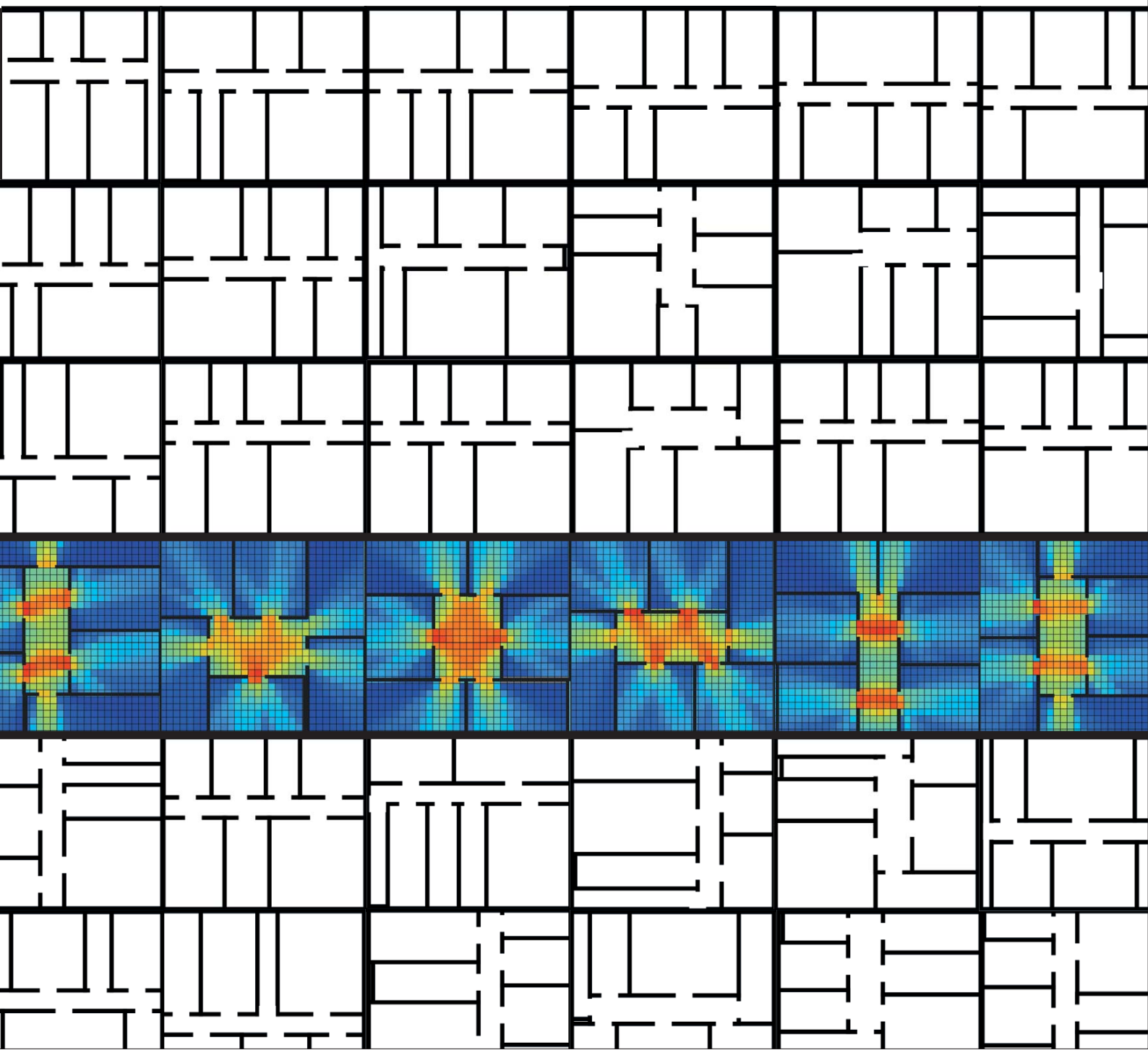
KREMLAS

Entwicklung einer kreativen evolutionären Entwurfsmethode für Layoutprobleme in Architektur und Städtebau

Herausgegeben von
Dirk Donath, Reinhard König und Frank Petzold



KREMLAS

Entwicklung einer kreativen evolutionären Entwurfsmethode für Layoutprobleme in Architektur und Städtebau

Dirk Donath, Reinhard König, Frank Petzold (Hrsg.)

Bauhaus-Universität Weimar
Verlag

Impressum

KREMLAS Entwicklung einer kreativen evolutionären Entwurfsmethode
für Layoutprobleme in Architektur und Städtebau

Dirk Donath, Reinhard König, Frank Petzold (Hrsg.)

Satz und Gestaltung:
Reinhard König

Druck:
Thüringer Papierwarenfabrik
C. Schröter GmbH + Co. KG Mühlhausen

Verlag der Bauhaus-Universität Weimar 2012
verlag@uni-weimar.de
Fax: 03642 / 581156

ISBN: 978-3-86068-471-9

URN: http://nbn-resolving.de/urn:nbn:de:gbv:wim2-20120509-16504

Die Publikation wurde gefördert von der der Deutschen Forschungsgemeinschaft
(DO 551/19-1)

Vorwort der Herausgeber

Das vorliegende Buch fasst die Ergebnisse zusammen, welche von 2009 bis 2011 im Rahmen eines von der Deutschen Forschungsgemeinschaft (DFG) geförderten Forschungsprojekts (DO 551/19-1) entstanden sind. Der DFG sei an dieser Stelle herzlich für die finanzielle Unterstützung gedankt, ohne die dieses Forschungsvorhaben nicht möglich gewesen wäre.

Das Projekt wurde in Kooperation zwischen der Bauhaus-Universität Weimar und der Technischen Universität München durchgeführt und von Prof. Dr. Dirk Donath (Professur Informatik in der Architektur), der zeitweise von Dr. Reinhard König vertreten wurde, sowie von Prof. Dr. Frank Petzold (Lehrstuhl Architekturinformatik) geleitet. Den beiden universitären Einrichtungen gebührt unser Dank für die administrative Betreuung. Als wissenschaftliche Mitarbeiter waren Dipl.-Ing. Sven Schneider, MSc. Dipl.-Ing. (FH) Katja Knecht und Dipl.-Ing. Jan-Ruben Fischer in unterschiedlichen Rollen an den Untersuchungen beteiligt. Nicht zuletzt ihrem herausragenden Engagement ist das Zustandekommen dieses Buches zu verdanken.

Wie bereits die Überschrift dieses Buchs verrät, ging es bei dem im vorliegenden Buch dokumentierten Forschungsprojekt um die Entwicklung einer computerbasierten Entwurfsmethode für Layoutprobleme in Architektur und Städtebau. Dieses Thema wurde, zwar unter Berücksichtigung grundlegender entwurfstheoretischer Überlegungen, aus einer primär technischen Perspektive abgehandelt, sodass wir uns mit den Ausführungen vorwiegend an informationstechnisch interessierte Fachleute und weniger an praktisch arbeitende Architekten wenden.

Alle dargestellten Beispiele wurden mittels C# implementiert. Alle in diesem Buch beschriebenen Computerprogramme können von dieser Internetseite heruntergeladen werden:

http://infar.architektur.uni-weimar.de/service/drupal-cms/KremlasBuch

Über die Herausgeber und Autoren

Dirk Donath (Herausgeber)

Dirk Donath studierte von 1981 bis 1986 Architektur und Angewandte Informatik und promovierte 1988 mit einem Thema zu architektonischen Entwurfsmethoden. Er war von 1988 bis 1990 wissenschaftlicher Assistent und von 1990 bis 1992 Leiter des Bereiches „Computergestütztes Planen" an der Fakultät Architektur der Hochschule für Architektur und Bauwesen Weimar (heute Bauhaus-Universität Weimar). Seit 1993 leitet er die Professur für Informatik in der Architektur an der Bauhaus-Universität Weimar. Seit 2008 ist er Gastprofessor am EiABC der Addis Ababa University Ethiopia im Fach Baukonstruktion und Entwerfen.

Dirk Donath ist seit 1990 Freier Architekt und Mitglied der Architektenkammer Thüringen. Er war von 1990 bis 2011 Partner der Architektengemeinschaft Nitschke-Donath und gründete in dieser Zeit die Ingenieurgemeinschaft b.a.u.werk. Das zentrale Forschungsinteresse von Dirk Donath umfasst unterschiedliche Themenschwerpunkte zu Planungssystemen in der Architektur.

Frank Petzold (Herausgeber)

Frank Petzold studierte angewandte Informatik mit Vertiefung Architektur und Bauingenieurwesen an der Hochschule für Architektur und Bauwesen Weimar. Nach seinem Diplom war Frank Petzold wissenschaftlicher Mitarbeiter an der Professur für Informatik in der Architektur an der Bauhaus-Universität Weimar. Nach seiner Promotion im Jahre 2001 hatte er eine Juniorprofessur für Architekturinformatik bis zu seiner Tätigkeit als Ordinus an der TU München ab 2009 inne. Prof. Petzold ist Mitglied des Arbeitskreises für Architekturinformatik sowie Mitglied in verschiedenen internationalen Gremien.

In Forschung und Lehre setzt sich Frank Petzold mit Fragestellungen der informationstechnischen Unterstützung architektonischer Entwurfsprozesse auseinander. Dabei werden die Tätigkeitsfelder in der Architektur analysiert, Anforderungen an digi-

tale Werkzeuge erarbeitet und mit dem Wissen über neue Technologien Konzepte erstellt, prototypische Lösungen entwickelt und evaluiert.

Reinhard König (Herausgeber und Autor)

Reinhard König hat Architektur und Städtebau an der Hochschule München sowie der TU Kaiserslautern studiert und jeweils mit einem Diplom abgeschlossen. Seine Promotion hat er 2009 an der Universität Karlsruhe am Institut für Orts-, Regional- und Landesplanung mit Auszeichnung beendet. Von 2009 bis 2011 war er Vertretungsprofessor der Professur für Informatik in der Architektur an der Bauhaus-Universität Weimar. Seit 2012 ist er kommissarischer Leiter dieser Professur und verantwortlich für Forschungsprojekte zu computerbasierten Methoden für eine sozial nachhaltige Stadt- und Raumplanung und zur Entwicklung evolutionärer Entwurfsmethoden für Layout-Probleme in Architektur und Städtebau. Forschungsschwerpunkte von Herrn König sind die Komplexität urbaner Systeme sowie die Anwendung generativer Methoden und evolutionärer Algorithmen in Planungs- und Entwurfsprozessen.

Sven Schneider (Autor)

Sven Schneider studierte Medieninformatik an der TU Chemnitz sowie Architektur an der TU Dresden und der Bauhaus-Universität Weimar. Sein Diplom erlangte er an letztgenannter Einrichtung im Jahre 2009. Seitdem war bzw. ist er wissenschaftlicher Mitarbeiter am Lehrstuhl für Architekturinformatik an der TU München bzw. am Lehrstuhl für Informatik in der Architektur an der Bauhaus-Universität Weimar.

Seine Forschungsinteressen liegen bei der Entwicklung generativer Systeme zur Unterstützung von Entwurfs- bzw. Planungsprozessen sowie der Anwendung und Weiterentwicklung von Methoden zur Analyse räumlicher Konfigurationen.

KATJA KNECHT (Autorin)

Katja Knecht studierte Audiovisuelle Medien an der Hochschule der Medien, Stuttgart, und ist Absolventin des Masterprogramms MediaArchitecture der Bauhaus-Universität Weimar. Seit ihrem Abschluss im Sommer 2011 arbeitet sie als wissenschaftliche Mitarbeiterin an der Professur Informatik in der Architektur an der Bauhaus-Universität Weimar.

Im Fokus ihrer Forschung stehen die Schnittstellen zwischen Architektur und Medien. Sie beschäftigt sich sowohl mit der computer- und medienbasierten Unterstützung architektonischer Entwurfs- und Arbeitsprozesse als auch der Entwicklung von ortsbasierten Medienanwendungen und interaktiven, räumlichen Installationen.

Inhaltsverzeichnis

II. Methoden

III. Anwendungsstudien

I. Einführung und Grundlagen

1. Projekteinführung

Reinhard König

Die im vorliegenden Buch dokumentierten Untersuchungen befassen sich mit der Entwicklung von Methoden zur algorithmischen Lösung von Layoutaufgaben im architektonischen Kontext. Die Grundlage dieser Verfahren besteht in einem zirkulär gekoppelten Trial- and Error-System mit auf Evolutionären Algorithmen (EA) basierenden Generierungs- und Evaluationsmechanismen. Die einzelnen Kapitel sollen insbesondere einen Beitrag zur Synthese von Optimierungs- und Gestaltungsstrategien leisten.

In diesem Kapitel wird in das Thema eingeführt und auf die Problematik eingegangen, mit der sich die einzelnen Beiträge aus verschiedenen Perspektiven befassen werden. Ferner wird die Zielstellung beschrieben, welcher im dem Buch zugrunde liegenden Forschungsprojekt nachgegangen wurde, und es wird eine thematische Einordung vorgenommen.

1.1. Einleitung

Layout bezeichnet die Anordnung verschiedener Elemente innerhalb eines gegebenen Raums zu einem bestimmten Zweck. Layoutprobleme findet man in verschiedenen Gebieten, angefangen beim Entwurf von Stundenplänen oder Computerplatinen über die Anordnung von abstrakten Elementen eines Graphen, der Organisation von Kisten einer Schiffsladung oder der Platzierung von auszuschneidenden Blechteilen bis hin zur Anordnung der Räume eines Grundrisses oder der Gebäude einer städtebaulichen Nachbarschaft. Bei der Lösung von Layoutproblemen sind in der Regel bestimmte Kriterien zu berücksichtigen. Beispielsweise sollen sich die Elemente möglichst nicht überlappen und ihre Organisation soll möglichst platzsparend und effizient erfolgen.

Bei den Untersuchungen in diesem Buch geht es um die gestalterisch und funktional sinnvolle Anordnung räumlicher Elemente (z.B. von Parzellen, Gebäuden, Räumen) auf bestimmten Maßstabsebenen (Abb. 1). Maßstabsebenen bezeichnen im

Folgenden die verschiedenen Kontexte einer Planung (z.B. städtebauliche Nachbarschaft, Gebäudemassen, Funktionsbereiche, Grundrisse). Die Anordnung räumlicher Elemente entscheidet maßgeblich über Qualität und Nachhaltigkeit von Gebäuden, Stadtteilen oder ganzen Städten. Dabei muss je nach Projekt und Kontext eine Fülle an verschiedenen Anforderungen zuerst definiert und dann durch einen Lösungsvorschlag möglichst gut erfüllt werden. Das Erarbeiten eines Lösungsvorschlages wird als Entwurfsprozess bezeichnet. Sofern das Ergebnis dieses Prozesses Eigenschaften aufweist, die nicht als direkte Folge der Anforderungen bzw. nicht aus der genauen Beschreibung des Problems hervorgehen, betrachten wir den Entwurfsprozess im Rahmen der vorliegenden Arbeit als einen kreativen Prozess, unabhängig davon, wie „originell" das Ergebnis angesehen wird.

Im Rahmen der vorliegenden Untersuchungen wird dargestellt, wie sich anhand verschiedener Methoden Layoutaufgaben in wesentlichen Teilen automatisch lösen lassen. In diesem Zusammenhang sprechen wir von computerbasiertem Layoutentwurf bzw. Computer Aided Layout Design (CALD). Die folgenden Beiträge konzentrieren sich auf das Layout von Räumen in einem gegebenen Gebäude, sind aber mit geringfügigen Anpassungen grundsätzlich auf andere Maßstabsebenen übertragbar.

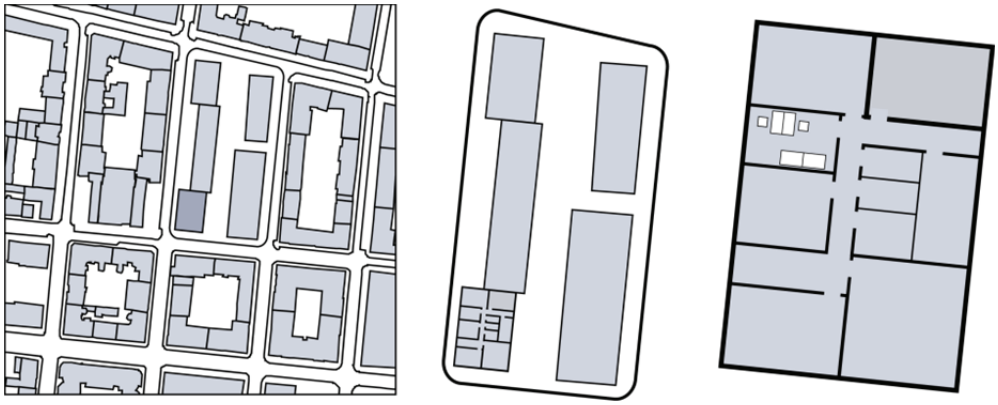


Abb. 1: Verschiedene Maßstabsebenen für Layoutaufgaben.

Mittels computerbasierter generativer Systeme können für bestimmte Problemstellungen automatisch Lösungen generiert werden. Solche Systeme eignen sich insofern zur Unterstützung des Entwurfsprozesses, als dass sie es erlauben, in relativ kurzer Zeit eine große Anzahl von Entwürfen zu generieren und zu überprüfen. Kriterien, die vom Menschen schlecht oder nur langsam überprüfbar sind, können mithilfe generativer Systeme effektiv in den Entwurfsprozess eingebunden werden

(Eckert, Kelly, & Stacey, 1999). Bei einer komplexen Entwurfsaufgabe können beispielsweise Teilprobleme an den Computer übergeben werden, der dafür schnell gute Lösungen finden kann. Bei diesen Teilproblemen handelt es sich um formulierbare bzw. operationalisierbare Probleme, welche in der Regel vor allem die funktionalen Anforderungen an einen Entwurf darstellen.

Beim traditionellen Entwerfen (ohne Unterstützung durch computerbasierte generative Systeme) werden in der Regel Entwurfskonzepte aufgestellt, von denen man annimmt, dass sie alle im Rahmen der Entwurfsaufgabe definierte Kriterien zufriedenstellend berücksichtigen können. Die mithilfe solcher Entwurfshypothesen entwickelten Lösungen werden im Laufe des Entwurfsprozess immer wieder überprüft und verfeinert. Voraussetzung für dieses Vorgehen ist eine wesentliche Reduktion der Komplexität einer Entwurfsaufgabe unter zu Hilfenahme von Leitbildern[1] (Lawson, 2006) auf einer möglichst hohen Maßstabsebene. Diese Leitbilder geben den Rahmen bzw. die Regeln für die Gestaltung auf untergeordneten Maßstabsebenen vor. Eine solche Arbeitsweise kann als Top-Down-Entwurfsstrategie bezeichnet werden, wobei funktionale Anforderungen oft den Leitbildern untergeordnet werden. Die gestalterische Ausarbeitung einer Entwurfssaufgabe findet bei dieser Herangehensweise weitgehend unabhängig von der Optimierung funktionaler Gesichtspunkte statt.

Die automatisierte Lösung von Entwurfssaufgaben durch CALD-Methoden findet dagegen weitgehend unabhängig von gestalterischen Absichten bzw. subjektiven Vorstellungen statt. So werden bei CALD in aller Regel funktionale Kriterien verwendet, um ein Layout zu erzeugen, welches diesen Kriterien optimal entspricht. Hierbei ist zu bemerken, dass Entwurfsaufgaben nicht allein durch Optimierungsmethoden gelöst werden können, da sich gestalterische Absichten nicht oder nur teilweise operationalisieren lassen. Aus diesem Grund kreisen die Auseinandersetzungen in den einzelnen Kapiteln dieses Buchs um eine Strategie, die eine flexible Kombination von Gestaltungs- und Optimierungsmethoden für Grundrisslayouts mittels Methoden zur Nutzerinteraktion mit dem CALD ermöglicht. Folglich werden wir neben einer detaillierten Beschreibung der Optimierungsmethoden immer wie-

[1] Lawson spricht von *„guiding images“* bzw. *„guiding principles“*.

der Aspekte der Nutzerinteraktion beleuchten und untersuchen, wie diese am besten mit Optimierungsmethoden kombiniert werden können.

1.2. Entwurfsprobleme

Begeistert von den Fortschritten in der Computertechnik und künstlichen Intelligenz beschreibt Negroponte 1970 in seinem Buch *The architecture machine* einen intelligenten Entwurfsautomat: *"Imagine a machine that can follow your design methodology and at the same time discern and assimilate your conversational idiosyncrasies. This same machine, after observing your behavior, could build a predictive model of your conversational performance. Such a machine could then reinforce the dialogue by using the predictive model to respond to you in a manner that is in rhythm with your personal behavior and conversational idiosyncrasies"* (Negroponte, 1970). Eine Beschreibung der Methoden, die die Umsetzung einer solchen Maschine erlauben würden, liefert Negroponte jedoch nicht. Die Idee einer intelligenten Entwurfsmaschine bleibt bis heute lediglich eine Vision. So schreibt Donald Schön (1992) resignierend, dass Software, die ähnlich wie ein Entwerfender handeln soll, dies nur kann in einer *"highly restricted situation, a narrowly defined chunk of a design process, where the design world employed by designers can feasibly be assumed as given and fixed"* (Schön, 1992, p. 146). Zu einem ähnlichen Resultat gelangt auch Bryan Lawson: *"Of course the human design process in architecture is not a process of suboptimisation. So the computer as 'oracle' has not so far proved to be helpful and is not likely to do so"* (Lawson, 2005, p. 384).

Warum eine vollständige Entwurfsautomatisierung nicht nur ein technisches, sondern auch ein konzeptionelles Problem ist, wird deutlich, wenn man sich die Charakteristika von Entwurfsproblemen vergegenwärtigt. Dazu ist eine Unterscheidung zwischen operationalen und nicht-operationalen Problemen hilfreich. Ein Problem ist operational, wenn es so genau beschrieben werden kann, dass sich angeben lässt, durch welche Schritte es zu lösen ist. Dies geschieht dadurch, dass im Rahmen einer Analyse ein komplexes Problem in Teilprobleme zerlegt wird, welche dann immer genauer dargestellt werden können. *„Das Ziel der Analyse eines Problems ist eine Beschreibung, die so genau wird, dass sie die Lösung enthält"* (Franck &

Elezkurtaj, 2002). Die konkret definierbaren, handfesten Kriterien zur Problembeschreibung werden als operationale Kriterien bezeichnet.

Dagegen sind nicht-operationale Probleme *„vage definiert, bedeutende Elemente der Aufgabestellung sind unbekannt oder nicht genau (quantitativ) erfassbar, ihr Lösungskriterium ist nicht eindeutig formuliert; der Entscheidungsprozeß beschäftigt sich weniger mit der Suche nach Lösungen, sondern vielmehr mit der Konkretisierung und Abgrenzung des Problems sowie der Schließung offener Beschränkungen"* (Röpke, 1977). Rittel und Webber (1973) nennen diese Probleme aufgrund ihrer Eigenschaften auch *„wicked problems"* oder bösartige bzw. verzwickte Probleme. Für diese Probleme gibt es keine eindeutig richtige oder falsche Lösung. Bei Entwurfsproblemen handelt es sich in der Regel um nicht-operationale Probleme, wodurch sie sich von den meisten Problemen naturwissenschaftlicher Forschung unterscheiden (Simon, 1969). Die Lösung nicht-operationaler Probleme hängt immer von subjektiven und kontextabhängigen Aspekten ab.

Für die Bearbeitung von Entwurfsproblemen sind Heuristiken unerlässlich, welche sinnvolle Annahmen zur Lösung eines Problems angeben, allerdings keine Lösung garantieren. Dass eine Problemlösung durch ein solches Verfahren nicht garantiert werden kann, bedeutet, dass das Verfahren (vorprogrammierte) Fehler bei der Lösungssuche macht. Im besten Fall könnten diese Fehler produktiv verwendet werden, um neue, unerwartete Lösungswege aufzuzeigen. Eine solche Systemeigenschaft könnte zu kreativen Problemlösungen führen, zu welchen Computersysteme bis heute allerdings kaum in der Lage sind.

Ein pragmatischer Ansatz zum Umgang mit diesem Dilemma im Kontext von computerbasierten Entwurfssystemen besteht in der Einbeziehung des Nutzers in den generativen Prozess: Während sich operationale Probleme meist sehr gut algorithmisch lösen lassen, ist man bei nicht-operationalen Problemen auf die Interpretation eines Problems durch den Menschen angewiesen. Die Einbeziehung menschlicher Fähigkeiten in ein generatives Entwurfssystem ermöglicht es, bestimmte Aspekte des Entwurfsprozesses auf operationale Probleme zu reduzieren, denn, *"all that is possible is the conversion of particular problems from ill-structured to well-structured via the one transducer that exists, namely, man"* (Ernst & Newell, 1969).

1.3. Ziele

Trotz vielfältiger Forschungsarbeiten ist bis heute keine umfassende kreative computergestützte Entwurfsmethode entstanden. Viele der bis heute entwickelten generativen Systeme, welche kreative Aufgaben erfüllen sollen, produzieren meist wenig brauchbare Ergebnisse, deren grafische Repräsentation dann lediglich als Inspiration für einen Entwurf verwendet werden kann . Diese Ergebnisverwertung widerspricht allerdings der ursprünglichen Absicht bei der Entwicklung generativer Systeme, wesentliche Anteile bei der Problemlösung zu automatisieren.

Für komplexe Entwurfsaufgaben existieren sehr viele Lösungen. Die Menge dieser Lösungen bilden den Lösungsraum. Kreative Lösungen können als Inseln im Ozean der Möglichkeiten (Suchraum) aufgefasst werden. Das Ziel des dem vorliegenden Buch zugrunde liegenden Forschungsprojekts besteht darin, mittels EA den Suchraum nach diesen Inseln zu durchforsten. Zu diesem Zweck soll eine generative zirkulär gekoppelte Entwurfsumgebung entwickelt werden, die es erlaubt, den Suchraum einerseits flexibel zu definieren und diesen andererseits zu erkunden, anstatt nach Optima für ein parametrisiertes Problem zu suchen. Für die technische Umsetzung werden evolutionäre Strategien verwendet, welche sich aufgrund ihrer Funktionsweise hervorragend für die genannten Absichten eignen: *"…through experimentation and analysis we have learned that evolutionary techniques have excellent abilities as general-purpose problem solvers. Indeed, as Goldberg (1989) states, the genetic algorithm is 'a search algorithm with some of the innovative flair of human search'"* (P. J. Bentley & Corne, 2002, S. 60).

Das vorliegende Buch ist in folgende drei Bereiche gegliedert, welche die Teilziele des Forschungsprojekts Kremlas widerspiegeln:

1. Grundlagen: In den ersten Kapiteln wird nach einer thematischen Einführung und der Darstellung des derzeitigen Forschungsstands die Konzeption eines allgemeingültigen Layout-Entwurfssystems (ALES) für die Lösung von Layoutproblemen vorgestellt (Kapitel 3). Anhand des ALES sollen theoretisch drei Aufgabenbereiche bearbeitet werden können: Das Layout der Bebauungsstruktur einer städtischen Nachbarschaft, die Anordnung der Gebäudemassen und das Layout der möglichen Gebäudegrundrisse. Diese drei Beispiele sind notwendig für den Nachweis der prin-

zipiellen Allgemeingültigkeit der zu entwickelnden Methode. In der methodischen Ausarbeitung konzentrieren wir uns allerdings auf das Layout von Grundrissen auf Gebäudeebene.

2. Methoden: In den Kapiteln 5 bis 8 werden verschiedene Methoden zur Lösung von Layoutproblemen vorgestellt. Diese müssen so gestaltet werden, dass sie sich in das Konzept einer zirkulär gekoppelten computerbasierten Entwurfsmethode, basierend auf dem ALES-Konzept integrieren lassen. Der Entwerfer soll stets die Möglichkeit haben, mit dem automatisierten Lösungsfindungsprozess auf den Ebenen der Problemdefinition, der Definition der Entwurfsgrammatik (formale, geometrische Restriktionen), der Festlegung der operationalen Kriterien sowie der Ergebnisevaluation zu interagieren (Abb. 2).

3. Anwendungsstudien: In den letzten Kapiteln werden mögliche Szenarien zur Anwendung der entwickelten Methoden und des ALES-Konzepts untersucht. Dabei spielt vor allem die Gestaltung einer für Architekten und Städtebauer angemessenen und verständlichen Nutzerinteraktion eine wichtige Rolle. Dazu gehört auch das Ausarbeiten eines Konzeptes für die Eingabe verschiedener Problemstellungen.

1.4. Einordnung des Forschungsvorhabens

Bei den gängigen computerbasierten Entwurfssystemen in Architektur und Ingenieurwesen (Río-Cidoncha, Iglesias, & Martínez-Palacios, 2007) wird für die Problemlösung im Wesentlichen eine lineare Strategie angewandt (vgl. Abb. 6 auf Seite 36). Nach der Definition einer Zielfunktion durch den Nutzer erfolgt so lange eine automatische Lösungssuche ohne weitere Interaktionsmöglichkeit, bis ein zufriedenstellendes Resultat gefunden wurde oder der Suchvorgang abgebrochen wird. Das vorliegende Forschungsvorhaben zielt darauf ab, eine zirkulär gekoppelte Problemlösungsstrategie zu entwickeln, deren wichtigste Bestandteile in Abb. 2 dargestellt sind. Die Randbedingungen sollen während der Lösungssuche vom Nutzer auf den Ebenen der Problemdefinition, der Entwurfsgrammatik, der operationalen Kriterien sowie der Ergebnisevaluation variiert werden können.

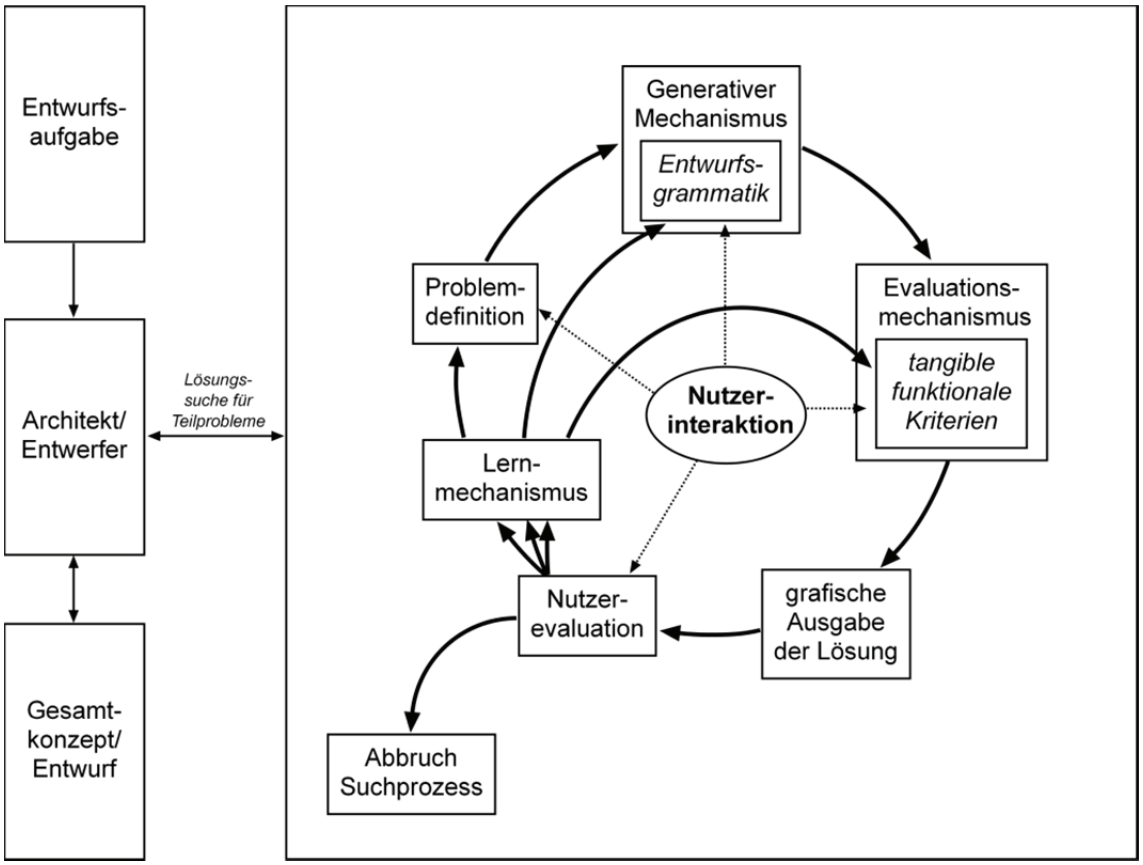


Abb. 2: Forschungsdesign für die Entwicklung einer zirkulär gekoppelten computerbasierten Entwurfsmethode.

Die Intention des Projekts soll abschließend durch folgendes Zitat illustriert werden: *"Again it is worth stressing that such systems are not intended to replace people, but increase productivity and creativity by allowing people to explore more and a wider variety of solutions than they could without such computer systems"* (P. J. Bentley & Corne, 2002, S. 36).

2. Stand der Forschung

Reinhard König, Sven Schneider, Katja Knecht

Forschungsarbeiten zur Computeranwendung in der Architektur (Computer Aided Architectural Design: CAAD) haben sich seit den 1980er Jahren schwerpunktmäßig auf die Entwicklung von Zeichen-, Modellier- und Darstellungswerkzeugen beschränkt. Erst seit der Jahrtausendwende ist ein wiedererwachendes Interesse an den in den 60er- und 70er-Jahren begonnenen Forschungen zu kreativen oder intelligenten generativen Systemen zur Entwurfsunterstützung erkennbar. Generative Systeme sind in der Lage, auf Basis weniger Regeln verschiedene Strukturen zu erzeugen. Abhängig von den Regeln bzw. deren Parametern können stark variierende und unvorhersehbare Strukturen entstehen. In den folgenden Unterkapiteln wird der Stand der Forschung zu den wichtigsten generativen Methoden vorgestellt.

Die Lösung von Layoutproblemen durch computerbasierte Methoden ist ein zentrales Thema in der Anwendung Künstlicher Intelligenz im Bereich der Architektur. Layoutprobleme sind in der Regel sehr komplexe Probleme, welche eine Vielzahl von Anforderungen erfüllen müssen. Mit jedem Faktor, der bei einem Layoutentwurf berücksichtigt werden soll (z.B. Anzahl der Räume), steigt die Anzahl der Lösungsmöglichkeiten exponentiell an (March & Steadman, 1974). Aus der Perspektive der Komplexitätstheorie fallen Layoutprobleme in die Kategorie der sogenannten NP-vollständigen Probleme. Das bedeutet, dass diese Probleme nicht effizient gelöst werden können, da alle bekannten deterministischen Algorithmen für diese Probleme exponentiellen Rechenaufwand erfordern[2].

Zur computerbasierten Lösung von Layoutproblemen wurden seit Anfang der 1960er Jahre (Whitehead & Eldars, 1964) verschiedene Methoden entwickelt (Frew, 1980). Allen diesen Methoden ist gemein, dass sie einen generativen Mechanismus zur Produktion von Lösungsvarianten und einen Evaluationsmechanismus zur Bewertung dieser Varianten beinhalten (Mitchell, 1998). Der Unterschied,

[2] Vgl.: http://de.wikipedia.org/wiki/NP_%28Komplexit%C3%A4tsklasse%29, zuletzt besucht am 05.03.2012.

den wir im Folgenden näher betrachten wollen, besteht in der Ausprägung der beiden Mechanismen.

2.1. Problemwissen

Bei entwurfsunterstützenden Systemen lässt sich zwischen direkten und iterativen Verfahren unterscheiden. Direkte Verfahren liefern nach endlicher Zeit eine exakte Lösung für ein Problem. Sie beruhen meist auf einer umfassenden analytischen Durchdringung des Problems. Iterative Verfahren dagegen liefern in der Regel nur näherungsweise optimale Lösungen für ein Problem, indem sie sich schrittweise an die Ideallösung herantasten.

Neben der Unterscheidung zwischen direkten und iterativen Verfahren ist ein weiteres wichtiges Differenzierungskriterium die Menge an notwendigem Problemwissen, welche anfangs erforderlich ist, um zu brauchbaren Ergebnissen zu gelangen. Auf der einen Seite stehen Methoden, die für den generativen Mechanismus viel und für den Evaluationsmechanismus wenig Problemwissen benötigen und bereits zu Beginn sicherstellen, dass ein entsprechendes generatives System stets akzeptable Ergebnisse liefert. Auf der anderen Seite finden sich Methoden, die mit wenig Problemwissen für den generativen Mechanismus auskommen, dafür aber einen aufwändigen Evaluationsmechanismus benötigen. Basierend auf den Evaluationsergebnissen werden erste Lösungsvarianten in einem iterativen Prozess weiter verbessert, bis sie eine bestimmte Qualität erreichen.

In diesem Zusammenhang ist es wichtig, sich den Unterschied zwischen Problemwissen und Evaluationskriterien zu verdeutlichen. Problemwissen umfasst die genaue Analyse eines Ist-Zustands sowie die präzise Angabe der Schritte, die auszuführen sind, um zu einem bestimmten Soll-Zustand zu gelangen. Evaluationskriterien dagegen geben an, welche Eigenschaften einer Lösungsvariante wie bewertet werden sollen. Da es in der Regel einfacher ist, Evaluationskriterien zu ergänzen, als einen generativen Mechanismus an ein neues Problem anzupassen, können iterative Verfahren, die wenig Problemwissen erfordern, im Vergleich zu direkten Verfahren leichter an sich ändernde Problemstellungen angepasst werden.

Verallgemeinernd können wir feststellen, dass bei Ansätzen, die viel Problemwissen erfordern, die Lösung eines Problems größtenteils in den Vorgaben enthalten ist und bei Verfahren, die wenig Problemwissen verlangen, die Lösung eines Problems nicht explizit im generativen Mechanismus enthalten ist, sondern erst durch die Evaluationskriterien definiert wird. Zur Kategorie von Ansätzen, die viel Problemwissen erfordern, zählen die Methoden der Logischen Programmierung (Coyne, 1988), der Shape Grammar (Duarte, 2001; Stiny & Mitchell, 1978) und der reinen Constraint-Based Systeme (Li, Frazer, & Tang, 2000; Medjdoub & Yannou, 2001). Diese Methoden eignen sich insbesondere zur Bearbeitung gut definierter Probleme wie z.B. der Nachahmung bestimmter Formen, basierend auf Formgrammatiken (Shape Grammar). Zur Klasse der Ansätze, die wenig Problemwissen verlangen, zählen die Methoden der Zellulären Automaten (M Batty & Xie, 1994; Coates, Healy, Lamb, & Voon, 1996), agentenbasierten Systeme (Coates & Schmid, 2000; Derix, 2009) und Evolutionären Algorithmen (Hower, 1997; Jo & Gero, 1998; M A Rosenman, 1997).

Bezugnehmend auf die getroffene Unterscheidung zwischen Verfahren, welche viel und welche wenig Problemwissen erfordern, kann abschließend ergänzt werden, dass beide Verfahren auf operationalen Kriterien basieren (vgl. 1.2), letzteres aber keine vollständige Analyse des komplexen Problems beinhalten muss.

2.2. Constraint-based Verfahren

Eine umfassende Besprechung verschiedener Constraint-Based Verfahren zur Layout-Generierung bis 1996 findet sich bei Hower und Graf (1996). Genau genommen sind alle generativen Verfahren mehr oder weniger durch bestimmte Bedingungen (Constraints) gekennzeichnet und fallen daher in die Kategorie der Constraint-Based Verfahren. Es gibt allerdings eine spezielle Methode der Constraint-Programmierung, die auf einem besonderen Programmierparadigma beruht und daher hier als eigenständige Methode angeführt wird. Ziel der Constraint-Programmierung ist es, eine Menge mathematischer Gleichungen oder logischer

Prädikate (Constraints) auf Widerspruchsfreiheit zu prüfen und gegebenenfalls zu vereinfachen[3].

Ein ausgereiftes System, das mittels Constraint-Programmierung Layouts generiert, ist SEED (Flemming & Woodbury, 1995), welches auf dem ABLOOS Framework basiert (Coyne & Flemming, 1990). ABLOOS wiederum ist eine hierarchische Erweiterung eines Systems namens LOOS (Flemming, 1989), welches orthogonale Strukturen zur Repräsentation von lose gepackten Arrangements von Rechtecken verwendet.

2.3. Shape-Grammar

Shape-Grammar-Methoden sind im Zuge der Entwicklung der Künstlichen Intelligenz (KI) - genauer der symbolischen KI - entstanden. *"Die symbolische KI stellt den groß angelegten Versuch dar, die Grammatik, die die Verknüpfung von Symbolen zu bedeutenden Aussagen regelt, auf rein syntaktische […] zu reduzieren"* (Franck & Elezkurtaj, 2002). Eine Shape-Grammar besteht aus einer Menge elementarer Formen und Symbolen sowie syntaktischer Regeln. Die Regeln dienen der Transformation einer Form oder Formenkollektion in eine neue Form. Rekursiv auf eine Initialform angewendet, ergeben die Regeln Strukturen, von denen man sagen kann, dass sie zu einer Formen-Sprache gehören (Stiny, 1975; Stiny & Gips, 1972). Theorie und Anwendungsmöglichkeiten der Shape-Grammar wurden in den Büchern von Mitchell (1998) und Stiny (2006) umfassend zusammengetragen. Der Vollständigkeit halber ist anzuführen, dass das Prinzip der Shape-Grammar sehr dem der L-Systeme (Fernau, 1994) ähnelt, wobei die Produktionsregeln der L-Systeme den syntaktischen Regeln der Shape-Grammar entsprechen.

Shape-Grammar wurden seit ihrer Einführung hauptsächlich für analytische Zwecke im Rahmen der generativen Beschreibung von Stilen verwendet (Fleming, 1987). So konnte man beispielsweise anhand der Untersuchung von Palladio-Villen (Stiny & Mitchell, 1978), Wrens Stadtkirchen (Buelinckx, 1993), Frank Lloyd Wrights Prairie-Häusern (Koning & Eizenberg, 1981), Wrights Fensterdesigns (Rollo, 1995), japanischen Teehäusern (Knight, 1981), Mughul-Gärten (Stiny & Mitchell, 1980), Hepp-

[3] Vgl.: http://de.wikipedia.org/wiki/Constraintprogrammierung, zuletzt besucht am 05.03.2012.

lewhite Stühlen (Knight, 1994), Stadtvierteln von Marrakesh (Duarte, Rocha, & Soares, 2007) sowie Siza-Häusern (Duarte, 2001, 2005) in überzeugendem Ausmaß belegen, dass sich gestalterische auf syntaktische Wohlgeformtheit reduzieren lässt (Abb. 3). Die Entwurfszeichnungen, welche mittels Shape-Grammar-Methoden generiert wurden, könnten ohne Weiteres für bisher verschollene Originale gehalten werden. Sie wären ernst zu nehmende Kandidaten für einen architektonischen Turing-Test (Elezkurtaj & Franck, 2002, S. 2).

In gewissem Rahmen können die angeführten analytischen Verfahren auch als Beispiele für generative Mechanismen verstanden werden, da zumindest im Prinzip durch eine Manipulation der Formen und Regeln neue *„Design-Sprachen"* abgeleitet werden können. Ein aktuelles Beispiel für die synthetische Verwendung von Shape-Grammar-Techniken ist das Projekt CityEngine (Pascal Müller, 2007), welches die automatische Erzeugung von Gebäudeformen (P Müller, Wonka, Haegler, Ulmer, & Van Gool, 2006) und deren Fassaden (P Müller, Zeng, Wonka, & Van Gool, 2007) sowie von ganzen Städten (Parish & Müller, 2001) umfasst.

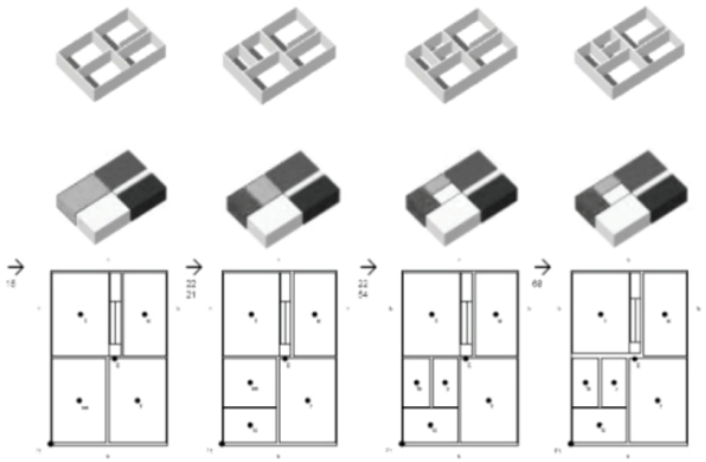


Abb. 3: Jose Duarte, Customizing Mass Housing. Abbildung aus (Duarte, 2001).

Trotz der vielversprechenden Möglichkeiten konnte mittels der Shape-Grammar-Methode bisher kein praxistaugliches System zur Entwurfsunterstützung entwickelt werden. Das Problem der Methode liegt vor allem darin, dass sich der architektonische Entwurf nicht darauf reduzieren lässt, Grafiken zu produzieren und Stile zu imitieren. Entwurfszeichnungen haben auch eine (semantische) Bedeutung, die sich beispielsweise aus der Funktion ergibt, die die Elemente eines Gebäudes oder einer Siedlungsstruktur haben sollen. Funktion ist nun aber in der Architektur stark kon-

textabhängig, was es unmöglich macht, Entwurfsprobleme vollständig zu beschreiben (vgl. Punkt 1.2).

2.4. Kräftebasierte Systeme

Kräftebasierte Systeme bezeichnen Systeme, die, basierend auf der Simulation physikalischer Kräfte, z.B. Anziehung und Abstoßung (Abb. 4), die Layoutorganisation zu automatisieren versuchen (Arvin & House, 2002).

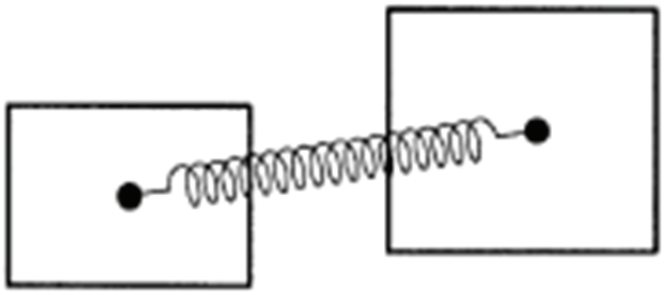


Abb. 4: System virtueller Federn. Abbildung aus (Arvin & House, 2002).

Planelementen werden dabei durch den Entwerfer *„Kraftfelder"* zugeordnet, die zum einen die topologische Beziehung der Elemente beeinflussen und zum anderen auch die geometrische Form der Elemente verändern können. Ein solches System soll eine möglichst natürliche, interaktive, intuitive und flexible Art des Entwerfens ermöglichen. Dabei werden die Designabsichten z.B. in ein Masse-Feder-System übersetzt, welches im Anfangszustand in einem kraftmäßigen Ungleichgewicht ist und durch den Simulationsprozess ein Gleichgewicht anstrebt.

Die Nachteile von kräftebasierten Systemen liegen vor allem in der Begrenzung der Menge möglicher Entwurfsziele, da diese sich nicht immer in Form von anziehenden oder abstoßenden Kräften modellieren lassen. Ferner blockieren sich die Elemente durch ihre körperhaften Eigenschaften gegenseitig beim Versuch, ein Kräftegleichgewicht zu erreichen.

2.5. Zelluläre Automaten und agentenbasierte Systeme

Anwendungen für Zelluläre Automaten (ZA) und agentenbasierte Systeme finden sich eher auf der stadt- und regionalplanerischen Ebene. ZA bestehen in ihrer einfachsten Ausführung aus einem Zellenraster, dessen Zellen ihre Zustände in Abhän-

gigkeit von den Zuständen der Nachbarzellen ändern können (Toffoli & Margolus, 1987). Die Zellenzustände können beispielsweise funktionale Elemente wie Straßen, Freiflächen oder Gebäudeteile darstellen. Basierend auf Regeln für die Zustandsänderung einer Zelle können räumliche Anordnungen bestimmter Elemente generiert werden (M Batty & Xie, 1994; Koenig, 2012; Koenig & Bauriedel, 2004).

Agentenbasierte Systeme bestehen aus autarken Entitäten (Agenten), welche untereinander und mit ihrer Umgebung Informationen austauschen können. Auf diese Weise lassen sich z.B. Wegestrukturen in einer Landschaft (Schweitzer, 1997) oder Interaktionsraten zwischen Siedlungen (Michael Batty, 2005) erzeugen oder Gebäude in Abhängigkeit vom urbanen Kontext platzieren (Coates & Schmid, 2000). Dabei wird das agentenbasierte System oft mit einem ZA zur Repräsentation einer Landschaft oder Stadt kombiniert.

Die Verwendung von ZA ist insofern eingeschränkt, als diese nur bestimmte geometrische Strukturen abbilden können (meist regelmäßige Zellenraster). Zwar gibt es auch Übertragungen auf unregelmäßige Zellensysteme (Dillenburger, Braach, & Hovestadt, 2009; O'Sullivan, 2001), trotzdem bleibt stets eine gewisse geometrische Einschränkung erhalten, da die Nachbarschaftsverhältnisse immer eindeutig definiert sein müssen. Agentenbasierte Modelle erlauben zwar eine größere geometrische Freiheit als ZA (Braach, 2002; Coates, Appels, Simon, & Derix, 2001), können aber nur indirekt über die Interaktionsregeln der Agenten gesteuert werden. Außerdem müssen sie für die Erzeugung komplexer Geometrie mit weiteren generativen Methoden kombiniert werden.

2.6. Unterteilungsalgorithmen

Unterteilungsalgorithmen, auch Raumpartitionierungsalgorithmen genannt, sind Algorithmen, die Flächen oder mehrdimensionale Räume sowie Datenräume mithilfe von Schnittlinien oder -ebenen nach bestimmten Regeln oder einer festgelegten Abfolge in kleinere Unterräume unterteilen. Sie stammen unter anderem aus dem Gebiet der Computergrafik, wo sie beispielsweise zur Unterteilung von Polygonen eingesetzt werden, um gekrümmte Flächen im dreidimensionalen Raum annähernd darstellen zu können (Catmull & Clark, 1978).

Die Unterteilung der Fläche oder eines Raums erfolgt bei den meisten Algorithmen rekursiv in immer kleinere Unterflächen bzw. Unterräume. Die Abfolge und Lage der Schnittlinien kann als Baumstruktur, als sogenannter *Slicing Tree*, gespeichert und organisiert werden, bei dem die Unterflächen als Knoten und die resultierenden Endflächen als Blätter abgebildet werden. Diese Form der Datenstruktur lässt sich besonders effizient erstellen, durchsuchen und verarbeiten, weshalb Raumpartitionierungsalgorithmen häufig in der Computergeometrie, zum Beispiel bei der Suche nach den nächsten Nachbarn, der *Nearest Neighbor Query* (Moore, 1991), Anwendung finden. Außerdem werden sie für Suchalgorithmen bei klassischen Datenbankapplikationen eingesetzt (J. L. Bentley, 1990).

Unterteilungsalgorithmen zeichnen sich außerdem dadurch aus, dass Elemente durch Unterteilung einer vorgegebenen Grundfläche dicht gepackt werden können. Diese Eigenschaft macht sie interessant für den Einsatz im *Floorplanning*[4] (Young & Wong, 1997) oder zur Lösung von Facility Layout Problemen (Kado, 1995).

Das Unterteilen einer vorgegebenen Grundfläche in Zonen und Räume stellt darüber hinaus eine im Architekturentwurf häufig eingesetzte Methode zur Grundrissentwicklung dar, weshalb Unterteilungsalgorithmen auch zunehmend zur Automatisierung der Generierung von architektonischen Layouts herangezogen werden. Beispielsweise wurden sie zur automatischen Generierung von Stadtstrukturen und Gebäudegrundrissen insbesondere bei Computerspielen eingesetzt (Hahn, Bose, & Whitehead, 2006). Marson und Musse (2010) haben zudem die Verwendung von quadratisierten Unterteilungsbäumen zur Echtzeitgenerierung von architektonisch sinnvollen Grundrissen untersucht.

Es existieren verschiedene Arten von Unterteilungsalgorithmen, die sich im Aufbau der Datenstruktur und den verwendeten Unterteilungsregeln unterscheiden sowie unterschiedliche Charakteristika besitzen. Die Auswahl der Schnittdimension und die Bestimmung der Lage der Unterteilung können beispielsweise zufällig oder nach festgelegten Regeln erfolgen. Zum Beispiel raum- oder flächenbasiert, indem eine Fläche immer nach der längeren Seite in einem festgelegten Proportionsverhältnis

[4] *Floorplanning* betrachtet die Optimierung von Lagebeziehungen zwischen Bauteilen, wie beispielsweise im Platinen-, Chip- oder Anlagendesign.

unterteilt wird, oder so, dass alle resultierenden Unterräume den gleichen Flächeninhalt besitzen. Unterteilungsalgorithmen, die auf k-dimensionalen Bäumen basieren (siehe auch Kapitel 6), nutzen darüber hinaus Punktmengen zur Raumpartitionierung und berechnen die Schnittlinien durch Mittel- oder Medianwertberechnung aller Punktkoordinaten. Der Aufbau einer Unterteilungsstruktur kann außerdem auch durch Interpretation einer Zeichenfolge, der sogenannten Unterteilungssyntax, erfolgen (siehe auch Kapitel 7.1.1.3).

In den letzten Jahren entstanden darüber hinaus einige Arbeiten auf Basis von Voronoi-Diagrammen, beispielsweise zur Generierung von städtischen Straßennetzwerken (Anders & König, 2011) sowie von architektonischen Raumstrukturen (Coates, Derix, Krakhofer, & Karanouh, 2005; Harding & Derix, 2010) (Abb. 5). Unterteilungsalgorithmen auf Basis von Voronoi-Diagrammen besitzen wiederum andere Unterteilungssystematiken und -abläufe. Hier entstehen in Abhängigkeit von der Verteilung der Raumzentren durch Unterteilung sogenannte Zellen. Die Unterteilung und damit das Diagramm werden aus den Zellgrenzen gebildet. Eine solche Grenze besteht aus all jenen Punkten, die gleichweit von mehreren Zentren entfernt liegen (De Berg, Cheong, Van Kreveld, & Overmars, 1997). Es entstehen Zellen- bzw. schaumartige Strukturen, deren Form durch Lage, Anzahl und Verteilung der Raumzentren bestimmt werden.

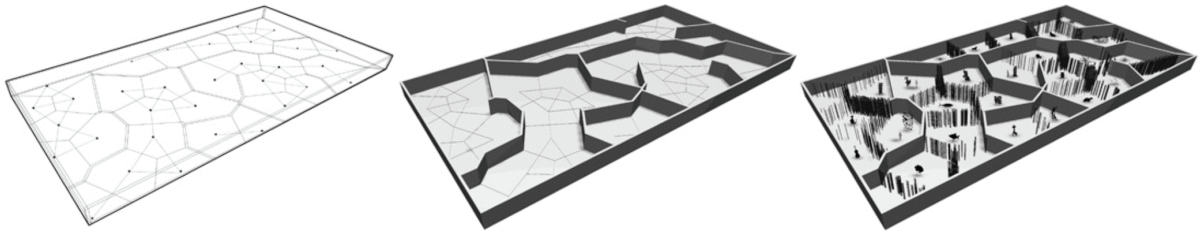


Abb. 5: Voronoi-Diagramm (Harding & Derix, 2010).

Ein entscheidender Nachteil von Unterteilungsalgorithmen resultiert aus der Tatsache, dass sie als subtraktive Methode nur auf bereits bestehende, vorgegebene und vordefinierte Flächen, Räume, Volumen oder Körper angewendet werden können, d.h. auf ihrer Basis keine initialen und neuen Flächen, Räume, Volumen oder Körper erstellt oder dem Entwurf hinzugefügt werden können. Dadurch werden ihre Einsatzmöglichkeiten stark eingeschränkt. Es empfiehlt sich aus diesem Grund eine

Kombination mit anderen Methoden (P Müller, et al., 2006; Wonka, Wimmer, Sillion, & Ribarsky, 2003).

2.7. Kombinierte Methoden und Performancesimulation

Eine weitere Möglichkeit für das Erzeugen komplexer Geometrie besteht in der Kombination der Shape-Grammar-Methode mit Genetischen Algorithmen (GA). Dabei werden die syntaktischen Regeln als Chromosomen kodiert (Koutamanis, 2000). Dieses Verfahren wird in der Arbeit von Chouchoulas und Day (2007) als Shape-Code bezeichnet. Das Problem besteht allerdings nach wie vor im Fehlen der semantischen Informationen, also im Fehlen funktionaler Aussagen über einzelne Elemente. Dementsprechend ist es nicht möglich, einen sinnvollen Evaluations-Algorithmus zu entwickeln (Chouchoulas & Day, 2007, S. 31).

Einen ähnlichen Ansatz wie Chouchoulas und Day (2007) verfolgt Doulgerakis (2007). Er bezieht sich zwar nicht explizit auf die Shape-Grammar-Methode, wendet aber ebenso Transformationsregeln, die mittels Genetischer Programmierung (GP) kodiert werden, auf eine Ausgangsform an. Im Gegensatz zu Elezkurtaj und Franck (2002), die mittels GA eine Lösung für die Addition einer vorgegebenen Anzahl von Räumen gefunden haben, konnte Doulgerakis (2007) mittels GP keine befriedigenden Resultate für ein additives Verfahren erzielen. Daher hat er sich auf ein subtrahierendes Verfahren konzentriert, mittels welchem sich Lösungen mit anderen geometrischen Eigenschaften ergeben.

Weitere Möglichkeiten zur Kombination generativer Methoden bestehen darin, diese durch evaluierende Funktionen zu ergänzen. Die Abgrenzung zu den Evaluationsmechanismen, die in Kapitel 2.1 eingehend beschrieben wurden, ist im Einzelfall nicht immer eindeutig möglich. Im Allgemeinen handelt es sich bei den hier gemeinten Funktionen um komplexere Performancesimulationen. Solche Bewertungsverfahren dienen zur Generierung assoziativer Daten für erzeugte Lösungen (Chao et al., 1997). Bei Doulgerakis (2007) wird beispielsweise zuerst die räumliche Struktur generiert und anschließend dahingehend bewertet, wie gut sich ein gewünschtes Raumprogramm darin unterbringen lässt und wie nutzbar das Gebäude hinsichtlich bestimmter Kriterien ist. Das letztgenannte Bewertungskriterium wird mittels agentenbasierter Simulation ermittelt (siehe auch die Arbeit von Miranda, 2004).

Die Schwierigkeit bei Performancesimulationen besteht allerdings in den ihnen zugrunde liegenden Annahmen und Parametern, da es für diese in der Regel nur ein mangelhaftes empirisches Fundament gibt. Das Gleiche gilt für Versuche, die menschliche Wahrnehmung mittels künstlicher neuronaler Netze nachzubilden (Derix, 2004), insofern, als es bisher keine Methode gibt, die menschliche Wahrnehmung angemessen zu operationalisieren.

Weitere Ansätze für Performancesimulationen finden sich beispielsweise bei Watanabe (1990, 2002). Bei diesem wird auf dem Maßstab des Gebäudes die Besonnungszeit einer Raumeinheit als Bewertungskriterium für die Gebäudegestalt gewählt und auf städtebaulichem Maßstab die Verschlungenheit der Straßenführung und die Erreichbarkeiten verschiedener Versorgungsfunktionen als Evaluationskriterien für die städtebauliche Gestaltung herangezogen. Ein theoretischer Rahmen für Performance als Forschungs- und Entwurfskonzept findet sich bei Hensel und Menges (2008).

Eine andere, relativ ausgereifte Methode für Performancesimulationen bietet die Bewertung der Erschließungsstruktur mittels graphenbasierter Messmethoden, die allen voran von Space Syntax propagiert werden (Hillier, 2007; Hillier & Hanson, 1984). Diese erlauben es, Kennwerte zu Zentralität (Integration) und Durchgangspotential (Choice) einzelner Straßensegmente innerhalb eines Straßennetzes zu berechnen. Diese Kennwerte wurden in zahlreichen Untersuchungen der Space Syntax Gruppe mit verschiedenen Nutzungsmustern, z.B. dem Fußgängerverkehr, korreliert (Hillier, 2005).

2.8. Konklusion

In der Regel werden anhand der in diesem Kapitel betrachteten generativen Methoden zweidimensionale Strukturen erstellt. Einen Überblick zur Praxisrelevanz der hier dargestellten Methoden liefert (Derix, 2009). Exemplarische Herangehensweisen zur Umsetzung generativer Systeme im dreidimensionalen Raum, welche realitätsnahe Ergebnisse liefern, finden sich bei Chouchoulas und Day (2007), Miranda (2004) sowie Doulgerakis (2007).

Die aus Sicht der Autoren vielversprechendsten Ansätze in der automatischen Layoutgenerierung bestehen aus einer Kombination von generativen Mechanismen mit EA. Aufgrund der enormen Relevanz von EA für die in diesem Buch dargestellten Untersuchungen widmen wir deren Darstellung ein eigenes Kapitel (siehe Kapitel 4).

Eine grundsätzliche Schwierigkeit bei generativen Systemen besteht im Signature-Problem (Schnier, 2008), welches besagt, dass die Wahl der generativen Methode das Design stark definiert. *"Most evolutionary art implementations share one characteristic: being locked into a fixed, highly distinguishable 'style' (the 'signature problem')"* (Schnier, 2008). Dieses Problem ist bei ZA und auf ZA basierenden agentenbasierten Systemen besonders ausgeprägt. Das liegt vor allem daran, dass ZA mit relativ unflexiblen geometrischen Strukturen zur Repräsentation der Zellen arbeiten.

Trotz der prototypischen Entwicklung von CALD-Systemen wie SEED (Flemming & Woodbury, 1995) oder ARCHiPLAN (Medjdoub & Yannou, 2001) ist den Autoren kein kommerziell verfügbares System bekannt, welches in der Lage ist, automatisch Grundrisslayouts zu erzeugen, die dicht gepackt sind und definierte Nachbarschaftsbeziehungen für die Räume erfüllen.

3. Konzeption eines Allgemeinen Layout-Entwurfssystems (ALES)[5]

Reinhard König, Sven Schneider

Das Kapitel behandelt die Konzeption eines allgemeinen Layoutentwurfssystems. Hierzu werden auf Grundlage spezifischer Charakteristika von Entwurfsprozessen Anforderungen abgeleitet, die ein solches System erfüllen muss. Aus diesen Anforderungen werden konkrete Kriterien abgeleitet, welche für die Entwicklung von Layoutsystemen beachtet werden sollten. Auf diese Kriterien wird vor allem im Methodenteil (Kapitel 5 bis 9) an verschiedenen Stellen Bezug genommen.

3.1. Einleitung

In Kapitel 2 wurden verschiedene Ansätze zur Lösung von Layoutproblemen dargestellt und die bei diesen verwendeten Methoden mit ihren jeweiligen Potentialen und Beschränkungen für die Entwicklung von computerbasierten Entwurfssystemen aufgezeigt. Zu bemerken ist, dass keines der genannten Projekte bislang Verwendung bei praktizierenden Architekten gefunden hat. Die fehlende Akzeptanz, diese Systeme im Entwurfsprozess einzusetzen, ist zu Teilen auf die mangelnde Berücksichtigung spezifischer Charakteristika von Entwurfsprozessen zurückzuführen. Zum einen behandeln alle bisher entwickelten Methoden operationale Probleme. Kreative Lösungen sind allerdings die Folge schlecht definierter Situationen (vgl. Punkt 1.2). Zum anderen fehlt es den Projekten an Universalität, da meist nur bestimmte Teilprobleme behandelt werden. Diese Teilprobleme ergeben sich jedoch oft erst im Laufe des Entwurfs, und selbst da sind sie meist nicht als solche konkret erkennbar. Darüber hinaus ist zu erwähnen, dass bei den gängigen computerbasierten Entwurfssystemen oft ein lineares Prinzip verfolgt wird: Nach der Definition einer Zielfunktion durch den Nutzer erfolgt so lange eine automatische Lösungssuche oh-

[5] Dieses Kapitel beruht zu Teilen auf dem Artikel von Schneider, S., Fischer, J. R. & Koenig, R. (2010). Rethinking Automated Layout Design: Developing a Creative Evolutionary Design Method for the Layout Problems in Architecture and Urban Design. Paper presented at the Design Computing and Cognition (DCC '10), Stuttgart.

ne weitere Interaktionsmöglichkeit, bis ein zufriedenstellendes Resultat gefunden wurde oder der Suchvorgang abgebrochen wird (siehe Abb. 6).

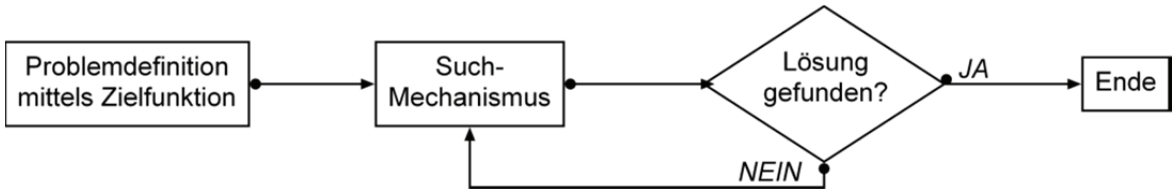


Abb. 6: Lineare Problemlösungsstrategie.

Welche Faktoren berücksichtigt werden müssen, um ein Entwurfssystem besser in den Prozess des kreativen Entwerfens integrieren zu können, wird im Folgenden diskutiert.

3.2. Anforderungen an ein entwurfsunterstützendes Layoutsystem

Für die Konzeption von generativen Entwurfssystemen ist es hilfreich, sich das Zusammenspiels von Entwerfer und Werkzeug (siehe Abb. 7) zu vergegenwärtigen. Dieses Zusammenspiel kann als eine Art Kreislauf aufgefasst werden (Gänshirt, 2007): Entwerfer benutzen Werkzeuge, um ihre (zunächst unsichtbaren) Gedanken und Ideen zu externalisieren, sprich in Form von Artefakten (Skizzen, Modelle, Bilder, etc.) sichtbar zu machen. Diese Artefakte werden in der Folge bewertet, mit den Entwurfskriterien abgeglichen und mittels verschiedener Entwurfswerkzeuge weiterentwickelt. Schön (1983) nennt diesen Kreislauf auch *„reflective conversation with the situation"*. Hinsichtlich der Rolle der Werkzeuge in diesem Prozess ist zu bemerken, dass jedes Werkzeug nur einen bestimmten Raum an möglichen Operationen abdeckt. Diese begrenzen den Handlungsspielraum des Benutzers eines bestimmten Werkzeuges, sprich die Menge möglicher Lösungen, die mit einem Werkzeug innerhalb eines bestimmten Zeitraumes realisierbar ist. Je größer dieser Handlungsspielraum, desto uneingeschränkter ist man bei der Suche nach geeigneten Lösungen für ein Problem.

Für die Entwicklung eines generativen Entwurfssystems zur Unterstützung von Entwurfsprozessen bedeutet dies zweierlei: Zum einen muss es sich möglichst nahtlos in den beschriebenen Kreislauf (Abb. 7) einfügen. Schön (1992) spricht hier von der Berücksichtigung des elementarsten Bestandteils des Entwurfsprozesses, dem *„see-*

ing-moving-seeing". Zum anderen ist es entscheidend, den Handlungsspielraum, den das Werkzeug bietet, möglichst groß zu halten. Die Qualität einer Software hängt also entscheidend von ihrem Potential zum Dialog mit dem Nutzer *(conversation)* und ihrer Unvorherbestimmtheit *(experience of surprise)* ab (Schön, 1992). Im Folgenden werden vier Eigenschaften beschrieben, die ein generatives Entwurfssystem aufweisen muss, um den genannten Anforderungen gerecht zu werden.

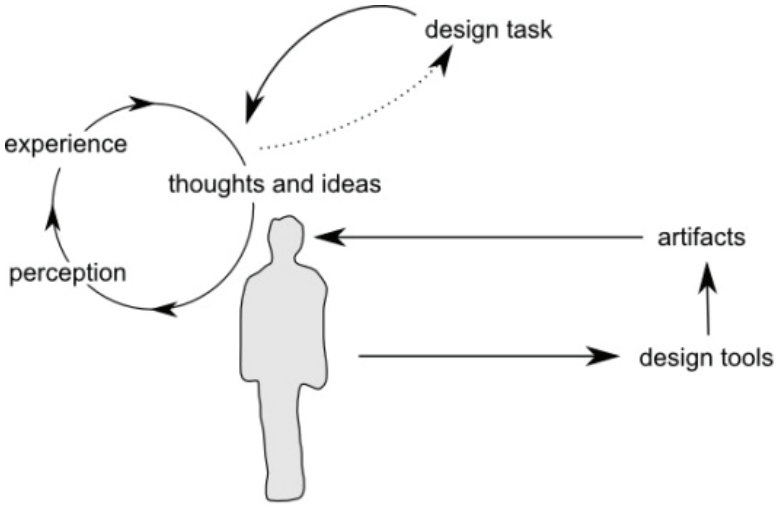


Abb. 7: Entwurfskreislauf als Zusammenspiel zwischen Mensch und Werkzeug.

3.2.1. Adaptivität

Die erste Eigenschaft betrifft die Adaptivität bzw. Anpassbarkeit eines Entwurfssystems. Da sich die Wahrnehmung der Probleme und die Reaktion auf diese im Entwurfsprozess parallel entwickeln (Schön, 1983), können Entwurfsprobleme nicht ad hoc mittels konkreten, vorgefertigten Mustern abgebildet werden. Im Gegensatz zu *„presented problem situations"* hat man es beim Entwerfen mit *„discovered problem situations"* zu tun (Getzels & Csikszentmihalyi, 1967). Entwurfsprobleme sind also nicht vorgegeben, sondern werden Schritt für Schritt während des Entwurfsprozesses entdeckt bzw. definiert.

Ein Entwurfssystem muss es daher erlauben, während des Suchprozesses neue Anforderungen für ein Entwurfsproblem definieren bzw. bestehende Anforderungen ändern zu können. Adaptiv bedeutet folglich, dass die Problemdefinition möglichst flexibel anpassbar ist. Voraussetzung für eine solche Adaptivität ist die im Folgenden beschriebene Zirkularität.

3.2.2. Zirkularität

Entwerfen ist ein iterativer Prozess. Jede gefundene Lösung kann aus einer anderen Perspektive wieder neue Probleme bergen, die in weiteren Entwurfsiterationen formuliert und behandelt werden müssen. Die Reihenfolge für die Definition der Probleme und die mit diesen verbundenen Anforderungen ist jedoch nicht festgeschrieben. Der fortwährende Anpassungsprozess folgt keinem vordefinierten Schema (siehe 3.2.1).

Entscheidend für ein Entwurfssystem ist es daher, dass dieses kontinuierlich arbeitet. Es gibt also keinen konkreten Anfangs- oder Endpunkt für den Lösungsprozess, sondern es muss sich immer in Bereitschaft befinden, um auf sich ändernde Problemstellungen reagieren zu können.

3.2.3. Explorativität

Generative Systeme können Lösungen erzeugen, welche bestimmten operationalen Kriterien entsprechen. Diese Kriterien und deren Gewichtung können bei jedem Entwerfer bzw. bei jedem Entwurf variieren. Entscheidend ist, dass die Bewertungsfunktion des generativen Systems flexibel definiert werden kann. Anstatt nach Optima für ein parametrisiertes Problem zu suchen, muss das System es ermöglichen, einen Suchraum flexibel zu definieren, zu verändern und auf verschiedenen Pfaden zu erkunden.

Hinsichtlich sich widersprechender Kriterien, die sich während eines Entwurfsprozesses ergeben, ist es wichtig, dass das generative System ein möglichst breites Spektrum an gleichwertigen Kompromisslösungen anbietet. Das bedeutet, dass das System Lösungen anbieten soll, die möglichst gleichmäßig über die sogenannte Pareto-Front (auf der alle Kompromisslösungen liegen) verteilt sind (siehe 5.2.3). Dadurch wird sichergestellt, dass alle relevanten Kompromisse angegeben werden. Die Auswahl zwischen den gleichberechtigten Kompromisslösungen muss durch einen Nutzer erfolgen, sofern keine weiteren Kriterien eingeführt werden.

3.2.5. Unmittelbarkeit

Während des Entwurfsprozesses gibt es sowohl Phasen, in denen der Entwerfende die erarbeitete Lösung in aller Ruhe reflektiert, als auch Phasen, in denen er in hoher Geschwindigkeit verschiedene Dinge mehr oder weniger gleichzeitig zu organisieren versucht (Lawson, 2005). Der Entwerfer muss in den entscheidenden Momenten die Auswirkungen seiner Handlungen bzw. Entscheidungen sofort sehen können. Für ein generatives System bedeutet dies, dass es Lösungen unmittelbar, d.h. ohne zeitliche Verzögerung, erzeugen muss. Erst diese Unmittelbarkeit ermöglicht ein Verständnis der Effekte, welche durch bestimmte Veränderungen von Regeln bzw. Kriterien hervorgerufen werden.

3.3. Wahl der generativen Methode

Ein wesentlicher Punkt für die Entwicklung eines interaktiven Entwurfssystems betrifft die Wahl der generativen Methode, da diese das Verhalten des Systems bei der Problemlösung bestimmt. Bei generativen Systemen kann, wie in Abschnitt 2.1 beschrieben, zwischen direkten und iterativen Verfahren unterschieden werden. Direkte Verfahren liefern nach endlicher Zeit eine exakte Lösung für ein Problem. Sie beruhen meist auf einer umfassenden analytischen Durchdringung des Problems, welche die notwendigen Informationen zur Berechnung einer Lösung liefert. Rechenberg (1994) nennt diese Verfahren auch problemorientierte Verfahren. Da sie keinen Rückkopplungsmechanismus beinhalten, ist das Lösungsverfahren linear. Es ist kein Eingriff während des Problemlösungsprozesses möglich. Die Funktionsweise iterativer Methoden besteht darin, sich in einem Prozess, bestehend aus Generierung und Bewertung, schrittweise an eine Ideallösung heranzutasten. Diese Verfahren liefern zwar oft nur näherungsweise optimale Lösungen für ein Problem, jedoch ist es möglich, nach jeder Iteration Änderungen an den Regeln bzw. Kriterien, die im Problemlösungsprozess angewandt werden, vorzunehmen. Dies ist entscheidend, um die unter Punkt 3.2 genannten Kriterien Adaptivität, Zirkularität und Explorativität angemessen zu berücksichtigen.

Hinsichtlich der Menge an notwendigem Problemwissen, welche erforderlich ist, um zu brauchbaren Entwurfsvarianten zu gelangen, ist es für die Entwicklung eines ALES entscheidend, dass das Entwurfssystem mit möglichst wenig Problemwissen

(vgl. Punkt 2.1) auskommt, um auch für unstrukturierte Probleme Lösungen anbieten zu können. *„Denn hochgradiges Nichtwissen ist ja gerade das, was ein Problem zum Problem macht"* (Rechenberg, 1994, p. 218).

Zur Bearbeitung von Problemen, bei denen die Lösung noch nicht in den Vorgaben enthalten sein soll, eignen sich insbesondere Evolutionäre Algorithmen (EA). Die an der biologischen Evolution orientierten Methoden lassen in einem zirkulären Trial-and-Error Prozess Lösungen entstehen, die mittels zufälliger Variationen schrittweise an bestimmte Anforderungen (Evaluations- oder Fitnesskriterien) angepasst werden. Dabei ist es nicht nötig, konkrete Verbesserungsanweisungen (wie sie im Falle direkter Methoden formuliert werden müssen) zu definieren (Rechenberg, 1994). Entscheidend für diesen Prozess sind die Generierungsregeln und die Evaluationskriterien. Die Generierungsregeln geben den Rahmen vor, innerhalb dessen nach optimalen Lösungen gesucht werden kann. Die Evaluationskriterien geben die Richtung vor, in welcher im Lösungsraum (der durch die Generierungsregeln vorgegeben ist) gesucht wird. Die Funktionsweise von EA werden im Kapitel 4 ausführlich beschrieben. Entscheidend für die Implementierung eines EA sind die Evaluationskriterien bzw. die Generierungsregeln. Um diese zu bestimmen, ist eine formale Beschreibung der Probleme notwendig, die mithilfe des Entwurfssystems gelöst werden sollen.

3.4. Formalisierung des Entwurfsproblems

Ein wesentlicher Aspekt bei der Implementierung eines interaktiven generativen Systems ist die Art und Weise der Repräsentation eines Entwurfsproblems durch dessen geschickte Formalisierung. Durch sie wird der Einsatzbereich des Entwurfssystems festgelegt, sprich die Frage beantwortet: Was kann man mit dem System entwerfen? Je eingeschränkter die Möglichkeiten eines Systems sind, Probleme zu repräsentieren, desto eingeschränkter ist der Entwerfende bei der Suche nach Lösungen für seine Probleme, da diese sich, wie unter Punkt 3.2.2 dargestellt, oft erst während des eigentlichen Entwurfsprozesses ergeben.

Für die Problemrepräsentation eines ALES sind drei Punkte relevant: Erstens die Definition von Layout, welche ausschlaggebend ist für den Anwendungsbereich, der sich mit dem System abdecken lässt. Zweitens die Regeln zur Formerzeugung, wel-

che die Variantenvielfalt determinieren, die das System erzeugen kann und damit auch das Potential des Systems bestimmen, zu kreativen Lösungen zu gelangen. Drittens die Bewertungskriterien, mit deren Hilfe der Nutzer des Systems die Suche nach Lösungen steuern kann.

3.4.1. Definition Layout

Entwerfen ist ein Prozess, der auf mehreren Maßstabsebenen simultan stattfindet. Ein System zur Unterstützung dieses Prozesses darf sich dementsprechend nicht auf eine Ebene beschränken, sondern muss auf verschiedenen Ebenen funktionieren. Daher muss auch der Begriff Layout so definiert werden, dass er möglichst universell verwendbar ist, d.h. für Entwurfsprobleme auf verschiedenen Maßstabsebenen Gültigkeit besitzt. Ganz allgemein beschreibt Layout in Architektur und Städtebau die sinnfällige Anordnung verschiedener Elemente wie Parzellen, Gebäude, Räume, Zonen, Bauteile, Möbel etc. auf unterschiedlichen Maßstabsebenen (Abb. 1). Diese Elemente können als geometrische Objekte repräsentiert werden[6]. Zusätzlich zu der geometrischen Information müssen die Elemente auch Informationen zu den an sie gestellten Anforderungen enthalten können. Dabei kann es sich um Mindestmaße oder Relationen zu anderen Elementen (z.B. Nähe oder Abstände zu anderen Elementen) handeln.

Um die verschiedenen Maßstabsebenen miteinander zu verknüpfen, wird eine hierarchische Verschachtelung der Elemente vorgeschlagen. Dies bedeutet, dass sich innerhalb eines Elementes andere Elemente befinden können. Die Tiefe der Verschachtelung kann dabei beliebig sein. So umfasst beispielsweise ein Grundstück Häuser, innerhalb derer sich Wohnungen befinden, die wiederum Räume enthalten, usw. Dies ermöglicht es dem Nutzer des Systems, ein komplexes Entwurfsproblem mit vielen Abhängigkeiten zu definieren. Eine genaue Beschreibung zu dieser hierarchischen Gliederung findet sich in Kapitel 10.

[6] Bei den in diesem Buch dargestellten Beispielen werden ausschließlich 2-dimensionale Objekte (Rechtecke, Polygone) verwendet.

3.4.2. Regeln zur Formgenerierung

Mittels EA kann die Suche nach Lösungen nur in einem gewissen Rahmen geschehen. Dieser Rahmen ist definiert durch die im Generierungsalgorithmus festgeschriebenen Regeln zur Erzeugung einer Lösung. Damit sich die Lösungssuche nicht im Überprüfen von Trivialfällen erschöpft, sollten die Generierungsregeln ein möglichst großes Spektrum an Lösungen ermöglichen (große Varianz). Dadurch zeichnet sich letztlich auch die Fähigkeit eines Systems aus, kreative Lösungen zu erzeugen.

Die Herausforderung bei der Definition der Regeln zur Formerzeugung hinsichtlich der Nutzerinteraktion besteht darin, eine große Formvielfalt erzeugen zu können und gleichzeitig den Suchraum so einzugrenzen, dass der Aufwand für die Suche nach Lösungen so gering wie möglich gehalten wird. Dies ist wichtig, um dem Nutzer Lösungen nach möglichst kurzer Rechenzeit zu liefern (siehe 3.2.4). Verschiedene Ansätze zur Erzeugung von Layouts finden sich in den Kapiteln 5 – 8. Dabei kann grundsätzlich zwischen additiven und dividierenden Verfahren unterschieden werden. Die Vor- und Nachteile dieser Ansätze werden in den jeweiligen Kapiteln beschrieben.

3.4.3. Evaluationskriterien

Neben der Erzeugung einer möglichst großen Vielfalt geometrischer Varianten ist die Einbeziehung verschiedenster Evaluationskriterien notwendig, um beispielsweise Lösungen mit bestimmten funktionalen Eigenschaften zu finden. Bei diesen Kriterien lassen sich für den vorliegenden Fall zwei Kategorien unterscheiden. Erstens Kriterien, die sich problemlos auf verschiedenen Maßstabsebenen anwenden lassen. Diese werden hier als kontextunabhängige Kriterien bezeichnet. Dazu zählen beispielsweise die Überlappung von Elementen, Abstände zwischen Elementen, Orientierung von Elementen (z.B. hinsichtlich Himmelsrichtungen), topologische Relationen sowie die Proportion und Größe von Elementen. Zweitens finden sich Kriterien, die sich nur speziell auf einer Maßstabsebene oder einen Elementtyp anwenden lassen. Diese werden hier kontextabhängige Kriterien genannt. Solche Kriterien sind beispielsweise Flächennutzungskennwerte, Nutzungsdichten, Abstandsflächen, Verschattungen.

Da die Kriterien zur Evaluation abhängig sind von den Absichten des Entwerfers bzw. der Interpretation des dargestellten Layouts, also der Bedeutung, die die Elemente für einen Nutzer haben, müssen die Evaluationskriterien flexibel zugewiesen und gewichtet werden können. Das Gewichten der unterschiedlichen Kriterien ist insbesondere von Bedeutung, da es dem Entwerfenden erlaubt, den Lösungsraum in bestimmte Richtungen zu durchsuchen und so Erkenntnisse über das zu entwerfende Objekt zu gewinnen. So könnte man beispielsweise herausfinden, ob sich eine bestimmte Grundflächenanzahl in einem Stadtquartier unter Einhaltung einer bestimmten Maximalhöhe und Mindestabstand der Gebäude realisieren lässt.

3.5. Zusammenfassung

Im vorliegenden Kapitel wurden Anforderungen an ein ALES formuliert. Diese Anforderungen wurden auf Grundlage der Überlegungen zu Entwurfsproblemen im Allgemeinen als auch zu Charakteristika von Entwurfsprozessen im Speziellen erarbeitet. Da die Lösung von Entwurfsproblemen die Bearbeitung operationalisierbarer als auch nicht-operationalisierbarer Aspekte verlangt, ist ein entscheidender Faktor für das Design eines Entwurfssystems die Einbeziehung des Nutzers. Um diese möglichst reibungslos zu gestalten, ist die Berücksichtigung spezifischer Charakteristika von Entwurfsprozessen notwendig. Wesentliche Aspekte, die daraus abgeleitet wurden, sind die Adaptivität, Zirkularität, Explorativität und Unmittelbarkeit.

Das Gesamtkonzept für die Entwicklung eines ALES kann, wie in Abb. 8 dargestellt, zusammengefasst werden. Es besteht im Wesentlichen aus vier Teilen: dem generativen System, der Grafischen Ausgabe, dem Nutzer und den Interaktions- bzw. Eingabemöglichkeiten. Das generative System erzeugt Lösungen, welche den in einer Problembeschreibung definierten (operationalen) Anforderungen genügen. Diese Lösungen werden dem Nutzer grafisch dargestellt. Auf Basis dieser Darstellung kann der Nutzer die Lösungen (auf Grundlage nicht-operationaler Aspekte) bewerten. Über verschiedene Interaktionsmöglichkeiten können Änderungen an der Problembeschreibung vorgenommen werden.

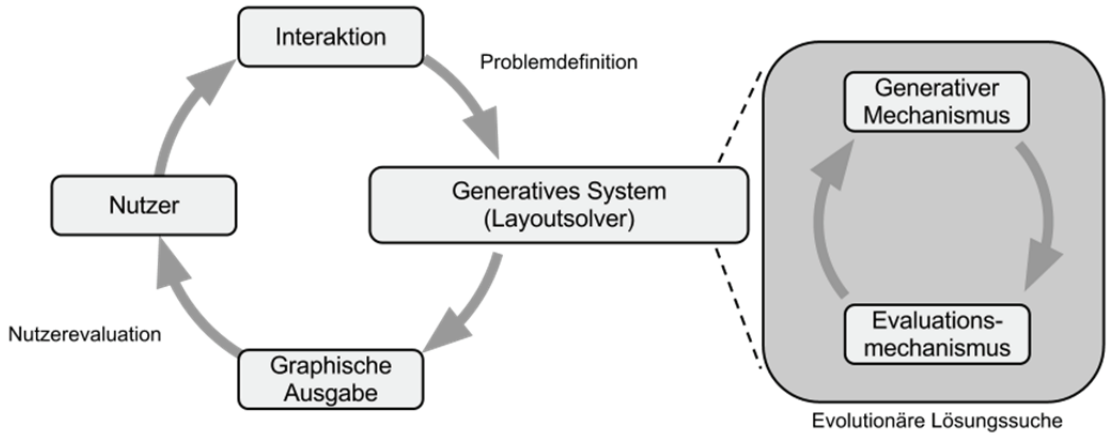


Abb. 8: Design für die Entwicklung eines ALES.

Das generative System, welches aus einem generativen Mechanismus und einem Evaluationsmechanismus besteht (Abb. 8), wird im Folgenden auch als Layoutsolver bezeichnet, da wir uns in den restlichen Kapiteln primär mit der Lösung von Layoutproblemen befassen.

4. Evolutionäre Algorithmen und formale Konventionen

Reinhard König, Katja Knecht

Es wurde bereits unter Punkt 1.2 dargestellt, dass man es bei Entwurfsaufgaben mit nicht-operationalen Fragen zu tun hat. Für eine analytische Bearbeitung dieser Aufgaben ist die Auseinandersetzung mit einer großen Anzahl sich teilweise widersprechender Zielstellungen, mit vielen Einschränkungen und mit einer Fülle vielfältiger Parameter notwendig (Parmee, 1999). Zur Handhabung dieses komplexen Sachverhalts ist eine Strategie erforderlich, welche eine flexible Erkundung des Suchraums ermöglicht. Als Suchmethode eignen sich Evolutionäre Algorithmen (EA), da: *"Using such methods [EA] we have turned evolution into an explorer of what is possible, instead of an optimizer of what is already there"* (P. J. Bentley & Corne, 2002, S. 54).

Aufgrund der für unsere Zwecke positiven Eigenschaften von EA werden diese bei allen in diesem Buch dargestellten Methoden zur Layoutgenerierung verwendet (siehe Kapitel 5 bis 8). Aus diesem Grund widmen wir dieses Kapitel der Beschreibung verschiedener EA und deren bis heute erprobten Anwendungen für Entwurfs- und Planungsprobleme sowie der Darstellung ihrer Funktionsweise.

4.1. Beschreibung evolutionärer Algorithmen

Bei EA handelt es sich um sogenannte heuristische Methoden, die im Einzelfall die Lösung einer Aufgabe nicht garantieren, wohl aber den Zeitaufwand zur Problemlösung erheblich verringern. EA, die als Nachbildung der biologischen Evolution, dem kreativsten aller bekannten Prozesse, verstanden werden können, sind beim derzeitigen Stand der Forschung die einzig verfügbare computerbasierte Methode für die Lösung von schlecht definierten Problemen (P. J. Bentley & Corne, 2002). Sie ermöglichen es, neue, nicht schon in den Vorgaben enthaltene Lösungen zu finden. Unter EA werden die folgenden vier Algorithmen subsumiert. Eine detaillierte Beschreibung der vier EA findet sich unter anderen bei Bentley und Corne (2002).

4.1.1. Genetischer Algorithmus (GA)

Der von Holland (1973, 1992) entwickelte GA wird mittlerweile für eine Vielzahl sehr unterschiedlicher Probleme verwendet. Eine der wichtigsten Eigenschaften eines GA ist die Trennung von Such- und Lösungsraum (Abb. 9). Der Suchraum beinhaltet die kodierten Lösungen, die Genotypen, welche durch Kreuzung und Mutation variiert werden. Die kodierten Lösungen werden anhand eines Verfahrens, das als Mapping bezeichnet wird, in den Lösungsraum überführt und bilden dort die Phänotypen, die dem Selektionsprozess ausgesetzt werden (Abb. 9). Alle zu einem Zeitpunkt vorhandenen Phäno- bzw. Genotypen bilden die Individuen einer Generation. Geeignete Individuen werden in die nächste Generation überführt. Welche Individuen als geeignet betrachtet werden, wird durch eine Fitnessfunktion bestimmt. GAs zeichnen sich durch ihre Robustheit aus, d.h., sie liefern auch bei schlechter Implementierung gute Ergebnisse (Goldberg, 1989).

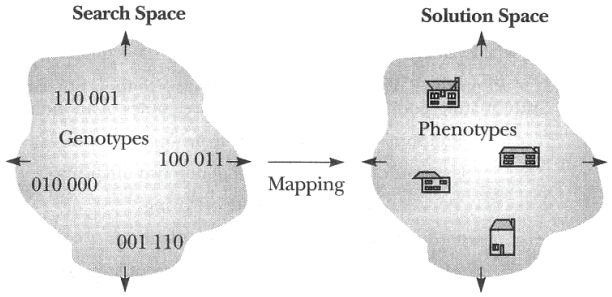


Abb. 9: Mapping der Genotypen des Suchraums auf die Phänotypen des Lösungsraums. Abbildung aus Bentley und Corne (2002, S. 11).

4.1.2. Genetische Programmierung (GP)

Die GP wurde von Koza (1992) begründet und ist in ihrer Funktionsweise der des GA sehr ähnlich. Der Unterschied besteht darin, dass bei der GP nicht nur Parameterwerte, sondern auch Teile von Funktionen oder Programmen zusammengefasst und kombiniert werden können.

4.1.3. Evolutions-Strategie (ES)

Die in den 1960er Jahren von Bienert, Rechenberg und Schwefel ausgearbeitete ES (Bäck, 1994) ist relativ einfach zu implementieren und hinsichtlich der Rechenzeit ein vergleichsweise schneller EA. Bei der ES werden die Genotypen nicht mittels binärer Kodierung repräsentiert, sondern anhand der unkodierten Parameterwerte

(in der Regeln in Form von Dezimalwerten). Die Variation mittels Kreuzung und Mutation wird direkt auf die Parameterwerte angewandt. Dadurch ist kein De- und Entkodierungsverfahren wie bei GA erforderlich. Folglich unterscheidet sich die ES vom GA hauptsächlich dadurch, wie die Parameterwerte repräsentiert werden.

4.1.4. Evolutionäre Programmierung (EP)

Die EP geht auf Lawrence Fogel zurück (Fogel, Owens, & Walsch, 1966). Sie ist der ES sehr ähnlich, aber parallel zu dieser entwickelt worden. Sie wird an dieser Stelle nur der Vollständigkeit halber erwähnt. Wir verwenden in den in diesem Buch dargestellten Methoden statt der EP die ES.

4.2. Anwendungen von EA in der Planung

Im Bereich der Architektur wurden die ersten Experimente mit EA von Frazer (1974, 1995) veröffentlicht. Die Ergebnisse dieser Studien haben allerdings noch sehr abstrakten Charakter und sind von rein akademischem Interesse. Auch die Arbeiten, die im Umfeld von Paul Coates in den 1990er Jahren am CECA[7] entstanden sind (Broughton, Tan, & Coates, 1997; Coates & Hazarika, 1999), resultieren in abstrakten räumlichen Strukturen, die als Inspiration für eine weitere Ausarbeitung einer Entwurfslösung dienen. Erste überzeugende Beispiele für die Anwendung von EA im Bereich der computerbasierten Grundrissentwicklung finden sich bei Jo und Gero (1998) sowie bei Rosenman (1997). Allerdings basieren diese frühen Beispiele noch auf einem rechtwinkligen Raster, wodurch die mögliche Geometrie stark eingeschränkt ist.

Unter den vielfältigen computerbasierten Grundrissentwicklungssystemen, die in den letzten Jahren entstanden sind (Flemming & Woodbury, 1995; Hower & Graf, 1996; Medjdoub & Yannou, 2001), zählt die Arbeit von Elezkurtaj und Franck (2002), die ein additives Verfahren benutzt (Abb. 10), in mehrerlei Hinsicht zu den am besten ausgearbeiteten. Erstens berechnet das System Entwurfsvorschläge sehr schnell, wodurch zweitens eine sinnvolle Interaktionsmöglichkeit zwischen der generativen Software und dem Nutzer ermöglicht wird, die ein reibungsloses Wech-

[7] Center for Evolutionary Computing in Architecture an der University of East London: http://uelceca.net/

selspiel zwischen Entwerfer und Computer erlaubt. Drittens kann der Entwurf kontinuierlich weiterentwickelt werden. Viertens kann die Methode auch für städtebauliche Entwurfsaufgaben verwendet werden (Elezkurtaj & Franck, 2001).

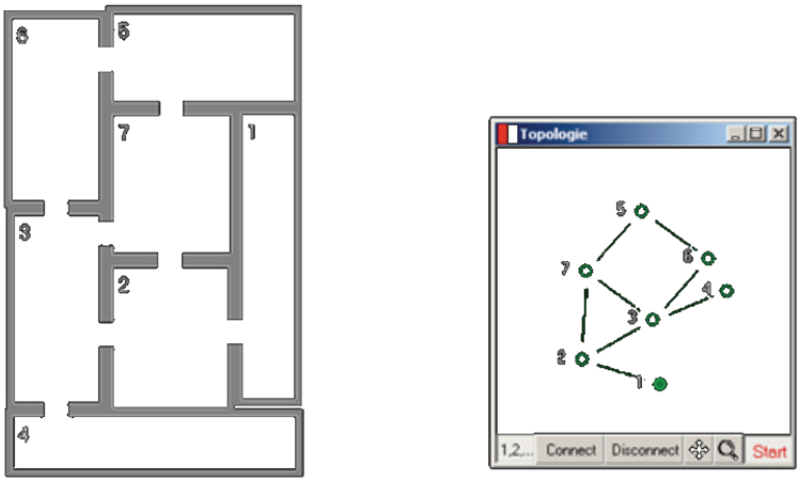


Abb. 10: Tomor Elezkurtaj, Interaktives Layout System. Abbildung aus (2002).

Beim Einsatz von EA für die Layoutplanung muss man im Allgemeinen zwischen der Optimierung vorgegebener Layouts und der Generierung neuer Layouts unterscheiden (Coates et al, 1999). Die zweitgenannte Anwendung stellt hierbei die ungleich komplexere dar, da der kreative Entwurfsprozess die Lösung sowohl operationaler wie auch nicht operationaler Kriterien umfasst (vgl. Punkt 1.2).

Für die Grundlagen und Hintergründe zu verschiedenen EA mit Bezug zu gestalterischen Fragestellungen sind die Texte von Bentley (1999) sowie Bentley und Corne (2002) besonders empfehlenswert.

4.3. Formale Konventionen

Für ein besseres Verständnis der Funktionsweise von EA, insbesondere ES, die im Rahmen der folgenden Kapitel in verschiedenen Varianten eingesetzt werden, betrachten wir im Folgenden die wichtigsten Grundlagen in Form eines allgemeinen formalen Rahmens. Wir haben uns bemüht, alle formalen Zusammenhänge so einfach wie möglich darzustellen. Die formale Notation ist notwendig für ein tieferes Verständnis der Mechanismen der Algorithmen und für eine kompakte Darstellung aller notwendigen Begriffe und deren Zusammenhänge. Zusätzlich werden stets alle mathematischen Beschreibungen in möglichst allgemeinverständlicher Sprache be-

schrieben. Wir orientieren uns an den formalen Darstellungen bei Bäck, Hoffmeister und Schwefel (1991), Deb (2001) sowie Bäck (2000b).

Bei den in diesem Buch vorgestellten Verfahren beginnen wir in der Regel mit solchen, die auf die wesentlichen Mechanismen beschränkt sind, und erweitern diese Schritt für Schritt. Dieses Vorgehen soll deutlich machen, welche Effekte bestimmte Erweiterungen der grundlegenden Verfahren haben und wie sinnvoll diese sind.

Wir beginnen hier mit einer Auflistung der wichtigsten formalen Elemente (Tab. 1), welche sich auf eine sogenannte (μ+λ)-ES beziehen. Auf die Bedeutung dieser Schreibweise wird später eingegangen.

ES verwenden Populationen P von Individuen a. Die Variablen μ und λ bezeichnen die Anzahl der Eltern- und Kinderindividuen in einer Population. $P^t=(a^t_1, ..., a^t_\mu)$ charakterisiert eine Population in Generation t.

Die einfachste Form stellt somit die (*1*+*1*)-ES$=(P^\circ, m, s, c_d, c_i, f, g, t)$ dar, bei der ein Elternindividuum kopiert und diese Kopie mutiert wird. In die nächste Generation wird das laut Bewertungsfunktion bessere (fittere) Individuum übernommen. Demzufolge kann die (*1*+*1*)-ES als eine Art wahrscheinlichkeitsbasiertes Gradientenverfahren betrachtet werden (Bäck, et al., 1991).

$P^\circ = (x^\circ, \sigma^\circ) \in I$	*Population;* $I = R^n \times R^n$
$a \in \mathbb{N}$	*Individiuum*
$X \in \mathbb{N}$	*Objektparameter*
$m: I \rightarrow I$	*Mutationsoperator*
$s: I^\lambda \rightarrow I^\mu$	*Selektionsoperator*
$r: I^\mu \rightarrow I$	*Rekombinationsoperator*
$c_d, c_i \in \mathbb{R}$	*Schrittweitenkontrolle*
$f: \mathbb{R}^n \rightarrow \mathbb{R}$	*Zielfunktion*
$g_j: \mathbb{R}^n \rightarrow \mathbb{R}$	*Nebenbedingungen* $j \in \{1, ..., q\}$
$t: I \times I \rightarrow \{0,1\}$	*Abbruchbedingung*
$\mu \in \mathbb{N}$	*Anzahl der Eltern*
$\lambda \in \mathbb{N}$	*Anzahl der Kinder*
$\rho \in \mathbb{N}$	*Anzahl Eltern für eine Rekombination*
$\Delta\sigma \in \mathbb{R}$	*Meta – Schrittweitenkontrolle*
$\sigma \in \mathbb{R}$	*Standardabweichung*

Tab. 1: Häufig verwendete formale Elemente.

Mit der Nomenklatur (μ+λ)-ES wird angegeben, dass aus μ Eltern λ Kinder erzeugt und mittels Selektionsoperator s wieder auf μ Eltern der nächsten Generation reduziert werden. In allen Varianten der (μ+λ)-ES werden die Eltern zusammen mit den Kindern bewertet und selektiert. Folglich kann bei der (μ+λ)-ES ein Individuum so lange überleben, bis es von einem besseren Nachkommen verdrängt wird.

Eine ES optimiert nach einer Bewertungsfunktion f in Bezug auf eine Menge von Objektparameter $\boldsymbol{X}=(X_1, X_2 \ldots X_i)$ (X_i werden auch Entscheidungsvariablen genannt):

$$f(\boldsymbol{X}) \rightarrow Optimum \tag{1}$$

Die Anzahl der Objektparameter X_i definieren die Anzahl der Dimensionen eines Suchraums. Je mehr Objektparameter in der Bewertungsfunktion vorkommen, desto komplizierter und langwieriger ist in der Regel die Suche nach einem Optimum, da jede Dimension die Anzahl möglicher Variablenkombinationen potenziert. Die Geschwindigkeit, mit der sich ein EA einem Optimum annähert, wird als Konvergenzgeschwindigkeit bezeichnet. Ziel bei der Anwendung eines EA ist, dass dieser gegen ein globales Optimum konvergiert und nicht in lokalen Optima hängen bleibt.

Ein Individuum a_k mit dem Index k umfasst in der einfachsten Form einer ES den spezifischen Satz der Objektparameter $\boldsymbol{X}_k$ und eine Bewertungsfunktion $f_k(\boldsymbol{X})$:

$$a_k := \left(\boldsymbol{X}_k, f_k\left(\boldsymbol{X}\right)\right) \tag{2}$$

Bei einer Mutation wird zu jedem Objektparameter eines Individuums ein zufälliger Wert addiert:

$$X_i' = X_i + \mathrm{N}_0(\sigma) \tag{3}$$

Die Zufallszahlen werden anhand einer Normalverteilung $N_0(\sigma)$ generiert. Die Angaben zur Normalverteilung N bedeuten, dass zufällige Werte mit dem Erwartungswert 0 und der Standardabweichung σ erzeugt werden. Bei der Anwendung für ES wird die Standardabweichung σ als Mutationsschrittweite bezeichnet. Existiert nur eine globale Mutationsschrittweite σ, werden alle Individuen mit dieser mutiert. Durch eine geschickte Adaption der Schrittweite kann der Erfolgsfaktor einer Mutation verbessert werden (Kramer, 2008).

Angenommen, die Bewertungsfunktion f soll minimiert werden, können wir die einfachste Form einer (*1*+*1*)-ES mit folgendem iterativen Schema definieren:

$$P^{t+1} = s\left(P''^{t}\right) = \begin{cases} a_2'^{t} & if \begin{cases} f\left(\boldsymbol{X}''^{t}\right) \le f\left(\boldsymbol{X}^{t}\right) \wedge \\ g\left(\boldsymbol{X}''^{t}\right) \ge 0 \end{cases} \\ a_1'^{t} \in P^{t} & else \end{cases} \tag{4}$$

Diese ES wird aufgrund ihres Selektionsschemas (4) als (*1*+*1*)-ES bezeichnet, da, wie oben beschrieben, die Eltern- und Kinderpopulationen jeweils nur ein Individuum umfassen und beide Populationen für die Selektion verwendet werden.

4.4. Schematische Beschreibung Evolutionärer Algorithmen

Alle in diesem Buch beschriebenen EA besitzen eine gemeinsame Struktur, die aus einem einfachen Kreislauf aus Rekombination, Mutation, Evaluation und Selektion von Individuen einer Population besteht. Auf deren Basis lässt sich ein allgemeingültiges Ablaufschema für EA erstellen, das nach Bäck (2000b), wie in Tab. 2 beschrieben, dargestellt werden kann.

Algorithmus:	Evolutionärer Algorithmus
Input:	μ – Größe der Elternpopulation λ – Größe der Kinderpopulation r – Rekombinationsoperator m – Mutationsoperator s – Selektionsoperator ι – Abbruchkriterium
Output:	a^* das beste Individuum während des Durchlaufs P^* die beste Population während des Durchlaufs
Logik:	*Schritt 1:* Generation t = 0 *Schritt 2:* Initialisiere $P(t)$ mit μ Individuen *Schritt 3:* Evaluiere alle Individuen in $P(t)$ mit der Evaluationsfunktion $F(t)$ *Schritt 4:* Rekombiniere $P(t)$ mittels $r \rightarrow P'(t)$ *Schritt 5:* Mutiere $P'(t)$ mittels $m \rightarrow P''(t)$ *Schritt 6:* Evaluiere alle Individuen in $P''(t)$ mit der Evaluationsfunktion $F(t)$

	Schritt 7: Selektiere μ Individuen mittels s aus $P''(t)$ entsprechend ihrer Fitnesswerte $F(t) \rightarrow P(t+1)$
	Schritt 8: $t = t + 1$
	Schritt 9: Beginne wieder bei *Schritt 4* solange $\iota \neq$ true

Tab. 2: Evolutionärer Algorithmus nach Bäck (2000b).

Zunächst erfolgt die Initialisierung des Algorithmus mit Generation $t = 0$ und der Bildung einer Ausgangspopulation $P(t)$. $P(t)$ wird mit μ Individuen initialisiert, die jeweils einen Punkt des Lösungsraums repräsentieren. Die Evolutionsschleife beginnt mit der Bildung von Nachkommen durch Rekombination von Individuen der Ausgangspopulation mit dem Rekombinationsoperator r und Mutation mit dem Mutationsoperator m. Diese generierten Nachkommen bilden die Kinderpopulation $P''(t)$ der Größe λ und werden im nächsten Schritt mithilfe der Evaluationsfunktion $f(t)$ bewertet. Durch Selektion mit dem Selektionsoperator s wird aus den Nachkommen in $P''(t)$ die neue Population $P(t)$ der Generation $t+1$ gebildet. Die Evaluationsschleife wird so lange ausgeführt, bis ein vorher definiertes Abbruchkriterium ι erreicht ist. Als Abbruchkriterium kann beispielsweise eine maximale Anzahl an Durchläufen oder das Erreichen einer Lösung mit bestimmten Eigenschaften definiert werden. Das Abbruchkriterium sowie die Charakteristiken und Parameter der Rekombination, Mutation und Selektion unterscheiden sich Algorithmus spezifisch bzw. von Implementierung zu Implementierung.

II. Methoden

5. Layouts mittels Dichter Packung[8]

Reinhard König

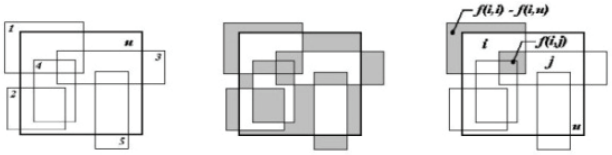
In diesem Kapitel wird eine Methode dargestellt, die es erlaubt, Grundrisslayouts anhand des Prinzips der dichten Packung geometrischer Elemente zu erzeugen. Mittels ES werden die Überlappungsflächen minimiert und verschiedene Heuristiken so eingesetzt, dass eine Hybride ES entsteht. Es wird die Performance verschieden komplexer EA miteinander verglichen und untersucht, wie diese auf Störungen reagieren. Darüber hinaus werden als weiteres Kriterium topologische Beziehungen zwischen den Elementen berücksichtigt, sodass geforderte Nachbarschaftsbeziehungen zwischen Räumen in einem Grundriss hergestellt werden können. Abschließend erfolgt eine Analyse des erstellten Systems hinsichtlich der Zuverlässigkeit und Geschwindigkeit, mit der Lösungen für ein bestimmtes Layoutproblem gefunden werden können.

5.1. Dichte Packung

Das Problem der dichten Packung tritt auf, wenn eine bestimmte Anzahl räumlicher Elemente innerhalb eines gegebenen Raums überlappungsfrei und möglichst lückenlos angeordnet werden muss. Die Elemente und der umgebende Raum können unveränderbare Größen haben, wie es beispielsweise bei einer dichten Packung von Frachtkisten in einem Lastwagen der Fall ist. Beim Grundrisslayout können sowohl die zu packenden Elemente, die Räume, als auch der umfassende Raum, das Gebäude, innerhalb eines gewissen Spielraums variieren. Bei dem in diesem Abschnitt behandelten Testszenario gehen wir von einem festgelegten Gebäudeumriss aus und befassen uns mit der zweidimensionalen Packung einzelner Räume, die in ihren Abmessungen innerhalb definierter Mindest- und Höchstgrenzen flexibel sind, aber immer einen gegebenen Flächeninhalt einhalten müssen. Die Summe der Flächeninhalte der zu packenden Räume entspricht dabei dem Flächeninhalt des Gebäudeumrisses.

[8] Dieses Kapitel basiert auf dem Artikel von Koenig, R. (2011b). *Generierung von Grundriss-Layouts mittels hybrider Evolutions-Strategie*. Weimar: Bauhaus-Universität Weimar.

Das zu lösende Problem wurde von Elezkurtaj und Franck (2001, 2002) formal folgendermaßen beschrieben (Abb. 11): Zu minimieren ist die Summe aller Schnittflächen Sg. Diese errechnet sich aus der Summe der Schnittflächen aller zu packender Räume ($Si \cap Sj$) und der gewichteten Summe der Schnittflächen, die sich aus der Überlappung der zu packenden Räume mit dem Umgebungsrechteck ergeben ($Si \setminus Su$).



$$Sg = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (S[i] \cap S[j]) + \lambda \sum_{i=1}^{n} (S[i] \setminus S[u])$$

Abb. 11: Grafische und formale Darstellung der Berechnung der gesamten Schnittflächen beim Problem der dichten Packung. Abbildung aus Elezkurtaj und Franck (2001).

5.1.1. Kollisionserkennung

Zur algorithmischen Lösung des in Abb. 11 dargestellten Problems bedienen wir uns zuerst des Verfahrens der Kollisionserkennung zwischen zwei geometrischen Elementen (wir beschränken uns hier auf Rechtecke) mit dem Ziel, dass sich diese voneinander abstoßen und dadurch ihre Überlappungen verringern.

Um zu prüfen, ob zwei Rechtecke miteinander kollidieren bzw. sich überschneiden, wird das *„Separating Axis Theorem"* verwendet. Bei diesem wird überprüft, ob es eine Linie gibt, die beide Rechtecke voneinander separiert, die also ohne Überschneidung zwischen den Rechtecken verläuft. Existiert so eine Line, überschneiden sich die beiden Rechtecke nicht.

Überschneiden sich zwei Rechtecke, wie in Abb. 12 dargestellt, wird geprüft, welcher Abprall-Vektor (v_x oder v_y) der kleinste ist, sodass nach einer Verschiebung beider Rechtecke um je die Hälfte dieses Vektors in je verschiedene Richtungen (v_i und $-v_i$) keine Überlappung mehr auftritt. Das beschriebene Beispiel wurde anhand des Algorithmus von Cozic (2006) implementiert.

Das voneinander Abprallen der Rechtecke ist eine effiziente Heuristik, um die Summe aller Schnittflächen S_g zu verringern. Allerdings können auf diese Weise nur

die Positionen der Rechtecke angepasst werden. Für eine überlappungsfreie Packung der Rechtecke ($S_g = 0$) ist es notwendig, auch die Proportionen, also Länge und Breite der Rechtecke zu verändern.

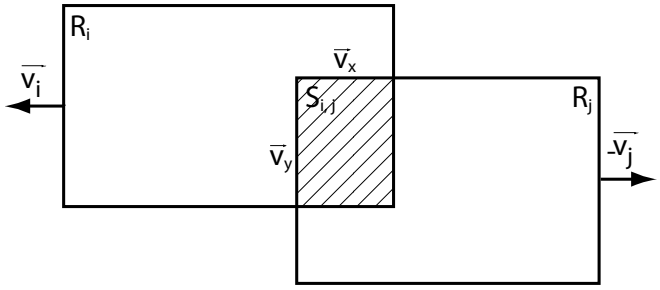


Abb. 12: Berechnung der Abstoßungs-Vektoren zweier sich überlappender Rechtecke.

Für die Anpassung der Proportionen nutzen wir das gleiche Verfahren wie für das Abprallen der Rechtecke. Der einzige Unterschied besteht darin, dass in diesem Fall die Vektoren v_i nicht für das Verschieben der Räume verwendet werden, sondern für die Änderung der Höhe oder Breite der Rechtecke. Bei dem Beispiel in Abb. 12 würde die Breite von R_i um die Hälfte des Betrags des Vektors v_i verringert und gleichzeitig die Höhe von R_i erhöht, sodass die Fläche F_i nach der Proportionsänderung konstant bleibt. Genauso wird beim zweiten Rechteck R_j verfahren.

Für die Umsetzung des bisher beschriebenen Verfahrens werden zusätzlich folgende Operatoren eingeführt: Die Gewichtung β für die Abprall-Vektoren, die Wahrscheinlichkeit ρ_L, mit der im Falle einer Überlagerung zweier Rechtecke die Abprallfunktion ausgeführt wird (Lokalisationsänderung), die Gewichtung α für die Proportionsänderungsvektoren sowie die Wahrscheinlichkeit ρ_G, mit der im Falle einer Überlagerung zweier Rechtecke die Proportionsänderungsfunktion ausgeführt wird. Zusammenfassend können wir die beiden wichtigsten Funktionen für die Anpassung der Rechtecke folgendermaßen darstellen:

$$\begin{bmatrix} \triangle G_i^t = \overrightarrow{v''_{iG}} = \alpha * \overrightarrow{v_i^t} & if \quad \rho_G > U \\ \triangle G_i^t = 0 & else \\ \triangle L_i^t = \overrightarrow{v'}^t_{iL} = \beta * \overrightarrow{v_i^t} & if \quad \rho_L > U \\ \triangle L_i^t = 0 & else \end{bmatrix}$$

$$\overrightarrow{v'_i} = -\overrightarrow{v'_j} = \begin{cases} \begin{pmatrix} 0.5 * v'_{xi} \\ 0 \end{pmatrix} & if \; v'_{xi} > v'_{yi} \\ \begin{pmatrix} 0 \\ 0.5 * v'_{yi} \end{pmatrix} & else \end{cases} \qquad (5)$$

wobei ΔG_i die Proportionsänderung und ΔL_i die Lokalisationsänderung zum Zeitpunkt t angeben. U ist eine gleichverteilte Zufallszahl auf dem Intervall [0, 1], die bei jedem Aufruf neu gezogen wird.

Mit dem in Abb. 13 dargestellten Softwaremuster können dichte Packungen mittels Abprall- und Proportionsänderungsfunktion hergestellt werden. Wir verwenden ein Testszenario mit einem Umgebungsquadrat von 600 x 600 Pixel und zehn einzupassenden Rechtecken gleichen Flächeninhalts. Die Einheit der Werte für S_g sind dementsprechend Pixel, werden im Folgenden aber nicht mehr angegeben, da die dargestellten Werte lediglich dem Vergleich verschiedener Algorithmen dienen. Das in diesem Abschnitt beschriebene Verfahren liefert erste brauchbare Ergebnisse bei den Parametereinstellungen α=0,1; ρ_G=0,5; β=0,5; ρ_L=1.

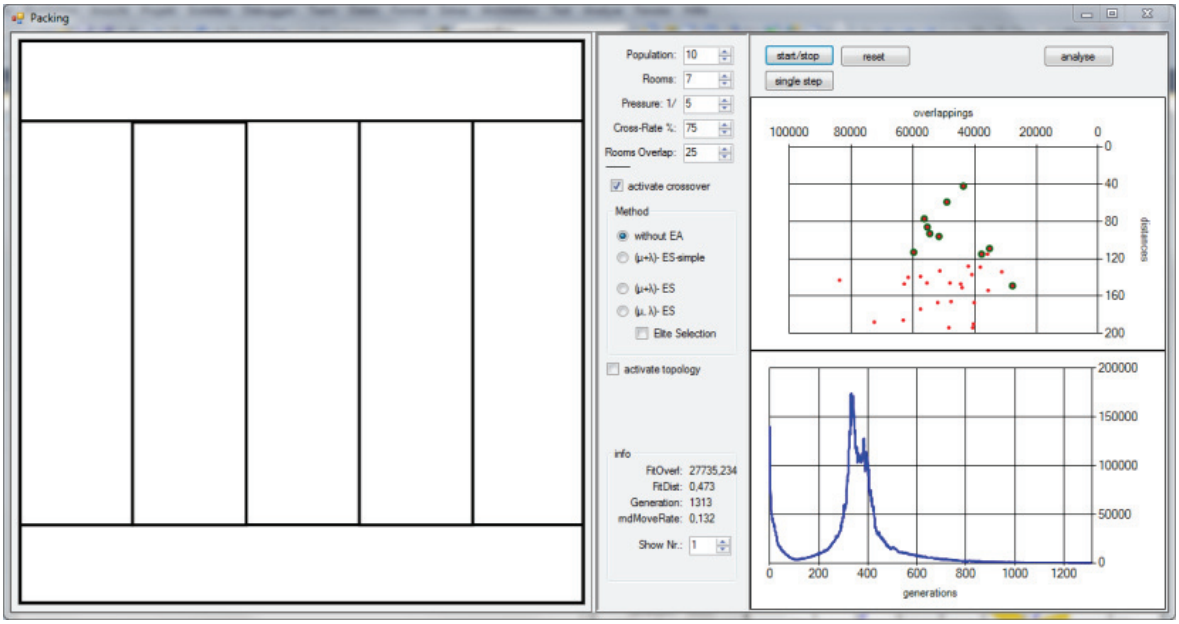


Abb. 13: Beispiel für ein automatisch generiertes Layout (rechts) mittels Abprall- und Proportionsänderungsfunktion. Das Diagramm unten rechts plottet die Überlappungswerte (y-Achse) gegen die Generationen (x-Achse). Das Programm wurde bis Generation t = 1300 ausgeführt.

Aus dem Diagramm in Abb. 13 (rechts unten) ist zu erkennen, dass ca. zwischen t=200 und t=400 eine starke Zunahme der Überlappungswerte zu verzeichnen ist, obwohl mit dem oben beschriebenen Verfahren beabsichtigt ist, diese Werte kontinuierlich zu verringern. Dieser Umstand kommt dadurch zustande, dass es bei der gleichzeitigen Anwendung der beiden Funktionen für ΔG_i und ΔL_i (5) zu Rückkopplungen kommen kann, durch die bei bestimmten Konstellationen der Rechtecke unerwünschte Veränderungen aufgeschaukelt werden.

Für eine bessere Bewertung der Systemeigenschaften des in diesem Abschnitt dargestellten Verfahrens führen wir eine Analysemethode (Performancetest) ein, bei der das in Abb. 13 dargestellte Layoutprogramm mehrfach ausgeführt wird und die Überlappungswerte aller Programmdurchläufe in einem Diagramm zusammengefasst werden (Abb. 14).

In Abb. 14 ist zu erkennen, dass die Rückkoppelungseffekte nicht bei allen Durchläufen auftreten. An der Mittelwertkurve ist allerdings gut abzulesen, dass die Rückkoppelungen im Durchschnitt zu keinen brauchbaren Ergebnissen führen und dass die Zeit bzw. die Anzahl zu durchlaufender Generationen t relativ groß ist, um gute Lösungen zu erreichen. Um diese Probleme zu beheben, führen wir im nächsten Schritt eine einfache ES ein.

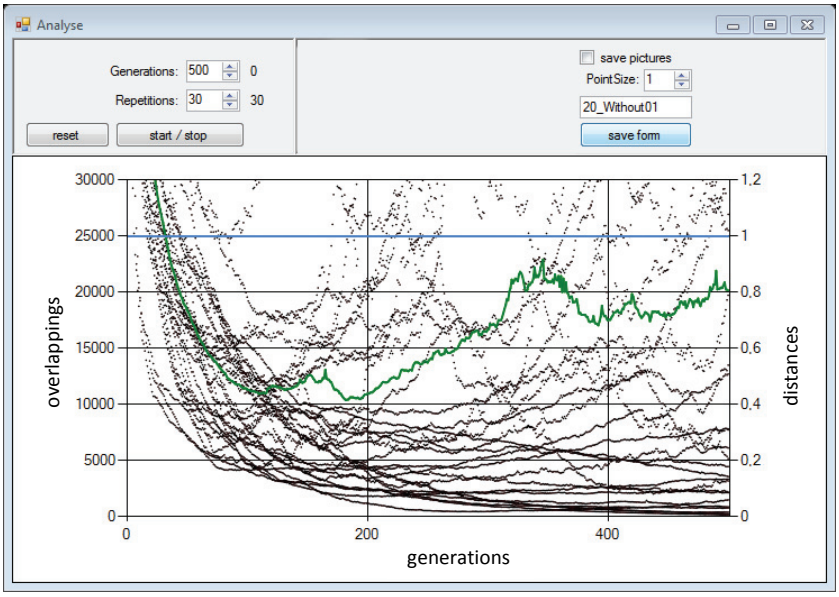


Abb. 14: Analysediagramm für die Layout-Generierung mittels Kollisionserkennung nach 30 Durchläufen des Programms mit jeweils t=500. Die durchgezogene grüne Linie stellt die Mittelwertkurve dar.

Bevor wir uns mit den Möglichkeiten der ES befassen, soll anhand des Diagramms in Abb. 14 beschrieben werden, welche Eigenschaften das gesuchte computerbasierte Layout-System aufweisen soll. Wesentlich für eine Interaktion mit einem Nutzer ist die Konvergenzgeschwindigkeit, mit der das System brauchbare Lösungsvorschläge anbietet. Wir werden weiter unten sehen, dass die Geschwindigkeit für eine akzeptable Interaktion dadurch angegeben werden kann, dass nach ca. 5 Sekunden (im Folgenden ca. 20 Generationen auf dem verwendeten Testcomputer) der Wert

für die Summe aller Überlappungen eines Layouts bei ca. S_g=5000 liegen sollte. Mit in etwa der gleichen Geschwindigkeit sollte das System auf die Interaktionen eines Nutzers reagieren, die als Störungen eines einmal gefundenen optimierten Layouts aufgefasst werden können. Betrachten wir unter diesem Gesichtspunkt noch einmal das Diagramm in Abb. 14, können wir feststellen, dass das System im Mittel noch zu träge ist.

5.1.2. (*1+1*) Evolutions-Strategie

Die einfachste Form einer ES ist die sogenannte zweigliedrige (*1+1*)-ES (Rechenberg, 1994). Der Notation aus Kapitel 4.3 folgend bedeutet dies für die Variablen μ=λ=1. Bei der (*1+1*)-ES umfasst eine Generation also lediglich ein Individuum. Von diesem wird eine Kopie angelegt, welche mutiert wird, sodass zwei verschiedene Individuen miteinander verglichen werden können. Die Qualität eines Individuums wird anhand der Bewertungsfunktion f ermittelt. Das Individuum mit der höchsten Qualität wird in die nächste Generation übertragen, das andere gelöscht. Als Bewertungsfunktion dient in unserem Fall die in Abb. 11 angegebene Funktion zur Berechnung von S_g. Je kleiner S_g ausfällt, desto höher ist die Qualität eines Individuums:

$$f(\boldsymbol{X}) \rightarrow \min(S_g) \tag{6}$$

Jedes Individuum a repräsentiert eine Layout-Lösung. Der Satz an Objektparametern X_i umfasst die x- und y-Positionen px und py, Höhen h und Breiten b der Rechtecke eines Layouts. Der Definition in Kapitel 4.3 folgend können wir diese Parameter folgendermaßen angeben:

$$X_i = (px_i,\ py_i,\ h_i,\ b_i) \tag{7}$$

Auf diesen Parametersatz wenden wir nun das iterative Schema (4) der ES an. Die Mutationsoperationen können angegeben werden mit:

$$f(X^t + Z) = \left(px_i^t + Z,\ py_i^t + Z,\ b_i^t + Z\right) \tag{8}$$

Es werden alle Objektparameter mutiert. Die Höhe $h'_i(t)$ ergibt sich nach der Mutation von $b_i(t)$ aus der Division der Rechteckfläche durch die neue Breite, da der Flä-

cheninhalt eines Rechtecks immer konstant bleiben muss. Die Mutationsschrittweite σ für die Normalverteilung $Z \sim N(0, \sigma)$ wird anhand der 1/5-Regel adaptiert, welche besagt, dass das Verhältnis p_s der erfolgreichen Mutationen zu allen Mutationen 1/5 sein sollte. Wenn es größer ist als 1/5, erhöhe σ, wenn es kleiner ist, verringere σ:

$$\sigma^{t+1} = \begin{cases} c_d \cdot \sigma^t, & if \quad p_s^t < 1/5 \\ c_i \cdot \sigma^t, & if \quad p_s^t > 1/5 \\ \sigma^t, & if \quad p_s^t = 1/5 \end{cases} \tag{9}$$

wobei wir nach Schwefel (1995) definieren, dass c_d = 0.82, c_i = 1/0.82. Um den Rechtecken größere Sprünge zu erlauben, führen wir einen weiteren Mutationsoperator ein, der mit der Wahrscheinlichkeit ρ_R die zufällige Positionsänderung mit dem Faktor κ multipliziert:

$$\begin{aligned} px_i'^t &= \begin{cases} px_i^t + \mathrm{N}_0(\sigma) * \kappa & if \quad \rho_Z > U \\ px_i^t & else \end{cases} \\ py_i'^t &= \begin{cases} py_i^t + \mathrm{N}_0(\sigma) * \kappa & if \quad \rho_Z > U \\ py_i^t & else \end{cases} \end{aligned} \tag{10}$$

Wir verwenden für κ=100 und ρ_Z =0.01. U ist eine gleichverteilte Zufallszahl auf dem Intervall [0, 1].

Die Restriktion, dass die Rechtecke nicht über die Grenzen des Umgebungsrechtecks hinausgehen sollen und eine entsprechende Überlappung zu vermeiden ist, kann, wie in Abb. 11 dargestellt, in die Berechnung von S_g einfließen und dadurch für die Minimierung von S_g verwendet werden. Als Alternative zu diesem Minimierungsverfahren kann eine Überlappung mit dem Umgebungsrechteck von vornherein ausgeschlossen werden, indem die Rechtecke im Falle einer Überlappung gezielt verschoben werden, sodass keine Überlappung mit dem Umgebungsrechteck mehr auftritt. Diese Alternative vereinfacht die Berechnung von S_g, die wir im Folgenden in dieser Form verwenden:

$$S_g(\boldsymbol{X}) = f_1^t(\boldsymbol{X}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left(S_i \cap S_j \right); \quad f_1^t(\boldsymbol{X}) \to \min \tag{11}$$

Die gezielte Verschiebung der Rechtecke im Fall einer Überlappung mit dem Umgebungsrechtecke kann folgendermaßen angegeben werden:

$$
\begin{aligned}
px_i'^t &= \begin{cases} px_i^t + (x_{\min} - px_i^t - 0.5 * b_i^t) & if \quad px_i^t - 0.5 * b_i^t < x_{\min} \\ px_i^t + (px_i + 0.5 * b_i^t - x_{\max}) & if \quad px_i^t - 0.5 * b_i^t > x_{\max} \\ px_i^t & else \end{cases} \\
py_i'^t &= \begin{cases} py_i^t + \left(y_{\min} - py_i^t - 0.5 * h_i^t\right) & if \quad py_i^t - 0.5 * h_i^t < y_{\min} \\ py_i^t + (py_i^t + 0.5 * h_i^t - y_{\max}) & if \quad py_i^t - 0.5 * h_i^t > y_{\max} \\ py_i^t & else \end{cases}
\end{aligned}
\tag{12}
$$

wobei die Variablen x_{min}, y_{min}, x_{max} und y_{max} die Grenzen des Umgebungsrechtecks definieren.

Die in diesem Abschnitt beschriebene ES zum Generieren von Layouts allein liefert noch keine akzeptablen Ergebnisse und dient lediglich der Einführung grundlegender Mechanismen. Aus diesem Grund führen wir für das in diesem Abschnitt vorgestellte System keine Performanceanalyse durch. Erwähnenswert ist jedoch, dass die Einführung der 1/5-Regel keine nennenswert positiven Effekte auf die Konvergenzgeschwindigkeit hat, weshalb wir diese bei unseren weiteren Betrachtungen nicht mehr aufgreifen. Im nächsten Abschnitt wird dargestellt, wie die Methoden der Kollisionserkennung und die der ES miteinander kombiniert werden können, um die Performance des Layout-Systems zu verbessern.

5.1.3. Kombination von (μ+λ)-Evolutions-Strategie und Kollisionserkennung

Wir beginnen mit der Kombination einer (*1+1*)-ES mit den Abprall- und Proportionsänderungsfunktionen, wodurch ein sogenannter hybrider Algorithmus entsteht (Jakob, 2004, p. 15), den wir im Folgenden als hybride ES bezeichnen. Mittels dieser Hybridisierung lassen sich die Vorteile der bisher beschriebenen Verfahren kombinieren und deren Nachteile umgehen. Zu diesem Zweck werden die Mutationsoperatoren der ES (8) wie folgt erweitert:

$$
px_i'^t = \begin{cases} px_i^t + \mathrm{N}_0(\sigma) * \kappa & if \quad \rho_{Z1} > U \\ px_i^t + \mathrm{N}_0(\sigma) & if \quad \rho_{Z2} > U \\ px_i^t & else \end{cases}
$$

$$
py_i'^t = \begin{cases} py_i^t + \mathrm{N}_0(\sigma) * \kappa & if \quad \rho_{Z1} > U \\ py_i^t + \mathrm{N}_0(\sigma) & if \quad \rho_{Z2} > U \\ py_i^t & else \end{cases} \qquad (13)
$$

$$
b_i'^t = \begin{cases} b_i^t + \mathrm{N}_0(\sigma) & if \quad \rho_{Z3} > U \\ b_i^t & else \end{cases}
$$

Wir verwenden für κ = 100 und ρ_{Z1} =0.01, ρ_{Z2} =0.5, ρ_{Z3} =0.1. Nachdem die λ Kinder anhand des Zufallsvektors $N(\sigma)$ mutiert wurden, werden auf sie die Anpassungsfunktionen (5) der Kollisionserkennung mit den Parametereinstellungen α=0,1; ρ_G=0,5; β=0,5; ρ_L=1 angewendet. Anschließen werden entsprechend Schema (4) die μ Individuen mit den geringsten Werten für S_g in die nächste Elterngeneration kopiert und die nächste Iteration wird begonnen.

Da nur Individuen mit besserer Fitness (niedrigeren Werten für S_g) in die nächste Generation kopiert werden, kann sich eine einmal gefundene Lösung nicht mehr verschlechtern. Auf diese Weise können unerwünschte Rückkoppelungseffekte, wie sie in Abschnitt 5.1.1 bei der Kollisionserkennung beschrieben wurden, vermieden werden. Das kleine Diagramm in Abb. 15, links, zeigt den entsprechenden Verlauf von S_g. Ferner ist ein typisches Layout der hybriden ES nach t=215 Generationen zu sehen.

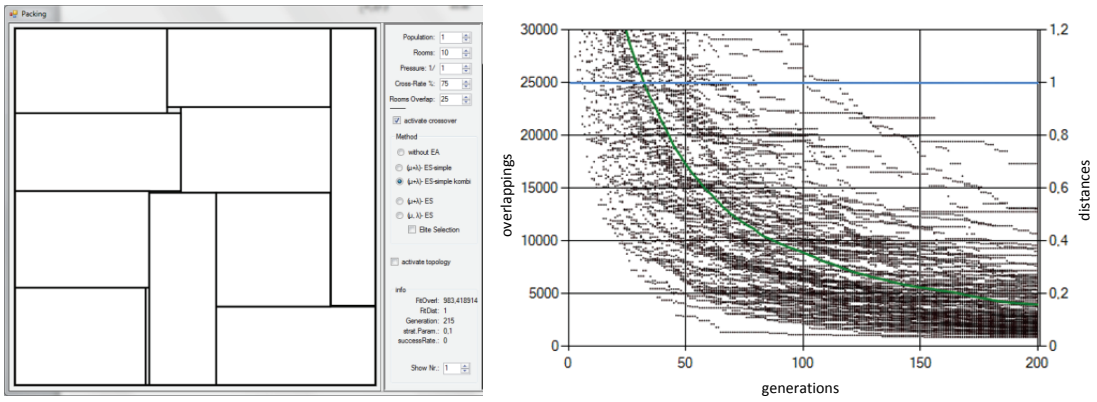


Abb. 15, Links: Typisches Ergebnis der hybriden ES, bestehend aus (1+1)-ES und Kollisionserkennung bei t=215. Rechts: Analysediagramm für die Layout-Generierung mittels hybrider ES nach 100 Durchläufen des Programms mit jeweils t=200 Generationen. Die durchgezogene grüne Linie stellt die Mittelwertkurve dar.

Mittels der hybriden ES kann die Performance des Layout-Systems im Vergleich zu den bisher betrachteten Verfahren deutlich gesteigert werden. Das Ergebnis des Performancetests ist im Diagramm in Abb. 15, rechts dargestellt. Nach t=200 Generationen erreichen wir einen durchschnittlichen Wert für S_g≈4000. Der Verlauf der Mittelwertkurve zeigt allerdings, dass wir von dem oben genannten Ziel, bei t≈20 einen Wert S_g≈5000 zu erreichen, noch weit entfernt sind.

Der soweit entwickelte hybride Algorithmus kann auf verschiedene Arten variiert werden. Eine naheliegende Möglichkeit besteht in der Erhöhung der λ Kinder, die pro Generation erzeugt werden. Im linken Diagramm in Abb. 16 ist das Ergebnis des Performancetests einer hybriden (1+6)-ES zu sehen. Im Vergleich mit der hybriden (1+1)-ES (Abb. 15) wird hier nach t=200 Generationen ein wesentlich besserer durchschnittlicher Wert für S_g≈2700 erreicht. Zudem ist der Verlauf der Mittelwertkurve günstiger, da diese steiler abfällt, was bedeutet, dass die Qualität der Layouts schneller steigt. Eine weitere Verbesserung der Systemperformance kann durch die Erhöhung der μ Eltern erzielt werden. Das mittlere Diagramm in Abb. 16 zeigt das Ergebnis des Performancetests einer hybriden (6+6)-ES. Nach t=200 Generationen wird ein durchschnittlicher Wert für S_g≈1900 erreicht.

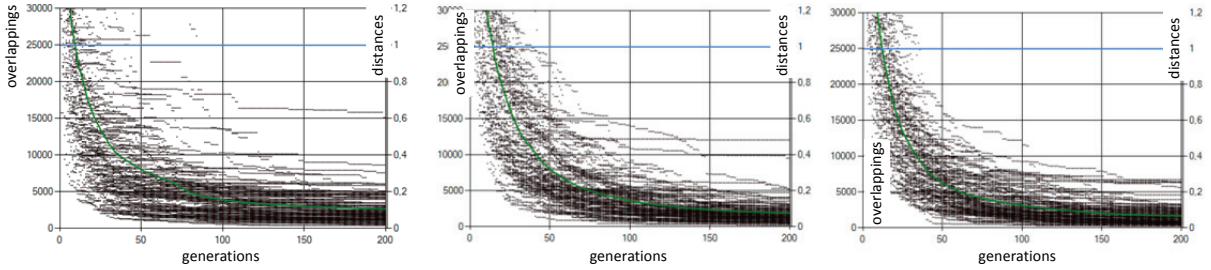


Abb. 16: Analysediagramme für die Layout-Generierung mittels hybrider ES nach je 100 Durchläufen des Programms mit jeweils t=200 Generationen. Die durchgezogenen grünen Linien stellen die Mittelwertkurven dar. Links: (1+6)-ES. Mitte: (6+6)-ES. Rechts: (6+6)-ES mit selbst-adaptivem Strategieparameter.

Bei der Erhöhung der λ Kinder und μ Eltern einer ES ist zu beachten, dass für jedes zusätzliche Individuum die Bewertungsfunktion ausgeführt werden muss, was zu einer entsprechend längeren Laufzeit pro Generation führt. Wir haben bisher die Geschwindigkeit des Systems in Generationen gemessen und angenommen, dass eine gute Qualität eines Layouts (S_g≈5000) nach t≈20 eine angemessene Nutzerinteraktion mit dem System ermöglicht. Diese Annahme gilt hier zwar weiterhin, wird

aber durch die längeren Laufzeiten bei mehr zu berechnenden Individuen pro Generation relativiert.

Eine weitere Variation der hybriden-ES besteht darin, eine oder mehrere Variablen der Anpassungsfunktionen (5) als selbst-adaptiven Strategieparameter (Bäck, 2000b; Weicker, 2007, p. 135) zu verwenden:

$$\begin{aligned} &\alpha' = \alpha * \exp\left(\mathrm{N}_0(\sigma)\right) \quad \alpha' \in [0{,}1;\ 1] \\ &\triangle G_i^t = \alpha' * r_i^t \qquad if \quad \rho_G > U \end{aligned} \tag{14}$$

wobei σ=0,1. Die Festlegung in (14), dass α' nicht kleiner als 0,1 sein darf, ist dadurch begründet, dass eine sehr kleine Gewichtung für die Proportionsänderung der Rechtecke zwar kurzfristig in der Nähe von lokalen Optima von Vorteil sein kann, bei der Suche nach globalen Optima in der Regel aber kontraproduktiv sind.

Das Ergebnis des Performancetests einer hybriden (6+6)-ES mit integriertem Strategieparameter (14) ist im rechten Diagramm in Abb. 16 dargestellt. Nach t=200 Generationen wird ein durchschnittlicher Wert für S_g≈1700 erreicht, was einer Verbesserung der Systemperformance im Vergleich mit der (6+6)-ES ohne Strategieparameter (Abb. 16, Mitte) um ca. 10% entspricht. Ferner ist zu erkennen, dass die Mittelwertkurve etwas steiler abfällt, was z.B. daran zu erkennen ist, nach wie vielen Generationen diese Kurve einen Wert für S_g≈5000 erreicht.

Untersucht wurde ferner, ob mittels weiterer Strategieparameter für die Anpassungsfunktionen (5), also für die Selbstadaption von ρ_G, β, oder ρ_L, zusätzliche Verbesserungen erzielt werden können. Herausgestellt hat sich, dass sich durch die Selbstadaption dieser Variablen keine positiven Effekte ergeben haben, sondern es im Gegenteil zu Variablenkombinationen kommen kann, welche das System in einem lokalen Optimum halten und dadurch ein Fortschreiten zum globalen Optimum blockieren.

Bei allen bisher beschriebenen Untersuchungen wurde die (+)-Selektion für die ES verwendet, da diese während der Optimierung kontinuierlich vorgeht, d.h. die Eigenschaft hat, dass es nicht zu sprunghaften Wechseln zwischen verschiedenen Layouts kommt wie bei der (,)-Selektion. Allerdings bringt die (+)-Selektion auch Nachteile mit sich (Bäck, 1994), deren gewichtigster im vorliegenden Kontext darin

besteht, dass sich die *(μ+λ)*-ES nur schlecht an sich verändernde Zielvorgaben anpassen kann und dazu tendiert, in einem einmal gefunden Optimum zu verharren. Insbesondere durch Nutzerinteraktion kann es bei dem hier untersuchten System häufig zu sich verändernden Zielvorgaben kommen. Wir werden allerdings noch sehen, dass sich diese Schwierigkeiten umgehen lassen und die Vorteile der (+)-Selektion deren Nachteile überwiegen.

Wir betrachten im Folgenden die Möglichkeiten für die weitere Verbesserung der Systemperformance, die sich durch die sich durch die Durchmischung verschiedener Individuen mittels Rekombination ergeben.

5.1.4. Hybride (μ+λ)-ES mit Rekombinationsoperator

Bei der Verwendung mehrerer Eltern pro Generation $\mu > 1$ können ρ Eltern an der Produktion eines Kindes beteiligt sein. Die Schreibweise einer ES wird dementsprechend um den Parameter ρ ergänzt: *(μ/ρ+λ)*-ES. Für unsere Zwecke betrachten wir die intermediäre und die diskrete Rekombination (Bäck, et al., 1991). Bei der intermediären Rekombination werden die Mittelwerte der Objektparameter berechnet:

$$X_i' = \frac{1}{\rho}\sum_{i=1}^{\rho} X_i \tag{15}$$

Bei der diskreten Rekombination wird jeder Objektparameter zufällig von einem der ρ Eltern gewählt (Deb, 2001, p. 137) (ρ=3):

$$\begin{array}{lllllll} Elter1: & X_1^{(1)} & X_2^{(1)} & X_3^{(1)} & X_4^{(1)} & X_5^{(1)} & X_6^{(1)} \\ Elter2: & X_1^{(2)} & X_2^{(2)} & X_3^{(2)} & X_4^{(2)} & X_5^{(2)} & X_6^{(2)} \\ Elter3: & X_1^{(3)} & X_2^{(3)} & X_3^{(3)} & X_4^{(3)} & X_5^{(3)} & X_6^{(3)} \\ & & & & & & \\ Rekombinant\ \boldsymbol{X}': & X_1^{(3)} & X_2^{(1)} & X_3^{(2)} & X_4^{(2)} & X_5^{(1)} & X_6^{(3)} \end{array} \tag{16}$$

Für die in diesem Abschnitt betrachtete Rekombinationsmethode verwenden wir ρ=2. Eingeführt wird ferner der Rekombinationsoperator *r:*

$$
\begin{aligned}
&r\left(P^t\right) = a' = \left(X', \sigma'\right) \in I \quad X' \in \mathbb{R}^n,\ \sigma' \in \mathbb{R}^n \\
&X'_i = \begin{cases} X_{b,i}\ ,\ U \le 0{,}5 \\ X_{c,i}\ ,\ U > 0{,}5 \end{cases} \quad \forall i \in \{1,\ldots,n\} \\
&\sigma'_i = \left(\sigma_{b,i} + \sigma_{c,i}\right) * 0{,}5 \quad \forall i \in \{1,\ldots,n\}
\end{aligned}
\tag{17}
$$

wobei $b=(X_b, \sigma_b)$, $c=(X_c, \sigma_c)$ ϵ I zwei Eltern darstellen, die vom Rekombinationsoperator r ausgewählt werden. An dieser Stelle ist es wichtig, sich zu vergegenwärtigen, dass, wie in Gleichung (7) angegeben, ein Gen X_i den kompletten Satz von Eigenschaften eines Rechtecks umfasst, der durch Rekombination nicht zerstört werden darf. Diese Tatsache ist relevant, da wir den einmal definierten Flächeninhalt eines Rechtecks durch keine Operation des EA verändern wollen. Würden wir die Eigenschaften der Rechtecke der Rekombination aussetzen, würden sich neue Rechtecke ergeben, die aufgrund neuer Werte für Höhen und Breiten zwangsläufig neue Flächeninhalte erhielten, was hier zu vermeiden ist, da jeder Raum seine ihm einmal zugewiesene Größe beibehalten muss. Könnten die Flächeninhalte variieren, würde eine Population schnell von kleinen Räumen dominiert werden, da diese geringere Überlappungsflächen erzeugen. Die Summe der Rechteckflächen muss daher immer gleich der Fläche des Umgebungsrechtecks sein.

Die Auswahl der Eltern b und c erfolgt mittels gewichteter Wahrscheinlichkeit, auch als Roulette-Wheel-Verfahren bezeichnet, bei dem sich die Auswahlwahrscheinlichkeit an der Fitness eines Individuums orientiert, sodass bessere Individuen häufiger für eine Rekombination herangezogen werden. Um zu verhindern, dass besonders gute Individuen eine Population zu schnell dominieren, wird die Auswahlwahrscheinlichkeit mittels linearer Skalierung (Goldberg, 1989, p. 79) so angepasst, dass das beste Individuum mit einer doppelt so großen Wahrscheinlichkeit ausgewählt wird wie ein Individuum mit mittlerer Fitness. Die Wahrscheinlichkeit für das schlechteste Individuum wird gleich null gesetzt, sodass dieses nicht für die Rekombination verwendet wird.

Für die Erzeugung von X'_i werden nach (17) die Vektoren X zweier Eltern ab einer bestimmten, mittels U zufällig gewählten Stelle i miteinander vertauscht. U bezeichnet eine gleichverteilte Zufallszahl auf dem Intervall [0, 1], die bei jedem Aufruf neu gezogen wird. Die Variable σ'_i ergibt sich durch intermediäre Rekombination. Der Rekombinationsoperator wird nicht für die Erzeugung jedes Kindes ange-

wandt, sondern mit einer bestimmten Wahrscheinlichkeit ρ_r, für die in der vorliegenden Untersuchung der Wert ρ_r=0,75 verwendet wurde. Das bedeutet, dass 75% der Kinder mittels Rekombinationsoperator generiert und 25% unverändert von der Eltern- in die Kindergeneration kopiert werden. Unabhängig von der Anwendung des Rekombinationsoperators wird der Mutationsoperator (13) und der selbstadaptive Strategieparameter (14) bei allen Kindern verwendet.

Bei der Auswertung der in diesem Abschnitt entwickelten hybriden (6/2+6)-ES im linken Diagramm in Abb. 17 sehen wir, dass nach t=200 Generationen ein durchschnittlicher Wert für S_g≈1400 erreicht wird, was einer deutlichen Verbesserung der Systemperformance um über 20% im Vergleich mit der (6+6)-ES ohne Rekombinationsoperator (Abb. 16, rechts) entspricht. Ferner ist zu erkennen, dass, verglichen mit der letztgenannten Variante, die Mittelwertkurve etwas steiler abfällt, was bedeutet, dass brauchbare Layoutlösungen schneller gefunden werden.

Wir haben bisher das Prinzip der (+)-Selektion befolgt, welches besagt, dass die Individuen der Elternpopulation nicht mutiert werden, wodurch es möglich ist, dass sehr gute Individuen immer wieder unverändert in die nächste Generation kopiert werden. Dadurch können diese eine Population sehr schnell dominieren, da die neuen Varianten immer wieder auf Basis der bereits sehr guten Individuen erzeugt werden. Dieser Umstand hat den positiven Effekt, dass es zu keinen sprunghaften Veränderungen einer einmal gefundenen Lösung kommt, sondern diese aus Sicht eines Nutzers relativ kontinuierlich immer weiter verbessert wird.

Für eine weitere Optimierung der Systemperformance werden nun zwei Operationen eingeführt, welche auf die Elternindividuen angewandt werden. Die Eltern werden in jeder Generation verändert, bevor die Kinderindividuen mittels Rekombinations- und Mutationsoperatoren erzeugt werden. Im mittleren Diagramm in Abb. 17 sind die Ergebnisse des Performancetests einer Variante der hybriden (6/2+6)-ES dargestellt, bei der für alle Elternindividuen die Anpassungsfunktionen (5) der Kollisionserkennung mit den Parametereinstellungen α=0,1; ρ_G=0,5; β=0,5; ρ_L=1 angewendet wurden. Die Elternindividuen werden folglich ausschließlich anhand der Mechanismen der Kollisionserkennung verändert, so wie sie in Abschnitt 5.1.1 beschrieben wurden. Wir erinnern uns daran, dass durch die Operationen der Kollisionserkennung unerwünschte Rückkoppelungseffekte auftreten können, wel-

che gelegentlich die Qualität einer einmal gefundenen Layout-Lösung verringern. Dieser Umstand hat zur Folge, dass es nun keine Individuen mehr gibt, die grundsätzlich die Verschlechterung einer gefundenen Layoutlösung verhindern. Tatsächlich sind in den Verläufen der einzelnen Graphen im mittleren Diagramm in Abb. 17 gelegentlich leichte Verschlechterungen (leichte Zunahme der Überlappungswerte S_g) zu erkennen. Trotzdem ist die Performance dieser Variante insgesamt deutlich besser als bei der vorangegangenen Variante der hybriden (6/2+6)-ES ohne Manipulation der Eltern. Nach t=200 Generationen wird ein durchschnittlicher Wert für S_g≈800 erreicht, was einer deutlichen Verbesserung der Systemperformance um über 40% im Vergleich mit der (6/2+6)-ES (Abb. 17, links) entspricht. Ferner ist zu erkennen, dass, verglichen mit der letztgenannten Variante, die Mittelwertkurve wiederum etwas steiler abfällt.

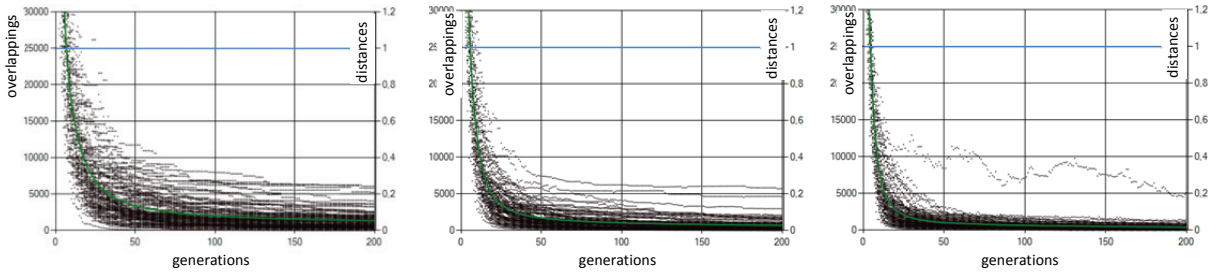


Abb. 17: Analysediagramme für die Layout-Generierung mittels hybrider (μ/ρ +λ)-ES nach je 100 Durchläufen des Programms mit jeweils t=200 Generationen. Die durchgezogenen grünen Linien stellen die Mittelwertkurven dar. Links: (6/2+6)-ES mit ρ_r=0.75. Mitte: (6/2+6)-ES mit ρ_r=0.75 und Kollisionserkennung bei den Eltern. Rechts: (10/2+7)-ES mit ρ_r=0.75 sowie Kollisionserkennung und Zufallsbewegung bei den Eltern.

Bei der nächsten in diesem Abschnitt betrachteten Variante der ES mit Rekombinationsoperator wird vor der bereits dargestellten Kollisionserkennung der Mutationsoperator (10), der eine zufällige Positionsänderung bewirkt, auf die Elternindividuen angewandt. Wir verwenden für den Faktor $\kappa=S_g(t)/1000$, wodurch die zufälligen Positionsänderungen durchschnittlich umso größer ausfallen, je größer die Summe der Überlappungsflächen ist. Die Wahrscheinlichkeit für diese Mutation hängt von der Größe der Elternpopulation ab: $\rho_Z = 1/\mu$. Der hier eingeführte Mutationsoperator für eine zufällige Positionsänderung soll vor allem verhindern, dass das System in einem lokalen Optimum verharrt.

Das Ergebnis des Performancetests der entsprechend erweiterten hybriden (10/2+7)-ES ist im rechten Diagramm in Abb. 17 dargestellt. Nach t=200 Generationen wird ein durchschnittlicher Wert für S_g≈500 erreicht, was einer Verbesserung der Systemperformance im Vergleich mit vorausgehenden Variante (Abb. 17, Mitte) um über 40% entspricht. Ferner ist zu erkennen, dass, verglichen mit der letztgenannten Variante, die Mittelwertkurve noch steiler abfällt.

Die gute Performance der zuletzt beschriebenen Variante (Abb. 17, rechts) beruht zum einen auf dem Mutationsoperator für zufällige Positionsänderungen und zum anderen auf den etwas größeren Eltern- und Kinderpopulationen. Durch die Manipulation der Elternindividuen haben wir das Prinzip der reinen (+)-Selektion einer ES verletzt und uns einer (,)-Selektion angenähert.

Bei der Standardimplementation einer rekombinativen ES werden die ρ Eltern nicht wie in unserem Beispiel mittels gewichteter Wahrscheinlichkeit, sondern rein zufällig ausgewählt (Deb, 2001, pp. 136-137). Da die Zufallsauswahl (16) einfacher als das Roulette-Wheel-Verfahren ist, stellen wir in Abb. 18 beide Verfahren gegenüber. Im linken Diagramm in Abb. 18 ist noch einmal das Ergebnis des Performancetests der oben eingeführten (10/2+7)-ES mit ρ_r=0.75 abgebildet. Nach t=100 Generationen wird ein durchschnittlicher Wert für S_g≈700 erreicht. Bei allen Layoutsystemen, deren Performancediagramme in Abb. 18 dargestellt sind, wurde auf die Eltern eine Kollisionserkennung und Zufallsbewegung angewandt. Die Diagramme in der Mitte und rechts in Abb. 18 zeigen Performancetests mit zufällig ausgewählten ρ Eltern. Im mittleren Diagramm wurden drei Eltern (ρ=3, S_g≈600) und im rechten Diagramm zwei Eltern (ρ=2, S_g≈500) für die Rekombination herangezogen. Darüber hinaus fallen die Mittelwertkurven bei den beiden letztgenannten Varianten etwas steiler ab als bei der erstgenannten Systemvariante mit gewichteter Wahrscheinlichkeit.

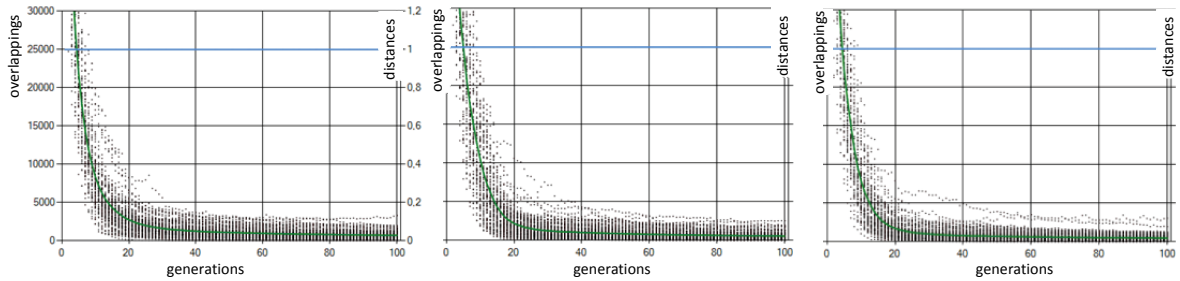


Abb. 18: Analysediagramme für die Layout-Generierung mittels hybrider ($\mu/\rho+\lambda$)-ES nach je 100 Durchläufen des Programms mit jeweils t=100 Generationen. Die durchgezogenen grünen Linien stellen die Mittelwertkurven dar. Links: (10/2+7)-ES mit ρ_r=0.75 (Roulette-Wheel-Verfahren). Mitte: (10/2+7)-ES. Rechts: (10/3+7)-ES mit ρ_r=0.75 sowie Kollisionserkennung und Zufallsbewegung bei den Eltern.

Da die zufällige Auswahl der ρ Eltern die einfachere Variante ist und zudem geringfügig bessere Ergebnisse liefert, bietet die zuletzt vorgestellte Variante einer hybriden (10/2+7)-ES für die Lösung des Problems der dichten Packung die beste Performance der hier untersuchten Systemvarianten. Im folgenden Abschnitt betrachten wir die Unterschiede zwischen einer reinen (10/2,7)-ES und der hier zuletzt vorgestellten (10/2+7)-ES hinsichtlich der Reaktion auf Störungen.

5.1.5. Reaktion auf Störungen

Bereits im ersten Kapitel wurde erwähnt, dass ein wichtiger Teil der vorliegenden Untersuchung darin besteht, bei der Bearbeitung von Layoutaufgaben neben Optimierungsmethoden in gleichgewichtiger Weise Gestaltungsaspekte zu berücksichtigen. Da letztere lediglich in Ansätzen operationalisierbar sind, fällt der Nutzerinteraktion eine wichtige Rolle zu. Diese ermöglicht es, nicht-operationalisierbare Aspekte, die bei der Bewertung eines Layouts der menschlichen Intuition obliegen, in das System einfließen zu lassen.

In Abschnitt 5.1.1 wurde festgestellt, dass für eine angemessene Nutzerinteraktion eine bestimmte Konvergenzgeschwindigkeit erreicht werden muss, mit der das System brauchbare Lösungsvorschläge anbietet. Für diese Geschwindigkeit haben wir definiert, dass nach $t\approx$20 Generationen der Wert für die Summe aller Überlappungen eines Layouts bei $S_g\approx$5000 liegen soll.

Vor diesem Hintergrund betrachten wir in diesem Abschnitt, wie und mit welcher Geschwindigkeit das soweit entwickelte Layoutsystem auf Störungen reagiert. Unter

einer Störung kann beispielsweise die plötzliche Verschiebung oder Skalierung eines Rechtecks durch einen Nutzer verstanden werden. Im Folgenden untersuchen wir die Reaktion des Systems auf zufällige Positionsänderungen beliebig ausgewählter Rechtecke.

Untersucht wird wie gehabt die Performance des Layoutsystems, indem das Programm 100-mal hintereinander ausgeführt wird und die Entwicklung der S_g-Werte in ein Diagramm geplottet werden (Abb. 19). Wir betrachten einen Zeitraum von 100 Generationen. Bei t=34 und t=68 Generationen findet eine Positionsänderung eines beliebigen Rechtecks statt. Das entsprechende Rechteck wird mit zufälligen x- und y-Werten verschoben, deren Maximum durch die Breite bzw. Höhe des Umgebungsrechtecks festgelegt ist.

In Abb. 19 sind die Ergebnisse der Performancetests dargestellt, die mit verschiedenen ES durchgeführt wurden. Das linke Diagramm in Abb. 19 zeigt die Ergebnisse der im vorangegangenen Abschnitt 5.1.4 zuletzt erläuterten (10/2+7)-ES. Nach den beiden Störungen wird nach t=100 Generationen ein durchschnittlicher Wert für S_g≈1500 erreicht. Das oben genannte Kriterium für eine angemessene Nutzerinteraktion ist hiermit erfüllt. Das System reagiert sehr schnell auf die Interaktion eines Nutzers, sodass sich unmittelbar nach einer Änderung wieder eine gute Layout-Lösung einstellt.

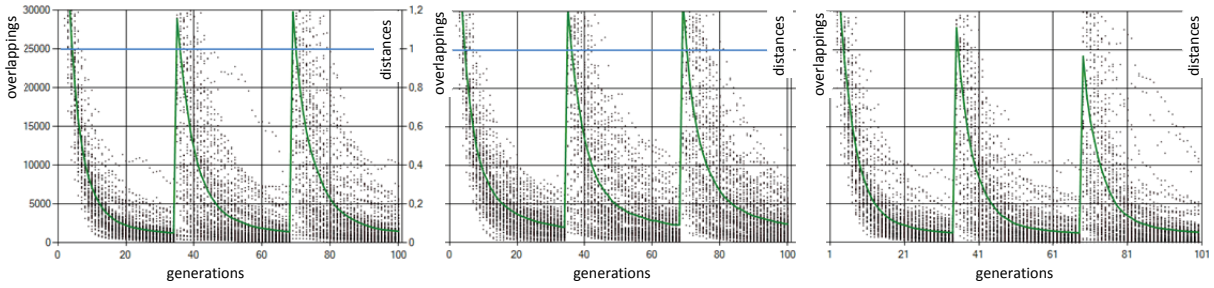


Abb. 19: Analysediagramme für die Reaktion auf Störungen. Die durchgezogenen grünen Linien stellen die Mittelwertkurven dar. Bei allen Varianten erfolgte eine Kollisionserkennung und Zufallsbewegung bei den Eltern. Pro Diagramm wurden 100 Durchläufe des Programms mit jeweils t=100 Generationen aufgezeichnet. *Links:* (10/2+7)-ES mit ρ_i=0.75. *Mitte:* (10/2,7)-ES. *Rechts:* Standard (10/2+7)-ES.

Im mittleren Diagramm in Abb. 19 ist das Ergebnis des Performancetests einer (10/2,7)-ES dargestellt. Bei dieser wird nach t=100 Generationen ein durchschnittlicher Wert für S_g≈2500 erreicht. Verwenden wir Elite-Selektion (das beste Individu-

um wird in die nächste Generation weitergegeben) für die (10/2,7)-ES, verbessert sich der durchschnittlicher Wert für Sg≈2000 nach t=100 Generationen. Das rechte Diagramm in Abb. 19 zeigt das Ergebnis des Performancetests für eine Standard (10/2+7)-ES mit zufälliger Auswahl der ρ Eltern. Bei dieser wird nach t=100 Generationen ein durchschnittlicher Wert für S_g≈1400 erreicht.

Wir können feststellen, dass die Standard (10/2+7)-ES die beste Performance aller bisher untersuchten Layoutsysteme aufweist. Die Befürchtung, dass das System nach einer Störung bei einer einmal gefundenen Layoutvariante hängen bleibt, war besonders bei der (+)-Selektion, wie in Abschnitt 5.1.3 beschrieben, naheliegend. Entgegen dieser Erwartung haben alle in Abb. 19 untersuchten Systemvarianten gezeigt, dass sie schnell auf Störungen reagieren und angepasste Layoutlösungen erzeugen können. Die (10/2+7)-ES hat neben der besten Performance den Vorteil, dass das System am wenigsten sprunghaft auf Nutzerinteraktionen reagiert und entsprechende Anpassungen vergleichsweise kontinuierlichsten erscheinen. Das Problem der dichten Packung kann folglich mit der (10/2+7)-ES hinreichend gut gelöst werden.

Bevor wir uns im nächsten Kapitel mit Methoden zur Berücksichtigung bestimmter Raumbeziehungen befassen, sei hier zusammenfassend der bisher in diesem Kapitel erarbeitete Algorithmus angegeben, der im Folgenden als Standard-ES bezeichnet wird (Tab. 3).

Algorithmus:	Standard-ES
Input:	Population P mit μ Individuen a mit je zufällig erzeugten Objektparametern
Output:	Individuum a_k mit den Objektparameter $\boldsymbol{X}_k$
Logik:	*Schritt 1:* Erstelle λ Kinder durch a) Mutation der Eltern [Zufallsbewegung (10) und Kollisionserkennung (5)] b) Rekombination von ρ Eltern [(15)-(17)] c) Mutation der Rekombinanten [(13) und Kollisionserkennung (5)] *Schritt 2:* Selektiere die μ besten Individuen aus Eltern und Kinderpopulation für die nächste Elterngeneration *Schritt 3:* Wenn das Abbruchkriterium erfüllt ist, beende den Algorithmus, andernfalls gehe zu Schritt 1.

Tab. 3: Schematischer Algorithmus der Standard-ES.

5.2. Raumbeziehungen

Unter Raumbeziehungen verstehen wir hier die Nachbarschaftsverhältnisse einzelner Räume. Zwei Räume sind miteinander benachbart, wenn sie keinen Abstand zueinander aufweisen und sich an je einer Kante mit einer bestimmten Länge berühren. Interpretieren wir das Layout als einen Grundriss mit einzelnen Räumen, soll diese Anforderung eine minimale Durchgangsbreite zwischen zwei Räumen gewährleisten.

Abstrahieren wir die Rechtecke als Repräsentation für einen Raum und verwenden stattdessen einen Punkt bzw. Knoten, so lassen sich die Raumbeziehungen als Verbindungslinien (Kanten) zwischen den Knoten darstellen. Auf diese Weise können wir einen Graphen konstruieren, der die topologischen Beziehungen des Layouts abbildet. Knoten und Kanten sind die topologischen Grundformen, die grafisch als Punkte und Linien dargestellt werden. Sind zwei Koten mit einer Kante verbunden, bedeutet dies, dass die beiden den Knoten entsprechende Räume miteinander benachbart sein sollen.

Wir betrachten lediglich Fälle, in denen Räume entweder benachbart miteinander sind oder nicht. Weitere Restriktionen, die beispielsweise angeben, wie nah ein Raum dem anderen sein soll oder dass zwei Räume möglichst weit voneinander entfernt liegen sollen, werden hier nicht berücksichtigt. Die Einbeziehung solcher Restriktionen würde nach Ansicht des Autors wenig Sinn machen, da auch zwei miteinander benachbarte Räume in der Wahrnehmung eines Nutzers relativ weit voneinander entfernt sein können, wenn keine Verbindung zwischen den Räumen besteht und ihre jeweiligen Eingänge in entsprechend großem Abstand zueinander platziert sind.

Für die folgenden Untersuchungen nehmen wir eine sternförmige Topologie als Restriktion für die Raumbeziehungen an (Abb. 20). Das bedeutet, dass alle Rechtecke zu einem zentralen Rechteck (z.B. der Diele) benachbart liegen sollen. Wir gehen davon aus, dass das Generieren eines Layouts mit sternförmiger Topologie relativ komplex ist und ein System, welches Lösungen für dieses Problem findet, auch Lösungen für Layoutprobleme mit anderen topologischen Restriktionen finden wird,

solange dies mit den geometrischen Variationsmöglichkeiten unsers Systems möglich ist. Formal lässt sich die topologische Restriktion folgendermaßen ausdrücken:

$$f_2^t(\boldsymbol{X}) = \sum_{i=1}^{n}\sum_{j=1}^{n}\left(d_{ij} + c_{ij}\right); \qquad f_2^t(\boldsymbol{X}) \to \min \tag{18}$$

Für die Berechnung der Funktion f werden zu den Abständen d zwischen zwei Rechtecken mit den Indizes i und j die Werte c für die Berührungslängen der Rechtecke addiert. Damit die gewünschten Raumbeziehungen entstehen, ist die Zielfunktion f zu minimieren.

Wir werden in diesem Kapitel drei Möglichkeiten untersuchen, um die geforderten Raumbeziehungen zu erreichen. Bei der ersten werden die Proportionen der Räume verändert (Abb. 20, rechts), bei der zweiten werden die Räume miteinander vertauscht (Permutation, Abb. 20, Mitte) und bei der dritten werden virtuelle Federn zwischen den Rechtecken gespannt, zwischen welchen eine topologische Beziehung besteht.

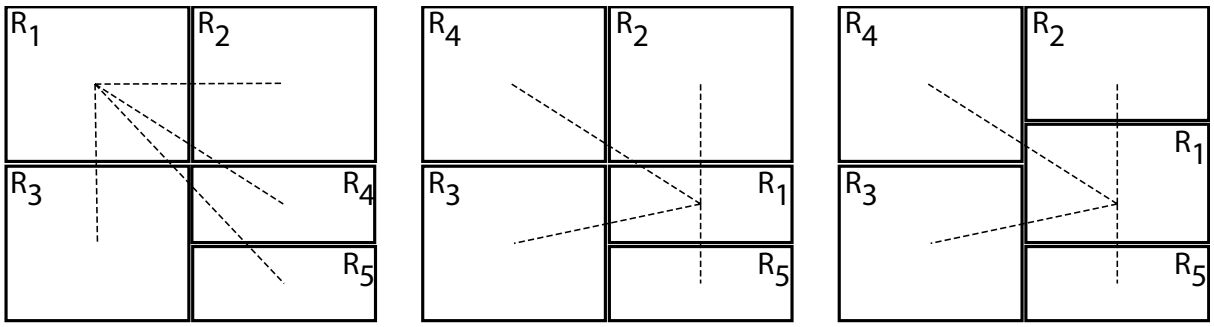


Abb. 20: Erfüllung der geforderten topologischen Beziehungen (gestrichelte Linien) durch Permutation (vgl. linke und mittlere Layout-Konfiguration) und Proportionsänderung (vgl. mittlere und rechte Layout-Konfiguration). Die Darstellung berücksichtigt nicht den Flächenerhalt der einzelnen Räume.

Die Funktion zur Proportionsänderung der Räume wurde bereits mit Gleichung (5) angegeben. Die Funktionen für Permutation und virtuelle Federn werden in den folgenden beiden Abschnitten erläutert.

5.2.1. Permutation

Nach einer Konfiguration der Räume, welche die gewünschten topologischen Beziehungen erfüllt, kann gesucht werden, indem man die Räume miteinander ver-

tauscht. Das Vertauschen von Elementen und die damit einhergehende Veränderung ihrer Anordnung bezeichnet man als Permutation.

Jeder Raum beziehungsweise jedes Gen X, welches alle Raumeigenschaften kapselt, besitzt einen Index i. Mittels diesem Index kann jedem Raum eine bestimmte Funktion zugewiesen werden und die topologischen Beziehungen können eindeutig festgelegt werden. Bei den Rekombinations- und Mutationsoperatoren ist nun allerdings darauf zu achten, dass ein Individuum, welches eine Layoutlösung repräsentiert, immer nur einen Raum mit einem bestimmten Index besitzt. Für die Rekombination wird diese Anforderung durch die von Goldberg und Lingle (1985) entwickelte Methode des *„Partially Mapped Crossover"* (PMX) gewährleistet. Eine genaue Beschreibung im Kontext der Grundrissgenerierung findet sich bei Elezkurtaj (2004, p. 77). Bei der Mutation eines Individuums werden entweder zwei Gene miteinander vertauscht oder eine zufällig gewählte Genreihenfolge wird invertiert.

5.2.2. Anziehungskräfte mittels virtueller Federn

Die Berechnung der Anziehungsvektoren zwischen zwei Rechtecken, die zueinander benachbart liegen sollen, erfolgt auf ähnliche Weise wie die der Abstoßungsvektoren in Abschnitt 5.1.1. Zuerst wird ermittelt, ob es einen Abstand $d_{i,j}$ zwischen den beiden Rechtecken gibt (Abb. 21). Wenn $d_{i,j} > 0$, wird geprüft, ob die x- oder y-Vektorkomponente größer ist (19). Die Anziehung der beiden Rechtecke erfolgt in die Richtung der größeren Vektorkomponente (19). Die Anziehungsvektoren ergeben sich folglich entweder für die x- oder y-Richtung, sodass sich die Rechtecke nur orthogonal aufeinander zu bewegen.

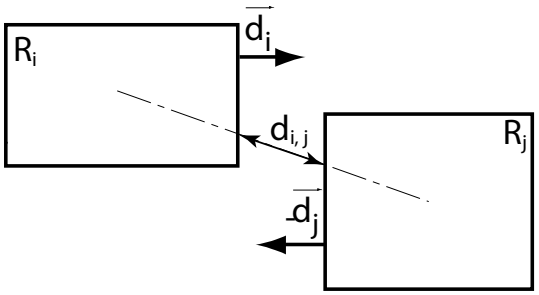


Abb. 21: Berechnung der Anziehungs-Vektoren zweier Rechtecke, die zueinander benachbart liegen sollen.

$$\vec{d}'^{t}_{i} = \gamma * \vec{d}^{t}_{i}$$
$$\vec{d}_i = -\vec{d}_j = \begin{cases} \begin{pmatrix} 0.5 * d_{xi} \\ 0 \end{pmatrix} & if\ d_{xi} > d_{yi} \\ \begin{pmatrix} 0 \\ 0.5 * d_{yi} \end{pmatrix} & else \end{cases} \quad (19)$$

Die größte Schwierigkeit besteht nun darin, alle soweit vorgestellten Methoden so miteinander zu kombinieren, dass mehrere Zielfunktionen gleichzeitig erfüllt werden können. Wie dieses Problem zu handhaben ist, wird im nächsten Abschnitt erläutert.

5.2.3. Multikriterielles Optimierungsproblem (MOOP)

Mit der Einführung der Raumbeziehungen als weitere Restriktion haben wir nun zwei Zielfunktionen f_1 (11) und f_2 (18), die beide durch das Layoutsystem optimiert (minimiert) werden sollen. Bei MOOP unterscheidet man zwischen widersprüchlichen und nicht widersprüchlichen Zielfunktionen. Bei sich widersprechenden Zielen kann nicht jedes Kriterium vollständig erfüllt, sondern im besten Fall ein möglichst guter Kompromiss gefunden werden. Für einen solchen Kompromiss gibt es in der Regel viele verschiedene Möglichkeiten, die als pareto-optimal bezeichnet werden. Ein pareto-optimaler Zustand (der auf der pareto-optimalen Front liegt) ist dadurch charakterisiert, dass es nicht möglich ist, eine Lösung weiter zu verbessern, ohne zugleich eine andere zu verschlechtern. Bei sich nicht widersprechenden Zielfunktionen kann jedes Kriterium vollständig erfüllt werden. Bei den beiden hier behandelten Zielfunktionen f_1 und f_2 handelt es sich in der Regel um sich nicht widersprechende Zielfunktionen. Das gilt zumindest, solange wir keine topologische Konfiguration für f_2 vorgeben, für die es keine überlappungsfreie Anordnung der Räume für f_1 gibt.

Die klassische Methode, MOOP zu lösen, besteht darin, sie auf einfache Optimierungsprobleme zu reduzieren, indem alle Kriterien zu einem kombinierten Kriterium zusammengefasst werden (Deb, 2001, pp. 13, 46). Für die Zusammenfassung auf ein Kriterium gibt es mehrere Methoden, von denen die der gewichteten Summe die gängigste und einfachste darstellt. Aufgrund der Schwierigkeiten, die die klassischen Methoden mit sich bringen, beispielsweise das Problem, die Gewichtungen

der einzelnen Kriterien festzulegen (Deb, 2001, pp. 49-80), wurden diese für das hier zu bearbeitende MOOP nicht verwendet.

In der Dissertation von Elezkurtaj (2004, pp. 65-68, 79) wurde das MOOP mittels Co-Evolution der EAs gelöst. Wie der co-evolutionäre Mechanismus im Detail funktioniert, konnte anhand der Beschreibung von Elezkurtaj vom Autor der vorliegenden Untersuchung nicht nachvollzogen werden. Wir betrachten im Folgenden eine einfache Implementation zur Lösung des hier beschriebenen MOOP, die als Kombination aus dem Vector Evaluated Genetic Algorithm (VEGA) nach Schaffer (1985) und der Vector-Optimized Evolution Strategy nach Kursawe (1990) verstanden werden kann und die wir als einfache MOES bezeichnen. Beide Basisverfahren sind detailliert beschrieben bei Deb (2001, pp. 179-189).

Als EA verwenden wir die in Abschnitt 5.1.5 als Standard definierte $(\mu/\rho+\lambda)$-ES; allerdings ohne selbst-adaptiven Strategieparameter, da dieser sich für das MOOP als kontraproduktiv erwiesen hat. Als Grundeinstellungen wird eine (10/2+5)-ES mit α=0,1; ρ_G=0,5; β=0,45; ρ_L=1 (vgl. Gleichung (5)) festgelegt. Die wesentliche Erweiterung der Standard-ES besteht in der Art, wie Individuen für die nächste Generation selektiert werden. Wie gehabt werden aus μ Eltern λ mutierte Kinder erzeugt. Anschließend werden per (+)-Selektion Eltern und Kinder gemeinsam selektiert. Dazu berechnen wir für jedes Individuum zwei Fitnesswerte für je eine der beiden Zielfunktionen f_1 und f_2. Anschließend wird die Kinderpopulation der ersten Zielfunktion entsprechend sortiert und ein bestimmter Anteil an Individuen für die nächste Elterngeneration ausgewählt. Genauso verfahren wir für die zweite Zielfunktion. Die Anzahl von Individuen, die für die beiden Zielfunktionen f_1 und f_2 ausgewählt werden, ist durch das Verhältnis $V=(f_1 / f_2)$ definiert. Bei der Initialisierung wird für $V=(\mu*0.1 / \mu*0.9)$ gewählt. Das bedeutet bei μ=10, dass nur ein Individuum für die geringste Überlappungsfläche und neun für die geringste Distanz selektiert werden. Bei geringeren Werten für μ wird allerdings immer ein Individuum für f_1 selektiert. Erhöht man von Beginn an die Anzahl an Individuen, die für f_1 selektiert werden, konvergiert das System leichter bei einer suboptimalen Lösung, die in etwa einem lokalen Optimum bei Optimierungsproblemen mit einem Kriterium entspricht. Man spricht hier von einer lokalen pareto-optimalen Front.

Damit bei Layouts, welche nach f_2 eine sehr geringe Distanzsumme aufweisen, möglichst rasch auch die Überlappungsflächen minimiert werden, führen wir einen Schwellenwert Ψ_V zur Veränderung des Verhältnisses V ein. Als effektiv hat sich die Regel zur Veränderung des Schwellenwertes erwiesen, bei der V verändert wird, sobald jenes Individuum mit dem geringsten Wert für f_1 (Überlappungsflächen) einen Wert für f_2 (Distanzen) unterschreitet, der kleiner ist als die doppelte Anzahl der Räume N_R:

$$V^t = \begin{cases} (\mu * 0.4 \ / \ \mu * 0.6) & if \quad f_2^t(\boldsymbol{X}_{(f_1 \min)}) < \Psi_V; \quad \Psi_V = 2 * N_R \\ (\mu * 0.1 \ / \ \mu * 0.9) & else \end{cases} \tag{20}$$

Eine ähnliche Regel wenden wir für die Selektionsmethode an. Solange ein Elternindividuum a einen Wert für f_2 aufweist, der größer als der Schwellenwert Ψ_V ist, wird eine (,)-Selektion angewandt, andernfalls eine (+)-Selektion mit Kollisionserkennung und Zufallsbewegung bei den Eltern, die wir als Standard-ES eingeführt haben:

$$Selektion \quad s = \begin{cases} (,) & if \quad f_2^t(\boldsymbol{X}_k) < \Psi_S; \quad \Psi_S = 4 * N_R \\ (+) & else \end{cases} \tag{21}$$

Der Einsatz von ES für MOOP, die auf dem Konzept nicht-dominierter Lösungen beruhen, wie beispielsweise PESA (Corne, Knowles, & Martin, 2000), haben im vorliegenden Problemkontext keine Verbesserung der Systemperformance gezeigt. Dies lag vor allem an der Eigenschaft, die besten bisher gefundenen Lösungen (non-dominated set) als Elternpopulation zu verwenden (elite-preserving). Diese Eigenschaft hat dazu geführt, dass das System lokale Pareto-Fronten oft nicht überwinden konnte.

Zusammenfassend kann der Algorithmus für die einfache (10/2+5)-MOES folgendermaßen angeben werden:

Algorithmus:	MOES
Input:	Population P mit μ Individuen a mit je zufällig erzeugten Objektparametern
Output:	Individuum a_k mit den Objektparameter $\boldsymbol{X}_k$

Logik:	*Schritt 1:* Erstelle λ Kinder durch a) nur bei (+) Selektion, vgl. (21) Mutation der Eltern [Zufallsbewegung (10), virtuelle Federn (19) und Kollisionserkennung (5)] b) Rekombination von ρ Eltern [(15)-(17)] c) Mutation der Rekombinanten [(13), Permutation (vgl. Punkt 5.2.1), Kollisionserkennung (5) und virtuelle Federn (19)]
	Schritt 2: Selektiere die μ besten Individuen nach den Regeln (20) und (21) für die nächste Elterngeneration
	Schritt 3: Wenn ein Abbruchkriterium erfüllt ist, beende den Algorithmus, andernfalls gehe zu Schritt 1. (In unserem Beispiel verwenden wir kein Abbruchkriterium. Dieses könnte aber z.B. so definiert werden, dass die Qualität der besten Lösung einen bestimmten Wert unterschreiten muss.)

Tab. 4: Schematischer Algorithmus für die einfache MOES.

5.2.4. Analyse der einfachen (10/2+5)-MOES

Auf die gleiche Art und Weise, wie wir in Kapitel 5.2 vorgegangen sind, betrachten wir nun die Performance der im vorangegangenen Abschnitt beschriebenen einfachen (10/2+5)-MOES. In Abb. 22 sind die Ergebnisse der Performancetests dargestellt, die jeweils mit verschieden vielen Räumen bei Vorgabe einer Sterntopologie (Abb. 20 und Abb. 23) durchgeführt wurden. Das linke Diagramm in Abb. 22 zeigt die Ergebnisse für 7 Räume. Nach t=20 Generationen wird ein durchschnittlicher Wert für S_g≈5000 erreicht, womit die oben festgelegte Grenze für eine akzeptable Konvergenzgeschwindigkeit erreicht wird. Nach t=100 Generationen erreicht das System einen durchschnittlichen Wert für S_g≈1100. Es ist offensichtlich, dass das schlechtere Ergebnis, verglichen mit der Standard-ES in Abschnitt 5.1.5, damit zusammenhängt, dass wir hier ein weiteres Kriterium (f_2) zu berücksichtigen haben.

Das mittlere Diagramm in Abb. 22 zeigt die Ergebnisse des Performancetests für acht Räume. Nach t=20 Generationen wird ein durchschnittlicher Wert für S_g≈10000 und nach t=100 Generationen ein durchschnittlicher Wert für S_g≈2700 erreicht. Dieses Ergebnis genügt nicht mehr den Anforderungen an eine akzeptable Konvergenzgeschwindigkeit und das System findet ab acht Räumen oftmals keine global optimale Lösung innerhalb der betrachteten 100 Generationen. Die Ergeb-

nisse des Performancetests für neun Räume sind im rechten Diagramm in Abb. 22 dargestellt, welche offensichtlich noch schlechter ausfallen als bei acht Räumen.

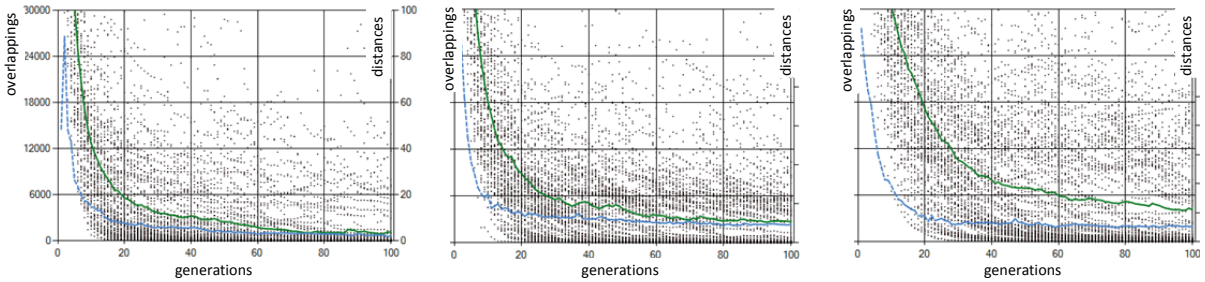


Abb. 22: Analysediagramme für die Layout-Generierung mittels MOEA nach je 100 Durchläufen des Programms mit jeweils t=100 Generationen. Die durchgezogenen grünen Linien stellen die Mittelwertkurven für die Überlappungsflächen f_1 und die gestrichelten blaue Linien die Mittelwertkurven für die Summen der Distanzen f_2 dar. Als Topologische Restriktion wurde eine Sterntopologie verwendet (vgl. Abb. 20). *Links:* 7 Räume. *Mitte:* 8 Räume. *Rechts:* 9 Räume.

Für die Beurteilung der Qualität des Lösungsverfahrens anhand des vorgestellten einfachen MOES ist es relevant, ob die gefundenen Lösungen ein großes Spektrum an verschiedenen Varianten abdecken oder ob immer die gleiche geometrische Lösung gefunden wird. Nach Deb (2001, p. 22) gibt es zwei ideale Eigenschaften für ein System zur multi-kriteriellen Optimierung: Erstens muss es einen Satz an Lösungen finden, die so nah wie möglich an der pareto-optimalen Front liegen bzw. die bei nicht widersprüchlichen Zielfunktionen für alle Funktionen ein globales Optimum darstellen. Zweitens muss es einen Satz an Lösungen finden, die so verschieden wie möglich sind und natürlich alle das erste Kriterium erfüllen – also an der pareto-optimalen Front liegen. Diese Systemeigenschaften werden in Kapitel 9 anhand eines differenzierten Testszenarios noch einmal eingehend untersucht.

Ob und in welcher Geschwindigkeit unser System für beide Zielfunktionen f_1 und f_2 ein globales Optimum findet, wurde mit den vorangegangenen Performanceanalysen in Abb. 22 untersucht. Die Frage nach der Diversität der Lösungen wird in Abb. 23 beantwortet. Im linken Feld finden sich elf grundlegend verschiedene geometrische Varianten für sieben Räume, deren Nachbarschaften eine Sterntopologie darstellen. Diese elf Varianten wurden aus den Ergebnissen der 100 Durchläufe des Systems nach t=100 Generationen ausgewählt (vgl. Abb. 22, links). Unter den 100 Ergebnissen befanden sich viele Variationen dieser elf Grundvarianten, die gespiegelte oder gedrehte Anordnungen der Räume umfassten. In der untersten Reihe in

allen Feldern in Abb. 23 sind typische unzureichende Layoutlösungen dargestellt, die bei sieben, acht oder neun Räumen mindestens für eine Zielfunktion in einem lokalen Optimum hängen geblieben sind und entweder die Forderung einer minimalen Durchgangsbreite von einem zum anderen Raum nicht erfüllen oder relativ hohe Werte für die Überlappungen der einzelnen Räume aufweisen.

Im mittleren Feld in Abb. 23 finden sich die gefundenen zwölf Grundvarianten, die aus den Ergebnissen der 100 Durchläufe des Systems nach t=100 Generationen ausgewählt wurden (vgl. Abb. 22, Mitte). Dementsprechend finden sich im rechten Feld in Abb. 23 die gefundenen elf Grundvarianten, die aus den Ergebnissen der 100 Durchläufe des Systems nach t=100 Generationen ausgewählt wurden (vgl. Abb. 22, rechts). Die mit der Anzahl an Räumen zunehmend schlechtere Qualität der Layoutlösungen, die bereits oben erwähnt wurde, ist auch im Vergleich der drei Felder in Abb. 23 zu erkennen.

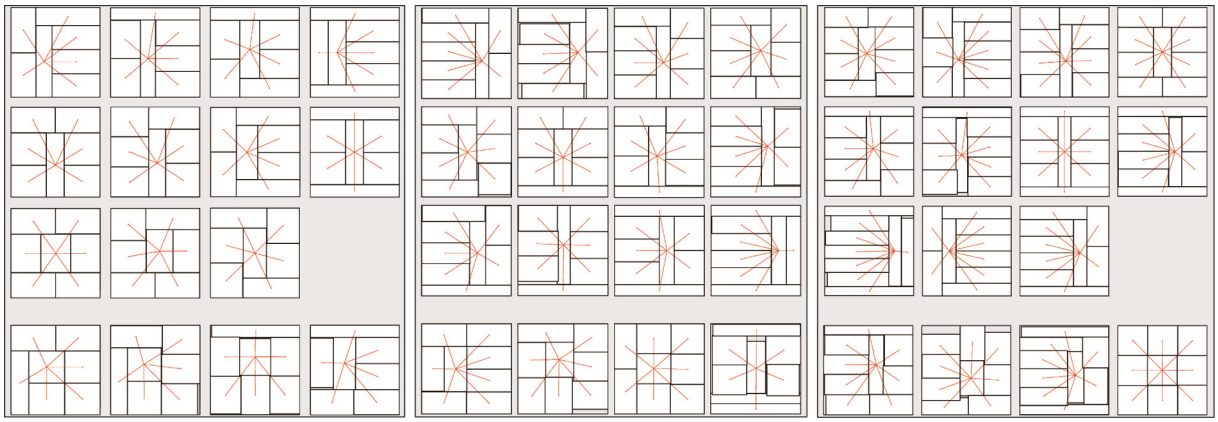


Abb. 23: Verschiedene Layouts mit Sterntopologie nach je 100 Durchläufen des Programms mit jeweils t=100 Generationen. Links: 7 Räume. Mitte: 8 Räume. Rechts: 9 Räume. In den oberen drei Reihen sind typische Layoutlösungen abgebildet. In der unteren Reihe finden sich typische unzureichende Layoutlösungen.

Bei der visuellen Auswertung der Layoutlösungen in Abb. 23 können wir feststellen, dass das System eine gewisse Diversität bei der Anordnung und Proportionierung der Rechtecke abdeckt. Allerdings können wir uns noch weitere Lösungen für das gestellte Problem vorstellen, bei denen beispielsweise der zentrale Raum die ganze verfügbare Höhe oder Breite des Umgebungsrechtecks ausnutzt. Dass sich eine solche Lösung bei keiner der in Abb. 23 dargestellten Varianten findet, lässt darauf schließen, dass es bei dem vorgestellten System einen Mechanismus gibt, der solche Lösungen nicht zulässt bzw. solche Lösungen relativ instabil sind, sodass sie schnell

wieder verworfen werden. Vermutlich kommt dieses Systemverhalten durch die Mechanismen der Kollisionserkennung (5) zustande. Folglich bietet das soweit vorgestellte System nur bedingt die oben genannte ideale Eigenschaft, die potentielle Diversität möglicher Lösungen vollständig abzudecken. Diese Einschränkung kann behoben werden, indem mehrere Populationen parallel zur Lösungssuche verwendet werden (vgl. Kapitel 9), wobei eine als (+)-Selektion als konservative Population verwendet wird, bei der die beste Variante erhalten bleibt, und eine (,)-Selektion als innovative Population, bei der lokale Optima leichter überwunden werden können, da auch beste Lösungen verloren gehen können.

5.3. Konklusion und Ausblick

Das Ziel der in diesem Kapitel durchgeführten Untersuchung bestand darin, ein System zur Lösung von Layoutproblemen in Architektur und Städtebau zu entwickeln, das anfangs mit möglichst wenig Problemwissen auskommt (vgl. Abschnitt 2.1) und schnell brauchbare Ergebnisse liefert, die durch schrittweises Hinzufügen von Problemwissen interaktiv weiter ausgearbeitet werden können.

Die vorgestellten Ergebnisse belegen die Fruchtbarkeit des Ansatzes, bei der Suche nach Layoutlösungen mit wenig Problemwissen zu beginnen. Diese Vorgehensweise steht jenen entgegen, die zuerst möglichst viel Wissen über ein Problem erfassen und darauf aufbauend mittels eines relativ unflexiblen Verfahrens Lösungen erzeugen. Unflexibel bedeutet in diesem Zusammenhang, dass die Lösungsverfahren ohne das anfängliche Problemwissen nicht arbeiten können. Beispiele für solche wissensbasierten Systeme finden sich bei den Arbeiten von Coyne (1988) oder Duarte (2001). An dieser Stelle muss darauf hingewiesen werden, dass die Unterschiede zwischen den beiden angeführten und prinzipiell verschiedenen Ansätzen nichts über die Qualität der entsprechenden Ergebnisse aussagen, wohl aber etwas darüber, als wie kreativ man ein entsprechendes System bezeichnen kann. Was wir im vorliegenden Kontext unter einem kreativ arbeitenden Computersystem verstehen, wurde in Kapitel 1.1 beschrieben. Der Autor des vorliegenden Texts ist der Ansicht, dass im Vergleich mit ähnlichen wissensbasierten Systemen mit dem hier vorgestellten evolutionären Layoutsystem ein wesentlich kreativeres entstanden ist.

Bei dem hier besprochenen System zur Lösung von Layoutproblemen wurde ein iterativer Ansatz, basierend auf EA, gewählt, welcher eine überlappungsfreie Anordnung von Elementen sowie deren topologische Beziehungen zueinander sicherstellt. Das entwickelte System ist auf rechteckige Elemente beschränkt. Die Verwendung freier Formen würde den Such- und Lösungsraum erheblich vergrößern und damit die Konvergenzgeschwindigkeit deutlich reduzieren. Die angestrebte Echtzeitinteraktion wäre damit nicht möglich. Dennoch ist es ein Ziel zukünftiger Arbeiten, das Layoutsystem für freie Formen zu erweitern (Schneider, Koenig, & Pohle, 2011).

Eine zentrale Schwierigkeit ergab sich bei der Kalibrierung des Layoutsystems, da sich die verschiedenen Parameter gegenseitig beeinflussen und die günstigsten Werte für einen Parameter hinsichtlich der Konvergenzgeschwindigkeit des Systems von der erreichten Qualität einer Lösung abhängen können. Das bedeutet, dass es erforderlich ist, erstens nach einer optimalen Kombination der Parameterwerte zu suchen und zweitens die Parameterwerte im Laufe eines Optimierungsprozesses anzupassen. In Abschnitt 5.1.3 wurde dargestellt, dass mittels eines selbstadaptiven Strategieparameters (Bäck, 2000b; Weicker, 2007, p. 135) die beschriebene Schwierigkeit nicht hinreichend behoben werden konnte. Es wurden alternative Strategien für die Adaption der Parameterwerte erprobt, bei denen erstens Schwellenwerte an die Fitnesswerte (vgl. die Regeln (20) und (21)), zweitens die Mutationsstärke an die Fitnesswerte (vgl. S. 22, $\kappa=S_g(t)/1000$) und drittens die Mutationswahrscheinlichkeit an die Größe der Elternpopulation (vgl. S. 22, $\rho_Z=1/\mu$) gekoppelt wurden.

Es konnte gezeigt werden, dass das vorgestellte Layoutsystem bis zu einem bestimmten Komplexitätsgrad, der sich anhand der Anzahl an Räumen und Relationen bestimmen lässt, gute Ergebnisse liefert, die eine brauchbare Unterstützung für einen Planer während des Planungsprozesses gewährleisten. Wichtig ist in diesem Zusammenhang, dass die Lösungssuche bei komplexen Problemen zwar relativ lange dauern kann, das System aber in der Regel eine Lösung findet.

Das hier vorgestellte Layoutsystem bietet aus Sicht des Autors die Grundfunktionalität für die computerbasierte Generierung von Layoutvarianten. Das Potential des MOOP-Ansatzes liegt darin, dass theoretisch beliebig viele weitere Kriterien in das

System integriert werden können. Für alle Kriterienkombinationen kann nach pareto-optimalen Lösungen gesucht werden. Beispielsweise könnten die Ausrichtung und Belichtung verschiedener Räume (Lobos & Donath, 2010), Sichtbarkeitsanalysen (Michael Batty, 2001) oder kürzeste Wegeverbindungen (Derix, 2009) berücksichtigt werden. Die genannten Kriterien können im Rahmen des iterativen Verfahrens beliebig aktiviert oder deaktiviert werden.

Abschließend ist zu erwähnen, dass die Performance des hier beschriebenen Systems durch Parallelisierung und Auslagerung der Rechenkapazitäten weiter gesteigert werden kann. Eine Parallelisierung von Populationen mit (+)- und (,)-Selektion erfolgt bei der in Kapitel 9 beschriebenen Vergleichsuntersuchung.

Ferner ließen sich vermutlich Performanceverbesserungen erreichen, indem für die einzelnen Räume keine festen Flächeninhalte festgelegt werden, sondern die Flächen innerhalb eines bestimmten Bereichs variiert werden können, wobei die Summe der Einzelflächen konstant bleiben muss.

6. Layouts mittels K-dimensionaler Bäume[9]

Katja Knecht, Reinhard König

K-dimensionale Bäume, im Englischen verkürzt auch K-d Trees genannt, sind binäre Such- und Partitionierungsbäume, die ursprünglich aus dem Bereich der Computergeometrie stammen und deren Algorithmen sich durch Effizienz und Schnelligkeit auszeichnen. Im Rahmen des in diesem Buch vorgestellten Forschungsprojekts beschäftigten wir uns mit der Frage, ob und wie das generative und gestalterische Potential von K-d Tree Algorithmen eingesetzt werden kann, um architektonische Layouts zu generieren. Die Raumpartitionierung durch K-d Trees als mögliche dividierende Methode zur Erstellung von Grundrissen wird in diesem Kapitel beschrieben. Darüber hinaus wird untersucht, wie die auf Basis von K-d Tree Algorithmen entstandenen Layouts in Kombination mit EA in Bezug auf topologische Beziehungen und Raumgrößen optimiert werden können und wie sie sich in das in Kapitel 3 beschriebene ALES einpassen.

6.1. K-d Trees

K-d Trees nach Bentley (J. L. Bentley, 1975) sind binäre Suchbäume, die eine Menge von n Punkten in einem multidimensionalen Raum organisieren (J. L. Bentley, 1990) und damit Datenstrukturen darstellen, die der Raumpartitionierung dienen (Goodrich & Tamassia, 2002).

Eingesetzt werden K-d Tree Datenstrukturen ursprünglich bei der Suche nach den nächsten Nachbarn, der *Nearest Neighbor Query* (Moore, 1991), und in weiteren Suchalgorithmen vor allem für klassische Datenbankapplikationen (J. L. Bentley, 1990). K-d Tree Datenstrukturen werden in Verbindung mit den genannten Suchalgorithmen aufgrund ihrer hohen Durchsuchungsgeschwindigkeit, ihrer Genauigkeit und Effizienz in vielfältigen Anwendungsgebieten eingesetzt. Zu diesen zählt beispielsweise die Gesichtserkennung im Rahmen des interaktiven Trackings von Ge-

[9] Dieses Kapitel basiert teilweise auf dem Artikel von Knecht, K., & Koenig, R. (2010). *Generating Floor Plan Layouts with K-d Trees and Evolutionary Algorithms.* Paper presented at the GA2010 - 13th Generative Art Conference, Milan, Italy.

sichtszügen in Videoframes, für die Geschwindigkeit und Genauigkeit der Suchergebnisse von Relevanz ist (Buchanan & Fitzgibbon, 2006). Darüber hinaus hat sich der Einsatz von K-d Tree Datenstrukturen in Raytracing Verfahren bewährt (Fussell & Subramanian, 1988; Wald & Havran, 2006). In bisherigen Anwendungen werden K-d Trees folglich vor allem verwendet, um räumliche Daten zu organisieren und zu speichern sowie die erstellten Datenstrukturen schnell und effizient zu durchsuchen und zu durchqueren.

Eine K-d Tree Struktur wird folgendermaßen aufgebaut (siehe hierzu Tab. 5 sowie Abb. 24 und Abb. 25): Ausgehend von einer bekannten, finiten Punktemenge P wird der k-dimensionale Raum durch zu den Koordinatenachsen senkrechten Schnittebenen, sogenannten *partition planes*, unterteilt. Im zweidimensionalen Raum entspricht die Unterteilung einer Linie, in drei- und mehrdimensionalen Räumen handelt es sich um eine an einer Koordinatenachse ausgerichtete Hyperebene (Goodrich & Tamassia, 2002). Zur Bestimmung der Lage der Unterteilung wird aus P zunächst der Median- oder Mittelwert der Punktkoordinaten in der Schnittdimension, der *split dimension*, gebildet. Es entsteht der erste Knoten, englisch *node*, $N1$ mit dem berechneten Mittel- oder Medianwert als Schnittlinienwert, dem *split value,* der die Punktemenge P in zwei Untermengen $P1$ und $P2$, die sogenannten *sub trees*, unterteilt. Alle Punkte mit Koordinaten, die in der Schnittdimension kleiner als der Schnittlinienwert sind, bilden den linken Ast und sind Bestandteil von sub tree $P1$, alle Punkte mit Koordinaten, die größer sind, bilden den rechten Ast und sind Bestandteil von $P2$. $N1$ gilt dabei als *root node* für die Unterknoten $N2$ und $N3$ der sub trees $P1$ und $P2$, $N2$ und $N3$ sind *child nodes* von $N1$. Diese Teilräume werden nun nach dem gleichen Prinzip weiter unterteilt, wie es für die Ausgangspunktmenge beschrieben wurde. Die Unterteilung wird so lange fortgeführt, bis keine weitere Unterteilung mehr möglich oder ein bestimmter Grenzwert bzw. die vorgegebene Unterteilungstiefe erreicht ist (Abb. 24 und Abb. 25).

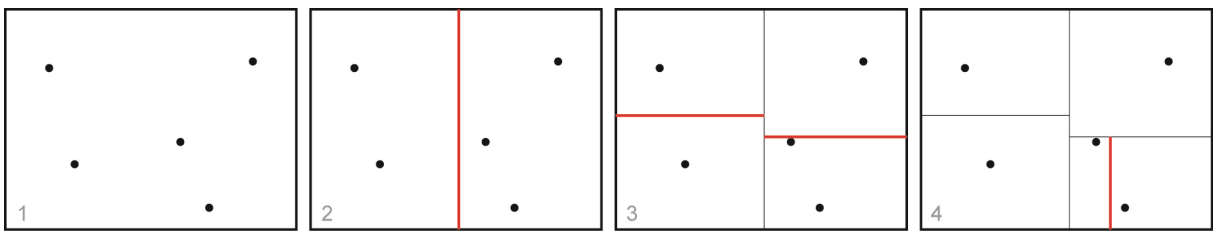


Abb. 24: Raumunterteilung durch eine K-d Tree Struktur mithilfe der Mittelwertberechnung.

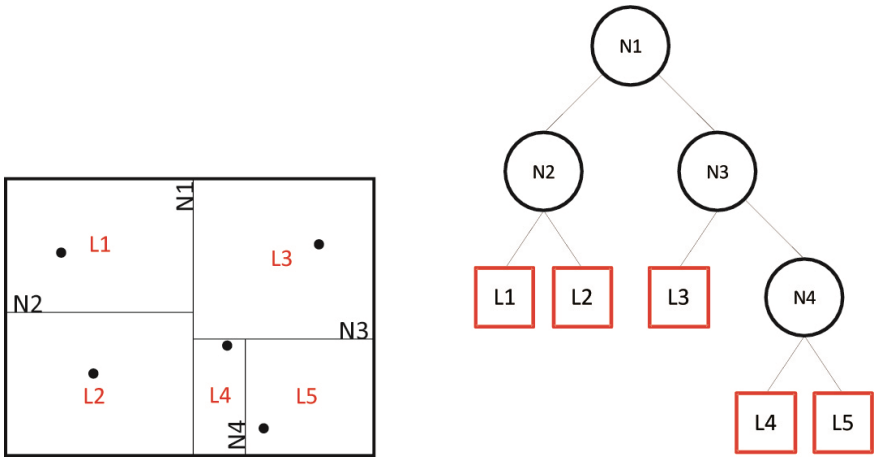


Abb. 25: Geometrische und grafische Darstellung der Unterteilung.

Algorithmus:	Aufbau eines K-d Trees
Input:	Punktemenge P
Output:	*kd* vom Typ KDTree
Logik:	*Schritt 1:* Generation t = 0 *Schritt 1*: If P ist leer return leeres KDTree *Schritt 2:* Bestimme die Schnittebene und generiere den Knoten N mit den folgenden Werten: *SplitDim* = die Schnittdimension *SplitVal* = der nach der festgelegten Schnittregel berechnete Schnittlinienwert in *SplitDim* *Schritt 3:* Fülle die rechten und linken Untermengen Pleft und Pright: *Pleft* = alle Punkte ϵ *P*, mit Vektor v[*SplitDim*] ≤ *SplitVal* *Pright* = alle Punkte ϵ *P*, mit Vektor v[*SplitDim*] > *SplitVal* *Schritt 4:* *kdleft* = linker Ast; Konstruiere das K-d Tree rekursiv mit *Pleft* *Schritt 5:* *kdright* = rechter Ast; Konstruiere des K-d Tree rekursiv mit *Pright* *Schritt 6:* return *kd*

Tab. 5: Schematischer Algorithmus des Aufbaus eines K-d Trees nach (Moore, 1991).

Jedem Knoten wird eine Region zugeordnet, welche wiederum Unterknoten und Unterregionen enthalten kann. Im Inneren des Suchbaums bezeichnet man die Knoten als interne Knoten, *internal nodes*. Sie unterteilen den Raum durch eine Schnittebene in einer der k Dimensionen (J. L. Bentley, 1990). Darüber hinaus wer-

den in den inneren Knoten die Information zur Schnittebene sowie die Verweise zu den rechten und linken Unterknoten abgelegt. Die Endpunkte des Baums, an denen keine weitere Unterteilung mehr möglich ist bzw. an denen die vorgegebene Unterteilungstiefe erreicht wurde, bezeichnet man als Blätter, englisch *leaves*, oder auch als externe Knoten, *external nodes* oder als *buckets* (J. L. Bentley, 1990).

Die Datenstruktur erfüllt damit drei Funktionen auf einmal: Sie speichert den Datensatz, unterteilt den Raum in Hyperrechtecke und liefert gleichzeitig ein Verzeichnis dieser Hyperrechtecke (J. L. Bentley & Friedman, 1979).

Die Struktur des Baums und damit auch die geometrische Repräsentation sind abhängig von der Unterteilungsregel, d.h. wie, in welcher Dimension und in welcher Abfolge die Schnittebenen gebildet werden (Maneewongvatana & Mount, 1999). Man unterscheidet hier zwei Grundtypen, punktbasierte und raumbasierte K-d Trees (Goodrich & Tamassia, 2002), die auf verschiedenen Schnittregeln basieren.

Bei Punkt-basierten K-d Trees werden Unterteilungen abhängig von der Verteilung der Punkte im k-dimensionalen Raum vollzogen. Eine einfache Schnittregel legt die Reihenfolge der Schnittdimensionen als kontinuierliche Abfolge von Schnitten in x-, y- bis n-Richtung fest (J. L. Bentley, 1975; De Berg, et al., 1997). Der Zyklus beginnt nach dem Schnitt in der n-ten Dimension anschließend wieder von vorne. Eine weitere mögliche Schnittregel, nach der unterteilt werden kann, ist die Standard Splitting Rule nach (Friedman, Bentley, & Finkel, 1977). Hier wird die Schnittdimension in der Dimension gewählt, in der eine Punktemenge die größte Verteilung aufweist (Goodrich & Tamassia, 2002). Der Schnittlinienwert wird in diesem Fall aus dem Median der Punktkoordinaten berechnet.

Ein raumbasierter K-d Tree entsteht beispielsweise durch Anwendung der Midpoint Splitting Rule (Maneewongvatana & Mount, 1999). Hier wird die Schnittebene durch den Mittelpunkt der zu unterteilenden Fläche gelegt und unterteilt diese nach der längsten Seite. Eine Weiterentwicklung mit Punkt- sowie raumbasierten Eigenschaften findet sich in der von Maneewongvatana und Mount (1999) vorgestellten Sliding Midpoint Rule, bei der zwar die Region zunächst mittig unterteilt wird, die Schnittebene aber anschließend zum nächstgelegenen Punkt hin verschoben wird (Abb. 26).

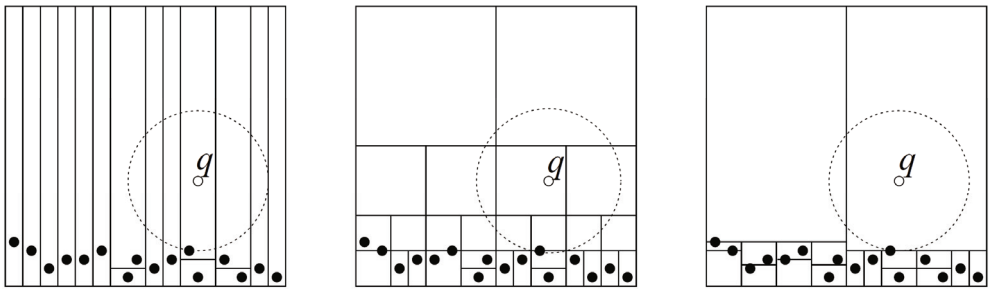


Abb. 26: Unterteilungsergebnisse mit unterschiedlichen Schnittregeln: Standard Splitting Rule, Midpoint Splitting Rule und Sliding Midpoint Rule. Abbildung aus (Maneewongvatana & Mount, 1999)

Die Schnittlinienwerte können darüber hinaus aus Median- oder Mittelwertbildung der Punktkoordinaten berechnet werden, wobei sich der strukturelle Aufbau der entstehenden K-d Trees unterscheidet. Bei der Medianberechnung werden zunächst alle Punkte der Punktmenge nach ihren Koordinaten in der Schnittdimension sortiert. Es wird dann derjenige Punkt bestimmt, dessen Koordinate an der mittleren Stelle der Folge steht. Die Schnittebene wird anschließend durch diesen Medianpunkt geführt (Abb. 27, links). In der Folge können jedem internen Knoten des K-D Trees je ein Punkt der Punktemenge zugeordnet werden. Im Gegensatz dazu liegen alle Punkte in einem durch Mittelwertberechnung gebildeten Baum in den Blattregionen (Abb. 27, rechts). Die Schnittlinienwerte werden hier aus dem arithmetischen Mittelwert aller Punktkoordinaten in der Schnittdimension berechnet.

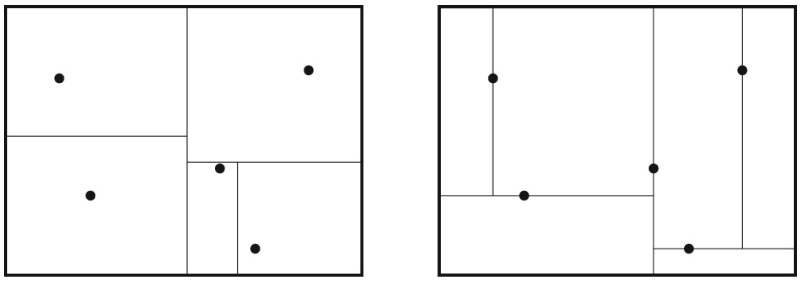


Abb. 27: Unterteilung durch Median- (links) bzw. Mittelwertberechnung (rechts).

6.2. Generierung von Grundrisslayouts mit K-d Trees

6.2.1. Generativer Mechanismus

Die Idee, K-d Tree Algorithmen zur Generierung von Grundrisslayouts im Rahmen des Kremlas Forschungsprojekts einzusetzen, entstand aus einer Betrachtung ihrer geometrischen Eigenschaften sowie den Möglichkeiten, die ihre Unterteilungsstruktur und Datenstruktur bieten. In der Folge wird die Entwicklung des generativen

Mechanismus beschrieben, d.h. der Aufbau der allgemeinen Datenstruktur sowie die eingesetzten EA zur Suche nach bestimmten Raumgrößen und Nachbarschaftsverhältnissen. Der Einsatz des K-d Trees zur Generierung von Grundrissen wurde zunächst nur auf Gebäudeebene untersucht. Die dargestellten Softwaremuster wurden in *C#* implementiert.

6.2.2. Allgemeine Datenstruktur

Die ersten Arbeitsschritte bestanden zunächst in der Umsetzung eines einfachen K-d Tree Algorithmus auf Basis der in Kapitel 6.1 dargestellten Logik, nach der eine vorgegebene Fläche durch Mittelwert- bzw. Medianberechnung in Raumachsenabfolge unterteilt wird (Abb. 28). Die Schnittlinien entsprechen dabei möglichen Raumgrenzen.

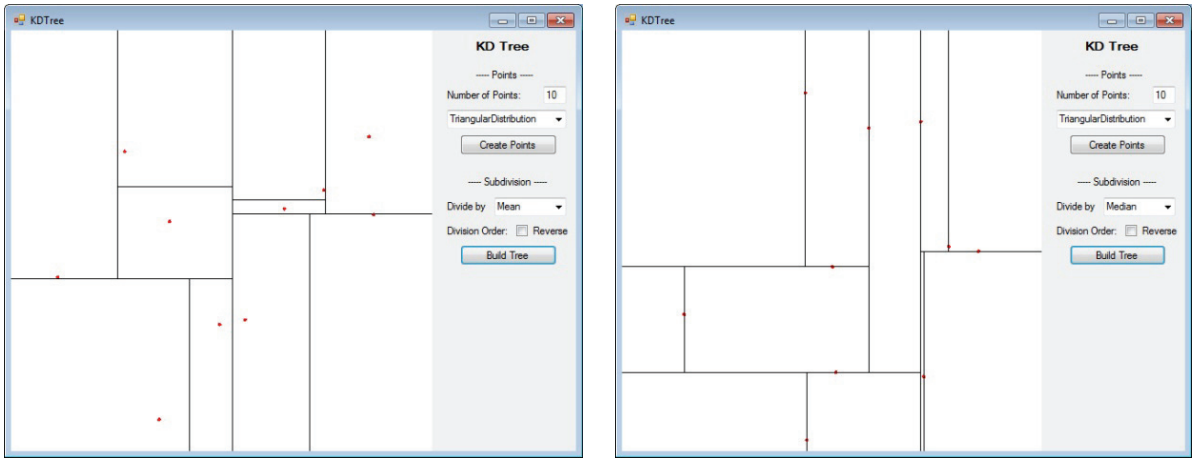


Abb. 28: Erster Prototyp; Raumunterteilung durch einen einfachen K-d Tree Algorithmus mit Mittelwert- (rechts) und Medianwertberechnung (links).

Beide Berechnungsvarianten besitzen Vor- und Nachteile hinsichtlich Interaktion und Datenmanagement. Beim Aufbau der K-d Tree Datenstruktur durch Mittelwertberechnung repräsentiert jeder Punkt der Punktemenge einen Raum. Diese Repräsentationsform hat den Vorteil, dass die Größe der Punktmenge gleichzeitig die Anzahl der generierten Räume im Grundriss bestimmt und jedem Punkt folglich ein Raum zugeordnet werden kann. Erfolgt der Aufbau der K-d Tree Datenstruktur hingegen durch Medianberechnung, so existiert für jeden Punkt der Punktemenge eine Schnittlinie. Die Anzahl der entstehenden Räume ergibt sich folglich aus der Anzahl Punkte + 1.

Während durch Mittelwertberechnung entstandene K-d Trees eine logische Verknüpfung von Punkten und Räumen erlauben und damit die Zuordnung von Raumeigenschaften erleichtern, zeichnen sich durch Medianwertberechnung entstandene K-d Trees durch eine nachvollziehbarere geometrische Unterteilung und Zuordnung aus.

Die K-d Tree Datenstruktur bildete die Grundlage für alle folgenden Arbeitsschritte. Darauf aufbauend wurde zunächst ein einfacher EA nach dem in Kapitel 4.4 in Tab. 2 beschriebenen Schema umgesetzt, der die mittels K-d Tree generierten Layouts im Hinblick auf die Raumgrößen optimierte (Abb. 29). Die hierzu eingesetzte ES wurde in der Folge gekoppelt mit einem GA, der die Nachbarschaftsbeziehungen der Räume optimiert. Abschließend wurde der generative Mechanismus als multikriterielles System implementiert, das die beiden Kriterienparameter so anpasst, dass sie zu optimalen Layoutlösungen führen.

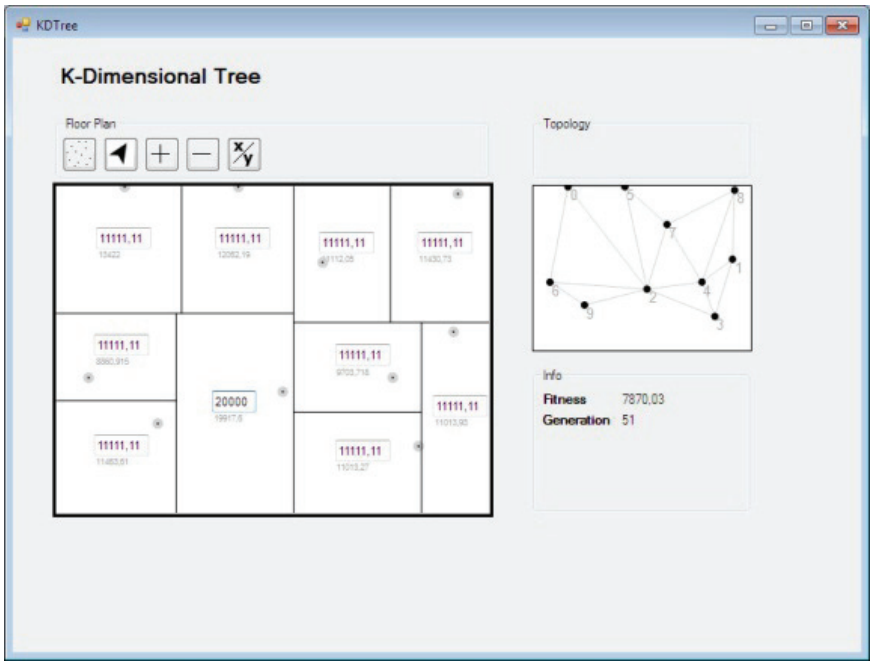


Abb. 29: Arbeitsversion zur Optimierung der Raumgrößen.

6.2.3. Suche nach bestimmten Raumgrössen

Die Optimierung der Raumgrößen ist ein geometrisches Problem, das als Formfindungsproblem bezeichnet werden kann. Es wird im Rahmen des K-d Tree Algorithmus durch die Größe der durch Unterteilung generierten Räume bestimmt. Die Unterteilung hängt ihrerseits von der Lage und Verteilung der Punkte im Raum ab. Um

die Raumgrößen zu optimieren, müssen die Positionen der einzelnen Punkte so verändert werden (Abb. 30), dass ihre Verteilung nach der Unterteilung die gewünschten Raumgrößen entstehen lässt. Zur Optimierung der Raumgrößen erwies sich der Einsatz einer ES als hilfreich.

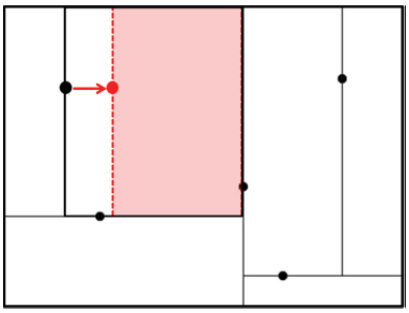


Abb. 30: Raumgrößenänderung durch Punktverschiebung.

ES zeichnen sich dadurch aus, dass die Individuen vor der Anwendung von Selektions-, Mutations- und Rekombinationsoperatoren nicht kodiert und direkt durch die evolutionären Operationen manipuliert werden.

Wie in Kapitel 4.3 beschrieben, werden bei ES verschiedene Selektionsschemen unterschieden, die auf der Anzahl sowie dem Verhältnis der Eltern zur Anzahl der generierten Kinder pro Elter sowie auf dem Übertrag der Eltern in die neue Generation basieren. Im Rahmen des in diesem Kapitel behandelten K-d Tree basierten Layoutsolvers wurde die (μ + λ)-ES eingesetzt, um gute Lösungen aus einer Elterngeneration nicht zu verlieren, sondern ihnen die Möglichkeit zu geben, erneut Nachkommen zu bilden. Dies birgt jedoch grundsätzlich die Gefahr, dass eine gute Lösung und ihre Nachkommen den Lösungsraum mit Fortschreiten der Evolution dominieren und die Evolution in lokalen Maxima verharrt. Dem wurde durch einen Anteil an zufällig ausgewählten, nichtoptimalen Lösungen aus der Population bei der Bildung der Elterngeneration entgegengewirkt.

Der evolutionäre Prozess gestaltet sich folgendermaßen. Zunächst wird eine Elterngeneration initialisiert und die Fitness der Individuen bestimmt. Die eingesetzte Bewertungsfunktion ergibt sich aus der Abweichung der tatsächlichen Flächeninhalte der Räume von ihren Idealwerten:

$$f = \sum_{i=1}^{n} \left| A_i - A_i' \right| \tag{22}$$

wobei n = Anzahl der Flächen; A = tatsächlicher Flächeninhalt; A' = idealer Flächeninhalt. Ein Individuum ist folglich umso fitter, je geringer die Abweichungen von den tatsächlichen zu den idealen Flächeninhalten der Räume ausfallen. Ziel ist es, diese Abweichungen zu minimieren, der Fitnesswert einer idealen Layoutlösung beträgt folglich 0.

Anschließend werden durch Mutation von jedem Elternindividuum λ Kinder erzeugt. Zur Erzeugung eines Kindes werden mit einer Mutationswahrscheinlichkeit m die Positionen der Punkte mutiert, d.h. ihre Lage verändert (Abb. 31). Nach der Evaluierung aller Lösungen werden aus der Kinderpopulation und ihrer Elterngeneration μ Individuen ausgewählt. Hierbei kann es sich um die μ Individuen mit den besten Fitnesswerten handeln oder, wie bereits erwähnt, um eine Kombination aus den fittesten mit einigen zufällig ausgewählten Individuen. Diese μ Individuen bilden die neue Elterngeneration.

Abb. 31 zeigt das Beispiel eines typischen Layouts mit zehn Räumen, dass durch die reine Raumgrößenoptimierung eines mittelwertberechneten K-d Trees mit einer einfachen Schnittregel, d.h. die Reihenfolge der Schnittdimensionen wird als kontinuierliche Abfolge von Schnitten in x- und y-Richtung festgelegt, generiert wurde.

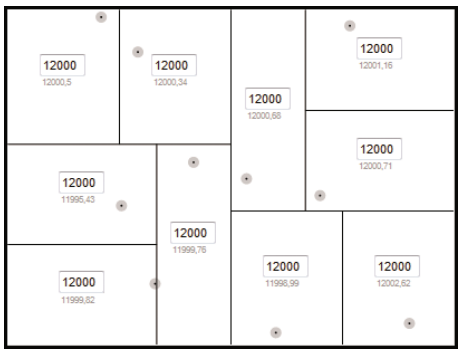


Abb. 31: Optimierung der Raumgröße von K-d Tree generierten Layouts durch eine ES.

6.2.4. Suche nach bestimmten Nachbarschaftsverhältnissen

Die Suche nach bestimmten Nachbarschaftsverhältnissen zwischen den Punkten bzw. den ihnen zugeordneten Räumen ist ein topologisches Problem. Es besteht darin, den Räumen die Funktionen so zuzuordnen, dass sie die gewünschten Nachbarschaftsbeziehungen aufweisen (Eckert, et al., 1999; Elezkurtaj, 2004). Darüber hinaus sind die Schnittdimensionen so zu wählen, dass durch die resultierende Un-

terteilung die gewünschten Lagebeziehungen zwischen Räumen entstehen. Die gewünschten Nachbarschaftsbeziehungen zwischen Räumen werden vom Nutzer definiert.

Um die Nachbarschaftsbeziehungen zu optimieren, müssen zum einen die den Flächen zugeordneten Funktionen und zum anderen die Schnittdimensionen so lange vertauscht werden, bis die entstandene Layoutlösung die vorgegebene Topologie aufweist (Abb. 32). Allen unseren weiteren Untersuchungen liegt die in Kapitel 5.2 beschriebene Sterntopologie mit der in (18) formulierten Restriktion zugrunde.

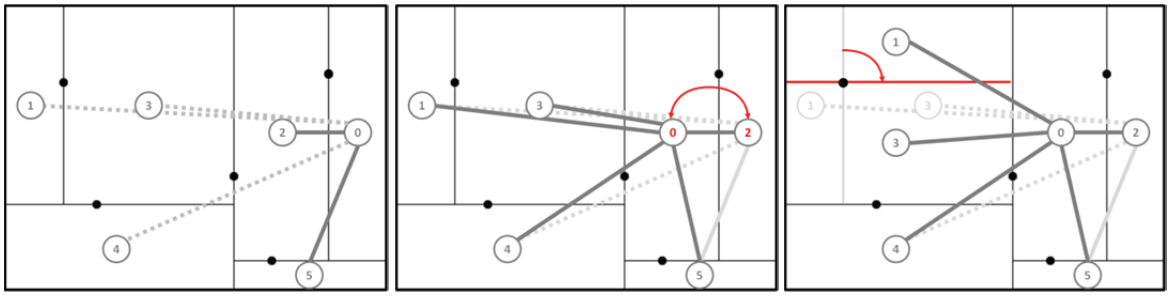


Abb. 32: Topologisches Problem (links), Optimierung der Nachbarschaften durch Vertauschen von Indizes (Mitte) und Änderung der Schnittrichtung (rechts).

Die topologische Optimierung lässt sich durch GAs vornehmen. Ein grundlegender Aspekt der GA ist, dass die Parameter der Problemstellung zunächst im Rahmen der Suche kodiert werden. Die kodierten Elemente werden als Chromosomen bezeichnet. In der ursprünglichen Variante des GA stellen sie eine Bitfolge dar, welche die Parameter des Problems repräsentiert (Whitley, 1994). Sie sind die Genotypen, die vom GA manipuliert werden. In abgewandelter Form kann der Genotyp auch aus einer Folge von Elementen anderer Typen bestehen (Bäck, 2000a).

Im konkreten Fall werden die Raumindizes als Chromosom kodiert und repräsentieren so den Phänotyp, eine Layoutlösung. Die Raumindizes können bei mittelwertberechneten K-d Trees Werte zwischen 0 und $n\text{-}1$ annehmen, bei medianberechneten K-d Trees Werte zwischen 0 und $n\text{-}2$, wobei n die Anzahl der Räume darstellt. Voraussetzung ist, dass jeder Indexwert nur einmal vorkommt und folglich jedem Index ein Raum eindeutig zugeordnet werden kann und umgekehrt. Die Raumfolge ergibt sich aus der Abfolge ihrer Entstehung beim Aufbau des K-d Trees, die Kodierung der Raumindizes erfolgt über deren Zuordnung zur Raumfolge (Abb. 33).

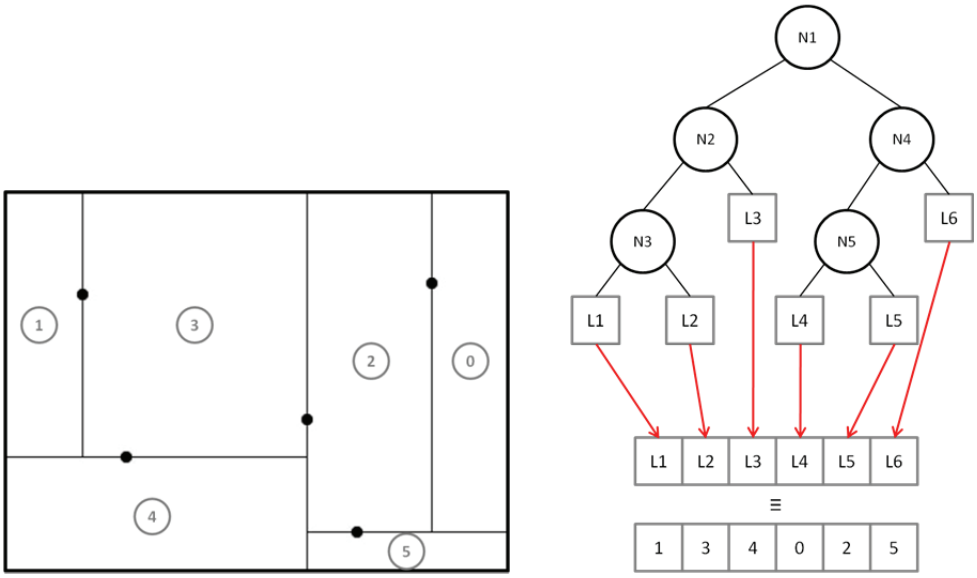


Abb. 33: Kodierung der Raumindizes aus der Raumabfolge, Phänotyp (links), Genotyp (rechts).

Die Schnittdimensionen der Layoutlösung werden als Unterteilungsfolge kodiert, die sich ebenfalls aus der Baumstruktur ergibt (Abb. 34). Anders als bei herkömmlichen K-d Trees werden durch die Verwendung eines GA die Schnittdimensionen nicht im Voraus oder durch die Punktverteilung festgelegt, sondern können dynamisch variiert werden. Die Schnittdimensionen können Werte zwischen 0 und $k\text{-}1$ annehmen, wobei k der Anzahl der Raumdimensionen entspricht. In einem zweidimensionalen Raum repräsentiert 0 eine Unterteilung in x-Richtung, 1 eine Unterteilung in y-Richtung (Abb. 34).

Das Genom beschreibt die Abfolge der Schnittdimensionen in den Knoten in Reihenfolge der Baumebenen von der Wurzel bis zu den Blättern. Das Genom hat eine Länge von:

$$L = \sum_{i=0}^{n-1} 2^i \tag{23}$$

wobei n die Anzahl der Baumebenen darstellt.

Das Genom hat damit in der Regel mehr Stellen, als eine Layoutlösung tatsächliche Unterteilungen bzw. ein K-d Tree interne Knoten besitzt. Stattdessen besitzt es jeweils eine Stelle für jeden theoretisch möglichen Knoten auf allen Baumebenen (Abb. 34). Diese Form der Kodierung wurde gewählt, um jede Unterteilung eindeutig einer Position im Baum zuordnen zu können und umgekehrt. Würden die Unterteilungen stattdessen als kontinuierliche Folge ähnlich der Raumindizes kodiert, so

könnten, bedingt durch die unterschiedliche Verteilung der Punkte und die damit in Verbindung stehende Lage der Knoten im Baum, zwei Lösungen mit demselben Unterteilungsgenom zu zwei unterschiedlichen Unterteilungsstrukturen auf Ebene der Phänotypen führen.

Das Genom wird immer vollständig initialisiert, sodass jederzeit Veränderungen in der Baumstruktur, d.h. Verschiebungen von Knoten und Ästen, vorgenommen werden können. Ist ein Knoten im Baum nicht besetzt, wird der zugehörige Schnittwert nicht aus dem Genom ausgelesen bzw. ist für die Bildung der aktuellen Layoutlösung nicht relevant.

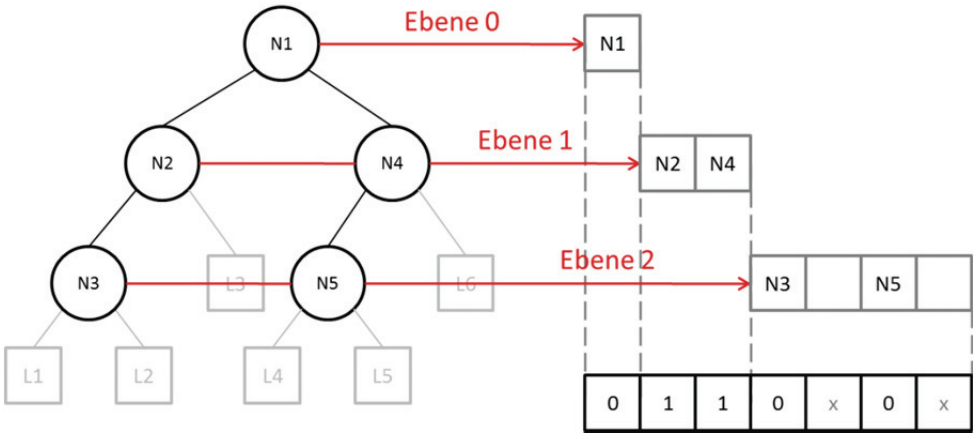


Abb. 34: Kodierung der Unterteilungsfolge.

Die Umsetzung des GA zur Optimierung der Nachbarschaften erfolgte anhand des in Tab. 2 beschriebenen Schemas für EA. Die im Rahmen des GA eingesetzte Evaluierungsfunktion betrachtet die gewünschten Nachbarschaftsbeziehungen und vergleicht sie mit der tatsächlichen Lage der betreffenden Räume in der aktuellen Layoutlösung. Sie berechnet die Summe aller Distanzen zwischen den Räumen gewünschter Nachbarschaften und lässt sich folgendermaßen beschreiben:

$$f = \sum_{i=1}^{n} \overline{A_l B_l} \tag{24}$$

wobei A und B die Räume darstellen, die benachbart sein sollen, und n die Anzahl der gewünschten Nachbarschaften.

Der Abstand zwischen zwei direkt benachbarten Räumen beträgt null, wobei eine Mindestdurchgangsbreite bei der Überlappung gegeben sein muss, um sie als be-

nachbart gelten zu lassen. Eine Mindestdurchgangsbreite wird berücksichtigt, um später das Platzieren von Verbindungen zwischen den Räumen, wie Türen, zu ermöglichen. Der aus der Evaluierungsfunktion resultierende Fitnesswert ist umso niedriger, je mehr gewünschte Nachbarschaftsbeziehungen tatsächlich in der aktuellen Layoutlösung existieren und je näher diese Räume zueinander liegen. Ein niedriger Fitnesswert ist folglich vorteilhaft für die Performance einer Lösung. Ihr Idealwert tendiert gegen null.

Die Rekombination zweier Elternindividuen wird mittels One-Point-Crossover der Unterteilungs- und Raumindexgenome durchgeführt, bei dem die Chromosomen der Eltern an einer Schnittstelle getauscht werden, sodass zwei neue Chromosomen entstehen (Abb. 35). Die Selektion der Eltern zur Rekombination kann auf verschiedene Arten stattfinden, beispielsweise über Roulette-Wheel-Selection oder wie die in diesem Fall verwendete Binary Tournament Selection. Hier werden zweimal zwei Individuen der Elterngeneration zufällig ausgewählt, wobei aus dem direkten Vergleich jeweils das Individuum mit dem besseren Fitnesswert zur Reproduktion bestimmt wird.

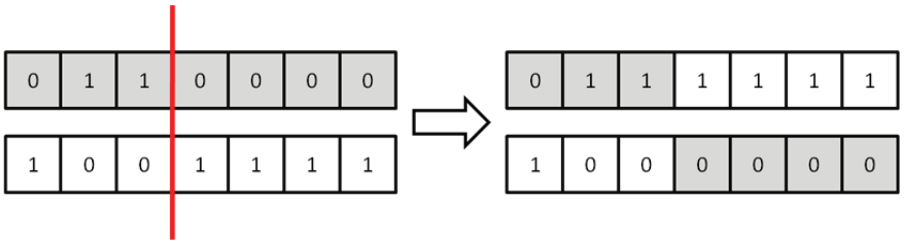


Abb. 35: One-Point-Crossover.

Mithilfe des Mutationsoperators *m* werden die durch Rekombination entstandenen Kinder anschließend mutiert. Mutation bedeutet in diesem Fall, dass die Indizes einzelner Räume miteinander vertauscht werden bzw. die Dimension einer Stelle im Genom geändert wird. Durch (μ + λ)-Selektion wird eine neue Elterngeneration gebildet.

Die Suche nach bestimmten Nachbarschaftsverhältnissen erfolgt, ebenso wie die Suche nach bestimmten Raumgrößen, in einer eigenständigen Population. Die Nachbarschaftsbeziehungen bzw. die Raumgrößen wurden folglich in zwei Populationen parallel optimiert. Zusätzlich wurden die besten Individuen bzw. die besten Lösungen in jeder Generation zwischen den Populationen ausgetauscht, d.h. mig-

riert (Abb. 36). Diese erste Optimierungsstrategie wurde in der Folge durch ein multikriterielles System ersetzt, da die Migration die Generierung von in beiden Kriteriendimensionen optimalen Lösungen nur in geringem Maße beeinflusste bzw. begünstigte.

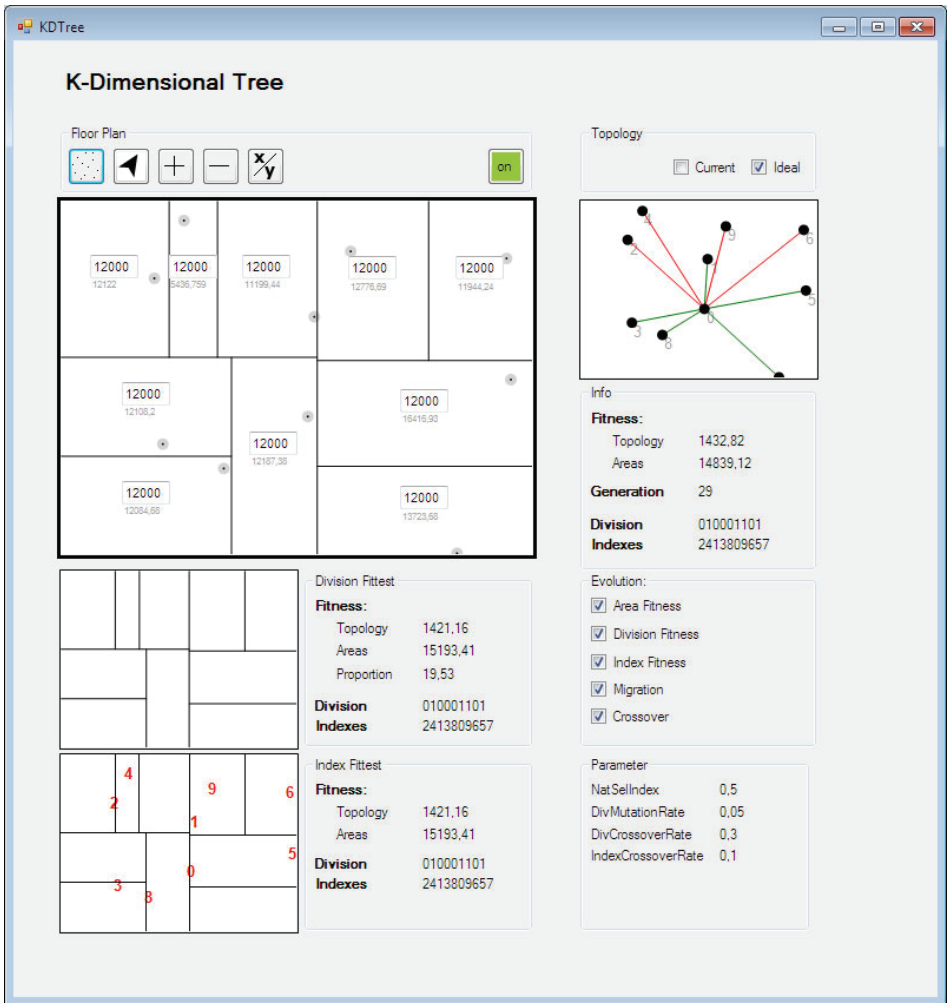


Abb. 36: Optimierung in zwei Kriteriendimensionen, nach Raumgrößen und Nachbarschaften.

6.2.5. Multikriterielles System

Wie gesehen beinhalten Layoutprobleme oft konkurrierende Parametersets. Die Lösung solcher Probleme besteht darin, die Parameterwerte so anzupassen, dass sie eine optimale Lösung ergeben. Im Gegensatz zu Problemstellungen mit nur einem Optimierungsziel bedeutet dies, das bei der Lösung Kompromisse eingegangen werden müssen, da häufig die Verbesserung einer Lösung hinsichtlich eines Parameters die Verschlechterung des Ergebnisses hinsichtlich eines anderen Parameters bedeutet. Wie bereits in Abschnitt 5.2.3 eingeführt, spricht man hier von MOOP (Zitzler, 1999).

Ein MOOP wird definiert als ein Problem zur Findung eines *"vector of decision variables which satisfies constraints and optimizes a vector function whose elements represent the objective functions"* (Coello, 1998, p. 270).

Im Allgemeinen können, wie bei (Horn, 1997) beschrieben, zwei Problemstellungen bei der Lösung von MOOPs ausgemacht werden: Die Suche nach geeigneten Lösungen in einem komplexen Lösungsraum und die multikriterielle Entscheidungsbildung, d.h. die Auswahl einer geeigneten Kompromisslösung aus einem Set an Lösungen, die von einem menschlichen Entscheidungsträger getroffen werden muss.

Abhängig von der Kombination von Suche und Entscheidungsfindung im Optimierungsprozess können die eingesetzten Optimierungsmethoden unterschieden werden in solche, bei denen die Entscheidungsbildung vor, nach oder während der Suche stattfinden (Horn, 1997). In der ersten Kategorie werden die Kriterien vor der Suche durch Präferenzbildung in ein einzelnes Kriterium überführt, wie beispielsweise im Weighted Sum Approach (Cohon, 1978). Dadurch können die Optimierungsprobleme wie einzelkriterielle Probleme und mit traditionellen Methoden gelöst werden. Dieser Ansatz setzt jedoch bereits ein erhebliches Vorwissen über die Gewichtung der verschiedenen Parameter für die Lösungssuche voraus, die nicht bei allen Problemstellungen gegeben ist (Zitzler, 1999).

Methoden der zweiten Kategorie führen zunächst die Optimierung ohne Präferenzen durch. Es ergibt sich ein Set an Lösungen im Entscheidungsraum, pareto-optimales Set oder auch Pareto-Front genannt, aus dem der Entscheidungsträger auswählt (Zitzler, Laumanns, & Bleuler, 2003). Lösungen im pareto-optimalen Set gelten dann als pareto-optimal, wenn eine Lösung von keiner anderen Lösung dominiert wird, d.h. wenn ihnen unter Betrachtung aller Kriterien keine andere Lösung überlegen ist (Zitzler, 1999). Sie können in keinem Kriterium verbessert werden, ohne dass sie sich nicht in mindestens einem anderen Kriterium verschlechtern.

In der dritten Kategorie kann der Entscheidungsträger bereits während der Suche Präferenzen treffen. Er erhält nach jedem Suchlauf eine Auswahl an Alternativlösungen, aufgrund derer er weitere Kriterien bestimmt, die in den weiteren Suchlauf mit einfließen. Dieser sowie der zuvor beschriebene Ansatz erhöhen die Komplexität

des Suchraums dadurch, dass zunächst keine Eingrenzung getroffen wird. Eine Schwierigkeit besteht insbesondere in höher-dimensionalen MOOPs in der Darstellung der Ergebnisse (Zitzler, 1999).

Da die Suche nach dem pareto-optimalen Set rechenintensiv und aufgrund der Komplexität nicht direkt berechnet werden kann, wurden für die Lösungsprozesse in der zweiten und dritten Kategorie stochastische Suchstrategien entwickelt, die sich der Pareto-Front langsam annähern. Dazu zählen neben der Ant Colony Optimization, dem Simulated Anealing oder der Tabu Search insbesondere auch die Evolutionäre Algorithmen, da man mit ihnen zum einen große Suchräume sowie komplexe Problemstellungen handhaben und zum anderen durch ihre auf Populationen basierende Struktur innerhalb eines Evolutionslaufs mehrere pareto-optimale Lösungen finden kann (Zitzler, et al., 2003).

Im Bereich der sogenannten Multiobjective Evolutionary Algorithms (MOEAs) existieren verschiedene Ansätze. Sie unterscheiden sich im Wesentlichen durch die verwendeten Selektionsstrategien, d.h. durch die Art und Weise, wie die Fitness der Individuen bestimmt wird und diese zur Mutation und Bildung der Population ausgewählt werden. MOEAs wie der Vector-Evaluated Genetic Algorithm (VEGA) optimieren beispielsweise einzelne Kriterien parallel. Andere wie der Multiobjective Genetic Algorithm (MOGA) gewichten Kriterien und verwenden eine summierte Fitnessfunktion. Darüber hinaus kann sich der Optimierungsprozess an der Pareto-Front orientieren, wie zum Beispiel bei Pareto-Ranking (Goldberg, 1989), bei dem die Individuen einer Population nach deren Pareto-Optimalität sortiert werden. Weitere Selektionskriterien bieten das Niching, das die Fitness bezogen auf die Umgebung eines Individuums im Lösungsraum berechnet, und verschiedene Arten der Tournament Selection, in denen eine Gruppe an Individuen ausgewählt und verglichen werden, um das fitteste Individuum zur Selektion zu bestimmen (Horn, 1997). Viele MOEAs basieren auf einer Kombination der verschiedenen dargestellten Möglichkeiten und variieren oder erweitern deren Grundkonzepte.

Im Rahmen des oben vorgestellten generativen Mechanismus handelte es sich bei den zu optimierenden Parametern um die Raumgrößen und die Nachbarschaftsbeziehungen in architektonischen Layouts. Die Schwierigkeit besteht darin, diese unterschiedlichen Parameter so zu kombinieren, dass Lösungen mit möglichst optima-

len Raumgrößen und Nachbarschaften entstehen können. Dazu wurden in einer ersten Versuchsreihe die Kriterien zunächst auf Basis des VEGA-Ansatzes in jeweils eigenen Populationen parallel optimiert. Die Individuen dieser Subpopulationen wurden zu einer gemeinsamen Gesamtpopulation vermischt (schematische Darstellung in Abb. 37). Zur Generierung der Kindergeneration wurden die in 6.2.1 beschriebenen evolutionären Operationen verwendet. Aus den entstandenen Kindern wurden anschließend die neuen Subpopulationen erstellt.

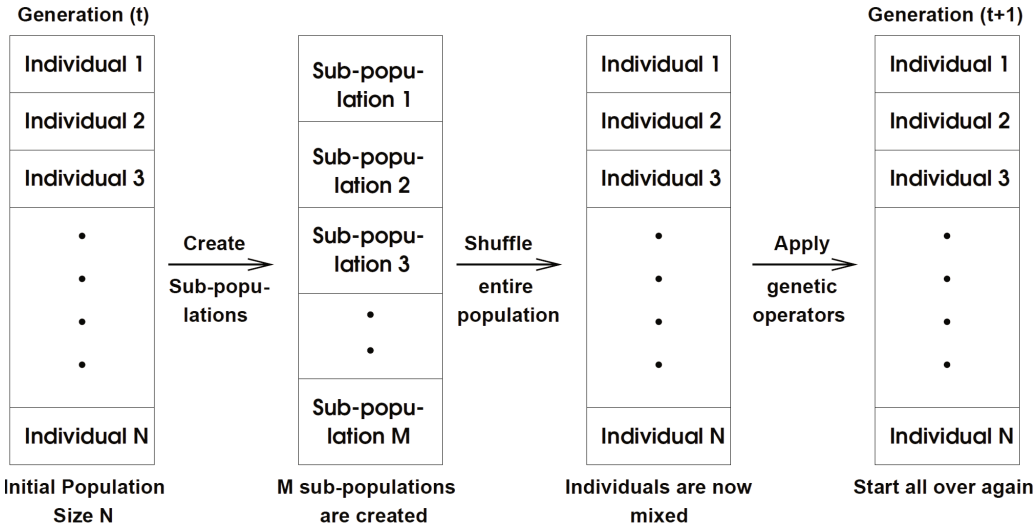


Abb. 37: Schematische Darstellung des VEGA, wobei N die Größe der Gesamtpopulation und M die Anzahl der Kriterienfunktionen darstellt. Abbildung aus (Coello, 1998).

Der VEGA zeichnet sich vor allem durch seine Einfachheit aus. Aus seinem Aufbau ergeben sich jedoch auch Probleme. Zum einen ist ein lokal in einer Subpopulation nicht-dominiertes Individuum nicht notwendigerweise auch global nicht-dominiert, d.h. lokal optimale Individuen sind nicht notwendigerweise auch global besser als andere. Zum anderen unterstützt der Selektionsprozess die Bildung von Spezies, also Gruppen von Individuen, die in verschiedenen Kriterien besonders gute Ergebnisse liefern, da vor allem diese Individuen selektiert werden. Dadurch gehen solche Lösungen verloren, die eine mittelmäßige Performance in allen Kriteriendimensionen erreichen und die zur Auswahl von Kompromisslösungen günstig wären (Coello, 1998).

Aufgrund der genannten Nachteile von VEGA wurde der Pareto-Envelope-based Selection Algorithm (PESA) getestet, der bei der Selektion und Optimierung die Lö-

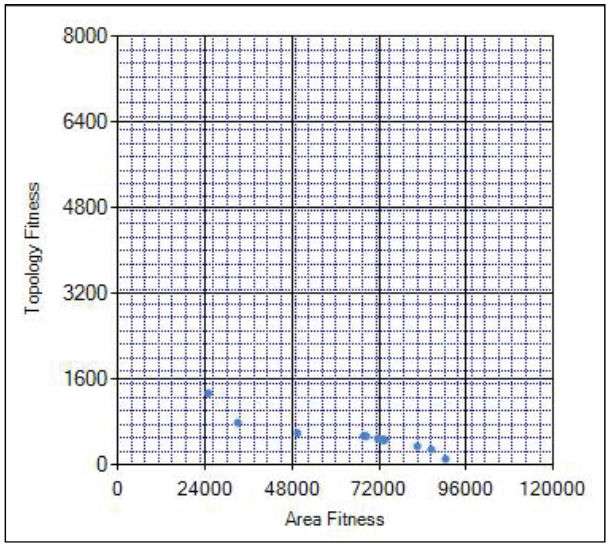
sungsvielfalt über den Lösungsraum hinweg erhält und die Pareto-Front speziell in noch unerforschte Bereiche vorantreibt (Corne, et al., 2000).

PESA zeichnet sich durch eine kleine interne Population aus, welche die neugebildeten Individuen enthält, die noch evaluiert werden müssen, und verfügt über eine meist größere externe Population, auch als Archiv bezeichnet, die die Individuen enthält, welche die aktuelle Pareto-Front bilden. Darüber hinaus wird der Lösungsraum gerastert, sodass ein sogenanntes Hypergrid entsteht, anhand dessen die Verteilung bzw. die Häufung der Lösungen auf der Pareto-Front kontrolliert wird. Dieser Häufungswert, der sogenannte Squeezefaktor, dient als Auswahlkriterium für die Individuen (Corne, et al., 2000). Der PESA Algorithmus ist in Tab. 6 schematisch dargestellt.

Algorithmus:	Pareto-Envelope-based Selection Algorithm (PESA)
Parameter:	IP = Interne Population P_i = Anzahl der Individuen in der internen Population $P_{e\,max}$ = Maximale Anzahl an Individuen in der Externen Population r = Rekombinationswahrscheinlichkeit
Output:	EP = Externe Population, Pareto-Front von optimalen Lösungen
Logik:	*Schritt 1:* Generation t = 0 *Schritt 2:* Initialisiere $IP(t)$ mit P_i Individuen und initialisiere eine leere $EP(t)$ *Schritt 3:* Evaluiere $IP(t)$ und verschiebe nicht-dominierte Elemente von $IP(t)$ nach $EP(t)$ *Schritt 4:* Wenn die maximal zugelassene Anzahl an Individuen $P_{e\,max}$ in $EP(t)$ überschritten ist, lösche Individuen aus $EP(t)$ bis Anzahl gleich $P_{e\,max}$ *Schritt 5:* Wenn ein Abbruchkriterium erreicht wurde, halte den Algorithmus an und gebe das Set an Individuen in $EP(t)$ als Ergebnis zurück Sonst leere $IP(t)$ und führe die folgenden Anweisungen aus bis P_i neue Lösungen generiert wurden: a) Wähle zwei Eltern mit einer Wahrscheinlichkeit r aus $EP(t)$ aus, produziere ein Kind durch Crossover und mutiere das Kind b) Wähle mit der Wahrscheinlichkeit $(1 - r)$ ein Elternindividuum aus $EP(t)$ aus und mutiere es, um ein Kind zu erstellen *Schritt 6:* $t = t + 1$ *Schritt 7:* Beginne wieder bei Schritt 3

Tab. 6: Pareto-Envelope-based Selection Algorithm (Corne, et al., 2000).

Die Rekombination und Mutation (Tab. 6, Schritt 5) erfolgten analog zu den in den Kapiteln 6.2.3 und 6.2.4 beschriebenen Schemata. Die Dominanz zwischen den Individuen wurde über die ebenfalls dort beschriebenen Bewertungsfunktionen bestimmt. Bei der Verschiebung der nicht-dominierten Elemente aus der internen in die externe Population wird Individuum für Individuum geprüft, ob es von keiner anderen Lösung in der internen und der externen Population dominiert ist. Nur wenn es nicht dominiert ist, wird es ins Archiv übernommen (Abb. 38). Gleichzeitig werden auch alle Archivlösungen neu evaluiert und dominierte Individuen gegebenenfalls aus der externen Population entfernt. Überschreitet die Anzahl an Individuen in der Externen Population eine festgelegte Höchstzahl, werden Individuen auf Basis des Squeezefaktors entfernt (Corne, et al., 2000).



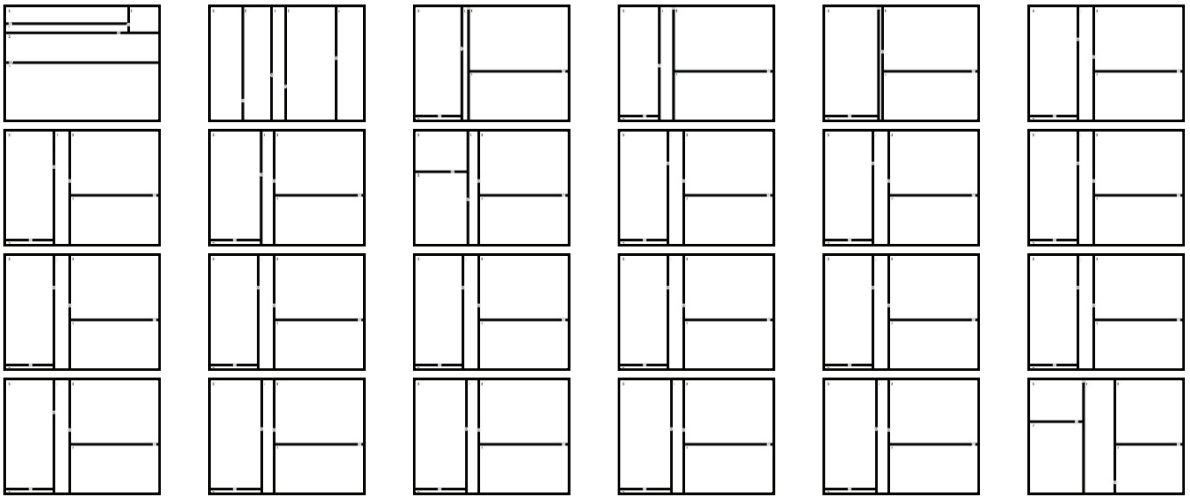


Abb. 38: Pareto-Front (oben) und zugehörige Archivlösungen (unten) (Generation: 9).

Der Squeezefaktor berechnet sich folgendermaßen: Der mehrdimensionale Lösungsraum wird durch ein Hyperraster in Hyperboxen unterteilt, wobei jedes Individuum einer Hyperbox zugeordnet werden kann. Die Anzahl der Lösungen in einer Box bestimmen deren Squeezefaktor. Für das Update des Archivs wird zunächst der maximale Squeezefaktor der Population bestimmt und anschließend ein Element aus der Hyperbox mit dem größten Squeezefaktor zufällig ausgewählt und entfernt.

Die Auswahl der Eltern erfolgt auf Basis der Häufung der Archivlösungen in bestimmten Regionen durch Binary Tournament Selection. Zwei Lösungen werden hierbei durch Zufall aus dem Archiv ausgewählt. Das Individuum mit dem kleinsten Squeezefaktor wird zur Generierung des Kinds hergenommen, um die Pareto-Front in weniger frequentierte Regionen zu erschließen.

Durch den PESA entwickelte sich eine Pareto-Front, die relativ schnell gute Ergebnisse in der Optimierung der Nachbarschaftsbeziehungen und der Raumgrößen zeigte (Abb. 39).

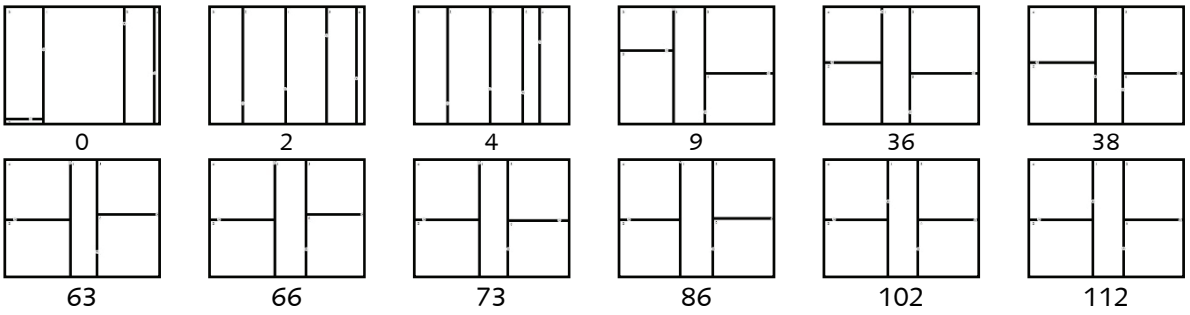


Abb. 39: Optimierung der Raumgrößen: Fitteste Individuen nach Generationen 0 bis 112.

6.2.6. Verschachtelte K-d Trees

Die Datenstruktur des K-d Trees unterstützt das Konzept des Entwurfs auf verschiedenen Maßstabsebenen mit entsprechend verschachtelten Elementen, wie es in Kapitel 10 detailliert beschrieben wird. Layoutelemente entsprechen, bezogen auf die Datenstruktur, externen Knoten, also Elementen untergeordneter Hierarchieebenen, denen anschließend bestimmte Funktionen innerhalb des Layouts zugeordnet werden können. Interne Knoten entsprechen Elementen übergeordneter Hierarchieebenen, die untergeordnete interne und externe Knoten beispielsweise zu Funktionsbereichen bündeln können. Die internen und externen Knoten stellen in der Implementierung folglich zentrale Objekte dar, die durch einen Datenpunkt

und seine Lage sowie die zugehörige Region definiert werden. Gleichzeitig kann auch der K-d Tree selbst als Objekt betrachtet werden, das die Baumstruktur als Referenzen zwischen Root und Child Nodes speichert.

Im Rahmen einiger Prototypen wurde das Prinzip der verschachtelten K-d Trees versuchsweise implementiert (und Abb. 41). Dazu wurde zunächst die Grunddatenstruktur des K-d Trees angepasst und zusätzliche Verweise zwischen parent und child trees eingefügt, die es ermöglichen, in den leaves eines K-d Trees ein diesem Baum untergeordnetes child tree zu pflanzen. Die dem leaf zugeordnete Region gibt die Begrenzungen für dieses child tree vor.

Jeder verschachtelte K-d Tree stellt eine Layoutproblematik für sich dar, die es im Rahmen des generativen Mechanismus zu lösen gilt. Folglich ist auch der generative Mechanismus verschachtelt. Jede Layoutanpassung und -veränderung durch evolutionäre Prozesse oder Nutzerinteraktion auf einer höheren Hierarchiestufe bedingt automatisch die Anpassung und Veränderung der abhängigen, untergeordneten Strukturen und ihrer evolutionären Prozesse.

Theoretisch ist es denkbar, K-d Trees unendlich tief zu verschachteln, praktisch stößt die Verschachtelung allerdings an Grenzen. Beispielsweise kommt es mit zunehmender Verschachtelungstiefe zu Darstellungs- und Rechenleistungsproblemen oder die Tiefe der Verschachtelung ist begrenzt durch die Existenz von konkreten, sinnvollen Entsprechungen auf architektonischer und städtebaulicher Ebene.

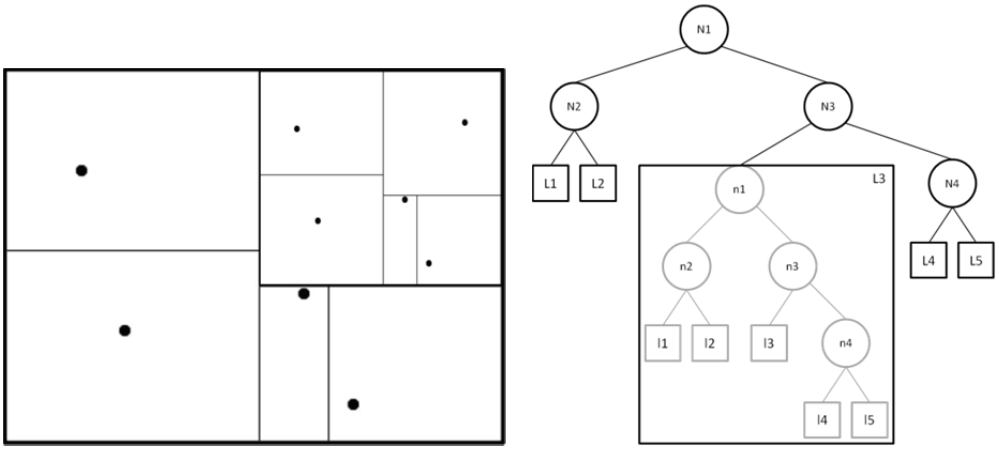


Abb. 40: Geometrische und grafische Darstellung eines verschachtelten K-d Trees.

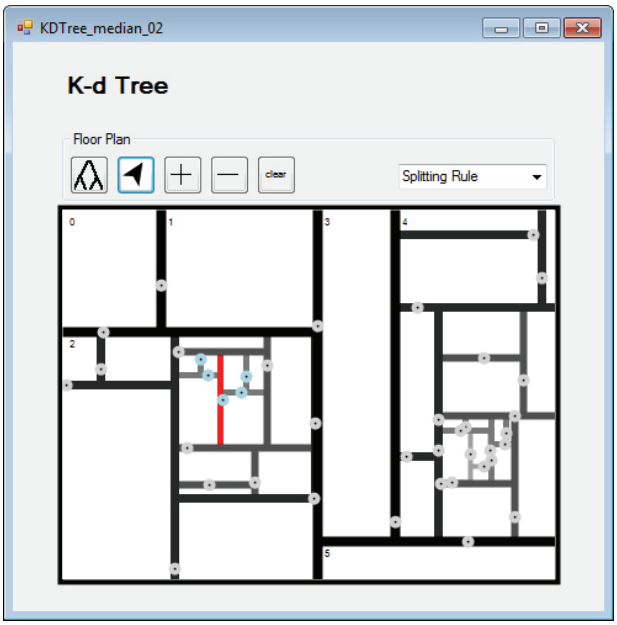


Abb. 41: Prototypische Umsetzung eines verschachtelten K-d Trees.

6.3. Konklusion und Ausblick

In diesem Kapitel wurden die geometrischen Qualitäten und gestalterischen Möglichkeiten der Raumunterteilung mittels K-d Tree Algorithmen zur Lösung von Layoutproblemen in Architektur und Städtebau ausgelotet. Es wurde zunächst ihre Einsatzmöglichkeit zur Generierung von Grundrissen betrachtet. In den entsprechenden Untersuchungen hat sich gezeigt, dass K-d Tree Algorithmen aufgrund ihrer geometrischen Struktur in Verbindung mit ES und GA sinnvolle Layoutlösungen für Grundrisse generieren können. Durch einen flexiblen Aufbau der K-d Tree Struktur ohne festgelegte Teilungsregel und den Einsatz median- und mittelwertbezogener Berechnungen zeigen sich die entstandenen Layout-Lösungen flexibel, divers und können dynamisch angepasst werden. Die Verwertung und Weiterverarbeitung der erhaltenen Lösungen bedarf jedoch weiterer Untersuchungen.

Durch evolutionäre Algorithmen konnten Raumgrößen und Nachbarschaften der Unterräume der K-d Trees gezielt beeinflusst werden. Die punktbasierte Charakteristik des K-d Tree Algorithmus bietet dabei Vorteile bei der Generierung der Lösungen sowie deren Optimierung. Durch den Aufbau eines multikriteriellen Optimierungssystems können Zielkriterien über den gesamten Lösungsraum hinweg optimiert werden. Die entstehende Pareto-Front aus nicht dominierten Lösungen kann dem Nutzer bei der Entscheidungsfindung und der Auswahl von Kompromiss-

lösungen unterstützen. Im Idealfall soll es dem Anwender möglich sein, den Lösungsraum flexibel zu durchsuchen und in seinem Sinne zu verändern. Hierzu muss das bisher entwickelte System allerdings noch erweitert werden, um dem Nutzer sinnvolle Funktionen und Interaktionsmöglichkeiten zur Verfügung stellen zu können.

Im Rahmen der weiteren Ausarbeitung des vorgestellten Systems wären Erweiterungen des multikriteriellen Optimierungsmodells sinnvoll. Zu untersuchen wären zum Beispiel die Erweiterung um topologische Kriterien wie die Raumausrichtung oder die Integration von klassischen Analysewerkzeugen für Sonnenstand und Sichtbarkeiten. Darüber hinaus erschließen die umgesetzten generativen Mechanismen zum aktuellen Zeitpunkt nur einen begrenzten Teil des möglichen Lösungsraums, da zunächst eine Beschränkung auf orthogonale Unterteilungen und rechteckige Raumstrukturen stattfand. Im Rahmen einer Weiterbearbeitung bestünde die Möglichkeit, die Generierung von komplexer Geometrie zu untersuchen und damit die orthogonale Unterteilungsweise der K-d Trees auf nichtrechtwinklige Ebenen zu erweitern.

Abschließend kann festgestellt werden, dass die Raumpartitionierung durch K-d Trees in Verbindung mit evolutionären Algorithmen bei der Entwicklung von Methoden zur kreativen algorithmischen Lösung von Layoutaufgaben in Architektur und Städtebau eine interessante und vielversprechende Variante zu bereits bekannten Strategien darstellt.

7. Layouts mittels Unterteilungsalgorithmen[10]

Katja Knecht, Reinhard König

Das Unterteilen einer vorgegebenen Grundfläche in Zonen und Räume ist eine im Architekturentwurf häufig eingesetzte Methode zur Grundrissentwicklung. Für deren Automatisierung wurden im Rahmen des Forschungsprojekts Kremlas sogenannte Slicing Trees, die im konkreten Fall einen vorgegebenen, mehrdimensionalen Raum nach einer festgelegten Syntax unterteilen und in einer Baumstruktur organisieren, betrachtet und ihr generatives und gestalterisches Potential im Hinblick auf architektonische Fragestellungen und ihre Einsatzmöglichkeiten zur Layoutgenerierung untersucht.

Es entstand ein Layoutsolver, der eine Unterteilungsfolge zufällig erstellt und Grundrisse mit einer festgelegten Anzahl an Räumen mit bestimmter Raumgröße durch Unterteilung generiert. In Kombination mit evolutionären Algorithmen werden die Layoutlösungen hinsichtlich der Nachbarschaftsbeziehungen zwischen einzelnen Räumen optimiert. Dieses Kapitel dokumentiert unsere Untersuchungen und ihre Ergebnisse.

7.1. Slicing Trees

Unterteilungsalgorithmen sind Algorithmen, die Flächen oder mehrdimensionale Räume nach bestimmten Regeln oder einer festgelegten Abfolge unterteilen und in Form einer Baumstruktur, dem sogenannten Slicing Tree, organisieren. Sie zeichnen sich unter anderem dadurch aus, dass Elemente durch Unterteilung einer vorgegebenen Grundfläche dicht gepackt werden können, und entsprechen einer in Architektur und Stadtplanung oft eingesetzten Entwurfsmethodik. Die Computerspielebranche machte sich diese Methodik, jedoch zunächst ohne architektonischen Anspruch, zunutze und setzte Unterteilungsalgorithmen in der automatischen Generie-

[10] Dieses Kapitel basiert auf dem Artikel von Knecht, K. & Koenig, R. (2011). Evolutionäre Generierung von Grundriss-Layouts mithilfe von Unterteilungsalgorithmen. Weimar: Bauhaus-Universität Weimar.

rung von Stadtgrundrissen und -strukturen bis hin zu Gebäudegrundrissen ein, um flexible Spielwelten zu schaffen und begehbar zu machen (Hahn, et al., 2006; P Müller, et al., 2006). Marson und Musse (2010) haben die Verwendung von quadratisierten Unterteilungsbäumen zur Echtzeitgenerierung von architektonisch sinnvollen Grundrissen und deren Einsatzmöglichkeiten untersucht.

Im Rahmen unserer Untersuchungen verstehen wir unter einem Slicing Tree Algorithmus die rekursive Unterteilung einer Fläche durch kantenparallele Schnittlinien in kleinere Rechtecksflächen. Die Unterteilung kann auf die entstehenden Unterflächen bis zu einer festgelegten Tiefe oder nach einer festgelegten Abfolge weiter angewendet werden (Tab. 7) (Otten, 1982). Die Auswahl der Schnittdimension und die Bestimmung der Lage der Unterteilung kann zufällig oder nach festgelegten Regeln erfolgen, d.h., dass eine Fläche zum Beispiel immer nach der längeren Seite in einem festgelegten Proportionsverhältnis (Abb. 42, links) unterteilt wird oder so, dass alle resultierenden Unterräume den gleichen Flächeninhalt besitzen (Abb. 42, rechts).

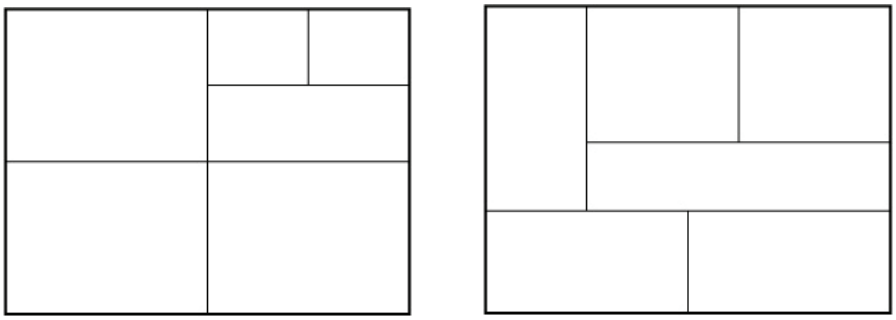


Abb. 42: Unterteilung im Verhältnis 1:1 (links) und nach gleichen Raumgrößen (rechts).

Algorithmus:	Unterteilung einer Fläche und Aufbau eines Unterteilungsbaums
Input:	Rechtecksfläche *R*
Output:	Unterteilungslayout, *t* vom Typ Slicing Tree
Logik:	*Schritt 1:* Generation $t = 0$ *Schritt 1*: If *R* ist leer return leeres Slicing Tree *Schritt 2:* Bestimme und generiere die Schnittebene s sowie den Unterteilungsknoten *N* mit den folgenden Werten: *SplitDim* = die Schnittdimension *SplitVal* = der nach der Schnittabfolge bzw. Schnittverhältnis berechnete Schnittlinienwert in *SplitDim*

	Schritt 3:	Bestimme die rechten und linken Unterflächen *Rleft* und *Rright*: *Rleft* = Teilfläche von *R* links oder oberhalb von *s*, mit Mittelpunktsvektor v[*SplitDim*] ≤ *SplitVal* *Rright* = Teilfläche von R rechts oder unterhalb von *s*, mit Mittelpunktsvektor v[*SplitDim*] > *SplitVal*
	Schritt 4:	*tleft* = linker Ast; Unterteile die Fläche ab Schritt 2 weiter rekursiv mit *Rleft* bis das festgelegte Abbruchkriterium erreicht ist
	Schritt 5:	*tright* = rechter Ast; Unterteile die Fläche ab Schritt 2 weiter rekursiv mit *Rright* bis das festgelegte Abbruchkriterium erreicht ist
	Schritt 6:	return *t*

Tab. 7: Schematischer Unterteilungsalgorithmus und Aufbau eines Slicing Trees.

Wie bereits erwähnt, zeichnen sich Unterteilungsalgorithmen dadurch aus, dass die Elemente durch die Unterteilung einer vorgegebenen Grundfläche automatisch dicht gepackt sind. Zusätzlich verfügen sie über Repräsentationsformen, die einfach zu verarbeiten sind, und damit beispielsweise die Entwicklung von Optimierungsstrategien erleichtern.

7.1.1. Darstellungsformen

Unterteilungsalgorithmen bzw. ihre Regeln lassen sich auf verschiedene Arten darstellen. Neben der konkreten grafischen Darstellung als sogenannter *Slicing Floorplan* wird die Unterteilungsfolge häufig in Form einer binären Baumstruktur, dem sogenannten *Slicing Tree*, dargestellt (Lai & Wong, 2001). Darüber hinaus lässt sie sich als Syntax in Präfixnotation beschreiben. Grundsätzlich lässt sich jede Repräsentationsform aus der Interpretation einer der anderen Darstellungsformen erstellen.

7.1.1.1. Slicing Floorplan

Die dem Unterteilungsalgorithmus entsprechende geometrische Darstellungsform ist der konkrete Unterteilungsgrundriss, der *Slicing Floorplan*. Er wird durch die rekursive Unterteilung einer vorgegebenen, rechteckigen Fläche in kleinere Rechtecksflächen erstellt (Abb. 43). Die Unterteilung erfolgt durch eine Abfolge von horizontalen und vertikalen Schnitten von Kante zu Kante (Valenzuela & Wang, 2002), die wie bereits beschrieben nach bestimmten Regeln oder zufällig erfolgen kann.

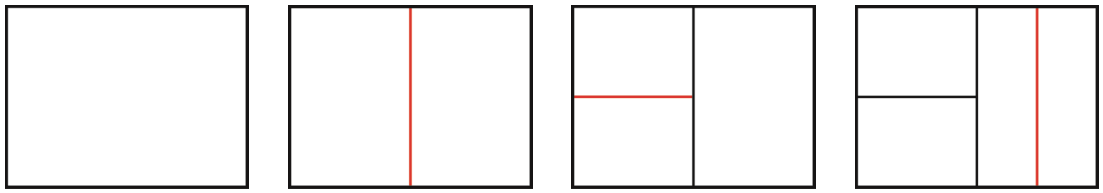


Abb. 43: Erstellung eines Slicing Floorplans.

7.1.1.2. Slicing Tree

Der *Slicing Tree* ist eine schematische Darstellungsform des Unterteilungsalgorithmus, die einen Binärbaum darstellt. Operatoren werden im *Slicing Tree* durch innere Knoten repräsentiert, Terminals bilden die externen Knoten oder Blätter.

Ein *Slicing Tree* kann beispielsweise aus der Unterteilungsfolge im Layout oder der Interpretation der Syntax von links nach rechts erstellt werden. Die Interpretation erfolgt folgendermaßen: Für eine Unterteilung bzw. einen Operator wird ein interner Knoten gesetzt. Der erste Knoten des Baums wird in der Regel als Wurzel (*root node*) bezeichnet. Die aus der Unterteilung entstandenen Unterrechtecke bilden den linken und rechten Ast des Knotens bzw. die einem Operator folgenden, von einem Klammerpaar umschlossenen Unterebenen.

Mit jedem entstandenen Unterrechteck bzw. mit jeder aufgehenden Klammer wird der Baum um eine Ebene erweitert. Die Interpretation springt eine Ebene nach unten. Es folgt entweder ein weiterer interner Knoten mit einem Operator, der zu einer weiteren Unterteilung der Fläche und einer weiteren Verästelung führt, oder ein Terminal, welches den Endpunkt der aktuellen Verästelung darstellt. Mit jeder schließenden Klammer bzw. jedem Raum springt die Interpretation eine Ebene im Baum zurück nach oben und erreicht mit der letzten Klammer bzw. Unterteilung wieder ihren Ausgangspunkt, die Wurzel. Die Baumrepräsentation einer Syntax ist in Abb. 44 (rechts) zu sehen.

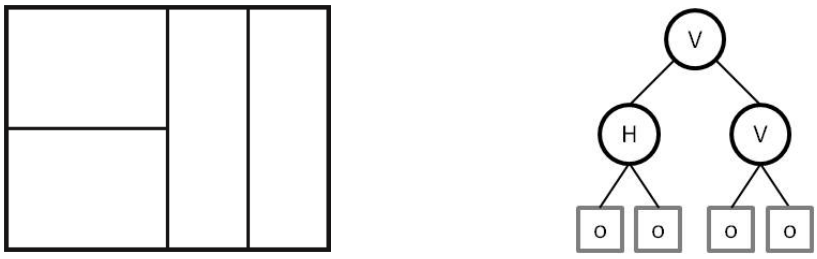


Abb. 44: Interpretation der Unterteilungssyntax V(H(o)(o))(V(o)(o)) bzw. des Slicing Floorplans (links) als Slicing Tree (rechts).

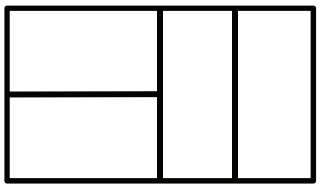
7.1.1.3. Unterteilungssyntax

Die Unterteilungssyntax ist eine Zeichenkette, die die Abfolge von Unterteilungen in horizontaler oder vertikaler Richtung sowie die Lage der Räume in einem Layout beschreibt. Die Zeichenkette besteht aus Operatoren, strukturierenden Zeichen und Operanden, den sogenannten Terminals.

Ein Operator steht stellvertretend für eine Unterteilung. Unterteilungen können im zweidimensionalen Raum in horizontaler oder vertikaler Richtung ausgeführt werden, die Operatoren werden entsprechend mit ‚H' und ‚V' bezeichnet. Ein Terminal stellt einen Endpunkt der Unterteilungsabfolge dar und entspricht im Layout einem Raum. Terminals werden in der Syntax mit ‚o' bezeichnet.

Die Syntax wird nach den Grundregeln der Präfixnotation gebildet, d.h., die Operatoren stehen immer vor den Operanden (Hamblin, 1962). Zur besseren Lesbarkeit der Syntax werden darüber hinaus die strukturierenden Zeichen ‚(' und ‚)' eingesetzt. Ein Klammerpaar umschließt einen Zweig in der Unterteilungshierarchie, d.h. eine Unterteilungsebene. Zur Verarbeitung der Zeichenkette sind die Klammern wegen der Eigenschaften der Präfixnotation jedoch nicht nötig.

Da sich unsere Untersuchungen zunächst auf binäre Datenstrukturen beschränken, gehen von jedem Operator zwei Unterebenen ab, die der linken bzw. oberen und der rechten bzw. unteren Teilungsfläche entsprechen. Abb. 45 zeigt eine Unterteilungssyntax (unten) und ihre grafische Darstellung als Layoutstruktur (oben).



V(H(o)(o))(V(o)(o))

Abb. 45: Grafische Interpretation und Unterteilungssyntax.

7.1.2. Unterteilungsalgorithmen im Architekturentwurf

Die Methode des Unterteilens wird häufig beim Entwurf von Grundrissen und anderen Layouts eingesetzt. Eine vorgegebene Fläche, beispielsweise ein Baugrundstück, wird dabei durch horizontale und vertikale Linien unterteilt. Durch diese Methode wird die zur Verfügung stehende Fläche vollständig genutzt.

Unterteilungen ermöglichen die hierarchische Gliederung eines Layouts in Zonen und Räume. Die so entstandene Grundrisstopologie kann als Baumstruktur dargestellt werden, die einer *Slicing Tree* Struktur ähnelt. In der hierarchischen Ordnung der Baumstruktur lassen sich beispielsweise die Abhängigkeiten der Zonen und Räume innerhalb des Layouts abbilden und ablesen. Funktionale Einheiten repräsentieren in diesem Fall innere Knoten und können eine Gesamtfläche beispielsweise in private oder öffentliche Zonen unterteilen (Marson & Musse, 2010) (Abb. 46, links). Die Blätter oder externen Knoten des *Slicing Tree* stellen die eigentlichen Räume des Grundrisses dar und entsprechen den in der Struktur geschaffenen Rechtecken (Abb. 46, rechts).

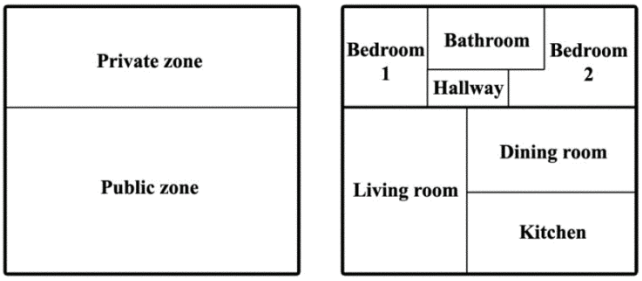


Abb. 46: Hierarchische Ordnung in Grundrissen: Zonierung (links), Raumaufteilung (rechts). Abbilding aus (Lopes, Tutenel, Smelik, de Kraker, & Bidarra, 2010).

Die Codierung des Grundrisses als Baumstruktur bzw. als Unterteilungssyntax vereinfacht dessen Verarbeitung und bildet folglich die Grundlage für die automatische Generierung von Layouts. Dabei muss beachtet werden, dass sich der Lösungsraum aller möglichen Grundrisse durch die Generierung mithilfe von Unterteilungsalgorithmen einschränkt, da sich viele existierende Grundrissformen nicht durch Unterteilung erstellen lassen. Hierzu gehören zum Beispiel verwinkelte Räume wie L-Formen (Abb. 47, rechts), ringartige oder netzartige räumliche Verbindungen und offene Grundrisse.

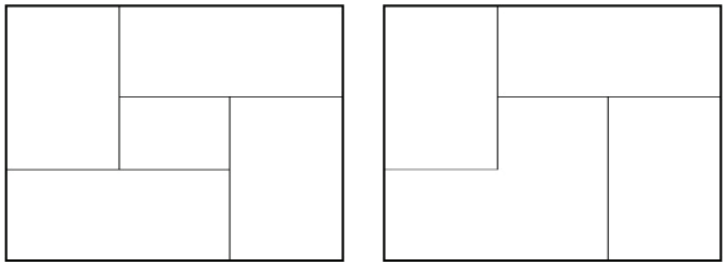


Abb. 47: Layoutbeispiele, die nicht durch Unterteilung erstellt werden können.

7.2. Generierung von Grundrisslayouts durch Unterteilung

Da sich Unterteilungsalgorithmen, wie oben beschrieben, an gängigen Entwurfsmethoden anlehnen, wurden im Rahmen des Forschungsprojekts Kremlas ihre geometrischen und strukturellen Eigenschaften genutzt, um sie zur Erzeugung von Grundrisslayouts einzusetzen.

7.2.1. Allgemeine Datenstruktur

Die binäre Baumstruktur des *Slicing Trees* bildet das zentrale Element der Datenstruktur. Die Unterteilungssyntax bildet den Ausgangspunkt für den Aufbau dieser Datenstruktur. Der Aufbau der Baumstruktur kann auf Basis der Generierung der Syntax erfolgen. Der Algorithmus hierzu wurde, wie in Tab. 7 beschrieben, umgesetzt. Im Baum ist die Unterteilungsfolge als Verbindungen zwischen Knoten und Blättern abgelegt. Die Eigenschaften der Räume des späteren Grundrisses sind den Blättern zugeordnet und werden über die Raumindizes referenziert.

7.2.2. Generierung von Unterteilungssyntax und Slicing Tree

Im Hinblick auf die Zielstellung, die Generierung von Grundrisslayouts, besteht der erste Schritt in der Generierung einer zufälligen Unterteilungsfolge, die gleichzeitig als Basis für den Aufbau der Datenstruktur dient.

Bei der Bildung der Syntax lassen sich zwei aus der genetischen Programmierung entlehnte Strategien anwenden, die anhand der Baumrepräsentation verdeutlicht werden können: Die Initialisierung als *full tree* bzw. als *grow tree*. Im ersten Fall werden alle Knoten des Baums bis zu einer festgelegten maximalen Tiefe gefüllt (Abb. 48, links). Im zweiten Fall werden die Knoten zufällig bis zur maximalen Tiefe mit Operatoren oder Terminals belegt (Abb. 48, rechts).

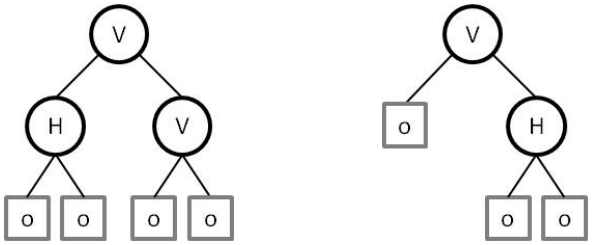


Abb. 48: Voll initialisierte Baumstruktur (links) und gewachsene Baumstruktur (rechts).

Bei der Entwicklung des generativen Mechanismus wurde berücksichtigt, dass Entwurfsaufgaben in der Regel über ein Raumprogramm verfügen, d.h. die Anzahl der im Grundriss zu erstellenden Räume weitestgehend vorgegeben ist. Der Algorithmus muss folglich Unterteilungsfolgen erstellen, die eine Fläche so unterteilen, dass jeweils eine vorgegebene Anzahl an Räumen entsteht.

Die Bildung der Syntax erfolgt deshalb auf Basis eines *grow trees* durch ein reglementiertes Wachstum. Das bedeutet, dass die zufällige Auswahl von Operatoren und Terminals beim Belegen der Knoten durch den Einsatz eines Raumzählers beschränkt wird, sodass nicht mehr oder weniger als die gewünschte Anzahl an Räumen in der Syntax beschrieben wird.

Darüber hinaus werden einige Baumeigenschaften im Voraus festgelegt bzw. begrenzt. Die maximal zulässige Baumtiefe wurde zunächst als Minimaltiefe + 1 definiert, um gleichmäßige Bäume zu generieren. Die Ausweitung der zulässigen Baumtiefe auf die tatsächliche Maximaltiefe ist möglich und würde die Anzahl an möglichen Lösungen zusätzlich erhöhen. Die Auswahl eines der Operatoren H und V oder des Terminals o in den Knoten erfolgt zunächst mit gleicher Wahrscheinlichkeit. Grundsätzlich ist eine andere Verteilung der Wahrscheinlichkeiten, d.h. zufällige Auswahl der Operatoren, denkbar.

Die Datenstruktur wird entweder gleichzeitig mit der Generierung der Syntax oder durch deren Interpretation erstellt. Das Layout wird wiederum aus der grafischen Interpretation des Baums gewonnen. Abb. 49 zeigt einen ersten, einfachen Prototyp, in dem eine Syntax zufällig generiert und anschließend grafisch interpretiert wird. Die Anzahl der zu generierenden Räume kann vom Nutzer eingegeben werden.

7.3. Optimierung von Grundrisslayouts

Die durch den im vorangegangenen Kapitel beschriebenen generativen Mechanismus erstellten Layouts werden zunächst zufällig generiert. Ein nächster Schritt besteht darin, die erzeugten Grundrisse hinsichtlich architektonisch relevanter Kriterien zu evaluieren und zu optimieren. Hierzu wird im Folgenden die Optimierung

von Layouts in Hinblick auf bestimmte Raumgrößen sowie die Erzeugung bestimmter Nachbarschaftsbeziehungen zwischen Räumen untersucht.

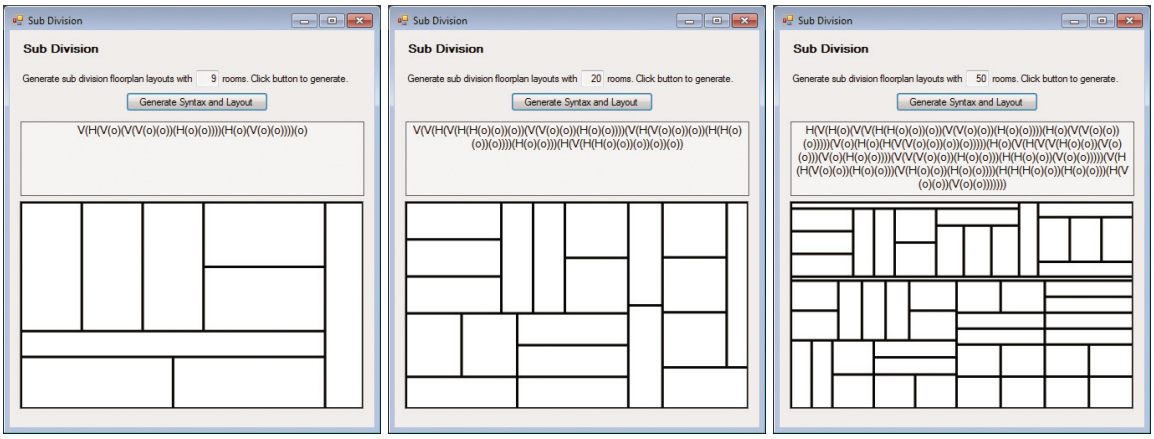


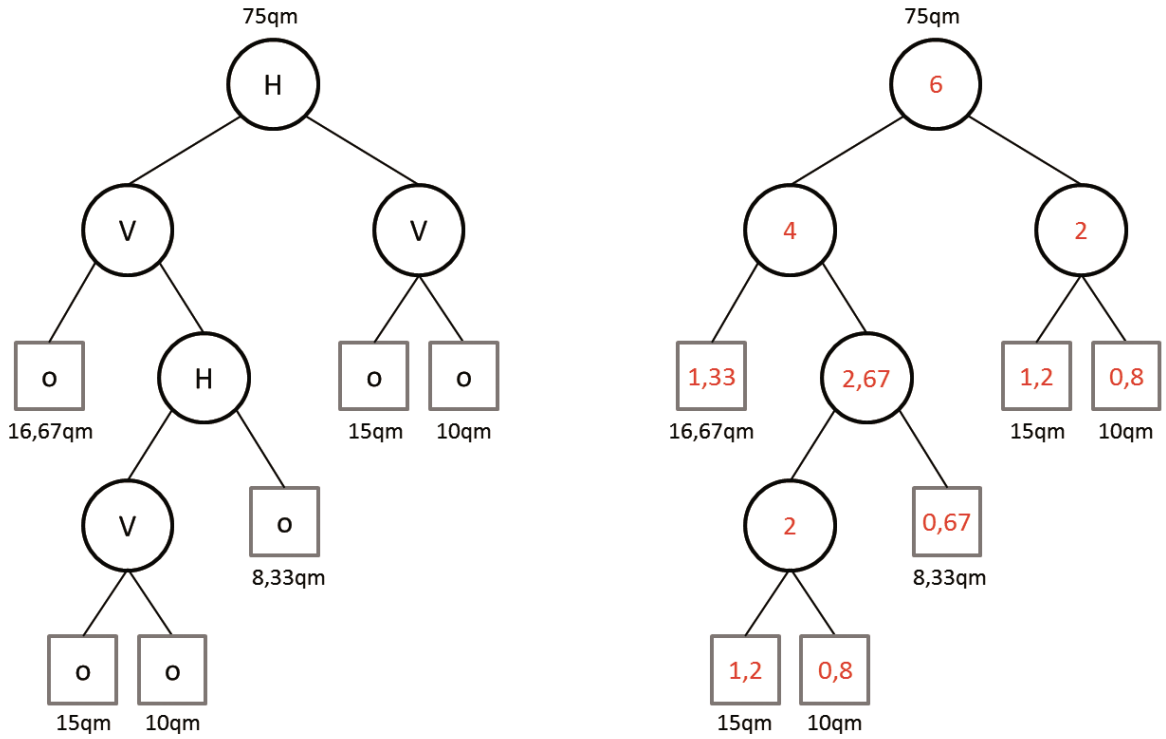
Abb. 49: Prototyp zur zufälligen Generierung einer Unterteilungssyntax mit beispielsweise 9, 20 oder 50 Räumen (von links nach rechts).

7.3.1. Berechnung bestimmter Raumgrössen

Die Erstellung von Layouts mit Räumen vorgegebener Größe wurde in den in Kapitel 5 und 6 vorgestellten Methoden mithilfe von Optimierungsalgorithmen gelöst. Im Fall des Unterteilungsalgorithmus lassen sich die Unterteilungswerte bzw. das Teilungsverhältnis, basierend auf den vorgegebenen Raumgrößen und einer bekannten Unterteilungsfolge, direkt berechnen. Eine Optimierung der Lage der Unterteilungen ist folglich überflüssig, da nach Ausführung aller Unterteilungen auf Basis der Unterteilungswerte jeder entstandene Raum die vorgegebene, gewünschte Größe besitzt.

Die Berechnung der Teilungswerte, der sogenannten *split values*, erfolgt von den Blättern ausgehend. Zunächst wird das Gewicht eines Raums bzw. Blatts aus dem Verhältnis seiner gewünschten Größe zum Flächenmittel berechnet. Das Flächenmittel ergibt sich aus der Teilung der zur Verfügung stehenden Gesamtfläche durch die Anzahl der Räume. Bei einer Gesamtfläche von 75qm beträgt das Flächenmittel bei sechs Räumen beispielsweise 12.5qm. Ein Raum mit 15qm besitzt in diesem Fall einen Wichtungswert von 1.2, ein Raum mit 10qm einen Wert von 0.8. Der Wichtungswert eines Knotens berechnet sich aus der Summe der Wichtungswerte seiner beiden Äste, d.h. ein den beiden genannten Räumen zugeordneter Knoten hätte einen Wichtungswert von 2. So werden von den Blättern zur Wurzel die Wich-

tungswerte aller Räume und Knoten bestimmt (Abb. 50). Die Wichtung des Wurzelknotens aus der Summe der beiden Hauptäste muss gleich der Anzahl der Räume sein.



Syntax: H(V(o)(H(V(o)(o))(o)))(V(o)(o))

Abb. 50: Slicing Tree: 6 Räume, mit Angabe der Raumgrößen (links) und Berechnung der Wichtungswerte in den Blättern und Knoten (rechts).

Die Lage der Unterteilung sowie die den Unterästen zugeordnete Fläche wird anschließend, beginnend vom *root node*, für jeden Knoten aus dem Verhältnis der Wichtungswerte seiner zwei Unteräste bezogen auf die Knotenfläche berechnet (Abb. 51). Der im vorigen Beispiel erwähnte Knoten besitzt beispielsweise ein Teilungsverhältnis von 1.2 zu 0.8, d.h. von 3 zu 2. Der zugehörige Teilungswert lässt sich aus der Unterteilung der Breite oder Höhe der zugehörigen Knotenfläche im berechneten Teilungsverhältnis bestimmen. Bei einer Unterteilung in horizontaler Richtung wird hierzu die Höhe der Knotenfläche herangezogen, man erhält den y-Wert der horizontalen Schnittlinie. Bei einer Unterteilung in vertikaler Richtung wird das Verhältnis der Unteräste auf die Breite der Knotenfläche bezogen, um den x-Wert der vertikalen Schnittlinie zu bestimmen.

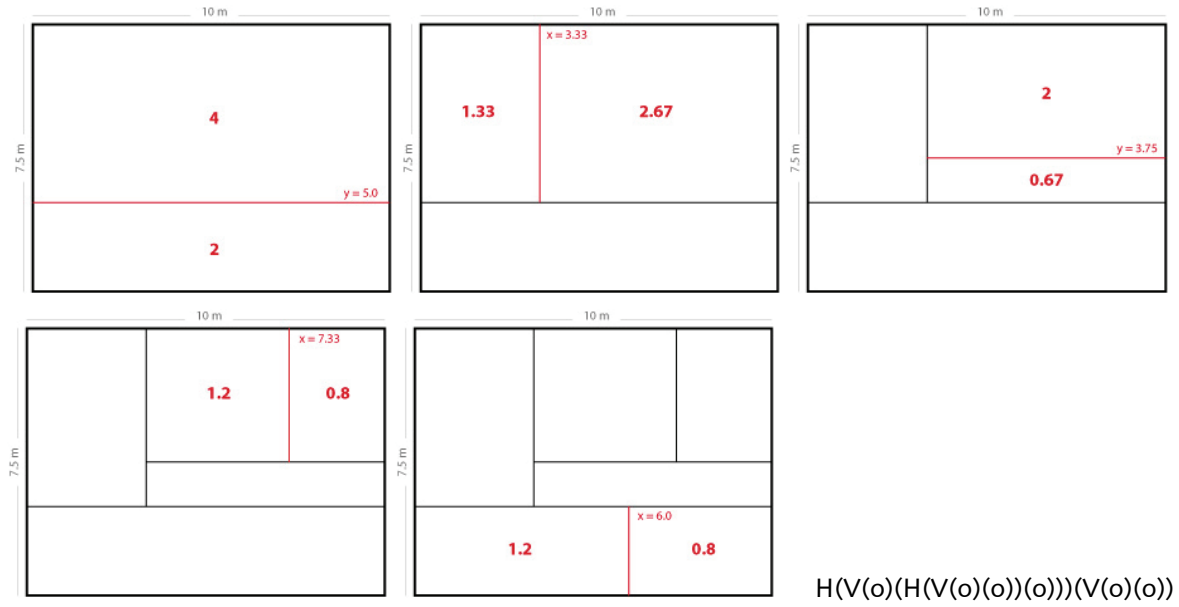


Abb. 51: Abfolge und Bestimmung der Lage der Unterteilungen im Layout.

7.3.2. Suche nach bestimmten Nachbarschaftsverhältnissen

Bei der Suche nach bestimmten Nachbarschaftsverhältnissen handelt sich um ein topologisches Problem. Es besteht darin, den Räumen die Funktionen so zuzuordnen bzw. die Schnittdimensionen in den Knoten so zu legen, dass nach der Unterteilung die gewünschten Nachbarschaften zwischen Räumen bestehen. Im Unterschied zur Problematik der Erstellung von Räumen bestimmter Raumgrößen lassen sich bestimmte Nachbarschaftsverhältnisse nicht direkt berechnen.

Optimale Lösungen werden mithilfe von Optimierungsstrategien gesucht. Im konkreten Fall wurde eine $(\mu + \lambda)$-ES in Kombination mit GA und GP implementiert. Der schematische Ablauf des evolutionären Prozesses ist in Tab. 2 dargestellt. GA und GP bilden dabei die Basis für die Rekombination und Mutation der Elternindividuen in Schritt 4 und 5 des EA.

Der GA wird eingesetzt, um die Zuordnung der Funktionen und Indizes zu den Räumen hinsichtlich der gesuchten Nachbarschaftsverhältnisse zu optimieren. Hierbei werden zunächst die Indizes der Räume in der Abfolge ihres Entstehens bei der Unterteilung als Indexsequenz kodiert (Abb. 52). Im Rahmen der Optimierung wird diese Indexsequenz zur Erzeugung neuer Lösungen mutiert und rekombiniert. Die Mutation erfolgt durch das Vertauschen von zwei Indizes innerhalb der Sequenz und entspricht dem Vertauschen der Indizes zweier Räume (Abb. 53). Bei der Re-

kombination entstehen neue Lösungen aus der Kreuzung der Indexsequenzen zweier Lösungen durch One-Point-Crossover (Abb. 54).

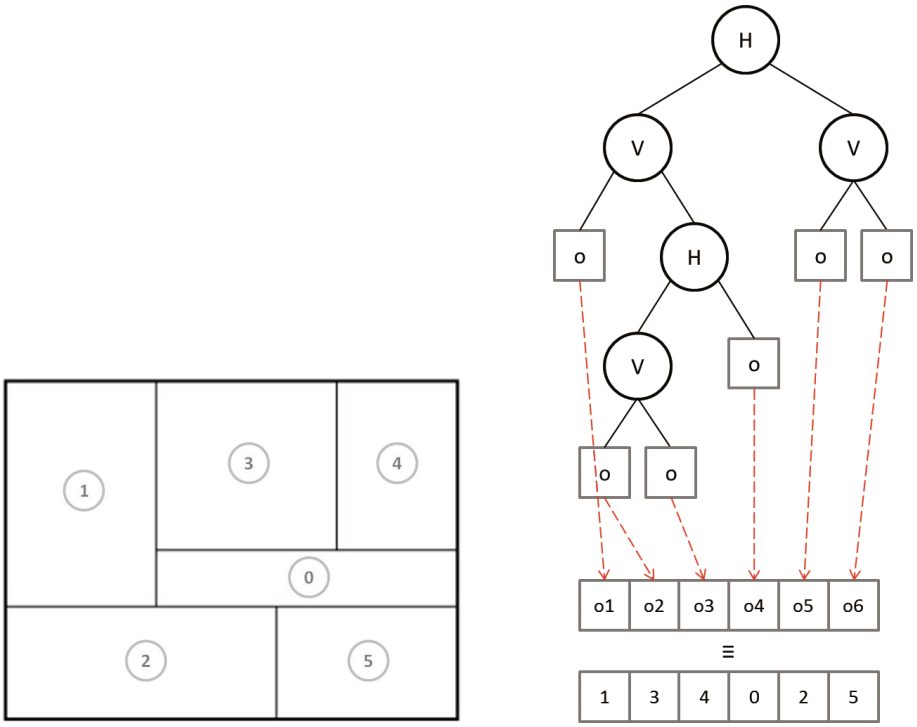


Abb. 52: Kodierung der Raumindizes aus der Entstehungsabfolge, Phänotyp (links), Genotyp (rechts).

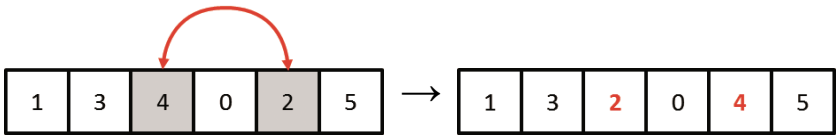


Abb. 53: Genetischer Algorithmus: Vertauschen von Indizes.

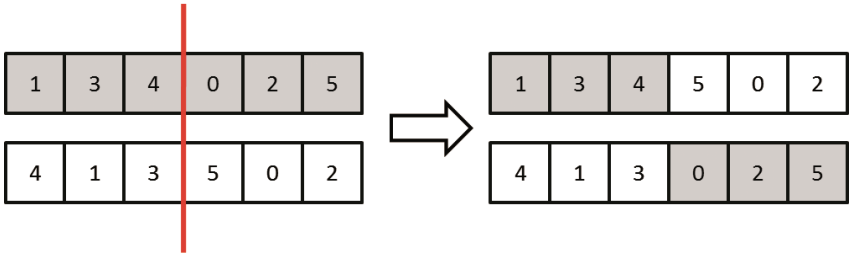
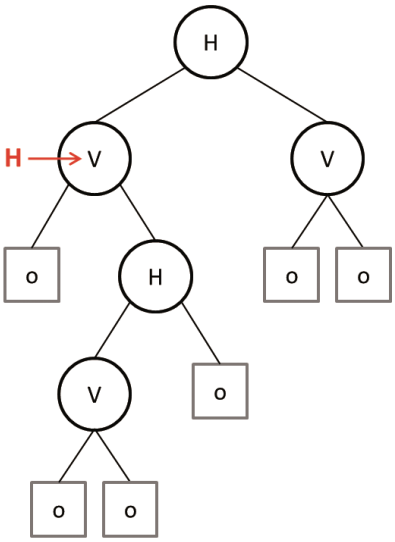


Abb. 54: Genetischer Algorithmus: One-Point-Crossover zweier Genome.

Die GP dient zur Optimierung der Struktur, d.h. der Unterteilungsabfolge, des *Slicing Trees.* Neue Lösungen werden entweder durch zufällige Mutation eines Elternindividuums, d.h. dem Umschalten einer Unterteilung in einem Knoten von horizontal nach vertikal oder umgekehrt (Abb. 55), oder durch Crossover generiert.



H(H(o)(H(V(o)(o))(o)))(V(o)(o))

Abb. 55: Genetische Programmierung: Mutation eines Individuums durch Änderung der Unterteilungsrichtung.

Beim Crossover werden Äste zwischen den Bäumen zweier Elternindividuen ausgetauscht. Die beiden Elternindividuen werden zunächst durch Binary Tournament Selection ausgewählt. Anschließend werden in den *Slicing Trees* Äste zum Austausch bestimmt, die jeweils die gleiche Anzahl an Blättern besitzen, damit die Gesamtzahl der Räume im Kindindividuum gleich der der Eltern bleibt (Abb. 56).

In der Evaluationsfunktion wird die Summe aller Abstände zwischen Räumen mit gewünschter Nachbarschaftsbeziehung berechnet. Sie lässt sich folgendermaßen beschreiben:

$$f = \sum_{i=1}^{n} \overline{A_l B_l} \tag{25}$$

wobei A und B die Räume darstellen, die benachbart sein sollen, und n die Anzahl der Nachbarschaften.

Der Abstand zwischen zwei direkt benachbarten Räumen beträgt null. Folglich ist der aus der Evaluierungsfunktion resultierende Fitnesswert umso niedriger, je mehr gewünschte Nachbarschaftsbeziehungen in einer Layoutlösung existieren oder je näher die entsprechenden Räume zueinander liegen. Sein Idealwert beträgt null.

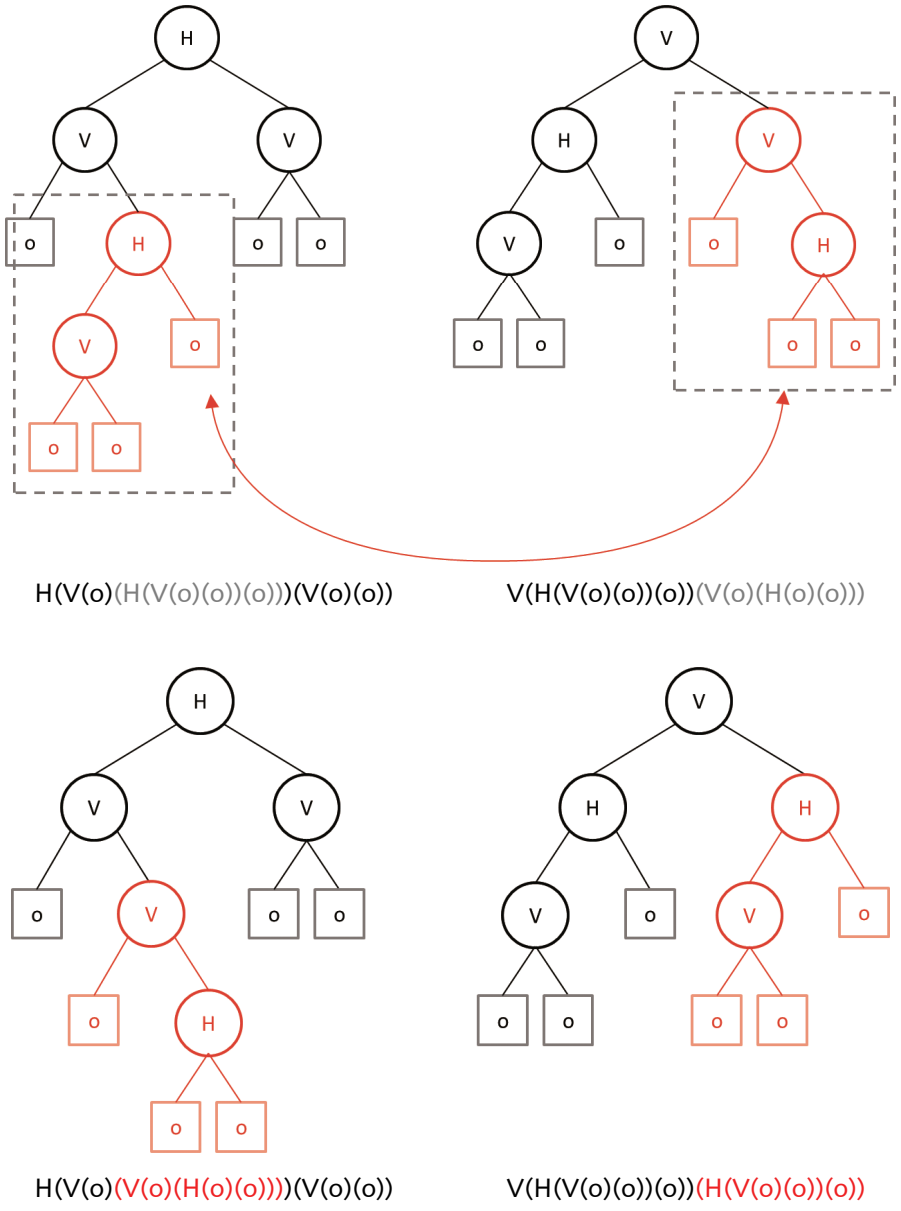


Abb. 56: Genetische Programmierung: Crossover von Ästen zwischen zwei Syntaxbäumen.

Um die Fitnesswerte des Unterteilungsalgorithmus mit anderen Methoden der Grundrissgenerierung besser vergleichen zu können (siehe hierzu Kapitel 9), wurde der Wertebereich der in (26) beschriebenen Evaluationsfunktion f auf das Intervall]0;1] normiert. Die normierte Evaluationsfunktion F stellt sich folgendermaßen dar:

$$F = \frac{1}{f+1} \tag{26}$$

Dies bedeutet, dass der Fitnesswert umso kleiner ist, je weniger gewünschte Nachbarschaften bestehen und je weiter die nicht benachbarten Räume auseinander liegen, und dass die Kennwerte topologisch optimaler Lösungen gegen 1 tendieren.

7.3.3. Prototypische Umsetzung

Der generative Mechanismus wurde in Kombination mit dem beschriebenen Evolutionären Algorithmus zur Suche nach bestimmten Nachbarschaftsverhältnissen prototypisch umgesetzt (Abb. 57). Als zu optimierende topologische Struktur wurde wie in den vorhergehend beschriebenen Methoden (siehe Kapitel 5.2 und 6.2.4) die im Grundrissentwurf relativ häufig vorkommende Sterntopologie gewählt. Eine Sterntopologie zeichnet sich dadurch aus, dass alle Räume des Layouts mit einem zentralen Raum verbunden und von diesem aus erreichbar sind.

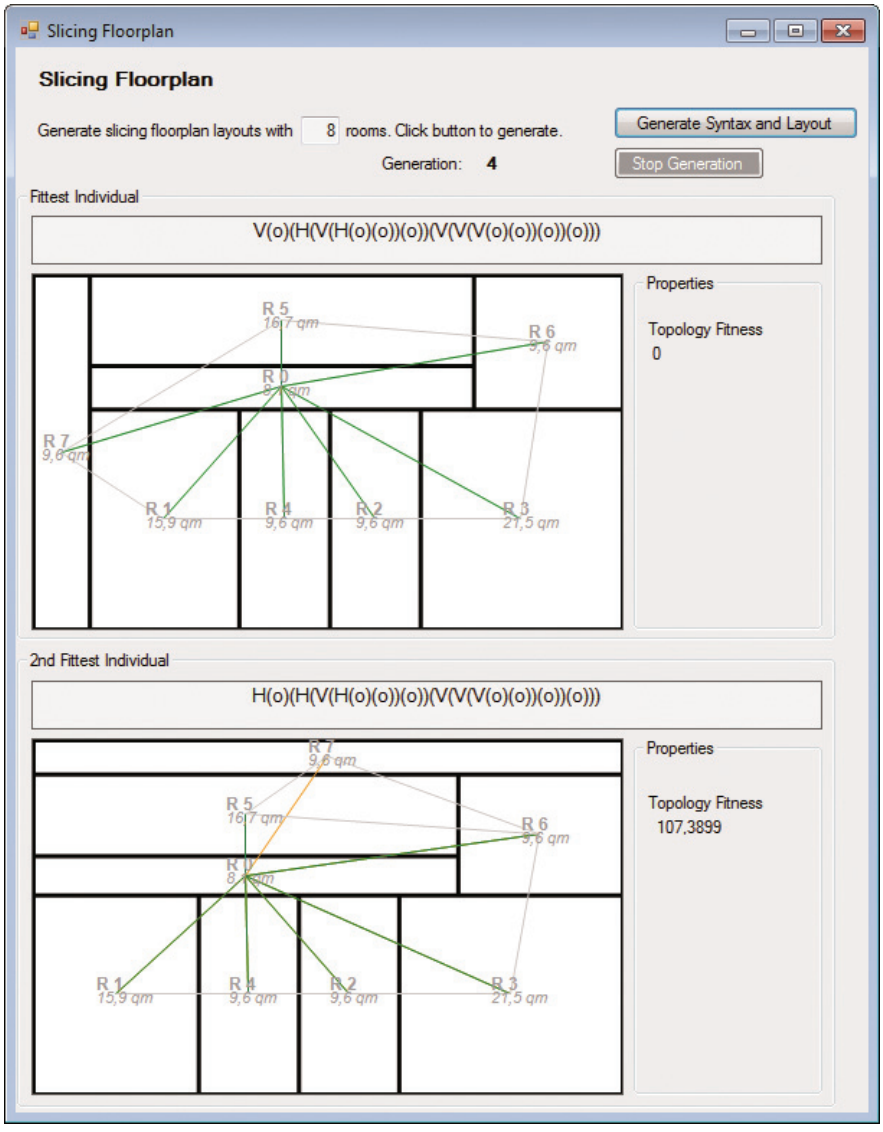


Abb. 57: Optimierung der Nachbarschaftsbeziehungen, Prototypische Umsetzung.

Die Umsetzung des Algorithmus erfolgte auf Basis des in Kapitel 4.4, Tab. 2 dargestellten Schemas. Die Individuen wurde mithilfe des in Kapitel 4.1 dargestellten GAs bzw. der GP rekombiniert und mutiert. Die Auswahl von GA oder GP zur Re-

kombination und Mutation erfolgte zufällig und mit gleicher Wahrscheinlichkeit. Bei der Berechnung des Fitnesswerts einer Lösung auf Basis des Abstands zwischen zwei Räumen wurde zusätzlich eine Mindestdurchgangsbreite berücksichtigt, um später das Platzieren von Verbindungen wie Türen zwischen den Räumen zu ermöglichen.

Der entwickelte Softwareprototyp liefert schnell topologisch optimierte Lösungen, insbesondere für eine niedrige Anzahl an Räumen. Aufgrund der gewählten Topologie weisen mit steigender Raumanzahl die mit dem aktuellen Prototyp generierten Layouts viele lange, schmale Räume auf, die jedoch aus architektonischer Sicht für viele Nutzungen ungeeignet sind.

7.4. Konklusion und Ausblick

Es wurde ein generativer Mechanismus vorgestellt, der es erlaubt, innerhalb einer vorgegebenen rechteckigen Grundstücksfläche *Slicing Floorplans* mit einer festgelegten Anzahl an Räumen mit bestimmten Raumgrößen zu erzeugen. Der eingesetzte Algorithmus imitiert eine häufig in der Architektur eingesetzte Entwurfsmethode, bei der eine vorgegebene Grundfläche zunächst in Zonen, Abschnitte und schließlich in Räume unterteilt wird. Als Grundlage für das Datenmodell dienten uns *Slicing Trees* und Unterteilungssyntax. Die Kombination des generativen Mechanismus mit EA (einer Kombination aus GA und GP) ermöglichte die Optimierung des Grundrisses im Hinblick auf gewünschte Nachbarschaftsbeziehungen zwischen bestimmten Räumen innerhalb des Layouts.

Es kann festgestellt werden, dass durch die Raumpartitionierung durch Unterteilungsalgorithmen in Kombination mit evolutionären Algorithmen interessante und architektonisch relevante Layoutlösungen entstehen. Die Unterteilungsmethode stellt folglich eine vielversprechende Variante zu bereits bekannten Strategien bei der kreativen algorithmischen Lösung von Layoutaufgaben in Architektur und Städtebau dar, die im Zuge weiterer Untersuchungen vertieft werden kann.

Zur Weiterbearbeitung der Thematik ist die Betrachtung von Mehrzieloptimierungsalgorithmen und eine damit verbundene die Optimierung der Grundrisse hinsichtlich weiterer Kriterien denkbar. Zu architektonisch relevanten Kriterien, die im Zuge

einer Mehrzieloptimierung betrachtet werden könnten, zählen Raumcharakteristiken wie Raumproportionen und weitere topologischen Kriterien wie die Raumausrichtung. Darüber hinaus ist die Integration von Analysen zu Sonnenstand und Sichtbarkeiten zu untersuchen.

8. Layouts mittels Voronoi-Diagramm[11]

Sven Schneider, Reinhard König

In diesem Kapitel wird eine auf Voronoi-Diagrammen basierende Methode zur Generierung von Layouts innerhalb geometrisch frei definierbarer Begrenzungen beschrieben. Dabei werden Layouts erzeugt, bei dem die Winkel in den Ecken möglichst an rechte Winkel angenähert werden. Dies ist relevant, da bei realen Layoutproblemen als Ausgangspunkt oft geometrische Unregelmäßigkeiten existieren (z.B. bei dem Umriss eines Grundrisses, dem Zuschnitt einer Parzelle oder einer bestehenden Wegestruktur), welche bei der Planung berücksichtigt werden müssen.

8.1. Rechtwinkligkeit in architektonischen Layouts

Rechtwinkligkeit ist eine häufig auftretende Eigenschaft in architektonischen Layouts. Welche Umstände ursprünglich zu dieser Präferenz rechter Winkel geführt haben, ist bis heute nicht vollständig geklärt (Bafna & Shah, 2007). Jedoch können zahlreiche Vorteile genannt werden, die die Verwendung rechter Winkel mit sich bringen. So erweist er sich in vielerlei Hinsicht als praktikabel, beispielsweise bei der Möblierung von Grundrissen (aufgrund rechteckiger Möbel) oder als Hilfsmittel zur Vereinfachung der Baukonstruktion (Einfluss der verfügbaren Materialien und Bauweisen). Für die Bewegung im Raum ist festgestellt worden, dass rechte Winkel die räumliche Orientierung erleichtern. So haben Sadalla und Montello (1989) durch Experimente herausgefunden, dass sich Menschen beim Zurücklegen von Wegen primär an Winkeländerungen, die einem Vielfachen von 90° entsprechen, erinnern. Außerdem ist es häufig sinnvoll, Räume mit konvexen Formen zu schaffen, da diese für den Nutzer optimal einsehbar sind und ein Gefühl von Sicherheit und Wohlbefinden erzeugen (Alexander, Ishikawa, & Silverstein, 1977). Sollen mehrere solcher (konvexer) Räume aneinandergefügt werden, so hilft die Verwendung gera-

[11] Teile dieses Kapitels beruhen auf dem Artikel von Schneider, S., Koenig, R. & Pohle, R. (2011). Who cares about right angles? Overcoming barriers in creating rectangularity in layout structures. Paper presented at the eCAADe 2011: Respecting Fragile Places, Ljubljana, Slovenia.

der Linien und rechter Winkel, den Einfluss der Form eines Raums auf die Form eines anderen möglichst gering zu halten (siehe Abb. 59, rechts).

8.2. Geometrische Unregelmässigkeiten in Layouts

Trotz des häufig auftretenden rechten Winkels in der Architektur findet sich eine ausschließliche Rechtwinkligkeit in Städten und Gebäuden eher selten. Oft existieren geometrische Unregelmäßigkeiten (stumpfe oder spitze Winkel) in einer zu bebauenden Parzelle. Sollen diese beim Entwurf einer räumlichen Konfiguration berücksichtigt werden, führt dies unweigerlich zu nicht vollständig rechtwinkligen Formen.

Bei den bisher entwickelten Systemen zur Generierung von Layouts findet man stets vereinfachte Annahmen hinsichtlich der Form der Räume (bzw. den innerhalb eines Raumes auftretenden Winkel). Beispielhaft seien die Arbeiten von Elezkurtaj (2004), Arvin und House (2002) sowie die Kapitel 5 vorgestellte Methode genannt. Die Arbeiten verwenden als Grundform Rechtecke, welche möglichst lückenlos innerhalb einer rechteckigen Begrenzung dicht gepackt werden. Ein solches Verfahren kann auch als additives Verfahren bezeichnet werden. Durch die getroffenen Einschränkungen hinsichtlich der Form von Räumen vereinfachen sich natürlich auch die Algorithmen, die zu Generierung und Bewertung eingesetzt werden. Dies verringert zum einen die Rechenzeit, die nötig ist, um zu einer Lösung zu gelangen[12]. Zum anderen vereinfacht die Verwendung von einfachen Grundformen und bekannten Operationen die Interaktion des Nutzers mit dem System (siehe Punkt 11.2).

Neben den additiven Verfahren existieren dividierende Verfahren, die durch Aufteilung einer gegebenen Fläche ein Layout erzeugen. Zu den Projekten, welche mit dividierenden Verfahren arbeiten, zählen beispielsweise die Arbeiten von Flemming (1977), Duarte (2001) sowie die in den Kapiteln 6 bis 7 vorgestellten Methoden. Hinsichtlich der erzeugten Formen lässt sich jedoch auch hier feststellen, dass sich bei allen genannten Projekten die Erkundung des Lösungsraumes auf rechtwinklige Layouts beschränkt. Doulgerakis (2007) stellt eine Methode vor, die es erlaubt,

[12] Zu bemerken ist außerdem, dass bei allen Projekten neben der reinen Form auch funktionale Kriterien wie die topologischen Beziehungen der Räume zueinander berücksichtigt werden.

nicht rechtwinklige Layouts zu erzeugen. Jedoch ist bei dieser festzustellen, dass durch die innerhalb der erzeugten Layouts auftretenden Winkel oft schlecht nutzbare bzw. unkomfortable Räume entstehen.

Um geometrische Unregelmäßigkeiten zu berücksichtigen, aber gleichzeitig möglichst rechtwinklige Layouts zu erhalten, ist ein differenzierter Umgang mit Winkeln im Generierungsprozess vonnöten. Dabei ist zum einen zu bemerken, dass die Form des Rechteckes keine Vorrausetzung für architektonische Räume ist. Zum anderen ist auch der perfekte rechte Winkel nicht zwingend erforderlich. So schreibt Alexander, dass es zwar Gründe dafür gibt dass Wände *„(…) annähernd gerade sein sollten; (...) Aber nichts spricht dafür, daß ihre Seiten vollkommen gleich sind oder ihre Ecken absolut rechtwinklig sind"* (Alexander, Ishikawa, Silverstein, & Czech, 2000, S. 956). Zu vermeiden sind nach Alexander spitze Winkel < 80 Grad und überstumpfe Winkel > 180 Grad (siehe auch Abb. 58), da diese das Wohlbefinden der sich darin aufhaltenden Menschen beeinträchtigen.

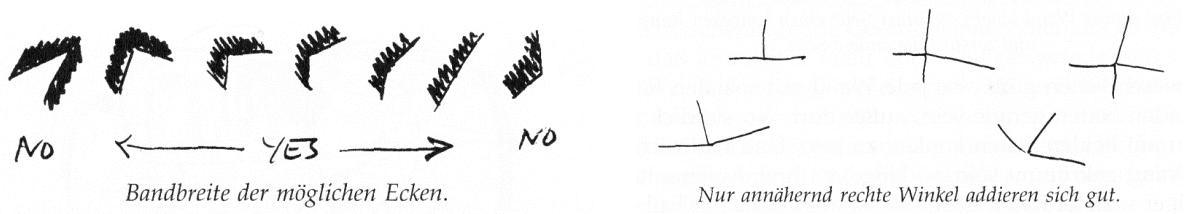


Abb. 58 Bandbreite der möglichen Winkel. Abbildung aus (Alexander, et al., 2000).

Im Folgenden wird dargestellt, wie Layouts hinsichtlich der Anforderung nach möglichst rechten Winkeln generiert werden können. Das Szenario, welches dazu verwendet wird, besteht in der Generierung eines Layouts innerhalb einer frei definierbaren Begrenzung. Frei definierbar bedeutet hierbei, dass die polygonale Begrenzung sowohl konvex als auch konkav geformt sein sowie beliebige Winkel aufweisen kann. Bei der Begrenzung kann es sich sowohl um einen Gebäudeumriss als auch um ein zu parzellierendes Grundstück handeln. Für die Generierung wird eine dividierende Vorgehensweise (Aufteilen einer Fläche) verwendet.

8.3. Algorithmus zur Generierung annähernd rechtwinkliger Layouts

Um annähernd rechtwinklige Layouts zu erzeugen, wird eine ES verwendet (siehe Kapitel 4). Entscheidend für eine erfolgreiche Anwendung dieser Strategie ist ein generativer Mechanismus, der eine möglichst große Zahl unterschiedlicher Lösungen erzeugen kann. Dieser Mechanismus spannt den Lösungsraum auf, innerhalb dessen nach optimalen Lösungen gesucht werden kann. Für den generativen Mechanismus wird im Folgenden die Aufteilung einer Fläche durch Voronoi-Zerlegung gewählt. Die Voronoi-Zerlegung eignet sich insofern zur Generierung von Layouts, da das resultierende Voronoi-Diagramm ein großes Spektrum möglicher Lösungen abdeckt. So lassen sich mittels Voronoi-Zerlegung neben Rechteckraster über Waben-Raster bis hin zu völlig freien Aufteilungen vielfältigste Strukturen erzeugen. Entscheidend für das Resultat der Zerlegung ist die Anordnung der Zentren: Ein Voronoi-Diagramm bezeichnet die Aufteilung einer Fläche in Regionen ausgehend von einer definierten Punktmenge, den Voronoi-Zentren. Dabei sind alle Punkte, die sich innerhalb einer Region befinden, näher am Zentrum ihrer Region als zu allen anderen Zentren. An den Kanten oder Grenzen der Regionen ist der Abstand zwischen zwei Zentren identisch. Zur Erzeugung von Voronoi-Diagrammen existieren zahlreiche Konstruktionsmethoden, auf die im vorliegenden Artikel nicht näher eingegangen werden soll[13].

Als Evaluationskriterium für die ES dient die Summe der Abweichungen vom rechten Winkel bei den innerhalb einer Lösung auftretenden Winkel. Das Voronoi-Diagramm wird dahingehend untersucht, welche Kantenpunkte sich innerhalb des umgebenden Polygons befinden bzw. welche Schnittpunkte sich zwischen Kanten und dem umgebenden Polygon ergeben. Für die innenliegenden Punkte der Voronoi-Kanten werden jeweils die drei zusammenhängenden Winkel berechnet. Bei Voronoi-Diagrammen treffen meistens drei Kanten in einem Punkt aufeinander[14].

[13] Eine Übersicht zu verschiedenen Konstruktionsprinzipien findet sich bspw. in Goodman & O'Rourke Goodman, J. E., & O'Rourke, J. (2004). *Handbook of discrete and computational geometry*. Chapman & Hall/CRC.

[14] Der Fall, dass vier Kanten einen gemeinsamen Schnittpunkt haben, kann auftreten, ist aber sehr unwahrscheinlich, und wird bei der vorgestellten Methode vernachlässigt.

Für die Schnittpunkte der Voronoi-Kanten mit der Begrenzung wird jeweils ein Schnittwinkel berechnet (siehe Abb. 59).

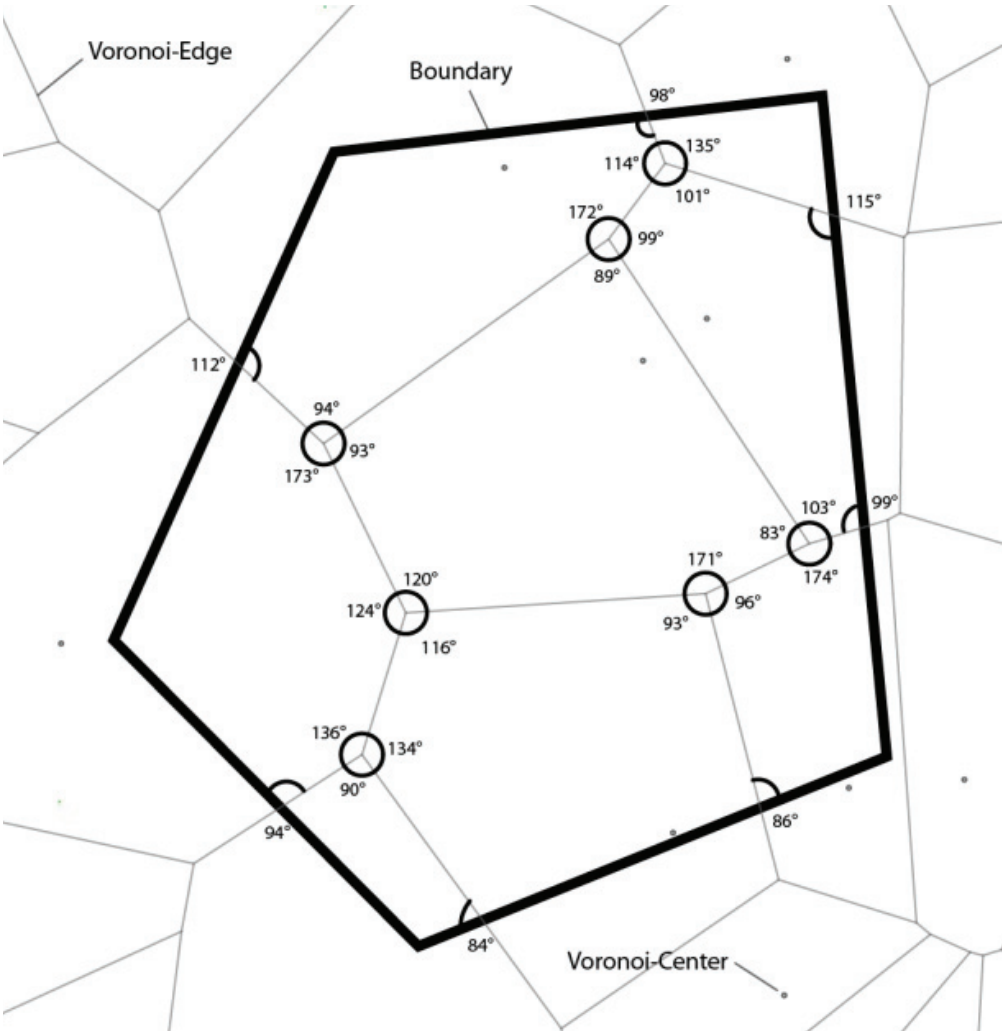


Abb. 59: Auftretende Winkel der Struktur mit Bezug zur Begrenzung.

Die Fitnessfunktion beschreibt die Summe der Differenz der einzelnen Winkel zum 90° Winkel, was einer Ecke im Raum entspricht, bzw. zum 180° Winkel, was einer geraden Wand entspricht. Alle Winkel, die unter 135° liegen, werden hinsichtlich ihrer Abweichung von 90°, alle Winkel, die zwischen 135° und 180° liegen, hinsichtlich ihrer Abweichung von 180° beurteilt. Winkel über 180° können innerhalb eines Voronoi-Diagramms nicht auftauchen und müssen daher nicht berücksichtigt werden. Da jedes Individuum (Layout) eine unterschiedliche Anzahl an auftretenden Winkel besitzen kann, ist eine Normierung der Winkelsummen notwendig. Die Fitnessfunktion für k auftretende Winkel lautet daher folgendermaßen:

$$f = \frac{\sum_{i=0}^{k} \begin{cases} |\alpha_i - 90| & if\ 0 < \alpha_i < 135 \\ |\alpha_i - 180| & else \end{cases}}{k} \tag{27}$$

Die Funktionsweise der zur Optimierung der auftretenden Winkel implementierten ES ist analog zu der in Kapitel 4, Tab. 2 beschriebenen. Zuerst wird eine Population P mit μ Individuen initialisiert. Ein Individuum besteht aus n anfangs zufällig platzierten Punkten, aus welchen das Voronoi-Diagramm generiert wird. Die μ Individuen werden nach (27) bewertet. Danach werden sie für die nächste Generation selektiert, rekombiniert und mutiert. Bei der Rekombination werden zufällig Punkte zwischen verschiedenen Individuen vertauscht. Die Mutation variiert die Position zufällig ausgewählter Punkte eines Individuums geringfügig. Dieser Prozess wiederholt sich so lange bis der Fitnesswert minimal ist.

8.4. Testscenarien

Der oben beschriebene Algorithmus wurde für verschiedene Begrenzungsformen getestet (Abb. 60). Festzustellen ist, dass in keinem der getesteten Fälle die Fitnessfunktion den Minimalwert 0 erreichen konnte. Das bedeutet, dass die ES in lokalen Optima verharrt, was sich auf die Verwendung der Voronoi-Zerlegung als generative Methode zurückführen lässt. So haben Änderungen der Position eines Voronoi-Zentrums direkten Einfluss auf die Form der Voronoi-Regionen anderer Zentren und damit auch auf die in diesen Voronoi-Regionen auftretenden Winkel. Diese Interaktion zwischen den Regionen erschwert das Finden eines globalen Optimums.

Des Weiteren kann anhand der Ergebnisse festgestellt werden, dass bestimmte Typen von Lösungen gehäuft auftreten. Dazu zählt in erster Linie das Entstehen von Linienzügen. Einige dieser Linienzüge entstehen relativ zentral innerhalb der Begrenzung und bilden in ihrer Krümmung die Form der Begrenzung in abstrahierter Form nach (Abb. 60: 1b, 2b, 3a).

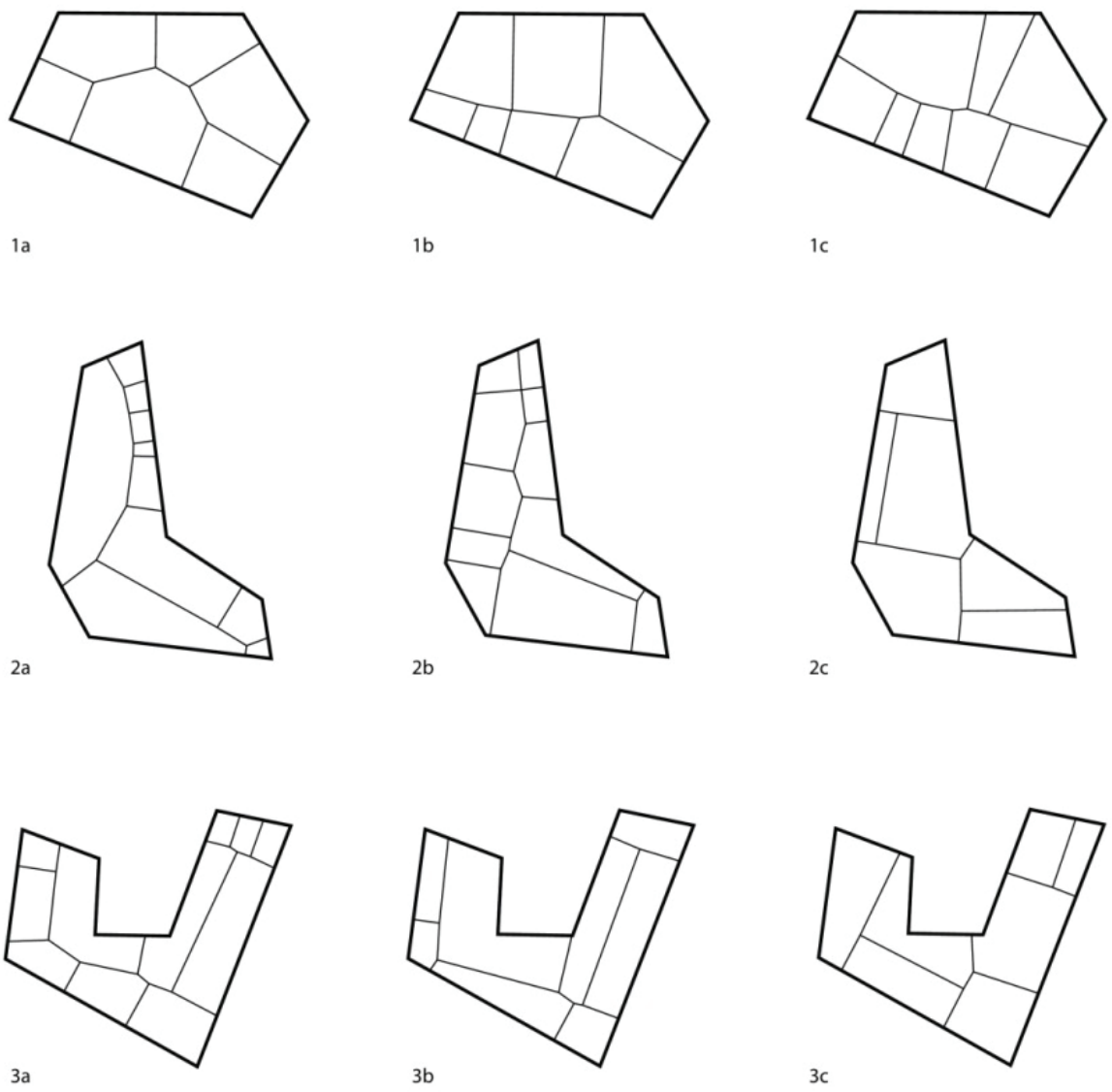


Abb. 60 Lösungsvarianten für jeweils drei verschiedene Begrenzungsformen.

Anhand der generierten Lösungsvarianten lassen sich problematische Stellen erkennen. Sie sind exemplarisch in Abb. 61 dargestellt. Abb. 61a zeigt einen konkaven Raum. Zwar enthalten Voronoi-Diagramme laut Definition ausschließlich konvexe Regionen, jedoch werden bei der hier der vorgeschlagenen Methode diese konvexen Regionen mit einer frei definierbaren Begrenzung geschnitten. Ist diese Begrenzung konkav, können an den Stellen, an denen der Innenwinkel der Begrenzung mehr als 180° beträgt, konkave Räume entstehen. Eine weitere problematische Stelle ergibt sich, wenn die Voronoi-Kanten in die Nähe der Eckkanten eine Begrenzungslinie schneiden (Abb. 61b). Bei dieser Situation besteht das Problem darin, dass der auftretende Winkel nur an einer der Begrenzungskanten optimal ist. Die zweite Kante wird außer Acht gelassen, obwohl sie sich in geringem Abstand

zur Voronoi-Kante befindet. Schließlich ist das Entstehen von Zick-Zack-Linien zu beobachten (Abb. 61c), welche zustande kommen können, da sich die Formen der Voronoi-Regionen, wie bereits beschrieben, durch das Verschieben der Voronoi-Zentren gegenseitig beeinflussen.

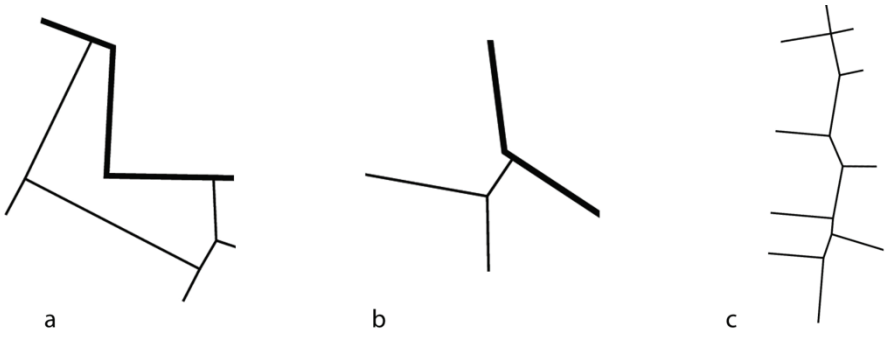


Abb. 61 Problematische Stellen der generierten Strukturen: (a) konkave Regionen (b) Kanten nah an Ecken (c) Zick-Zack-Linien.

Um die genannten Problemstellen bei der Generierung zu umgehen, bieten sich zwei Möglichkeiten an. Erstens ließe sich der Evaluationsmechanismus um weitere Kriterien, z.B. die Überprüfung der Konvexität der entstehenden Regionen, erweitern. Dies führt jedoch einerseits zur Verlangsamung des Optimierungsprozess, da letztlich für jedes Individuum aufwendige geometrische Analysen durchgeführt werden müssen. Andererseits besteht nach wie vor das Problem, globale Optima zu finden, da es sich bei den Evaluationskriterien auch um sich gegenseitig widersprüchliche handeln kann (siehe 5.2.3). Die zweite Möglichkeit zur Vermeidung der genannten Problemstellen ist eine Kombination der iterativ arbeitenden ES mit einem Verfahren zur direkten Verbesserung der Geometrie eines Individuums (Anders & König, 2011). Ein solches Verfahren könnte beispielsweise Zick-Zack Linien begradigen (Abb. 61d) oder innerhalb eines gewissen Toleranzbereiches Linien an die Eckpunkte der Begrenzung haften (Abb. 61b). Für die ES würde das bedeuten, dass nach jeder Generation nicht die ursprünglichen, sondern die korrigierten Varianten beurteilt werden, was die Qualität der Lösungen und auch die Berechnungsgeschwindigkeit steigern könnte.

Um eine große Anzahl von Regionen innerhalb der Begrenzung anzuordnen, ist weiterhin zu überprüfen, inwieweit eine Hierarchisierung des Unterteilungsalgorithmus (die wiederholte Unterteilung der annähernd rechtwinkligen Regionen) sinnvoll ist.

Für das Herstellen einer dichten Packung von Rechtecken konnte bereits gezeigt werden, dass eine solche Herangehensweise einen enormen Einfluss auf die Geschwindigkeit zur Berechnung einer Lösung hat, da sich die Zahl der gleichzeitig zu optimierenden Elemente verringert (siehe Kapitel 10).

8.5. Konklusion und Ausblick

In diesem Kapitel wurde der rechte Winkel als Ausgangspunkt für die Erzeugung von Strukturen behandelt. Es wurde eine Methode entwickelt, wie man unabhängig von vordefinierten Elementen (Rechtecken) Layouts innerhalb frei definierbarer Begrenzungen so generieren kann, dass die darin auftretenden Winkel möglichst rechtwinklig sind. Die Teilung einer Fläche durch Voronoi-Zerlegung und deren Optimierung hinsichtlich der in den Strukturen auftretenden Winkel ist ein vielversprechender Ansatz zur Bearbeitung des in diesem Kapitel definierten Layoutproblems. Die verwendete ES ist derzeit noch nicht vollständig ausgereift. Die Berechnung der oben dargestellten Beispiele nimmt in etwa 10 Sekunden (ca. 50 Generationen) in Anspruch. Für eine optimale Nutzerinteraktion wäre die Geschwindigkeit zur Berechnung einer Lösung auf den Bruchteil einer Sekunde zu reduzieren (siehe Punkt 3.2.4). Dadurch würde es beispielsweise möglich, in Echtzeit neue Räume hinzuzufügen oder bestehende Räume zu verschieben, wobei die innerhalb der Struktur auftretenden Winkel jederzeit möglichst gut an rechte Winkel angenähert bleiben. Um die Berechnungsgeschwindigkeit zu erhöhen, ist eine Kombination des vorgestellten iterativen Verfahrens mit direkten Verfahren zu überprüfen. Ein weiterer Ausarbeitungsschritt für das vorgestellte Layoutsystem bestünde darin, den Evaluationsmechanismus um funktionale Restriktionen (z.B. topologische Beziehungen und Raumgrößen) zu erweitern.

9. Vergleich zweier Methoden zur Erzeugung von Grundriss-Layouts

Dichte Packung vs. Unterteilungsalgorithmen

Reinhard König, Katja Knecht

In diesem Kapitel werden die Methoden zur Erzeugung von Grundrisslayouts mittels dichter Packung (Kapitel 5) und mittels Unterteilungsalgorithmen (Kapitel 7) miteinander verglichen. Basierend auf zwei Grundrissszenarien werden die Eigenschaften der beiden Layoutsolver analysiert. Die Analysen betrachten erstens die Geschwindigkeit, mit der Lösungen gefunden werden, zweitens die Zuverlässigkeit, mit der optimale Lösung gefunden werden, sowie drittens die Anzahl an unterschiedlichen Lösungen, die überhaupt gefunden werden können. Die Ergebnisse der Analysen werden einer vergleichbaren Studie von Flemming, Baykan et al. (1992) gegenübergestellt. Abschließend werden die Unterschiede der beiden verglichenen Methoden im Hinblick auf die Möglichkeiten der Nutzerinteraktion besprochen. In einer abschließenden Betrachtung wird dargestellt, dass keine der betrachteten Methoden der anderen überlegen ist, sondern dass sie sich aufgrund ihrer unterschiedlichen Eigenschaften für verschiedene Anwendungsszenarien mehr oder weniger eignen.

9.1. Szenarios

Den Vergleichsanalysen für die beiden Layoutsolver (Punkt 9.2) liegen zwei einheitliche Szenarien zugrunde. In der Gestaltung des ersten Szenarios haben wir uns an einem von Flemming et al. (1992) beschriebenen Referenz-Layoutproblem mit entsprechenden Restriktionen orientiert. Die Anzahl und Größe der acht Räume dieses Referenzproblems ist in Tab. 8 dargestellt und wird im Folgenden als Layoutszenario 1 bezeichnet.

Im Unterschied zum Referenzproblem haben wir die topologische Restriktion, also die Vorgabe, welche Räume miteinander benachbart sein müssen, verändert, da bei den hier verglichenen Systemen keine Relationen bezüglich Himmelsrichtungen

vorgesehen sind. Als topologische Restriktion wird für beide Szenarien eine Sterntopologie festgelegt, bei der alle Räume zu einem zentralen Raum benachbart sein müssen. Da diese Restriktion im Vergleich zum Referenzproblem relativ schwer zu erfüllen ist, sind aus unserer Sicht die Ergebnisse der Untersuchungen von (Flemming, et al., 1992) mit den hier durchgeführten Analysen gut vergleichbar. Als zweites Szenario wird ein Problem mit zehn Räumen definiert, deren Größen in Tab. 9 angegeben sind. Das zweite Szenario ist schwieriger zu lösen als das erste.

Spaces		
No.	Name	Size
0	hall	10 m²
1	court	7 m²
2	living room	22 m²
3	master bedroom	14 m²
4	bedroom 1	10 m²
5	bedroom 2	10 m²
6	kitchen	8 m²
7	bathroom	5 m²

Tab. 8: Layoutszenario 1.

Wir gehen davon aus, dass mit den betrachteten Szenarien ein Großteil real vorkommender Planungsaufgaben abgedeckt werden kann, da diese in der Regel nicht komplexer als die hier betrachteten Szenarien sind. Bei Aufgaben, bei denen man es mit umfangreicheren Raumprogrammen, also mehr Räumen als bei unseren Testszenarien zu tun hat, werden in der Regel funktional zusammenhängende Teilbereiche gebildet, welche über verschiedene Hierarchieebenen miteinander in Beziehung gesetzt werden können (vgl. Kapitel 10).

Spaces		
No.	Name	Size
0	hall	10 m²
1	court	7 m²
2	living room	12 m²
3	master bedroom	12 m²
4	bedroom 1	10 m²
5	bedroom 2	8 m²
6	kitchen	8 m²
7	bathroom	5 m²
8	bathroom2	4 m²
9	dining room	10 m²

Tab. 9: Layoutproblem 2.

9.2. Vergleichsanalysen

In diesem Abschnitt vergleichen wir die Eigenschaften der oben eingeführten Systeme zur Erzeugung von Grundrisslayouts mittels dichter Packung (Kapitel 5) und mittels Unterteilungsalgorithmen (Kapitel 7). Der Layoutsolver „Dichte Packung" wurde für diese Vergleichsanalyse erweitert, indem mehrere Populationen parallel zur Lösungssuche verwendet werden, wobei eine konservative Population als (+)-Selektion implementiert wurde, bei der stets die beste Variante erhalten bleibt, und mehrere innovative Population als (,)-Selektion umgesetzt wurden, bei denen lokale Optima leichter überwunden werden können, da beste Lösungen verloren gehen können, sodass diese die Suche in anderen Richtungen nicht blockieren. Die jeweils besten Lösungen der (,)-Selektions-Populationen werden regelmäßig in die (+)-Selektions-Population kopiert. Auf diese Weise können einige Schwierigkeiten des Systems „Dichte Packung" überwunden werden, die am Ende von Kapitel 5.2.4 beschrieben wurden. Die Gegenüberstellung der beiden Systeme erfolgt anhand der oben dargestellten Layoutszenarien (Tab. 8 und Tab. 9). Interessant für die Anwendung der entwickelten Systeme ist vor allem, wie schnell Lösungen gefunden werden können (Performance), wie sicher man sein kann, dass in einer bestimmten Zeitspanne eine Lösung gefunden wird, wenn sie theoretisch möglich ist (Zuverlässigkeit), und wie viele unterschiedliche Lösungen überhaupt gefunden werden können (Varianz).

9.2.1. Performance

Für die Analyse der Performance der zu vergleichenden Layoutsolver greifen wir auf die Methode zurück, die bereits in Kapitel 5.1.1 für Performancetests zur Evaluation des Layoutsolvers „Dichte-Packung" verwendet wurde. Jedes Layoutsystem wird je 100-mal durchlaufen, wobei die Qualität der Ergebnisse in einem Diagramm aufgezeichnet wird (Abb. 62). Dadurch wird sowohl die Geschwindigkeit, mit der Lösungen mit bestimmter Qualität im Durchschnitt erzielt werden, als auch die durchschnittliche Qualität der Lösungen nach einer definierten Zeit ablesbar. Beide Systeme verwenden eine $(\mu+\lambda)$-ES, wie in den entsprechenden Kapiteln 5 und 7 detailliert beschrieben. Für die charakteristischen Parameterwerte wurden gewählt μ=7, λ=35, ρ=2, ρ_r=0,75. Alle Berechnungen beider Layoutsysteme wurden auf einem

Dell Precision T7500-2 (mit Intel Xenon CPU; 2,40 GHz; 48 GB Arbeitsspeicher und Windows 7, 64Bit) ausgeführt.

Die Diagramme in Abb. 62 und Abb. 63 zeigen die Ergebnisse der Performancetests. Jeweils links sind die Diagramme für den Layoutsolver „Dichte Packung" zu sehen, bei dem an den y-Achsen je die Summen aller Überlappungsflächen (linke Ordinate) und aller Distanzen (rechte Ordinate) als Fitnesswerte angetragen sind. Die Mittelwerte der beiden Fitnesswerte sind über den Verlauf der 100 aufgezeichneten Generationen (bzw. Iterationen) als Mittelwertlinien geplottet, wobei die grüne durchgezogene Linie die Mittelwerte der Überlappungsflächen und die blaue gestrichelte Linie die Mittelwerte der Distanzen darstellen. Beide Linien fallen exponentiell ab und erreichen ab ca. 50 Generationen sehr gute Werte, welche sich im weiteren Verlauf nur noch marginal verbessern. Das bedeutet, dass wir davon ausgehen können, dass der Layoutsolver im Mittel nach 50 Generationen brauchbare Lösungen für Layoutproblem 1 findet. Für die Berechnung von 100 Generationen benötigt der Layoutsolver durchschnittlich 27 Sekunden und für 50 Generationen entsprechend 13,5 Sekunden. Bei allen hier gemachten Zeitangaben ist zu berücksichtigen, dass diese Durchschnittswerte von 100 wiederholten Programmdurchläufen darstellen. Aufgrund des nicht optimierten Programmcodes, der Erstellung der Performancediagramme und der Berechnung der Mittelwertkurven wird die Dauer eines Durchlaufs mit der Zeit immer größer. Ein einzelner Lauf des Programms ist ca. 2-3-mal schneller als die hier angegebenen Durchschnittswerte.

Auf der rechten Seite in Abb. 62 und Abb. 63 sind jeweils die Diagramme für den Layoutsolver „Unterteilungsalgorithmus" zu sehen. Da sich die geforderten Raumgrößen hier direkt berechnen lassen, müssen die Unterteilungen so vorgenommen werden, dass die korrekten Nachbarschaftsbeziehungen zustande kommen. Dementsprechend ist bei diesem Layoutsolver nur ein Kriterium zu optimieren, nämlich jenes der zu erfüllenden Nachbarschaftsrelationen. Wie gut diese erfüllt werden, ist in den Diagrammen in Abb. 62 und Abb. 63 auf der rechten Seite an der y-Achse angetragen, die mit „topology" bezeichnet ist. Der Wert 1 bedeutet, dass alle Nachbarschaftsrelationen vollständig erfüllt sind, und der Wert 0, dass die Räume, die miteinander benachbart sein sollen, in maximalen Abständen zueinander liegen und folglich keine der geforderten Nachbarschaftsbeziehungen erfüllt wird. Die rote

durchgezogene Linie stellt die Mittelwertlinie der Relationen über den Verlauf der 100 aufgezeichneten Generationen dar. Die Linie fällt exponentiell ab und erreicht ab ca. 50 Generationen sehr gute Werte, welche sich im weiteren Verlauf nur noch marginal verbessern. Das bedeutet, dass wir davon ausgehen können, dass dieser Layoutsolver ebenfalls im Mittel nach 50 Generationen brauchbare Lösungen für Layoutproblem 1 findet. Für die Berechnung von 100 Generationen benötigt das System durchschnittlich 11,69 Sekunden und für 50 Generationen 3,58 Sekunden. Folglich erhöht sich auch bei diesem Layoutsolver aufgrund der fehlenden Programmcodeoptimierung, der grafischen Ausgabe der Diagramme und Berechnungen der Mittelwertkurven die Durchlaufdauer mit zunehmender Generationenzahl.

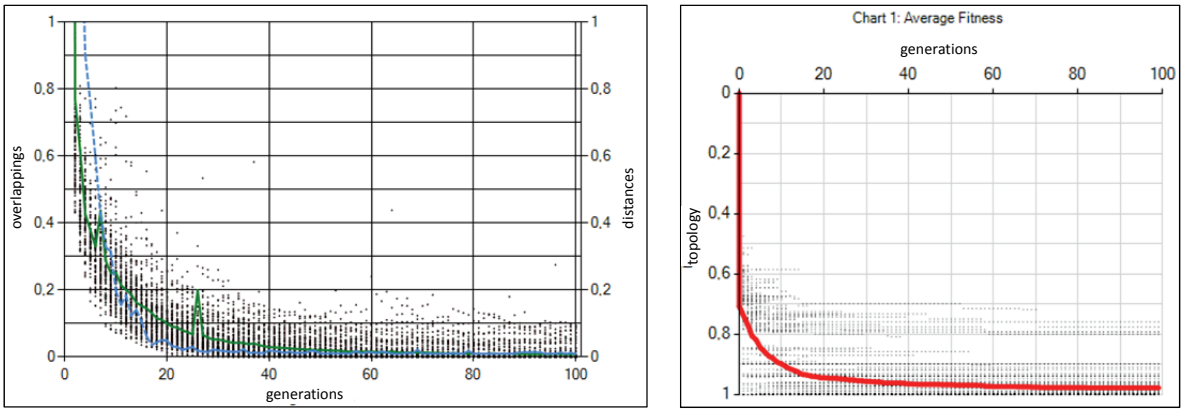


Abb. 62: Gegenüberstellung der Performance für Layoutproblem 1 (8 Räume). Links: Layoutsolver „Dichte Packung“; an den y-Achsen sind je die Summen aller Überlappungsflächen und aller Distanzen als Fitnesswerte angetragen. Rechts: Layoutsolver „Unterteilungsalgorithmus“; an der y-Achse sind die erfüllten Nachbarschaftsrelationen angetragen. Die x-Achse gibt die Generationen der ES an.

Die Performancetests beider Layoutsolver für Layoutproblem 2 sind in Abb. 63 dargestellt und folgen der gleichen Konzeption wie oben erläutert. Die Mittelwertlinien, welche die durchschnittliche Qualität der Lösungen angeben, fallen hier ebenfalls bei beiden Systemen exponentiell ab und erreichen ab ca. 50 Generationen gute Werte, welche sich im weiteren Verlauf nur noch marginal verbessern. Wir können folglich auch bei Layoutproblem 2 davon ausgehen, dass beide Systeme im Mittel nach 50 Generationen brauchbare Lösungen finden. Für die Berechnung von 100 Generationen benötigt der Layoutsolver „Dichte Packung“ 44 Sekunden. Der Layoutsolver „Unterteilungsalgorithmus“ benötigt für die Berechnung von 100 Generationen 12,66 Sekunden.

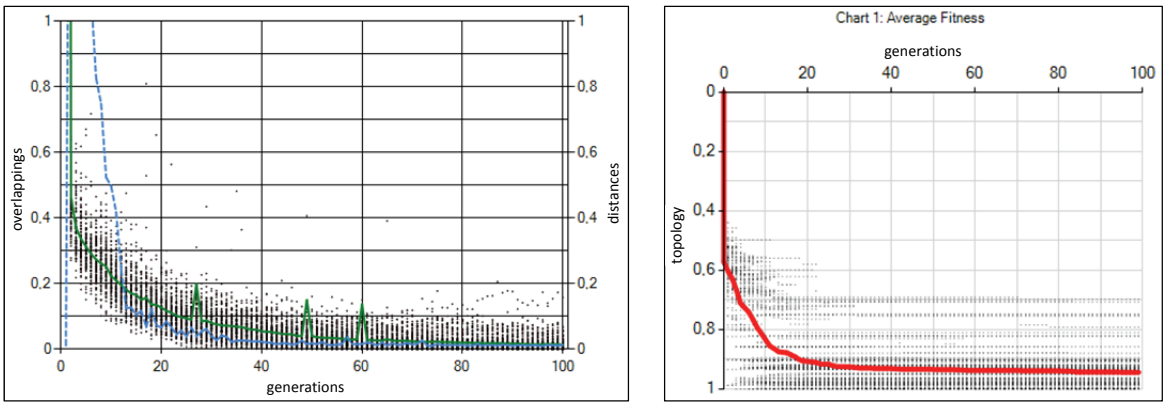


Abb. 63: Gegenüberstellung der Performance für Layoutproblem 2 (10 Räume). Links: Layoutsolver „Dichte Packung“; an den y-Achsen sind je die Summen aller Überlappungsflächen und aller Distanzen als Fitnesswerte angetragen. Rechts: Layoutsolver „Unterteilungsalgorithmus“; an der y-Achse sind die erfüllten Nachbarschaftsrelationen angetragen. Die x-Achse gibt die Generationen der ES an.

9.2.2. Zuverlässigkeit

In diesem Abschnitt wird untersucht, wie zuverlässig die betrachteten Systeme Lösungen für die beiden Layoutprobleme finden. Für eine gute Vergleichbarkeit betrachten wir die Layouts, welche von den Systemen nach 100 Generationen gefunden werden. Die Punktwolken in den Diagrammen in Abb. 62 und Abb. 63 zeigen, dass es bei manchen Durchläufen zu wesentlich schlechteren Ergebnissen kommt, als die jeweiligen Mittelwertkurven anzeigen.

Beim Layoutsolver „Dichte Packung“ werden Lösungen als brauchbar betrachtet, wenn ihre Überlappungsflächen einen bestimmten Wert unterschreiten. Dieser Wert muss in den Skalen der Diagramme in Abb. 62 und Abb. 63 kleiner 0.1 sein (die Punkte geben die Überlappungsflächen an). Wir können in den Diagrammen zum Layoutsolver „Dichte Packung“ erkennen, dass einige Punkte bei Generation 100 über diesem Wert liegen – sie stellen unbrauchbare Lösungen dar. Es lässt sich feststellen, dass der Layoutsolver „Dichte Packung“ bei beiden Layoutproblemen ca. 10% unbrauchbare Lösungen produziert. Es hat dementsprechend eine Zuverlässigkeit von ca. 90%. Diese kann durch die Berechnung weiterer Generationen auf annähernd 100% erhöht werden.

Beim Layoutsolver „Unterteilungsalgorithmus“ werden Lösungen als brauchbar betrachtet, wenn alle Nachbarschaftsbeziehungen erfüllt sind und zwischen benachbarten Räumen eine festgelegte Mindestüberlappung besteht. Den Diagrammen auf der rechten Seite in Abb. 62 und Abb. 63 zufolge werden im Mittel nach 100 Gene-

rationen die Abstände der Räume, die miteinander benachbart sein sollen, zu ca. 95% erfüllt. Wir können in den Diagrammen zum Layoutsolver „Unterteilungsalgorithmus" eine geringe Streuung der Messpunkte für erfüllte Nachbarschaftsbeziehungen bei Generation 100 erkennen. In Abb. 65 sind die einzelnen Layouts dargestellt, deren Qualitäten durch die Punkte in den Diagrammen repräsentiert werden. Es lässt sich feststellen, dass der Layoutsolver „Unterteilungsalgorithmus" bei Layoutproblem 1 ca. 10% und bei Layoutproblem 2 bis zu ca. 20% unbrauchbare Lösungen produziert. Es hat dementsprechend eine Zuverlässigkeit von 90% bzw. 80%. Die Diskrepanz zwischen dem relativ guten mittleren Fitnesswert und der Zuverlässigkeit der Lösungsausgabe ergibt sich aus der Eigenschaft des Layoutsolver, insbesondere bei einer größeren Anzahl von Räumen, wie in Layoutproblem 2 der Fall, Lösungen zu entwickeln, die zwar die topologische Vorgabe erfüllen, aber die vorgeschriebene Mindestüberlappung nicht für alle Nachbarschaften einhalten.

9.2.3. Varianz

Varianz bezeichnet das Spektrum unterschiedlicher Lösungen (Rittel, 1992), das von den beiden hier betrachteten Systemen erzeugt werden kann. Je größer die Varianz, desto mehr verschiedene Lösungen kann ein Layoutsolver generieren. Varianz kann folglich als Messwert für die Vielfalt möglicher Lösungen verstanden werden. Ideaerweise erwarten wir von einem System zur automatischen Erzeugung von Grundrisslayouts, dass es jede beliebige Konfiguration generieren kann. Einen Überblick über die Varianz der beiden hier verglichenen Layoutsolver bieten Abb. 64 und Abb. 65. Die Schwierigkeit besteht nun darin, die verschiedenen Lösungen anhand eines Kennwerts voneinander zu unterscheiden und einen Kennwert für die Varianz eines Systems zu formulieren.

Für die Differenzierung der Lösungen führen wir einen Proportionskennwert θ ein, der anhand folgender Funktion ermittelt wird:

$$\theta = \sum_{i=1}^{n} \left(\frac{kS_i}{lS_i} \right) \Big/ N_R \qquad (28)$$

Wobei kS_i die kürzere und lS_i die längere Seite eines Raums bezeichnen und N_R die Anzahl der Räume repräsentiert. Die Proportionskennwerte der einzelnen Lösungen

sind am Anfang der Dateinamen in Abb. 64 und Abb. 65 angegeben. Die Kennwerte erlauben eine grobe Ordnung der unterschiedlichen Lösungen. Allerdings haben teilweise sehr verschiedene Lösungen einen ähnlichen Kennwert. Daher ist für einen genauen Vergleich eine visuelle Inspektion erforderlich.

Mithilfe der Proportionskennwerte und dem visuellen Vergleich der Layouts in Abb. 64 und Abb. 65 wird deutlich, dass ein Großteil der Lösungen der beiden Systeme gleich ist, trotz der unterschiedlichen Charakteristika der in den Systemen eingesetzten Algorithmen. So kann beispielsweise der Layoutsolver „Unterteilungsalgorithmus" nur Lösungen finden, bei denen mindestens eine durchgehende Wand von einer zur anderen Außenseite des Umgebungsrechtecks existiert. Wie in Abb. 64 zu erkennen ist, findet auch der Layoutsolver „Dichte Packung", zumindest bei der den Untersuchungen zugrunde liegenden Sterntopologie als topologischer Restriktion, nur selten Layouts, bei denen dies nicht der Fall ist. Allerdings existieren auch Layoutlösungen, die vorwiegend von nur einem der beiden Systeme gefunden werden. So generiert der Layoutsolver „Unterteilungsalgorithmus" relativ häufig Layouts, bei denen der zentrale Raum der Sterntopologie an der Außenwand angeordnet ist. Diese Variante kommt beim Layoutsolver „Dichte Packung" äußerst selten vor und ist bei den in Abb. 64 gezeigten Lösungen nicht vertreten.

Als einfacher Kennwert für die Varianz v eines Systems bietet sich die Differenz zwischen maximalem und minimalem Proportionskennwert θ an:

$$v = \theta_{max} - \theta_{min} \tag{29}$$

Für den Layoutsolver „Dichte Packung" ergibt sich bei 100 Durchläufen für acht Räume eine Varianz v_8=0,52 und für 10 Räume v_{10}=0,44. Für den Layoutsolver „Unterteilungsalgorithmus" ergeben sich die entsprechenden Varianzen v_8=0,52 und v_{10}=0,45. Die Werte für die Varianz v hängen von den Durchläufen ab und können geringfügig von den angegebenen Werten abweichen.

Neben dem dargestellten Kennwert für die Varianz v eines Systems ist die Verteilung der Proportionskennwerte interessant. Diese vermittelt einen Eindruck, mit welcher Wahrscheinlichkeit welche Varianten generiert werden und ob der Layoutsolver zu Lösungen in einem bestimmten Proportionskennwerte-Bereich ten-

diert. Ideal wäre eine möglichst große Varianz mit einer gleichmäßigen Verteilung der Proportionskennwerte.

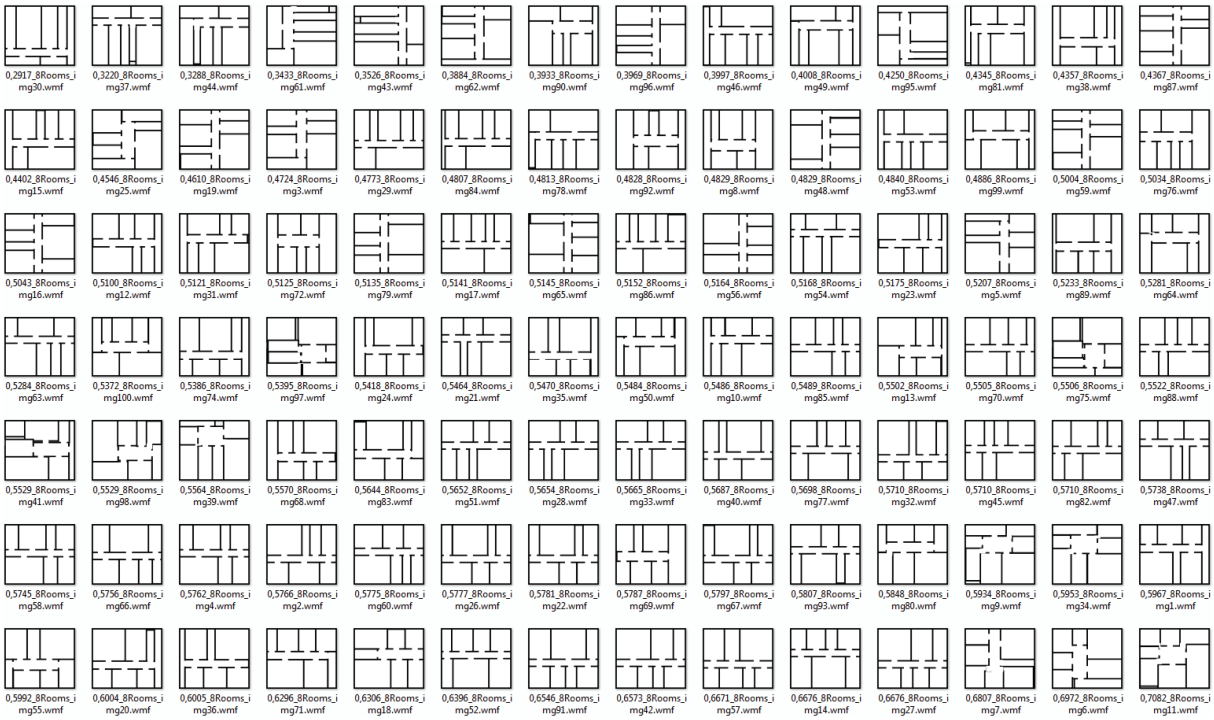


Abb. 64: Screenshot der Lösungen des Systems „Dichte Packung“ nach je 100 Generationen.

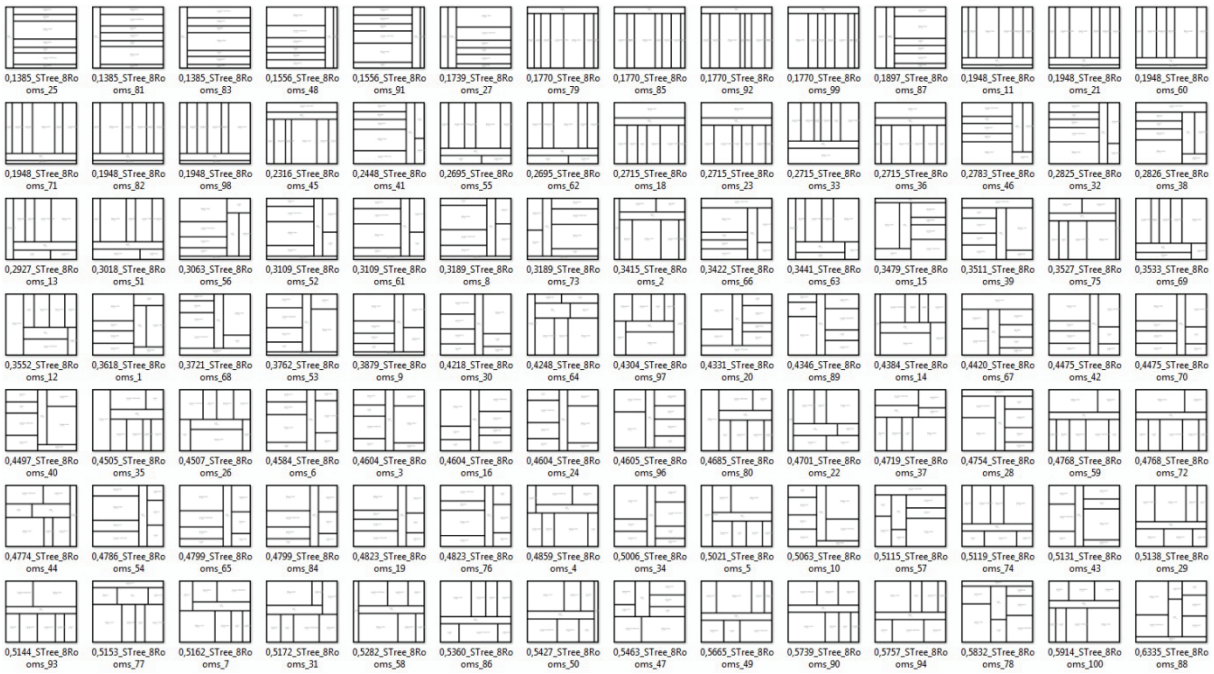


Abb. 65: Screenshot der Lösungen des Systems „Unterteilungsalgorithmus“ nach je 100 Generationen.

In den Histogrammen in Abb. 66 und Abb. 67 sind die entsprechenden Verteilungen für je 100 generierte Layouts der beiden Systeme dargestellt. In den Diagrammen zum Layoutsolver „Dichte Packung" in Abb. 66 ist gut zu erkennen, dass sich die Proportionskennwerte jeweils in einem bestimmten Bereich stark häufen und die Verteilung tendenziell einer Normalverteilungskurve folgt. Das bedeutet, dass gewisse Konfigurationen bevorzugt generiert werden (vgl. auch Abb. 64). In den Diagrammen zum Layoutsolver „Unterteilungsalgorithmus" in Abb. 67 wird deutlich, dass sich die Proportionskennwerte vergleichsweise gleichmäßig über das gesamte Varianz-Spektrum des Layoutsolvers verteilen. Dieser scheint keine starke Tendenz zu bestimmten Konfigurationen zu haben, lediglich die oberen Proportionskennwerte kommen seltener vor.

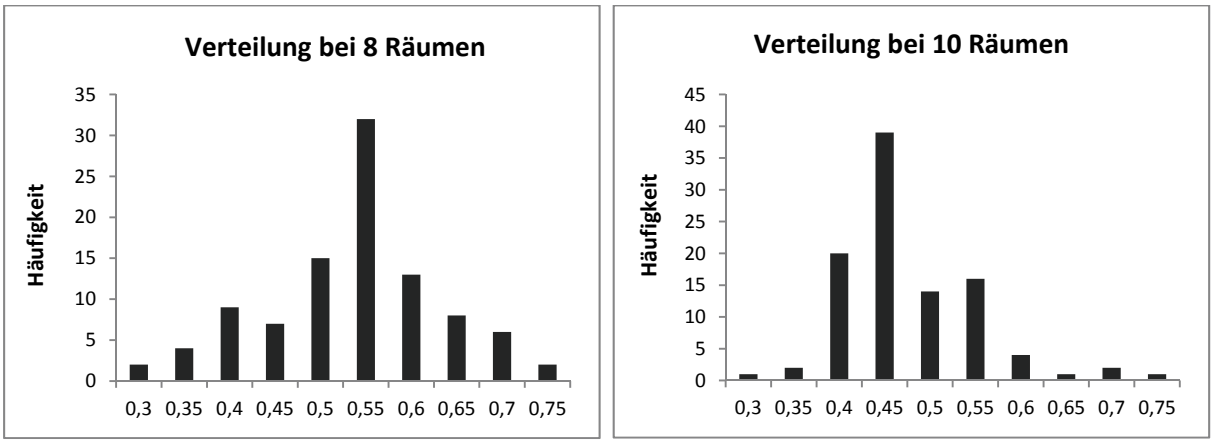


Abb. 66: Histogramme der Proportionskennwerte für den Layoutsolver „Dichte Packung" bei 100 Durchläufen. Links: Für 8 Räume, Rechts: Für 10 Räume.

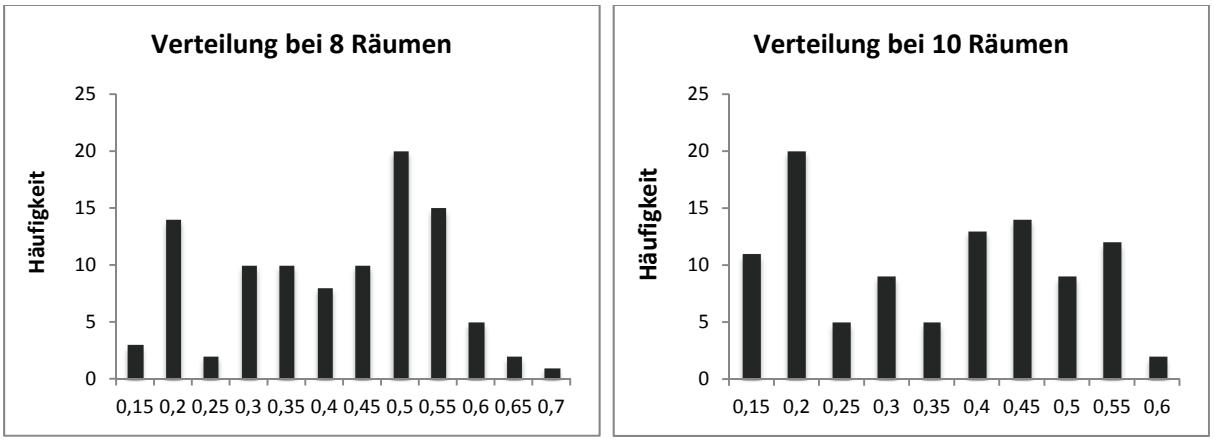


Abb. 67: Histogramme der Proportionskennwerte für den Layoutsolver „Unterteilungsalgorithmus" bei 100 Durchläufen. Links: Für 8 Räume, Rechts: Für 10 Räume.

9.2.4. Bewertung

Der Layoutsolver „Unterteilungsalgorithmus“ schneidet im Vergleich mit dem Layoutsolver „Dichte Packung“ bei der Performanceanalyse besser ab, da es Lösungen um ein Vielfaches schneller generiert, wobei die Qualität der Lösungen und die Zuverlässigkeit beider Systeme in etwa gleich gut sind. Auch die Varianz beider Systeme kann als gleich groß bewertet werden. Jeder Layoutsolver ist mit wenigen Ausnahmen in der Lage, alle möglichen Konfigurationen zu finden. Bei der Gegenüberstellung der Verteilung der Proportionskennwerte liefert wiederum der Layoutsolver „Unterteilungsalgorithmus“ die vorteilhafteren Ergebnisse, da die Kennwerte der Lösungen, die von diesem Layoutsolver erzeugt werden, relativ gleichmäßig verteilt sind und keine Konfigurationen gehäuft auftreten.

9.3. Interaktionscharakteristiken

In diesem Abschnitt werden die Unterschiede der verglichenen Layoutsolver hinsichtlich der Möglichkeiten zur Nutzerinteraktion mit den Lösungen knapp umrissen. Eine ausführlichere Darstellung zur Nutzerinteraktion mit generativen Layout-Systemen findet sich in Kapitel 11.

Die beiden Layoutsolver unterscheiden sich im Wesentlichen darin, wie und in welchem Umfang mit einzelnen Elementen des Layouts interagiert werden kann. Die Interaktionsmöglichkeiten und -formen sind abhängig von den Charakteristiken der eingesetzten Algorithmen. So erlaubt der „Unterteilungsalgorithmus“ lediglich eine linienbasierte Interaktion. Das bedeutet, dass hier die Wände eines Layouts verschoben werden können, wobei die Verschiebung der Unterteilungslinie um die Strecke s auch die Raumgrenze grafisch um s verschiebt. Folglich lassen sich durch die Verschiebung in erster Linie die Raumgrößen verändern. Probleme ergeben sich daraus, dass lange Schnittlinien nicht unterteilt werden können, d.h. dass sich bei ihrem Verschieben gleichzeitig die Raumgrenzen mehrerer Räume und damit ihre Raumgrößen verändern. Die Positionen einzelner Räume können in diesem System hingegen nicht direkt manipuliert werden. Diese Eigenschaft erschwert es, bestimmte Räume, z.B. Treppenhäuser bei mehrgeschossigen Bauten, präzise übereinander zu platzieren und damit innerhalb eines Layouts zu fixieren.

Der Layoutsolver „Dichte Packung“ ermöglicht hingegen zusätzlich eine raumbasierte Interaktion, welche weitreichendere Möglichkeiten bietet als die linienbasierte. Neben der Anpassung der Raumgrößen durch Manipulation der Begrenzungslinien der Elemente können hier einzelne Räume innerhalb des Layouts verschoben bzw. in ihrer Position fixiert werden. Dadurch können auch exakte vertikale räumliche Verbindungen sowie Verbindungen zu übergeordneten Hierarchieebenen realisiert werden.

Bei den hier verglichenen Systemen wurden die Interaktionsfunktionen nur rudimentär implementiert, weshalb eine detailliertere Betrachtung bzw. Unterscheidung der Systeme nicht sinnvoll erscheint.

9.4. Konklusion und Ausblick

Beide Systeme erscheinen praxistauglich für das automatische Erstellen von Grundrissen im Rahmen komplexer Planungsprozesse. Die Unterschiede liegen eher im Detail. So ist die Performance beim „Unterteilungsalgorithmus“ besser, wobei sich der Layoutsolver „Dichte Packung“ besser zur Nutzerinteraktion eignet. Die Varianz beider Systeme ist vergleichbar groß. Diese Eigenschaft ist entscheidend, damit sichergestellt ist, dass eine möglichst große Menge verschiedener Layouts generierbar ist, welche durch weitere Restriktionen eingeschränkt werden kann.

Verglichen mit dem von Flemming et al. (1992) analysierten Layoutsolver zur Lösung des oben erwähnten Referenz-Layoutproblems ist Folgendes festzustellen: Zwar ist die Performance der hier vorgestellten Layoutsolver auf den ersten Blick besser, da eine einzelne Lösung in ca. 4 Sekunden gefunden wird, allerdings sind die Vergleichssysteme von Flemming et al. bereits mehr als 20 Jahre alt. Deren schnellstes System benötigte damals 12 Sekunden für die Lösung des im Rahmen ihrer Untersuchungen beschriebenen Szenarios. Es ist zu anzunehmen, dass die Performance dieses Systems auf aktueller Computerhardware wesentlich bessere Werte erzielen würde.

Abschließend soll darauf hingewiesen werden, dass mit den hier vorgestellten Layoutsolver auch komplexere Grundrisse generiert werden können. So lassen sich beispielsweise durch das Entfernen einer gemeinsamen Trennwand einzelne Räume

zusammenlegen, um komplexere Formen, wie z.B. L-förmige Räume, zu bilden. Auf diese Weise können zunächst größere Räume und Flächen in Unterräume und -flächen zerlegt und anschließend geeignete Unterräume wieder verbunden werden. Auf diese Weise wird z.B. die Generierung langer verzweigter Gänge in komplizierten Gebäudekomplexen unterstützt.

III. Anwendungsstudien

10. Hierarchische Gliederung von Layouts[15]

Reinhard König, Sven Schneider

In den vorangegangenen Kapiteln wurde die Anordnung von Elementen unter Berücksichtigung topologischer Relationen auf einer bestimmten Maßstabsebene behandelt. Diese Art von Layoutproblem wird im Folgenden als einfach gegliedertes oder nicht-hierarchisches Layoutproblem (NHLP) bezeichnet.

In diesem Kapitel wird untersucht, wie sich hierarchisch gegliederte Layoutprobleme (HLP) mittels des in Kapitel 5 vorgestellten Layoutsolvers lösen lassen. Es wird gezeigt, dass es zur Lösung eines HLP notwendig ist, Relationen über Hierarchiegrenzen hinweg definieren zu können, und welche Anforderungen HLPs hinsichtlich der Problembeschreibung haben. Anhand eines Testszenarios werden NHLP und HLP miteinander verglichen.

10.1. Hierarchische Gliederung der Elemente und grenzübergreifende Relationen

Als hierarchisch gegliedert werden Layouts bezeichnet, bei denen Elemente mehrfach ineinander verschachtelt sind. Das bedeutet, dass sich Elemente innerhalb anderer Elemente befinden und selbst Teile anderer Elemente sein können. Als Beispiel für ein HLP betrachten wir ein Wohnhaus, bei dem auf einem Geschoss die Erschließung und mehrere Wohnungen angeordnet werden müssen. Innerhalb der Wohnungen müssen wiederum mehrere Zimmer untergebracht werden und innerhalb der Zimmer sind die Möbel zu platzieren. Die Art und Weise einer hierarchischen Gliederung ist situationsabhängig und kann vom Entwerfenden frei bestimmt werden. So ist es möglich, ein Haus entweder in einzelne Wohnungen oder aber in bestimmte Zonen (wie z.B. privat und öffentlich) zu gliedern.

[15] Teile dieses Kapitels beruhen auf dem Artikel von Koenig, R. & Schneider, S. (2012). Hierarchical structuring of layout problems in an interactive evolutionary layout system. AIEDAM: Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 26(2).

Bezugnehmend auf die Beziehungen zwischen den Elementen, ist es bei HLP erforderlich, hierarchieebenen-übergreifende (grenzübergreifende) Relationen einzuführen. Relationen stellen im Folgenden die topologischen Beziehungen dar, welche in Kapitel 5.2 beschrieben wurden. Grenzübergreifende Relationen sind erforderlich, da die Hierarchisierung zwar angibt, dass sich bestimmte Elemente nur innerhalb eines anderen Elementes anordnen können, funktionale Anforderungen an diese Elemente jedoch auch abhängig von Elementen sein können, die sich auf anderen Hierarchieebenen befinden (siehe Abb. 68). Bezugnehmend auf das oben angeführte Beispiel eines Wohnhauses ist eine grenzübergreifende Relation erforderlich, um die wohnungsinterne Erschließung mit der Erschließung des Wohnhauses zu verbinden. Folglich sind grenzübergreifende Relationen für die Beschreibung von Layoutproblemen mit hierarchisch verschachtelten Elementen unentbehrlich.

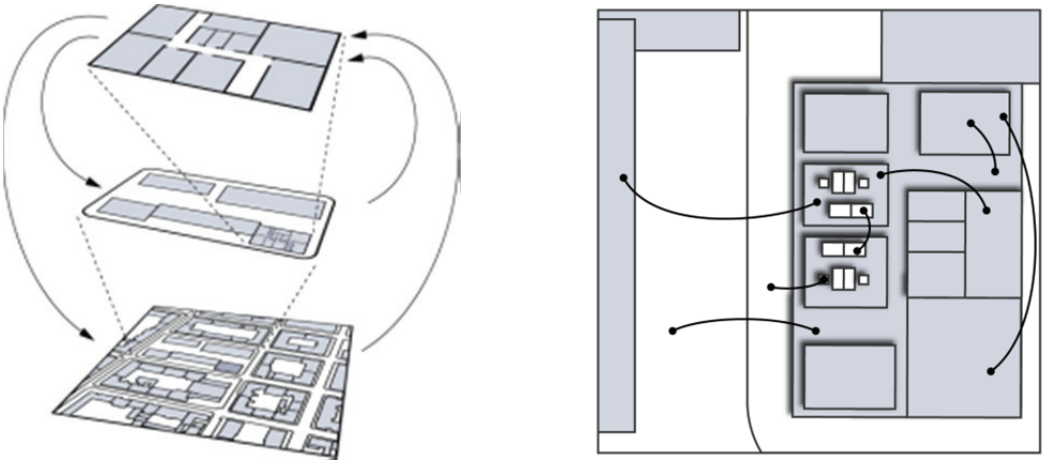


Abb. 68: Schemaskizzen zur hierarchischen Gliederung von Layouts (links) und zu grenzübergreifenden Relationen (rechts).

HLPs beinhalten also zum einen verschachtelte Elemente. Zum anderen können sie Beziehungen zwischen Elementen beinhalten, die sich auf unterschiedlichen Hierarchieebenen befinden. Inwieweit sich HLPs mit grenzübergreifenden Relationen mit dem generativen System aus Kapitel 5 bearbeiten lassen und was dabei zu berücksichtigen ist, wird im Folgenden geklärt.

10.2. Explizite und implizite interne und externe Relationen

Hinsichtlich der Interaktion eines Nutzers mit dem Layoutsystem lassen sich explizite und implizite Anforderungen an ein Layout unterscheiden. Explizite Anforderun-

gen sind Anforderungen, welche ein Nutzer durch seine Eingaben ausdrücklich vorgibt. Implizite Anforderungen sind dagegen Anforderungen, die sich aus den expliziten Vorgaben ergeben und nicht ausdrücklich festgelegt wurden. Am Beispiel der Definition topologischer Beziehungen wird dies deutlich: Es gibt prinzipiell zwei verschiedene Arten von Relationen, interne und externe. Intern sind Relationen zwischen Elementen innerhalb einer Hierarchieebene, wie z.B. die topologische Beziehung zwischen Zimmern innerhalb einer Wohnung. Externe Relationen hingegen sind Relationen zwischen Elementen auf jeweils unterschiedlichen Hierarchieebenen, z.B. die gewünschte Nachbarschaft eines Wohnungsflurs zum Gang des Gebäudes, in dem sich die Wohnung befindet. Im ersten Fall liegen Wohnung und Gang auf der gleichen Hierarchieebene, im zweiten Wohnungsflur und Gang auf unterschiedlichen Ebenen. Das bedeutet, dass auf der Ebene des Gebäudes sowohl die Wohnung mit dem Gang als auch der Wohnungsflur mit dem Gang benachbart sein muss. Da die Raumbeziehung Wohnung – Gang nicht explizit vorgegeben wurde, bedeutet das, dass diese Relation implizit durch die Verknüpfung Wohnungsflur – Gang entstehen muss. Sowohl interne als auch externe Relationen können explizit (durch den Nutzer) vorgegeben werden. Implizite Relationen entstehen ausschließlich indirekt durch die Vorgabe externer Relationen. Dabei können sowohl interne-implizite als auch externe-implizite Relationen bei tiefer verschachtelten Layoutproblemen entstehen (siehe Abb. 69).

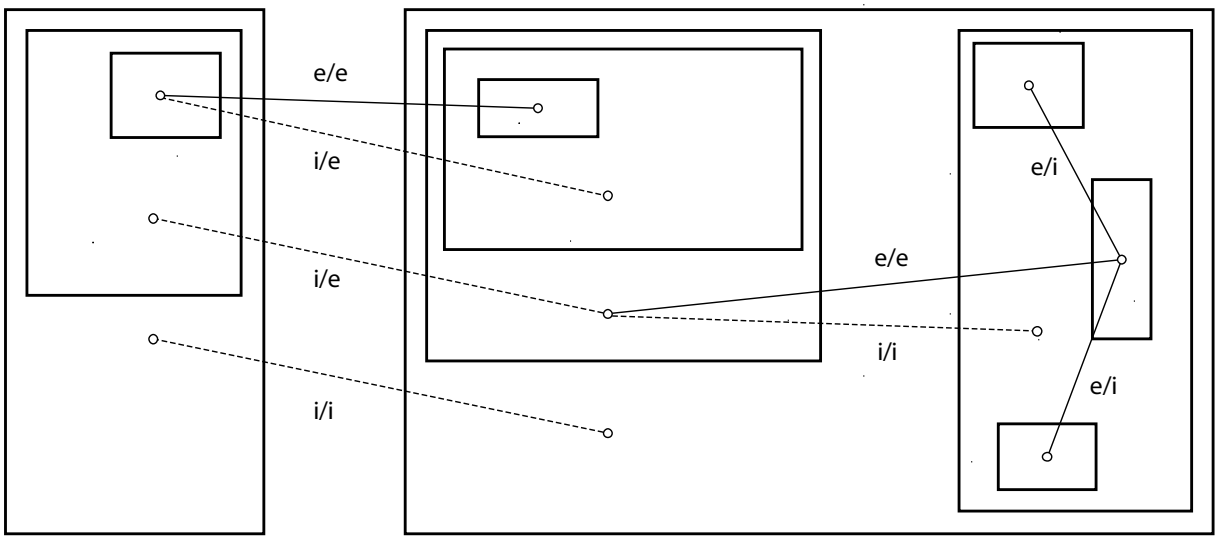


Abb. 69: Unterschiedliche Arten von Relationen in einer Problembeschreibung für ein HLP (e/i = explizit-intern, e/e = explizit-extern, i/i = implizit-intern, i/e = implizit-extern).

Wird der unter Kapitel 5 vorgestellte Layoutsolver auf HLPs angewandt, müssen verschiedene Fälle berücksichtigt werden. Für Layoutprobleme, bei denen ausschließlich interne Relationen existieren, ist der Mechanismus ohne weiteres anwendbar. Für Layoutprobleme, bei denen externe Relationen vorhanden sind, müssen diese besonders berücksichtigt werden. Die zuvor eingeführte Unterscheidung zwischen impliziten und expliziten Relationen ist ausschließlich für die Nutzerinteraktion von Bedeutung, da der Nutzer sich nur um die von ihm gesetzten expliziten Anforderungen kümmern muss und die für ihn unsichtbaren impliziten Anforderungen automatisch erzeugt werden. Der Layoutsolver behandelt implizite und explizite Anforderungen prinzipiell gleich. Sonderfälle stellen lediglich die externen Relationen dar. Für ein besseres Verständnis dieser Sonderfälle muss auf den Ablauf des Layoutsolvers bei HLPs eingegangen werden. Sobald ein Element andere Elemente enthält, versucht der Solver, die Unterelemente innerhalb des übergeordneten Elements anzuordnen. Existieren mehrere übergeordnete Elemente, werden diese gleichzeitig bearbeitet. Es laufen dann mehrere Optimierungsprozesse parallel ab. Das HLP wird demzufolge aufgeteilt in mehrere NHLP. Beispielhaft wird dies in Abb. 70 und Abb. 71 deutlich. Das HLP aus Abb. 70 wird in Abb. 71 in zwei Unterprobleme aufgeteilt: die Anordnung von $R_{1.1}$, $R_{1.2}$ und $R_{1.3}$ in R_1 und die Anordnung von $R_{1.2.1}$, $R_{1.2.2}$ und $R_{1.2.3}$ in $R_{1.2}$.

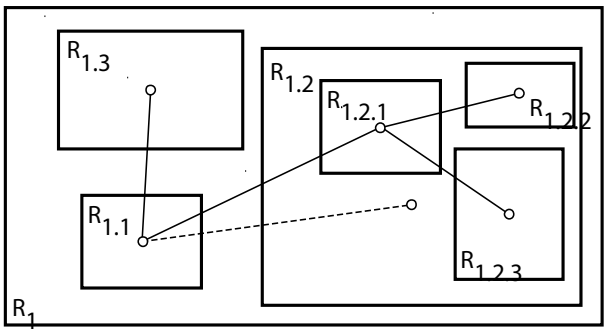


Abb. 70: Implizite und Explizite Relationen in einem einfachen Beispiellayout.

Hinsichtlich des Erstellens dichter Packungen (vgl. Kapitel 5) sind beide in Abb. 71 dargestellte Probleme (R_1 und $R_{1.2}$) getrennt voneinander lösbar. Das Herstellen einer dichten Packung in R_1 kann unabhängig vom Herstellen der dichten Packung in $R_{1.2}$ stattfinden, da es für die Unterelemente $R_{1.2.1}$, $R_{1.2.2}$ und $R_{1.2.3}$ lediglich bedeutet, dass sich die Höhe h und Breite b des übergeordneten Elements $R_{1.2}$ verändern kann, innerhalb dessen sie zu packen sind.

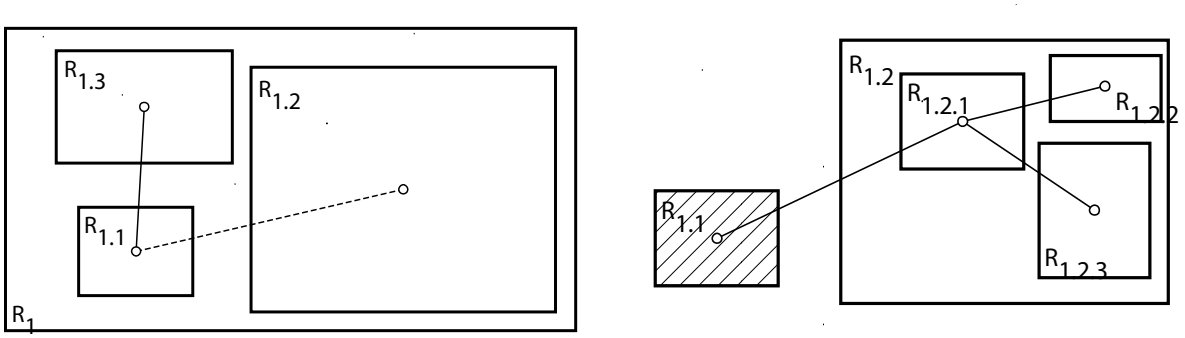


Abb. 71: Unterschiedliche Verknüpfungsarten auf unterschiedlichen Hierarchiestufen. Links: das Teilproblem $R1$, Rechts: das Teilproblem $R1.2$.

Hinsichtlich der zu berücksichtigenden topologischen Beziehungen ist zwischen internen und externen Relationen zu unterscheiden. Während interne Relationen wie unter Punkt 5.2 beschrieben behandelt werden, müssen externe Relationen gesondert berücksichtigt werden. Zunächst ist es wichtig, dass sich externe Relationen immer nur in eine Richtung auswirken, und zwar von der tieferliegenden zur höherliegenden Hierarchieebene. Das hängt zum einen damit zusammen, dass durch eine implizite Verknüpfung (von $R_{1.1}$ und $R_{1.2}$) die Anforderung für eine entsprechende Benachbarung (von $R_{1.1}$ und $R_{1.2.1}$) hinreichend gegeben ist. Zum anderen werden durch die einseitige Wirkrichtung potentielle Rückkoppelungen zwischen den Teilproblemen vermieden. Für die Lösung von R_1 müssen also lediglich die topologischen Beziehungen zwischen $R_{1.1}$ und $R_{1.2}$, sowie zwischen $R_{1.1}$ und $R_{1.3}$ berücksichtigt werden. Für die Lösung von $R_{1.2}$ hingegen ist zusätzlich zu den internen Relationen die externe Relation zu $R_{1.1}$ zu berücksichtigen. Das externe Element $R_{1.1}$ darf bei der Lösung von $R_{1.2}$ nicht in Lage und Größe verändert werden. Die Größenänderung von $R_{1.1}$ findet in R_1 statt. Der Layoutsolver für $R_{1.2}$ behandelt $R_{1.1}$ nur passiv, und zwar insofern, dass er versucht, das Element $R_{1.2.1}$ so nah wie möglich zu $R_{1.1}$ zu platzieren. Für die Berechnung des Abstandes zwischen den Elementen, die durch eine externe Relation verbunden sind, muss die Summe der Distanzen des übergeordneten Elements zu den externen Elementen von der Summe der Distanzen aller zu berücksichtigenden Elemente abgezogen werden. Für unser Beispiel bedeutet dies, dass für die Berechnung der Distanz zwischen $R_{1.2.1}$ und $R_{1.1}$ die Distanz zwischen $R_{1.2}$ und $R_{1.1}$ abzuziehen ist.

10.3. Performancevergleich von HLP vs. NHLP

In diesem Abschnitt werden HLP und NHLP hinsichtlich ihrer Eigenschaften bei der Lösung eines vorgegebenen Testszenarios miteinander verglichen. Dieses besteht in der Aufgabe, ein Layout für ein Wohngebäude mit drei Wohneinheiten zu generieren (Abb. 72). Für die Formulierung eines HLP sind auf der ersten Hierarchieebene die drei Wohnungen und ein Hausflur anzuordnen. Auf der zweiten Hierarchieebene müssen innerhalb jeder Wohnung die einzelnen Zimmer angeordnet werden. Die internen Relationen auf Hierarchieebene der Wohnung sind jeweils sternförmig ausgeprägt. Von den jeweiligen Wohnungsfluren ist eine externe Relation zum Hausgang erforderlich. Diese erzeugen implizite interne Relationen von den Wohnungen zum Hausgang (Abb. 72, links). Die Formulierung des NHLPs erfolgt anhand der Relationen zwischen den einzelnen Räumen der Wohnungen und des Hausgangs auf einer einzigen Ebene. Folglich sind ausschließlich interne Relationen erforderlich (Abb. 72, rechts).

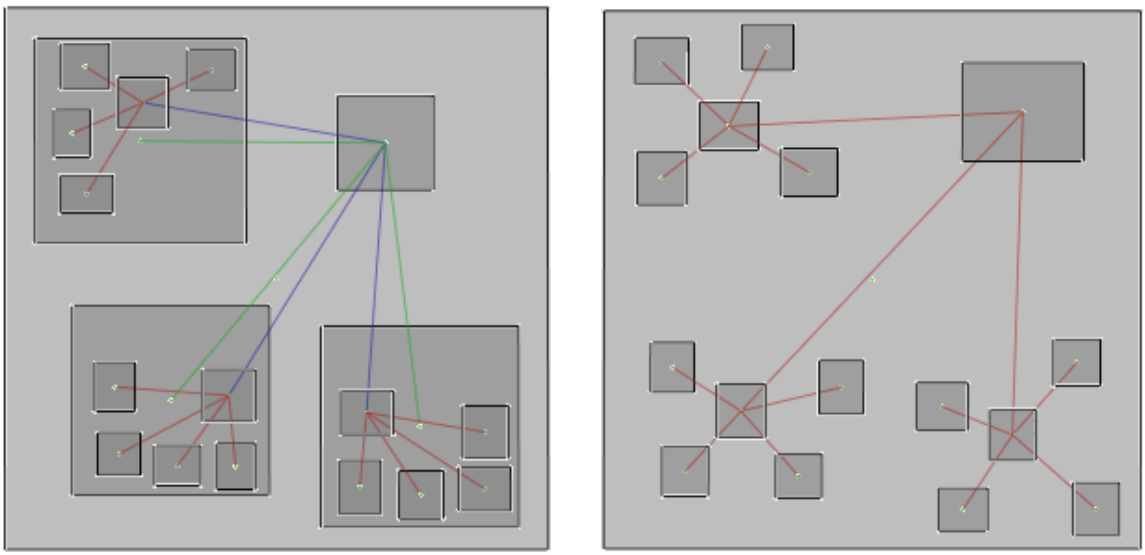


Abb. 72: Schematische Darstellung der Testszenarien. Links HLP (rot = interne, blau = externe, grün = implizite Relationen) und rechts NHLP.

Der Fokus des Vergleichs zwischen den beiden Testszenarien (HLP vs. NHLP) liegt auf der Konvergenzgeschwindigkeit, mit der ein System brauchbare Lösungsvorschläge findet. Diese ist entscheidend, um eine angemessene Interaktion des Nutzers mit dem System zu gewährleisten. Zur Definition dessen, was man unter einer brauchbaren Layoutlösung verstehen kann, werden die Fitnesswerte für f_l (Überlap-

pungsflächen) und f_2 (Distanzen) auf das Intervall [0, 1] normiert. Diese Normierung erfolgt, indem wir für f_1 und f_2 Maximalwerte annehmen, welche zu einer Skalierung der Fitnesswerte führen, die primär den Wertebereich brauchbarer Lösungen abdeckt. Für die folgenden Analysen und die Darstellung in Abb. 73 wurden folgende Skalierungsfunktionen verwendet:

$$f_1(\boldsymbol{X})_{norm} = f_1(\boldsymbol{X}) / (b_p * h_p * 1/3)$$
$$f_2(\boldsymbol{X})_{norm} = f_2(\boldsymbol{X}) / N_{rel}(b_p * h_p * 1/10) \quad (30)$$

wobei b_p und h_p die Breite und Höhe des übergeordneten Rechtecks angeben und N_{rel} die Anzahl an Relationen in einem Layoutproblem bezeichnet. Anhand dieser Normierung lässt sich nun festlegen, dass Layoutlösungen als brauchbar bezeichnet werden können, sobald sie bestimmte Werte unterschreiten, die wir folgendermaßen festlegen:

$$f_1(\boldsymbol{X})_{norm}^{accept} > 0{,}3; \qquad f_2(\boldsymbol{X})_{norm}^{accept} > 0{,}1 \quad (31)$$

Die Konvergenzgeschwindigkeit, mit der das System brauchbare Lösungen liefert, wird zu Vergleichszwecken in Generationen des L-MOES angegeben. Für eine angenehme Nutzerinteraktion (echtzeit-nahes Feedback) sollten die unter (31) angegebenen Werte nach spätestens t=20 Generationen erreicht werden. Mit in etwa der gleichen Geschwindigkeit sollte das System auf die Interaktionen eines Nutzers reagieren, die als Störungen eines einmal gefundenen optimierten Layouts aufgefasst werden kann.

Mit den soweit getroffenen Definitionen für die Brauchbarkeit von Lösungen und die Geschwindigkeit, mit der diese generiert werden müssen, können wir nun die Analysediagramme in Abb. 73 auswerten. Die Diagramme stellen Messwerte dar, die während 100 Wiederholungen des Layoutsystems anhand der beiden Testszenarien aus Abb. 72 gesammelt wurden. In beiden Diagrammen in Abb. 73 geben die Datenpunkte die Fitnesswerte für f_1 (Überlappungen) jedes einzelnen Durchlaufs an. Die grüne, durchgezogene Linie repräsentiert die Mittelwerte für f_1 und die blaue, gepunktete Linie die Mittelwerte für f_2 nach 100 wiederholten Durchläufen.

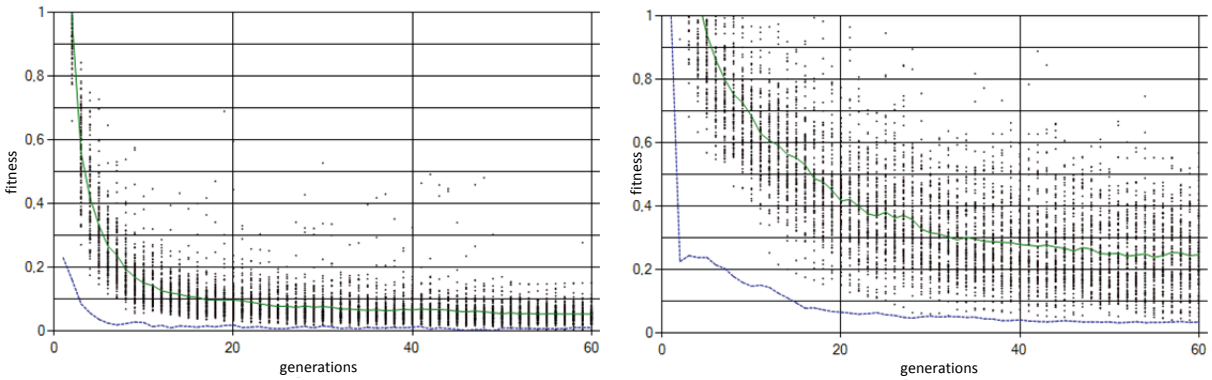


Abb. 73: Analysediagramme für die Layout-Generierung mittels L-MOEA nach je 100 Durchläufen des Programms mit jeweils t=60 Generationen. Die durchgezogenen, grünen Linien stellen die Mittelwertkurven für die Überlappungsflächen $f1$ und die gestrichelten, blauen Linien die Mittelwertkurven für die Summen der Distanzen $f2$ dar. Ausgewertet wurde links das HLP und rechts das NHLP (vgl. Abb. 72).

Im Vergleich der beiden Diagramme in Abb. 73 lässt sich gut erkennen, dass bei HLP (linkes Diagramm) die Mittelwertkurve sowohl für f_1 als auch für f_2 steiler abfällt und niedrigere Werte erreicht. Entsprechend der vorangegangenen Definitionen bedeutet dies, dass mittels HLP in einer für eine angemessene Nutzerinteraktion akzeptablen Geschwindigkeit gute Lösungen generiert werden können. Dagegen verlaufen die beiden Mittelwertkurven bei NHLP (Abb. 73, rechtes Diagramm) wesentlich flacher und erreichen zu keinem Zeitpunkt ähnlich niedrige Werte wie bei HLP. Es kann festgestellt werden, dass für f_1 im Mittel erst nach ca. t=40 Generationen brauchbare Lösungen gefunden werden, was für eine angemessene Nutzerinteraktion zu langsam ist. Ferner weist die Verteilung der Datenpunkte bei NHLP eine wesentlich größere Streuung auf, woraus sich erkennen lässt, dass das System des Öfteren keine brauchbaren Lösungen in der betrachteten Generationenspanne findet. Bei beiden Diagrammen in Abb. 73 wird ersichtlich, dass die Mittelwertkurve für f_2 (Distanzen) schnell gute Ergebnisse liefert, was auf die Regeln (20) und (21) der L-MOES zurückzuführen ist, welche die Optimierung von f_2 bevorzugt behandeln.

Um einen Eindruck der Layoutergebnisse zu vermitteln, die sich hinter den diagrammatischen Darstellungen in Abb. 73 verbergen, zeigt Abb. 74 die geometrischen Lösungen für ein HLP (links) und ein NHLP (rechts), die ausgehend von den Testszenarien aus Abb. 72 nach t=60 Generationen mittels L-MOES generiert wurden. Die Auswirkungen des Einsatzes der verschiedenen Strategien sind daran zu

erkennen, dass sich beim HLP (links) die Wohnungen innerhalb der übergeordneten Rechtecke organisieren, die durch die Hierarchisierung des Problems eingefügt wurden. Dagegen sind die Räume beim NHLP an keine übergeordneten Begrenzungen gebunden, wodurch es zu einer räumlichen Verschränkung der drei Wohnungen kommt.

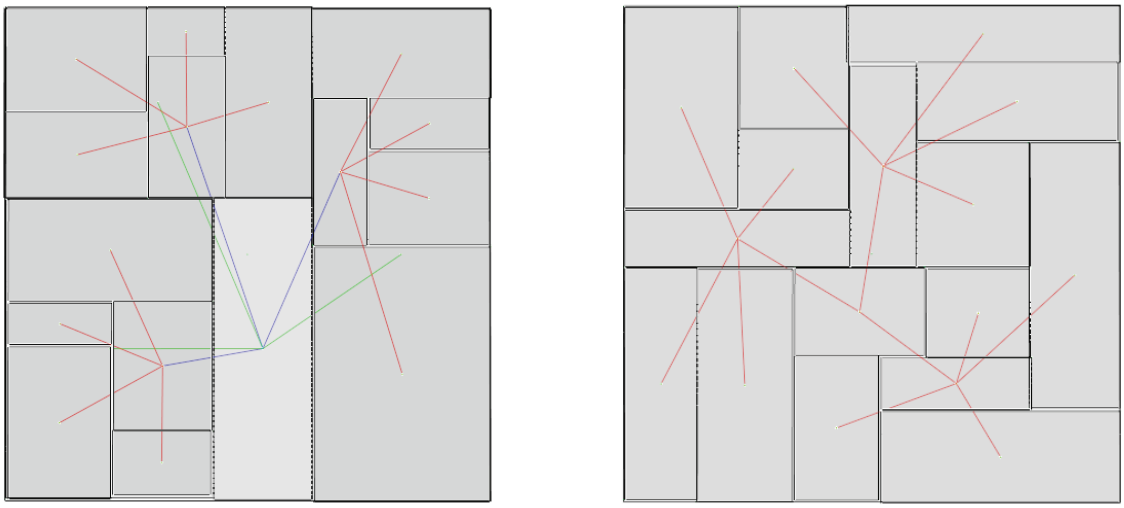


Abb. 74: Typische Lösungen bei der Berechnung eines HLP (links) und eines NHLP (rechts) mittels L-MOEA nach t=60 Generationen. Beide Lösungen unterschreiten die Anforderungen nach (31).

10.4. Konklusion und Ausblick

Abschließend lassen sich die Vor- und Nachteile der in diesem Abschnitt verglichenen Strategien (NHLP und HLP) wie folgt zusammenfassen. Die wesentliche Schwäche bei NHLP liegt in der langsamen Generierung brauchbarer Lösungen. Diese Eigenschaft wird insbesondere dann problematisch, wenn eine große Menge an Elementen mit vielen Relationen angeordnet werden muss. Der wichtigste Vorteil bei NHLP besteht darin, dass keine a priori Hierarchisierung der anzuordnenden Elemente vorgenommen werden muss. Dadurch entstehen Lösungen, welche von bestimmten Konventionen unberührt bleiben, die bei der Angabe von Hierarchien vorgegeben werden. Eine solche Konvention stellt bei dem hier dargestellten Beispiel die Zusammenfassung von Räumen einer Wohnung innerhalb eines übergeordneten rechteckigen Elements dar (Abb. 74).

Bezugnehmend auf das in diesem Beitrag vorgestellte interaktive Layoutsystem bergen HLP maßgeblich zwei Vorteile. Erstens entsteht hinsichtlich der Nutzerinteraktion durch die Hierarchisierung eine bessere Übersicht über die zu bearbeitenden

Elemente. Dies ist vorteilhaft, da ein Nutzer in der Regel nur eine begrenzte Anzahl an Elementen gleichzeitig berücksichtigen kann (Miller, 1956). Zweitens können, verglichen mit NHLP, komplexe Probleme mit vielen Elementen und Relationen wesentlich schneller gelöst werden. Mit Verweis auf die Forderung in Abschnitt 3.2.3, dass ein System möglichst explorativ sein soll, kann bei HLP festgestellt werden, dass durch die a priori Festlegung der Hierarchien bestimmte Lösungen ausgeschlossen werden. Folglich wird das Spektrum gleichwertiger Kompromisslösungen bei HLP eingeschränkt, indem der Such- und Lösungsraum verkleinert wird. Der kleinere Suchraum bei HLP erklärt zum Teil auch die höhere Konvergenzgeschwindigkeit bei HLP.

11. Darstellung und Nutzereingabe[16]

Sven Schneider, Reinhard König

Die automatisierte Lösung von Entwurfsaufgaben birgt für das Entwerfen zwei grundlegende Probleme: Zum einen die in Kapitel 1.2 beschriebene Schwierigkeit, nicht-operationale auf operationale Fragen zu reduzieren, um sie formal beschreibbar zu machen. Zum anderen, das sich die Probleme während des Entwurfsprozesses verändern, sich oft sogar erst entwickeln. Daher ist die Einbeziehung menschlicher Fähigkeiten ein entscheidender Faktor für ein Layoutsystem.

Vergewissern wir uns noch einmal des Zusammenspiels zwischen Entwerfer und Werkzeug (vgl. Punkt 3.2), so ist es wichtig, dass der Entwerfer das, was er mithilfe der Werkzeuge geschaffen hat, adäquat bewerten bzw., wenn nötig, weiterentwickeln kann. Im Falle eines generativen Entwurfssystems ist hierfür eine geeignete Schnittstelle (Interface) zu entwickeln. Die Überlegungen, die bei der Entwicklung des im Projekt implementierten Interfaces eine besondere Rolle gespielt haben, werden im Folgenden dargestellt. Zur Demonstration der Methoden wurde ein Softwaremuster implementiert, welches den in Kapitel 5 beschriebenen Layoutsolver zur Lösung von Layoutproblemen verwendet.

11.1. Grafisches Nutzerinterface

Die Interaktion eines Nutzers mit dem Entwurfssystem findet über ein grafisches Nutzerinterface statt. Dieses Nutzerinterface ermöglicht es einem Nutzer einerseits, generierte Lösungen visuell zu bewerten, andererseits die Beschreibung für das Layoutproblem zu definieren. Für die Darstellung der Lösungen muss eine geeignete Repräsentationsform gefunden werden, welche ein schnelles Erfassen aller relevanten Eigenschaften ermöglicht. Dass möglichst viele Eigenschaften einer Lösungsvariante erfassbar sind, ist insbesondere von Bedeutung, da es zum einen sehr viele Kri-

[16] Teile dieses Kapitels wurden entnommen aus dem Arbeitspapier von Schneider, S. & Koenig, R. (2011). Nutzerinteraktion bei der computergestützten Generierung von Layouts. Weimar: Bauhaus-Universität Weimar.

terien gibt, nach denen sich eine Lösung bewerten lässt. Zum anderen verwenden unterschiedliche Nutzer (Entwerfer) unterschiedliche Kriterien für die Bewertung.

Nicht alle Eigenschaften einer Lösungsvariante lassen sich innerhalb einer Darstellung und in Form einer Darstellungsmethode abbilden. Es ist folglich sinnvoll, bei der grafischen Nutzerschnittstelle mehrere, unterschiedliche Darstellungsmethoden parallel anzubieten. Damit der Nutzer eine Layoutlösung schnell in ihrer Gesamtheit bewerten kann, empfiehlt sich eine grafische Repräsentation mittels konventioneller Darstellungsmethoden, wie z.B. einer grundrisshaften Darstellung. Diagrammatische Darstellungen eignen sich insbesondere für Informationen wie jene über topologische Beziehungen zwischen Elementen. Textbasierte Ausgaben kommen bei der Darstellung numerischer Werte, z.B. der Größe von Räumen, oder alphabetischer Werte, wie z.B. der Bezeichnung von Räumen, zum Einsatz. Textbasierte Ausgaben können sowohl in tabellarischer Form dargestellt als auch mit der grafischen Ausgabe von Lösungen kombiniert werden. Abb. 75 zeigt ein Softwaremuster, in dem die verschiedenen Arten der Informationsvisualisierung implementiert wurden.

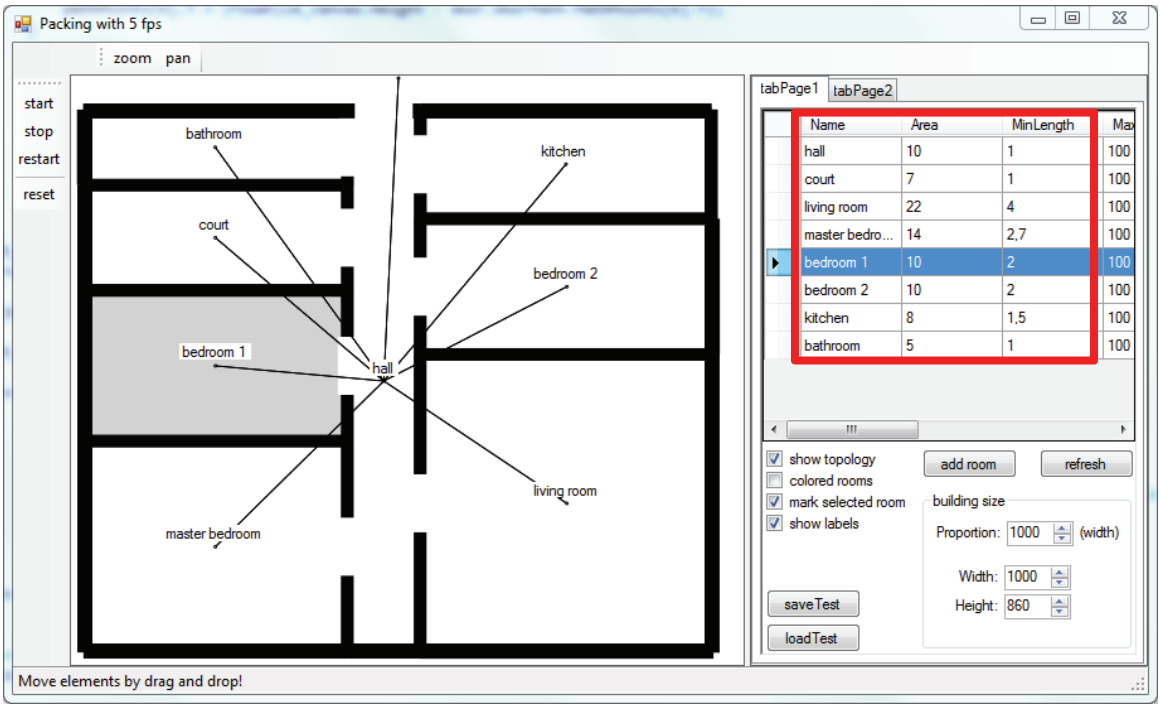


Abb. 75: Ändern der Werte der Evaluationskriterien über Tabelleneinträge (rot umrahmt).

Das System muss es dem Nutzer erlauben, zwischen den verschiedenen Darstellungsarten wechseln zu können. Dabei bleibt die Datenbasis für jede Darstellung jeweils dieselbe. Dies entspricht im Wesentlichen dem Multiple-View-Konzept, das

Rosenman und Gero (1996) für die Entwicklung von CAD Systemen vorschlagen[17]. Wichtig ist hierbei, dass die verschiedenen Darstellungsmodi dem Nutzer nicht nur Informationen zeigen, sondern dass dieser auch die Möglichkeit hat, mit diesen zu interagieren, sprich die dargestellten Parameter und Elemente zu ändern. Beispielsweise könnte ein Layout in Form eines Grundrisses sowie tabellarisch in Form eines Raumprogramms angezeigt werden. Änderungen im Raumprogramm hätten dann sofortige Auswirkungen auf den Grundriss zur Folge und vice versa.

11.2. Direkte und Indirekte Manipulation

Der generative Mechanismus liefert beständig Lösungen für ein gestelltes Layoutproblem. Bei Änderungen der Problembeschreibung reagiert das System innerhalb eines sehr kurzen Zeitintervalls (siehe Abschnitt 5.1.1). Dies ist wichtig, um dem Nutzer ein unmittelbares Feedback für die von ihm vorgenommenen Aktionen zu liefern (siehe Abschnitt 3.2.4). Da wir ein iteratives System verwenden, kann die Problembeschreibung jederzeit geändert und angepasst werden. Dies geschieht mittels verschiedener Interaktionsmöglichkeiten. Dabei kann zwischen direkten und indirekten Eingriffen unterschieden werden. Direkte Eingriffe bezeichnen die unmittelbare Manipulation der ausgegebenen grafischen Repräsentation eines Layouts. Indirekte Eingriffe bezeichnen die Manipulation der generativen Regeln bzw. Evaluationskriterien auf deren Basis Lösungen erzeugt werden.

Bei der direkten Manipulation operiert der Nutzer in einem räumlichen Bezugssystem. Das bedeutet, er nimmt Änderungen vor, indem er die Form und Lage von Elementen anhand ihrer grafischen Repräsentation manipuliert. Da dieses Eingreifen eher intuitiv und spielerisch geschieht, adressiert die direkte Manipulation im Wesentlichen die nicht-operationalisierbaren Kriterien einer Entwurfsaufgabe. Zu den Operationen, die auf dieser Ebene möglich sind, gehören das Verschieben und Skalieren von Elementen, das Hinzuzufügen und Löschen von Elementen und das Erstellen bzw. Löschen von Relationen. Diese Operationen wurden in einem Softwaremuster mittels Mausinteraktion umgesetzt (siehe Abb. 76).

[17] Heutigen BIM-Systemen (Building Information Modeling) liegt dasselbe Konzept zugrunde.

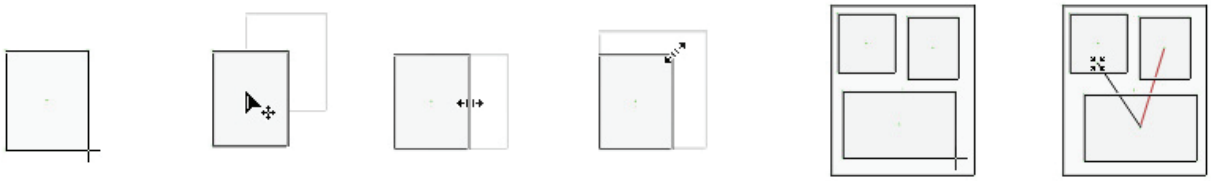


Abb. 76: Verschiedene Mausinteraktionen für ein einzelnes Element.

Hinsichtlich des Zusammenspiels der Nutzerinteraktion mit dem generativen Mechanismus müssen im Wesentlichen zwei Dinge beachtet werden. Erstens müssen das Element bzw. die Elemente, an denen zu einem bestimmten Zeitpunkt Änderungen vorgenommen werden, ihre veränderten Objektparameter auf die entsprechenden Elemente aller anderen Individuen der Population übertragen. Zweitens dürfen während der Interaktion des Nutzers keine Änderungen an den Individuen vorgenommen werden. Da das System iterativ arbeitet, würde eine Nicht-Beachtung dieser Regeln zur Folge haben, dass die Lösung, an welcher der Nutzer Änderungen vornimmt bzw. vorgenommen hat, vom Evaluationsmechanismus aussortiert wird, da sie in aller Regel zunächst eine schlechtere Fitness aufweist als die anderen Individuen. Das würde die Nutzerinteraktion erheblich beeinträchtigen und den Nutzer irritieren.

Einen weiteren, wichtigen Punkt im Kontext direkter Interaktionen stellt das Festlegen von Teillösungen dar. Da der Mensch immer nur eine begrenzte Anzahl an Elementen gleichzeitig im Blick behalten kann (Miller, 1956), ist es insbesondere bei komplexeren Problembeschreibungen wichtig, dass bestimmte Teile/Gruppen von Lösungen fixiert werden können. Solche Teillösungen können einzelne oder mehrere Elemente umfassen, die aus bestimmten Gründen unverändert bleiben sollen (z.B. Größe und Position eines Funktionstraktes in einem Gebäude). Das Fixieren von Teillösungen geschieht, indem die Objektparameter der zu fixierenden Elemente auf alle entsprechenden Individuen übertragen und für Änderungen durch den generativen Mechanismus gesperrt werden.

Bei der indirekten Manipulation greift der Nutzer nicht direkt über die grafische Ausgabe ein, sondern verändert die Regeln des generativen Mechanismus oder die Werte der Bewertungskriterien des Evaluationsmechanismus. Der Nutzer steuert dadurch die Richtung, in der das System den Lösungsraum durchsucht. Evaluationskriterien werden immer mittels numerischer Parameter definiert. Da die Änderung

dieser Parameter sehr bewusst geschieht und die Regeln für die Evaluation eindeutig beschrieben werden müssen, werden mittels indirekter Manipulation die operationalen Kriterien einer Entwurfsaufgabe definiert.

Die Eingabe bzw. Änderung dieser numerischen Werte kann über Texteingaben oder sogenannte Schieberegler stattfinden. Für Evaluationskriterien, die sich nicht mittels eines numerischen Wertes beschreiben lassen (z.B. Verteilungsdichten von Elementen), eignet sich eine grafische Definition der Zielwerte über Kurven oder Diagramme.

11.3. Zuschreiben von Bedeutungen

Form und Funktion sind die zwei wesentlichen Kategorien, nach denen die Kriterien zur Bewertung von Architektur eingeteilt werden können (Hillier, 1996). Das Aufeinander abstimmen dieser beiden Kategorien kann als ein wesentliches Ziel eines Entwurfsprozesses angesehen werden. Eine Hierarchisierung dieser Kategorien (das eine folgt dem anderen) ist dabei nicht möglich. Im Entwurfsprozess bedeutet das einerseits, dass Annahmen über die Form gemacht werden müssen, um funktionale Kriterien überprüfen zu können. Andererseits müssen, um Formen evaluieren zu können, Annahmen über die Funktion getroffen werden. Übertragen wir die Unterscheidung zwischen Form und Funktion auf den Entwurf von Layouts mit der unter Punkt 3.4.1 genannten Definition, so bezeichnen die geometrischen Attribute der Elemente (Höhe, Breite, Position und relative Lage) die Form. Die Zuschreibung einer Bedeutung für ein Element, also das, was ein Element repräsentiert, entscheidet darüber, wie der Nutzer dessen Funktion im jeweiligen Kontext beurteilt. Beispielsweise sind in Abb. 77 drei Layouts abgebildet, die sich hinsichtlich ihrer Form nicht unterscheiden, jedoch aufgrund der unterschiedlichen Bedeutung der Einzelelemente etwas Unterschiedliches repräsentieren und demensprechend auch anders bewertet werden.

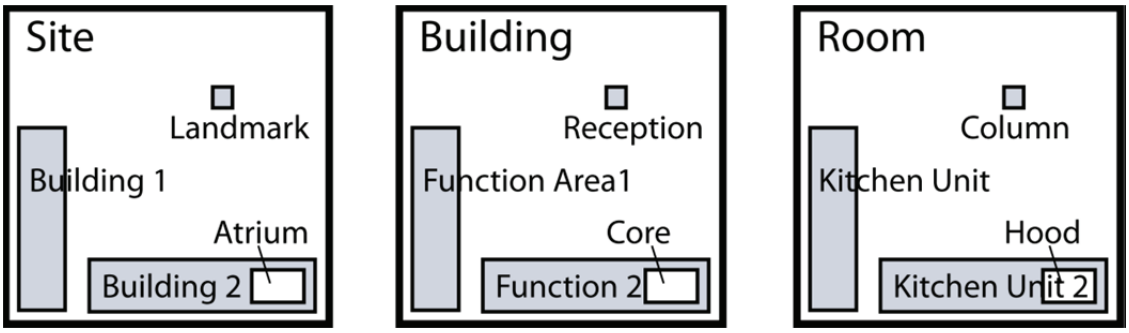


Abb. 77: Gleiche Repräsentation, unterschiedliche Interpretation.

Um ein CALD-System für möglichst viele Anwendungsfälle nutzen zu können, ist es wichtig, dass dem Nutzer die Möglichkeit geboten wird, das Dargestellte frei zu interpretieren. Der Nutzer muss den verwendeten Elementen selbst Bedeutungen zuschreiben können bzw. diese Bedeutungen auch während des Entwurfsprozesses ändern können. So muss sich der Nutzer des Systems nicht im Vorfeld auf vordefinierte Elemente festlegen, sondern kann diese je nach Situation anpassen. In diesem Zusammenhang sei auf den bereits mehrfach zitierten Artikel Donald Schöns verwiesen, welcher darauf hinweist, dass ein Entwurfssystem dem Nutzer ermöglichen muss, sich seine eigene *"Designworld"* konstruieren zu können (Schön, 1992).

11.4. Kontextsensitive Darstellung

Aufgrund der Möglichkeit, einem Element verschiedene Bedeutungen zuweisen zu können, ergeben sich weitere Fragestellungen bezüglich einer kontextsensitiven Darstellung. Die Darstellung des Kontextes erleichtert bzw. ermöglicht überhaupt erst die Bewertung eines Lösungsvorschlages, denn der Kontext, so Guski (2000), beeinflusst die Interpretation. Dies lässt sich eindrücklich an einem einfachen Beispiel (Abb. 78) zeigen: *„Wenn wir z.B. einen Text lesen, werden wir eher Buchstaben erwarten als Zahlen, und wenn eine Reizkonfiguration gleichermaßen als Zahl wie als Buchstabe interpretiert werden kann, dann wird sie als Buchstabe interpretiert"* (Guski, 2000, p. 69).

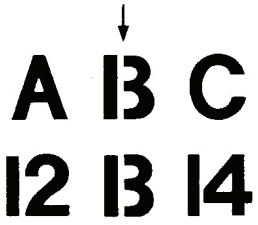


Abb. 78: Der Kontext beeinflusst die Interpretation. Abbildung aus (Guski, 2000).

Gleiches trifft auf die Bewertung von Layouts zu. Für ein einzelnes Gebäude ist es beispielsweise notwendig, das städtebauliche Umfeld, in dem es sich befindet, darzustellen. Dies ist zum einen wichtig, um die Form (den Umriss) des Gebäudes besser beurteilen zu können, zum anderen ermöglicht es kontextbezogene Aussagen über die innere Organisation des Gebäudes, wie z.B. die Platzierung des Haupteinganges oder die Orientierung lärmempfindlicher Räume.

Für die Visualisierung von Layoutlösungen im vorgestellten CALD-System ist daher eine kontextsensitive Darstellung sinnvoll. Diese kann in Abhängigkeit vom Kontext und der Bedeutung der Elemente die Darstellung des Layouts anpassen. Ein Beispiel hierfür ist die bisher im System implementierte Methode zur Anpassung der Detaillierungstiefe in Abhängigkeit von der Zoomstufe (siehe Abb. 79). Diese Art der kontextsensitiven Darstellung wird auch als semantischer Zoom bezeichnet.

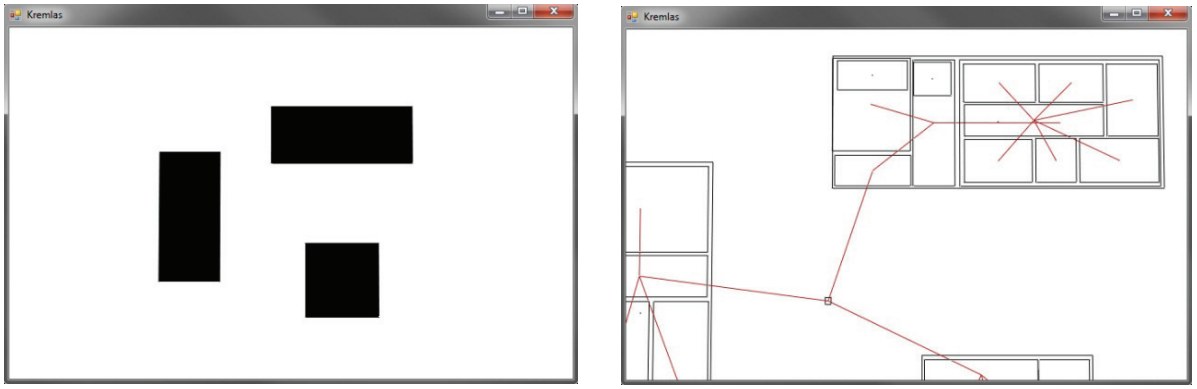


Abb. 79: Darstellung eines Layouts in Abhängigkeit vom betrachteten Bildausschnitt bzw. der Zoomstufe (Screenshot eines Softwaremusters).

Was bei der kontextsensitiven Darstellung des hier vorgestellte CALD-System als problematisch angesehen werden kann, ist, dass sich das, was der Nutzer in einem bestimmten Kontext betrachten will, situationsabhängig verändern kann. Dieser Gedanke kann anhand des folgenden Beispielszenarios verdeutlicht werden: der Generierung eines Hauses in einem städtischen Kontext. Ein Nutzer könnte einerseits nur die äußere Form des Gebäudes, andererseits aber auch den Grundriss des Gebäudes im Kontext des städtebaulichen Umfeldes betrachten wollen. Die Entscheidung, was wann in welchem Kontext visualisiert werden soll, muss vom Nutzer bestimmt werden können. Infolgedessen scheint eine vollständig automatisierte Kontextsensitivität der Darstellung nicht möglich zu sein.

11.5. Konklusion und Ausblick

Generative Systeme bieten ein großes Potential zur Unterstützung von Entwurfsprozessen. Dabei muss jedoch beachtet werden, dass eine algorithmische Lösung von Problemen nur möglich ist, wenn diese operationalisierbar sind. Entwurfsprobleme sind in aller Regel nicht operationalisierbar und entziehen sich einer vollständigen formalen Beschreibung. Die Lösung solcher Probleme ist abhängig von subjektiven und kontextabhängigen Faktoren. Um generative Systeme dennoch zur Unterstützung von Entwurfsprozessen einsetzen zu können, ist die Einbeziehung menschlicher Fähigkeiten in den computerbasierten Problemlösungsprozess notwendig. Die Benutzeroberfläche stellt hierbei die Schnittstelle zwischen generativem System und Nutzer dar. Im vorliegenden Kapitel wurden die zwei wesentlichen Aspekte beschrieben, die bei ihrer Gestaltung berücksichtigt werden müssen. Es handelt sich dabei zum einen um eine kontextsensitive Darstellung von Lösungen anhand verschiedener Arten der Informationsvisualisierung, zum anderen um eine adäquate Gestaltung von direkten und indirekten Interaktionsmöglichkeiten.

Die in diesem Kapitel dargestellten Überlegungen beleuchten nur einen kleinen Ausschnitt aus dem Gebiet der Nutzerinteraktion und Datenvisualisierung. Es wurden hauptsächlich Fragestellungen betrachtet, die sich mit dem Einbinden eines CALD-Systems in den Entwurfsprozess befassen. Fragen zur Ergonomie, also der optimierten Nutzereingabe, wurden z.B. nicht behandelt. Eine vielversprechende Möglichkeit zur Weiterentwicklung des vorgestellten Systems wäre die Anbindung an ein haptisches Interface (Knecht, 2011; Schubert, Artinger, Petzold, & Klinker, 2011).

12. Visibility-Based Floorplan Design

Automatische Generierung von Grundrisslayouts unter Berücksichtigung von Sichtfeldeigenschaften[18]

Sven Schneider, Reinhard König

Anhand von Sichtfeldern ist es möglich, wichtige verhaltensrelevante Dimensionen der gebauten Umwelt quantitativ zu erfassen. In diesem Kapitel wird untersucht, inwieweit sich Sichtfeldeigenschaften zur Generierung von Layouts verwenden lassen. Es werden zwei unterschiedliche Ansätze verfolgt. Im ersten Ansatz wird der im Kapitel 5 vorgestellte Layoutsolver um das Evaluationskriterium globaler Sichtfeldeigenschaften erweitert. Im zweiten Ansatz wird versucht, Layouts auf Grundlage bestimmter lokaler Sichtfeldeigenschaften zu erzeugen. Beide Ansätze werden miteinander verglichen. Darüber hinaus werden Ansatzpunkte für eine zukünftige Weiterentwicklung der vorgestellten Ansätze diskutiert.

12.1. Einleitung

Kriterien, die sich auf die Wirkung des Raumes auf den Menschen beziehen, finden bei den bisher entwickelten generativen Methoden zur Erzeugung von Layouts (Kapitel 5 bis 8) nur wenig Beachtung. Dabei spielen diese Kriterien für die Bewertung der Qualität eines Layouts eine zentrale Rolle. Um solche wirkungsrelevanten Kriterien in einen Layoutsolver einbinden zu können, müssen sich diese quantitativ beschreiben lassen. Eine Methodik, die es erlaubt, die Wirkung von Räumen umfassend und eindeutig zu quantifizieren, existiert bislang nicht. Jedoch existieren Methoden, mithilfe derer sich zumindest einige wirkungsbezogene Aspekte räumlicher Konfigurationen erfassen lassen. Dazu gehören beispielsweise die Sichtfeldanalysen (Isovist-Analysis).

[18] Dieses Kapitel basiert auf dem Artikel von Schneider, S. & Koenig, R. (2011). Visibility-based Floor Plan Design – The Automatic Generation of Floor Plans based on Isovist Properties. Paper presented at the Symposium Spatial Cognition for Architectural Design (SCAD 11).

Die Untersuchung, die in diesem Kapitel dokumentiert ist, baut auf einem Konzept von Benedikt (1979) auf, räumliche Konfiguration auf Basis von Sichtfeldern zu generieren. Dieses Konzept wird von Benedikt folgendermaßen formuliert: *"One might well ask: when is it possible, given one or more isovist fields (…) to (re)generate E [the spatial configuration] as a whole? (…) a direction seems clear: to design environments not by the initial specification of real surfaces but by specification of the desired (potential) experience in space (…)"* (Benedikt, 1979, S.63).

Im folgenden Abschnitt 12.2 werden die Methoden zur Berechnung von Sichtfeldern genauer erläutert. Ansätze, die solche Analysemethoden in den Generierungsprozess einbinden, werden im dritten Abschnitt 12.3 vorgestellt.

12.2. Quantifizieren visueller Eigenschaften von Raum mittels Isovists & Isovist Fields

Räume werden von den sich in ihnen aufhaltenden Menschen durch ihre Sinne wahrgenommen. Ein Großteil räumlicher Eigenschaften wird über den Sehsinn erfasst. Diese Eigenschaften werden im Folgenden als visuelle Eigenschaften bezeichnet. Sie werden maßgeblich durch zwei Faktoren beeinflusst: Einerseits durch die Oberflächenbeschaffenheit der Begrenzungen (Materialien, Texturen und Farbe) und andererseits durch die Anordnung und Größe der Begrenzungen. So regulieren Begrenzungen wie Wände und Decken die Bewegungsfreiheit und definieren, was man von einem konkreten Standpunkt aus sehen bzw. nicht sehen kann.

Eine Methode, visuelle Eigenschaften, die sich aus der Anordnung der Begrenzungen ergeben, quantitativ zu beschreiben, sind Sichtfelder (Isovists). Unter einem Isovist (auch als *Viewshed* bezeichnet) versteht man den Teil eines Raums, der von einem einzelnen Punkt aus gesehen werden kann (Benedikt, 1979). Sichtfelder sind ein *"intuitively attractive way of thinking about a spatial environment, because they provide a description of the space 'from inside', from the point of view of individuals, as they perceive it, interact with it, and move through it."* (Turner, Doxa, O'Sullivan, & Penn, 2001, S. 103).

Für die Berechnung eines Sichtfeldes wird ein bestimmter Ausschnitt (die Region D) aus dem euklidischen Raum E^3 gewählt[19] (Abb. 80, links). In dieser betrachteten Region D existieren Begrenzungen (Objekte bzw. Oberflächen, welche eine Durchsicht verhindern). Die Menge an Begrenzungen wird als Environment E bezeichnet (Abb. 80, Mitte). Sie kann mit der im Einleitungsteil definierten Bezeichnung *„räumliche Konfiguration"* gleichgesetzt werden. Ein Sichtfeld Vx beschreibt die Fläche innerhalb von D, die von einem bestimmten Blickpunkt x aus gesehen werden nzbildung in ein einzelnes Kriterium überführt, wie beispielsweise im Weighted riss ∂V des Sichtfeldes Vx kann in drei verschiedene Bestandteile untergliedert werden: (1) Sx: das Sichtfeld trifft auf begrenzende Oberflächen von E (2) Rx: die nicht physisch begrenzten Ränder des Sichtfeldes und (3) ∂Dx: das Sichtfeld trifft auf die Grenzen der Region D (siehe Abb. 80, rechts).

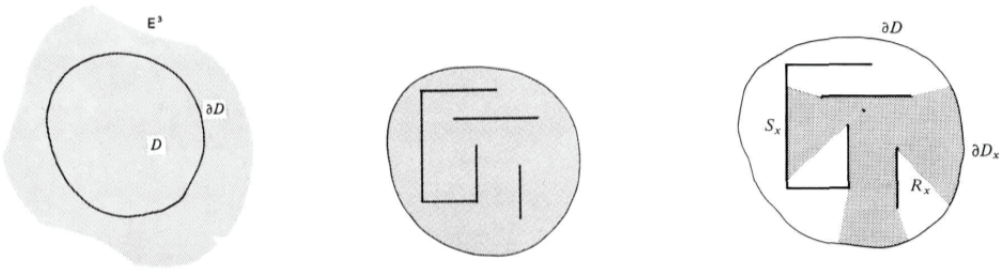


Abb. 80: Links: eine Region D im Euklidischen Raum E^3; Mitte: eine räumliche Konfiguration (Environment) E in D; Rechts: ein Sichtfeld Vx, dessen Begrenzung ∂V in die 3 Bestandteile Sx (Sichtfeld trifft auf begrenzende Oberfläche), Rx (die nicht physisch begrenzten Ränder des Sichtfeldes) und ∂Dx (Sichtfeld trifft auf die Grenzen der Region D) zerlegt werden kann. (Abbildung aus Benedikt, 1979).

Die Berechnung eines Isovists kann auf verschiedene Arten erfolgen. Benedikt (1979) beschreibt eine Berechnungsvariante basierend auf Strahlen, welche ausgehend vom Blickpunkt die zu untersuchenden Konfiguration abtasten. Das Isovist Vx wird demnach als eine Menge an Strahlen angegeben, die sich durch den Blickpunkt x und dem Schnittpunkt v' mit der Begrenzung definieren. Ausgehend von den Endpunkten und der Länge der Strahlen und den zuvor genannten unterschiedlichen Bestandteilen (Sx, Rx, Dx), lassen sich verschiedene Kennwerte ableiten, die zur Charakterisierung eines Sichtfeldes dienen. Dazu zählen unter anderem die Flä-

[19] Hierbei ist anzumerken, dass die von Benedikt angeführten Beispiele im 2-dimensionalen Raum stattfinden. In Abb. 80 müsste es also korrekterweise E^2 heißen.

che, welche beschreibt, wie viel man von einem bestimmten Standpunkt aus sieht, der Umfang, welcher in Relation zur Fläche ein Maß für die Kompaktheit des Sichtfeldes ist, *Occlusivity*, das die Summe der Länge offener, nicht von physischen Flächen begrenzte Kanten eines Isovists bezeichnet, sowie *MinRadial*, welches die kürzeste, und *MaxRadial*, welches die längste Distanz vom Blickpunkt zu E beschreibt. Auf eine ausführliche Beschreibung der für die Berechnung der Kennwerte notwendigen Formeln wird hier verzichtet. Eine sehr gute und knappe formale Darstellung findet sich bei Conroy (2001).

Die Analyse eines einzelnen Sichtfeldes erlaubt Aussagen über eine räumliche Konfiguration ausgehend von einem bestimmten Blickpunkt. Soll eine räumliche Konfiguration als Ganzes bewertet werden, ist es nötig, eine Konfiguration nicht nur von einem Punkt, sondern von möglichst allen Punkten einer Konfiguration aus zu betrachten. Hierfür schlägt Benedikt die Erstellung von Isovist-Feldern vor. Deren Berechnung wird bei Batty (2001) beschrieben. Für diese wird für eine Umgebung D ein regelmäßiges Raster erzeugt. Für jeden Punkt in diesem Raster wird ein Isovist errechnet. Die jeweiligen Messwerte der Isovisten können dann mittels Grauwerten dargestellt werden. Dunkle Bereiche bedeuteten dabei niedrige, helle hohe Werte (Abb. 81).

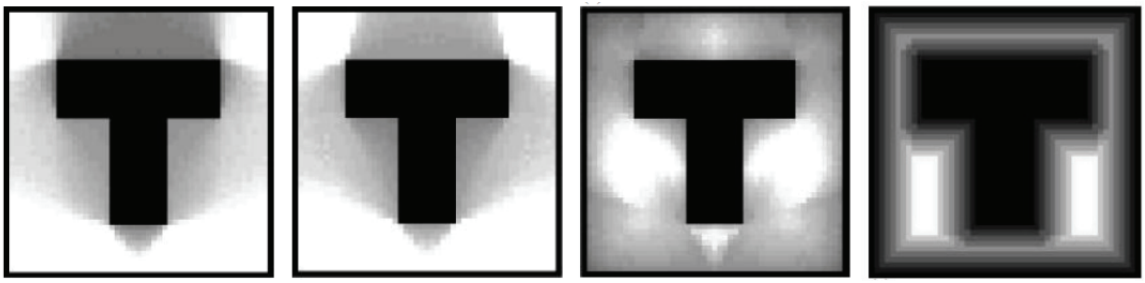


Abb. 81: Isovist fields für eine T-förmige Konfiguration, Berechnungswerte von links nach rechts: Fläche, Umfang, Kompaktheit, minimale Sehstrahlen, Abbildung aus (Michael Batty, 2001).

Aus den Isovist-Feldern lassen sich leicht Extremwerte bzw. Verläufe der oben genannten Isovisteigenschaften ablesen. So kann z.B. schnell erfasst werden, wo sich die am stärksten einsehbaren Bereiche innerhalb eines Grundrisses befinden.

Die wirkungs- bzw. verhaltensrelevanten Dimensionen von Isovist-Eigenschaften sind bisher noch nicht vollständig erforscht. Jedoch konnten in empirischen Untersuchungen verschiedene Korrelationen zwischen Isovisteigenschaften und tatsächli-

cher menschlicher Raumwahrnehmung nachgewiesen werden. So zeigen Franz und Wiener (2008) mittels VR-Experimenten, dass Isovist-Attribute wie Größe, *Jaggedness* (*Compactness*) und *Occlusivity* mit Bewertungen der Testpersonen für empfundene Schönheit, Komplexität und Geräumigkeit einer Konfiguration korrelieren. Weiterhin konnte gezeigt werden, dass die Testpersonen in der Lage sind, in einer Raumkonfiguration die Punkte mit größter bzw. kleinster Isovist-Area zu finden. Das Isovisteigenschaften relevante Informationen für das Entscheidungsverhalten bei Wegfindungsproblemen liefern, wurde beispielsweise von Conroy (2001) und Wiener et al (2011) festgestellt. Mit einer Beschreibung für das Maß an Geschlossenheit von Räumen mittels Isovisten beschäftigt sich Stamps (2005). Stamps stellt fest, dass ein Wert wie *Occlusivity* (Menge an offenen Kanten) allein nicht ausreicht, um Geschlossenheit zu beschreiben, sondern die Entfernung der Testperson zu den umschließenden Wänden eine ebenso wichtige Rolle spielt.

Wenn sich, wie oben dargestellt, aus der räumlichen Konfiguration Aussagen über ihre Wirkung treffen lassen, so sollten sich, um auf das am Anfang des Kapitels vorgestellte Konzept zurückzukommen, im Umkehrschluss aus der beabsichtigten Wirkung räumliche Konfigurationen ableiten lassen. Wie dieses Konzept umgesetzt werden kann, wird im nächsten Abschnitt anhand zweier experimenteller Ansätze zur Generierung von Grundrissen untersucht.

12.3. Generierung von Grundrissen auf Basis von Sichtfeldeigenschaften

Im Folgenden werden zwei Vorgehensweisen zur Generierung von Grundrissen unter Berücksichtigung von Sichtfeldeigenschaften untersucht: Bei der ersten wird eine auf Basis funktionaler Kriterien erzeugte Konfiguration nachträglich hinsichtlich gewünschter Sichtfeldeigenschaften optimiert. Bei der zweiten wird versucht, eine Konfiguration, die bestimmte, anfangs definierte räumliche Wirkungen aufweisen soll, direkt aus Sichtfeldeigenschaften abzuleiten. Die funktionalen Kriterien werden hier nicht explizit berücksichtigt, sondern ergeben sich im Idealfall aus der Definition der Sichtfeldeigenschaften.

12.3.1. Optimierung auf Basis von Isovist-Feld-Eigenschaften

Der in diesem Abschnitt dargestellte Ansatz ist zweistufig und basiert auf dem Layoutsolver „Dichte Packung“, der in Kapitel 5 vorgestellt wurde. Im ersten Schritt werden die Konfigurationen in ihren Grundzügen erstellt. Im zweiten Schritt werden diese innerhalb der durch den ersten Schritt vorgegebenen Grenzen hinsichtlich einer Sichtfeldeigenschaft optimiert.

Der erste Schritt besteht in der Positionierung und Dimensionierung von Rechtecken innerhalb eines vorgegebenen Rechteckes. Die Kriterien, die dafür verwendet werden, sind das Herstellen einer dichten Packung sowie das Sicherstellen von gewünschten Nachbarschaftsbeziehungen (Abb. 82). Die Prinzipien zur Lösung dieses Optimierungsproblems wurden in Kapitel 5 detailliert beschrieben.

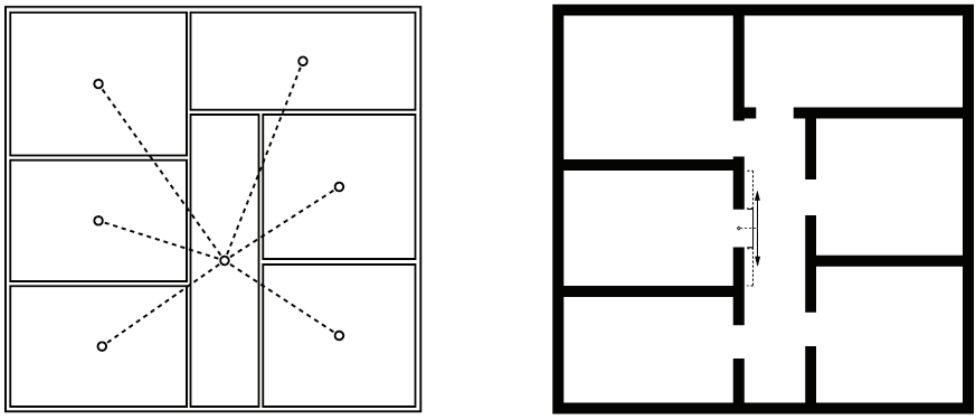


Abb. 82: Links: Typisches Resultat des Layoutsolvers „Dichte Packung“; Rechts: Darstellung eines Grundrisses, wobei die Positionen der Türen zu einem gewissen Maß variabel sind.

Aus der ersten Stufe resultieren Grundrisse, die durch eine lückenlose und überlappungsfreie Anordnung von Rechtecken innerhalb eines rechteckigen Umrisses charakterisiert sind. Dabei werden die Räume, die benachbart sein sollen, so angeordnet, dass sie sich über eine gewisse Mindestlänge berühren. Innerhalb dieser Berührungslänge werden Öffnungen mittig platziert, welche die Räume miteinander verbinden (Abb. 82). Die generierten Grundrisse werden nun als Grundlage für eine weitere Optimierung genutzt, welche als Fitnessfunktion die Maximierung des durchschnittlichen Werts aller Flächen eines Sichtbarkeitsfelds (*Mean Isovist Area*) verwendet. Diese Funktion kann formal folgendermaßen dargestellt werden:

$$f(x)\min = | tAv - \sum_{i=1}^{n} Avi | \tag{32}$$

Die Erstellung von Varianten für den Optimierungsprozess erfolgt durch die Änderung der Positionen der Türen (Abb. 82, rechts). Die geometrische Anordnung der Rechtecke wird nicht mehr verändert, sodass die Qualitäten, die während des ersten Schritts erzeugt wurden, weiterhin erhalten bleiben. Die Ergebnisse des Optimierungsprozesses der zweiten Stufe sind in Abb. 83 dargestellt, wobei in der oberen Reihe die Varianten vor der Sichtfeldoptimierung abgebildet sind und unten jene, die durch die Optimierung erzeugt wurden.

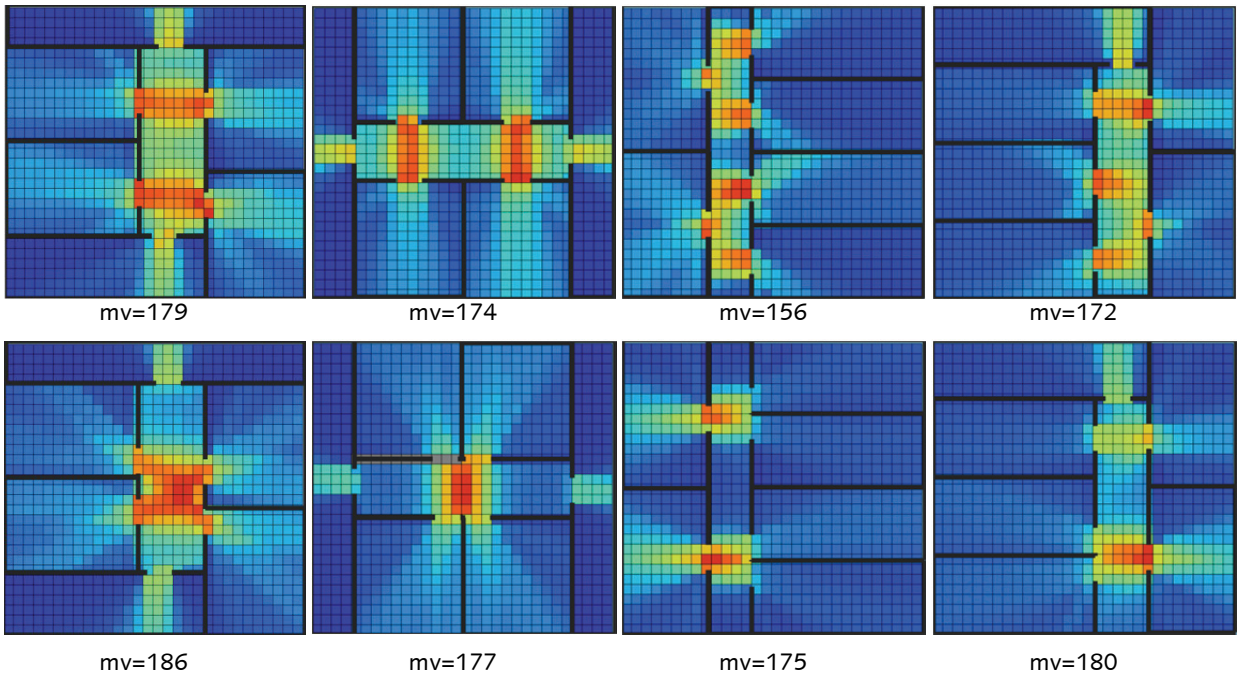


Abb. 83: Vier verschiedene Grundrisse mit je 7 Räumen und sternförmiger Topologie. Die Mean Isovist Area (mv) ist für jeden Grundriss angegeben. Die roten Felder zeigen Punkte mit hoher Sichtbarkeit, die blauen solche mit geringer Sichtbarkeit. Die Abbildung zeigt die 4 Grundrisse vor der Sichtfeldoptimierung (obere Reihe) und nach der Optimierung (untere Reihe).

12.3.2. Generierung von Layouts auf Basis einzelner Isovists

Der zweite Ansatz verwendet ein einfacheres Modell zur Generierung von Layouts als der erste Ansatz. Das Modell besteht aus einem gleichmäßigen Raster, in welchem horizontale und vertikale Linien erzeugt werden können (Abb. 84). Ein ähnlicher Ansatz wurde bereits von Krämer und Kunze (2005) zur Erzeugung von Layouts verwendet.

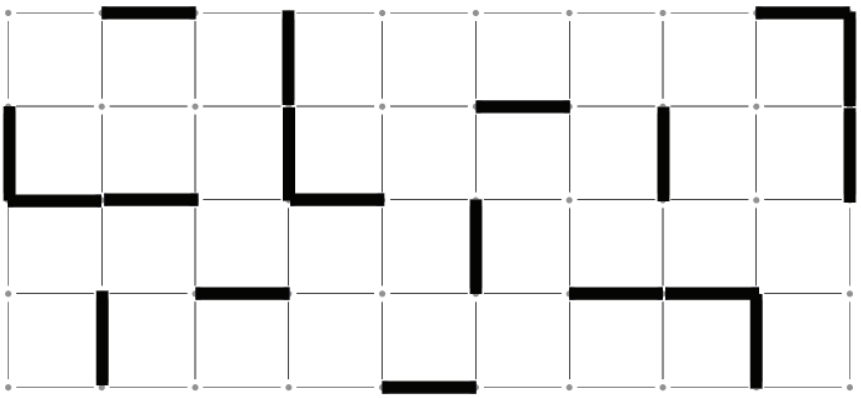


Abb. 84: Raster mit horizontalen und vertikalen Linien, welche die Wände eines Grundrisses darstellen.

Der Evaluationsmechanismus besteht in der Bewertung der Sichtfeldeigenschaften ausgehend von einem einzelnen Blickpunkt. Dieser Blickpunkt ist fix und muss vor Beginn des Optimierungsprozesses festgelegt werden. Die Fitnesskriterien, die zur Optimierung herangezogen werden, sind *Area, Perimeter, Occlusivity*. Diese sollen die Form und die Größe der Räume definieren. Durch Mutation bzw. Rekombination werden die horizontalen bzw. vertikalen Linien der Individuen so lange ein- und ausgeblendet bzw. vertauscht, bis eine den Fitnesskriterien genügende Lösung gefunden wird. Die Zielfunktion für die Optimierung der Konfiguration besteht in der Minimierung der Abweichung von den Zielwerten für Area tA, Perimeter tV und Occlusivity tQ. Die Abweichungen werden normiert und können in der Fitnessfunktion individuell gewichtet werden:

$$f1(x) = norm(|\,Ax - tA\,|) * wA + norm(|\,Vx - tV\,|) * wV + norm(|\,Qx - tQ\,|) * wQ \qquad (33)$$

wA, wV und wQ bezeichnen die Gewichtungen, mit denen die Abweichungen von den Zielwerten in die Fitness eines Individuums eingehen. In Abb. 85 sind exemplarische Lösungen dargestellt, die mittels unterschiedlicher Zielwerte und Gewichtungen entstanden sind.

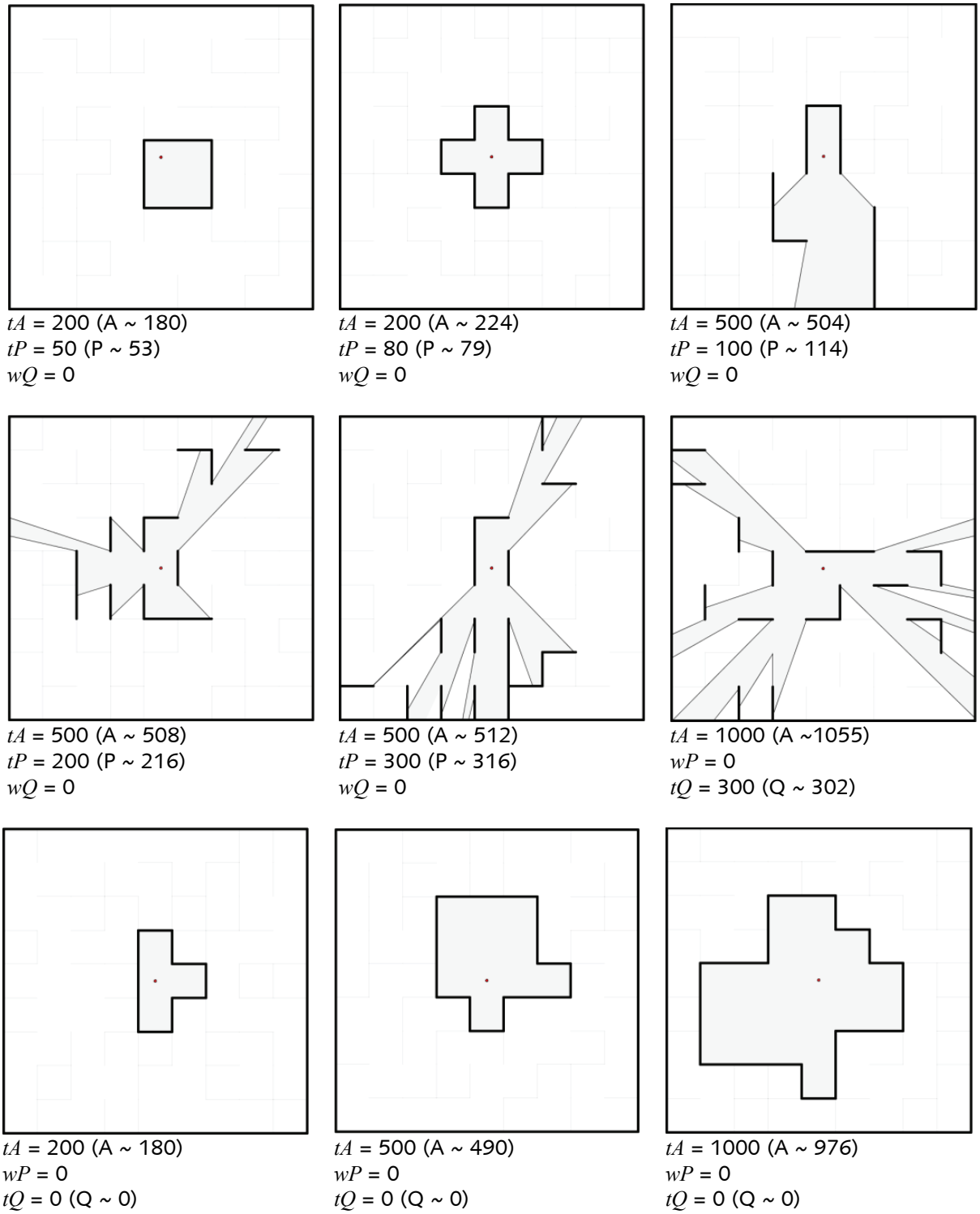


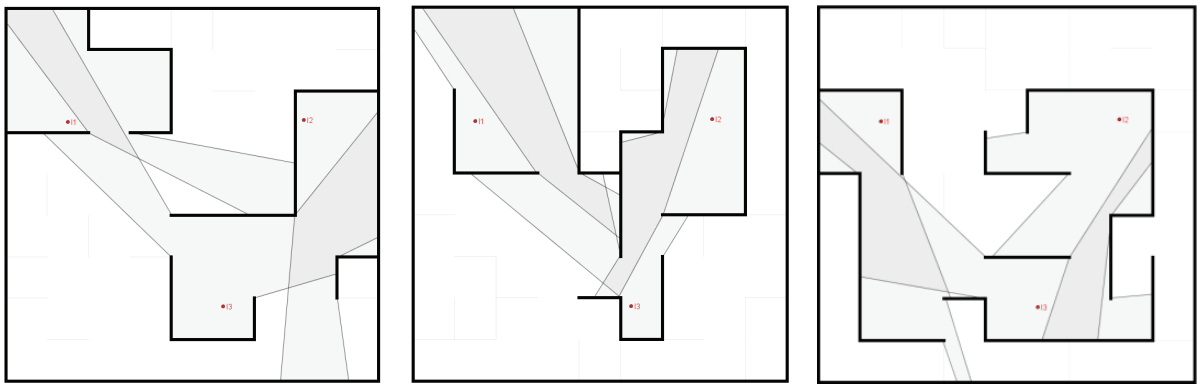
Abb. 85 9 Varianten von Konfigurationen ausgehend von einem Blickpunkt mit unterschiedlichen Zielkriterien.

Um Relationen zwischen verschiedenen Bereichen eines Grundrisses zu definieren, ist es nötig, nicht nur die Eigenschaften eines Isovisten, sondern auch die Beziehungen zwischen mehreren Isovisten zu definieren. Hierfür haben wir das System um ein weiteres Fitnesskriterium erweitert: die Überlappungsfläche mehrer Isovisten.

Ziel dieser Erweiterung ist es, zu prüfen, ob sich durch die gezielte Überlagerung von Sichtfeldern topologische Beziehungen herstellen lassen.

$$f2(x) = f1(x) + \sum_{i=1}^{n} |Ovlp(Ai, Ai+1) - tOvlp(i, j)| \tag{34}$$

Ovlp bezeichnet die Überlappungsfläche zwischen zwei Isovisten und *tOvlp* den Zielwert für die Überlappung zwischen zwei Isovisten. In einem Testszenario haben wir die Position dreier Blickpunkte *p1, p2* und *p3* definiert. Ausgehend von diesen Blickpunkten werden die dazugehörigen Isovisten *I1, I2* und *I3* erzeugt. Die Überlagerung der Isovisten *I1+I3* und *I2+I3* sollte 300 betragen. Zwischen *I1* und *I2* sollte keine Überlappung auftreten (*tOvlp(1,2)* = 0). Abb. 86 zeigt drei exemplarische Ergebnisse entsprechender Optimierungen.



Ovlp(1,2) = 0
Ovlp(1,3) = 318
Ovlp(2,3) = 388

Ovlp(1,2) = 0
Ovlp(1,3) = 332
Ovlp(2,3) = 324

Ovlp(1,2) = 0
Ovlp(1,3) = 338
Ovlp(2,3) = 291

Abb. 86 Varianten für eine Konfiguration, die durch die drei Blickpunkte *P1, P2 und P3* definiert ist. Dabei sollen sich laut Zielfunktion die Isovistpolygone von *P1* und *P2* nicht überlappen.

Auffällig ist, dass die entstehenden Räume nicht abgeschlossen sind. So ist es möglich, dass Bereiche, deren Isovisten sich nicht überlappen sollen (*I1* und *I2*), dennoch einander zugänglich sind. Die Erschließungsbeziehungen von Räumen in einem Grundriss lassen sich also nicht allein durch Überlappung der Isovisten beschreiben.

12.4. Diskussion

Vorgestellt wurden zwei verschiedene Ansätze, bei denen für die Erzeugung einer räumlichen Konfiguration Sichtfeldeigenschaften als formbeeinflussende Kriterien

verwendet wurden. Trotz der grundsätzlichen Verschiedenheit beider Ansätze lassen sich ihre Vor- und Nachteile gegenüberstellen. Der erste Ansatz optimiert die Sichtfeldeigenschaften anhand eines zweistufigen Verfahrens erst in der zweiten Stufe und behandelt diese damit im Vergleich zu den funktionalen Kriterien nachrangig. Der zweite Ansatz kann als *„Generierung von innen heraus"* bezeichnet werden, da die Anordnung der Wände vorrangig auf den Eigenschaften der Sichtfelder bestimmter Positionen im Raum basiert.

Vorteilhaft beim ersten Ansatz ist die effektive Erzeugung konventioneller Grundrisse (zellenartig, mit Türöffnungen), da für den Layoutsolver bereits Grundannahmen über den Zusammenhang von Form und Funktion getroffen wurden (ein Raum ist ein rechteckiges Element, Räume dürfen sich nicht überlappen, verbundene Räume müssen sich berühren). Hieraus resultiert jedoch gleichzeitig ein Nachteil hinsichtlich der Integration von Sichtfeldeigenschaften. So gibt es aufgrund der festgelegten Grundannahmen nur wenig Spielraum für die Formgebung durch die Isovisteigenschaften. Um Isovist-Eigenschaften zu beeinflussen, kann lediglich die Position der Türen verändert werden. Dieses Problem ist nicht zuletzt auf die Zweistufigkeit des Verfahrens zurückzuführen. Eine Einbindung von Isovist-Feld-Eigenschaften in die erste Stufe der Generierung gestaltet sich jedoch schwierig, da zunächst einmal ein bewertbarer Grundriss vorhanden sein muss, um eine Lösung hinsichtlich dieser Eigenschaften zu bewerten. Es ist nicht sinnvoll, Isovisten für überlappende Rechtecke ohne Türen zu berechnen. Ein bewertbarer Grundriss ist erst nach der ersten Stufe verfügbar.

Beim zweiten Ansatz haben die Isovisten einen direkteren Einfluss auf die Anordnung der Wände. Die Methode zur Erzeugung von Varianten arbeitet mit wenigen Grundannahmen. Die Anordnung der horizontalen bzw. vertikalen Linien entsteht ausschließlich durch Optimierung der Sichtfeldeigenschaften. Jedoch ist der Lösungsraum durch dieses einfache Modell zur Generierung von Varianten stark eingeschränkt. Das Raster, in dem die Linien platziert werden (Abb. 84), lässt darüber hinaus nur wenig Spielraum zur Variation der Isovisteigenschaften. Entweder existiert eine Linie oder nicht, es sind keine Teillängen für Linien möglich, um beispielsweise gezielt Öffnungen zu generieren. Ferner entstehen zwischen den defi-

nierten Blickpunkten Flächen ohne eindeutige Zuordnung. Dadurch ist es schwierig, funktionale Kriterien zu integrieren.

Hinsichtlich des Evaluationsmechanismus unterscheiden sich beide Ansätze durch die Art der Verwendung von Sichtfeldern bei der Bewertung der Konfigurationen. Im ersten Ansatz wird aufgrund der Verwendung von Isovist-Feldern die Konfiguration global mittels Durchschnittswerten bewertet. Dies ermöglicht eine Charakterisierung des Grundrisses als Ganzes. Nachteilig daran ist jedoch, dass die Bildung des Durchschnitts eines Isovist-Feld-Werts (in diesem Fall Area) nur wenig Aussagen über lokale Qualitäten der Konfiguration liefert. Der zweite Ansatz verwendet definierte (lokale) Standpunkte zur Bewertung einer Lösung. Dies ist einerseits als Vorteil zu betrachten, da es ein gezieltes Verorten von Sichtfeldeigenschaften ermöglicht. Andererseits kann dies als nachteilig aufgefasst werden, da die Platzierung dieser Standpunkte zu einem hohen Grad über die erzielbaren Ergebnisse entscheidet.

12.5. Konklusion und Ausblick

Die Generierung räumlicher Konfigurationen basierend auf Sichtfeldeigenschaften stellt einen interessanten Ansatz für die zukünftige Entwicklung von Entwurfsautomaten dar, da räumliche Konfigurationen hier ausgehend von ihrer Wirkung auf den Menschen entworfen werden. Die im vorliegenden Kapitel vorgestellten Ansätze stellen jedoch lediglich erste Ideenskizzen dar, wie sich diese Aufgabe bewältigen lässt.

Die vorgestellten Methoden können auf zwei Ebenen weiterentwickelt werden. Auf Ebene des generativen Mechanismus ist das Modell zur Generierung von Varianten dahingehend zu verbessern, dass eine größere Vielfalt unterschiedlicher geometrischer Konfigurationen erzeugt werden kann. Dabei ist zu berücksichtigen, dass in jedem Modell, das einem generativen Mechanismus zugrunde liegt, Konventionen darüber stecken, wie Raum geometrisch repräsentiert wird. Je mehr Konventionen in einem solchen Modell existieren (z.B. ein Raum besitzt die Form eines Rechteckes), desto weniger Möglichkeiten ergeben sich, die Form eines Raumes auf Basis von Isovisteigenschaften zu bilden. Um räumliche Konfigurationen möglichst direkt aus den Isovisteigenschaften ableiten zu können, ist es wichtig, ein Modell zur Ge-

nerierung zu verwenden, das mit relativ wenig Konventionen auskommt, welche die Form von Räumen betreffen. Das Modell des zweiten Ansatzes scheint für die Optimierung von Isovisteigenschaften geeigneter zu sein als das des ersten Ansatzes. Jedoch ist zu bemängeln, dass das zweite Modell aufgrund des verwendeten Rasters nur eine kleine Varianz hinsichtlich der erzeugbaren Geometrie besitzt. Dies ist nachteilig, da bereits kleine geometrische Änderungen große Auswirkungen auf die Eigenschaften der Sichtfelder haben können.

Bei beiden Ansätzen wird deutlich, dass auf Ebene des Evaluationsmechanismus die verwendeten Bewertungskriterien nicht ausreichen, um die Eigenschaften, die ein Grundriss hinsichtlich seiner räumlichen Wirkungen haben soll, umfassend zu beschreiben. Das größte Defizit liegt aus unserer Sicht in der ausschließlichen Verwendung von entweder lokalen oder globalen Sichtfeldeigenschaften. Da Räume von Menschen nie ausschließlich in Bewegung bzw. nie ausschließlich von einem Standpunkt aus wahrgenommen werden, lässt sich Raumwirkung weder auf globale noch auf lokale Eigenschaften reduzieren. Für den Evaluationsmechanismus sehen wir ein vielversprechendes Potential in der Kombination lokaler und globaler Eigenschaften. Einen interessanten Ansatz hierfür könnte in der Erstellung von *„Place Graphs“* liegen, wie sie Franz & Wiener (2011) vorschlagen. Dort werden die Min-Radial-Werte eines Isovist-Fields auf Hochpunkte und Grate untersucht, um aus einem zweidimensionalen Grundriss die Zentren von Räumen und deren Verbindungen untereinander zu extrahieren.

Als weitere Evaluationskriterien bieten sich die Beziehungen zwischen verschiedenen Sichtfeldern an. Diese können am besten in Form eines Graphen aus gegenseitig sichtbaren Punkten analysiert werden (Turner, et al., 2001). Aus diesem sogenannten Visibility-Graphen können Messwerte wie *Integration, Clustering Coefficient, Control* bzw. *Controllability* abgeleitet werden. Ein erster Ansatz, den Messwert Integration in einen Layoutsolver einzubinden, findet sich bei Krämer und Kunze (2005).

13. Zusammenfassung und Ausblick

Reinhard König

In diesem abschließendem Kapitel werden alle wesentlichen Arbeitsschritte und Ergebnisse der diesem Buch zugrunde liegenden Forschungsarbeit rekapituliert. Außerdem betrachten wir mögliche Implikationen der vorgestellten Methoden für eine computerbasierte Entwurfs- und Planungspraxis.

13.1. Zusammenfassung

Im einführenden Teil der vorliegenden Untersuchung wurde neben der Projektbeschreibung und der Darstellung zum Stand der Forschung im Rahmen der Konzeption eines Allgemeinen Layout-Entwurfssystems (ALES) eine allgemeine Beschreibung eines CALD-Systems geliefert (Kapitel 3). Es wurde zunächst erläutert, warum es notwendig ist, den Nutzer eines solchen Systems in den Generierungsprozess zu involvieren. Darauf aufbauend konnten als wesentliche Merkmale, die ein interaktives Layoutsystem aufweisen sollte, folgende vier identifiziert werden: Adaptivität, Zirkularität, Explorativität und Unmittelbarkeit. Darüber hinaus wurden Methoden diskutiert, um diesen Anforderungen gerecht zu werden. Daraus ergab sich die Schlussfolgerung, dass sich insbesondere evolutionäre Algorithmen für die Umsetzung eines interaktiven entwurfsunterstützenden Systems eignen.

Im Hauptteil der vorliegenden Arbeit (Kapitel 5 bis 8) wurden verschiedene Algorithmen implementiert, anhand derer Layouts automatisch generiert werden können. Das erste vorgestellte Verfahren löst Layoutprobleme mittels dichter Packung, wohingegen die drei darauffolgend beschriebenen Methoden Unterteilungsalgorithmen zur Layoutgenerierung applizieren. Allen vorgestellten Systemen gemein ist, dass sie sich Evolutionärer Algorithmen bedienen, um hinsichtlich definierter Randbedingungen wie Raumgrößen und Nachbarschaftsbeziehungen optimale Layoutstrukturen zu finden. Nicht betrachtet wurden Methoden, die mittels Constraint-Programmierung Layouts generieren (vgl. Punkt 2.2), da sich diese aus Sicht der Autoren für eine angemessene Nutzerinteraktion weniger gut eignen.

Für einen fundierten Vergleich der untersuchten generativen Verfahren wurden zwei Systeme exemplarisch gegenübergestellt (Kapitel 9). Das eine verwendet die dichte Packung und das andere einen Unterteilungsalgorithmus zur Erzeugung von Layouts. Es hat sich gezeigt, dass sich die Systeme zwar hinsichtlich Geschwindigkeit und den Möglichkeiten zur Nutzerinteraktion unterscheiden, dass aber beide zuverlässig gute Layoutlösungen für die betrachteten Problemszenarien liefern. Beide Verfahren haben folglich das Potential, zu praxistauglichen entwurfsunterstützenden Anwendungen weiterentwickelt zu werden.

Im letzten Teil der Arbeit wurden spezielle Themen zu CALD betrachtet. In Kapitel 10 erfolgte eine Auseinandersetzung mit den Bedingungen und Folgen einer hierarchischen Gliederung von Layoutproblemen. Es wird gezeigt, dass es bei hierarchisch gegliederten Layoutproblemen notwendig ist, topologische Relationen räumlicher Elemente über Hierarchiegrenzen hinweg definieren zu können. Die Vor- und Nachteile von Layoutproblemen, die hierarchisch gegliedert wurden (HLP), und denen, die auf einer Hierarchieebene abgehandelt werden (NHLP), lassen sich wie folgt zusammenfassen. Die wesentliche Schwäche bei NHLP liegt in der langsamen Generierung brauchbarer Lösungen. Der wichtigste Vorteil bei NHLP besteht darin, dass keine a priori Hierarchisierung der anzuordnenden Elemente vorgenommen werden muss. Dagegen bergen HLP die Vorteile, dass erstens hinsichtlich der Nutzerinteraktion durch die Hierarchisierung eine bessere Übersicht über die zu bearbeitenden Elemente entsteht und dass zweitens komplexe Probleme mit vielen Elementen und Relationen relativ schnell gelöst werden können.

Anschließend wurden in Kapitel 11 neben den verschiedenen Darstellungsformen räumlicher Elemente vor allem die Nutzerinteraktionsaspekte betrachtet und Anforderungen für ein interaktives generatives System formuliert. Die Einbeziehung menschlicher Fähigkeiten in den computerbasierten Problemlösungsprozess ist notwendig, um generative Systeme optimal zur Unterstützung von Entwurfsprozessen einsetzen zu können. Der wichtigste Aspekt besteht darin, dass ein Nutzer jederzeit die Möglichkeit hat, auf Grundlage der im grafischen Interface dargestellten Lösungsvariante in den Problemlösungsprozess einzugreifen. Dies kann durch das Ändern der Problembeschreibung geschehen, wobei durch direkte Interaktion ge-

ometrische Eigenschaften der Elemente manipulierbar und durch indirekte Interaktion Restriktionen für die Generierung veränderbar sein sollten.

Abschließend werden die Potentiale betrachtet, die sich aus einer Einbindung weiterer Zielfunktionen in die entwickelten Layoutsysteme ergeben (Kapitel 12). Dabei lag der Fokus darauf, Sichtfelder explizit zu berücksichtigen. Die Generierung räumlicher Konfigurationen basierend auf Sichtfeldeigenschaften stellt einen vielversprechenden Ansatz für die zukünftige Entwicklung von Entwurfsautomaten dar, da hier Raum bzw. dessen grafische Repräsentation ausgehend von seiner Wirkung auf den Menschen erzeugt wird.

13.2. Konklusion und Ausblick

In der Einleitung (Kapitel 1.1) wurde beschrieben, dass wir einen Entwurfsprozess als kreativ betrachten, wenn dessen Ergebnis, also der Entwurf, Eigenschaften aufweist, die nicht in der Aufgabenstellung enthalten waren. Diese Sichtweise entspricht der Feststellung Rittels (1992, p. 21), dass nicht-operationale (bösartige) Probleme schlecht definiert sind, sodass deren Lösung auf einer unzuverlässigen Entscheidung beruht. Die Definition des Problems beinhaltet in diesem Fall bereits wesentliche Aspekte seiner Lösung.

Die Systeme, welche in der vorliegenden Arbeit beschrieben wurden, verwenden zwei Anforderungen (Restriktionen) für die zu erzeugenden Layouts: Erstens die Größen der gewünschten Räume und zweitens deren topologische Nachbarschaftsbeziehungen (welcher Raum neben welchem liegen soll). Basierend auf diesen beiden Anforderungen können verschiedene geometrische Layoutlösungen erzeugt werden, welche zwar alle die Restriktionen erfüllen, allerdings geometrisch sehr verschieden ausfallen können. Allein das große geometrische Spektrum möglicher Lösungen, deren Ausformung nicht Teil der Anforderungen war, veranlasst uns hier, mit einem gewissen Optimismus davon auszugehen, dass die realisierten Layoutsysteme ein beachtliches kreatives Potential haben.

Dagegen ließe sich einwenden, dass die Algorithmen, mit denen die Lösungen erzeugt werden, auch als Teil der Problemdefinition betrachtet werden können, da diese beispielsweise nur rechtwinklige Layouts erzeugen können, womit ein we-

sentliches Charakteristikum der möglichen Lösungen bereits vorweggenommen wird. Dass dieser Einwand durch elaborierte Algorithmen weitgehend entkräftet werden kann, wurde im Rahmen dieser Arbeit z.B. mit der Erzeugung nicht rechtwinkliger Layoutstrukturen in Kapitel 8 gezeigt.

Die Entwurfsphilosophie, welche der in diesem Buch vorgestellten Methoden zugrunde liegt, ist vorwiegend von rationalen Überlegungen charakterisiert. Das entwerferische Vorgehen ist hier dadurch geprägt, dass man versucht, zu einem möglichst guten Entwurf zu gelangen, indem man mit funktionalen Kriterien beginnt, die man während des Entwurfsprozesses möglichst nicht aus den Augen verliert. Dem entgegen steht eine intuitive Entwurfspraxis, welche bestrebt ist, ausgehend von formalen Inspirationen zu einer interessanten Geometrie zu gelangen, welcher erst am Ende die notwendigen Funktionen so gut wie möglich zugeschrieben werden. Eine detaillierte Auseinandersetzung zu dieser Unterscheidung findet sich bei Koenig (2010, 2011a).

Im Kontext der vorliegenden Auseinandersetzung soll an dieser Stelle ein weiterführender Diskurs angestoßen werden, welcher den aktuellen Entwicklungen im Bereich des computergestützten Entwerfen und Planens Rechnung trägt. Die Verbindung von rationalen Entwurfsstrategien mit algorithmischen Methoden, wie sie in dieser Untersuchung dargestellt wurde, wird aus Sicht der Autoren am treffendsten mit dem Begriff *Computational Planning* bezeichnet. Daraus leitet sich das Forschungsfeld der *Computational Planning Science* ab, welche als Wissenschaft vom computerbasierten Planen nur unzureichend übersetzt werden kann, da der Begriff *Computational* die Synthese aus Ingenieurswissenschaften, Mathematik und Informatik beschreibt. Folglich stellt die *Computational Planning Science* den Rahmen dar, innerhalb dessen verschiedene rationale Verfahren aus unterschiedlichen wissenschaftlichen Gebieten integriert werden können mit dem Ziel, effektive Softwaresysteme zur Planungsunterstützung zu entwickeln, ohne dabei jedoch die kreativen Aspekte des Entwerfens gänzlich aus dem Blick zu verlieren.

Es erscheint angebracht, hier eine Abgrenzung zum *Computational Design* vorzunehmen, welches in den letzten Jahren verstärkt zur Beschreibung einer bestimmten Verwendung neuer kommerziell verfügbarer Softwaresysteme für Entwurfsaufgaben herangezogen wurde. *Computational Design* zielt primär darauf ab, originelle, geo-

metrisch komplexe Ergebnisse zu erreichen. Funktionale Aspekte werden dabei vor allem als Mittel zum Zweck bzw. als „*Formgeneratoren*" benutzt. An den vielfältigen Beispielen, welche komplexe Formen durch die Optimierung isoliert betrachteter Kriterien rechtfertigen (Hensel & Menges, 2008), wird dieses Herangehen deutlich. Die Autoren des vorliegenden Buchs betrachten dieses Konzept als unzureichend, da es die komplexen Wechselwirkungen verschiedener Kriterien, die beim Entwerfen und Planen zu berücksichtigen sind, vernachlässigt und oft auf rein formale Aspekte reduziert. Diese sind zwar wichtig, aber für sich genommen nicht tragfähig.

Um den komplexen Gegebenheiten von Entwurfsprozessen gerecht zu werden, bieten beispielsweise multikriterielle Optimierungsverfahren einen vielversprechenden Ansatz, der im Rahmen der *Computational Planning Science* zu erkunden ist. Selbstverständlich sind die Konzepte des *Computational Design* ebenfalls in diesen Rahmen zu integrieren. Denn nur dadurch wird es möglich, das Gegeneinander von Entwurfsphilosophien wie intuitiv vs. rational (poetisch vs. regelbasiert, analog vs. digital) in ein Miteinander im Sinne einer gemeinsamen Arbeit an der Erforschung von Entwurfsmethoden und -qualitäten zu verwandeln.

Abkürzungen

ALES	Allgemeines Layout-Entwurfssystem
CAAD	Computer Aided Architectural Design
CALD	Computer Aided Layout Design, bzw. computerbasierter Layoutentwurf
EA	Evolutionäre Algorithmen
ES	Evolutionäre Strategie
GA	Genetischer Algorithmus
GP	Genetische Programmierung
HLP	hierarchisch gegliedertes Layoutproblem
KI	Künstliche Intelligenz
MOOP	Multikriterielles Optimierungsproblem
NHLP	nicht-hierarchisch gegliedertes Layoutproblem
NP	nichtdeterministisch polynomielle (Vollständigkeit)
ZA	Zelluläre Automaten

Glossar

Constraints: In der Regel explizit vorgegebene Anforderungen die eine Lösung eines Entwurfsproblems haben muss.

Evaluationsmechanismus: Gesamtheit aller zur Bewertung einer Lösungsvariante verwendeten Berechnungen (Fitnessfunktionen).

Fitness: Gibt die Qualität einer Variante (eines Individuums) hinsichtlich eines bestimmten Kriteriums an. Diese wird anhand einer Fitnessfunktion bestimmt. Eine Variante kann hinsichtlich verschiedener Kriterien bewertetet werden. Die Berechnung aller Kriterien erfolgt im Rahmen des Evaluationsmechanismus.

Generativer Mechanismus: Gesamtheit aller generativen Methoden, die zur Erzeugung der Lösungsvarianten verwendet werden.

Generatives System: Generative Systeme sind in der Lage, auf Basis einfacher Regeln verschiedene Strukturen zu erzeugen. Abhängig von den Regeln bzw. deren Parametern können stark variierende und unvorhersehbare Strukturen entstehen. Ein generatives System besteht aus einem generativen Mechanismus und einem Evaluationsmechanismus (siehe auch Abb. 8 auf Seite 44).

Genotypen und Phänotypen: Genotypen sind (z.B. binär) kodierte Lösungsvarianten eines Problems. Kodiert werden die Parameterwerte, welche vom generativen Mechanismus verwendet werden. Operationen wie Mutation und Rekombination werden auf die Genotypen angewandt. Genotypen werden mittels Mapping in Phänotypen überführt, die dem Selektionsprozess ausgesetzt werden (siehe Abb. 9 auf Seite 46). Alle zu einem Zeitpunkt vorhandenen Phäno- bzw. Genotypen bilden die Individuen einer Generation.

Hierarchische Gliederung: Bedeutet, dass sich innerhalb eines Elementes andere Elemente befinden können. Die Tiefe der Verschachtelung kann dabei beliebig sein. Beispielsweise kann ein Grundstück Häuser umfassen, innerhalb derer sich Wohnungen befinden, die wiederum Räume enthalten, usw.

Konvergenzgeschwindigkeit: Die Geschwindigkeit, mit der ein EA sich einem (lokalen oder globalen) Optimum annähert bzw. dieses erreicht.

Layoutsolver: Spezielles generatives System zur Lösung von Layoutproblemen (siehe auch Abb. 8 auf Seite 44).

Mapping: Verfahren, welches (meist binär) kodierte Lösungen vom Such- in den Lösungsraum überführt, die dort die Phänotypen bilden, welche dem Selektionsprozess ausgesetzt werden (siehe auch Abb. 9 auf Seite 46).

Maßstabsebenen: Bezeichnen die verschiedenen Kontexte einer Planung (z.B. städtebauliche Nachbarschaft, Gebäudemassen, Funktionsbereiche, Grundrisse).

Mutation: Zufällige Variation des Parameterwerts eines Individuums.

Operationale und nicht-operationale Kriterien: Ein Problem ist operational, wenn es so genau beschrieben werden kann, dass sich angeben lässt, durch welche Schritte es zu lösen ist. Dies geschieht dadurch, dass im Rahmen einer Analyse ein komplexes Problem in Teilprobleme zerlegt wird, welche dann immer genauer dargestellt werden können. Die konkret definierbaren, handfesten Kriterien zur Problembeschreibung werden als operationale Kriterien bezeichnet. Dagegen sind nicht-operationale Probleme vage definiert und bedeutende Elemente der Aufgabestellung sind unbekannt oder nicht genau (quantitativ) erfassbar. Die Lösung des Problems liegt hier größtenteils in dessen Definition. Nicht-operationale Kriterien sind dementsprechend schwer quantitativ zu fassen und können meist nur ungefähr umschrieben werden.

Pareto-Front: Alle pareto-optimalen Lösungen für ein Problem liegen auf der Pareto-optimalen-Front. Die Pareto-Front ist dadurch charakterisiert, dass es nicht möglich ist, eine Lösung, die auf ihr liegt, weiter zu verbessern, ohne zugleich eine andere zu verschlechtern.

Pareto-Optimalität: Bei sich widersprechenden Zielen kann keine Lösung gefunden werden, die jedes Zielkriterium vollständig erfüllt. Im besten Fall erhält man Lösungen, die einen möglichst guten Kompromiss bezüglich der Zielkriterien darstellen. Für einen solchen Kompromiss gibt es in der Regel viele verschiedene Möglichkeiten, die als pareto-optimal bezeichnet werden.

Rekombination: Kombination der Parameterwerte von mindestens zwei (Eltern-) Individuen zur Erzeugung eines oder mehrerer neuer (Kind-) Individuen.

Selektion: Beschreibt den Prozess, der die besten Individuen einer Generation auswählt, welche die (Eltern-) Individuen der nächsten Generation bilden.

Such- und Lösungsraum: Grundsätzlich enthält der Suchraum die Menge aller möglichen Varianten, die für ein bestimmtes Problem denkbar sind. Die meisten dieser Varianten sind allerdings völlig unbrauchbar. Technisch gesehen wird die Dimension des Suchraums durch die parametrisierten Kriterien definiert, welche vom generativen Mechanismus variiert werden können. Die Größe der Dimensionen hängt von den Wertebereichen der Kriterien ab. Der Lösungsraum enthält die Menge aller Lösungen für ein bestimmtes Problem und ist theoretisch eine Teilmenge des Suchraums. Praktisch enthält der Suchraum, der von einem generativen Mechanismus abgedeckt werden kann, meistens nicht alle Lösungen. Daher bilden Such- und Lösungsraum bei den in diesem Buch betrachteten Beispielen eine Schnittmenge. Das heißt, im Suchraum befinden sich zumindest einige brauchbare Lösungen.

Topologische Relationen: Geben abstrakte Beziehungen zwischen Elementen an, z.B. dass diese zueinander benachbart sein sollen. Diese Beziehungen sagen noch nichts über die konkrete geometrische Konfiguration der Elemente aus. Im Kontext dieses Buchs werden topologische Relationen in der Regel verwendet, um geforderte Nachbarschaftsbeziehungen zwischen Räumen in einem Grundriss anzugeben (siehe auch Abb. 20 auf Seite 77).

Varianz: Bezeichnet das Spektrum unterschiedlicher Lösungen, das von einem generativen System erzeugt werden kann.

Referenzen

Alexander, C., Ishikawa, S., & Silverstein, M. (1977). *A pattern language: towns, buildings, construction*: Oxford University Press.

Alexander, C., Ishikawa, S., Silverstein, M., & Czech, H. (2000). *Eine Muster-Sprache: Städte, Gebäude, Konstruktion*: Löcker.

Anders, F., & König, R. (2011). *Analyse und Generierung von Straßennetzwerken mittels graphenbasierter Methoden*. Weimar: Bauhaus-Universität Weimar, Professur Informatik in der Architektur.

Arvin, S. A., & House, D. H. (2002). Modeling architectural design objectives in physically based space planning. *Automation in Construction, 11*, 213–225.

Bäck, T. (1994). *Evolutionary Algorithm in Theory and Practice*: Oxford University Press.

Bäck, T. (2000a). Binary Strings *Evolutionary Computation 1 - Basic Algorithms and Operators* (pp. 132-135). New York, NY: Taylor & Francis Group, LLC.

Bäck, T. (2000b). Introduction to Evolutionary Algorithms. In T. Bäck, D. B. Fogel & T. Michalewicz (Eds.), *Evolutionary Computation 1: Basic Algorithms and Operators* (pp. 59-64). New York: Taylor & Francis.

Bäck, T., Hoffmeister, F., & Schwefel, H.-P. (1991). *A Survey of Evolution Strategies.* Paper presented at the Fourth International Conference on Genetic Algorithms

Bafna, S., & Shah, S. (2007). *The Evolution of Orthogonality in Built Space: An Argument From Space Syntax.* Paper presented at the 6th International Space Syntax Symposium, Istanbul.

Batty, M. (2001). Exploring isovist fields: space and shape in architectural and urban morphology. *Environment and Planning B: Planning and Design, 28*, 123-150.

Batty, M. (2005). *Cities and Complexity: Understanding Cities with Cellular Automata, Agent-Based Models, and Fractals*. London: MIT Press.

Batty, M., & Xie, Y. (1994). From cells to cities. *Environment and Planning B: Planning and Design 21*(7), 31-48.

Benedikt, M. L. (1979). To take hold of space: isovists and isovist fields. *Environment and Planning B, 6*(1), 47-65.

Bentley, J. L. (1975). Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM, 18,* 509 - 517.

Bentley, J. L. (1990). *K-d Trees for Semidynamic Point Sets*. Paper presented at the Sixth Annual Symposium on Computational Geometry.

Bentley, J. L., & Friedman, J. H. (1979). Data Structures for Range Searching. *ACM Computing Surveys (CSUR), 11,* 397-409.

Bentley, P. J. (1999). An Introduction to Evolutionary Design by Computers. In P. J. Bentley (Ed.), *Evolutionary Design by Computers* (pp. 1 - 74). San Francisco: Morgan Kaufmann.

Bentley, P. J., & Corne, D. W. (2002). An Introduction to Creative Evolutionary Systems. In P. J. Bentley & D. W. Corne (Eds.), *Creative Evolutionary Systems* (pp. 1-76). San Francisco: Morgan Kaufmann.

Braach, M. (2002). Kaisersrot - computergestützter individualisierter Städtebau. *Werk Bauen Wohnen*, 4.

Broughton, T., Tan, A., & Coates, P. (1997). *The Use of Genetic Programming In Exploring 3D Design Worlds: A Report of Two Projects by Msc Students at CECA UEL.* Paper presented at the CAAD futures, München.

Buchanan, A., & Fitzgibbon, A. (2006). *Interactive Feature Tracking using K-D Trees and Dynamic Programming*. Paper presented at the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition.

Buelinckx, H. (1993). Wren's language of City church designs: a formal generative classification. *Environment and Planning B: Planning and Design, 20*(6), 645 – 676.

Catmull, E., & Clark, J. (1978, November). Recursively generated B-spline surfaces on arbitrary topological meshes. *Computer Aided Design, vol. 10, no. 6,* 350-355.

Chao, K.-M., Guenov, M., Hills, B., Smith, P., Buxton, I., & Tsai, C.-F. (1997). An expert system to generate associativity data for layout design. *Artificial Intelligence in Engineering, 11*, 191-196.

Chouchoulas, O., & Day, A. (2007). Design Exploration Using a Shape Grammar with a Genetic Algorithm. *open house international, 32*(2).

Coates, P., Appels, T., Simon, C., & Derix, C. (2001). *Current work at CECA, Three projects: Dust, Plates & Blobs.* Paper presented at the Generative Art International Conference, Milan.

Coates, P., Derix, C., Krakhofer, I. S. P., & Karanouh, A. (2005). *Generating architectural spatial configurations. Two approaches using Voronoi tessellations and particle systems*. Paper presented at the Proceedings of the VIII Generative Art International Conference (GA2005).

Coates, P., & Hazarika, L. (1999). *The use of Genetic Programming for applications in the field of spatial composition.* Paper presented at the Generative Art Conference, Milan.

Coates, P., Healy, N., Lamb, C., & Voon, W. L. (1996). The use of Cellular Automata to explore bottom up architectonic rules. In J. Rossignac & F. Sillion (Eds.), *Eurographics '96*. Poitiers: Blackwell Publishers.

Coates, P., & Schmid, C. (2000). Agent Based modelling. from CECA - The centre for computing & environment in architecture, University of East London School of Architecture: http://uelceca.net/research/ECAADE/agentnotes%20the%20paper398liverpool%201999.pdf

Coello, C. A. (1998). A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems,* 269-308.

Cohon, J. L. (1978). *Multiobjective Programming and Planning*. New York, San Francisco, London: Academic Press.

Conroy, R. (2001). *Spatial Navigation in immersive virtual environments.* Unpublished Dissertation, London.

Corne, D. W., Knowles, J. D., & Martin, J. O. (2000). *The Pareto Envelope-based Selection Algorithm for Multiobjective Optimization.* Paper presented at the Parallel Problem Solving from Nature VI Conference.

Coyne, R. F. (1988). *Logic Models of Design*. London: Pitman Publishing.

Coyne, R. F., & Flemming, U. (1990). Planning in design synthesis: abstraction-based LOOS. In J. S. Gero (Ed.), *Artificial Intelligence in Engineering V* (Vol. 1, pp. 91-111). New York: Springer.

Cozic, L. (2006). 2D Polygon Collision Detection. *The Code Project*, 2010, from http://www.codeproject.com/KB/GDI-plus/PolygonCollision.aspx

De Berg, M., Cheong, O., Van Kreveld, M., & Overmars, M. (1997). *Computational Geometry: Algorithms and Applications*. Berlin: Springer-Verlag.

Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms*. New York: John Wiley & Sons.

Derix, C. (2004). *Building a Synthetic Cognizer.* Paper presented at the Design Computational Cognition, MIT, Boston.

Derix, C. (2009). In-Between Architecture Computation. *International Journal of Architectural Computing, 7*(4).

Dillenburger, B., Braach, M., & Hovestadt, L. (2009). *Building design as an individual compromise between qualities and costs: A general approach for automated building generation under permanent cost and quality control.* Paper presented at the CAADFutures 2009.

Doulgerakis, A. (2007). *Genetic Programming + Unfolding Embryology in Automated Layout Planning*. University College London, London.

Duarte, J. P. (2001). *Customizing mass housing: a discursive grammar for Siza's Malagueira houses.* Massachusetts Institute of Technology.

Duarte, J. P. (2005). Towards the mass customization of housing: the grammar of Siza's houses at Malagueira. *Environment and Planning B: Planning and Design, 32*(3), 347 – 380.

Duarte, J. P., Rocha, J. d. M., & Soares, G. D. (2007). Unveiling the structure of the Marrakech Medina: A shape grammar and an interpreter for generating urban form. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 21*(in press).

Eckert, C., Kelly, I., & Stacey, M. (1999). Interactive generative systems for conceptual design: An empirical perspective. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 13*(4), 303-320.

Elezkurtaj, T. (2004). *Evolutionäre Algorithmen zur Unterstützung des kreativen architektonischen Entwerfens.* Technische Universität Wien, Vienna, Austria.

Elezkurtaj, T., & Franck, G. (2001). *Evolutionary Algorithm in Urban Planning.* Paper presented at the CORP 2001, Information Technology in Urban- and Spatial Planning, Vienna.

Elezkurtaj, T., & Franck, G. (2002). Algorithmic Support of Creative Architectural Design. *Umbau, 19*, 129-137.

Ernst, G., & Newell, A. (1969). *GPS: A Case Study in Generality and Problem Solving*. New York: Academic Press.

Fernau, H. (1994). *Iterierte Funktionen, Sprachen und Fraktale*. Mannheim: B. I. Wissenschaftsverlag.

Fleming, U. (1987). The Role of Shape Grammars in the Analysis and Creation of Designs. In Y. E. Kalay (Ed.), *Computability of Design* (pp. 245-272). New York: Wiley.

Flemming, U. (1977). *Automatisierter Grundrissentwurf: Darstellung, Erzeugung und Dimensionierung von dicht gepackten, rechtwinkligen Flächenanordnungen* Technische Universitat Berlin.

Flemming, U. (1989). More on the representation and generation of loosely packed arrangements of rectangles. *Environment and Planning B: Planning and Design, 16*(3), 327-359.

Flemming, U., Baykan, C. A., Coyne, R. F., & Fox, M. S. (1992). Hierarchical generate-and-test vs. constraint-directed search. A comparison in the context of layout synthesis. In J. Gero (Ed.), *Artificial Intelligence in Design '92* (pp. 817-838). Boston: Kluwer Academic Publisher.

Flemming, U., & Woodbury, R. (1995). Software environment to support early phases in building design (SEED): Overview. *Journal of Architectural Engineering, 1*, 147-152.

Fogel, L., Owens, A. J., & Walsch, M. J. (1966). Artificial Intelligence through Simulated Evolution: Wiley.

Franck, G., & Elezkurtaj, T. (2002). Design Methods. Retrieved 10.08.2008, from http://www.iemar.tuwien.ac.at/assets/docs/design_methods.pdf

Franz, G., & Wiener, J. M. (2008). From space syntax to space semantics: a behaviorally and perceptually oriented methodology for the efficient description of the geometry and topology of environments. *Environment and Planning B: Planning and Design, 35*(4), 574-592.

Frazer, J. (1974). Reptiles. *Architectural Desig*(April), 231-239.

Frazer, J. (1995). *An Evolutionalry Architecture*. London: Architectural Assocoation Publications.

Frew, R. S. (1980). *A survey of space allocation algorithms in use in architec-tural design in the past twenty years*. Paper presented at the DAC '80 Proceedings of the 17th Design Automation Conference.

Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1977). *An Algorithm for Finding Best Matches in Logarithmic Expected Time*. Paper presented at the ACM Transactions on Mathematical Software (TOMS).

Fussell, D. S., & Subramanian, K. R. (1988). *Fast Ray Tracing Using K-d Trees*. Austin, TX, USA: University of Texas at Austin, Department of Computer Sciences.

Gänshirt, C. (2007). *Tools for Ideas: An Introduction to Architectural Design*. Basel: Birkhäuser.

Getzels, J. W., & Csikszentmihalyi, M. (1967). Scientific creativity. *Science Journal, 3*, 80–84.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning* (1 ed.). Boston: Addison-Wesley.

Goldberg, D. E., & Lingle, R. (1985). *Alleles, loci and the traveling salesman problem.* Paper presented at the International Conference on Genetic Algorithms.

Goodman, J. E., & O'Rourke, J. (2004). *Handbook of discrete and computational geometry*. Chapman & Hall/CRC.

Goodrich, M. T., & Tamassia, R. (2002). *Algorithm Design: Foundations, Analysis and Internet Examples*. New York, United States: John Wiley & Sons

Guski, R. (2000). *Wahrnehmung: eine Einführung in die Psychologie der menschlichen Informationsaufnahme*: Kohlhammer.

Hahn, E., Bose, P., & Whitehead, A. (2006). *Persistent Realtime Building Interior Generation.* Paper presented at the Sandbox Symposium 2006, Boston, MA, USA.

Hamblin, C. L. (1962). Translation to and from Polish notation. *The Computer Journal, Volume 5, Issue 3*, 210-213.

Harding, J., & Derix, C. (2010). *Associative Spatial Networks in Architectural Design: Artificial Cognition of Space using Neural Networks with Spectral Graph Theory*. Paper presented at the Design Computing and Cognition DCC'10.

Hensel, M., & Menges, A. (2008). Performance als Forschungs- und Entwurfskonzept. *Arch +, 188*, 31-35.

Hillier, B. (1996). *Space is the machine: a configurational theory of architecture*: Cambridge University Press.

Hillier, B. (2005). The art of place and the science of space. *World Architecture, 11*(Special Issue on Space Syntax), 24-34.

Hillier, B. (2007). The Common Language of Space: A way of looking at the social, economic and environmental functioning of cities on a common basis. Retrieved from http://www.spacesyntax.org/publications/commonlang.html

Hillier, B., & Hanson, J. (1984). *The social logic of space.* (Reprinted paperback edition 2003 ed.). Cambridge: Cambridge University Press.

Holland, J. (1973). Genetic Algorithm and the Optimal Allocations of Trials. *SIAM Journal of Computing, 2*(2), 88-105.

Holland, J. (1992). *Adaption in Natural and Artificial Systems: An Introduction with Applications to Biology, Control, and Artificial Intelligence* (2 ed.): MIT Press.

Horn, J. (1997). F1.9 Multicriterion decision making *Handbook of Evolutionary Computation* (pp. 1-9). Bristol: IOP Publishing Ltd.

Hower, W. (1997). Placing computations by adaptive procedures. *Artificial Intelligence in Engineering, 11*, 307-317.

Hower, W., & Graf, W. H. (1996). A bibliographical survey of constraint-based approches to CAD, graphics, layout, visualization, and related topics. *Knowledge-Based Systems, 9*, 449-464.

Jakob, W. (2004). *Eine neue Methodik zur Erhöhung der Leistungsfähigkeit Evolutionärer Algorithmen durch die Integration lokaler Suchverfahren*. Karlsruhe: Forschungszentrum Karlsruhe.

Jo, J. H., & Gero, J. S. (1998). Space layout planning using an evolutionary approach. *Artificial Intelligence in Engineering, 12*, 149-162.

Kado, K. (1995). *An Investigation of Genetic Algorithms for Facility Layout Problems.* University of Edinburgh, Edinburgh.

Knecht, K. (2011). *Augmented Urban Model: Ein Tangible User Interface zur Unterstützung von Stadtplanungsprozessen.*

Knecht, K., & Koenig, R. (2010). *Generating Floor Plan Layouts with K-d Trees and Evolutionary Algorithms.* Paper presented at the GA2010 - 13th Generative Art Conference, Milan, Italy.

Knight, T. W. (1981). The forty-one steps. *Environment and Planning B: Planning and Design, 8*(1), 97 – 114.

Knight, T. W. (1994). *Transformations in Design: a Formal Approach to Stylistic Change and Innovation in the Visual Arts*. Cambridge, UK: Cambridge University Press.

Koenig, R. (2010). *Generative Planungsmethoden aus strukturalistischer Perspektive*. Weimar: Bauhaus-Universität Weimar.

Koenig, R. (2011a). Generative planning methods from a structuralist perspective. In T. Valena, T. Avermaete & G. Vrachliotis (Eds.), *Strukturalismus Reloaded: Rule-Based Design in Architecture and Urbanism* (pp. 275-280). Stuttgart/London: Axel Menges.

Koenig, R. (2011b). *Generierung von Grundriss-Layouts mittels hybrider Evolutions-Strategie*. Weimar: Bauhaus-Universität Weimar.

Koenig, R. (2012). Generating urban structures: A method for urban planning supported by multi-agent systems and cellular automata. *Space & Form, 16*.

Koenig, R., & Bauriedel, C. (2004). *Computer-generated City Structures*. Paper presented at the Generative Art Conference.

Koning, H., & Eizenberg, J. (1981). The language of the prairie: Frank Lloyd Wright's prairie houses. *Environment and Planning B: Planning and Design, 8*(3), 295 – 323.

Koutamanis, A. (2000). *Representations from generative systems.* Paper presented at the Artificial Intelligence in Design, Worcester, Massachusetts.

Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*: MIT Press.

Krämer, J., & Kunze, J.-O. (2005). *Design Code*. Berlin: TU Berlin.

Kramer, O. (2008). *Self-adaptive heuristics for evolutionary computation*: Springer.

Kursawe, F. (1990). A variant of evolution strategies for vector optimization *Parallel Problem Solving from Nature I* (pp. 193-197).

Lai, M., & Wong, D. F. (2001). *Slicing Tree is a Complete Floorplan Representation.* Paper presented at the DATE '01 Proceedings of the conference on Design, automation and test in Europe Piscataway, NJ, USA.

Lawson, B. (2005). Oracles, draughtsman and agents: the nature of knowledge and creativity in design and the role of IT. *Automation in Construction, 14*(3), 383-391.

Lawson, B. (2006). *How Designers Think: The Design Process Demystified* (4 ed.). Oxford: Architectural Press.

Li, S.-P., Frazer, J. H., & Tang, M.-X. (2000). *A Constraint Based Generative System for Floor Layouts.* Paper presented at the Fifth Conference on Computer Aided Architectural Design Research in Asia (CAADRIA), Singapore

Lobos, D., & Donath, D. (2010). Space Layout Problem in Architecture. A survey and reflections. *Arquiteturarevista, 6*(2).

Lopes, R., Tutenel, T., Smelik, R. M., de Kraker, K. J., & Bidarra, R. (2010). *A constrained growth method for procedural floor plan generation.* Paper presented at the GAME-ON 2010, Leicester, United Kingdom.

Maneewongvatana, S., & Mount, D. M. (1999). *It's okay to be skinny if your friends are fat*. Paper presented at the Center for Geometric Computing 4th Annual Workshop on Computational Geometry.

March, L., & Steadman, P. (1974). *The geometry of environment: an introduction to spatial organization in design* (2nd ed.): M.I.T. Press.

Marson, F., & Musse, S. R. (2010, January). Automatic Real-Time Generation of Floor Plans Based on Squarified Treemaps Algorithm. *International Journal of Computer Games Technology*.

Medjdoub, B., & Yannou, B. (2001). Dynamic space ordering at a topological level in space plannung. *Artificial Intelligence in Engineering, 15*, 47-60.

Miller, G. A. (1956). The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *The Psychological Review, 63*, 81-97.

Miranda, P. (2004). ArchiKluge. Retrieved 12.08.2008, from http://www.armyofclerks.net/ArchiKluge/index.htm

Mitchell, W. J. (1998). *The Logic of Architecture.: Design, Computation, and Cognition* (6 ed.). Cambridge, Massachusetts: MIT Press.

Moore, A. W. (1991). An introductory tutorial on kd-trees, *Extract from: Efficient Memory-based Learning for Robot Control Technical, PhD Thesis: Report No. 209*. Cambridge: University of Cambridge.

Müller, P. (2007). Pascal Mueller's Wiki. Retrieved 07.08.2008, from http://www.vision.ee.ethz.ch/~pmueller/wiki/Main/Front

Müller, P., Wonka, P., Haegler, S., Ulmer, A., & Van Gool, L. (2006). *Procedural Modeling of Buildings*. Paper presented at the Proceedings of ACM SIGGRAPH 2006 / ACM Transactions on Graphics (TOG)

Müller, P., Zeng, G., Wonka, P., & Van Gool, L. (2007). *Image-based Procedural Modeling of Facades.* Paper presented at the ACM SIGGRAPH 2007 / ACM Transactions on Graphics, San Diego.

Negroponte, N. (1970). *The architecture machine*: M.I.T. Press.

O'Sullivan, D. (2001). Exploring spatial process dynamics using irregular cellular automaton models. *Geographical Analysis, 33*, 1-18.

Otten, R. H. J. M. (1982). *Automatic Floorplan Design.* Paper presented at the Proceedings of the 19th Design Automation Conference, Piscataway, NJ, USA.

Parish, Y. I. H., & Müller, P. (2001). *Procedural Modeling of Cities.* Paper presented at the ACM SIGGRAPH, Los Angeles, CA.

Parmee, I. (1999). Exploring the Design Potential of Evolutionary Search, Exploration and Optimization. In P. J. Bentley (Ed.), *Evolutionary Design by Computers* (pp. 119-143). San Francisco: Morgan Kaufmann.

Rechenberg, I. (1994). *Evolutionsstrategie '94*: Frommann Holzboog.

Río-Cidoncha, D. M. G., Iglesias, J. E., & Martínez-Palacios, J. (2007). A comparison of floorplan design strategies in architecture and engineering. *Automation in Construction, 16*, 559–568.

Rittel, H. W. J. (1992). *Planen, Entwerfen, Design: Ausgewählte Schriften zu Theorie und Methodik*. Stuttgart: Kohlhammer.

Rittel, H. W. J., & Webber, M. M. (1973). Dilemmas in a General Theory of Planning. *Policy Sciences, 4*, 155-169.

Rollo, J. (1995). Triangle and T-square: the windows of Frank Lloyd Wright. *Environment and Planning B: Planning and Design, 22*(1), 75 – 92.

Röpke, J. (1977). *Die Strategie der Innovation: Eine systemtheoretische Untersuchung der Interaktion von Individuum Organisation und Markt im Neuerungsprozess*. Tübingen: Mohr Siebeck.

Rosenman, M. A. (1997). The generation of form using an evolutionary approach. In D. Dasgupta & Z. Michalewicz (Eds.), *Evolutionary Algorithms in Engineering Applications*: Springer.

Rosenman, M. A., & Gero, J. S. (1996). Modelling multiple views of design objects in a collaborative CAD environment. *Computer-Aided Design, 28*(3), 193-205.

Sadalla, E. K., & Montello, D. R. (1989). Remembering Changes in Direction. *Environment and Behavior, 21*(3), 346-363.

Schaffer, J. D. (1985). *Multiple objective optimization with vector evaluated genetic algorithms.* Paper presented at the First International Conference on Genetic Algorithms.

Schneider, S., Koenig, R., & Pohle, R. (2011). *Who cares about right angles? Overcoming barriers in creating rectangularity in layout structures.* Paper presented at the eCAADe 2011: Respecting Fragile Places, Ljubljana, Slovenia.

Schnier, T. (2008). *Evolving out of the Box: Overcoming the Signature Problem in Evolutionary Art.* Paper presented at the NSF International Workshop on Studying Design Creativity, University of Provence.

Schön, D. A. (1983). *The reflective practitioner: how professionals think in action*: Basic Books.

Schön, D. A. (1992). Designing as reflective conversation with the materials of a design situation. *Knowledge-Based Systems, 5*(1), 3-14.

Schubert, G., Artinger, E., Petzold, F., & Klinker, G. (2011). *Tangible tools for architectural design - seamless integration into the architectural workflow.* Paper presented at the ACADIA 2011, Calgary and Banff.

Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*: Wiley-Interscience.

Schweitzer, F. (1997). Wege und Agenten: Reduktion und Konstruktion in der Selbstorganisationstheorie. In H.-J. Krug & L. Pohlmann (Eds.), *Selbstorganisation* (Vol. 8, pp. 113-135). Berlin: Duncker & Humblot.

Simon, H. (1969). *The Science of the Artificial*: MIT Press.

Stamps, A. (2005). Enclosure and Safety in Urbanscapes. *Environment and Behaviour, 37* 102-133.

Stiny, G. (1975). *Pictorial and Formal Aspects of Shape and Shape Grammar*. Stuttgart: Birkhäuser.

Stiny, G. (2006). *Shape: Talking about Seeing and Doing*. Cambridge; London: MIT Press.

Stiny, G., & Gips, J. (1972). *Shape Grammars and the Generative Specification of Painting and Sculpture.* Paper presented at the IFIP Congress 1971., Amsterdam.

Stiny, G., & Mitchell, W. J. (1978). The Palladian Grammar. *Environment and Planning B: Planning and Design, 5*(1), 5-18.

Stiny, G., & Mitchell, W. J. (1980). The grammar of paradise: on the generation of Mughul gardens. *Environment and Planning B: Planning and Design, 7*(2), 209 – 226.

Toffoli, T., & Margolus, N. (1987). *Cellular Automata Machines: A New Environment for Modeling*. Cambridge: MIT Press.

Turner, A., Doxa, M., O'Sullivan, D., & Penn, A. (2001). From isovists to visibility graphs: a methodology for the analysis of architectural space. *Environment and Planning B: Planning and Design, 28*(1), 103-121.

Valenzuela, C. L., & Wang, P. Y. (2002). VLSI Placement and Area Optimization Using a Genetic Algorithm to Breed Normalized Postfix Expressions. *IEEE Transactions on Evolutionary Computation, Vol. 6, Issue 4,* 390-401.

Wald, I., & Havran, V. (2006). *On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N)*. Paper presented at the Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing.

Watanabe, M. S. (1990, 2003). The Induction Cities: Induction Design Project (1990-). from http://www.makoto-architect.com/idc2000/index.htm

Watanabe, M. S. (2002). *Induction Design: A Method for Evoutionary Design: The IT Revolution in Architecture*. Basel: Birkhäuser.

Weicker, K. (2007). *Evolutionäre Algorithmen* (2 ed.). Wiesbaden: Teubner.

Whitehead, B., & Eldars, M. Z. (1964). An approach to the optimum layout of single-storey buildings. *The Architects Journal*, 1373-1380.

Whitley, D. (1994). A Genetic Algorithm Tutorial. *Statistics and Computing, Vol. 4, No.2,* 65-85.

Wiener, J. M., Hölscher, C., Büchner, S., & Konieczny, L. (2011). Gaze Behaviour during Space Perception and Spatial Decision Making. *Psychological Research, in press*.

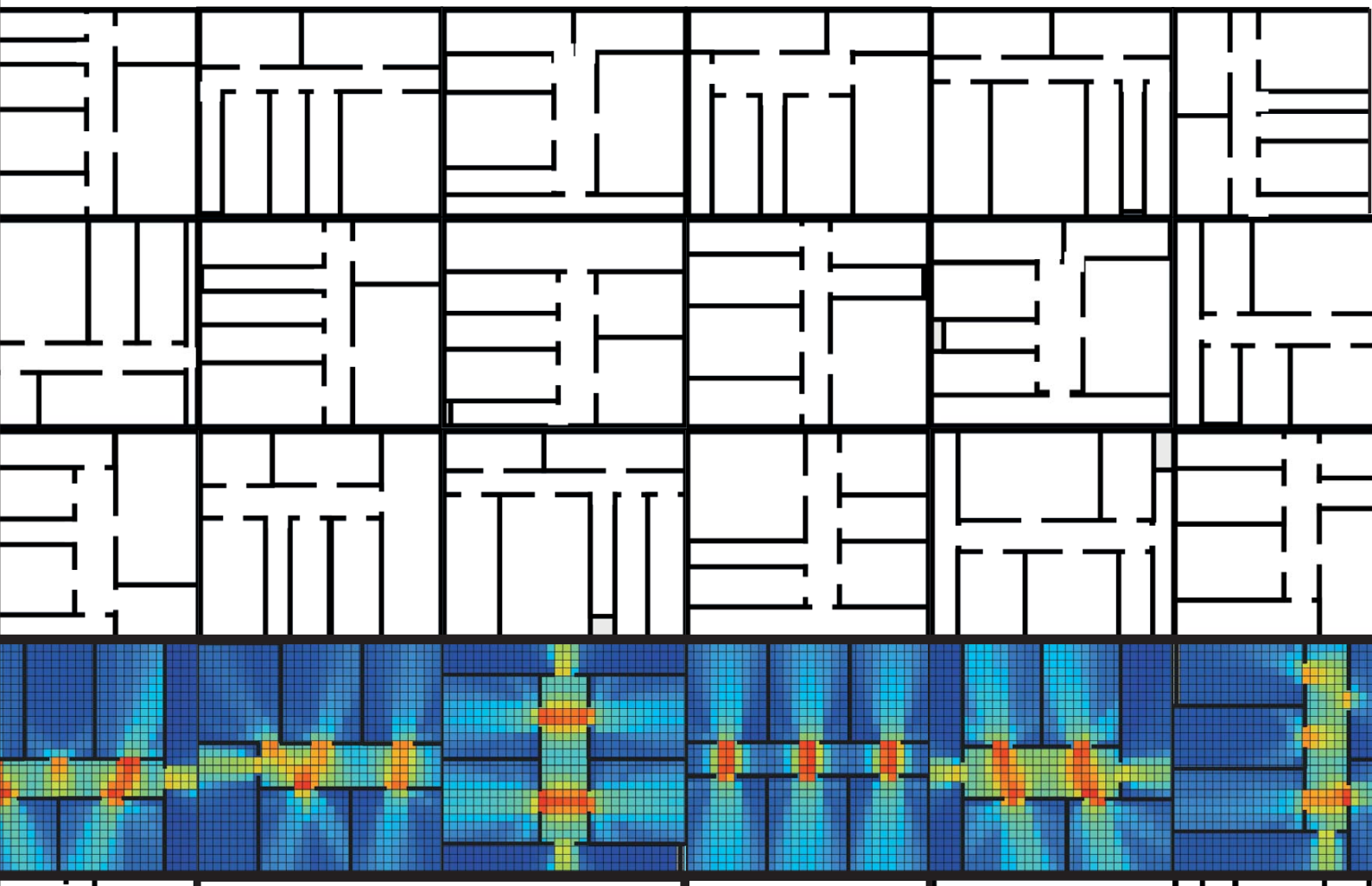
Wonka, P., Wimmer, M., Sillion, F., & Ribarsky, W. (Eds.). (2003) Proceedings ACM SIGGRAPH 2003 (Vols. ACM Transaction on Graphics). San Diego, USA: ACM.

Young, F. Y., & Wong, D. F. (1997). *How good are slicing floorplans?* Paper presented at the Proceedings of the International Symposium on Physical Design ISPD'97, Napa Valley, California, USA.

Zitzler, E. (1999). Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications, *TIK-SCHRIFTENREIHE NR. 30*. Zürich: Institut für Technische Informatik und Kommunikationsnetze, Eidgenössische Technische Hochschule Zürich.

Zitzler, E., Laumanns, M., & Bleuler, S. (2003). A Tutorial on Evolutionary Multiobjective Optimization *In Metaheuristics for Multiobjective Optimisation* (pp. 3-38): Springer-Verlag.

Die im vorliegenden Buch dokumentierten Untersuchungen befassen sich mit der Entwicklung von Methoden zur algorithmischen Lösung von Layoutaufgaben im architektonischen Kontext. Layout bezeichnet hier die gestalterisch und funktional sinnvolle Anordnung räumlicher Elemente, z.B. von Parzellen, Gebäuden, Räumen auf bestimmten Maßstabsebenen. Die vorliegenden Untersuchungen sind im Rahmen eines von der Deutschen Forschungsgemeinschaft geförderten Forschungsprojekts entstanden.



Bauhaus-Universität Weimar
Verlag

gefördert durch die DFG