Bauhaus-Universität Weimar
Informatik in der Architektur | InfAR

Arbeitspapiere
Working Papers

Nr. 10, Dezember 2011

Evolutionäre Generierung von Grundriss-Layouts mithilfe von Unterteilungsalgorithmen

Katja Knecht, Reinhard König

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

Katja Knecht, Reinhard König
Evolutionäre Generierung von Grundriss-Layouts mithilfe von Unterteilungsalgorithmen

Weimar 2011
Arbeitspapiere (Working Papers) Informatik in der Architektur, Nr. 10
Herausgegeben von Prof. Dr. Dirk Donath und Dr. Reinhard König
ISSN 2191-2416

Bauhaus-Universität Weimar, Professur Informatik in der Architektur
Belvederer Allee 1, 99425 Weimar
http://infar.architektur.uni-weimar.de

Redaktionelle Anmerkung:

M.Sc., Dipl. Ing. (FH) Katja Knecht ist wissenschaftliche Mitarbeiterin an der Professur Informatik in der Architektur an der Bauhaus-Universität Weimar. 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

Evolutionäre Generierung von Grundriss-Layouts mithilfe von Unterteilungsalgorithmen

Katja Knecht, Reinhard König
katja.knecht@uni-weimar.de, reinhard.könig@uni-weimar.de
Professur Informatik in der Architektur
Fakultät Architektur, Bauhaus-Universität Weimar, Belvederer Allee 1, 99421 Weimar, Germany

Abstract

Das Unterteilen einer vorgegebenen Grundfläche in Zonen und Räume ist eine im Architekturentwurf häufig eingesetzte Methode zur Grundrissentwicklung. Für deren Automatisierung können Unterteilungsalgorithmen betrachtet werden, die einen vorgegebenen, mehrdimensionalen Raum nach einer festgelegten Regel unterteilen. Neben dem Einsatz in der Computergrafik zur Polygondarstellung und im Floorplanning zur Optimierung von Platinen-, Chip- und Anlagenlayouts finden Unterteilungsalgorithmen zunehmend Anwendung bei der automatischen Generierung von Stadt- und Gebäudegrundrissen, insbesondere in Computerspielen.

Im Rahmen des Forschungsprojekts Kremlas wurde das gestalterische und generative Potential von Unterteilungsalgorithmen im Hinblick auf architektonische Fragestellungen und ihre Einsatzmöglichkeiten zur Entwicklung einer kreativen evolutionären Entwurfsmethode zur Lösung von Layoutproblemen in Architektur und Städtebau untersucht. Es entstand ein generativer Mechanismus, der eine Unterteilungsfolge zufällig erstellt und Grundrisse mit einer festgelegten Anzahl an Räumen mit bestimmter Raumgröße durch Unterteilung generiert. In Kombination mit evolutionären Algorithmen lassen sich die erhaltenen Layoutlösungen zudem hinsichtlich architektonisch relevanter Kriterien optimieren, für die im vorliegenden Fall Nachbarschaftsbeziehungen zwischen einzelnen Räumen betrachtet wurden.

Keywords: Grundrissgenerierung, Optimierung, Evolutionäre Algorithmen, Genetische Algorithmen, Genetische Programmierung, Unterteilungsalgorithmen, Computational Design

1. Einleitung

Unterteilungsalgorithmen sind Algorithmen, die Flächen oder mehrdimensionale Räume nach bestimmten Regeln oder einer festgelegten Abfolge unterteilen. Eingesetzt wurden sie zunächst in der Computergrafik zur Unterteilung von Polygonen, um gekrümmte Flächen im dreidimensionalen Raum annähernd darstellen zu können (Catmull & Clark, 1978). Da sich Unterteilungsalgorithmen zudem dadurch auszeichnen, dass Elemente durch Unterteilung einer vorgegebenen Grundfläche dicht gepackt werden können, fanden sie ein weiteres Einsatzgebiet im Floorplanning (Young & Wong, 1997). Floorplanning betrachtet die Optimierung von Lagebeziehungen zwischen Bauteilen (Otten, 1982), wie beispielsweise im Platinen-, Chip- oder Anlagendesign, oder auch zwischen Einrichtungen in Gebäuden (Kado, 1995).

Der Vorgang des Unterteilens imitiert eine in Architektur und Stadtplanung oft genutzte Entwurfsmethodik. Diese Imitation, jedoch zunächst ohne architektonischen Anspruch, machte sich die Computerspielebranche zu Nutze und verwendete Unterteilungsalgorithmen in der automatischen Generierung von Stadtgrundrissen und -strukturen bis hin zu Gebäudegrundrissen, um flexible Spielwelten zu schaffen und begehbar zu machen (Hahn, Bose, & Whitehead, 2006; Müller et al., 2006). Marson und Musse (2010) haben die Verwendung von quadratisierten Unterteilungsbäumen zur Echtzeitgenerierung von architektonisch sinnvollen Grundrissen und deren Einsatzmöglichkeiten untersucht.

Der Einsatz von Algorithmen zur Generierung von Grundrissen wird bereits seit einiger Zeit erforscht. Da der Entwurf und die Entwicklung von Layouts funktionale und gleichzeitig kreative Lösungen erfordern, werden dabei zunehmend evolutionären Methoden eingesetzt (Elezkurtaj & Franck, 2002). Im Rahmen eines Forschungsprojekts zur Suche nach einer kreativen algorithmischen Methode zur Lösung von Layoutaufgaben in Architektur und Städtebau (Kremlas) haben wir uns mit dem generativen und gestalterischen Potential von Unterteilungsalgorithmen auseinandergesetzt und insbesondere untersucht, wie sie in Kombination mit genetischen Algorithmen und genetischer Programmierung zur Grundrissgenerierung eingesetzt werden können.

Unsere Untersuchungen und ihre Ergebnisse dokumentiert dieses Arbeitspapier in den folgenden Kapiteln. Kapitel 2 gibt zunächst eine Einführung in Unterteilungsalgorithmen und ihre Darstellungsformen und beleuchtet ihre Vor- und Nachteile im Hinblick auf die Generierung von architektonischen Grundrissen. Der generative Mechanismus, der im Rahmen der Untersuchungen zur Generierung von Unterteilungslayouts entwickelt wurde, wird in

Kapitel 3 vorgestellt. Kapitel 4 betrachtet die Optimierungsprobleme der Grundrisse, die mithilfe von evolutionären Algorithmen gelöst wurden, sowie den entwickelten Softwareprototyp.

2. Unterteilungsalgorithmen

Im Rahmen unserer Untersuchungen verstehen wir unter einem Unterteilungsalgorithmus die rekursive Unterteilung einer Fläche durch kantenparallele Schnittlinien in kleinere Rechtecksflächen. Die Unterteilung kann auf die entstehenden Unterflächen bis zu einer festgelegten Tiefe oder nach einer festgelegten Abfolge weiter angewendet werden (Tab.1) (Otten, 1982). Die Auswahl der Schnittdimension und die Bestimmung der Lage der Unterteilung kann zufällig oder nach festgelegten Regeln erfolgen, d.h. zum Beispiel, dass eine Fläche immer nach der längeren Seite, in einem festgelegten Proportionsverhältnis (Abb. 1, links), oder so unterteilt wird, dass alle resultierenden Unterräume den gleichen Flächeninhalt besitzen (Abb. 1, rechts).

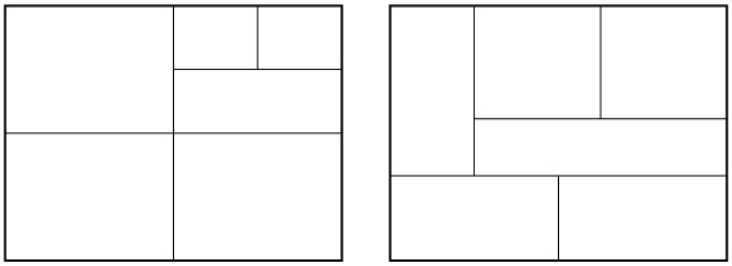


Abb. 1: Unterteilung im Verhältnis 1:1 (links) und nach gleichen Raumgrößen (rechts)

Algorithmus:	Unterteilung einer Fläche und Aufbau eines Unterteilungsbaums
Input:	Rechtecksfläche $\boldsymbol{R}$
Output:	Unterteilungslayout, $\boldsymbol{t}$ vom Typ **Slicing Tree**
Logik:	1. If $\boldsymbol{R}$ ist leer return leeres **Slicing Tree** 2. Bestimme und generiere die Schnittebene **s** sowie den Unterteilungsknoten $\boldsymbol{N}$ mit den folgenden Werten: a. **SplitDim** = die Schnittdimension b. **SplitVal** = der nach der Schnittabfolge bzw. Schnittverhältnis berechnete Schnittlinienwert in SplitDim 3. Bestimme die rechten und linken Unterflächen **Rleft** und **Rright**: a. **Rleft** = Teilfläche von $\boldsymbol{R}$ links oder oberhalb von $\boldsymbol{s}$, mit Mittelpunktsvektor v[SplitDim] ≤ SplitVal b. **Rright** = Teilfläche von $\boldsymbol{R}$ rechts oder unterhalb von $\boldsymbol{s}$, mit Mittelpunktsvektor v[SplitDim] > SplitVal

	4. **tleft** = linker Ast; Unterteile die Fläche ab Schritt 2 weiter rekursiv mit **Rleft** bis das festgelegte Abbruchkriterium erreicht ist 5. **tright** = rechter Ast; Unterteile die Fläche ab Schritt 2 weiter rekursiv mit **Rright** bis das festgelegte Abbruchkriterium erreicht ist 6. return ***t***

Tab. 1: Schematischer Unterteilungsalgorithmus und Aufbau eines Slicing Trees

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

2.1. Darstellungsformen

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

2.1.1. Slicing Floorplan

Die dem Unterteilungsalgorithmus entsprechende geometrische Darstellungsform ist der konkrete Unterteilungsgrundriss, der *Slicing Floorplan*. Er wird durch die rekursive Unterteilung einer vorgegebenen, rechteckigen Fläche in kleinere Rechtecksflächen erstellt (Abb. 2). Die Unterteilung erfolgt durch eine Abfolge von horizontalen und vertikalen Schnitten von Kante zu Kante (Valenzuela & Wang, 2002), die wie bereits beschrieben nach bestimmten Regeln oder zufällig erfolgen kann.

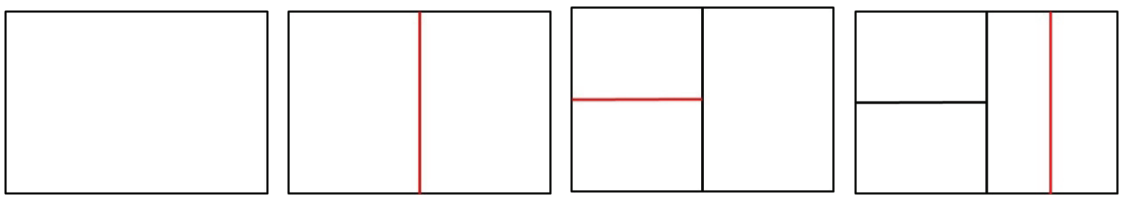


Abb. 2: Erstellung eines Slicing Floorplans

2.1.2. Unterteilungssyntax

Die Unterteilungssyntax ist eine Zeichenkette, die die Abfolge von Unterteilungen in horizontaler oder vertikaler Richtung sowie die Lage der Räume in einem Layout beschreibt.

Die Zeichenkette besteht aus Operatoren, strukturierenden Zeichen und Operanden, den sogenannten Terminals.

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

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

Da sich unsere Untersuchungen zunächst auf binäre Datenstrukturen beschränken, gehen von jedem Operator zwei Unterebenen ab, die der linken bzw. oberen und der rechten bzw. unteren Teilungsfläche entsprechen. Abbildung 3 zeigt eine Unterteilungssyntax (unten) und ihre grafische Darstellung als Layoutstruktur (oben).

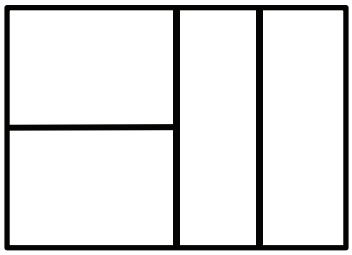


Abb. 3: Grafische Interpretation der Unterteilungssyntax V(H(o)(o))(V(o)(o))

2.1.3. Slicing Tree

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

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

tens bzw. die einem Operator folgenden, von einem Klammerpaar umschlossenen Unterebenen.

Mit jedem entstandenen Unterrechteck bzw. mit jeder aufgehenden Klammer wird der Baum um eine Ebene erweitert. Die Interpretation springt eine Ebene nach unten. Es folgt entweder ein weiterer interner Knoten mit einem Operator, der zu einer weiteren Unterteilung der Fläche und einer weiteren Verästelung führt, oder ein Terminal, welches den Endpunkt der aktuellen Verästelung darstellt. Mit jeder schließenden Klammer bzw. jedem Raum springt die Interpretation eine Ebene im Baum zurück nach oben und erreicht mit der letzten Klammer bzw. Unterteilung wieder ihren Ausgangspunkt, die Wurzel. Die Baumrepräsentation einer Syntax ist in Abbildung 4 (rechts) zu sehen.

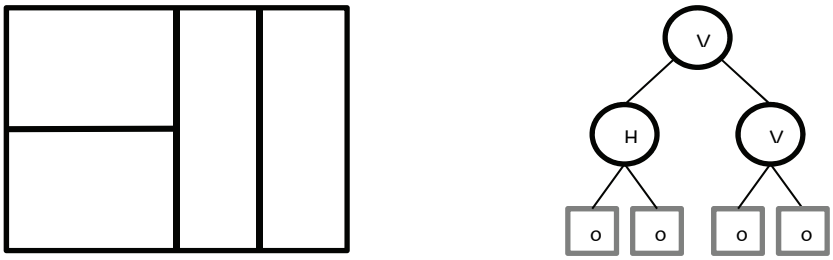


Abb. 4: Interpretation der Unterteilungssyntax V(H(o)(o))(V(o)(o)) bzw. des Slicing Floorplans (links) als Slicing Tree (rechts)

2.2. Unterteilungsalgorithmen im Architekturentwurf

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

Unterteilungen ermöglichen die hierarchische Gliederung eines Layouts in Zonen und Räume. Die so entstandene Grundrisstopologie kann als Baumstruktur dargestellt werden, die einer *Slicing Tree* Struktur ähnelt. In der hierarchischen Ordnung der Baumstruktur lassen sich beispielsweise die Abhängigkeiten der Zonen und Räume innerhalb des Layouts abbilden und ablesen. Funktionale Einheiten repräsentieren in diesem Fall innere Knoten und können eine Gesamtfläche beispielsweise in private oder öffentliche Zonen unterteilen (Marson & Musse, 2010) (Abb. 5, links). Die Blätter oder externen Knoten des *Slicing Tree* stellen die eigentlichen Räume des Grundrisses dar und entsprechen den in der Struktur geschaffenen Rechtecken (Abb. 5, rechts).

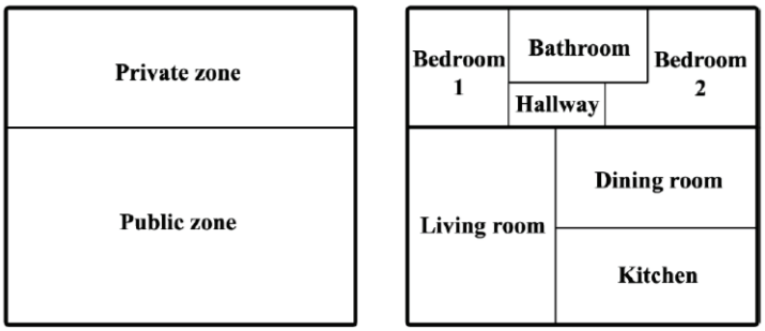


Abb. 5: Hierarchische Ordnung in Grundrissen: Zonierung (links), Raumaufteilung (rechts), entnommen aus Lopes, Tutenel, Smelik, de Kraker und Bidarra (2010)

Die Codierung des Grundrisses als Baumstruktur bzw. als Unterteilungssyntax vereinfacht dessen Verarbeitung und bildet folglich die Grundlage für die automatische Generierung von Layouts. Dabei muss beachtet werden, dass sich der Lösungsraum aller möglichen Grundrisse durch die Generierung mithilfe von Unterteilungsalgorithmen einschränkt, da sich viele existierende Grundrissformen nicht durch Unterteilung erstellen lassen. Hierzu gehören zum Beispiel verwinkelte Räume wie L-Formen (Abb. 6, rechts), ringartige oder netzartige räumliche Verbindungen und offene Grundrisse.

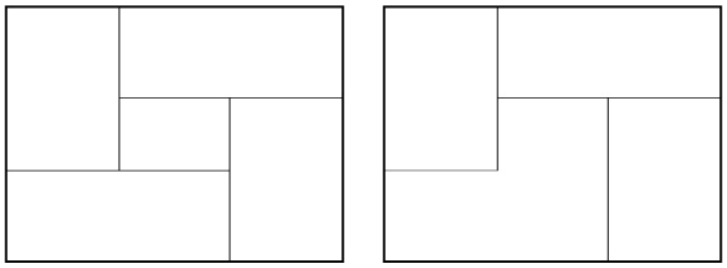


Abb. 6: Layoutbeispiele, die nicht durch Unterteilung erstellt werden können

3. Generierung von Grundrisslayouts durch Unterteilung

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

3.1. Allgemeine Datenstruktur

Die binäre Baumstruktur des *Slicing Trees* bildet das zentrale Element der Datenstruktur. Die Unterteilungssyntax bildet den Ausgangspunkt für den Aufbau dieser Datenstruktur. Der Aufbau der Baumstruktur kann auf Basis der Generierung der Syntax erfolgen. Der Algorithmus hierzu wurde wie in Tabelle 1 beschrieben umgesetzt. Im Baum ist die Unterte-

lungsfolge als Verbindungen zwischen Knoten und Blättern abgelegt. Die Eigenschaften der Räume des späteren Grundrisses sind den Blättern zugeordnet und werden über die Raumindizes referenziert.

3.2. Generierung von Unterteilungssyntax und Slicing Tree

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

Bei der Bildung der Syntax lassen sich zwei, aus der genetischen Programmierung entlehnte Strategien anwenden, die anhand der Baumrepräsentation verdeutlicht werden können: Die Initialisierung als *full tree* bzw. als *grow tree*. Im ersten Fall werden alle Knoten des Baums bis zu einer festgelegten maximalen Tiefe gefüllt (Abb. 7, links). Im zweiten Fall werden die Knoten zufällig bis zur maximalen Tiefe mit Operatoren oder Terminals belegt (Abb. 7, rechts).

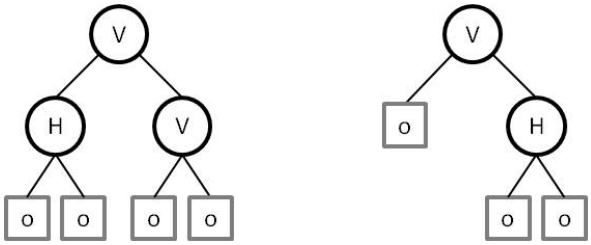


Abb. 7: Voll initialisierte Baumstruktur (links) und gewachsene Baumstruktur (rechts)

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

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

Darüber hinaus werden einige Baumeigenschaften im Voraus festgelegt bzw. begrenzt. Die maximal zulässige Baumtiefe wurde zunächst als Minimaltiefe + 1 definiert, um gleichmäßige Bäume zu generieren. Die Ausweitung der zulässigen Baumtiefe auf die tatsächliche Maximaltiefe ist möglich und würde die Anzahl an möglichen Lösungen zusätzlich erhöhen.

Die Auswahl eines der Operatoren H und V oder des Terminals o in den Knoten erfolgt zunächst mit gleicher Wahrscheinlichkeit. Grundsätzlich ist eine andere Verteilung der Wahrscheinlichkeiten, d.h. zufällige Auswahl der Operatoren, denkbar.

Die Datenstruktur wird entweder gleichzeitig mit der Generierung der Syntax oder durch deren Interpretation erstellt. Das Layout wird wiederum aus der grafischen Interpretation des Baums gewonnen. Abbildung 8 zeigt einen ersten, einfachen Prototyp, in dem eine Syntax zufällig generiert und anschließend grafisch interpretiert wird. Die Anzahl der zu generierenden Räume kann vom Nutzer eingegeben werden.

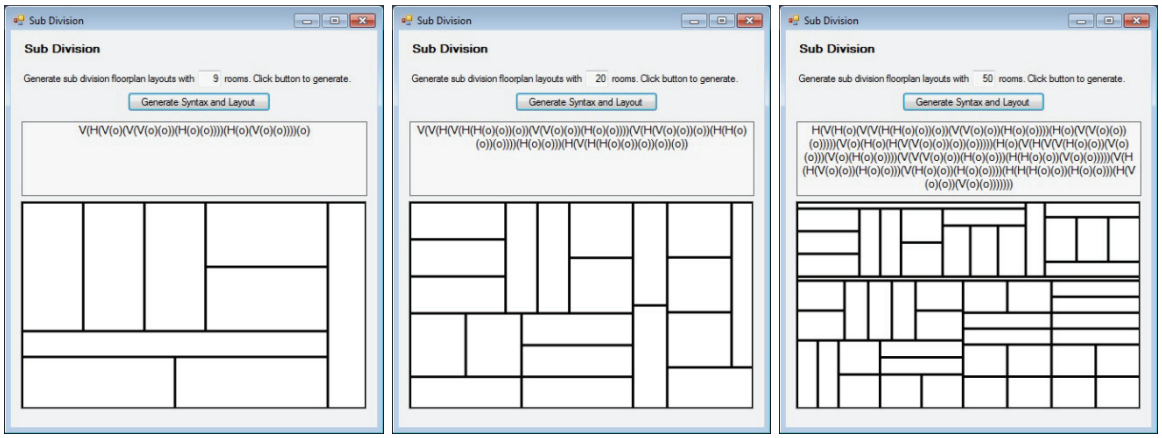


Abb. 8: Prototyp zur zufälligen Generierung einer Unterteilungssyntax mit beispielsweise 9, 20 oder 50 Räumen (von links nach rechts)

4. Optimierung von Grundrisslayouts

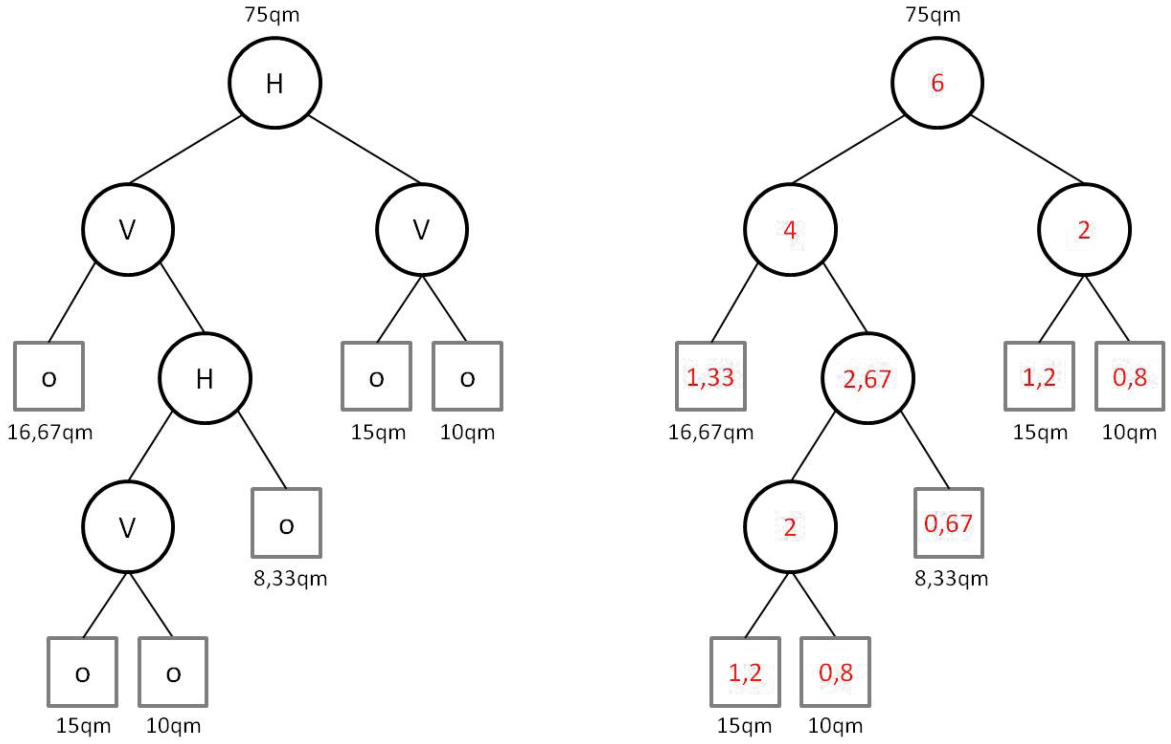
Die durch den im vorangegangenen Kapitel beschriebenen generativen Mechanismus erstellten Layouts werden zunächst zufällig generiert. Ein nächster Schritt besteht darin, die erzeugten Grundrisse hinsichtlich architektonisch relevanter Kriterien zu evaluieren und zu optimieren, um sie anschließend dem Nutzer zur ästhetischen Begutachtung, Auswahl und Weiterbearbeitung anzuzeigen. Zu den betrachteten architektonisch relevanten Problemstellungen gehört die Optimierung von Layouts im Hinblick auf bestimmte Raumgrößen sowie die Erzeugung bestimmter Nachbarschaftsbeziehungen zwischen Räumen.

4.1. Berechnung bestimmter Raumgrößen

Die Erstellung von Grundrissen, die Räume vorgegebener Größe enthalten, ist ein Grundproblem des Architekturentwurfs. Als operationales Problem wird es häufig mithilfe von Optimierungsalgorithmen gelöst. Im konkreten Fall des Unterteilungsalgorithmus lassen sich die Unterteilungswerte bzw. das Teilungsverhältnis, basierend auf den vorgegebenen Raumgrößen und einer bekannten Unterteilungsfolge, direkt berechnen. Eine Optimierung

der Lage der Unterteilungen ist folglich überflüssig, da nach Ausführung aller Unterteilungen auf Basis der Unterteilungswerte jeder entstandene Raum die vorgegebene, gewünschte Größe besitzt.

Die Berechnung der Teilungswerte, der sogenannten *split values*, erfolgt von den Blättern ausgehend. Zunächst wird das Gewicht eines Raums bzw. Blatts aus dem Verhältnis seiner gewünschten Größe zum Flächenmittel berechnet. Das Flächenmittel ergibt sich aus der Teilung der zur Verfügung stehenden Gesamtfläche durch die Anzahl der Räume. Bei einer Gesamtfläche von 75qm beträgt das Flächenmittel bei 6 Räumen beispielsweise 12.5qm. Ein Raum mit 15qm besitzt in diesem Fall einen Wichtungswert von 1.2, ein Raum mit 10qm einen Wert von 0.8. Der Wichtungswert eines Knotens berechnet sich aus der Summe der Wichtungswerte seiner beiden Äste, d.h. ein den beiden genannten Räumen zugeordneter Knoten hätte einen Wichtungswert von 2. So werden von den Blättern zur Wurzel die Wichtungswerte aller Räume und Knoten bestimmt (Abb. 9). Die Wichtung des Wurzelknotens aus der Summe der beiden Hauptäste muss gleich der Anzahl der Räume sein.



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

Abb. 9: Slicing Tree: 6 Räume, mit Angabe der Raumgrößen (links) und Berechnung der Wichtungswerte in den Blättern und Knoten (rechts)

Die Lage der Unterteilung sowie die den Unterästen zugeordnete Fläche wird anschließend, beginnend vom *root node*, für jeden Knoten aus dem Verhältnis der Wichtungswerte seiner zwei Unteräste bezogen auf die Knotenfläche berechnet (Abb. 10). Der im vorigen Beispiel

erwähnte Knoten besitzt beispielsweise ein Teilungsverhältnis von 1.2 zu 0.8, d.h. von 3 zu 2. Der zugehörige Teilungswert lässt sich aus der Unterteilung der Breite oder Höhe der zugehörigen Knotenfläche im berechneten Teilungsverhältnis bestimmen. Bei einer Unterteilung in horizontaler Richtung wird hierzu die Höhe der Knotenfläche herangezogen, man erhält den y-Wert der horizontalen Schnittlinie. Bei einer Unterteilung in vertikaler Richtung wird das Verhältnis der Unteräste auf die Breite der Knotenfläche bezogen, um den x-Wert der vertikalen Schnittlinie zu bestimmen.

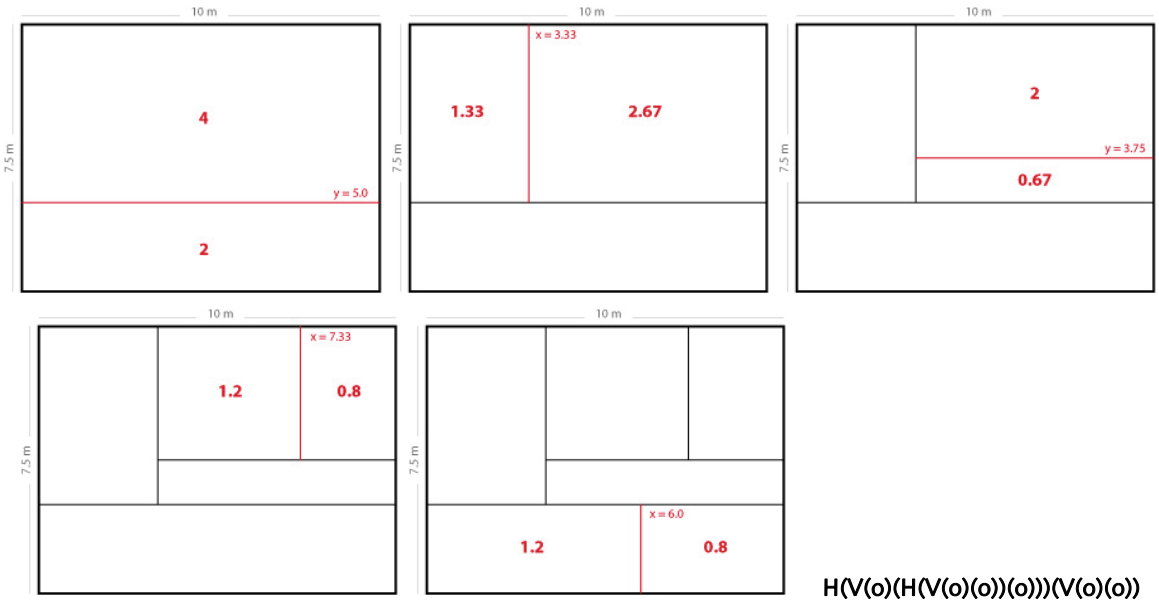


Abb. 10: Abfolge und Bestimmung der Lage der Unterteilungen im Layout

4.2. Suche nach bestimmten Nachbarschaftsverhältnissen

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

Optimale Lösungen werden mithilfe von Optimierungsstrategien gesucht. Im konkreten Fall wurde eine (μ + λ)-Evolutionsstrategie in Kombination mit genetischem Algorithmus und genetischer Programmierung implementiert. Der schematische Ablauf des evolutionären Prozesses ist in Tabelle 2 dargestellt. Genetischer Algorithmus und genetische Programmierung bilden dabei die Basis für die Rekombination und Mutation der Elternindividuen in Schritt 4 und 5 des Evolutionären Algorithmus.

Algorithmus:	Evolutionärer Algorithmus
Input:	μ – Größe der Elternpopulation λ – Größe der Kindpopulation Θr – Rekombinationsparameter Θm – Mutationsparameter Θs – Selektionsparameter Θi – Abbruchkriterium
Output:	**a*** das beste Individuum während des Durchlaufs oder **P*** die beste Population während des Durchlaufs
Logik:	1. Generation **t** = 0 2. Initialisiere **P(t)** mit μ Individuen 3. Evaluiere alle Individuen in **P(t)** mit der Evaluationsfunktion **F(t)** 4. Rekombiniere **P(t)** mit der Wahrscheinlichkeit Θr → **P'(t)** 5. Mutiere **P'(t)** mit der Wahrscheinlichkeit Θm → **P''(t)** 6. Evaluiere alle Individuen in **P''(t)** mit der Evaluationsfunktion **F(t)** 7. Selektiere μ Individuen aus **P''(t)** nach **F(t)** und dem Selektionsparameter Θs → **P(t+1)** 8. **t** = **t** + 1 9. Beginne wieder bei Schritt 4 solange das Abbruchkriterium Θi nicht erreicht ist. 10. return **a*** oder **P***.

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

Der genetische Algorithmus wird eingesetzt, um die Zuordnung der Funktionen und Indizes zu den Räumen hinsichtlich der gesuchten Nachbarschaftsverhältnisse zu optimieren. Hierbei werden zunächst die Indizes der Räume in der Abfolge ihres Entstehens bei der Unterteilung als Indexsequenz kodiert (Abb. 11). Im Rahmen der Optimierung wird diese Indexsequenz zur Erzeugung neuer Lösungen mutiert und rekombiniert. Die Mutation erfolgt durch das Vertauschen von zwei Indizes innerhalb der Sequenz und entspricht dem Vertauschen der Indizes zweier Räume (Abb. 12). Bei der Rekombination entstehen neue Lösungen aus der Kreuzung der Indexsequenzen zweier Lösungen durch One-Point-Crossover (Abb. 13).

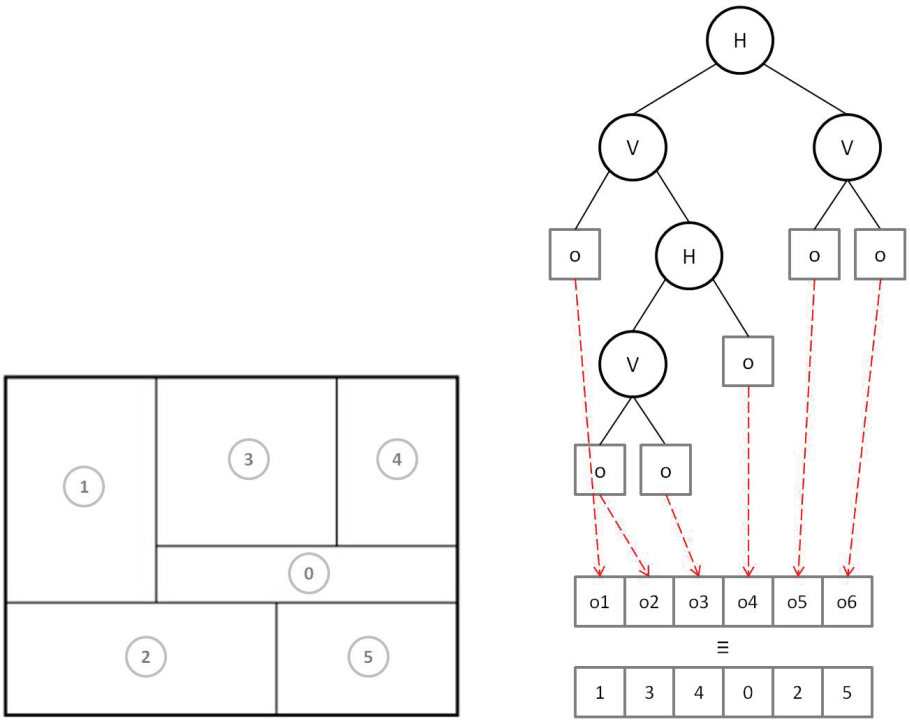


Abb. 11: Kodierung der Raumindizes aus der Entstehungsabfolge, Phänotyp (links), Genotyp (rechts)

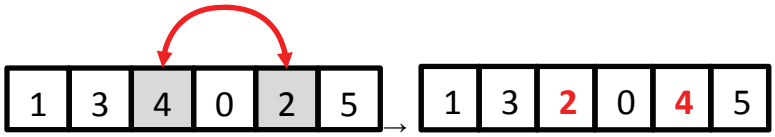


Abb. 12: Genetischer Algorithmus: Vertauschen von Indizes

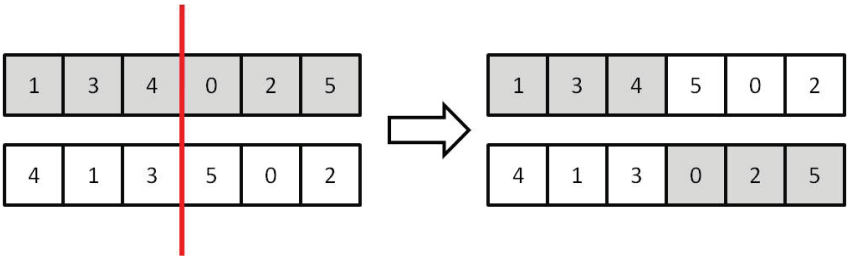


Abb. 13: Genetischer Algorithmus: One-Point-Crossover zweier Genome

Die genetische Programmierung dient zur Optimierung der Struktur, d.h. der Unterteilungsabfolge, des *Slicing Trees*. Neue Lösungen werden entweder durch zufällige Mutation eines Elternindividuums, d.h. dem Umschalten einer Unterteilung in einem Knoten von horizontal nach vertikal oder umgekehrt (Abb. 14), oder durch Crossover generiert. Beim Crossover werden Äste zwischen den Bäumen zweier Elternindividuen ausgetauscht. Die beiden Elternindividuen werden zunächst durch Binary Tournament Selection ausgewählt. Anschließend werden in den *Slicing Trees* Äste zum Austausch bestimmt, die jeweils die gleiche Anzahl an Blättern besitzen, damit die Gesamtzahl der Räume im Kindindividuum gleich der der Eltern bleibt (Abb. 15).

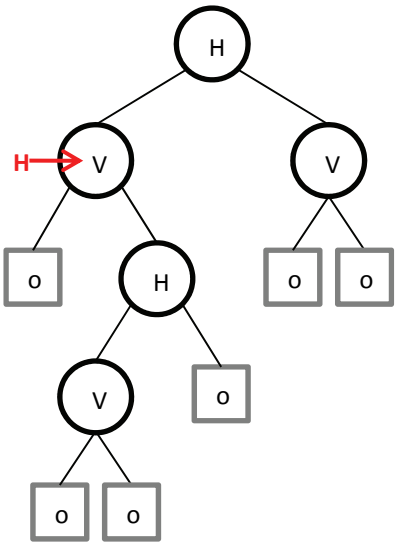
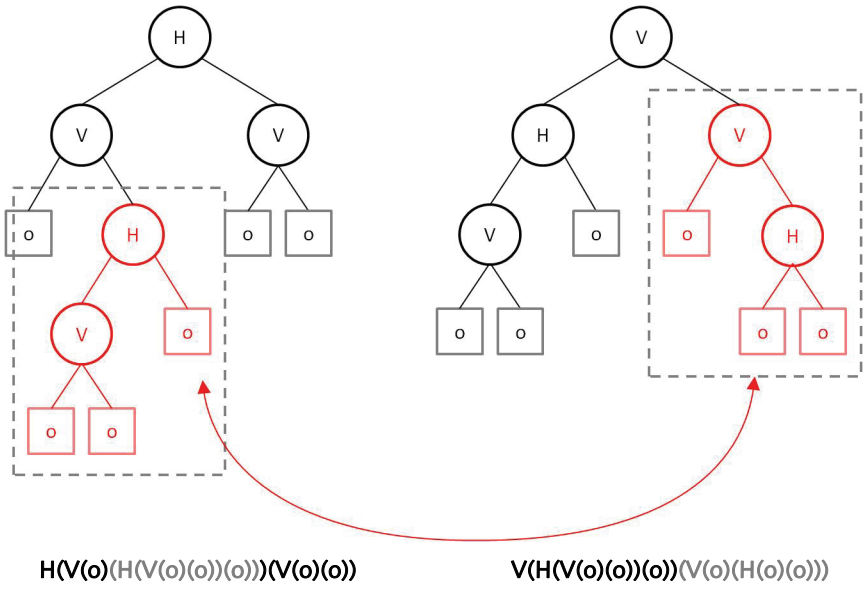


Abb. 14: Genetische Programmierung: Mutation eines Individuums durch Änderung der Unterteilungsrichtung



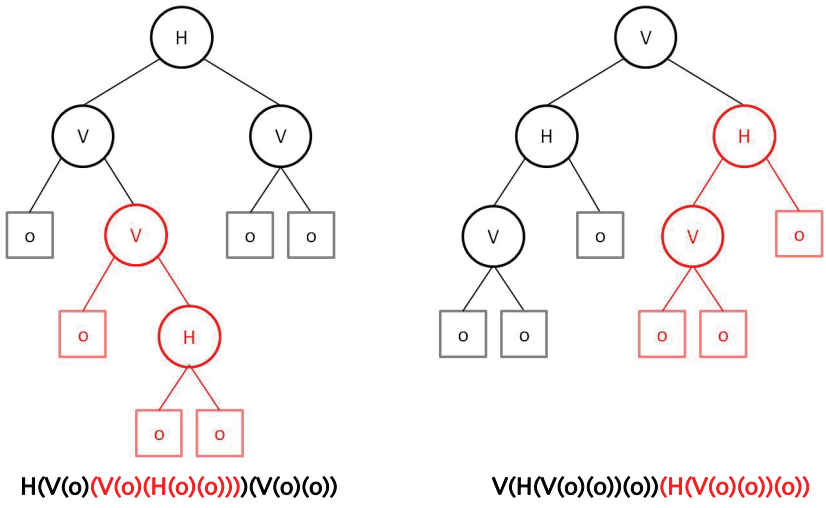


Abb. 15: Genetische Programmierung: Crossover von Ästen zwischen zwei Syntaxbäumen

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

$$f = \sum_{i=1}^{n} \overline{A_i B_i}$$

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

Der Abstand zwischen zwei direkt benachbarten Räumen beträgt 0. Folglich ist der aus der Evaluierungsfunktion resultierende Fitnesswert umso niedriger, je mehr gewünschte Nachbarschaftsbeziehungen in einer Layoutlösung existieren oder je näher die entsprechenden Räume zueinander liegen. Sein Idealwert tendiert gegen 0.

4.3. Prototypische Umsetzung

Der generative Mechanismus wurde in Kombination mit dem beschriebenen Evolutionären Algorithmus zur Suche nach bestimmten Nachbarschaftsverhältnissen prototypisch umgesetzt (Abb. 16). Als zu optimierende topologische Struktur wurde die im Grundrissentwurf relativ häufig vorkommende Sterntopologie gewählt. Eine Sterntopologie zeichnet sich dadurch aus, dass alle Räume des Layouts mit einem zentralen Raum verbunden und von diesem aus erreichbar sind.

Die Umsetzung des Algorithmus erfolgte auf Basis des in Tabelle 2 dargestellten Schemas. Die Individuen wurde mithilfe des in 4.2. dargestellten genetischem Algorithmus bzw. der genetischen Programmierung rekombiniert und mutiert. Die Auswahl von genetischem Algorithmus oder genetischer Programmierung zur Rekombination und Mutation erfolgte zufällig und mit gleicher Wahrscheinlichkeit. Bei der Berechnung des Fitnesswerts einer Lösung auf Basis des Abstands zwischen zwei Räumen wurde zusätzlich eine Mindestdurchgangsbreite berücksichtigt, um später das Platzieren von Verbindungen wie Türen zwischen den Räumen zu ermöglichen.

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

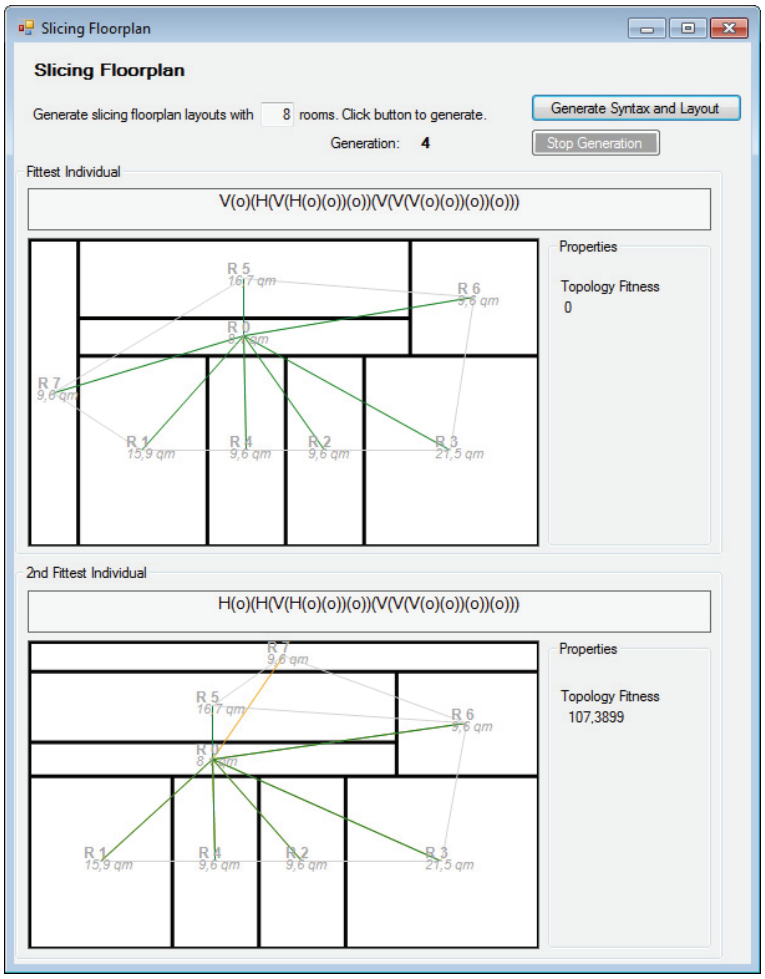


Abb. 16: Optimierung der Nachbarschaftsbeziehungen, Prototypische Umsetzung

5. Schlussbetrachtung und Ausblick

Im Rahmen der vorgestellten Untersuchung haben wir das generative und gestalterische Potential von Unterteilungsalgorithmen genutzt, um Layoutaufgaben für architektonische und städtebauliche Anwendungen zu lösen. Wir haben einen generativen Mechanismus entwickelt, der es uns erlaubt innerhalb einer vorgegebenen rechteckigen Grundstücksfläche *Slicing Floorplans* mit einer festgelegten Anzahl an Räumen mit bestimmten Raumgrößen zu erzeugen. Der Algorithmus imitiert eine häufig in der Architektur eingesetzte Entwurfsmethode, bei der eine vorgegebene Grundfläche zunächst in Zonen, Abschnitte und schließlich in Räume unterteilt wird. Als Grundlage für das Datenmodell dienten uns *Slicing Trees* und Unterteilungssyntax.

Die Kombination des generativen Mechanismus mit evolutionären Algorithmen (einer Kombination aus genetischem Algorithmus und genetischer Programmierung) ermöglichte die Optimierung des Grundrisses im Hinblick auf gewünschte Nachbarschaftsbeziehungen zwischen bestimmten Räumen innerhalb des Layouts.

Im Zuge einer Weiterbearbeitung der Thematik ist die Betrachtung von Mehrzieloptimierungsalgorithmen und eine damit verbundene die Optimierung der Grundrisse hinsichtlich weiterer Kriterien denkbar. Zu architektonisch relevanten Kriterien, die im Zuge einer Mehrzieloptimierung betrachtet werden könnten, zählen Raumcharakteristiken wie Raumproportionen und weitere topologischen Kriterien wie die Raumausrichtung. Darüber hinaus ist die Integration von Analysen zu Sonnenstand und Sichtbarkeiten zu untersuchen.

Desweiteren könnte der entwickelte Mechanismus im Rahmen einer Anwendung eingesetzt werden, die den Entwerfer beim Layoutentwurf unterstützt. Es sollte dem Anwender ermöglicht werden, den Lösungsraum für eine Entwurfsaufgabe flexibel zu durchsuchen und gegebenenfalls anzupassen. Hierzu müssten Interaktionsmöglichkeiten mit dem vorgestellten System untersucht und entwickelt werden.

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

Referenzen

Bäck, T. (2000). Introduction to Evolutionary Algorithms. In T. Bäck, D. B. Fogel, & Z. Michalewicz, *Evolutionary Computation 1 Basic Algorithms and Operators* (S. 59-63). New York, NY: Taylor & Francis Group, LLC.

Banerjee, A., Quiroz, J. C., & Louis, S. J. (2008). A model of creative design using collaborative interactive genetic algorithms. *Design Computing and Cognition DCC'08* (S. 397–416). Springer.

Bruls, M., Huizing, K., & Van Wijk, J. J. (2000). Squarified Treemaps. *Proceedings of Joint Eurographics and IEEE TCVG Symp. on Visualization* (S. 33-42). IEEE.

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

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

Flack, R. W. (January 2011). Evolution of Architectural Floor Plans. *Technical Report # CS-11-03* . St. Catharines, Ontario, Canada: Brock University, Department of Computer Science.

Hahn, E., Bose, P., & Whitehead, A. (2006). Persistent Realtime Building Interior Generation. *Sandbox Symposium 2006* (S. 179-186). Boston, MA, USA: ACM.

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

Johnson, B., & Shneiderman, B. (1991). Treemaps: a space-filling approach to the visualization of hierarchical information structures. *Proc. of the 2nd International IEEE Visualization Conference* (S. 284-291). San Diego: IEEE.

Kado, K. (1995). An Investigation of Genetic Algorithms for Facility Layout Problems. *Thesis* . Edinburgh: University of Edinburgh.

Lai, M., & Wong, D. F. (2001). Slicing Tree is a Complete Floorplan Representation. *DATE '01 Proceedings of the conference on Design, automation and test in Europe* (S. 228 - 232). Piscataway, NJ, USA: IEEE Press.

Lopes, R., Tutenel, T., Smelik, R. M., de Kraker, K. J., & Bidarra, R. (2010). A constrained growth method for procedural floor plan generation. *GAME-ON 2010.* Leicester, United Kingdom.

Marson, F., & Musse, S. R. (January 2010). Automatic Real-Time Generation of Floor Plans Based on Squarified Treemaps Algorithm. *International Journal of Computer Games Technology* .

Müller, P., Wonka, P., Haegler, S., Ulmer, A., & Van Gool, L. (2006). Procedural Modeling of Buildings. *Proceedings of ACM SIGGRAPH 2006 / ACM Transactions on Graphics (TOG)* (S.614-623). ACM Press, Vol. 25, No. 3.

Otten, R. H. (1982). Automatic Floorplan Design. *Proceedings of the 19th Design Automation Conference* (S. 261-267). Piscataway, NJ, USA: IEEE.

Sedgewick, R. (1991). *Algorithmen.* Bonn; München; Reading, Mass.: Addison Wesley Longman Verlag.

Valenzuela, C. L., & Wang, P. Y. (2002). VLSI Placement and Area Optimization Using a Genetic Algorithm to Breed Normalized Postfix Expressions. *IEEE Transactions on Evolutionary Computation, Vol. 6, Issue 4* , S. 390-401.

Wong, D. F., & Liu, C. L. (1986). A new algorithm for floorplan design. *Proceedings of the 23rd Design Automation Conference* (S. 101-107). IEEE.

Young, F. Y., & Wong, D. F. (1997). How good are Slicing Floorplans? *Proceedings of the International Symposium on Physical Design ISPD'97* (S. 144-149). Napa Valley, California, USA: ACM.