Wissensintegrierende Modellierung der Geometrie von Bauwerken mit Intervallen und Constraints für Parameter

vorgelegt von
M. Sc.
Jakob Vinzenz Kirchner

an der Fakultät VI – Planen Bauen Umwelt
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr.-Ing. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Timo Hartmann
Gutachter:	Prof. Dr.-Ing. Wolfgang Huhnt
Gutachter:	Prof. Dr. Jakob Beetz

Tag der wissenschaftlichen Aussprache: 14. Juli 2020

Berlin 2021

Zusammenfassung

Digitale Bauwerksmodelle stehen im Zentrum der modellbasierten Arbeitsweise in den Prozessen der Planung und Bewirtschaftung von Bauwerken. Sie bestehen dabei aus Daten, die semantische, geometrische und andere Aspekte des repräsentierten Bauwerks abbilden und algorithmisch nutzbar machen. Aktuelle Softwareprodukte ermöglichen zwar das Erstellen von Bauwerksmodellen, erfordern aber bei nachträglichen Änderungen des Modells aufwändige menschliche Eingriffe, da das Wissen über die Zusammenhänge und Randbedingungen des Entwurfs nicht im Modell gespeichert wird. In dieser Arbeit wird untersucht, wie sich Entwurfswissen für geometrische Daten strukturiert in ein Bauwerksmodell integrieren lässt. Dabei werden Randbedingungen und Zusammenhänge betrachtet, die sich bei dem architektonischen oder ingenieurtechnischen Entwurf ergeben. Diese lassen sich in Form von Intervallen für Parameter und sogenannten Constraints beschreiben und integrieren. Auf Grund deren großer Anzahl und damit einhergehender Komplexität, ist es notwendig, sie nachvollziehbar im Modell vorzuhalten. Es wird ein Bauwerksmodell vorgestellt, dessen Struktur Geometrie, Bauteile, Constraints und Parameter verbindet.

Die Auswertung des Wissens eines solchen Modells erfolgt auf Basis von Optimierungsverfahren aus dem Bereich der Constraint-basierten Modellierung mit Intervallen. Dabei werden die Möglichkeiten an Beispielen mit einer prototypischen Implementierung nachgewiesen und die Grenzen diskutiert.

Die Arbeit steht dabei im Zusammenhang mit den aus dem Forschungsbereich aufgestellten Forderungen nach Softwaresystemen, die den Menschen „intelligent“ bei der Lösungsfindung eines Entwurfs unterstützen können.

Abstract

Digital building models are the heart of the model-based working method in processes of planning and managing buildings. They consist of data, representing semantics, geometry and other aspects of the building. This data is appropriate for further processing. State of the art software products support the creation of building models, but require costly adjustments by users in the case of subsequent modifications. The reason is the lack of stored knowledge in the model, which is about the design's relations and boundary conditions.

This thesis investigates in the structured integration of design knowledge about geometric data in a building model. Especially the boundary conditions and relations originating from architectural or engineering ideas are considered. They can be described and integrated as intervals for parameters and constraints. Because of their huge number and the resulting complexity, it is necessary to structure them comprehensibly. A building model is presented, which includes geometry, construction elements, constraints and parameters.

Optimization methods from the domain of constraint-based modeling with intervals can be used for the evaluation of such model-included knowledge. Possibilities of these methods are demonstrated by examples, made in a prototypical implementation. Limitations are discussed subsequently.

This thesis is associated with demands made by researchers of this domain, for software systems, supporting users in finding a solution of the design process in an 'intelligent' way.

Besonderer Dank gilt meiner Familie, die mich die ganze Zeit unterstützte und mir den Rücken freihielt.

Inhaltsverzeichnis

Abkürzungen und Symbole

Abkürzungen

AABB	axis-aligned minimum bounding box
BIM	Building Information Modeling
B-rep	Boundary representation
CAD	Computer-Aided Design
CSG	Constructive Solid Geometry
CSP	Constraint-Satisfaction-Problem
GARM	General AEC Reference Model
GCS	Geometrisches Constraint-System
GCSP	Geometrisches Constraint-Satisfaction-Problem
ICS	Intervall-Constraint-System
ICSP	Intervall-Constraint-Satisfaction-Problem
IEEE	Institute of Electrical and Electronics Engineers
IFC	Industry Foundation Classes
NCS	Numerisches Constraint-System
NCSP	Numerisches Constraint-Satisfaction-Problem
OCL	Object Constraint Language
STEP	Standard for the exchange of product model data

KAPITEL 1

Einführung

1.1 Entwurfsentscheidungen in Bauwerksmodellen

Die voranschreitende Implementierung digitaler Prozesse in der Planung im Bauwesen geht einher mit der Entwicklung komplexer Datenstrukturen. Eine Form dieser Datenstrukturen stellt das digitale Bauwerksmodell dar, welches den Kern der modellbasierten Arbeitsweise (englisch: Building Information Modeling; Abk. BIM) darstellt. Dabei stellt das Modell die Datenbasis für den aktuellen und alle folgenden Prozessschritte dar und bildet, so weit wie nötig, das real existierende oder zu erschaffende Bauwerk ab. Ein solches Modell umfasst geometrische Daten zur Beschreibung der geometrischen Form, sowie nicht-geometrische Daten zur Beschreibung semantischer, physikalischer oder anderer Inhalte. Der Umfang und die sich daraus ergebende Komplexität solcher Modelle lässt sich nur beherrschen, indem sie sinnvoll strukturiert werden. Mit dem Repräsentationscharakter des digitalen Bauwerksmodells ergibt sich die Strukturierung anhand aller relevanten Einzelteile des abgebildeten Gebäudes, wie zum Beispiel Bauteile und durch sie begrenzte Räume.

Das eigentliche Ziel des Modells, als Quelle für die Generierung aller für die Umsetzung relevanter Informationen zu dienen, folgt dem Grundsatz, dass das digitale Bauen vor dem realen Bauen zu erfolgen hat. Die modellbasierte Arbeitsweise bringt es mit sich, dass das Modell datentechnisch konsistent sein soll, gleichzeitig aber auch die Anforderungen des realen Bauwerks, soweit es digital möglich ist, erfüllt werden müssen. Da das Modell entsprechend der üblichen iterativen Arbeitsweise beim Entwurf eines Bauwerks erstellt und verändert wird, stellt dies eine komplexe Aufgabe dar. Dies umfasst das Erstellen von Varianten, wie auch das Zurücknehmen von Entscheidungen. Sämtliche Arbeitsschritte

werden auf Grund von Entscheidungen menschlicher Experten, wie Ingenieure oder Architekten, getroffen.

Die Entscheidungen im Entwurfsprozess gehen auf Überlegungen der verantwortlichen Experten zurück und bilden die Synthese aus ihnen bekannten Annahmen und Anforderungen an das Bauwerk. Die Anforderungen an das Bauwerk sind vielfältig und haben ihren Ursprung in Vorschriften und Normen, baupraktischen Überlegungen, individuellen Projektanforderungen, ästhetischen Ansprüchen, aber auch in den Erfahrungen der Experten. Das digitale Modell hat dabei bestimmte Anforderungen zwingend – und andere wenn möglich – zu erfüllen. Der Experte hat die Aufgabe, aus diesem Spannungsfeld zwischen zwingenden und möglichen Anforderungen die Randbedingungen abzuleiten und auf deren Basis die Entwurfsentscheidungen zu treffen. Er hat diese so im Modell umzusetzen, dass sie – nach seiner Einschätzung – zum gewünschten Ergebnis führen. Dieses Prinzip existiert ebenso bei der Entwicklung digitaler Software (vgl. Berg u. a., 2009).

1.2 Motivation

Das digitale Bauwerksmodell selbst wird nach heutigem Stand der Technik, hauptsächlich als aktueller Planungsstand bzw. als das Produkt der durch einen Menschen getroffenen Entscheidungen, betrachtet. Die darüberhinausgehende Integration des Wissens und der Überlegungen, welche hinter den Entscheidungen steckt, spielen bisher nur eine begrenzte Rolle, obwohl sich Wissen und Überlegungen aus den Anforderungen als mehr oder weniger exakt formulierte Zwänge und *Regeln* ergeben. Eine typische Form solcher *Regeln* im Bauwesen ist die Festlegung von Mindest- oder Höchstmaßen unter gegebenen Bedingungen. Beispielsweise hat die lichte Breite der Fluchtwege bei bis zu 20 Personen im Einzugsgebiet entsprechend der technischen Regeln mindestens 1,00 m zu betragen (entnommen aus Bundesanstalt für Arbeitsschutz und Arbeitsmedizin: Ausschuss für Arbeitsstätten (ASTA), 2007, Tabelle 1). Wird nun die lichte Breite als *Parameter* zwischen begrenzenden Bauteilen des Fluchtwegs aufgefasst, so ist es in aktuell verfügbaren Softwareprodukten zwar möglich, einen konkreten Wert für jeden solchen *Parameter* der lichten Breite festzulegen, die Information über den möglichen Wertebereich des *Parameters* wird dadurch nicht abgebildet. Dazu kommt zwar die Umsetzung des Prinzips des *parametrischen Modellierens*, bei dem der Entwurf in Abhängigkeit von bestimmten *Parametern* beschrieben wird, doch führt dies auf Grund eines fest definierten *Konstruktionsplans* zu einem steifen Korsett. Dieses ist insofern unflexibel, als dass bestimmte *Parameter* nicht direkt veränderbar sind und nur indirekt über andere *Parameter* gesteu-

ert werden können. Damit ergibt sich ein gutes Ergebnis des *parametrischen Modellierens* durch Versuch und Irrtum. Es ist aber im Hinblick auf die Anforderungen nur in wenigen Fällen das beste bzw. optimale Ergebnis, welches möglich wäre. Ein solches Vorgehen steht der Idee des Baus eines unter aktuellen Randbedingungen möglichst optimalen Gebäudes entgegen. Bei der aktuellen Entwicklung, von einer steigenden Anzahl zu berücksichtigender Anforderungen durch umfangreichere und strengere Bauvorschriften, sowie der Berücksichtigung zusätzlicher Interessen, stellt es eine komplexer werdende Aufgabe dar. Dazu machen nachträgliche Änderungen des Entwurfs teure Umplanungen notwendig. Aus diesem Grund ist es zu prüfen, in wie weit die Bauwerksmodelle erweitert werden können, dass sie den Menschen bei solchen Aufgaben unterstützen.

Dies beinhaltet zum einen, dass Entwurfsentscheidungen und die ihnen zugrunde liegenden begründbaren Überlegungen nachvollziehbar im Modell zu finden sind, und zum anderen, dass die Randbedingungen und Zusammenhänge direkt in das Modell integriert werden. Beides umfasst die Integration von Entwurfswissen in das Modell auf eine formalisierte Art und Weise und würde es ermöglichen, den Spielraum der möglichen Änderungen eines Modells algorithmisch zu berechnen.

Es existieren sinnvolle Argumente für die Integration der Überlegungen und Entwurfsentscheidungen. Ihre digitale Repräsentation würde zu einer verbesserten Kommunikation und einem verbesserten Arbeiten am digitalen Modell führen (vgl. Otey u. a., 2018), da eine erhöhte Nachvollziehbarkeit der Absichten hinter den Entscheidungen möglich wäre (vgl. Shum u. a., 1994). Die Randbedingungen und Zusammenhänge könnten dazu genutzt werden, Änderungen direkt im Modell umzusetzen, solange alle Zwänge erfüllt wären. Auch wiederholt auftretende Muster in Form von wiederauftretenden Anforderungen ließen sich über Projekte hinweg nutzen. Insgesamt würde sich die Anzahl der Iterationsschritte im Prozess reduzieren, da direkt bei Ausführung von Änderungen und nicht am Ende eines Iterationsschritts die Einhaltung aller Randbedingungen geprüft würden. Die Möglichkeit, das Entwurfswissen in das Modell zu integrieren, zu verarbeiten und auszulesen, würde schlussendlich die Produktivität deutlich erhöhen (vgl. Peña-Mora u. a., 1993).

Die Anforderungen an solche Systeme und die Gründe für ihren sinnvollen Einsatz wurden bereits beschrieben. G. Lee u. a. (2006) bezeichnen die Integration von Entwurfs- und Ingenieurwissen als Voraussetzung für leistungsfähige Softwaresysteme. Ein solches System nennt Galle (1995) „intelligent“, da es damit möglich ist, bei dem sich weiterentwickelnden Entwurf die semantische Integrität des Modells konsistent zu erhalten. Darüber hinaus nennt er die Möglichkeit, dass ein solches „intelligentes“ System Lösungen zu verschiedenen Aspekten des Entwurfsproblems generieren und vorschlagen

kann. Eastman, Sacks u. a. (2004) bezweifeln, dass ein manuelles Erstellen und Platzieren der in die Millionen gehenden Einzelteile eines mittelgroßen Gebäudemodells sinnvoll durchgeführt werden kann. Die benötigte Zeit und die Anzahl der gemachten Fehler wäre zu groß. Sie fordern daher die Entwicklung geeigneter Softwareprodukte, die den Entwurfsprozess unterstützen. Valdes u. a. (2016) schlagen dazu einen neuartigen Ansatz des Bauwerksentwurfs vor. Kern des Ganzen ist eine Systemumgebung, in der Gebäudeanforderungen und Entwurfsspezifikationen definiert und anschließend ausgewertet werden können. Dies soll mit der geometrischen Konstruktion gekoppelt werden. Der sich ergebende Vorteil wäre die Reduzierung von Fehlern und Erhöhung der Konsistenz der Modelle.

Auch Bettig, Bapat u. a. (2005) erkennen, dass aktuelle Softwaresysteme kaum Entwurfswissen integrieren und beschreiben die Vision eines interaktiven Entwurfssystems. In diesem könnten die Anforderungen an das Modell während des Entwurfsprozesses durch den Menschen spezifiziert und durch eine Wissensdatenbank ergänzt werden. Dabei könnte – bei Änderungen – ein Algorithmus auftretende Probleme unter Berücksichtigung der implementierten Anforderungen lösen. Nach Bettig, Bapat u. a. (eigene Übersetzung 2005, Seite 8) würde dies zu einem neuen Paradigma für die menschlichen Experten, hier genannt „Designer“, beim Entwurf der Modelle führen:

> „Designer könnten die Anzahl und Varianten von Entwürfen durch das Hinzufügen, Entfernen und Verändern der Entwurfsanforderungen kontrollieren. Designer könnten die Entwurfsalternativen untersuchen und durch aktives Anpassen die Grenzen erkennen.“ (eigene Übersetzung Bettig, Bapat u. a., 2005, Seite 8)

Die aufgeführten Veröffentlichungen enthalten alle die Forderung nach wissensintegrierenden Systemen für Bauwerksmodelle und machen Vorschläge, wie diese umgesetzt werden könnten. Es existiert aber aktuell keine Software, welche die Forderungen weder für einen Teilbereich noch für ein Gesamtsystem ermöglicht.

1.3 Ziel und Abgrenzung

Das Ziel dieser Arbeit ergibt sich aus den im vorangegangenen Abschnitt aufgezählten Forderungen an Softwaresysteme bzw. Bauwerksmodelle und die zugehörigen Visionen:

> Ziel ist die Entwicklung eines Bauwerksmodells mit Fokus auf die Geometrie, welches neben dem konkreten Entwurf das den einzelnen Entwurfsentschei-

> dungen zugrunde liegende Expertenwissen integriert. Das Expertenwissen soll bei der Entwicklung eines möglichst optimalen Entwurfs genutzt werden.

Auf Grund der Komplexität der Aufgabe soll nur ein Teil der Forderungen erfüllt werden. Deshalb richtet sich der Fokus der Arbeit ausdrücklich auf das Expertenwissen in Bezug auf die Geometrie und auf die Strukturierung eines Bauwerksmodells. Dafür werden zwei Formen des Expertenwissens, die Intervalle in *Parametern* und die *Constraints* in das Modell integriert.

Dies ist zum einen das Prinzip des *Mengen-basierten Entwurfs* (englisch: set-based design), welcher in den Wertebereichen der *Parametern* Ausdruck findet. Bettig und Kale (2012) fordern von der nächsten Generation von Software-Systemen für das Modellieren von Geometrie, dass sie speziell Ungleichungen unterstützen, die es ermöglichen, Wertebereiche von *Parametern* festzulegen. Aus diesem Grund wird in dieser Arbeit geprüft, in wie weit dies möglich ist:

> Zur Beschreibung der gültigen Wertebereiche von numerischen Randbedingungen werden Intervalle genutzt.

Zum anderen soll das *Constraint-basierte Modellieren* (englisch: constraint-based modeling) verwendet werden, das es ermöglicht, deklarativ und flexibel zu modellieren:

> Zur Beschreibung der Zusammenhänge zwischen den Einzelteilen des Modells werden sogenannte *Constraints* (deutsch: Zwangsbedingungen) genutzt.

Beide Konzepte erhöhen die Komplexität eines Bauwerksmodells beträchtlich. Der Mehrwert des Konzepts ergibt sich, wenn der Mensch beim Lösen der Entwurfsprobleme unterstützt werden kann. Erst damit kann der Rechner zu mehr dienen, als nur zur mathematischen Berechnung und zur Datenspeicherung (vgl. Vikram u. a., 2007). Dabei stellt die eigentliche Schwierigkeit *Constraint-basierter Modelle* deren Komplexität und Umfang dar. Shah (2001) fordert deshalb, dass der Mensch wissen muss, welche Änderungen im System unter den aktuell gesetzten *Constraints* möglich sind. Es stellen sich folgende Fragen:

> Wie kann effizient und gezielt eine sinnvolle Lösung gefunden und ausgewählt werden, wenn im Modell – unter den gegebenen Anforderungen – eine große Anzahl Lösungen existiert? Wie können die verbliebenen Möglichkeiten in Form der möglichen Werte der *Parameter* des Modells bestimmt werden?

Diese Arbeit gehört ausdrücklich nicht zu den Bereichen des maschinellen Lernens (englisch: machine learning) bzw. der Parameterstudien. Sie betrachtet keine Algorithmen,

die nach heuristischen Prinzipien ablaufen. Ebenso wenig werden über mehrere Durchläufe Eingangsdaten variiert und das Ergebnis statistisch ausgewertet, wie es beispielsweise bei der Optimierung in der *generativen Gestaltung* (engl. generative design) der Fall ist. Dies bedeutet, dass alle Algorithmen in dem durch das Expertenwissen im Bauwerksmodells spezifizierten Rahmen keinen möglichen gültigen Entwurf ausschließen dürfen.

1.4 Einbezogene Forschungsgebiete

Zur Erreichung der Ziele dieser Arbeit werden verschiedene Forschungsgebiete betrachtet und die auf diesen Gebieten entwickelten Ansätze und gewonnenen Erkenntnisse zusammengeführt. Die einbezogenen Forschungsgebiete sind:

CAD-Modellierung: Der *rechnergestützte Entwurf* beinhaltet die Beschreibung und den Aufbau von Objekten auf geometrischer Grundlage für architektonische oder ingenieurtechnische Anwendungen.

Constraint-basiertes Modellieren von Geometrie: Umfasst das Paradigma, Geometrie auf Basis von *Constraints* zu beschreiben. Da die Durchführung der Berechnung eines Entwurfs mit *Constraints* nur in wenigen Fällen trivial ist, existiert dazu eine Vielzahl an Untersuchungen.

Mengen-basierter Entwurf: Ist ein Ansatz mit Ursprüngen im Automobilbau und stellt die Idee dar, im Entwicklungsprozess die *Entwurfsparameter* so spät wie möglich einzuschränken, damit keine sinnvolle Lösung zu früh ausgeschlossen werden kann.

Intervall-Methoden: Auf Grundlage der mathematischen Beschreibung von Zahlen als Menge eines Zahlenraums können *Constraint-Systeme* beschrieben und berechnet werden. Die dafür entwickelten Algorithmen unterteilen sich in Optimierungs- und Suchverfahren.

1.5 Eigene zugehörige Veröffentlichungen

Im Zuge der Forschungen für diese Arbeit wurden zwei Untersuchungen veröffentlicht. Diese Veröffentlichungen enthalten bereits Grundideen, gehen aber von stark vereinfachten Randbedingungen aus. Die darin gewonnenen Erkenntnisse finden sich auch in dieser Arbeit wieder.

In der ersten Untersuchung (Kirchner, Huhnt u. a., 2016), wird die Forderung nach korrekten *Geometrie-Objekten* aufgestellt, die bei nachträglichen Änderungen ein nachvollziehbares Verhalten zeigen. Dies wird für den stark vereinfachten Fall untersucht, wenn die Beziehungen zwischen Objekten ausschließlich in einer Achsrichtung definiert werden können und die *Geometrie-Objekte* in einer einfachen raumzerlegenden Datenstruktur definiert sind. Dass dies unter den gegebenen Vereinfachungen möglich ist, wird in einer prototypischen Implementierung gezeigt.

Die Vorzüge von Berechnungen mit Intervallen bei der geometrischen Modellierung wird bereits in der zweiten Untersuchung (Kirchner und Huhnt, 2018) erkannt. Dabei ist es möglich, beliebige nichtlineare arithmetische Ausdrücke auszuwerten. Die Vereinfachung liegt darin, dass keine Zyklen bei der Berechnung auftreten dürfen, was zwingend eine Baumstruktur des *Constraint-Graphen* voraussetzt.

1.6 Eindeutigkeit der Begriffe

Das Bild der geprägten Begriffe, welches sich bei der Bearbeitung des Themas ergibt, ist schwierig, da diese teilweise nicht eindeutig definiert und unterschiedlich verwendet werden. Es finden sich beispielsweise einzelne Begriffe, welche in unterschiedlichen Quellen und Domänen abweichend definiert sind. Dies geht soweit, dass solche Begriffe vollkommen gegensätzlich definiert und gebraucht werden. Andere Begriffe werden pauschal verwendet und nicht detaillierter definiert. Auch werden teilweise für denselben Sachverhalt eine Vielzahl von Begriffen geprägt. Trotz des Vorhandenseins von zusammenfassenden Veröffentlichungen und Büchern führender Experten ergibt sich daraus ein unbefriedigendes Gesamtbild, welches den tieferen Blick in die Thematik erschwert. Diese Kritik soll als Anregung verstanden werden, zukünftig Anstrengungen zur Verbesserung der Situation zu unternehmen.

1.7 Verwendete Fachterminologie und Zahlendarstellung

Die in dieser Arbeit verwendete Fachterminologie wird, so weit möglich, in deutscher Sprache geführt. Dazu gehört beispielsweise die konsequente Nutzung des deutschen Begriffs *Entwurf* für den englischen Begriff *design* in allen Kombinationen.

Zur Identifikation der Begriffe in dem fast vollständig englischsprachigen Forschungsbereich werden bei deren Erstnennung zusätzlich die englische Entsprechung und gegebe-

nenfalls deren Abkürzung genannt. Es existieren wenige, aber gewichtige Ausnahmen, wie zum Beispiel der Begriff *Constraint*, bei dem umgekehrt verfahren wird. Dies geschieht, da nach Meinung des Autors entweder der englische Begriff so dominant und präzise für die behandelte Domäne ist, oder eine deutsche Entsprechung nicht existiert, und eine Eigenübersetzung zu starken Eigenheiten mit sich bringen würde. Dies wird ebenfalls bei Abkürzungen angewendet, die fast ausschließlich auf der englischen Terminologie basieren.

Bei der Zahlendarstellung von Dezimalzahlen, wird auf die in der Informatik übliche Schreibweise, mit einem . (Punkt) als Dezimaltrenner zurückgegriffen. Dies vereinfacht die eindeutige Darstellung bei den in dieser Arbeit verwendeten Intervallen, in denen das Trennzeichen zwischen der oberen und unteren Grenze ein , (Komma) darstellt.

1.8 Aufbau der Arbeit

Nach diesem Einführungskapitel, in welchem die Motivation und die Ziele der Arbeit herausgestellt wurden, gliedert sich die Arbeit folgendermaßen.

In Kapitel 2 werden Definitionen zur Beschreibung von Modellen und der Unterscheidung von *Regeln* und *Constraints* gegeben. Es werden anschließend die Grundlagen des *Constraint-basierten Modellierens* und der sich daraus ergebenden Probleme vorgestellt. Zusätzlich werden die im weiteren Verlauf der Arbeit vorausgesetzten Grundlagen über den Aufbau und die Eigenschaften der geometrischen Modelle des *Solid Modelings* näher erläutert.

Das Ziel der *Wissensintegration* speziell für die Geometrie von Bauwerksmodellen erfordert, die einzelnen Bestandteile solcher Modelle näher zu erläutern. Dafür wird in Kapitel 3 das Prinzip erläutert, welche Formen von *Wissen* in digitale Modelle integriert werden können und wie sich daraus ein entsprechendes *Expertensystem* ergibt. Anschließend wird das *digitale Bauwerksmodell*, dessen Aufbau und seine Bestandteile bzw. verschiedenen Aspekte, analysiert und veranschaulicht. Es wird dazu auch der Bezug zum *Mengen-basierten Entwurf* hergestellt, welcher das übergeordnete Prinzip darstellt, wie mit Modellen mit integriertem Wissen zu arbeiten ist.

Die Frage nach den Formen des Modellierens von Geometrie, und wie dabei Wissen integriert werden kann, wird in Kapitel 4 betrachtet. Dazu wird der Stand der Technik und der Forschung im Bereich der Modelle des *Computer-Aided Designs* (CAD) wiedergegeben. Insbesondere das Prinzip des *parametrischen Modellierens* und der Probleme von Lösungsverfahren beim *Constraint-basierten Modellieren* von Geometrie werden näher erläutert. Dabei werden speziell die Ermittlung gültiger Parameterbereiche und die Suche

nach Entwürfen betrachtet, die der Idee bzw. der Absicht des Menschen nahekommen. Es wird anschließend auf die Forschungen in Bezug auf das Bauwesen im Bereich der Modellierung von Geometrie eingegangen, insbesondere auf das *Constraint-basierte Modellieren*. Die Ergebnisse der Betrachtung des aktuellen Entwicklungsstands zu diesem Thema werden diskutiert und daraus die weiteren Untersuchungen für diese Arbeit abgeleitet. Dabei wird die numerische Berechnung von *Constraint-Systemen* auf Basis von Intervallen für die Geometrie für die weitere Untersuchung ausgewählt.

Die Berechnung auf Grundlage von Intervallen und deren Anwendung auf das Modellieren und Lösen von *Constraint-Systemen* wird in Kapitel 5 erläutert. Dabei werden die Möglichkeiten und Grenzen verfügbarer Algorithmen zur Ermittlung gültiger Parameterbereiche, der Optimierung und Lösungsfindung, genauer beschrieben.

In Kapitel 6 wird ein Modell vorgestellt, welches im Zuge dieser Arbeit entwickelt wurde. Dieses beinhaltet die Geometrie von Bauteilen und ähnlichen Entitäten. Dabei wird das *Wissen* auf Basis von *Constraints* und Intervallen strukturiert in das Modell integriert.

Wie das in das Modell integrierte *Wissen* ausgewertet werden kann, wird anhand einer prototypischen Implementierung vorgestellt. Diese wird in Kapitel 7 beschrieben. Dabei werden bekannte Algorithmen aus dem Bereich der Optimierungsverfahren auf Basis von Intervallen genutzt, um mögliche Parameterbereiche und möglichst geometrisch naheliegende Lösungen zu berechnen.

Die Möglichkeiten der prototypischen Implementierung werden in Kapitel 8 an Beispielen gezeigt und bewertet. Die Arbeit wird mit einer Zusammenfassung, einem Fazit und einem Ausblick in Kapitel 9 abgeschlossen.

KAPITEL 2

Grundlagen und Definitionen

In diesem Kapitel werden Begriffe definiert, die im Verlauf dieser Arbeit wiederholt Verwendung finden. Außerdem werden Grundlagen zum *Constraint-basierten Modellieren* und der Modellierung von geometrischen Körpern erläutert, die in den folgenden Kapiteln vorausgesetzt werden.

2.1 Grundsätzliche Modell-Klassifikation

Modelle zur Beschreibung von Sachverhalten besitzen Merkmale, die sich von anderen Modellen dieser Art unterscheiden. Da es in verschiedenen Domänen und auf Grund des Kontexts zu einer Verwendung von Synonymen zur Beschreibung von Modellen kommen kann, obwohl diesen unterschiedliche Konzepte zugrunde liegen, ist es notwendig, grundlegende Typen von Modellen festzulegen. Auf diese Weise ist eine eindeutige Beschreibung möglich, und an entsprechender Stelle kann auf Analogien Bezug genommen werden.

Für die Repräsentation eines Modells zur Beschreibung eines Sachverhalts lassen sich zwei Formen unterscheiden.

Definition 2.1.1 (explizites Modell)**.** Ein *explizites Modell* (englisch: explicit model) repräsentiert eindeutig die Lösung eines Problems in Form von Daten, die ohne weitere Berechnungen abgerufen werden können. Dabei muss die Struktur des Problems nicht mitabgebildet werden.

Definition 2.1.2 (implizites Modell)**.** Ein *implizites Modell* (englisch: implicit model) repräsentiert ein Problem durch Beschreibung der Struktur des Problems. Mögliche

Lösungen sind dabei algorithmisch zu ermitteln. Eine Lösung eines impliziten Modells ist ein *ausgewertetes Modell* (englisch: evaluated model). Dieses gehört wiederum zu den *expliziten Modellen*.

Für *implizite Modelle* existieren zwei mögliche Paradigmen: *imperative Schemata* und *deklarative Schemata*.

Definition 2.1.3 (imperatives Schema)**.** Ein *imperatives Schema* (englisch: imperative scheme) stellt das Problem und die Lösung in einer strukturieren Form dar, sodass die Reihenfolge der einzelnen Lösungsschritte klar definiert ist. Existiert mindestens eine Lösung, kann eine solche durch serielle Bearbeitung der einzelnen Schritte gefunden werden.

Definition 2.1.4 (deklaratives Schema)**.** Ein *deklaratives Schema* (englisch: declarative scheme) stellt das Problem in einzelnen, beschreibenden Aussagen dar, deren Reihenfolge nicht klar definiert sein muss. Existiert mindestens eine Lösung, kann diese gegebenenfalls nur durch komplexe Lösungsverfahren gefunden werden.

2.2 Regeln und Constraints

In dieser Arbeit wird streng zwischen den Begriffen *Regel* und *Constraint* unterschieden. Dies ist notwendig, da sich nur durch diese Unterscheidung verschiedene Konzepte erklären lassen.

Definition 2.2.1 (Regeln)**.** *Regeln* bestehen aus einer Bedingung für Eingangsgrößen (englisch: input) und einem eindeutigen Ergebnis (englisch: output) in Form von Ausgangsgrößen. Das Ergebnis kann unabhängig von anderen *Regeln* jederzeit bestimmt werden, wenn die Eingangsgrößen vollständig bekannt sind.

Das Ergebnis kann dabei in einfachster Form eine Aussage der Gestalt *Wahr* oder *Falsch* sein, wie zum Beispiel „Ist der Flur exakt 3 m breit?“. Komplexere *Regeln* können *Konstruktionsregeln* sein, wie zum Beispiel „Sind die Punkte P1 und P2 vorhanden, erzeuge eine Linie zwischen ihnen“. *Regeln* sind dabei vergleichbar mit der Kontrollstruktur aus Programmiersprachen: der *bedingten Anweisung* (englisch: if-then-else-expression).

Definition 2.2.2 (Constraints)**.** *Constraints* (deutsch: Zwangsbedingungen) stellen im Gegensatz zu *Regeln* eine Beschreibung dar, die etwas einschränkt. Dabei ist etwas *constraint*, wenn es durch mindestens eine *Constraint* eingeschränkt wird. Ihre Aussagen entsprechen dabei der Prädikatenlogik der ersten Stufe (vgl. Ebbinghaus u. a., 2018, Seite 15).

Bei *Constraints* ist es immer möglich, die Negation der Aussage als Umkehrschluss abzuleiten. Dadurch muss bei der Auswertung von *Constraints* nicht zwischen Eingangsgrößen und Ausgangsgrößen unterschieden werden. Wenn die Eingangsgrößen vollständig bekannt sind, ist es jederzeit möglich zu prüfen, ob die Aussage erfüllt und die *Constraint* damit *valide* ist. Ist die Aussage nicht erfüllt, gilt die *Constraint* als *verletzt*.

Zur Spezifizierung von *Constraints* auf Basis einer formalen Sprache kann die *Object Constraint Language* (OCL) (Object Management Group, 2014) genutzt werden.

Eine grundlegende Art der *Constraints* sind *algebraische Constraints*.

Definition 2.2.3 (Algebraische Constraints)**.** *Algebraische Constraints* beschränken *Variablen*, indem sie einen mathematischen Ausdruck, üblicherweise in Form einer Gleichung oder Ungleichung, abbilden. Dies ist beispielsweise für die *Variablen* a, b und c die Gleichung $a + b = c$.

Fasst man *Regeln* und *Constraints* als Relationen zwischen Objekten auf, so sind *Regeln* bei der Auswertung unidirektional, während *Constraints* bidirektional interpretiert werden können. Dadurch bietet die Verwendung von *Constraints* deutlich mehr Möglichkeiten und Flexibilität als *Regeln* bei der Modellierung von Systemen. Je nach Struktur des Problems fällt die Auswertung von *Constraints* deutlich komplexer aus. Dies wird im folgenden Abschnitt genauer erläutert.

2.3 Constraint-Satisfaction-Problem

Das *Constraint-basierte Modellieren* ist die Beschreibung eines Problems durch Aufbau eines *Constraint-Systems*.

Definition 2.3.1 (Constraint-System)**.** Ein *Constraint-System* besteht aus folgenden Teilen:

- Eine Menge von $T = \{T_1, ..., T_n\}$, wobei jedes T_i einen Datentyp darstellt.
- Eine Menge $V = \{v_1, ..., v_n\}$ von *Variablen*, wobei v_i vom Typ T_i ist
- Ein Zustand $S = \{D_1 \times ... \times D_n\}$, gilt $D_i \subseteq T_i$ bei $i \in 1, ..., n,$. Dabei ist D_i der mögliche Wertebereich (englisch: feasible domain) der *Variablen* v_i.
- Eine Menge $C = \{C_1, ..., C_m\}$ von *Constraints*. C_i mit $i = 1, ..., m$ besteht dabei aus einer Menge von *Variablen* $C_i = \{v_1, ..., v_k\}$ wobei $C_i \subseteq V$. C_i ist eine atomare Formel der Prädikatenlogik 1. Stufe. Dabei repräsentiert sie eine Teilmenge des

kartesischen Produkts $D_1 \times ... \times D_k$, welche die Werte darstellt, die untereinander kompatibel sind.

- Ein Anfangszustand $S_{initial}$, wobei für jeden weiteren Zustand gilt $S \subseteq S_{initial}$.

Constraints besitzen dabei bezüglich des Zustands eines *Constraint-Systems* unterschiedliche Relevanz. Eine *Constraint* ist dabei

- *aktiv*, wenn ihr Entfernen zu einer „Entspannung“ des Systems führen würde. Das bedeutet, dass sie wirklich für einen Teil der Einschränkungen, die den Zustand beschreibt, allein verantwortlich ist.
- *inaktiv* (englisch auch: „drowned“), wenn für den aktuellen Zustand des Systems mindestens eine andere *Constraint* existiert, die mindestens dieselbe Einschränkung verursacht.

Die reine Beschreibung des Problems in Form eines *Constraint-Systems* ist deklarativ, sodass sich die Lösungen für dieses Problem nicht direkt bestimmen lassen. Da, die Bestimmung der Lösung ausschließlich in bestimmten Fällen trivial ist, wird das Problem als *Constraint-Satisfaction-Problem* bezeichnet.

Definition 2.3.2 (Constraint-Satisfaction-Problem)**.** Das *Constraint-Satisfaction-Problem* (deutsch: Bedingungserfüllungsproblem; Abkz. CSP) beschreibt die Schwierigkeit, einen bzw. die Zustände eines *Constraint-Systems* zu finden, für die alle *Constraints* gleichzeitig erfüllt bzw. *valide* sind.

Es existieren verschiedene Arten des CSPs. In dieser Arbeit werden *Numerische Constraint-Satisfaction-Probleme* (NCSP), *geometrische Constraint-Satisfaction-Probleme* (GCSP) (siehe Abschnitt 4.3.4) und *Intervall-Constraint-Satisfaction-Probleme* (ICSP) (siehe Abschnitt 5.4.2) eingeführt. Der Zusammenhang der verschiedenen CSP-Arten ist in Abbildung 2.1 abgebildet.

Definition 2.3.3 (Numerisches Constraint-Satisfaction-Problem)**.** Ein *numerisches Constraint-Satisfaction-Problem* (NCSP) stellt eine spezielle Form des CSPs dar. Dabei sind die Elemente der Menge $T = T_1, ..., T_n$ Zahlenwerte. Das *numerische Constraint-System* (NCS) selbst besteht aus den *Variablen* und Gleichungen und gegebenenfalls Ungleichungen in einem Gleichungssystem.

Ein CSP kann auf Basis der Anzahl der möglichen Lösungen l unterschieden werden:

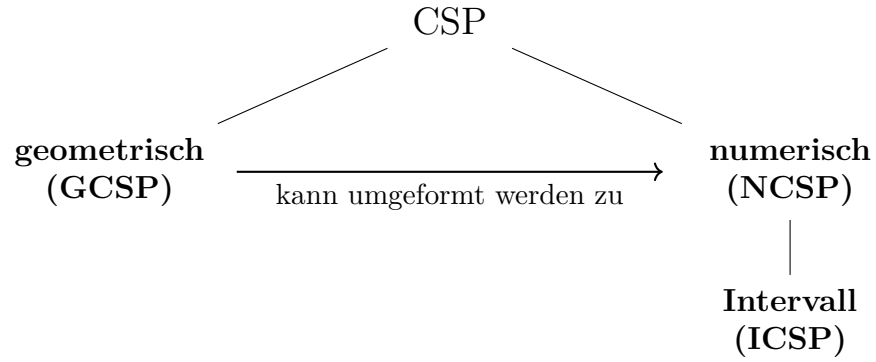


Abbildung 2.1: Taxonomie der *Constraint-Satisfaction-Probleme* in dieser Arbeit

Überbestimmung Ein *Constraint-System* ist *überbestimmt* (englisch: over-constraint), wenn keine Lösung möglich ist: $l = 0$. Das bedeutet, es existiert mindestens ein Widerspruch durch Verletzung einer oder mehrerer *Constraints*.

Wohlbestimmung Ein *Constraint-System* ist *wohlbestimmt* (englisch: well-constraint), wenn exakt eine Lösung möglich ist: $l = 1$.

Unterbestimmung Ein *Constraint-System* ist *unterbestimmt* (englisch: under-constraint), wenn mehr als eine Lösung möglich ist: $l > 1$. Darüber hinaus ist zu unterscheiden, ob es sich in einem *unterbestimmten* CSP um eine begrenzte Anzahl disjunkter *Einzellösungen* handelt, oder *kontinuierliche Lösungen* vorhanden sind. Kontinuierliche Lösungen zeichnen sich dadurch aus, dass sie eine unbegrenzte Anzahl an nicht-disjunkten *Einzellösungen* enthalten.

Es sei hiermit darauf hingewiesen, dass abweichende Definitionen der Bestimmtheit existieren:

- Sitharam, St. John u. a. (2019, Seite 140) definieren *wohlbestimmte Constraint-Systeme* als die, für die eine endliche Anzahl an diskreten *(Einzel-)Lösungen* existiert.
- In der linearen Algebra für Gleichungssysteme bedeutet *überbestimmt*, dass mehr Gleichungen m als *Variablen* n existieren. Dies resultiert nicht automatisch in einem Widerspruch im Gleichungssystem, sodass es gegebenenfalls lösbar ist.

Das Lösen eines CSPs ist in vielen Fällen *NP-schwer*[1] und stellt somit einen nicht zu unterschätzenden Aufwand dar. Die Software bzw. der Algorithmus, der ein Lösungsverfahren implementiert bzw. darstellt, wird als *Löser* (englisch: solver) bezeichnet.

[1] Ein Problem ist NP-schwer (englisch: NP-hard) wenn es in nichtdeterministischer polynomialer Zeit gelöst werden kann (vgl. Kelleners, 1999, Seite 12-13)

Es existieren dabei folgende Grundstrategien für effiziente *Solver*, welche üblicherweise kombiniert werden (nach Kelleners, 1999, Seite 13-15):

Problemreduktion: Verkleinerung des zu prüfenden Bereichs durch Ausschließen von redundanten Werten oder Werten, die zu keiner Lösung führen.

Baumsuche: Durchsuchen des Lösungsraums, in dem für einzelne *Variablen* der Wertebereich zerlegt wird und anschließend nur die Teilbereiche durchsucht und gegebenenfalls wieder zerlegt werden.

Lösungssynthese: Schrittweises Erzeugen der Lösung auf konstruktive Art durch Ausnutzung der Struktur des CSPs.

Die Fragestellungen, die bei einem CSP beantwortet werden können, unterteilen sich in folgende Klassen (nach Kelleners, 1999, Seite 15-16):

Einzelsuchproblem: Das Finden einer Lösung, die alle *Constraints* erfüllt.

Suche aller Lösungen: Das Finden aller Lösungen, die jeweils alle *Constraints* für sich erfüllen.

Optimierungsproblem: Das Finden der besten Lösung in Bezug auf eine vordefinierte Zielfunktion.

Überbestimmtes Problem: Für den Fall, dass keine Lösung möglich ist, das Finden einer Pseudolösung, die einer eigentlichen Lösung möglichst „nahe“ kommt.

Die Fragestellungen resultieren in teilweise sehr unterschiedlichen Vorgehensweisen und Laufzeiten. Es ist daher immer zu prüfen, welche ingenieurtechnische Fragestellung welcher Klasse an CSP-Fragestellungen zuzuordnen ist. In dieser Arbeit wird keine Suche von Lösungen für *überbestimmte* Probleme behandelt, da davon ausgegangen wird, dass sich der Suchraum ausschließlich aus den vorhanden *Constraints* ergibt.

2.4 Constraint-Graph

Zur Darstellung und Analyse von *Constraint-Systemen* existieren – in Abhängigkeit von der Art des beschriebenen Problems – verschiedene Möglichkeiten. Eine grundlegende Form ist der *Constraint-Graph*.

Definition 2.4.1 (Constraint-Graph)**.** Ein *Constraint-Graph* $G_S = (N, E)$ ist ein ungerichteter, bipartiter Graph. Dabei repräsentieren die Knoten N die beschränkten Objekte V und die Kanten E die *Constraints* C (vgl. Jermann u. a., 2006). Handelt es sich ausschließlich um einstellige (eine *Constraint* beschränkt genau ein Objekt) oder zweistellige *Constraints* (eine *Constraint* beschränkt genau zwei Objekte), so handelt es sich um einen einfachen Graphen. Bei höherwertigen *Constraints* (mehr als zwei beschränkte Objekte) handelt es sich um einen Hypergraphen (vgl. Freuder und Mackworth, 2006, Seite 14). Alternativ kann auch ein bipartiter Graph genutzt werden, wobei jeder Knoten ein beschränktes Objekt aus V oder eine *Constraint* aus C repräsentiert. Jede Kante verbindet dabei eine *Constraint* mit einem von ihr beschränkten Objekt (vgl. Jermann u. a., 2006).

Eine spezielle Form des *Constraint-Graphen* bei NCSP ist der *Gleichungsgraph*. Dabei handelt es sich bei den *Constraints* um Gleichungen und bei den beschränkten Objekten um *Variablen* in diesen Gleichungen (vgl. Jermann u. a., 2006).

2.5 Solid Modeling

Ein einzelner Körper ist eine abgeschlossene, dreidimensionale, volumetrische Form, die im Sinne des Modells als gesondert von den anderen Körpern betrachtet wird. Die Einteilung in einzelne Körper kann beispielsweise auf Basis von physischem Material geschehen. So können Ortbeton und die Bügel des Bewehrungsstahls in einer Betonwand einzelne Körper darstellen. Auch andere Einteilungen, beispielsweise auf Basis von Prozessen, ist möglich. So kann die Geometrie eines großen Betonfundaments, entsprechend der Fertigungsabschnitte in einzelne Körper zerlegt werden.

Die geometrischen Beschreibungsformen solcher (Fest-)Körper, (englisch: solids), ihre Erstellung und das Arbeiten mit ihnen werden unter dem Begriff *Solid Modeling* (deutsch: Festkörper-Modellierung) zusammengefasst.

Durch die teils widersprüchliche oder synonyme Verwendung von Begriffen für die Taxonomie der Repräsentationen ist eine klare begriffliche Unterscheidung notwendig. Beispielsweise bezeichnet Hillyard (1982, Seite 44) die Begriffe „unevaluated, implicit, direct or algorithmic schemes“ als Synonyme.

Das Ziel des *Solid Modelings* ist die Beschreibung valider Körper als eindeutige Teilmenge des dreidimensionalen Raums. Für die Validität kann dabei zwischen der Abgrenzung zum Raum außerhalb und der eindeutigen Belegung des Raums innerhalb eines Körpers unterschieden werden. Daraus ergeben sich angelehnt an Mäntylä (nach 1988, Abschnitt 6.3) folgende Kriterien:

Wasserdichtigkeit: (auch „geschlossene Haut“) Die Teilmenge des Raums muss eindeutig von dem nicht zugehörigen Raum abgegrenzt sein.

Überschneidungsfreiheit: Die Referenzierung von zugehörigen Teilen des Raums erfolgt eineindeutig. Das bedeutet, jeder Teil des Raums ist genau einmal von der Repräsentationsbeschreibung zu referenzieren.

Wird der zu beschreibende Körper als Teilmenge des dreidimensionalen euklidischen Raums aufgefasst, ergeben sich folgende mathematischen Konzepte der Beschreibung (nach Mäntylä, 1988, Abschnitt 3.4 - 3.5):

Direkte Mengenbeschreibung: In der Literatur auch als *Point-set models* bekannt. Dabei wird die Teilmenge des Raums als vollständige Punktmenge beschrieben.

Indirekte Mengenbeschreibung: In der Literatur auch als *Surface-Set Models* bekannt. Dabei erfolgt die Beschreibung der Menge indirekt über die topologische Oberfläche. Hierbei ist zu beachten, dass eine Orientierung notwendig ist, die eindeutig festlegt, auf welcher Seite der Oberfläche sich die zu beschreibende Menge befindet.

Ein Beispiel für eine *direkte Mengenbeschreibung* einer volumetrischen Einheitskugel ist die Ungleichung: $\sqrt{x^2+y^2+z^2} \leq 1$. Die indirekte Beschreibung ist dagegen: $\sqrt{x^2+y^2+z^2} = 1$ mit einer zusätzlichen Angabe eines Punkts (0/0/0) aus der Menge.

Darüber hinaus ist zu unterscheiden, ob alle benötigten Teilmengen, unabhängig davon, ob sie eine *direkte* oder *indirekte Mengenbeschreibung* ermöglichen, vollständig als *explizites Modell* (siehe Definition 2.1.1) vorliegen, oder, weil dies nicht möglich bzw. vorgesehen ist, als *implizites Modell* (siehe Definition 2.1.2).

Ein *implizites Schema* wird üblicherweise dann verwendet, wenn ein *explizites Schema* sehr kompliziert zu bilden oder speicherintensiv ist. Dabei liegen alle benötigten Teilmengen nicht *explizit* vor. Stattdessen ist eine Erzeugungsvorschrift enthalten, die den Lösungsweg darstellt (vgl. VDI 2209, Seite 25). Dafür kann die Zugehörigkeit zur Teilmenge des Körpers oder seiner Oberfläche eindeutig abgeleitet werden. Dies wird in der Literatur auch als *konstruktiv* oder *prozedural* beschrieben (vgl. ISO 10303-55, Definition 4.2).

Grundsätzlich existieren drei Hauptklassen, in die sich die Arten der Repräsentationen zuordnen lassen, welche individuelle Vor- und Nachteile mit sich bringen (vgl. Mäntylä, 1988, Kapitel 4-6; Shah und Mäntylä, 1995, Seite 52-53):

Zerlegungsmodelle: Der Körper als Teilmenge des Raums wird direkt beschrieben, indem benachbarte, vorher fest definierte Raumeinheiten referenziert und zu einer

Gesamtmenge zusammengefügt werden (vgl. Shah und Mäntylä, 1995, Kapitel 2.4.2). Die Raumeinheiten können den Raum gleichmäßig wie ein reguläres Gitter zerlegen. Die Blöcke, sogenannte Voxels, besitzen dabei identische Form und Größe. Sind die zusammenhängenden Raumeinheiten unregelmäßig, so handelt es sich um eine zelluläre Zerlegung, wie sie in der FEM (Finite Elemente Methode) für die Repräsentation der Netze genutzt wird.

Bei der rekursiven Zerlegung des Raums (englisch: space subdivision) werden Datenstrukturen wie zum Beispiel der Octree[2] genutzt um Raumeinheiten zu repräsentieren, die der Teilmenge des Körpers zugeordnet werden können. Form und Größe der Raumeinheiten sind entsprechend der Datenstruktur der Raumzerlegung definiert.

Konstruktive Modelle: *Konstruktive Modelle* sind *implizit* mit einem üblicherweise *imperativen Schema* (siehe Definition 2.1.3), finden sich aber auch unter den Synonymen *prozedurale* oder *generative Konstruktionsgeschichten-Modelle* (vgl. ISO 10303-108, Abschnitt 4.2) (vgl. VDI 2209, Seite 25) (vgl. Mäntylä, 1988, Kapitel 5; Shah und Mäntylä, 1995, Seite 52-53). Bei *konstruktiven Modellen* wird die Teilmenge des Körpers auf Basis von vordefinierten Teilmengen (Primitiven oder Sketches) und deren Kombination oder Manipulation durch Konstruktions-Operatoren beschrieben. Vordefinierte Teilmengen können dabei vorgefertigte geometrische Primitive, wie zum Beispiel Würfel, Kugel, etc., sein. Alternativ dazu können Sketches (deutsch: Zeichnungen oder Skizzen) in Form von meist zweidimensionalen, zusammengesetzten Formen, wie zum Beispiel Polygonen, genutzt werden. Dabei steht für die reine Repräsentation im Vordergrund, dass die Operatoren für die Auswertungen der Geometrie genutzt werden und nicht für die Erzeugung anderer Repräsentationen. Durch die Operatoren ist es nicht notwendig, die gesamte Teilmenge des Raums, die der Körper einnimmt, vorzuhalten, was gegebenenfalls auch nicht gewollt oder möglich ist.

Beispiele für Konstruktions-Operatoren sind *Boolesche Mengenoperatoren* (Vereinigung, Schnitt, Differenz) oder *translationales Sweeping*, bei dem eine zweidimensionale Form beschreibende Menge entlang einer Raumkurve „gezogen“ wird.

Die Umsetzung eines rein *konstruktiven Modells* ist beispielsweise die *Constructive Solid Geometry* (CSG). Dabei wird eine Binärbaumstruktur aufgebaut. In den Blättern finden sich definierte Mengen in Form von dreidimensionalen Primitiven.

[2]Zerlegt den dreidimensionalen Raum rekursiv. Die Datenstruktur ist ein Baum mit je 8 Kindern pro Knoten. Jeder Knoten repräsentiert einen Oktanten seines Elternknotens.

Die Knoten des Baums repräsentieren *Boolesche Mengenoperatoren*, welche die jeweiligen Unterbäume kombinieren. Eigenschaften, bzw. Vor- und Nachteile des CSG, werden von Hillyard (1982, Seite 44) und Mäntylä (1988, Abschnitt 5.24) genauer beschrieben.

Eine neuere Variante der *konstruktiven Modelle* sind die sogenannten *Sweeps* oder *Swept Surfaces*, bei denen die eigentliche Operation aus einer zweidimensionalen Form einen dreidimensionalen Körper formt (vgl. VDI 2209, Seite 262).

Berandungsmodelle: Diese Modelle werden auch als *Boundary representation* (deutsch: Begrenzungsflächenmodell; Abk. B-rep) bezeichnet. Die Beschreibung der Teilmenge des Raums erfolgt *indirekt* über den Rand als Zusammenfassung der gesamten Oberfläche eines Körpers. Der Rand eines Körpers ist dabei eine Menge von Randobjekten der zweiten Dimension, den *Faces* (deutsch: Facetten) (vgl. Mäntylä, 1988, Abschnitt 6.1). Da es sich grundsätzlich um eine *indirekte Mengenbeschreibungen* handelt, besitzt jede *Face* eine Orientierung. Die Orientierung dient dazu, Außen- und Innenraum des Körpers strikt voneinander zu unterscheiden. Die Ränder der *Faces* können dabei rekursiv betrachtet jeweils in um eine Dimension reduzierte Randobjekte zerlegt werden (vgl. Hoffmann und Rossignac, 1996). Dies kann auch als ein hierarchisches Modell aufgefasst werden, in dem höher-dimensionale Randobjekte durch eine Anzahl niederdimensionaler Randobjekte repräsentiert werden.

Für dreidimensionale Körper sind die sich ergebenden topologischen Objekte in Tabelle 2.1 dargestellt.

Tabelle 2.1: Topologischen Objekte einer *Boundary representation*, entsprechend ihrer Dimension, mit deutscher und englischer Bezeichnung

Dimension	**Deutscher Begriff**	**Englischer Begriff**
0	Eckpunkt	Vertex (auch Point)
1	Kante	Edge
2	Facette	Face
3	(Fest-)Körper	Solid

In der einfachsten Ausprägung sind B-rep-Modelle „Polygon-basierte Berandungsmodelle". Dabei wird der Rand ausschließlich aus einer Menge planarer Polygone beschrieben, welche die Koordinaten der Eckpunkte geordnet speichern (vgl. Mäntylä, 1988, Abschnitt 6.2.1). Dies ist für Visualisierungszwecke ausreichend. Die Prüfung, ob zwei Polygone eine gemeinsame Kante besitzen, ist ausschließlich

auf geometrischer Ebene möglich, indem die numerischen Werte der Koordinaten verglichen werden.

In den sogenannten Eckpunkt-basierten Modellen werden von den Polygonen die Koordinaten tragenden Eckpunkte referenziert (vgl. Mäntylä, 1988, Abschnitt 6.2.2). Dadurch wird der topologische Zusammenhang der *Faces* festgehalten.

Deutlich versiertere Formen stellen die *Kanten-basierten Berandungsmodelle* dar, in denen die vollständige topologische Hierarchie von der 3. bis zur 0. Dimension objektorientiert abgebildet wird (Mäntylä, 1988, Abschnitt 6.2.3). Ein Beispiel für kantenbasierte Modelle ist die Half-Edge Datenstruktur (vgl. Mäntylä, 1988, Kapitel 10; Shah und Mäntylä, 1995, Anhang B).

Berandungsmodelle haben gegenüber den anderen Repräsentationen direkte Vorteile. Beispielsweise ist für das Darstellen (englisch: rendering) von Geometrie im Rechner die *explizite Beschreibung* in Form einer B-rep die vorgeschriebene Form der verfügbaren Grafikschnittstellen (vgl. Mäntylä, 1988, Abschnitt 2.4.4). Aus diesem Grund sind *indirekte Beschreibungen* zur Darstellung erst in eine B-rep zu konvertieren. Die Konversion der einzelnen Repräsentationsformen ist dabei möglich, aber selten trivial. Hoffmann (2004) nennt speziell CSG zu B-rep als durchführbar, die Gegenrichtung nur in bestimmten Fällen. Dies ist speziell notwendig für duale Modelle, in denen zusätzlich zur führenden *indirekten Beschreibung* eine jederzeit darauf basierende B-rep vorgehalten wird (vgl. ISO 10303-55, Abschnitt 4.2.4).

Werden *implizite* und *explizite Schemen* kombiniert, spricht man von einem *Hybriden Modell* (ISO 10303-55, Abschnitt 4.2.2) (vgl. Mäntylä, 1988, Kapitel 7). *Hybride Modelle* sind in der Modellierungs-Software Stand der Technik (vgl. VDI 2209, Seite 157).

KAPITEL 3

Wissensrepräsentation und Modellierung digitaler Bauwerksmodelle

In diesem Kapitel wird eine Einführung in *Wissen* und in die Wissensrepräsentation in *Expertensystemen* gegeben. Die Konzepte zur Modellierung digitaler Bauwerksmodells werden vorgestellt und die unterschiedlichen Aspekte der eines Bauwerksmodells identifiziert und benannt. Es wird der *Mengen-basierte Entwurf* vorgestellt, der das Prinzip darstellt, wie mit einem Modell mit integriertem *Wissen* gearbeitet werden kann. Daran schließt eine Diskussion an über die *Wissensintegration* in Bezug auf Bauwerksmodelle.

3.1 Wissen und wissensbasierte Modellierung

Wissen ist kein einfach zu fassender Begriff und wird in dieser Arbeit folgendermaßen definiert:

Definition 3.1.1 (Wissen)**.** „[…] [ist] die (menschliche) Fähigkeit, Regeln zu erzeugen und anzuwenden (basierend auf Erfahrung, gestützt durch Vertrauen), also destillierte bzw. raffinierte oder neutralisierte bzw. abstrahierte menschliche Erfahrung […]“ (eigene Übersetzung aus Vajna u. a., 2009, Seite 429)

Dabei kann sich *Wissen* anhand des Grades der möglichen Formalisierung unterscheiden (vgl. Vajna u. a., 2009, Seite 432):

Implizites Wissen ist schwer zu formalisierendes *Wissen*, in Form unscharfer *Regeln* und vernetzter Informationen, das meist aus Erfahrung kommt.

Explizites Wissen ist leicht zu formulieren, beispielsweise in Form von Anleitungen und Dokumentationen.

Auch die Unterscheidung anhand der *Art des Wissens* ist dabei möglich:

Deklaratives Wissen ist die Beschreibung von Objekten und die Relationen untereinander auf Basis von Fakten und definierten Begriffen.

Prozedurales Wissen enthält Formeln bzw. *Regeln*, welche Abläufe für die Objekte des deklarativen *Wissens* definieren.

Da nur formal beschreibbare Formen des *Wissens* im Rechner abgebildet werden können, ist es notwendig, Strukturen zu nutzen, die dies ermöglichen. Dazu gehören beispielsweise Taxonomien, Relationen, *Regeln* bzw. Meta-Regeln. Das bedeutet, dass nur *explizites Wissen* identifizierbar in Modellen abgebildet werden kann. Die Integration von *Wissen* in das Modell wird unter dem Begriff *wissensbasierte Modellierung* zusammengefasst.

Definition 3.1.2 (Wissensbasierte Modellierung)**.** *Wissensbasierte Modellierung* ist die Integration von *Regeln*, Randbedingungen, Zwängen, Zusammenhängen und Beschreibungen, die ein Mensch formalisiert und in ein digitales Modell integriert. Das *Wissen* liegt dabei im Modell in der Form vor, dass der Rechner daraus Algorithmen ableiten oder es für Algorithmen verwenden kann.

3.2 Wissensrepräsentation in Expertensystemen

Digitale Wissensspeicherung bzw. Wissensrepräsentation erfordert, dass das *Wissen* in Form von *expliziten Modellen* bzw. der sie beschreibenden Sprachen vorgehalten wird. Die Idee, *Wissen* in einem Modell zu hinterlegen, welches anschließend für das Lösen von Fragestellungen im Kontext des Modells genutzt werden kann, wird unter dem Begriff *Expertensystem* zusammengefasst. Nach Haun (eigener Übersetzung, 2016, Seite 304) stellen *Expertensysteme* „[…]eine Weiterentwicklung der klassischen Wissensspeicherung durch schriftliche oder elektronische Datenbanken dar. *Expertensysteme* speichern nicht nur *Wissen*, sondern ziehen das *Wissen* auch zu Schlussfolgerungen heran. Sie besitzen daher die Fähigkeit, das abgespeicherte *Wissen* zielgerichtet zu kombinieren. Bestandteile eines *Expertensystems* sind dementsprechend neben der Wissensbasis und der Wissenserwerbskomponente auch die Problemlösungs-, Erklärungs- und Dialogkomponente. *Expertensysteme* lassen sich im Bereich der Wissensspeicherung deshalb insbesondere dann sinnvoll einsetzen, wenn Fakten-, Handlungs- und Rezeptwissen abgespeichert wird,

die den Anwender später bei der Diagnose, Planung, Beratung, Entscheidungsfindung und Koordination unterstützen sollen."

Probleme, die von einem solchen *Expertensystem* gelöst werden sollen, besitzen nach Neumann u. a. (1987) unter anderem folgende Merkmale:

1. Es existiert ein großer Lösungsraum.
2. Der Lösungsweg lässt sich oft hierarchisch strukturieren.
3. Die Abhängigkeiten der einzelnen Objekte ist von zentraler Bedeutung.

Ein *Expertensystem* dient also dazu, auf Basis von (Experten-) *Wissen* ein Problem zu lösen und dabei ein Modell so zu konfigurieren, dass es unter der Prämisse, dass die Wissensbasis ausreichend ist, möglichst sinnvolle Ergebnisse liefert. Dabei existieren laut Günter u. a. (1999) unterschiedliche Konfigurationsmethoden für *Expertensysteme*. Diese definieren die Form von *Wissen*, wie es in ein digitales Modell integriert werden kann:

Regel-basierte Systeme: Eine Menge von Wenn-Dann-Entscheidungen auf Basis von fest definierten Regeln.

Konzept-Hierarchien: Klassifikation von Objekten und damit Bildung von Konzepten.

Struktur-basierter Ansatz: Konfiguration auf Basis von vorgegebenen Entscheidungsbäumen.

Constraint-basierte Systeme: Beschränkungen werden als Relationen zwischen Objekten repräsentiert und können ausgewertet werden.

Ressourcen-basierter Ansatz: Komponenten eines technischen Systems werden gewählt und genutzt, weil sie vom Gesamtsystem oder einzelnen anderen Komponente benötigt werden. Die Schnittstellen zwischen den Komponenten definieren die ausgetauschten „Ressourcen".

Fall-basierte Konfigurationen: Identifikation und Wahl einer Lösung auf Basis bereits gelöster ähnlicher Fälle.

Es existierte ein Forschungsprojekt „FLAKON: Expertensystemkern für Planungs- und Konfigurierungsaufgaben" (1986-1990), in dessen Zuge ein generisches Modell entwickelt wurde, für das einige dieser Konfigurationsmethoden umgesetzt wurden (Zusammenfassung in Cunis u. a., 1991).

Neuere Forschungen beinhalten die Entwicklung eines Datenmodells für *Expertensysteme*. Runte (2006, Seite 2) stellt ein solches vor, welches „[…]den problemlosen Austausch bzw. den domänenspezifischen Einsatz von Constraint-Lösungstechnologien ermöglicht.“.

Eine Anwendung solcher Systeme auf Bauwerksmodelle ist nicht bekannt.

3.3 Definition eines digitalen Bauwerksmodells

In verschiedenen Quellen finden sich Definitionen des digitalen Bauwerksmodells – oft *Building (Information) Model* genannt –, die sich wenig bis stark unterscheiden, bzw. sehr allgemein bis sehr detailliert sind. Nach DIN EN ISO 29481 (Definition 3.2) ist es eine „[…][digitale] Repräsentanz eines Bauwerks (inkl. Gebäude und Infrastrukturbauwerke, usw.), um die Prozesse der Bauplanung, der Baukonstruktion und des Bauwerksbetriebs zu erleichtern und eine verlässliche Entscheidungsgrundlage bereitzustellen“. Dies umfasst „[…]alles, was konstruiert wird oder das Ergebnis baulicher Tätigkeiten ist.“ (DIN EN ISO 29481, Definition 3.7). Ein digitales Bauwerksmodel ist nach dieser Definition ein Produkt(informations)Modell, da es „[…]ein Informations Modell [ist], welches eine abstrakte Beschreibung von Fakten, Konzepten und Instruktionen eines Produktes [darstellt] […]“ (ISO 10303-1, Definition 3.2.29). Das zugehörige *Produkt* ist das „Bauwerk“, das ein „[…]Ding […][ist, welches] durch einen natürlichen oder künstlichen Prozesses erstellt wird“ (ISO 10303-1, Definition 3.2.26).

Wichtigste Anforderung und größter Unterschied zu vorherigen Technologien – wie beispielsweise technische Zeichnungen als Resultat des klassischen *Computer-Aided Draftings* – ist dabei, dass ein solches Bauwerksmodell die Daten auf eine formale, computerinterpretierbare Art speichert (vgl. Underwood u. a., 2010, Seite 2).

Nach Eastman, Teicholz u. a. (2011, Seite 16) und in ähnlicher Form auch bei Borrmann, König u. a. (2015, Kapitel 1.2) zeichnet sich ein Bauwerksmodell folgendermaßen aus: Es besteht aus digitalen Repräsentationen (Objekten) von Bauteilen. Diese Objekte besitzen eine Geometrie und Eigenschaften, die sie für Software-Anwendungen nutzbar machen.

Darüber hinaus nennen Eastman, Teicholz u. a. (2011, Seite 16) noch einen weiteren Punkt: Die Objekte besitzen *parametrische Regeln*, die es ermöglichen, sie auf intelligente Art zu verändern. Dies geht über die reine Repräsentation eines realen Bauteils hinaus, da hier das Arbeiten mit dem digitalen Modell im Vordergrund steht.

Die Natur eines digitalen Bauwerksmodells lässt sich entsprechend über die Beantwortung dreier Fragen beschreiben:

Was wird repräsentiert? Ein Bauwerksmodell repräsentiert ein Bauwerk, wobei es, aufgrund der Menge der vorhandenen Informationen, in Objekte strukturiert wird. Diese stellen üblicherweise architektonische oder ingenieurtechnische Einteilungen dar, beispielsweise in Form von Etagen, Räumen, Bauteilen, etc.

In welcher Art wird es repräsentiert? Es sind unterschiedliche Arten von Daten zu unterscheiden: Die *geometrischen Daten*, welche die geometrische Form des Bauwerks und seiner Teile beschreiben und nicht-geometrische Daten. Dazu gehören beispielsweise die *semantischen Daten*, welche die Grundlage der Organisation des Modells bilden und Funktion und Zusammenhänge der Objekte beschreiben. Andere nicht-geometrische Daten sind zum Beispiel *physikalische Daten*, welche die physikalischen Eigenschaften und Zusammenhänge der Komponenten beschreiben.

Was ist über die reine Repräsentation hinaus enthalten? Soll das Modell als reiner Datencontainer dienen und einen Zustand des Bauwerks abbilden, so sind keine weiteren Daten als die der reinen Repräsentation notwendig. Es können aber zur Verwaltung, Kommunikation und Fortschreibung des Modells weitere Daten enthalten sein. Ein Anwendungsfall ist beispielsweise die computergestützte Modellerstellung und -veränderung, bei der die Daten der Kommunikation der Beteiligten oder zur Auswertung von *Regeln* dienen, welche automatisiert auf das Modell angewendet werden können.

Entgegen anderer Definitionen, welche einem digitalen Bauwerksmodell die Notwendigkeit zusprechen, alle Daten für alle Prozesse des Gebäude-Lebenszyklus zu enthalten (vgl. Underwood u. a., 2010, Seite 2), wird in dieser Arbeit davon ausgegangen, dass ein digitales Bauwerksmodell mindestens einen Zustand des repräsentierten Bauwerks beschreibt und nur dieser eine Zustand relevant ist. Andere Zustände, die sich zum Beispiel aus verschiedenen Bauphasen und deren Zusammenhang ergeben, werden nicht betrachtet.

3.4 Der Modellcharakter eines digitalen Bauwerksmodells

Zur Abgrenzung, was ein digitales Bauwerksmodell beschreibt und in was es sich von anderen Modellen unterschiedet, ist der Begriff „Modell“ zu erklären. Allgemein besitzen Modelle laut Stachowiak (1973, Abschnitt 2.1.1) drei Hauptmerkmale:

Abbildungsmerkmal: „Modelle sind stets […] Abbildungen, Repräsentation natürlicher oder künstlicher Originale […]“

Verkürzungsmerkmal: „Modelle erfassen im allgemeinen […] nicht alle Attribute des repräsentierten Originals, sondern nur solche, die den jeweiligen […] Modellbenutzern relevant erscheinen“

Pragmatisches Merkmal: „Modelle sind ihren Originalen nicht per se eindeutig zugeordnet. Sie erfüllen ihre Ersetzungsfunktion […] innerhalb bestimmter Zeitintervalle und unter Einschränkung auf bestimmte gedankliche oder tatsächliche Operationen.“

Für ein digitales Bauwerksmodell folgt entsprechend dem *Abbildungsmerkmal*, dass es die Repräsentation eines Bauwerks ist, wobei die enthaltenen Daten mindestens einen konkreten Zustand des Bauwerks beschreiben. Bei diesen Zuständen kann es sich um verschiedene Varianten oder Baufortschritte handeln. In der Planung ist typischerweise der führende Zustand derjenige, der das finale Bauwerk am Ende des Errichtungsprozesses repräsentiert. Hingegen ist der führende Zustand der Bewirtschaftung eines existierenden Gebäudes der aktuelle Stand der Realität. Daraus folgt, dass die im Bauwerksmodell gespeicherten Daten unterschiedliche Ursprünge besitzen: Sie können auf Basis realer existierender Informationen, auf Basis menschlicher Planung und Vorhersehung oder einer Kombination beider entstehen.

Ein digitales Bauwerksmodell kann aufgrund des *Verkürzungsmerkmals* nur eine Teilmenge der Eigenschaften des realen Bauwerks repräsentieren. Deshalb obliegt es dem Menschen – die Unterscheidung zwischen einem Modellersteller und einem Modellnutzer ist im Kontext dieser Arbeit nicht relevant –, die Vorgaben und Planungsideen, welche zum Zeitpunkt der Modellbearbeitung existieren, soweit wie möglich umzusetzen. Hierbei ist zu bemerken, dass auch Eigenschaften des digitalen Bauwerksmodells vorhanden sein können, die keine Entsprechung im realen Bauwerk besitzen. So sind zum Beispiel Identifikatoren einzelner Bauteile im digitalen Prozess relevant, fließen aber in keiner Form in das reale Bauwerk ein. Auch geometrisch-konstruktive Ideen, wie beispielsweise Hilfslinien oder die Parallelität bestimmter Linien, können zwar im digitalen Modell als *Konstruktionsregeln* enthalten sein und eine Rolle spielen, in der Realität ist aber maximal ihre Auswirkung auf das Ergebnis sichtbar. In dieser Arbeit soll dies unterschieden werden, indem die Existenz eines Objekts auf zwei Arten begründbar ist. Entweder ist es *ingenieurtechnisch relevant*, das bedeutet, es hat eine Entsprechung in der Realität oder wird für Prozesse der Realität benötigt, oder es ist *modelliertechnisch relevant*, was wiederum bedeutet, dass es allein in dem virtuellen Raum und den dort aktiven Prozessen einen Zweck erfüllt.

Das pragmatische Merkmal ergibt sich aus dem *Verkürzungsmerkmal*. Da nur eine definierte Menge an Eigenschaften im digitalen Modell existieren kann, ist der Zweck des Modells die Grundlage, auf derer diese Menge definiert wird. So müssen zum Beispiel in einem Modell, welches als Datengrundlage für die Errichtung eines Gebäudes dienen soll, die notwendigen Eigenschaften für die Bestellung der zu liefernden Bauteile enthalten sein. Dies kann die Produktnummer eines Fenstertyps eines konkreten Herstellers oder die geometrische Form eines Stahlträgers sein. Hierbei ist zu beachten, dass ein Modell einen übergeordneten Zweck haben kann, der sich aus der Vereinigung aller Anforderungen aus den verschiedenen Prozessen, für die das Modell genutzt werden soll, ergeben kann. Die unterschiedlichen Zwecke eines digitalen Bauwerksmodells lassen sich folgendermaßen kategorisieren:

Kommunikationszweck: Das Modell dient als Kommunikationsgrundlage für Beteiligte, beispielsweise in Form einer dreidimensionalen Visualisierung des finalen Bauwerks. Auch kann der Zweck eines Modells eine gemeinsame Datenquelle zur Kommunikation sein, zum Beispiel in Form von sprachlich verfassten Kommentaren im Modell.

Zweck der realen Ausführung: Das Modell dient als Grundlage für die Erstellung von Daten und Dokumenten für Genehmigungsverfahren oder dem realen Bauprozess des Bauwerks, wie zum Beispiel Bestellungen von Bauprodukten, Ausschreibungen oder Plänen und Datenblättern.

Zweck der Anforderungsprüfung: Da der Anteil der Bauwerke, die typisiert sind und in großer Stückzahl errichtet werden, aus ästhetischen wie auch praktischen Gründen gering ist, haben sie im Allgemeinen einen Unikats-Charakter. Auch die Qualität des Bauwerks und der Prozesse seiner baulichen Änderungen wird durch unterschiedliche, teilweise nicht direkt beeinflussbare Bedingungen geprägt, wie zum Beispiel die Wetterlage, der Baugrund, Normen und Gesetze, die Ausführungsqualität, Verfügbarkeit und Preis von Bauprodukten und die beteiligten Unternehmen. Daraus ergibt sich, dass es nicht sinnvoll, praktikabel und finanzierbar ist, Erprobungen, Optimierungen und Verbesserungen vorab an realen Prototypen des Bauwerks durchzuführen, wie es bei Produkten in anderen Industriezweigen üblich ist. Da aber ein reales Bauwerk unterschiedlichsten Anforderungen gerecht werden muss, wie zum Beispiel gesetzlichen Vorgaben oder den Wünschen des Bauherrn, verspricht das digitale Bauwerksmodell eine Lösung. Es kann durch seinen prototypischen Charakter vorab für notwendig Prüfungen der Anforderungen herangezogen werden. Dazu zählen beispielsweise statische und energetische Nachweise, Ent-

fluchtungssimulationen, bauliche Umsetzbarkeit (dazu gehört beispielsweise die Kollisionsfreiheit von Bauteilen) usw.

3.5 Objektorientierung in einem digitalen Bauwerksmodell

Wie bereits in Abschnitt 3.3 beschrieben, setzt sich ein digitales Bauwerksmodell aus Objekten zusammen. Das digitale Objekt als Teil des Modells ist ein Produkt des Paradigmas der *Objektorientierung*, welches eine große Rolle im *Building Information Modeling* (BIM) spielt (vgl. Underwood u. a., 2010, Seite 179-180).

Das aus der Informatik stammende Konzept der *Objektorientierung* fußt darauf, dass ein Modell, welches ein komplexes System darstellt, besser verstanden und verarbeitet werden kann, wenn es in eine Struktur aus kleineren Teilen, den Objekten, zerlegt wird. Diese Objekte sind dabei klar identifizierbare Teile des Systems. Sie enthalten Daten in Form von Attributen und gegebenenfalls *Regeln*.

Nach Booch u. a. (2008, Abschnitt 1.5) ergeben sich mit der *Objektorientierung* folgende Möglichkeiten der Ordnung des Systems:

Zerlegung: Durch die Zerlegung in kleine, „autonome“ Einheiten können die Daten eindeutig den repräsentierten Entitäten zugeordnet werden. Dabei geschieht die Verwaltung der Daten eigenverantwortlich in den Objekten (vgl. Shalloway u. a., 2002, Seite 110-112).

Abstraktion: Objekte können Gemeinsamkeiten aufweisen, welche es ermöglichen, eine Abstraktion vorzunehmen und die Objekte gleichzeitig zu klassifizieren.

Hierarchie: Durch die Einführung einer hierarchischen Abstufung der Einheiten ist es möglich, den semantischen Gehalt eines Modells zu erhöhen. Die Identifikation einer sinnvollen Hierarchie ist dabei selten einfach, führt aber zu einer starken Erhöhung der Nachvollziehbarkeit und Verständlichkeit des Modells.

Im Folgenden sollen diese Konzepte bei der Modellbildung eines digitalen Bauwerksmodells vorgestellt werden.

3.5.1 Zerlegung des Bauwerksmodells

Die Zerlegung von Modellen in einzelne identifizierbare Teile (englisch: chunks), ist eine natürliche Art, *Wissen* in ein Modell zu integrieren (vgl. Shah und Mäntylä, 1995, Seite 10-11).

Bei einem digitalen Bauwerksmodell, welches ein ingenieurtechnisches Bauwerk repräsentiert, ist es naheliegend, die objektorientierte Struktur analog zu der Struktur des realen Bauwerks im ingenieurtechnischen Sinne aufzubauen. Das bedeutet beispielsweise, dass die Zerlegung in Einheiten – wie Bauteile, Räume, Etage, Leitungssystem etc. – vorzunehmen ist. Diese Einheiten bilden zusammen das System des Modells und können gegebenenfalls noch weiter zerlegt werden. Wie fein eine solche Zerlegung erfolgt und welche ingenieurtechnischen Domänen dabei berücksichtigt werden, hängt von dem Zweck des Modells ab.

Außerhalb des Bauwesens finden sich vergleichbare Konzepte. So existiert im Maschinenbau das *Feature-basierte Modellieren*. Dabei liegt der Fokus auf dem Herstellungsprozess einzelner Bauteile und damit vor allem dem Herausarbeiten der geometrischen Form (vgl. Hoffmann und Rossignac, 1996). Ein *Feature* (deutsch: Merkmal bzw. Funktion) ist dabei die Repräsentation von physischen Teilen oder Baugruppen, die ingeneurtechnische Relevanz hat (vgl. Shah und Mäntylä, 1995, Seite 97). Shah, Sreevalsan u. a. (1988) fassen folgende Eigenschaften eines *Features* zusammen:

- Es ist ein physischer Bestandteil eines (Bau-)Teils.
- Es kann allgemein auf eine geometrische Form abgebildet werden.
- Es hat ingenieurtechnische Relevanz.
- Es hat vorbestimmte Eigenschaften.

Das *Feature-basierte Modellieren* steht auch im Zusammenhang mit dem *parametrischen Modellieren* (siehe Abschnitt 4.3.1), da die Eigenschaften eines *Features* zur Steuerung der geometrischen Form genutzt werden können.

3.5.2 Abstraktion zur Strukturierung

In der *objektorientierten Modellierung* findet die Abstraktion mithilfe der *Klassen* statt. Jedes Objekt ist dabei die Instanz einer *Klasse* und besitzt deren Eigenschaften und deren *Regeln*. Objekte derselben *Klasse* besitzen dieselbe Menge an Eigenschaften, welche sie individuell ausprägen. Die Abstraktion findet im Bauwerksmodell entsprechend der ingenieurtechnischen Einteilung statt. Dies geschieht beispielsweise auf Basis von Gemeinsamkeiten, die entsprechend im Anwendungskontext identifiziert werden. Sie kann aber auch auf Basis eines Klassifikationssystems erfolgen. ISO 12006-2 dient als Grundlage für Klassifikationssysteme im Bauwesen, das die Bauteile repräsentierenden Objekte

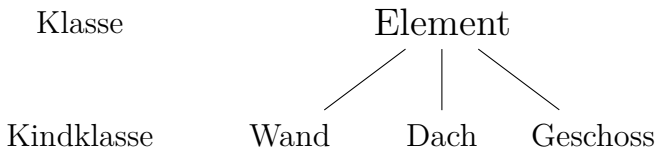


Abbildung 3.1: Klassifikation von Komponenten eines Bauwerks in einer Hierarchie (in eigene Übersetzung entnommen aus ISO 12006-2, Abbildung 2, links)

entsprechend in *Klassen* und *Unterklassen* unterteilt. Ein Beispiel ist in Abbildung 3.1 gegeben.

Eine Möglichkeit der Abstraktion von digitalen Bauteilobjekten wird von Gielingh (2008) genannt. Dieser unterteilt die Beschreibung von Objekten in vier Ebenen:

Generische Ebene: Eine Beschreibung auf Basis von *Parametern*, wobei nicht alle *Parameter* mit Werten spezifiziert sind.

Spezifische Ebene: Eine Beschreibung, die alle Parameterwerte spezifiziert, aber für mehrere Objekte verwendet werden kann.

Individuelle Ebene: Individuelle Objekte, deren Eigenschaften über den Lebenszyklus variieren können.

Auftritts-Ebene: Der Zustand eines Objekts zu einem bestimmten Zeitpunkt.

Diese Einteilung findet sich auch in aktuellen Softwareprodukten zur Modellierung von Bauwerksmodellen, wie zum Beispiel *Autodesk Revit*. Dabei wird in *Autodesk Revit* der generische Typ als *Familie* (englisch: family), der spezifische Typ als *Typ* (englisch: type) und das Exemplar als *Instanz* (englisch: instance) bezeichnet. Eine Entsprechung für das *Auftreten* ist in *Autodesk Revit* ausschließlich über die Zuordnung zu genau einer *Projekt-Phase* vorgesehen.

3.5.3 Hierarchische Strukturierung

Die hierarchische Strukturierung kann in der *objektorientierten Modellierung* auf verschiedene Art und Weise stattfinden. Durch die Klasse-Objekt-Struktur kann eine Klassenhierarchie aufgebaut werden, die eine Beschreibung der verschiedenen Ebenen der Abstraktion abbildet. Außerdem können direkte Relationen zwischen den Objekten bestehen, woraus sich eine Hierarchie auf Basis der Bedeutung der Relationen ergibt. Beispielsweise kann dies eine Aggregation sein, deren Semantik darin besteht, dass ein Objekt aus einem oder mehreren anderen Objekten zusammengesetzt wird. Dabei kann eines der Objekte

auch ohne das Ganze existieren (vgl. Booch u. a., 2008, Seite 63-64; Neumann u. a., 1987). Für ein digitales Bauwerksmodell wird üblicherweise eine Komposition gewählt, bei der ein Teil nur existieren kann, wenn auch das Ganze existiert. Dies ergibt beispielsweise folgende Einteilung: Ein Gebäude besteht aus Gebäudeteilen, diese bestehen aus Etagen, diese bestehen aus Räumen und Bauteilen, Bauteile bestehen aus Einzelkomponenten, usw. Nach ISO 12006-2 (Abschnitt B.3) kann eine Klassifikation von Bauteilen und ihren Komponenten ebenfalls in einer Hierarchie abgebildet werden. Ein Beispiel für ein Gebäude und seine Zusammensetzung ist in Abbildung 3.2 gegeben.

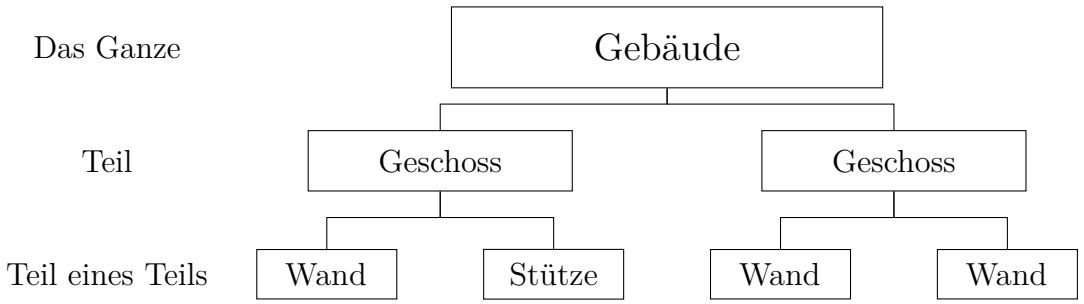


Abbildung 3.2: Strukturierung von Komponenten eines Bauwerks als Kompositions-Hierarchie (angelehnt an ISO 12006-2, Abbildung 2, rechts)

3.6 Aspekte eines digitalen Bauwerksmodells

3.6.1 Ein erläuterndes Beispiel

An einem Beispiel sollen die einzelnen Aspekte eines digitalen Bauwerksmodells gezeigt werden, wie sie für diese Arbeit identifiziert werden. Ein Bauwerksmodell ist üblicherweise bauteilzentriert bzw. objektorientiert aufgebaut. Dies bedeutet, dass darin als kleinste Einheiten Bauteile oder vergleichbare Einheiten, wie zum Beispiel Räume, enthalten sind.

Am Beispiel eines Fensters wird deutlich, welche unterschiedlichen Aspekte abgebildet werden. Das Fenster ist in Abbildung 3.3 dargestellt.

Das Fenster selbst besteht aus Attributen (englisch: attribute oder property), welche die Werte enthalten, die verschiedene Eigenschaften des Fensters festlegen. So kann es beispielsweise die Attribute „Verglasung“ oder „Rahmenbreite“ besitzen. Die Attribute werden den semantischen Daten des Fensters zugeordnet. Daneben gehört zu einem Fenster üblicherweise eine geometrische Beschreibung. Hierbei handelt es sich nicht um Attribute mit Maßen des Fensters, wie „Höhe“, „Breite“, „Tiefe“, sondern um eine geometrische Repräsentation des Fensters. Die „klassische“ geometrische Repräsentation

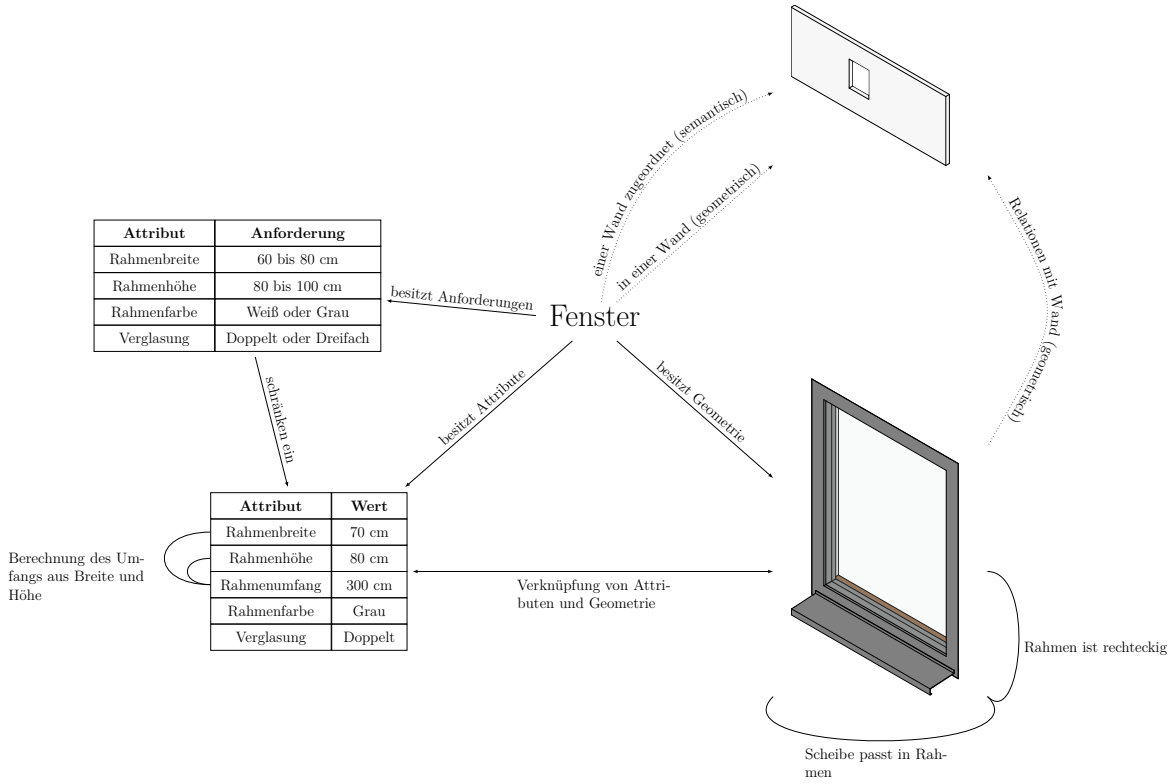


Abbildung 3.3: Auszug verschiedener Aspekte eines Bauwerksmodells am Beispiel einer Entität „Fenster“

im Bauwesen stellt die technische Zeichnung dar, welche aber von dreidimensionalen, geometrischen Modellen abgelöst werden. Diese Modelle gehen über die reine Visualisierung hinaus. Sie können zur algorithmischen Ableitung von Informationen genutzt werden. Der Zweck einer geometrischen Zeichnung – als zweidimensionales Bild – ist es üblicherweise, Maße, Form und Material, als von Menschenen interpretierbar darzustellen: In geometrischen Modellen sind solche Informationen vom Comptuter interpretierbar abgelegt. Die Daten sind dabei *explizit* enthalten, oder sie können *implizit* abgeleitet werden. So lässt sich die Brüstungshöhe eines Fensters aus der Höhendifferenz zwischen Fensterbrüstung und Fertigfußboden oder Rohdecke berechnen. Dies ist in technischen Zeichnungen nicht möglich, da die Zahl dort explizit in Form einer Beschriftung dargestellt wird. Eine Ausnahme stellt die Ableitung einer technischen Zeichnung aus einem dreidimensionalen geometrischen Modell dar. Dabei geht üblicherweise der Bezug zwischen technischer Zeichnung und dem 3D-Modell durch die verwendeten Austauschformate verloren. Neben der Semantik in Form von Attributen, Relationen und Klassifikation und der geometrischen Repräsentation kann ein Bauteil in einem Bauwerksmodell auch Gren-

zen oder Zusammenhänge dazwischen beinhalten. Dazu zählen Zusammenhänge zwischen Attributen, zwischen Attributen und geometrischer Repräsentation oder innerhalb der geometrischen Repräsentation. Diese Zusammenhänge stellen eine mögliche Logik in Form von *Regeln* oder *Constraints* dar. Grenzen und Zusammenhänge können als eine Form von *Wissen* betrachtet werden, da sie nicht immer notwendig sind, um den Zustand des Modells zu beschreiben. Sie sind aber sehr wohl nützlich, wenn Änderungen am Modell vorgenommen werden, da sie auswertbar sind und das Resultat beeinflussen können. Werden beispielsweise Attribute mit anderen Attributen oder mit der geometrischen Repräsentation verknüpft, so wird dies als *parametrisches Modellieren* bezeichnet (siehe Abschnitt 4.3.1). Es folgt daraus, dass sich auf der Ebene eines Bauteils drei Aspekte ergeben: die Semantik, die Geometrie und das *Wissen* über den Zusammenhang und die Anforderungen. Weitere, wie beispielsweise physikalische oder prozessorientierte Aspekte werden in dieser Arbeit nicht betrachtet, da sie für die Ziele der Arbeit keine Relevanz besitzen.

3.6.2 Semantische Objektbildung und -strukturierung

Jeder Aspekt erfordert die Beschreibung der erforderlichen Objekte und deren Strukturierung. Die Repräsentation der Entitäten als Objekte und deren Attribute und strukturierende Relationen sind der Semantik zuzuordnen. So kann die semantische Struktur eines Gebäudes beispielsweise als eine Hierarchie *Gebäude* →*Etage* →*Raum* →*Bauteil* aufgebaut sein (vgl. Rasmussen u. a., 2017).

3.6.3 Geometrische Objektbildung und -strukturierung

Die Geometrie des digitalen Bauwerksmodell ist entsprechend der Zerlegung den einzelnen Einheiten zuzuordnen. Wird eine Einheit in kleinere Einheiten zerlegt, ist dies auch in der Geometrie abzubilden, da sich sonst semantische oder geometrische Strukturierung widersprechen. Dies kann zu nicht nachvollziehbaren Ergebnissen bei der Auswertung des Modells führen.

Der reale, physikalische, dreidimensionale Raum kann nicht vollständig in den begrenzten Speicherplatz eines Rechners abgebildet werden. Deshalb ist es notwendig, eine digitale Repräsentation der Geometrie zu wählen, welche die Geometrie korrekt und für den Zweck des Modells vollständig abbilden kann. Für ein Bauwerksmodell, welches für eine bauphysikalische Überprüfung auf Wärmebrücken genutzt werden soll, bedeutet dies, dass beispielsweise keine Dimensionsreduzierung stattfinden darf. Reduzierte Darstellungsformen, wie zweidimensionale Schnitte, kommen also nicht in Frage. Darüber hinaus

muss die Geometrie algorithmisch auswertbar sein. Dies schließt klassische zweidimensionale Pläne ebenfalls aus, da deren grafische Darstellung nur eingeschränkt automatisch interpretiert werden können und Fehlinterpretationen seitens der Algorithmen nicht ausgeschlossen werden können. Das bedeutet, dass die grafische Darstellung in praktikabler Weise nur vom Menschen sinnvoll interpretiert werden kann.

3.6.4 Wissensbildung über die Anforderungen und den Zusammenhang und deren Strukturierung

Da eine Entität innerhalb des Bauwerksmodells, wie beispielsweise ein Bauteil, in dem Modell unter verschiedenen Aspekten repräsentiert wird, sind die unterschiedlichen Aspekte aufeinander abzustimmen. Bezogen auf die Objektbildung stellt dies die Konsistenz zwischen den semantischen Attributen und der geometrischen Repräsentation dar. Auch die Strukturierung ist zwischen den Aspekten abzustimmen. Ein Widerspruch zwischen den Aspekten tritt dann auf, wenn sich beispielsweise eine Treppe geometrisch zwischen der 5. und 6. Etage eines Hauses befindet, sie semantisch aber als Teil der 2. Etage im Modell hinterlegt ist. Diese Zusammenhänge und Grenzen sind dem meist wenig berücksichtigten Aspekt des *Wissens* zuzuordnen. Dieser setzt sich aus der *Wissensbildung* und der *Wissensstrukturierung* zusammen.

Dabei werden die Grenzen zum einen aus den Anforderungen gebildet. Die Anforderungen an ein Bauwerksmodell können beispielsweise in einem Anforderungsmodell integriert sein (vgl. Kiviniemi, 2005, Kapitel 6). Zum anderen ergeben sie sich aus Zwängen, welche die Modellvalidität und -konsistenz sicherstellen sollen.

Das *Wissen* über den Zusammenhang stellt die Verbindung zwischen und innerhalb von Semantik und Geometrie dar. Dabei liegt in der Beschreibung des Zusammenhangs das eigentliche *Wissen* über den Entwurf. Diese Bestandteile des *Wissens* sind ebenfalls zu strukturieren. Dabei ist auch die Zuordnung zu einzelnen Entitäten möglich. Ist der Zusammenhang nicht oder nur unzureichend vorhanden, ist die Konsistenz zwischen den einzelnen Aspekten durch den Menschen sicherzustellen, was erfahrungsgemäß selbst bei weniger umfangreichen Modellen zu Fehlern führt.

Es ergibt sich ein System der Wechselwirkungen zwischen den Aspekten (siehe Abbildung 3.4).

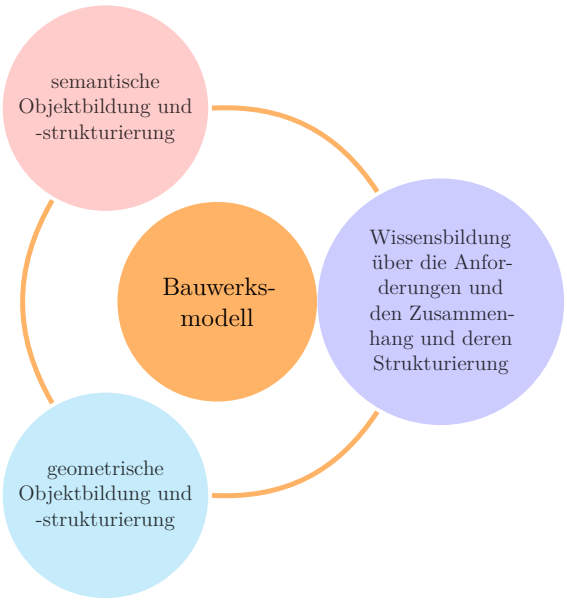


Abbildung 3.4: Aspekte eines Bauwerksmodells und ihre Wechselwirkungen untereinander

3.7 Aktueller Stand der digitalen Bauwerksmodelle

In aktuell verfügbaren Softwareprodukten zur Bauwerksmodellierung werden verschiedene Aspekte adressiert. Betrachtet man aktuell verfügbare Softwareprodukte, wie *Nemtschek Allplan*[1], *Graphisoft ArchiCAD*[2], *Autodesk Revit*[3] oder *Bentley Microstation*[4], so fällt auf, dass sich die Aspekte der *semantischen Objektbildung und -strukturierung* und der *geometrischen Objektbildung und -strukturierung* vollständig wiederfinden. Die *Wissensbildung* ist hingegen nur in Ansätzen integriert. Zwar ist es möglich, Zusammenhänge und Abhängigkeiten zu modellieren, betrachtet man allerdings die Geometrie, so ist dies nur bis zu einem gewissen Grad möglich. Speziell auf die Probleme des geometrischen Modellierens wird genauer in Kapitel 4 eingegangen.

Anforderungen beispielsweise in Form von gültigen Wertebereichen für *Parameter* können in den Vanilla-Versionen (die ausgelieferte, nicht veränderte Software, die nicht durch zusätzliche Softwarekomponenten erweitert wurde) nicht integriert werden und können auf Grund der zur Auswertung notwendigen Zusammenhänge nicht für den Entwurf genutzt werden. Auf Grund des Fehlens solcher *Wissensintegration* in den Softwareprodukten fehlt folgerichtig auch die Strukturierung des *Wissens*.

[1]https://www.allplan.com/ (abgerufen am 25.10.2019)
[2]https://www.graphisoft.de/archicad/ (abgerufen am 25.10.2019)
[3]https://www.autodesk.de/products/revit/overview (abgerufen am 25.10.2019)
[4]https://www.bentley.com/de/products/brands/microstation (abgerufen am 25.10.2019)

Bei dem offenen Austauschformat IFC (Industry Foundation Classes) ist der Zustand vergleichbar. Anforderungen lassen sich nicht für einzelne *Merkmale* (Synonym in den IFC für *Attribut*; englisch: property) festlegen. Zwar existiert der Ansatz, *Merkmale* in Zusammenhang zu setzen, doch dies kann nach aktuellem Stand ausschließlich als textlicher Ausdruck geschehen[5]

Anforderungen werden nach aktuellem Stand der Technik nicht in das Bauwerksmodell integriert. Die Prüfung kann aber durch nachträgliche Validierung der Bauwerksmodelle durchgeführt werden. Dies wird im folgenden Abschnitt beschrieben.

3.8 Validierung von Bauwerksmodellen

Die Validierung von digitalen Bauwerksmodellen (englisch: model checking) ist heute ein wichtiger Prozessschritt in der Planung von Bauwerken. Dabei wird ein Zustand des Modells gegen verschiedene Anforderungen in Form von *Regeln* geprüft. Die aufgestellten *Regeln* prüfen verschiedene Teilaspekte des Modells und werden aus unterschiedlichen Anforderungen abgeleitet. Grundsätzlich lassen sich dabei *Fehlerfreiheit*, *Konsistenz*, *Konformität* und *Entwurfsabsicht* unterscheiden (vgl. Borrmann, König u. a., 2015, Seite 323-325).

Die *Fehlerfreiheit* umfasst den Bereich der Prüfung auf grundsätzliche physikalische Widersprüche. Diese kann einzelne geometrische Körper umfassen (siehe *Solid Modeling* in Abschnitt 2.5), die sich nicht selbst überschneiden dürfen oder durch falsche Orientierung der Volumina bzw. Oberflächen auffallen. Dazu gehört auch die geometrische Überschneidung verschiedener Körper, welche mit einer Kollisionsprüfung (englisch: clash detection) festgestellt werden kann (vgl. Borrmann, König u. a., 2015, Abschnitt 16.2). Darüber hinaus existiert auch das Gegenstück der Kollisionsprüfung: Das Aufspüren ungewollter Lücken zwischen geometrischen Körpern (englisch: defiency detection) (vgl. Kraft, 2016, Abschnitt 2.5.2).

Die *Konsistenz* beinhaltet, dass alle im Modell vorhandenen Daten dieselben Informationen repräsentieren und keine Widersprüche existieren. Ein klassisches Beispiel sind unterschiedliche Sichten auf das Modell in Form von Planschnitten, welche übereinstimmende Lokalisierungen und Maße der Bauteile beinhalten müssen.

Das Prüfen der *Konformität* (englisch: compliance checking) soll sicherstellen, dass technische bzw. gesetzliche Regularien, aber auch projektspezifische Anforderungen erfüllt

[5] http://standards.buildingsmart.org/IFC/RELEASE/IFC4_1/FINAL/HTML/schema/ifcpropertyresource/lexical/ifcpropertydependencyrelationship.htm (abgerufen am 22.10.2019)

sind. Die Grundlage der technischen bzw. gesetzlichen *Konformität* (englisch: code compliance) bilden Normen und Richtlinien, aus denen entsprechende *Regeln* abgeleitet werden können (vgl. Borrmann, König u. a., 2015, Kapitel 20). Ist die technische bzw. gesetzliche *Konformität* sichergestellt, lassen sich projektspezifische Anforderungen ebenfalls sicherstellen. Dazu gehören die Entwurfsabsichten der Architekten und Ingenieure. Diese umfassen beispielsweise ästhetische, bauphysikalische oder finanzielle Randbedingungen (vgl. Niemeijer, 2011, Seite 32). Grundsätzlich besteht bei der Prüfung der *Konformität* die Schwierigkeit in der Übersetzung der schriftlich verfassten Texte oder gedanklichen Überlegungen in algorithmisch prüfbare *Regeln*.

Eastman, J.-m. Lee u. a. (2009) geben eine Übersicht zu dem Prozess der Validierung und den dafür verfügbaren Softwaresystemen. Sie nennen dabei als Probleme die Herausforderung des Entwickelns von Sprachen für das Definieren der *Prüfregeln*, und die Bewertung und Einordnung der Ergebnisse der Validierung durch die Nutzer. Dass die geometrische *Fehlerfreiheit* und *Konsistenz* als grundsätzliche Anforderungen an Bauwerksmodelle erfüllt sein müssen, wird zwar vorausgesetzt, die Erfüllung ist aber keine triviale Aufgabe, wie aktuelle Forschungen zeigen (vgl. Huhnt, 2018).

Die Validierung von Bauwerksmodellen findet aktuell, üblicherweise nachgelagert, zur Modellierung statt und ist in einen iterativen Prozess eingebunden. Dieser beinhaltet die Anpassung des Modells bei negativem Ergebnis der Validierung und ein erneutes Prüfen. Dies unterliegt der Annahme, dass auf diese Weise der Prozess zu einem guten, aber nicht zwangsläufig optimalen Ergebnis des Modells führt. Der Grund dafür liegt darin, dass die *Regeln* nur dazu dienen, den aktuellen Zustand des Modells zu bewerten, was dazu führt, dass gegebenenfalls sinnvolle Lösungen übersehen werden (vgl. Niemeijer, 2011, Abschnitt 3.4.2).

3.9 Modellierungsansatz: Der Mengen-basierte Entwurf

Wie beschrieben wird der Integration der Anforderungen in Bauwerksmodelle bisher wenig Beachtung geschenkt. Dies führt dazu, dass im Bauwesen und anderen Ingenieurdisziplinen heute der Prozess des *Punkt-basierten Entwurfs* (englisch: point-based design) vorherrscht. Dabei wird bei der Bearbeitung einer ingenieurtechnischen Fragestellung eine Lösung in Form einer Modellvariante entwickelt, welche zwar die Randbedingung erfüllt, aber nicht die Randbedingungen selbst vorhält. Auch werden Zusammenhänge, wie Abhängigkeiten der *Parameter* untereinander, selten digital integriert. Im Gegensatz dazu steht das Prinzip des *Mengen-basierten Entwurfs* (englisch: set-based design), das bei Toyota (vgl.

Sobek u. a., 1999) entwickelt und in Prozessen umgesetzt wurde. Dieses steht in keinem Zusammenhang zu Mengen aus der bauingenieurtechnischen Mengenermittlung von Baustoffen und Bauteilen. Es beschreibt stattdessen die Idee, dass bei ingenieurtechnischen Fragestellungen Varianten möglichst spät ausgeschlossen werden (vgl. Ghosh u. a., 2014). Dabei ist es möglich, dass erst durch eine grobe Beschreibung der Randbedingungen und anschließend durch weitere Anforderungen ein verkleinerter Raum der Möglichkeiten entsteht. Im Idealfall ergibt dies zum Ende eine oder mehrere gute Lösungsvarianten, die umgesetzt werden können. Es gilt die Annahme, dass ein Effizienzvorteil darin liegt, dass im Gegensatz zum *Punkt-basierten Entwurf*, bei dem gegebenenfalls viele einzelne Lösungen entwickelt, geprüft und gegebenenfalls verworfen werden müssen, nur ein *Mengen-basierter Entwurf* vorgehalten und weiterentwickelt werden muss (vgl. Liker u. a., 1996, Seite 167).

Ein häufiger Vorschlag ist es, den *Mengen-basierter Entwurf* dazu zu nutzen, das gemeinsame nebenläufige Arbeiten an digitalen Modellen zu verbessern. Sobek u. a. (1999) definieren dafür drei Grundprinzipien, welche sie aus der Anwendung des *Mengen-basierten Entwurfs* bei Toyota ableiten:

Abbilden des Entwurfs-Raums: Es müssen Bereiche festgelegt werden, die aufgrund der Randbedingungen möglich sind. Einschränkungen der Bereiche sollten gegenüber anderen Beteiligten sinnvoll begründet werden.

Integration durch Verschneidung: Es muss geprüft werden, ob die Verschneidung bzw. die Bildung der Schnittmenge der verschienden Varianten möglich ist, ohne einen Widerspruch zu erzeugen. Dabei sollte der Entwurf möglichst so entwickelt werden, dass er so weit wie möglich unabhängig von Entscheidungen anderer Beteiligter ist.

Vor Festlegungen, die Machbarkeit sicherstellen: Bei der fortschreitenden Verfeinerung des Entwurfs sind die Mengen langsam zu reduzieren. Es sollte vermieden werden, Mengen zu vergrößern, wenn sie einmal festgelegt wurden.

Es existieren des Weiteren Vorschläge, *Mengen-basierte Entwürfe* für Prozesse bei großen Bauprojekten wie zum Beispiel Flughäfen (vgl. Gil u. a., 2008) umzusetzen.

Die Idee des *Mengen-basierten Entwurfs* wurde in verschiedenen Arbeiten auf digitale Modelle übertragen. Von Yannou u. a. (2013) wurde ein Modell entwickelt, in dem die *Parameter* einer elektrischen Stichsäge entsprechend der verschiedenen Randbedingungen aus den Nutzungsszenarien unterschiedlicher Kundengruppen gebildet werden. Die Anforderungen werden in Form von Intervallen angegeben und durch *Intervallarithmetik*

zur Berechnung der möglichen Werte der *Parameter* herangezogen. Panchal u. a. (2007) schlagen vor, *Parameter* in einem Projekt als Intervalle zu definieren, welche die Menge an möglichen Werten repräsentieren. Anschließend können unterschiedliche Personen das Modell bearbeiten, was sich in einer Verkleinerung der Intervalle äußert. Dadurch ist es beim Zusammenführen der unterschiedlichen Arbeitsstände möglich zu prüfen, ob nach Bildung der Schnittmenge der Werte der *Variablen* noch Varianten möglich sind, oder ob Widersprüche existieren, da sich Werte ausschließen.

3.10 Vorschlag eines Entwicklungsprozesses bei der Entwurfsfindung

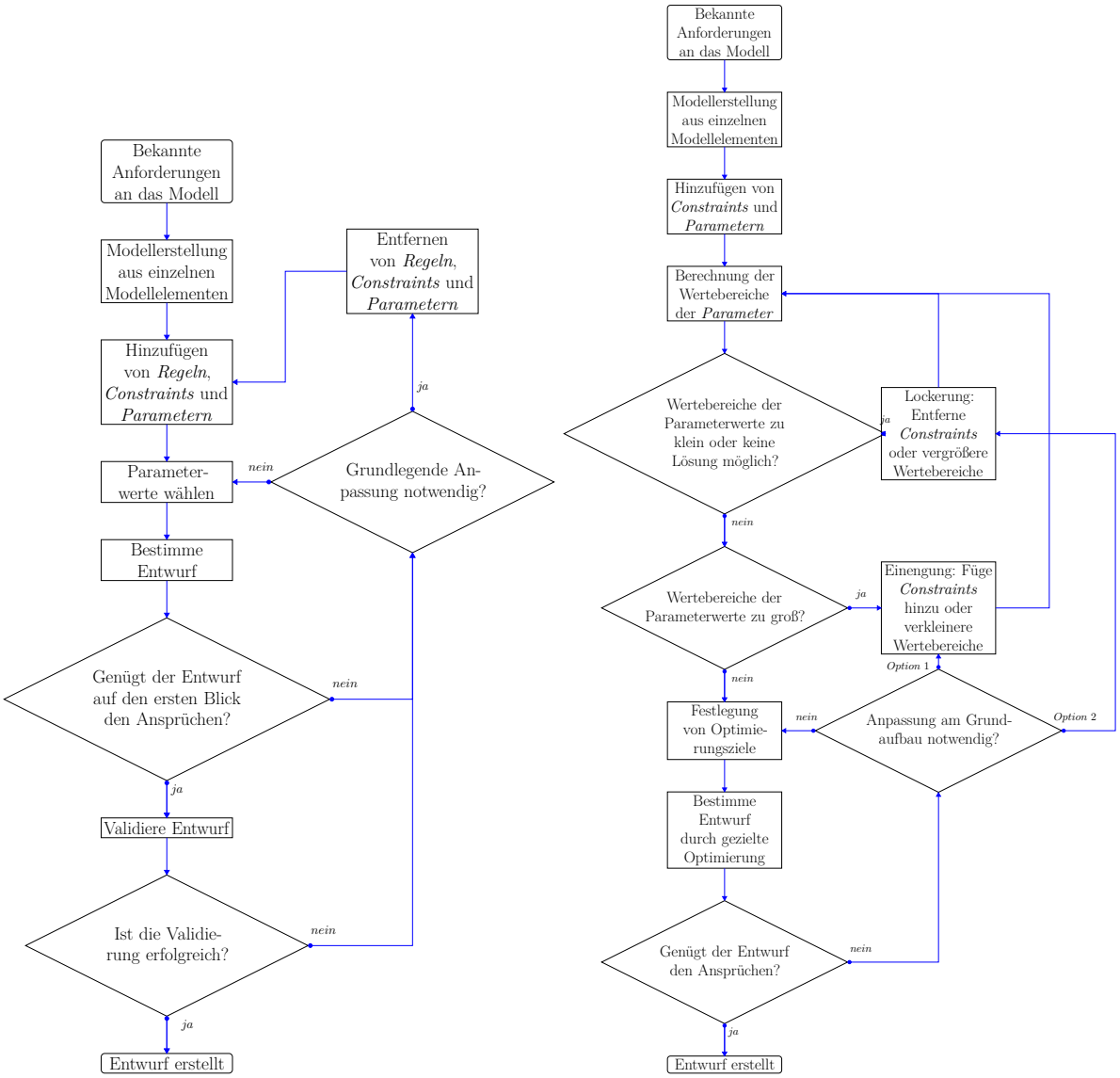
Der für diese Arbeit vorgeschlagene Entwicklungsprozess in der Entwurfsfindung mit einem Bauwerksmodell unterscheidet sich von dem bisher angewandten Prozess. Dabei steht im Vordergrund, das *Wissen* in das Modell zu integrieren und im Sinne des *Mengen-basierten Entwurfs* Wertebereiche der *Parameter* festzulegen. Anschließend soll möglichst innerhalb der gesetzten Randbedingungen ein Entwurf entwickelt werden. Dabei geschieht die Entwurfsfindung so weit wie möglich innerhalb der gesetzten Grenzen, beispielsweise durch Optimierung und kleinere Anpassungen. In Abbildung 3.5 sind der aktuell übliche und der vorgeschlagene Entwicklungsprozess zum Vergleich nebeneinander dargestellt.

3.11 Diskussion des Wissens in Bauwerksmodellen

Wie gezeigt wurde, kann *Wissen* in Bauwerksmodellen verschiedene Bereiche des Modells betreffen. Das vermehrte Modellieren von Zusammenhängen und Anforderungen würde es ermöglichen, im Modell selbst die Validität sicher zu stellen. Dies beträfe den Erhalt der Validität der Geometrie konsistent zu den Anforderungen und die Konsistenz von Semantik und Geometrie.

Für die Beherrschung der Komplexität werden Klassifikation und Strukturierung genutzt. Diese sind eindeutig mit dem Paradigma der *Objektorientierung* verbunden. Der Umfang solcher Modelle erfordert es, dass eine Zerlegung in sinnvolle Einheiten erfolgt. Der Ansatz der Zerlegung soll in dieser Arbeit bei der Beschreibung eines Bauwerksmodells und des integrierten *Wissens* besondere Berücksichtigung finden.

Der Erhalt der Modellvalidität ist bedeutend schwieriger umzusetzen, da er erfordert, dass in verschiedenen Bereichen die Konsistenz erhalten bleiben muss. Soll das *Wissen* beispielsweise über den bündigen Anschluss zweier Wände untereinander im Modell



(a) Bisher vorherrschender Entwicklungsprozess **(b)** Vorgeschlagener Entwicklungsprozess

Abbildung 3.5: Entwicklungsprozesse in der Entwurfsfindung mit einem Bauwerksmodell

hinterlegt werden, ist zwischen den Wänden, aber auch zwischen den geometrischen Repräsentationen der Wände, diese Relation zu definieren. Das *Wissen* über diese Relation kann entweder zur Prüfung genutzt werden oder aber zur Suche nach Lösungen nach einem Zustand des Modells, in dem der bündige Anschluss sichergestellt ist.

Die Integration von *Wissen* im Modell kann speziell mit der Idee des *Mengen-basierten Entwurfs* sinnvoll kombiniert werden. Dies kann so umgesetzt werden, dass das *Wissen* über die Randbedingungen und die Beziehungen der Objekte genutzt wird,

die verbliebenen *Mengen* des Entwurfs, in Form der Möglichkeiten bzw. Varianten des Modells unter diesen Zwängen, zu bestimmen.

Im folgenden Kapitel soll untersucht werden, welche Möglichkeiten existieren, Geometrie zu modellieren und dabei Zusammenhänge zwischen den einzelnen Formen bzw. Objekten zu berücksichtigen. Darüber hinaus soll geprüft werden, welche Lösungsverfahren, mit integriertem *Wissen* über geometrische Beziehungen, zur Verfügung stehen, um sinnvolle Entwürfe des Modells abzuleiten.

KAPITEL 4

Stand der Technik, der Entwicklung und der Forschung bei Geometriemodellen

In diesem Kapitel werden die Grundsätze des Modellierens von Geometrie erläutert. Dazu wird der Stand der Technik und der Forschung im Bereich der Modelle des *Computer-Aided Designs* aufgeführt und verschiedenen Ansätze gegenübergestellt. Es wird dabei speziell auf das *parametrische Modellieren* und *Constraint-basierte Modellieren* eingegangen. Der aktuelle Stand der Forschung zur Ermittlung gültiger Parameterbereiche und der Suche nach Entwürfen, die der Idee des Menschen nahekommen, wird erläutert. Anschließend werden die Ergebnisse der Betrachtung des aktuellen Entwicklungsstands zu diesem Thema diskutiert und daraus die weiteren Untersuchungen für diese Arbeit abgeleitet.

4.1 Einleitung

Geometriemodelle bilden die Grundlage für Produktmodelle, wie sie digitale Bauwerksmodelle darstellen. Dabei ergeben sich die Anforderungen und Möglichkeiten von Geometriemodellen aus dem Zweck der Repräsentation, der Art der geometrischen Repräsentation einer Entität des Modells und der nachträglichen Änderungen. Es wird angestrebt, dass die Repräsentation in sich konsistent ist und allen Anforderungen genügt, die an die Geometrie der Entitäten gestellt werden. Die Repräsentation sollte nach dem Prinzip des *Solid Modelings* (siehe Abschnitt 2.5) aus validen geometrischen Einheiten bestehen. Wie diese angepasst, wie Abhängigkeiten zwischen den Einheiten definiert und wie daraus Erkenntnisse für den Entwurf gewonnen werden können, soll in den folgenden Abschnitten untersucht werden.

4.2 Computer-Aided Design

Unter *Computer-Aided Design* (CAD) versteht man das „rechnerunterstützte[s] […] Entwerfen und Konstruieren“ (Vajna u. a., 2009, Seite 11). Es dient dazu, die Geometrie einer zu beschreibenden Entität im Rechner in Form eines CAD-Modells abzubilden. Dieses CAD-Modell dient als Basis für weitere Auswertungen, wie zum Beispiel technische Zeichnungen, Simulationen oder Fertigungsmaschinen.

Im Laufe der Zeit fand die Hinwendung von anfangs zweidimensionalen zu dreidimensionalen Konstruktionsbeschreibungen in CAD-Systemen statt. Aktuelle 3D-CAD-Softwaresysteme haben das Spektrum so erweitert, dass ihre Grundlage 3D-Produktmodelle darstellen (vgl. VDI 2209, Seite 5). Damit ist ein digitales Bauwerksmodell ein komplexes CAD-Modell.

In VDI 2209 (Abschnitt 1.2) werden dreidimensionale CAD-Modelle entsprechend ihrer zugrunde liegenden Technik klassifiziert. In Abbildung 4.1 ist dazu eine Übersicht dargestellt. Diese Klassifizierung entspricht den historischen Stufen von Modellen, beginnend mit dem *CAD (konventionell)*, bestehend aus unabhängig voneinander platzierten Geometrieelementen, über *CAD (parametrisch)*, bei dem das Gesamtmodell über *Parameter* verändert werden kann (siehe auch Abschnitt 4.3.1). Der Stand der Technik ist dabei *Feature-basiertes CAD*, in dem das Erfassen der Funktion, der Identifikation und Organisation der einzelnen Bauteile eine große Rolle spielt. Die höchste Stufe, das *wissensbasierte CAD*, kann nur in Ansätzen als Stand der Technik angesehen werden, ist aber das erstrebenswerte Ziel (siehe Abschnitt 1.2). *Wissensbasiertes CAD* setzt voraus, dass einzelne Modellelemente *selbstorganisiert* sind und auf Veränderungen reagieren können. In der Literatur und dem wissenschaftlichen Diskurs werden solche Modellelemente auch als „intelligent“ oder „smart“ bezeichnet.

4.3 Ansätze zur Geometriemodellierung

Bei der Erstellung ingenieurtechnischer Modelle kommt es durch den üblicherweise iterativen Prozess zwangsläufig zu nachträglichen Änderungen bereits modellierter Objekte. Diese Anpassung erfolgt entweder durch vollständige Neukonstruktion, direktes Manipulieren vorhandener Geometrie oder durch indirekte Steuerung über die Änderung von Werten (vgl. Roller, 2013, Seite 10). Die vollständige Neukonstruktion ist dabei das zeitaufwändigste Verfahren, da bisherige Modellaufbauten verworfen werden. Es steht der effizienten Nutzung des Rechners bei der Modellierung entgegen.

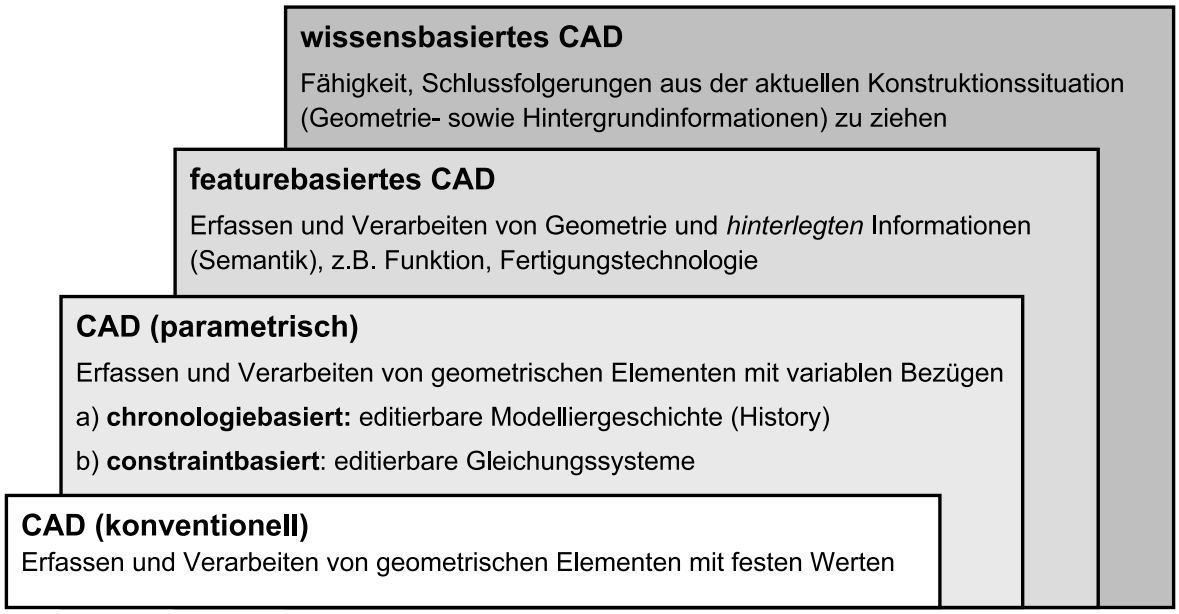


Abbildung 4.1: Klassifizierung der Stufen von 3D-CAD-Software (entnommen aus VDI 2209, Bild 5)

Die Art der Manipulation, welche bei einem Modell möglich ist, entscheidet über die Flexibilität bei nachträglichen Änderungen. Dabei existieren zwei grundlegend unterschiedliche Ansätze: Das *variationale Modellieren* und das *prozedurale Modellieren*.

Beim *variationalen Modellieren* erfolgt die Manipulation durch einen direkten Eingriff in den aktuellen Zustand der geometrischen Objekte. Die Manipulation geschieht dabei beispielsweise durch das Verschieben, das Rotieren oder durch Größenanpassungen der Formen. *Variationale Modelle* lassen sich üblicherweise den *deklarativen Schemata* zuordnen (siehe Definition 2.1.4). Ausprägungen des *variationalen Modellierens* sind *direktes Modellieren* und *Constraint-basiertes Modellieren*.

Der indirekten Steuerung der Geometrie liegen *prozedurale Modelle* zugrunde, auch *sequentielle Modelle* genannt. Diese finden sich in der Literatur auch unter dem Begriff *historienbasierte Modelle* (englisch: history-based models) bzw. *Chronologie-basierte Modelle* (englisch: chronology-based models). Es handelt sich hierbei um Modelle mit einem *imperativen Schema* (siehe Definition 2.1.3), bei denen die Beschreibung des Modells als *prozedural* zu verarbeitender *Konstruktionsplan* erfolgt.

Je nach verwendetem Ansatz kann die Manipulation des Modells anhand der Änderung von Parameterwerten durchgeführt werden. Dies wird als *parametrischer* Ansatz bezeichnet.

In Tabelle 4.1 ist eine Übersicht der Kombination der Ansätze für das Modellieren von Geometrie und deren konkrete Ausprägungen gegeben. Es sei darauf hingewiesen, dass

nach aktuellem Wissen des Autors kein sinnvolles *nichtparametrisches Modell* existiert, welches gleichzeitig *prozedural* ist. Der Grund dafür ist, dass der Verzicht auf *Parameter* eine zu starke Einschränkung der Ausdrucksmöglichkeiten solcher Modelle darstellen würde.

Tabelle 4.1: Konkrete Ausprägungen des Modellierens von Geometrie anhand der Manipulationsform und der vorhandenen Parametrik

	prozedural	**variational**
nicht-parametrisch	(nicht sinnvoll)	*Direktes Modellieren* *Constraint-basiertes Modellieren* (ohne *dimensionale Constraints*)
parametrisch	*(parametrisch) prozedurales Modellieren*	*Constraint-basiertes Modellieren* (mit *dimensionalen Constraints*)

In den folgenden Abschnitten wird zuerst allgemein auf die Grundsätze des *parametrischen Modellierens* und des *nichtparametrischen Modellierens* eingegangen. Anschließend wird das *parametrisch prozedurale Modellieren* – als einzig sinnvolle Form des *prozeduralen Modellierens* – genauer beschrieben. Das *Constraint-basierte Modellieren* wird umfassend erläutert, da es den in dieser Arbeit verfolgten Ansatz darstellt. Diese Modellierungsformen werden anschließend verglichen und es wird diskutiert, in wie weit sie sich für die Repräsentation von *Wissen* in Bauwerksmodellen eignen.

4.3.1 Parametrisches Modellieren

Das *parametrische Modellieren* (auch *parametrischer Entwurf*; englisch: parametric modeling/design) stellt den Grundsatz dar, die Geometrie durch das Ändern von *Variablen*, den so genannten *Parametern*, anzupassen (vgl. Eastman, Teicholz u. a., 2011, Seite 17-18). Damit sind *parametrische Modelle* immer *implizit* (siehe Definition 2.1.2). Den Kern des *parametrischen Modellierens* bildet dabei die Beschreibung des Zusammenhangs zwischen den wertetragenden *Parametern* und den *Geometrie-Objekten*.

Parameter lassen sich dabei je nach Kontext klassifizieren. Im Kontext der Relevanz für die Geometrie eines Modells schlagen Vajna u. a. (2009) folgende Unterscheidung vor:

Geometrische Parameter besitzen numerische Werte, sind mit der Geometrie verknüpft und definieren die Gestalt des Produkts.

Nicht geometrische Parameter haben keinen direkten Einfluss auf die Geometrie. Dazu gehören:

Topologische Parameter steuern zu nutzende Konstruktionselemente, wie zum Beispiel unterschiedliche Rohrtypen.

Physikalische Parameter enthalten Werkstoffeigenschaften.

Prozess- oder Technologieparameter legen Werte für die Herstellungsverfahren fest.

Wird unterschieden, welche *Parameter* zur Veränderung des Modells und welche ausschließlich zur Aufnahme der Ergebnisse von Berechnungen dienen, so ergeben sich:

Eingabeparameter (auch Führungsparameter, englisch: input/driving parameter) sind *Parameter* deren Wert durch den Nutzer des Systems direkt geändert werden kann. Anschließend wird auf Basis der neuen Werte ein neuer Zustand berechnet.

Ausgabeparameter (englisch: output/reporting parameter) können nicht direkt verändert werden, sondern ergeben sich indirekt durch Berechnungen aus anderen Parametern.

Die Änderung eines *Parameters* oder mehrerer *Parameter* führt zu einem Zustandswechsel. G. Lee u. a. (2006) beschreiben dabei eine Parameteränderung als einen *Stimulus*. Auf diesen folgt eine Reaktion des Modells, welche das Verhalten des Modells darstellt. Tritt ein invalider Zustand auf, da die Parameteränderung zu einem nicht berechenbaren Zustand führt, ist abzubrechen und der Zustandswechsel scheitert. Wird ein neuer valider Zustand erreicht, wird dieser durch ein neues *ausgewertetes Modell* repräsentiert.

Nachträgliche Änderungen eines Modells aus *Geometrie-Objekten* sind auf Basis eines *parametrischen Modells* umsetzbar. Das bedeutet, dass mit *Parametern* nicht nur einzelne *Geometrie-Objekte* in Form von *Solids* (siehe Abschnitt 2.5), sondern auch ein System von *Geometrie-Objekten* parametrisiert werden können. Es sind also auch Abhängigkeiten der *Geometrie-Objekte* untereinander möglich. In Softwaresystemen zur *parametrischen Modellierung* werden meist *parametrische Familien* unterstützt (vgl. Shah, 2001). Diese ermöglichen es, eine im Vorfeld digital modellierte *Familie* (siehe Abschnitt 3.5.2) mehrfach und projektübergreifend zu verwenden. Die *Parameter* dienen zur Steuerung der individuellen Ausprägung der *Instanz* bzw. des *Typs* im Projekt.

Für das *parametrische Modellieren* existieren zwei unterschiedliche Modellierungsschemen, welche sich analog zu den Modellklassifikationen aus Abschnitt 2.1 ergeben. Im Folgenden wird auf das *nichtparametrische Modellieren* und die zwei Modellierungsschemen des *parametrischen Modellierens* genauer eingegangen.

Es sei noch darauf hingewiesen, dass in dieser Arbeit folgende Bereiche des *parametrischen Modellierens* von Geometrie nicht genauer betrachtet werden:

Dynamische Modelle: Da in dieser Arbeit ausschließlich Bauwerksmodelle beschrieben werden, bei denen *Parameter* keinen zeitlichen Änderungen unterworfen sind, werden *parametrische Modelle* unter der Verwendung der Zeitdimension nicht betrachtet. Das bedeutet, dass keine kontinuierliche Zustandswechsel vorgesehen sind. *Parametrische Modelle* mit Berücksichtigung solcher dynamischen Veränderungen, existieren beispielsweise im Maschinenbau für Kurbelwellen, Zahnräder, usw.

Generative Gestaltung: Entgegen der regelmäßigen synonymen Verwendung von *parametrischer Modellierung* und dem *generativen Entwurf* (englisch: generative design), stellt der *generative Entwurf* eine weitere spezielle Form des *parametrischen Modellierens* dar. Dabei wird das *parametrische Modell* genutzt, einen geometrischen Entwurf durch einen Algorithmus umzusetzen, wobei durch *Eingabeparameter* Varianten generiert werden können. Es spielt hauptsächlich die Form und weniger der funktionale Zweck zur Verwendung in einem Produktmodell eine Rolle.

4.3.2 Nichtparametrisches Modellieren

Werden keine *Parameter* verwendet, so wird dies als *nichtparametrisches Modellieren* bzw. *historienfreies Modellieren* (englisch: history-free modeling) bezeichnet.

Die einzig mögliche Form des *nichtparametrischen Modellierens* ist das *direkte Modellieren* (englisch: direct modeling). Dieses beschreibt das Vorgehen, eine explizite Repräsentation von *Geometrie-Objekten* direkt zu verändern (vgl. Ault u. a., 2016) (vgl. VDI 2209, Seite 158). Üblicherweise basieren diese Modelle auf einem B-rep (siehe Abschnitt 2.5). Dies hat den Vorteil, dass Geometrien „intuitiv“ durch Greifen und Ziehen einzelner *Geometrie-Objekte* manipuliert werden können. Die Änderungsoperatoren, wie zum Beispiel Teilextrusionen von Flächen oder das Verschieben von Eckpunkten, werden direkt umgesetzt. Es existiert dabei weder eine vorhaltende Speicherung der Änderungsoperatoren, noch *Eingabeparameter* zur Steuerung der Änderungen.

Der Vorteil des *direkten Modellierens* liegt darin, dass es eine große Freiheit des Modellierens und Änderns ermöglicht, wodurch sich unvorhergesehene Änderungen schneller realisieren lassen (vgl. Abulawi, 2012, Tabelle 4.3). Dem gegenüber steht die Einschränkung, dass der Weg, wie diese Form entwickelt wurde, nicht vorgehalten wird. Auf Grund des Fehlens eines solchen *Konstruktionswissens*, ist die Erstellung von Varianten nur manuell durch direkte Anpassung möglich.

4.3.3 Parametrisch prozedurales Modellieren

Dem *parametrisch prozeduralen Modellieren* liegt die Idee zugrunde, dass die Beschreibung des Modells als *prozedural* zu verarbeitender *Konstruktionsplan* erfolgen soll.

Definition 4.3.1 (Konstruktionsplan)**.** Ein *Konstruktionsplan* (auch *Konstruktionsgeschichte*), stellt eine geordnete Abfolge einzelner *Konstruktionsoperatoren* dar (vgl. ISO 10303-108, Abschnitt 4.2.1). *Konstruktionsoperatoren* sind *Regeln* (siehe Definition 2.2.1), die auf Basis von Eingangsdaten in Form von *Parametern* und/oder *Geometrie-Objekten*, neue *Geometrie-Objekte* erzeugen bzw. bereits erzeugte verändern. Das Ergebnis eines *Konstruktionsplans* ist immer eindeutig.

Eine alternative Form der Repräsentation und Darstellung eines *prozeduralen Modells* kann auch als *Konstruktionsgraph* geschehen.

Definition 4.3.2 (Konstruktionsgraph)**.** Ein *Konstruktionsgraph* ist ein gerichteter, zyklenfreier, bipartiter Graph der disjunkten Teilmengen A und B. Die Knoten der Teilmengen A repräsentieren *Geometrie-Objekte* und *Parameter*. Die Knoten der Teilmengen B repräsentieren die *Konstruktionsoperatoren*.

Auf Grund der vorgegeben Richtung und der Zyklenfreiheit kann aus einem *Konstruktionsgraphen* ein *Konstruktionsplan* abgeleitet werden. Das Ergebnis eines Durchlaufs des *Konstruktionsplans* ist eine Lösung in Form eines *ausgewerteten Modells*. Durch die Änderung von Parameterwerten kommt es zu einer erneuten Erzeugung oder einer Aktualisierung des *ausgewerteten Modells*, mindestens ab dem ersten Auftauchen des *Parameters* in einem *Konstruktionsoperator* des *Konstruktionsplans*. Eine häufig verwendete Sonderform des *Konstruktionsgraphen* stellt der *Konstruktionsbaum* (vgl. Historienbaum Abulawi, 2012, Seite 73) dar. *Konstruktionsbäume* schränken die Möglichkeiten der Modellierung ein, da die Äste des Baums nicht mehr zusammengeführt werden. *Konstruktionsbäume* vermeiden Probleme bzw. Widersprüche, die an den zusammenführenden Knoten eines *Konstruktionsgraphen* bei der *prozeduralen* Erzeugung eines *ausgewerteten Modells* auftreten können.

Da *prozedurale Modelle* erzeugende Modelle sind, können bei bestimmen Parameteränderungen nicht eindeutige Abbildungen auftreten. Wird bei einem Durchlauf festgestellt, dass die Bedingungen eines Operators nicht erfüllt werden können, kommt es zu einem Abbruch, und es kann kein valider Zustand berechnet werden.

Durch das erneute Durchlaufen des *Konstruktionsplans* bei Parameteränderungen ist es notwendig, den vorherigen Zustand auf den neuen Zustand abzubilden, wenn sichergestellt werden soll, dass Objekte über alle Zustände hinweg persistent identifizierbar sein sollen.

Das Konzept der *persistenten Objekte* (englisch: persistent objects) sorgt dafür, dass nur die Geometrie eines Objekts neu erstellt wird, nicht aber das Objekt selbst (vgl. Hoffmann und Rossignac, 1996). Dies ist mit Problemen verbunden, wenn im *Konstruktionsplan* Operatoren eingebaut sind, welche nur bei bestimmten Parameterwerten durchführbar sind. Beispielsweise kann der Radius einer Kugel nicht für den nächsten Operationsschritt verwendet werden, wenn durch eine Parameteränderung aus einer Kugel ein Tetraeder wird.

Analog zur *prozeduralen Modellierung* sind die *konstruktiven Modelle* für *Solids* zu sehen (siehe Abschnitt 2.5). Offenkundig *prozedurale Modelle* liegen in den visuellen Programmiersprachen, wie *Grasshopper*[1] oder *Dynamo*[2], zur Generierung von Geometrie vor. Dabei lässt sich der Graph der Programmiersprache als eine verallgemeinerte Form eines *Konstruktionsgraphen* auffassen.

4.3.4 Constraint-basiertes Modellieren von Geometrie

Die Modelle für das *Constraint-basierte Modellieren* werden auch als *Constraint-basierte, variationale Modelle* (englisch: constraint-based variational models oder flexible constraint models) bezeichnet. Da die *Geometrie-Objekte* im Gegensatz zu *prozeduralen Modellen* explizit vorhanden sind, werden keine Erzeugungsmethoden wie *Konstruktionsoperatoren* vorgehalten.

Die Grundlage bilden hierbei die *Geometrie-Constraints*.

Definition 4.3.3 (Geometrie-Constraint)**.** *Geometrie-Constraints* sind eine spezielle Art *Constraints* (siehe Definition 2.2.2), welche geometrische Relationen und Einschränkungen beschreiben, die mit *Geometrie-Objekten* in Zusammenhang stehen. Es existieren verschiedene Typen von *Geometrie-Constraints* (vgl. Bettig und Shah, 2001; Obergrießer, 2017, Abschnitt 4.3.1; Pratt, 1998):

Dimensionale Constraints: Geometrische Zusammenhänge, die von einer Größe abhängig sind. Dazu gehören unter anderem Abstand, Winkel und Radius.

Logische Constraints: Logische geometrische Zusammenhänge. Dazu gehören Koinzidenz, Kollinearität und Koplanarität.

Orientierungs-Constraints: Geometrische Zusammenhänge der Orientierung. Dazu gehören Achs- und Geradeparallelität, Orthogonalität, Punkt- oder Achsensymmetrie und Tangentialität.

[1] https://www.grasshopper3d.com/ (abgerufen am 29.10.2019)
[2] https://dynamobim.org/ (abgerufen am 29.10.2019)

Das Modell für das *Constraint-basierte Modellieren* von Geometrie wird als *geometrisches Constraint-System* bezeichnet. Angelegt an die Definitionen von Jermann u. a. (2006) und Hoffmann und Joan-Arinyo (2005), lässt sich dies folgendermaßen definieren:

Definition 4.3.4 (Geometrisches Constraint-System)**.** Ein *geometrisches Constraint-System* (GCS) setzt sich aus den folgenden Teilen zusammen:

- Eine Menge von *Geometrie-Objekten* $O = \{o_1, ..., o_n\}$. Diese sind eingebettet in den euklidischen Raum R^d der Dimension d.
- Eine Menge von *Parametern* $V = \{v_1, ..., v_n\}$. Diese beschreiben geometrische Größen.
- Eine Menge von *Geometrie-Constraints* $C = \{C_1, ..., C_m\}$, die *Geometrie-Objekte* aus O in Zusammenhang setzen und einschränken.

Ein wichtiges Prinzip eines GCS ist: Wird jede *Geometrie-Constraint* als eine Menge von *algebraischen Constraints* (siehe Definition 2.2.3) ausgedrückt, so lässt sich das GCS als NCS (siehe Definition 2.3.3) betrachten. Lässt sich dabei eine *Geometrie-Constraint* mit genau einer Gleichung beschreiben, handelt es sich um eine *Einzel-Constraint* (englisch: single constraint). Ist mehr als eine Gleichung notwendig, ist es eine *Kompositions-Constraint* (englisch: compound constraint) (vgl. Brüderlin u. a., 1998, Seite 89-90).

Ein valider Zustand S eines GCS ist dann möglich, wenn die *Geometrie-Objekte* O so geformt und platziert werden können, dass alle *Constraints* C gleichzeitig erfüllt sind. Die in den meisten Fällen nichttriviale Suche nach validen Zuständen wird als *geometrisches Constraint-Satisfaction-Problem* (GCSP) bezeichnet und stellt eine Abwandlung des allgemeinen CSPs (Abschnitt 2.3) dar.

Zur Repräsentation und Darstellung der Zusammenhänge eines GCS lässt sich dieses als *Constraint-Graph der Geometrie* modellieren.

Definition 4.3.5 (Constraint-Graph der Geometrie)**.** Ein GCSP lässt sich in Form eines *Constraint-Graphen der Geometrie* beschreiben. In diesem ungerichteten Graphen sind die Knoten *Geometrie-Objekte* oder *Parameter* und die Kanten *Constraints*.

In Abbildung 4.2 ist ein Beispiel für einen *Constraint-Graph der Geometrie* dargestellt.

Ein erster Schritt zur Untersuchung, ob einzelne geometrische Objekte durch *Constraints* fixiert werden, lässt sich auf der Basis der aus der Mechanik bekannten *Freiheitsgrade* durchführen.

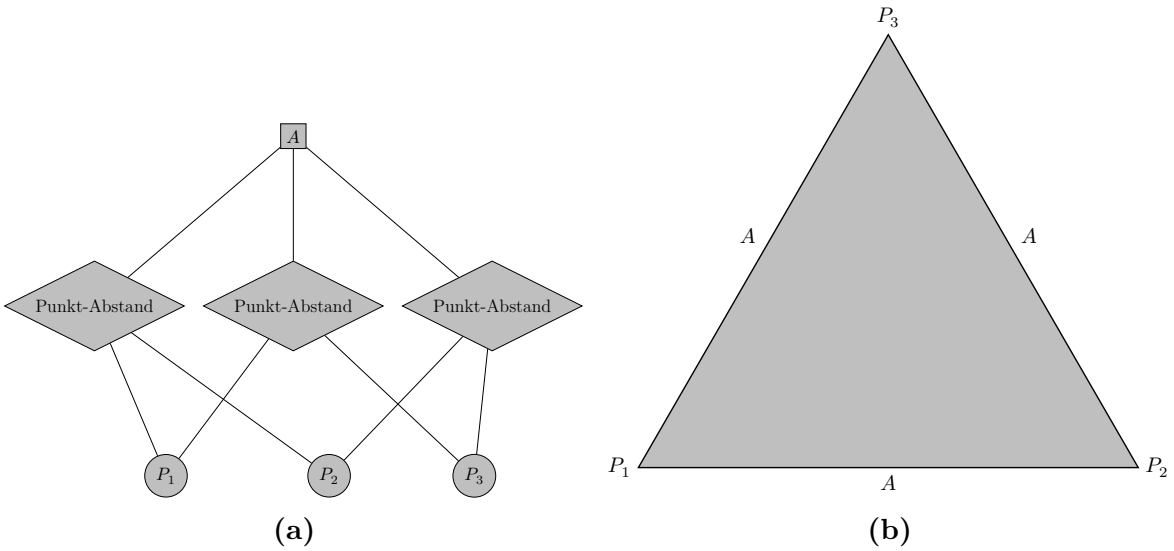


Abbildung 4.2: Beispiel eines *geometrischen Constraint-Graphen* (a) für die Beschreibung der Gleichseitigkeit mit der Seitenlänge A für das Dreieck (b). Die Bestandteile des *Constraint-Graphens* sind *Constraints* (Rauten), *Parameter* (Rechteck) und Punkte (Kreis).

Definition 4.3.6 (Freiheitsgrad)**.** Ein *Freiheitsgrad* (englisch: degree of freedom; Abkz. DOF) stellt die Möglichkeit einer eindimensionalen Translation oder Rotation eines Starrkörpers dar.

Dabei ist die Anzahl möglicher *Freiheitsgrade*, die ein Starrkörper in Abhängigkeit von der Dimension des euklidischen Raum R^d besitzt:

R^1 : 1 *Freiheitsgrad*, bestehend aus einer Verschiebung.

R^2 : 3 *Freiheitsgrade*, bestehend aus zwei Verschiebungen (XY) und einer Rotation.

R^3 : 6 *Freiheitsgrade*, bestehend aus drei Verschiebungen (XYZ) und drei Rotationen.

Handelt es sich nicht um Starrkörper, kann die Anzahl variieren. Für einen Punkt entspricht die Anzahl der *Freiheitsgrade* der Anzahl der Raumdimensionen.

Die *Constraints* lassen sich über die *Valenz* mit den *Freiheitsgraden* in Zusammenhang bringen.

Definition 4.3.7 (Valenz)**.** Die *Valenz* ist die Anzahl der *Freiheitsgrade*, die durch eine *Constraint* gebunden werden können (Brüderlin u. a., 1998, Seite 132).

Es ist zu beachten, dass die *Valenz* einiger *Constraints*, je nach numerischem Wert der beteiligten *Parameter*, unterschiedlich ausfallen kann. Als Beispiel nennt Döring (2011,

Seite 17) die *Constraint* für den Abstand A zweier Punkte mit der *Valenz* 1, solange der Abstand größer 0 ist. Fallen die beiden Punkte zusammen, beträgt der Abstand genau 0, so ist die *Valenz* 2. *Valenzen* lassen sich nur in speziellen Fällen gegen *Freiheitsgrade* aufrechnen, da unterschiedliche *Constraints* direkt oder indirekt dieselben *Freiheitsgrade* binden können. Schon bei wenig komplexen Modellen fällt es Menschen schwer, dies an der Struktur des Systems zu erkennen. Das algorithmische Lösen eines GCSP lässt sich in den wenigsten Fällen trivial bestimmen. Es existieren verschiedene Ansätze, die in Abschnitt 4.4 genauer beschrieben werden.

Es ist zu beachten, dass Konzepte existieren, welche aus der Reihenfolge der *Constraints* bei der Nutzereingabe direkt einen *Konstruktionsplan* ableiten. Das bedeutet, dass ein scheinbar *Constraint-basiertes Modell* eigentlich ein *prozedurales* ist (vgl. Rigid Constraint Satisfaction in Shah und Mäntylä, 1995, Seite 80-82).

4.3.5 Vergleich der Modellierungsformen

Betrachtet man die verschiedenen Modellierungsformen im Hinblick auf die Möglichkeit, ingenieurtechnisches *Wissen* zu repräsentieren – dies umfasst eine nachvollziehbare Strukturierung, Flexibilität bei nachträglichen Änderungen und die Ausdrucksstärke bei der Beschreibung der Geometrie – so ergibt sich ein unterschiedliches Bild.

Das *direkte Modellieren* als einzige Form des *nichtparametrischen Modellierens* ist zwar sehr flexibel bei nachträglichen Änderungen, verzichtet aber auf Strukturierung und beinhaltet kein weiteres *Wissen* über die Konstruktion oder über die zugrunde liegenden Überlegungen.

Prozedurale Modelle hingegen besitzen den Vorteil, dass sie durch die starke Strukturierung mit einem *Konstruktionsplan* die funktionale Zuordnung sichtbar abbilden. Darüber hinaus lassen sich Konstruktionsvarianten, im Rahmen des gewählten *Konstruktionsplans*, schnell und einfach umsetzen (vgl. Abulawi, 2012, Tabelle 4.3). Hingegen ist es notwendig, ein solches Modell von Anfang an sorgfältig zu planen, da eine nachträgliche Anpassung des *Konstruktionsplans* aufwändig sein kann. Abulawi (vgl. 2012, Tabelle 4.3) nennt auch die starke „Eigenkomplexität“ als Faktor, der zu schwer nachvollziehbaren Resultaten führen kann.

Die größtmögliche Flexibilität bietet das *Constraint-basierte Modellieren*, in welchem sich unvorhergesehene Änderungen schneller realisieren lassen (vgl. Abulawi, 2012, Tabelle 4.3). Die verwendeten *Constraints* und Parameterwerte stellen dabei das *Entwurfswissen* dar (siehe Definition 3.1.2), welches ausgewertet werden kann. Der Nachteil ist dabei, dass es bei komplexen Modellen zu einer ausgeprägten *Unterbestimmtheit* oder zu einer Vielzahl an Konflikten zwischen *Constraints* kommen kann, was von Menschen nur mit

Hilfe nachvollziehbar ist. Es ist dafür Sorge zu tragen, dass in so einem Fall der Mensch bei der Modellierung und Analyse unterstützt wird (vgl. Shah, 2001).

Stellt man das *Constraint-basierte Modellieren* und das *prozedurale Modellieren* von Geometrie gegenüber, wird deutlich, dass beide zu einem unterschiedlichen Ansatz im Modellierungsprozess führen. Beim *Constraint-basierten Modellieren* können theoretisch alle Anforderungen über *Constraints* modelliert und anschließend Lösungen berechnet werden. Das *prozedurale Modellieren* erlaubt ausschließlich die Umsetzung eines kleinen Teils der Anforderungen. Beispielsweise wird ein festgelegter Abstand zweier Wände nur dann mit Sicherheit in einem *prozeduralen Modell* hinterlegt, wenn dies ein Teil des *Konstruktionsplans* ist. Weitere Abstände können auf Grund der geforderten Zyklenfreiheit nicht umgesetzt werden. Dies führt dazu, dass in diesem Fall nachträglich geprüft und gegebenenfalls der *Konstruktionsplan* geändert werden muss (vgl. dazu Abbildung 3.5). Daraus folgt, dass das *prozedurale Modellieren* für Untersuchungen des Entwurfs und „Was-wäre-Wenn-Analysen“ eher ungeeignet ist (vgl. Shah und Mäntylä, 1995, Seite 82).

Es existieren auch Mischformen, welche in aktuellen Softwaresystemen umgesetzt werden. Ein in kommerziellen Systemen umgesetztes Prinzip ist *2D-Constraint mit 3D-Geschichte* (vgl. „2D constraint with 3D history“ in Shah, 2001). Dabei handelt es sich um einen *Konstruktionsplan*, welcher Geometrien im dreidimensionalen Raum beschreibt. Darin befinden sich in einzelnen Konstruktionsschritten Skizzen-Ebenen, auf denen planare Topologie zur Geometriebeschreibung definiert wird. Diese Ebenen können an einer beliebigen Stelle des Raums oder relativ zu vorgeschalteten Geometrien des *Konstruktionsplans* liegen. In einer Skizzen-Ebene ist es möglich, die planare Topologie mit *Geometrie-Constraints* zu beschreiben. Dies dient normalerweise als Basis für *Sweep-Operationen*. Eine Software, die erkennbar dieses Prinzip umsetzt, ist beispielsweise *Autodesk Revit*.

Eine solche Mischform führt zu einer geometrischen Struktur von Bauwerksmodellen, die bestimmten Begrenzungen unterliegt. Dies soll an einem Beispiel aus dem Hochbau erläutert werden:

Ein Gebäude besteht aus verschiedenen Etagen, wobei die Grundebene jeder Etage in Abhängigkeit zu der Grundebene der darunter liegenden Etage beschrieben wird. Wird nun der *Parameter* „Höhe einer Etage“ geändert, so verschiebt dies immer die darüber liegenden Ebenen. Es handelt sich dabei um ein *parametrisch prozedurales Modell*, da die Parameteränderungen nur in eine Richtung zulässig sind. So ist es nicht möglich in die Gegenrichtung Änderungen vorzunehmen, indem beispielsweise die Wand einer Etage nach unten verschoben wird und sich die Höhe der darunter liegenden Etage anpasst.

Wird eine Etage betrachtet, so werden die Bauteile und Räume innerhalb der projizierten Grundebene der Etage unabhängig voneinander platziert. Die Grundebene kann als Skizze betrachtet werden, auf dem die Bauteile und Räume zusätzlich mit *Constraints* versehen werden, welche bei Änderungen erhalten bleiben. Bauteile können darin *parametrisch* aufgebaut sein, wobei für die einzelnen Bauteile wiederum ein *parametrisch prozedurales Modell* genutzt werden kann und gegebenenfalls weitere Skizzen in Ebenen zum Einsatz kommen können. Solch eine Modellierung entkoppelt durch die *prozeduralen* Abschnitte die einzelnen *Constraint-Systeme* in den Skizzen-Ebenen voneinander. Zwar lassen sich auch *Constraints* über die Ebenen hinweg definieren, diese können aber auf Grund der unidirektionalen *Prozeduralität* nicht für den Lösungsprozess genutzt werden. Ihr Zweck besteht maximal aus der Beschreibung der Zwangsbedingung und der Meldung, dass diese gegebenenfalls nicht mehr eingehalten werden kann, wenn eine nachträgliche Änderung ausgeführt wird.

4.4 Lösungsverfahren für geometrische CSP

Die Vielfalt der Ansätze der Lösungsverfahren für GCSP wird in verschiedenen Literaturquellen beschrieben und zusammengefasst (Hoffmann und Joan-Arinyo, 2005; Joan-Arinyo, 2009; Brüderlin u. a., 1998; Ait-Aoudia u. a., 2009; Obergrießer, 2017, Kapitel 4.6; Sitharam, St. John u. a., 2019). Für einen Überblick der geschichtlichen Entwicklung ist dabei die Veröffentlichung von Bettig und Hoffmann (2011) hervorzuheben. In dieser Arbeit wird in den nächsten Abschnitten ein kurzer Überblick über die Lösungsansätze gegeben, der auf der Veröffentlichung von Bettig und Hoffmann (2011) basiert. Für weitergehende Informationen sei auf die genannten Quellen verwiesen.

Lösungsverfahren für GCSP lassen sich auf verschiedene Weise klassifizieren. Eine Möglichkeit ist dabei die Einteilung in *ordnende Verfahren* und *nicht-ordnende Verfahren*. Diese unterscheiden sich darin, dass bei *ordnenden Verfahren* der letzte Schritt grundsätzlich das sequentielle Durchlaufen eines *Konstruktionsplans* (siehe Definition 4.3.1) beinhaltet. Ein *Konstruktionsplan* stellt die generierte bzw. gefundene Ordnung aus einzelnen *Konstruktionsoperatoren* dar.

Bei *nicht-ordnenden Verfahren* wird keine Ordnung gesucht. Dem Autor dieser Arbeit sind dabei einzig die *algebraischen Methoden* als *nicht-ordnende Verfahren* bekannt.

Eine weitere Möglichkeit der Klassifizierung bietet sich die verwendete Repräsentation des GCSPs in den einzelnen Ansätzen an. Es existieren *Graphen-basierte Ansätze*, *algebraische Methoden*, *Theorembeweis-Verfahren* und *Logik-basierte Ansätze*. Eine mögliche Taxonomie wird in Abbildung 4.3 gezeigt.

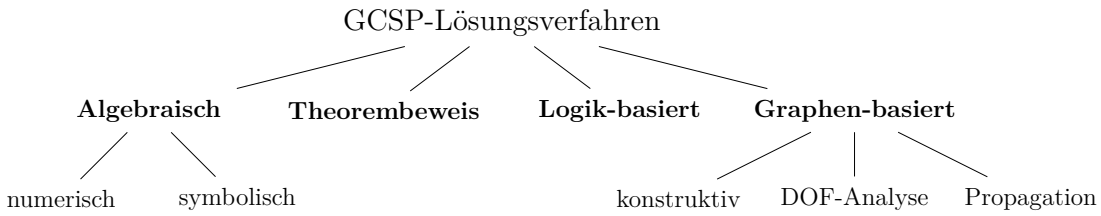


Abbildung 4.3: Taxonomie der Lösungsverfahren des *geometrischen Constraint-Satisfaction-Problems* (angelehnt an Obergrießer, 2017, Abbildung 4.26; Lipson u. a., 1999, Abbildung 3)

.

4.4.1 Anforderungen und Eigenschaften

Es existieren für Lösungsverfahren des GCSPs Anforderungen, die den sinnvollen Einsatz in der praktischen Anwendung ermöglichen sollen. Hoffmann und Joan-Arinyo (2005) schlagen unter anderem folgende Anforderungen vor:

Vollständigkeit: Alle möglichen GCSP sollen lösbar sein und unlösbare als solche erkannt werden.

Persistenz: Es sollen bei der wiederholten Suche nach einer Lösung in einem *Constraint-System* dieselben Lösungen berechnet werden.

Stabilität: Kleine Änderungen am System führen zu Lösungen in der „Nähe“. Da der Begriff „Nähe“ nicht eindeutig interpretierbar ist, wird dieser in Abschnitt 7.5.5 genauer untersucht.

Robustheit: Das Finden von Lösungen wird nicht von der begrenzten Zahlendarstellung des Rechners negativ beeinflusst.

Darüber hinaus führen Hoffmann und Joan-Arinyo die *Kompetenz* ein. Diese besagt, dass kein aktuelles Verfahren effizient und *vollständig* ist, es aber *kompetent* ist, wenn es im vorgesehenen Anwendungsbereich *vollständig* ist.

Der Funktionsumfang der Verfahren lässt sich dabei unter folgenden Gesichtspunkten bewerten:

Raumdimension: Werden zwei oder dreidimensionale Räume unterstützt (vgl. Hoffmann und Joan-Arinyo, 2005)?

Constrainttypen: Welche *Geometrie-Constraints* und darüber hinaus welche *algebraischen Constraints* werden unterstützt (vgl. Hoffmann und Joan-Arinyo, 2005)?

System-Struktur: Wie sind die Eigenschaften des zugrunde liegenden *Constraint-Graphen* (siehe Abschnitt 2.4) bzw. des Gleichungssystems? Ist ein System *serialisierbar* (vgl. Sitharam, St. John u. a., 2019, Seiten 145, 152), so lässt sich der *Constraint-Graph* analog zu einem *Konstruktionsplan* (siehe Definition 4.3.1) durchlaufen, bzw. das Gleichungssystem in eine Dreiecksform bringen. Komplexere und damit schwerer lösbare Systeme sind *nicht serialisierbar* (als „variational" bezeichnet in Sitharam, St. John u. a., 2019, Seite 145), da Teile des Systems sich gegenseitig bedingen.

Bestimmtheit: Werden ausschließlich *wohlbestimmte* oder auch *unterbestimmte* (siehe Abschnitt 2.3) Systeme unterstützt?

4.4.2 Algebraische Methoden

Bei den *algebraischen Methoden* wird das GCSP als NCSP betrachtet. Die Lösungen des GCSP entspricht dabei den Lösungen des Gleichungssystems des NCSPs.

Gleichungssysteme können mit unterschiedlichen Methoden gelöst werden:

Numerische Methoden: Die Nullstellen des Gleichungssystem werden mit numerischen Methoden berechnet (englisch: root finding), wie zum Beispiel dem *Newton-Raphson-Verfahren.*

Symbolische Methoden: Das Gleichungssystem wird auf eine Dreiecksform gebracht, welche ein einfaches Lösen, beginnend mit der Gleichung mit nur einer Unbekannten, ermöglicht.

Analyse des Gleichungssystems: Das Gleichungssystem wird analysiert und in Teilsysteme zerlegt, die unabhängig voneinander gelöst werden können.

Die Schwierigkeit beim Lösen von Gleichungssystemen besteht darin, dass *Geometrie-Constraints* fast immer nichtlineare Gleichungen darstellen (siehe Abschnitt 7.3). Dies lässt einfache Methoden des Lösens linearer Gleichungssysteme nicht zu. Erschwerend hinzu kommt, dass das Lösen des Gleichungssystems bei weniger oder mehr Gleichungen als *Variablen*, zusätzlichen Aufwand erfordern kann.

Das Prinzip der Zerlegung bei der *Analyse des Gleichungssystems* und die Ordnung der Lösung der *symbolischen Methoden* entspricht den Prinzipien der *Graphen-basierten Ansätze*, welche noch vorgestellt werden. Das Grundprinzip der *numerischen Methoden* besitzt hingegen keine Entsprechung bei den *Graphen-basierten Ansätzen.*

4.4.3 Logik-basierter Ansatz

Beim *Logik-basierten Ansatz* wird das *Constraint-System* des GCSPs in eine Menge von Annahmen und Axiomen übersetzt und anschließend durch einen *regelbasierten Reasoner* geprüft. Dabei wird das *Constraint-System* mit Hilfe der *Reasoner-Logik* so umgebaut (englisch: rewriting), dass die Lösungsschritte prozedural durchgeführt werden können (vgl. Hoffmann und Joan-Arinyo, 2005).

4.4.4 Theorembeweis-Verfahren

Das *Theorembeweis-Verfahren* (englisch: theorem proving) umfasst die Analyse der *Constraints* und Ermittlung geeigneter Axiome. Diese werden anschließend mit Hilfe von Theoremen verknüpft. Daraus kann eine valide Lösung ermittelt werden.

4.4.5 Graphen-basierte Ansätze

Das allgemeine Prinzip der *Graphen-basierten Ansätze* folgt dem Prinzip des *Teile-und-herrsche* (englisch: divide-and-conquor) mit folgendem Ablauf:

Schritt 1: Durchführung einer Analyse des *Constraint-Graphen*, der dem GCSP zugrunde liegt. Dabei wird der Graph in Teilprobleme zerlegt, für die Lösungsstrategien bekannt sind.

Schritt 2: Lösen der Teilprobleme.

Schritt 3: Zusammensetzen der Teilprobleme zu einer Lösung des Gesamtsystems.

Der *Konstruktionsplan* wird in diesem Fall aus einem *DR-Plan* (englisch: decomposition-recombination plan) abgeleitet (vgl. Sitharam, St. John u. a., 2019, Seite 182).

Die *Graphen-basierten Ansätze* teilen sich in drei unterschiedliche Herangehensweisen auf:

Konstruktiver Ansatz: Dabei werden im *Constraint-Graph* durch rekursive Mustererkennung Komponenten identifiziert, für die ein Zerlegungs-Muster bekannt ist. Üblicherweise besteht die Schwierigkeit - vor allem im dreidimensionalen Raum - darin, lösbare Teilprobleme zu definieren und zu identifizieren.

Freiheitsgradanalyse: Im *Constraint-Graph* wird jeder Knoten, der ein *Geometrie-Objekt* repräsentiert, mit der Anzahl der offenen *Freiheitsgrade* versehen. Jede Kante, die eine *Constraint* repräsentiert, verringert die Anzahl der offenen *Freiheitsgrade*. Auf Basis der verbliebenen *Freiheitsgrade* werden Lösungsstrategien gewählt.

Propagationsansatz: Das GCSP wird als NCSP betrachtet. Der zugehörige *Gleichungsgraph* wird so geordnet und umgeformt, dass er einen gerichteten Graphen ohne Zyklen darstellt. Das bedeutet, dass für jede unbekannte *Variable* genau eine Gleichung existiert, die nach der *Variablen* aufgelöst wird. Es existieren dabei Ansätze, die benötigte Zyklenfreiheit aufzubrechen.

Nach Bettig und Hoffmann (2011) stellen die *Graphen-basierten Ansätze* die führende Art der Lösungsverfahren dar. Auch Schultz u. a. (2017) nennen speziell den *konstruktiven Ansatz* als wichtiges Grundprinzip, welcher aber in einer hybriden Kombination mit *numerischen Methoden* Anwendung findet. Die Grundlage dafür wurde von Owen (1991) gelegt. Die *Constraint-Löser Siemens D-Cubed*, *Ledas LGS 2D*[3] und *FreeCAD*[4] sollen auf solch Hybriden-Ansätzen basieren (vgl. Schultz u. a., 2017). Dabei stellt beispielsweise *Siemens D-Cubed* die Grundlage für *Autodesk Inventor*[5] dar.

4.5 Das Problem der multiplen Lösungen in geometrischen Constraint-Systemen

4.5.1 Hintergründe

Das häufige Ziel bei der Beschreibung eines *geometrischen Constraint-Systems* ist, dass nur eine Lösung dafür existiert, es also *wohlbestimmt* (siehe Abschnitt 2.3) ist. In *unterbestimmten Constraint-Systemen* tritt mehr als eine Lösung auf. Wird ein *unterbestimmtes Constraint-System* angepasst, beispielsweise durch das Bewegen eines *Geometrie-Objekts*, ist die Bestimmung einer „beabsichtigten“ Lösung nicht eindeutig möglich. Dies wird als *Multiple-Solution-Problem* (deutsch: „Problem der multiplen Lösungen“) bezeichnet. Wird ein GCSP in ein NCSP umgeformt, so stellt jede mögliche Lösung des GCSPs genau eine mögliche Nullstelle des Gleichungssystems des NCSP dar. Das *Multiple-Solution-Problem* wird aus diesem Grund auch als Problem der *Nullstellenidentifikation* (englisch: root identification problem) bezeichnet.

Es lassen sich zwei Gründe identifizieren, warum ein geometrisches System *unterbestimmt* sein kann:

- Es ist ungewollt *unterbestimmt* und war eigentlich als *wohlbestimmt* beabsichtigt.

[3] http://ledas.com/en/expertise/3d-modeling/ (abgerufen am 29.10.2019)
[4] https://www.freecadweb.org/ (Abgerufen am 29.10.2019)
[5] www.autodesk.com/products/inventor/overview (abgerufen am 29.10.2019)

- Es ist gewollt *unterbestimmt*, und es wird eine beliebige oder eine bestimmte Lösung darin gesucht, oder es sollen die möglichen Parameterwerte bestimmt werden, für die Lösungen möglich sind.

Ungewollt *unterbestimmte* GCS werden in der Literatur als der übliche Fall bei multiplen Lösungen betrachtet. Der Grund dafür ist, dass davon ausgegangen wird, dass der Mensch ein Ergebnis mit einer eindeutigen Form konstruieren will. Ein häufiges Resultat *unterbestimmter* GCS ist das „Überschlagen“ von Flächen, wodurch sich die Orientierung der Fläche umdreht. Dies führt üblicherweise zu nicht korrekten Körpern im Sinne des *Solid Modelings* (siehe Abschnitt 2.5). Shimizu u. a. (1997) geben an, dass der Mensch in vielen Fällen zu wenige oder zu viele *Constraints* setzt. Sie leiten daraus ab, dass der Mensch Schwierigkeiten bei der Überprüfung des Systems auf *Wohlbestimmung* hat und aus diesem Grund dazu neigt, mehr *Constraints* zu setzen als notwendig sind.

Es wird im Folgenden von einem gewollt *unterbestimmten geometrischen Constraint-System* ausgegangen, für welches – wie es in den Zielen dieser Arbeit definiert ist – der Mensch bei der Entwurfsfindung zu unterstützen ist. Die Suche nach einer Lösung, die der Absicht des Menschen nahe kommt, und die Bestimmung der möglichen Parameterwerte, wird dabei genauer betrachtet.

4.5.2 Suche nach der beabsichtigten Lösung

Das Finden einer beliebigen Lösung in einem *unterbestimmten geometrischen Constraint-System* wird von Fudos u. a. (1997) als *NP-schwer* identifiziert.

Soll aber eine bestimmte Lösung, bzw. eine Lösung mit bestimmten Eigenschaften, gesucht werden, stellt dies das Problem der Lösungswahl (englisch: solution selector) dar. Es ergibt sich die Frage nach der Lösung, welche der Absicht des Menschen am nächsten kommt. Bouma u. a. (1995) schlagen dazu vier mögliche Vorgehensweisen vor, wobei sie dabei von einem zweidimensionalen geometrischen Entwurf ausgehen:

- Der Mensch verändert bestimmte *Geometrie-Objekte* so, dass sie seiner Vorstellung einer Lösung entsprechen. Anschließend wird eine möglichst ähnliche bzw. nahe Lösung gesucht, die alle *Constraints* erfüllt.

- Es werden solange zusätzliche *Constraints* durch den Menschen hinzugefügt, bis nur noch eine Lösung möglich ist.

- Die passende Lösung wird innerhalb des Lösungsprozesses interaktiv zwischen Mensch und Software bestimmt.

- Es wird ein hierarchisch strukturierter Entwurf eingeführt, der eindeutige *Regeln* enthält, sodass das Problem der multiplen Lösungen nicht auftritt.

Da der letzte Punkt keine Lösung im eigentlichen Sinne darstellt, lassen sich aus den anderen Vorgehensweisen vier Ansätze identifizieren: Dazu gehören heuristische Ansätze, die Wahl durch Interaktion zwischen Mensch und Software, die Einführung zusätzlicher *Constraints* und die *geometrische Anfrage*.

Heuristische Ansätze

Bei heuristischen Ansätzen wird versucht, während des Lösungsprozesses auf Basis von vordefinierten Strategien Entscheidungen zu treffen. Diese sollen zu einer Lösung führen, welche die topologische Orientierung des skizzierten Systems berücksichtigen (vgl. Fudos u. a., 1997)(Bouma u. a., 1995). Beispielsweise wird versucht, einen Punkt auf derselben Seite einer Geraden zu platzieren, auf der er sich bereits im vorher skizzierten System befand.

Heuristische Verfahren bieten zwar eine Lösung, aber es ist nicht sichergestellt, dass diese auch nur annähernde Ähnlichkeit mit der beabsichtigten Lösung des Menschen aufweist, selbst wenn eine solche möglich ist.

Zusätzliche Constraints

Die Einführung zusätzlicher *Constraints* erfolgt im Vorfeld des Lösungsprozesses, wobei gegebenenfalls die Prüfung, ob ein *geometrisches Constraint-System unterbestimmt* ist, die Voraussetzung dafür bietet.

Noort u. a. (1998) bestimmen auf Basis einer *Freiheitsgradanalyse* die Teile des Systems, welche *unterbestimmt* sind, und fordern vom Menschen das Einfügen weitere *Constraints*. Die Spezifizierung möglichst vieler zusätzlicher *Constraints* durch den Menschen wird auch von Luzón u. a. (2005) und Döring (2011, Kapitel 4) verfolgt.

Ein weiterer Ansatz ist, im Vorfeld weitere Angaben über die Art der gewünschten Lösung vorzunehmen. Dazu schlagen Bettig und Shah (2003) vor, weitere *Constraints* in Form von Ungleichungen einzuführen, welche den Lösungsraum weiter einschränken. Dabei handelt es sich um einfache Prädikate, wie zum Beispiel der relativen Lageangaben, wie *vor/dahinter*, *innen/außen* oder der relativen Orientierung, wie *selbe/unterschiedliche Richtung*. Sie leiten daraus weitere ab, wie beispielsweise *konkav/konvex*. Eine Implementierung des Konzepts wird nicht erwähnt. Weitere Prädikate in Form von Zielfunktionen werden im Abschnitt *Geometrische Anfrage* erwähnt.

Zur Vermeidung des Falls, dass bei zusätzlichen *Constraints* eine *Überbestimmung* auftritt und somit keine Lösung möglich ist, wenden Joan-Arinyo, Luzón u. a. (2003) einen *genetischen Algorithmus* an. Dieser versucht nach dem Prinzip der evolutionären Entwicklung, Lösungen zu finden, die möglichst viele der zusätzlichen *Constraints* erfüllen.

Interaktion im Lösungsprozess

Für die Interaktion im Lösungsprozess kann bei *konstruktiven Lösungsverfahren* im Lösungsprozess selbst die Entscheidung durch den Menschen erfolgen, wie *Geometrie-Objekte* zu platzieren sind. Dies setzt voraus, dass im Algorithmus jeder Fall identifiziert wird, der eine Entscheidung benötigt, und dafür alle Entscheidungsmöglichkeiten bekannt sind. Alternativ wird zuerst eine Lösung berechnet. Diese kann beliebig oder auf Grund von Heuristiken bestimmt werden. Anschließend werden durch den Menschen Änderungen am Prozess vorgenommen, sodass eine Lösung entsteht, die seinen Vorstellungen nahe kommt.

Für die Auswahl einer Lösung durch den Menschen stellen Bouma u. a. (1995) eine visuelle Navigation durch den *DR-Plan* des *Constraint-Systems* vor. Dabei soll der Mensch schrittweise Teillösungen für einzelne Teile des *Constraint-Systems* wählen. Sitharam, Arbree u. a. (2006) erweitern diesen Ansatz für numerische Lösungsverfahren und bieten eine Navigation zwischen den *Einzellösungen*.

Alle Ansätze erfordern ein hohes Maß an Nutzerinteraktion im Lösungsprozess bei den zu treffenden Entscheidungen. Die Schwierigkeit besteht darin, dass der Mensch nicht zwangsweise versteht, wie der Lösungsprozess intern aufgebaut ist und die *Constraints* darin repräsentiert werden (vgl. Bettig und Hoffmann, 2011). Dies ist bei umfangreichen Modellen mit vielen Lösungen in der Praxis kaum anwendbar.

Geometrische Anfrage

Die Idee der *geometrischen Anfrage* liegt darin, dass bestimmte Zielwerte vorgegeben werden, die bei der Lösung möglichst erfüllt werden sollen. Die Anfrage kann einerseits über eine Zielfunktion formuliert werden, wie sie Bettig und Shah (2003) konzipieren. Dabei soll das Minimum oder das Maximum einer Zielfunktion gesucht werden. Das führt beispielsweise zu der Lösung, in der ein Objekt *am weitesten links/rechts* liegt. Die Anfragen können andererseits auch *prototypisch* formuliert werden, indem der Mensch die *Geometrie-Objekte* so platziert, dass sie seiner Absicht möglichst gut entsprechen. Auch Parameterwerte können auf diese Weise einen prototypischen Wert zugewiesen bekommen. Anschließend wird eine Lösung gesucht, die dieser Absicht möglichst „nahe“ kommt.

Essert-Villard u. a. (2000) präsentieren ein Konzept, welches Ähnlichkeiten zwischen zweidimensionalen geometrischen Skizzen vergleicht. Sie schlagen vor, dieses bei der Lösungsfindung von *geometrischen Constraint-Systemen* umzusetzen. Darauf aufbauend definieren Meiden und Bronsvoort (2005) einfache geometrische Eigenschaften, die es ermöglichen, den Lösungsraum zu zerlegen. Dafür wird beispielsweise unterschieden, ob der Winkel an einer Ecke eines Dreiecks spitz oder stumpf ist. Alle Lösungen, welche einen spitzen Winkel enthalten, gehören zu einem Teil des Lösungsraums, alle mit einem stumpfen Winkel in einen anderen Teil des Lösungsraums. Alle Lösungen eines Teils des Lösungsraums ähneln sich und sind sich laut Meiden und Bronsvoort (2005) geometrisch „nah“. Die *prototypische Skizze* wird anhand solcher Eigenschaften klassifiziert und daraus der passende Teil des Lösungsraums identifiziert. Anschließend wird versucht, genau eine Lösung aus diesem Teil zu finden, wenn eine solche existiert. Andere Teile des Lösungsraums werden nicht weiter betrachtet. Speziell die Zerlegung des Lösungsraums von dreidimensionalen Problemen wird in einer späteren Veröffentlichung genauer untersucht (Meiden und Bronsvoort, 2010).

Die *geometrische Anfrage* stellt einen vielversprechenden Ansatz dar, da sie intuitiv durch den Menschen definiert werden kann. Die bisherigen Vorschläge konzentrierten sich dabei vor allem auf die Erkennung von Muster-Eigenschaften, die in den Lösungsprozess einfließen. Diese stellen im eigentlichen Sinne zusätzliche *Constraints* dar, welche aber automatisch erzeugt werden.

4.5.3 Bestimmung möglicher Parameterwerte

Die Bestimmung möglicher Parameterwerte ist ein aufwändiges Verfahren. Dabei ist zu unterscheiden, ob es sich um alle, möglicherweise disjunkten, gültigen Parameterbereiche oder den gesamten Parameterbereich (englisch: parameter range) in Form einer unteren und oberen Grenze handelt. Die Bestimmung der möglichen Werte ist durch den Menschen aufgrund des Aufwands nur selten möglich.

Die Bestimmung der Parameterwerte, für die Lösungen möglich sind, wird von Bettig und Hoffmann (2011) als offene Frage des GCSPs genannt. Chang (2014, Seite 242) erklärt, dass die möglichen Parameterwerte und deren Grenzen ein essentielles Ziel sind bei der Parametrisierung digitaler Bauteile. Er nennt auch das Problem, dass sich der Aufwand bei der Ermittlung der möglichen Parameterwerte bei mehr als einem variablen *Parameter* multipliziert.

Ein Algorithmus zur konstruktiven Berechnung möglicher Werte wird von Hoffmann und Kim (2001) vorgestellt. Das Ziel ist es dabei, die Topologie von Polygonen zu erhalten und zu bestimmen, wie groß der Spielraum einer *Constraint* ist, wenn sie angepasst

werden soll. Der vorgestellte Algorithmus ist dabei ausschließlich für eindimensionale, lineare Modelle anwendbar. Es ist damit möglich, die Wertebereiche der *Parameter* aller *Constraints* für den Abstand zweier Linien in Y-Richtung zu bestimmen. Das Beispiel von Hoffmann und Kim (2001) wird in dieser Arbeit für eine Auswertung mit der prototypischen Implementierung verwendet (siehe Abschnitt 8.4).

Die Berechnung der möglichen Werte genau eines *Parameters* unter der Bedingung, dass alle anderen *Parameter* nur einen diskreten Wert annehmen, wird in verschiedenen Arbeiten betrachtet. Mata Burgarolas u. a. (1999) stellen fest, dass die Berechnung des Parameterbereichs eines *Parameters* in einem *unterbestimmten* System mit genau einem *Freiheitsgrad* unter diesen Bedingungen *NP-schwer* ist. Einen Ansatz zur Berechnung bieten Meiden und Bronsvoort (2006) (erweitert in Meiden, 2008). Dabei ist es möglich, mit Abständen und Winkeln zu arbeiten, wobei der gültige Wertebereich eines speziellen *Varianten-Parameters* gesucht wird. Die Berechnung erfolgt auf einem *konstruktiven Graphen-basierten* Ansatz, bei dem bestimmte Punkte berechnet werden, an denen der *Varianten-Parameter* kritische Werte annimmt. Anschließend werden diejenigen bestimmt, an denen das Minimum bzw. das Maximum des Wertebereichs erreicht wird. Hidalgo Garcia (2013, Kapitel 4) untersucht diesen Algorithmus und führt den Nachweis, dass auch disjunkte Wertebereiche bestimmt werden können.

Für mehrere *Parameter* existiert der Vorschlag von Joan-Arinyo, Mata u. a. (2001), der auf Basis von *Intervall-Arithmetik* (siehe Abschnitt 5.3) mit einem Gradientenverfahren die Parameterbereiche aller *Parameter* eines GCS zu berechnen. Der von ihnen verwendete Algorithmus liefert aber nur bedingt brauchbare Ergebnisse, da diese überschätzt werden. Das bedeutet, dass die berechneten Wertebereiche deutlich größer sind, als sie es eigentlich sein sollten. Einen vergleichbaren Ansatz verwenden Wang u. a. (2007), die zur Lösung das *Gauß-Seidel-Verfahren* (siehe Abschnitt 5.4.6) einsetzen. Der von Wang u. a. (2007) vorgestellte Algorithmus funktioniert für das vorgestellte Beispiel. Wie aber in Abschnitt 5.8 genauer erläutert wird, ist dieses Verfahren nicht allgemein anwendbar, da es nur für einen Teil der Probleme exakte Intervalle liefert.

4.6 Constraint-basiertes Modellieren von Geometrie im Bauwesen

Im Bauwesen existieren verschiedene Ansätze, *Constraint-basiertes Modellieren* umzusetzen. Die Erstellung von *Constraints* mit Hilfe von verschiedenen Sprachen ist eines dieser Gebiete. Donath u. a. (2007) beschreiben einen Prototyp, in dem es möglich ist, mit Hilfe einer visuellen Skriptsprache verschiedene algebraische Ausdrücke zu definieren.

Daraus wird ein *Constraint-Graph* erzeugt. Für einfache Beispiele lassen sich Lösungen des Systems berechnen. Als Anwendungsgebiet definieren sie die Variantenuntersuchung im Beteiligungsprozess bei der Planung von Wohngebieten. Die Auswertung von Anforderungen in menschlicher Sprache wird von Niemeijer u. a. (2014) (auch in Niemeijer, 2011) vorgeschlagen und in einem Prototyp umgesetzt. Dabei werden Sätze wie „Die Länge der Wand muss mindestens 3 Meter und maximal 6 Meter betragen“ syntaktisch zerlegt und formalisiert. Die entstehenden algebraischen Ausdrücke können für einfache Beispiele Lösungen liefern. Ziel ist es, architektonische Anforderungen an Gebäude deklarativ zu beschreiben und auswerten zu können.

Die Modellierung von Tunnelbauwerken mit *Constraints* wird von Vilgertshofer u. a. (2017) beschrieben. Dabei handelt sich eigentlich um ein *prozedurales* Modell (siehe Abschnitt 4.3.3), vergleichbar mit dem von Borrmann, Ji u. a. (2012) beschriebenen, welches in den zweidimensionalen Skizzen-Ebenen *wohlbestimmte Constraint-Systeme* enthält. Die Idee ist, dass die Geometrie eines Tunnelbauwerks auf Basis eines Graphen modelliert wird. Entsprechende Veränderungen, wie zum Beispiel eine Erhöhung des Detailierungsgrads, findet dabei durch Ersetzung von Knoten durch neue Teilgraphen statt. Dies wird als „Graph Rewriting“ bezeichnet. Der hierarchische Graph stellt dabei den *prozeduralen Konstruktionsplan* dar, wobei in den Skizzen-Ebenen die entsprechenden *Constraint-Graphen* vorgehalten werden. Der *parametrische Ansatz* zur Modellierung von Infrastruktur-Bauwerken wird von Obergrießer (2017) genauer untersucht. Er entwickelt ein Konzept der Modellierung, in dem *Geometrie-Constraints* für den dreidimensionalen Raum eine Rolle spielen, aber die genaue Umsetzung nicht genau beschrieben wird. Darüber hinaus werden *Constraints* in zweidimensionalen Skizzen-Ebenen genutzt. Die *Freiheitsgrade* solcher Skizzen werden konzeptionell von Hand bestimmt und bewertet, da das Ziel ein parametrisch sinnvoll zu steuerndes Modell darstellt.

Schultz u. a. (2017) definieren mit Hilfe von *Geometrie-Constraints* und konstruktiven Hilfsobjekten Erzeugungsoperatoren, welche sie als *höherwertige Constraints* (englisch: high-level constraints) bezeichnen. Sie zeigen, wie komplexe Randbedingungen algorithmisch erzeugt und dem Modell hinzugefügt werden können. Dazu gehören beispielsweise Sichtlinien von Personen oder die Einhaltung der Mindestbreite von Wegen in Gebäuden, durch Generierung von Hilfslinien oder der Triangulation von Flächen, in Kombination mit einfachen *Geometrie-Constraints*, wie „Parallele-Linien“ oder „Orthogonale-Linien“.

4.7 Diskussion und Forschungsbedarf

Geometriemodelle stellen die Grundlage des CADs dar. Dabei existieren verschiedene Möglichkeiten, diese Modelle aufzubauen. Ein abschnittsweises *prozedurales Modellieren* kann durchaus sinnvoll sein, wenn von Anfang an die parametrischen Abhängigkeiten bekannt sind und nicht nachträglich geändert werden sollen. Es schränkt aber auch insofern ein, dass eine nachträgliche Änderung nur durch Anpassung von Parameterwerten oder einem vollständigen Neuaufbau der *prozeduralen* Schritte geschehen kann. *Prozedurales Modellieren* kann in Verbindung mit der *generativen Gestaltung* oder Parameterstudien genutzt werden, um Lösungen zu erzeugen und anschließend die Eingangswerte zu variieren und den Prozess wiederholt durchzuführen. Dabei ist nicht sichergestellt, dass die Lösung gefunden wird, die den Vorstellungen des Menschen nahe kommt, falls eine solche existiert. Das Resultat ist die heute gebräuchliche Methode aus *Versuch und Irrtum*, durch Variation der Eingangsdaten möglicherweise ein sinnvolles Ergebnis zu erzielen. Da *prozedurale Modelle* bei einem festen *Konstruktionsplan* nur bestimmte Zustände annehmen können und diese nur indirekt durch das Verändern der fest definierten *Eingabeparameter* variiert werden können, ist es bei komplexen Modellen selten möglich, sinnvolle Lösungen bei nachträglichen Änderungen zu finden.

Die *Wissensintegration* in Form von *Constraints* wird bereits in Forschungsarbeiten umgesetzt. Dabei ist YACS (Runte, 2006) zu nennen, welches ein generisches Framework für *Solver* eines *Constraint-Systems* mit *wissensbasierter* Konfigurierung darstellt. Es werden zwar alle Daten abgebildet, aber bestimmte Anforderungen können nicht erfüllt werden. Zum einen werden *Constraints* und Wertebereiche der *Parameter* zwar vorgehalten, dies geschieht aber ohne weitere Zuordnung zu *Wissen* aus einer spezifischen Domäne. Das bedeutet, es handelt sich um eine ungeordnete Sammlung von *Constraints*. *Wissen* darüber, wer sie integriert hat, was ihr Entwurfs-Zweck bzw. -Ursprung ist und wie sie semantisch zusammengehören, wird nicht vorgehalten. Zum anderen wird die eigentliche Schwierigkeit, die Bearbeitung und die Auswertung eines solchen Modells, nicht behandelt.

Betrachtet man den Bereich der *geometrischen Modellierung* von Produktmodellen, so existiert für die Beschreibung des Datenaustauschs die im Zuge von STEP (Standard for the exchange of product model data) entwickelte ISO 10303-108. Diese umfasst eine umfangreiche Beschreibung von Modellen und Konzepten zum *parametrischen Modellieren* von Geometrie mit Hilfe von *Parametern* und *Constraints*. Darin enthalten sind Modelle zur Beschreibung *geometrischer Constraint-Systeme*, bestehend aus *Parametern*, *Constraints* und expliziten *Geometrie-Objekten*. Sie enthält beispielsweise eine Menge

möglicher *Geometrie-Constraints*, zweidimensionaler Skizzen und Profile für *Sweeps* und erlaubt die Spezifizierung des möglichen Wertebereichs der Parameterwerte. Auch hierbei bleibt die Frage nach der Bearbeitung und der Auswertung eines solchen Modells ungelöst.

Ein weiteres Modell, das *Wissen* und Entwurfsideen auf Basis von *Constraints* integrieren soll, wird von Shih u. a. (1997) vorgeschlagen. Der Beitrag geht über die reine Konzeption und Beschreibung des Modells nicht hinaus.

Da keine Anwendung existiert, die Auswertungen für *Wissen* über den geometrischen Entwurf in *Bauwerksmodellen* ermöglicht und eine sinnvolle Integration des *Wissens* in ein solches Modell beschreibt, liegt der Entwicklungs- und Forschungsbedarf in zwei Bereichen:

- Ein Domänen-spezifisches Modell, welches Wissen über den geometrischen Entwurf eines Bauwerks auf Grundlage von *Constraints* und *Parametern* strukturiert und identifizierbar integriert.

- Die Untersuchung der Möglichkeiten, wie solche *geometrischen Constraint-Systeme* ausgewertet und sinnvoll genutzt werden können.

Die Integration von Wissen über *geometrische Constraint-Systeme* ist nur sinnvoll, wenn es zur Unterstützung des Menschen bei der Bestimmung geeigneter Entwürfe genutzt werden kann. Das System sollte *unterbestimmt* sein oder zumindest als solches erkannt werden. Nur in einem *unterbestimmten* System sind mehrere Lösungen möglich, die als Varianten vom Menschen gesucht und gefunden werden können.

Ausschließlich der *Graphen-basierte Ansatz* und die *Algebraischen Methoden* werden in aktuellen Softwareprodukten genutzt. Dabei ist es nicht trivial, Systeme zu berechnen, die mehr als die benötigen *Constraints* zur Beschreibung einer Lösung enthalten. Die Herausforderung besteht aber vor allem bei *geometrischen Constraint-Systemen*, die weniger als die benötigten *Constraints* besitzen, um genau eine Lösung zu bestimmen und die damit *unterbestimmt* sind. Für die Wahl einer Lösung, die der Vorstellung des Menschen möglichst nahe kommt, existieren zwar vielversprechende Ansätze, doch diese besitzen den Nachteil, dass sie es erfordern, mit zusätzlichen *Constraints* so lange Bereiche auszuschließen, bis eine Lösung zur Verfügung steht. Dies stellt bei komplexen Systemen einen langen iterativen Prozess mit umfangreichen manuellen Eingaben dar.

Die Bestimmung der gültigen Wertebereiche von *Parametern* ist bisher unbefriedigend (siehe Abschnitt 4.5). Für den *konstruktiven Ansatz* auf Basis von Graphen existiert lediglich eine Bestimmung für einen freien *Parameter*. Bei den *numerischen Verfahren* auf Basis von Intervallen findet laut den bisherigen Untersuchungen eine Überschätzung der gültigen Wertebereiche statt, oder ihre Ergebnisse sind nur eingeschränkt gültig.

Die Herausforderung ist es, in beliebig *unterbestimmten geometrischen Constraint-Systemen* Lösungen und Parameterbereiche zu berechnen. Der Ansatz auf Basis von Berechnungen mit *Intervallen* stellt, trotz der bisher beschriebenen Einschränkungen für Geometrie, einen vielversprechenden Ansatz dar. Im anschließenden Kapitel wird auf die bereits vorhandene Theorie und diverse Algorithmen für Intervalle vertieft eingegangen und geprüft in wie weit sie für die Ziele der Arbeit geeignet sind.

KAPITEL 5

Rechnen und Constraint-basiertes Modellieren mit Intervallen

In diesem Kapitel werden die Mathematik der Intervalle und das *Constraint-basierte Modellieren* mit Intervallen erläutert. Die Algorithmen zur Auswertung solcher Systeme auf Intervall-Basis und deren Eigenschaften werden vorgestellt und diskutiert.

5.1 Intervalle

In dieser Arbeit wird die Konvention genutzt, die zu den üblichen verwendeten gehört. Dabei werden Intervalle mit Großbuchstaben und die untere bzw. die obere Grenze mit einem Unter- bzw. Überstrich versehen:

$$X = [\underline{X}, \overline{X}] \tag{5.1}$$

Definition 5.1.1 (Intervall)**.** Ein Intervall ist eine definierte Menge zusammenhängender reeller Zahlen, definiert durch (vgl. Moore u. a., 2009, Seite 7):

$$X = [\underline{X}, \overline{X}] = x \in \mathbb{R} : \underline{X} \leq x \leq \overline{X} \text{ mit } X \in \mathbb{IR} \tag{5.2}$$

Da ein solches Intervall alle reellen Zahlen zwischen der unteren und oberen Grenze umschließt, ist es *kontinuierlich* (englisch: continous).

Dabei muss es sich nicht zwangsweise um ein geschlossenes Intervall handeln. Bei offenen oder halb-offenen Intervallen werden beide bzw. eine Intervallgrenze(n) ausgeschlossen.

Ein offenes Intervall ist definiert als:

$$X = (\underline{X}, \overline{X}) = x \in \mathbb{R} : \underline{X} < x < \overline{X} \text{ mit } X \in \mathbb{IR} \tag{5.3}$$

Ist dabei $\underline{X} = \overline{X}$ so spricht man von einem degenerierten Intervall: $X = [x]$.

Ein Vektor aus Intervallen wird als *Box* bezeichnet (vgl Moore u. a., 2009, Seite 15). Dies entspringt der Vorstellung, dass bei zwei *Variablen* die Intervalle in einem zweidimensionalen Raum ein Rechteck, in einem dreidimensionalen einen Quader, usw., ergeben. In Abbildung 5.1 wird eine zweidimensionale *Box* gezeigt.

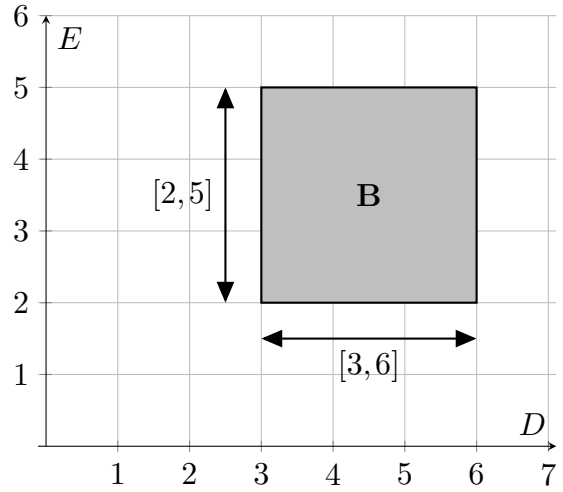


Abbildung 5.1: Beispielhafte grafische Darstellung einer *Box* $\mathbf{B} = (D \times E)$, die sich aus den Intervallen der Variablen $D = [3,6]$ und $E = [2,5]$ ergibt.

Das Intervall besitzt als Beschreibung der Teilmenge eines Zahlenraums verschieden Kenngrößen. Zum einen ergibt sich die Breite eines Intervalls als Differenz der oberen und unteren Grenze $width(X) = \overline{X} - \underline{X}$, zum anderen die Mitte des Intervalls als $mid(X) = \frac{\overline{X}+\underline{X}}{2}$.

Durch die Definition als Menge lassen sich auch die Operatoren aus der Mengentheorie auf *Intervalle* anwenden. Beispielsweise existieren die *Boolesche Mengenoperatoren* Schnitt $\cap$, Vereinigung $\cup$ und Differenz $\setminus$. Dabei ist speziell die Schnittmenge in Algorithmen von großer Relevanz:

$$A \cap B = [max(\underline{A}, \underline{B}), min(\overline{A}, \overline{B})] \tag{5.4}$$

Mit ihr lässt sich beispielsweise bei einer berechneten Einschränkung des Intervalls, ein bisheriger Wert mit einem neuberechneten Wert schneiden.

5.2 Grundlegende Literatur zu Intervallen

Durch Moore (1966) wurde die moderne Repräsentation von Werten und das Rechnen durch bzw. mit Intervallen eingeführt. Anschließend wurde dies durch eine Vielzahl an Forschungsprojekten und Untersuchungen weiter vorangetrieben. Grundlegende Erläuterungen der mathematischen Grundlagen und Algorithmen finden sich in einer Vielzahl von Büchern und zusammenfassenden Veröffentlichungen (Neumaier, 1990; Hansen und Walster, 2003; Neumaier, 2004; Moore u. a., 2009; Dawood, 2011; Kearfott, 2013; U. Kulisch, 2013; Kubica, 2015).

Darüber hinaus existiert auch der Implementierungsstandard IEEE Std 1788-2015, welcher die Repräsentation und die Operatoren der Arithmetik von Intervallen in Computersystemen beschreibt. Dieser baut auf dem Standard für Fließkommazahlen (IEEE Std 754-2008) auf, da Fließkommazahlen zur Repräsentation der Intervallgrenzen genutzt werden können. Eine hardwareseitige Umsetzung von Intervallen in Computer-Prozessoren ist aktuell nicht verfügbar. Es existieren aber bereits Vorschläge dafür (vgl. U. Kulisch, 2013, Kapitel 7).

Im Folgenden werden die Operationen der *Intervallarithmetik* beschrieben. Anschließend wird auf das *Constraint-basierte Modellieren* mit Intervallen und auf Algorithmen zur Optimierung und für das Finden von Lösungen in solchen *Constraint-Systemen* eingegangen.

5.3 Intervallarithmetik

5.3.1 Einführung

Das Rechnen mit Intervallen basiert darauf, dass mathematische Operatoren und die sich daraus zusammengesetzten Funktionen gemäß dem Inklusions-Theorem genutzt werden. Dieses fundamentale Theorem besagt, dass eine Funktion $F(X)$ eine inklusionsisotone Intervallerweiterung einer Funktion $f(x) : \mathbb{R} \to \mathbb{R}$ ist, wenn folgendes gilt (vgl. Ratz, 1997):

$$\text{F(X) ist inkludierend, wenn gilt } x \in X \text{ dann } f(x) \in F(X) \tag{5.5}$$

$$\text{und isoton, wenn } X \subseteq Y \text{ impliziert, dass } F(X) \subseteq F(Y) \tag{5.6}$$

Dies bedeutet, dass bei der Inklusionsisotonie sichergestellt ist, dass keine Teile der Wertemenge verloren gehen. Es bedeutet aber auch, dass die Ergebnismenge von $F(X)$ Bereiche umfassen kann, welche keine Entsprechung in der Definitionsmenge X besit-

zen. Dabei ist nicht sichergestellt, dass ein *Intervall* als Argument einer Funktion den Definitionsbereich überschreitet.

Ist bei einer inklusionsisotonen Funktion $F(X)$ sichergestellt, dass eine oder beide Grenzen eine Entsprechung in X besitzen, so werden solche Grenzen als *scharf* bezeichnet.

Die mathematischen Grundoperatoren $+$, $-$, $\cdot$,/ sind inklusionsisoton (vgl. Ratz, 1997). In verschiedenen Veröffentlichungen werden *Regeln* für Addition, Subtraktion, Multiplikation und Division aufgestellt. Diese unterscheiden sich ausschließlich in der Darstellung der Fallunterscheidungen, welche unterschiedlich zusammengefasst oder mehr oder weniger detailliert behandelt werden müssen. In der Konsequenz liefern sie aber dieselben Ergebnisse.

Die Grundoperatoren von zwei Intervallen sind:

$$A+B = [\underline{A}+\underline{B}, \overline{A}+\overline{B}] \tag{5.7}$$

$$A-B = [\underline{A}-\overline{B}, \overline{A}-\underline{B}] \tag{5.8}$$

$$A \cdot B = [min\{\underline{A}\cdot\underline{B}, \underline{A}\cdot\overline{B}, \overline{A}\cdot\underline{B}, \overline{A}\cdot\overline{B}\}, max\{\underline{A}\cdot\underline{B}, \underline{A}\cdot\overline{B}, \overline{A}\cdot\underline{B}, \overline{A}\cdot\overline{B}\}] \tag{5.9}$$

$$A/B = A\cdot[1/\overline{B}, 1/\underline{B}] \text{ für } 0 \notin B \tag{5.10}$$

Für die in *Geometrie-Constraints* wichtigen Operatoren X^2 und $\sqrt{X}$ lassen sich ebenfalls inklusionsisotone Intervallerweiterungen definieren:

$$A^2 = [\left\{ \begin{array}{ll} 0, & \text{wenn } 0 \in A \\ min\{\underline{A}^2, \overline{A}^2\}, & \text{wenn } 0 \notin A \end{array} \right\}, max\{\underline{A}^2, \overline{A}^2\}] \tag{5.11}$$

$$\sqrt{A} = [-\sqrt{\overline{A}}, \sqrt{\overline{A}}] \qquad \text{wenn } \underline{A} \geq 0 \tag{5.12}$$

Weitere Operatoren können ebenfalls für Intervalle erweitert werden. Diese werden in dieser Arbeit aber nicht betrachtet, da sie für die Gleichungen der hier genutzten *Geometrie-Constraints* nicht benötigt werden. Für eine Übersicht an erweiterten Operatoren und Funktionen, wie zum Beispiel trigonometrische Funktionen, sei auf Daumas u. a. (2009) verwiesen.

Bei der Implementierung von Intervallen mit Fließkommazahlen ergibt sich aus dem Inklusions-Theorem die Konsequenz, dass die immanente Ungenauigkeit der Fließkommazahloperationen durch entsprechendes Runden nach außen (englisch: outward rounding) berücksichtigt wird. Dies bedeutet für die untere Grenze eine mathematische Ab- und für die obere eine entsprechende Aufrundung. So wird sichergestellt, dass trotz der numerischen Ungenauigkeit, das Ergebnis durch das Intervall umschlossen wird. Ein Beispiel

ist in Tabelle 5.1 gezeigt, in dem verschiedene Ergebnisse der Fließkommazahlarithmetik mit der *Intervallarithmetik* gegenübergestellt werden.

Tabelle 5.1: Vergleich der Ergebnisse der Division 1000/7777 mit Fließkommaarithmetik und der gerundeten *Intervallarithmetik*, bei der Verwendung von Fließkommazahlen einfacher (single) oder doppelter (double) Präzision. Unterschiede in den Dezimalstellen sind **fett** markiert.

Arithmetik	**Ergebnis**
Fließkommazahl (single)	0.12858428
Fließkommazahl (double)	0.12858428**700012858**
Intervallarithmetik (single)	[0.12858428, 0.12858429]
Intervallarithmetik (double)	[0.12858428**700012858**, 0.12858428**700012859**]

5.3.2 Erweiterte Intervalle und Intervallarithmetik

Da die Inklusionsisotonie es nicht zulässt, dass Werte aus dem Argument keine Zugehörigkeit im Ergebnis besitzen, kann der Zahlenraum um die leere Menge erweitert werden. Dadurch ist beispielsweise $\sqrt{[-5,4]} = [-2,2]$ oder $\sqrt{[-2,-1]} = \emptyset$.

Zusätzlich lassen sich auch unbegrenzte Intervallgrenzen einführen, wie zum Beispiel bei dem *vollständigen* Intervall (englisch: entire): $(-\infty, \infty)$. Dadurch lassen sich, wie bei Fließkommazahlen nach IEEE Std 754-2008, bestimmte Grenzfälle abbilden. Mit diesen *erweiterten Intervallen* (vgl. U. Kulisch, 2013, Seite 140) lassen sich die Division durch Intervalle, die 0 enthalten, definieren. Beispielsweise ist die erweiterte Division $[12,12]/[0,3] = [4,\infty)$ oder $[12,12]/[0,0] = \emptyset$. Die Erweiterung für die Basis-Operatoren um nichtbegrenzte Intervallgrenzen wird unter anderem von U. Kulisch (2013, Seite 141-143) beschrieben.

Ratz (1997) stellt fest, dass für die erweiterte Division verschiedene Varianten entwickelt wurden, wobei nur eine den Anforderungen entspricht, korrekte und *scharfe* Grenzen zu liefern. Stellt man darüber hinaus die Anforderung, dass für jeden Funktionswert $y \in F(X)$ gelten soll, dass $x \in X$ existieren muss, sind bestimmte Operatoren wie die Division oder die Quadratwurzel zu erweitern. Dabei existieren Fälle, in denen das Ergebniss mehr als ein Intervall liefert. Dies ist beispielsweise bei folgender Division der Fall: $[12,12]/[-1,3] = (-\infty,-12] \cup [4,\infty)$. Eine Übersicht aller auftretenden Kombinationen für die erweiterte Division wird von U. Kulisch (2013, Tabellen 4.12 und 4.13) gegeben. Algorithmisch lässt sich dies entsprechend der Fälle in Tabelle 5.2 umsetzen. Die Fälle der Quadratwurzelfunktion sind in Tabelle 5.3 dargestellt.

Fall	$A = [\underline{A}, \overline{A}]$	$B = [\underline{B}, \overline{B}]$	A/B
1	$0 \in A$	$0 \in B$	$(-\infty, +\infty)$
2	$0 \notin A$	$B = [0, 0]$	$\emptyset$
3	$\overline{A} < 0$	$\underline{B} < \overline{B} = 0$	$[\overline{A}/\underline{B}, \infty)$
4	$\overline{A} < 0$	$\underline{B} < 0 < \overline{B}$	$(-\infty, \overline{A}/\overline{B}] \cup [\overline{A}/\underline{B}, +\infty)$
5	$\overline{A} < 0$	$0 = \underline{B} < \overline{B}$	$(-\infty, \overline{A}/\overline{B}]$
6	$\underline{A} > 0$	$\underline{B} < \overline{B} = 0$	$(-\infty, \underline{A}/\underline{B}]$
7	$\underline{A} > 0$	$\underline{B} < 0 < \overline{B}$	$(-\infty, \underline{A}/\underline{B}] \cup [\underline{A}/\overline{B}, +\infty)$
8	$\underline{A} > 0$	$0 = \underline{B} < \overline{B}$	$[\underline{A}/\overline{B}, +\infty)$

Tabelle 5.2: Algorithmisch zu behandelnde Fälle bei der Division in der *erweiterten Intervallarithmetik* (aus U. W. Kulisch, 2009, Seite 3). Die Division der Intervallgrenzen erfolgt nach den *Regeln* der Fließkommaarithmetik gemäß IEEE Std 754-2008.

Fall	$A = [\underline{A}, \overline{A}]$	$\sqrt{A}$
1	$\overline{A} < 0$	$\emptyset$
2	$\underline{A} \leq 0$	$[0, \sqrt{\overline{A}}]$
3	$\underline{A} > 0$	$[\sqrt{\underline{A}}, \sqrt{\overline{A}}]$

Tabelle 5.3: Algorithmisch zu behandelnde Fälle bei der Quadratwurzelfunktion in der *erweiterten Intervallarithmetik* (aufbauend auf Daumas u. a., 2009). Die Quadratwurzel der Intervallgrenzen erfolgt nach den *Regeln* der Fließkommaarithmetik gemäß IEEE Std 754-2008.

5.3.3 Intervall-Abhängigkeits-Problem

Das *Intervall-Abhängigkeits-Problem* (englisch: interval dependency problem) tritt auf, wenn *Variablen* in Berechnungen mehrfach auftreten (vgl. Moore u. a., 2009, Seite 38). Dabei bleibt die eigentliche Abhängigkeit zwischen gleichen reelen Werten einer mehrfach auftretenden *Variablen* unberücksichtigt.

Ein Beispiel ist der Vergleich der Quadrierung einer *Variablen* mit der Multiplikation einer *Variablen* mit sich selbst. Bei Verwendung einer reellen Arithmetik führt dies zum identischen Ergebnis, während die *Intervallarithmetik* bei der Multiplikation das Ergebnis überschätzt:

$$A = [-2, 2] \tag{5.13}$$

$$A \cdot A = [-4, 4] \tag{5.14}$$

$$A^2 = [0, 4] \tag{5.15}$$

Dabei berücksichtigt die Quadratfunktion die Abhängigkeit, dass bei der Multiplikation von A mit sich selbst ausschließlich dieselben reellen Zahlen aus der Menge von A multipliziert werden. Im Gegensatz dazu wird bei der aufgeführten Multiplikation davon ausgegangen, dass die zwei Faktoren voneinander unabhängig sind und auch so in der Berechnung berücksichtigt werden. Grundsätzlich gilt durch die Inklusionsisotonie, dass das Ergebnis einer Berechnung, welche das multiple Auftreten von *Variablen* berücksichtigt, eine Teilmenge des Ergebnisses ist, bei dem dies keine Berücksichtigung findet.

Dies ist für einzelne Operatoren und für Algorithmen insofern problematisch, da es zu einem Überschätzen der Grenzen führt, und diese ihre *Schärfe* verlieren können. Aus diesem Grund sollte das multiple Auftreten reduziert werden, soweit es möglich ist. Dies führt auch zu einer Effizienzsteigerung von Algorithmen, da Wertebereiche deutlich schneller reduziert werden können. Im Falle des Ersetzens der Multiplikation mit sich selbst durch die Quadratfunktion, ergibt sich beispielsweise eine halbierte Breite des Wertebereich des Ergebnisses.

5.4 Constraint-basierte Systeme mit Intervallen

5.4.1 Intervall-Constraints

Intervall-Constraints sind Relationen zwischen einer oder mehrerer *Variablen*. Dabei beschreibt eine *Constraint* den Zusammenhang der *Variablen* in Form einer algebraischen Gleichung oder Ungleichung.

Es lassen sich eine Vielzahl von komplexeren Gleichungen als *zusammengesetzte Constraints* beschreiben. Diese bilden sich aus einer Verkettung einfacher mathematischer Operatoren, den so genannten *primitiven Constraints*, bei denen jede *Variable* nur einmal auftaucht (vgl. Cruz, 2003, Seite 55; Dawood, 2011, Seite 36; Ceberio u. a., 2001).

Die Gleichungen lassen sich nach jeder beliebigen *Variablen* umstellen bzw. invertieren, sodass diese aus den anderen *Variablen* berechnet werden kann (vgl. Cruz, 2003, Tabelle 4.1). Dieses Konzept bezeichnet man als *Projektion einer Constraint* (vgl. Cruz, 2003, Seite 53; Lhomme, 1993).

Als Beispiel sei hier die Quadratwurzelfunktion $\sqrt{X} = Y$ genannt, welche umgestellt werden kann zu $X = Y^2$. Umgekehrt ergibt sich dabei aber ein Sonderfall: $X = Y^2$ ist invertiert $\pm\sqrt{X} = Y$.

Eine *Constraint* kann dabei *disjunktiv* sein, wenn eine Projektion die Konvexität der Eingangswerte nicht im Ergebnis erhält, also mehrere Intervalle das Ergebnis beschreiben müssten (vgl. Lhomme, 1993). Dabei können *Constraints*, welche die Operatoren der „klassischen“ *Intervallarithmetik* (siehe Abschnitt 5.3) nutzen, *disjunktiv* sein, während die *erweiterte Intervallarithmetik* für nicht *disjunktive Constraints* verwendet werden kann. Ein Beispiel für die *Disjunktivität* ist in Tabelle 5.4 dargestellt.

Arithmetik	**Projektion auf X**	**disjunktiv**
klassisch	$X = [-4, 4]$	ja
erweitert	$X = [-4, -1] \cap [1, 4]$	nein

Tabelle 5.4: Beispiel der Disjunktivität einer *Constraint* mit der Gleichung $X = Y^2$, bei $X = [-16, 16]$ und $Y = [1, 4]$, projiziert auf X, bei der Anwendung unterschiedlicher Arithmetiken

Die Vorteile der Nutzung von *Intervall-Constraints* sind die mögliche Invertierung und die sich daraus ergebende multidirektionale Projektion von Werten, die je nach Bedarf gewählt werden kann. Darüber hinaus kann bei differenzierbaren Funktionen die partielle Ableitung nach einer *Variablen* je nach Bedarf bestimmt werden.

5.4.2 Intervall-Constraint-Satisfaction-Problem

Ein *Intervall-Constraint-System* (ICS) ist die Erweiterung eines NCSs (siehe Abschnitt 2.3) um *kontinuierliche* Intervalle. Die *Variablen* des ICS besitzen als Wertebereiche (englisch: domains) reellwertige Intervalle der Form $X \in \mathbb{IR}$. Darüber hinaus besteht ein ICS aus einer Menge von *Intervall-Constraints*. Es kann dabei als Gleichungssystem aufgefasst werden, in dem jede Gleichung einer *Constraint* nach 0 aufgelöst wird. Die Schwierigkeit der

Bestimmung einer Lösung eines solchen Gleichungssystems wird als *Intervall-Constraint-Satisfaction-Problem* (ICSP) bezeichnet. Da die Intervalle selbst eine untere und obere Grenze besitzen, können sie auch als *Variable* eines NCS betrachtet werden, welche die valide Intervallmenge der *Variablen* mit je einer Ungleichung für die untere und für die obere Grenze beschränken.

Es ist dabei zu unterschieden, ob eine Gesamtlösung für das Gleichungssystem bzw. das ICS gesucht wird oder eine *Einzellösung* (englisch: canonical solution). Dabei ist eine *Einzellösung* die Kombination von reellen Zahlen für jede *Variable*, für die alle *Constraints* valide sind. Das bedeutet für das zugrunde liegende Gleichungssystem, dass es sich bei der Kombination der Zahlenwerte der *Einzellösung* um Nullstellen in allen Gleichungen handeln muss. Eine Gesamtlösung bedeutet, dass für jede *Variable* die Intervalle mit *scharfen* Grenzen gefunden werden, in denen ausschließlich Werte für *Einzellösung* existieren.

Es ergibt sich die Grundfrage: Welche Wertekombinationen erfüllen dabei alle gegebenen Gleichungen bzw. Ungleichungen? Darüber hinaus existieren weitere Fragen, wie zum Beispiel die nach dem globalen Minimum oder Maximum einer Funktion unter der Erfüllung aller *Constraints* (englisch: rigorous global optimization) oder die nach der unteren und oberen Grenze des Werts jeder *Variablen*, für die eine valide Wertekombination existiert. Dies ist analog zu der Suche nach den Wertebereichen der *Parameter* in *geometrischen Constraint-Systemen* zu sehen (siehe Abschnitt 4.5.3).

5.4.3 Konsistenzen eines Intervall-Constraint-Systems

Die Qualität der Gültigkeit eines ICSPs für eine bestimmte *Box* wird als *Konsistenz* bezeichnet. Es existieren *lokale* bzw. *partielle Konsistenzen*, die nur für einen bestimmten Teil des *Constraint-Systems* gültig sind, und *globale Konsistenzen*, die sich auf das *Constraint-System* als Ganzes beziehen. Dabei gilt: Entsprechen alle *Constraints* eines ICS einer *Konsistenz*, so entspricht auch das gesamte ICS dieser *Konsistenz*.

Es lassen sich allgemein für CSP verschiedene *Konsistenzgrade* unterscheiden. Dafür wird die *k-Konsistenz* mit $k \geq 2$ eingeführt. Diese besagt, dass eine *Variable* in Bezug auf die Intervalle von $k-1$ anderen *Variablen* des ICS Teil einer Lösung sein muss. Dies gilt in jeder Kombination von *Variablen* des ICS. Handelt es sich zusätzlich um eine *starke Konsistenz*, dann gilt auf rekursive Art und Weise die nächst niedrigere *Konsistenz*, die sogenannte *(k-1)-Konsistenz*, (vgl. Cruz, 2003, Seite 73-74).

Es existieren speziell für Intervalle verschiedene Arten von *Konsistenzen*, die sich anhand des *Konsistenzgrads* und ihres Ziels unterscheiden:

Kanten-Konsistenz (englisch: arc-consistency) ist eine *lokale Konsistenz*, welche für jede einzelne *primitive Constraint* autonom gültig ist (vgl. Mackworth, 1981; Miguel u. a., 2001). Dabei existiert bezüglich der betrachteten *Constraint* für das vollständige Intervall jedes Wertebereichs einer *Variablen* immer ein entsprechender Wert im Wertebereich einer anderen *Variablen*.

Box-Konsistenz: (englisch: box-consistency) Jedes Intervall eines Wertebereichs einer *Variablen* ist genau so groß, dass seine untere und obere Grenze für sich konsistent sind in Bezug auf alle anderen *Variablen* einer *Constraint*. Dabei wird ein multiples Auftreten einzelner *Variablen* in *Constraints* (siehe Abschnitt 5.3.3) nicht berücksichtigt (vgl. Benhamou, McAllester u. a., 1994).

Hüllen-Konsistenz ist eine *Konsistenz*, die sich ausschließlich auf die untere bzw. obere Grenze einer *Variablen* bezieht. Sie besagt, dass die Grenzen einer *Variablen* in Bezug auf andere *Variablen* zu einer Lösung führen können. Auf welche anderen *Variablen* dabei Bezug genommen wird, hängt von der Anzahl $k \geq 2$ der beteiligten *Variablen* ab. Lhomme (1993) führte dafür analog zur *k-Konsistenz*, welche sich auf alle Werte im Intervall bezieht, die *kB-Konsistenz* (B für englisch: bounds; deutsch: Grenzen) ein. Diese besagt, dass die Grenzen einer *Variablen* in Bezug auf die Intervalle von $k-1$ anderen *Variablen* des ICS Teil einer Lösung sein müssen. Dies gilt dabei in jeder Kombination von *Variablen* des ICS. Daraus ergeben sich folgende Formen der *Hüllen-Konsistenz*:

2B-Konsistenz: (auch (Lokale) *Hüllen-Konsistenz* bzw. englisch: hull-consistency) Jedes Intervall eines Wertebereichs einer *Variablen* ist genau so groß, dass seine untere und obere Grenze, in Bezug auf alle anderen *Variablen* einer *Constraint* für sich konsistent sind (vgl. Collavizza u. a., 1999).

3B-Konsistenz ist eine *globale Konsistenz*. Jedes Intervall eines Wertebereichs einer *Variablen* ist genau so groß, dass seine untere und obere Grenze in Bezug auf die oberen und unteren Grenzen alle anderen *Variablen* des ICS konsistent sind (vgl. Collavizza u. a., 1999).

Globale-Hüllen-Konsistenz (englisch: global hull-consistency) ist identisch mit der *(n+1)B-Konsistenz*, wobei gilt: $n =$ Anzahl aller Variablen des ICS. Jedes Intervall eines Wertebereichs einer *Variablen* ist genau so groß, dass dessen Grenzen Teil von *Einzellösungen* sind und das Intervall gegebenenfalls weitere *Einzellösungen* umschließt (vgl. Cruz, 2003, Seite 81).

5.4.4 Ansätze für Lösungsverfahren und ihre Einflussgrößen

Das Erreichen einer *Konsistenz* und die Berechnung von Lösungen hängen stark von der Art und Struktur des zugrunde liegenden Gleichungssystems ab. Dabei existieren verschiedene Unterscheidungsmerkmale, die den Schwierigkeitsgrad eines ICSPs bestimmen und teilweise Algorithmen ausschließen, wenn diese bestimmte Fälle nicht ausreichend behandeln können. Unterscheidungsmerkmale sind:

1. Es handelt sich nur um eine einzelne Gleichung oder ein Gleichungssystem.
2. Eine Gleichung ist *univariat* (besteht nur aus einer *Variablen*) oder *multivariat* (besteht aus mehreren *Variablen*).
3. Das Verhältnis der Anzahl unbekannter *Variablen* zur Anzahl der Gleichungen ist entweder kleiner, größer oder gleich 1.
4. Es ist ein lineares oder nichtlineares Gleichungssystem.
5. Einfaches oder mehrfaches Auftreten einer *Variablen* in einer oder in mehreren Gleichungen.
6. Das Gleichungssystem ist in Form eines *Constraint-Graphen* zyklenfrei, oder es enthält Zyklen.

Da in dieser Arbeit *geometrische Constraint-Systeme* betrachtet werden, die in algebraischer Form nichtlineare Gleichungssysteme darstellen (vgl. Kubica, 2015), werden ausschließlich Algorithmen betrachtet, die für nichtlineare Gleichungssystemen zum Einsatz kommen können.

Für Lösung eines ICSP können verschiedene Berechnungsmethoden verwendet werden. Es existieren dabei zwei unterschiedliche Ansätze:

Intervall-Constraint-Propagation: Dabei werden Werte der *Variablen* mit Hilfe der Berechnungsvorschrift der *Constraints* aufeinander *projiziert* bzw. abgeglichen. Die zum Einsatz kommenden Abgleich-Operatoren werden als *Kontraktoren* (englisch: contractors) bezeichnet. Die *Intervall-Constraint-Propagation* kann chaotisch ablaufen, wobei sich die geordneten Verfahren als effizienter erweisen (vgl. Kjøller u. a., 2007).

Intervall-Newton-Methoden: Dabei wird das ICS als Gleichungssystem betrachtet, für das Nullstellen gesucht werden. Mit Hilfe der Differentiation lassen sich diese Nullstellen eingrenzen.

Bei einem CSP mit diskreten Zahlenwerten führen diese Ansätze grundsätzlich zu einem Widerspruch, einer Lösung oder mehreren Lösungen. Im Gegensatz dazu kommt es bei intervallwertigen *Variablen* zu einem möglichen weiteren Ergebnis: Das Ergebnis einer Berechnungsmethode ist nicht ausreichend, eine Aussage zu treffen, da eine Überschätzung des Ergebnisses auftreten kann (siehe Abschnitt 5.3.3).

Es führt dazu, dass für *Boxen* eine leere Menge als Ergebnis möglich ist, da das ICS für eine solche *Box* keine Lösung enthält. Dies wird als *Unmöglichkeits-Test* (englisch: infeasibility test) bezeichnet (vgl. Ratschek u. a., 1991). Der Umkehrschluss in Form eines *Möglichkeits-Tests* (englisch: feasibility test) ist nicht ohne weiteres möglich und hängt von der Struktur des ICS ab.

Für das Finden von Lösungen für ein ICSP hat sich das *Branch-and-Prune-Verfahren* etabliert, welches im nächsten Abschnitt genauer erläutert wird. *Intervall-Constraint-Propagation* und *Intervall-Newton-Verfahren* können dafür als *Pruning-Operatoren* (vgl. Benhamou und Granvilliers, 1996) zum Einsatz kommen, da sie Werte von *Variablen* einschränken, ohne Lösungen auszuschließen. Darüber hinaus stellen sie *Unmöglichkeits-Tests* dar.

5.4.5 Das Branch-and-Prune-Verfahren

Grundsätzlich kann für das Lösen eines ICSPs das *Branch-and-Bound-Verfahren* angewendet werden. Der zu durchsuchende Intervallraum wird dabei in eine Baumstruktur unterteilt. Der Wurzelknoten umfasst dabei den gesamten Raum und die Kindknoten jeweils eine disjunkte Teilmenge ihres Elternknotens. Jeder Knoten repräsentiert somit einen Teilbaum bzw. *Ast* (englisch: branch) in Form einer *Box*. Jede Teilmenge eines *Asts* wird dahingehend bewertet, ob sie eine Grenze nicht überschreitet, bzw. ob sie eine sinnvolle Lösung liefern kann. Ist das nicht der Fall, wird die Teilmenge nicht weiter betrachtet. Ansonsten wird sie weiter durchsucht, indem sie wiederum in Teilmengen zerteilt (englisch: branching) wird. Dies wird solange fortgesetzt, bis eine Teilmenge dem Anspruch einer Lösung genügt oder festgestellt wird, dass keine solche Lösung gefunden werden kann.

Eine Effizienzsteigerung ergibt sich durch eine Variante des *Branch-and-Bound-Verfahrens*: das *Branch-and-Prune-Verfahren*. Das *Branch-and-Prune-Verfahren* stellt das grundlegende Schema eines Großteils der Algorithmen für das Lösen eines ICSPs dar. Der grundsätzliche Ablauf des *Branch-and-Prune-Verfahrens* erfolgt dabei in vier Phasen und wird theoretisch solange wiederholt, bis der Suchraum vollständig durchsucht oder eine bestimmte Lösung gefundenen wurde:

1. *Extract*: Wähle einen zu durchsuchenden *Ast*.
2. *Prune*: Verkleinere Suchraum, wenn möglich.
3. *Prove*: Prüfe, ob der Suchraum ausgeschlossen werden kann.
4. *Branch*: Zerteile den Suchraum für den nächsten Durchlauf.

In der *Extract*-Phase erfolgt die Auswahl des nächsten zu prüfenden *Asts*, der einen Bereich des Suchraums darstellt. Anschließend erfolgt das Beschneiden (englisch: prune) eines *Asts* mit Hilfe von *Pruning-Operatoren* (auch unter dem Begriff *Narrowing Operatoren* bekannt) (vgl. Benhamou und Granvilliers, 1996). Dabei kann es bereits zum Ausschluss eines *Asts* kommen, da ein *Pruning-Operator* feststellt, dass keine Lösung enthalten sein kann. Jeder *Pruning-Operator* bewirkt entweder das Verwerfen eines *Asts*, da dieser ausgeschlossen werden kann (*Nicht-Existenz-Tests*) oder erreicht einen *Fixpunkt*. Ab diesem *Fixpunkt* würde die wiederholte Anwendung des *Pruning-Operators* zu keiner weiteren Verkleinerung der *Box* des *Asts* führen (vgl. Cruz, 2003, Definition 4.1-3).

In der *Prove*-Phase wird mit Hilfe von *Unmöglichkeits-Tests* geprüft, ob der *Ast* verworfen werden kann. Durch die Überschätzung von Intervallen durch die *Intervallarithmetik*, ist es bei den meisten Verfahren nur möglich, *Äste* auszuschließen. Eine eindeutige Aussage, ob eine Lösung enthalten sein muss, ist nur in bestimmten Fällen möglich. Wenn nicht verworfen, wird ein *Ast* in der *Branch*-Phase in neue *Äste* aufgespalten bzw. die *Box* weiter unterteilt. Die Unterteilung, das sogenannte *Splitten* (englisch: splitting), erfolgt durch das Zerteilen eines Intervalls ausreichender Breite in neue Intervalle. Dabei wird jedes neue Intervall einem neuen *Ast* zugeordnet. In den meisten Fällen erfolgt das Splitten in genau zwei Teile, dem sogenannten *Bisektions-Verfahren*. Der *Split* erfolgt üblicherweise in der Mitte des Intervalls, sodass die resultierenden Intervalle eine identische Breite besitzen. Ist kein Split mehr möglich, da alle Intervalle auf eine Breite zerteilt wurden, die einen vordefinierten Grenzwert unterschreitet, so wird der *Ast* als *Einzellösung* betrachtet. Entstandene neue *Äste* stehen für den nächsten Durchlauf in der *Extract*-Phase zur Verfügung. Ein Beispiel für das Ergebnis des *Branch-and-Prune-Verfahrens* ist in Abbildung 5.2 dargestellt.

Die praktische Effizienz des *Branch-and Prune-Verfahrens* für ICSP hängt dabei vor allem von der schnellst möglichen Verkleinerung der Intervalle ab (vgl. Hansen und Walster, 2003, Abschnitt 4.2.2). Dafür sind speziell drei Faktoren maßgebend:

Auswahl des nächsten Asts: Bei *Optimierungsproblemen* kann in der *Extract*-Phase mit Hilfe einer Heuristik gearbeitet werden. Dabei werden bestimmte *Äste* höher priorisiert, da sie erfolgsversprechender sind.

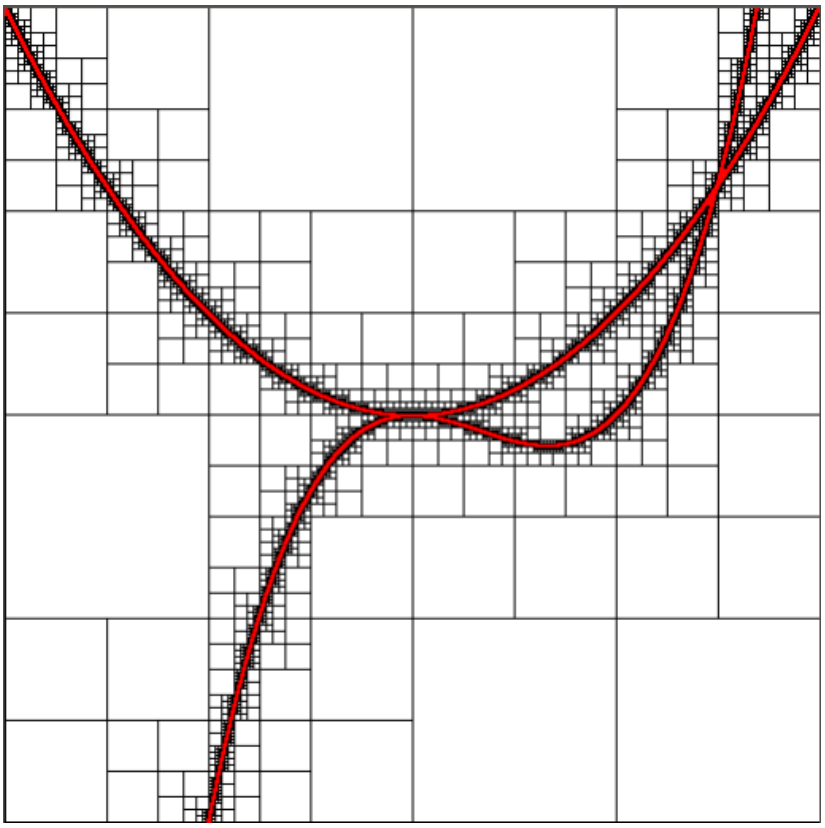


Abbildung 5.2: Beispielhafte Visualisierung der *Einzellösungen* (rot) und der Zerlegung in *Boxen* (schwarze Quadrate) durch ein *Branch-and-Prune-Verfahren* (Entnommen aus Moa, 2007, Abbildung 9.4)

Auswahl der Split-Variablen: Eine *Variable* kann mehr oder weniger zur Einschränkung anderer *Variablen* führen oder die *Nicht-Existenz-Tests* beeinflussen.

Effiziente Pruning-Operatoren: Der bzw. die verwendeten *Pruning-Operatoren* können den zu durchsuchenden Bereich stark oder weniger stark einschränken.

Es existiert kein Patentrezept, welche Strategie für die einzelnen Faktoren die höchste Effizienz bietet. Dies kann individuell vom untersuchten *Constraint-System* abhängen. Es existieren aber durchaus Strategien, die in einem Großteil der Fälle deutlich bessere Ergebnisse liefern als andere.

Es existieren verschiedene Forschungen, die das *Branch-and-Prune-Verfahren* für ICSP beschreiben, genauer untersuchen und verschiedene Strategien vergleichen (Ishii u. a., 2012, vgl. Araya u. a., 2016; Kjøller u. a., 2007; Vu, Silaghi u. a., 2006; Granvilliers, 2001; Beelitz, 2006). Umfangreiche Forschungen wurden von Bartłomiej Jacek Kubica durchgeführt. Er veröffentlichte für die Suche nach *Einzellösungen* in ICSP verschiedene Beiträge (siehe Kubica, 2011; Kubica, 2015) und ein Buch (siehe Kubica, 2019). Dabei nutzt er

einen generischen *Branch-and-Prune-Algorithmus* und untersucht verschiedene Strategien bei der Auswahl der *Split-Variablen* und der *Pruning-Operatoren* und entwickelt diese weiter. Er identifiziert und diskutiert außerdem Potentiale der Parallelisierung (Kubica, 2009; Kubica, 2017). Daneben existieren Ansätze, die Berechnungen beschleunigt auf Grafikkartenkernen (GPU) durchführen zu lassen (vgl. Lin u. a., 2018).

Speziell die Auswahl der zu splittenden *Variablen* für das *Bisektion-Verfahren* wurde von Beelitz (2006) und Kubica (2015) genauer untersucht. Beide stellen fest, dass die einseitige Festlegung auf eine der Grundstrategien in bestimmten Beispielen in sehr langen Laufzeiten resultieren. Eine Auswahl an Grundstrategien für die *Bisektion* ist:

MaxDiam (deutsch: größter Durchmesser): Es wird die *Variable* mit der größten Intervallbreite gewählt (vgl. Neumaier, 2004, Seite 26).

MaxSmear (deutsch: größte Verschmierung): Es werden alle Funktionen betrachtet. Es wird die *Variable* mit der größten Intervallbreite in der 1. Ableitung gewählt.

MaxSumMagnitude Es wird die *Variable* gewählt, deren Summe der partiellen Ableitungen (entsprechende Koeffizienten der Jacobi-Matrix) multipliziert mit der Intervallbreite den größten Wert liefert (vgl. Hansen und Walster, 2003, Abschnitt 11.8).

Für eine im Allgemeinen bessere *Bisektions-Strategie* werden hybride Strategien vorgeschlagen. Beelitz (2006, Abschnitt 2.4.6) verwendet für seine hybride Strategie verschiedene Kriterien, wie Intervallbreite, Ableitungen und Ursprungs-Symmetrien der Intervalle. Diese Kriterien entscheiden über die Wahl der entsprechenden Grundstrategie. Kubica (2015) setzt ebenfalls auf solche Kriterien, berücksichtigt aber auch die *Bestimmtheit* des *Constraint-Systems* und die von dem vorhergehenden Schritt des *Intervall-Newton-Verfahrens* nicht veränderten *Variablen*.

Weitere Strategien berücksichtigen, wie häufig *Variablen* für einen Split ausgewählt werden. So schlagen Ishii u. a. (2012) vor, für *unterbestimmte Intervall-Constraint-Systeme* alle *Variablen* alternierend zu splitten. Bestimmte *Variablen* sollen dabei höher, andere niedriger priorisiert werden.

5.4.6 Iterative Suchverfahren bei differenzierbaren Funktionen

Grundlagen

Bei den iterativen Suchverfahren ist das Ziel, eine oder mehrere Lösungen für ein ICSP zu finden. Dies kann für den linearen, wie auch für den nichtlinearen Fall als $m \times n$

Gleichungssystem $\mathbf{M} \cdot \mathbf{X} = \mathbf{0}$ betrachtet werden. Die n *Variablen* in $\mathbf{X}$ sind dabei die Unbekannten, stellen aber auch gleichzeitig die initiale *Box* dar, welche die *Variablen* im Gleichungssystem beschränkt. Jede einzelne der m Gleichungen repräsentiert dabei eine *Constraint*.

Zum Lösen des Gleichungssystems kann theoretisch ein konventioneller Gleichungslöser genutzt werden. Das Gauß'sche Eliminationsverfahren kommt dabei nicht in Frage, da das Runden nach außen und der Einfluss des *Intervall-Abhängigkeits-Problems* (siehe Abschnitt 5.3.3) zu unbefriedigenden Ergebnissen führt. Die Ergebnisse sind stark überschätzt und damit ineffizient (vgl. Hansen und Walster, 2003, Kapitel 5.5).

Zur Lösung des Problems wurden verschiedene Verfahren entwickelt, die davon ausgehen, dass alle Funktionen stetig und differenzierbar sind. Diese Verfahren haben gemeinsam, dass sie als Projektionsfunktion $N(X_t)$ betrachtet werden können. Diese Projektionsfunktion ist iterativ anzuwenden $X_{t+1} = X_t \cap N(X_t)$ mit $t \in 0, 1, 2, ...$ bis ein Terminierungskriterium erreicht wird.

Daneben ist üblicherweise auch ein *Existenz-Test* möglich. Dieser unterscheidet drei Fälle (vgl. Neumaier, 1990, Kapitel 5.1):

1. Wenn $X_{t+1} \subseteq X_t$, dann enthält X_t bzw. X_{t+1} genau eine Lösung.
2. Wenn $X_{t+1} \cap X_t = \emptyset$, dann ist keine Lösung enthalten.
3. In anderen Fällen ist keine Aussage über die Existenz einer Lösung möglich.

Zur weiteren Näherung reellwertiger Lösungen oder im Fall 3, kann ein solches Lösungsverfahren in einen *Branch-and-Prune-Algorithmus* eingebunden werden.

Als gebräuchlichstes Verfahren hat sich das *Intervall-Newton-Verfahren* erwiesen, welches effizient und schnell ist (vgl. Hansen und Greenberg, 1983). Es stellt die Erweiterung des bekannten *Newton-Raphson-Verfahrens* dar, einem iterativen Näherungsverfahren. Im eindimensionalen, *univariaten* Fall lautet die Rechenvorschrift:

$$N_t(X, F) = X_t - \frac{F(X_t)}{F'(X_t)} \tag{5.16}$$

$$X_{t+1} = X_t \cap N_t(X, F) \tag{5.17}$$

Die Idee des *Intervall-Newton-Verfahrens* kann im eindimensionalen Fall folgendermaßen erklärt werden: Es wird die 1. Ableitung $F'(X_t)$ mit Hilfe der *Intervallarithmetik* bestimmt. Durch die *Regeln* der *Intervallarithmetik* ist sichergestellt, dass die obere und untere Grenze von $F'(X_t)$ die minimal und maximal mögliche Ableitung in dem Intervall umschließen. Üblicherweise wird für den Funktionswert $F(X_t)$ nicht das gesamte

Intervall verwendet, sondern nur ein reeller Wert – üblicherweise die Mitte $mid(X_t)$. Der Funktionswert an der Mitte $F(mid(X_t))$ ist ein Stützwert im Intervall X_t, von dem aus die obere und die untere Grenze der 1. Ableitung alle möglichen Werte für Nullstellen $(F(x) = 0)$ umschließt, welche die Funktion $F(X_t)$ im Intervall X_t besitzen kann. Es kann anschließend der Schnitt aller umschlossener Werte mit X_t gebildet werden. Dieser muss alle potentiell möglichen Lösungen enthalten. Eine geometrische Darstellung des Prinzips ist in Abbildung 5.3 dargestellt.

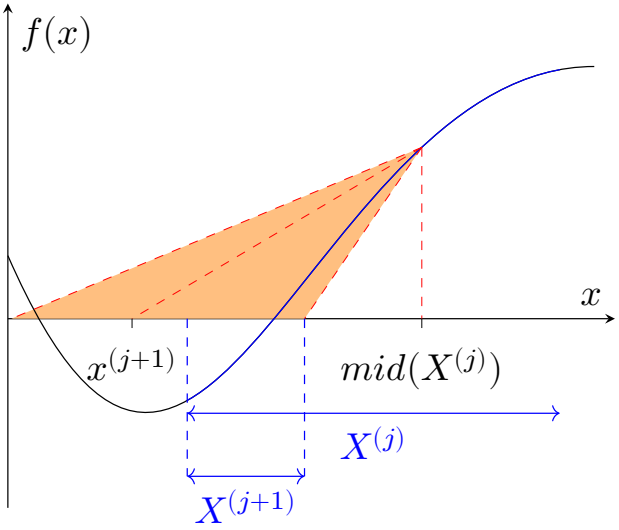


Abbildung 5.3: Prinzip eines Iterationsschritts des *Intervall-Newton-Verfahrens* für eine Funktion $f(x)$ im Intervall $X_{(j)}$ (blau). Das Ergebnis ist $X_{(j+1)}$ (mit *Intervallarithmetik*), $x_{(j+1)}$ (ohne *Intervallarithmetik*) (basierend auf Sainudiin u. a., 2005, Abbildung 20.3).

Im eindimensionalen Fall lässt sich die Berechnung des Bereichs der umschlossenen Werte der X-Achse mit einer Division durchführen. Dabei kann auf die *erweiterte Intervallarithmetik* zurückgegriffen werden, damit 1. Ableitungen, die 0 enthalten, zu zwei disjunkten Bereichen führen.

Die Generalisierung der Methode als mehrdimensionaler, *multivariater* Fall ist folgendermaßen definiert:

$$\mathbf{N}_t(\mathbf{X}_t, \mathbf{F}) = \mathbf{X}_t - \mathbf{Jacobi}(\mathbf{X}_t)^{-1} \cdot \mathbf{F}(\mathbf{X}_t) \tag{5.18}$$

$$\mathbf{X}_{t+1} = \mathbf{X}_t \cap \mathbf{N}_t \tag{5.19}$$

Dabei ergibt sich das Problem, dass die Inverse der Jacobi-Matrix – wie das Gauß'sche Eliminationsverfahren – aufgrund der Überschätzung ein nicht geeignetes Ergebnis liefert. Aus diesem Grund existieren Verfahren, die darauf verzichten, die Jacobi-Matrix

mit der *Intervallarithmetik* zu bestimmen. Zur Lösung kann entweder die *Krawczyk-Iteration* (vgl. Krawczyk, 1969), bei der eine reellwertige Jacobi-Matrix invertiert werden muss, die Iterationen nach dem *Gauß-Seidel-Verfahren* oder das *komponentenweise Berechnungsverfahren* genutzt werden. Dabei sind Algorithmen nach dem Prinzip des *Gauß-Seidel-Verfahrens* der *Krawczyk-Iteration* insofern überlegen, als dass sie im Allgemeinen schmalere Intervalle bzw. kleinere *Boxen* liefern (vgl. Hansen und Walster, 2003, Kapitel 11.3; Neumaier, 1990, Seite 138; Hansen und Greenberg, 1983). Auch der *Existenz-Test* einer Lösung führt bei reinen *Intervall-Newton-Verfahren*, wie beispielsweise dem *Gauß-Seidel-Verfahren*, zu klareren Aussagen. Goldsztejn (2007) veranschaulicht dies mit einem Vergleich bekannter Verfahren. Aufgrund der Überlegenheit des *Gauß-Seidel-Verfahrens* und des *komponentenweisen Berechnungsverfahren*, werden ausschließlich diese hier vorgestellt.

Gauß-Seidel mit Präkonditionierung

Das *Gauß-Seidel-Verfahren* für Intervalle wird für ein Gleichungssystem $n \times n$ wie folgt definiert:

$$i \in 1,...,n \text{ und } t \in 0,1,2,...$$
$$X_i^{t+1} = -\left[\sum_{j=1}^{i-1} M_{i,j} X_j^{t+1} - \sum_{j=i+1}^{n} M_{i,j} X_j^t\right] / M_{i,i} \tag{5.20}$$

Dabei wird eine Linearisierung unter Berücksichtigung aller *Variablen* durchgeführt.

Es existieren verschiedene Varianten, welche zusätzlich eine Präkonditionierung vornehmen. Deren Zweck ist es, möglichst schmale Intervalle bzw. kleine *Boxen* zu liefern. Als effiziente Konditionierungsmatrix hat sich die Inverse der Jacobi-Matrix der Intervallmitten etabliert (vgl. Kearfott, 2013, Kapitel 1.2.2):

$$\mathbf{Jacobi}(mid(\mathbf{X}))^{-1} \tag{5.21}$$

Für die Präkonditionierung eines nicht quadratischen, also eines *über-* ($n < m$) oder *unterbestimmten* ($n > m$) Systems, kann eine Teilmatrix genutzt werden. In diesem Fall werden entweder maximal so viele Gleichungen wie *Variablen* genutzt oder maximal so viele *Variablen* wie Gleichungen. Daneben ist es möglich, eine Pseudoinverse, wie beispielsweise die *Moore-Penrose-Inverse*, für die Konditionierung einzusetzen.

Komponentenweise Berechnung

Alternativ zum *Gauß-Seidel-Verfahren* mit Präkonditionierung lässt sich das *Intervall-Newton-Verfahren* auch *komponentenweise* durchführen. Es wird bei dieser Rechenvorschrift auf jede Gleichung und jede dort enthaltene *Variable* ein eindimensionaler *Intervall-Newton-Rechenschritt* $N_{cmp}(\mathbf{X}, i, j)$ durchgeführt (vgl. Herbort u. a., 1997). Das bedeutet, dass eine Linearisierung nur bezüglich einer *Variablen* gleichzeitig stattfindet. Dabei wird die i-te Komponente der *Box* $\mathbf{X}$ mit der j-ten Funktion des Gleichungssystems kombiniert:

$$N_{cmp}(\mathbf{X}, i, j) := mid(X_j) - \frac{F_i(X_1, \ldots, X_{j-1}, mid(X_j), X_{j+1}, \ldots X_n)}{\frac{\partial F_i}{\partial X_j}(X_1, \ldots, X_n)} \qquad \begin{matrix} \\ i \in 1, \ldots, m;\ j \in 1, \ldots, n \end{matrix} \tag{5.22}$$

Die Wahl der Paare (i,j) und die Reihenfolge der Paare ist dafür entscheidend, wie effizient der Algorithmus das System löst. Die einfachste Heuristik nutzt dafür alle Paare für die der Koeffizient der Jacobi-Matrix $\frac{\partial F_i}{\partial X_j} \neq 0$ ist. Das *komponentenweise Berechnungsverfahren* erfordert keine Präkonditionierung und ist für *unter-*, *wohl-* und *überbestimmte* Gleichungssysteme ohne zusätzlichen Aufwand nutzbar. Herbort u. a. (1997) beschreiben darüber hinaus die Möglichkeit, das Verfahren zu verbessern, da beispielsweise bei Gleichungssystemen, die nahezu linear sind, eine schlechte Einschränkung der *Box* auftritt. Sie schlagen eine Kombination mit einem *Gauß-Seidel-Verfahren* vor.

Kubica (2011) vergleicht für *unterbestimmte* Gleichungssysteme ($n > m$) verschiedene Lösungsverfahren. Dabei wird die Aussage gemacht, dass der *komponentenweise* Rechenschritt bei großen *Boxen*, sprich großen Intervallbreiten, deutlich effizienter ist, während das *Gauß-Seidel-Verfahren* mit invertierter Mittelpunktsmatrix zur Präkonditionierung, bei kleinen *Boxen* eine höhere Effizienz aufweist, und damit eine bessere Reduzierung liefert.

Zur Kombination beider Verfahren stellt Kubica (2015) eine Heuristik vor, bei der die Auswahl des Suchverfahrens von den Breiten der Intervalle der *Such-Box*, der Anzahl der *Variablen* n und der Zahl der Gleichungen m abhängt.

5.4.7 Algorithmen zur Berechnung der Hüllen-Konsistenz

Es existieren verschiedene Algorithmen zur Berechnung der Hülle bezogen auf eine *Constraint* oder ein *Constraint-System*. Im Folgende werden die effizientesten Verfahren zur Bestimmung der Hülle vorgestellt.

3B: Zur Berechnung der *3B-Konsistenz* kann der *3B*-Algorithmus genutzt werden (vgl. Lhomme, 1993). Dabei wird mit Hilfe des *Bisektion-Verfahrens* das Intervall einer *Variablen* so lange verkleinert, bis für jede Grenze ein *Fixpunkt* erreicht wird, an dem alle *Constraints* erfüllt sind.

HC4Revise: Benhamou, Goualard u. a. (1999) beschreiben einen Algorithmus zur Berechnung der Hülle einer einzelnen *Constraint* bzw. einer Gleichung. Dabei wird die Gleichung in einen binären Berechnungsbaum zerlegt, in dem die Blätter *Variablen* und Konstanten und die Knoten die mathematischen Operatoren enthalten. Am Wurzelknoten befindet sich das Gleichheitszeichen der Gleichung. Die Berechnung wird in zwei Phasen unterteilt:

1. Vorwärts-Rechnungs-Phase (englisch: forward evaluation phase): Beginnend mit den Blättern des Berechnungsbaums wird eine *Intervall-Propagation* aufsteigend bis zur Wurzel durchgeführt. Dabei wird an Knoten und an der Wurzel die Schnittmenge der Ergebnisse der Teilbäume gebildet.
2. Rückwärts-Rechnungs-Phase (englisch: backward evaluation phase): Das Ergebnis der Schnittmenge an der Wurzel aus der Vorwärts-Rechnungs-Phase wird in die Teilbäume in Richtung der Blätter propagiert.

Der Algorithmus ist sehr effizient und liefert bei einmaligem Auftreten von *Variablen* in der Gleichung die *Hüllen-Konsistenz*. Sollte die *erweiterte Intervallarithmetik* genutzt werden (siehe Abschnitt 5.3.2), so ist das Ergebnis die Gesamtlösung. Sind *Variablen* mehrfach vorhanden, wird das Ergebnis überschätzt.

HC4: Wiederholt so lange den *HC4Revise*-Algorithmus für alle *Constraints*, bis ein *Fixpunkt* erreicht wird und keine Verkleinerung der Intervalle der *Variablen* des ICS mehr stattfindet (Benhamou, Goualard u. a., 1999). Er ist wie der *HC4Revise*-Algorithmus vom *Intervall-Abhängigkeits-Problem* betroffen.

FBPD: Vu, Schichl u. a. (2009) beschreiben eine Erweiterung des *HC4*, wobei die Baumstruktur des *HC4* bzw. *HC4Revise* durch einen gerichteten, azyklischen Graphen (englisch: directed acyclic graph; Abkürzung: DAG) ersetzt wird. Dies ermöglicht es, das multiple Auftreten von *Variablen* besser zu berücksichtigen, indem der Graph geschickt in Teilgraphen zerlegt wird und gezielt Knoten für das Traversieren bei der Berechnung ausgewählt werden.

Globale-Hüllen-Konsistenz nach Cruz: Cruz (2003) beschreibt einen Algorithmus, mit dem sich für alle *Variablen* des ICS die vollständige globale Hülle bestimmen

lässt. Es wird dabei immer die aktuelle, anfangs leere Hülle vorgehalten, welche beim Finden von Lösungen entsprechend vergrößert wird. Dabei wird aus Effizienzgründen eine Baumdatenstruktur verwendet, in der jeder Knoten eine *Box* des möglichen Teilraums repräsentiert. Seine Kindknoten enthalten dabei wiederum *Boxen* mit den möglichen Teilräumen des Elternknotens. Für jede *Variable* wird bestimmt, ob ein *Blatt* bzw. dessen *Box* zu einer Vergrößerung der bisherigen Hülle führen kann, wenn es denn eine Lösung enthält. Nun werden für jede in Frage kommende Kombination aus *Variable* und *Blatt* drei Phasen durchlaufen:

1. Beschneiden (englisch: Prune): Es wird versucht, ein *Blatt* bzw. seine *Box* zu verkleinern, dazu kann beispielsweise *HC4* genutzt werden. Blätter ohne potentielle Lösung werden entfernt. Blätter mit potentieller Lösung werden in der Suche-Phase weiterverarbeitet.
2. Suche (englisch: Search): Es wird nach einer Lösung im *Blatt* bzw. seiner *Box* gesucht. Dafür wird ein lokaler Such-Algorithmus verwendet, der eine Lösung liefert, die für die betrachtete *Variable* ein möglichst großes bzw. kleines Ergebnis liefert. Es kann aber auch ein beliebiger anderer Such-Algorithmus verwendet werden.
3. Split: Sollte der Such-Algorithmus keine Lösung finden können, da er nicht vollständig ist – nicht mit Sicherheit den Suchraum ausschließen können –, wird das *Blatt* bzw. dessen *Box* gesplittet, und es entstehen zwei neue Blätter als Kindknoten. Diese werden in der Beschneiden-Phase weiterverarbeitet. Sollte der Such-Algorithmus vollständig sein, und es wird keine Lösung gefunden, so kann das *Blatt* entfernt werden.

Der Algorithmus endet, wenn keine Blätter mehr existieren, die potentielle Lösungen enthalten können, oder wenn nur noch Blätter existieren, die die Hülle einzelner *Variablen* potentiell nicht mehr vergrößern können. Der Aufwand ist trotz effizienter Datenstruktur als sehr hoch einzuschätzen.

Die *globale Hüllen-Konsistenz* nach Cruz (2003) ist nach Wissen des Autors dieser Arbeit die einzige Beschreibung eines Algorithmus, der die *globale Hülle* vollständig und trotz des hohen Aufwands so effizient wie möglich berechnet.

5.4.8 Algorithmen zur Berechnung der Box-Konsistenz

Für die Bestimmung der *Box-Konsistenz* finden die folgenden Algorithmen häufig Anwendung:

BC3: Benhamou, McAllester u. a. (1994) beschreiben einen Algorithmus, der mit dem *Bisektions-Verfahren* die untere und obere Grenze der Intervalle bestimmen soll. Dabei wird für jede *Variable* der Wertebereich des Intervalls mit Hilfe des *Intervall-Newton-Verfahrens* und einem anschließenden Split so weit reduziert, dass die Grenzen für alle *Constraints* eine sogenannte *Pseudonull* liefern. Das bedeutet, dass es für die einzelnen *Constraints* eine Nullstelle ist, aber nicht zwangsweise für das gesamte Gleichungssystem.

BC4: Benhamou, Goualard u. a. (1999) nutzt den Algorithmus *HC4* und erweitert ihn um *BC3*. Dadurch wird die Effizienz von *HC4* mit der Möglichkeit von *BC3*, das multiple Auftreten von *Variablen* zu berücksichtigen, kombiniert.

5.5 Optimierungsverfahren

Es existieren Algorithmen für das Finden globaler Minima- bzw. Maxima (Optimierung) von Funktionen oder der Berechnung von Lösungen (Suche) eines ICSP. Üblicherweise werden Optimierungen als das Finden eines Minimums definiert. Dabei ist ein globales Optimum der Funktionswert $\mathbf{W}$ einer Zielfunktion $\mathbf{F}$, so dass $\mathbf{W} \leq \mathbf{F}(\mathbf{X})$, wobei (X) jede mögliche *Einzellösung* eines ICSP darstellt (vgl. Moore u. a., 2009, Seite 165).

Bei Such- und Optimierungsverfahren für ICS mit nichtlinearen *Constraints* wird auf Filterverfahren gesetzt. Dabei werden *Boxen* herausgefiltert, die keine Relevanz für das Ziel des Verfahrens besitzen. Dies wird als *Methode der oberen Begrenzung* (englisch: upper bounding) bezeichnet (vgl. Araya u. a., 2016). Die Methode folgt dem Prinzip des *Branch-and-Prune*.

Die obere Grenze aller relevanter *Äste* muss bessere bzw. mindestens so gute Werte der Zielfunktion liefern, wie die bisher gefundene beste Lösung. Das bedeutet zum einen, dass so schnell wie möglich eine Lösung gefunden werden sollte, mit welcher der Suchraum begrenzt werden kann, zum anderen, dass in jedem Fall, wenn eine bessere Lösung gefunden wird, die bisher als beste betrachteten Lösungen verworfen werden (vgl. Neumaier, 2004, Seite 35; Rump, 2018). In dieser Arbeit wird ein Algorithmus, der diese Methode umsetzt, in Abschnitt 7.5.4 detailliert beschrieben.

Zur Beschleunigung des Verfahrens können verschiedene Strategien eingesetzt werden. Zum einen kann in der *Extract*-Phase des *Branch-and-Prune-Verfahrens* die Auswahl der nächsten *Box* bzw. des nächsten *Asts* beeinflusst werden. Messine (2004) schlägt beispielsweise vor, immer die *Box* zu wählen, für welche die Zielfunktion den Funktionswert mit der kleinsten unteren Grenze annimmt. Dies unterliegt der Annahme, dass diese *Box* wahrscheinlicher das Optimum liefern kann, als die anderen. Es ist nicht auszuschließen,

dass diese Annahme nicht zutrifft, da die *Box* nach weiteren Durchläufen keine Lösung oder eine schlechte Lösung liefert. Der Grund dafür ist, dass die beste enthaltene *Einzellösung* eine größere untere Grenze besitzt. Zum anderen kann die Suche nach dem Optimum beschleunigt werden, indem so früh wie möglich eine Lösung gefunden wird. Dazu kann beispielsweise der Mittelwert der aktuellen *Box* darauf geprüft werden, ob er eine *Einzellösung* darstellt (vgl. Messine, 2004). Ist dies der Fall, kann diese Lösung berücksichtigt werden und gegebenenfalls einen neuen Minimalwert der Zielfunktion liefern.

5.6 Eignung und Aufwand der Algorithmen

Die Berechnung der möglichen Wertebereiche von *Variablen* – äquivalent zum Berechnen der Wertebereiche von *Parametern* – und die Suche nach Lösungen unter einer Zielfunktion machen das *Constraint-basierte Modellieren* mit Intervallen zu einem vielversprechenden Ansatz, die Ziele dieser Arbeit zu erreichen. Es steht mit dem *Branch-And-Prune-Verfahren* ein flexibles Konzept zur Verfügung, welches gut parallelisierbar und einfach umgesetzt werden kann.

Die Ziele dieser Arbeit beinhalten die Berechnung der aller gültigen Werte von *Parametern*, welche vollständig mit der Bestimmung der *(n+1)-Konsistenz*. Die Umsetzung ist theoretisch möglich, aber nicht praktikabel. Sollen alle möglichen Werte eines Parameters berechnet werden, kann das Ergebnis aus einer großen Anzahl an disjunkten Teilmengen bestehen. Da durch das Auftreten der Lücken ein stark wachsender kombinatorischer Aufwand entsteht (vgl. Hyvönen, 1989; Cruz, 2003, Seite 69) ist dies, auf Grund der langen Laufzeiten für komplexe ICS, nicht umsetzbar. Eine sinnvolle Möglichkeit, den kombinatorischen Aufwand zu reduzieren und trotzdem ein brauchbares Ergebnis zu erreichen, ist die Berechnung der *Hüllen-Konsistenz* (Lhomme, 1993). Dadurch kann der explodierende kombinatorische Aufwand vermieden werden.

Lhomme (1993) benennt die Suche nach der *globalen Konsistenz* als ein *NP-schweres* Problem, solange eine feste Präzision genutzt wird. Mit der *(n+1)B-Konsistenz* reduziert sich der Aufwand, ist aber von folgenden *Parametern* des ICSP abhängig:

- n: Anzahl der *Variablen*
- m: Anzahl der *Constraints*
- d: Anzahl der möglichen reellen Werte im größten Intervall

Bordeaux, Monfroy u. a. (2001) beziffern den Aufwand bezüglich der Berechnungsschritte für einen von ihnen entwickelten Algorithmus für die *kB-Konsistenz* auf:

$$\begin{array}{ll} (2k')\text{B} & \mathcal{O}(md(d^2n^3)^{k'-1}) \\ (2k'+1)\text{B} & \mathcal{O}(md^2n(d^2n^3)^{k'-1}) \end{array} \text{ mit } k' \geq 1 \tag{5.23}$$

Damit liegt der Aufwand für die *(n+1)B-Konsistenz* bei:

$$\begin{array}{ll} \text{wenn } n \text{ gerade, dann} & \mathcal{O}(md(d^2n^3)^{\frac{n}{2}-1}) \\ \text{wenn } n \text{ ungerade, dann} & \mathcal{O}(md^2n(d^2n^3)^{\frac{n}{2}-1}) \end{array} \tag{5.24}$$

Die praktische Laufzeit kann sich in bestimmten Fällen deutlich verschlechtern, wenn die Algorithmen nur langsam zu einem *Fixpunkt* konvergieren. Es existieren Fälle in denen zusätzliche Abbruchkriterien bei Algorithmen der Intervall-Propagation notwendig sind, da sie sehr langsam konvergieren (vgl. Bordeaux, Hamadi u. a., 2007). Dies trifft auch für iterative Suchverfahren bei differenzierbaren Funktionen zu (vgl. Csallner u. a., 1996). Diese Probleme sind zu berücksichtigen, ihre Auswirkungen können aber durch zusätzliche Terminierungskriterien begrenzt werden.

Für den Sonderfall, dass der *Constraint-Graph* des ICS einem Baum entspricht, ergeben sich aus den fehlenden Zyklen und den nicht mehrfach auftretenden *Variablen* folgende Eigenschaften (vgl. Kirchner und Huhnt, 2018):

- Es ist exakt lösbar, wenn eine exakte *Intervallarithmetik* verwendet wird (Kearfott, 2013, Seite 14).
- Es gilt, wie für alle *Constraint-Systeme* dieser Form: Es ist in linearer Zeit lösbar (vgl. Freuder, 1982).

Für alle anderen ICS ist mit einem deutlich höheren Aufwand zu rechnen, da das ICSP nicht in polynomialer Zeit lösbar ist (vgl. Freuder, 1982).

Die Möglichkeiten der *globalen Hüllen-Konsistenz* werden in dieser Arbeit als ausreichend betrachtet, da sie zwei wichtige Aussagen möglich macht:

- Für jede *Variable* – bzw. jeden *Parameter* – enthält die Hülle mit der unteren und oberen Grenze ein bis zwei (eine, wenn es sich um ein degeneriertes Intervall handelt) Werte, für die eine *Einzellösung* existiert.
- Alle weiteren gültigen Werte müssen innerhalb der Hülle des Intervalls liegen.

Diese Möglichkeit, kann mit Hilfe von Optimierungsalgorithmen angewendet auf *Intervall-Constraint-Systeme* zur gezielten Berechnung von Varianten und beabsichtigten Lösungen genutzt werden.

5.7 Softwarebibliotheken für Intervalle

Für Intervalle existieren verschiedene Softwarebibliotheken, die vornehmlich quelloffen sind. Für die Implementierung von *Intervallarithmetik* in C++ sind dies beispielsweise *GAOL*[1], *PROFIL/BIAS*[2] oder *FILIB*[3]. Auch in der Programmiersprache *Julia*[4] existiert eine solche Implementierung[5]. Algorithmen für die Optimierung und das Lösen von *Constraint-Systemen* nutzen die genannten Bibliotheken oder besitzen eigene Implementierungen. Softwarebibliotheken in diesem Bereich sind unter anderem *INTLAB*[6] für *Matlab*, *IBEX*[7] in C++ und *Realpaver*[8] in *C*. Die existierenden Bibliotheken für *Julia*[9] [10] besitzen gegenüber den vorher genannten einen deutlich geringeren Funktionsumfang.

5.8 Vergleichbare Arbeiten und Entwicklungsbedarf

Modellierung und Berechnungen auf Basis von Intervallen stellen einen Forschungsbereich dar, der bereits in voller Breite untersucht wurde. Granvilliers u. a. (2004) nennen verschiedene Problemklassen, die mit Intervallen und den zugehörigen Algorithmen bearbeitet werden können. Diese sind:

- Modellierung von Unsicherheit, zum Beispiel bei Messungenauigkeiten
- Rundungsfehler bei arithmetischen Operationen im Rechner
- Mengen-Modellierung von reellen Zahlen durch logische Schlussfolgerungen
- Robuste numerische Algorithmen für Integration, Gleichungslöser, Differentiation oder globale Optimierung

[1] https://gitlab.univ-nantes.fr/goualard-f/gaol (abgerufen am 22.09.2019)
[2] http://www.ti3.tu-harburg.de/keil/profil/index_e.html (abgerufen am 22.09.2019)
[3] http://www2.math.uni-wuppertal.de/wrswt/software/filib.html (abgerufen am 22.09.2019)
[4] https://julialang.org/ (abgerufen am 22.09.2019)
[5] https://github.com/JuliaIntervals/IntervalArithmetic.jl (abgerufen am 22.09.2019)
[6] http://www.ti3.tu-harburg.de/rump/intlab/ (abgerufen am 22.09.2019)
[7] http://www.ibex-lib.org/ (abgerufen am 22.09.2019)
[8] http://pagesperso.lina.univ-nantes.fr/~granvilliers-l/realpaver/ (abgerufen am 22.09.2019)
[9] https://github.com/JuliaIntervals/IntervalConstraintProgramming.jl (abgerufen am 22.09.2019)
[10] https://github.com/JuliaIntervals/IntervalOptimisation.jl (abgerufen am 22.09.2019)

Die globale Optimierung mit Intervallen wird dabei beispielsweise in den Gebieten der künstlichen Intelligenz, den Modellen der Wasserverteilungsnetze oder den Differentialgleichungen der Strukturmechanik, untersucht (vgl. Moore u. a., 2009, Kapitel 11).

Intervalle eignen sich ebenfalls als Grundlage des *Mengen-basierten Entwurfs* (siehe Abschnitt 3.11). Es stellt sich die Frage, welche Anstrengungen bisher unternommen wurden, ein ICS aus einem *geometrischen Constraint-System* abzuleiten und darauf aufbauend Modelle zu entwickeln, die sinnvoll ausgewertet werden können.

Nahm u. a. (2006) präsentieren den Prototyp eines CAD-System, in dem geometrische und nicht-geometrische Daten modelliert werden können. Sie identifizieren dabei die unzureichende Unterstützung durch vorhandene CAD-Software in frühen Entwurfsphasen, in denen noch große Unsicherheiten im Entwurf bestehen. Die entwickelte Software entspricht einem *Expertensystem*, in dem *Constraints* und *Parameter* integriert sind. Sie nutzen dafür Intervalle zur Beschreibung der Wertebereiche von *Parametern*. Der von ihnen verwendete Algorithmus der *Intervall-Propagation* wird nicht genau beschrieben, basiert aber laut den Autoren auf dem von Finch u. a. (1996). Dieser stellt eine große Einschränkung dar, da damit Gleichungen nicht simultan lösbar sind.

Wang u. a. (2007) stellen bereits die Möglichkeit des *Constraint-basierten Modellierens* mit Intervallen für *Geometrie-Objekte* vor. Das Ziel ist dabei ebenfalls die Berechnung aller Wertebereiche der Parameter. Sie nutzen dafür ein Lösungsverfahren, welches die nichtlinearen Gleichungen mit abschnittsweiser Linearisierung durch ein *Gauß-Seidel-Verfahren* löst. Dies führt aber nur für bestimmte Systeme zur *globalen Hüllen-Konsistenz*, da es vom *Intervall-Abhängigkeits-Problem* betroffen ist. Im Zuge dieser Arbeit wurde das Beispiel von Wang u. a. (2007) überprüft, und es wurde festgestellt, dass die Lösung bereits mit dem *HC4*-Algorithmus durch reine *Intervall-Constraint-Propagation* erreicht werden kann.

Speziell für die Bestimmung der Wertebereiche von *Parametern* und für Algorithmen zur Optimierung stehen, wie in diesem Kapitel beschrieben, bereits Konzepte zur Verfügung. Auf den vorgestellten Forschungen kann aufgebaut werden. In den weiteren Kapiteln dieser Arbeit soll ein Modell vorgestellt werden, welches unter anderem *Wissen* in Form von Intervallen und *Constraints* berücksichtigen kann. Die Umsetzung erfolgt entsprechend des *Mengen-basierten Entwurfs*. Die Möglichkeiten des sich daraus ergebenden *Intervall-Constraint-Systems* werden anschließend mit ausgewählten Algorithmen gezeigt und bewertet.

KAPITEL 6

Ein Modell der Funktionalen Einheiten und Constraints

In diesem Kapitel wird die *Wissensintegration* in die Geometrie eines Bauwerksmodells vorgestellt. In diesem sollen Entitäten, Geometrien, *Constraints* und *Parameter* mit Intervall-Wertebereichen sinnvoll strukturiert werden. Dies soll, in Verbindung mit den im darauffolgenden Kapitel beschriebenen Algorithmen, ein *Expertensystem* bilden.

6.1 Ziele und Idee des Modells

Die Ziele des Modells sind es, die geometrischen Aspekte eines Bauwerksmodells abzubilden. Damit eine Modellierung gemäß des *Mengen-basierten Entwurfs* (siehe Abschnitt 3.9) erfolgen kann, sollen für alle Werte der *Parameter* Intervalle genutzt werden.

Die grundlegende Strukturierung des Modells erfolgt durch rekursive Zerlegung des Bauwerks in *Modellobjekte*. Daraus ergibt sich eine Hierarchie in Form eines Baums. Das Modell ist im Grundsatz *parametrisch* angelegt. Die dabei genutzten numerischen *Parameter* bilden gleichzeitig die Attribute der einzelnen *Modellobjekte*.

Das Ziel, *Wissen* zu integrieren, kann nur erreicht werden, wenn es strukturiert und erkennbar im Modell hinterlegt werden kann. Dabei ist die Ausprägung bzw. Repräsentation des *Wissens* und seine Integration zu unterscheiden. Die Repräsentation des *Wissens* erfolgt als einzelne Einheiten. Diese umfassen die *Constraints* zur Beschreibung der Zusammenhänge der *Parameter* und der Geometrie, und die zulässigen Wertebereiche der *Parameter*. Die Integration erfolgt auf zwei Ebenen. Während die Einschränkungen der Parameterwerte direkte Auswirkungen auf die betroffenen *Parameter* haben und die

Constraints direkt den einzelnen *Modellobjekte* zugeordnet sind, ist für die Einordnung der *Wissenseinheiten* eine weitere Strukturierung anzulegen.

Die Einschränkung dieses Modells liegt darin, dass sich das *Wissen* nur auf bereits vorhandene Teile beziehen kann. Das heißt, die Topologie der *Geometrie-Objekte* bzw. *Modellobjekte* wird fest vorgegeben und kann sich durch *Wissen* nicht verändern. In der Konsequenz ist es nicht möglich, *Wissen* über die Erzeugung von Objekten zu integrieren. Ob beispielsweise der Querschnitt einer Stütze quadratisch oder kreisförmig ist, kann nicht auf Basis von *Wissen* in diesem Modell entschieden werden.

Im Folgenden werden die einzelnen Teile des Modells und ihr Zusammenspiel vorgestellt. Die eingeführten Teile des Modells werden in Deutsch und Englisch benannt, finden sich in den Abbildungen aber ausschließlich in englischer Sprache wieder.

6.2 Beschreibung der Wissensintegration

6.2.1 Zerlegung in Einheiten

Die *Wissensintegration* stellt ein Kernziel dieser Arbeit dar. Dabei ist zu unterscheiden, welches ingenieurtechnische Wissen in ein Modell integriert werden kann und in welcher Form es sich in dem Modell wiederfindet. Dabei ist die Strukturierung ein wichtiger Punkt, da es nur so möglich ist, die technischen Ideen im Modell zu hinterlegen, sodass sie später nachvollziehbar identifizierbar und zuordenbar sind. Das *Wissen* ergibt sich dabei aus den einzelnen Überlegungen und Entscheidungen der beteiligten Menschen, welche auch im Modell erkennbar sein müssen.

Die Basis des *Wissens* stellen dabei die *Constraints* und die festgelegten gültigen Wertebereiche der *Parameter*, die *Parameterbegrenzung*, dar. Hier sei darauf hingewiesen, dass in anderen Arbeiten auch *Parameterbegrenzungen* als *Constraints* aufgefasst werden. Dies wird in dieser Arbeit vermieden, da sonst Widersprüche mit den Definitionen der *Intervall-Constraints* entstünden, die sich ausschließlich auf die Beschreibung der Zusammenhänge zwischen intervallwertigen *Variablen* beziehen.

Zur Strukturierung und Identifikation soll das *Wissen* in *Wissens-Einheiten* zerlegt werden.

Definition 6.2.1 (Wissens-Einheit)**.** Eine *Wissens-Einheit* (englisch: knowledge unit) repräsentiert ein Stück des *Entwurfswissens* und stellt eine Gruppierung der Entwurfsentscheidung zugrunde gelegten Parameterwerte – als *Parameterbegrenzung* – und benötigten *Constraints* dar, die mit der *Wissens-Einheit* in das Modell integriert werden.

Hierzu sei ein negatives Beispiel gegeben: Es existiert die Idee, zwischen zwei Wänden aus Gründen der Entfluchtung, einen bestimmten Abstand vorzugeben. Dies kann mit Hilfe von *Constraints* in das Modell integriert werden. Es werden aus architektonischen Gründen weitere *Constraints* zwischen den Wänden gesetzt, die ebenfalls den Abstand zwischen den Wänden beeinflussen. Werden diese *Constraints* nicht den einzelnen *Wissens-Einheiten* zugeordnet, ist später der Ursprung und Zweck der *Constraints* nicht mehr nachvollziehbar.

Daraus ergibt sich die Frage, inwieweit *Wissen* im Entwurf in solche Einheiten zerlegt werden kann. Im Folgenden sollen zwei in Frage kommende Konzepte vorgestellt werden, welche aufeinander aufbauen. Dafür wird zuerst der *konstruktive Grundgedanke* eingeführt, damit anschließend darauf aufbauend, die weiteren Randbedingungen integriert werden können.

6.2.2 Konstruktiver Grundgedanke

Der *konstruktive Grundgedanke* beinhaltet die Beantwortung der Fragen:

- Welche Entitäten sind zu erstellen und welche *Parameter* besitzen sie?
- Wie sind diese Entitäten hierarchisch zu strukturieren?
- Welche *Constraints* sind notwendig, damit sie ihre geometrische Grundform und grundlegenden Eigenschaften erhalten?

Das bedeutet: Es ist im Vorfeld festzulegen, wie das Grundgerüst des Bauwerks im Modell abgebildet werden soll. Dazu gehört zum einen die Struktur und zum anderen die jederzeit zu erfüllenden grundlegenden Eigenschaften.

Üblicherweise werden dabei Strukturierungsmöglichkeiten wie Ebenen bzw. Stockwerke, Gebäudeflügel oder Bauabschnitte verwendet, die gleichzeitig hierarchische, aber auch geometrische Einteilungen beinhalten. Die Einteilung in horizontale Ebenen ist auch in aktuellen Softwaresystemen des Hochbaus wie *Autodesk Revit*, *Autodesk Inventor*, *ArchiCAD* von *Graphisoft* oder *Nemetschek Allplan* vorgesehen. Daraus ergibt sich eine Strukturierung als Baum, bei dem ausgehend von einem Wurzelknoten, welcher das Gesamtmodell repräsentiert, das Bauwerk rekursiv in seine einzelnen Teile zerlegt wird. Das heißt, ein Bauteil wird gegebenenfalls in weitere Unterbauteile zerlegt. Die Blätter des Baums stellen die kleinstmögliche Entität in dem Teil des Baums dar. Der Aufbau eines solchen *Strukturbaums* kann teilweise mithilfe konstruktiver Operatoren geschehen, welche durch einmaliges Ausführen wiederholt auftretende Geometrien und *Modellobjekte* erzeugen.

Ein einzelnes Bauteil muss zur Erfüllung der Grundeigenschaften alle *Parameter*, geometrische Formen, Unterbauteile und *Constraints* besitzen, welche die Funktion des Bauteils und seiner Unterbauteile vollständig beschreiben und modelltechnischen Problemen vorbeugen können.

Das bedeutet, dass folgende Anforderungen eingehalten werden müssen:

Korrekte Repräsentation: Dabei müssen sich alle geometrischen Körper entsprechend der Ziele des *Solid Modelings* (Abschnitt 2.5) verhalten.

Korrekte Physik: Entsprechend der Physik darf ein physikalischer Raum nur von einer Materie belegt werden. Auch sind die Gesetze der Schwerkraft zu berücksichtigen. Beispielsweise darf in einem statischen Modell kein Objekt „schweben“ oder andere Objekte mit Materie durchdringen.

Korrekte Grundform: Besitzt das Bauteil eine bestimmte geometrische Form, so ist diese zu erhalten. Ein beabsichtigt quaderförmiges Bauteil bleibt bei allen möglichen Parameterwerten ein quaderförmiges Bauteil.

Korrekte Konstruktionslogik: Die Unterbauteile sind korrekt aufeinander abzustimmen. Bei einem Bauteil Tür ist das Türblatt immer mit dem Scharnier verbunden.

Herstellermaße: Handelt es sich um ein konfektioniertes Bauteil eines konkreten Herstellers, sind die vorgegebenen Maße einzuhalten.

Diese Anforderungen sind der Entität zuzuordnen, für welche sie verwendet werden. Der *Parameter* „Dicke“ einer Innenwand ist also direkt der Wand zuzuordnen und nicht dem übergeordneten Stockwerk. Ist dieser *Parameter* auch für das Stockwerk relevant, da das Stockwert selbst einen *Parameter* zur Steuerung der Dicke aller Innenwände besitzt, so sind diese *Parameter* mit den „Dicke“-*Parametern* der einzelnen Wände über *Constraints* zu verknüpfen bzw. in Zusammenhang zu setzen.

6.2.3 Integration weiterer Randbedingungen

Über den *konstruktiven Grundgedanken* hinaus müssen weitere Randbedingungen integriert werden. Die Randbedingungen unterschieden sich dabei in ihrer Wichtigkeit und ihrer Herkunft.

Als mögliche Ursprünge für weitere Randbedingungen bei Bauprojekten lassen sich beispielsweise folgende identifizieren:

Rechtlicher Rahmen: Die direkte Einhaltung von Gesetzen, Vorschriften und Normen.

Projektanforderungen: Weitere Randbedingungen des Bauherrn oder anderer Planungsbeteiligter. Beispielsweise Raumgrößen oder minimale Wanddicken auf Grund baustatischer Nachweise.

Konstruktive Nebenbedingungen: Ideen des Modellierers, ein sinnvoll konstruiertes Modell umzusetzen. Beispielsweise Deckenhöhen (architektonisch) oder Mindestgrößen für Baustellen-Arbeitsräume (baupraktisch).

Diese Randbedingungen sind sinnvoll zu gruppieren und können in zwei Bereichen integriert werden, zum einen durch die Verwendung weiterer *Constraints* zwischen den bereits im *Konstruktiven Grundgedanken* erstellten Einheiten und deren geometrischen Formen und zum anderen durch die Einschränkung des Wertebereichs von *Parametern*. Diese sind entsprechenden den *Wissens-Einheiten* zuzuordnen.

6.3 Beschreibung des Modells

6.3.1 Ausgeprägte Objektorientierung

Ein Bauwerksmodell zeichnet sich durch eine Strukturierung und Zerlegung aus, beispielsweise in Bauteile und Bereiche. Nur so ist es möglich, die Fülle an Daten ingenieurtechnisch nachvollziehbar aufzubereiten und die Komplexität des Modells zu beherrschen. Grundsätzlich bietet es sich dabei an, eine ausgeprägte Objektorientierung umzusetzen. Dies ermöglicht es, die einzelnen Entitäten klar zu identifizieren. Die Abbildung realer Objekte führt zur Vereinfachung bei der Bearbeitung und Weiterverwendung des Modells (vgl. Sommerville, 2011, Seite 178). Damit wird sichergestellt, dass einzelne Objekte im Inneren funktional vollständig definiert und überprüfbar sind. Das Gesamtmodell ergibt sich anschließend aus dem Zusammenspiel der einzelnen Objekte.

Das hier vorgestellte Konzept unterscheidet für das Modell folgende Objektarten:

- *Parameter*, die auch Attribute von Objekten darstellen können.
- *Constraints* beschreiben die Zusammenhänge.
- *Funktionale Einheiten* repräsentieren die Entitäten und Geometrie als abgeschlossene Einheiten.
- *Wissenseinheiten* fassen *Constraints* und Parameterbegrenzungen strukturiert zusammen.

Die *Wissenseinheiten* besitzen unterschiedliche Ausprägungen und sind genauer zu definieren. Dabei werden Auszüge des Diagramms zur Beschreibung der Zusammenhänge der einzelnen Klassen (siehe Abbildung A.1) verwendet. Die Notation des Diagramms ist dabei an UML-Klassendiagramme angelehnt, wobei sich die Verwendung auf *Klassen* ohne Attribute und Methoden, so wie *Assoziationen*, *Kompositionen*, *Kardinalitäten* und *Vererbung* beschränkt. Anschließend wird auf die sich ergebende Struktur des Modells weiter eingegangen und die für den Aufbau wichtigen *Erzeugungsoperatoren* eingeführt.

6.3.2 Parameter mit Intervallen

Alle *Parameter* besitzen einen Wertebereich in Form eines Intervalls entsprechend Abschnitt 5.1. Dieser Wertebereich kann durch die später eingeführten *Wissenseinheiten* mit ihren *Parameterbegrenzungen* weiter eingeschränkt werden. Wird beispielsweise ein *Parameter* „Höhe des Gebäudes“ eingeführt, so kann dies durch die *Wissenseinheit* „Bauvorschrift Firsthöhe max. 30 m“ auf ein Intervall $(0,30]$ (bei Verwendung der Einheit Meter für alle Berechnungen) begrenzt werden. Dies geschieht durch Bildung der Schnittmenge aller den *Parameter* einschränkenden *Parameterbegrenzungen*.

6.3.3 Constraints

Für das Modell sind unterschiedliche Arten von *Constraints* notwendig. Bei der Verwendung numerischer Lösungsverfahren, wie es in dieser Arbeit beschrieben wird, setzen sich *Constraints* aus einer oder mehreren Gleichungen zusammen. Dies ist notwendig, da ein numerisches Lösungsverfahren, im Gegensatz zu konstruktiven Lösungsverfahren (siehe Abschnitt 4.4), ein Gleichungssystem voraussetzt. Jede *Constraint* besteht dabei aus einer Anzahl an Gleichungen (englisch: Equations), welche verschiedene *Parameter* in Beziehung setzen. Es sei hier darauf hingewiesen, dass bestimmte Ungleichungen mit Hilfe von Gleichungen und Intervallen umgesetzt werden können (vgl. Neumaier, 2004, Seite 12; Kjøller u. a., 2007):

$$a < b \text{ ist als Gleichung } a = (-\infty, 0) + b$$

$$a > b \text{ ist als Gleichung } a = (0, \infty) + b$$

Die *Constraints* des Modells lassen sich dabei drei verschiedenen Kategorien zuordnen:

Einfache Constraint: (englisch: Simple Constraint) Diese setzt *Parameter* in Beziehung, ohne weitere Zusammenhänge.

Geometrie-Constraint: (englisch: Geometry Constraint) Diese beschreibt eine geometrische Zwangsbedingung in Form von Relationen zwischen *Geometrie-Objekten*, wobei auch *Parameter* mit einbezogen werden können.

Höhere Constraint: (englisch: Higher Constraint) Diese beschreibt eine Zwangsbedingung zwischen den höheren Objekten und den noch einzuführenden *Funktionalen Einheiten*, wie zum Beispiel Bauteilen. Beispielsweise kann der Anschluss eines Rohrs an eine Muffe als *Höhere Constraint* modelliert werden.

Der Zusammenhang der *Constraints*, Gleichungen und *Parameter* ist in Abbildung 6.1 dargestellt.

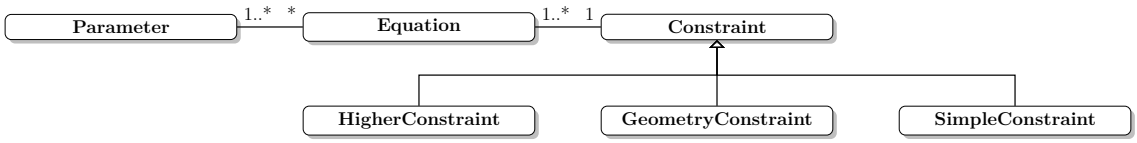


Abbildung 6.1: Auszug aus dem Klassendiagramm: *Constraints*, *Gleichungen* und *Parameter*

6.3.4 Wissenseinheit

Für die sinnvolle Zusammenfassen von Randbedingungen für den Entwurf wird die *Wissenseinheit* (englisch: Knowledge Unit) eingeführt. Die im Zusammenhang stehenden Klassen sind in Abbildung 6.2 dargestellt.

Eine *Wissenseinheit* besteht aus *Constraints* und den *Einschränkungen von Parameterwerten* (englisch: Parameter Domain Limitations), die sinnvoll zusammengefasst werden. Ein *Parameter* kann verschiedenen *Einschränkungen von Parameterwerten* aus verschiedenen *Wissenseinheiten* unterliegen. Alle Punktkoordinaten der Bauteile eines Projekts können beispielsweise durch die Beschränkung aller Koordinaten auf den zu bebauenden Grundstücksbereich eingeschränkt werden. Zusätzlich können einzelne Punktkoordinaten zur Fixierung bestimmter Bauteile bei Anschlüssen an die Nachbarbebauung begrenzt werden.

Es ist möglich, *Wissenseinheiten* zu schachteln, da sich eine *Wissenseinheit* wiederum aus einer beliebigen Anzahl an *Wissenseinheiten* zusammensetzen kann. Der Grund für die Schachtelung wird an folgendem Beispiel deutlich: Die Bedingung, dass ein Fluchtweg mindestens 2 m breit sein muss, wird als eine *Wissenseinheit* „globales Fluchtweg-Mindestmaß“ integriert. Diese *Wissenseinheit* setzt sich wiederum zusammen aus je einer *Wissenseinheit* „lokales Fluchtweg-Mindestmaß“ für ein Paar der Fluchtweg-begrenzenden

Bauteile. Jedes „lokale Fluchtweg-Mindestmaß“ besteht aus einer *Constraint*, die den Abstand a der Fluchtweg zugewandten Seiten zwischen den Bauteilen darstellt und aus einer Einschränkung des Abstands a auf $[2, \infty)$.

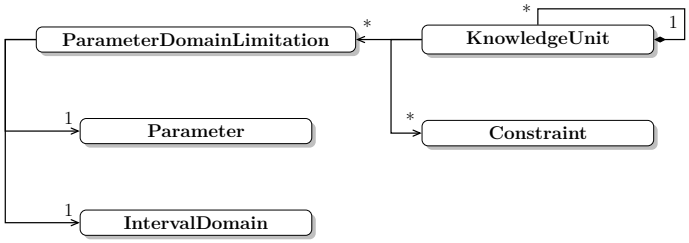


Abbildung 6.2: Auszug aus dem Klassendiagramm: *Wissenseinheiten*, *Parameter* und *Constraints*

6.3.5 Funktionale Einheit

Funktionale Einheiten (englisch: Functional Unit) stellen die Entitäten und Geometrien dar und sind angelehnt an die Objekte selben Namens aus dem *General AEC Reference Model* (GARM) (vgl. Gielingh, 1988). Sie stellen ein Objekt dar, welches einer geometrischen Form oder einer Entität im ingenieurtechnischen bzw. modelliertechnischen Sinne entspricht. Sie besitzt nach außen klar definierte *Parameter* und ihre Funktion ist im Inneren mit Hilfe von *Wissenseinheiten* vollständig beschrieben.

Die *Funktionale Einheit* ist entweder ein *Modellobjekt* oder ein *Geometrie-Objekt*, wobei sich *Geometrie-Objekte* wiederum in zwei Arten unterteilen lassen. Dies wird im Folgenden genauer beschrieben:

Basisgeometrie(-objekt): (englisch: Base Geometry) Dabei handelt es sich um eine einfache topologische Struktur, deren Form auf der einfachen *Zeichenregel* der geraden Verbindung zweier Punkte beruht. Dazu gehören beispielsweise Punkt, Linie, Polygon oder Polyeder, welche sich selbst keinen Einschränkungen in Form von *Constraints* in *Wissenseinheiten* unterwerfen und keine weiteren *Parameter* besitzen. Die einzige Ausnahme bildet der Punkt, welcher, entsprechend der Dimension des Raums, *Parameter* für die Koordinaten besitzt. Alle anderen *Basisgeometrien* setzen sich aus anderen *Basisgeometrien* zusammen. Sie basieren damit geometrisch direkt oder indirekt auf den Koordinaten der Punkte. *Basisgeometrien* besitzen im Gegensatz zu anderen *Funktionalen Einheiten* ausschließlich *Basisgeometrien* als untergeordnete *Funktionale Einheiten*.

Komplexe(s) Geometrie(-objekt): (englisch: Complex Geometry) Eine *komplexe Geometrie* ergibt sich aus mindestens einer Geometrie (Basis oder Komplex), mindestens einer Constraint und gegebenenfalls *Parametern*, welche die Geometrie im mathematischen Sinne beschreiben. Es seien hier der „Radius“ bei einem Kreis oder die „Breite“ und „Länge“ bei einem Rechteck als Beispiele genannt. Alle *Parameter* sind direkt oder indirekt mit den enthaltenen Geometrien verbunden. Zusätzlich können weitere *Constraints* unabhängig von den *Parametern* zwischen den untergeordneten Geometrien definiert sein.

Modellobjekt: (englisch: Model Object) Ein *Modellobjekt* stellt eine ingenieurtechnische oder modelliertechnische Entität dar und ist folgendermaßen definiert:

- Es besteht aus anderen, untergeordneten *Funktionalen Einheiten*, wobei sie sich direkt oder transitiv auf *Geometrie-Objekte* zurückführen lässt.
- Es kann *Parameter* besitzen, die ingenieurtechnisch oder modelliertechnisch relevant sind und mit Hilfe von *Constraints* mit *Parametern* oder Geometrien von untergeordneten *Funktionalen Einheiten* verknüpft sind.
- Es kann weitere *Constraints* beinhalten, die Geometrieteile von untergeordneten *Funktionalen Einheiten* verknüpfen.

Der Zusammenhang der *Funktionalen Einheiten* und *Parameter* ist in Abbildung 6.3 dargestellt.

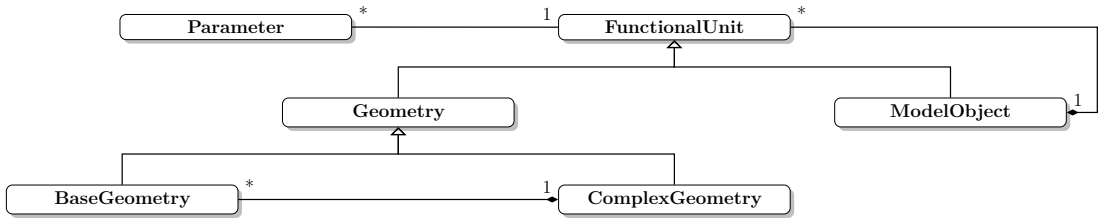


Abbildung 6.3: Auszug aus dem Klassendiagramm: *Funktionalen Einheiten* und *Parameter*

Werden *Basisgeometrien* und *Komplexe Geometrien* verwendet, so können alle denkbaren geometrischen Körper konstruiert werden. Dabei können alle *expliziten Beschreibungen* von Geometrien über *Basisgeometrie-Objekte* modelliert werden. Komplexere Geometrien beschreiben *parametrische Geometrien*, wie zum Beispiel Kreis oder Spline (deutsch: Freiformkurve). Sie können aber auch *Basisgeometrien* weiter beschränken, sodass sie nur bestimmte Formen annehmen können. Beispielsweise kann die *Basisgeometrie* „Polygon mit vier Ecken“ durch Hinzufügen von *Constraints* für verschiedenste *Komplexe*

Geometrien verwendet werden. Dies führt dazu, dass es für ein und dieselbe Geometrie verschiedene Möglichkeiten geben kann, sie mit *Constraints* korrekt zu beschreiben. Das schlägt sich im *konstruktiven Grundgedanken* nieder.

In einfachen Fällen ist es möglich, durch unterschiedliche Kombinationen von *Constraints*, dasselbe Ergebnis zu erreichen. Dies soll an einem Beispiel dreier paralleler Geraden *G1*, *G2* und *G3* mit den Abständen a_{12}, a_{23}, a_{31} gezeigt werden. Dabei ist es ausreichend, zwei Abstände zu kennen, da sich der dritte jederzeit aus den zwei bekannten ableiten lässt. Für die Modellierung existieren in diesem Fall vier Möglichkeiten:

1. a_{12} und a_{23}
2. a_{23} und a_{31}
3. a_{12} und a_{31}
4. a_{12}, a_{23}, a_{31}

Wobei die letzte Möglichkeit eine Redundanz zulässt. Jede mögliche Kombination wird im Modell zugelassen. Dies stellt sich im Modell als nicht weiter problematisch dar, da jede Gerade als *Funktionale Einheit* klare Anforderungen erfüllen muss: Alle notwendigen *Parameter* und alle notwendigen geometrischen Bestandteile müssen vorhanden sein. Im in Abbildung 6.4 dargestellten Teilmodell der *Funktionalen Einheiten* und *Constraints* wird deutlich, wie sich die unterschiedlichen *Constraint*-Arten im Modell eingeordnet werden. Die *Geometrie-Constraints* verbinden ausschließlich *Geometrie-Objekte*, während die *Höheren Constraints* auch zwischen *Funktionalen Einheiten* gesetzt werden können.

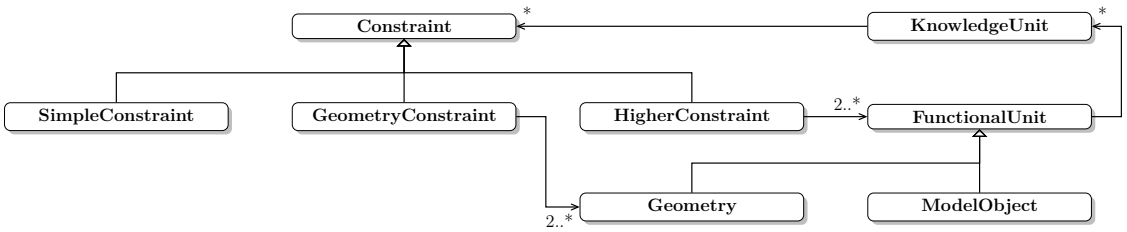


Abbildung 6.4: Auszug aus dem Klassendiagramm: *Funktionalen Einheiten* und *Constraints*

6.3.6 Erzeugungsoperatoren der Geometrie

Für die Modellierung der Geometrie einer *Funktionalen Einheit* werden Operatoren eingeführt, welche die Erzeugung von Geometrie vereinfachen. Diese werden als *Erzeugungsoperatoren der Geometrie* (englisch: Geometry Modeling Operation) bezeichnet.

Typische Operatoren sind *Extrusion* oder *Sweep*. Beim *parametrisch prozeduralen Modellieren* (siehe Abschnitt 4.3.1) dienen solche Operatoren eigentlich als Bauplan von Geometrien für das wiederholte Erzeugen bei Parameter-Änderungen. In diesem Fall stellen sie einen Bauplan dar, der beschreibt, wie sich komplexere *Geometrie-Objekte* erzeugen lassen und wie diese dabei mit *Constraints* und *Parametern* versehen werden. Eine Übersicht über den Zusammenhang der Klassen ist in Abbildung 6.5 gegeben.

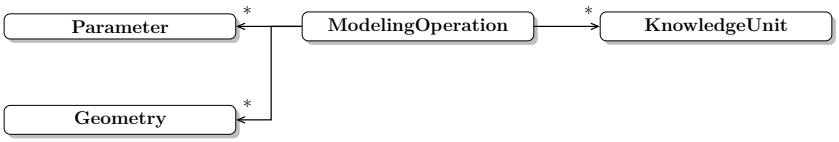


Abbildung 6.5: Auszug aus dem Klassendiagramm: *Erzeugungsoperatoren der Geometrie*

Diese *Erzeugungsoperatoren* besitzen dabei einen Input in Form von bereits im Modell vorhandenen *Geometrie-Objekten* und *Parametern*. Der Input wird dabei genutzt, basierend auf den Regeln des *Operators*, einen Output in Form von *Geometrie-Objekten* und *Constraints* zu erzeugen. Dabei können Teile des Inputs nicht aus dem Modell entfernt werden, sondern bleiben darin nach Ausführen des Operators enthalten. Damit ist ein *Erzeugungsoperator* in dieser Arbeit ein Teil des *Wissens*, wie ein *Geometrie-Objekt* topologisch aufgebaut und mit *Parametern* verknüpft ist. Das schließt nicht aus, dass im Nachgang weitere *Constraints* und *Parameter* für die erzeugten *Geometrie-Objekte* eingefügt werden.

Ein *Erzeugungsoperator* muss für den eingeführten Zweck zu einem validen Ergebnis führen und kann darüber hinaus auch *Prüfregeln* besitzen. Diese können für die Validierung des Inputs und damit die Überprüfung, ob der Operator ein valides Ergebnis liefern kann, genutzt werden. Für komplexere *Regeln* bei der Erzeugung können entsprechend komplexere Operatoren implementiert werden, welche sich wiederum aus anderen Operatoren zusammensetzen. Ein möglicher Operator ist beispielsweise die Extrusion eines Polygons in der *XY*-Ebene in Richtung der *Z*-Achse um den *Parameter* h. Das Resultat ist in diesem Fall eine Sonderform des Polyeders: Ein Prisma mit der Grund- und Oberseite in Form des Polygons.

Der mögliche Aufbau eines solchen *Extrusions-Operators* ist in Abbildung 6.6 gezeigt. Dabei wird für jeden Eckpunkt des zu extrudierenden Polygons ein Eckpunkt für das Polygon der Oberseite erzeugt. Jeder dieser Eckpunkte wird mit seiner Entsprechung auf der Unterseite über *Constraints* verbunden. Dazu sind die X- und Y-Koordinaten gleichzusetzen und die Z-Koordinaten mit einem positiven Abstand h in Zusammenhang zu bringen. Die seitlichen Polygone werden entsprechend der Eckpunkte gebildet. Beispielsweise P_1, P_2, P_2',P_1'. Ist sichergestellt, dass das zu extrudierende Polygon korrekt

orientiert und überschneidungsfrei ist, erzeugt der beschriebene *Extrusions-Operator* einen korrekten Polyeder.

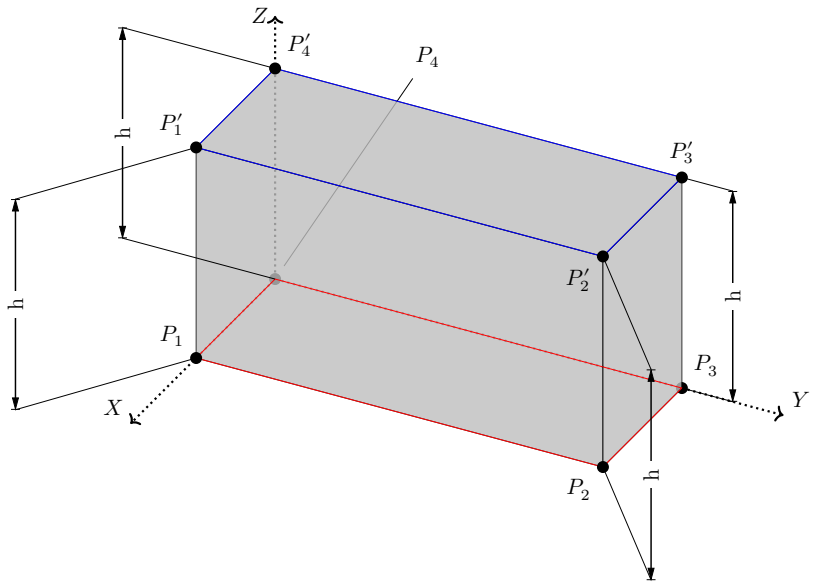


Abbildung 6.6: Beispiel eines *Erzeugungsoperators*: *Extrusions-Operator* für ein Polygon (rotes Rechteck) in XY-Ebene in Z-Richtung. Das entstehende Polygon der Oberseite ist blau dargestellt.

Ein *Erzeugungsoperator* kann über den Erzeugungsprozess hinaus im Modell erhalten bleiben. Er stellt dabei *Wissen* über die ursprüngliche Erstellung der Topologie und damit den *konstruktiven Grundgedanken* dar. Dazu wird er der *Funktionalen Einheit* zugeordnet, für die er Einzelteile erzeugt.

6.3.7 Konsequenzen und Diskussion des Modells

Hierarchiebaum der Funktionalen Einheiten

Aus dem Aufbau der *Funktionalen Einheiten* und *Wissenseinheiten* als geschachtelte Objekte ergibt sich jeweils eine hierarchische Strukturierung. Diese entspricht einer Trennung der Strukturierung der verschiedenen Aspekte des Modells (siehe Abschnitt 3.6). Der sich ergebende Hierarchiebaum besteht aus *Basisgeometrien* in den Blättern und *Komplexen Geometrien* bzw. *Modellobjekten* in den Knoten. Ein Beispiel ist in Abbildung 6.7 gezeigt.

Ebenenmodell

Betrachtet man zusätzlich zur Hierarchie der *Funktionalen Einheiten*, die *Constraints*, *Parameter* und die zugeordneten Gleichungen, so lassen sich diese verschiedenen Ebenen

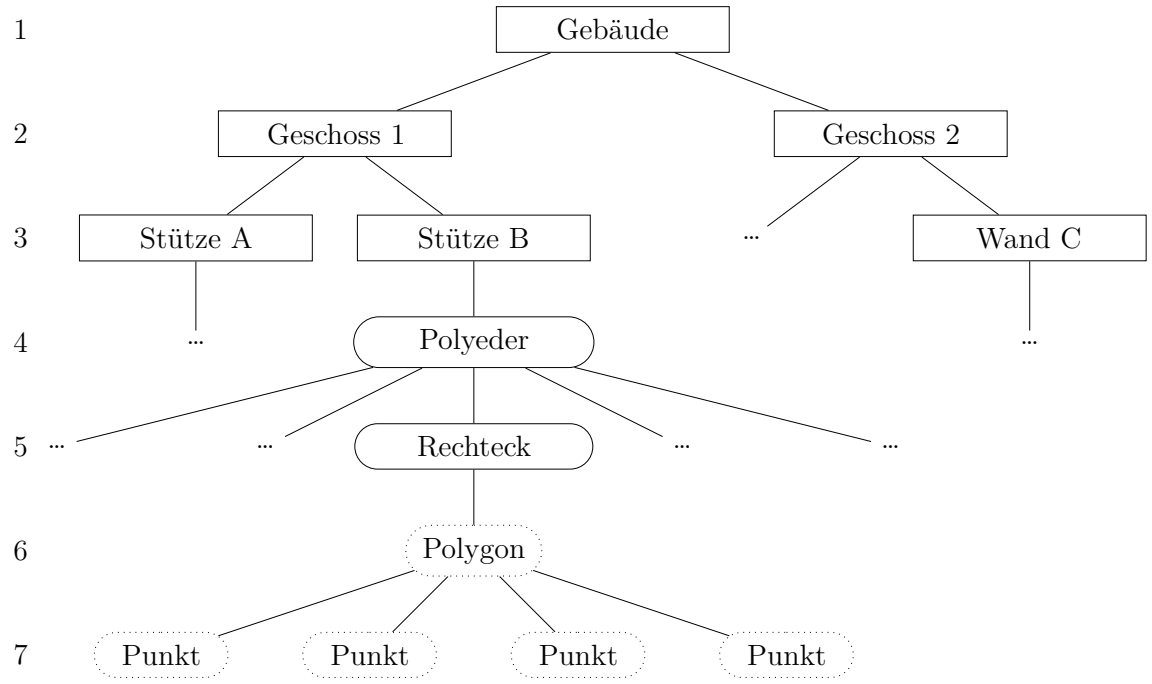


Abbildung 6.7: Beispiel des Hierarchiebaums für *Funktionale Einheiten*; *Modellobjekte* (Rechteck), *Komplexe Geometrien* (durchgezogenes abgerundetes Rechteck), *Basisgeometrien* (gepunktetes abgerundetes Rechteck).

zuordnen. Dabei können verschiedene Objekte auf unterschiedlichen Ebenen gleichzeitig vorkommen. Die Ebenen sind:

Gleichungen: Auf dieser Ebene befinden sich alle *Parameter* und Gleichungen. Sie entspricht dem *Gleichungsgraph* (Abschnitt 2.4).

Basisgeometrie: Auf dieser Ebene befinden sich *Geometrie-Constraints*, *Basisgeometrien* und *Parameter*.

Komplexe Geometrie und Modellelemente: Auf dieser Ebene befinden sich *Geometrie-Constraints*, *Höhere Constraints*, *Komplexe Geometrien* und *Modellobjekte*. Diese Ebene kann auf Grund der Hierarchie der *Funktionalen Einheiten* in weitere Ebenen, analog zum Hierarchiebaum, zerlegt werden.

Ein Beispiel der Betrachtung als Ebenenmodell befindet sich in Abbildung 6.8.

Das Ebenenmodell ist insofern von Vorteil, da es dem Menschen ermöglicht, die Zusammenhänge in verschiedenen Detailierungsstufen zu betrachten. Ein vergleichbares

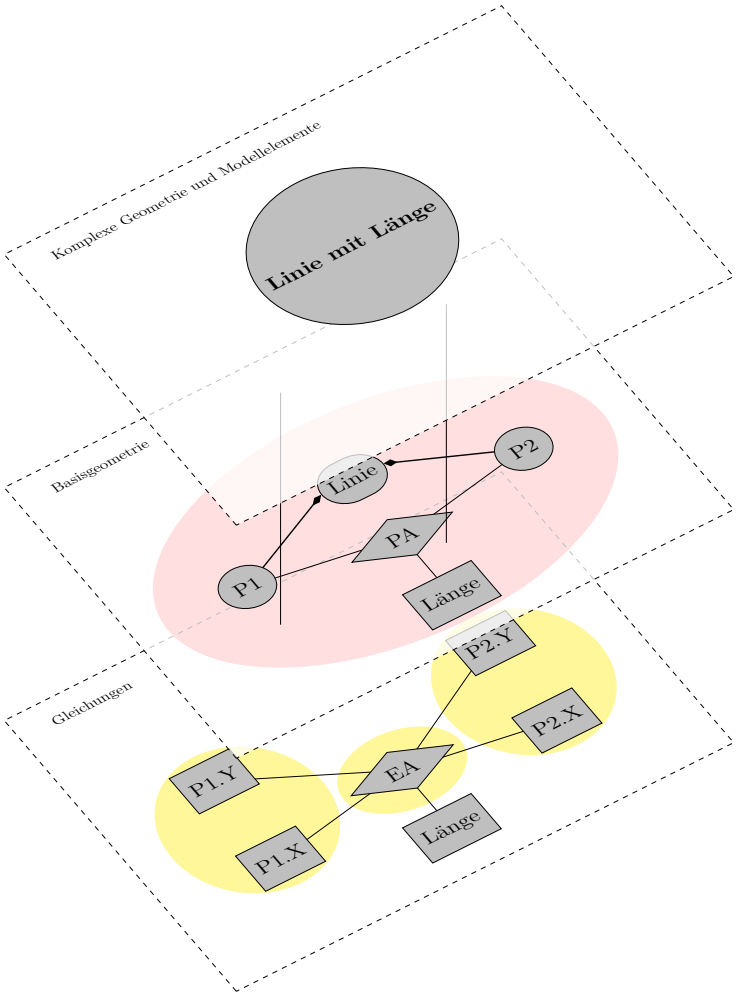


Abbildung 6.8: Ebenenmodell am Beispiel einer *Linie mit Länge*. Erklärung: P1, P2 sind die Endpunkte einer Linie; *Länge* ist ein *Parameter*; PA ist die *Constraint* „Punktabstand“; EA ist die Gleichung des euklidischen Abstands; Farblich hervorgehoben, sind Bereiche, die den Objekten der nächst höheren Ebene zusammengefasst zugeordnet werden können.

Konzept zur Kombination der Objekthierarchie mit *Constraints* und Gleichungen wurde bereits von Böger (2017)[1] vorgeschlagen.

Diskussion

Das vorgestellte Modell erlaubt es, die verschiedenen Aspekte eines Bauwerksmodells miteinander in Verbindung zu setzen und gleichzeitig strukturiert abzubilden. Die Einführung der *Funktionalen Einheiten* und deren Spezialisierungen ermöglicht es, gemäß

[1] Der Autor war an der Erarbeitung der Aufgabenstellung und der Betreuung der hier genannten Arbeit maßgeblich beteiligt.

der *Objektorientierung* einzelne abgeschlossene Einheiten zu bilden. Dadurch können die Repräsentationen der Entitäten bzw. deren Geometrien eindeutig definiert werden. Zusätzlich wird mit der Gruppierung von *Constraints* und *Parameterbegrenzungen* in *Wissenseinheiten* erreicht, dass *Wissen* sinnvoll zusammengefasst und der jeweiligen *Funktionalen Einheit* zugeordnet werden kann. Die *Wissenseinheiten* stellen die eigentliche Neuerung dar, da sie es ermöglichen nachzuvollziehen, wie Beschränkungen zusammenhängen und welchen – gegebenenfalls gemeinsamen – Ursprung sie besitzen.

Insgesamt ist das Modell durch die allgemeine Definition der *Funktionalen Einheiten* so flexibel gestaltet, dass unterschiedliche Bauwerke – Hochbau, Tiefbau, Infrastruktur, etc. – abgebildet werden können. Ein *Modellobjekt* als Repräsentation der Entität des Teils eines Bauwerks ist nicht zwangsläufig einem Bauteil zuzuordnen, sondern kann beliebig verwendet werden. Beispielsweise kann ein *Modellobjekt* einen Bauabschnitt darstellen oder den Überbau einer Brücke zusammenfassen. Der verwendete Ansatz lässt im Entwurfsprozess eine Vielzahl an Möglichkeiten zu, welche Entitäten abgebildet und wie detailliert diese unterteilt werden sollen.

Die Nutzung der *Erzeugungsoperatoren der Geometrie* vereinfacht den Erstellungsprozess und unterstützt den *konstruktiven Grundgedanken*. Damit ist es möglich sich wiederholende Schritte bei der Erstellung des Modells zusammenzufassen und gegebenenfalls mit einzubinden.

Es ist in Zukunft zu überlegen, ob es sinnvoll ist, Schnittstellen – ähnlich der im GARM vorgesehenen *Anschlüsse* (englisch: ports) bzw. *freien Enden* (englisch: free ends) – zu integrieren (vgl. Gielingh, 1988). Diese würden vordefinierte Anknüpfungspunkte für *Constraints*, bestehend aus *Parametern* und untergeordneten *Funktionalen Einheiten*, darstellen. Die Konsequenz wären detailliert ausgearbeitete Bauteilfamilien, die so vorbereitet sind, dass sie im Modell funktional aneinandergebunden werden könnten.

KAPITEL 7

Prototypische Implementierung

In diesem Kapitel wird die prototypische Implementierung erläutert, die im Zuge dieser Arbeit entwickelt wurde. Dazu gehören die umgesetzten *Funktionalen Einheiten* und *Constraints*. Es werden die eingesetzten Algorithmen zur Auswertung beschrieben, sowie auf die Visualisierung der Lösungsräume eingegangen.

7.1 Technische Grundlagen

Als technische Grundlage der prototypischen Implementierung dient eine für diese Arbeit entwickelte Bibliothek für Intervalle und *Intervallarithmetik* in der Programmiersprache *Java*[1]. Der darin verwendete numerische Datentyp ist *BigDecimal*[2]. Dies stellt eine vorzeichenbehaftete Dezimalzahl mit beliebiger Präzision dar und entspricht einer Software-seitigen Implementierung der Fließkommazahlen nach IEEE Std 754-2008. Der Nachteil der Software-seitigen Berechnung gegenüber der Hardware-seitigen Berechnung von Fließkommazahlen mit fester einfacher oder doppelter Präzision ist die geringere Performance. Diesem steht die direkte und flexible Kontrolle der Präzision und Rundungsmodi für alle verwendeten Operatoren als Vorteil gegenüber. Zu den implementierten Operatoren gehören Mengenoperatoren, Vergleichsoperatoren, arithmetische Operatoren und Funktionen. Die arithmetischen Operatoren wurden den algebraischen Ausdrücken der *Geometrie-Constraints* entsprechend gewählt. Dazu gehören die mathematischen Grundoperatoren $+$, $-$, $\cdot$ und $/$ und die erweiterten Operationen ^{2} (Quadrierung) und $\sqrt{}$

[1] Die genutzt Java-Version ist `11.0.2`

[2] https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/math/BigDecimal.html (abgerufen am 28.10.2019)

(Quadratwurzel). Für die Berechnung von Ableitungen ist die Methode der *automatischen Differentiation* (vgl. U. Kulisch u. a., 1993, Kapitel 5) implementiert.

Die dreidimensionale Visualisierung erfolgt mit der *jMonkey-Engine*[3]. Dabei sind die *Basisgeometrien* Punkt und Linie direkt zur Darstellung in 3D-Visualisierungen geeignet. Polygone sind vorher in Dreiecke zu zerlegen. Dies erfolgt mittels *Delaunay-Triangulation* der Bibliothek *Poly2Tri*[4].

7.2 Implementierte Funktionale Einheiten

Als *Funktionale Einheiten* (siehe Abschnitt 6.3.5) wurden verschiedene *Modellobjekte* und die für diese benötigten Geometrie-Objekte implementiert.

Als *Basisgeometrie-Objekte* wurden folgende Klassen implementiert:

Punkt: Mit drei Koordinaten als Parameter.

Linie: Bestehend aus zwei Punkten; je einer als Endpunkt.

Polygon: Mit k-Ecken; besteht aus k Punkten und k Linien; wobei $k > 2$.

Die *Basisgeometrie-Objekte* dienen zusätzlich als Eingangsdaten für die dreidimensionale Visualisierung. Daneben wurden *Komplexe Geometrie-Objekte* aus den *Basisgeometrie-Objekten*, *Constraints* und *Parameter* zusammengesetzt:

Rechteck: Bestehend aus einem Polygon mit 4 Eckpunkten und den zusätzlichen *Parametern* „Länge“ und „Breite“.

Trapez: Bestehend aus einem Polygon mit 4 Eckpunkten und den zusätzlichen *Parametern* „Länge Seite 1“, „Länge Seite 2“ und „Höhe“.

Die *Basisgeometrie-Objekte* und *komplexen Geometrie-Objekte* dienen als Formen für die *Modellobjekte*. Diese wurden aus dem Bereich des Hochbaus entnommen. Dazu wurden Bauteile und Räume betrachtet. Die implementierten *Modellobjekte* sind:

Wand mit rechteckiger Grundfläche: Mit einem Rechteck als Grundfläche, welches um den *Parameter* „Höhe“ in Z-Richtung extrudiert wird. Die *Parameter* „Länge“ und „Dicke“ der Wand entsprechend den *Parametern* „Länge“ und „Breite“ des Rechtecks.

[3] https://jmonkeyengine.org/ in Version `3.2.4`
[4] https://github.com/orbisgis/poly2tri.java in Version 0.1.2

Wand mit trapezförmiger Grundfläche: Mit einem Trapez als Grundfläche, welches um den *Parameter* „Höhe“ in Z-Richtung extrudiert wird. Der *Parameter* „Dicke“ entspricht dem *Parameter* „Höhe“ des Trapezes.

Raum mit polygonaler Grundfläche: Mit einem Polygon als Grundfläche. Zusätzlich besitzt der Raum die *Parameter* „Fläche“ und „Umfang“

7.3 Implementierte Geometrie-Constraints

In dieser Arbeit wurden verschiedene *Geometrie-Constraints* implementiert. Diese besitzen eine geometrische Bedeutung, werden aber als eine oder mehrere Gleichungen beschrieben. Einen Überblick über *Geometrie-Constraints* und deren Gleichungen geben Wang u. a. (2007, Tabelle 1). Diese gehören zu der von Bettig und Shah (2001) definierten Standardmenge der *Geometrie-Constraints*.

Die im Folgenden beschriebenen *Geometrie-Constraints* orientieren sich daran und finden auch in den Beispielen dieser Arbeit Verwendung. Teilweise gehen sie aber auch darüber hinaus.

In den folgenden Abschnitten ist $\odot$ das Skalarprodukt und $\times$ das Kreuzprodukt im jeweiligen Raum.

7.3.1 Punkt-Koinzidenz

Entsprechend der gewählten Koordinatenrichtungen lassen sich die Koordinaten zweier Punkte $P1$ und $P2$ gleichsetzen und so die Koinzidenz herstellen. Es kann dabei die Koinzidenz bezogen auf eine Hauptachse (1 Gleichung), auf eine Hauptebene (2 Gleichungen) oder auf alle Koordinatenrichtungen (3 Gleichungen) umgesetzt werden. Für alle Koordinatenrichtungen werden die *Parameter* der Punktkoordinaten „X“, „Y“ und „Z“ gleichgesetzt:

- $P1.X = P2.X$
- $P1.Y = P2.Y$
- $P1.Z = P2.Z$

7.3.2 Gerichtete Linien-Koinzidenz

Sollen zwei Linien koinzident liegen, so müssen ihre Endpunkte koinzident liegen. Aus diesem Grund wird die Linien-Koinzidenz aus dem Prinzip zweier *Constraints* für Punkt-

Koinzidenz gebildet. Die Richtung ergibt sich daraus, dass jeder Endpunkt einen Partner in der anderen Linie zugeordnet bekommt. Es ist also nicht möglich, dass die Linien den bei einer allgemeinen Koinzidenz möglichen zweiten Zustand mit der inversen Richtung annehmen können.

7.3.3 Abstand zweier Punkte

Aus dem Abstand zweier Punkte $P1$ und $P2$ ergibt sich der Abstand a in einer Hauptachse, einer Hauptebene oder im euklidischen Raum R^3 als Betrag des Differenzvektors. Dies führt im R^3 zum euklidischen Abstand in Form der Gleichung:

$$(P2.X - P1.X)^2 + (P2.Y - P1.Y)^2 + (P2.Z - P1.Z)^2 = a^2 \tag{7.1}$$

Ein Abstand in einer Hauptrichtung kann auch vorzeichenbehaftet sein. Ist der Abstand positiv, liegt $P2$ in, ist er negativ, liegt er entgegen der Hauptrichtung. So ist beispielsweise der Abstand a_Z in Z-Richtung:

$$P2.Z - P1.Z = a_Z \tag{7.2}$$

7.3.4 Abstand Punkt zu Gerade

Für den Abstand a eines Punkts P von der Geraden g in der Zweipunkteform (mit den Punkten $P1$ und $P2$) gilt folgende Berechnungsvorschrift:

$$\frac{|(\overrightarrow{g.P2} - \overrightarrow{g.P1}) \times (\vec{P} - \overrightarrow{g.P1})|}{|\overrightarrow{g.P2} - \overrightarrow{g.P1}|} = a \tag{7.3}$$

Als Sonderform lässt sich der Abstand in der XY-Ebene (bei $Z = 0$) auch vorzeichenbehaftet ausdrücken, indem auf den Betrag im Zähler verzichtet wird. Dabei lässt sich unterscheiden, ob P links $a < 0$ oder rechts $a > 0$ von der Geraden g liegt, bezogen auf den Richtungsvektor $\overrightarrow{g.P1\ g.P2}$.

7.3.5 Rechter Winkel zweier Geraden

Für die *Constraint* für den Erhalt des rechten Winkels zweier Geraden g_1 und g_2 in der Zweipunkteform gilt folgende Berechnungsvorschrift:

$$(\overrightarrow{g_1.P2} - \overrightarrow{g_1.P1}) \odot (\overrightarrow{g_2.P2} - \overrightarrow{g_2.P1}) = 0 \tag{7.4}$$

7.3.6 Parallele Geraden

Für die Parallelität zweier Geraden g_1 und g_2 in der Zweipunkteform gilt folgende Berechnungsvorschrift:

$$(\overrightarrow{g_1.P2} - \overrightarrow{g_1.P1}) \times (\overrightarrow{g_2.P2} - \overrightarrow{g_1.P1}) = \vec{0} \tag{7.5}$$

7.3.7 Abstand zweier Geraden

Für den Abstand a zweier (paralleler) Geraden g_1 und g_2 in der Zweipunkteform gilt folgende Berechnungsvorschrift:

$$\begin{aligned} \frac{|(\overrightarrow{g_1.P2} - \overrightarrow{g_1.P1}) \times (\overrightarrow{g_2.P1} - \overrightarrow{g_1.P1})|}{|\overrightarrow{g_1.P2} - \overrightarrow{g_1.P1}|} &= a \\ \frac{|(\overrightarrow{g_1.P2} - \overrightarrow{g_1.P1}) \times (\overrightarrow{g_2.P2} - \overrightarrow{g_1.P1})|}{|\overrightarrow{g_1.P2} - \overrightarrow{g_1.P1}|} &= a \end{aligned} \tag{7.6}$$

Als Sonderform lässt sich der Abstand in der XY-Ebene ($Z = 0$) auch vorzeichenbehaftet formulieren, in dem jeweils auf den Betrag im Zähler verzichtet wird. Dabei lässt sich unterscheiden, ob g_2 links $a < 0$ oder rechts $a > 0$ von der Geraden g_1 liegt, bezogen auf den Richtungsvektor $\overrightarrow{g.P1g.P2}$.

7.3.8 Rechter Winkel mit vorzeichenbehaftetem Abstand

Der rechte Winkel in XY-Ebene zwischen $P1$, $P2$ und $P3$ mit einem vorzeichenbehaftetem Abstand a zwischen $P1$ und $P2$ lässt sich aus der Kombination der zwei *Constraints Rechter Winkel zwischen Geraden* und *Abstand Punkt zu Gerade* definieren. Dabei ergibt sich die Gerade g_1 in der Zweipunkteform aus P_2 und P_3 und die Gerade g_2 aus $P1$ und $P2$.

7.3.9 Positive Orientierung von 3 Punkten

Die positive Orientierung (gegen den Uhrzeigersinn) von 3 Punkten $P1$, $P2$ und $P3$ in der XY-Ebene ergibt sich mit folgender Berechnungsvorschrift:

$$\overrightarrow{P1\ P2} \odot \overrightarrow{P1\ P3} = (0, \infty) \tag{7.7}$$

7.3.10 Länge eines Kantenzugs

Für die Länge l eines Kantenzugs, beschrieben durch die Punkte $P_1, \ldots, P_n$, gilt folgende Berechnungsvorschrift:

$$\sum_{i=1}^{n-1} |\vec{P}_{i+1} - \vec{P}_i| = l \tag{7.8}$$

7.3.11 Fläche eines Polygons

Für die Fläche f eines Polygons, beschrieben durch die Punkte $P_1, \ldots, P_n$ (mit $n+1=1$), in der XY-Ebene lässt sich die Gaußsche Trapezformel anwenden:

$$\sum_{i=1}^{n} (P_iY + P_{i+1}Y) \cdot (P_iX + P_{i+1}X) = 2 \cdot |f| \tag{7.9}$$

Als Sonderform lässt sich der Abstand auch vorzeichenbehaftet formulieren, in dem auf den Betrag von f verzichtet wird. Ist $f > 0$ ist das Polygon, solange es nicht selbstüberschlagend ist, gegen den Uhrzeigersinn, bei $f < 0$ ist es im Uhrzeigersinn orientiert.

7.4 Implementierte geometrische Erzeugungsoperatoren

7.4.1 Linien-Extrusion

Bei der Linien-Extrusion wird eine Line L, beschrieben durch die Eckpunkte $P1$, $P2$, in einer der Hauptebenen, rechtwinklig mit einem Abstands-*Parameter* „a“ extrudiert, wobei ein Rechteck R entsteht. Dabei besitzt R, neben $P1$ und $P2$, die Eckpunkte $P3$ und $P4$. Wird L als orientierte Linie mit Start und Endpunkt betrachtet, so befindet sich $P3$ auf der linken Seite von L. $P4$ befindet sich rechtwinklig zu $P1$ ebenfalls auf der linken Seite von L.

Es wird dabei eine *Constraint* für den rechten Winkel mit vorzeichenbehaftetem Abstand für $P3$ (rechts) zum Punkt $P2$ bezogen auf $P1$ erzeugt. Analog geschieht dies für $P4$ mit einem rechten Winkel mit vorzeichenbehaftetem Abstand (links) zu $P1$ bezogen auf $P2$.

7.4.2 Polygon-Extrusion

Bei Polygon-Extrusion wird ein Polygon *Poly*1 in Richtung einer gewählten Hauptachse (X, Y oder Z) um einen Wert „e“ extrudiert. Dies resultiert zum einen in einem neuen Polygon *Poly*2, welches kongruent zu *Poly*1 und um e versetzt ist, zum anderen wird jeder Eckpunkt von *Poly*1 mit seiner Entsprechung in *Poly*2 mit Hilfe einer *Constraint vorzeichenbehafteten Punktabstand* in Extrusionsrichtung in Beziehung gesetzt. Zusätzlich werden für jede berandende Linie von *Poly*1 bzw. *Poly*2 Polygone als Seitenfläche des entstehenden Festkörpers erzeugt.

7.5 Such- und Optimierungsalgorithmen

7.5.1 Grundalgorithmus

Alle Such- und Optimierungsalgorithmen folgen einem Basis-Algorithmus nach dem Prinzip des *Branch-and-Prune*. Dieser Basis-Algorithmus ist in Algorithmus 7.1 dargestellt. Die vier Phasen (siehe Abschnitt 5.4.5) werden dabei variabel gestaltet. Das bedeutet, dass je nach Zielsetzung eine andere Strategie in der jeweiligen Phase genutzt werden kann. In der *Extract*-Phase wird auf Basis einer *Extract-Strategie* eine *Box* bzw. der *Ast*, der diesen Suchraum repräsentiert, ausgewählt. Anschließend können in der *Prune*-Phase eine oder mehrere *Prune-Strategien* zum Einsatz kommen. Das Ziel ist, soweit wie möglich bisherige Lösungen und vordefinierte *Prune-Operatoren* zur Verkleinerung der betrachteten *Box* zu nutzen. Sollte in der *Prove*-Phase die *Box* nicht ausgeschlossen werden, so wird überprüft, ob ein weiterer Split der *Box* möglich ist. Ist dies nicht möglich, da ein unterer Schwellenwert für die Intervallbreite erreicht wurde, so wird überprüft, ob die gefundene Lösung relevant für das aktuelle Ziel ist. Diese Überprüfung wird über die *Prove-Strategie* festgelegt. Stellt sie die einzige Lösung dar, die gefunden werden soll, kann direkt im Anschluss darauf verzichtet werden, weitere *Boxen* zu durchsuchen, und der Algorithmus wird beendet. Ist ein weiterer *Bisektions-Split* der *Box* möglich, so wird dieser in der *Branch*-Phase durchgeführt. Auch hier kann eine *Bisektions-Strategie* gewählt werden. Diese entscheidet darüber, auf welcher Grundlage eine *Variable* für den Split ausgewählt wird. Der Algorithmus endet, wenn entweder in der *Prove*-Phase ein definiertes Abbruchkriterium durch Finden einer Lösung erreicht wurde, oder wenn keine weiteren zu durchsuchenden *Boxen* des Suchraums zur Verfügung stehen.

Algorithmus 7.1: Basis-Algorithmus nach dem Prinzip des *Branch-and-Prune*

Input: $\mathbf{B}_{init}$ is the initial box representing the complete search space
f is the set of functions to satistfy

Output: no, one or more solutions represented as boxes

Preliminaries: L is the set of all boxes possibly containing solutions
$extract_{strat}(L)$ is a function that pops a box from L by a specific strategy
$prune_{strat}(\mathbf{X})$ is a function that runs pruning-operators on $\mathbf{X}$
$prove_{strat}(\mathbf{X})$ is a function that checks, whether a solution $\mathbf{X}$ is relevant
$bisect_{strat}(\mathbf{X})$ is a function that splits $\mathbf{X}$ by choosing a splittable interval

```
1  push(L, B)
2  repeat
3      X := extract_strat(L)
4      prune_strat(X)
5      if X cannot contain solutions then
6          discard X
7      if X cannot be split anymore then
8          prove_strat(X)
9      else
10         {X_1, X_2} := bisect_strat(X)
11         push(L, X_1 and X_2)
12     end
13 until L is ∅ OR all relevant solutions have been evaluated
```

Als *Bisektions-Strategie* wurde die Heuristik nach Kubica (2015) gewählt, welche die *Bestimmtheit* des *Constraint-Systems*, das Ergebnis der letzten Anwendung des *Intervall-Newton-Verfahrens* und die Breiten der Intervalle der 1. Ableitungen berücksichtigt.

7.5.2 Allgemeine Pruning-Operatoren

Als allgemeine *Pruning-Operatoren* wurden *HC4* und das *komponentenweises Intervall-Newton-Verfahren* ausgewählt. Diese werden grundsätzlich in der *Prune*-Phase ausgeführt und stellen jeweils *Nicht-Existenz-Tests* dar, die *Boxen* ausschließen und verwerfen können.

HC4

Eine effiziente Berechnung einer *Hüllen-Konsistenz* kann über den *Pruning-Operator HC4* (siehe Abschnitt 5.4.7) erfolgen. Er ist zwar vom *Intervall-Abhängigkeits-Problem*

betroffen, hat aber im Vergleich zu anderen *Pruning-Operatoren* eine geringe Laufzeit mit einer angemessenen Verkleinerung von Intervallen.

Zur Verhinderung, dass bei langsam konvergierenden Funktionen der Rechenaufwand steigt, wird neben dem Abbruchkriterium, des Erreichens des *Fixpunkts*, eine Begrenzung auf $3 \cdot e$ Aufrufe von *HC4Revise* (e ist die Anzahl der Gleichungen des ICS). Dies ist ein legitimer Ansatz, da *Pruning-Operatoren* nur verkleinern, aber nicht zwangsläufig einen *Fixpunkt* erreichen müssen um von Nutzen zu sein. Auch wenn kein *Fixpunkt* erreicht wurde, kann das *Branch-and-Prune-Verfahren* von den verkleinerten *Boxen* profitieren.

Komponentenweises Intervall-Newton-Verfahren

Als *Pruning-Operator*, der die 1. Ableitung berücksichtigt, wurde das *Intervall-Newton-Verfahren* der *komponentenweisen* Berechnung implementiert (siehe Abschnitt 5.4.6). Dieses Verfahren hat gegenüber anderen den Vorteil, dass es ohne Einschränkung für jede Kombination von n Variablen mit m Gleichungen durchführbar ist und keine zusätzliche Berechnung von Pseudoinversen benötigt. Als erstes Abbruchkriterium des Operators dient das Erreichen eines *Fixpunkts*. Dabei wird die Schrittweite in den einzelnen Iterationsschritten berücksichtigt. Sollte keine Variable eine signifikante Änderung größer einem Wert $\epsilon_{Fixpoint}$ aufweisen, so gilt der *Fixpunkt* als erreicht. Das zweite Abbruchkriterium ist die Begrenzung der Anzahl der Durchläufe q auf einen festen Wert. Hierbei wurde $q = 5$ gewählt. Das zweite Abbruchkriterium hat den Zweck, bei langsam konvergierenden Funktionen den Rechenaufwand zu begrenzen.

7.5.3 Optimierung und Optimierungsziele

Im Falle einer nachträglichen Änderung oder bei der Variantendiskussion kann es notwendig sein, eine oder mehrere optimierte Lösungen für ein GCS zu finden. Voraussetzung für eine sinnvoll durchgeführte Optimierung ist, dass es sich um ein *unterbestimmtes Constraint-System* handelt (siehe Abschnitt 2.3). Bei einem *wohlbestimmten* System würde die Optimierung immer zu der einen möglichen Lösung führen. Roller (2013) stellt die grundlegende Frage „[…] [Welche] der Lösungen der Vorstellung des Konstrukteurs am nächsten kommt […] [?]“. Da diese Aussage formal nicht eindeutig beschreibbar ist, wird in den folgenden Abschnitten die numerische Optimierung in einem ICSP beschrieben und ein Algorithmus dafür vorgestellt. Anschließend wird die geometrische „Nähe“ diskutiert und ein Vorschlag gemacht, wie diese bei der Optimierung berücksichtigt werden kann.

7.5.4 Methode der oberen Begrenzung

Für die Optimierung wird die *Methode der oberen Begrenzung* (siehe Abschnitt 5.5) genutzt. Da die Zielfunktion $F_{Ziel}(\mathbf{X})$ als algebraische Gleichung formuliert werden kann, gelten für sie dieselben Eigenschaften wie für *Intervall-Constraints* (Abschnitt 5.4.1). Dazu gehört, dass sie nach jeder beliebigen Variablen aufgelöst werden kann. Mit diesem Prinzip kann jederzeit geprüft werden, ob eine *Box* $\mathbf{X}$ Lösungen enthalten kann, die einen kleineren Wert für $F_{Ziel}(\mathbf{X})$ liefern können, als die bisher gefundenen Lösungen.

Dies soll an einem Beispiel erläutert werden. Es seien folgende Komponenten gegeben:

- Ein *Constraint-System* bestehend aus zwei Variablen A und B.
- Die Initiale-*Box* für A und B: $(X_{init}) = \{[2,4],[3,7]\}$.
- Die zu minimierende Zielfunktion $F_{Ziel}(\mathbf{X}) = A + B$.
- Ein vordefinierter bester Wert als obere Begrenzung $\overline{G}_{best} = 10$.

Der Wert der Zielfunktion zu Beginn beträgt $F_{Ziel}(\mathbf{X_{init}}) = [5,11]$. Es ist zu erkennen, dass die obere Grenze des Intervalls größer ist als $\overline{G_{best}}$. Durch eine Umstellung von F_{Ziel} nach A und einmal nach B lässt sich die aktuell obere Grenze von 10 auf $\mathbf{X}_{init}$ projizieren. Durch Bildung der Schnittmenge mit der Projektion ergibt sich die neue *Box* $\mathbf{X}_{neu} = \{[2,4],[3,6]\}$. Die obere Begrenzung 10 führt in diesem Beispiel zu einem Abschneiden der Werte, für die keine mindestens gleich gute Lösung möglich ist.

Die *Methode der oberen Begrenzung* kann als Kombination aus einem *Pruning-Operator* und einem *Prove-Operator* umgesetzt werden. Der *Pruning-Operator* führt dabei die obere Begrenzung durch. Der Algorithmus ist in Algorithmus 7.2 dargestellt. Zu beachten ist, dass der *Pruning-Operator* zwar die *Box* einschränken kann, diese aber gegebenenfalls erst im Anschluss mit den anderen *Pruning-Operatoren* (siehe Abschnitt 7.5.2) verworfen wird.

Algorithmus 7.2: *Prune-Operator* für die Minimierung nach der *Methode der oberen Begrenzung*

Input: **B** is the box of a branch, that should be pruned
$f_O(\mathbf{X})$ is the objective function for minimization
Output: no, one or many solutions represented as boxes

Preliminaries: X_{best} is the interval representing the lowest found value for $f_O(\mathbf{X})$ with a solution's box **X** so far

1 **if** $mid(\mathbf{X})$ *is a solution* **then**
2 prove solution $mid(\mathbf{X})$ by Prove-Operator
3 **end**
4 **if** X_{best} *has not been set* **then**
5 **return** *no pruning possible; Branch-and-Prune has to continue, until a first solution is found*
6 **end**
7 **foreach** *variable V included in* f_O **do**
8 f_{inv} := invert $f_O(\mathbf{X})$ for V
9 $V_{prunedvalue} := f_{inv}(X_{best}, \mathbf{B})$
10 $V_{newvalue} := V_{prunedvalue} \cap V_{oldvalue}$
11 **if** $V = \emptyset$ **then**
12 **return** *discard* ***B****, that does not include a better value than* X_{best}
13 **else**
14 put $V_{newvalue}$ into **B**
15 **end**
16 **end**
17 **return** *deliver* ***B*** *to Branch-and-Prune*

Der *Prove-Operator* dient dazu, gefundene Lösungen mit der Zielfunktion zu bewerten und sie gegebenenfalls zu speichern und den bisherigen besten Wert der Zielfunktion zu aktualisieren. Der für die *Methode der oberen Begrenzung* genutzte *Prove-Operator* ist in Algorithmus 7.3 dargestellt.

Algorithmus 7.3: *Prove-Operator* für die Aktualisierung der bisher besten Lösung bei der Minimierung nach der *Methode der oberen Begrenzung*

Input: **B** is the box of a branch, that represents a solution
$f_O(\mathbf{X})$ is the objective function for minimization

Preliminaries: X_{best} is the interval representing the lowest found value for $f_O(\mathbf{X})$ with a solution's box **X** so far

1 $B_{candidate}$:=$f_O(\mathbf{B})$
2 **if** X_{best} *has not been set OR* $\underline{B}_{candidate} < \underline{X}_{best}$ **then**
3 X_{best}:=$B_{candidate}$
4 $\mathbf{X} := \mathbf{B}$
5 **else**
6 **end**

Als weitere Einstellung wird in der *Extract*-Phase immer der *Ast* ausgewählt, dessen *Box* X_e am vielversprechendsten ist. Dazu wird die Zielfunktion mit $Y_e = F_{Ziel}(\mathbf{X}_e)$ berechnet und eine der *Boxen* ausgewählt, für welche die untere Grenze $\underline{Y}_e$ am niedrigsten ist.

7.5.5 Zielfunktion für die geometrische Anfrage

Wie schon in Abschnitt 4.5.2 dargestellt, ist die Suche nach Lösungen, die möglichst mit der Absicht des Menschen übereinstimmen, keineswegs trivial. Die *geometrische Anfrage* (siehe Abschnitt 4.5.2), in der bestimmte Zielwerte vorgegeben und möglichst gut bzw. „nah“ erreicht werden, stellt einen solchen Ansatz dar. Die Besonderheit ist hierbei, dass in der Anfrage *Geometrie-Objekte* vom Menschen platziert werden, und der Algorithmus die Lösung für das GCSP bestimmen soll, in der die *Geometrie-Objekte* dieser Konfiguration möglichst nahekommen. In Abschnitt 4.4.1 wird darüber hinaus auf die geforderte Eigenschaft der *Stabilität* für die Lösungsverfahren eines GCSP hingewiesen, dass kleine Änderungen am System zu Lösungen in der „Nähe“ führen. Auch dies kann als *geometrische Anfrage* interpretiert werden.

Es ergibt sich das Problem, dass „Nähe“ nicht immer eindeutig definiert ist. Ausschließlich für das primitivste *Geometrie-Objekt*, den Punkt, ist die Bestimmung einer geometrische Nähe allgemein und eindeutig über den euklidischen Abstand möglich. Schultz u. a. (2017) definieren beispielsweise das Prädikat „der Punkt C liegt näher an Punkt A als an Punkt B“ über die Ungleichung der *quadrierten euklidischen Abstände*:

$$(a_x - c_x)^2 + (a_y - c_y)^2 < (b_x - b_x)^2 + (b_y - b_y)^2 \tag{7.10}$$

Bei anderen *Geometrie-Objekten*, wie zum Beispiel Linien, Polygonen oder Polyeder lassen sich hingegen verschiedene Definitionen der geometrischen Nähe entwickeln. Eine Möglichkeit besteht beispielsweise für zwei Polygone, die größte geometrische Nähe über den kleinsten möglichen Abstand zwischen zwei ihrer Liniensegmenten des Randes zu definieren. Damit wird zwar die Nähe als ein möglichst kleiner Leerraum zwischen den Objekten definiert, dies führt aber zu Problemen, wenn sich die *Geometrie-Objekte* überlappen bzw. durchdringen. Dies ist unzureichend, da es zum einen bei der *geometrischen Anfrage* nicht auszuschließen, sondern sogar gewollt ist, dass solche Durchdringungen zwischen der prototypischen Skizze und der betrachteten möglichen Lösung auftreten. Zum anderen soll die vollständige Lage des *Geometrie-Objekts* in der Anfrage berücksichtigt werden. Das bedeutet beispielsweise, dass bei einem Polygon eine Lösung gesucht wird, bei der alle Eckpunkte des Polygons ihrer Lage aus der Anfrage so weit wie möglich nahekommen.

Betrachtet man die geometrische Nähe als eine Menge von Punkten zu ihrem prototypischen Abbild aus der *geometrischen Anfrage*, so kann dies als Summe der *quadrierten euklidischen Abstände* jedes einzelnen Punkts zu seinem Abbild aufgefasst werden. Wird die Minimierung einer solchen Funktion gefordert, entspricht dies der *Methode der kleinsten Quadrate* (vgl. Papageorgiou u. a., 2012, Kapitel 6):

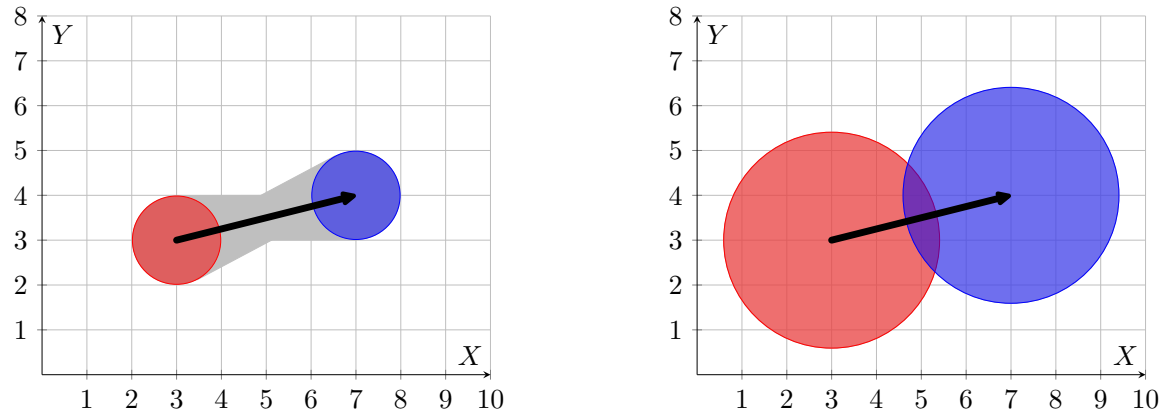
$$d^2(\vec{u}, \vec{v}) = \sum_{i=1}^{n} (x_{ui} - x_{vi})^2 \tag{7.11}$$

Dabei steht x_{ui} für den Wert des *Parameters* in der *geometrischen Anfrage*, welcher das prototypische Abbild darstellt, und x_{vi} für den Wert des *Parameters* in der aktuell betrachteten *Box* des Lösungsprozesses. Angewendet für die *geometrische Anfrage* stellt die *Methode der kleinsten Quadrate* die Summe der Quadrate der Abweichungen der Koordinaten bzw. *Parameter* von der prototypischen Skizze dar.

In dem in dieser Arbeit vorgestellten Bauwerksmodell lassen sich alle *Parameter* als *Variablen* in dem ICS darstellen. Damit ergibt sich für ein solches *Geometrie-Objekt* die *geometrische Anfrage* als das Finden des Minimums der Zielfunktion der Summe der kleinsten Quadrate aller seiner (Eck-)Punktkoordinaten. Die Auswirkung auf einen einzelnen Eckpunkt ist, dass die Zielfunktion einen konstanten Wert liefert, wenn sich die Abweichung des Eckpunkts auf einem Kreis in $\mathbb{R}^2$ bzw. einer Kugel in $\mathbb{R}^3$ um den

Eckpunkt befindet. Die Menge der *Parameter* der Koordinaten der Eckpunkte eines *Geometrie-Objekts* kann als Gruppe aufgefasst werden, die die geometrische Lage des *Geometrie-Objekts* repräsentiert.

In Abbildung 7.1 sind beispielhaft die möglichen Lagen einer Linie dargestellt – bei unterschiedlichen Werten für eine Zielfunktion aus den kleinsten Quadraten der Endpunktkoordinaten. Es wird deutlich, dass sich die mögliche Lage der Linie aus der *konvexen Hülle* aller möglichen Positionen der Endpunkte ergibt. Dies ist darin begründet, dass die Endpunkte jede beliebige Position innerhalb des sich ergebenden Kreises – der Radius beträgt $\sqrt{\text{Wert der Zielfunktion}}$ – annehmen kann. Da zwischen den Punkten bzw. ihren *Parametern* keine Abhängigkeit besteht, können sie ihre Positionen nicht beeinflussen.

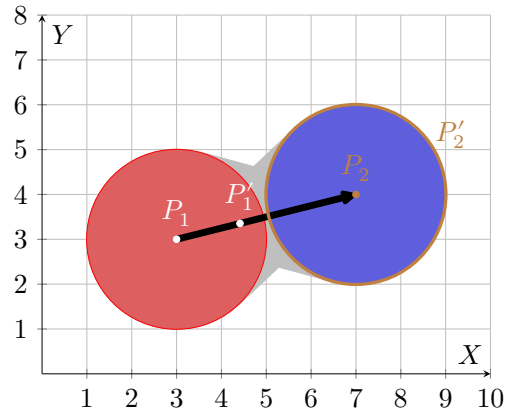


(a) Zielfunktion mit dem Wert 1 **(b)** Zielfunktion mit dem Wert 5

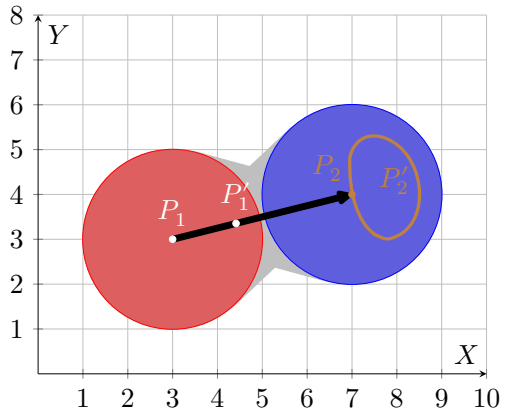
Abbildung 7.1: Mögliche Lage des Startpunkts (rot) und des Endpunkts (blau) einer Linie (schwarz) und die mögliche Lage der Linie im Zwischenraum (hellgrauer Bereich) bei unterschiedlichen Zielfunktionswerten; die Zielfunktion entspricht der Summe der Quadrate der Abweichungen der Koordinaten der Endpunkte.

Werden weitere *Parameter*, die keine Koordinaten repräsentieren, in eine solche Zielfunktion aufgenommen, so kommt es gegebenenfalls zu Abhängigkeiten der *Parameter* untereinander. Beispielsweise sorgt bei einer Linie die Einbeziehung des *Parameters* „Länge“ für eine Optimierung, in der Lösungen bevorzugt werden, in denen die „Länge“ möglichst wenig abweicht. Bei einem konstant angenommenen Wert der Zielfunktion ändert sich zwar nicht die äußere Grenze des möglichen Lagebereichs der Punkte, aber durch die Abhängigkeit wird die Lage der Endpunkte zueinander eingeschränkt. Dies ist in Abbildung 7.2 dargestellt.

Speziell bei komplexeren *Geometrie-Objekten* kann es sinnvoll sein, einzelne oder gruppierte *Parameter* höher zu gewichten. Die Erweiterung der kleinsten Quadrate um die



(a) Zielfunktion berücksichtigt ausschließlich die Koordinaten der Punkte



(b) Zusätzliche Berücksichtigung der quadrierten Abweichung des *Parameters* der Länge der Linie in der Zielfunktion

Abbildung 7.2: Mögliche Lage der Endpunkte P_1 (rot) und P_2 (blau) einer Linie (schwarz) und die mögliche Lage der Linie im Zwischenraum (hellgrauer Bereich) bei dem Zielfunktionswert 4; die Zielfunktion entspricht der Summe der Quadrate der Abweichung der insgesamt vier Koordinaten der Endpunkte. Der Punkt P_1' ist eine beispielhafte Abweichung von P_1 um $\sqrt{2}$ auf der ursprünglichen Linie. Daraus ergibt sich die mögliche Lage von P_2' (orangener Umriss).

Gewichte (vgl. Papageorgiou u. a., 2012, Abschnitt 6.1.2) führt zu folgender Zielfunktion:

$$d^2_{gewichtet}(\vec{u},\vec{v},\vec{p}) = \sum_{i=1}^{n}(x_{ui} - x_{vi})^2 \cdot x_{gi} \tag{7.12}$$

Dabei wird jedem *Parameter* zusätzlich ein Gewicht x_{gi} zugeordnet. Je höher das Gewicht, desto stärker wird der Einfluss des *Parameters* auf die Zielfunktion, und um so weniger wird eine Abweichung des *Parameters* bei der Optimierung akzeptiert. Die Auswirkungen der Gewichtung ist an einem Beispiel in Abbildung 7.3 dargestellt.

In Abbildung 7.4 ist für eine Linie und ein Dreieck dargestellt, welche Lagen in einer Optimierung als gleichwertig betrachtet werden. Es fällt dabei auf, dass das Konzept der kleinsten Quadrate der Abweichung der Koordinaten von Eckpunkten nicht zwangsläufig zu ähnlichen Formen führt. Dies liegt darin begründet, dass in den Beispielen kein Zusammenhang zwischen den Eckpunkten in Form von *Constraints* berücksichtigt wurde.

Es wird vorgeschlagen, das vorgestellte Konzept für die *geometrische Anfrage* mit der Summe der quadrierten Abweichungen folgendermaßen anzuwenden:

- Im einfachsten Fall werden nur die Eckpunktkoordinaten der Eckpunkte genutzt.

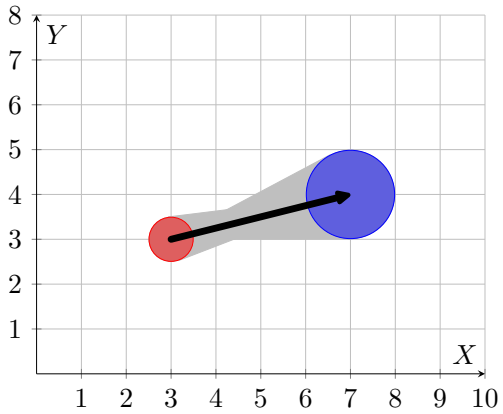
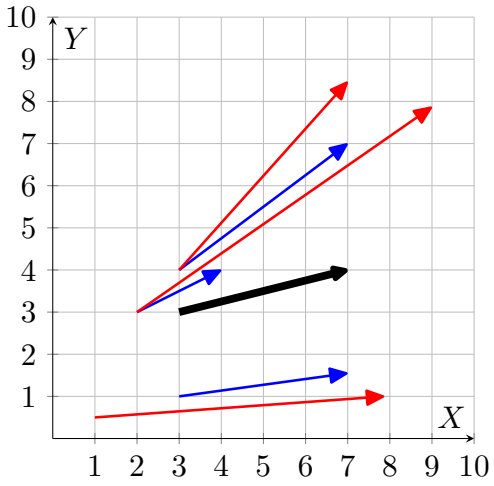
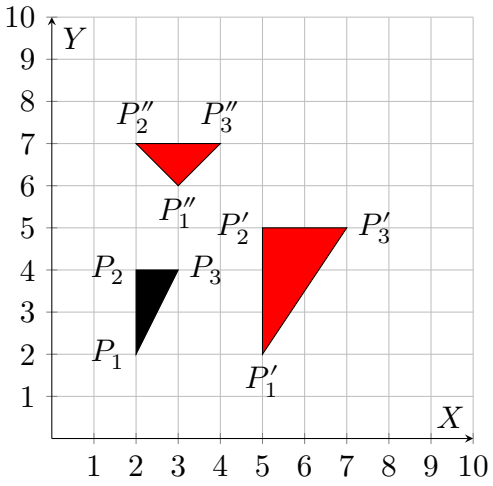


Abbildung 7.3: Mögliche Lage des Startpunkts (rot) und des Endpunkts (blau) einer Linie (schwarz) und die mögliche Lage der Linie im Zwischenraum (hellgrauer Bereich) bei einem Zielfunktionswert von 1; Die Zielfunktion entspricht der Summe der Quadrate der Abweichung der insgesamt vier Koordinaten der Endpunkte. Dabei besitzt der Startpunkt ein Gewicht von 2 und der Endpunkt von 1.



(a) Linie (als Pfeil dargestellt) in der Ausgangslage (schwarz) und bei den Zielfunktionswerten 10 (blau) und 20 (rot)



(b) Dreieck in der Ausgangslage (schwarz) und bei dem Zielfunktionswert 36 (rot)

Abbildung 7.4: Beispielhafte Lagen von *Geometrie-Objekten* bei unterschiedlichen Zielfunktionswerten. Die Zielfunktion entspricht der Summe der Quadrate der Abweichungen der Koordinaten der End- bzw. Eckpunkte

- Sollen bestimmte Eigenschaften der Objekte zwingend erhalten bleiben, so sind diese für die Optimierung mit zusätzlichen *Constraints* zu erzwingen. Dies sollte auch durch weiteres Einschränken der entsprechenden Wertebereiche der *Parameter* geschehen.

- Sollen bestimmte Eigenschaften der Objekte möglichst erhalten bleiben und können diese über *Parameter* spezifiziert werden, sind diese höher zu gewichten.
- Fehlen bestimmte *Parameter* und die sie in Zusammenhang-bringenden *Constraints*, sind diese dem Modell hinzuzufügen, zumindest für den Prozess der Optimierung. Vorschläge hierfür sind Schwerpunktskoordinaten und Winkel.

Die Anwendung des Konzepts wird an Beispielen in Kapitel 8 gezeigt.

7.6 Wertebereiche der Parameter

7.6.1 Berechnung der Wertebereiche

Für die Berechnung der Wertebereiche von *Parametern* wurden zwei Algorithmen implementiert.

Der erste Schritt ist die Berechnung einer *Hülle* mit Hilfe des *HC4 Prune-Operators* (siehe Abschnitt 7.5.2). Dieser ist effizient, was bei den getesteten Beispielen der Implementierung bestätigt wurde. Das umfangreichste Beispiel *Raum-begrenzt-durch-Wände* (siehe Abschnitt 8.5) besteht dabei aus 161 Funktionen und 165 Variablen, wobei davon 134 Variablen initial ein nicht degeneriertes Intervall enthalten. Dabei ergab sich für alle Beispiele auf 8 Logischen Prozessoren (Intel Core i7-4700MQ 2.50 GHz) mit 16 GB Arbeitsspeicher (DDR3-RAM) eine Berechnungszeit < 1 Sekunde. Da *HC4* – außer im Fall von baumartigen *Gleichungsgraphen* – keine Aussage über enthaltene Lösungen erlaubt und in der Regel das Ergebnis überschätzt, ist er ausschließlich zur ersten Abschätzung geeignet.

Wie in Abschnitt 5.4.7 beschrieben, erfolgt die eigentliche Berechnung der Wertebereiche durch die Bestimmung der *globalen Hüllen-Konsistenz* nach Cruz (2003). Dabei wird für jeden betrachteten *Parameter* die untere und obere Grenze seines Wertebereichs berechnet, für die jeweils eine Einzellösung existiert.

Der Algorithmus zur Berechnung der *globalen Hülle* basiert, wie das vorgestellte *Branch-and-Prune-Verfahren*, auf einer Zerlegung des Suchraums. Dazu wird ein Zerlegungsbaum eingeführt, der in jedem *Blatt* einen Teil des Suchraums als *Box* vorhält. Dabei ist die Idee, eine anfangs leere *Box*, welche die bisher bestimmte *globale Hülle* repräsentiert, mit jeder gefunden Lösung zu erweitern. Die Suche nach Lösungen wird solange fortgesetzt, bis ausgeschlossen werden kann, dass eine weitere Lösung existiert, die diese *Hülle* erweitern könnte.

Es wird ein Feld (englisch: array) angelegt (Index mit 1 beginnend), in der für jede *Variable* zwei aufeinanderfolgende Einträge gespeichert werden:

- Ein Eintrag mit ungeradem Index repräsentiert die untere Grenze.
- Ein Eintrag mit geradem Index repräsentiert die obere Grenze.

Jeder Eintrag enthält eine Liste von Referenzen auf *Blätter* des Zerlegungsbaums, die für die jeweiligen Grenzen der *Variablen* zu einer Vergrößerung der aktuellen *Hülle* führen könnten. Die Berechnung erfolgt für jeden Eintrag der Liste in drei Phasen, solange er nicht als irrelevant erkannt und verworfen wird:

1. Anwendung von *Prune-Operatoren* zur Verkleinerung der *Box* des zugehörigen *Blatts*. Sollten die *Prune-Operatoren* keine Verkleinerung bewirken folgt…
2. die Suche in einem *Blatt* nach einer Einzellösung, die eine möglichst starke Vergrößerung der *Hülle* – für die betrachtete Grenze der *Variablen* – bewirkt. Sollte die Lösung nicht mit Sicherheit die stärkste Vergrößerung der *Hülle* für die betrachtete Grenze aus diesem Suchraum bieten, oder sind Teile des Suchraums für andere Grenzen relevant, …
3. wird das *Blatt* bzw. die zugehörige *Box* zerlegt (englisch: branching).

Bei jeder Veränderung eines *Blatts* bzw. seiner *Box*, durch *Prune-Operatoren* oder Zerlegung, und bei jedem Finden einer weiteren Lösung, wird das Feld der Listen für die Grenzen der *Variablen* aktualisiert. Das Grundprinzip des Algorithmus ist in Algorithmus 7.4 dargestellt.

In der prototypischen Implementierung ist der Algorithmus vollständig umgesetzt, es existieren aber zwei Abweichungen zum originalen Prinzip:

- Zu Beginn des Verfahrens wird eine beliebige Einzellösung gesucht. Diese repräsentiert die initiale *Hülle*. Das hat zum einen den Vorteil, dass direkt für eine Anzahl der Grenzen der *Variablen* bzw. *Parametern* das beste Ergebnis erzielt wird und sie nicht weiter berücksichtigt werden müssen. Zum anderen existiert eine erste untere bzw. obere Grenze der Hülle, die für einzelne *Variablen* bzw. *Parameter* einen Teil des Suchraums von vornherein ausschließt.
- Das implementierte Suchverfahren in einem *Blatt* basiert im Gegensatz zum originalen Prinzip nicht auf einer Kombination aus *Branch-and-Prune-Verfahren* und lokaler Suche. Stattdessen wird der Grundalgorithmus (siehe Abschnitt 7.5.1) mit der *Methode der oberen Begrenzung* genutzt (siehe Abschnitt 7.5.4). Dabei stellt die

Algorithmus 7.4: *Globale Hülle nach Cruz* (angelehnt an Cruz, 2003, Abbildung 6.15)

Input: $\mathbf{B}_{init}$ is the initial box representing the complete search space
f is the set of functions to satistfy
V is the set of all variables of allfunctions in f
Output: no, one or more solutions represented as boxes

Preliminaries: n is the count of all variables in V
B_{in} is the current state of the global hull
T is a tree-datastructure which partitions the search space
L is an array (1..2n) of lists; each array-entry belongs to a variable's bound (odd → lower; even → upper); each list includes all leafs for the boundary that might increase the hull B_{in} and the $ACTION$, what to do next with each leaf

1 $B_{in} := \emptyset$
2 init T with B as root node
3 **foreach** *list* l *in* L **do**
4 $l := (B, ACTION\_PRUNE)$
5 **end**
6 **repeat**
7 **for** j=*1* **to** $2n$ **do**
`/* j/2 → number of variable v; j mod2 → (odd → lower; even → upper) */`
8 $(leaf_t, ACTION)$:= poll head of list of $L[i]$
9 B_t := box of $leaf_t$
10 $B_{relevant}$:= relevantSubbox(B_t, j)
11 **if** $ACTION$=$ACTION\_PRUNE$ **then**
12 pruneAction($leaf_t$,$B_{relevant}$,j)
13 **if** $ACTION$=$ACTION\_SEARCH$ **then**
14 searchAction($leaf_t$,$B_{relevant}$,j)
15 **if** $ACTION$=$ACTION\_SPLIT$ **then**
16 splitAction($leaf_t$,$B_{relevant}$,j)
17 **end**
18 **until** L *includes at least one entry with a LEAF AND a corresponding ACTION*

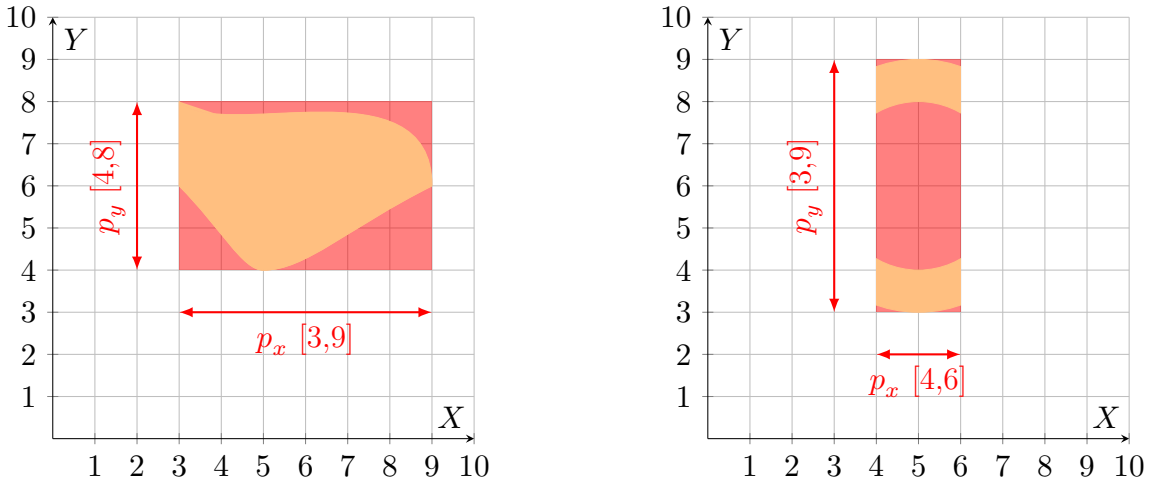
Optimierung die Suche nach jenem Wert dar, für den die aktuell betrachtete Grenze die größte Vergrößerung der aktuellen *Hülle* bietet. Werden dabei Einzellösungen gefunden, die sich nicht als die besten herausstellen, werden diese gegebenenfalls direkt zur Vergrößerung der aktuellen *Hülle* genutzt.

7.6.2 Visualisierung der Lösungsräume

Wurde die *globale Hülle* bestimmt, stellt sich die Frage, wie diese über die numerische Darstellung hinaus visualisiert werden kann.

Da das modellierte *Constraint-System* im Kern ein nichtlineares Gleichungssystem darstellt, existieren beliebige geometrische Formen des Lösungsraums von *Geometrie-Objekten*. Die *globale Hülle* stellt nur die Begrenzung der *Box* in die Richtung der jeweiligen Dimension dar.

Betrachtet man eine *Box*, welche ausschließlich die *globale Hülle* der Koordinaten eines Punkts darstellt, dann ist das geometrische Äquivalent der *Box* die *Achsen-ausgerichtete minimale Bounding-Box* (englisch: axis-aligned minimum bounding box; Abkz. AABB). Die Intervalle der einzelnen *Parameter* stellen die Abmessungen der *Bounding-Box* dar. Damit lässt sich der exakte Lösungsraum des Punkts in Form eines Hüllkörpers – Rechteck (in $\mathbb{R}^2$) oder als Quader (in $\mathbb{R}^3$) – umfassen. In Abbildung 7.5 werden Beispiele für die Lösungsräume und *Bounding-Boxen* gezeigt.

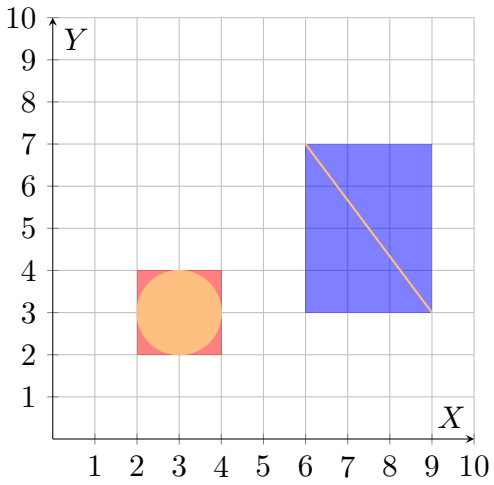


(a) Zusammenhängend **(b)** Disjunkt

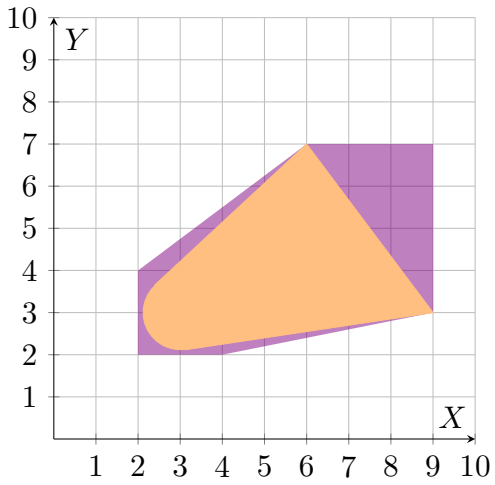
Abbildung 7.5: Visualisierung des Lösungsraums für Punkte: Exakter Lösungsraum (orange), *Bounding-Box* (rot)

Werden höhere *Geometrie-Objekte* als Punkte betrachtet, ist es möglich, auch für diese Punkte Hüllkörper als Repräsentation des Lösungsraums zu erzeugen. Werden ausschließlich *Geometrie-Objekte* mit geraden Kanten bzw. planaren Oberflächen verwendet, kann vereinfacht eine *Bounding-Box* berechnet werden, die alle *Bounding-Boxen* der Lage der Eckpunkte umhüllt. Eine genauere Eingrenzung ist mit der *konvexen Hülle* möglich. Diese stellt den kleinsten konvexen Polyeder dar, der alle betrachteten Punkte umschließt

(vgl. Dattorro, 2016, Abschnitt 2.3.2). Mit „betrachteten Punkten“ sind in diesem Fall nicht das *Geometrie-Objekt* Punkt, sondern die Eckpunkte der *Bounding-Boxen* der möglichen Lage der Punkte gemeint. Damit ist es möglich, für Linien, Polygone oder Polyeder einen Hüllkörper zu erstellen. Der mögliche Lösungsraum als *konvexe Hülle* einer Linie ist in Abbildung 7.6 dargestellt. In der prototypischen Implementierung ist die *konvexe Hülle* unter Nutzung der Bibliothek *QuickHull3D*[5] für 2D und 3D umgesetzt.



(a) Gemeinsame Hülle der Lage der Endpunkte als *Bounding-Boxen* (rot und blau)



(b) *Konvexe Hülle* der *Bounding-Boxen* der Endpunkte (lila)

Abbildung 7.6: Lösungsräume einer Linie bzw. derer Endpunkte; Exakter Lösungsraum (orange)

7.7 Diskussion

Die prototypische Implementierung enthält einen Grundstock an *Constraints*, *Geometrie-Objekten* und *Modellobjekten*. In der Umsetzung der *Constraints* können bei umfangreicheren *Constraints* mehrere Möglichkeiten existieren, diese durch verschiedene Gleichungen zu beschreiben. Das ist insofern unproblematisch, da durch den Menschen in der Modellierung die *Constraints* und nicht die zugrunde liegenden Gleichungen bearbeitet werden. Die Möglichkeiten, *Constraints* aus Gleichungen zusammenzusetzen, sind dabei vielfältig.

Betrachtet man die genutzten Algorithmen zur Optimierung und Bestimmung der gültigen Werte der *Parameter* zeigt sich, dass die gewählte Umsetzung des *Branch-and-Prune-Verfahrens* so allgemein wie möglich gehalten wird. Es können für die einzelnen Phasen beliebige Strategien hinzugefügt oder kombiniert werden. Beispielsweise können

[5] https://www.cs.ubc.ca/~lloyd/java/quickhull3d.html (abgerufen am 07.10.2019)

weitere *Pruning-Operatoren* entwickelt oder beliebige weitere Zielfunktionen für die Optimierung integriert werden.

Das umgesetzte Konzept für die *geometrische Nähe* mittels der *Methode der kleinsten Quadrate* der Abweichungen von der prototypischen Skizze stellt die Grundlage dar, beliebige Anfragen umzusetzen. Es ist leicht erweiterbar und detailliert konfigurierbar, indem über die Koordinaten der Punkte hinaus weitere *Parameter* hinzugefügt werden.

Die Berechnung und Visualisierung des Lösungsraums gestalten sich als schwierig. Mit dem Konzept der *Bounding-Box* für den Lösungsraum der Punkte in Kombination mit der *konvexen Hülle* für höherwertige Objekte steht zumindest ein Hilfsmittel zur Verfügung, welches die Lösungsräume mehr oder weniger (abhängig von der Struktur des Modells) eingrenzen kann. In Kombination mit der *geometrischen Anfrage* stellt dies ein geeignetes Werkzeug dar, welches den Menschen bei der Suche nach Lösungen in der berechneten Näherung des Lösungsraums unterstützen kann.

Es ist zu überlegen, ob zukünftig ein Algorithmus umgesetzt werden kann, der die Ränder des exakten Lösungsraums bestimmt. Der dafür hohe Aufwand folgt aus der Notwendigkeit, den Rand des exakten Lösungsraums zu bestimmen. Da der Rand aus Einzellösungen besteht, wäre ein Ansatz ihn schrittweise durch lokale Suchen der Lösungen zu bestimmen. Die Schwierigkeit besteht vor allem darin, dass sich ein Lösungsraum aus mehreren disjunkten Bereichen ergeben oder Löcher bzw. Blasen enthalten könnte.

Die prototypische Implementierung setzt aktuell nur auf *Geometrie-Objekte*, die auf der linearen Verbindung von Punkten basieren. Dazu gehören Linien und Polygone. Andere *Geometrie-Objekte* wie Kreise, Ellipsen, Freiformkurven und -flächen sind bisher noch nicht implementiert und untersucht worden.

Die Funktionalität der prototypischen Implementierung und die Auswirkungen der zugrunde liegenden Konzepte, sollen im folgenden Kapitel an Beispielen gezeigt und bewertet werden.

KAPITEL 8

Modellversuche und Ergebnisse

Nach den vorangegangenen Kapiteln, die das Modell und die Algorithmen beschreiben, werden in diesem Kapitel verschiedene Beispielmodelle vorgestellt. An diesen wird der Aufbau unterschiedlicher Modellinstanzen und die Durchführung von Modellversuchen mit den implementierten Algorithmen vorgenommen und ausgewertet. Im Anschluss werden die Ergebnisse bewertet.

8.1 Aufbau und Vorgehen

Es werden dreidimensionale Modelle verwendet, die mit Hilfe der gewählten Werte auf zweidimensionale Probleme reduziert wurden. Die Gründe hierfür sind, dass zweidimensionale Beispiele in zweidimensionalen Zeichnungen nachvollziehbarer erklärt werden können und sich der Berechnungsaufwand in Grenzen hält. Darüber hinaus ist speziell die Größe und Dichte der relevanten Teile der zu erläuternden *Constraint-Graphen* deutlich reduziert.

Das grundsätzliche Vorgehen bei den einzelnen Beispielen ist:

1. Beschreibung der Ziele und des Hintergrunds des Beispiels.
2. Beschreibung des Aufbaus, der Parametrisierung und der *Constraints*.
3. Durchführung der Algorithmen mit Interpretation und Bewertung der Ergebnisse. Dabei werden unter anderem die *globale Hülle nach Cruz* (siehe Abschnitt 7.6), die Hülle aus dem *HC4*-Algorithmus (siehe Abschnitt 7.5.2) ohne das *Branch-and-Prune-Verfahren*, die Optimierung der Parameterwerte (siehe Abschnitt 7.5.3) und

die *geometrische Anfrage* in Form von prototypischen Skizzen (siehe Abschnitt 7.5.5) betrachtet. Die numerischen Ergebnisse finden sich in Appendix B.

Zusätzlich sind die Beispiele in Varianten unterteilt, bei denen Eingangsdaten und/oder *Constraints* verändert werden. Dies dient zur Veranschaulichung des Einflusses der verschiedenen Möglichkeiten des Modellierens.

Gemäß der üblichen Konvention wird in diesem Beispiel die Konvention berücksichtigt, dass die Z-Achse entgegen der Richtung der Gravitationskraft gerichtet ist. Eine *Box* wird als erreichte *Einzellösung* betrachtet, wenn sie ausschließlich Intervalle der Breite $wid(X) < 0.01$ enthält. Alle Berechnungen wurden mit einer Präzision von mindestens 14 Stellen durchgeführt, dies entspricht der *double precision* nach IEEE Std 754-2008.

8.2 Beispiel: Rechteck-Wand

8.2.1 Ziele

Am Beispiel einer *Funktionalen Einheit* „Wand“, welche als übliches Bauteil in Bauwerksmodellen des Hochbaus anzusehen ist, sollen die Prinzipien des Aufbaus eines einzelnen Bauteils gezeigt werden. Dazu werden die *Parameter* der Wand so gewählt, dass sich ein *unterbestimmtes Constraint-System* ergibt. Es werden zuerst die möglichen Parameterwerte in Form der *globalen Hülle* berechnet und anschließend durch Optimierung nach Lösungen gesucht.

8.2.2 Aufbau

Der grundsätzliche Aufbau der Wand ist:

- Unter- und Oberseite sind kongruente Rechtecke in unterschiedlichen Ebenen, parallel zur XY-Ebene.
- Die Seitenflächen sind Rechtecke, welche Ober- und Unterseite verbinden.
- Die Wand besitzt die Punkte *P1* und *P2*, welche eine Längskante der Unterseite beschreiben.

Darüber hinaus kann die Wand durch folgende *Parameter* beschrieben werden:

Dicke ist der Abstand zwischen den beiden Längsseiten.

Höhe ist der vertikale Abstand zwischen Unter- und Oberseite.

Länge ist der Abstand zwischen den Stirnflächen.

Die Parametrisierung und die Bezeichnung der Eckpunkte sind in Abbildung 8.1 dargestellt.

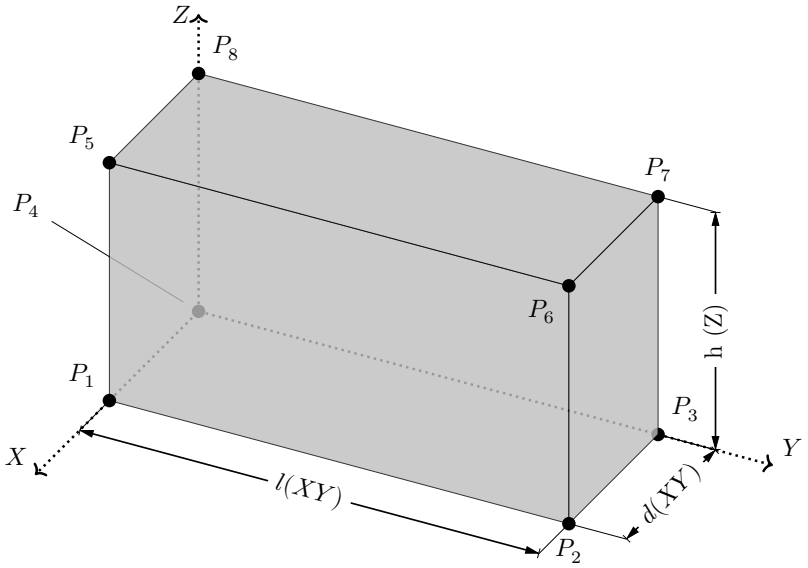


Abbildung 8.1: Parametrisierung der *Rechteck-Wand* (semi-transparent). Der Ursprung und die Achsrichtung X und Y sind beispielhaft, da lediglich die aufgespannte Ebene definiert sein muss.

Als Grundidee des Entwurfs wird davon ausgegangen, dass sich die Eckpunkte der Oberseite (P_5, P_6, P_7, P_8) bezüglich der Eckpunkte der Unterseite (P_1, P_2, P_3, P_4) genau um den Parameterwert der „Höhe“ in Z-Richtung befinden. Die Parametrisierung und die *Constraints* des Rechtecks in XY-Ebene ist in Abbildung 8.2 dargestellt. Dies wird mit Hilfe des *Erzeugungsoperators Polygon-Extrusion* in Richtung der Z-Achse um die „Höhe“ umgesetzt. Dabei wird das Rechteck der Unterseite extrudiert und so der *Solid* und damit das Rechteck der Oberseite erzeugt.

In Z-Richtung erfolgt die Parametrisierung mit dem *Parameter* „Höhe“. Der *konstruktiven Grundgedanke* ergibt sich mit folgender Erzeugungsreihenfolge:

1. Die Punkte P_1 und P_2 werden als gegeben betrachtet.

2. Deklaration einer *Punkt-Koinzidenz in Z*, die den Abstand in Z-Richtung zwischen P_1 und P_2 gleichsetzt.

3. Die *Parameter* „Höhe“, „Dicke“ und „Länge“ werden mit $(0, \infty)$ initialisiert.

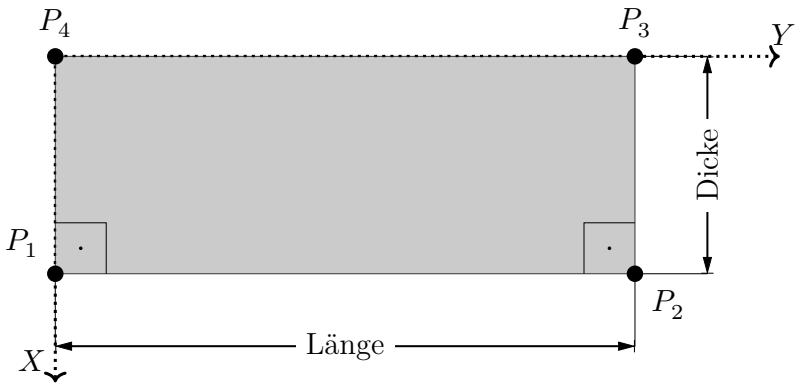


Abbildung 8.2: *Parameter* und *Constraints* der Unterseite der *Rechteck-Wand* in der XY-Ebene (Draufsicht).

4. Aufspannen der rechteckigen Grundseite mit einer *Linien-Extrusion* in XY-Ebene von der Linie $\overline{P_1P_2}$ um die „Dicke“.

5. Extrusion der Grundseite in Z-Richtung mit einer *Polygon-Extrusion* des Polygons in Z-Richtung um die „Höhe“. Dadurch entsteht ein Körper, bestehend aus den vier seitlichen Polygonen und den Polygonen der Unter- bzw. Oberseite.

6. Füge *Constraint Abstand zweier Punkte* zwischen P_1 und P_2 mit dem Abstand als *Parameter* „Länge“ ein.

Der sich daraus ergebende *Constraint-Graph* ist in Abbildung 8.3 dargestellt. In dem *Constraint-Graph* ist nicht direkt erkennbar, dass durch die Extrusion senkrecht zur XY-Ebene, alle Koordinaten bzw. *Parameter* der XY-Richtung von denen der Z-Richtung entkoppelt sind. Das bedeutet, dass im *Gleichungs-Graph* die *Parameter* „Länge“, „Dicke“ und alle X- bzw. Y-Koordinaten der Punkte einen Teilgraph bilden. Der andere dazu disjunkte Teilgraph besteht aus dem *Parameter* „Höhe“ und den Z-Koordinaten der Punkte.

Es handelt sich um ein *wohlbestimmtes* System, wenn die Koordinaten von P_1 und P_2, die „Dicke“ und die „Höhe“ valide reelle Zahlen sind. Der gewählte Aufbau ermöglicht es, dass die Wand ausschließlich als korrekter geometrischer Körper repräsentiert wird. Beschränkungen der *Parameter* auf sinnvolle Werte, wie zum Beispiel die „Höhe“ und „Dicke“ auf positive Werte, führen dazu, dass geometrisch degenerierte Zustände verhindert werden. Ein degenerierter Zustand tritt beispielsweise auf, wenn die Höhe 0 betragen und somit die Wand in einem geometrischen Körper mit einem 0-Volumen resultieren würde.

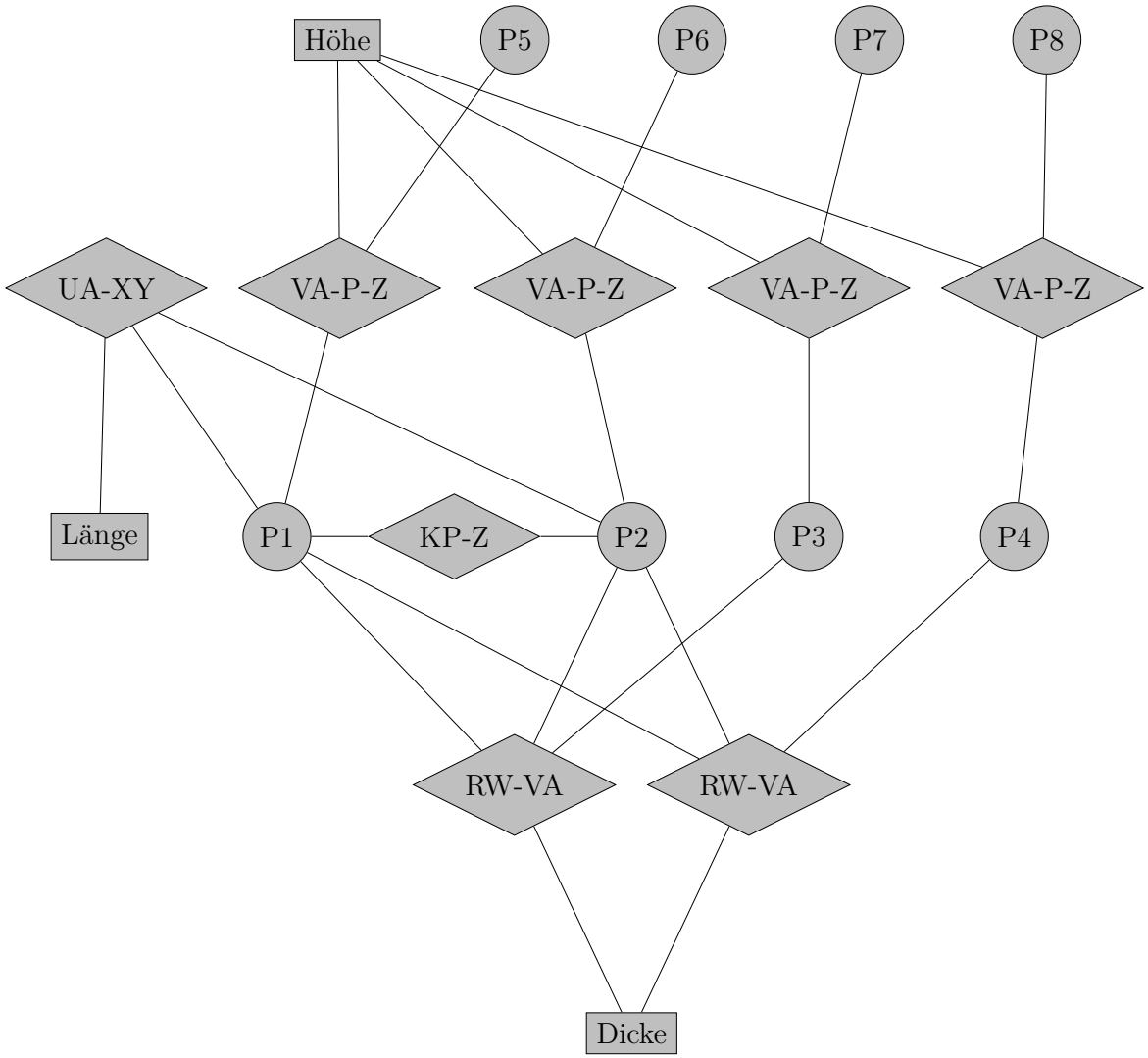


Abbildung 8.3: *Constraint-Graph* einer *Rechteck-Wand*. Darin sind ausschließlich Punkte (Kreis), *Parameter* (Rechteck) und die *Constraints* (Raute) zwischen diesen enthalten. Die *Geometrie-Constraints* sind *Koinzidenz von Punkten in Z* (KP-Z), *vorzeichenbehafteter Abstand von Punkten in Z-Richtung* (VA-P-Z) und *rechter Winkel mit vorzeichenbehaftetem Abstand* (RW-VA).

In den folgenden Berechnungen wird die Höhe der Wand auf einen festen Wert von 2.2 gesetzt. Dadurch kann das Beispiel als 2D-Problem in der XY-Ebene betrachtet werden. Eine Visualisierung der Wand in der prototypischen Implementierung ist in Abbildung 8.4 zu sehen.

8.2.3 Berechnung der globalen Hülle

Die Berechnung der *globalen Hülle nach Cruz* erfolgte für das Beispiel in zwei Schritten:

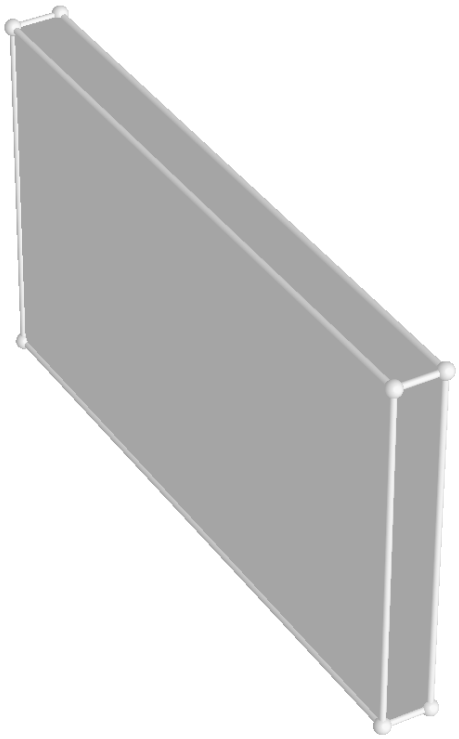


Abbildung 8.4: 3D-Visualisierung eines gültigen Zustands des Beispiels *Rechteck-Wand* aus der prototypischen Implementierung

1. Anwendung des *Prune-Operators HC4*.
2. Suche der *globalen Hülle* in der verbliebenen *Box*.

Dabei dient *HC4* zur schnellen Einschränkung der Startintervalle, während die Berechnung der *globalen Hülle* deutlich mehr Rechenzeit in Anspruch nimmt. Die Ergebnisse sind in Tabelle B.1 aufgelistet. Es ist zu erkennen, dass für die Koordinaten der „freien“ Punkte (keine Einschränkung der Startwerte der Koordinaten) $P3$ und $P4$ der *HC4*-Algorithmus ein Ergebnis mit zu breiten Intervallen liefert. Hingegen liefert die *globale Hülle nach Cruz* die korrekten Grenzen der Parameter, was im folgenden Abschnitt deutlich wird.

8.2.4 Optimierung

Es wurden verschiedene Optimierungen mit der *Methode der kleinsten Quadrate* durchgeführt. Dabei wurden insgesamt vier Zielfunktionen gebildet:

Max. Dicke: Für die Dicke wird eine Lösung nahe dem reellen Wert 1 gesucht. Da der Wert oberhalb der *globalen Hülle* liegt, entspricht dies der Suche nach dem Maximum.

Dicke und Länge: Es wird nach einer Zielfunktion mit den quadrierten Abweichungen der *Parameter* „Dicke“ zu 0.5 und „Länge“ zu 6 optimiert.

Skizze (a) Eine Zielfunktion über eine *geometrische Anfrage* der Eckpunkte der Unterseite der Wand. Die prototypische *Skizze (a)* liegt dabei so, dass für eine mögliche Lösung ausschließlich eine Verschiebung, aber keine Rotation notwendig ist.

Skizze (b) Eine Zielfunktion über eine *geometrische Anfrage* der Eckpunkte der Unterseite der Wand. Die prototypische *Skizze (b)* liegt dabei so, dass die Skizze zu allen möglichen Lösungen um ca. 90° verdreht liegt.

Die *geometrischen Anfragen* sind zusammen mit den Resultaten und dem exakten Lösungsraum (dieser wurde logisch bestimmt und nicht berechnet) in Abbildung 8.5 dargestellt. Die Ergebnisse der Optimierungen nach den vier Zielfunktionen ist in Tabelle B.2 zu sehen. Bei der Bestimmung der maximalen „Dicke“ wird eine Lösung erreicht, die wie zu erwarten der oberen Grenze der *globalen Hülle* entspricht. Bei der Optimierung nach der „Dicke“ und „Länge“, wird bei der Optimierung nur für die „Dicke“ der angegebene Zielwert erreicht, da die „Länge“ nicht größer als 5 werden kann. Das bedeutet, dass kein Kompromiss zwischen den Werten gemacht werden muss.

Nicht anders verhält es sich bei der *Skizze (a)* der *geometrischen Anfrage*. Hier zeigt sich das Ergebnis in Form einer Translation in den exakten Lösungsraum. Sieht man von der „Dicke“ ab, ist die Skizze bereits so ausgerichtet, wie die nächste mögliche Lösung am Rand des Lösungsraums. Hierbei kann man von einer intuitiven Lösung sprechen.

Die *Skizze (b)* der *geometrischen Anfrage* liegt verdreht zu allen möglichen Lösungen. Die eigentlich in X-Richtung liegende Seite P1-P2, befindet sich in der Skizze in Richtung der Y-Achse. In dem berechneten Ergebnis der Optimierung liegt der Eckpunkt P1 soweit in Y-Richtung wie möglich, und die „Dicke“ der Wand ist auf ein Minimum reduziert, da die Eckpunkte P3 und P4 in X-Richtung gedrückt werden.

8.2.5 Bewertung

Das Beispiel der Wand mit rechteckigem Grundriss zeigt, dass die Algorithmen Ergebnisse liefern, die nachvollziehbar sind. Die beabsichtigte „Nähe“ ist in den verschiedenen *geometrischen Anfragen* erkennbar.

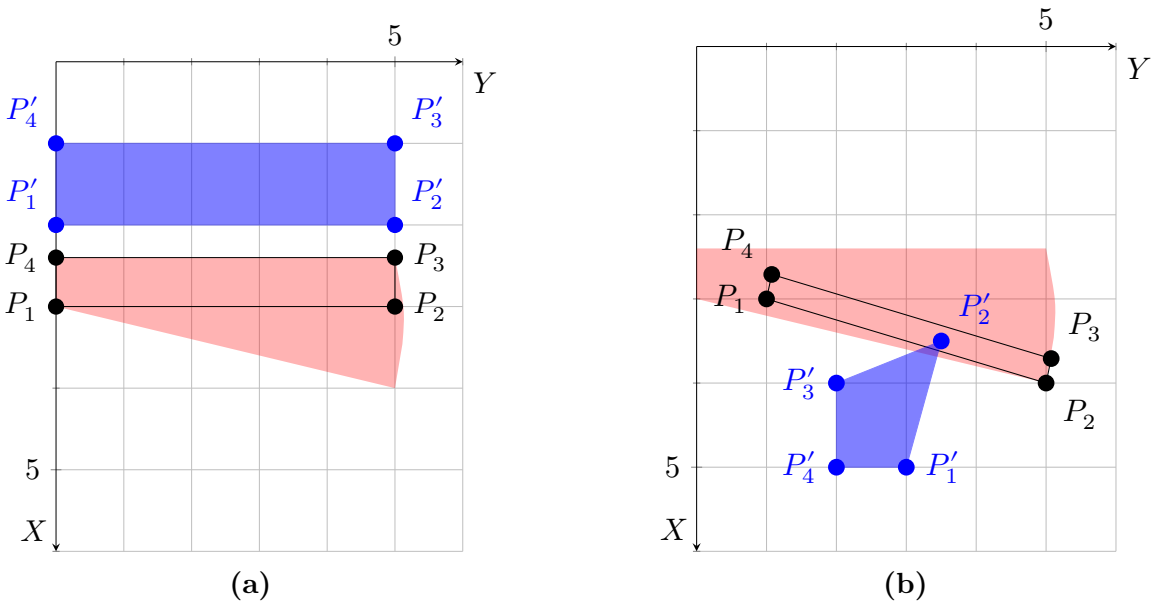


Abbildung 8.5: *Geometrische Anfragen* mit Skizze (blau) mit dem Ergebnis (schwarz umrandet) für das Beispiel *Rechteck-Wand*; der exakte Lösungsraum der Grundseite der Wand ist rot dargestellt.

8.3 Beispiel: Trapez-Wand

8.3.1 Ziele

Als weiteres Beispiel einer *Funktionalen Einheit* Wand soll diese mit einer Grundfläche in Form eines Trapezes untersucht werden. Die Wand besitzt dabei eine feste „Dicke“ über ihre Längsachse hinweg. Die Stirnflächen der Wand sind aber nicht zwangsweise parallel zueinander. Solche Wände bieten die typische Form, dass an Ecken mit beliebigem Winkel ein Anschluss zwischen zwei Wänden, gegebenenfalls unterschiedlicher „Dicke“, hergestellt werden kann. Dieser Anschluss wird als Gehrung bezeichnet.

Für die Berechnung ergibt sich gegenüber dem Beispiel der *Rechteck-Wand* folgende Besonderheit: Das Trapez ist gegenüber dem Rechteck ein deutlich flexiblerer Körper, da ausschließlich der Abstand und die Parallelität zweier gegenüberliegenden Seiten gewährleistet werden muss. Dafür muss in der Ebene des Trapezes die Orientierung der Punkte erhalten bleiben, sodass kein „Überschlagen“ im Sinne einer invertierten Orientierung möglich ist.

8.3.2 Aufbau

Der Aufbau erfolgt analog zum Beispiel *Rechteck-Wand*, mit der Abweichung, dass die Grundseite ein Trapez ist. Daraus ergeben sich statt einer „Länge“, eine „Länge“ der Seite $P1-P2$ und eine „Länge“ der Seite $P3-P4$. Diese werden jeweils mit einer *Constraint* für den Abstand der Punkte definiert. Das Trapez besitzt außerdem eine *Constraint* für den vorzeichenbehafteten Abstand der zwei Geraden durch $P1-P2$ bzw. $P4-P3$, welcher die „Höhe“ des Trapezes darstellt. Mit dieser *Constraint* ist auch die Parallelität sichergestellt. Zur Verhinderung einer falschen Orientierung wird eine *Constraint* für die positive Orientierung des Dreiecks $P4$, $P1$, $P3$ und eine für das Dreieck $P2$, $P3$, $P4$ genutzt. Der Aufbau ist in Abbildung 8.6 dargestellt. Der sonstige Aufbau ist dem Beispiel *Rechteck-Wand* in Abschnitt 8.2 zu entnehmen.

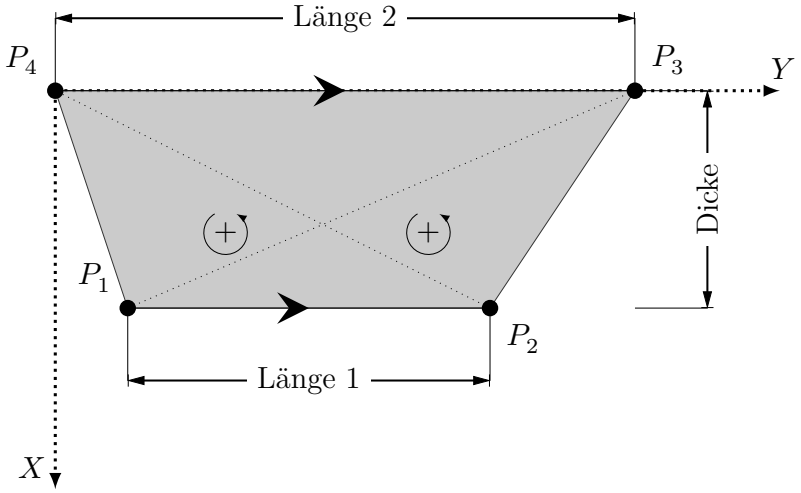


Abbildung 8.6: *Parameter* und *Constraints* der Unterseite der *Trapez-Wand* in der XY-Ebene (Draufsicht).

Der *Constraint-Graph* ist auf Grund der zusätzlichen *Constraints* für die zwei *Parameter* der „Länge“ und die positive Orientierung komplexer als bei der *Rechteck-Wand*. Eine reduzierte Darstellung ist in Abbildung 8.7 zu sehen.

8.3.3 Berechnung der globalen Hülle

Die Berechnung der *globalen Hülle* erfolgt wiederum in zwei Schritten. Die Ergebnisse sind in Tabelle B.3 zu sehen. Der *HC4*-Algorithmus bewirkt zwar eine Verkleinerung der *Such-Box*, es wird aber nicht die *globale Hülle* erreicht. Für die „Länge $\overline{P1-P2}$“ ergibt sich ein auf den ersten Blick überraschendes Ergebnis, dass die Werte von *HC4* und der *globalen Hülle* übereinstimmen. Da sich die maximale „Länge $\overline{P1-P2}$“ dann einstellt,

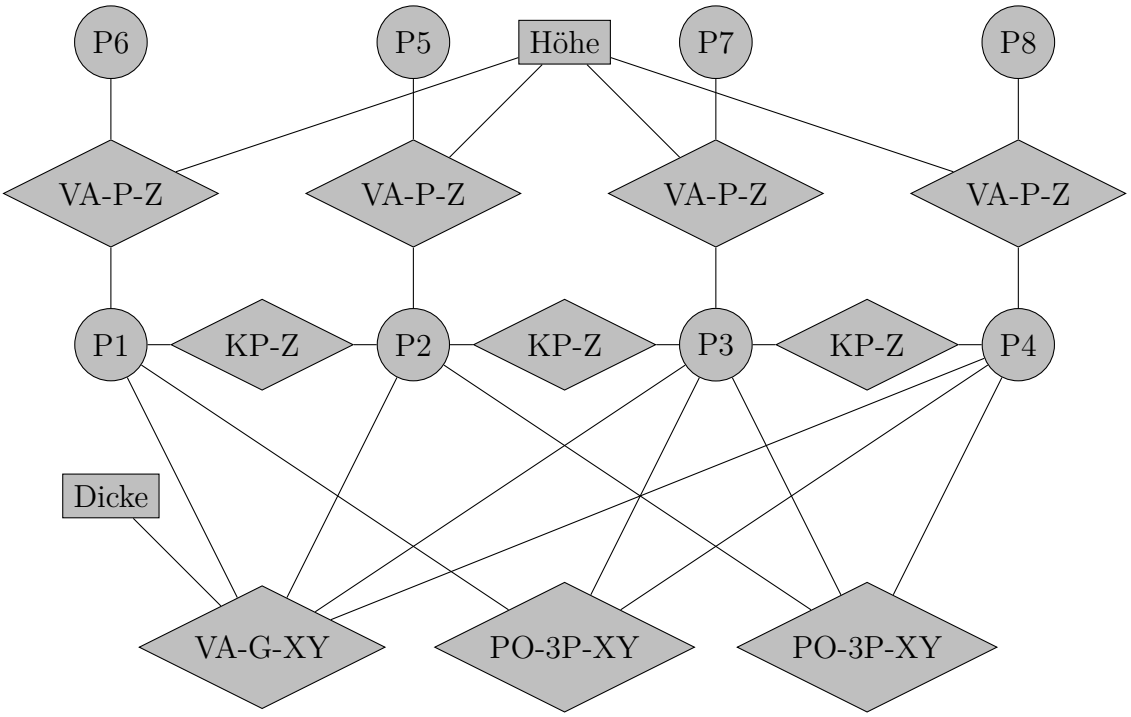


Abbildung 8.7: *Constraint-Graph* des Beispiels *Trapez-Wand*. Darin sind ausschließlich Punkte (Kreis), *Parameter* (Rechteck) und die *Constraints* (Raute) zwischen diesen enthalten. Nicht enthalten sind die *Constraints* für die Längen der Seiten. Die *Geometrie-Constraints* sind *Koinzidenz von Punkten in Z* (KP-Z), *Vorzeichenbehafteter Abstand von Punkten in Z-Richtung* (VA-P-Z) und *Positive Orientierung von 3 Punkten* (PO-3P-XY).

wenn diese sich auf einer Geraden parallel zur Y-Achse bei $X = 3$ befinden, ist der Wert zwischen *HC4* und der *globalen Hülle* in diesem Fall identisch.

Zusätzlich wurde bei diesem Beispiel der Lösungsraum der Linie zwischen $P1$ und $P2$ entsprechend dem in Abschnitt 7.6.2 vorgeschlagenen Konzept visualisiert. Das Ergebnis ist in Abbildung 8.8 dargestellt.

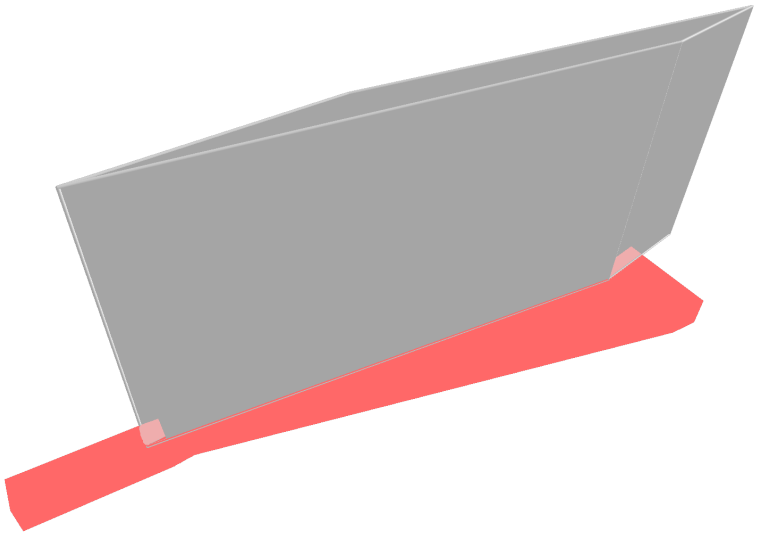


Abbildung 8.8: Visualisierung des Lösungsraums (rot) der Linie $P1$ und $P2$ (Unterkante vorne) des Beispiels *Trapez-Wand* in der prototypischen Implementierung

8.3.4 Optimierung

Es wurden drei Optimierungen durchgeführt:

Maximale Dicke: Es wurde die maximale Dicke bestimmt, dazu wurde der Wert der oberen Grenze der *globalen Hülle* als Ziel für die Optimierung genutzt.

Skizze (a): Es wurde die gleiche Skizze wie in der *Skizze (b)* aus dem Beispiel *Rechteck-Wand* gewählt.

Skizze (b): Die Skizze enthält einen relativ großen Abstand zwischen $P3$ und $P4$. Zusätzlich wurde in der Zielfunktion, neben den Koordinaten der Punkte, auch die Abweichung der „Länge $\overline{P3-P4}$“ zu dem Wert 6 berücksichtigt. Dies zeigt, dass in einer *geometrischen Anfrage* auch Werte von *Parametern* berücksichtigt werden können, deren abweichender Wert nicht dem aus der Skizze abgeleiteten entsprechend muss.

Die Ergebnisse der Parameterwerte durch die Optimierungen sind in Tabelle B.4 aufgeführt. Für die Maximierung der „Dicke“ ergibt sich das zu erwartende Ergebnis von 0.6. Die berechneten Lösungen aus den Skizzen sind in Abbildung 8.9 dargestellt. Dabei fällt auf, dass in *Skizze (a)* die Punkte $P3$ und $P4$ zusammenfallen, während sie in *Skizze (b)* auf Grund der Berücksichtigung der „Länge $\overline{P3-P4}$“ und des deutlich größeren

Abstands in der Geometrie der Skizze weiter auseinander liegen. Für $P1$ und $P2$ stellt sich in beiden Skizzen dasselbe Ergebnis ein. Für $P1$ wurde der Zielwert auf gleicher Höhe oder weiter rechts angegeben. In beiden Skizzen wurde für $P2$ der maximale X-Wert erreicht, obwohl in ein Ergebnis für $P2$ möglich war, welches eine kleinere Abweichung dargestellt hätte. Dieses stellte sich aber nicht ein, da die Abweichung der anderen Punkte überwiegt und zusätzlich die Mindestdicke der Wand von 0.3 erhalten bleiben muss.

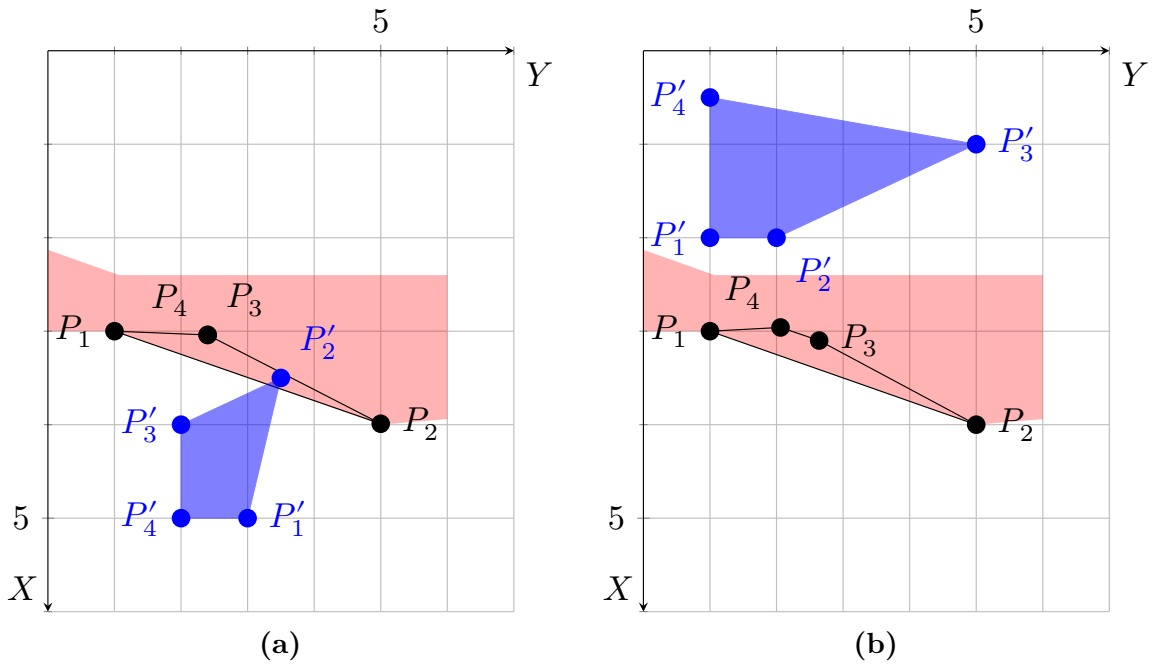


Abbildung 8.9: *Geometrische Anfragen* mit Skizze (blau) mit dem Ergebnis (schwarz umrandet) für das Beispiel *Trapez-Wand*; der exakte Lösungsraum der Grundseite der Wand ist rot dargestellt.

8.3.5 Bewertung

Das Beispiel der Wand mit dem trapezförmigen Grundriss macht deutlich, dass für komplexere Formen deutlich mehr *Constraints* benötigt werden, um die Form in einem validen Zustand zu erhalten. Die Ergebnisse der Algorithmen zeigen, dass zusätzlich *Parameter* in der *geometrischen Anfrage* berücksichtigt werden können, um bestimmte Eigenschaften in der Optimierung hervorzuheben.

8.4 Beispiel: Valides Polygon

8.4.1 Ziele

Die Grundlage und ein Teil der Durchführung des Beispiels wurden von Hoffmann und Kim (2001) vorgestellt. Dabei werden die möglichen Wertebereiche von einzelnen, festen *Constraints* (der Abstand ist kein Intervall, sondern ein reeller Wer) bestimmt für den Fall, dass sie gelöst werden müssen. Die Nebenbedingung ist, dass ein Polygon bzw. seine Kanten topologisch erhalten bleiben müssen. Das Beispiel soll zeigen, es ist unter Nutzung von Intervallen und *Constraints* möglich, dieselben Ergebnisse wie Hoffmann und Kim (2001) zu erreichen. Es wird anschließend die Problemstellung erweitert, indem die Wertebereichen der *Parameter* bestimmt werden, wenn zwei *Constraints* gleichzeitig gelockert werden. Dieses Problem wird von Hoffmann und Kim nicht gelöst.

Ziel des Beispiels ist es, die *globale Hülle* für unterschiedliche Konfigurationen der *Parameter* zu bestimmen. Für die Lockerung von mehreren *Constraints* soll zusätzlich eine Optimierung durchgeführt werden. Da es sich – wie im folgenden Abschnitt genauer erklärt wird – eigentlich um ein eindimensionales Problem handelt, sind die Ergebnisse der Optimierung gut nachvollziehbar.

8.4.2 Aufbau

Da alle betrachteten Kanten des Polygons parallel zur X-Achse bzw. horizontal liegen, kann dies, wie von Hoffmann und Kim vereinfacht angenommen, als eindimensionales Problem in Y-Richtung betrachtet werden. Die Kanten werden dazu mit H_i mit $i \in (0..11)$ bezeichnet und können für das Problem als Geraden betrachtet werden. Bei den verwendeten *Constraints* handelt es sich so ausnahmslos um den Abstand zweier Geraden. Alle *Constraints* definieren den vorzeichenbehafteten Abstand zweier Punkte in Y-Richtung, wobei jeder Punkt eine der Geraden bzw. Kanten repräsentiert. Die *Constraints*, welche den festen Abstand zwischen Geraden definieren – für dieses Beispiel als *blaue Constraint* bezeichnet –, werden als C_i mit $i \in (1..11)$ benannt. Zur Fixierung des Systems wird die Y-Koordinate der Kante H_{10} auf 0 festgelegt

Aufgrund der in Hoffmann und Kim (2001) nicht explizit vorhandenen Auflistung der *Constraints*, welche notwendig sind, die Topologie des Polygons zu erhalten, werden diese für dieses Beispiel zusätzlich definiert. Die Topologie-erhaltenden *Constraints* – für dieses Beispiel als *rote Constraint* bezeichnet – sind dabei grundsätzlich mit dem initialen Abstand $(0, \infty)$ versehen. Sie werden im Beispiel als C_{A-B} bezeichnet, wobei A die Nummer der Kante ist, die einen kleineren Y-Wert besitzen soll als die Kante B, damit das Polygon topologisch erhalten bleibt. Dies dient dazu, zwischen zwei Kanten des Polygons einen Mindestabstand größer 0 zu definieren und durch die Vorzeichenbehaftung

sicherzustellen, dass kein „Überschlagen“ stattfindet. In Abbildung 8.10 ist das System dargestellt. Die *blauen Constraints* lassen sich als Baum anordnen, während durch die *roten Constraints* verschiedene Zyklen integriert werden. Dies ist im *Constraint-Graphen* erkennbar, siehe Abbildung 8.11.

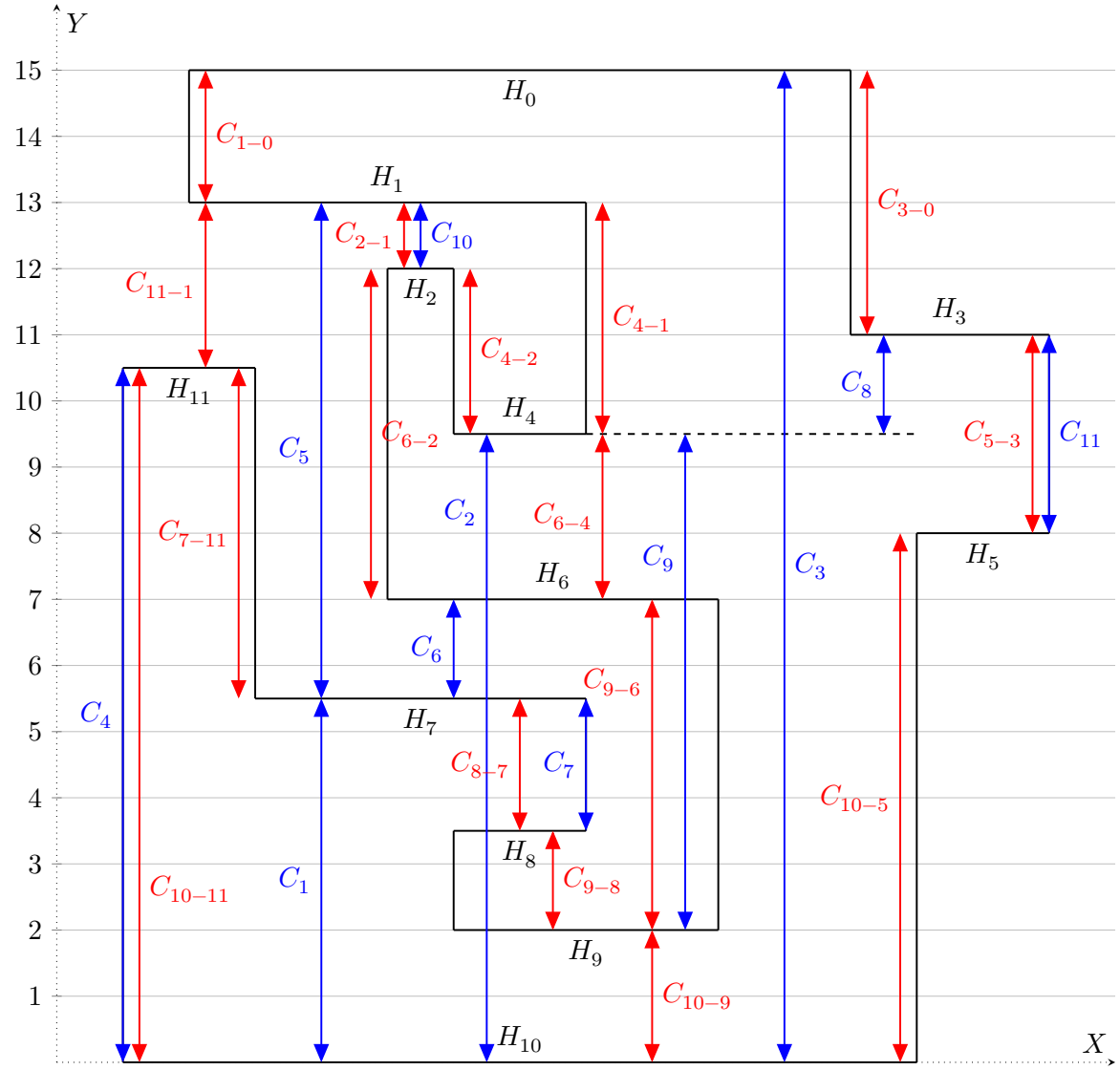


Abbildung 8.10: Draufsicht mit Parametrisierung des Beispiels *Valides Polygon* (In Anlehnung an Hoffmann und Kim, 2001, Abbildung 5). Alle *Constraints* (blau und rot) sind vom Typ *vorzeichenbehafteter Abstand zweier Punkte in Y. Schwarze Constraints* sind mit festem Abstand, *rote Constraints* besitzen einen Abstand von $(0, \infty)$.

8.4.3 Bestimmung der Startkonfiguration

Die Startkonfiguration stellt die Parameterwerte dar, wenn keine *Constraints* gelockert werden. Da die *blauen Constraints* feste Abstände darstellen und mit den horizontalen Kanten des Polygons einen Baum bilden, ist das System in der Ausgangslage *wohlbestimmt* und kann nicht verändert werden, ohne eine *Constraint* zu verletzen. Aus diesem Grund wurde vor der eigentlichen Durchführung die Ausgangslage der Wertebereiche der Abstände der *roten Constraints* bestimmt. Dabei liefert die *globale Hülle nach Cruz* und *HC4* dasselbe Ergebnis, wie in Tabelle B.5 aufgeführt. Es existieren dafür zwei Erklärungen. Da es sich um ein *wohlbestimmtes System* handelt, umfasst die *globale Hülle nach Cruz* genau die eine mögliche Lösung. Der *HC4*-Algorithmus findet diese ebenfalls, da theoretisch der Wertebereich des Abstands für jede *rote Constraint* ausschließlich aus der Kombination der festen *blauen Constraints* berechnet werden kann. Das bedeutet, die Berechnung findet auf Basis der diskreten Parameterwerte der *blauen Constraints* statt, bei der ausschließlich die mathematischen Operatoren + und − Anwendung finden. Eine Überschätzung der Ergebnisse findet somit nicht statt.

Die im Folgenden betrachteten Konfigurationen unterscheiden sich dadurch, dass in der ersten eine *schwarze Constraint* gelockert wird und anschließend die *globale Hülle* bestimmt, wohingegen in der zweiten zwei *schwarze Constraints* gelockert werden, die *globale Hülle* bestimmt wird und eine Optimierung stattfindet.

8.4.4 Konfiguration: Lockerung einer schwarzen Constraint

In dieser Konfiguration wird die *schwarze Constraint* C_2 gelockert, indem ihr Abstand auf $[-100, 100]$ gesetzt wird. Anschließend werden die Werte der *globalen Hülle* bestimmt. Diese sind in Tabelle B.6 abgebildet. Dabei tritt ein Unterschied zwischen der *globalen Hülle nach Cruz* und *HC4* zu Tage. Beide liefern zwar für die Y-Werte der Kanten des Polygons dieselben Ergebnisse, aber für den Abstand von C_{5-3} wird mit *Cruz* eine diskrete Lösung errechnet (mit erwartetem numerischem Fehler), während *HC4* zu einem Intervall der Breite 6.5 führt. Der Grund dafür liegt in dem dazugewonnenen Spielraum der beiden Kanten H_3 und H_5. Diese können sich als Paar – über die *Constraint* C_{11} fest verbunden – zwischen H_{10} und $H0$ frei bewegen. Die Y-Werte dieser Kanten sind Intervalle, wie in den Ergebnissen zu sehen ist. Da die *Constraint* C_{5-3} direkt von diesen Y-Werten abhängt, wird auch sie einen Intervallwert annehmen. *HC4* kann zwar C_{11} berücksichtigen, dies geschieht aber ausschließlich für die untere und die obere Grenze von H_5 in indirekten Bezug zu H_4.

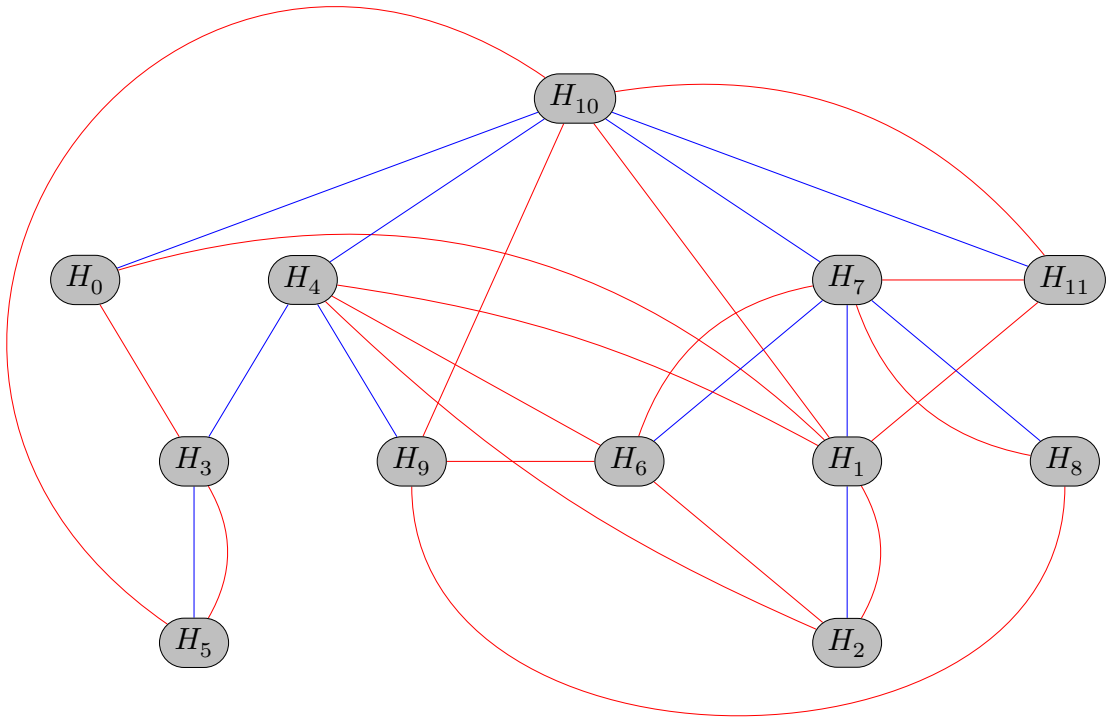


Abbildung 8.11: *Constraint-Graph* des Beispiels *Valides Polygon*; die horizontalen Linien bilden die Knoten; alle *Constraints* sind in diesem Fall als Kanten dargestellt: Fester Abstand (blau) und positiver Abstand in y-Richtung (rot)

Das Ergebnis für die betrachtete *Constraint* C_2 ist hingegen bei beiden Algorithmen identisch und stimmt mit dem von Hoffmann und Kim (2001) überein. Es besagt, dass in Folge der Lockerung von C_2 die Kante H_4 in Bezug auf ihre Ausgangslage von 9.5 zwischen -2 und 1.5 bewegt werden kann.

8.4.5 Konfiguration: Lockerung zweier schwarzer Constraints

In dieser Konfiguration werden die zwei *schwarzen Constraints* C_2 und C_5 gelockert, indem ihr Abstand auf $[-100, 100]$ gesetzt wird. Die Ergebnisse der *globalen Hülle* sind in Tabelle B.7 abgebildet. Der Unterschied zwischen der *globalen Hülle nach Cruz* und *HC4* tritt im Vergleich zu der Konfiguration bei der Lockerung nur einer *schwarzen Constraint* deutlicher hervor, da neben dem einen *Parameter* bei der Lockerung einer *schwarzen Constraint* bereits drei weitere unterschiedliche Intervalle aufweisen. Dabei sind wieder ausschließlich *Parameter* der *roten Constraints* betroffen. Alle anderen stimmen überein. Die Gründe sind identisch mit denen aus der anderen Konfiguration: Die *roten*

Constraints werden direkt oder transitiv durch *schwarze Constraints* überbrückt. Dies wird in der *globalen Hülle nach Cruz* berücksichtigt.

Für die Optimierung wurde von der *globalen Hülle* ausgegangen und es wurden zwei Zielfunktionen untersucht. Zum einen wurde für C_2 eine Lösung gesucht, die dem Wert 11 nahe kommt, zum anderen wurde eine Skizze vorgegeben, in der die Y-Werte von $H2$ bei 13 und von $H9$ bei 3 liegen. Anschließend wurde die *Summe der Quadrate* von $H2$ und $H9$ minimiert. Die numerischen Ergebnisse sind in Tabelle B.8 aufgeführt. Dabei wurde im ersten Durchgang für C_2 eine Lösung gefunden. Diese muss auf Grund der oberen Grenze der *globalen Hülle nach Cruz* existieren. Für die Skizze ergibt sich ein Ergebnis, welches erkennen lässt, dass beide Werte gleich gewichtet wurden. Zwar wäre für C_2 das mögliche Ergebnis von 11 näher an 13, aber durch die Berücksichtigung von 5 für C_5 wurde eine Lösung gefunden, die beide Werte annähert.

8.4.6 Bewertung

Das Beispiel zeigt die Möglichkeit mit der prototypischen Implementierung ein *Constraint-System* zu berechnen, in dem der *Constraint-Graph* keinen Baum darstellt, sondern Zyklen enthalten kann. Es ist möglich, zumindest in eine Koordinatenrichtung eine polygonale Grundform mit *Constraints* zu erhalten, ohne dass es zu topologischen Problemen oder geometrischen Überschneidungen kommt.

8.5 Beispiel: Raum-begrenzt-durch-Wände

8.5.1 Ziele

In diesem Beispiel soll gezeigt werden, dass es möglich ist, mehrere *Modellobjekte* mit *Constraints* in Zusammenhang zu bringen und die Wertebereiche von abgeleiteten *Parametern* wie „Fläche“ oder „Umfang“ zu bestimmen.

8.5.2 Aufbau

Es wurden vier Wände mit trapezförmiger Grundseite modelliert (siehe Abschnitt 8.3), die an den Ecken in Form einer Gehrung verbunden sind. Dies wird umgesetzt, indem an den Berührungsflächen die unteren Kanten mit je einer *Constraint Linien-Koinzidenz* verbunden werden. Die Wände begrenzen in ihrer Mitte einen Raum bzw. eine Raumfläche mit einem polygonalen Grundriss. Die Begrenzung wird ebenfalls über *Linien-Koinzidenz* zwischen der Unterkante der Wand und der Außenkante des Raums erreicht. Der Raum

besitzt neben den Koordinaten seiner *Eckpunkte* die *Parameter* „Umfang“ und „Fläche“. Diese wurden jeweils mit den *Constraints* für die *Länge eines Kantenzugs* bzw. *Fläche eines Polygons* mit den Eckpunkten des Flächen-bestimmenden Polygons verknüpft. Zwischen zwei sich gegenüberstehenden Wänden wurden zusätzlich *Constraints* zwischen den Linien der Unterkanten gesetzt, die diese parallel halten. Ein Bild des Modells aus der prototypischen Implementierung ist in Abbildung 8.12 gezeigt. Auf Grund der großen Anzahl an *Constraints* sind im *Constraint-Graph* ausschließlich die *Modellobjekte* und die zwischen ihnen gesetzten *Constraints* enthalten. Dieser *Constraint-Graph* ist in Abbildung 8.13 dargestellt.

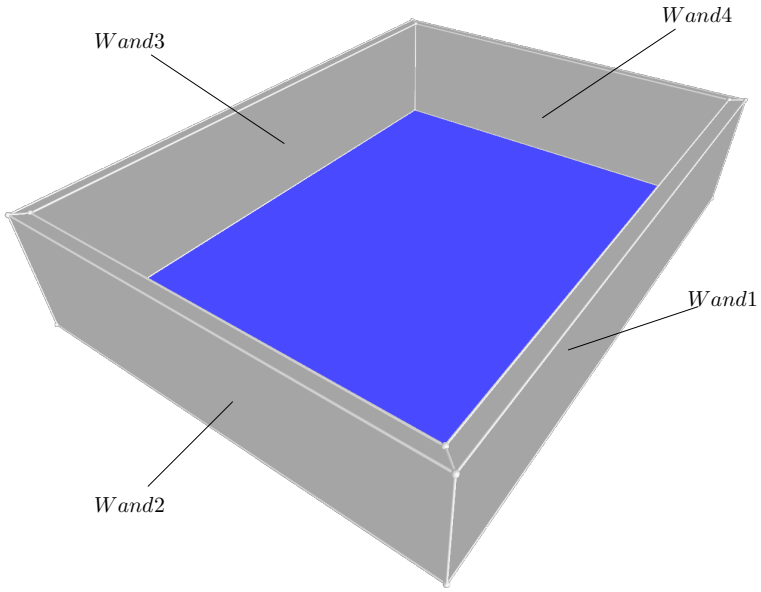


Abbildung 8.12: 3D-Visualisierung eines gültigen Zustands des Beispiels *Raum-begrenzt-durch-Wände* aus der prototypischen Implementierung mit hinzugefügten Bezeichnungen. Der Raum ist als Fläche blau dargestellt.

Berechnung der globalen Hülle

Die Ergebnisse der Berechnungen sind in Tabelle B.9 aufgeführt. Sie zeigen zwischen *HC4* und der *globalen Hülle* speziell bei der „Fläche“ und dem „Umfang“ des Raums große Unterschiede. Dies ist zurückzuführen auf die Überschätzung der Intervalle in der Berechnung durch mehrfaches Auftreten von Variablen in den zugehörigen Gleichungen. Da der Raumumfang aus den Eckpunkten des Raums berechnet wird, werden die relativ kleinen Unterschiede in deren Koordinaten zu einem großen Unterschied in der resultierenden Fläche. Die Berechnungszeit der *globalen Hülle* steigt bei solchen Überschätzungen deutlich an.

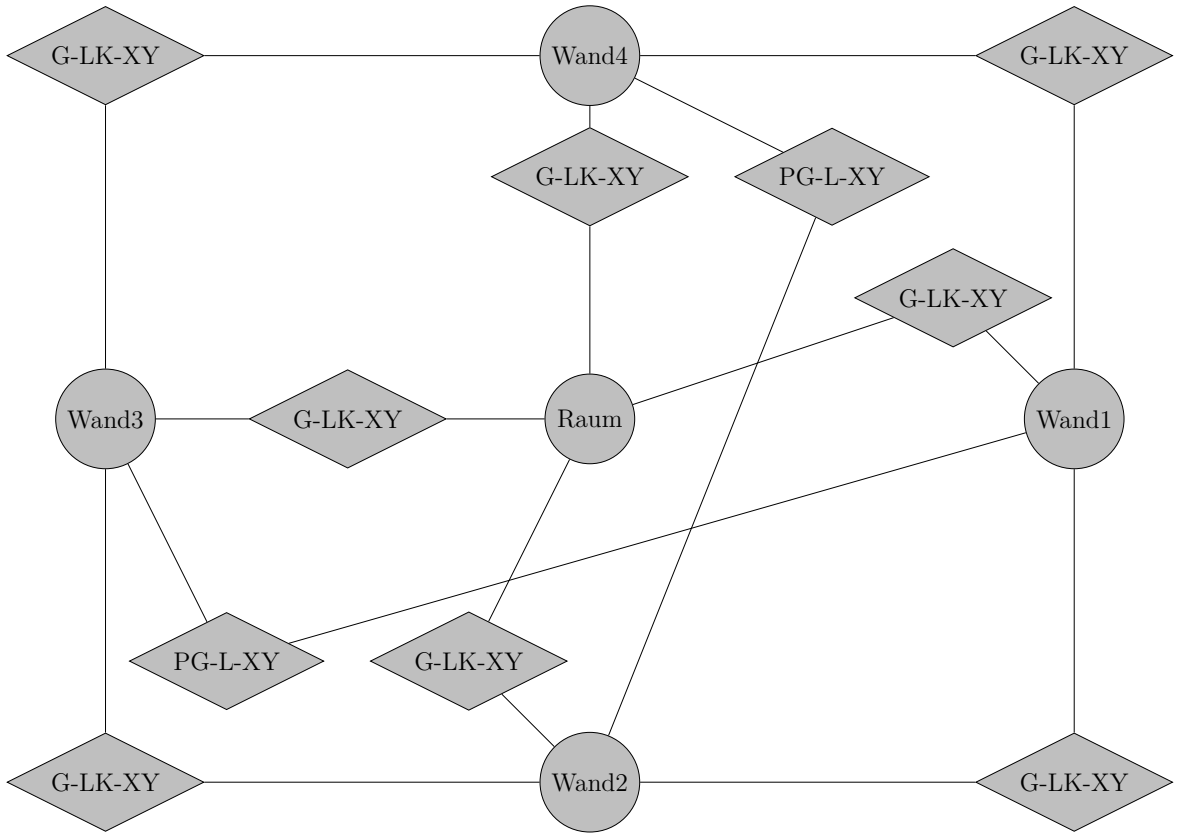


Abbildung 8.13: *Constraint-Graph* des Beispiels *Raum-begrenzt-durch-Wände*. Darin sind ausschließlich *Modellobjekte* (Kreis) und die *Constraints* (Raute) zwischen diesen dargestellt. Die *Geometrie-Constraints* sind *Gerichtete Koinzidenz von Linien* (G-LP-XY) und *Parallele Geraden durch Linien* (PG-L-XY)

8.5.3 Bewertung

Das aufgeführte Beispiel zeigt, dass auch komplexere Modelle mit mehreren *Modellobjekten* erstellt und berechnet werden können. Die *Constraint* für den Zusammenhang zwischen Eckpunkten des Polygons und der Polygonfläche basiert auf der Gaußschen Trapezformel. In dieser Formel ist jede Koordinate zweimal enthalten, was dazu führt, dass es aufgrund des *Intervall-Abhängigkeits-Problems* (siehe Abschnitt 5.3.3) zu einer deutlichen Überschätzung der Intervalle kommt. Das bedeutet in der Konsequenz, die Berechnungszeit der *globalen Hülle* steigt bei solch einem Beispiel stark an.

8.6 Fazit der Beispiele

Die Beispiele zeigen, dass das gewählte Konzept der *geometrischen Anfrage* geeignet ist, eine Form der geometrischen Nähe zu bestimmen. Auch die Berechnung der *globalen Hülle nach Cruz* ist möglich und stellt ein Mittel dar, die Wertebereiche der *Parameter* zu bestimmen. Das bedeutet, dass *Wissen* in Form von *Parametern* und *Constraints* mit diesen Algorithmen ausgewertet werden kann.

Das Beispiel *Valides Polygon* macht deutlich, dass gegenüber den vergleichbaren Forschungen ein Ergebnis erzielt werden kann, welches über das bisher erreichte hinausgeht. Die Erfahrungen, die bei der Erstellung der Beispiele gemacht wurden, liegen darin, dass *Funktionale Einheiten* erst ausgiebig für ihren Einsatz getestet werden müssen. Sind sie einmal erstellt und ihre Funktionalität ist sichergestellt, lassen sie sich mit anderen *Funktionalen Einheiten* sinnvoll zusammensetzen. Damit ist der Aufbau komplexerer Systeme möglich.

Die in dieser Arbeit vorgestellten Beispiele sind im Umfang und der Komplexität begrenzt. Zum einen sind sie so deutlich anschaulicher zu erklären, zum anderen ist die prototypische Implementierung nur teilweise optimiert. Die während der Bearbeitung der Beispiele umgesetzten Optimierungen der Implementierung zeigten bereits ein deutlich verbessertes Laufzeitverhalten.

KAPITEL 9

Zusammenfassung, Fazit und Ausblick

9.1 Zusammenfassung und Fazit

In dieser Arbeit wurde untersucht, wie *Wissen* in der Modellierung der Geometrie von digitalen Bauwerksmodellen integriert werden kann. Dazu wurde festgestellt, dass *Wissen* ein Aspekt eines solchen Modells ist und den Zusammenhang zwischen und innerhalb des semantischen und des geometrischen Aspekts darstellt. Das *Constraint-basierte Modellieren* zeigt sich als eine Möglichkeit, *Wissen* in Form der *Constraints* zu berücksichtigen. Den Ansatz des *Mengen-basierten Entwurfs* digital umzusetzen und dabei alle Werte der *Parameter* als Intervalle zuzulassen, ist in diesem Zusammenhang sinnvoll.

Der aktuelle Stand der Technik und der Forschung wurde beleuchtet. Dabei stellte sich heraus, dass aktuelle Softwareprodukte keine Intervalle unterstützen und *Constraint-basiertes Modellieren* von Geometrie nur eingeschränkt Einsatz findet. Es wurde gezeigt, dass der aktuelle Stand der Forschung zwar Ansätze zu *unterbestimmten* Systemen und der Verwendung von Intervallen verfolgt, die Ergebnisse aber unter Einschränkungen zustande kommen. Weder die Berechnung der gültigen Parameterbereiche, noch die Berechnung einer geometrischen „Nähe“ mit der *geometrischen Anfrage* wurden allgemein als gelöst betrachtet.

Mit dem *Constraint-basierten Modellieren* auf Basis von Intervallen wurde ein Verfahren genutzt, mit dem die Berechnung der gültigen Parameterbereiche und die *geometrischen Anfrage* möglich sind. Es wurden dafür die verschiedenen Ansätze zur Lösung und Optimierung von *Intervall-Constraint-Systemen* vorgestellt und ihre Grenzen beschrieben.

Auf Basis der verschiedenen Ansätze wurde in einem ersten Schritt ein Modell entwickelt, welches es ermöglicht, *Constraints* und *Parameter* mit Intervallen als *Wissen* in ein Bauwerksmodell zu integrieren und mit der Geometrie in Beziehung zu setzen.

In einem zweiten Schritt konnte eine prototypische Implementierung entwickelt werden, mit der es möglich ist, *Intervall-Constraint-Systeme* für Geometrie zu nutzen und die Parameterbereiche zu berechnen. Auch für die *geometrische Anfrage* mit einer prototypischen Skizze konnte ein Ansatz auf Basis einer numerischen Optimierung entwickelt werden.

Die Implementierung stellt ein einfaches *Expertensystem* dar, mit dem es möglich ist, die verbliebenen Spielräume eines Modells zu ermitteln und den Menschen bei der Suche nach einem guten Entwurf zu unterstützen. Dabei können zweidimensionale, wie auch dreidimensionale *Constraints* genutzt werden, so lange sie sich als algebraische Gleichung ausdrücken lassen. Anhand von Beispielen wurde die Tauglichkeit der Implementierung deutlich gemacht.

9.2 Fazit

Die verschiedenen Ziele dieser Arbeit wurden größtenteils erreicht. Es konnte gezeigt werden, dass es zumindest für kleine Beispiele möglich ist, die Forderungen nach einer Modellierung, die umfangreiche Anforderungen unterstützt, zu ermöglichen.

Der Ansatz der Integration von *Wissenseinheiten* in das Bauwerksmodell, welches aus den Anforderungen an das reale Bauwerk besteht, sollte unabhängig von der Umsetzung dieser Arbeit Beachtung finden. Dies bezieht sich ausdrücklich nicht nur auf das *Wissen* in Bezug auf das *geometrische Modellieren*. Der Autor dieser Arbeit sieht dies als eine sinnvolle Möglichkeit an, die Qualität der Bauwerksmodelle und der Planung allgemein zu verbessern. Dies kann in verschiedenen Stufen umgesetzt werden:

1. Das *Wissen* ist eine Form der Dokumentation, sodass jede *Wissenseinheit* eine Markierung darstellt, die Entwurfsentscheidungen nachvollziehbar und gegebenenfalls einer Person zugeordnet werden kann.

2. Das *Wissen* ist formalisierbar, sodass *Regeln* abgeleitet werden können, die bei der nachträgliche Prüfung des Bauwerksmodells ausgewertet werden können. Dies ist vergleichbar mit dem Prinzip der Test-getriebenen Softwareentwicklung (englisch: test-driven development). Dabei werden die zu absolvierenden Tests als *Regeln* vor der Entwicklung der zu testenden Softwarekomponenten implementiert und mit ihnen verwaltet.

3. Das *Wissen* wird, wie in dieser Arbeit beschrieben, dazu genutzt, im Modellierungsprozess selbst Schlussfolgerungen zu ziehen und den Menschen bei der Entwurfsentscheidung zu unterstützen.

Betrachtet man die Algorithmen auf Basis der Intervalle, so gestaltet sich das Fazit zwiespältig. Systeme mit umfangreichen modellierten Zusammenhängen, wie ein Bauwerksmodell, sind komplex und stellen bei der Findung einer sinnvollen Lösung eine Herausforderung dar. Aus diesem Grund ist ausschließlich eine Berechnung der *globalen Hülle* aller *Parameter* mit vertretbarem Aufwand möglich. Für jeden *Parameter* die obere und untere Grenze zu berechnen, für die Lösungen existieren, ist sinnvoll. Eine Aussage über das Innere des Intervalls ist nicht möglich, sodass darin im Extremfall keine weiteren Lösungen enthalten sind. Die Aussagekraft ist somit zwar stark, aber trotzdem beschränkt. Theoretisch wäre auch die Berechnung der inneren Grenzen möglich, dies stellt aber ein kombinatorisches Problem dar, sodass sich das Laufzeitverhalten als nicht praktikabel und damit akzeptabel ergäbe. Die Visualisierung des Lösungsraums ist ebenfalls nur begrenzt möglich, da sich durch die komplexen Formen, die möglich sind, der genaue Umriss nur unter hohem Aufwand bestimmen lässt. An dieser Stelle konnte nur der vereinfachte Ansatz der *konvexen Hülle* genutzt werden, der lediglich einen groben Umriss darstellt.

Betrachtet man die Optimierungsalgorithmen und die Suche nach möglichst „nahen“ Lösungen, so ist der vorgestellte Ansatz sinnvoll, sollte aber in Kombination mit weiteren Möglichkeiten der Analyse genutzt werden. Dies stellt somit nur den ersten Schritt dar. Es kann aktuell nicht abgeschätzt werden, wie praktikabel das Modellieren und Arbeiten mit einem solchen Bauwerksmodell wirklich ist. Das hängt in erster Linie davon ab inwieweit eine Modellierungs-Software den Menschen bei der Analyse des Systems und der berechneten Lösungen unterstützt.

Wie in dieser Arbeit beschrieben, sind die Probleme bei der Berechnung von Lösungen für *Constraint-Systeme* in vielen Fällen NP-schwer. Dies ist in der Anwendung der nur teilweise optimierten prototypischen Implementierung bereits bei einfachen Beispielen festzustellen, in denen die Laufzeit stark ansteigt, und stellt kein unbekanntes Problem beim Lösen von *Intervall-Constraint-Systemen* dar (vgl. Neumaier, 2004, Seite 70). Die Laufzeit ist aber stark vom gewählten Beispiel abhängig und kann aber durch verschiedene Strategien bei der Lösungsfindung deutlich verbessert werden.

9.3 Ausblick

Die Ansätze dieser Arbeit sollen Möglichkeiten aufzeigen, wie Bauwerksmodelle in Zukunft aufgebaut sein sollten und ausgewertet werden können. Es existiert weiterhin eine Vielzahl von Problemen, die gelöst werden müssen, bevor der Ansatz mit Intervallen und *Constraints* in einem Produktivsystem zum Einsatz kommen kann. Aufgrund der Komplexität der Systeme und der Menge an zusätzlichen Daten, die durch den Menschen im Modellierungsprozess spezifiziert werden müssen, ist dies nur mit einer massiven Unterstützung durch die Software möglich.

Für die Integration von *Wissen* ist zu untersuchen, inwieweit es praktikabel ist, die Anforderungen in Form von Parameterbereichen und *Constraints* zu spezifizieren. Dies kann möglicherweise durch die Wiederverwendung von *Funktionalen Einheiten* und Anforderungen vereinfacht werden. Inwieweit der Mensch bei der Analyse unterstützt werden kann, ist ebenfalls zu prüfen. Ein Vorschlag in diese Richtung ist der Gewinn von zusätzlichen Informationen und die Darstellung der Zusammenhänge aus dem *Constraint-Graphen*. Dies erfordert gegebenenfalls eine Arbeitsweise, bei der auf Basis des *Constraint-Graphen* das Modell analysiert und bearbeitet wird.

Es ist aktuell nicht sichergestellt, dass sich alle Anforderungen an eine valide Geometrie als *Constraints* in Kombination mit Intervallen umsetzen lassen können. Dazu sind weitere Untersuchungen notwendig, mit denen überprüft wird, ob dies machbar und mit der Laufzeit vereinbar ist.

Sollte sich herausstellen, dass der massive Einsatz von *Constraints* bei der Modellierung größerer Bauwerke nicht umsetzbar ist, da die Systeme zu komplex oder die Laufzeiten in der Praxis nicht sinnvoll wären, so könnte eine Kombination mit bestehenden Verfahren geprüft werden. Die Vereinfachung des Systems durch ein *prozedurales Modell* in Kombination mit abschnittsweise *Constraint-basierten* Modellen und Intervallen würde bereits einen Fortschritt ermöglichen.

Im Bereich der *generativen Gestaltung* und bei Parameterstudien ist zu untersuchen, ob der Einsatz von Intervallen sinnvoll ist. Durch die dabei verwendeten zyklenfreien gerichteten Graphen wäre auch bei der Berechnung mit Intervallen ein Lösungsverfahren notwendig, welches das *Intervall-Abhängigkeitsproblem* berücksichtigt. Zur Optimierung und Berechnung von möglichen Parameterwerten könnten die Algorithmen aus dieser Arbeit genutzt werden.

In Bezug auf die Algorithmen sind verschiedene Untersuchungen denkbar. So könnte im Sinne des *Mengen-basierten Entwurfs* die *globale Hülle* mit einer Optimierung bestimmter Werte gekoppelt werden. Damit wäre es möglich, für Optima die verbliebenen

Möglichkeiten zu berechnen. Zwar ist die Parallelisierbarkeit der Algorithmen durch die Natur des *Branch-and-Prune-Verfahrens* gegeben (vgl. Dawood, 2011, Seite 71), doch sind Gewinne aus der Wahl geschickter Strategien für die verschiedenen Phasen der Algorithmen möglich. Beispielsweise könnte im *Branch-and-Prune-Verfahren* der *Gleichungsgraph* weiter zerlegt werden, wenn zwischen den Teilgraphen nur noch ein Zusammenhang auf Basis eines degenerierten Intervalls besteht. In dieser Richtung besteht weiterer Untersuchungsbedarf. Eine offene Frage ist ebenfalls die der numerischen Stabilität und Robustheit der Algorithmen, welche in dieser Arbeit nicht untersucht wurden.

Zum Abschluss dieser Arbeit ist herauszustellen, dass die Arbeitsweise des vorgestellten Entwicklungsprozesses bei der Entwurfsfindung, bzw. die *wissensbasierte Modellierung*, deutlich von der aktuell praktizierten Arbeitsweise abweicht. Der Mensch würde dabei den Rechner zum Finden einer Lösung nutzen und konstruiert nicht mehr selbst. Es ist erforderlich, dass alle integrierten Anforderungen für das System und alle Anfragen an das System in geeigneter Art und Weise formalisiert werden müssen, bevor sie algorithmisch nutzbar sind. Möglicherweise ist dies der Kerngedanke der nächsten Generation von Softwareprodukten: Der Mensch integriert das *Wissen* über das zu lösende Problem in das Bauwerksmodell und trifft die Entscheidung des konkreten Entwurfs im Rahmen der verbliebenen Möglichkeiten und der Optimierung der Zielgrößen.

Literaturverzeichnis

Abulawi, Jutta (2012). „Ansatz zur Beherrschung der Komplexität von vernetzten 3D-CAD-Modellen“. Dissertation. Hamburg, Germany: Helmut-Schmidt-Universität.

Ait-Aoudia, Samy, Mehdi Bahriz und Lyes Salhi (2009). „2D Geometric Constraint Solving: An Overview“. In: *Visualisation, 2009. VIZ '09. Second International Conference in*, S. 201–206.

Araya, Ignacio und Victor Reyes (2016). „Interval Branch-and-Bound algorithms for optimization and constraint satisfaction: a survey and prospects“. In: *Journal of Global Optimization* 65.4, S. 837–866.

Ault, Holly K. und Arnold Phillips (2016). „Direct Modeling: Easy Changes in CAD?“ In: *Proceedings of the ASEE EDGD 70th Midyear Conference*. Daytona Beach Shores, Florida, S. 99–106.

Beelitz, Thomas (2006). „Effiziente Methoden zum Verifizierten Lösen von Optimierungsaufgaben und Nichtlinearen Gleichungssystemen“. Dissertation. Wuppertal: Bergischen Universität Wuppertal.

Benhamou, Frédéric, Frédéric Goualard, Laurent Granvilliers und Jean-Francois Puget (1999). „Revising Hull and Box Consistency“. In: *Logic Programming: The 1999 International Conference, Las Cruces, New Mexico, USA, November 29 - December 4, 1999*. MIT press, S. 230–244.

Benhamou, Frédéric und Laurent Granvilliers (1996). „Combining local consistency, symbolic rewriting and interval methods“. In: *Artificial Intelligence and Symbolic Mathematical Computation*. Hrsg. von G. Goos, J. Hartmanis, J. Leeuwen, Jacques Calmet, John A. Campbell und Jochen Pfalzgraf. Bd. 1138. Springer Berlin Heidelberg, S. 144–159. ISBN: 978-3-540-61732-7 978-3-540-70740-0.

Benhamou, Frédéric, David Allen McAllester und Pascal Van Hentenryck (1994). *CLP (Intervals) Revisited*. Techn. Ber. Providence, RI, USA: Brown University.

Berg, Mark van den, Antony Tang und Rik Farenhorst (2009). „A Constraint-Oriented Approach to Software Architecture Design“. In: *Proceedings of the Ninth International Conference on Quality Software*. Jeju, Korea, S. 396–405. ISBN: 978-0-7695-3828-0.

Bettig, Bernhard, Vikram Bapat und Balaji Bharadwaj (2005). „Limitations of parametric operators for supporting systematic design“. In: *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, S. 131–142.

Bettig, Bernhard und Christoph M. Hoffmann (2011). „Geometric Constraint Solving in Parametric Computer-Aided Design“. In: *Journal of Computing and Information Science in Engineering* 11.2, S. 021001–021001.

Bettig, Bernhard und Vaibhav Kale (2012). „Geometric Constraint Solving With Solution Selectors“. In: *Journal of Computing and Information Science in Engineering* 12.4.

Bettig, Bernhard und Jami J. Shah (2001). „Derivation of a standard set of geometric constraints for parametric modeling and data exchange“. In: *Computer-Aided Design* 33.1, S. 17–33.

— (2003). „Solution Selectors: A User-Oriented Answer to the Multiple Solution Problem in Constraint Solving“. In: *Journal of Mechanical Design* 125.3, S. 443.

Böger, Laura (2017). „Operationen auf dem Constraint-Graphen eines parametrischen Gebäudemodells“. Masterarbeit. Berlin: Technische Universität Berlin.

Booch, Grady, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Connallen und Kelli A. Houston (2008). *Object-oriented analysis and design with applications, third edition*. Bd. 33. Redwood City, CA, US: Addison Wesley Longman Publishing Co., Inc.

Bordeaux, Lucas, Youssef Hamadi und Moshe Y. Vardi (2007). „An Analysis of Slow Convergence in Interval Propagation“. In: *Principles and Practice of Constraint Programming – CP 2007*. Hrsg. von Christian Bessière. Bd. 4741. Springer Berlin Heidelberg, S. 790–797. ISBN: 978-3-540-74969-1.

Bordeaux, Lucas, Eric Monfroy und Frédéric Benhamou (2001). „Improved bounds on the complexity of kB-consistency“. In: *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, S. 303–308.

Borrmann, André, Yang Ji, Javier Ramos Jubierre und Matthias Flurl (2012). „Procedural Modeling: A new approach to multi-scale design in infrastructure projects“. In: *Proceedings of the 19th EG-ICE International Workshop*. München: Technische Universität München. ISBN: 978-3-00-038455-4.

Borrmann, André, Markus König, Christian Koch und Jakob Beetz, Hrsg. (2015). *Building Information Modeling*. Springer Fachmedien Wiesbaden. ISBN: 978-3-658-05605-6 978-3-658-05606-3.

Bouma, William, Ioannis Fudos, Christoph M. Hoffmann, Jiazhen Cai und Robert Paige (1995). „A Geometric Constraint Solver“. In: *Computer-Aided Design* 27.6, S. 487–501.

Brüderlin, Beat und Dieter Roller, Hrsg. (1998). *Geometric Constraint Solving and Applications*. Springer Berlin Heidelberg. ISBN: 978-3-642-63781-0 978-3-642-58898-3.

Bundesanstalt für Arbeitsschutz und Arbeitsmedizin: Ausschuss für Arbeitsstätten (ASTA) (2007). *Technische Regeln für Arbeitsstätten (ASR) - Fluchtwege und Notausgänge, Flucht- und Rettungsplan.* Technische Regeln ASR A2.3. Bundesanstalt für Arbeitsschutz und Arbeitsmedizin: Ausschuss für Arbeitsstätten (ASTA).

Ceberio, Martine und Laurent Granvilliers (2001). „Solving Nonlinear Systems by Constraint Inversion and Interval Arithmetic". In: *Artificial Intelligence and Symbolic Computation.* Hrsg. von G. Goos, J. Hartmanis, J. van Leeuwen, John A. Campbell und Eugenio Roanes-Lozano. Bd. 1930. Springer Berlin Heidelberg, S. 127–141. ISBN: 978-3-540-42071-2 978-3-540-44990-4.

Chang, Kuang-Hua (2014). *Product Design Modeling using CAD/CAE: The Computer Aided Engineering Design Series.* Academic Press. ISBN: 978-0-12-398517-0.

Collavizza, Hélène, François Delobel und Michel Rueher (1999). „Comparing Partial Consistencies". In: *Reliable Computing* 5.3, S. 213–228.

Cruz, Jorge Carlos Ferreira Rodrigues da (2003). „Constraint reasoning for differential models". Disseration. Lisboa: Universidade Nova de Lisboa.

Csallner, A. E. und T. Csendes (1996). „The convergence speed of interval methods for global optimization". In: *Computers & Mathematics with Applications.* Selected Topics in Numerical Methods 31.4, S. 173–178.

Cunis, Roman, Andreas Günter und Helmut Strecker, Hrsg. (1991). *Das PLAKON-Buch: Ein Expertensystemkern für Planungs- und Konfigurierungsaufgaben in technischen Domänen.* Subreihe Künstliche Intelligenz. Springer Berlin Heidelberg. ISBN: 978-3-540-53683-3.

Dattorro, Jon (2016). *Convex optimization euclidean distance geometry.* 2. Aufl. Palo Alto, California: Meboo Publishing. ISBN: 978-0-578-16140-2.

Daumas, Marc, David Lester und César Muñoz (2009). „Verified Real Number Calculations: A Library for Interval Arithmetic". In: *IEEE Trans. Comput.* 58.2, S. 226–237.

Dawood, Hend (2011). *Theories of Interval Arithmetic: Mathematical Foundations and Applications.* LAP Lambert Academic Publishing. ISBN: 978-3-8465-0154-2.

DIN EN ISO (2018). *DIN EN ISO 29481-1:2017 - BIM - Informations-Lieferungs-Handbuch - Teil 1: Methodik und Format.*

Donath, Dirk und Luis Felipe González Böhme (2007). „Constraint-Based Design in Participatory Housing Planning". In: *International Journal of Architectural Computing.* International Journal of Architectural Computing 6.1, S. 97–117.

Döring, Ulf (2011). „Ficucs, Ein Constraint-Solver für Geometrische Constraints in 2D und 3D". Disseration. enau: Technische Universität Ilmenau.

Eastman, Charles M., Jae-min Lee, Yeon-suk Jeong und Jin-kook Lee (2009). „Automatic rule-based checking of building designs". In: *Automation in Construction* 18.8, S. 1011–1033.

Eastman, Charles M., Rafael Sacks und Ghang Lee (2004). „Functional modeling in parametric CAD systems“. In: *Generative CAD (G-CAD) conference*. Hrsg. von O. Akin. Carnegie Mellon University, Pittsburgh, PA.

Eastman, Charles M., Paul Teicholz, Rafael Sacks und Kathleen Liston (2011). *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*. John Wiley & Sons. ISBN: 978-1-118-02169-9.

Ebbinghaus, Heinz-Dieter, Jörg Flum und Wolfgang Thomas (2018). *Einführung in die mathematische Logik*. 6. Aufl. Springer Spektrum. ISBN: 978-3-662-58028-8.

Essert-Villard, C., P. Schreck und J. -F. Dufourd (2000). „Sketch-based pruning of a solution space within a formal geometric constraint solver“. In: *Artificial Intelligence* 124.1, S. 139–159.

Finch, William W. und Allen C. Ward (1996). „Quantified relations: a class of predicate logic design constraints among sets of manufacturing, operating and other variables“. In: *Proceedings of the 1996 ASME Design Engineering Technical Conference*. Irvine, USA.

Freuder, Eugene C. (1982). „A Sufficient Condition for Backtrack-Free Search“. In: *Journal of the ACM* 29.1, S. 24–32.

Freuder, Eugene C. und Alan K. Mackworth (2006). „Constraint Satisfaction: An Emerging Paradigm“. In: *Handbook of Constraint Programming*. Hrsg. von Francesca Rossi, Peter Van Beek und Toby Walsh. Elsevier Science & Technology. ISBN: 978-0-08-046380-3.

Fudos, Ioannis und Christoph M. Hoffmann (1997). „A Graph-constructive Approach to Solving Systems of Geometric Constraints“. In: *ACM Transactions on Graphics* 16.2, S. 179–216.

Galle, Per (1995). „Towards integrated,“intelligent”, and compliant computer modeling of buildings“. In: *Automation in construction* 4.3, S. 189–211.

Ghosh, Sourobh und Warren Seering (2014). „Set-Based Thinking in the Engineering Design Community and Beyond“. In: *Volume 7: 2nd Biennial International Conference on Dynamics for Design; 26th International Conference on Design Theory and Methodology*. Buffalo, New York, USA: ASME. ISBN: 978-0-7918-4640-7.

Gielingh, Wim (2008). „A theory for the modelling of complex and dynamic systems“. In: *The Journal of Information Technology in Construction (ITcon)* 13, S. 421–475.

— (1988). „General AEC Reference Model (GARM). An aid for the integration of application specific product definition models“. In: *Proceedings of CIB W74 + W78 Workshop*. Bd. 126.

Gil, N., S. Beckman und I. D. Tommelein (2008). „Upstream Problem Solving Under Uncertainty and Ambiguity: Evidence From Airport Expansion Projects“. In: *IEEE Transactions on Engineering Management* 55.3, S. 508–522.

Goldsztejn, Alexandre (2007). „Comparison of the Hansen-Sengupta and the Frommer-Lang-Schnurr existence tests“. In: *Computing* 79.1, S. 53–60.

Granvilliers, Laurent (2001). „On the Combination of Interval Constraint Solvers“. In: *Reliable Computing* 7.6, S. 467–483.

Granvilliers, Laurent, Jorge Cruz und Pedro Barahona (2004). „Parameter Estimation Using Interval Computations“. In: *SIAM Journal on Scientific Computing* 26.2, S. 591–612.

Günter, Andreas und Christian Kühn (1999). „Knowledge-Based Configuration- Survey and Future Directions“. In: *XPS-99: Knowledge-Based Systems. Survey and Future Directions*. Hrsg. von Frank Puppe. Lecture Notes in Computer Science. Springer Berlin Heidelberg, S. 47–66. ISBN: 978-3-540-49149-1.

Hansen, Eldon R. und Ronald I. Greenberg (1983). „An Interval Newton Method“. In: *Applied Mathematics and Computation* 12.2, S. 89–98.

Hansen, Eldon R. und G. William Walster (2003). *Global Optimization Using Interval Analysis*. 2nd ed., revised and expanded. Monographs and textbooks in pure and applied mathematics 264. New York: Marcel Dekker. ISBN: 978-0-8247-4059-7.

Haun, Matthias (2016). *Cognitive Organisation*. Springer Berlin Heidelberg. ISBN: 978-3-662-52951-5 978-3-662-52952-2.

Herbort, Stefan und Dietmar Ratz (1997). *Improving the efficiency of a nonlinear-system-solver using a componentwise Newton method*. Techn. Ber. Karlsruhe: Institut für Angewandte Mathematik, Universität Karlsruhe (TH).

Hidalgo Garcia, Marta R. (2013). „Geometric constraint solving in a dynamic geometry framework“. PhD Dissertation. Barcelona: Universitat Politècnica de Catalunya.

Hillyard, Robin (1982). „The Build Group of Solid Modelers“. In: *IEEE Computer Graphics and Applications* 2.2, S. 43–52.

Hoffmann, Christoph M. (2004). „Solid modeling“. In: *Handbook of Discrete and Computational Geometry, Second Edition*. Hrsg. von Jacob E. Goodman und Joseph O'Rourke. Chapman und Hall/CRC, S. 1257–1278. ISBN: 978-1-58488-301-2.

Hoffmann, Christoph M. und Robert Joan-Arinyo (2005). „A Brief on Constraint Solving“. In: *Computer-Aided Design and Applications* 2.5, S. 655–663.

Hoffmann, Christoph M. und Ku-Jin Kim (2001). „Towards valid parametric CAD models“. In: *Computer-Aided Design* 33.1, S. 81–90.

Hoffmann, Christoph M. und Jaroslaw R. Rossignac (1996). „A road map to solid modeling“. In: *IEEE Transactions on Visualization and Computer Graphics* 2.1, S. 3–10.

Huhnt, Wolfgang (2018). „Reconstruction of edges in digital building models“. In: *Advanced Engineering Informatics* 38, S. 474–487.

Hyvönen, Eero (1989). „Constraint Reasoning Based on Interval Arithmetic.“ In: *IJCAI*, S. 1193–1198.

IEEE (2015). *IEEE 1788-2008 - Standard for Interval Arithmetic*.

— (2008). *IEEE 754-2008 - Standard for Floating-Point Arithmetic*.

Ishii, Daisuke, Alexandre Goldsztejn und Christophe Jermann (2012). „Interval-based projection method for under-constrained numerical systems“. In: *Constraints* 17.4, S. 432–460.

ISO (1994). *ISO 10303-1:1994 - Industrial automation systems - Product data representation and exchange - Part 1: Overview and fundamental principles*.

— (2005a). *ISO 10303-108:2005 - Industrial automation systems and integration - Product data representation and exchange - Part 108: Integrated application resource: Parameterization and constraints for explicit geometric product models*.

— (2005b). *ISO 10303-55:2005 - Industrial automation systems - Product data representation and exchange - Part 55: Integrated generic resource: Procedural and hybrid representation*.

— (2015). *ISO 12006-2:2015: Building construction - Organization of information about construction works - Part 2: Framework for classification*.

Jermann, Christophe, Gilles Trombettoni, Bertrand Neveu und Pascal Mathis (2006). „Decomposition of geometric constraint systems: a survey“. In: *International Journal of Computational Geometry & Applications* 16.5-6, S. 379–414.

Joan-Arinyo, Robert (2009). „Basics on Geometric Constraint Solving“. In: *Proceedings of 13th Encuentros de Geometrfa Computacional (EGC09)*. Zaragoza (Spain).

Joan-Arinyo, Robert, Maria Victoria Luzón und Antoni Soto (2003). „Genetic algorithms for root multiselection in constructive geometric constraint solving“. In: *Computers & Graphics* 27.1, S. 51–60.

Joan-Arinyo, Robert, Nuria Mata und Antoni Soto-Riera (2001). „A Constraint Solving-Based Approach to Analyze 2D Geometric Problems with Interval Parameters“. In: *Journal of Computing and Information Science in Engineering* 1.4, S. 341–346.

Kearfott, R. Baker (2013). *Rigorous Global Search: Continuous Problems*. Springer Science + Business Media. ISBN: 978-1-4757-2495-0.

Kelleners, Richard Hendrikus Maria Christina (1999). „Constraints in object-oriented graphics“. PhD Thesis. Eindhoven: Eindhoven University of Technology.

Kirchner, Jakob und Wolfgang Huhnt (2018). „Benefits and Limits of Interval Constraints in Acyclic Constraint Graphs for Geometric Modeling“. In: *Proceeding of the 17th International Conference on Computing in Civil and Building Engineering*. Tampere, Finland, S. 716–723.

Kirchner, Jakob, Wolfgang Huhnt, Thomas Gust und Konrad Polthier (2016). „Geometric correct objects and systems with a predictable and traceable behavior“. In: *Proceedings of the 23rd International Workshop on Intelligent Computing in Engineering*. Krakow, Poland.

Kiviniemi, Arto (2005). „Requirements management interface to building product models“. Dissertation. Stanford, CA, USA: Stanford University.

Kjøller, Steffen, Pavel Kozine, Kaj Madsen und Ole Stauning (2007). „Non-linear global optimization using interval arithmetic and constraint propagation“. In: *Models and Algorithms for Global Optimization*, S. 45–58.

Kraft, Bernold (2016). „Ein Verfahren der Raumzerlegung als Grundlage zur Prüfung von Geometrie und Topologie digitaler Bauwerksmodelle“. Dissertation. Berlin: Technische Universität Berlin.

Krawczyk, Rudolf (1969). „Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken“. In: *Computing* 4.3, S. 187–201.

Kubica, Bartłomiej Jacek (2019). *Interval Methods for Solving Nonlinear Constraint Satisfaction, Optimization and Similar Problems: From Inequalities Systems to Game Solutions*. Studies in Computational Intelligence. Springer International Publishing. ISBN: 978-3-030-13794-6.

— (2011). „Interval Methods for Solving Underdetermined Nonlinear Equations Systems“. In: *Reliable Computing*.

— (2017). „Parallelization of a bound-consistency enforcing procedure and its application in solving nonlinear systems“. In: *Journal of Parallel and Distributed Computing* 107, S. 57–66.

— (2015). „Presentation of a highly tuned multithreaded interval solver for underdetermined and well-determined nonlinear systems: Empirical evaluation of innovations“. In: *Numerical Algorithms* 70.4, S. 929–963.

— (2009). „Shared-Memory Parallelization of an Interval Equations Systems Solver – Comparison of Tools“. In: *KAEiOG 2009 Proceedings*.

Kulisch, Ulrich (2013). *Computer Arithmetic and Validity, Theory, Implementation, and Applications*. Berlin, Boston: De Gruyter. ISBN: 978-3-11-030173-1.

Kulisch, Ulrich, Rolf Hammer, Dietmar Ratz und Matthias Hocks (1993). *Numerical Toolbox for Verified Computing I: Basic Numerical Problems Theory, Algorithms, and Pascal-XSC Programs*. Springer Berlin Heidelberg. ISBN: 978-3-642-78423-1.

Kulisch, Ulrich W. (2009). „Complete Interval Arithmetic and Its Implementation on the Computer“. In: *Numerical Validation in Current Hardware Architectures*. Hrsg. von Annie Cuyt, Walter Krämer, Wolfram Luther und Peter Markstein. Bd. 5492. Springer Berlin Heidelberg, S. 7–26. ISBN: 978-3-642-01590-8 978-3-642-01591-5.

Lee, Ghang, Rafael Sacks und Charles M. Eastman (2006). „Specifying parametric building object behavior (BOB) for a building information modeling system“. In: *Automation in Construction*. Knowledge Enabled Information System Applications in Construction 15.6, S. 758–776.

Lhomme, Olivier (1993). „Consistency Techniques for Numeric CSPs“. In: *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1*. IJCAI'93. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., S. 232–238.

Liker, Jeffrey K., Durward K. Sobek, Allen C. Ward und John J. Cristiano (1996). „Involving suppliers in product development in the United States and Japan: evidence for set-based concurrent engineering“. In: *IEEE Transactions on Engineering Management* 43.2, S. 165–178.

Lin, Dang und Liangyu Chen (2018). „An efficient algorithm for global interval solution of nonlinear algebraic equations and its GPGPU implementation“. In: *CoRR*.

Lipson, Hod, Fumihiko Kimura und Moshe Shpitalni (1999). *Solving Geometric Constraints by Auxiliary Constructions*.

Luzón, Maria Victoria, Antoni Soto, Juan F. Gálvez und Robert Joan-Arinyo (2005). „Searching the Solution Space in Constructive Geometric Constraint Solving with Genetic Algorithms“. In: *Applied Intelligence* 22.2, S. 109–124.

Mackworth, Alan K. (1981). „Consistency in Networks of Relations“. In: *Readings in Artificial Intelligence*. Hrsg. von Bonnie Lynn Webber und Nils J. Nilsson. Morgan Kaufmann, S. 69–78. ISBN: 978-0-934613-03-3.

Mäntylä, Martti (1988). *Introduction to Solid Modeling*. New York, NY, USA: W. H. Freeman & Co. ISBN: 978-0-88175-108-6.

Mata Burgarolas, Nuria und Vladik Kreinovich (1999). „NP-Hardness In Geometric Construction Problems With One Interval Parameter“. In: *Applications of Interval Analysis to Systems and Control with special emphasis on recent advances in Modal Interval Analysis (MISC'99)*. Girona, Spain, S. 85–98.

Meiden, Hilderick A. van der (2008). „Semantics of Families of Objects“. PhD Thesis. Delft, Netherlands: Technische Universiteit Delft.

Meiden, Hilderick A. van der und Willem F. Bronsvoort (2006). „A constructive approach to calculate parameter ranges for systems of geometric constraints“. In: *Computer-Aided Design* 38.4, S. 275–283.

— (2010). „A non-rigid cluster rewriting approach to solve systems of 3D geometric constraints“. In: *Computer-Aided Design* 42.1, S. 36–49.

— (2005). „An Efficient Method to Determine the Intended Solution for a System of Geometric Constraints“. In: *International Journal of Computational Geometry & Applications* 15.03, S. 279–298.

Messine, Frederic (2004). „Deterministic global optimization using interval constraint propagation techniques“. In: *RAIRO - Operations Research* 38.4, S. 277–293.

Miguel, Ian und Qiang Shen (2001). „Solution Techniques for Constraint Satisfaction Problems: Foundations“. In: *Artificial Intelligence Review* 15.4, S. 243–267.

Moa, Belaid (2007). „Interval Methods for Global Optimization“. PhD Thesis. Victoria, BC, Canada: University of Victoria.

Moore, Ramon E. (1966). *Interval analysis*. Englewood Cliffs, NJ: Prentice-Hall.

Moore, Ramon E., R. Baker Kearfott und Michael J. Cloud (2009). *Introduction to Interval Analysis*. Philadelphia, PA, USA: Society for Industrial und Applied Mathematics. ISBN: 978-0-89871-669-6.

Nahm, Yoon-Eui und Haruo Ishikawa (2006). „A new 3D-CAD system for set-based parametric design“. In: *The International Journal of Advanced Manufacturing Technology* 29.1-2, S. 137–150.

Neumaier, Arnold (2004). „Complete Search in Continuous Global Optimization and Constraint Satisfaction“. In: *Acta Numerica* 13. Hrsg. von A. Iserles, S. 271–369.

— (1990). *Interval Methods for Systems of Equations*. Encyclopedia of Mathematics and its Applications. Cambridge University Press.

Neumann, Bernd, Roman Cunis, Andreas Günter und Ingo Syska (1987). „Wissensbasierte Planung und Konfigurierung“. In: *Wissensbasierte Systeme*. Hrsg. von W. Brauer und W. Wahlster. Informatik - Fachberichte. Springer Berlin Heidelberg, S. 347–357. ISBN: 978-3-642-88719-2.

Niemeijer, Remco Arnout (2011). „Constraint specification in architecture : a user-oriented approach for mass customization“. PhD Thesis. Eindhoven: Eindhoven University of Technology.

Niemeijer, Remco Arnout, Bauke de Vries und Jakob Beetz (2014). „Freedom through constraints: User-oriented architectural design“. In: *Advanced Engineering Informatics* 28.1, S. 28–36.

Noort, Alex, Maurice Dohmen und Willem F. Bronsvoort (1998). „Solving Over- and Underconstrained Geometric Models“. In: *Geometric Constraint Solving and Applications*. Hrsg. von Beat Brüderlin und Dieter Roller. Springer Berlin Heidelberg, S. 107–126. ISBN: 978-3-642-63781-0 978-3-642-58898-3.

Obergrießer, Mathias (2017). *Digitale Werkzeuge zur integrierten Infrastrukturbauwerksplanung*. Wiesbaden: Springer Vieweg. ISBN: 978-3-658-16781-3 978-3-658-16782-0.

Object Management Group (2014). *Object Constraint Language, Version 2.4*. URL: http://www.omg.org/spec/OCL/2.4.

Otey, Jeffrey, Pedro Company, Manuel Contero und Jorge D. Camba (2018). „Revisiting the design intent concept in the context of mechanical CAD education“. In: *Computer-Aided Design and Applications* 15.1, S. 47–60.

Owen, J. C. (1991). „Algebraic Solution for Geometry from Dimensional Constraints“. In: *Proceedings of the First ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications*. SMA '91. New York, NY, USA: ACM, S. 397–407. ISBN: 978-0-89791-427-7.

Panchal, Jitesh H., Marco Gero Fernández, Christiaan J. J. Paredis, Janet K. Allen und Farrokh Mistree (2007). „An Interval-based Constraint Satisfaction (IBCS) Method for Decentralized, Collaborative Multifunctional Design“. In: *Concurrent Engineering* 15.3, S. 309–323.

Papageorgiou, Markos, Marion Leibold und Martin Buss (2012). *Optimierung: statische, dynamische, stochastische Verfahren für die Anwendung*. 3. Berlin Heidelberg: Springer Vieweg. ISBN: 978-3-540-34012-6 978-3-540-34013-3.

Peña-Mora, Feniosky, Duvvuru Sriram und Robert Logcher (1993). „SHARED-DRIMS: SHARED design recommendation-intent management system“. In: *Proceedings Second Workshop on Enabling Technologies - Infrastructure for Collaborative Enterprises*. Morgantown, WV, USA, S. 213–221.

Pratt, Mike J. (1998). „Extension of the Standards ISO 10303 (STEP) for the Exchange of Parametric and Variational CAD Models“. In: *Proceedings of the 10th IFIP WG5.2/5.3 PROLAMAT Conference*. Trento, Italy.

Rasmussen, Mads Holten, Pieter Pauwels, Christian Anker Hviid und Jan Karlshøj (2017). „Proposing a Central AEC Ontology That Allows for Domain Specific Extensions“. In: *Lean and Computing in Construction Congress - Volume 1: Proceedings of the Joint Conference on Computing in Construction*. Heraklion, Crete, Greece: Heriot-Watt University, S. 237–244. ISBN: 978-0-9565951-6-4.

Ratschek, Helmut und R. L. Voller (1991). „What can interval analysis do for global optimization?“ In: *Journal of Global Optimization* 1.2, S. 111–130.

Ratz, Dietmar (1997). *Inclusion isotone extended interval arithmetic. A toolbox update*. Techn. Ber. Karlsruhe: Institut für Angewandte Mathematik, KIT.

Roller, Dieter (2013). *CAD: Effiziente Anpassungs- und Variantenkonstruktion*. Springer Berlin Heidelberg. ISBN: 978-3-642-79449-0.

Rump, Siegfried M. (2018). „Mathematically rigorous global optimization in floating-point arithmetic“. In: *Optimization Methods and Software* 33.4-6, S. 771–798.

Runte, Wolfgang (2006). „YACS: Ein hybrides Framework für Constraint-Solver zur Unterstützung wissensbasierter Konfigurierung“. Diplomarbeit. Bremen: Universität Bremen.

Sainudiin, Raazesh und Ruriko Yoshida (2005). „Applications of Interval Methods to Phylogenetic trees“. In: *Algebraic Statistics for Computational Biology*. New York, NY: Cambridge University Press. Cambridge: Cambridge University Press, S. 359–374.

Schultz, Carl, Mehul Bhatt und André Borrmann (2017). „Bridging qualitative spatial constraints and feature-based parametric modelling: Expressing visibility and movement constraints“. In: *Advanced Engineering Informatics* 31, S. 2–17.

Shah, Jami J. (2001). „Designing with Parametric CAD: Classification and comparison of construction techniques“. In: *Geometric Modelling*. Hrsg. von Fumihiko Kimura. New York: Springer Science+Business Media, S. 53–68. ISBN: 978-1-4757-5322-6 978-0-387-35490-3.

Shah, Jami J. und Martti Mäntylä (1995). *Parametric and Feature-Based CAD/CAM: Concepts, Techniques, and Applications*. John Wiley & Sons. ISBN: 978-0-471-00214-7.

Shah, Jami J., P. Sreevalsan, M. Roger, R. Billo und A. Matthew (1988). *Current Status of features technology*.

Shalloway, Alan, James R. Trott und James Trott (2002). *Design Patterns Explained: A New Perspective on Object-oriented Design*. Addison-Wesley Professional. ISBN: 978-0-201-71594-1.

Shih, Chia-Hui und Bill Anderson (1997). „A design/constraint model to capture design intent“. In: *Proceedings of the fourth ACM symposium on Solid modeling and applications - SMA '97*. Atlanta, Georgia, United States: ACM Press, S. 255–264. ISBN: 978-0-89791-946-3.

Shimizu, Shuichi und Masayuki Numao (1997). „Constraint-based design for 3D shapes“. In: *Artificial Intelligence*. Artificial Intelligence Research in Japan 91.1, S. 51–69.

Shum, Simon Buckingham und Nick Hammond (1994). „Argumentation-based design rationale: what use at what cost?“ In: *International Journal of Human-Computer Studies* 40.4, S. 603–652.

Sitharam, Meera, Adam Arbree, Yong Zhou und Naganandhini Kohareswaran (2006). „Solution space navigation for geometric constraint systems“. In: *ACM Transactions on Graphics* 25, S. 194–213.

Sitharam, Meera, Audrey St. John und Jessica Sidman (2019). *Handbook of geometric constraint systems principles*. Boca Raton: CRC Press, Taylor & Francis Group. ISBN: 978-1-4987-3891-0.

Sobek, Durward, A.C. Ward und Jeffrey Liker (1999). „Toyota's Principles of Set-Based Concurrent Engineering“. In: *Sloan Management Review* 40.2, S. 67–83.

Sommerville, Ian (2011). *Software engineering*. 9th ed. Boston: Pearson. ISBN: 978-0-13-703515-1 978-0-13-705346-9.

Stachowiak, Herbert (1973). *Allgemeine Modelltheorie*. Springer Wien New York. ISBN: 978-3-211-81106-1 978-0-387-81106-2.

Underwood, Jason, Umit Isikdag und Ioan Dima, Hrsg. (2010). *Handbook of Research on Building Information Modeling and Construction Informatics: Concepts and Technologies*. Advances in Civil and Industrial Engineering. IGI Global. ISBN: 978-1-60566-928-1 978-1-60566-929-8.

Vajna, Sandor, Christian Weber, Helmut Bley und Klaus Zeman (2009). *CAx für Ingenieure: eine praxisbezogene Einführung*. 2. Springer Berlin Heidelberg. ISBN: 978-3-540-36038-4 978-3-540-36039-1.

Valdes, Francisco, Russell Gentry, Charles Eastman und Stephen Forrest (2016). „Applying Systems Modeling Approaches to Building Construction“. In: *Proceeding of the 33th International Symposium on Automation and Robotics in Construction*. Auburn, AL, USA. ISBN: 978-1-5108-2992-3.

VDI (2009). *VDI 2209:2009 - 3-D-Produktmodellierung Technische und organisatorische Voraussetzungen Verfahren, Werkzeuge und Anwendungen Wirtschaftlicher Einsatz in der Praxis*.

Vikram, Bapat, Bettig Bernhard und Joshua Summers (2007). „Requirements-driven design computations in next-generation CAD“. In: *Proceedings of ICED 2007, the 16th International Conference on Engineering Design*. Paris, France.

Vilgertshofer, Simon und André Borrmann (2017). „Using graph rewriting methods for the semi-automatic generation of parametric infrastructure models“. In: *Advanced Engineering Informatics* 33, S. 502–515.

Vu, Xuan-Ha, Hermann Schichl und Djamila Sam-Haroud (2009). „Interval propagation and search on directed acyclic graphs for numerical constraint solving“. In: *Journal of Global Optimization* 45.4, S. 499.

Vu, Xuan-Ha, Marius-Calin Silaghi, Djamila Sam-Haroud und Boi Faltings (2006). *Branch-and-prune search strategies for numerical constraint solving*. Techn. Ber. LIA-REPORT-2006-007. Swiss Federak Istitute if Technology (EPFL).

Wang, Yan und Bartholomew O. Nnaji (2007). „Solving Interval Constraints by Linearization in Computer-Aided Design“. In: *Reliable Computing* 13.2, S. 211–244.

Yannou, Bernard, Pierre-Alain Yvars, Chris Hoyle und Wei Chen (2013). „Set-based design by simulation of usage scenario coverage“. In: *Journal of Engineering Design* 24.8, S. 575–603.

Abbildungsverzeichnis

Tabellenverzeichnis

Übersicht der Algorithmen

ANHANG A

Modell

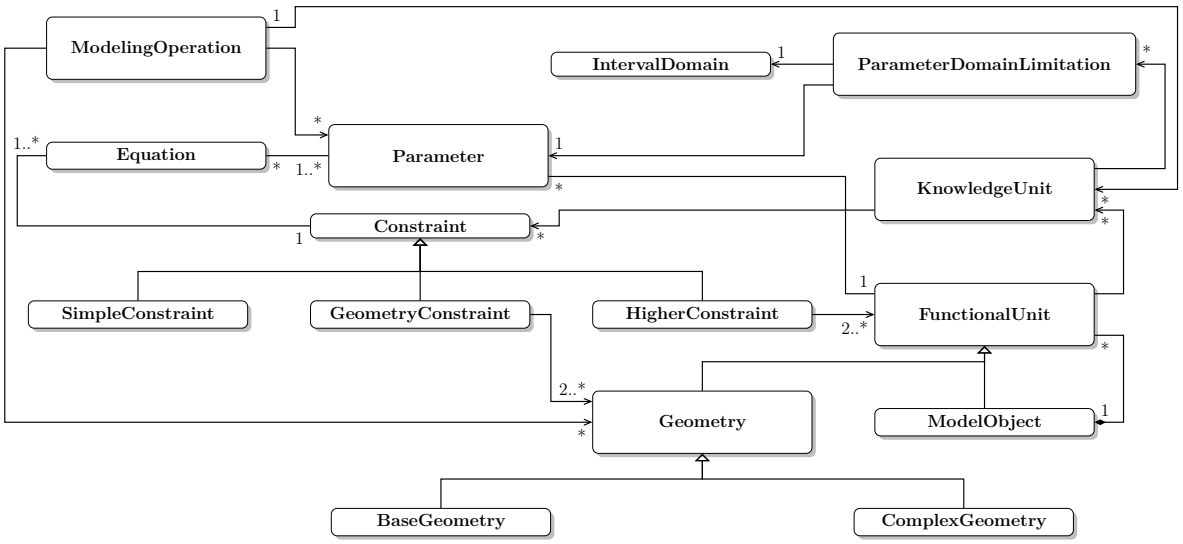


Abbildung A.1: Vollständiges Klassendiagramm des entwickelten Bauwerksmodells aus Kapitel 6

ANHANG B

Werte der Auswertung der Beispiele

B.1 Beispiel: Rechteck-Wand

Tabelle B.1: Beispiel *Rechteck-Wand*: Ergebnisse der *globalen Hülle* für ausgewählte *Parameter* (auf die 3. Dezimalstelle gerundet). Unterschiede der Algorithmen sind grau eingefärbt.

Parameter	**Startwert**	**HC4**	**Berechnete globale Hülle**
P1.X	$[3]$	$[3]$	$[3]$
P1.Y	$[0,1]$	$[0,1]$	$[0,1]$
P2.X	$[3,4]$	$[3,4]$	$[3,4]$
P2.Y	$[5]$	$[5]$	$[5]$
P3.X	$[-\infty,\infty]$	$[2.25,3.765]$	$[3.39,3.710]$
P3.Y	$[-\infty,\infty]$	$[5,5.15]$	$[5,5.15]$
P4.X	$[-\infty,\infty]$	$[2.25,2.765]$	$[2.39,2.709[$
P4.Y	$[-\infty,\infty]$	$[0,1.15]$	$[0,1.15]$
Dicke	$[0.3,0.6]$	$[0.3,0.6]$	$[0.3,0.6]$
Höhe	$[2.2]$	$[2.2]$	$[2.2]$
Länge	$]0,\infty[$	$[4,5.099]$	$[4,5.099]$

Tabelle B.2: Beispiel *Rechteck-Wand*: Ergebnisse der Optimierungen für ausgewählte *Parameter* (auf die 3. Dezimalstelle gerundet)

Parameter	Max. Dicke	Dicke 0.5 & Länge 6	Skizze (a)	Skizze (b)
P1.X	3	3	3	3
P1.Y	0.2	0.01	0	1
P2.X	3.54	3.99	3	4
P2.Y	5	5	5	5
P3.X	2.943	3.51	2.400	3.709
P3.Y	5.067	5.100	5	5.073
P4.X	2.404	2.51	2.400	2.709
P4.Y	0.2667	0.11	0	1.075
Dicke	0.6	0.5	0.600	0.309
Höhe	2.2	2.2	2.2	2.2
Länge	4.830	5.099	5.000	4.123

B.2 Beispiel: Trapez-Wand

Tabelle B.3: Beispiel *Trapez-Wand*: Ergebnisse der *globalen Hülle* für ausgewählte *Parameter* (auf die 3. Dezimalstelle gerundet). Unterschiede der Algorithmen sind grau eingefärbt.

Parameter	Startwerte	HC4	Berechnete globale Hülle
P1.X	$[3]$	$[3]$	$[3]$
P1.Y	$[0,1]$	$[0,1]$	$[0,1]$
P2.X	$[3,4]$	$[3,4]$	$[3,4]$
P2.Y	$[5]$	$[5]$	$[5]$
P3.X	$[0,6]$	$[1.985,4.2]$	$[2.130,4.005)$
P3.Y	$[0,6]$	$[0,6]$	$[0,6]$
P4.X	$[0,6]$	$[1.985,4.2]$	$[2.130,3.96)$
P4.Y	$[0,6]$	$[0,6]$	$[0,6]$
Dicke	$[0.3,0.6]$	$[0.3,0.6]$	$[0.3,0.6]$
Höhe	$[2.2]$	$[2.2]$	$[2.2]$
Länge $\overline{P1-P2}$	$(0,\infty)$	$[4,5.100]$	$[4,5.100]$
Länge $\overline{P3-P4}$	$(0,\infty)$	$(0,6.396]$	$(0,6.195)$

Tabelle B.4: Beispiel *Trapez-Wand*: Ergebnisse der Optimierungen für ausgewählte *Parameter* (auf die 3. Dezimalstelle gerundet)

Parameter	**Max. Dicke**	**Skizze (a)**	**Skizze (b) (+ Länge 2)**
P1.X	3	3	3
P1.Y	0.856	1	1
P2.X	3.983	3.99	4
P2.Y	5	5	5
P3.X	3.498	3.036	3.100
P3.Y	5.55	2.37	2.637
P4.X	2.22	3.033	2.958
P4.Y	0.170	2.36	2.06
Dicke	0.6	0.303	0.301
Höhe	2.2	2.2	2.2
Länge $\overline{P1-P2}$	4.26	4.123	4.123
Länge $\overline{P3-P4}$	5.53	0.02	0.59

B.3 Beispiel: Valides Polygon

Tabelle B.5: Beispiel *Valides Polygon*: Berechnung der *globalen Hülle* für die Ausgangslage

Parameter	**Startwerte**	**Globale Hülle nach Cruz & HC4**
C_{1-0}	$(0, \infty)$	[2]
C_{2-1}	$(0, \infty)$	[1]
C_{3-0}	$(0, \infty)$	[4]
C_{4-1}	$(0, \infty)$	[3.5]
C_{4-2}	$(0, \infty)$	[2.5]
C_{5-3}	$(0, \infty)$	[3]
C_{6-2}	$(0, \infty)$	[5]
C_{6-4}	$(0, \infty)$	[2.5]
C_{7-6}	$(0, \infty)$	[1.5]
C_{7-11}	$(0, \infty)$	[5]
C_{8-7}	$(0, \infty)$	[2]
C_{9-6}	$(0, \infty)$	[5]
C_{9-8}	$(0, \infty)$	[1.5]
C_{10-5}	$(0, \infty)$	[8]
C_{10-9}	$(0, \infty)$	[2]
C_{10-11}	$(0, \infty)$	[10.5]
C_{11-1}	$(0, \infty)$	[2.5]

Tabelle B.6: Beispiel *Valides Polygon*: Ergebnisse der *globalen Hülle* bei der Lockerung der *schwarzen Constraint* C_2 (auf die 3. Dezimalstelle gerundet). Unterschiede der Algorithmen sind grau eingefärbt.

Parameter	**Startwerte**	**HC4**	**Globale Hülle**
C_2	$[-100, 100]$	$(7.5, 11)$	$(7.5, 11)$
H0.Y	$[0, 15]$	$[15]$	$[15]$
H1.Y	$[0, 15]$	$[13]$	$[13]$
H2.Y	$[0, 15]$	$[12]$	$[12]$
H3.Y	$[0, 15]$	$(9, 12.5)$	$(9, 12.5)$
H4.Y	$[0, 15]$	$(7.5, 11)$	$(7.5, 11)$
H5.Y	$[0, 15]$	$(6, 9.5)$	$(6, 9.5)$
H6.Y	$[0, 15]$	$[7]$	$[7]$
H7.Y	$[0, 15]$	$[5.5]$	$[5.5]$
H8.Y	$[0, 15]$	$[3.5]$	$[3.5]$
H9.Y	$[0, 15]$	$(0, 3.5)$	$(0, 3.5)$
H10.Y	$[0, 15]$	$[0]$	$[0]$
H11.Y	$[0, 15]$	$[10.5]$	$[10.5]$
C_{1-0}	$(0, \infty)$	$[2]$	$[2]$
C_{2-1}	$(0, \infty)$	$[1]$	$[1]$
C_{3-0}	$(0, \infty)$	$(2.5, 6)$	$(2.5, 6)$
C_{4-1}	$(0, \infty)$	$(2, 5.5)$	$(2, 5.5)$
C_{4-2}	$(0, \infty)$	$(1, 4.5)$	$(1, 4.5)$
C_{5-3}	$(0, \infty)$	$(0, 6.5)$	$(2.99, 3.02)$
C_{6-2}	$(0, \infty)$	$[5]$	$[5]$
C_{6-4}	$(0, \infty)$	$(0.5, 4)$	$(0.5, 4)$
C_{7-6}	$(0, \infty)$	$[1.5]$	$[1.5]$
C_{7-11}	$(0, \infty)$	$[5]$	$[5]$
C_{8-7}	$(0, \infty)$	$[2]$	$[2]$
C_{9-6}	$(0, \infty)$	$(3.5, 7)$	$(3.5, 7)$
C_{9-8}	$(0, \infty)$	$(0, 3.5)$	$(0, 3.5)$
C_{10-5}	$(0, \infty)$	$(6, 9.5)$	$(6, 9.5)$
C_{10-9}	$(0, \infty)$	$(0, 3.5)$	$(0, 3.5)$
C_{10-11}	$(0, \infty)$	$[10.5]$	$[10.5]$
C_{11-1}	$(0, \infty)$	$[2.5]$	$[2.5]$

Tabelle B.7: Beispiel *Valides Polygon*: Ergebnisse der *globalen Hülle* bei der Lockerung der *schwarzen Constraints* C_2 und C_5 (auf die 3. Dezimalstelle gerundet). Unterschiede der Algorithmen sind grau eingefärbt.

Parameter	**Startwerte**	**HC4**	**Globale Hülle**
C_2	$[-100, 100]$	$(7.5, 11)$	$(7.5, 11)$
C_5	$[-100, 100]$	$(5, 9.5)$	$(5, 9.5)$
H0.Y	$[0, 15]$	$[15]$	$[15]$
H1.Y	$[0, 15]$	$(10.5, 15)$	$(10.5, 15)$
H2.Y	$[0, 15]$	$(9.5, 14)$	$(9.5, 14)$
H3.Y	$[0, 15]$	$(9, 12.5)$	$(9, 12.5)$
H4.Y	$[0, 15]$	$(7.5, 11)$	$(7.5, 11)$
H5.Y	$[0, 15]$	$(6, 9.5)$	$(6, 9.5)$
H6.Y	$[0, 15]$	$[7]$	$[7]$
H7.Y	$[0, 15]$	$[5.5]$	$[5.5]$
H8.Y	$[0, 15]$	$[3.5]$	$[3.5]$
H9.Y	$[0, 15]$	$(0, 3.5)$	$(0, 3.5)$
H10.Y	$[0, 15]$	$[0]$	$[0]$
H11.Y	$[0, 15]$	$[10.5]$	$[10.5]$
C_{1-0}	$(0, \infty)$	$(0, 4.5)$	$(0, 4.5)$
C_{2-1}	$(0, \infty)$	$(0, 5.5)$	$(0.98, 1.01]$
C_{3-0}	$(0, \infty)$	$(2.5, 6)$	$(2.5, 6)$
C_{4-1}	$(0, \infty)$	$(0, 7.5)$	$(0.99, 7.5)$
C_{4-2}	$(0, \infty)$	$(0, 6.5)$	$(0, 6.5)$
C_{5-3}	$(0, \infty)$	$(0, 6.5)$	$(2.99, 3.02)$
C_{6-2}	$(0, \infty)$	$(2.5, 7)$	$(2.5, 7)$
C_{6-4}	$(0, \infty)$	$(0.5, 4)$	$(0.5, 4)$
C_{7-6}	$(0, \infty)$	$[1.5]$	$[1.5]$
C_{7-11}	$(0, \infty)$	$[5]$	$[5]$
C_{8-7}	$(0, \infty)$	$[2]$	$[2]$
C_{9-6}	$(0, \infty)$	$(3.5, 7)$	$(3.5, 7)$
C_{9-8}	$(0, \infty)$	$(0, 3.5)$	$(0, 3.5)$
C_{10-5}	$(0, \infty)$	$(6, 12)$	$(6, 9.5)$
C_{10-9}	$(0, \infty)$	$(0, 3.5)$	$(0, 3.5)$
C_{10-11}	$(0, \infty)$	$[10.5]$	$[10.5]$
C_{11-1}	$(0, \infty)$	$(0, 4.5)$	$(0, 4.5)$

Tabelle B.8: Beispiel *Valides Polygon*: Ergebnisse der Optimierungen bei der Lockerung der *schwarzen Constraints* C_2 und C_5 (auf die 3. Dezimalstelle gerundet)

Parameter	C_2 **für** 11	**Skizze (H2.Y=13, H9=3)**
C_2	11.00	10.26
C_5	8.63	5.76
H1.Y	14.13	11.26
H2.Y	13.13	10.25
H3.Y	12.49	11.76
H4.Y	10.99	10.26
H5.Y	9.49	8.76
H9.Y	3.49	2.76
C_{1-0}	0.87	3.74
C_{2-1}	1.00	1.00
C_{3-0}	2.51	3.24
C_{4-1}	3.13	1.00
C_{4-2}	2.13	0.01
C_{6-2}	6.13	3.26
C_{6-4}	3.99	3.26
C_{9-6}	3.51	4.24
C_{9-8}	0.01	0.74
C_{10-5}	9.49	8.76
C_{10-9}	3.49	2.76
C_{11-1}	3.63	0.76

B.4 Beispiel: Raum-begrenzt-durch-Wände

Tabelle B.9: Beispiel *Raum-begrenzt-durch-Wände*: Ergebnisse der *globalen Hülle* für ausgewählte *Parameter* (auf die 3. Dezimalstelle gerundet). Unterschiede der Algorithmen sind grau eingefärbt.

Parameter	**Startwert**	**HC4**	**Globale Hülle**
Raum.P1.X	$(-\infty,\infty)$	$[17,23]$	$(18.215,20)$
Raum.P1.Y	$(-\infty,\infty)$	$[17,20]$	$[17,20]$
Raum.P2.X	$(-\infty,\infty)$	$[20]$	$[20]$
Raum.P2.Y	$(-\infty,\infty)$	$[10]$	$[10]$
Raum.P3.X	$(-\infty,\infty)$	$[25.714,34.286]$	$(30,31.788)$

Parameter	Startwert	HC4	Globale Hülle
Raum.P3.Y	$(-\infty,\infty)$	$[10,16.429]$	$(10,13.002)$
Raum.P4.X	$(-\infty,\infty)$	$[30]$	$[30]$
Raum.P4.Y	$(-\infty,\infty)$	$[20]$	$[20]$
Raum.Fläche	$(0,\infty)$	$(0,325.918]$	$[87.65,100.17)$
Raum.Umfang	$(0,\infty)$	$[22.57,54.183]$	$(38.739,40.020)$
Wand1.P1.X	$[17,23]$	$(18.215,20)$	$[17,23]$
Wand1.P1.Y	$[17,23]$	$[17,20]$	$[17,20]$
Wand1.P2.X	$[30]$	$[30]$	$[30]$
Wand1.P2.Y	$[30]$	$[20]$	$[20]$
Wand1.P3.X	$[0,40]$	$[14.135,40]$	$(30.164,30.251)$
Wand1.P3.Y	$[0,40]$	$[13.500,30.429]$	$(20.289,20.366)$
Wand1.P4.X	$[0,40]$	$[12.812,26.690]$	$(17.944,19.801)$
Wand1.P4.Y	$[0,40]$	$[11.547,26.076]$	$(17.239,20.301)$
Wand2.P1.X	$[0,40]$	$[30]$	$[30]$
Wand2.P1.Y	$[0,40]$	$[20]$	$[20]$
Wand2.P2.X	$[0,40]$	$[25.714,34.286]$	$(30,31.788)$
Wand2.P2.Y	$[0,40]$	$[10,16.429]$	$(10,13.002)$
Wand2.P3.X	$[0,40]$	$[6.249,40]$	$(30.249,32.09)$
Wand2.P3.Y	$[0,40]$	$[0,35.465]$	$(9.749,12.82)$
Wand2.P4.X	$[0,40]$	$[17.735,40]$	$(30.164,30.251)$
Wand2.P4.Y	$[0,40]$	$[13.500,30.429]$	$(20.289,20.365)$
Wand3.P1.X	$[25,35]$	$[25.714,34.286]$	$(30,31.788)$
Wand3.P1.Y	$[10,17]$	$[10,16.429]$	$(10,13.002)$
Wand3.P2.X	$[20]$	$[20]$	$[20]$
Wand3.P2.Y	$[10]$	$[10]$	$[10]$
Wand3.P3.X	$[0,40]$	$[13.366,26.136]$	$(19.799,19.872)$
Wand3.P3.Y	$[0,40]$	$[0,22.905]$	$(9.692,9.761)$
Wand3.P4.X	$[0,40]$	$[6.250,40]$	$(30.249,32.09)$
Wand3.P4.Y	$[0,40]$	$[0,32.250]$	$(9.749,12.82)$
Wand3.P1.X	$[17,23]$	$[20]$	$[20]$
Wand3.P1.Y	$[10,23]$	$[10]$	$[10]$
Wand3.P2.X	$[17,23]$	$[17,23]$	$(18.215,20)$
Wand3.P2.Y	$[17,23]$	$[17,23]$	$[17,20)$
Wand3.P3.X	$[0,40]$	$[12.812,26.690]$	$(17.944,19.801)$
Wand3.P3.Y	$[0,40]$	$[11.547,26.076]$	$(17.239,20.301)$

Parameter	**Startwert**	**HC4**	**Globale Hülle**
Wand3.P4.X	$[0, 40]$	$[13.365, 26.136]$	$(19.799, 19.872)$
Wand3.P4.Y	$[0, 40]$	$[0, 24.784]$	$(9.692, 9.761)$