et al. [16], design strategy-proof mechanisms for online scheduling in which agents bid for access to a re-usable resource such as processor time or wireless network access, and each agent is assumed to arrive and depart dynamically. Blum et al. [5], consider online auctions, in which a single commodity is bought by multiple buyers and sellers whose bids arrive and expire at different times. They present an $O(\log(p_{max} - p_{min}))$ -competitive algorithm for maximizing profit and an $O(\log(p_{max}/p_{min}))$ -competitive algorithm for maximizing volume where the bids are in the range $[p_{min}, p_{max}]$, and a strategy-proof algorithm for maximizing social welfare. They also show that their algorithms achieve almost optimal competitive ratios. Bredin and Parkes [6] consider online *double auctions*, which are matching problems with incentives, where agents arrive and depart dynamically. They show how to design strategy-proof mechanisms for this setting.

In Section 2 we introduce our model. In Section 3 we provide an optimal algorithm for the returning secretary problem. In the full version of the paper we give an over 2-competitive algorithm for the returning matroid secretary problem; we show that we can improve this competitive ratio to $16/9$ for transversal matroids (and more generally, returning edge-weighted bipartite matching); and we analyze the cases of the k -returning secretary problem for $k = 3$ and $k = \Theta(\log n)$.

2 Model and Preliminaries

Consider the following scenario. There are n items which arrive in an online fashion, and each item arrives k times. Each arrival of an item is called a *round*; there are kn rounds. The order of arrivals is selected uniformly at random from the $(kn)!$ possible permutations. An algorithm observes the items as they arrive, and must make an irrevocable decision about each item upon the item's last appearance. We call such an algorithm a $(k-1)$ -returning online algorithm and the problem it solves a $(k-1)$ -returning online problem. Because the problem is most natural when $k = 2$, for the rest of the paper, we assume that $k = 2$ (and instead of “1-returning”, we simply say “returning”). In the full version of the paper, we consider scenarios when $k > 2$.

We use the following definition of matroids:

► **Definition 2.1.** A matroid $\mathcal{M} = (E, \mathcal{I})$ is an ordered pair, where E is a finite set of elements (called the *ground set*), and \mathcal{I} is a family of subsets of E , (called the *independent sets*), which satisfies the following properties:

1. $\emptyset \in \mathcal{I}$,
2. If $X \in \mathcal{I}$ and $Y \subseteq X$ then $Y \in \mathcal{I}$,
3. If $X, Y \in \mathcal{I}$ and $|Y| < |X|$ then there is an element $e \in X$ such that $Y \cup \{e\} \in \mathcal{I}$.

Property (2) is called the *hereditary property*. Property (3) is called the *exchange property*. An independent set that becomes dependent upon adding any element of E is called a *basis* for the matroid. In a *weighted matroid*, each element $e \in E$ is associated with a weight $w(e)$. The *returning matroid secretary problem* is the following: Each element of a weighted matroid $\mathcal{M} = (E, \mathcal{I})$ arrives twice, in an order selected uniformly at random out of the $(2n)!$ possible permutations of arrivals. The algorithm maintains a set of selected elements, S , and may add any element to S at any time between (and including) the first and second appearances of the element, as long as $S \cup \{e\} \in \mathcal{I}$. The goal of the algorithm is to maximize the sum of the weights of the elements in S . The success of the algorithm is defined by its *competitive ratio*.

► **Definition 2.2** (competitive ratio, c -competitive algorithm). If the weight of a maximal-weight basis of a matroid is at most c times the expected weight of the set selected by an algorithm (where the expectation is over the arrival order), the algorithm is said to be c -competitive, and its *competitive ratio* is said to be c .

A special case of the returning matroid secretary problem is the *returning secretary problem*, in which there are n secretaries, each of whom arrives twice. The goal of the algorithm is to identify the best secretary. The algorithm is *successful* if and only if it chooses the best secretary, and we quantify how “good” the algorithm is by its success probability.

Without loss of generality, we assume throughout this paper that the weights of all the elements are distinct (this applies to secretaries as well - given any two secretaries, one must be strictly better than the other).⁷ Although we do not discuss computational efficiency in this work, all the algorithms in this paper are polynomial in the succinct representation of the matroid.

We denote the set $\{1, 2, \dots, n\}$ by $[n]$.

3 The Returning Secretary

Assume that there are n secretaries that arrive in an online fashion. Each secretary arrives twice, and the order is selected uniformly at random from the $(2n)!$ possible orders. At all times, we keep note of who the best secretary is out of all the secretaries seen so far. We call this secretary the *candidate*. That is, in each round, if the secretary that arrived is better than all other secretaries that arrived before this round, he becomes the candidate. Note that it is possible that in a given round, the candidate will have already arrived twice. At any point between immediately after first arrival and immediately after the second arrival, we can *accept* or *reject* a secretary; an acceptance is final, a rejection is only final if made upon the second arrival. Once we accept a secretary, the process ends. We *win* if we accept (or *choose*) the best secretary. We would like to maximize the probability of winning.

⁷ Babaioff et al., [3] show that we do not lose generality by this assumption in the matroid secretary problem. The result immediately applies to our model.

3.1 Optimal Family of Rules

What is the best strategy for maximizing the probability of winning? We first show that the optimal rule must be taken from the family of stopping rules as described in the following lemma.

► **Lemma 3.1.** *The optimal strategy for choosing the best secretary in the returning secretary problem has the following structure: wait until d distinct secretaries have arrived; thereafter, accept the best secretary out of the secretaries seen so far, when he returns.*

Proof. Without loss of generality, we can restrict our attention to strategies that make decisions regarding a secretary s only upon s 's arrivals, as every strategy that makes decisions between the two arrival times has an equivalent strategy that defers the decision making to the second arrival. Let d_r be the random variable denoting the number of distinct secretaries that have arrived up to (and including) round r ($r \in [2n]$). Denote by $H(r) = \{x_1, x_2, \dots, x_{r-1}\}$ the history at round r , where $x_i = (y_i, z_i)$: y_i is the relative rank (among the secretaries that have arrived until now) of the secretary that arrived at time i , and z_i represents whether this is the first or second time that this secretary has arrived (i.e. $y_i \in [d_r]$, $z_i \in \{1, 2\}$). Any (deterministic) strategy \mathcal{S} must have the following structure: for every realization of $x_r = (y_r, z_r)$, and $H(r)$, \mathcal{S} must accept or reject. That is $S : (H_r, y_r, z_r) \rightarrow \{\text{accept}, \text{reject}\}$. Denote the optimal strategy by \mathcal{S}^* . Clearly,

1. If the t^{th} secretary is not the best, we will not choose him: $\forall y_r \neq 1, \mathcal{S}^*(H_r, y_r, z_r) = \text{reject}$.
2. If this is the first time we have seen a secretary, we cannot gain anything by choosing him now. It is better to wait for the second arrival, as we lose nothing by waiting: $\mathcal{S}^*(H_r, y_r, 1) = \text{reject}$.

Therefore, we only need to consider choosing the best secretary we have seen so far when he returns; i.e., we only accept at time t such that $y_r = 1, z_r = 2$. For all other values of y_i and z_i , \mathcal{S}^* must reject; henceforth, we only focus on the case that $y_r = 1, z_r = 2$, and omit this from the notation. Denote the event that \mathcal{S}^* accepts on history H_r by $\text{Acc}(H_r)$. As \mathcal{S}^* is a probability-maximizing strategy,

$$\mathcal{S}^*(H_r) = \text{accept} \iff \Pr[\text{win} | \text{Acc}(H_r)] \geq \Pr[\text{win} | \neg \text{Acc}(H_r)]. \quad (1)$$

Given that $d_r = d$, $\Pr[\text{win} | \text{Acc}(H_r)] = d/n$, as this is exactly the probability that the best secretary is part of a group of d secretaries selected uniformly at random. Although we cannot give such an elegant formula for $\Pr[\text{win} | \neg \text{Acc}(H_r)]$, we know that it is the probability of winning given that we have seen d secretaries, rejected them all, and have $(n-d)$ secretaries remaining to observe; hence, the probability is dependent only on d (as n is fixed). Denote this probability function by $g(d)$. We do not attempt to describe g , other than to say that g must be non-increasing in d . (This is easy to see: $g(d) \geq g(d+1)$ as a possible strategy is to always reject the d^{th} secretary.)

As the left side of (1) is an increasing function of d , and the right side is a decreasing function of d (and as \mathcal{S}^* is a probability-maximizing function), \mathcal{S}^* will accept only if the number of distinct secretaries that have arrived is at least d^* , the minimal d such that $d/n \geq g(d)$. We can conclude that the optimal strategy is to observe the first d^* secretaries without hiring any and to choose the first suitable secretary thereafter. It is easy to see (similarly to [8]), that randomization cannot lead to a better stopping rule. ◀

From Lemma 3.1, we can conclude that there is some function $f : n \rightarrow [0, n]$ for which the optimal algorithm for the returning secretary problem is Algorithm 1.

Algorithm 1: Returning secretary algorithm with function $f : n \rightarrow [0, n]$

Input : n , the number of secretaries

Output: A secretary s

the Candidate = \emptyset ;

for round $r = 1$ to $2n$ **do**

 Let i_r be the secretary that arrives on round r ;

 Denote by d_r the distinct number of secretaries that have arrived up to round r ;

 if i_r is the Candidate **then**

 if $d_r > f(n)$ **then**

 Return i_r ;

 if i_r is better than the Candidate **then**

 the Candidate = i_r ;

We do not, at this time, attempt to find the function f for which Algorithm 1 is optimized; we will optimize the parameter of a similar algorithm for a slightly different setting in Subsection 3.3. For now, we focus on the special case where $f(n) \equiv 0$, which we call the *no waiting* case. Aside from being interesting in their own right, these results will come in useful later on, for tightly bounding the success probability.

3.2 The No Waiting Case

In the classical secretary problem, even if we don't know n in advance, we can still find the best secretary with a reasonable probability, assuming we have some other information regarding the secretaries. For example, the secretaries can have an known arrival time density over $[0, 1]$ [7]⁸; n can be selected from some known distribution [26]; there are other, similar scenarios (see e.g., [29, 1, 25]). However, with no advance knowledge at all, it is impossible to attain a success probability better than $1/n$ (with a deterministic algorithm): if we don't accept the first item, we run the risk of there being no other items, while if we do accept it, we have accepted the best secretary with probability $1/n$. It is easy to see that while randomization may help a little, it cannot lead to a constant success probability. In the returning-online scenario, though, we have the following result.

► **Theorem 3.2.** *In the returning secretary problem, even if we have no previous information on the secretaries, including the number of secretaries that will arrive, we can hire the best secretary with probability at least $2/3$.*

Denote by win the event that we hire the best secretary. Theorem 3.2 is immediate from the following lemma.

► **Lemma 3.3.** *When applying Algorithm 1 to the returning secretary problem with $f(n) \equiv 0$,*

$$\Pr[\text{win}] = \frac{2n + 1}{3n}.$$

⁸ Note that this is different from the alternative formulation described in the introduction as in this case n is unknown.

Proof. Let us call the best secretary Don. If we reach round i and see Don, we say we *win* on round i , and denote this event win_i . (Notice that we can say that we win at this point even though this is the first time we see Don, as we will certainly hire him). The probability of winning on round 1 is exactly the probability that Don arrives first:

$$\Pr[\text{win}_1] = \frac{2}{2n}.$$

We win on round 2 if any secretary other than Don arrived on round 1, and Don arrived on round 2.

$$\Pr[\text{win}_2] = \left(\frac{2n-2}{2n}\right) \left(\frac{2}{2n-1}\right).$$

The probability of winning on round $i > 2$ is the following (the best secretary we had seen until that point did not return between rounds 2 and $i-1$, and Don arrived on round i):

$$\Pr[\text{win}_i] = \left(\frac{2n-2}{2n}\right) \left(\frac{2n-4}{2n-1}\right) \left(\frac{2n-5}{2n-2}\right) \left(\frac{2n-6}{2n-3}\right) \cdots \left(\frac{2n-i-1}{2n-i+2}\right) \left(\frac{2}{2n-i+1}\right).$$

Therefore

$$\begin{aligned} \Pr[\text{win}] &= \frac{1}{n} + \frac{1}{n(2n-1)(2n-3)} \sum_{i=2}^{2n-2} (2n-i)(2n-i-1) \\ &= \frac{1}{n} + \frac{2(n-1)(2n-1)(2n-3)}{3n(2n-1)(2n-3)} \\ &= \frac{3}{3n} + \frac{2(n-1)}{3n} \\ &= \frac{2n+1}{3n}, \end{aligned} \tag{2}$$

where (2) is reached by substituting $j = 2n - i$ and simplifying the sum. ◀

3.3 Optimizing the Success Probability

We would now like to optimize f in Algorithm 1 in order to maximize the algorithm's success probability. For ease of analysis, we turn to the alternative model for the secretary problem: instead of generating a random permutation over the secretaries, each secretary i is allocated, uniformly and independently at random, two real numbers $r_i^1, r_i^2 \in [0, 1)$, representing his two arrival times. Assume that f^* is the optimal function for Algorithm 1. Fix n and let μ denote the time of the arrival of the $(f^*(n))^{th}$ distinct secretary. It is easy to see that the two models are asymptotically identical: for large n , $\Pr[i^j \text{ is one of the first } f^*(n) \text{ arrivals}] \cong \Pr[i^j \in [0, \mu)]$. The analysis in this model is much cleaner, and so, for simplicity, (and at the expense of accuracy for small n), we use it to obtain our bounds. The optimal algorithm for the returning secretary problem in this model is Algorithm 2.

We introduce some new notation.

- Denote by $\text{win}(\mu)$ the event that we hire the best secretary when using Algorithm 2 with parameter μ .
- Let $\alpha_i(\mu)$ be the event that $r_i^1, r_i^2 \in [0, \mu)$.
- Let $\beta_i(\mu)$ be the event that $r_i^1 \in [0, \mu)$ and $r_i^2 \in [\mu, 1)$ or vice versa.
- Let $\gamma_i(\mu)$ be the event that $r_i^1, r_i^2 \in [\mu, 1)$.

Algorithm 2: Returning secretary algorithm with parameter $\mu \in [0, 1)$

Output: A secretary s

the Candidate = \emptyset ;
 Observe the first secretary;
while *there are secretaries that have not arrived* **do**

- Let i be the observed secretary;
- Let t_i be the time that i is observed;
- if** i *is the Candidate* **then**
 - if** $t_i \geq \mu$ **then**
 - Return i ;
- if** i *is better than the Candidate* **then**
 - the Candidate = i ;
- Observe the next secretary;

We omit (μ) from the notation when it is clear from context. Label the best secretary by 1, the second best by 2 and so on. Denote by $\text{win}(NW_i)$ the event that we find the best secretary in the no waiting scenario with i secretaries (recall that this is $\frac{2i+1}{3^i}$). We make the following observations, which rely on the arrival times being independent.

► **Observation 3.4.** $\forall i \in [n], \Pr[\alpha_i(\mu)] = \mu^2, \Pr[\beta_i(\mu)] = 2\mu(1 - \mu), \Pr[\gamma_i(\mu)] = (1 - \mu)^2$.

► **Observation 3.5.** $\Pr[\text{win} | \gamma_1, \gamma_2, \dots, \gamma_i, \alpha_{i+1}] = \Pr[\text{win}(NW_i)]$.

Proof. If $\gamma_1, \gamma_2, \dots, \gamma_i$ hold then all of the appearances of the best i secretaries are in the interval $[\mu, 1)$. Both appearances of the $(i + 1)^{\text{th}}$ best secretary are in $[0, \mu)$; therefore we will definitely choose one of the i best secretaries, and the probability of choosing the best is as in the no waiting scenario. ◀

► **Observation 3.6.**

$$\Pr[\text{win} | \gamma_1, \gamma_2, \dots, \gamma_i, \beta_{i+1}] = \Pr[\text{win}(NW_{i+1}) | \text{secretary } i + 1 \text{ is the first to arrive}].$$

Proof. If $\gamma_1, \gamma_2, \dots, \gamma_i$ and β_{i+1} hold then all appearances of the best i secretaries are in the interval $[\mu, 1)$, and the $(i + 1)^{\text{th}}$ secretary arrived once by time μ . This reduces to the problem of choosing the best secretary in the no waiting scenario, given that the $(i + 1)^{\text{th}}$ secretary arrives first. ◀

► **Claim 3.7.** $\Pr[\text{win}(NW_{i+1}) | \text{secretary } i + 1 \text{ is the first to arrive}] = \frac{2i}{2i+1} \Pr[\text{win}(NW_i)]$.

Proof. Given that $i + 1$ is the first to arrive, if $i + 1$ arrives second, we lost. If not, $i + 1$ cannot be chosen anymore, and we are exactly in the no waiting scenario with i secretaries. The probability that $i + 1$ arrives second given that he also arrives first is $\frac{1}{2i+1}$. ◀

Combining Observation 3.6 and Claim 3.7 gives the following corollary.

► **Corollary 3.8.** $\Pr[\text{win} | \gamma_1, \gamma_2, \dots, \gamma_i, \beta_{i+1}] = \frac{2i}{2i+1} \Pr[\text{win}(NW_i)]$.

We are now able to obtain a recursive representation of $\Pr[\text{win} | \gamma_1, \gamma_2, \dots, \gamma_i]$.

► **Claim 3.9.** $\Pr[\text{win} | \gamma_1, \gamma_2, \dots, \gamma_i] = \frac{\mu^2 + 4\mu i - 2\mu^2 i}{3^i} + (1 - \mu)^2 \Pr[\text{win} | \gamma_1, \gamma_2, \dots, \gamma_{i+1}]$.

Proof.

$$\begin{aligned} \Pr[\text{win} \mid \gamma_1, \gamma_2, \dots, \gamma_i] &= \Pr[\text{win} \mid \gamma_1, \gamma_2, \dots, \gamma_i, \alpha_{i+1}] \Pr[\alpha_{i+1}] \\ &\quad + \Pr[\text{win} \mid \gamma_1, \gamma_2, \dots, \gamma_i, \beta_{i+1}] \Pr[\beta_{i+1}] \\ &\quad + \Pr[\text{win} \mid \gamma_1, \gamma_2, \dots, \gamma_i, \gamma_{i+1}] \Pr[\gamma_{i+1}] \\ &= \mu^2 \Pr[\text{win}(NW_i)] + \frac{4\mu i(1-\mu)}{2i+1} \Pr[\text{win}(NW_i)] \end{aligned} \quad (3)$$

$$\begin{aligned} &\quad + (1-\mu)^2 \Pr[\text{win} \mid \gamma_1, \gamma_2, \dots, \gamma_{i+1}] \\ &= \frac{\mu^2 + 4\mu i - 2\mu^2 i}{2i+1} \Pr[\text{win}(NW_i)] \\ &\quad + (1-\mu)^2 \Pr[\text{win} \mid \gamma_1, \gamma_2, \dots, \gamma_{i+1}], \\ &= \frac{\mu^2 + 4\mu i - 2\mu^2 i}{3i} + (1-\mu)^2 \Pr[\text{win} \mid \gamma_1, \gamma_2, \dots, \gamma_{i+1}], \end{aligned} \quad (4)$$

where (3) is due to Observations 3.4, and 3.5 and Corollary 3.8, and (4) is due to Lemma 3.3. \blacktriangleleft

► **Claim 3.10.** For any constant k , and any $\mu \in [0, 1)$,

$$\Pr[\text{win}] \geq 2\mu(1-\mu) + \sum_{i=1}^k \left(\frac{(1-\mu)^{2i}(\mu^2 + 4\mu i - 2\mu^2 i)}{3i} \right) + \frac{2}{3}(1-\mu)^{2k+1}. \quad (5)$$

Proof.

$$\begin{aligned} \Pr[\text{win}] &= \Pr[\text{win} \mid \alpha_1] \cdot \Pr[\alpha_1] + \Pr[\text{win} \mid \beta_1] \cdot \Pr[\beta_1] + \Pr[\text{win} \mid \gamma_1] \cdot \Pr[\gamma_1] \\ &= 0 \cdot (\mu^2) + 1 \cdot 2\mu(1-\mu) + \Pr[\text{win} \mid \gamma_1] \cdot (1-\mu)^2, \end{aligned} \quad (6)$$

where (6) is due to Observation 3.4.

Recursively applying Claim 3.9, and noticing that $\Pr[\text{win} \mid \gamma_1, \gamma_2, \dots, \gamma_i] \geq \frac{2}{3}$, for all i , completes the claim. \blacktriangleleft

► **Lemma 3.11.** For any $x \in [0, 1)$, $\Pr[\text{win}] \geq 2x - \frac{4}{3}x^2 - \frac{1}{3}(1-x)^2 \log(1-x^2)$.

Proof. Substituting $x = 1 - \mu$ in (5), and ignoring the lowest order term, we get

$$\begin{aligned} \Pr[\text{win}] &\geq 2x(1-x) + \frac{1}{3} \sum_{i=1}^k x^{2i} \left(\frac{(1-x)^2 + 4(1-x)i - 2(1-x)^2 i}{i} \right) \\ &= 2x(1-x) + \frac{1}{3}(1-x)^2 \sum_{i=1}^k \frac{x^{2i}}{i} + \frac{1}{3} \sum_{i=1}^k x^{2i}(4-4x) - 2(1-x)^2 \\ &= 2x(1-x) + \frac{1}{3}(1-x)^2 \sum_{i=1}^k \frac{x^{2i}}{i} + \frac{1}{3} \sum_{i=1}^k x^{2i}(2-2x^2) \\ &= 2x(1-x) + \frac{1}{3}(1-x)^2 \sum_{i=1}^k \frac{x^{2i}}{i} + \frac{1}{3} \left(\sum_{i=1}^k 2x^{2i} - \sum_{i=1}^k 2x^{2(i+1)} \right) \\ &\geq 2x(1-x) + \frac{1}{3}(1-x)^2 \sum_{i=1}^k \frac{x^{2i}}{i} + \frac{2}{3}x^2 \end{aligned} \quad (7)$$

$$\xrightarrow[k \rightarrow \infty]{} 2x - \frac{4}{3}x^2 - \frac{1}{3}(1-x)^2 \log(1-x^2), \quad (8)$$

Algorithm 3: Returning matroid secretary algorithm

Input : a cardinality $n = |E|$ of the matroid $\mathcal{M} = (E, \mathcal{I})$

Output: an independent set $S \in \mathcal{I}$

Let n elements arrive, without choosing any element;

Let E' denote the elements which only arrived once thus far;

Relabel the elements of E' by $1, 2, \dots, |E'|$, such that $w_1 \geq w_2 \geq \dots \geq w_{|E'|}$;

$S \leftarrow \emptyset$;

for $i = 1$ *to* $|E'|$ **do**

if $S \cup i \in \mathcal{I}$ **then**
| $S \leftarrow S \cup i$;

Return S ;

where in (7), we once again ignore the lowest order term, and (8) is because $\sum_{i=1}^{\infty} \frac{y^i}{i}$ is the Taylor series for $-\log(1-y)$, for $|y| < 1$. ◀

Differentiating (8), we find that the winning probability is maximized at

$$x = \sqrt{\frac{e^5 - e^{W(2e^5)}}{e^{5/2}}} \approx 0.727374\dots$$

, where $W(z)$ is the Lambert W function (also known as the the product log function). This implies $\mu \approx 0.272626\dots$, and for this value, $\Pr[\text{win}] \approx 0.767974\dots$. This gives our main result of the section.

► **Theorem 3.12.** *The success probability of Algorithm 2 with $\mu = 0.272626\dots$ converges to the success probability of the optimal algorithm for the returning secretary problem, as n tends to infinity; the probability of hiring the best secretary using Algorithm 2 is at least 0.767974.*

4

 Extension to Matroid Secretary Problems

We extend our results to the matroid secretary problem. Due to space restrictions, we only provide an outline of the results, and defer the proofs to the full version of the paper.

4.1 The Returning Matroid Secretary

We show that in the returning online model, when $k = 2$, a simple algorithm obtains a 2-approximation to the maximum-weight basis of the matroid. It is a well known property of matroids (e.g., [27]), that the Greedy algorithm always finds a maximum-weight basis. Algorithm 3, in essence, lets n elements arrive, and then runs the Greedy algorithm on the elements which have only arrived once.

► **Theorem 4.1.** *There is a simple algorithm for the returning matroid secretary problem that is 2-competitive in expectation.*

We use the following algorithm. Due to space considerations, the analysis of the algorithm and proof of Theorem 4.1 is deferred to the full version of the paper.

Algorithm 4: Returning bipartite edge-weighted matching algorithm

Input : vertex set R and a cardinality $n = |L|$
Output: a matching M

Let L_r be the vertices that arrived until round r ;
 Let $L' \subset L_n$ denote the vertices that only arrived once until round n ;
 $M =$ optimal matching on $G[L' \cup R]$;
for each subsequent round $t > n$, when vertex $\ell_t \in L$ arrives **do**

$M_t =$ optimal matching on $G[L_t \cup R]$;
Let e_t be the edge assigned to ℓ_t in M_t ;
if $M \cup e_t$ is a matching then
$M = M \cup e_t$;

Return M ;

We also show that in some cases, this algorithm can be improved to give better bounds; specifically in the case of bipartite edge-weighted matching (a generalization of transversal matroids).

4.2 Returning Bipartite Edge-Weighted Matching

The *returning bipartite edge-weighted matching* problem is a generalization of the returning transversal matroid problem.⁹ Let $G = (L \cup R, E)$ be a bipartite graph with a weight function $w : E \rightarrow \mathbb{R}^+$. We are initially given R and $n = |L|$. In each step, a vertex $v \in L$ arrives together with its edges (and the edges' weights). Each vertex arrives twice, and the order of arrival is selected uniformly at random from the $(2n)!$ possible arrival orders. When a vertex $\ell \in L$ arrives for the second time, it is either matched to one of the free vertices in R that are adjacent to ℓ , or left unmatched. The goal of the algorithm is to maximize the weight of the matching. Note that if $|R| = 1$, and we succeed only if we find the maximum matching, this is exactly the returning secretary problem.

We present a variation on the returning matroid secretary algorithm, where instead of the Greedy algorithm, we use a maximum-matching algorithm (using any maximum matching algorithm, e.g., the Hungarian method [21]). We then use local improvements, similarly to [18]. Once again, we present the algorithm here, but due to space considerations, leave its analysis and the proof of Theorem 4.2 to the full version.

► **Theorem 4.2.** *Algorithm 4 is 16/9-competitive in expectation.*

Acknowledgements I would like to thank Yishay Mansour and the anonymous reviewers for their insightful comments.

References

- 1 A.R. Abdel-Hamid, J.A. Bather, and G.B. Trustrum. The secretary problem with an unknown number of candidates. *J. Appl. Prob.*, 19:619–630, 1982.

⁹ A *transversal matroid* is a bipartite graph $G = (L \cup R, E)$ where the elements are the vertices of L and the independent sets are sets of endpoints of matchings in the graph. Transversal matroids are a special case of bipartite edge-weighted matching, in which all the edges incident on the same vertex $\ell \in L$ have the same weight.

- 2 Moshe Babaioff, Michael Dinitz, Anupam Gupta, Nicole Immorlica, and Kunal Talwar. Secretary problems: Weights and discounts. In *SODA*, pages 1245–1254, 2009.
- 3 Moshe Babaioff, Nicole Immorlica, and Robert Kleinberg. Matroids, secretary problems, and online mechanisms. In *SODA*, pages 434–443, 2007.
- 4 Bahman Bahmani, Aranyak Mehta, and Rajeev Motwani. Online graph edge-coloring in the random-order arrival model. *Theory of Computing*, 8(1):567–595, 2012.
- 5 Avrim Blum, Tuomas Sandholm, and Martin Zinkevich. Online algorithms for market clearing. *J. ACM*, 53(5):845–879, 2006.
- 6 Jonathan Bredin and David C. Parkes. Models for truthful online double auctions. *CoRR*, abs/1207.1360, 2012.
- 7 F. Thomas Bruss. A unified approach to a class of best choice problems with an unknown number of options. *The Annals of Probability*, 12(3):882–889, 08 1984.
- 8 F. Thomas Bruss. Sum the odds to one and stop. *Annals of Probability*, 28:1384–1391, 2000.
- 9 Michael Dinitz. Recent advances on the matroid secretary problem. *SIGACT News*, 44(2):126–142, 2013.
- 10 E. B. Dynkin. The optimal choice of the instant for stopping a Markov process. *Soviet Math. Dokl.*, 4:627–629, 1963.
- 11 Thomas S. Ferguson. Who solved the secretary problem? *Statistical Science*, 4:282–296, 1989.
- 12 P.R. Freeman. The secretary problem and its extensions: A review. *International Statistical Review*, 51(2):189–206, 1983.
- 13 M. Gardner. Mathematical games. *Scientific American*, 202:152,178–179, 1960.
- 14 Shayan Oveis Gharan and Jan Vondrák. On variants of the matroid secretary problem. *Algorithmica*, 67(4):472–497, 2013.
- 15 John P. Gilbert and Frederick Mosteller. Recognizing the maximum of a sequence. *J. Amer. Statist. Assoc.*, 61(313):35–73, 1966.
- 16 Mohammad Taghi Hajiaghayi, Robert D. Kleinberg, Mohammad Mahdian, and David C. Parkes. Online auctions with re-usable goods. In *EC*, pages 165–174, 2005.
- 17 Mohammad Taghi Hajiaghayi, Robert D. Kleinberg, and David C. Parkes. Adaptive limited-supply online auctions. In *EC*, pages 71–80, 2004.
- 18 Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. An optimal online algorithm for weighted bipartite matching and extensions to combinatorial auctions. In *ESA*, pages 589–600, 2013.
- 19 Robert D. Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *SODA*, pages 630–631, 2005.
- 20 Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In *ICALP (2)*, pages 508–520, 2009.
- 21 H.W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.
- 22 Oded Lachish. $O(\log \log \text{rank})$ competitive ratio for the matroid secretary problem. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 326–335, 2014.
- 23 D.V. Lindley. Dynamic programming and decision theory. *Appl. Statist.*, 10:39–52, 1961.
- 24 Mohammad Mahdian and Qiqi Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs. In *STOC*, pages 597–606, 2011.
- 25 Zdzisław Porosiński. The full-information best choice problem with a random number of observations. *Stochastic Processes and their Applications*, 24(2):293–307, 1987.
- 26 E.L. Presman and I.M. Sonin. The best choice problem for a random number of objects. *Theory Prob. Applic.*, 17:657–668, 1982.

- 27 R. Rado. A note on independence functions. *Proc. London Math. Soc.*, 7:300–320, 1957.
- 28 José A. Soto. Matroid secretary problem in the random-assignment model. *SIAM J. Comput.*, 42(1):178–211, 2013.
- 29 T.J. Stewart. The secretary problem with an unknown number of options. *Operations Research*, 29:130–145, 1981.

Homomorphism Reconfiguration via Homotopy

Marcin Wrochna

Faculty of Mathematics, Informatics and Mechanics,
University of Warsaw, Warsaw, Poland
m.wrochna@mimuw.edu.pl

Abstract

We consider the following problem for a fixed graph H : given a graph G and two H -colorings of G , i.e. homomorphisms from G to H , can one be transformed into the other by changing one color at a time, maintaining an H -coloring throughout. This is the same as finding a path in the $\text{Hom}(G, H)$ complex. For $H = K_k$ this is the problem of finding paths between k -colorings, which was recently shown to be in P for $k \leq 3$ and PSPACE-complete otherwise (Bonsma and Cereceda 2009, Cereceda et al. 2011). We generalize the positive side of this dichotomy by providing an algorithm that solves the problem in polynomial time for any H with no C_4 subgraph. This gives a large class of constraints for which finding solutions to the Constraint Satisfaction Problem is NP-complete, but paths in the solution space can be found in polynomial time.

The algorithm uses a characterization of possible reconfiguration sequences (that is, paths in $\text{Hom}(G, H)$), whose main part is a purely topological condition described in terms of the fundamental groupoid of H seen as a topological space.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases reconfiguration, recoloring, homomorphisms, homotopy, hom complex

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.730

1 Introduction

Reconfiguration

Reconfiguration is a framework in which we study how discrete structures, constrained in various ways, can be carefully transformed with small steps. This is often best described by finding paths in a solution graph, whose vertices are all solutions to a combinatorial problem and whose edges define the steps between solutions one is allowed to make.

For example, in k -RECOLORING [3, 4, 8, 17], one is given two proper k -colorings of a graph G and the question is whether one can be transformed into the other by changing one color at a time, maintaining a proper coloring throughout. In other words, the solution graph has proper k -colorings as vertices (solutions) and edges (reconfigurations steps) between any two colorings that differ only at one vertex of G . Another well studied example is TOKEN JUMPING [12, 18], where the solutions are independent sets of some given size (seen as tokens on the graph's vertices) and a reconfiguration step removes one vertex from the set to add another (jumps one token to a different vertex). Yet another example is the reconfiguration of generalized SAT problems [11, 21, 24], where solutions are satisfying assignments of a given formula, and a reconfiguration step flips one variable of the assignment.



© Marcin Wrochna;

licensed under Creative Commons License CC-BY

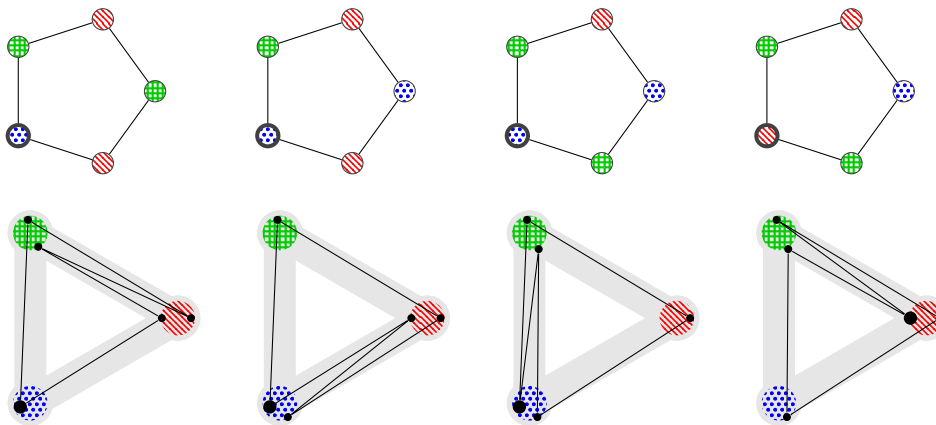
32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).

Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 730–742

Leibniz International Proceedings in Informatics

LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany





■ **Figure 1** A sequence of 3-colorings of C_5 and the same sequence seen as mappings from C_5 to K_3 (a graph with three vertices: red, green, blue), that is, a path in $\text{Hom}_1(C_5, K_3)$. One vertex of C_5 is thickened for clarity.

Homomorphisms

A homomorphism from a graph G to a graph H is a mapping $\sigma : V(G) \rightarrow V(H)$ such that edges are mapped to edges, that is, $uv \in E(G)$ implies $\sigma(u)\sigma(v) \in E(H)$. We also use the name *H-coloring*, especially when H is fixed in the context. Vertices of H are then called *colors*. Note that a K_k -coloring (so $H = K_k$ is the graph with all edges except loops) is the same as a (proper) k -coloring.

The solution graph $\text{Hom}_1(G, H)$ is defined to be the graph with H -colorings of G as vertices and edges between any two H -colorings that differ in the color of only one vertex (Figure 1). For a fixed graph H , H -RECOLORING is the problem where given a graph G and two H -colorings of G we are asked whether they are connected by a path in $\text{Hom}_1(G, H)$. SHORTEST H -RECOLORING asks whether there is such a path of at most some given length.

We use graph homomorphisms as a tool to explore how different constraints influence the complexity of reconfiguration. As our aim is to give more general statements about reconfiguration, they should be seen as a special case of Constraint Satisfaction Problems (CSPs), which can express a range of problems including k -COLORING, generalized SAT or INDEPENDENT SET (in weighted variants). However, graph homomorphisms already display many features of general CSPs and arise naturally in various situations. See [22] for an excellent survey and [16] for an in-depth book on the subject.

This approach allowed us to argue in [26] that the only notion of sparseness that can apply to (unparameterized) reconfiguration problems in general is treedepth. The reduction showed there explains why reconfiguration variants of easy combinatorial problems can be hard. In this paper we explore why reconfiguration variants of hard problems can be easy.

Motivations

The primary motivation for studying reconfiguration problems is to investigate the solution space of combinatorial problems, especially from the perspective of local search heuristics and random solution sampling. In particular, the success of Survey Propagation as a method for solving random Constraint Satisfaction Problems (CSPs) is connected to several conjectures about the structure of clusters of satisfying assignments (connected components in the solution graph) and frozen variables (variables/vertices that cannot change their value/color by any sequence of steps), see [1].

While finding paths in the solution graph is, for the above purposes, mostly a toy problem, it arises more directly in some settings. For example, the construction of Hearn and Demaine [12] allowed to show that many popular puzzles are PSPACE-complete [13, 20], and more interestingly, so is the problem of proof equivalence in a certain proof system [14], which answered an earlier question about normal forms of proofs.

Related work

H -RECOLORING has been shown PSPACE-complete for $H = K_k$ where $k \geq 4$ by Bonsma and Cereceda [3], and in P for $k \leq 3$ by Cereceda et al. [8]. The latter result has been extended to show that SHORTEST K_3 -RECOLORING is also in P by Johnson et al. [17].

For CSPs in the Boolean domain, a dichotomy was shown by Gopalan et al. [11] – for a fixed set of Boolean constraints Γ (i.e., Boolean relations, clause types), the problem of finding paths in the solution graph of a $\text{SAT}(\Gamma)$ instance is either in P or PSPACE-complete. In particular it is always in P when the corresponding satisfiability problem is in P (e.g. 2-SAT or Horn-SAT), but it is also in P for some Γ for which satisfiability is NP-complete. This was slightly corrected and extended to several similar problems by Schwerdtfeger [24], while a trichotomy was shown for the problem of finding *shortest* paths by Mouawad et al. [21]. Both [11] and [24] asked whether their results could be extended to larger domains. Our work can be seen as a step in this direction, but limited to only one symmetric relation of arity 2.

The corresponding extension of Schaefer’s dichotomy [23] for satisfiability (deciding the existence of a solution) to CSPs with arbitrary finite domains is a long-standing open problem stated by Feder and Vardi [10]. They showed that the conjecture is equivalent when limited to one relation of arity 2 (digraph homomorphisms). Hell and Nešetřil proved the dichotomy in the case the relation is additionally assumed to be symmetric (graph homomorphism) [15]: the problem of deciding the existence of an H -coloring of a given graph is in P for H bipartite or containing a loop, and NP-complete otherwise.

Results

It is natural to ask whether the unexpected tractability of K_3 -RECOLORING (in light of 3-COLORABILITY being NP-complete) is caused by the following property: whenever a vertex changes its color in a 3-coloring (e.g. from red to green), all of its neighbors must have one common color (blue).

We answer this in the positive considering the following definition: a graph H has the *monochromatic neighborhood property* if for every two colors $a, b \in V(H)$, the set of common neighbors $N_H(a) \cap N_H(b)$ contains at most one color. For graphs without loops this is equivalent to not containing C_4 (the cycle on 4 vertices) as a subgraph (not necessarily induced). This includes K_3 and all graphs of girth ≥ 5 , for example. For graphs with loops allowed, this is equivalent to not containing C_4 , K_3 with one loop added, nor K_2 with both loops added.

We show an algorithm that solves SHORTEST H -RECOLORING in polynomial time for all H with the monochromatic neighborhood property. To achieve this, we characterize possible paths in $\text{Hom}_1(G, H)$ by describing sequences of colors one vertex of G takes throughout an H -recoloring. We observe that the H -colorings of G correspond to continuous maps from G to H (seen as topological spaces) and that recoloring corresponds to a continuous transformation of these maps – a homotopy. This gives a topological condition on how a path in $\text{Hom}_1(G, H)$ can look like. It turns out that to give a complete characterization we only need to add a simple parity condition and a condition freezing some easily found

vertices. Thus a combinatorial problem is reduced to describing possible homotopies, which we do with standard algebraic calculations. This gives an unexpected connection that might be interesting on its own.

In combinatorial algebraic topology

Reconfiguration of homomorphisms has already been studied independently in the field of combinatorial algebraic topology, though from a different angle. The notion of \times -homotopy of homomorphisms as defined by Dochtermann [9] is identical to reachability in the solution graph $\text{Hom}_1(G, H)$. The solution graph $\text{Hom}_1(G, H)$ arises as the subgraph of the exponential graph H^G induced by looped vertices and as the 1-skeleton of the Hom-complex $\text{Hom}(G, H)$, see [2]. The Hom-complex, studied for its interesting categorical and topological properties, is a construction similar (and homotopy equivalent) to the clique complex of $\text{Hom}_1(G, H)$. These definitions were introduced to provide lower bounds on the chromatic number of graphs, a notoriously hard problem. A typical theorem derived from such methods is that for loopless graphs G, H , if $\text{Hom}_1(G, H)$ is connected for all G of degree at most d , then the chromatic number of H is at least $d/2$ (and is conjectured to be at least d) [5]. Studies have thus been mostly concerned with highly regular graphs for which the solution graph can be proved to be in some sense tightly connected.

We do not know of any prior work on the computational complexity of deciding \times -homotopy. More surprisingly, we do not know of any prior example where \times -homotopy (of homomorphisms) would be related to homotopy (of continuous mappings between graphs), more than through analogy.

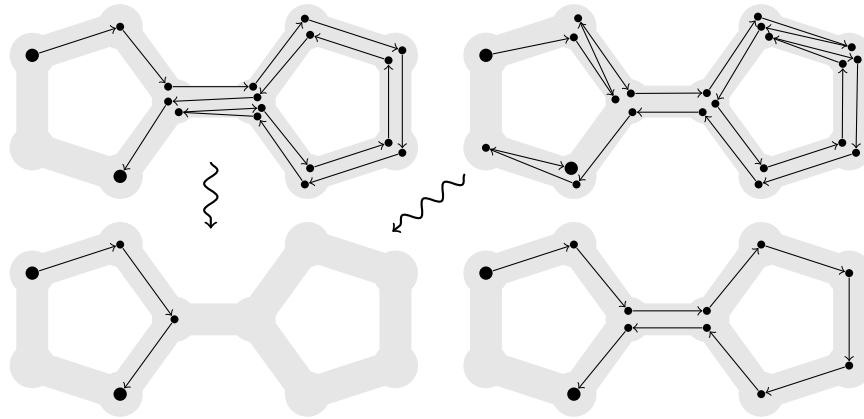
A longer version of this paper, with full proofs, is available at <http://arxiv.org/abs/1408.2812>.

2 Preliminaries

An (undirected) graph G is a pair $(V(G), E(G))$ where $V(G)$ is a finite set of *vertices*, while $E(G)$ is the set of *edges* – unordered vertex pairs $\{u, v\}$, $u, v \in V(G)$, written uv for short. A *loop* is an edge uu for some vertex u . The neighborhood $N_G(v)$ is defined as $\{w \in V(G) \mid vw \in E(G)\}$. Hence $a \in N_H(a)$ iff H has a loop at a .

G and H in this paper are always connected undirected graphs with at least one edge. G is always assumed to have no loops. H can have loops, but is always assumed to have the monochromatic neighborhood property.

An *H-recoloring sequence* or *reconfiguration sequence* is a path in $\text{Hom}_1(G, H)$, that is, a sequence of H -colorings of G in which consecutive colorings differ at one vertex. Consider a step of an H -recoloring sequence – a vertex $v \in V(G)$ changes color from $a \in V(H)$ to $b \in V(H)$. Since G is connected, loopless and has an edge, v has a neighbor, say $w \neq v$. As only v changes its color in the step, w has the same color, say $h \in V(H)$, before and after the step. The H -coloring before the step implies that $ha \in E(H)$, while the one after the step implies that $hb \in E(H)$. Thus $h \in N_H(a) \cap N_H(b)$. From the monochromatic neighborhood property we infer that $N_H(a) \cap N_H(b) = \{h\}$. We will often call h ‘*the color that all neighbors of v have during the step*’ (that is, in the H -colorings before and after the step), without arguing its existence and uniqueness anymore.



■ **Figure 2** Examples of two walks (in a graph H on 10 vertices) which reduce to the same, bottom left one. The bottom right one is a different reduced walk; when its endpoints are fixed, it cannot be distorted as a curve to give any of the others.

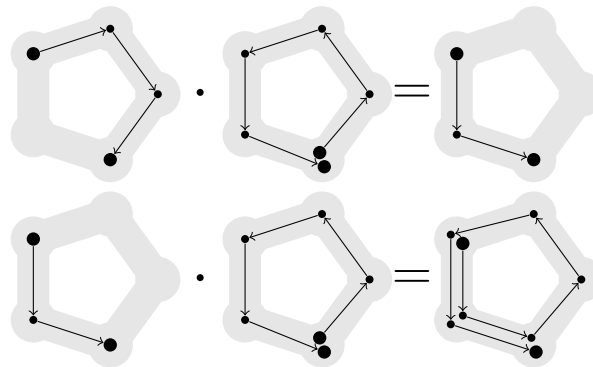
3 Fundamental groupoid

An *oriented edge* of a graph H is an ordered pair $e = (u, v)$ such that $\{u, v\}$ is an edge of H ; we denote its initial vertex u as $\iota(e)$ and its target vertex v as $\tau(e)$. We write e^{-1} for $(\tau(e), \iota(e))$. A *walk* from u to v in a graph H is a sequence of oriented edges $e_1 e_2 \dots e_l$ of H such that $\iota(e_1) = u$, $\tau(e_l) = v$ and $\tau(e_i) = \iota(e_{i+1})$ for $i = 0, \dots, l-1$. We write ε for an empty walk (the endpoints of ε will be irrelevant for us). The length of a walk is the number of edges in it. A walk W_1 from u to v can be concatenated to a walk W_2 from v to w to form a walk $W_1 W_2$ from u to w .

We call a walk *reduced* if it contains no two consecutive edges $e_i e_{i+1}$ such that $e_{i+1} = e_i^{-1}$. One can *reduce* a walk by removing any such two consecutive edges from the sequence. It can easily be seen that by iteratively reducing a walk W , one always gets the same reduced walk, which we therefore denote as \overline{W} , see Figure 2. For any two reduced walks $\overline{W}_1, \overline{W}_2$ such that \overline{W}_2 starts where \overline{W}_1 ends, we write $\overline{W}_1 \cdot \overline{W}_2$ for $\overline{W}_1 \overline{W}_2$ and similarly one can observe that \cdot is associative. For any walk $W = e_1 e_2 \dots e_l$ we write W^{-1} for the reversed walk $e_l^{-1} \dots e_2^{-1} e_1^{-1}$. Clearly $\overline{W} \cdot \overline{W}^{-1} = \varepsilon = \overline{W}^{-1} \cdot \overline{W}$ and $\varepsilon \cdot \overline{W} = \overline{W} \cdot \varepsilon = \overline{W}$. Therefore, the set of reduced walks of a graph forms together with the operations \cdot and $()^{-1}$ a *groupoid* – a group, except for the binary operation being a partial function. This particular groupoid is called the *fundamental groupoid* $\pi(H)$ of H , see Figure 3.

A groupoid is in many ways similar to a group. Identities such as $(e \cdot f)^{-1} = f^{-1} \cdot e^{-1}$ known from group theory are easily reproved in groupoids. Using the fundamental groupoid as opposed to the better known fundamental group allows us to describe calculations much more uniformly, without the tedious change of base points, for example. Readers familiar with category theory may benefit from the view given by equivalent definitions: a groupoid is a category in which every morphism is invertible, and a group is a groupoid with one object. A brief survey on groupoids and their application in topology [6] and a more complete exposition [7] have been written by R. Brown.

Note that if H has a loop at u , then $(u, u) = (u, u)^{-1}$ is an oriented edge and a reduced walk of length 1. On the other hand, $(u, u) \cdot (u, u) = \varepsilon$. (Hence $\pi(H)$ is strictly speaking not a *free groupoid*, see [19]).



■ **Figure 3** Examples of \cdot multiplication in the fundamental groupoid of $H = C_5$.

Topological interpretation

Let us comment on how this algebraic structure captures the topology of curves in the graph. When referring to topology, we give only informal interpretations, as the statements are not needed in any of the proofs.

A graph H without loops can be naturally associated with a topological space, constructed from copies of the $[0, 1]$ interval for each edge, with endpoints merged into vertices accordingly. A *curve* in this space is a continuous map $f : [0, 1] \rightarrow H$ (where H is meant as the topological space), not necessarily injective. Two curves f_0, f_1 are *homotopic* if one can be continuously transformed into the other, which means there is a set of functions $\{\phi_t : t \in [0, 1]\}$ such that $\phi_0 = f_0, \phi_1 = f_1$ and the mapping $\phi : (t, x) \mapsto \phi_t(x)$ is continuous as a function from $[0, 1] \times [0, 1]$ to H .

The fundamental groupoid fully describes curves up to homotopy. For any two vertices u, v of H , two curves f_0, f_1 are homotopic via a homotopy ϕ_t that fixes the endpoints ($\phi_t(0) = u, \phi_t(1) = v$ for all t) if and only if the corresponding reduced walks in $\pi(H)$ are equal. Considering only reduced walks that start and end in the same vertex v , we obtain a group, which is known as the fundamental group $\pi_1(H, v)$ of the topological space H . When no vertex is fixed, a closed curve is homotopic to another, via a homotopy such that $\phi_t(1) = \phi_t(0)$ for all t , if and only if the corresponding elements C_1, C_2 of $\pi(H)$ are *conjugate*, i.e. $C_2 = P^{-1} \cdot C_1 \cdot P$ for some $P \in \pi(H)$.

4 Vertex walks and realizability

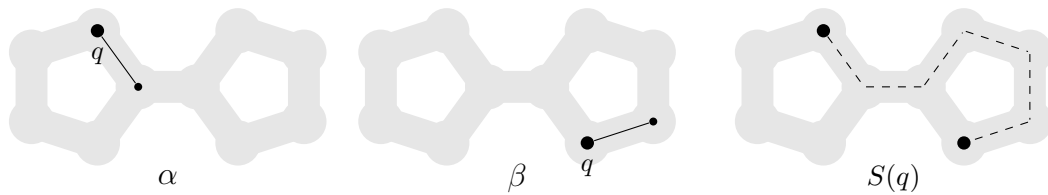
Consider an H -recoloring sequence $S = \sigma_0, \dots, \sigma_l$ of G and any vertex $v \in V(G)$. We define $S(v)$ as the following walk in H :

- If $l = 0$, that is, S is empty, then $S(v) = \varepsilon$.
- If $l = 1$, that is, S contains a single reconfiguration step, then

$S(v) = \varepsilon$	when $\sigma_0(v) = \sigma_1(v)$, and
$S(v) = (\sigma_0(v), h)(h, \sigma_1(v))$	otherwise,

 where h is the color that all neighbors of v have in σ_0 and σ_1 .
- If $l > 1$, then $S(v) = S_1(v)S_2(v) \dots S_l(v)$, where $S_i(v)$ is the walk corresponding to the reconfiguration step from σ_i to σ_{i+1} .

For two H -colorings α, β of G and a vertex $q \in V(G)$, we call an element $Q \in \pi(H)$ *realizable* if there is an H -recoloring sequence $S = \sigma_0, \dots, \sigma_l$ such that $\sigma_0 = \alpha, \sigma_l = \beta$ and $\overline{S(q)} = Q$. We focus on the following question: which $Q \in \pi(H)$ are realizable? It is



■ **Figure 4** A realizable walk for $\alpha, \beta : K_2 \rightarrow H$ and q . Note the shortest walk from $\alpha(q)$ to $\beta(q)$ (of length 3) is not realizable because of parity.

immediate from the definition that Q must be a reduced walk from $\alpha(q)$ to $\beta(q)$ and have even length (notice that the parity of the length of walks is preserved by reducing), see Figure 4. This parity condition will be one of three conditions characterizing realizable walks.

If $W = (v_1, v_2)(v_2, v_3) \dots (v_{n-1}, v_n)$ is a walk in G and α is an H -coloring of G , then $\alpha(W) = (\alpha(v_1), \alpha(v_2)) \dots (\alpha(v_{n-1}), \alpha(v_n))$ is a walk in H .

5 Topological validity

With a homomorphism from G to H one can associate a continuous map from G to H (seen as topological spaces as described above). The following lemma is the key to the monochromatic neighborhood property of H .

► **Lemma 1.** *Let S be an H -recoloring sequence of G from α to β . Consider any walk W from vertex u to v in G . Then $\overline{S(v)} = \overline{\alpha(W)}^{-1} \cdot \overline{S(u)} \cdot \overline{\beta(W)}$.*

The proof follows directly from the monochromatic neighborhood property and induction. The statement can be understood as an algebraic expression of the fact that when an H -coloring is reconfigured into another, the corresponding continuous mappings can be continuously transformed into one another, and the walk $S(v)$ of each vertex v is (up to reduction) the curve traced by the point v (see Figure 5). Let us focus on corollaries.

First, we see that in a given instance of H -RECOLORING, the reduced vertex walk $\overline{S(q)}$ of one vertex q in a solution S determines all other vertex walks up to reductions (in other words, up to homotopy). This is why we can focus on the realizability of one element $Q \in \pi(H)$ instead of a whole recoloring sequence. If we decide that Q is realizable, we will later use this lemma to completely recover a recoloring sequence S such that $\overline{S(q)} = Q$.

Second, note that the statement doesn't depend on how we choose walks in G . This means that for every pair of walks in G with equal endpoints, there is a topological condition on how solutions look like.

► **Definition 2.** Let α, β be two H -colorings of G and let q be a vertex of G . Let $Q \in \pi(H)$. We say Q is *topologically valid* for α, β, q if for every vertex w and every two walks W_1, W_2 from q to w in G we have $\overline{\alpha(W_1)}^{-1} \cdot Q \cdot \overline{\beta(W_1)} = \overline{\alpha(W_2)}^{-1} \cdot Q \cdot \overline{\beta(W_2)}$.

► **Corollary 3.** *Let α, β be two H -colorings of G and let q be a vertex of G . If $Q \in \pi(H)$ is realizable for α, β, q then Q is topologically valid for α, β, q .*

Using the groupoid structure, this condition can easily be rephrased in terms of single closed walks, instead of pairs of walks with equal starts and equal ends.

► **Lemma 4.** *Let α, β be two H -colorings of G and let q be a vertex of G . Then $Q \in \pi(H)$ is topologically valid for α, β, q if and only if for every closed walk C from q to q we have $\overline{\beta(C)} = Q^{-1} \cdot \overline{\alpha(C)} \cdot Q$.*

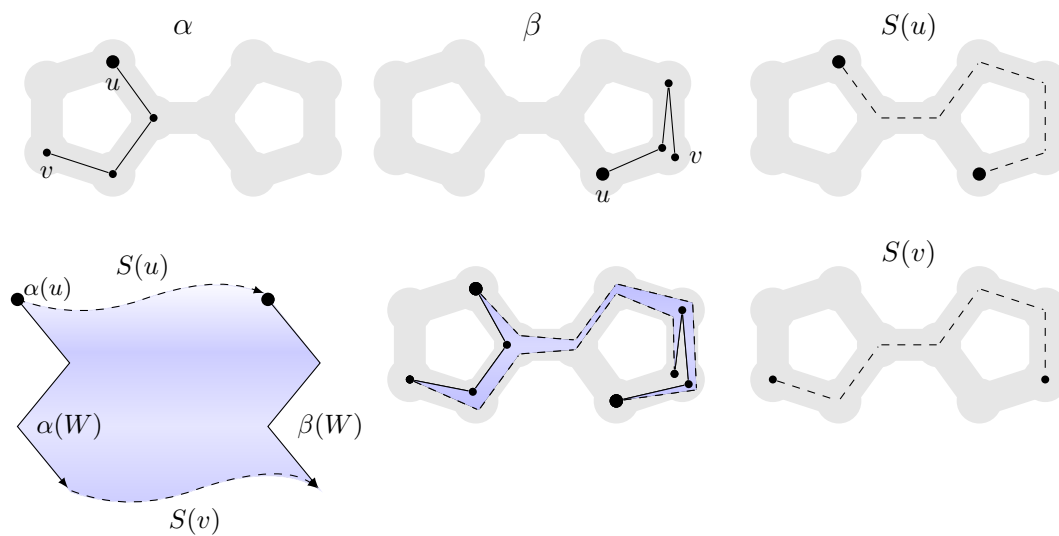


Figure 5 If α can be transformed to β by reconfiguration, then it can also be transformed by a homotopy. Restricting this homotopy to a walk W from u to v in G gives a map $\phi : [0, 1] \times [0, |W|] \rightarrow H$ such that $\phi(0, \cdot) = \alpha(W)$ and $\phi(1, \cdot) = \beta(W)$, $\phi(\cdot, 0) = S(u)$ and $\phi(\cdot, |W|) = S(v)$. Since ϕ is a continuous mapping of a rectangle to H , it's boundary, and so the image of it's boundary, can be contracted to a point. This is the meaning of the equality $\overline{\alpha(W)}^{-1} \cdot S(u) \cdot \overline{\beta(W)} \cdot S(v)^{-1} = \varepsilon$.

The name is motivated by the following fact: Q is *topologically valid* for α, β, q if and only if there is a homotopy continuously transforming α to β such that q traces the curve Q throughout this transformation ($\phi_0 = \alpha, \phi_1 = \beta$ and the image of $t \mapsto \phi_t(q)$ is Q).

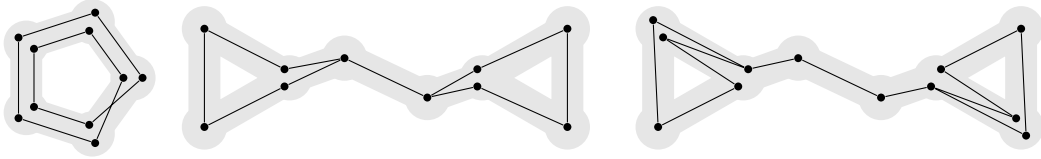
We note that the existence of a topologically valid walk for α, β implies in particular that cycles have the same homotopy class in α as in β , which algebraically is expressed in the fact that they have the same conjugacy class: $\overline{\beta(C)} = Q^{-1} \cdot \overline{\alpha(C)} \cdot Q$ for some $Q \in \pi(H)$. This generalizes one of the conditions of a characterization of 3-colorings connected by recoloring sequences given in [8] (Theorem 7 (C2)).

Another implication is that, intuitively, for each cycle C in G such that $\overline{\alpha(C)} \neq \varepsilon$, two solutions can differ only in the number of times vertex walks wind around $\overline{\alpha(C)}$. This is analyzed in more detail in the full version of the paper, but we will see this reflected when describing the set of realizable walks (e.g. Figure 8).

6 Tight closed walks and frozen vertices

We have seen that any realizable walk must have even length and must be topologically valid. There is one more obstruction to reconfiguration: closed walks whose sequences of colors are in a sense tightly stretched around H . In an H -coloring α of G , a vertex v of G is called *frozen* if for every H -recoloring sequence from α the resulting H -coloring β has $\beta(v) = \alpha(v)$. A closed walk $C = e_1 e_2 \dots e_l$ is *cyclically reduced* if it is reduced and also $e_l \neq e_1^{-1}$. In other words, repeating C gives an infinite reduced walk. A closed walk C is α -tight if $\alpha(C)$ is cyclically reduced. We show that vertices on tight closed walks are frozen.

► **Lemma 5.** *Let α be an H -coloring of G and let C be an α -tight walk in G . Then all vertices of C are frozen in α .*



■ **Figure 6** In the left example a C_5 -coloring of C_{10} is shown. there is a tight closed walk visiting all 10 vertices of the cycle in order. Hence no reconfiguration step is possible. Similarly in the middle example, there is a tight closed walk going around one 5-cycle, along the bridge, around the other 5-cycle and back along the bridge. In the right example no closed walk is tight, but the 4 middle vertices are frozen.

Tight closed walks can be found by starting from some vertex and exploring walks W such that $\alpha(W)$ is reduced. If they are arbitrarily long, then they contain the same oriented edge twice, giving an α -tight closed walk. We can answer whether an oriented edge is reachable from itself by a non-trivial reduced walk in time $\mathcal{O}(|E(G)|)$. The (potentially infinite) prefix tree of reduced walks starting from one vertex gives a generalization of the layer construction of [8] used to characterize frozen vertices when $H = K_3$. For other H , frozen vertices can arise in other situations (Figure 6), but it turns out we won't need to identify them.

Finding a frozen vertex v allows us to limit potentially realizable walks Q to a single one. Since $S(v) = \varepsilon$, Lemma 1 implies $Q = \overline{S(q)} = \overline{\alpha(W)}^{-1} \cdot \overline{S(v)} \cdot \overline{\beta(W)} = \overline{\alpha(W)}^{-1} \cdot \overline{\beta(W)}$.

► **Corollary 6.** *Let $Q \in \pi(H)$ be realizable for α, β – H -colorings of G and $q \in V(G)$. Let C be an α -tight closed walk in H . Then for every vertex v on C and every walk W from v to q we have $Q = \overline{\alpha(W)}^{-1} \cdot \overline{\beta(W)}$.*

7 Characterization of realizable walks

In the following theorem we show that there are no more conditions for a walk to be realizable than the three we described: even length, topological validity and frozenness of tight closed walks. This is very unexpected – the fact that edges are actually discrete and cannot be stretched arbitrarily, for example, turns out to imply no further obstructions to reconfiguration (it only restricts the input H -colorings).

► **Theorem 7.** *Let α, β be two H -colorings of G . Consider any vertex q of G and let $Q \in \pi(H)$ be a reduced walk in H from $\alpha(q)$ to $\beta(q)$. Then Q is realizable for α, β, q if and only if*

- Q is topologically valid for α, β, q ,
 - Q has even length,
 - if there is an α -tight walk, then for any walk W from it to q , $Q = \overline{\alpha(W)}^{-1} \cdot \overline{\beta(W)}$.
- Furthermore, there is an algorithm working in time $\mathcal{O}(|V(G)|^2 + |V(G)| \cdot |Q| + |E(H)|)$ that given G, H, α, β and a walk Q , checks whether Q satisfies the above conditions and if so, outputs a recoloring sequence such that $S(q) = Q$ and $S(v)$ is reduced for all $v \in V(G)$.

See Figure 7 for an example on how the conditions apply. While proving that realizability implies the conditions is easy, the proof of the converse spans 2 pages. The idea is that Lemma 1 gives vertex walk in a consistent way thanks to the first condition, they correspond to recoloring steps thanks to the second condition, and those can be ordered to give a recoloring sequence thanks to the third condition. In particular, the proof gives an effective way to obtain from a realizable $Q \in \pi(H)$ a recoloring sequence in which every vertex walk is reduced.

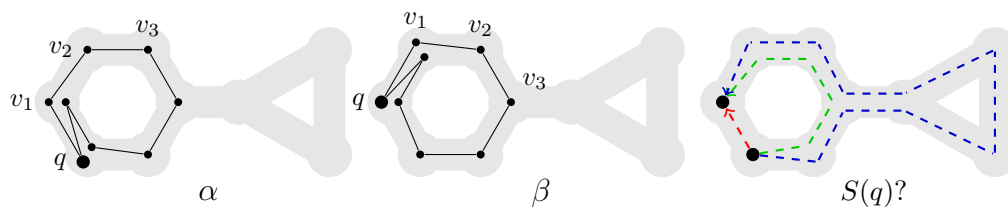


Figure 7 Even though no vertex is frozen, α cannot be reconfigured to β . The red and green walks are not realizable for α, β, q because of parity. The dark blue walk has good parity, but is not topologically valid (imagine pulling q along it).

► **Corollary 8.** Let α, β be two H -colorings of G . Let $S = \sigma_0, \dots, \sigma_l$ be a shortest H -recoloring sequence between $\sigma_0 = \alpha$ and $\sigma_l = \beta$. Then $S(v)$ is reduced for each vertex v .

Proof. Suppose $S(q)$ is not reduced for some q . Let $Q = \overline{S(q)}$. By the above theorem we know from one side that Q is realizable. From the other side we obtain a solution S' such that $S'(v) = \overline{S'(v)}$ for all v and $S'(q) = Q$. By Lemma 1, this implies $S'(v) = \overline{S(v)}$, for all v . But $\overline{S(v)}$ is always no longer than $S(v)$, and $\overline{S(q)}$ is strictly shorter than $S(q)$. Since the number of recoloring steps is equal to half the sum of lengths of all $S(v)$, S was not shortest. ◀

8 An algorithm

The following lemma follows from well-known calculations in the fundamental groupoid, which we recall in the appendix of the full version of the paper.

► **Lemma 9.** Let α, β be H -colorings of G and q a vertex of G . Consider the set $\Pi \subseteq \pi(H)$ of topologically valid walks for α, β, q . One of the following holds:

0. $\Pi = \emptyset$.
1. $\Pi = \{Q\}$ for some $Q \in \pi(H)$.
2. $\Pi = \{R^n \cdot P \mid n \in \mathbb{Z}\}$ for some $R, P \in \pi(H)$.
3. Π contains all reduced walks from $\alpha(q)$ to $\beta(q)$.

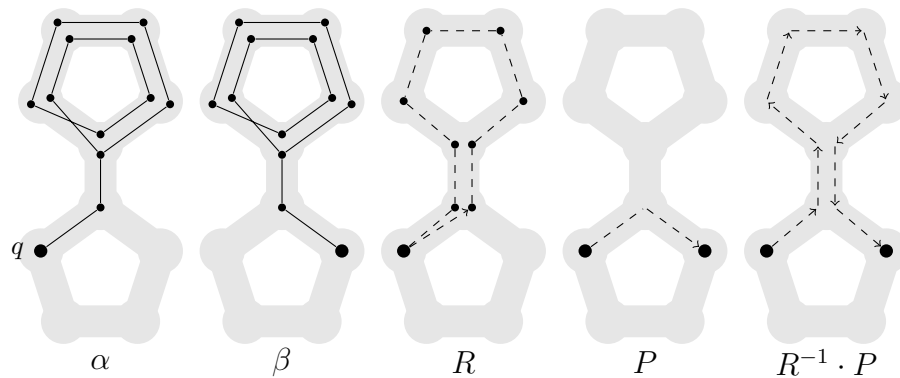
Furthermore, there is an algorithm working in time $\mathcal{O}(|E(G)| \cdot |V(G)| + |E(H)|)$ that given G, H, α, β, q outputs which case holds and outputs Q or R, P in cases 1, 2.

Intuitively, the algorithm finds an appropriate set of at most $E(G)$ closed walks in G and reduce the problem to a system of equations given by Lemma 4 for those closed walks. Consider a closed walk C and what this walk maps to, up to homotopy. If this is non-trivial, that is, $\overline{\alpha(C)} \neq \varepsilon$, then one can show that two topologically valid walks may differ only in the number of times they wind around $\overline{\alpha(C)}$ (or the shortest root R such that $R^n = \overline{\alpha(C)}$), see Figure 8. We solve the system of equations to get one solution P , if it exists. If two closed walks wind around different roots, this implies P is the only solution. If all non-trivial roots are equal R , then case (2.) of the lemma holds. If all roots are trivial, then case (3.) holds. Thus the problem reduces to calculating and comparing roots of the chosen closed walks.

The remaining constraints for realizability don't change the picture much, as stated in the next theorem.

► **Theorem 10.** Let α, β be H -colorings of G and q a vertex of G . Consider the set $\Pi' \subseteq \pi(H)$ of realizable walks for α, β, q . One of the following holds:

0. $\Pi' = \emptyset$.



■ **Figure 8** There is essentially only one closed walk C from q to q in this example, which winds around the root R ($\overline{\alpha(C)} = R^2$). The topologically valid paths are exactly $\{R^n \cdot P \mid n \in \mathbb{Z}\}$ (think about deforming α by pulling q : one can pull it once or more around the top cycle by rotating all of α , but this is impossible for the bottom cycle if we are to reach β).

1. $\Pi' = \{Q\}$ for some $Q \in \pi(H)$.
2. $\Pi' = \{R^n \cdot P \mid n \in \mathbb{Z}\}$ for some $R, P \in \pi(H)$.
3. Π' contains all reduced walks of even length from $\alpha(q)$ to $\beta(q)$.

Furthermore, there is an algorithm working in time $\mathcal{O}(|E(G)|^2 + |E(H)|)$ that given α, β, q outputs in polynomial time which case holds and outputs Q or R, P in cases 1, 2.

The algorithm of Theorem 10 simply runs the algorithm of Lemma 9 to handle the topological condition. To handle the parity condition it discards Q if it has odd length; replaces P with $R \cdot P$ if both R and P are odd; replaces R with R^2 if odd; returns an empty set if P is odd and R is even. To handle the tight closed walk condition, it searches for such walks as described before Corollary 6 and restricts the set to the single walk implied by this corollary, if it applies. To solve H -RECOLORING, that is, to decide whether the set of realizable walks is non-empty, we have to additionally check whether the set of even reduced walks from $\alpha(q)$ to $\beta(q)$ in H is empty. It is empty if and only if H is bipartite and $\alpha(q), \beta(q)$ are on different sides of a bipartition, which is easily checked in time $\mathcal{O}(|E(H)|)$.

► **Corollary 11.** *Let H be a graph (possibly with loops) with the monochromatic neighborhood property. Then H -RECOLORING is in P .*

Shortest recoloring sequences can also be found in polynomial time using Theorem 10, but this requires some more care, mostly in the case we need to find a reduced walk of even length that will correspond to a shortest solution.

► **Theorem 12.** *Let H be a graph (possibly with loops) with the monochromatic neighborhood property. Then SHORTEST H -RECOLORING is in P .*

9 Conclusions and future work

Our result generalizes the algorithm for K_3 -RECOLORING of [8] and recovers many of its features in a more general and perhaps more intuitive setting. When limited to $H = K_3$, we may observe that there is essentially only one possible root R for closed walks in H (a 3-cycle), and all the reduced walks in H with the same endpoints differ only by powers of R . This can be used to show that either no walk is topologically valid (that is, α, β are not homotopic), or all are. In the latter case, it suffices to find tight cycles and either there is

one, immediately implying the vertex walks in the only 3-recoloring sequence; or there is none, in which case any vertex walk for one vertex can be realized in a solution, as long as it has even length. In other words, either no walk is realizable, or $Q = \varepsilon$ is realizable for some (frozen) vertex, or for any vertex q , the walk from $\alpha(q)$ to $\beta(q)$ of length 0 or 2 is realizable. In particular, we don't need to find frozen vertices or do any of the calculations in Lemma 9, it suffices to run $|V(G)| + 1$ times the simpler algorithm of Theorem 7.

We note that none of the proofs used any structural properties of H . If we consider H -recoloring for any graph H , but only allow recoloring a vertex if all of its neighbors have one common color (in other words, a reconfiguration step is allowed only when the homotopy class of the mapping doesn't change), the same results will follow.

An obvious question is how far can our results be extended to more general CSPs – to the asymmetric case, to multiple constraints, to hypergraphs (relations of arbitrary arity)? Is there any connection with the tractable cases of generalized SAT problems? Another question is whether the problems of graph homomorphism reconfiguration exhibit a dichotomy. For which graphs H is H -RECOLORING in P or PSPACE-complete? Some basic reductions are given in the author's master thesis [25]. Finally, it could be interesting to explore the implications of the monochromatic neighborhood property for the whole Hom complex.

Acknowledgments. The author would like to thank Amer E. Mouawad and Naomi Nishimura for helpful discussions and their hospitality. Many thanks to Jarosław Błasiok for sharing his knowledge of algebraic topology, in a remarkably concise way.

References

- 1 Dimitris Achlioptas, Amin Coja-Oghlan, and Federico Ricci-Tersenghi. On the solution-space geometry of random constraint satisfaction problems. *Random Structures and Algorithms*, 38(3):251–268, 2011.
- 2 Eric Babson and Dmitry N. Kozlov. Complexes of graph homomorphisms. *Israel Journal of Mathematics*, 152(1):285–312, 2006.
- 3 Paul Bonsma and Luis Cereceda. Finding paths between graph colourings: PSPACE-completeness and superpolynomial distances. *Theor. Comput. Sci.*, 410(50):5215–5226, 2009.
- 4 Paul Bonsma and Amer E. Mouawad. The complexity of bounded length graph recoloring. *arXiv*, 1404.0337, 2014.
- 5 Graham R Brightwell and Peter Winkler. Graph homomorphisms and long range action. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 63:29–48, 2004.
- 6 Ronald Brown. From groups to groupoids: a brief survey. *Bull. London Math. Soc.*, 19(2):113–134, 1987.
- 7 Ronald Brown. Topology and groupoids. 2006.
- 8 Luis Cereceda, Jan van den Heuvel, and Matthew Johnson. Finding paths between 3-colorings. *Journal of Graph Theory*, 67(1):69–82, 2011.
- 9 Anton Dochtermann. Hom complexes and homotopy theory in the category of graphs. *European J. Combin.*, 30(2):490–509, 2009.
- 10 Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, 1998.

- 11 Parikshit Gopalan, Phokion G. Kolaitis, Elitza N. Maneva, and Christos H. Papadimitriou. The connectivity of boolean satisfiability: Computational and structural dichotomies. *SIAM Journal on Computing*, 38(6):2330–2355, 2009.
- 12 Robert A. Hearn and Erik D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1-2):72–96, 2005.
- 13 Robert A. Hearn and Erik D. Demaine. *Games, puzzles and computation*. A K Peters, 2009.
- 14 Willem Heijltjes and Robin Houston. No proof nets for MLL with units: Proof equivalence in MLL is PSPACE-complete. In *Proceedings of the CSL-LICS 2014 Joint Meeting*, pages 50:1–50:10. ACM, 2014.
- 15 Pavol Hell and Jaroslav Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48(1):92–110, 1990.
- 16 Pavol Hell and Jaroslav Nešetřil. *Graphs and homomorphisms*, volume 28. Oxford University Press, Oxford, 2004.
- 17 Matthew Johnson, Dieter Kratsch, Stefan Kratsch, Viresh Patel, and Daniël Paulusma. Colouring reconfiguration is fixed-parameter tractable. *arXiv*, 1403.6347, 2014.
- 18 Marcin Kamiński, Paul Medvedev, and Martin Milanič. Complexity of independent set reconfigurability problems. *Theoretical Computer Science*, 439:9–15, 2012.
- 19 Jin Ho Kwak and Roman Nedela. Graphs and their coverings. <http://www.savbb.sk/~nedela/graphcov.pdf>, 2007.
- 20 Rahul Mehta. 2048 is (PSPACE) hard, but sometimes easy. *arXiv*, 1408.6315, 2014.
- 21 Amer E. Mouawad, Naomi Nishimura, Vinayak Pathak, and Venkatesh Raman. Shortest reconfiguration paths in the solution space of boolean formulas. *arXiv*, 1404.3801, 2014.
- 22 Jaroslav Nešetřil. Homomorphisms of structures (concepts and highlights). *Physics and Theoretical Computer Science: From Numbers and Languages to (Quantum) Cryptography Security*, 7:295, 2007.
- 23 Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of STOC '78*, pages 216–226, New York, NY, USA, 1978. ACM.
- 24 Konrad W. Schwerdtfeger. A computational trichotomy for connectivity of boolean satisfiability. *arXiv*, 1312.4524, 2013.
- 25 Marcin Wrochna. *Homomorphism reconfiguration in general graphs*, chapter 4. 2014. Master thesis, <http://mimuw.edu.pl/~mw290715/thesis.pdf>.
- 26 Marcin Wrochna. Reconfiguration in bounded bandwidth and treedepth. *arXiv*, 1405.0847, 2014.

Computing Downward Closures for Stacked Counter Automata

Georg Zetsche

AG Concurrency Theory
Fachbereich Informatik
TU Kaiserslautern
zetsche@cs.uni-kl.de

Abstract

The downward closure of a language L of words is the set of all (not necessarily contiguous) subwords of members of L . It is well known that the downward closure of any language is regular. Although the downward closure seems to be a promising abstraction, there are only few language classes for which an automaton for the downward closure is known to be computable.

It is shown here that for stacked counter automata, the downward closure is computable. Stacked counter automata are finite automata with a storage mechanism obtained by *adding blind counters* and *building stacks*. Hence, they generalize pushdown and blind counter automata.

The class of languages accepted by these automata are precisely those in the hierarchy obtained from the context-free languages by alternating two closure operators: imposing semilinear constraints and taking the algebraic extension. The main tool for computing downward closures is the new concept of Parikh annotations. As a second application of Parikh annotations, it is shown that the hierarchy above is strict at every level.

1998 ACM Subject Classification F.4.3 Formal languages

Keywords and phrases abstraction, downward closure, obstruction set, computability

Digital Object Identifier 10.4230/LIPIcs.STACS.2015.743

1 Introduction

In the analysis of systems whose behavior is given by formal languages, it is a fruitful idea to consider abstractions: simpler objects that preserve relevant properties of the language and are amenable to algorithmic examination. A well-known such type of abstraction is the *Parikh image*, which counts the number of occurrences of each letter. For a variety of language classes, the Parikh image of every language is known to be effectively semilinear, which facilitates a range of analysis techniques for formal languages (see [12] for applications).

A promising alternative to Parikh images is the *downward closure* $L\downarrow$, which consists of all (not necessarily contiguous) subwords of members of L . Whereas for many interesting classes of languages the Parikh image is not semilinear in general, the downward closure is regular *for any language* [10], suggesting wide applicability. Moreover, the downward closure encodes properties not visible in the Parikh image: Suppose L describes the behavior of a system that is observed through a lossy channel, meaning that on the way to the observer, arbitrary actions can get lost. Then, $L\downarrow$ is the set of words received by the observer [9]. Hence, given the downward closure as a finite automaton, we can decide whether two systems are equivalent under such observations, and even whether the behavior of one system includes the other. Hence, even if Parikh images are effectively semilinear for a class of languages, computing the downward closure is still an important task. See [2, 3, 13] for further applications.



© Georg Zetsche;

licensed under Creative Commons License CC-BY

32nd Symposium on Theoretical Aspects of Computer Science (STACS 2015).

Editors: Ernst W. Mayr and Nicolas Ollinger; pp. 743–756

Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



However, while there always *exists* a finite automaton for the downward closure, it seems difficult to *compute* them and there are few language classes for which computability has been established. The downward closure is known to be computable for context-free languages and algebraic extensions [5, 16], 0L-systems and context-free FIFO rewriting systems [1], and Petri net languages [9]. It is not computable for reachability sets of lossy channel systems [14] and for Church-Rosser languages [7]. For considerations of complexity, both descriptional and computational, see [3, 8, 11, 15] and the references therein.

It is shown here that downward closures are computable for *stacked counter automata*. These are automata with a finite state control and a storage mechanism obtained by two constructions (of storage mechanisms): One can *build stacks* and *add blind counters*. The former is to construct a new mechanism that stores a stack whose entries are configurations of an old mechanism. One can then manipulate the topmost entry, pop it if empty, or start a new one on top. Adding a blind counter to an old mechanism yields a new mechanism in which the old one and a blind counter (i.e., a counter that can attain negative values and has to be zero in the end of a run) can be used simultaneously.

Stacked counter automata are interesting because among a large class of automata with storage, they are *expressively complete* for those storage mechanisms that guarantee semilinear Parikh images. This is due to the fact that they accept precisely those languages in the hierarchy obtained from the context-free languages by alternating two closure operators: imposing semilinear constraints (with respect to the Parikh image) and taking the algebraic extension. These two closure operators correspond to the constructions of storage mechanisms in stacked counter automata (see Section 3).

The main tool to show the computability of downward closures is the concept of *Parikh annotations*. As another application of this concept, it is shown that the aforementioned hierarchy is strict at every level.

The paper is structured as follows. After Section 2 defines basic concepts and notation, Section 3 introduces the hierarchy of language classes. Section 4 presents Parikh annotations, the main ingredient for the computation of downward closures. The main result is then presented in Section 5, where it is shown that downward closures are computable for stacked counter automata. As a second application of Parikh annotations, it is then shown in Section 6 that the hierarchy defined in Section 3 is strict at every level. Because of space restrictions, most proofs can only be found in the long version of this work [18].

2 Preliminaries

A *monoid* is a set M together with a binary associative operation such that M contains a neutral element. Unless the monoid at hand warrants a different notation, we will denote the neutral element by 1 and the product of $x, y \in M$ by xy . The trivial monoid that contains only the neutral element is denoted by $\mathbf{1}$.

If X is an alphabet, X^* denoted the set of words over X . The empty word is denoted by $\varepsilon \in X^*$. For a symbol $x \in X$ and a word $w \in X^*$, let $|w|_x$ be the number of occurrences of x in w and $|w| = \sum_{x \in X} |w|_x$. For an alphabet X and languages $L, K \subseteq X^*$, the *shuffle product* $L \sqcup K$ is the set of all words $u_0 v_1 u_1 \cdots v_n u_n$ where $u_0, \dots, u_n, v_1, \dots, v_n \in X^*$, $u_0 \cdots u_n \in L$, and $v_1 \cdots v_n \in K$. For a subset $Y \subseteq X$, we define the *projection morphism* $\pi_Y: X^* \rightarrow Y^*$ by $\pi_Y(y) = y$ for $y \in Y$ and $\pi_Y(x) = \varepsilon$ for $x \in X \setminus Y$. By $\mathcal{P}(S)$, we denote the power set of the set S . A *substitution* is a map $\sigma: X \rightarrow \mathcal{P}(Y^*)$ and given $L \subseteq X^*$, we write $\sigma(L)$ for the set of all words $v_1 \cdots v_n$, where $v_i \in \sigma(x_i)$, $1 \leq i \leq n$, for $x_1 \cdots x_n \in L$ and $x_1, \dots, x_n \in X$. If $\sigma(x) \subseteq Y$ for each $x \in X$, we call σ a *letter substitution*.

For words $u, v \in X^*$, we write $u \preceq v$ if $u = u_1 \cdots u_n$ and $v = v_0 u_1 v_1 \cdots u_n v_n$ for some $u_1, \dots, u_n, v_0, \dots, v_n \in X^*$. It is well-known that \preceq is a well-quasi-order on X^* and that therefore the *downward closure* $L\downarrow = \{u \in X^* \mid \exists v \in L: u \preceq v\}$ is regular for any $L \subseteq X^*$ [10].

If X is an alphabet, X^\oplus denotes the set of maps $\alpha: X \rightarrow \mathbb{N}$. The elements of X^\oplus are called *multisets*. Let $\alpha + \beta \in X^\oplus$ be defined by $(\alpha + \beta)(x) = \alpha(x) + \beta(x)$. With this operation, X^\oplus is a monoid. We consider each $x \in X$ to be an element of X^\oplus . For a subset $S \subseteq X^\oplus$, we write S^\oplus for the smallest submonoid of X^\oplus containing S . For $\alpha \in X^\oplus$ and $k \in \mathbb{N}$, we define $(k \cdot \alpha)(x) = k \cdot \alpha(x)$, meaning $k \cdot \alpha \in X^\oplus$. A subset of the form $\mu + F^\oplus$ for $\mu \in X^\oplus$ and a finite $F \subseteq X^\oplus$ is called *linear*. A finite union of linear sets is called *semilinear*. The *Parikh map* is the map $\Psi: X^* \rightarrow X^\oplus$ defined by $\Psi(w)(x) = |w|_x$ for all $w \in X^*$ and $x \in X$. Given a morphism $\varphi: X^\oplus \rightarrow Y^\oplus$ and a word $w \in X^*$, we use $\varphi(w)$ as a shorthand for $\varphi(\Psi(w))$. We lift Ψ to sets in the usual way: $\Psi(L) = \{\Psi(w) \mid w \in L\}$. If $\Psi(L)$ is semilinear, we will also call L itself semilinear.

Let M be a monoid. An *automaton over M* is a tuple $A = (Q, M, E, q_0, F)$, in which

- (i) Q is a finite set of *states*,
- (ii) E is a finite subset of $Q \times M \times Q$ called the set of *edges*,
- (iii) $q_0 \in Q$ is the *initial state*, and
- (iv) $F \subseteq Q$ is the set of *final states*.

We write $(q, m) \rightarrow_A (q', m')$ if there is an edge $(q, r, q') \in E$ such that $m' = mr$. The set *generated* by A is then $S(A) = \{m \in M \mid (q_0, 1) \rightarrow_A^* (f, m) \text{ for some } f \in F\}$.

A *class of languages* is a collection of languages that contains at least one non-empty language. The class of regular languages is denoted by REG. A *finite state transducer* is an automaton over $Y^* \times X^*$ for alphabets X, Y . Relations of the form $S(A)$ for finite state transducers A are called *rational transductions*. For $L \subseteq X^*$ and $T \subseteq Y^* \times X^*$, we write $TL = \{u \in Y^* \mid \exists v \in L: (u, v) \in T\}$. If TF is finite for every finite language F , T is said to be *locally finite*. A class \mathcal{C} of languages is called a *full trio* (*full semi-trio*) if it is closed under (locally finite) rational transductions, i.e. if $TL \in \mathcal{C}$ for every $L \in \mathcal{C}$ and every (locally finite) rational transduction T . A *full semi-AFL* is a union closed full trio.

Stacked counter automata In order to define stacked counter automata, we use the concept of valence automata, which combine a finite state control with a storage mechanism defined by a monoid M . A *valence automaton over M* is an automaton A over $X^* \times M$ for an alphabet X . The *language accepted by A* is then $L(A) = \{w \in X^* \mid (w, 1) \in S(A)\}$. The class of languages accepted by valence automata over M is denoted $\text{VA}(M)$. By choosing suitable monoids M , one can obtain various kinds of automata with storage as valence automata. For example, blind counters, partially blind counters, pushdown storages, and combinations thereof can all be realized by appropriate monoids [19].

If one storage mechanism is realized by a monoid M , then the mechanism that *builds stacks* is realized by the monoid $\mathbb{B} * M$. Here, \mathbb{B} denotes the bicyclic monoid, presented by $\langle a, \bar{a} \mid a\bar{a} = 1 \rangle$, and $*$ denotes the free product of monoids. For readers not familiar with these concepts, it will suffice to know that a configuration of the storage mechanism described by $\mathbb{B} * M$ consists of a sequence $c_0 a c_1 \cdots a c_n$, where c_0, \dots, c_n are configurations of the mechanism realized by M . We interpret this as a stack with the entries c_0, \dots, c_n . One can open a new stack entry on top (by multiplying $a \in \mathbb{B}$), remove the topmost entry if empty (by multiplying $\bar{a} \in \mathbb{B}$) and operate on the topmost entry using the old mechanism (by multiplying elements from M). For example, the monoid \mathbb{B} describes a partially blind counter (i.e. a counter that cannot go below zero and is only tested for zero in the end) and

$\mathbb{B} * \mathbb{B}$ describes a pushdown with two stack symbols. Given a storage mechanism realized by a monoid M , we can *add a blind counter* by using the monoid $M \times \mathbb{Z}$, where \mathbb{Z} denotes the group of integers. We define SC to be the smallest class of monoids with $\mathbf{1} \in \text{SC}$ such that whenever $M \in \text{SC}$, we also have $M \times \mathbb{Z} \in \text{SC}$ and $\mathbb{B} * M \in \text{SC}$. A *stacked counter automaton* is a valence automaton over M for some $M \in \text{SC}$. For more details, see [19]. In Section 3, we will turn to a different description of the languages accepted by stacked counter automata.

3 A hierarchy of language classes

This section introduces a hierarchy of language classes that divides the class of languages accepted by stacked counter automata into levels. This will allow us to apply recursion with respect to these levels. The hierarchy is defined by alternating two operators on language classes, algebraic extensions and semilinear intersections.

Algebraic extensions Let \mathcal{C} be a class of languages. A \mathcal{C} -*grammar* is a quadruple $G = (N, T, P, S)$ where N and T are disjoint alphabets and $S \in N$. The symbols in N and T are called the *nonterminals* and the *terminals*, respectively. P is a finite set of pairs (A, M) with $A \in N$ and $M \subseteq (N \cup T)^*$, $M \in \mathcal{C}$. A pair $(A, M) \in P$ is called a *production of G* and also denoted by $A \rightarrow M$. The set M is the *right-hand side* of the production $A \rightarrow M$.

We write $x \Rightarrow_G y$ if $x = uAv$ and $y = uwv$ for some $u, v, w \in (N \cup T)^*$ and $(A, M) \in P$ with $w \in M$. A word w with $S \Rightarrow_G^* w$ is called a *sentential form* of G and we write $\text{SF}(G)$ for the set of sentential forms of G . The *language generated by G* is $L(G) = \text{SF}(G) \cap T^*$. Languages generated by \mathcal{C} -grammars are called *algebraic over \mathcal{C}* . The class of all languages that are algebraic over \mathcal{C} is called the *algebraic extension* of \mathcal{C} and denoted $\text{Alg}(\mathcal{C})$. We say a language class \mathcal{C} is *algebraically closed* if $\text{Alg}(\mathcal{C}) = \mathcal{C}$. If \mathcal{C} is the class of finite languages, \mathcal{C} -grammars are also called *context-free grammars*.

We will use the operator $\text{Alg}(\cdot)$ to describe the effect of *building stacks* on the accepted languages of valence automata. In [19], it was shown that $\text{VA}(M_0 * M_1) \subseteq \text{Alg}(\text{VA}(M_0) \cup \text{VA}(M_1))$. Here, we complement this by showing that if one of the factors is $\mathbb{B} * \mathbb{B}$, the inclusion becomes an equality. Observe that since $\text{VA}(\mathbb{B} * \mathbb{B})$ is the class of languages accepted by pushdown automata and $\text{Alg}(\text{REG}) = \text{Alg}(\text{VA}(\mathbf{1}))$ is clearly the class of languages generated by context-free grammars, the first statement of the following theorem generalizes the equivalence between pushdown automata and context-free grammars.

► **Theorem 1.** *For every monoid M , $\text{Alg}(\text{VA}(M)) = \text{VA}(\mathbb{B} * \mathbb{B} * M)$.*

Semilinear intersections The second operator on language classes lets us describe the languages in $\text{VA}(M \times \mathbb{Z}^n)$ in terms of those in $\text{VA}(M)$. Consider a language class \mathcal{C} . By $\text{SLI}(\mathcal{C})$, we denote the class of languages of the form $h(L \cap \Psi^{-1}(S))$, where $L \subseteq X^*$ is in \mathcal{C} , the set $S \subseteq X^\oplus$ is semilinear, and $h: X^* \rightarrow Y^*$ is a morphism. We call a language class \mathcal{C} *Presburger closed* if $\text{SLI}(\mathcal{C}) = \mathcal{C}$. Proving the following requires only standard techniques.

► **Proposition 2.** *Let M be a monoid. Then $\text{SLI}(\text{VA}(M)) = \bigcup_{n \geq 0} \text{VA}(M \times \mathbb{Z}^n)$.*

The hierarchy is now obtained by alternating the operators $\text{Alg}(\cdot)$ and $\text{SLI}(\cdot)$. Let F_0 be the class of finite languages and let

$$G_i = \text{Alg}(F_i), \quad F_{i+1} = \text{SLI}(G_i) \quad \text{for each } i \geq 0, \quad F = \bigcup_{i \geq 0} F_i.$$

Then we clearly have the inclusions $F_0 \subseteq G_0 \subseteq F_1 \subseteq G_1 \subseteq \dots$. Furthermore, G_0 is the class of context-free languages, F_1 is the smallest Presburger closed class containing CF, G_1 the algebraic extension of F_1 , etc. In particular, F is the smallest Presburger closed and algebraically closed language class containing the context-free languages.

The following proposition is due to the fact that both $\text{Alg}(\cdot)$ and $\text{SLI}(\cdot)$ preserve (effective) semilinearity. The former has been shown by van Leeuwen [16].

► **Proposition 3.** *The class F is effectively semilinear.*

The work [4] characterized all those storage mechanisms among a large class (namely among those defined by graph products of the bicyclic monoid and the integers) that guarantee semilinear Parikh images. Each of the corresponding language classes was obtained by alternating the operators $\text{Alg}(\cdot)$ and $\text{SLI}(\cdot)$, meaning that all these classes are contained in F . Hence, the following means that stacked counter automata are expressively complete for these storage mechanisms. It follows directly from Theorem 1 and Proposition 2.

► **Theorem 4.** *Stacked counter automata accept precisely the languages in F .*

One might wonder why F_0 is not chosen to be the regular languages. While this would be a natural choice, our recursive algorithm for computing downward closures relies on the following fact. Note that the regular languages are not Presburger closed.

► **Proposition 5.** *For each $i \geq 0$, the class F_i is an effective Presburger closed full semi-trio. Moreover, for each $i \geq 0$, G_i is an effective full semi-AFL.*

4 Parikh annotations

This section introduces Parikh annotations, the key tool in our procedure for computing downward closures. Suppose L is a semilinear language. Then for each $w \in L$, $\Psi(w)$ can be decomposed into a constant vector and a linear combination of period vectors from the semilinear representation of $\Psi(L)$. We call such a decomposition a *Parikh decomposition*. The main purpose of Parikh annotations is to provide transformations of languages that *make reference to Parikh decompositions* without leaving the respective language class. For example, suppose we want to transform a context-free language L into the language L' of all those words $w \in L$ whose Parikh decomposition does not contain a specified period vector. This may not be possible with rational transductions: If $L_\vee = \{a^n b^m \mid m = n \text{ or } m = 2n\}$, then the Parikh image is $(a+b)^\oplus \cup (a+2b)^\oplus$, but a finite state transducer cannot determine whether the input word has a Parikh image in $(a+b)^\oplus$ or in $(a+2b)^\oplus$. Therefore, a Parikh annotation for L is a language K in the same class with additional symbols that allow a finite state transducer (that is applied to K) to access the Parikh decomposition.

► **Definition 6.** *Let $L \subseteq X^*$ be a language and \mathcal{C} be a language class. A Parikh annotation (PA) for L in \mathcal{C} is a tuple $(K, C, P, (P_c)_{c \in C}, \varphi)$, where*

1. C, P are alphabets such that X, C, P are pairwise disjoint,
2. $K \subseteq C(X \cup P)^*$ is in \mathcal{C} ,
3. φ is a morphism $\varphi: (C \cup P)^\oplus \rightarrow X^\oplus$,
4. P_c is a subset $P_c \subseteq P$ for each $c \in C$,

such that

- (i) $\pi_X(K) = L$ (the projection property),
- (ii) $\varphi(\pi_{C \cup P}(w)) = \Psi(\pi_X(w))$ for each $w \in K$ (the counting property), and
- (iii) $\Psi(\pi_{C \cup P}(K)) = \bigcup_{c \in C} c + P_c^\oplus$ (the commutative projection property).

A Parikh annotation describes for each w in L one or more Parikh decompositions of $\Psi(w)$. The symbols in C represent constant vectors and symbols in P represent period vectors. The symbols in $P_c \subseteq P$ correspond to those that can be added to the constant vector corresponding to $c \in C$. Furthermore, for each $x \in C \cup P$, $\varphi(x)$ is the vector represented by x . The projection property states that removing the symbols in $C \cup P$ from words in K yields L . The commutative projection property requires that after $c \in C$ only symbols representing periods in P_c are allowed and that all their combinations occur. Finally, the counting property says that the additional symbols in $C \cup P$ indeed describe a Parikh decomposition of $\Psi(\pi_X(w))$. Of course, only semilinear languages can have a Parikh annotations.

► **Example 7.** Let $X = \{a, b, c, d\}$ and consider the regular set $L = (ab)^*(ca^* \cup db^*)$. For $K = e(pab)^*c(qa)^* \cup f(rab)^*d(sb)^*$, $P = \{p, q, r, s\}$, $C = \{e, f\}$, $P_e = \{p, q\}$, $P_f = \{r, s\}$, and $\varphi: (C \cup P)^\oplus \rightarrow X^\oplus$ with $e \mapsto c$, $f \mapsto d$, $p \mapsto a + b$, $q \mapsto a$, $r \mapsto a + b$, and $s \mapsto b$, the tuple $(K, C, P, (P_g)_{g \in C}, \varphi)$ is a Parikh annotation for L in REG.

In a Parikh annotation, for each $cw \in K$ and $\mu \in P_c^\oplus$, we can find a word cw' in K such that $\Psi(\pi_{C \cup P}(cw')) = \Psi(\pi_{C \cup P}(cw)) + \mu$. In particular, this implies the equality $\Psi(\pi_X(cw')) = \Psi(\pi_X(cw)) + \varphi(\mu)$. In our applications, we will need a further guarantee that provides such words, but with additional information on their structure. Such a guarantee is granted by Parikh annotations with insertion marker. Suppose $\diamond \notin X$ and $u \in (X \cup \{\diamond\})^*$ with $u = u_0 \diamond u_1 \cdots \diamond u_n$ for $u_0, \dots, u_n \in X^*$. Then we write $u \preceq_\diamond v$ if $v = u_0 v_1 u_1 \cdots v_n u_n$ for some $v_1, \dots, v_n \in X^*$.

► **Definition 8.** Let $L \subseteq X^*$ be a language and \mathcal{C} be a language class. A Parikh annotation with insertion marker (PAIM) for L in \mathcal{C} is a tuple $(K, C, P, (P_c)_{c \in C}, \varphi, \diamond)$ such that:

- (i) $\diamond \notin X$ and $K \subseteq C(X \cup P \cup \{\diamond\})^*$ is in \mathcal{C} ,
- (ii) $(\pi_{C \cup X \cup P}(K), C, P, (P_c)_{c \in C}, \varphi)$ is a Parikh annotation for L in \mathcal{C} ,
- (iii) there is a $k \in \mathbb{N}$ such that every $w \in K$ satisfies $|w|_\diamond \leq k$ (boundedness), and
- (iv) for each $cw \in K$ and $\mu \in P_c^\oplus$, there is a $w' \in L$ with $\pi_{X \cup \diamond}(cw) \preceq_\diamond w'$ and with $\Psi(w') = \Psi(\pi_X(cw)) + \varphi(\mu)$. This property is called the insertion property.

If $|C| = 1$, then the PAIM is called linear and we also write $(K, c, P_c, \varphi, \diamond)$ for the PAIM, where $C = \{c\}$.

In other words, in a PAIM, each $v \in L$ has an annotation $cw \in K$ in which a bounded number of positions is marked such that for each $\mu \in P_c^\oplus$, we can find a $v' \in L$ with $\Psi(v') = \Psi(v) + \varphi(\mu)$ such that v' is obtained from v by inserting words in corresponding positions in v . In particular, this guarantees $v \preceq v'$.

► **Example 9.** Let L and $(K, C, P, (P_c)_{c \in C}, \varphi)$ be as in Example 7. Furthermore, let $K' = e \diamond (pab)^*c \diamond (qa)^* \cup f \diamond (rab)^*d \diamond (sb)^*$. Then $(K', C, P, (P_c)_{c \in C}, \varphi, \diamond)$ is a PAIM for L in REG. Indeed, every word in K' has at most two occurrences of \diamond . Moreover, if $ew = e \diamond (pab)^m c \diamond (qa)^n \in K'$ and $\mu \in P_e^\oplus$, $\mu = k \cdot p + \ell \cdot q$, then $w' = (ab)^{k+m} c a^{\ell+n} \in L$ satisfies $\pi_{X \cup \diamond}(ew) = \diamond (ab)^m c \diamond a^n \preceq_\diamond (ab)^k (ab)^m c a^\ell a^n = w'$ and clearly $\Psi(\pi_X(w')) = \Psi(\pi_X(ew)) + \varphi(\mu)$ (and similarly for words $fw \in K'$).

The main result of this section is that there is an algorithm that, given a language $L \in \mathbb{F}_i$ or $L \in \mathbb{G}_i$, constructs a PAIM for L in \mathbb{F}_i or \mathbb{G}_i , respectively.

► **Theorem 10.** Given $i \in \mathbb{N}$ and L in \mathbb{F}_i (\mathbb{G}_i), one can construct a PAIM for L in \mathbb{F}_i (\mathbb{G}_i).

Outline of the proof The rest of this section is devoted to the proof of Theorem 10. The construction of PAIM proceeds recursively with respect to the level of our hierarchy. This means, we show that if PAIM can be constructed for F_i , then we can compute them for G_i (Lemma 17) and if they can be constructed for G_i , then they can be computed for F_{i+1} (Lemma 18). While the latter can be done with a direct construction, the former requires a series of involved steps:

- The general idea is to use recursion with respect to the number of nonterminals: Given a F_i -grammar for $L \in G_i$, we present L in terms of languages whose grammars use fewer nonterminals. This presentation is done via substitutions and by using grammars with one nonterminal. The idea of presenting a language in $\text{Alg}(\mathcal{C})$ using one-nonterminal grammars and substitutions follows van Leeuwen's proof of Parikh's theorem [16].
- We construct PAIM for languages generated by one-nonterminal grammars where we are given PAIM for the right-hand-sides (Lemma 16).
- We construct PAIM for languages $\sigma(L)$, where σ is a substitution, a PAIM is given for L and for each $\sigma(x)$ (Lemma 15). This construction is again divided into the case where σ is a letter substitution (i.e., one in which each symbol is mapped to a set of letters) and the general case. Since the case of letter substitutions constitutes the conceptually most involved step, part of its proof is contained in this extended abstract (Proposition 13).

Maybe surprisingly, the most conceptually involved step in the construction of PAIM lies within obtaining a Parikh annotation for $\sigma(L)$ in $\text{Alg}(\mathcal{C})$, where σ is a letter substitution and a PAIM for $L \subseteq X^*$ in $\text{Alg}(\mathcal{C})$ is given. This is due to the fact that one has to substitute the symbols in X consistently with the symbols in $C \cup P$; more precisely, one has to maintain the agreement between $\varphi(\pi_{C \cup P}(\cdot))$ and $\Psi(\pi_X(\cdot))$.

In order to exploit the fact that this agreement exists in the first place, we use the following simple yet very useful lemma. It states that for a morphism ψ into a group, the only way a grammar G can guarantee $L(G) \subseteq \psi^{-1}(h)$ is by encoding into each nonterminal A the value $\psi(u)$ for the words u that A derives. The G -compatible extension of ψ reconstructs this value for each nonterminal. Let $G = (N, T, P, S)$ be a \mathcal{C} -grammar and M be a monoid. A morphism $\psi: (N \cup T)^* \rightarrow M$ is called G -compatible if $u \Rightarrow_G^* v$ implies $\psi(u) = \psi(v)$ for $u, v \in (N \cup T)^*$. Moreover, we call G reduced if for each $A \in N$, we have $A \Rightarrow_G^* w$ for some $w \in T^*$ and $S \Rightarrow_G^* uAv$ for some $u, v \in (N \cup T)^*$.

► **Lemma 11.** *Let H be a group, $\psi: T^* \rightarrow H$ be a morphism, and $G = (N, T, P, S)$ be a reduced \mathcal{C} -grammar with $L(G) \subseteq \psi^{-1}(h)$ for some $h \in H$. Then ψ has a unique G -compatible extension $\hat{\psi}: (N \cup T)^* \rightarrow H$. If $H = \mathbb{Z}^n$ and $\mathcal{C} = F_i$, $\hat{\psi}$ can be computed.*

We will essentially apply Lemma 11 by regarding X^\oplus as a subset of \mathbb{Z}^n and defining $\psi: (C \cup P \cup X)^* \rightarrow \mathbb{Z}^n$ as the morphism with $\psi(w) = \Psi(\pi_X(w)) - \varphi(\pi_{C \cup P}(w))$. In the case that G generates the corresponding Parikh annotation, the counting property implies that $L(G) \subseteq \psi^{-1}(0)$. The lemma then states that each nonterminal in G encodes the imbalance between $\Psi(\pi_X(\cdot))$ and $\varphi(\pi_{C \cup P}(\cdot))$ on the words it generates.

We continue with the problem of replacing $C \cup P$ and X consistently. For constructing the PAIM for $\sigma(L)$, it is easy to see that it suffices to consider the case where $\sigma(a) = \{a, b\}$ for some $a \in X$ and $\sigma(x) = \{x\}$ for $x \in X \setminus \{a\}$. In order to simplify the setting and exploit the symmetry of the roles played by $C \cup P$ and X , we consider a slightly more general situation. There is an alphabet $X = X_0 \uplus X_1$, morphisms $\gamma_i: X_i^* \rightarrow \mathbb{N}$, $i = 0, 1$, and a language $L \subseteq X^*$, $L \in \text{Alg}(F_i)$ with $\gamma_0(\pi_{X_0}(w)) = \gamma_1(\pi_{X_1}(w))$ for every $w \in L$. Roughly speaking, X_1 will later play the role of $C \cup P$ and X_0 will play the role of X . Then, $\gamma_0(w)$

will be the number of a 's in w and $\gamma_1(w)$ will be the number of a 's represented by symbols from $C \cup P$ in w . Therefore, we wish to construct a language L' in $\text{Alg}(\mathbf{F}_i)$ such that each word in L' is obtained from a word in L as follows. We substitute each occurrence of $x \in X_i$ by one of $\gamma_i(x) + 1$ many symbols y in an alphabet Y_i , each of which will be assigned a value $0 \leq \eta_i(y) \leq \gamma_i(x)$. Here, we want to guarantee that in every resulting word $w \in (Y_0 \cup Y_1)^*$, we have $\eta_0(\pi_{Y_0}(w)) = \eta_1(\pi_{Y_1}(w))$, meaning that the symbols in X_0 and in X_1 are replaced *consistently*. Formally, we have

$$Y_i = \{(x, j) \mid x \in X_i, 0 \leq j \leq \gamma_i(x)\}, \quad i = 0, 1, \quad Y = Y_0 \cup Y_1, \quad (1)$$

and the morphisms

$$\begin{aligned} h_i: Y_i^* &\longrightarrow X_i^*, & h: Y^* &\longrightarrow X^*, & \eta_i: Y_i^* &\longrightarrow \mathbb{N}, \\ (x, j) &\longmapsto x, & (x, j) &\longmapsto x, & (x, j) &\longmapsto j, \end{aligned} \quad (2)$$

and we want to construct a subset of $\hat{L} = \{w \in h^{-1}(L) \mid \eta_0(\pi_{Y_0}(w)) = \eta_1(\pi_{Y_1}(w))\}$ in $\text{Alg}(\mathbf{F}_i)$. Observe that we cannot hope to find \hat{L} itself in $\text{Alg}(\mathbf{F}_i)$ in general. Take, for example, the context-free language $E = \{a^n b^n \mid n \geq 0\}$ and $X_0 = \{a\}$, $X_1 = \{b\}$, $\gamma_0(a) = 1$, $\gamma_1(b) = 1$. Then the language \hat{E} would not be context-free. However, the language $E' = \{wg(w)^R \mid w \in \{(a, 0), (a, 1)\}^*\}$, where g is the morphism with $(a, j) \mapsto (b, j)$ for $j = 0, 1$, is context-free. Although it is only a proper subset of \hat{E} , it is large enough to satisfy $\pi_{Y_i}(E') = \pi_{Y_i}(\hat{E}) = \pi_{Y_i}(h^{-1}(E))$ for $i = 0, 1$. We will see that in order to construct Parikh annotations, it suffices to use such under-approximations of \hat{L} .

Derivation trees and matchings In this work, by an X -labeled tree, we mean a finite ordered unranked tree in which each node carries a label from $X \cup \{\varepsilon\}$ for an alphabet X . For each node, there is a linear order on the set of its children. For each node x , we write $c(x) \in X^*$ for the word obtained by reading the labels of x 's children in this order. Furthermore, $\text{yield}(x) \in X^*$ denotes the word obtained by reading leaf labels below the node x according to the linear order induced on the leaves. Moreover, if r is the root of t , we also write $\text{yield}(t)$ for $\text{yield}(r)$. The *height* of a tree is the maximal length of a path from the root to a leaf, i.e. a tree consisting of a single node has height 0. A *subtree* of a tree t is the tree consisting of all nodes below some node x of t . If x is a child of t 's root, the subtree is a *direct subtree*.

Let $G = (N, T, P, S)$ be a \mathcal{C} -grammar. A *partial derivation tree (for G)* is an $(N \cup T)$ -labeled tree t in which

- (i) each inner node x has a label $A \in N$ and there is some $A \rightarrow L$ in P with $c(x) \in L$, and
- (ii) no ε -labeled node has a sibling.

If, in addition, the root is labeled S and every leaf is labeled by $T \cup \{\varepsilon\}$, it is called a *derivation tree for G* .

Let t be a tree whose leaves are $X \cup \{\varepsilon\}$ -labeled. Let L_i denote the set of X_i -labeled leaves of t . An *arrow collection for t* is a finite set A together with maps $\nu_i: A \rightarrow L_i$ for $i = 0, 1$. Hence, A can be thought of as a set of arrows pointing from X_0 -labeled leaves to X_1 -labeled leaves. We say an arrow $a \in A$ is *incident* to a leaf ℓ if $\nu_0(a) = \ell$ or $\nu_1(a) = \ell$. If ℓ is a leaf, then $d_A(\ell)$ denotes the number of arrows incident to ℓ . More generally, for a subtree s of t , $d_A(s)$ denotes the number of arrows incident to some leaf in s and some leaf outside of s . A is called a *k -matching* if

- (i) each leaf labeled $x \in X_i$ has precisely $\gamma_i(x)$ incident arrows, and
- (ii) $d_A(s) \leq k$ for every subtree s of t .

The following lemma applies Lemma 11. The latter implies that for nodes x of a derivation tree, the balance $\gamma_0(\pi_{X_0}(\text{yield}(x))) - \gamma_1(\pi_{X_1}(\text{yield}(x)))$ is bounded. This can be used to construct k -matchings in a bottom-up manner.

► **Lemma 12.** *Let $X = X_0 \uplus X_1$ and $\gamma_i: X_i^* \rightarrow \mathbb{N}$ for $i = 0, 1$ be a morphism. Let G be a reduced F_i -grammar with $L(G) \subseteq X^*$ and $\gamma_0(\pi_{X_0}(w)) = \gamma_1(\pi_{X_1}(w))$ for every $w \in L(G)$. Then one can compute a bound k such that each derivation tree of G admits a k -matching.*

We are now ready to construct the approximations necessary for obtaining PAIM.

► **Proposition 13 (Consistent substitution).** *Let $X = X_0 \uplus X_1$ and $\gamma_i: X_i^\oplus \rightarrow \mathbb{N}$ for $i = 0, 1$ be a morphism. Let $L \in \text{Alg}(F_i)$, $L \subseteq X^*$, be a language with $\gamma_0(\pi_{X_0}(w)) = \gamma_1(\pi_{X_1}(w))$ for every $w \in L$. Furthermore, let Y_i, h_i, η_i for $i = 0, 1$ and Y, h be defined as in Eq. (1) and Eq. (2). Moreover, let L be given by a reduced grammar. Then one can construct a language $L' \in \text{Alg}(F_i)$, $L' \subseteq Y^*$, with*

- (i) $L' \subseteq h^{-1}(L)$,
- (ii) $\pi_{Y_i}(L') = \pi_{Y_i}(h^{-1}(L))$ for $i = 0, 1$,
- (iii) $\eta_0(\pi_{Y_0}(w)) = \eta_1(\pi_{Y_1}(w))$ for every $w \in L'$.

Proof. Let $G_0 = (N, X, P_0, S)$ be a reduced F_i -grammar with $L(G_0) = L$. Let $G_1 = (N, Y, P_1, S)$ be the grammar with $P_1 = \{A \rightarrow \hat{h}^{-1}(K) \mid A \rightarrow K \in P_0\}$, where $\hat{h}: (N \cup Y)^* \rightarrow (N \cup X)^*$ is the extension of h that fixes N . With $L_1 = L(G_1)$, we have $L_1 = h^{-1}(L)$.

According to Lemma 12, we can find a $k \in \mathbb{N}$ such that every derivation tree of G_0 admits a k -matching. With this, let $F = \{z \in \mathbb{Z} \mid |z| \leq k\}$, $N_2 = N \times F$, and η be the morphism $\eta: (N_2 \cup Y)^* \rightarrow \mathbb{Z}$ with $(A, z) \mapsto z$ for $(A, z) \in N_2$, and $y \mapsto \eta_0(\pi_{Y_0}(y)) - \eta_1(\pi_{Y_1}(y))$ for $y \in Y$. Moreover, let $g: (N_2 \cup Y)^* \rightarrow (N \cup Y)^*$ be the morphism with $g((A, z)) = A$ for $(A, z) \in N_2$ and $g(y) = y$ for $y \in Y$. This allows us to define the set of productions $P_2 = \{(A, z) \rightarrow g^{-1}(L) \cap \eta^{-1}(z) \mid A \rightarrow K \in P_1\}$. Note that since F_i is an effective Presburger closed full semi-trio, we have effectively $g^{-1}(K) \cap \eta^{-1}(z) \in F_i$ for $K \in F_i$. Finally, let G_2 be the grammar $G_2 = (N_2, Y, P_2, (S, 0))$. We claim that $L' = L(G_2)$ has the desired properties. Since $L' \subseteq L_1 = h^{-1}(L)$, Item 1 is satisfied. Furthermore, the construction guarantees that for a production $(A, z) \rightarrow w$ in G_2 , we have $\eta(w) = z$. In particular, every $w \in Y^*$ with $(S, 0) \Rightarrow_{G_2}^* w$ exhibits $\eta_0(\pi_{Y_0}(w)) - \eta_1(\pi_{Y_1}(w)) = \eta(w) = 0$. Thus, we have shown Item 3.

Note that the inclusion “ \subseteq ” of Item 2 follows from Item 1. In order to prove “ \supseteq ”, we shall use k -matchings in G_0 to construct derivations in G_2 . See Fig. 1 for an example of the following construction of derivation trees. Let $w \in h^{-1}(L) = L(G_1)$ and consider a derivation tree t for w in G_1 . Let \bar{t} be the $(N \cup X)$ -tree obtained from t by replacing each leaf label $y \in Y$ by $h(y)$. Then \bar{t} is a derivation tree of G_0 and admits a k -matching \bar{A} . Since \bar{t} and t are isomorphic up to labels, we can obtain a corresponding arrow collection A in t (see Fig. 1a).

Let L_i denote the set of Y_i -labeled leaves of t for $i = 0, 1$. Now fix $i \in \{0, 1\}$. We choose a subset $A' \subseteq A$ as follows. Since \bar{A} is a k -matching, each leaf $\ell \in L_i$ of t has precisely $\gamma_i(h(\lambda(\ell))) \geq \eta_i(\lambda(\ell))$ incident arrows in A . For each such $\ell \in L_i$, we include some arbitrary choice of $\eta_i(\lambda(\ell))$ arrows in A' (see Fig. 1b). The tree t' is obtained from t by changing the label of each leaf $\ell \in L_{1-i}$ from (x, j) to (x, j') , where j' is the number of arrows in A' incident to ℓ (see Fig. 1c). Note that since we only change labels of leaves in L_{1-i} , we have $\pi_{Y_i}(\text{yield}(t')) = \pi_{Y_i}(\text{yield}(t)) = \pi_{Y_i}(w)$.

For every subtree s of t' , we define $\beta(s) = \eta_0(\pi_{Y_0}(\text{yield}(s))) - \eta_1(\pi_{Y_1}(\text{yield}(s)))$. By construction of A' , each leaf $\ell \in L_j$ has precisely $\eta_j(\lambda(\ell))$ incident arrows in A' for $j = 0, 1$.

The basic idea for the case of general substitutions is to replace each x by a PAIM for $\sigma(x)$. Here, Lemma 14 allows us to assume that the PAIM for each $\sigma(x)$ is linear. However, we have to make sure that the number of occurrences of \diamond remains bounded.

► **Lemma 15** (Substitutions). *Let $L \subseteq X^*$ in \mathbb{G}_i and σ be a \mathbb{G}_i -substitution. Given a PAIM in \mathbb{G}_i for L and for each $\sigma(x)$, $x \in X$, one can construct a PAIM for $\sigma(L)$ in \mathbb{G}_i .*

The next step is to construct PAIM for languages $\mathbb{L}(G)$, where G has just one nonterminal S and PAIM are given for the right-hand-sides. Here, it suffices to obtain a PAIM for $\text{SF}(G)$ in the case that S occurs in every word on the right hand side: Then $\mathbb{L}(G)$ can be obtained from $\text{SF}(G)$ using a substitution. Applying $S \rightarrow R$ then means that for some $w \in R$, $\Psi(w) - S$ is added to the Parikh image of the sentential form. Therefore, computing a PAIM for $\text{SF}(G)$ is akin to computing a semilinear representation for S^\oplus , where S is semilinear.

► **Lemma 16** (One nonterminal). *Let G be a \mathbb{G}_i -grammar with one nonterminal. Furthermore, suppose PAIM in \mathbb{G}_i are given for the right-hand-sides in G . Then we can construct a PAIM for $\mathbb{L}(G)$ in \mathbb{G}_i .*

Using Lemmas 15 and 16, we can now construct PAIM recursively with respect to the number of nonterminals in G .

► **Lemma 17** (PAIM for algebraic extensions). *Given $i \in \mathbb{N}$ and an \mathbb{F}_i -grammar G , along with a PAIM in \mathbb{F}_i for each right hand side, one can construct a PAIM for $\mathbb{L}(G)$ in \mathbb{G}_i .*

The last step is to compute PAIM for languages in $\text{SLI}(\mathbb{G}_i)$. Then, Theorem 10 follows.

► **Lemma 18** (PAIM for semilinear intersections). *Given $i \in \mathbb{N}$, a language $L \subseteq X^*$ in \mathbb{G}_i , a semilinear set $S \subseteq X^\oplus$, and a morphism $h: X^* \rightarrow Y^*$, along with a PAIM in \mathbb{G}_i for L , one can construct a PAIM for $h(L \cap \Psi^{-1}(S))$ in $\text{SLI}(\mathbb{G}_i)$.*

5 Computing downward closures

The procedure for computing downward closures works recursively with respect to the hierarchy $\mathbb{F}_0 \subseteq \mathbb{G}_0 \subseteq \dots$. For languages in $\mathbb{G}_i = \text{Alg}(\mathbb{F}_i)$, we use an idea by van Leeuwen [17], who proved that downward closures are computable for $\text{Alg}(\mathcal{C})$ if and only if this is the case for \mathcal{C} . This means we can compute downward closures for \mathbb{G}_i if we can compute them for \mathbb{F}_i . For the latter, we use Lemma 19, which is based on the following idea. Using a PAIM for L in \mathbb{G}_i , one constructs a language $L' \supseteq L \cap \Psi^{-1}(S)$ in which every word admits insertions that yield a word in $L \cap \Psi^{-1}(S)$, meaning that $L' \downarrow = (L \cap \Psi^{-1}(S)) \downarrow$. Here, L' is obtained from the PAIM using a rational transduction, which implies $L' \in \mathbb{G}_i$.

► **Lemma 19.** *Given $i \in \mathbb{N}$, a language $L \subseteq X^*$ in \mathbb{G}_i , and a semilinear set $S \subseteq X^\oplus$, one can compute a language $L' \in \mathbb{G}_i$ with $L' \downarrow = (L \cap \Psi^{-1}(S)) \downarrow$.*

Proof. We call $\alpha \in X^\oplus$ a *submultiset* of $\beta \in X^\oplus$ if $\alpha(x) \leq \beta(x)$ for each $x \in X$. In analogy with words, we write $T \downarrow$ for the set of all submultisets of elements of T for $T \subseteq X^\oplus$. We use Theorem 10 to construct a PAIM $(K, C, P, (P_c)_{c \in C}, \varphi, \diamond)$ for L in \mathbb{G}_i . For each $c \in C$, consider the set $S_c = \{\mu \in P_c^\oplus \mid \varphi(c + \mu) \in S\}$. Since \leq is a well-quasi-ordering on X^\oplus [6], membership in $S_c \downarrow$ can be characterized by a finite set of forbidden submultisets, which is Presburger definable and thus computable. Therefore, the language $\Psi^{-1}(S_c \downarrow)$ is effectively regular. Hence, the language

$$L' = \{\pi_X(cv) \mid c \in C, cv \in K, \pi_{P_c}(v) \in \Psi^{-1}(S_c \downarrow)\}.$$

effectively belongs to G_i , since G_i is an effective full semi-AFL. We claim that $L \cap \Psi^{-1}(S) \subseteq L' \subseteq (L \cap \Psi^{-1}(S))\downarrow$. The latter clearly implies $L\downarrow = (L \cap \Psi^{-1}(S))\downarrow$.

The counting property of the PAIM entails the inclusion $L \cap \Psi^{-1}(S) \subseteq L'$. In order to show $L' \subseteq (L \cap \Psi^{-1}(S))\downarrow$, suppose $w \in L'$. Then there is a $cv \in K$ with $w = \pi_X(cv)$ and $\pi_{P_c}(v) \in \Psi^{-1}(S_c\downarrow)$. This means there is a $\nu \in P_c^\oplus$ with $\Psi(\pi_{P_c}(v)) + \nu \in S_c$. The insertion property of $(K, C, P, (P_c)_{c \in C}, \varphi, \diamond)$ allows us to find a word $v' \in L$ such that

$$\Psi(v') = \Psi(\pi_X(cv)) + \varphi(\nu), \quad \pi_{X \cup \{\diamond\}}(cv) \preceq_\diamond v'. \quad (6)$$

By definition of S_c , the first part of Eq. (6) implies that $\Psi(v') \in S$. The second part of Eq. (6) means in particular that $w = \pi_X(cv) \preceq v'$. Thus, we have $w \preceq v' \in L \cap \Psi^{-1}(S)$. ◀

► **Theorem 20.** *Given a language L in F , one can compute a finite automaton for $L\downarrow$.*

Proof. We perform the computation recursively with respect to the level of the hierarchy.

- If $L \in F_0$, then L is finite and we can clearly compute $L\downarrow$.
- If $L \in F_i$ with $i \geq 1$, then $L = h(L' \cap \Psi^{-1}(S))$ for some $L' \subseteq X^*$ in G_{i-1} , a semilinear set $S \subseteq X^\oplus$, and a morphism h . Since $h(M)\downarrow = h(M\downarrow)\downarrow$ for any $M \subseteq X^*$, it suffices to describe how to compute $(L' \cap \Psi^{-1}(S))\downarrow$. Using Lemma 19, we construct a language $L'' \in G_{i-1}$ with $L''\downarrow = (L' \cap \Psi^{-1}(S))\downarrow$ and then recursively compute $L''\downarrow$.
- If $L \in G_i$, then L is given by an F_i -grammar G . Using recursion, we compute the downward closure of each right-hand-side of G . We obtain a new REG-grammar G' by replacing each right-hand-side in G with its downward closure. Then $L(G')\downarrow = L\downarrow$. Since we can construct a context-free grammar for $L(G')$, we can compute $L(G')\downarrow$ using the available algorithms by van Leeuwen [16] or Courcelle [5].

◀

6 Strictness of the hierarchy

In this section, we present another application of Parikh annotations. Using PAIM, one can show that the inclusions $F_0 \subseteq G_0 \subseteq F_1 \subseteq G_1 \subseteq \dots$ in the hierarchy are, in fact, all strict. It is of course easy to see that $F_0 \subsetneq G_0 \subsetneq F_1$, since F_0 contains only finite sets and F_1 contains, for example, $\{a^n b^n c^n \mid n \geq 0\}$. In order to prove strictness at higher levels, we present two transformations: The first turns a language from $F_i \setminus G_{i-1}$ into one in $G_i \setminus F_i$ (Proposition 21) and the second turns one from $G_i \setminus F_i$ into one in $F_{i+1} \setminus G_i$ (Proposition 25).

The essential idea of the next proposition is as follows. For the sake of simplicity, assume $(L\#)^* = L' \cap \Psi^{-1}(S)$ for $L' \in \mathcal{C}$, $L' \subseteq (X \cup \{\#\})^*$. Consider a PAIM $(K', C, P, (P_c)_{c \in C}, \varphi, \diamond)$ for L' in \mathcal{C} . Similar to Lemma 19, we obtain from K' a language $\hat{L} \subseteq (X \cup \{\#, \diamond\})^*$ in \mathcal{C} such that every member of \hat{L} admits an insertion at \diamond that yields a word from $(L\#)^* = L' \cap \Psi^{-1}(S)$. Using a rational transduction, we can then pick all words that appear between two $\#$ in some member of \hat{L} and contain no \diamond . Since there is a bound on the number of \diamond in K' (and hence in \hat{L}), every word from L has to occur in this way. On the other hand, since inserting at \diamond yields a word in $(L\#)^*$, every such word without \diamond must be in L .

► **Proposition 21.** *Let \mathcal{C} be a full trio such that every language in \mathcal{C} has a PAIM in \mathcal{C} . Moreover, let X be an alphabet with $\# \notin X$. If $(L\#)^* \in \text{SLI}(\mathcal{C})$ for $L \subseteq X^*$, then $L \in \mathcal{C}$.*

Using induction on the structure of a rational expression, it is not hard to show that we can construct PAIM for regular languages. This means Propositions 2 and 21 imply the following, which might be of independent interest.

► **Corollary 22.** *Let $L \subseteq X^*$, $\# \notin X$, and $(L\#)^* \in \text{VA}(\mathbb{Z}^n)$. Then L is regular.*

In order to prove Proposition 25, we need a new concept. A bursting grammar is one in which essentially (meaning: aside from a subsequent replacement by terminal words of bounded length) the whole word is generated in a single application of a production.

► **Definition 23.** *Let \mathcal{C} be a language class and $k \in \mathbb{N}$. A \mathcal{C} -grammar G is called k -bursting if for every derivation tree t for G and every node x of t we have: $|\text{yield}(x)| > k$ implies $\text{yield}(x) = \text{yield}(t)$. A grammar is said to be bursting if it is k -bursting for some $k \in \mathbb{N}$.*

► **Lemma 24.** *If \mathcal{C} is a union closed full semi-trio and G a bursting \mathcal{C} -grammar, then $L(G) \in \mathcal{C}$.*

The essential idea for Proposition 25 is the following. We construct a \mathcal{C} -grammar G' for L by removing from a \mathcal{C} -grammar G for $M = (L \sqcup \{a^n b^n c^n \mid n \geq 0\}) \cap a^*(bX)^*c^*$ all terminals a, b, c . Using Lemma 11, one can then show that G' is bursting.

► **Proposition 25.** *Let \mathcal{C} be a union closed full semi-trio and let $a, b, c \notin X$ and $L \subseteq X^*$. If $L \sqcup \{a^n b^n c^n \mid n \geq 0\} \in \text{Alg}(\mathcal{C})$, then $L \in \mathcal{C}$.*

► **Theorem 26.** *For $i \in \mathbb{N}$, define the alphabets $X_0 = \emptyset$, $Y_i = X_i \cup \{\#\}_i$, $X_{i+1} = Y_i \cup \{a_{i+1}, b_{i+1}, c_{i+1}\}$. Moreover, define $U_i \subseteq X_i^*$ and $V_i \subseteq Y_i^*$ as $U_0 = \{\varepsilon\}$, $V_i = (U_i \#_i)^*$, and $U_{i+1} = V_i \sqcup \{a_{i+1}^n b_{i+1}^n c_{i+1}^n \mid n \geq 0\}$ for $i \geq 0$. Then $V_i \in \mathbf{G}_i \setminus \mathbf{F}_i$ and $U_{i+1} \in \mathbf{F}_{i+1} \setminus \mathbf{G}_i$.*

References

- 1 Parosh Aziz Abdulla, Luc Boasson, and Ahmed Bouajjani. Effective lossy queue languages. In *Proc. of ICALP 2001*, volume 2076 of *LNCS*, pages 639–651. Springer, 2001.
- 2 Mohamed Faouzi Atig, Ahmed Bouajjani, and Shaz Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. In *Proc. of TACAS 2009*, volume 5505 of *LNCS*, pages 107–123. Springer, 2009.
- 3 Georg Bachmeier, Michael Luttenberger, and Maximilian Schlund. Finite automata for the sub- and superword closure of cfls: Descriptive and computational complexity, 2015. To appear in: *Proceedings of LATA 2015*.
- 4 P. Buckheister and Georg Zetsche. Semilinearity and context-freeness of languages accepted by valence automata. In *Proc. of MFCS 2013*, volume 8087 of *LNCS*, pages 231–242. Springer, 2013.
- 5 Bruno Courcelle. On constructing obstruction sets of words. *Bulletin of the EATCS*, 44:178–186, 1991.
- 6 Leonard Eugene Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics*, 35(4):413–422, 1913.
- 7 Hermann Gruber, Markus Holzer, and Martin Kutrib. The size of Higman-Haines sets. *Theoretical Computer Science*, 387(2):167–176, 2007.
- 8 Hermann Gruber, Markus Holzer, and Martin Kutrib. More on the size of higman-haines sets: effective constructions. *Fundamenta Informaticae*, 91(1):105–121, 2009.
- 9 Peter Habermehl, Roland Meyer, and Harro Wimmel. The downward-closure of Petri net languages. In *Proc. of ICALP 2010*, volume 6199 of *LNCS*, pages 466–477. Springer, 2010.
- 10 Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society. Third Series*, 2:326–336, 1952.
- 11 Prateek Karandikar and Philippe Schnoebelen. On the state complexity of closures and interiors of regular languages with subwords. In *Proc. of DCFS 2014*, volume 8614 of *LNCS*, pages 234–245. Springer, 2014.

- 12 Eryk Kopczynski and Anthony Widjaja To. Parikh images of grammars: Complexity and applications. In *Proc. of LICS 2010*, pages 80–89. IEEE, 2010.
- 13 Zhenyue Long, Georgel Calin, Rupak Majumdar, and Roland Meyer. Language-theoretic abstraction refinement. In *Proc. of FASE 2012*, volume 7212 of *LNCS*, pages 362–376. Springer, 2012.
- 14 Richard Mayr. Undecidable problems in unreliable computations. *Theoretical Computer Science*, 297(1-3):337–354, 2003.
- 15 Alexander Okhotin. On the state complexity of scattered substrings and superstrings. *Fundamenta Informaticae*, 99(3):325–338, 2010.
- 16 Jan van Leeuwen. A generalisation of Parikh’s theorem in formal language theory. In *Proc. of ICALP 1974*, volume 14 of *LNCS*, pages 17–26. Springer, 1974.
- 17 Jan van Leeuwen. Effective constructions in well-partially-ordered free monoids. *Discrete Mathematics*, 21(3):237–252, 1978.
- 18 Georg Zetsche. Computing downward closures for stacked counter automata.
- 19 Georg Zetsche. Silent transitions in automata with storage. In *Proc. of ICALP 2013*, volume 7966 of *LNCS*, pages 434–445. Springer, 2013.