Bauhaus-Universität Weimar
Informatik in der Architektur | InfAR

Arbeitspapiere
Working Papers

Nr. 4, März 2011

Generierung von Grundriss-Layouts mittels hybrider Evolutions-Strategie

Reinhard Koenig

ISSN 2191-2416

Bauhaus-Universität Weimar, Professur Informatik in der Architektur, Belvederer Allee 1, 99421 Weimar
Fon: +49/3643/584201, caad@architektur.uni-weimar.de, http://infar.architektur.uni-weimar.de

Reinhard Koenig
Generierung von Grundriss-Layouts mittels hybrider Evolutions-Strategie
Weimar 2011
Arbeitspapiere Informatik in der Architektur, Bauhaus Universität Weimar, Nr. 4
ISSN 2191-2416

Bauhaus-Universität Weimar, Professur Informatik in der Architektur
Belvederer Allee 1, 99421 Weimar
http://infar.architektur.uni-weimar.de

Titelbild: Jugendstil-Wendeltreppe im Hauptgebäude © Bauhaus-Universität Weimar

Redaktionelle Anmerkung:

Dr. Reinhard König ist Vertretungsprofessor der Professur Informatik in der Architektur an der Bauhaus-Universität Weimar.

Der Text ist im Rahmen des von der DFG geförderten Forschungsprojekts „KREMLAS: Entwicklung einer kreativen evolutionären Entwurfsmethode für Layoutprobleme in Architektur und Städtebau“ (DO 551/19-1) entstanden. http://infar.architektur.uni-weimar.de/service/drupal-cms/kremlas

Das in diesem Arbeitspapier vorgestellte Computerprogramm ist im Internet verfügbar unter http://infar.architektur.uni-weimar.de/service/drupal-cms/GenerativeMethoden

Generierung von Grundriss-Layouts mittels hybrider Evolutions-Strategie

Reinhard Koenig
reinhard.koenig@uni-weimar.de
Professur Informatik in der Architektur
Fakultät Architektur, Bauhaus-Universität Weimar, Belvederer Allee 1, 99421 Weimar, Germany

Abstract

Der vorliegende Text beschreibt ein computerbasiertes Verfahren zur Lösung von Layoutproblemen in Architektur und Städtebau, welches mit möglichst wenig Problemwissen auskommt und schnell brauchbare Ergebnisse liefert, die durch schrittweises Hinzufügen von Problemwissen interaktiv weiter ausgearbeitet werden können. Für das generative Verfahren wurde eine Evolutions-Strategie verwendet, die mit Mechanismen zur Kollisionserkennung und virtuellen Federn zu einem hybriden Algorithmus kombiniert wurde. Dieser dient erstens der Lösung des Problems der Dichten Packung von Rechtecken sowie zweitens der Herstellung bestimmter topologischer Beziehungen zwischen diesen Rechtecken. Die Bearbeitung beider Probleme wird durch schrittweise Erweiterung grundlegender Verfahren untersucht, wobei die einzelnen Schritte anhand von Performancetests miteinander verglichen werden. Am Ende wird ein iterativer Algorithmus vorgestellt, der einerseits optimale Lösungen garantiert und andererseits diese Lösungen in einer für eine akzeptable Nutzerinteraktion ausreichenden Geschwindigkeit generiert.

Keywords: Grundrissgenerierung, Multikriterielle Optimierung, Evolutions-Strategie, Dichte Packung, Computational Design, Kollisionserkennung, Kremlas

1. Einleitung

Der vorliegende Text befasst sich mit computerbasierten Methoden für Layout-Probleme in Architektur und Städtebau. Layout bezeichnet im Allgemeinen die Anordnung verschiedener Elemente meist innerhalb eines gegebenen Raums zu einem bestimmten Zweck. Beispielweise die Anordnung von räumlichen Elementen (z.B. Parzellen, Gebäuden, Räumen) auf bestimmten Maßstabsebenen. Maßstabsebenen bezeichnen im Folgenden die verschiedenen Kontexte einer Planung (z.B. städtebauliche Nachbarschaft, Gebäudemassen, Funktionsbereiche, Grundrisse).

Layout-Probleme findet man in den verschiedensten Gebieten, angefangen beim Entwurf eines Stundenplans oder einer Computerplatine, über die verschneidungsfreie Anordnung von abstrakten Elemente eines Graphen, der möglichst platzsparenden Organisation von Kisten einer Schiffsladung oder der Restflächen minimierenden Platzierung von auszuschneidenden Blechteilen, bis hin zur Anordnung der Räume eines Grundrisses oder der Gebäude einer städtebaulichen Nachbarschaft.

Wir befassen uns in dieser Arbeit mit dem Layout von Räumen in einem gegebenen Gebäude. Bei diesem Anwendungsbereich ist zu beachten, dass es sich nicht um eine reine Optimierungsaufgabe handelt, sondern Gestaltungsaspekte eine wichtige Rolle spielen. Aus diesem Grund konzentrieren wir uns in der vorliegenden Auseinandersetzung darauf, eine Strategie auszuarbeiten, die eine flexible Kombination von Gestaltungs- und Optimierungsmethoden für Grundriss-Layouts ermöglicht.

Im Rahmen der vorliegenden Arbeit wird dargestellt, wie sich anhand eines Computerprogramms Layout-Aufgaben in wesentlichen Teilen automatisch lösen lassen. In diesem Zusammenhang sprechen wir von computerbasiertem Layout. Ein wesentlicher Aspekt für die Integration von Gestaltungsmethoden in das Computerprogramm ist die Möglichkeit zur Interaktion zwischen einem Nutzer und dem Programm. Folglich werden wir neben einer detaillierten Beschreibung der Optimierungsmethoden immer wieder Aspekte der Nutzerinteraktion beleuchten und untersuchen, wie diese am besten mit Optimierungsmethoden kombiniert werden können.

2. Stand der Forschung

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.

Zur computerbasierten Lösung solcher Probleme wurden seit Anfang der 60er Jahre (Whitehead & Eldars, 1964) verschiedene Methoden entwickelt. Allen diesen Methoden ist gemein, dass sie einen generativen Mechanismus zur Produktion von Lösungsvarianten und einen Evaluations-Mechanismus zur Bewertung dieser Varianten beinhalten. Der Unterschied, den wir im Folgenden näher betrachten wollen, besteht in der Ausprägung der beiden Mechanismen.

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, welche für den generativen Mechanismus viel und für den Evaluations-Mechanismus wenig Problemwissen benötigen und bereits zu Beginn sicherstellen, dass ein entsprechendes generatives System akzeptable Ergebnisse liefert. Auf der anderen Seite finden sich Methoden, die mit wenig Problemwissen für den generativen Mechanismus auskommen, dafür aber einen aufwändigen Evaluations-Mechanismus 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 klassischen Shape Grammar (Duarte, 2000; 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 Zelluläre Automaten (Batty & Xie, 1994; Coates, Healy, Lamb, & Voon, 1996), agentenbasierten Systeme (Coates & Schmid, 2000; Derix, 2009) und Evolutionären Algorithmen (Hower, 1997; Jo & Gero, 1998; Rosenman, 1997). Aus welchem Grund sich aus Sicht der Autoren des vorliegenden Beitrags die letztgenannten Verfahren besser zur Lösung von Entwurfsproblemen eignen, soll im Folgenden verdeutlicht werden.

2.1. Entwurfsprobleme

Wie in der Einleitung dargestellt, treten Layout-Probleme beim Entwerfen auf verschiedenen Maßstabsebenen auf. Für die Bearbeitung von Entwurfsproblemen mittels Algorithmen ist es notwendig, zwischen wohl-definierten und schlecht-definierten Situationen zu unterscheiden. Eine schlecht-definierte Situation kann sich auf die Problemdefinition und/oder auf die Problemlösungs- und Ausführungsprogramme beziehen (Röpke, 1977). Für eine eindeutige Zuordnung wird hier der Vorschlag von Kirsch (1997) aufgegriffen, wohl-definierte Probleme als operational und schlecht-definierte Probleme als nicht-operational zu bezeichnen.

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). 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.

Bezugnehmend auf die oben getroffene Unterscheidung zwischen Verfahren welche viel und welche wenig Problemwissen erfordern, kann hier ergänzt werden, dass beiden Verfahren auf operationalen Kriterien basieren, letzteres aber keine vollständige Analyse des komplexen Problems beinhalten muss. In der Regel genügen Heuristiken, 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.

2.2. Generative 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-based Programmierung, die auf einem besonderen Programmierparadigma beruht und daher hier als eigenständige Methode angeführt wird. Gängige Methoden, die zur Generierung von Layout-Strukturen verwendet werden, sind Logische Programmierung (Coyne, 1988), Shape Grammars (Duarte, 2000; Stiny & Mitchell, 1978), constraint-based Systems (Li, et al., 2000; Medjdoub & Yannou, 2001), Zelluläre Automaten (Batty & Xie, 1994; Coates, et al., 1996), agentenbasierte Systeme (Coates & Schmid, 2000; Derix, 2009) und Evolutionäre Algorithmen (Jo & Gero, 1998; Rosenman, 1997).

2.3. Evolutionäre Algorithmen (EA)

Die unter Punkt 0 angeführten generativen Methoden behandeln in der Regel operationale Probleme. Kreative Lösungen sind allerdings die Folge schlecht-definierter Situationen.

Als kreativ kann eine Lösung erst dann bezeichnet werden, wenn sie nicht aus der genauen Beschreibung des Problems hervorgeht (vgl. Punkt 2.1). Dieser Sachverhalt macht deutlich, warum direkte Verfahren, die viel Problemwissen erfordern sich nicht zur Unterstützung kreativer Prozesse eignen. Aufgrund der unter Punkt 2.1 beschriebenen Schwierigkeit nicht-operationale auf operationale Fragen zu reduzieren ist die Einbeziehung menschlicher Fähigkeiten notwendig, um bestimmte Aspekte des Entwurfsprozesses auf operationale Probleme zu reduzieren und formal zu beschreiben. 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, S. 363).

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 biologische Evolution, dem kreativsten aller bekannten Prozesse verstanden werden können, sind beim derzeitigen Stand der Forschung die einzig verfügbare computerbasierte Methode für die Umsetzung eines zumindest in Teilbereichen schlecht-definierten Problemlösungssystems, welches es ermöglicht, neue, nicht schon in den Vorgaben enthaltene Lösungen zu finden (Franck, 2009). Natürlich funktionieren auch EA, wie alle Computerprogramme nur, wenn alle auszuführenden Einzelschritte genau definiert sind. Verglichen mit anderen Algorithmen kommen sie allerdings mit sehr wenig Problemwissen aus. Für eine gewisse Vielfalt an Lösungsvarianten sorgen stochastische Prozesse, die zu überraschende Ergebnissen führen können. Eines der anschaulichsten Beispiele liefert Sims (1999).

Bei Entwurfsproblemen hat man es in der Regel mit mehreren sich widersprechenden Anforderungen zu tun. Eine Lösung für ein solches multikriterielles Problem stellt immer einen Kompromiss in Form einer ausgewogenen Berücksichtigung der verschiedenen Anforderungen dar. Im Bereich der computerbasierten Optimierung werden derartige Problemstellungen als Multikriterielle Optimierungsprobleme (MOOP[1]) bezeichnet. EA bieten als einzige bekannte Methode, alle möglichen Kompromisslösungen zu finden, welche als pareto-

[1] Da die Literatur über Multikriterielle Optimierungsproble größtenteils englischsprachig ist (Coello, Lamont, & Veldhuizen, 2007; Deb, 2001), verwenden wir im Folgenden die Abkürzung der englischen Übersetzung für Multikriterielles Optimierungsproblem: Multi-Objective Optimisation Problem (MOOP).

optimale oder auch nicht-dominierte Lösungen bezeichnet werden (Coello, et al., 2007; Deb, 2001).

Unter EA werden folgende vier Algorithmen subsumiert:

a) 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. 1). Der Suchraum beinhaltet die kodierten Lösungen, die Genotypen, welche durch Kreuzung und Mutation variiert werden. Die kodierten Lösungen werden anhand eines Verfahrens, welches als Mapping (oder auch als Embryogenie) bezeichnet wird, in den Lösungsraum überführt und bilden dort die Phänotypen, die dem Selektionsprozess ausgesetzt werden. 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).

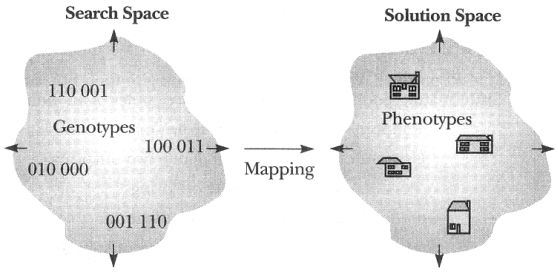


Abb. 1: Mapping der Genotypen des Suchraums auf die Phänotypen des Lösungsraums. Abbildung aus Bentley und Corne (2002, S. 11).

b) 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.

c) Evolutions-Strategie (ES)

Die in den 60er-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 findet keine Kodierung der Phänotypen als Genotypen statt (vgl. Abb. 1). Die Variation mittels Kreuzung und Mutation wird direkt auf

die Phänotypen angewandt. Dadurch entfällt das relativ aufwändige Mapping-Verfahren.

d) Evolutionäre Programmierung (EP)
Die EP geht auf Lawrence Fogel zurück (Fogel, Owens, & Walsch, 1966). Sie ist der ES zwar sehr ähnlich, aber parallel zu dieser entwickelt worden.

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 90er-Jahren am CECA[2] 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 des evolutionären Ansatzes im Bereich der computerbasierten Grundrissentwicklung finden sich bei Jo und Gero (1998) sowie bei Rosenman (1997) und bei Hower (1997).

Unter den vielfältigen computerbasierten Grundrisssystemen, die in den letzten Jahren entstanden sind, zählt die Arbeit von Elezkurtaj und Franck (2002), in mehrerlei Hinsicht zu den am besten ausgearbeiteten. Erstens berechnet das System Entwurfsvorschläge in Echtzeit, wodurch zweitens eine sinnvolle Interaktionsmöglichkeit zwischen der generativen Software und dem Nutzer ermöglicht wird, die ein kontinuierliches Wechselspiel zwischen Entwerfer und Computer erlaubt. Drittens kann der Entwurf kontinuierlich weiter entwickelt werden. Viertens sind nicht nur beliebige Rechtecke kombinierbar, sondern auch Polygone (Elezkurtaj & Franck, 2001).

Die vorliegende Arbeit folgt der Strategie, die in der Dissertation von Elezkurtaj (2004) beschrieben wurde. Dies betrifft vor allem die Kombination des Verfahrens zur Lösung des Packungsproblems mit dem zur Lösung des quadratischen Zuordnungsproblems. Allerdings sind in der Dissertation von Elezkurtaj (2004) keine ausreichenden Angaben über die verwendeten Algorithmen und die Methoden zur Kombination beider Verfahren zu finden, um das von Elezkurtaj entwickelte Layout-System zu reproduzieren.

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

3. Formaler Rahmen

Für ein besseres Verständnis der Funktionsweise Evolutionärer Strategien, die im Rahmen der vorliegenden Arbeit in verschiedenen Varianten eingesetzt werden, betrachten wir im Folgenden die wichtigsten Grundlagen in Form eines allgemeinen formalen Rahmens. Wir bemühen uns, alle formalen Zusammenhänge so einfach wie möglich darzustellen. Die formale Notation ist notwendig für ein tieferes Verständnis der Mechanismen der ES und einer kompakten Darstellung aller notwendigen Begriffe und deren Zusammenhänge. Zusätzlich werden in den folgenden Kapiteln alle mathematischen Beschreibungen in möglichst allgemeinverständlicher Sprache beschrieben. Wir orientieren uns im Folgenden an den formalen Darstellungen bei Bäck, Hoffmeister und Schwefel (1991), Deb (2001) sowie Bäck (2000). 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.

Bei den im Folgenden vorgestellten Verfahren beginnen wir 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 mit einer Auflistung der wichtigsten formalen Elemente, welche sich auf eine sogenannte (μ+λ)-ES beziehen. Auf die Bedeutung dieser Schreibweise wird später eingegangen.

$P^\circ = (x^\circ, \sigma^\circ) \in I$	*Population;* $I = R^n \times R^n$	
$a \in \mathbb{N}$	*Individuum*	
$X \in \mathbb{N}$	*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*	(1)
$g_j: \mathbb{R}^n \rightarrow \mathbb{R}$	*Nebenbedingungen* $j \in \{1, \ldots, q\}$	
$t: I \times I \rightarrow \{0,1\}$	*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*	

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).

Mit der Nomenklatur $(\mu+\lambda)$-ES$=(P^\circ, \mu, \lambda, r, m, s, \Delta\sigma, f, g, t)$ 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 $(\mu+\lambda)$-ES werden die Eltern zusammen mit den Kindern bewertet und selektiert. Folglich kann bei der $(\mu+\lambda)$-ES ein Individuum solange überleben, bis es von einem besseren Nachkommen verdrängt wird.

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

$$f(\boldsymbol{X}) \rightarrow Optimum \tag{2}$$

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 Variablen-Kombinationen potenziert. Die Geschwindigkeit, mit der ein EA sich einem Optimum annähert, wird als Konvergenzgeschwindigkeit bezeichnet. Ziel bei der Anwendung eines EAs 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 $f_k(\boldsymbol{X})$:

$$a_k := \left(\boldsymbol{X}_k, f_k\left(\boldsymbol{X}\right)\right) \tag{3}$$

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

$$X_i' = X_i + \mathrm{N}_0(\sigma) \tag{4}$$

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.

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

$$P^{t+1} = s\left(P'^t\right) = \begin{cases} a_2'^t & if \begin{cases} f\left(\mathbf{X}'^t\right) \le f\left(\mathbf{X}^t\right) \wedge \\ g\left(\mathbf{X}'^t\right) \ge 0 \end{cases} \\ a_1'^t \in P^t & else \end{cases} \tag{5}$$

Diese ES wird aufgrund ihres Selektionsschemas (5) 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. Methoden für das Grundriss-Layout

In diesem Kapitel setzten wir uns mit einer Methode auseinander, die für das computerbasierte generieren von Grundriss-Layouts geeignet erscheint. Sie beruht auf dem Prinzip der dichten Packung geometrischer Elemente. Die dargestellten Beispiele wurden mittels C# implementiert[3].

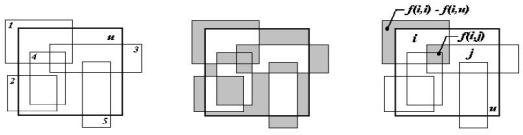
4.1. Dichte Packung

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 Grundriss-Layout 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 Testscenario gehen wir von einem festgelegten Gebäudeumriss aus und befassen uns mit der zweidimensionalen Packung

[3] Die in diesem Text beschriebenen Computerprogramme sind über folgende Internetseite verfügbar: http://infar.architektur.uni-weimar.de/service/drupal-cms/Softwaremuster

einzelner Räume, die in ihren Abmessungen innerhalb definierter Mindest- und Höchstgrenzen flexibel sind, aber immer einen gegebene Flächeninhalt einhalten müssen. Die Summe der Flächeninhalte der zu packenden Räume entspricht dabei dem Flächeninhalt des Gebäudeumrisses.

Das zu lösende Problem wurde von Elezkurtaj und Frank (Elezkurtaj & Franck, 2001, 2002) formal folgendermaßen beschrieben (Abb. 2): Zu minimieren ist die Summe aller Schnittflächen S_g. Diese errechnet sich aus der Summe der Schnittflächen aller zu packender Räume ($S_i \cap S_j$) und der gewichteten Summe der Schnittflächen die sich aus der Überlappung der zu packenden Räume mit dem Umgebungsrechteck ergeben ($S_i \setminus S_u$).



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

Abb. 2: Grafische und formale Darstellung der Berechnung der gesamten Schnittflächen beim Problem der Dichten Packung bei Elezkurtaj und Franck (2001).

4.1.1. Kollisionserkennung

Zur algorithmischen Lösung des in Abb. 2 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. 3 dargestellt, wird geprüft, welcher Abprall-Vektor (v_x oder v_y) der kleinste ist, so dass 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.

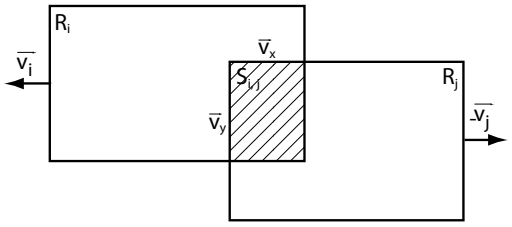


Abb. 3: Berechnung der Abstoßungs-Vektoren zweier sich überlappender Rechtecke.

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.

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 Breite oder Höhe der Rechtecke. Bei dem Beispiel in Abb. 3 würde die Breite von R_i um die Hälfte des Betrags des Vektors v_i verringert, und gleichzeitig die Höhe von R_i erhöht, so dass die Fläche F_i trotz der Proportionsänderung immer 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 Proportionsänderungs-Vektoren 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:

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

$$\overrightarrow{v_i'} = -\overrightarrow{v_j'} = \begin{cases} \begin{pmatrix} 0.5 * v_{xi}' \\ 0 \end{pmatrix} & if \; v_{xi}' > v_{yi}' \\ \begin{pmatrix} 0 \\ 0.5 * v_{yi}' \end{pmatrix} & else \end{cases} \tag{6}$$

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. 4 dargestellten Softwaremuster können dichte Packungen mittels Abprall- und Proportionsänderungsfunktion hergestellt werden. Wir verwenden ein Testscenario mit einem Umgebungsquadrat von 600 x 600 Pixel und 10 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.

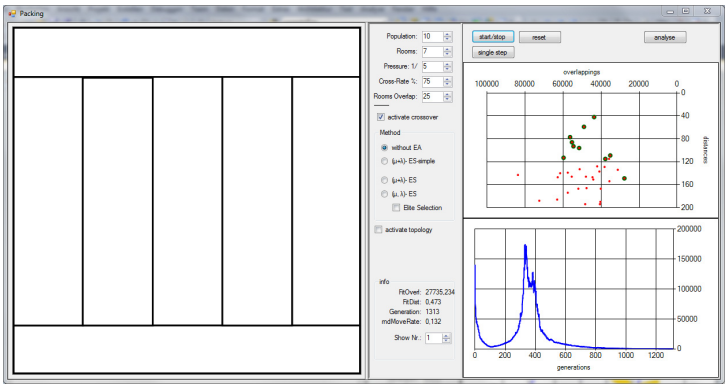


Abb. 4: Beispiel für ein automatisch generiertes Layout (rechts) mittels Abprall- und Proportionsänderungsfunktion. Das Diagramm unten rechts plottet die Überlappungswerte (y-Achse) gegen die Generationen (x-Achse). Das Programm wurde bis Generation t = 1313 ausgeführt.

Aus dem Diagramm in Abb. 4 (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 beschriebene 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 (6) zu Rückkoppelungen kommen kann, durch die bei bestimmten Konstellationen der Rechtecke unerwünschte Veränderungen aufgeschaukelt werden können.

Für eine bessere Bewertung der Systemeigenschaften des in diesem Abschnitt dargestellten Verfahrens führen wir eine Analysemethode (Performancetest) ein, bei der das in Abb. 4 dargestellte Layout-Programm mehrfach ausgeführt wird und die Überlappungswerte aller Programmdurchläufe in einem Diagramm zusammengefasst werden (Abb. 5).

In Abb. 5 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.

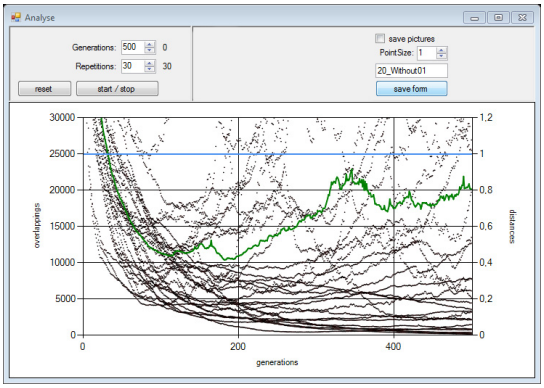


Abb. 5: Analysediagramm für die Layout-Generierung mittels Kollisionserkennung nach 30 Durchläufen des Programms mit jeweils t=500. Die durchgezogene grüne Linie stellt die Mittelwertkurve dar.

Bevor wir uns mit den Möglichkeiten der ES befassen, soll anhand des Diagramms in Abb. 5 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 angenehme Interaktion dadurch angegeben werden kann, dass nach ca. t=20 Generationen 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. 5 können wir feststellen, dass das System im Mittel noch zu träge ist.

4.1.2. (1+1) Evolutions-Strategie

Die einfachste Form einer ES ist die so genannte zweigliedrige $(1+1)$-ES (Rechenberg, 1994). Der Notation aus Kapitel 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, so dass zwei verschieden 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. 2 angegebene Funktion zur Berechnung von S_g. Je kleiner S_g ausfällt, desto höher ist die Qualität eines Individuums:

$$f(\boldsymbol{X}) \rightarrow \min(S_g) \tag{7}$$

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 3 folgend können wir diese Parameter folgendermaßen angeben:

$$X_i = \left(px_i,\ py_i,\ h_i,\ b_i\right) \tag{8}$$

Auf diesen Parametersatz wenden wir nun das iterative Schema (5) der ES an. Die Mutationsoperationen können angegeben werden mit:

$$f(X^t + Z) = \left(px_i^t + Z,\ py_i^t + Z,\ b_i^t + Z\right) \tag{9}$$

Es werden also alle Objektparametern 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 Flächeninhalt 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 weniger ist, verringere σ:

$$\sigma^{t+1} = \begin{cases} c_d \cdot \sigma^t, & if \quad p_s^t < 1/5 \\ c_i \cdot \sigma^t, & if \quad p_s^t > 1/5 \\ \sigma^t, & if \quad p_s^t = 1/5 \end{cases} \tag{10}$$

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:

$$\begin{aligned} px_i'^t &= \begin{cases} px_i^t + \mathrm{N}_0(\sigma) * \kappa & if \quad \rho_Z > U \\ px_i^t & else \end{cases} \\ py_i'^t &= \begin{cases} py_i^t + \mathrm{N}_0(\sigma) * \kappa & if \quad \rho_Z > U \\ py_i^t & else \end{cases} \end{aligned} \tag{11}$$

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. 2 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 vorn herein ausgeschlossen werden, indem die Rechtecke im Falle einer Überlappung gezielt verschoben werden, sodass keine Überlappung mit dem Umgebungsrechtecks mehr auftritt. Diese Alternative vereinfacht die Berechnung von S_g, die wir im Folgenden in dieser Form verwenden:

$$S_g(\mathbf{X}) = f_1^t(\mathbf{X}) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left(S_i \cap S_j \right); \quad f_1^t(\mathbf{X}) \to \min \tag{12}$$

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

$$\begin{aligned} px_i'^t &= \begin{cases} px_i^t + (x_{\min} - px_i^t - 0.5 * b_i^t) & if \quad px_i^t - 0.5 * b_i^t < x_{\min} \\ px_i^t + (px_i + 0.5 * b_i^t - x_{\max}) & if \quad px_i^t - 0.5 * b_i^t > x_{\max} \\ px_i^t & else \end{cases} \\ py_i'^t &= \begin{cases} py_i^t + \left(y_{\min} - py_i^t - 0.5 * h_i^t \right) & if \quad py_i^t - 0.5 * h_i^t < y_{\min} \\ py_i^t + (py_i^t + 0.5 * h_i^t - y_{\max}) & if \quad py_i^t - 0.5 * h_i^t > y_{\max} \\ py_i^t & else \end{cases} \end{aligned} \tag{13}$$

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 diente lediglich der Einführung grundlegender Mechanismen. Aus diesem Grund betrachten wir die Eigenschaften des in diesem Abschnitt vorgestellten Systems nicht im Detail. 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.

4.1.3. 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 (9) wie folgt erweitert:

$$
\begin{aligned}
px_i'^t &= \begin{cases} px_i^t + \mathrm{N}_0(\sigma) * \kappa & if \quad \rho_{Z1} > U \\ px_i^t + \mathrm{N}_0(\sigma) & if \quad \rho_{Z2} > U \\ px_i^t & else \end{cases} \\
py_i'^t &= \begin{cases} py_i^t + \mathrm{N}_0(\sigma) * \kappa & if \quad \rho_{Z1} > U \\ py_i^t + \mathrm{N}_0(\sigma) & if \quad \rho_{Z2} > U \\ py_i^t & else \end{cases} \\
b_i'^t &= \begin{cases} b_i^t + \mathrm{N}_0(\sigma) & if \quad \rho_{Z3} > U \\ b_i^t & else \end{cases}
\end{aligned} \tag{14}
$$

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 (6) der Kollisionserkennung mit den die Parametereinstellungen α=0,1; ρ_G=0,5; β=0,5; ρ_L=1 angewendet. Anschließen werden entsprechend Schema (5) 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 4.1.1 bei der Kollisionserkennung beschrieben wurden, vermieden werden. Das kleine Diagramm in Abb. 6, mitte zeigt den entsprechenden Verlauf von S_g. Ferner ist in Abb. 6, links ein typisches Layout der hybriden ES nach t=215 Generationen zu sehen.

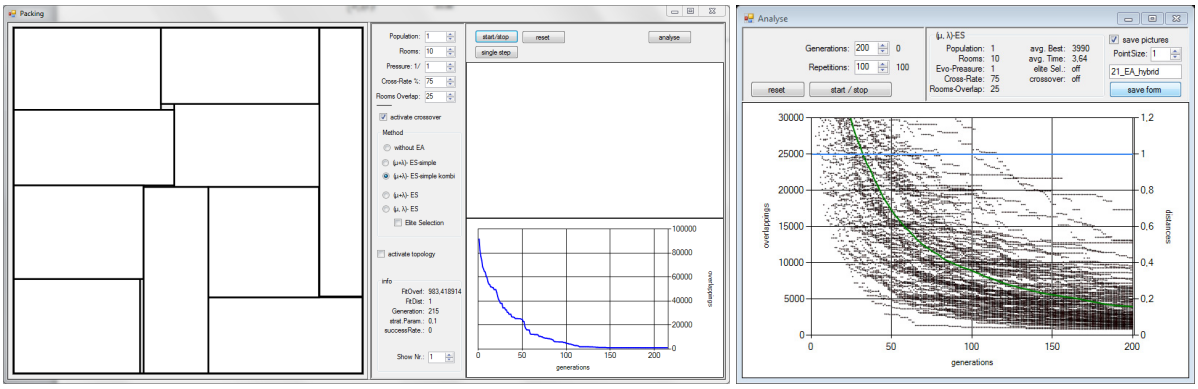


Abb. 6, Links: Typisches Ergebnis der hybriden ES bestehend aus (1+1)-ES und Kollisionserkennung bei t=215. *Rechts:* Analysediagramm für die Layout-Generierung mittels hybrider ES nach 100 Durchläufen des Programms mit jeweils t=200 Generationen. Die durchgezogene grüne Linie stellt die Mittelwertkurve 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. 6, rechts dargestellt. Nach t=200 Generationen erreichen wir einen durchschnittlichen Wert für $S_g \approx 4000$. 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. 7 ist das Ergebnis des Performancetests einer hybriden (1+6)-ES zu sehen. Im Vergleich mit der hybriden (1+1)-ES (Abb. 6) 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. 7 zeigt das Ergebnis des Performancetests einer hybriden (6+6)-ES. Nach t=200 Generationen wird ein durchschnittlicher Wert für S_g≈1900 erreicht.

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.

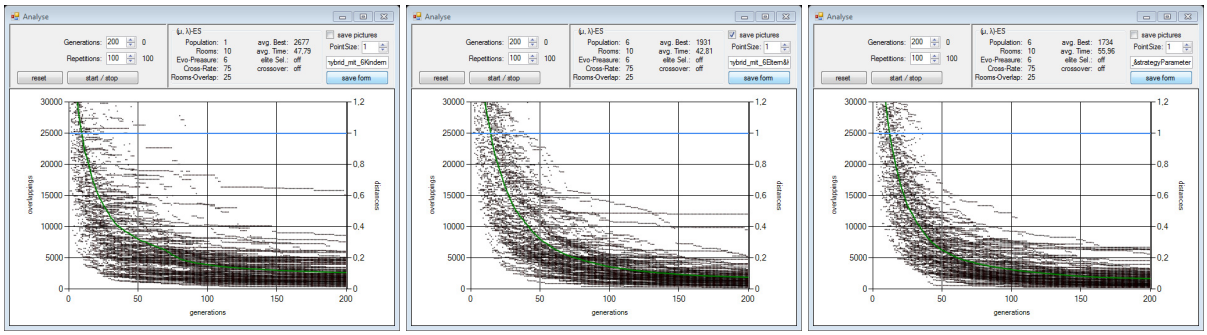


Abb. 7: Analysediagramme für die Layout-Generierung mittels hybrider ES nach je 100 Durchläufen des Programms mit jeweils t=200 Generationen. Die durchgezogenen grünen Linien stellen die Mittelwertkurven dar. *Links:* (1+6)-ES. *Mitte:* (6+6)-ES. *Rechts:* (6+6)-ES mit selbst-adaptivem Strategieparameter.

Eine weitere Variation der hybriden-ES besteht darin, eine oder mehrere Variablen der Anpassungsfunktionen (6) als selbst-adaptiven Strategieparameter (Bäck, 2000; Weicker, 2007, p. 135) zu verwenden:

$$\begin{aligned} \alpha' &= \alpha * \exp\left(\mathrm{N}_0(\sigma)\right) \quad \alpha' \in [0{,}1;\ 1] \\ {}_{\Delta}G_i^t &= \alpha' * r_i^t \qquad\quad if \quad \rho_G > U \end{aligned} \tag{15}$$

wobei σ=0,1. Die Festlegung in (15), dass α' nicht kleiner als 0,1 sein darf, ist dadurch begründet, dass sehr kleine Gewichtungen 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 (15) ist im rechten Diagramm in Abb. 7 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. 7, 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 (6), also für die Selbst-adaption von ρ_G, β, oder ρ_L, zusätzliche Verbesserungen erzielt werden können. Herausgestellt hat sich, dass sich durch die Selbst-adaption dieser Variablen keine positiven Effekte ergeben haben, sondern es im Gegenteil zu Variablenkombinationen kommen kann, welche das System in einem lokalen Optimum halten und eine weitere Optimierung dadurch blockieren.

Bei allen bisher beschriebenen Untersuchungen wurde die (+)-Selektion für die ES verwendet, da diese während der Optimierung kontinuierlicher 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 $(\mu+\lambda)$-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 Imitation sexueller Reproduktion ergeben.

4.1.4. Hybride $(\mu+\lambda)$-ES mit Rekombinationsoperator

Bei der Verwendung mehrere 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: $(\mu/\rho+\lambda)$-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:

$$X_i' = \frac{1}{\rho}\sum_{i=1}^{\rho} X_i \tag{16}$$

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

$$\begin{array}{lllllll} Elter1: & X_1^{(1)} & X_2^{(1)} & X_3^{(1)} & X_4^{(1)} & X_5^{(1)} & X_6^{(1)} \\ Elter2: & X_1^{(2)} & X_2^{(2)} & X_3^{(2)} & X_4^{(2)} & X_5^{(2)} & X_6^{(2)} \\ Elter3: & X_1^{(3)} & X_2^{(3)} & X_3^{(3)} & X_4^{(3)} & X_5^{(3)} & X_6^{(3)} \\ & & & & & & \\ Rekombinant\ \boldsymbol{X}': & X_1^{(3)} & X_2^{(1)} & X_3^{(2)} & X_4^{(2)} & X_5^{(1)} & X_6^{(3)} \end{array} \tag{17}$$

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

$$\begin{aligned} & r\left(P^t\right) = a' = \left(X', \sigma'\right) \in I \quad X' \in \mathbb{R}^n,\ \sigma' \in \mathbb{R}^n \\ & X_i' = \begin{cases} X_{b,i}\ , & U \le 0{,}5 \\ X_{c,i}\ , & U > 0{,}5 \end{cases} \quad \forall i \in \{1,\ldots,n\} \\ & \sigma_i' = \left(\sigma_{b,i} + \sigma_{c,i}\right) * 0{,}5 \quad \forall i \in \{1,\ldots,n\} \end{aligned} \tag{18}$$

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 (8) angegeben, ein Gen X_i den komplette 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, 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, so dass bessere Individuen häufiger für eine Rekombi-

nation 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 (18) 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 angewandt, sondern mit einer bestimmten Wahrscheinlich $\rho_{r,}$ für die für die vorliegende 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 (14) und der selbst-adaptive Strategieparameter (15) bei allen Individuen verwendet.

Bei der Auswertung der in diesem Abschnitt entwickelten hybriden (6/2+6)-ES linken Diagramm in Abb. 8, 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. 7, rechts) entspricht. Ferner ist zu erkennen, dass verglichen mit der letztgenannten Variante die Mittelwertkurve etwas steiler abfällt, was bedeutet, dass brauchbare Layout-Lösungen 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, wodurch diese eine Population sehr schnell dominieren können, 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 verändern, bevor die Kinderindividuen mittels Rekombinations- und Mutati-

onsoperatoren erzeugt werden. Im mittleren Diagramm in Abb. 8 sind die Ergebnisse des Performancetests einer Variante der hybriden (6/2+6)-ES dargestellt, bei der für alle Elternindividuen die Anpassungsfunktionen (6) der Kollisionserkennung mit den Parametereinstellungen α=0,1; ρ_G=0,5; β=0,5; ρ_L=1 angewendet wurde. Die Elternindividuen werden folglich ausschließlich anhand der Mechanismen der Kollisionserkennung verändert, so wie sie in Abschnitt 4.1.1 beschrieben wurden. Wir erinnern uns daran, dass durch die Operationen der Kollisionserkennung unerwünschte Rückkoppelungseffekte auftreten können, welche 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 Layout-Lösung verhindern. Tatsächlich sind in den Verläufen der einzelnen Graphen im mittleren Diagramm in Abb. 8 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. 8, links) entspricht. Ferner ist zu erkennen, dass verglichen mit der letztgenannten Variante die Mittelwertkurve wiederum etwas steiler abfällt.

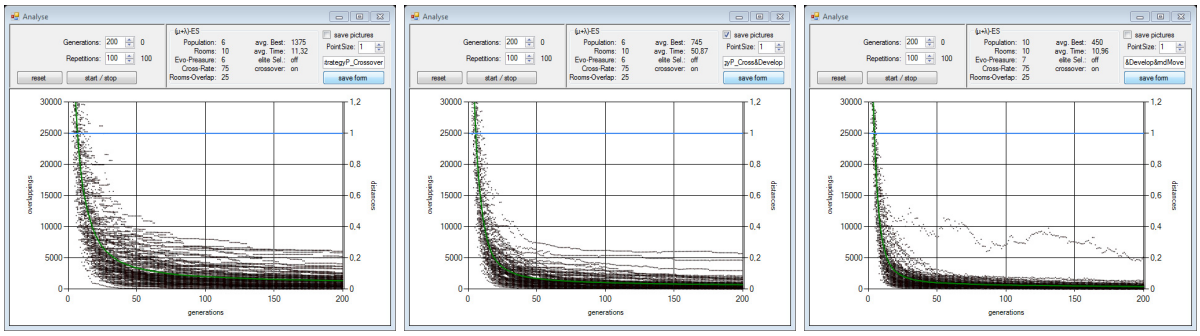


Abb. 8: Analysediagramme für die Layout-Generierung mittels hybrider $(\mu/\rho+\lambda)$-ES nach je 100 Durchläufen des Programms mit jeweils t=200 Generationen. Die durchgezogenen grünen Linien stellen die Mittelwertkurven dar. *Links:* (6/2+6)-ES mit ρ_r=0.75. *Mitte:* (6/2+6)-ES mit ρ_r=0.75 und 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 (11), 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. 8 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. 8, 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. 8, 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 (17) einfacher als das Roulette-Wheel-Verfahren ist, stellen wir in Abb. 9 beide Verfahren gegenüber. Im linken Diagramm in Abb. 9 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 Layout-Systeme, deren Performancediagramme in Abb. 9 dargestellt sind wurde auf die Eltern eine Kollisionserkennung und Zufallsbewegung angewandt. Die Diagramme in der Mitte und rechts in Abb. 9 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.

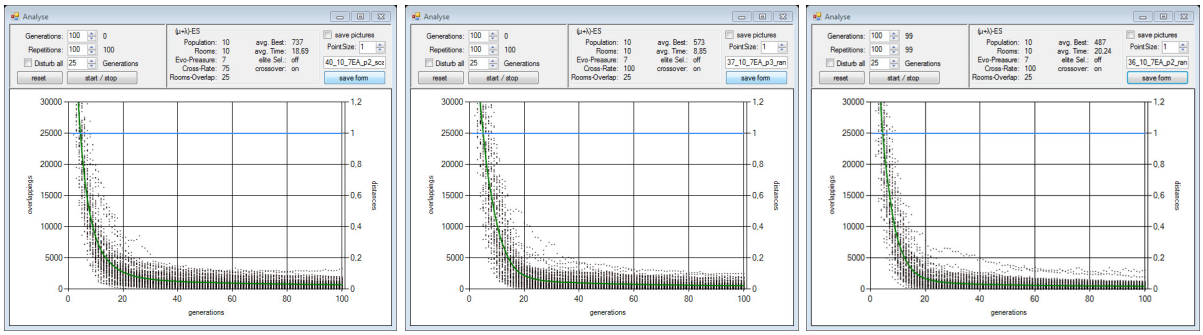


Abb. 9: Analysediagramme für die Layout-Generierung mittels hybrider $(\mu/\rho+\lambda)$-ES nach je 100 Durchläufen des Programms mit jeweils t=100 Generationen. Die durchgezogenen grünen Linien stellen die Mittelwertkurven dar. *Links:* (10/2+7)-ES mit ρ_r=0.75 (Roulette-Wheel-Verfahren). *Mitte:* (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 im Rahmen dieser Arbeit erzielte Performance. Im Folgenden Abschnitt betrachten wir im Kontext der Reaktion des Systems auf Störungen die Unterschiede zwischen einer reinen (10/2,7)-ES und der hier zuletzt vorgestellten (10/2+7)-ES.

4.1.5. Reaktion auf Störungen

Bereits in der Einleitung wurde erwähnt, dass ein wichtiger Teil der vorliegenden Arbeit darin besteht, bei der Bearbeitung von Layout-Aufgaben 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 Aspekt, die bei der Bewertung eines Layouts der menschlichen Intuition obliegen, in das System einfließen zu lassen.

In Abschnitt 4.1.1 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 Layout-System 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 Layout-Systems, indem das Programm 100-mal hintereinander ausgeführt wird und die Entwicklung der S_g-Werte in ein Diagramm geplottet werden (Abb. 10). 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. 10 sind die Ergebnisse der Performancetests dargestellt, die mit verschiedenen ES durchgeführt wurden. Das Linke Diagramm in Abb. 10 zeigt die Ergebnisse der im vorangegangenen Abschnitt 4.1.4 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, so dass sich unmittelbar nach einer Änderung wieder eine gute Layout-Lösung einstellt.

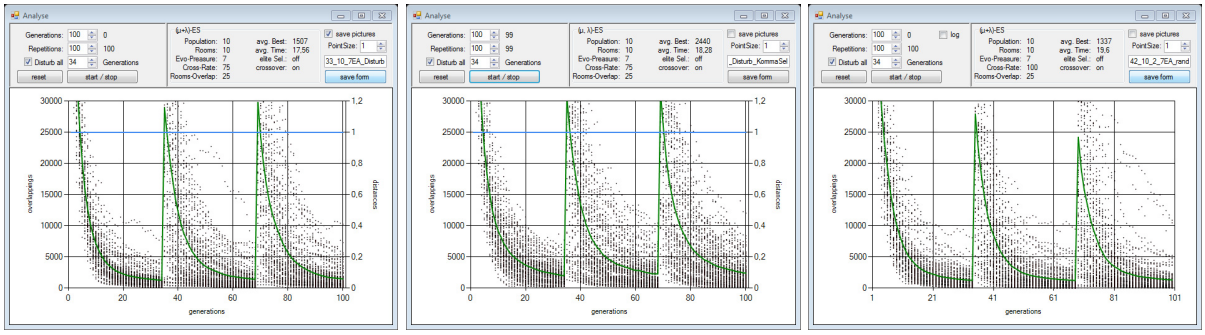


Abb. 10: Analysediagramme für die Reaktion auf Störungen. Die durchgezogenen grünen Linien stellen die Mittelwertkurven dar. Bei allen Varianten erfolgte eine Kollisionserkennung und Zufallsbewegung bei den Eltern. Pro Diagramm wurden 100 Durchläufen des Programms mit jeweils t=100 Generationen aufgezeichnet.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

Im mittleren Diagramm in Abb. 10 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 Individuum 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. 10 zeigt das Ergebnis des Performancetests einer 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 Layout-Systeme aufweist. Die Befürchtung, dass das System nach einer Störung bei einer einmal gefundenen Layout-Variante hängen bleibt, war besonders bei der (+)-Selektion, wie in Abschnitt 4.1.3 beschriebenen, naheliegend. Entgegen dieser Erwartung haben alle in Abb. 10 untersuchten Systemvarianten gezeigt, dass sie schnell auf Störungen reagieren und angepasste Layout-Lösungen 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 der ES angegeben, die im Folgenden als Standard-ES bezeichnet wird:

Schritt 1: Initialisiere eine Population P mit μ Individuen.

Schritt 2: Erstelle λ Kinder durch

a) Mutation der Eltern [Zufallsbewegung (11) und Kollisionserkennung (6)]

b) Rekombination von ρ Eltern [(16)-(18)]

c) Mutation der Rekombinanten [(14) und Kollisionserkennung (6)]

Schritt 3: Selektiere die μ besten Individuen aus Eltern und Kinderpopulation für die nächste Elterngeneration

Schritt 4: Wenn das Abbruchkriterium erfüllt ist, beende den Algorithmus, andernfalls gehe zu Schritt 2.

4.2. 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 von den Rechtecken als Repräsentation für einen Raum und verwenden stattdessen einen Punkte bzw. Knoten, so lassen sich die Raumbeziehungen als Verbindungslinien zwischen den Knoten, sogenannten Kanten darstellen. Auf diese Weise können wir einen Graphen konstruieren, der die topologischen Beziehungen des Layouts abbildet. Knoten und Kante sind die topologischen Grundformen, die graphisch 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. 11). 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 andere topologische Restriktionen finden wird, solange dies mit den geometrischen Variationsmöglichkeiten unsers Systems möglich ist. Formal lässt sich die topologische Restriktion folgendermaßen ausdrücken:

$$f_2^t(\boldsymbol{X}) = \sum_{i=1}^{n}\sum_{j=1}^{n}\left(d_{ij} + c_{ij}\right); \qquad f_2^t(\boldsymbol{X}) \to \min \tag{19}$$

Für die Berechnung der Funktion f werden zu den Abständen d zwischen zwei Rechtrecken 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. 11, rechts), bei der zweiten werden die Räume miteinander vertauscht (Permutation, (Abb. 11, Mitte) und bei der dritten werden virtuelle Federn zwischen den Rechtecken gespannt, zwischen welchen eine topologische Beziehung besteht.

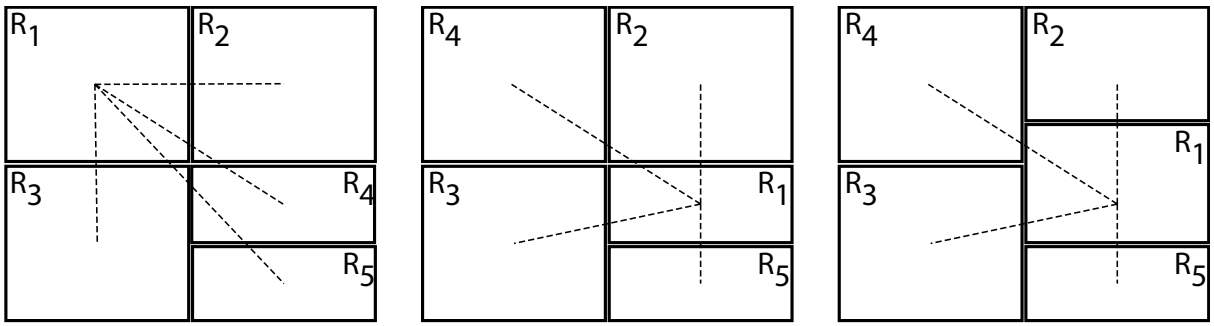


Abb. 11: Erfüllung der geforderten topologischen Beziehungen (gestrichelte Linien) durch Permutation (vgl. linke und mittlere Layout-Konfiguration) und Proportionsänderung (vgl. mittlere und rechte Layout-Konfiguration). Die Darstellung berücksichtigt nicht den Flächenerhalt der einzelnen Räume.

Die Funktion zur Proportionsänderung der Räume wurde bereits mit Gleichung (6) angegeben. Die Funktionen für Permutation und virtuelle Federn werden in den folgenden beiden Abschnitten erläutert.

4.2.1. Permutation

Nach einer Konfiguration der Räume, welche die gewünschten topologischen Beziehungen erfüllt, kann gesucht werden indem man die Räume miteinander vertauscht. 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 Layout-Lösung 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 Invertiert.

4.2.2. 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 4.1.1. Zuerst wird ermittelt, ob es einen Abstand $d_{i,j}$ zwischen den beiden Rechtecken gibt (Abb. 12). Wenn $d_{i,j} > 0$, wird geprüft, ob die x- oder y-Vektorkomponente größer ist (20). Die Anziehung der beiden Rechtecke erfolgt in die Richtung der größeren Vektorkomponente (20). Die Anziehungsvektoren ergeben sich folglich entweder für die x- oder y-Richtung, sodass sich die Rechtecke nur orthogonal aufeinander zu bewegen.

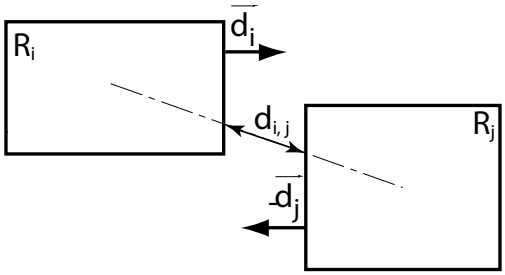


Abb. 12: Berechnung der Anziehungs-Vektoren zweier Rechtecke, die zueinander benachbart liegen sollen.

$$\vec{d'}_i^t = \gamma * \vec{d}_i^t$$
$$\vec{d}_i = -\vec{d}_j = \begin{cases} \begin{pmatrix} 0.5 * d_{xi} \\ 0 \end{pmatrix} & if\ d_{xi} > d_{yi} \\ \begin{pmatrix} 0 \\ 0.5 * d_{yi} \end{pmatrix} & else \end{cases} \quad (20)$$

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.

4.2.3. Multikriterielles Optimierungsproblem (MOOP)

Mit der Einführung der Raumbeziehungen als weitere Restriktion haben wir nun zwei Zielfunktionen f_1 (12) und f_2 (19), die beide durch das Layout-System 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 (pareto-optimale Front) 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 Arbeit nicht nachvollzogen werden.

Wir untersuchen 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) betrachtet 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 4.1.5 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 (6)) 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 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 das 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:

$$V^t = \begin{cases} (\mu*0.4 \ / \ \mu*0.6) & if \quad f_2^t(\mathbf{X}_{(f_1 \min)}) < \Psi_V; \quad \Psi_V = 2*N_R \\ (\mu*0.1 \ / \ \mu*0.9) & else \end{cases} \tag{21}$$

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:

$$Selektion \quad s = \begin{cases} (,) & if \quad f_2^t(\mathbf{X}_k) < \Psi_S; \qquad \Psi_S = 4 * N_R \\ (+) & else \end{cases} \tag{22}$$

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 Paretofronten oft nicht überwinden konnte.

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

Schritt 1: Initialisiere eine Population P mit μ Individuen.

Schritt 2: Erstelle λ Kinder durch
a) nur bei (+) Selektion, vgl. (22): Mutation der Eltern [Zufallsbewegung (11), virtuelle Federn (20) und Kollisionserkennung (6)]
b) Rekombination von ρ Eltern [(16)-(18)]
c) Mutation der Rekombinanten [(14), Permutation (vgl. Punkt 4.2.1), Kollisionserkennung (6) und virtuelle Federn (20)]

Schritt 3: Selektiere die μ besten Individuen nach den Regeln (21) und (22) für die nächste Elterngeneration

Schritt 4: Wenn das Abbruchkriterium erfüllt ist, beende den Algorithmus, andernfalls gehe zu Schritt 2.

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

Auf die gleiche Art und Weise, wie wir in Kapitel 4.2 vorgegangen sind, betrachten wir nun die Performance der im vorangegangenen Abschnitt beschriebenen einfachen (10/2+5)-MOES. In Abb. 13 sind die Ergebnisse der Performancetests dargestellt, die jeweils mit verschieden vielen Räumen bei Vorgabe einer Sterntopologie durchgeführt wurden. Das linke Diagramm in Abb. 13 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 4.1.5 damit zusammenhängt, das wir hier ein weiteres Kriterium (f_2) zu berücksichtigen haben.

Das mittlere Diagramm in Abb. 13 zeigt die Ergebnisse des Performancetests für 8 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 8 Räumen oftmals keine global optimale Lösung innerhalb der betrachteten 100 Generationen. Die Ergebnisse des Performancetests für 9 Räume sind im rechten Diagramm in Abb. 13 dargestellt, welche offensichtlich noch schlechter ausfallen als bei 8 Räumen.

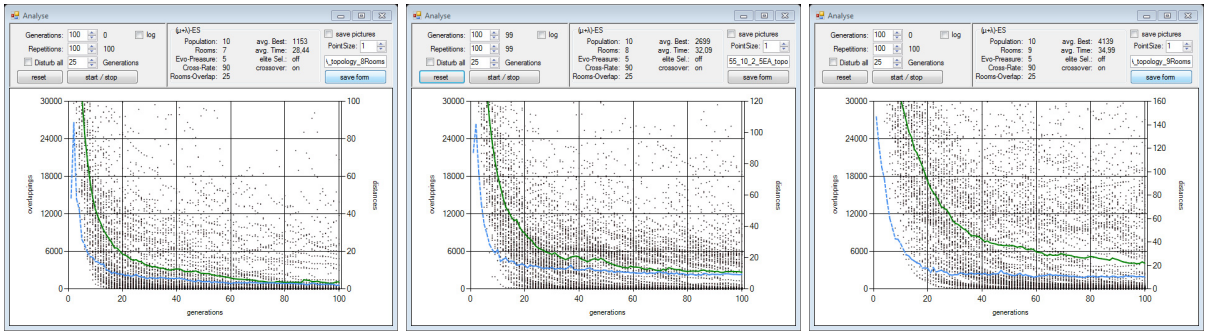


Abb. 13: Analysediagramme für die Layout-Generierung mittels MOEA nach je 100 Durchläufen des Programms mit jeweils t=100 Generationen. Die durchgezogenen grünen Linien stellen die Mittelwertkurven für die Überlappungsflächen f_1 und die gestrichelten blaue Linien die Mittelwertkurven für die Summen der Distanzen f_2 dar. Als Topologische Restriktion wurde eine Sterntopologie verwendet (vgl. *Abb. 14*). *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 und an der pareto-optimalen Front liegen.

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. 13 untersucht. Die Frage nach der Diversität der Lösungen wird in Abb. 14 beantwortet. Im

linken Feld finden sich 11 grundlegend verschiedene geometrische Varianten für 7 Räume, deren Nachbarschaften eine Sterntopologie darstellen. Diese 11 Varianten wurden aus den Ergebnissen der 100 Durchläufe des Systems nach t=100 Generationen ausgewählt (vgl. Abb. 13, links). Unter den 100 Ergebnissen befanden sich viele Variationen dieser 11 Grundvarianten, die gespiegelte oder gedrehte Anordnungen der Räume umfassten. In der untersten Reihe in allen Feldern in Abb. 13 sind typische unzureichende Layout-Lösungen dargestellt, die bei 7, 8 oder 9 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. 13 finden sich die gefundenen 12 Grundvarianten, die aus den Ergebnissen der 100 Durchläufe des Systems nach t=100 Generationen ausgewählt wurden (vgl. Abb. 13, Mitte). Dementsprechend finden sich im rechten Feld in Abb. 13 die gefundenen 11 Grundvarianten, die aus den Ergebnissen der 100 Durchläufe des Systems nach t=100 Generationen ausgewählt wurden (vgl. Abb. 13, rechts). Die mit der Anzahl an Räumen zunehmend schlechtere Qualität der Layout-Lösungen, die bereits oben erwähnt wurde, ist auch im Vergleich der Felder in Abb. 13 zu erkennen.

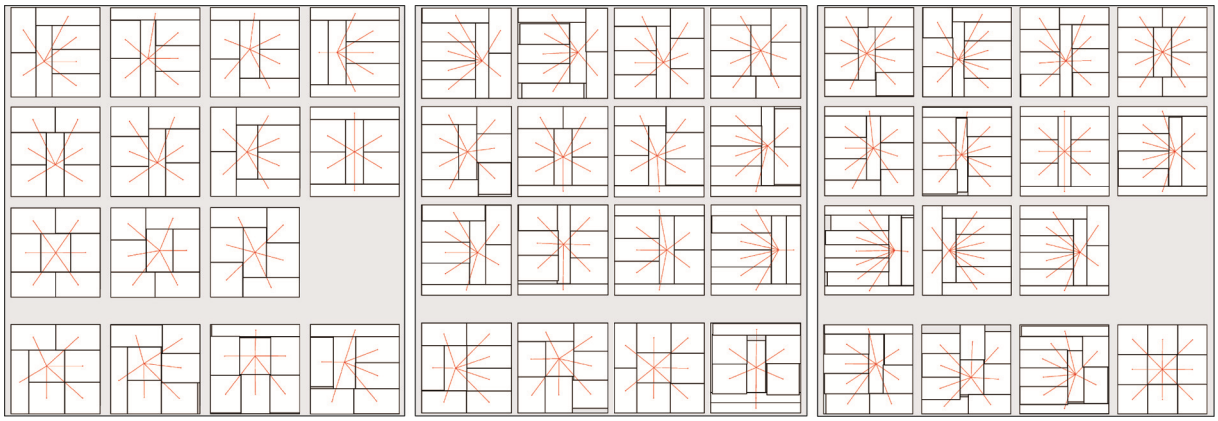


Abb. 14: Verschiedene Layouts mit Sterntopologie nach je 100 Durchläufen des Programms mit jeweils t=100 Generationen. *Links:* 7 Räume. *Mitte:* 8 Räume. *Rechts:* 9 Räume. In den oberen drei Reihen sind typische Layout-Lösungen abgebildet. In der unteren Reihe finden sich typische unzureichende Layout-Lösungen.

Bei der visuellen Auswertung der Layout-Lösungen in Abb. 13 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. Das sich eine solche Lösung bei keiner der in Abb. 13 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 (Gleichung (6)) zustande. Folglich bietet das soweit vorgestellte System nur bedingt die oben genannte Ideale Eigenschaft, die potentielle Diversität möglicher Lösungen vollständig abzudecken.

5. Konklusion und Ausblick

Das Ziel der vorliegenden Arbeit bestand darin, ein System zur Lösung von Layoutproblemen in Architektur und Städtebau zu entwickeln, welches anfangs mit möglichst wenig Problemwissen auskommt (vgl. Punkt 2) und schnell brauchbare Ergebnisse liefert, die durch schrittweises Hinzufügen von Problemwissen interaktiv weiter ausgearbeitet werden können.

Die hier vorgestellten Ergebnisse belegen die Fruchtbarkeit des Ansatzes, 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 (2000). 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 hier unter einem kreativ arbeitenden Computersystem verstehen, wurde in Abschnitt 2 beschrieben. Der Autor des vorliegenden Texts ist der Ansicht, dass verglichen mit ähnlichen wissensbasierten Systemen mit dem hier vorgestellten evolutionären Layout-System ein wesentlich kreativeres System 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, forthcoming).

Eine zentrale Schwierigkeit ergab sich bei der Kalibrierung des Layout-Systems, 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 4.1.3 wurde dargestellt, dass mittels eines selbst-adaptiven Strategieparameters (Bäck, 2000; 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 (21) und (22)), 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 Layout-System bis zu einem bestimmten Komplexitätsgrad, der sich anhand der Anzahl an Räumen und Relationen bestimmen lässt, gute Ergebnisse liefert, die eine brauchbare Unterstützung eines Planers 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 Layout-System 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 könnte die Ausrichtung und Belichtung verschiedener Räume (Lobos & Donath, 2010), Sichtbarkeitsanalysen (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. 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.

Glossar

Fitnesswert:
Der Fitnesswert ist ein Kennwert, der die Güte einer speziellen Lösung bzw. eines Individuums hinsichtlich eines Optimierungsproblems beschreibt. Ziel der Optimierung ist es, eine Lösung mit einem möglichst guten Fitnesswert zu finden.

Fitnessfunktion = Zielfunktion:
Eine Fitnessfunktion berechnet den Fitnesswert eines Individuums. Dabei kann sie z. B. die Position des Individuums im Such- oder Lösungsraum berücksichtigen.

Individuum:
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. Ein Individuum ist bei Evolutionsstrategie nicht codiert, sondern liegt meist als Vektor von n-reellwertigen Variablen vor. Gemäß dem biologischen Vorbild werden bei vielen Heuristiken Punkte im Suchraum Individuen genannt.

Multi-kriterielles Optimierungsproblem (MOOP):
Ein Problem, bei dem mehrere Zielfunktion optimiert (minimiert oder maximiert) werden sollen. Dabei können die Zielfunktionen entweder so beschaffen sein, dass für alle ein Optimum erreichet werden kann oder die Zielfunktionen wiedersprechen sich gegenseitig. In letzterem Fall kann lediglich ein Satz von pareto-optimalen bzw. nicht dominierten Lösungen gefunden werden, von denen hinsichtlich der Zielfunktionen keine besser oder schlechter ist als die andere.

Mutation:
Zufallsgesteuerte Veränderung des Werts eines Objektparameters mit einer bestimmten Wahrscheinlichkeit. Mutationen führen meist zu einer Verschlechterung des Fitnesswerts eines Individuums, können allerdings manchmal zu äußerst produktiven Sprüngen in neue Gebiete des Lösungsraums führen.

Objektparameter:
Objektparameter bezeichnen die Eigenschaften oder Parameterwerte einer Lösung und spannen den Suchraum auf. Eine beliebige Kombination von Objektparameterwerten stellt

einen Punkt im Suchraum dar, der mittels Fitnessfunktion auf einen Punkt im Lösungsraum gemappt werden kann. Die Position des Punktes im Lösungsraum gibt die Qualität einer Lösungsvariante an.

Population:
Als Population wird eine Menge strukturell gleichartiger Individuen bezeichnet.

Rekombination:
Als Rekombination bezeichnet man die Mischung, bzw. Neu-Kombination der Objektparameter verschiedener Individuen.

Such- und Lösungsraum:
Eine der wichtigsten Eigenschaften eines GA ist die Trennung von Such- und Lösungsraum. Der Suchraum beinhaltet die kodierten Lösungen, die Genotypen, welche durch Kreuzung und Mutation variiert werden. Die kodierten Lösungen werden anhand eines Verfahrens, welches als Mapping (oder auch als Embryogenie) bezeichnet wird, in den Lösungsraum überführt und bilden dort die Phänotypen, die dem Selektionsprozess ausgesetzt werden.

Referenzen

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

Bäck, T. (2000). 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

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

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.

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

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., Lamont, G. B., & Veldhuizen, D. A. v. (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems* (2 ed.). Berlin, Heidelberg: Springer.

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. (1988). *Logic Models of Design*. London: Pitman Publishing.

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

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

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

Duarte, J. P. (2000). *Customizing mass housing: a discursive grammar for Siza's Malagueira houses.* Massachusetts Institute of Technology.

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

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.

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

Franck, G. (2009). Maschinelle Entwurfshilfen: Was lehren Künstliche Intelligenz und Künstliche Kreativität über das architektonische Denken? In D. Gethmann & S. Hauser (Eds.), *Kulturtechnik Entwerfen. Praktiken, Konzepte und Medien in Architektur und Design Science* (pp. 227-242). Bielefeld: Transcript

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

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

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

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.

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.

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.
Kirsch, W. (1997). *Die Handhabung von Entscheidungsproblemen: Einführung in die Theorie der Entscheidungsprozesse* (5 ed.). München: Verlag Barbara Kirsch.
Koza, J. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*: MIT Press.
Kursawe, F. (1990). A variant of evolution strategies for vector optimization *Parallel Problem Solving from Nature I* (pp. 193-197).
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).
March, L., & Steadman, P. (1974). *The geometry of environment: an introduction to spatial organization in design* (2nd ed.): M.I.T. Press.
Medjdoub, B., & Yannou, B. (2001). Dynamic space ordering at a topological level in space plannung. *Artificial Intelligence in Engineering, 15*, 47-60.
Rechenberg, I. (1994). *Evolutionsstrategie '94*: Frommann Holzboog.
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.
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. (forthcoming). *Who cares about right angles? - Overcoming barriers in creating rectangularity in floor plan structures.* Paper presented at the 29th eCAADe 2011, Ljubljana.
Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*: Wiley-Interscience.
Simon, H. (1969). *The Science of the Artificial*: MIT Press.
Sims, K. (1999). Evolving Three Dimensional Morphology and Behaviour. In P. J. Bentley (Ed.), *Evolutionary Design by Computers*: Morgan Kaufmann.
Stiny, G., & Mitchell, W. J. (1978). The Palladian Grammar. *Environment and Planning B: Planning and Design, 5*(1), 5-18.
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.