Ficucs
Ein Constraint-Solver für Geometrische Constraints in 2D und 3D

-

Algorithmen, Einsatzfälle, Modellierung

Dissertation
zur Erlangung des akademischen Grades Doktoringenieur (Dr.-Ing.)

vorgelegt der Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau

von Dipl.-Inf. Ulf Döring
geboren am 4. Mai 1969 in Potsdam

vorgelegt am 12. Januar 2010

verteidigt am 21. Januar 2011

1. Gutachter: Prof. Dr. sc. techn. Beat D. Brüderlin, TU Ilmenau

2. Gutachter: Prof. Dr. rer. nat. habil. Dr. h. c. Karl-Heinz Modler, TU Dresden

3. Gutachter: Univ.-Prof. Dr.-Ing. Burkhard Corves, RWTH Aachen

URN: urn:nbn:de:gbv:ilm1-2011000437
URL: http://www.dmg-lib.org/dmglib/handler?docum=16733009

Ulf Döring
Technische Universität Ilmenau
Fachgebiet Computergrafik
Postfach 100565
D-98684 Ilmenau

Tel.: 03677 69 1211
Email: ulf.doering@tu-ilmenau.de

Ficucs
Ein Constraint-Solver für Geometrische Constraints in 2D und 3D

Algorithmen, Einsatzfälle, Modellierung

Die vorliegende Arbeit spiegelt Ergebnisse einer langjährigen Forschungs- und Entwicklungstätigkeit auf dem Gebiet der geometrischen Constraints zu Standardgeometrien in 2D und 3D wider. Die Erfahrungen zu konstruktiven und numerischen Ansätzen für das Berechnen der constraintbasierten Modelle flossen in die Entwicklung des Constraint-Solvers Ficucs ein. Neben den in Ficucs genutzten Algorithmen und Datenstrukturen werden diverse Applikationen, welche Ficucs als Berechnungsmodul nutzen, sowie dazugehörige Modelle beschrieben.

Ziel der Tätigkeiten war es, einen in den verschiedensten Projekten einsetzbaren Constraint-Solver zu entwickeln. Besonderes Augenmerk lag auf der Interaktivität sowie der Stabilität der Algorithmen, welche für eine Nutzerakzeptanz sehr wichtig sind. Hierzu wurden immer wieder Verbesserungsmöglichkeiten gefunden, zum Großteil auch implementiert und somit verifiziert. Es zeigte sich, dass die Kombination unterschiedlicher Konzepte in einer Anwendung oft problematisch ist. Die vorliegende Arbeit soll die erreichten Resultate interessierten Lesern zugänglich machen und zu einer weiteren Forschungstätigkeit in Richtung eines hybriden Ansatzes aus konstruktiven und numerischen Verfahren anregen.

Ficucs
a Constraint Solver for Geometric Constraints in 2D and 3D

Algorithms, Applications, Modeling

This thesis reflects the results of many years of research and development work in the field of geometric constraints for standard geometries in 2D and 3D. The experiences with constructive as well as numerical approaches for the calculation of constraint-based models have been continuously incorporated into the development of the constraint solver Ficucs. Besides the algorithms and data structures used in Ficucs the thesis describes various applications which use Ficucs as calculation module and corresponding models.

The aim was to develop a constraint solver which is usable in many different kinds of projects. The focus was on interactivity and stability of the algorithms, because that is very important for the user's acceptance. A lot of improvements were found, most of them were also implemented and thus verified. It arose that a combination of different concepts in one application is often problematic. The thesis shall make the achieved results accessible to the interested readers and motivate further research work concerning a hybrid approach which combines constructive as well as numerical methods.

Hiermit möchte ich mich ganz herzlich bei all denen bedanken, die mich direkt oder indirekt beim Verfassen dieser Arbeit unterstützt haben. Besonderer Dank geht an meine Kollegen aus den Fachgebieten Computergrafik und Konstruktionstechnik sowie die Partner aus den verschiedensten Forschungs- und Entwicklungsprojekten, durch die ich immer wieder wertvolle Anregungen für meine Forschungen erhielt. Ganz herzlich bedanke ich mich auch bei meiner Frau, deren Korrekturlesen sicher wesentlich zur Qualität dieser Arbeit beigetragen hat. Vielen Dank!

Inhaltsverzeichnis

Abkürzungsverzeichnis

0D	null-dimensional, siehe Abschnitt 3.7
1D	ein-dimensional, z. B. die Zahlengerade
2D	zwei-dimensional, z. B. die euklidische Ebene „in 2D“ entspricht „in der euklidischen Ebene“
3D	drei-dimensional, z. B. der euklidische Raum „in 3D“ entspricht „im euklidischen Raum“
agl	engl: angle, Winkel
allg.	allgemein
arg	Argument bzw. Parameter
BG_M0	Algorithmus zum Finden einer ersten Freiheitsgradverteilung, siehe Seite 52
bri	Kürzel für „Intervallbreite“ bzw. „Breite des Intervalls“
bzgl.	bezüglich
bzw.	beziehungsweise
C++	Programmiersprache, in der Ficucs implementiert wurde
ca.	zirka
CAD	engl.: Computer Aided Design
CA-Set	eine Art von Winkelcluster nach [VSR92], siehe Seite 94
CAx	Verallgemeinerung von *CAD* und ähnlichen Technologien
CD-Set	2D-Cluster nach [VSR92], siehe Seite 95
CLP	engl.: Constraint Logic Programming
Cns	Constraint
COSMOS	Forschungsprojekt für die Entwicklung eines Modellierers, dessen Name sich vom englischen „Conceptual Modeling System“ ableitet
d. h.	das heißt
dir	engl.: direction, Richtung von Geraden oder Vektoren
dist	engl.: distance, Abstand zwischen geometrischen Objekten
DMG-Lib	Digitale Mechanismen- und Getriebebibliothek
DOF	engl.: Degree of Freedom, Freiheitsgrad(e)
engl.	Englisch
et al.	und anderen
ext	Kürzel für „Extremwerte“
ff.	und folgende Seiten
FG	Freiheitsgrad(e), engl.: Degree of Freedom
Ficucs	Name des vorgestellten Constraint-Solvers (engl: Fast Iterative ConstrUctive Constraint Solver)
ges	Kürzel für „gesamt“
ggf.	gegebenenfalls
GML	Generative Modelling Language
gnd	Kürzel für „Grenze nicht definiert“
ID	Identifikator, z. B. in Form von Bezeichnern oder Speicheradressen
Kfz	Kraftfahrzeug

KONNI	KONsisteNte Modellerstellung und realitätsnahe Präsentation im Internet
LHS	Linkshändiges System, z. B. für Koordinaten oder als Hinweis
LnEq	lineare Gleichung
MAP	Vorgehensweise beim Finden einer nächsten Zuordnung. Der Fokus liegt auf der Beibehaltung hoch priorisierter Freiheitsgrade. Siehe Abschnitt 3.3.4 auf Seite 55
MAP1	einfacher *MAP*-Algorithmus, siehe Abschnitt 3.3.5
MAPG	*MAP*-Algorithmus, der Informationen über Gruppierungen beachtet, siehe Abschnitt 3.3.6
MASP	Programmsystem, der Name leitet sich vom englischen „Modeling and Analyses of Solution Principles“ ab
MIP	Vorgehensweise beim Finden einer nächsten Zuordnung. Der Fokus liegt auf der Vermeidung niedrig priorisierter Freiheitsgrade. Siehe Abschnitt 3.3.4 auf Seite 55
MIP1	einfacher *MIP*-Algorithmus, siehe Abschnitt 3.3.5
MIPG	*MAP*-Algorithmus, der Informationen über Gruppierungen beachtet, siehe Abschnitt 3.3.6
MXP	Vorgehensweise beim Finden einer nächsten Zuordnung, die *MAP* und *MIP* vereint
MXPGT	*MXP*-Algorithmus, der Informationen über Gruppierungen und Typen beachtet
nD	n-dimensional, Überbegriff für *0D* bis *3D* und höhere Dimensionen
nrm	Normale von Ebenen (Richtung, die senkrecht zur Ebene steht)
o. Ä.	oder Ähnliche[s]
Obj	Objekt
pln	engl.: plane, Ebene
pnt	engl.: point, Punkt in *2D* oder *3D*, im Constraint-Solver synonym zu Vertex (*vtx*)
PpdP	Constraint: „Plane parallel-distance Plane“, paralleler Abstand zweier Ebenen
RHS	Rechtshändiges System, z. B. für Koordinaten oder als Hinweis
SEN	Algorithmus zum Suchen der ersten Nullstelle (vgl. *SWN*), siehe Seite 85
SNI	Bezeichnung für Algorithmen zum Suchen einer Nullstelle in einem bestimmten Bereich, siehe Seite 84
SNI_{bri}	Algorithmus zur Nullstellensuche durch Intervallunterteilung, siehe Seite 85
SNI_{ext}	Algorithmus zur Nullstellensuche durch Untersuchung von Extrema, siehe Seite 84
SNI_{gnd}	Algorithmus zur Nullstellensuche durch Untersuchung von Grenzen zu nicht definierten Bereichen, siehe Seite 85
SNI_{vzw}	Algorithmus zur Nullstellensuche durch Untersuchung von Intervallen mit Vorzeichenwechseln, siehe Seite 84
s. o.	siehe oben
SWN	Algorithmus zum Suchen einer weiteren Nullstelle (vgl. *SEN*), siehe Seite 85
u. a.	unter anderem
usw.	und so weiter
u. U.	unter Umständen
Val	Valenz, negative Freiheitsgrade
VeqV	Constraint: „Vertex equal Vertex“, Gleichheit zweier Punkte
vgl.	vergleiche
VonL	Constraint: „Vertex on Line“, Punkt befindet sich auf der Geraden
vs.	versus, gegenüber, impliziert Vergleich
VtoL	Constraint: „Vertex to Line“, Abstand eines Punktes zu einer Geraden
vtx	Vertex, topologischer Punkt, im Constraint-Solver synonym zu Punkt (*pnt*)
vzw	Kürzel für „Vorzeichenwechsel“
XHS	Verallgemeinerung von *LHS* und *RHS*
z. B.	zum Beispiel

Kapitel 1

Einführung

1.1 Geometrische Constraints in Forschung und Praxis

Schon seit vielen Jahren laufen Forschungen hinsichtlich dessen, was heute typischerweise als „geometrische Constraints“ bezeichnet wird. Gemeint sind hiermit Zusammenhänge[1] zwischen geometrischen Objekten wie z. B. Punkten, Kreisen und Flächen, aber auch Parametern, die eine geometrische Bedeutung haben (z. B. Abstände und Winkel). Im Mittelpunkt der Forschung steht die rechentechnische Umsetzung einer constraint-basierten Modellierung und den darauf aufbauenden Berechnungsabläufen.

In den letzten Jahren haben die Forschungsergebnisse zunehmend Einzug in kommerzielle Applikationen gefunden, wobei der Schwerpunkt auf dem CAD-Bereich liegt. Aber auch in den Geowissenschaften, der Chemie, der Biochemie, der Unterhaltungsbranche sowie beim multimedialen Lernen finden sich eine Vielzahl von Anwendungen für geometrische Constraints.

1.1.1 Einführendes Beispiel

Zur Verdeutlichung der Semantik von geometrischen Constraints wird zunächst ein sehr einfaches Modell betrachtet.

Beispiel 1.1: Abstand eines Punktes von einer Geraden
Abbildung 1.1 zeigt ein einfaches Modell, in dem ein Punkt P einen bestimmten Abstand D zu einer Geraden G hat. Abstrakt gesehen handelt es sich hierbei um eine Beziehung zwischen den drei Objekten[2] P, D und G. Zur Sicherung der Konsistenz (respektive Einhaltung der Beziehung bzw. des Constraints) kann nun:

- getestet werden, ob der Punkt momentan den richtigen Abstand zur Geraden hat. Wenn das so ist, ist der Constraint erfüllt und es muss nichts weiter berechnet werden.
- Bei fehlgeschlagenem Test oder auch stets[3] berechnet werden:
  - P' aus gegebenen G und D,
  - G' aus gegebenen P und D,
  - D' aus gegebenen P und G.

---

[1] auch bezeichnet als: Beziehungen, Relationen, Restriktionen, Zwänge
[2] Näheres zur Verwendung des *Objekt*-Begriffs in dieser Arbeit siehe Seite 10.
[3] Berechnungen ohne vorherige Tests entsprechen einem eher pessimistischen Ansatz, der sich jedoch bei annähernd gleichem Rechenaufwand von Test und Berechnung bzw. meist fehlschlagenden Tests als der bessere erweist.

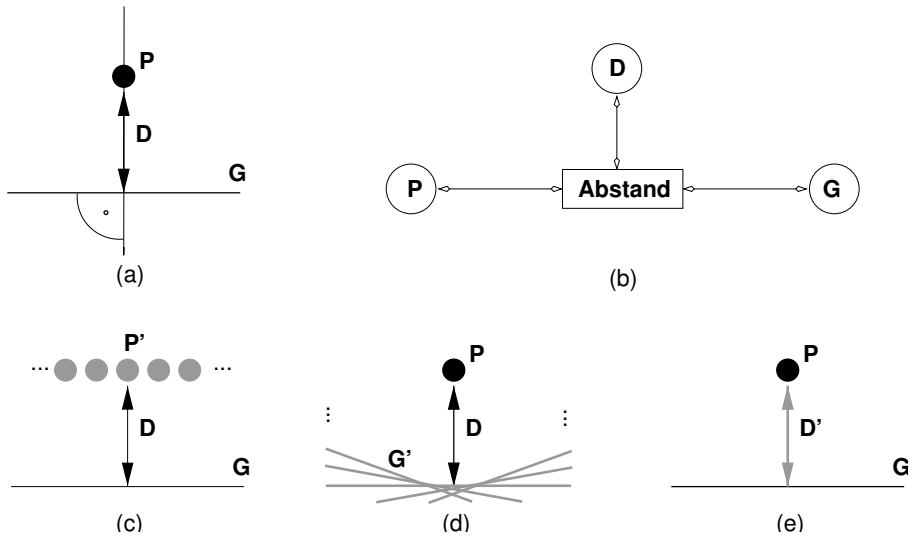


Abbildung 1.1: Einführendes Beispiel: (a) geometrische Beschreibung, (b) abstrakte Darstellung, (c) Berechnungsmöglichkeiten für P', (d) Berechnungsmöglichkeiten für G', (e) Berechnung von D'

Entsprechend den Vorstellungen des Nutzers[4] oder auch bestimmten Zwängen[5] ist die günstigste Aktion auszuwählen. Von den drei angegebenen Berechnungsvarianten ist allerdings nur die letzte (die Berechnung des Abstandes) eindeutig[6]. Bei der Berechnung von Punkt oder Gerade muss eine sogenannte Unterbestimmtheit in geeigneter Weise behandelt werden. In der Abbildung sind einige mögliche Lösungen grau dargestellt. Welche der (typischerweise unendlich vielen) Lösungen die beste ist, hängt von der Semantik des Modells ab. Eine einfache Möglichkeit ist die Auswahl des Punktes (bzw. der Geraden) die am nächsten zur letzten konsistenten Lösung ist[7].
Grundsätzlich sind jedoch alle Lösungen zulässig, bei denen der Abstands-Constraint erfüllt ist, also auch solche, bei denen sowohl Punkt und Grade als auch Parameter andere Werte angenommen haben. Ob eine solche Lösung vom Nutzer akzeptiert wird bzw. im Kontext der Lösung eines übergeordneten Problems sinnvoll ist, hängt wiederum vom konkreten Anwendungsfall ab. □

Im Laufe der Zeit wurden von Forschern unterschiedlichste Aspekte der Handhabung von Constraints untersucht und immer wieder andere Lösungsmöglichkeiten gefunden. Die Motivation war hierbei teilweise akademischer Natur zum anderen bestimmten aber auch praktische Erwägungen Forschungsrichtung und Lösungsansatz. Im folgenden Text soll auf verschiedene Aspekte eingegangen werden, wobei immer wieder Bezug auf das einführende Beispiel genommen wird.

1.1.2 Überblick über constraint-basierte Ansätze

Schon seit jeher werden in den Natur- und Ingenieurwissenschaften Zusammenhänge aus Natur und Technik beschrieben. Durch eine mehr oder weniger starke Abstraktion der dort beobachteten Eigenschaften und das Einhalten einer bestimmten Notation werden Modellbeschreibungen gewonnen. Aus Modellbeschreibungen lassen sich je nach Bedarf mittels Transformationen[8] weitere Modellbeschreibungen ableiten, die für die jeweiligen Nutzer von Interesse sind.

---

[4] Er kann z. B. interaktiv den Punkt bewegen und wird erwarten, dass die Gerade mit verschoben wird.
[5] Zum Beispiel können Gerade und Abstand fixiert sein, d. h. der Punkt muss sich bei Verschiebungen parallel zur Gerade bewegen.
[6] Man spricht hier von einem vollbestimmten Problem.
[7] So wird eine Art Trägheit modelliert.
[8] Das Spektrum reicht hierbei von menschlicher Intuition bis hin zu implementierten Algorithmen.

Tabelle 1.1: Vergleich einer allgemeinen Vorgehensweise mit der Vorgehensweise in Ficucs

Eine allgemeine Vorgehensweise	Vorgehensweise beim Umgang mit geometrischen Constraints für/in Ficucs
Erkennen von Zusammenhängen und Bildung von Modellen, z. B. als Gleichungssysteme	manuelles Modellieren oder automatische Constraint-Erkennung und -Generierung (evtl. in Verbindung mit einer Featureerkennung), Ziel ist eine constraint-solver-konforme Repräsentation
Umformung der Gleichungssysteme	automatische Erstellung eines Konstruktionsplanes, vgl. Abschnitt 2.1.2
Propagation von Werten in den umgeformten Gleichungssystemen	Berechnen eines sequenziellen Konstruktionsplanes, vgl. Abschnitt 2.1.2
falls nötig: Anwendung numerischer Methoden zum iterativen Lösen der Gleichungssysteme	falls nötig: Nullstellensuche mit iterativen Konstruktionen, vgl. Abschnitt 2.7

Auch die Anwendung von geometrischen Constraints soll diesem Zweck dienen. Geometrische Constraints sind im obigen Sinne ein Mittel zur Modellierung geometrischer Zusammenhänge, die aus der Umwelt abstrahiert wurden. Für die Bildung abgeleiteter Modelle stehen eine Vielzahl von Methoden zur Verfügung, die heutzutage meist computerunterstützt ausgeführt werden können. Ein typisches Ergebnis der entsprechenden constraint-basierten Berechnungen ist ein Modell, dessen Elemente (z. B. Punkte, Kanten und Ebenen) so angeordnet wurden, dass sowohl die in den Constraints beschriebenen Zusammenhänge als auch spezielle Anforderungen der Nutzer erfüllt werden[9]. In Tabelle 1.1 sind eine allgemeine Vorgehensweise sowie die Vorgehensweise in Ficucs, dem in dieser Arbeit erörterten Constraint-Solver, gegenübergestellt.

Beispiel 1.2: Abstand einer Radachse vom Untergrund
Das im einführenden Beispiel 1.1 beschriebene Modell könnte sehr abstrakt den Abstand einer Radachse vom Untergrund beschreiben. Bei der Modellbildung wurden sämtliche technischen Details des zugehörigen Fahrzeuges vernachlässigt. Außerdem wurde das ursprünglich räumliche Modell (*3D*) auf ein planares Modell (*2D*) abgebildet. Als Notation wurde eine auf geometrischen Constraints basierende Beschreibung verwendet. Eine alternative Notation könnte u. a. in der Angabe einer Gleichung bestehen.
Das zunächst gewonnene Modell könnte weiteren Transformationen unterworfen werden, zum Beispiel weil ein Nutzer eine Berechnungsvorschrift für die möglichen Lagen des Punktes benötigt (wenn Gerade und Abstand bekannt sind). Einen anderen Nutzer könnte hingegen eine Vorschrift für die Berechnung der möglichen Lagen der Gerade interessieren (wenn der Punkt und der Abstand bekannt sind). Die Herleitung der entsprechenden Vorschrift stellt eine problemabhängige Transformation dar. Das Ergebnis der Berechnungen ist jeweils ein konsistentes Modell, das sowohl den Abstands-Constraint als auch das aktuelle Interesse des Nutzers berücksichtigt. □

Ohne Zweifel ist eine Vielzahl mathematischer Verfahren, die heutzutage die Grundlage der rechentechnischen Realisierung constraint-basierter Ansätze bilden, bereits lange vor dem Einsatz von Computern in Benutzung gewesen. Im Allgemeinen handelt es sich hierbei um numerische Verfahren zur Lösung von linearen und nicht-linearen Gleichungssystemen. Speziell für den Um-

[9] Zum Beispiel könnte ein Nutzer fordern, dass in Beispiel 1.1 die Gerade bei Verschiebung des Punktes zwar per Constraint mit verschoben wird, sich ihre Richtung jedoch nicht (oder in komplexeren Modellen möglichst wenig) ändert. Die Berücksichtigung derartiger nur implizit gegebener Constraints ist oft wesentlich für die Nutzerakzeptanz.

gang mit geometrischen Constraints sind aber auch eine ganze Reihe weiterer Arbeiten interessant, u. a.:

- zu geometrischen Konstruktionen, die in der Elementargeometrie behandelt werden (z. B. [Adl06]),
- zur graphischen Statik, wo wesentliche Überlegungen bezüglich der Freiheitsgradanalyse gefunden werden können ([Hen11]) und
- zur graphischen Kinematik (Federhofer [Fed32] äußert sich z. B. zur Möglichkeit einer deklarativen Beschreibung graphischer Lösungswege).

Als das erste Programm, in dem geometrische Constraints rechentechnisch umgesetzt wurden, wird Sketchpad ([Sut63a] und [Sut63b]) angesehen. Seither erfolgte die Beschreibung einer Vielzahl von Ansätzen, die zur Umformung einer impliziten Modelldarstellung (eine Menge von Objekten und Constraints) in eine explizite Modelldarstellung (Objekte mit Zahlenwerten, die die Constraints erfüllen) dienen. Es lassen sich zwei Hauptgruppen unterscheiden:

- Verfahren, in denen die Menge der Constraints sehr abstrakt als ein *Gleichungssystem* repräsentiert ist[10], welches meist als Ganzes gelöst wird (*simultane* symbolische oder numerische Lösung, siehe Anhang B), und
- Verfahren, in denen eine *sequenzielle* Anwendung von *Konstruktionsregeln* erfolgt (*konstruktiver* Ansatz). Bei der Lösungsfindung lässt sich hier die eigentliche geometrische Aufgabenstellung wesentlich besser berücksichtigen.

Zudem existiert eine ganze Reihe *hybrider* Verfahren, die eine Kombination unterschiedlichster Ansätze darstellen, so vereinigte bereits Sketchpad in sich sowohl den konstruktiven als auch den algebraischen (numerischen) Ansatz. Die Überführung der Constraints in ein Gleichungssystem hat den Vorteil, dass entsprechend der Mächtigkeit des gewählten (Standard-)Algorithmus meist eine Lösung gefunden wird. Wie sich jedoch zeigen lässt, ist bei *numerischer* Lösung sowohl die Steuerbarkeit als auch die Robustheit der Lösungssuche nicht stets gewährleistet und bei *symbolischer* Lösung ist der enorme Bedarf an Rechenzeit und Speicher zu berücksichtigen. Die *konstruktive* Lösung läuft meist schneller ab, außerdem lassen sich grundsätzlich die Spezialfälle in den einzelnen Konstruktionen berücksichtigen. Jedoch gibt es viele relevante Modelle, die mit einem rein konstruktiven Ansatz nicht berechnet werden können. Zudem kann die Implementierung der Konstruktionen aufwändig sein. Im Beispiel 1.1 wurden zwar nur drei verschiedene Typen von Konstruktionen benötigt. Allgemein wächst die Anzahl der benötigten Typen von Konstruktionen jedoch kombinatorisch mit der Anzahl unterstützter Constraint-Typen, siehe Abschnitt 3.4.1.

In Ficucs werden konstruktiver und numerischer Lösungsansatz vereint, wobei der Schwerpunkt auf dem konstruktiven liegt. Allgemeine Verfahren zur numerischen Lösung sind in Abschnitt 4.2.3 sowie in Anhang B näher beschrieben. Die auf der Anwendung von Konstruktionsregeln basierten Ansätze lassen sich entsprechend der Handhabung des sogenannten Konstruktionsplanes[11] einteilen in:

- Verfahren mit statischem Plan

  Hier lassen sich wiederum zwei Gruppen unterscheiden:

  - *History-basierte Verfahren*
    Es existiert ein fest vorgegebener Plan, der üblicherweise ein Protokoll der vom Nutzer durchgeführten Modellierungsschritte darstellt. Das Protokoll kann als die Entstehungsgeschichte (*history*) des Modells angesehen werden. Dieser Plan lässt sich später auch mit anderen Parametern ausführen.

[10] Diese Verfahren werden deshalb auch als *algebraische* Verfahren bezeichnet.

[11] Der Konstruktionsplan stellt eine Sequenz von einzelnen Konstruktionsschritten dar. Im Folgenden wird auch vereinfacht vom Plan gesprochen.

– *Variantenprogrammierung*
Entsprechend den jeweils angebotenen Sprachkonstrukten (z. B. Verzweigungen, Schleifen, Datenbankzugriffe, Nutzung externer Programme...) ergeben sich Modellierwerkzeuge von unterschiedlicher Mächtigkeit. Mit ihnen könnte auch die Lösung von zyklischen Abhängigkeiten programmiert werden. Das Problem liegt insbesondere in der meist aufwändigen Erstellung und Wartung der Programmsysteme. Schon zu Anfängen des „Computerzeitalters“ erörterte Konrad Zuse das Problem der Erstellung von Rechenplänen [Zus48]. Aber auch heute noch erfolgt die Erzeugung der Befehlssequenzen meist „per Hand“.

Bei den Verfahren mit statischem Plan liegt das Hauptproblem darin, dass in jeder Berechnungssequenz implizit Ein- und Ausgänge festgelegt sind. Entsprechend einer durch den Nutzer interaktiv oder in einem Skript definierten Reihenfolge ist nur eine bestimmte Konstruktionsrichtung möglich. Im einführenden Beispiel 1.1 könnte also nur der Punkt ***oder*** die Gerade ***oder*** der Abstandsparameter berechnet werden[12]. Für komplexe Aufgabenstellungen mit wechselnden Ein- und Ausgängen ist u. U. eine kombinatorische Vielfalt von Berechnungssequenzen notwendig, deren Implementierung und Wartung nicht durch einen Programmierer vorgenommen werden kann. Dynamische Pläne, d. h. die automatische Generierung der Berechnungssequenzen in einem constraint-basierten System, stellen eine sinnvolle Alternative dar.

Entsprechend der Festlegung, dass sich aus Constraints verschieden gerichtete Abhängigkeiten ableiten lassen (was sich z. B. in verschiedenen Konstruktionsrichtungen bzw. unterschiedlichen Konstruktionssequenzen niederschlägt), sind history-basierte Ansätze im engeren Sinne nicht constraint-basiert, denn hier ist die Richtung der Abhängigkeiten fest vorgegeben. Analog ist die Variantenprogrammierung einzuschätzen. Es ist jedoch vorstellbar, dass sich auch mittels einer komplexeren Variantenprogrammierung aus einer Beziehung verschieden gerichtete Abhängigkeiten für die Konstruktion ableiten lassen[13].

- Verfahren mit dynamischen Plan

  Hier sollen konstruktive Verfahren eingeordnet werden, die in der Lage sind, auf Basis der gegebenen Objekte und Constraints selbständig einen Konstruktionsplan zu generieren, mit dessen Hilfe sich hinsichtlich der Constraints konsistente Geometriedaten ableiten lassen. Die Erstellung des Planes kann durch eine *graph-basierte* Suche sehr effizient erfolgen. Ältere Verfahren, die zumeist mit Hilfe von Expertensystemen oder direkt in der Programmiersprache Prolog umgesetzt wurden und auch als rein *regelbasiert* bezeichnet werden, sind gegenüber der graph-basierten Suche wesentlich langsamer. Die Mächtigkeit der Verfahren hängt im Wesentlichen davon ab, wie viele Konstruktoren implementiert wurden und ob eine Lösung zyklischer Abhängigkeiten (durch Nutzung numerischer Verfahren) möglich ist. Zudem gibt es verschiedenste Clusteringansätze, die dazu dienen, konstruktive (aber auch numerische) Verfahren effizienter anwenden zu können (vgl. Abschnitt 3.7).

Neben der obigen Einteilung der Evaluierungsmethoden lassen sich noch andere Unterscheidungen bzw. Zuordnungen vornehmen. So kann z. B. die Art der Parametrisierung in *struktur-* und *dimensionsbezogen* eingeteilt werden. Durch eine strukturbezogene Parametrisierung werden die Anzahl und der Typ von Objekten sowie die entsprechenden Verknüpfungen über Constraints manipulierbar gemacht. Das ist bei einer dimensionsbezogenen Parametrisierung nicht möglich. Hier können nur Werte von Variablen geändert werden. Die Nutzung von Strukturparametern bei der constraint-basierten Erzeugung der Geometriedaten ist nicht trivial und wird daher allgemein (noch) nicht unterstützt. Am besten lassen sich Strukturänderungen mittels Variantenprogrammierung umsetzen, die danach die Lösung entsprechend den dimensionsbezogenen Parametern berechnet oder in einem externen Constraint-Solver berechnen lässt.

[12] Da bei history-basierten Ansätzen die Parameter typischerweise als Eingang angenommen werden, wäre hier die Berechnung des Abstandes nicht vorgesehen.

[13] Möglicherweise auch durch die Einbindung eines Gleichungslösers oder eines speziellen Constraint-Solvers.

1.1.3 Einordnung des vorgestellten constraint-basierten Ansatzes

Der im Rahmen dieser Arbeit vorgestellte und im Constraint-Solver *Ficucs* realisierte Ansatz arbeitet grundsätzlich konstruktiv. Hierbei werden Standardgeometrien in *2D* und in *3D* unterstützt[14]. Für Teilprobleme, die nicht konstruktiv gelöst werden können, wird auf numerische Iterationen zurückgegriffen (siehe Erläuterungen ab Seite 82). Diese beruhen auf einem eigens für den Einsatz in Ficucs entwickelten Verfahren, das sowohl schnell als auch robust ist. Die Kombination von konstruktivem und numerischem Ansatz verschafft Ficucs einen wesentlichen Vorteil gegenüber rein konstruktiven Ansätzen, welche viele der in dieser Arbeit gezeigten Modelle nicht berechnen können. Zudem weist Ficucs bessere Eigenschaften als die oft genutzten numerischen Verfahren wie das Newton-Raphson-Verfahren oder die Homotopie auf (siehe Vergleich in Abschnitt 4.2.3). In Ficucs lässt sich die Auswahl der vom Nutzer erwarteten Lösung wesentlich besser steuern[15] und die Gefahr, dass eine Lösung nicht gefunden wird, ist bei weitem nicht so groß. Im Vergleich mit Constraint-Solvern, in denen komplexere Clustering-Ansätze implementiert sind (Abschnitt 3.7), zeigt sich ein Nachteil von Ficucs, denn in ihm sind nur sehr einfache Formen des Clusterings realisiert, so dass er für die Berechnung einiger Modelle auf numerische Iterationen zurückgreifen muss, obwohl eine rein konstruktive Lösung möglich wäre. Zudem kann Ficucs keine strukturbezogenen Constraints zur Erzeugung und Löschung von Objekten aus dem Constraint-Netz bearbeiten. Abschnitt 2.3 zeigt jedoch, dass sich ähnliche Effekte durch die Nutzung der von Ficucs unterstützten Conditional-Constraints erreichen lassen.

Wie in Abschnitt 5.1 gezeigt wird, lässt sich Ficucs grundsätzlich für die Lösung von Gleichungssystemen einsetzen. Im Allgemeinen ist hierfür jedoch die Nutzung entsprechender numerischer Verfahren zu empfehlen. So ist Ficucs auch zum Einsatz in physikbasierten Simulationen grundsätzlich ungeeignet. Die Lösung der entsprechenden Differentialgleichungen wird sinnvollerweise mit den hierfür entwickelten Verfahren durchgeführt. Ficucs lässt sich in derartigen Systemen allenfalls als Teilmodul zur Berechnung der Form starrer Körper einsetzen, deren Bewegungen simuliert werden sollen.

Ficucs ist durch seine Ausrichtung auf geometrische Konstruktionen sehr gut zur Ermittlung der Anordnungen bzw. Positionen von Standardgeometrien geeignet, wenn das entsprechende Modell vollbestimmt[16] ist. Zudem kann der Constraint-Solver auch oft bei Unterbestimmtheit und konsistenter Überbestimmtheit erfolgreich eingesetzt werden. Nicht möglich sind jedoch Ausgleichsrechnungen in überbestimmten Modellen, wie sie z. B. in der Geodäsie[17] oder der (Bio-)Chemie[18] nötig sind. In derartigen Fällen ist die Anwendung von Methoden wie die *Methode der kleinsten Quadrate* (nach Gauß bzw. Legendre [Wol90]) sinnvoll. Denkbar wäre für derartige Ausgleichsprobleme, dass mit Ficucs ein vollbestimmtes Teilmodell berechnet wird. Inkonsistente Überbestimmtheiten werden hierbei von Ficucs ignoriert[19]. Die Berechnungen erfolgen typischerweise sequenziell ähnlich einer Abfolge von Konstruktionsschritten mit Zirkel und Lineal, wo nötig können aber auch komplexere Teilmodelle numerisch gelöst werden. So kann mit Ficucs sehr schnell ein sinnvoller Startwert für die Ausgleichsrechnung gefunden werden. Die Problematik der Wahl von Startwerten für iterative numerische Berechnungen wird in Abschnitt 4.2.3 verdeutlicht.

---

[14] In *2D* sind das beim jetzigen Stand der Implementierungen Punkte, Geraden sowie Kreise mit festem Radius und in *3D* sind das Punkte sowie Ebenen. Komplexere Geometrien wie z. B. Zylinder werden nur teilweise von Ficucs direkt unterstützt. Es besteht jedoch die Möglichkeit, komplexere Geometrien aus einfachen Geometrien und entsprechenden Constraints zu modellieren.

[15] Bei iterativen numerischen Verfahren sollte besser überhaupt nicht von einer Steuerung bzw. Steuerbarkeit der Lösungsauswahl gesprochen werden (siehe Seite 117).

[16] Vergleiche Abschnitt 2.2.2 zur Bedeutung von Voll-, Über- und Unterbestimmtheit.

[17] Unterschieden werden Triangulation (in einem zu berechnenden Dreiecksnetz ist eine Länge bekannt, ansonsten nur die Winkel) und Trilateration (entfernungsmessungsbasiert). Hauptaugenmerk liegt meist auf der Unsicherheit der Messungen, weshalb Ausgleichsrechnungen für Messfehler durchzuführen sind. Entsprechende Verfahren werden heute auch zur Bestimmung von Molekülstrukturen angewendet. Ausgleichsrechnungen zur Behandlung von Überbestimmtheit sind mit Ficucs nicht möglich.

[18] In diesem Zusammenhang wird auch oft von Distanzgeometrie gesprochen, [Hav97].

[19] Oder als Hinweis zur Auswahl aus Mehrfachlösungen genutzt, vgl. Abschnitt 2.6.

Das Haupteinsatzgebiet von Ficucs stellen Anwendungen dar, in denen geometrische Modelle interaktiv aufgebaut werden und für die entsprechende Berechnungen zur Sicherstellung der Modellkonsistenz durchzuführen sind. Entsprechend dem Zweck der jeweiligen Applikation lassen sich so interaktive Modellanpassungen und auch Animationen für unterschiedlichste Modelle berechnen (siehe Kapitel 5). Außer der mehr oder weniger abstrakten geometrischen Beschreibung von gegenständlichen Objekten können die Modelle jedoch auch die Beschreibung geometrischer Verfahren beinhalten. In Abschnitt 5.2.3 wird hierfür ab Seite 137 ein entsprechendes Beispiel zur Berechnung von Kräften in Tragwerken gegeben.

1.2 Fokus dieser Arbeit

Die vorliegende Arbeit spiegelt die Ergebnisse einer langjährigen Forschungs- und Entwicklungstätigkeit auf dem Gebiet der geometrischen Constraints zu Standardgeometrien in *2D* und *3D* wider, das sind insbesondere:

- die Erfahrungen zu konstruktiven und numerischen Ansätzen,
- der Constraint-Solver Ficucs sowie
- diverse Applikationen, die Ficucs nutzen.

Ziel der Tätigkeiten war es, einen in den verschiedensten Projekten einsetzbaren Constraint-Solver zu entwickeln. Besonderes Augenmerk lag auf der Interaktivität sowie der Stabilität der Algorithmen, welche für eine Nutzerakzeptanz sehr wichtig sind. Hierzu wurden immer wieder Verbesserungsmöglichkeiten gefunden, zum Großteil auch implementiert und somit verifiziert. Es zeigte sich, dass die Kombination unterschiedlicher Konzepte in einer Anwendung oft problematisch ist. Die vorliegende Arbeit soll die erreichten Resultate interessierten Lesern zugänglich machen und zu einer weiteren Forschungstätigkeit in Richtung eines hybriden Ansatzes aus konstruktiven und numerischen Verfahren anregen.

Als Leser kommen sehr unterschiedliche Personen in Betracht:

- Endanwender
  Sie sind die eigentlichen Nutzer der Software (z. B. CAD Programmen). Entweder nutzt ein Endanwender constraint-basierter Software lediglich solche Modelle, die mit Hilfe von Constraints modelliert wurden, oder er erstellt selbst constraint-basierte Modelle mit den durch die Software bereitgestellten Werkzeugen. Insbesondere im zweiten Fall benötigt der Endanwender ein Grundverständnis, um im Falle eines Fehlschlagens der Berechnungen bzw. einer aus seiner Sicht falschen Lösungswahl die entsprechenden Probleme beheben zu können und sich somit eine positive Haltung gegenüber der Software zu bewahren.
- Anwendungsentwickler
  Sie sind die Entwickler der Software, mit der der Endanwender arbeitet. Das breite Spektrum von Anwendungsbeispielen in Kapitel 5 könnte Entwicklern Anregungen für weitere Einsatzmöglichkeiten constraint-basierter Modellierung und eines Constraint-Solvers bieten. Für die Konzeption und Implementierung müssen die Entwickler die Schwächen und Stärken eines Constraint-Solvers kennen, um sie in der Applikation angemessen zu berücksichtigen. Dies ist insbesondere für die Akzeptanz beim Endnutzer von großer Bedeutung.
- Programmierer eines Constraint-Moduls
  Hiervon gibt es offensichtlich am wenigsten. Für sie wird die vorliegende Arbeit am interessantesten sein, denn sie können sich eine Vielzahl von Anregungen für eine eigene Implementierung holen. Insbesondere die Ausführungen zur Nullstellensuche könnten aber auch von allgemeinem Interesse sein.

Die obige Dreiteilung spiegelt auch die unterschiedlichen Schichten einer constraint-basierten Software wider:

- die constraint-basierten Modelle,
- eine Anwendungssoftware, die die Modelle und die aktuellen Steuerungsinformationen in einer für den Constraint-Solver geeigneten Form an diesen übergibt und
- den Constraint-Solver, der die Berechnungen zur Herstellung bzw. Sicherung der Modellkonsistenz durchführt.

In der vorliegenden Arbeit wird auf alle drei Schichten eingegangen, wobei der Fokus auf den tieferen Schichten liegt und die äußeren eher zur Motivation und Demonstration herangezogen werden.

1.3 Struktur dieser Arbeit

Das Ziel von Kapitel 2 ist es, dem Leser das Grundwissen zu vermitteln, das er zum Verständnis der darauffolgenden Kapitel benötigt. Hierfür werden anhand kleinerer Beispiele Ansätze zum Umgang mit geometrischen Constraints sowie dabei möglicherweise auftretende Probleme erläutert. Verweise auf weiterführende Abschnitte ermöglichen dem interessierten Leser auch die sofortige Vertiefung des jeweiligen Themas.

In Kapitel 3 werden die grundlegenden in Ficucs verwendeten Ansätze detailliert erörtert. Nach einer Diskussion der Entwicklungsziele liegt der Fokus auf den intern verwendeten Konzepten sowie den entsprechenden Algorithmen zu ihrer Umsetzung.

Die Behandlung von Mehrfachlösungen, die ein zentrales Problem bei der Berechnung constraint-basierter Modelle darstellt, wird in Kapitel 4 tiefergehend behandelt. Hierbei erfolgt auch ein Vergleich der in Ficucs verwendeten Ansätze mit anderen in der Literatur beschriebenen Ansätzen.

Die Vielfalt der Einsatzmöglichkeiten von Constraints für das geometrische Modellieren im Allgemeinen und die Umsetzung mit dem vorgestellten Constraint-Solver im Besonderen soll Kapitel 5 zeigen.

Schließlich wird in Kapitel 6 eine Zusammenfassung der vorliegenden Arbeit sowie ein Ausblick auf die zukünftige Forschungs- und Entwicklungstätigkeit gegeben.

Im Anhang A sind eine Reihe von Modelldefinitionen und Plänen für ausgewählte Beispiele aufgeführt. Zudem ermöglichen zwei durchgängige Beispiele ein besseres Verständnis des Zusammenspiels der einzelnen Algorithmen. Anhang B erläutert einige der iterativen numerischen Verfahren, die in der Literatur zur Berechnung constraint-basierter Modelle herangezogen werden.

Kapitel 2

Grundtechniken und zu beachtende Probleme

An dieser Stelle sollen anhand von kleineren Beispielen einige Grundtechniken des Umgangs mit geometrischen Constraints sowie dabei auftretende Probleme erläutert werden. Ziel ist es, dem Leser ein Grundwissen zu vermitteln, das zum Verständnis der folgenden Kapitel benötigt wird. Verweise auf weiterführende Abschnitte ermöglichen dem interessierten Leser jedoch auch die sofortige Vertiefung des jeweiligen Themas.

2.1 Modelldefinition und -berechnung

Beim constraint-basierten Modellieren sind verschiedene Abstraktionsstufen der Modelle zu unterscheiden. In den Applikationen existieren mehr oder weniger stark ausmodellierte Modelle (z. B. CAx-Modelle) mit denen der Nutzer arbeitet. Um für Berechnungen einen Constraint-Solver zu nutzen, werden von diesen Modellen nur die für die Berechnung relevanten Modellbestandteile constraint-basiert beschrieben. Die Notwendigkeit der damit einhergehenden Modellreduktion ergibt sich daraus, dass ein Constraint-Solver typischerweise umso bessere Ergebnisse liefert[1], je kleiner die zu berechnenden Modelle sind.

In diesem Abschnitt wird auf die Übergabe der reduzierten Modelle von einer Applikation an einen Constraint-Solver (im folgenden Text als externe Modellierung bezeichnet) sowie auf die constraint-solver-interne Modellierung eingegangen. In den Beispielen wird zur Vereinfachung nur die Modellierung geometrischer Objekte betrachtet.

2.1.1 Modellierung von Geometrischen Objekten

Extern

Zur Beschreibung von Modellen muss stets auf solche Konstrukte zurückgegriffen werden, die in der Schnittstelle des Constraint-Solver-Moduls angeboten werden. Bei einem klassischen Gleichungslöser sind das z. B. Variablen und Gleichungen. Hierbei geht jedoch die Möglichkeit, spezielles Domänenwissen bei der Lösungsfindung zu nutzen, verloren[2], wodurch die Brauchbarkeit der Lö-

[1] bezüglich der Berechnungsgeschwindigkeit und der Güte der Lösung. Zu große Modelle können u. U. nicht mehr interaktiv bearbeitet werden oder die Lösungen entsprechen nicht den Erwartungen der Nutzer.

[2] Es sei denn, es wird versucht aufgrund der Struktur von Gleichungen und ggf. dem gleichzeitigen Auftreten von bestimmten Gleichungen auf die ursprüngliche geometrische Bedeutung zu schließen und die Lösungssuche entsprechend zu steuern. Ein solcher Ansatz ist jedoch nicht bekannt und ein Erfolg fraglich.

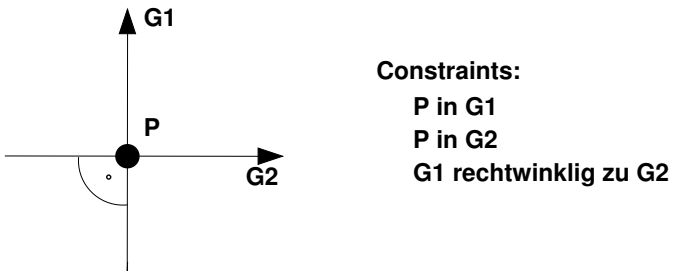


Abbildung 2.1: Modellierung eines *2D*-Koordinatensystems

sungen oft nicht gegeben ist. In Ficucs wird deshalb mit Standardgeometrien gearbeitet. Hierbei handelt es sich um:

- *2D*-Punkte,
- *2D*-Geraden,
- *2D*-Kreise,
- *3D*-Punkte,
- *3D*-Ebenen,
- *3D*-Zylinder.

Außerdem besteht die Möglichkeit Parameter zu definieren, wobei auch hier entsprechend dem Einsatz in der Domäne „Geometrie“ Abstands- und Winkelparameter unterschiedlich behandelt werden können.

Die in Ficucs verwendeten Standardgeometrien werden im folgenden Text auch als *einfache geometrische Objekte* oder *geometrische Objekte* bezeichnet. Komplexere geometrische Objekte sind durch diese einfachen geometrischen Objekte und entsprechende Constraints sowie ggf. Parameter zu beschreiben. Die Obermenge von geometrischen Objekten und Parametern wird in der vorliegenden Arbeit auch allgemein als *Objekte* bezeichnet.

Beispiel 2.1: Modellierung eines lokalen Koordinatensystems in 2D
Ein lokales Koordinatensystem ist ein komplexeres geometrisches Objekt. Es kann in *2D* wie folgt modelliert werden:

- einen Punkt für den Ursprung,
- zwei gerichtete Geraden für die Achsen,
- zwei „Punkt auf Gerade“-Constraints, um sicher zu stellen, dass die Achsen durch den Ursprung verlaufen sowie
- einen Orthogonalitäts-Constraint, um sicher zu stellen, dass die Achsen senkrecht aufeinander stehen,

siehe Abbildung 2.1. □

Freiformgeometrien können durch Ficucs nicht direkt bearbeitet werden. Das Bearbeiten von Kontrollpunkten ist jedoch möglich, wenn sie als *2D*- oder *3D*-Punkte hinterlegt werden. In Abbildung 5.2 ist ein Beispiel für die Modellierung einer Bezierkurve gezeigt.

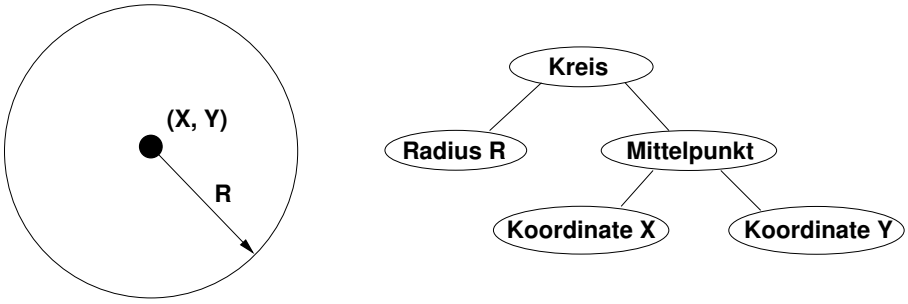


Abbildung 2.2: Interne Modellierung eines *2D*-Kreises

Intern

Es hat sich gezeigt, dass auch bei der constraint-solver-internen Modellierung der oben beschriebenen einfachen geometrischen Objekte die Verwendung von Subobjekten sinnvoll sein kann. Ein Kreis in *2D* wird z. B. durch einen *2D*-Punkt und einen Abstandsparameter für den Radius modelliert, der Mittelpunkt wiederum als x- und y-Koordinate (siehe Abbildung 2.2).

Vorteile der Nutzung von Subobjekten sind:

- Die separate Fixierung ist möglich. Werte fixierter Objekte werden vom Constraint-Solver nur gelesen.
- Constraints können Subobjekte direkt referenzieren. Das ist sinnvoll, da ansonsten spezielle Constraints und damit ggf. auch spezielle Konstruktoren nötig wären. Zum Beispiel kann der Mittelpunkt eines Kreises in einem Abstands-Constraint zwischen zwei Punkten referenziert werden, ein spezieller Constraint wie „Abstand zwischen dem Mittelpunkt eines Kreises und einem Punkt“ sind somit nicht nötig.
- Die Konsistenz von Modellen kann besser gewährleistet werden (siehe Beispiel 2.10).
- In einigen Fällen lassen sich Vereinfachungen im Modell vornehmen und so die Berechnungen effizienter durchführen.

  Beispiel 2.2: 3D-Richtungsvektoren als Subobjekte von Ebenen
  Ebenen seien intern durch ein Subobjekt für die *3D*-Richtung und den Abstand zum Ursprung des Koordinatensystems modelliert. Zur Definition einer Menge paralleler Ebenen können dann Gleichheits-Constraints zwischen den Subobjekten für die *3D*-Richtungen genutzt werden. Dies führt zu einer Vereinfachung des internen Modells, denn bei Gleichheit von Objekten erfolgt die Ersetzung der Menge gleicher Objekte (hier Richtungen) durch ein einziges Referenzobjekt (siehe Abschnitt 3.2.2). Ist nun beispielsweise die Normale einer Richtung bekannt, so steht diese Information automatisch in allen Ebenen zur Verfügung. □

Allerdings sind mit der Einführung von Subobjekten auch Konsequenzen, zum Teil sogar unerwünschte, verbunden:

- Es werden zusätzliche Konstruktoren benötigt, die ein Objekt[3] auf Basis bekannter (z. B. fixierter oder bereits berechneter) Subobjekte und ggf. noch zu erfüllender Constraints berechnen.
- Die Anzahl der (intern) zu bearbeitenden Objekte erhöht sich[4].

[3] bzw. seine noch unbekannten Subobjekte

[4] Zwar nur linear, aber ungenutzte Subobjekte sind schlecht für die Effizienz. Zum Beispiel wird in einigen Applikationen nicht auf die Subobjekte für die Koordinaten von *2D*-Punkten zugegriffen. Hier wäre es sinnvoll auf die Nutzung von Subobjekten für die Modellierung der Punkte zu verzichten.

- Es werden spezielle Constraints „ist Subobjekt von“ benötigt.

Oft sind verschiedene Varianten der Aufteilung in Subobjekte möglich, z. B. könnten Punkte auch in Polarkoordinaten hinterlegt sein. Allgemein ist die Entscheidung ob und wie aufgeteilt werden soll ein Kompromiss bezüglich:

- der wichtigsten Einsatzdomänen,
- der Anzahl der dort zu erwartenden direkten Zugriffe auf die Subobjekte einer bestimmten Aufteilung[5] und
- dem Aufwand der internen Modellierung und Verarbeitung solcher Subobjekte.

Zum Beispiel wurde zur Vermeidung eines erhöhten internen Programmieraufwandes in Ficucs auf die Unterteilung von *3D*-Punkten ganz verzichtet. Eine separate Fixierung der einzelnen Koordinaten ist nun nur über entsprechende Constraints, z. B. den fixen Abstand zu einer Hauptebene, möglich.

In einer aufwändigeren Implementierung könnten die Aufteilungen bedarfsabhängig vorgenommen werden.

Resultat

Das Ergebnis der Modellierung ist ein interner Graph aus Objekten und Constraints. Da Objekte per Definition keine Objekte referenzieren dürfen und Constraints keine Constraints, ist der Graph bipartit[6]. Dieser Graph wird auch als Constraint-Netz bezeichnet.

Außer dem Graphen stehen typischerweise noch diverse Informationen zur Verfügung, z. B.:

- die aktuelle Lage der Objekte,
- Namen von Objekten und Constraints (z. B. für die Fehlersuche),
- die Position innerhalb einer Featurehierarchie (siehe Beispiel 5.5),
- Priorisierungsinformation (siehe Abschnitt 2.2.5),
- Fixierungsinformation (siehe Abschnitt 2.2.4).

2.1.2 Ablauf von Modelldefinition und -berechnung

Der grundsätzliche Ablauf soll an einem kleinen Beispiel erläutert werden.

Beispiel 2.3: Ablauf einer Modelldefinition und -berechnung

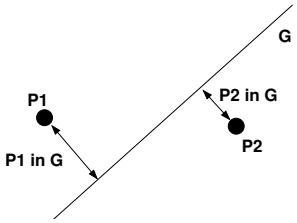
1. Deklaration von Objekten und Constraints[7]
   z. B.: zwei Punkte P_1 und P_2, eine Gerade G und zwei „Punkt auf Gerade“-Constraints werden

[5] In einer Anwendung, die *2D*-Punkte als Ecken von rechteckigen und achsenparallelen Fenstern nutzt, ist die Aufteilung in x- und y-Koordinate sinnvoll, denn es wird hier viele Constraints geben, die die Koordinaten direkt betreffen. Bei der Modellierung von Mechanismen hingegen (siehe z. B. Abschnitt 5.2.3) werden Koordinaten nur sehr selten oder nie referenziert.

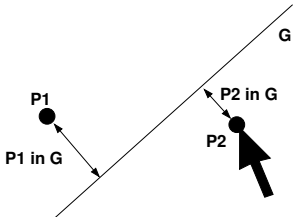
[6] zweigeteilt

[7] evtl. auch Zusatzinformation. Bei größeren Modellen ist eine geeignete Modellierung oft nicht trivial. In einigen Anwendungen wird auf vordefinierte Modelle zugegriffen. Die automatische Erstellung ist möglich.

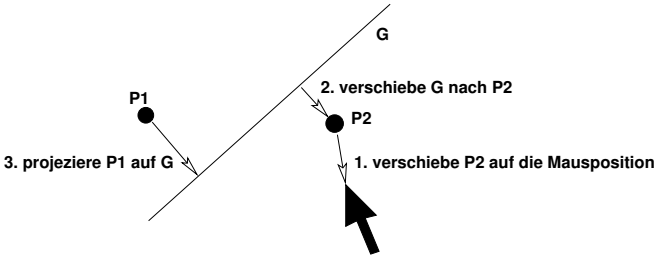
definiert. Die Abbildung zeigt, dass die Constraints initial nicht unbedingt erfüllt sein müssen.



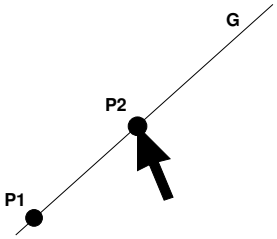
2. Festlegung einer Aktion
   z. B.: Auswahl von Punkt P_2 zur Verschiebung mit Hilfe der Maus.



3. Erstellung eines Konstruktionsplanes für das Modell entsprechend der Aktion[8]
   z. B. bestehend aus den folgenden drei Einträgen:



4. Ausführung des Konstruktionsplanes
   z. B. mit folgendem Resultat:



5. Evtl. weitere Berechnungszyklen mit dem gleichen Plan
   z. B. durch anhaltende Verschiebung des Mauszeigers. Die aktuellen Positionen von P_2, G und P_1 werden jeweils nachgeführt.
6. Verwerfen des Planes, wenn die Aktion beendet ist[9].

□

[8] Wir nehmen an, dass aufgrund einer Priorisierung (siehe Abschnitt 2.2.5) die Gerade nicht rotiert, sondern nur in den Punkt verschoben wird. Die Priorisierung ist eine Form der Zusatzinformation, die während der Freiheitsgradanalyse (Abschnitt 2.2.2) ausgewertet wird.

[9] Prinzipiell ist auch ein Speichern des Planes möglich. Vor einer Planerstellung kann dann der Versuch einer Planerkennung erfolgen, jedoch kann die Trefferquote durch unterschiedliche Fixierungen, Priorisierungen, usw. gering sein. Bei sich oft wiederholenden Interaktionen an unveränderten Modellen kann die Planspeicherung/Planerkennung aber durchaus sinnvoll sein, siehe Abschnitt 3.4.3.

2.2 Freiheitsgrade

Ein wichtiger Aspekt der Handhabung von constraint-basierten Modellen ist die Analyse der im Modell vorhandenen Freiheitsgrade. In diesem Abschnitt werden die hierfür wesentlichen Grundlagen vermittelt.

2.2.1 Abstraktion von Objekten und Constraints

Es ist üblich, den aus Objekten und Constraints gebildeten Graphen unabhängig von konkreten Werten und teilweise sogar unabhängig von den Objekt- bzw. Constraint-Typen zu analysieren. Die einzige typabhängige Information, die bei der Analyse verwendet wird, ist für Objekte ihr *Freiheitsgrad* und für Constraints ihre *Valenz*. Die Bedeutung definiert sich wie folgt:

- Der Freiheitsgrad[10] eines Objektes entspricht der Anzahl der Parameter, die für eine redundanzfreie Speicherung notwendig ist.
- Die Valenz eines Constraints entspricht der Summe der Freiheitsgrade, die (durch die Definition des Constraints) den von ihm referenzierten Objekten entzogen werden.

Beispiel 2.4: Freiheitsgrade und Valenzen an einem einfachen 2D-Beispiel
Das einführende Beispiel 1.1 besteht aus einem Punkt (*2FG*), einer Gerade (*2FG*) und einem Abstandsparameter (*1FG*) als Objektmenge. Ohne Definition von Constraints besitzt das Modell somit *5FG*. Nach dem Hinzufügen des Abstands-Constraints sind im Modell nur noch *4FG* vorhanden[11], dementsprechend hat ein Abstands-Constraint zwischen Punkt und Gerade eine Valenz von eins. Würde die Modellierung mit Gleichungen erfolgen, so wäre genau eine Gleichung für den Constraint zu definieren[12]. □

In den Tabellen 2.1 und 2.2 sind die Freiheitsgrade und Valenzen von verschiedenen Objekt- und Constraint-Typen aufgeführt. Es gibt jedoch auch spezielle Constraints, die keine fixe Valenz haben. Das trifft z. B. bei den Constraint-Typen „ist Subobjekt von“[13] und „ist fix“ (siehe Abschnitt 2.2.4) zu. Zudem kann die Valenz eines Constraints von den Werten der referenzierten Objekte abhängen[14].

2.2.2 Freiheitsgradanalyse

Bei der Freiheitsgradanalyse wird das Wechselspiel der Freiheitsgrade und Valenzen untersucht. Das kann zum einen pauschal durch Aufsummierung und Vergleich erfolgen, zum anderen kann die Untersuchung aber auch die Struktur des Graphen berücksichtigen.

Beispiel 2.5: Aufsummierung und Vergleich von Freiheitsgraden und Valenzen
Für das in Abschnitt 2.1.2 gezeigte Modell ergibt sich folgende Rechnung:

Objekt bzw. Constraint	Freiheitsgrad	Valenz
$P1$	2	
$P2$	2	
G	2	
$P1$ in G		1
$P2$ in G		1
Summen:	6	2

[10] Die Verwendung der Mehrzahl („Freiheitsgrade eines Objektes“ statt „Freiheitsgrad eines Objektes“) drückt die mögliche Unterteilung des Objektes in Subobjekte aus (siehe Abbildung 2.2), denen jeweils ein Freiheitsgrad oder mehrere Freiheitsgrade zugeordnet werden kann.

[11] Zum Beispiel könnten Punkt und Gerade mit je zwei Freiheitsgraden positioniert werden, der Abstandsparameter ist dann jedoch abhängig von dieser Positionierung.

[12] Unter der Annahme, dass die Variablen skalar sind.

[13] Seine Valenz ist gleich dem Freiheitsgrad des Subobjektes. Zum Beispiel hat eine Ebene drei *FG*, ihr Subobjekt für die Richtung hat zwei *FG* und dementsprechend der Constraint, der beide verbindet, eine Valenz von zwei.

[14] Siehe Beispiel 2.11.

Tabelle 2.1: Freiheitsgrad für verschiedene Objekttypen

Objekttyp	Freiheitsgrad	Anmerkung
Parameter	1	ein Zahlenwert
2D-Punkt	2	z. B. x- und y-Koordinaten
2D-Richtung	1	z. B. Winkel mit x-Achse
2D-(Richtungs-)Vektor	2	Werte für dx und dy
2D-Gerade	2	z. B. *2D*-Richtung und Abstand zum Ursprung
2D-Kreis	3	Mittelpunkt und Radius $(2 + 1)$
2D-Kreis mit fixem Radius	2	nur Mittelpunkt
3D-Punkt	3	z. B. x-, y- und z-Koordinaten
3D-Richtung	2	z. B. Winkel zur x- und y-Achse
3D-(Richtungs-)Vektor	3	Werte für dx, dy und dz
3D-Gerade	4	z. B. ihr *2D*- Durchstoßpunkt in der xy-Ebene und ihre *3D*-Richtung
Ebene	3	z. B. *3D*-Richtung und Abstand zum Ursprung
Zylinder (unbegrenzt)	5	z. B. Mittelachse als *3D*-Gerade und Radius als Parameter

Tabelle 2.2: Valenz von verschiedenen Constraint-Typen

Constraint-Typ	Valenz	Anmerkung
Gleichung	1	z. B. lineare Gleichungen
2D: Punkt in Gerade	1	beachte Unterschied zu *3D*
3D: Punkt in Gerade	2	es bleiben wahlweise ein *FG* für den Punkt oder zwei *FG* für die Gerade (Richtung)
2D: Punkt gleich Punkt	2	beachte den Unterschied zu *3D*
3D: Punkt gleich Punkt	3	beachte den Unterschied zu *2D*
Abstand Punkt zu Punkt	1	gilt in *2D* und in *3D*
Abstand Punkt zu Gerade	1	gilt in *2D* und in *3D*
Abstand Punkt zu Kreis	1	gilt in *2D* und in *3D*
2D: Punkt in Kreis	1	beachte den Unterschied zu *3D*
3D: Punkt in Kreis	2	beachte den Unterschied zu *2D*
3D: Punkt in Ebene	1	dann z. B. Punkt wie ein *2D*-Punkt in der euklidischen Ebene
Winkel zwischen Geraden	1	gilt in *2D* und in *3D*
2D: Gerade parallel Gerade	1	Gleichheit der *2D*-Richtungen
3D: Gerade parallel Gerade	2	Gleichheit der *3D*-Richtungen
2D: Parallelität und Abstand zweier Geraden	2	z. B. Gleichheit der Richtungen und Gleichung für Abstände zum Ursprung
3D: Parallelität und Abstand zweier Geraden	3	Richtungen sind gleich, aber eine Gerade könnte noch um die andere „kreisen“
3D: Parallelität und Abstand zweier Ebenen	3	z. B. Gleichheit der Richtungen und Gleichung für Abstände zum Ursprung
2D: Kreis tangential Gerade	1	z. B. Gleichung für „Abstand vom Kreismittelpunkt zur Geraden ist Kreisradius“
2D: Kreis tangential Kreis	1	z. B. Gleichung für Abstand der Mittelpunkte unter Berücksichtigung der Radien
2D: Konzentrische Kreise	2	Gleichheit der Mittelpunkte

Demnach hat das Gesamtmodell einen Freiheitsgrad von $6 - 4 = 2$. Im Falle des gezeigten Verschiebens von $P2$ war $P2$ auf den Mauszeiger gesetzt worden, was sich auch als Constraint der Valenz zwei (Punkt gleich Punkt) deuten lässt. Dementsprechend hatte das Modell beim Ziehen nur einen Freiheitsgrad von zwei, der z. B. durch die Richtung von G und den Abstand von $P1$ und $P2$ beschrieben werden könnte. □

Die Analyse kann folgende Zustände des Modells feststellen:

1. Überbestimmtheit
   Die Gesamtvalenz im Graphen ist größer als der Gesamtfreiheitsgrad. Es kann konsistente (alle Constraints sind erfüllbar) und inkonsistente Überbestimmtheit (es gibt mindestens einen nicht erfüllbaren Constraint) vorliegen.

   Beispiel 2.6: Überbestimmtheit bei zwei gleichen Punkten in 2D
   Zwischen zwei fixierten Punkten ($\sum FG = 0$) wird ein Gleichheits-Constraint ($Val = 2$) definiert. Sollten die beiden Punkte an derselben Position fixiert worden sein ist die Gleichheit gegeben und das Modell trotz Überbestimmtheit konsistent. Sind die Punkte jedoch an unterschiedlichen Positionen fixiert worden, so liegt inkonsistente Überbestimmtheit vor, da der Gleichheits-Constraint nicht erfüllbar ist. □

2. Unterbestimmtheit
   Die Gesamtvalenz im Graphen ist kleiner als der Gesamtfreiheitsgrad.

   Beispiel 2.7: Unterbestimmtheit bei zwei gleichen Punkten in 2D
   Zwischen zwei nicht fixierten Punkten ($\sum FG = 2{+}2 = 4$) wird ein Gleichheits-Constraint ($Val = 2$) definiert. Es bleiben für das Modell zwei Freiheitsgrade, z. B. die x- und y-Koordinate der gemeinsamen Position, die nicht bestimmt ist. Deshalb wird das Modell in diesem Zustand als unterbestimmt bezeichnet. □

3. Vollbestimmtheit
   Die Gesamtvalenz im Graphen ist gleich dem Gesamtfreiheitsgrad.

   Beispiel 2.8: Vollbestimmtheit bei zwei gleichen Punkten in 2D
   Zwischen einem fixierten und einem unfixierten Punkt ($\sum FG = 2 + 0 = 2$) wird ein Gleichheits-Constraint ($Val = 2$) definiert. Das Modell hat keinen Freiheitsgrad, es ist vollbestimmt. □

Das oben gezeigte Aufsummieren und Vergleichen berücksichtigt die Struktur des Graphen jedoch nicht. Dadurch kann es zum Beispiel dazu kommen, dass ein Modell als vollbestimmt angesehen wird, obwohl Teile des Modells unterbestimmt und andere überbestimmt sind. Graphbasierte Analysen können derartige Fälle erkennen. Sie suchen nach einer Zuordnung zwischen Freiheitsgraden und Valenzen[15].

Beispiel 2.9: Vorteil graphbasierter Freiheitsgradanalyse beim Erkennen von Über- und Unterbestimmtheit
Abbildung 2.3 zeigt ein Modell mit vier Punkten ($\sum FG = 0{+}0{+}2{+}2 = 4$) und drei Constraints ($\sum Val = 2 + 1 + 1 = 4$)[16]. Der pauschale Vergleich würde ein vollbestimmtes Modell vermuten. Die graphbasierte Analyse in Abbildung 2.3(c) zeigt jedoch, dass die Valenz des Gleichheits-Constraints ($Val = 2$) nicht zugeordnet werden kann, denn die referenzierten Objekte haben keine Freiheitsgrade. Es liegt also eine lokale Überbestimmtheit vor. Zugleich haben P_3 und P_4 noch je einen Freiheitsgrad (zwei Freiheitsgrade minus eine zugeordnete Valenz ergibt einen Freiheitsgrad)[17] und sind deshalb unterbestimmt. □

Beispiel 2.10: Vorteil der Modellierung von Subobjekten
Eine Ebene hat drei Freiheitsgrade. Zwei davon sind an ihr Richtungs-Subobjekt gebunden. Angenommen es seien zwischen einer Ebene E und den Achsen des Koordinatensystems drei Winkel-Constraints modelliert worden und die entsprechenden Parameter sowie das Koordinatensystem seien fixiert. Ohne

[15] Sie werden deshalb auch als Matching-Algorithmen bezeichnet, z. B. engl: *biparted graph matching* (siehe Abschnitt 3.3.1), da die Graphen bipartit sind (siehe 2.1.1).
[16] Die fixierten Abstände (FG=0) sind für die Betrachtung irrelevant.
[17] Sie könnten um P1 bzw. P2 rotieren.

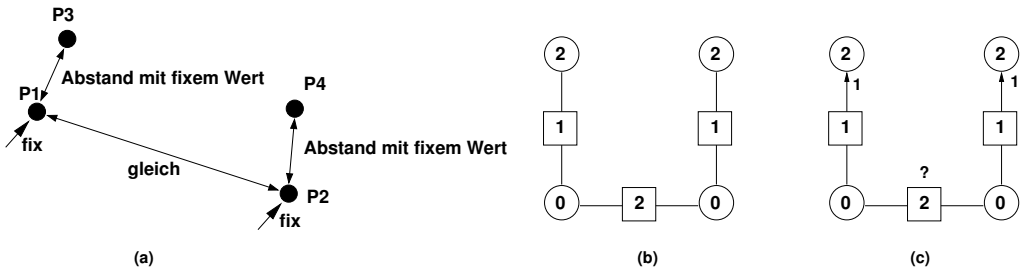


Abbildung 2.3: Modell mit lokaler Über- und Unterbestimmtheit und die entsprechende Zuordnung von Freiheitsgraden und Valenzen: (a) Modell, (b) Graph für die Freiheitsgradanalyse, (c) Zuordnung der Valenzen zu den Freiheitsgraden

die Nutzung von Subobjekten würde die Freiheitsgradanalyse fälschlicherweise feststellen, dass die Ebene vollbestimmt ist (drei Freiheitsgrade von E und drei Valenzen). Da Winkel-Constraints jedoch nur auf die Richtung der Ebene wirken, referenzieren sie im internen Constraint-Netz nicht die Ebene sondern direkt die Richtung. Somit wird die Überbestimmtheit (zwei Freiheitsgrade der Richtung gegenüber drei Valenzen der Winkel-Constraints) deutlich. □

Obwohl die graphbasierten Algorithmen bessere Resultate liefern als das pauschale Aufsummieren und Vergleichen, bleiben dennoch Unsicherheiten bezüglich des Zustandes des Modells. Zwei Möglichkeiten seien hier angeführt:

1. Die Valenzen von Constraints können für spezielle Zahlenwerte der referenzierten Objekte anders sein.

   Beispiel 2.11: Abhängigkeit der Valenz von Zahlenwerten referenzierter Objekte
   Ein Beispiel hierfür ist der Abstands-Constraint zwischen zwei Punkten. Wird bei den Berechnungen ein Abstands-Parameter mit dem Wert gleich null[18] verwendet, so ist die Valenz des Constraints nicht mehr eins, sondern zwei in *2D* und drei in *3D*, denn es handelt es sich nun um Gleichheits-Constraints. □

2. Die Art der Constraints (und ggf. auch die referenzierten Werte) können zu einer Redundanz führen. Diese ist vergleichbar mit der linearen Abhängigkeit von Gleichungen in einem linearen Gleichungssystem.

   Beispiel 2.12: Problematik der Abstraktion auf Freiheitsgradniveau
   Wird zwischen zwei Punkten in *2D* ein fixer Abstand ($Val = 1$) und ein fixer Anstieg ($Val = 1$) definiert, so ist bei Fixierung des einen Punktes (nun $FG = 0$) auch der andere Punkt bekannt, und das Modell ist vollbestimmt. Es ist jedoch auch denkbar, dass statt des Anstiegs-Constraints ein zweiter fixer Abstand zwischen den beiden Punkten modelliert wurde. Die Struktur des Abhängigkeitsgraphen und die betrachteten Freiheitsgrade und Valenzen ändern sich dadurch nicht. Es liegt jedoch offensichtlich eine konsistente (die Abstandswerte sind gleich) oder inkonsistente (die Abstandswerte sind verschieden) lokale Überbestimmtheit vor. □

Die beschriebenen (teilweise auch als degeneriert bezeichneten) Fälle werden oft in Veröffentlichungen und Implementierungen ignoriert[19]. In Abschnitt 3.2 werden Ansätze zum Umgang mit degenerierten Fällen gezeigt.

---

[18] Der Wert null kann konstant sein. Er wäre damit bei der Eingabe erkennbar und der Abstands-Constraint könnte wie ein Gleichheits-Constraint behandelt werden. Es ist jedoch möglich, dass sich der Wert null erst während einer Planausführung ergibt. Derartige Fälle verkomplizieren das Zusammenspiel von Freiheitsgradanalyse, Planerstellung und Planausführung, denn normalerweise wird die Freiheitsgradanalyse als unabhängig von Zahlenwerten betrachtet.

[19] ihre Berücksichtigung würde manchen Ansatz verkomplizieren

2.2.3 Starrheit von Objekten

Die Freiheitsgradanalyse kann ergeben, dass sich eine Menge von Objekten durch Constraints, die zwischen ihnen wirken, als starrer Körper behandeln lässt. Starr heißt hier, dass die relative Lage der einzelnen Objekte zueinander fix ist. Die Anzahl der restlichen Freiheitsgrade, die die Einbettung des starren Körpers im Raum repräsentieren, hängt wie folgt von der Dimension des Raumes ab:

Dimension	Freiheitsgrade	Bemerkung
1	1	Fixe Abstände zwischen Zahlenwerten, z. B. Winkeln. Ein (Referenz-)Wert ist frei wählbar.
2	3	z. B. Menge von Punkten, Geraden und Kreisen in der euklidischen Ebene. Es ist ein Referenzpunkt (zwei FG) und eine Referenzrichtung (ein FG) wählbar. Analoges gilt auch für eine Menge von *3D*-Richtungen.
3	6	z. B. starre Körper in *3D*, in denen Punkte, Geraden und Ebenen eingebettet sind.

Wie oben gezeigt läuft die Freiheitsgradanalyse jedoch auf einem sehr abstrakten Niveau ab. Es lassen sich deshalb leicht Modelle konstruieren, bei denen die Starrheit nicht so einfach feststellbar ist. Hierfür seien folgende Beispiele gegeben:

- **Beispiel 2.13: Punkte auf einer Geraden in 3D**
  Mehrere Punkte, die mit fixen Abständen zueinander auf einer Gerade in *3D* angeordnet sind, bilden zusammen mit der Gerade einen starren Körper. Dennoch besitzt dieser Körper nur fünf Freiheitsgrade. Die Rotation um die Gerade führt zu keiner Lageänderung des Körpers. □

- **Beispiel 2.14: Zwei Punkte mit einem variablen Abstand in 3D**
  Zwei Punkte in *3D* und ein nicht fixierter Abstand, zwischen den ein Abstands-Constraint definiert wurde, haben $2 * 3 + 1 - 1 = 6$ Freiheitsgrade[20]. Jedoch ist der entsprechende Körper keineswegs starr. □

- **Beispiel 2.15: Constraints für die absolute Lage**
  Constraints, die sich auf die absolute Lage im 1, 2 oder 3-dimensionalen Raum beziehen, dürfen in die Freiheitsgradrechnung für die Starrheit nicht eingehen. Dazu gehören Fixierungen aber auch z. B. ein „Richtung von einem *2D*-Punkt zu einem anderen"-Constraint[21], der sich üblicherweise auf die x-Achse bezieht. □

2.2.4 Fixieren von Freiheitsgraden

Für jedes definierte Objekt eines Modells (einschließlich der zugreifbaren Subobjekte) ist die Fixierung und damit die Wegnahme seiner Freiheitsgrade möglich. Damit werden sie in Analysen und der Planerstellung als durch den Solver unveränderlich angenommen. Ihre Werte werden im Laufe der Planausführung nur gelesen und nie gesetzt. Das kann unter Umständen auch zu einer inkonsistenten Überbestimmtheit führen. Die Fixierung wird auch als lokaler Constraint bezeichnet, dessen Valenz stets gleich den Freiheitsgraden des zugeordneten Objektes ist.

---

[20] Ebenso wie die Menge von 2 Punkten ohne jegliche Constraints.
[21] engl.: *slope*

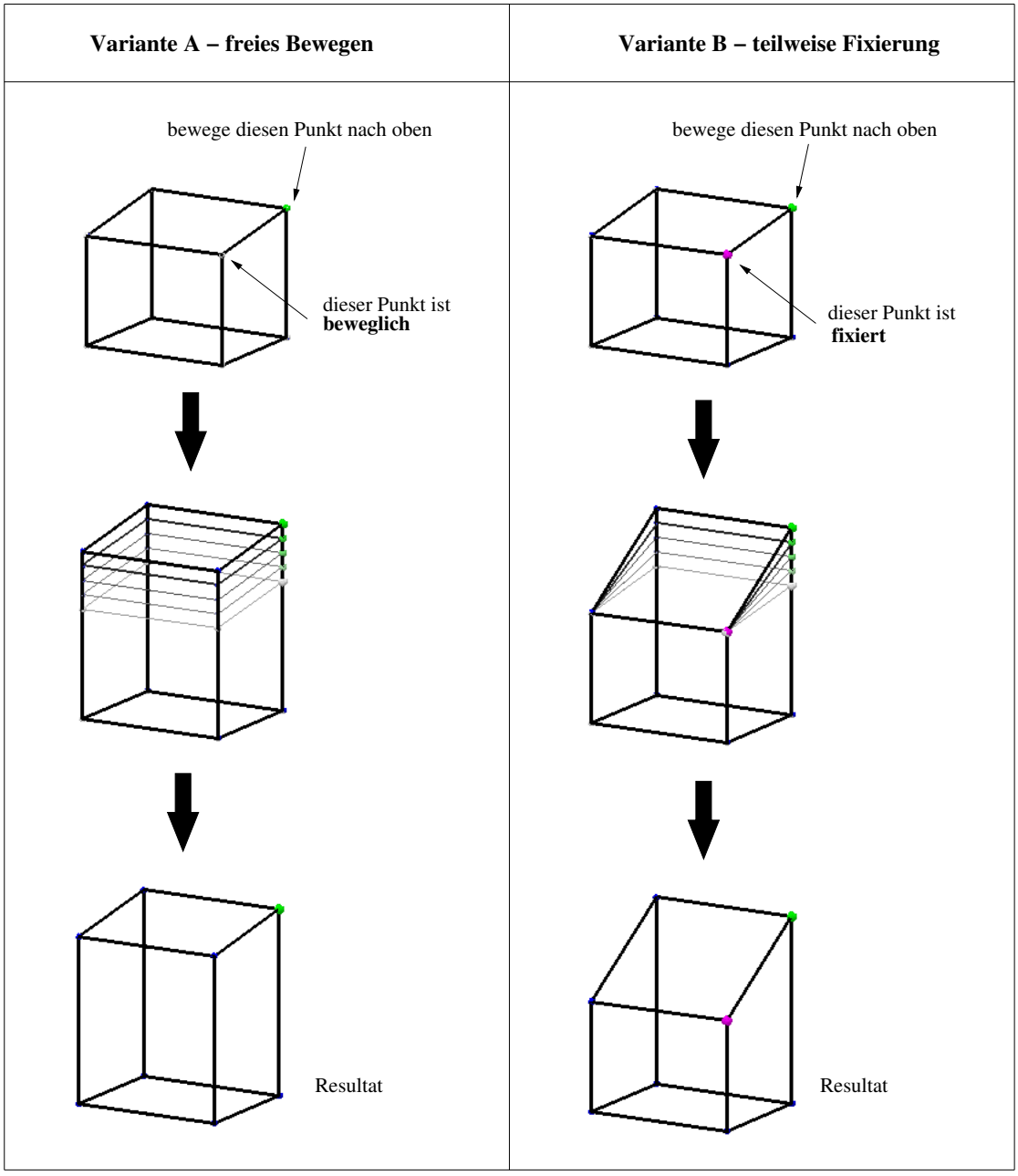


Abbildung 2.4: Kontrolle der berechneten Änderungen durch teilweises Fixieren

Tabelle 2.3: Mögliche Abbildung von Objekttypen auf Freiheitsgradtypen

Objekttyp	Freiheitsgradtyp	Kürzel
Punkt (*2D & 3D*)	Position	p
Richtung (*2D & 3D*)	Rotation	r
Parameter, z. B. Vermaßungen	Dimension	d
Gerade(*2D & 3D*)	gemischt	g
Kreis (*2D & 3D*)	gemischt	g
Ebene	gemischt	g

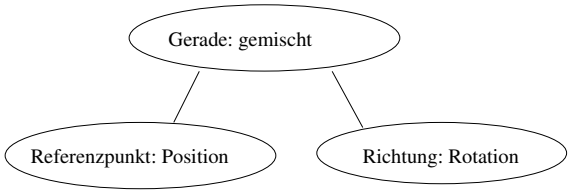


Abbildung 2.5: Die Gerade als Beispiel für ein Objekt, das aus Subobjekten mit unterschiedlichen Freiheitsgradtypen modelliert wurde

2.2.5 Typisierung und Priorisierung von Freiheitsgraden

Das in Abbildung 2.4 gezeigte Verhalten, dass ohne eine Fixierung die Ausrichtung von Polygonen möglichst nicht geändert wird, wurde durch eine Typisierung der Freiheitsgrade und eine auf den Typdefinitionen beruhende Priorisierung der Freiheitsgrade erreicht.

Default-Typen

Vom System können Freiheitsgradtypen entsprechend den Typen der zugehörigen Objekte vergeben werden (siehe z. B. auch [Hsu96]). Einige Beispiele für eine derartige Zuordnung sind in Tabelle 2.3 gegeben.

Wie der Tabelle 2.3 zu entnehmen ist, werden den Freiheitsgraden komplexerer Objekte keine Typen zugeordnet. So kann zum Beispiel für eine Gerade, die sowohl Rotations- als auch Positionsfreiheitsgrade besitzt, nur ein „gemischt“ vermerkt werden.

Priorisierung

Auf Basis der für die einzelnen Objekte definierten Typen kann nun eine Reihenfolge festgelegt werden, zum Beispiel „*prgd*“[22] für das Modell in Abbildung 2.4. Da hier ein unterbestimmter Fall vorliegt, lassen sich einige Freiheitsgrade so nutzen, dass die entsprechenden Objektwerte konstant bleiben. Im Beispiel treten keine Vermaßungen auf, dementsprechend findet das in der Reihenfolge auftretende „*d*“ keine Beachtung. „*p*“ steht für die Referenzpunkte der Ebenen (die den Offset der Ebene repräsentieren und damit einen Positionsfreiheitsgrad widerspiegeln) und für die Eckpunkte. Die Reihenfolge „*prgd*“ bewirkt nun, dass die Normalen der Ebenen mit ihren Rotationsfreiheitsgraden („*r*“) höher priorisiert sind als die Eckpunkte und folglich möglichst nicht verändert werden.

[22] beachte die Kürzel aus Tabelle 2.3

Verfeinerung durch externe Festlegungen

Für einige Einsatzfälle reichen die auf Basis der Objekttypen vom Constraint-Solver festgelegten Typen der Freiheitsgrade nicht aus. Hier ist es zweckmäßig durch externe Festlegungen entsprechend verfeinerte Typen anzugeben. Abschnitt 5.2.3 zeigt ab Seite 140, wie mit *kontextsensitiver* Priorisierung sehr effizient unterschiedlichste Interaktionen unterstützt werden können.

2.3 Werte- vs. existenzbestimmende Constraints

Bei den in dieser Arbeit behandelten Constraints handelt es sich stets um solche, die sich auf die Werte von Objekten beziehen, nie aber auf deren Existenz. Sie können somit als wertebestimmend bezeichnet werden. Um diese wertebestimmenden Constraints zu erfüllen, werden nur Werte geändert jedoch nie Objekte erschaffen oder gelöscht.

Derartige Funktionalität ist zwar durchaus sinnvoll, muss aber außerhalb von Ficucs implementiert werden. Das in Abschnitt 5.3.1 beschriebene Modelliersystem kann die Existenz der Objekte über sogenannte *programmierbare Features* steuern (siehe [BBDO00] oder [BBDO02]). In Ficucs selbst lassen sich existenzbestimmende Constraints für Objekte durch den Einsatz von sogenannten *Conditional Constraints*[23] nur nachempfinden. Die Idee besteht darin, dass alle wertebestimmenden Constraints mit der Exitenzbedingung für ein bestimmtes Objekt versehen werden, statt herkömmlicher Constraints liegen nun Conditional Constraints vor. Wenn die Existenzbedingung für das Objekt nicht erfüllt ist, würden alle diese Constraints deaktiviert werden und das Objekt hätte keine Verbindung mehr zum Constraint-Netz. Somit hätte das Objekt auch keinen Einfluss mehr auf die Berechnungen, was aus Sicht des Constraint-Solvers[24] gleichbedeutend mit der Nichtexistenz des Objektes wäre. Nachteilig ist an diesem Ansatz jedoch, dass schon von vornherein eine maximale Objektanzahl im Modell definiert sein müsste, was beim Einsatz programmierbarer Features nicht der Fall ist.

Im CAD-Bereich gibt es noch eine Reihe weiterer Veröffentlichungen, die Programmsysteme beschreiben, in denen die Existenz von Objekten gesteuert werden kann. So wird z. B. in [Rol91] und [Rol95] durch Roller gezeigt, wie sich die Anzahl von Bohrungen eines Werkstücks mittels existenzbestimmender Constraints steuern lässt[25]. Auch Rosendahl und Berling stellen in [RB98] ein Segmentkonzept vor, in dem Objekte mittels Alternativen und Iterationen gezielt erzeugt werden können.

Auch wenn es eine Reihe von Veröffentlichungen zu existenzbestimmenden Constraints gibt, eine ausführliche Beschreibung von Algorithmen, insbesondere solcher für ein effektives Zusammenspiel von werte- und strukturbestimmenden Constraints, ist nicht bekannt. Selbst die eigenen Arbeiten trennen stets die beiden Constraint-Arten.

2.4 Repräsentierung von Parametern

Eine grundsätzliche Entscheidung beim Design eines Constraint-Moduls liegt in der Beantwortung der Frage: „Sollen Parameter ein fester[26] Bestandteil der Constraints sein?“. Systeme, wie sie in

[23] Sie können auch als dynamische Constraints bezeichnet werden, deren Existenz von bestimmten Bedingungen abhängt. Für eine nähere Erörterung siehe [BBD99], hier wird u. a. am Beispiel eines Malteser-Kreuz-Getriebes gezeigt, wie sich mit derartigen dynamischen Constraints Schaltgetriebe modellieren lassen.

[24] Die Applikation müsste z. B. beachten, dass das Objekt nicht visualisiert wird.

[25] Ein Constraint zur Bestimmung des Radius der Bohrungen wäre hingegen wieder ein wertebestimmender Constraint.

[26] Fest heißt hier zum einen, dass die Werte der Parameter nicht innerhalb des Constraint-Moduls berechnet werden können (ggf. jedoch durch den Nutzer änderbar sind) und zum andern, dass die Parameter stets einem bestimmten Constraint zugeordnet sind.

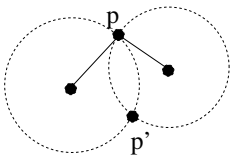


Abbildung 2.6: Zwei Lösungen für die Konstruktion eines Punktes

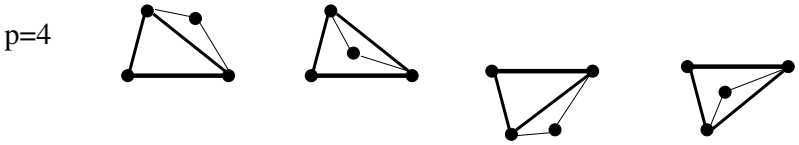


Abbildung 2.7: Mehrfachlösungen für eine Menge von vier Punkten

[Owe91], [BFH+93] und [HLS98] beschrieben werden, nutzen z. B. diesen Ansatz, der hier als der klassische Ansatz bezeichnet werden soll. Eine Alternative ist, Parameter wie Punkte, Flächen, usw. zu repräsentieren, d. h. als Objekte, die in Constraints referenzierbar sind.

Für die Repräsentierung als separate Objekte spricht Folgendes:

- Der gleiche Parameter kann in verschiedenen Constraints referenziert werden.
- Man hat immer noch die Möglichkeit, den Parameter zu fixieren. Er wird dann nicht berechnet, was dem klassischen Ansatz entspricht.
- Der Parameter kann, falls er nicht fixiert wurde, in die Freiheitsgradanalyse einbezogen werden.
- Der Parameter kann einen Typ und somit eine bestimmte Priorität für die FG-Analyse erhalten.
- Der Parameter kann ggf. auch berechnet werden.
- Durch das Berechnen von Parametern können ggf. Überbestimmtheiten vermieden werden.

Gegen die Repräsentierung als separates Objekt spricht Folgendes:

- Die Modellierung der Constraints wird komplizierter. Zum Beispiel betrachten viele Veröffentlichungen zur Freiheitsgradanalyse nur zweistellige Constraints, was einfachere Algorithmen erlaubt. Die Referenzierung von Parametern in entsprechenden Constraints würde jedoch zu einer Dreistelligkeit führen. Die Algorithmen müssten also angepasst werden.

In Ficucs werden Parameter genauso wie die geometrischen Objekte repräsentiert. Der überwiegende Teil der in Kapitel 5 gezeigten Beispiele wäre andernfalls nicht modellierbar gewesen.

2.5 Mehrfachlösungen

Auch wenn ein Modell durch entsprechende Constraints voll bestimmt ist, kann eine (endliche) Anzahl von korrekten Lösungen existieren. Man spricht hier von Mehrfachlösungen oder Varianten (engl. manchmal auch: roots). Ein einfaches Beispiel ist die in Abbildung 2.6 gezeigte Konstruktion eines Punktes P basierend auf den Abständen zu zwei anderen bekannten Punkten.

Das Problem der Mehrfachlösungen sollte nicht unterschätzt werden, denn es können exponentiell viele voneinander verschiedene Lösungen auftreten. Dies lässt sich am Beispiel einer Punktmenge in der Ebene mit den entsprechenden Abstands-Constraints leicht zeigen. Sind die notwendigen Konstruktionsschritte direkt ausführbar[27], dann beträgt die maximale Anzahl von Lösungen für p Punkte 2^{p-2}. Abbildung 2.7 zeigt ein Beispiel für $p = 4$ und somit $2^{4-2} = 4$ Mehrfachlösungen[28].

Für die Handhabung von Mehrfachlösungen gibt es eine ganze Reihe von Möglichkeiten, unter denen entsprechend der zu bearbeitenden Aufgabe gewählt werden kann:

- So ist es z. B. bei interaktiver Arbeit üblich, den Constraint-Solver die Lösung auswählen zu lassen[29], die am nächsten an der alten Lösung liegt. Dazu muss ihm diese allerdings auch zur Verfügung stehen. In einigen Domänen können auch andere Auswahlkriterien notwendig sein (siehe z. B. Abschnitt 4.1).
- Für Analysezwecke ist es hingegen von Vorteil, wenn der Geometriedatengenerator jede der möglichen Lösungen auf Anforderung erzeugen kann.
- Eine andere Herangehensweise ist, die Anzahl der Mehrfachlösungen von vorn herein einzuschränken. Hierzu lassen sich redundante geometrische Constraints einsetzen (im obigen Beispiel wären das weitere Abstände) oder Ungleichungen (z. B. $a - b < 20$). Auch topologische Constraints wie das Verbot von Selbstschneidungen (modellierbar über Ungleichungen für Abstandsparameter) können die Lösungsmenge begrenzen.

Die Möglichkeiten zur Auswahl von Mehrfachlösungen sind ein wichtiges Kriterium zur Einschätzung der Steuerbarkeit eines Constraint-Solvers (Abschnitt 3.1.2).

Die Schwierigkeit der eindeutigen Identifizierung und einer darauf beruhenden Auswahl aus mehreren möglichen Lösungen ist auch eine wichtige Ursache für Probleme beim Austausch von constraint-basierten Modellen zwischen unterschiedlichen CAx-Systemen [Hof94].

2.6 Hinweise

Im Zusammenhang mit Mehrfachlösungen wurde bereits erörtert, wie dem Constraint-Solver Informationen zur gezielten Auswahl der passenden Lösung gegeben werden können. In diesem Abschnitt werden spezielle Relationen - die Hinweise - vorgestellt, die in ihrer Definition ähnlich Constraints sind, jedoch im Lösungsprozess anders eingesetzt werden.

Hinweise dienen der Steuerung der Konstruktion. Sie sind für Planerzeugung und Planberechnung nutzbar. Der entscheidende Unterschied zu Constraints ist, dass Hinweise keine Valenz haben, d. h. sie senken nicht die Anzahl der Freiheitsgrade von Objekten[30][31].

Werden sie bei Berechnungen eingesetzt, so können sie die Auswahl von Mehrfachlösungen steuern, z. B.:

- die rechtshändige oder linkshändige Anordnung von drei *3D*-Richtungen[32]
  Ein solcher Hinweis wäre nutzbar bei Berechnungen für die Entscheidung, ob bei einer Wendeltreppe eine Stufe rechts oder links herum angesetzt werden soll, siehe Beispiel 3.20.

---

[27] vergleichbar der Konstruktion mit Zirkel und Lineal, d. h. ohne die Notwendigkeit numerischer Berechnungen

[28] In Abschnitt A.5.3 ist ein Beispiel gezeigt, in dem eine noch größere Anzahl von Lösungen möglich ist.

[29] Interaktivität soll hier heißen, dass Modelle kontinuierlich in der einen oder anderen Weise manipuliert werden, z. B. durch das Bewegen des Mauszeigers. Der Zwang zu einer manuellen Auswahl aus einer Menge von Lösungen würde hier den Arbeitsfluss stören.

[30] Sie steuern zwar die Suche bzw. schränken den Lösungsraum ein, senken jedoch nicht die Dimension des Lösungsraumes.

[31] Es gibt auch Sonderfälle: So liefert der Constraint „$a = b$“ ($Val = 1$) die gleiche Aussage wie die beiden Hinweise „$a \leq b$“ und „$a \geq b$“, die jeweils eine Valenz von null haben.

[32] siehe XHS-Hinweise

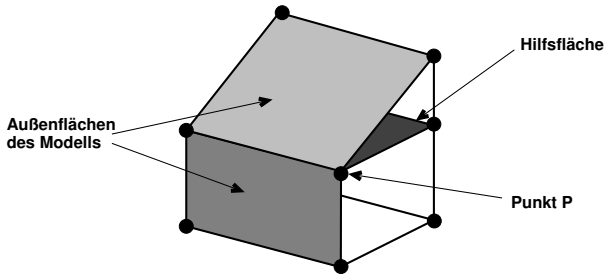


Abbildung 2.8: Degenerierter Fall für die Punktkonstruktion

- und die rechtshändige oder linkshändige Anordnung von drei *2D*-Punkten.
  Ein solcher Hinweis könnte in Abbildung 2.6 und Abbildung 2.7 zur Lösungsauswahl herangezogen werden.

Den Planerstellungsprozess können sie dadurch unterstützen, dass sie spezielle Informationen bereitstellen, z. B.:

- als Hinweise für die Konstruktion eines Punktes. Ein Punkt lässt sich *nie* eindeutig aus drei Ebenen, die senkrecht auf einer vierten stehen, konstruieren. Die Punktinzidenzen implizieren, dass sich die Ebenen in einer Gerade schneiden.

  Beispiel 2.16: Erkennung degenerierter Fälle in der Punktkonstruktion
  Abbildung 2.8 zeigt einen Punkt P, der in vier Flächen liegt. Das sind eine Hilfsfläche und drei Außenflächen. Zwei von ihnen wurden im Bild eingefärbt, durch die dritte kann man hindurch auf die Hilfsfläche schauen. Um einen Punkt in *3D* zu konstruieren werden im Allgemeinen drei Flächen benötigt. Hier stehen jedoch vier zur Verfügung. Wählt der Solver die Außenflächen aus, so kann P konstruiert werden. Wählt er aber beispielsweise die Hilfsfläche und die beiden gefärbten Außenflächen, so ist die Lage von P nicht eindeutig bestimmt. Er kann irgendwo auf der gemeinsamen Kante liegen. Die transparente Außenfläche ist aufgrund ihrer Lageparameter für die Konstruktion unabdingbar. □
  Derartige Mengen von Ebenen lassen sich schon im voraus bestimmen[33] und können dem Constraint-Solver übermittelt werden.

- In Hinweisen können Informationen über alternative Constraints hinterlegt sein. Ein derartiger Hinweis beschreibt die Menge der alternativen Constraints (die Summe ihrer Valenzen sei V_G) und die für die Konstruktion zulässige Valenz V_Z oder die überschüssige Valenz $V_{\ddot{U}} = V_G - V_Z$. Siehe Abschnitt 3.3.8 zur Behandlung von alternativen Constraints in der Freiheitsgradanalyse sowie Abschnitt 3.4.2, in dem die Behandlung bei der Planerstellung beschrieben ist.

  Beispiel 2.17: Einsatz alternativer Constraints bei der Definition eines Schweißpunktes auf einem Getriebeglied
  Ein Schweißpunkt ist auf einem Getriebeglied definiert durch den Abstand zu *einem* Ende des Getriebegliedes. Zur Vermeidung von Iterationen wäre die explizite Angabe der Abstände zu *beiden* Enden des Getriebegliedes sinnvoll. Die entsprechenden Abstands-Constraints würden dann eine Menge alternativer Constraints bilden, die eine überschüssige Valenz $V_{\ddot{U}} = 1$ haben. □

Ungleichungen sind spezielle Hinweise. Sie verwenden Relationen wie $<, >, \leq, \geq, \neq$ und gelten für Werte von Parametern, z. B. „Abstand Punkt zu Ebene“$< d$ oder $a \cdot b < 3$. Ungleichungen werden in Modellen insbesondere zur Auswahl von Mehrfachlösungen eingefügt.

[33] Eine Erkennung im Solver würde dessen Robustheit erhöhen, insbesondere dann, wenn sich die Zugehörigkeit einer Ebene zu einer solchen Ebenenmenge erst im Laufe einer Interaktion oder Optimierung (temporär) einstellt. Aber auch bei Erkennung im Solver muss die entsprechende Information für die Plangenerierung in einer geeigneten Form (z. B. diese Hinweise) hinterlegt werden.

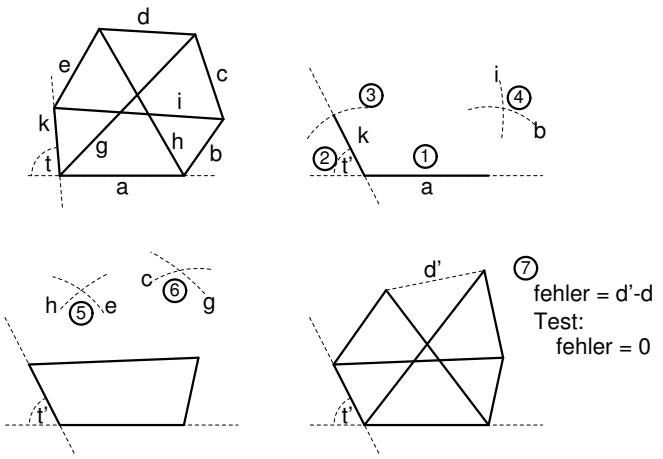


Abbildung 2.9: Sechseck mit einer Versteifung, die zu einer zyklischen Abhängigkeit führt

Bei der Modellierung ist es möglich durch die Definition von Constraints bewusst eine Überbestimmtheit zur Eingrenzung der Lösung bei Mehrfachlösungen zu erzeugen. Die explizite Formulierung als Hinweis ist jedoch günstiger. Um dies zu unterstützen lassen sich im beschriebenen System bei Bedarf Constraints als Hinweise markieren. Sie haben dann keine Valenz mehr, d. h. sie beeinflussen nicht mehr die Freiheitsgrade der Objekte, die sie referenzieren.

2.7 Zyklische Abhängigkeiten - iterative Lösungsfindung

In Abschnitt 2.1.2 wurde ein Beispiel gezeigt, wie ein Konstruktionsplan aussehen könnte. Er bestand aus 3 Teilschritten, die nacheinander durch Konstruktionen mit Zirkel und Lineal[34] lösbar waren. In einem solchen Fall soll von direkten Konstruktionen (direkten Lösungen) gesprochen werden. Derartige Konstruktionen können in linearer Zeit[35] abgearbeitet werden. Es ist jedoch nicht immer möglich, für ein gegebenes Modell einen Plan mit ausschließlich direkten Konstruktionen zu finden. Zyklische Abhängigkeiten können die Nutzung direkter Konstruktionen verhindern. Beim Auftreten zyklischer Abhängigkeiten besteht die Möglichkeit das Problem als Gleichungssystem zu formulieren und einen entsprechenden Gleichungslöser zu starten, jedoch hätte ein solches Vorgehen entscheidende Nachteile, z. B. eine potentiell hohe Zeitkomplexität und eine schlechte Steuerbarkeit (sie Abschnitt 4.2.3).

Um zum einen die günstige Zeitkomplexität sowie die gute Steuerbarkeit der direkten Konstruktion weitestgehend nutzen zu können und zum anderen trotzdem zu einer Lösung zu kommen, kann nach dem Erkennen von Zyklen auf ein hybrides Verfahren umgeschaltet werden [HB97].

Beispiel 2.18: Iterative Positionsberechnung von sechs Punkten in 2D
Abbildung 2.9 zeigt die entsprechenden Konstruktionsschritte für ein Sechseck. Zu den sechs Eckpunkten wurden neun Abstands-Constraints gesetzt, die das Sechseck voll bestimmen. Dadurch sind implizit auch die Winkel gegeben, z. B. der eingezeichnete Winkel t[36]. Nach Schritt (1) der Konstruktionssequenz kann keine direkte Konstruktion mehr ausgeführt werden. Deshalb wird hier mit (2) eine Iteration gestartet. Das geschieht durch die Wahl eines Wertes t' (der korrekte Wert t ist nur implizit durch die neun Constraints gegeben!). Aufbauend auf dieser Wahl können nun eine Reihe direkter Konstruktionsschritte (3)...(6) erfolgen, nach deren Ausführung die Lage aller Punkte bestimmt ist. Die Konstruktionsschritte berücksichtigen jedoch nicht den Parameter d. Statt dessen ergibt sich in Abhängigkeit von t' ein Abstand

[34] bzw. rechnerintern durch entsprechende Methoden
[35] bezüglich der Anzahl der zu konstruierenden Objekte
[36] Die Benennung mit t erfolgte, weil dieser Winkel als Parameter für die iterativen Konstruktionen dienen soll.

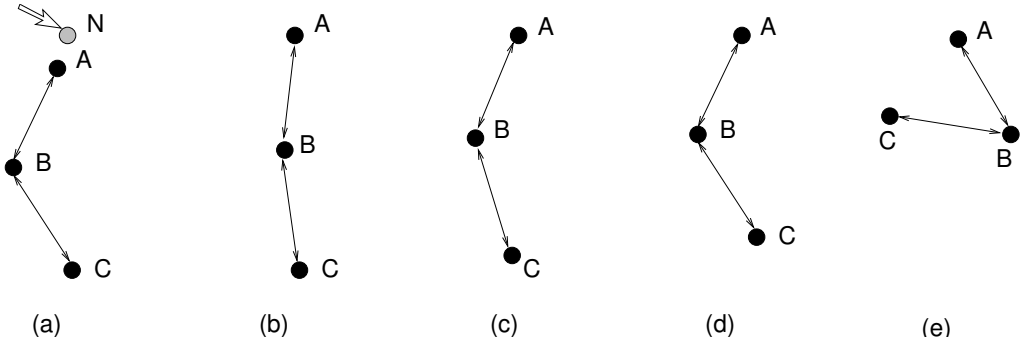


Abbildung 2.10: Beispiel eines unterbestimmten Modells aus drei Punkten und zwei Abstands-Constraints (die Abstände seien fixiert). (a) zeigt das Modell in seinem Ausgangszustand, wobei N die Position ist, zu der der Punkt A verschoben werden soll. (b) bis (e) stellen vier der unendlich vielen Lösungsmöglichkeiten dar.

d', so dass in Schritt (7) der aktuelle Abstand d' und der geforderte Abstand d zu vergleichen sind. Aus der Differenz der beiden ergibt sich jeweils ein Fehlerwert. Die gefundenen Paare von t' und entsprechendem Fehlerwert dienen in einem numerischen Verfahren zur Bestimmung einer Nullstelle (Fehlerwert $f(t) = 0$), d. h. einer Konstruktion, die allen neun Abstands-Constraints genügt[37]. Von Schritt (7) - dem Test - wird also so lange zu Schritt (2) - der Wahl eines neuen t' - zurückgesprungen, bis $f(t') = d' - d = 0$ ist. □

2.8 Plausibilität von Lösungen und Steuerbarkeit der Lösungssuche

Die Plausibilität von berechneten Lösungen spielt für die Akzeptanz einer Software durch den Nutzer eine sehr große Rolle. Dies gilt auch für CAx-Programme sowie einen von ihnen genutzten Constraint-Solver. Durch geeignete Modellierung[38] und Algorithmen muss stets die Steuerbarkeit der Berechnungsabläufe sichergestellt sein, insbesondere muss der Nutzer stets das Verhalten der Modelle nachvollziehen können. Die Nachvollziehbarkeit ist nicht gewährleistet:

- beim Springen zwischen möglichen Mehrfachlösungen und
- bei Modifikationen an solchen Teilen eines unterbestimmten Modells, von denen der Nutzer annimmt, dass sie unverändert bleiben.

Das Problem der Mehrfachlösungen und eine Lösungsmöglichkeit über die Auswertung von Hinweisen wurde in den Abschnitten 2.5 und 2.6 bereits diskutiert. Eine ausführlichere Betrachtung findet in Kapitel 4 anhand der Berechnung von Bewegungspfaden für Punkte in Gelenkketten sowie anhand der Berechnung starrer Modelle statt.

Zur Verdeutlichung, wie unterschiedlich Berechnungsresultate für Veränderungen an unterbestimmten Modellen seien können, sei folgendes Beispiel gegeben.

Beispiel 2.19: Mögliche Ergebnisse des Ziehens an einer Punktekette
In Abbildung 2.10 wird ein Modell aus drei Punkten und zwei festen Abständen gezeigt. Die Veränderung soll in der Verschiebung des Punktes A auf die Position N bestehen (Abbildung 2.10(a)). Da das Modell

[37] Der Abstand d ist mit dem Finden der Nullstelle erfüllt. Alle anderen Abstände wurden schon beim direkten Konstruieren eingehalten.

[38] Gemeint ist hier sowohl die Mächtigkeit der gewählten Repräsentationsform als auch das konkrete Modell, das sich dieser Mächtigkeit bedient.

unterbestimmt ist, können grundsätzlich unendlich viele verschiedene Lösungen für die neuen Positionen von B und C gefunden werden, die die beiden Abstands-Constraints erfüllen. Vier von ihnen sollen hier diskutiert werden.

- Bei hoch priorisiertem C könnte C konstant bleiben (vgl. Abschnitt 2.2.5). B würde sich dann als Schnittpunkt zweier Kreise ergeben (Abbildung 2.10(b)). Die Änderung bleibt hier nahe am Interaktionspunkt (A). Dieses Verhalten kann für Modifikationen an komplexen Modellen wichtig sein.
- Abbildung 2.10(c) zeigt ein Verhalten, wie es für Modelle mit bewegten Körpern plausibel sein kann, z. B. wenn das Modell eine Kette aus zwei starren Gliedern beschreibt. Ausgehend vom Interaktionspunkt werden hier zunächst B und dann C so berechnet, dass sie sich gegenüber ihrer vorhergehenden Position möglichst wenig verschieben.
- In Abbildung 2.10(d) erfolgt eine Verschiebung des Modells. Es wird hierbei (temporär) wie ein starrer Körper betrachtet. Ein entsprechender Verschiebemodus erspart dem Nutzer die Definition eines Abstands-Constraints zwischen A und C sowie seine Löschung nach der Verschiebung.
- Selbst das in Abbildung 2.10(e) gezeigte Ergebnis ist gültig, wenn auch das Finden einer plausiblen Erklärung, warum der Nutzer gerade diese Änderung erwarten sollte, schwer fällt. Denkbar wäre, dass das Ergebnis im Kontext eines Strebens nach gleichmäßiger Verteilung der Punkte akzeptabel ist.

□

Wie das obige Beispiel zeigt, benötigt ein allgemein einsetzbarer Constraint-Solver also zum einen unterschiedliche Berechnungsstrategien und zum anderen auch die Informationen, die ihn bei der Auswahl der Berechnungsstrategie und ggf. auch einer bestimmten Lösung steuern. Vertiefende Überlegungen zur Steuerbarkeit der Lösungssuche sind in Abschnitt 3.1.2 zu finden.

2.9 Clustering

Beim Clustering werden in einem (Gesamt-)Constraint-Netz, mit dem ein Modell beschrieben wird, geeignete Teilnetze (die sogenannten Cluster) ermittelt. Ihre Berechnung kann mit der Erstellung von Hilfskonstruktionen verglichen werden, die zur Beschleunigung des Berechnungsprozesses beitragen sollen.

Beispiel 2.20: Hilfskonstruktionen ermöglichen die direkte Konstruktion eines speziell modellierten Fünfecks
Abbildung 2.11 zeigt ein Fünfeck, das ohne die Zuhilfenahme von Clustering iterativ konstruiert werden müsste. Grund hierfür ist die Art und Weise der Modellierung, die zwar ein vollbestimmtes Modell zeigt[39], jedoch die für eine direkte Konstruktion nötigen Abstände nur implizit definiert. Mit zwei Hilfskonstruktionen gemäß Abbildung 2.12 lassen sich aber die Längen der Strecken P_1P_4 und P_2P_4 bestimmen. Im nächsten Schritt können dann die Längen von P_1P_4 und P_2P_4 genutzt werden, um P_4 zu konstruieren (Abbildung 2.12). Anschließend werden P_3 mittels B und C sowie P_5 mittels D und E gefunden. □

Die Algorithmen zum Auffinden der Cluster basieren in der Regel auf generischer Information, wie sie auch in der Freiheitsgradanalyse verwendet wird. Somit ergeben sich beim Clustering die gleichen Probleme mit degenerierten Fällen. Als unproblematisch haben sich jedoch sogenannte Äquivalenz-Cluster erwiesen, die auf Basis von Gleichheits-Constraints gebildet werden können. Ziel ist es hierbei, alle Constraints für eine Menge von Objekten, welche per Constraint als gleich definiert wurden, auf ein (Referenz-)Objekt umzuleiten, wodurch sich dieses Objekt eher direkt konstruieren lässt. Abbildung 2.13 zeigt ein entsprechendes Beispiel. Aufgrund der positiven Erfahrungen wurde die Behandlung von Äquivalenz-Clustern in Ficucs aufgenommen (siehe Abschnitt 3.2.2).

[39] Das gezeigte Fünfeck hat zunächst zehn Freiheitsgrade (Menge von fünf Punkten in $2D$, $5 * 2FG = 10FG$). Im Modell sind die fünf Seitenlängen und zwei Winkel entsprechend Abbildung 2.11 gegeben. Jeder der entsprechenden sieben Constraints hat eine Valenz von eins, so dass das Fünfeck $10FG - 7Val = 3FG$ hat. Es liegt somit ein voll bestimmter starrer Körper in allgemeiner Lage vor.

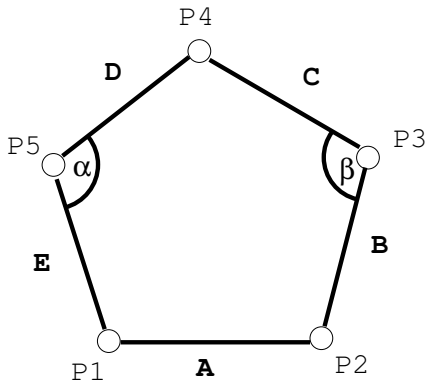


Abbildung 2.11: Modell eines Fünfecks, das durch Clustering effizient berechnet werden kann.

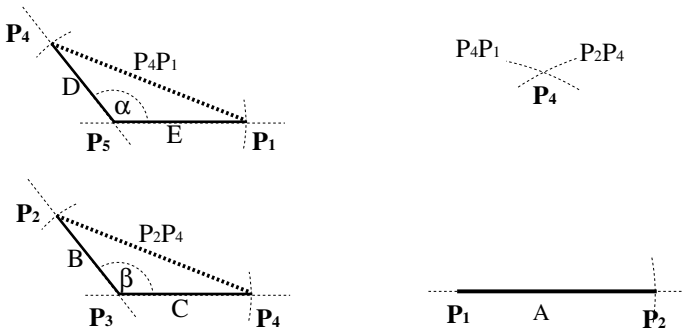


Abbildung 2.12: Hilfskonstruktionen und Konstruktion des Punktes P_4 aus Abbildung 2.11

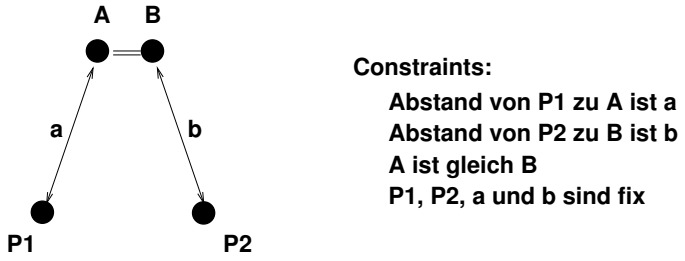


Abbildung 2.13: Beispiel, wie ein aus zwei Punkten gebildetes Äquivalenz-Cluster für die Konstruktion genutzt werden kann. Ohne Clustering ließen sich weder A noch B konstruieren. Leitet man jedoch den Abstands-Constraint von B nach A um, so kann A als Schnittpunkt zweier Kreise konstruiert werden.

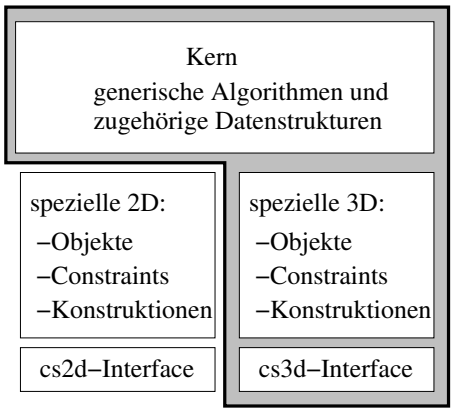


Abbildung 2.14: Übersicht über die Modulstruktur des Constraint Solvers. Der grau hinterlegte Bereich wird für die Modellierung räumlicher Modelle verwendet.

Eine tiefergehende Übersicht über Clustering-Ansätze und Diskussion der mit ihnen verbundenen Probleme wird in Abschnitt 3.7 vorgenommen.

2.10 Prinzipieller Aufbau des Constraint-Moduls

Ein erheblicher Teil der Algorithmen, z. B. Freiheitsgradanalyse, Umgang mit Parametern und iterative Konstruktionen sind unabhängig von den spezifischen Ausprägungen der *2D*- und *3D*-Constraints. Deshalb wurde die in Abbildung 2.14 gezeigte innere Modulstruktur gewählt, die generische Algorithmen und entsprechende Datenstrukturen in einem Modulkern zusammenfasst.

Alle problemraum-spezifischen Datenstrukturen und Algorithmen, wie:

- geometrische Objekte der Ebene oder des Raumes,
- Constraints, die solche Objekte referenzieren und
- Berechnungsalgorithmen, mit denen die jeweiligen geometrischen Objekte konstruiert werden,

sind außerhalb des Modulkerns in speziellen Erweiterungen angeordnet. Je nach Anwendungsfall werden sie in das verwendete Constraint-Modul aufgenommen. Der in Abbildung 2.14 grau hinterlegte Bereich zeigt die für die Modellierung von räumlichen Modellen relevanten Teile des Constraint-Solvers, die das sogenannte *3D*-Modul umfasst. Analog wird ein *2D*-Modul für die Arbeit in der xy-Ebene verwendet.

In Abbildung 2.15 wird der Aufbau von Ficucs am Beispiel des *2D*-Moduls detaillierter gezeigt. Über das Interface kann eine Applikation (siehe Kapitel 5) auf den Constraint-Solver zugreifen, das Modell aufbauen und modifizieren, die Lösungssuche steuern sowie Berechnungen starten. Bei der Übertragung des Modells von der Applikation an den Constraint-Solver wird eine Modell-Anpassung vorgenommen, um z. B. redundante Bestandteile des Modells zu entfernen und so die internen Arbeitsabläufe effizienter zu gestalten (Abschnitt 3.2).

Die Steuerung der Abläufe im Constraint-Solver wird vom Constraint-Manager vorgenommen. Die zentrale Datenstruktur im Constraint-Solver ist das Constraint-Netz, in dem die Objekte und Constraints mit ihren Verknüpfungen hinterlegt sind. Gegenüber Änderungen von Objektpositionen und Parameterwerten eines bestimmten Modells ist das Constraint-Netz invariant. Aus den

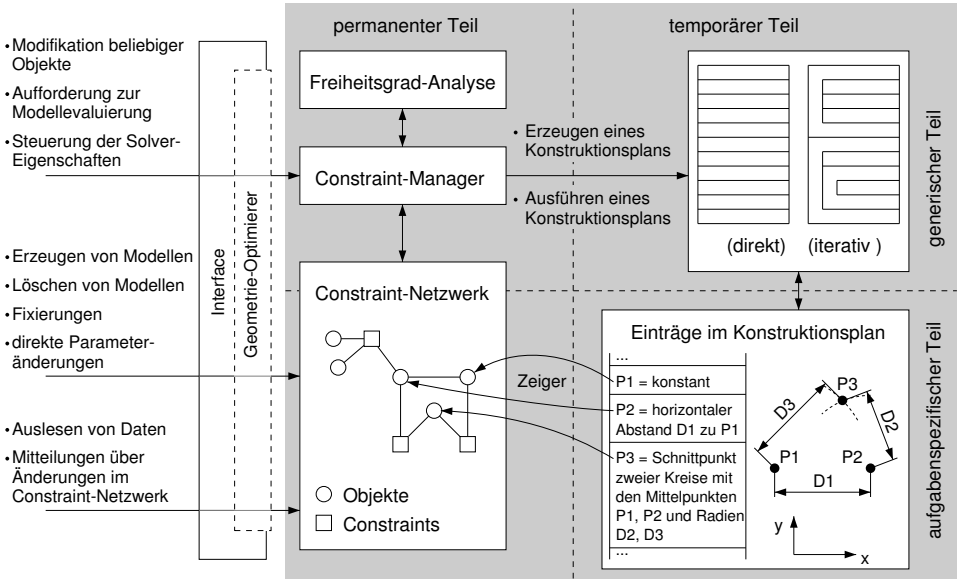


Abbildung 2.15: Schematische Darstellung der im Constraint-Solver verwendeten Datenstrukturen und Algorithmen sowie ihrer Zusammenhänge und der wichtigsten Aktionen.

in ihm gespeicherten Informationen zu den Freiheitsgraden wird für die Freiheitsgradanalyse ein spezieller bipartiter Graph aufgebaut und inkrementell[40] an die jeweiligen Ziele der Änderungen angepasst (Abschnitt 3.3).

Nachdem durch die Freiheitsgradanalyse Über- und Unterbestimmtheiten behandelt wurden, wird eine weitere Datenstruktur, der Konstruktionsplan, aufgebaut (Abschnitt 3.4). Er enthält alle Schritte, die für die Konstruktion der im Modell verwendeten Objekte notwendig sind. In den Berechnungen werden möglichst nur direkte Konstruktionsschritte durchgeführt, ähnlich den Konstruktionen mit Zirkel und Lineal. Erst wenn nicht weiter direkt konstruiert werden kann, wird auf iterative Konstruktionen zurückgegriffen (Abschnitt 3.6), wobei im Inneren der Iterationen wieder direkte Konstruktionen zur Anwendung kommen.

Wie oben gezeigt, ist Ficucs in einen generischen und in einen aufgabenspezifischen Teil gegliedert. Im generischen Teil sind im Sinne der objekt-orientierten Programmierung alle von den konkreten geometrischen Objekten unabhängigen Basisklassen für Objekte, Constraints, Konstruktionen usw. sowie die entsprechenden Algorithmen wie Freiheitsgradanalyse, Planerstellung und Nullstellensuche vereint. Der aufgabenspezifische Teil umfasst konkrete Objekte für die Ebene oder den Raum (z. B. Punkte, Geraden, Ebenen) inklusive den Konstruktionsschritten, die für die Berechnung derartiger Objekte benutzt werden sollen. Die Trennung von generischem und aufgabenspezifischem Teil hat sich für den Einsatz des Constraint-Solvers in den unterschiedlichsten Projekten bewährt.

[40] Der Vorteil einer inkrementellen Anpassung liegt darin, dass sie für kleine Änderungen sehr effizient vorgenommen werden kann, siehe Seite 53ff.

Kapitel 3

Datenstrukturen und Algorithmen

In diesem Kapitel wird der grundlegende Aufbau des entwickelten Constraint-Solvers erörtert. Nach einer Diskussion der Entwicklungsziele liegt der Fokus auf den intern verwendeten Datenstrukturen sowie den auf ihnen arbeitenden Algorithmen.

3.1 Ziele

In den meisten Projekten in denen der Constraint-Solver integriert ist, steht die Interaktivität der Berechnungen im Vordergrund, d. h. der Constraint-Solver soll schnell für den Nutzer plausible Lösungen liefern. Ruckende Bewegungen oder lange Wartezeiten sind hier nicht akzeptabel. Eine hohe Geschwindigkeit und gute Steuerbarkeit der Lösungssuche verursacht u. U. jedoch auch erheblichen Aufwand beim Erstellen der Modelle oder bei der Implementierung der entsprechenden Programme. Ein hoher Modellierungsaufwand senkt aber wiederum die Akzeptanz bei den jeweiligen Entwicklern. Diese durchaus gegenläufigen Ansichten sind bei der Entwicklung eines Constraint-Solvers zu berücksichtigen. Entsprechende Entwicklungsziele werden im folgenden Text diskutiert.

3.1.1 Geschwindigkeit

Trotz der stetig steigenden Rechenleistung der Computer wird die Komplexität der in einer akzeptablen Zeit berechenbaren Modelle stets begrenzt bleiben. Demzufolge ist es ein Ziel, möglichst solche Methoden und Algorithmen zu verwenden, die bezüglich der Rechenzeit effizient sind.

Im Kontext der Berechnungen von geometrischen Objekten heißt das, dass die einzelnen Objekte möglichst direkt zu konstruieren bzw. zu berechnen sind[1]. Die direkte Konstruktion ermöglicht exakte Berechnungen[2] bei einer bezüglich der Objektanzahl linearen Berechnungskomplexität. Andere Lösungsansätze, wie z. B. die Verwendung numerischer Gleichungslöser [Hüs88, Oun01] oder die Relaxation [SB98a], können grundsätzlich auch beliebig genaue Lösungen berechnen, die Geschwindigkeit ist jedoch sehr von der Wahl der Startparameter abhängig[3]. In Abschnitt 4.2.3 ist ein entsprechendes Beispiel gezeigt.

[1] In Abschnitt 3.4.4 wird eine Möglichkeit gezeigt, wie zeitaufwändige iterative Konstruktionen vermieden werden können.

[2] zumindest bezüglich der mit den verwendeten Zahlenformaten verbundenen Rechengenauigkeit

[3] sowie vom konkreten Modell und den Konvergenzeigenschaften des verwendeten Algorithmus

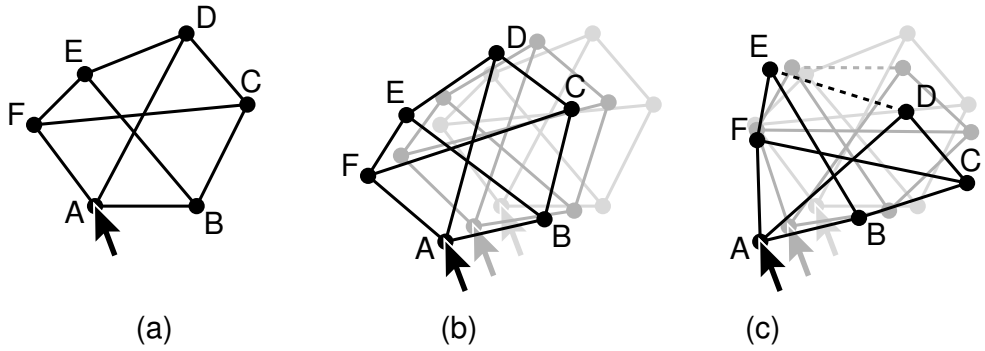


Abbildung 3.1: Beispiel, in dem die Iterationsvermeidung zu einer unerwünschten Deformierung führt. (a) Ausgangsmodell, (b) Modell mit unveränderter Priorisierung, dessen Abstände bei Bewegungen beibehalten werden. Allerdings sind iterative Berechnungen nötig. (c) Modell, bei dem zur Iterationsvermeidung der Parameter für den Abstand $|\overline{ED}|$ niedriger priorisiert wurde. $|\overline{ED}|$ bleibt dadurch nicht mehr konstant. Statt dessen erhält F einen Freiheitsgrad, so dass F nahe seiner alten Position konstruiert werden kann. Die Umpriorisierung führt zu einer merklichen und somit unerwünschten Deformierung.

Weitere Ansätze, die Rechenzeit einsparen helfen, sind:

- Bei sich wiederholenden Analyse- und Berechnungsabläufen sollten alle Schritte, deren Ergebnisse sich nicht ändern, in eine Vorverarbeitungsphase ausgelagert werden[4].
- Die Nutzung von Domänenwissen muss sich nicht auf die Verwendung vordefinierter Konstruktionsschritte (z. B. *3D*-Punkt aus drei Ebenen) in der Berechnung der Modelle beschränken. Auch in der Analysephase können spezielle Transformationen sinnvoll sein. Diese wirken dann ähnlich domänenspezifischen symbolischen Umformungen.
- Es sollten nur dort allgemeine symbolische oder numerische Gleichungslöser eingesetzt werden, wo kein spezielles Domänenwissen existiert.
- Bezüglich des vorgestellten Constraint-Solvers ist die Vermeidung iterativer Konstruktionen eine Möglichkeit, Geschwindigkeitsvorteile zu erzielen. Dies kann aber im Konflikt mit anderen Modellierungszielen stehen, z. B. dem Erfüllen der vom Nutzer gewünschten Priorisierung, denn um Iterationen zu vermeiden, ist oft eine (zumindest teilweise) Umpriorisierung nötig.

  Beispiel 3.1: Unerwünschter Fall beim Konflikt zwischen Iterationsvermeidung und Priorisierung
  In der Menge von sechs Punkten (Abbildung 2.9) seien die neun Abstandsparameter nicht fixiert, sondern nur sehr hoch priorisiert. Normalerweise werden die Freiheitsgrade dann so verteilt, dass das Sechseck starr bleibt. Ist die Iterationsvermeidung jedoch wichtiger als die Priorisierung, dann wird ein Abstandsparameter nicht frei bleiben, d. h. ihm wird an der Stelle, wo bei iterativen Konstruktionen die Korrektheit des Abstandes gemessen wird, einfach nur der gemessene Wert zugewiesen. □

- Inkrementelle Algorithmen zur Transformation, Freiheitsgradverteilung und Planerstellung können wesentlich zur Geschwindigkeitssteigerung beitragen, zumindest wenn nach kleineren Modelländerungen oft Berechnungen auszuführen sind.

[4]Das führt z. B. zur Trennung der Freiheitsgradanalyse sowie Planerstellung von der Planauswertung. Problematisch ist jedoch, dass die Freiheitsgradanalyse nicht stets unabhängig von den Berechnungsergebnissen ist. Derartige Fälle müssen erkannt werden und zu einer neuen Freiheitsgradanalyse und Planerstellung führen (siehe Beispiel 3.14).

Inkrementelle Änderungen am Modell sind sowohl für die interaktive Arbeit als auch für die programmgesteuerte lösungssuchende oder optimierende Struktursynthese[5] typisch, d. h. dem bestehenden Modell werden einzelne Objekte und Constraints hinzugefügt oder entnommen. Einfache Ansätze verwerfen bei jeder dieser Änderungen das alte interne Modell und bauen es komplett neu auf. Für interaktive Änderungen kleinerer Modelle ist diese Vorgehensweise mit heutiger Rechentechnik eher unkritisch. Bei programmgesteuerten Änderungen wird jedoch die inkrementelle und damit möglichst schnelle Anpassung des internen Modells stets von Bedeutung bleiben, insbesondere wenn sogenannte *brute force*-Ansätze[6] für die Struktursynthese zum Einsatz kommen.

Allgemein kann auch das Laden eines neuen Modells inkrementell erfolgen. Da es während des Ladevorganges jedoch nicht nötig ist die internen Datenstrukturen stets konsistent zu halten, ermöglicht die Verwendung von nicht-inkrementellen Algorithmen zur Erzeugung des internen Modells eine Zeiteinsparung. Zum Beispiel können der bipartite Graph für die Freiheitsgradanalyse und eine entsprechende Freiheitsgradverteilung auch nach dem Aufbau des internen Constraint-Netzes neu erstellt werden.

3.1.2 Steuerbarkeit

Neben der Geschwindigkeit, in der der Constraint-Solver zu den Lösungen kommt, ist die Steuerbarkeit der Lösungssuche von großer Bedeutung für die Akzeptanz durch den Nutzer. Es ist wichtig, dass dem Nutzer beim Aufbau der Modelle oder bei interaktiven Modifikationen der Modelle die Möglichkeit gegeben wird, das von ihm erwartete Verhalten bei der Behandlung von:

- Unterbestimmtheit,
- Überbestimmtheit und
- Mehrfachlösungen

zu formulieren. Zum anderen müssen die internen Algorithmen diese Vorgaben möglichst genau beachten. In Ficucs sind folgende Beschreibungsmittel vorgesehen:

- Die Steuerbarkeit der Lösung von Unterbestimmtheit[7] soll durch die Priorisierung der Objektfreiheitsgrade ermöglicht werden. Um den Modellierungsaufwand gering zu halten, ist die initiale Priorisierung typabhängig.

  Beispiel 3.2: Typabhängige Priorisierung in einem Dreieck
  Ein Dreieck hat in *2D* entsprechend seinen drei Eckpunkten zunächst $2FG * 3 = 6FG$, d. h. drei mehr als ein starrer Körper. Werden die Seitenlängen vermaßt, so kommen je drei Längenparameter und Abstands-Constraints hinzu. Pro Vermaßung sind das $1FG$ und $1Val$, die sich jeweils aufheben. In der Summe sind also immer noch $6FG$ vorhanden, so dass sich z. B. die Punkte verschieben lassen könnten und die Abstandsparameter berechnet werden würden. Da in den meisten Applikationen die Form von Objekten typischerweise erhalten bleiben soll, werden im Allgemeinen Parameter vom Constraint-Solver zunächst hoch priorisiert. Die Applikationssoftware oder der Nutzer können diese Vorgabe jedoch überschreiben, z. B. um die Parameter (oder auch nur einen ganz speziellen) zu fixieren oder aber sie sehr gering zu priorisieren, wodurch sie wieder wie oben beschrieben zum Messen der Abstände dienen können. □

[5] Eine lösungssuchende Struktursynthese impliziert, dass mit der Struktur des aktuellen Modells keine Konsistenz herstellbar ist. Eine optimierende Struktursynthese kann hingegen auf einer bekannten Struktur mit konsistenten Modellparametern aufbauen.

[6] Suchen den kompletten Lösungsraum nach einem sehr einfachen Algorithmus ab. Wegen dieser Einfachheit sind sie bei Programmierern beliebt, haben jedoch im Vergleich zu aufwändigen („intelligenten“) Suchstrategien ein schlechtes Laufzeitverhalten.

[7] Es ist zu beachten, dass auch starre Körper im Allgemeinen in ihrer Lage nicht bestimmt sind. Werden sie bewegt ist also ggf. zu entscheiden, wie mit dieser Unterbestimmtheit umzugehen ist.

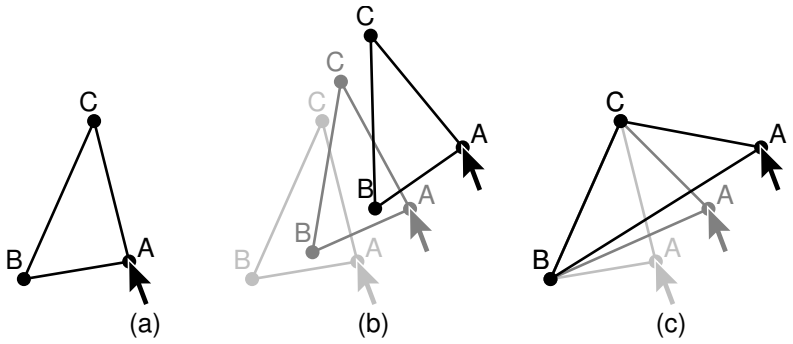


Abbildung 3.2: Auswirkung der Verschiebung eines Punktes bei unterschiedlicher Priorisierung. (a) Die Abstandsparameter haben eine höhere Priorität als die Punktkoordinaten von B und C. (b) Die Abstandsparameter haben eine niedrigere Priorität als die Punktkoordinaten von B und C.

Da den aktuellen Interaktionspunkten (z. B. das Objekt, an dem der Mauszeiger oder ein spezielles *Handle*[8] angreift) stets besondere Bedeutung zukommt, müssen sie vom Constraint-Solver standardmäßig hoch priorisiert werden. Ein Beispiel, wo dies nicht zutrifft, ist in Abschnitt 5.2.3 gezeigt.

- Zur Steuerung von Überbestimmtheit soll die Priorisierung von Constraints dienen. Dies führt zu den sogenannten *Soft-Constraints*. Vergleicht man die Priorisierung von Objekten und Constraints, so entsprechen die Soft-Constraints den nicht fixierten (aber in irgendeiner Weise priorisierten) Objekten und die „harten“ Constraints den fixierten Objekten. So wie die Freiheitsgradanalyse in unterbestimmten Fällen die Prioritäten der nicht fixierten Objekte gegeneinander abwägt, muss sie in überbestimmten Fällen auch die Prioritäten der Soft-Constraints berücksichtigen.

- Die gezielte Auswahl aus einer Menge von Mehrfachlösungen wird durch die Beachtung entsprechender Hinweise möglich. Es ist zu beachten, dass im vorgestellten iterativ-konstruktiv arbeitenden Constraint-Solver die Berücksichtigung globaler Kriterien wie „minimale Energie“ problematisch ist (vgl. Abschnitt 3.6.4).

Teilweise ist die Gültigkeit bzw. die Wirksamkeit von Constraints von bestimmten Bedingungen abhängig. Um Constraints gezielt aus- und einschalten zu können, wurde im Constraint-Solver das Konzept der *Conditional Constraints* eingeführt [BBD99]. Ein Beispiel für eine auf Applikationsebene automatisierte Generierung und Löschung von Constraints aber auch von Objekten zeigt Abschnitt 5.3.1 anhand der *Programmierbaren Features*.

In einigen Veröffentlichungen zum Thema „Geometrische Constraints“ wird das Problem der Steuerbarkeit allgemein mit Aussagen wie „Der Nutzer sollte die Möglichkeit des interaktiven Eingriffs zur Lösungsauswahl haben.“ abgetan (u. a. [Kra92]). Dies zeigt, dass das Problem von den Autoren stark unterschätzt wird, denn nur in Ausnahmefällen kann der Nutzer ein derartiges Vorgehen akzeptieren. Voraussetzung ist, dass die korrekte Lösung meist automatisch beibehalten wird. Dies ist möglich, wenn nur sehr wenige Lösungen zur Auswahl stehen oder Parameteränderungen an den Modellen sehr langsam vorgenommen werden. In der Praxis treten jedoch schnell größere Modelle mit entsprechenden Mehrfachlösungen auf, bei denen sich die richtige (bzw. erwartete) Lösung nicht stets durch die Nähe zur letzten Lösung bestimmen lässt. Die Wahl der falschen Lösung und der dadurch verursachte Korrekturaufwand führen regelmäßig zur Frustration der Nutzer.

[8] deutsch: Griff, Henkel, etwas das hilft mit einem Objekt umzugehen, es handzuhaben bzw. es gezielt zu verändern.

Zum Teil liegt die Ursache für eine verminderte Steuerbarkeit bereits in der Wahl des Berechnungsverfahrens. Hier führen insbesondere numerische Lösungsverfahren regelmäßig zu großen Problemen. Die Behauptung „Durch Wahl eines anderen Startpunktes kann eine andere Lösung gefunden werden.“ (z. B. [SB98b]) ist zwar korrekt, aber es besteht eben nur die Möglichkeit, dass eine andere Lösung gefunden wird. Eine für den Nutzer akzeptable Steuerbarkeit ist in der Regel nicht gegeben, denn bedingt durch den chaotischen Charakter der Verteilung der Lösungen in den Suchräumen (vgl. Abbildungen im Abschnitt 4.2.3) ist nicht vorhersagbar:

- wo der neue Startwert liegen soll,
- wie viele Versuche benötigt werden, bis eine Lösung gefunden wird,
- welche Lösung gefunden wird und
- ob es überhaupt noch weitere Lösungen im Suchraum gibt.

Aus diesem Grunde sollte die Verwendung einer numerischen Lösungssuche im Constraint-Solver möglichst vermieden werden. Wird sie dennoch benötigt (was bei *2D*- wie auch *3D*-Modellen durchaus typisch ist), so muss der Algorithmus zur numerischen Lösungssuche bzw. seine Implementierung das Problem der Steuerbarkeit berücksichtigen.

3.1.3 Robustheit

Es ist wichtig, dass die verwendeten Algorithmen robust gegenüber explizit oder implizit vorliegenden degenerierten Fällen sind. Einige solcher Fälle sind die folgenden:

- Es treten immer wieder doppelt definierte Constraints und redundante Fixierungen auf.

  Beispiel 3.3: Überbestimmtes Modell in MASP
  Ein Getriebeglied hat nach seiner Erzeugung in MASP eine fixe Länge. Wenn es seine Position nicht ändern soll, wird es oft durch den Nutzer am Anfangs- und Endpunkt fixiert (siehe auch Abbildung 3.13). Hierdurch wird aber offensichtlich eine Überbestimmtheit hervorgerufen, denn der Abstand zweier fixierter Punkte ist durch die relative Lage der beiden Punkte gegeben. Ein explizites Fixieren eines falschen Abstandsparameterwertes würde sogar zu einer inkonsistenten Überbestimmtheit führen. Erst die Freigabe (Defixierung) des Abstandsparameters würde die Überbestimmtheit aufheben. Der damit verbundene Mehraufwand bei der Modellierung könnte jedoch die Nutzerakzeptanz beeinträchtigen, so dass diese Art von Degeneriertheit durch den Constraint-Solver toleriert werden sollte. □

- Es gibt eine Reihe von Fällen, in denen die Degeneriertheit explizit durch Constraints festgeschrieben ist. Derartige Fälle sollten durch den Constraint-Solver erkannt und behandelt werden.

  Beispiel 3.4: Constraints führen explizit zu einem degenerierten Fall
  Allgemein lässt sich die Position eines Punktes ermitteln, wenn drei Ebenen, in denen er enthalten ist, bekannt sind. In den mit dem Constraint-Solver berechneten Modellen kam es jedoch mehrfach vor, dass die drei Ebenen senkrecht zu einer vierten Ebene standen. In diesem Fall schneiden sich die Ebenen nicht nur in einem Punkt sondern in einer Geraden[9]. Ein solcher Fall ist auch in Abbildung 2.8 gezeigt. Zur Lösung des Problems wurde im Abschnitt 2.6 die Definition spezieller Hinweise vorgeschlagen. □

[9] Dass sie sich (zumindest in konsistenten Modellen) schneiden wird durch die drei „Punkt in Ebene“-Constraints garantiert.

- Beim interaktiven Umgang mit den Modellen kommt es immer wieder vor, dass für bestimmte Positionen oder Parameterwerte Konstruktionen nicht mehr durchführbar sind, z. B. wenn sich zwei Kreise nicht mehr schneiden oder wenn ein Punkt in zwei Geraden liegen soll, die im Laufe der Interaktion parallel geworden sind.
  In solchen Fällen sollte eine andere Freiheitsgradverteilung gefunden werden, die wieder zu einem konstruierbaren Plan führt. Allerdings kann es kombinatorisch viele verschiedene Freiheitsgradverteilungen geben, so dass nur für kleinere oder nahezu vollbestimmte Modelle das Suchen einer nächsten Freiheitsgradverteilung sinnvoll ist. Bei großen bzw. stark unterbestimmten Modellen besteht die Gefahr, dass zu viel Rechenzeit bei der Suche nach einer konstruierbaren Freiheitsgradverteilung verbracht wird und die Arbeitsweise nicht mehr robust ist.
  Ein erster Ansatz zur Lösung des Problems könnten verzweigte Pläne sein, wo sinnvolle alternative Berechnungswege schon im Plan enthalten sind. Die Wahl des Weges könnte durch das Testen von kritischen Parametern vor der Konstruktion erfolgen - oder aber durch das Verzweigen nach einer fehlgeschlagenen Konstruktion. Derartige Tests wären in konstanter Zeit durchführbar und somit dem Zyklus von:

  – Generieren einer weiteren Freiheitsgradverteilung,
  – Erstellen eines entsprechenden Konstruktionsplanes und
  – Versuch der Konstruktion mit diesem Plan

  bezüglich der Zeitkomplexität vorzuziehen. Es wurden jedoch noch keine Algorithmen für die Generierung von verzweigten Plänen untersucht.

- Die Verwendung von numerischen Verfahren ist im Allgemeinen sehr schlecht für die Robustheit eines Constraint-Solvers. Wie Abschnitt 4.2.3 zeigt, ist die Gefahr des Springens zwischen verschiedenen Lösungen aber insbesondere auch die Gefahr der Divergenz der Suche sehr hoch. Der iterativ-konstruktive Ansatz stellt ein spezielles, relativ gut beherrschbares numerisches Verfahren dar, das eingesetzt werden kann, wenn kein rein konstruktiver Lösungsweg existiert, oder wenn dieser nicht gefunden werden kann. In Abschnitt 3.6.3 und in Kapitel 4 wird gezeigt, wie bei iterativen Konstruktionen zum einen die relevanten Nullstellen zu finden sind und zum anderen wie dann aus ihnen die richtige ausgewählt werden kann, so dass ein Springen zwischen verschiedenen Lösungen vermieden wird.
  Bezüglich der Robustheit des Suchalgorithmus sei an dieser Stelle darauf hingewiesen, dass bei der Suche ein stetiger Suchfortschritt gewährleistet sein muss (siehe Abschnitt 3.6.3 und insbesondere die Beispiele für die Gefahr von Endlosschleifen bei der Suche mit einfachen Verfahren).

- Wenn für die Berechnung eines Modells ein rein konstruktiver Lösungsweg existiert, so sollte der Constraint-Solver diesen zur Vermeidung von Iterationen auch finden. Hierfür sind gegebenenfalls Clusteringansätze zu nutzen. Eine Diskussion über Möglichkeiten zur Umsetzung von Clustering im vorgestellten Constraint-Solver wird in Abschnitt 3.7 vorgenommen.

Zusammenfassend lässt sich sagen, dass die Robustheit wesentlich von der Fähigkeit zum Finden einer passenden Lösung abhängt. Sie ist somit wesentlich von der Steuerbarkeit[10] und auch von der Geschwindigkeit[11] der Berechnungen abhängig. Das Erkennen von degenerierten Fällen (und die damit verbundene Vermeidung von langwierigen erfolglosen Berechnungen) bietet eine Möglichkeit die Robustheit des Constraint-Solvers zu erhöhen. Allerdings ist hiermit auch ein nicht unerheblicher Mehraufwand bei der Entwicklung des Constraint-Solvers verbunden.

[10] Beschreibung der passenden Lösung
[11] Eine Suche, die zu lange dauert, wird als Misserfolg bewertet und abgebrochen.

3.1.4 Allgemeingültigkeit der Algorithmen

Der Anspruch einen generischen Constraint-Solver bzw. zumindest einen generischen Constraint-Solver-Kern zu entwickeln wirft die Frage auf, wie allgemein die verwendeten Algorithmen und Datenstrukturen sein müssen. Ohne Zweifel kann die Entwicklung eines *General Problem Solvers*[12] nicht das Ziel sein. Andererseits zeigen Analysen von Veröffentlichungen, dass eine Vielzahl von Constraint-Solvern starken Einschränkungen unterliegt. Beispiele hierfür sind:

- Für Constraints wird stets die Valenz eins angenommen. Dies äußert sich dann z. B. darin, dass von der Anzahl der Constraints gesprochen wird, die notwendig ist, um ein Objekt zu berechnen (u. a. zwei Constraints für einen *2D*-Punkt). Bei einer Erweiterung auf Constraints mit einer größeren Valenz wären insbesondere die Freiheitsgradanalyse- und Planerstellungsalgorithmen anzupassen, was nicht trivial ist.
- Das System kann nur relative Constraints verarbeiten, d. h. es gibt keine Fixierungen und keine Anstiege (Anstiegswinkel von einem Punkt zu einem anderen bezüglich der x-Achse). Hier muss die Applikation die notwendige Positionierung bzw. Ausrichtung des berechneten Modells vornehmen.
- Constraints werden immer nur zwischen zwei Objekten definiert. Dies impliziert konstante Parameter. Auch hier sind bei einer Erweiterung auf freie Parameter insbesondere die Freiheitsgradanalyse- und Planerstellungsalgorithmen anzupassen. Zudem müssen für die Umsetzung eines konstruktiven Ansatzes die Konstruktoren zur Berechnung der entsprechenden Parameter implementiert werden.

Die obigen Einschränkungen gelten nicht für Ficucs. Allerdings weist er in seiner Leistungsfähigkeit noch wesentliche Grenzen bezüglich der Lösung von Gleichungssystemen und der Optimierung auf. Allgemein ist jedoch festzustellen, dass Ficucs bereits jetzt ein vielseitig einsetzbarer Constraint-Solver ist (siehe insbesondere Kapitel 5). Die weitere Entwicklung des Kernes aber auch der Erweiterungsmodule wird das Spektrum seiner möglichen Einsatzfälle entsprechend den jeweiligen Anforderungen kontinuierlich verbreitern.

3.1.5 Implementierbarkeit

Die insbesondere aus Gründen der Geschwindigkeit und Steuerbarkeit bevorzugten direkten Konstruktionen bedingen je nach geplantem Einsatzzweck einen mehr oder weniger hohen Implementierungsaufwand im Constraint-Solver (vgl. Abschnitt 3.4.1). Folgende Ansätze können helfen, den Entwicklungsaufwand dennoch in Grenzen zu halten:

- Trennung der generischen Teile von den speziellen Teilen der Datenstrukturen und vor allem der entsprechenden Algorithmen.

  Generisch sind z. B.:

  - Freiheitsgradanalyse,
  - Nullstellensuche und
  - Planerstellung.

  Speziell, d. h. domain-angepasst, sind z. B.:

  - Objektrepräsentation,
  - Objektkonstruktion und

[12] deutsch: *Allgemeiner Problemlöser*. Der englische Begriff wurde von Newell und Simon geprägt (siehe z. B. [NS63]).

  – geometrische Constraints.

  Auch zu den in *2D* und *3D* verwendeten Parametern gibt es genug Programm-Code, der keine domainspezifische Semantik hat, z. B. der Umgang mit linearen Gleichungen, trigonometrischen Funktionen sowie die Verwaltung von Konstanten[13].

- Die Verwendung eines allgemeinen Berechnungsansatzes, z. B. einer Software zum Lösen linearer und nichtlinearer Gleichungssysteme, könnte die Implementierung des Constraint-Solvers wesentlich vereinfachen, da im Wesentlichen nur die Umsetzung der Constraints in entsprechende Gleichungen zwingend notwendig zu implementieren wäre. Hoffmann, Lomonosov und Sitharam gehen jedoch noch wesentlich weiter und benutzen für die Unterteilung[14] der Gleichungen in Gleichungssysteme den DR-Algorithmus [HLS01a, HLS01b], der einen relativ allgemeinen Clustering-Ansatz realisiert. Sämtliche Berechnungen der Gleichungssysteme laufen jedoch in Fremdsoftware ab, wodurch die Gefahr des Verlustes der Steuerbarkeit sehr groß ist.

Denkbar ist auch eine Kombination beider Ansätze. Die Einbindung weitergehender Clustering-Ansätze in Ficucs würde aber erheblichen Aufwand bedeuten. Andererseits könnte eine einfache Anbindung eines allgemeinen Gleichungslösers an Ficucs ohne größeren Programmieraufwand realisiert werden. Es bestünde dann die Möglichkeit, diesen zur Lösungsfindung (zumindest für nichtzeitkritische Modellberechnungen) einzusetzen, falls der konstruktive bzw. iterativ-konstruktive Ansatz fehlschlägt.

3.1.6 Effizienz der Modellierung

Ein nicht zu unterschätzender Faktor für die Akzeptanz eines Constraint-Solvers ist die Effizienz der Modellierung. Die Notwendigkeit die Lösungsfindung zu steuern und bestimmte degenerierte Fälle zu vermeiden wurde in den vorangehenden Abschnitten bereits erörtert. Hierfür ist jedoch Kenntnis über die Arbeitsweise des jeweiligen Constraint-Solvers nötig, denn nur so können die degenerierten Fälle, die vom Constraint-Solver intern noch nicht beherrscht werden, vermieden werden und nur so kann man geeignete Mittel zur Beschreibung der gewünschten Lösung auswählen. Hilfe beim Erwerb der entsprechenden Kenntnisse kann gegeben werden durch:

- die Dokumentation der Schnittstellen sowie der allgemeinen Arbeitsweise des Constraint-Solvers, inklusive kleiner und größerer Beispiele für günstige und ungünstige Modellierung sowie
- umfangreiche Diagnosemöglichkeiten, die die Arbeitsweise des Constraint-Solvers, insbesondere auch bei Problemen, transparent macht.

Hat man dieses Wissen und beherrscht die Modellierung, so ist die Effizienz der Modellierung aus Sicht des Anwendungsentwicklers vom reinen Programmier- und Debugaufwand[15] abhängig. Hier können insbesondere bei kleineren Modellen sinnvolle Voreinstellungen helfen. Auch hat sich die Bereitstellung von einfachen Datentransferklassen[16] mit oft genutzten Methoden als hilfreich erwiesen.

---

[13] Möglicherweise wird es später Modelle mit speziellen *1D*-Domänen geben, die ein zusätzliches Erweiterungsmodul des Constraint-Solver-Kernes notwendig bzw. sinnvoll machen.

[14] Die geschickte Unterteilung ist wesentlich für die Berechnungsgeschwindigkeit.

[15] engl.: *debug* - deutsch: entwanzen. Im Kontext der Programmentwicklung eine gebräuchliche Bezeichnung für das Finden von Fehlern in Programmen. Der Begriff *bug* bedeutet dementsprechend „Fehler“ (ursprünglich: Käfer oder Wanze).

[16] Ficucs hat eine C++-Schnittstelle, in der der Datentransfer über spezielle Datentransferklassen definiert ist. Grundsätzlich sind diese Klassen abstrakt, so dass Programmierer z. B. nicht zu bestimmten Formen der Datenhaltung gezwungen werden. Um den Constraint-Solver an einer Anwendung anzubinden ist ihm durch Überladung der Lese- und Schreibmethoden für die Datentransferobjekte der Zugriff auf die spezielle Art und Weise der Datenhaltung der Applikation zu ermöglichen. Mitunter reicht (insbesondere in kleineren Projekten) jedoch eine einfache Form der Datenspeicherung. Hierfür werden in der Constraint-Solver-Schnittstelle Klassen angeboten, wodurch dem Anwendungsprogrammierer Entwicklungsaufwand abgenommen wird.

Aus Sicht eines Bedieners soll die Applikationssoftware den Constraint-Solver mit seinen Eigenheiten möglichst verbergen. Die automatische Objekt- und Constraint-Generierung in MASP (entsprechend dem jeweils vom Nutzer in das Modell eingefügten Symbol) oder in COSMOS (z. B. die automatische Erzeugung von „Punkt in Ebene“-Constraints für Polyeder) ist ein wichtiger Schritt in diese Richtung. Noch weitergehende Arbeiten wurden in Zusammenhang mit dem KONNI-Projekt gemacht (siehe Abschnitt 5.3.2), wo ausgehend von einer Punktmenge automatisch die Polygone der konvexen Hülle eines Modells, die Parameter und die Hilfsebenen (z. B. für symmetrische Modelle) sowie Constraints und Hinweise zum Erhalt der topologischen Eigenschaften und Anbindung der Parameter an die Geometrie erzeugt wurden.

3.2 Transformation des Constraint-Netzes

Das durch eine Applikation oder direkt durch einen Nutzer definierte Constraint-Netz lässt sich mitunter für die Bearbeitung im Constraint-Solver optimieren. Um nicht das externe (originale) Constraint-Netz ändern zu müssen, wird eine constraint-solver-interne Repräsentation des Constraint-Netzes aufgebaut. Die wesentlichen Schritte, die bei der Transformation vom externen zum internen Constraint-Netz vorgenommen werden, sind in den folgenden Abschnitten beschrieben.

3.2.1 Entfernung doppelter Constraints

Die Notwendigkeit der Entfernung von doppelten Constraints ergibt sich daraus, dass:

- einerseits das Auftreten von Dopplungen im internen Constraint-Netz für die Algorithmen aus unterschiedlichen Gründen nicht erwünscht ist und
- dass es andererseits für Applikationsentwickler und Nutzer wünschenswert sein kann, dass Dopplungen im externen Constraint-Netz zulässig sind.

Das doppelte (bzw. allgemeiner das mehrfache) Auftreten von Constraints ist insbesondere für die Freiheitsgradanalyse aber auch für die Erstellung einer Konstruktionssequenz problematisch. Zum einen würden bei der Freiheitsgradanalyse in den referenzierten Objekten zu wenig Freiheitsgrade verbleiben, zum anderen könnten Konstruktionen im Plan permanent degeneriert sein.

Beispiel 3.5: Doppelte Definition eines “Abstand von Punkt zu Punkt”-Constraints (lokale Auswirkung)
Gegeben sei ein fixer Punkt F und ein Punkt A, sowie ein Constraint C_{1a}, der den Abstand von F zu A als einen bestimmten konstanten Zahlenwert definiert. Die Freiheitsgradanalyse ergibt, dass der Punkt A durch C_{1a} nur noch einen Freiheitsgrad hat. Wird jedoch ein zu C_{1a} gleicher Constraint C_{1b} definiert, so würde die Freiheitsgradanalyse zweimal eine Valenz von eins berücksichtigen und fälschlicherweise feststellen, dass in A kein Freiheitsgrad verbleibt.
Sollten bei der Aufstellung des Konstruktionsplanes C_{1a} und C_{1b} in einer Operation (hier die Konstruktion eines Punktes durch das Schneiden von zwei Kreisen) Verwendung finden, so wäre diese permanent degeneriert. Im vorliegenden Beispiel wären beide Kreise stets identisch und es könnte eine Projektion der alten Werte von A auf den Kreis vorgenommen werden. Am Verhalten würde sich somit (lokal) nichts ändern. Es würde jedoch unnötig Rechenzeit für den Test auf Gleichheit verbraucht werden. □

Die unerwünschte Wirkung mehrfach definierter Constraints muss jedoch nicht lokal begrenzt sein. Es sind auch Auswirkungen auf andere Objekte möglich.

Beispiel 3.6: Doppelte Definition eines “Abstand von Punkt zu Punkt”-Constraints (globale Auswirkung)
Gegeben sei ein fixer Punkt F und ein Punkt A, sowie ein Constraint C_{1a}, der den Abstand von F zu A

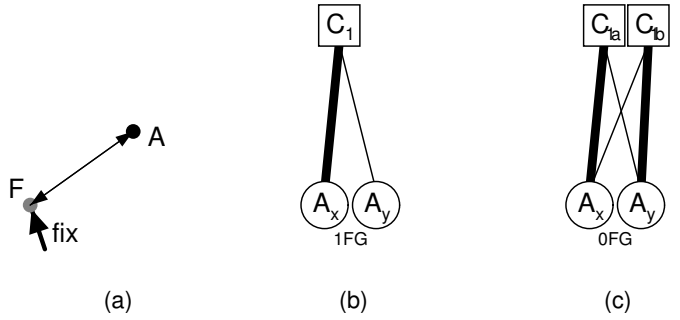


Abbildung 3.3: Beispiel für die lokale Auswirkung der doppelten Definition eines Abstands-Constraints. (a) Modell mit fixem Punkt F, bei dem sich A mit einem Freiheitsgrad um F bewegen kann. (b) Darstellung des entsprechenden Graphen zur Freiheitsgradanalyse, der zeigt, dass A noch einen Freiheitsgrad hat. (c) Es wird gezeigt, dass A nach der Definition des zweiten Abstands-Constraints zu F fälschlicherweise formal keinen Freiheitsgrad mehr hat.

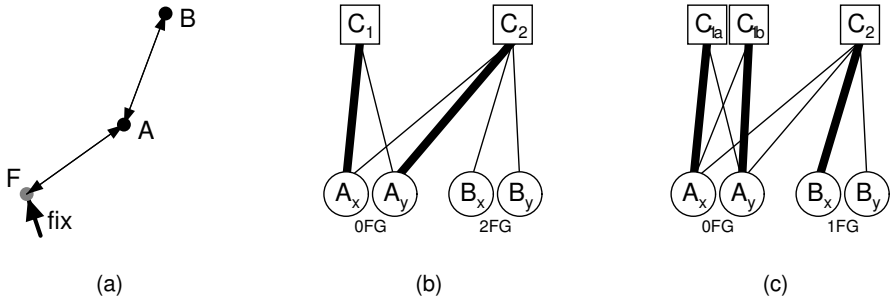


Abbildung 3.4: Beispiel für die globale Auswirkung der doppelten Definition eines Abstands-Constraints. (a) Modell mit fixem Punkt F, bei dem B zwei Freiheitsgrade hat und Punkt A durch die Abstände zu F und B bestimmt sei. (b) Darstellung des entsprechenden Graphen zur Freiheitsgradanalyse, der die Zuordnung der Freiheitsgrade zu A und B veranschaulicht. (c) Es wird gezeigt, dass B nach der Definition des zweiten Abstands-Constraints zwischen F und A fälschlicherweise formal nur noch einen Freiheitsgrad hat.

als einen bestimmten konstanten Zahlenwert definiert. Außerdem sei ein Punkt B mit einem konstanten Abstand zu A definiert (Constraint C_2). Ist Punkt B hoch priorisiert (z. B. wenn er per Mauszeiger verschoben werden soll), so stellt die Freiheitsgradanalyse fest, dass B mit zwei Freiheitsgraden (also frei) bewegt werden kann und A voll bestimmt ist. Wird jedoch ein zu C_{1a} gleicher Constraint C_{1b} definiert, stellt die Freiheitsgradanalyse fälschlicherweise fest, dass in B nur ein Freiheitsgrad verbleibt. □

Es gibt verschiedene Situationen, die zur Mehrfachdefinition eines Constraints führen können, so sind z. B.:

- die direkte Eingabe durch einen Nutzer,
- die automatische Generierung sowie
- Mischformen des Zusammenspiels von Nutzer und Applikation denkbar.

Es ist durchaus möglich, und in einigen Anwendungen auch sinnvoll, den Nutzer über das mehrfache Auftreten des Constraints zu informieren. Unter Umständen würde dies jedoch seinen Arbeitsablauf stören, so dass eine interne Behandlung (an dieser Stelle ein Entfernen) der Dopplung

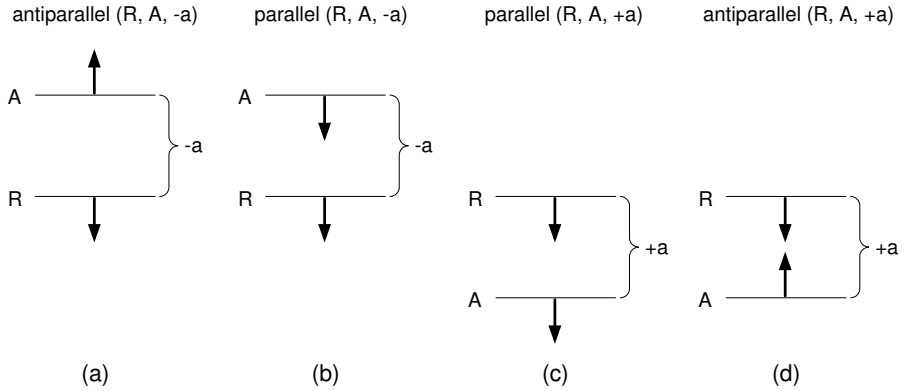


Abbildung 3.5: Die vier Fälle der Positionierung einer Ebene A (anti-)parallel zu einer Referenzebene R. (b) und (c) stellen die Parallelität im engeren Sinne dar, (a) und (d) hingegen die Antiparallelität, mit der die Entgegengerichtetheit der Normalen zum Ausdruck gebracht wird. Das Vorzeichen des Abstandsparameters bestimmt, ob die Ebene A vor (+) oder hinter (-) der Referenzebene R positioniert ist.

zweckmäßig ist. Auch das automatische Generieren von Constraints kann vereinfacht werden, wenn Dopplungen automatisch entfernt werden.

Beispiel 3.7: Doppelte Definition von "Punkt in Ebene"-Constraints
Bei der constraint-basierten Repräsentation von Polyedern sind „Punkt in Fläche"-Constraints zu definieren, die garantieren, dass alle Punkte eines Polygons in einer Ebene bleiben. Werden die Polyeder als B-Rep[17] repräsentiert, so wäre es denkbar, dass für jeden Punkt P_i entsprechend den „Punkt P_i begrenzt Kante K_j"-Relationen sowie den „Kante K_j begrenzt Fläche F_l"-Relationen jeweils die Inzidenz-Constraints zwischen P_i und allen Flächen F_l definiert werden. Bei einer derartigen Vorgehensweise wären jedoch alle „Punkt in Ebene"-Constraints doppelt definiert, denn jeder P_i ist in zwei Kanten K_j, die eine Ebene F_l begrenzen, enthalten. Das automatische Entfernen der Dopplungen im Constraint-Solver hilft somit den Implementierungsaufwand in der Applikation zu vermindern. □

Zur Implementierung der Erkennung von Dopplungen ist der Einsatz binärer (bzw. besser digitaler) Bäume sinnvoll (vgl. Abschnitt 3.6.3). Hierfür sind Schlüssel aus den IDs[18] der externen Objekte sowie dem Constraint-Typ zu bilden. Bei der Bildung der Schlüssel sollte Folgendes berücksichtigt werden:

- Referenziert ein Constraint n Objekte eines Typs, deren Position innerhalb des Constraints grundsätzlich unerheblich ist, so sollten die IDs dieser Objekte für die Schlüsselbildung in eine eindeutige Reihenfolge gebracht werden (z. B. durch Sortierung der als Ganzzahlen interpretierten ID-Werte). Alternativ hierzu könnten $n!$ Schlüssel gebildet werden.
- Bei parameterbehafteten Constraints kann es sinnvoll sein, Parameter nicht im Schlüssel zu verwenden.

  Beispiel 3.8: Umgang mit doppelten Abstands-Constraints zwischen Punkten
  Der Abstand zwischen zwei Punkten A und B sei mittels Constraint C_1 als freier Parameter d definiert. Aus den IDs der Punkte A und B wird ein Schlüssel gebildet[19], unter dem der Constraint C_1 gespeichert wird. Bei Auftreten eines Constraints C_2 der den Abstand zwischen B und A als

[17] engl.: *Boundary Representation*, Beschreibung der Umhüllung eines Objektes. Siehe z. B. [FvDFH97] oder [BM01].
[18] ID ist eine Abkürzung für Identifikator. Identifikatoren dienen zur eindeutigen Identifizierung von Objekten. Denkbar ist z. B. die Verwendung von Bezeichnern oder von Zeigern auf die Adresse im Speicher.
[19] Zum Beispiel die Verkettung der Bitmuster der Zeiger auf A und B. Denkbar wäre, dass der Zeiger, der im Speicher weiter nach hinten zeigt, stets zuletzt ins Bitmuster eingeht.

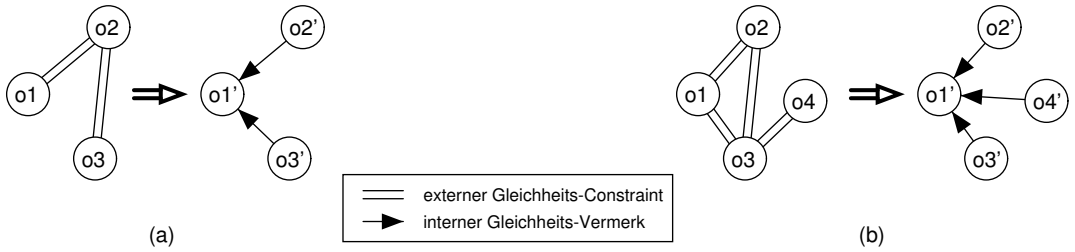


Abbildung 3.6: Transformation von Gleichheits-Constraints in interne Gleichheits-Vermerke. (a) Menge von drei gleichen Objekten. (b) Menge von vier gleichen Objekten, wobei ein Gleichheits-Constraint redundant ist. o_1 dient jeweils als Referenzobjekt.

fixierten Parameter f definiert, ist es nun möglich über den Schlüssel aus A und B den Constraint C_1 zu ermitteln. Ein Vergleich der Parameter ermöglicht es zu erkennen, dass hier keine Überbestimmtheit vorliegt (beide Parameter müssten fix sein). Im geschilderten Fall würde statt C_2 die Gleichheit der Parameter d und f im Constraint-Solver vermerkt werden (siehe Abschnitt 3.2.2).□

- Es ist nicht sinnvoll Parallelität und Anti-Parallelität als unterschiedliche Constraint-Typen anzusehen (sowohl in *2D* als auch in *3D*, gleiches gilt auch für Symmetrie und Anti-Symmetrie). Die in der Anti-Parallelität zum Ausdruck gebrachte Entgegengesetztheit lässt sich leicht in einer boolschen[20] Variable des Parallelitäts-Constraints speichern. Auf diese Art lassen sich schnell widersprüchliche Constraint-Definitionen bezüglich Gleich- und Entgegengerichtetheit erkennen (vgl. Abbildung 3.5 zur Definition von (Anti-)Parallelität).

3.2.2 Ausnutzung von Gleichheits-Constraints

Beim Auftreten von Gleichheits-Constraints lässt sich das Constraint-Netz vereinfachen, indem statt der gleichen Objekte nur ein Referenzobjekt o_r in das interne Constraint-Netz aufgenommen wird (ähnlich wie in [Bar87, Abschnitt 2.5.1]). Es seien durch n binäre Gleichheits-Constraints[21] eine Menge aus g gleichen Objekten O_g definiert worden, wobei $n \geq g-1$ ist. Der Trivialfall liegt für $n = 1$ und $g = 2$ vor. Es ist jedoch auch denkbar, dass es zu größeren Mengen gleicher Objekte kommt. Beispiele hierfür sind in Abbildung 3.6 gegeben. Während der Transformationsphase geschieht Folgendes:

- Alle Constraints, die zu einem Objekt $o \in O_g$ definiert sind, werden auf das Referenzobjekt o_r übertragen. Hierbei sind für einige Constraints Besonderheiten zu beachten, z. B. bei der Gleichheit von Richtungen.

  Beispiel 3.9: Transformation des Constraint-Netzes bei Gleichheit von Richtungen
  Die Gleichheit von Richtungen kann durch einen Parallelitäts-Constraint (er definiert die Gleichheit im engeren Sinne) oder durch einen Anti-Parallelitäts-Constraint definiert werden. Abbildung 3.7 zeigt, dass bei Parallelität ein simples Ersetzen ausreicht. Bei Anti-Parallelität muss hingegen ein Hilfsparameter erzeugt werden[22]. □

- Tritt eine Fixierung auf, so wird sie auf das Referenzobjekt o_r übertragen. Mehrere Fixierungen entsprechen einer lokalen Überbestimmtheit. Sind die Werte der fixierten Objekte

[20] Sie kann nur den Wert „wahr“ oder „falsch“ annehmen.
[21] d. h. es wird stets die Gleichheit zwischen zwei Objekten definiert
[22] Es ist zu beachten, dass z. B. bei der Definition von B-Reps Anti-Parallelitäts-Constraints teilweise häufiger vorkommen als Parallelitäts-Constraints, so sind es bei einem Würfel drei Anti-Parallelitäts-Constraints, die sicherstellen, dass alle sechs Würfelnormalen nach außen bzw. innen zeigen.

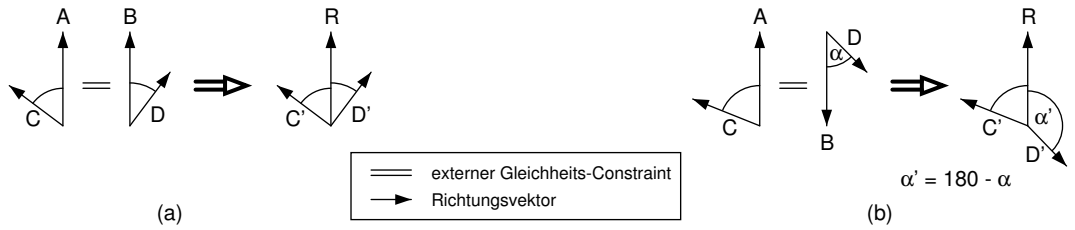


Abbildung 3.7: Transformation von Winkel-Constraints auf Basis von Gleichheits-Constraints. (a) Bei Parallelität der Richtungen reicht die Ersetzung. (b) Bei Anti-Parallelität der Richtungen muss ein Hilfswinkel sowie ein entsprechender Constraint eingefügt werden.

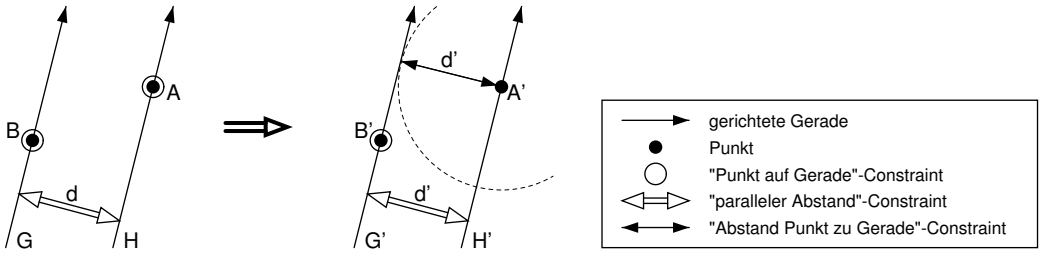


Abbildung 3.8: Transformation eines „Punkt auf Gerade“-Constraints auf Basis eines „paralleler Abstand“-Constraints.

gleich, so ist die Überbestimmtheit konsistent und die Werte können zur Initialisierung der Werte des Referenzobjektes verwendet werden. Treten bei einer mehrfachen Fixierung jedoch ungleiche Werte auf, so ist es unmöglich, das Constraint-Netz konsistent zu machen.

- Ist keines der Objekte fixiert, so werden die Werte des am höchsten priorisierten Objektes verwendet.

Die nicht im internen Constraint-Netz gespeicherten Gleichheiten werden in einer gesonderten Datenstruktur vermerkt. Diese wird nach jeder Berechnung des Modells durchlaufen, um die berechneten Werte der Referenzobjekte an die zu ihnen gleichen Objekte zu übertragen. Hierbei werden auch spezielle Fälle wie entgegengesetzte Richtungen berücksichtigt.

3.2.3 Ausnutzung spezieller Constraints

Um möglichst einfache (direkte) Konstruktionen vornehmen zu können, ist es sinnvoll, auch andere Constraints als die Gleichungs-Constraints für Transformationen auszunutzen. Ein Beispiel hierfür seien „paralleler Abstand“-Constraints, die zwischen zwei Geraden in der euklidischen Ebene oder zwei Ebenen im euklidischen Raum definiert sein könnten.

Beispiel 3.10: Transformation mittels “paralleler Abstand”-Constraints
Ein Punkt A liege auf der Geraden H und ein Punkt B auf der Geraden G. Die beiden Geraden G und H seien zueinander parallel mit einem Abstand d (siehe Abbildung 3.8). Wären A, B und d gegeben, so würde kein Konstruktionsplan gefunden werden, denn für keine der beiden Geraden steht (lokal) genug Information zur Berechnung bereit. Erst die in Abbildung 3.8 gezeigte Transformation sorgt dafür, dass die Constraints auf ein Objekt (hier die Gerade G) konzentriert werden. Bei bekannten Punkten A und B

kann hier zunächst G und dann unter Nutzung des „paralleler Abstand“-Constraints[23] auch H konstruiert werden[24]. □

Wichtig bei derartigen Transformationen ist, dass kein Informationsverlust auftritt. In Beispiel 3.10 heißt das, dass der „Abstand Punkt zu Gerade“-Constraint auch das Vorzeichen des Abstandsparameters auswerten muss.

3.2.4 Zusammenfassende Betrachtung zu Transformationen

Die in den vorhergehenden Abschnitten gezeigten Transformationen stellen nur einen Bruchteil der grundsätzlich möglichen Transformationen dar. Letztlich sind diese vergleichbar mit symbolischen Umformungen hin zu einer Normalform (vgl. [Brü87], wo ein entsprechendes regelbasiertes System beschrieben ist). An einer umfassenderen Umsetzung ist jedoch Folgendes problematisch:

- Die Implementierung in einer prozeduralen/objektorientierten Sprache wie C++ ist aufwändig, so dass generell eine regelbasierte Handhabung der Transformationen wünschenswert ist. Dies darf jedoch nicht zu einer wesentlichen Beeinträchtigung der Verarbeitungsgeschwindigkeit führen.
- Das letztliche Ziel der Transformationen ist unbekannt, denn der aktuelle Interaktionspunkt ist nicht Bestandteil des Modells. Um für unterschiedliche potentielle Interaktionspunkte sinnvolle Transformationen durchführen zu können, müssten ggf. alternative Constraints genutzt werden.
- Es sind Konzepte wie die Priorisierung von Objekten und Constraints zu berücksichtigen, so dass Über- und Unterbestimmtheit geeignet behandelt werden.
- Das externe Constraint-Netz unterliegt u. U. stetigen lokalen Änderungen (z. B. bei einer automatischen Strukturoptimierung). Für die Anwendung der Transformationen wären deshalb inkrementelle Ansätze wünschenswert.

Aus den beschriebenen Gründen wurden nur einige der möglichen (jedoch alle der in den vorhergehenden Abschnitten beschriebenen) Transformationen implementiert.

3.3 Freiheitsgradanalyse

Eine tiefgreifendere Freiheitsgradanalyse, die auch lokale Über- bzw. Unterbestimmtheiten erkennen kann, wird typischerweise graphbasiert durchgeführt. Zur Graphentheorie gibt es eine Vielzahl von Literaturquellen im Allgemeinen und insbesondere auch zu den für die Freiheitsgradanalyse interessanten bipartiten Graphen (u. a. [HK73], [MV80], [ABMP91], [MTYS94], [ADH98]). In diesem Abschnitt wird auf die Anwendbarkeit unterschiedlicher Ansätze sowie die Umsetzung geeigneter Erweiterungen eingegangen. Zunächst erfolgt jedoch eine Einführung in die verwendeten Begriffe und Symbole.

---

[23] Alternativ hierzu wäre z. B. auch ein „Gerade tangential zu Kreis“-Constaint nutzbar, wobei A den Mittelpunkt und d den Radius des Kreises definieren. Dann müssten jedoch auch Kreise mit vorzeichenbehaftetem Radius zulässig sein und zudem müsste das Vorzeichen bei der Lösungsfindung ausgewertet werden (Lage des Kreises rechts oder links von der Geraden).

[24] Man beachte, dass in Abb. 3.8 nur eine von 2 möglichen Lösungen für ein gegebenes vorzeichenbehaftetes d dargestellt wurde.

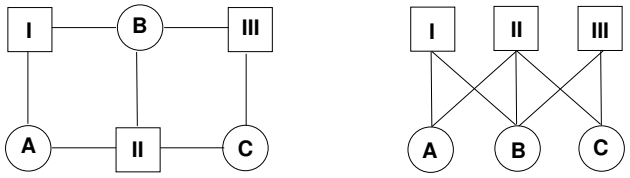


Abbildung 3.9: Der gleiche Graph in unterschiedlichen Darstellungen: (a) Ohne Überschneidung von Kanten, diese Darstellung lässt die Beziehungen gut erkennen. (b) Als zweigeteilte Knotenmenge, wodurch mögliche Zuordnungen besser erkennbar sind.

3.3.1 Begriffe und Symbole

- Bipartiter Graph

  Die Freiheitsgradanalyse verwendet einen bipartiten Graph[25], d. h. einen Graphen, dessen Knoten in zwei Untermengen aufgeteilt sind, innerhalb derer es keine Verbindungen gibt, denn Objekte sind nicht mit Objekten verbunden und Constraints nicht mit anderen Constraints. Während im (durch den Constraint-Manager verwalteten) Constraint-Netz diese Zweigeteiltheit nur logisch vorliegt, wird sie in den Datenstrukturen und Algorithmen der Freiheitsgradanalyse explizit berücksichtigt[26]. Zudem veranschaulicht ein bipartiter Graph Zuordnungsprobleme, wie sie in der Freiheitsgradanalyse von Interesse sind, deutlicher (Abbildung 3.9).

- Kardinalitäten K bzw. F und V:

  Die Kardinalitäten K von Objekten und Constraints hängen vom jeweiligen Typ ab. Die Kardinalität eines Objektes F ist sein maximal möglicher Freiheitsgrad (z. B. $F = 2$ für einen Punkt in der Ebene). Die Kardinalität eines Constraints V ist die seinem Typ entsprechende Valenz (z. B. $V = 3$ für die Gleichheit zweier Punkte im Raum).

- Freiheitsgrad f:

  Der in unterbestimmten Fällen (nach Abzug der auf das Objekt wirkenden Valenzen) auftretende Restfreiheitsgrad wird mit f bezeichnet.

- Zuordnung, Paarung bzw. Freiheitsgradverteilung Z:

  Eine Zuordnung oder Paarung Z ist eine Abbildung der Kardinalitäten von Constraints auf die der Objekte. Da im unterbestimmten Fall jede Zuordnung auch die Verteilung der in den Objekten verbleibenden Freiheitsgrade f festlegt, wird sie auch als Freiheitsgradverteilung bezeichnet.

- Fluss-basierte Ansätze

  Die Idee, Algorithmen zu Flüssen in Netzwerken (engl.: *network flow*) auf Zuordnungen in bipartiten Graphen zu übertragen, wird in der Literatur oft beschrieben, z. B. [McH90, HLS98]. Typisch ist hierbei das Hinzufügen einer zentralen Quelle, die mit allen Constraint-Knoten verbunden ist und eine zentrale Senke, die mit allen Objekt-Knoten verbunden ist (Abbildung 3.10). Die beschriebenen Algorithmen unterstützen aufgrund ihrer Herkunft die Übertragung von Valenzen, die größer als eins sind pro Verbindung zwischen einem Constraint und einem Objekt. Das Constraint-Netz ist dementsprechend direkt auf den bipartiten Graphen abbildbar.

[25] wird in der deutschsprachigen Literatur auch als paarer Graph bezeichnet
[26] D.h. das Erreichen einer günstigeren Datenspeicherung sowie Berechnungszeit ausgenutzt.

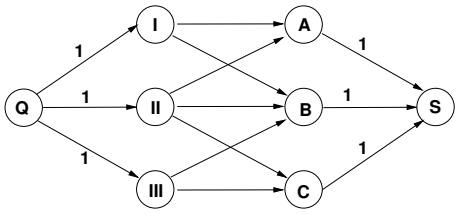


Abbildung 3.10: Graph, auf den ein Netzwerk-Fluss-Algorithmus angewendet werden kann. Der Graph gehört zu dem in Abbildung 3.9 gezeigten Beispiel, wobei zwei Knoten für die Quelle und die Senke des zu untersuchenden Flusses sowie entsprechende Kanten hinzugefügt wurden. Als Kardinalität der Constraints und Objekte wurde jeweils eins angenommen und in Form eines Gewichts an den hinzugefügten Kanten vermerkt. Die Ausrichtung der Kanten erfolgt von der Quelle Q zur Senke S .

- Matching-basierte Ansätze[27]

  Im Gegensatz zu den fluss-basierten Ansätzen wird bei matching-basierten Ansätzen [McH90, LM98, LIJ97] pro Verbindung zwischen Constraints und Objekten nur eine Valenz zugeordnet. Das heißt, dass für einen Constraint aus einem Constraint-Netz in einem entsprechenden bipartiten Graphen n einzelne Knoten anzulegen sind, wenn seine Kardinalität $V = n$ beträgt. Gleiches gilt für Objekte der Kardinalität $F = m$. Außerdem sind für jede Verbindung zwischen einem Constraint (mit $V = n$) und einem Objekt (mit $F = m$) $m * n$ Verbindungen anzulegen. Da für die in dieser Arbeit erörterten Problemstellungen sowohl die Kardinalitäten als auch die Anzahl der durch die Constraints jeweils referenzierten Objekte klein ist (≤ 4), kann von einer konstanten Verschlechterung des Laufzeitverhaltens ausgegangen werden[28].

- Perfekte oder auch vollständige Paarung (engl.: *perfect match*)

  Eine Zuordnung, in der alle Knoten berücksichtigt sind, wird als perfekte Paarung bezeichnet. Da stets ein Knoten nur genau einem anderen Knoten zugeordnet werden kann, muss hierfür die Anzahl der Knoten in den beiden Knotenmengen für Objektfreiheitsgrade und Valenzen gleich sein, d. h. die Summe der Objektfreiheitsgrade darf sich nicht von der Summe der Valenzen unterscheiden. Die perfekte Paarung kann also nur bei vollbestimmten Modellen auftreten.

- Maximale Paarung (engl.: *maximal matching*)

  Eine maximale Paarung stellt ein lokales Maximum der Anzahl der Kanten einer Zuordnung dar, d. h. es kann keine Kante mehr hinzugefügt werden, ohne die bisherigen Kanten zu verändern oder ohne die Bedingung, dass jeder Knoten nur in einer Kante sein darf, zu verletzen (Abbildung 3.11a).

- Größte Paarung (engl.: *maximum matching*)

  Eine maximale Paarung mit der größt möglichen Kantenanzahl wird als größte Paarung bezeichnet. Sie stellt somit ein globales Maximum der Anzahl der Kanten einer Zuordnung dar. Bei der Anwendung zur Freiheitsgradanalyse heißt das, dass es keine Zuordnung gibt, in der mehr Valenzen berücksichtigt werden (Abbildung 3.11c).

- Vollständige Paarung bzgl. Valenzen bzw. Freiheitsgraden

[27] engl.: *matching* heißt soviel wie zusammenpassend, angepasst, übereinstimmend

[28] Eine übermäßig große Anzahl von Verbindungen wird nur bei sehr wenigen ausgezeichneten Objekten auftreten, z. B. bei Ebenen von Koordinatensystemen, in denen viele Objekte durch entsprechende Constraints zu diesen Ebenen positioniert sind oder bei im gesamten Modell genutzten Variablen (vorstellbar ist z. B. eine Skalierungsvariable für alle Maße).

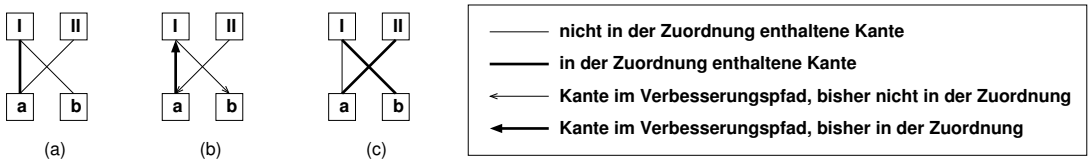


Abbildung 3.11: Wirkungsweise eines Verbesserungspfades. Die in (a) dargestellte maximale Paarung (lokales Maximum) kann über den Verbesserungspfad in (b) zu einer größten Paarung (globales Maximum) in (c) überführt werden.

Da perfekte Paarungen in interaktiv definierten Modellen zumindest während der Definitionsphase eher selten auftreten, werden für die Freiheitsgradanalyse die Begriffe:

1. vollständige Paarung bzgl. Valenzen,
   d. h. alle Valenzen sind in der Zuordnung enthalten, alle Constraints können also in die Berechnungen einfließen, es liegt ein voll- oder unterbestimmtes Modell vor, und
2. vollständige Paarung bzgl. Freiheitsgraden,
   d. h. alle Freiheitsgrade werden in der Zuordnung benutzt, es liegt somit Voll- oder Überbestimmtheit vor,

eingeführt.

- Verbesserungs- oder auch Erweiterungspfad (engl.: *augmenting path*)

  In Graphen können zwischen Knoten Pfade definiert werden. Eine spezielle Art von Pfaden in bipartiten Graphen sind sogenannte Verbesserungspfade, die zeigen, wie sich eine maximale Paarung erweitern lässt.

  Beispiel 3.11: Wirkungsweise eines Erweiterungspfades anhand der Analyse zweier linearer Gleichungen
  Es seien folgende zwei Gleichungen gegeben.

$$\begin{array}{rrcrcl} I: & a & + & b & = & 3 \\ II: & & & a & = & 7 \end{array} \tag{3.1}$$

  Nach der Zuordnung von I auf a kann II nicht mehr zugeordnet werden, denn a, der einzige Freiheitsgrad dem II zugeordnet werden kann, ist schon durch I belegt (Abbildung 3.11a). I wird deshalb über einen Verbesserungspfad nach b umgeleitet (Abbildung 3.11b). Das lokale Maximum kann dadurch wieder verlassen werden (Abbildung 3.11c). Verbesserungspfade führen von einem ungenutzten Knoten der einen Knotenmenge zu einem ungenutzten Knoten der anderen Knotenmenge (hier von II nach b bzw. umgekehrt). Dabei wechseln sich Kanten, die nicht in der Zuordnung enthalten sind, mit Kanten, die enthalten sind, ab. Beim Umschalten (engl.: *path switching*) wird für alle Kanten des Erweiterungspfades der Zustand des Enthaltenseins invertiert. Da der Verbesserungspfad zunächst aus n in der Paarung enthaltenen Kanten und aus $n+1$ nicht in der Paarung enthaltenen Kanten besteht, ist durch das Invertieren sichergestellt, dass sich die Zahl der Kanten in der Paarung von n auf $n+1$ erhöht. □

- Umleitungspfad

  Umleitungspfade führen von einem in der Zuordnung benutzten Knoten der einen Knotenmenge zu einem unbenutzten Knoten der gleichen Knotenmenge. Sie zeigen, wie sich eine Paarung ändern lässt, ohne die Anzahl der an der Paarung beteiligten Knoten zu verändern. Wie bei den Verbesserungspfaden erfolgt auch hier ein Umschalten der im Pfad enthaltenen Kanten. In den unten beschriebenen Algorithmen werden Umleitungspfade bei der Ermittlung der nächsten größten Paarung benutzt. Abbildung 3.12 zeigt dazu ein Beispiel.

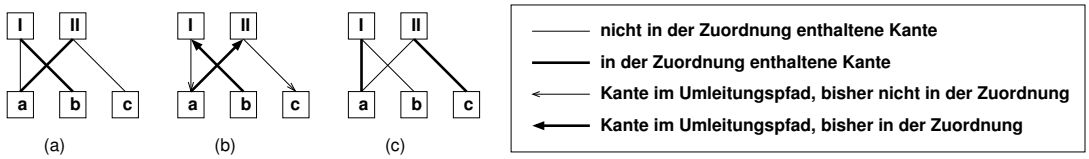


Abbildung 3.12: Wirkungsweise eines Umleitungspfades. Die in (a) dargestellte größte Paarung (globales Maximum) kann über den Umleitungspfad in (b) zu einer anderen größten Paarung in (c) überführt werden.

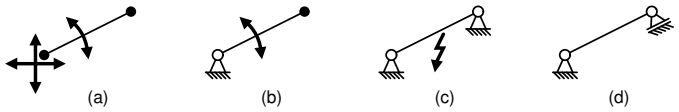


Abbildung 3.13: Fixierung eines Getriebegliedes (*2D*). (a) Nach der Generierung hat ein Getriebeglied drei Freiheitsgrade, zwei Positionsfreiheitsgrade und einen Rotationsfreiheitsgrad. (b) Wenn ein Endpunkt fixiert wird, dann kann sich das Getriebeglied noch um ihn drehen, es hat also noch einen Rotationsfreiheitsgrad. (c) Bei Fixierung des zweiten Endpunktes wird das Modell überbestimmt, denn der Abstand der beiden Endpunkte ist sowohl durch den fixierten Längenparameter des Getriebegliedes (und einen entsprechenden Constraint) als auch durch die beiden fixierten Punkte definiert. (d) Würde dem zweiten Endpunkt nur ein Freiheitsgrad genommen werden, so wäre das fixierte Getriebeglied vollbestimmt. Eine mögliche Alternative zu (d) wäre die Bereitstellung eines neuen Symbols zur Festeinspannung von Getriebegliedern oder aber die Defixierung des Längenparameters (durch den Nutzer oder die Software, die hierzu den in (c) gezeigten Fall jedoch erst erkennen muss).

- Gewichte vs. Prioritäten

  Insbesondere im Zusammenhang mit den Flüssen in Netzwerken können für Kanten Gewichte definiert werden. Diese Wichtungen dienen bei der Freiheitsgradanalyse zur Modellierung der Kardinalitäten für Freiheitsgrade und Valenzen. Eine Priorisierung wird so jedoch nicht erreicht. In den hier vorgestellten Ansätzen wird die Priorität von Objekten bzw. Constraints über die Position innerhalb des entsprechenden Knotenvektors modelliert.

- Gruppen

  Um eine für die Freiheitsgradanalyse angemessene Arbeitsweise der matching-basierten Algorithmen zu erreichen, wird die Zusammenfassung von Freiheitsgraden zu Gruppen unterstützt. Eine typische Anwendung ist, dass die Freiheitsgrade eines Objektes in einer Gruppe sind. Denkbar wäre aber auch, die Freiheitsgrade mehrerer gleichberechtigter Objekte zu einer Gruppe zusammenzufassen[29].

3.3.2 Probleme beim Einsatz der Freiheitsgradanalyse

Eine ganze Reihe von Veröffentlichungen beschäftigt sich mit Algorithmen zur Freiheitsgradanalyse und ihrer Anwendung in Constraint-Solvern, z. B. zur Dekomposition der Constraint-Netze oder (wie hier) zur Bestimmung unabhängiger Objekte. Im praktischen Einsatz kann bei Anwendung der in den Veröffentlichungen beschriebenen Algorithmen jedoch eine Vielzahl von Problemen auftreten. In diesem Abschnitt sollen einige dieser Probleme erörtert werden. Sie zeigen, dass die

[29] Zum Beispiel ist es im Allgemeinen unerheblich, ob der Parameter p oder der Parameter h als frei erkannt wird, wenn zwischen beiden ein Constraint $p = 2 * h$ definiert ist. Ausnahme: bei der Initialisierung des Modells durch die Applikation erfüllen die aktuellen Werte von p und h den Constraint nicht ($p \neq 2 * h$).

Freiheitsgradanalyse stets nur in begrenztem Rahmen[30] einsetzbar ist bzw. dass mit dem Auftreten von Fehleinschätzungen[31] gerechnet werden muss.

- Beim interaktiven Modellieren kommt es schnell zu einer lokalen Überbestimmtheit. Eine häufig auftretende Ursache ist das Fixieren von Objekten, zwischen denen Constraints definiert sind. Zur Verdeutlichung sei hier ein Beispiel aus der Modellierung von Mechanismen aufgeführt.

  Beispiel 3.12: Überbestimmtheit beim Fixieren eines Getriebegliedes
  Abbildung 3.13(b) zeigt ein Getriebeglied, das an einem Ende fixiert wurde. Wenn das Getriebeglied eine feste Länge hat (was nach seiner Erzeugung stets der Fall ist), kann sich das andere Ende noch auf einer Kreisbahn bewegen. Fixiert der Nutzer nun den zweiten Punkt, um diese Bewegungsmöglichkeit einzuschränken, so kommt es zu einer Überbestimmung. Der Abstand zwischen den beiden Punkten ist durch die Fixierungen festgelegt. Er kann nur noch mit dem ebenfalls fixierten Abstandsparameter verglichen werden. □

  Algorithmen zur Freiheitsgradanalyse müssen für praktische Anwendungen mit derartigen Überbestimmtheiten umgehen können.

- Die bisher in der Literatur beschriebenen Algorithmen berücksichtigen keine Prioritäten. Insbesondere beim konzeptuellen Modellieren treten jedoch stark unterbestimmte Modelle auf, in denen die Verteilung der verbleibenden Freiheitsgrade sinnvoll über Prioritäten gesteuert werden kann. Deshalb wurden in dieser Arbeit bekannte Algorithmen so erweitert, dass sie Informationen über Prioritäten von Freiheitsgraden in die Berechnungen einbeziehen können.

- Die ***Semantik*** von Objekten und Constraints kann sehr wichtig sein. Teilweise ist jedoch schon ein Vorzeichen ausschlaggebend für die Semantik, wie folgendes Beispiel zeigt.

  Beispiel 3.13: Redundantes Gleichungssystem aus Kettenvermaßung
  Es sei folgendes Gleichungssystem gegeben, das aus einer redundanten Kettenvermaßung resultieren könnte:

$$\begin{array}{rccccc} I: & b & - & a & = & x \\ II: & c & - & b & = & y \\ III: & c & - & a & = & z \end{array} \tag{3.2}$$

  Bei einer Priorisierungsreihenfolge $\{a, b, c, x, y, z\}$ würden in einer Freiheitsgradanalyse, wie sie in dieser Arbeit beschrieben ist, x, y und z als frei erkannt werden. Aufgrund der Struktur der Gleichungen können x, y und z jedoch nicht unabhängig voneinander gewählt werden. Deshalb und wegen der geringeren Priorität von x müsste die Freiheitsgradanalyse die Berechnung von x erzwingen. Wenn aber Gleichung II als $c + b = y$ definiert wäre, würde eine andere Semantik vorliegen, wodurch x, y und z tatsächlich unabhängig voneinander änderbar wären. □

  Ein ähnliches Beispiel ergibt sich auch bei folgendem Gleichungssystem: $\frac{a}{b} = x$; $\frac{b}{c} = y$; $\frac{a}{c} = z$.

- In den beschriebenen Algorithmen ist es stets das Ziel, möglichst hoch priorisierte Objekte zu finden, die nicht oder nur teilweise zu berechnen sind. Die Werte der entsprechenden Objekte werden dann oft konstant gelassen oder entsprechend lokalen Bedingungen berechnet. Für Optimierungen kann das jedoch unerwünscht sein, da Optimierungen oft global auf alle Objekte wirken sollen und somit bei den Berechnungen alle Freiheitsgrade genutzt werden müssen. Eine Freiheitsgradanalyse in der hier beschriebenen Art ist in diesen Fällen unnötig.

- Für eine korrekte Freiheitsgradanalyse ist es wichtig, dass die Information über das Vorliegen von degenerierten Fällen bzw. Spezialfällen im bipartiten Graph hinterlegt ist. So erhöht sich

[30] bei einer bestimmten Menge von Objekttypen und Relationstypen zwischen ihnen

[31] Da der bipartite Graph eine Abstraktion des Constraint-Netzes darstellt und insbesondere nicht auf (durch spezielle Variablen- oder Konstantenwerte hervorgerufene) degenerierte Fälle eingehen kann, ist es legitim, im allgemeinen Fall das Ergebnis der Freiheitsgradanalyse als eine Schätzung zu bezeichnen.

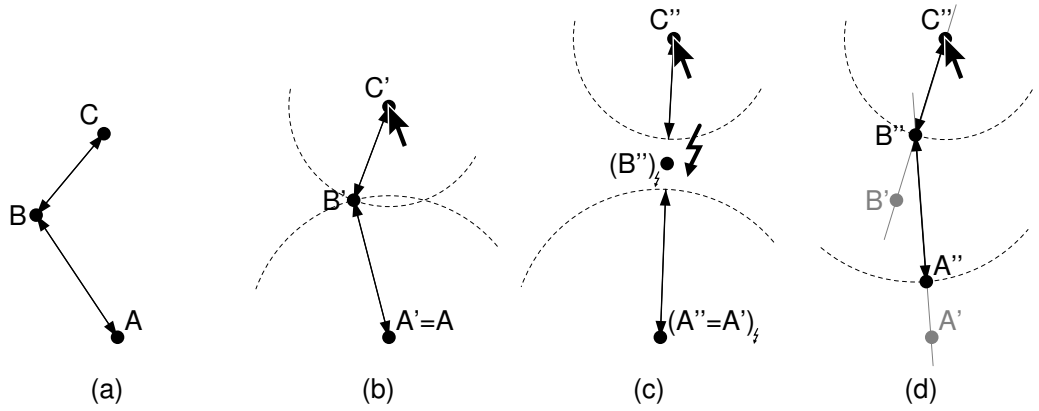


Abbildung 3.14: Nutzung einer nächsten Freiheitsgradverteilung, wenn sich zwei Kreise nicht mehr schneiden. (a) zeigt drei Punkte zwischen denen zwei Abstände definiert sind. (b) wird C bewegt, so kann zunächst A an seinem Ort belassen werden. Wird C jedoch zu weit von A entfernt (c), so ist eine Konstruktion von B (Schneiden zweier Kreise) nicht mehr möglich. Die Freiheitsgrade müssen anders verteilt werden. Für derartige Fälle wurden Algorithmen zur Bestimmung einer nächsten Freiheitsgradverteilung gefunden. (d) zeigt eine Lösung basierend auf einer anderen Freiheitsgradverteilung.

die Valenz eines Abstands-Constraints zwischen zwei Punkten der Ebene von eins auf zwei, wenn der Abstandswert gleich null wird. Lösungen für dieses Problem könnte eine Überprüfung aller potenziell auftretenden relevanten Spezialfälle sein. Vor der Freiheitsgradanalyse müsste für jeden erkannten Spezialfall der bipartite Graph angepasst werden.

- Es gibt Fälle in denen Objekte oder Parameter in Bereichen liegen, die eine Konstruktion verhindern, siehe folgendes Beispiel.

  Beispiel 3.14: Bedarf an einer nächsten Freiheitsgradverteilung, wenn sich zwei Kreise nicht mehr schneiden
  In Abbildung 3.14(a) seien die Ausgangspositionen der Punkte A, B und C dargestellt. Zwei Constraints definieren die Abstände zwischen den Punkten. In Abbildung 3.14(b) wird nun C interaktiv mit dem Mauszeiger an die Position C' bewegt. Die Freiheitsgradanalyse habe die Freiheitsgrade so aufgeteilt, dass C mit zwei Freiheitsgraden (also frei) bewegt werden kann und A an seiner Position verbleibt ($A' = A$). Die Konstruktion, die sicherstellt, dass die Constraints eingehalten werden, besteht im Schneiden zweier Kreise um A' und C'. Erst die Durchführung der Konstruktion und das Finden von B' zeigen, dass die Freiheitsgradverteilung zulässig war. In Abbildung 3.14(c) wurde der Mauszeiger noch weiter nach C'' bewegt. Nun können sich die Kreise nicht mehr schneiden, die Freiheitsgradverteilung ist ungültig. An dieser Stelle könnte die Bewegung nach C'' durch den Constraint-Solver zurückgewiesen werden, jedoch wäre dies für den Nutzer nicht plausibel, denn es gibt keinen Constraint, der A an seiner Position festhält. Deshalb wird durch den Constraint-Solver eine nächste Verteilung der Freiheitsgrade ermittelt (siehe Abschnitt 3.3.4). Diese weist A und B je einen Freiheitsgrad zu. Geometrisch heißt das, dass B auf einem Kreis um C und A auf einem Kreis um B zu konstruieren ist, siehe Abbildung 3.14(d). Die entsprechende Konstruktion kann darin bestehen, jeweils die Projektion der letzten gültigen Punktpositionen (B' und A') auf die Kreise zu verwenden. Dies sichert eine möglichst geringe Bewegung der Punkte und ist deshalb plausibel für den Nutzer. □

- Da die Zusammenstellung des Konstruktionsplanes nur die dem Constraint-Solver bekannten Konstruktionen enthalten kann, ist es möglich, dass für bestimmte Freiheitsgradverteilungen keine passenden Konstruktoren zu finden sind. Auch hier kann die Bestimmung einer nächsten Freiheitsgradverteilung helfen. In diesem Falle werden die Prioritäten u. U. nicht nach

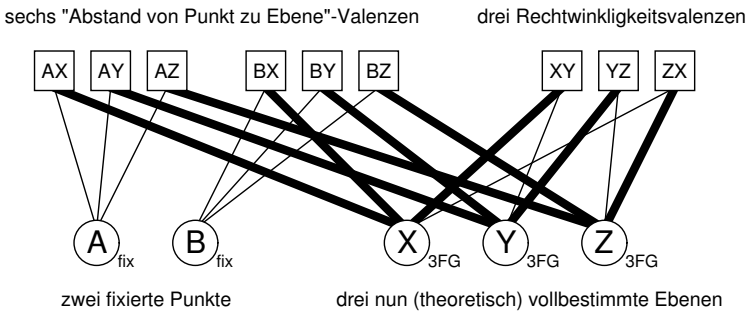


Abbildung 3.15: Problem der Freiheitsgradanalyse bei der Fixierung eines lokalen *3D*-Koordinatensystems anhand zweier Punkte. Die Abbildung zeigt die Zuordnung der Valenzen zu den neun Freiheitsgraden der drei Ebenen, was jedoch falsch ist. Die Ebenen haben noch einen Freiheitsgrad, sie können sich nämlich um eine Achse (durch A und B) drehen.

den Wünschen des Nutzers verteilt, es könnte jedoch wenigstens ein gültiger Konstruktionsplan erstellt werden.

- In großen Modellen, bei denen die Anzahl der unbenutzten Freiheitsgrade etwa die Hälfte der Summe der Freiheitsgradkardinalitäten ausmacht, kommt die kombinatorische Komplexität der Anzahl der Freiheitsgradverteilungen besonders stark zum Tragen. Wenn die Suche nach einer konstruierbaren Verteilung immer wieder die nächste Verteilung heranzieht, so sind die für interaktive Arbeit akzeptablen Berechnungszeiten schnell überschritten. Selbst für die nicht interaktive Lösungsfindung kann eine derartig simple Auswahl der nächsten Verteilung zu lange dauern, d. h. die Suche würde ohne eine Lösung abgebrochen werden. Deshalb wären Algorithmen wünschenswert, die Informationen zur Ursache der Suche nach der nächsten Verteilung (z. B. wenn der Konstruktionsplan nicht aufstellbar ist oder eine Konstruktion fehlschlägt) berücksichtigen können und somit effektiver zu einer passenden Verteilung kommen.

- Ein interessantes Problem, bei dem die Freiheitsgradanalyse fehlschlägt ist das folgende.

  Beispiel 3.15: Fixierung eines lokalen 3D-Koordinatensystems anhand zweier Punkte

  Durch drei jeweils (per Constraint) rechtwinklig zueinander stehende Ebenen sei ein lokales Koordinatensystem definiert. In ihm befinden sich zwei Punkte A und B, die jeweils drei Abstands-Constraints zu den Ebenen des Koordinatensystems haben. Die in den Abstands-Constraints verwendeten Parameter stellen dann die lokalen Koordinaten von A und B dar. Die Fixierung der Punkte (im globalen Koordinatensystem) und ihrer lokalen Koordinaten würde dem lokalen Koordinatensystem respektive den drei Ebenen einen Freiheitsgrad lassen - die Rotation um die Achse, die durch die Punkte geht. Abbildung 3.15 zeigt jedoch, dass die Freiheitsgradanalyse das Modell als voll bestimmt einschätzt. Sie erkennt nicht, dass die beschriebenen Objekte ein Cluster formen, in dem der Abstand zwischen A und B implizit festgelegt ist, eine Fixierung von A und B also nur noch fünf (statt zwei mal drei) Freiheitsgrade wegnimmt. □

In den folgenden Abschnitten werden wichtige Techniken der Freiheitsgradanalyse sowie Erweiterungen, die Lösungsansätze für einige der beschriebenen Probleme bieten, erörtert.

3.3.3 Bestimmung einer ersten Zuordnung in bipartiten Graphen

In der Literatur werden sowohl matching-basierte als auch fluss-basierte Ansätze zur Bestimmung von Zuordnungen genutzt (vgl. z. B. [HLS98, LM98]). Dadurch, dass der Graph in zwei Abstraktionsebenen hinterlegt ist, kann die Implementierung des vorgestellten Constraint-Solvers beide

Sichtweisen berücksichtigen. Es gibt zum einen die abstraktere Ebene (als Ebene 1 bezeichnet), die die Objekte und Constraints als Knoten beinhaltet. Links dieser Ebene können wie bei den fluss-basierten Algorithmen Valenzen von $v \geq 1$ entsprechend den Eigenschaften der angrenzenden Knoten übertragen werden[32]. In Ebene 2 werden nur Knoten verwendet, die eine Valenz respektive Freiheitsgradkardinalität von eins aufweisen. Entsprechend kann wie in den matching-basierten Ansätzen nur eine Valenz von eins übertragen werden. Die Ebene 2 lässt sich aus der Ebene 1 generieren. Dabei können auch Informationen zur Gruppierung gespeichert werden, so dass die matching-basierten Algorithmen für die Ebene 2 auf den gleichen Informationsgehalt über die Modelle zugreifen können, wie die fluss-basierten Algorithmen der Ebene 1.

Unabhängig davon, für welche Ebene ein Algorithmus implementiert wurde, lassen sich für ihn folgende Kriterien unterscheiden:

- Berücksichtigung von Prioritäten,
- Berücksichtigung von zusammengehörigen Freiheitsgraden,
- Zulässigkeit von Überbestimmtheit,
- Unterstützung inkrementeller bzw. dynamischer Änderungen und
- Suche nach einer anderen Lösung.

Statischer Ansatz

Die in der Literatur beschriebenen Ansätze für das Finden von maximalen Paarungen haben typischerweise statischen Charakter, d. h. bei Start eines entsprechenden Algorithmus ist stets der komplette bipartite Graph bekannt. Eine maximale Paarung wird dann genau für diesen Graphen berechnet. Wird er geändert, so kann der Algorithmus die vorher noch gültige Paarung nicht an den neuen Graphen anpassen. Die Information über die alte Paarung wird verworfen und eine Paarung vollständig neu berechnet.

Die Verwendung eines Algorithmus mit statischem Ansatz ist durchaus sinnvoll, wenn Modelle komplett neu geladen werden. Für die bei interaktiver Arbeit oder Strukturoptimierungen oft auftretenden inkrementellen Änderungen ist ein Algorithmus mit dynamischem Ansatz jedoch wesentlich effizienter. Er wird weiter unten beschrieben.

Der im Constraint-Solver entsprechend dem statischen Ansatz implementierte Algorithmus *BG_M0* arbeitet wie folgt:

$reserve := \sum F - \sum V$
$gefunden := 0$
Für jeden Freiheitsgrad f des Modells (beginnend mit der geringsten Priorität) mache Folgendes:

> Suche nach einem Vergrößerungspfad von f (stets unbenutzt) zu einer noch nicht benutzten Valenz v.
> Wenn ein Vergrößerungspfad gefunden wurde, dann:
>
> > schalte die Kanten im Vergrößerungspfad um,
> > $gefunden := gefunden + 1$,
> > wenn $gefunden = \sum V$, dann beende den Algorithmus und melde einen Erfolg,
>
> andernfalls:

[32] höchstens *Min(Valenz des Constraints, Kardinalität des Objektes)*

wenn $reserve = 0$ und Überbestimmtheit nicht zulässig ist, dann
beende den Algorithmus und melde einen Misserfolg,
$reserve := reserve - 1$.

Beende den Algorithmus und melde einen Erfolg, wenn eine Überbestimmtheit zulässig ist, ansonsten einen Misserfolg.

Das Auffinden eines Vergrößerungspfades erfolgt sinnvollerweise nach einer „Breite zuerst"-Strategie. Auf diese Weise wird sichergestellt, dass es keinen kürzeren Pfad gibt. Kurze Pfade werden schneller gefunden und können auch schneller umgeschaltet werden. Die Suche stoppt an schon betrachteten Knoten, denn sie werden offensichtlich über einen anderen kürzeren bzw. gleich langen Pfad erreicht. Die Komplexität ist somit im schlechtesten Fall pro Freiheitsgrad linear zur Anzahl der Knoten.

Da der Algorithmus die Freiheitsgrade entsprechend der vorgegebenen Reihenfolge abarbeitet, kann durch eine prioritätsabhängige Sortierung die Berücksichtigung von Prioritäten erreicht werden. Die Zulässigkeit von Überbestimmtheit ist über einen Parameter steuerbar. Für den obigen Algorithmus ist die Zusammengehörigkeit von Freiheitsgraden nicht von Bedeutung. Erst bei der Berechnung einer nächsten größten Paarung ist es sinnvoll, eine solche Information zu beachten.

Dynamischer bzw. inkrementeller Ansatz

Im Zuge inkrementeller Modelländerungen ist eine inkrementelle Anpassung des bipartiten Graphen für die Freiheitsgradanalyse sowie eine entsprechende Anpassung der aktuellen Freiheitsgradverteilung sinnvoll. Deshalb wurde ein Algorithmus für die dynamische Anpassung des Graphen sowie der Freiheitsgradverteilung entwickelt. Er ist auf vier Methoden aufgeteilt. Alle berücksichtigen Prioritäten. Sie arbeiten in der Ebene 1, also ähnlich einer fluss-basierten Vorgehensweise, wodurch die Berücksichtigung von Gruppen zusammengehöriger Freiheitsgrade stets gegeben ist. Die Zulässigkeit von Überbestimmtheit kann jeweils über einen Parameter gesteuert werden.

- Erhöhung der Kardinalität eines Objektes

  Wenn die Kardinalität eines Objektes erhöht wird (z. B. auch von 0 auf n beim Einfügen des Objektes oder bei der Aufhebung seiner Fixierung), so impliziert das, dass Valenzen, die bis jetzt höher priorisierten Objekten zugeordnet wurden, nun auf dieses Objekt umgeleitet werden können. Niedriger priorisierte Objekte sind von der Änderung nicht betroffen.

- Verringerung der Kardinalität eines Objektes

  Bei Verringerung der Kardinalität eines Objektes (z. B. von n auf 0 beim Löschen des Objektes oder bei seiner Fixierung) müssen alle Valenzen, die dem Objekt bis jetzt zugeordnet waren auf andere Objekte umgeleitet werden. Typischerweise sind das höherpriorisierte Objekte. Nach einer Suche von alternativen Freiheitsgradverteilungen (siehe Abschnitt 3.3.4) kann es jedoch auch vorkommen, dass die Valenzen auch niedriger priorisierten Objekten zugeordnet werden können[33].

- Erhöhung der Valenz eines Constraints

  Die Erhöhung der Valenz eines Constraints wird außer im überbestimmten Fall immer zu einer Senkung des Freiheitsgrades von Objekten führen. Dementsprechend wird ein Pfad vom Constraint zu einem Objekt gesucht, dem die Valenz zugeteilt werden kann. Zur Beachtung der Priorisierung ist es wichtig, dass das Objekt eine möglichst geringe Priorität hat.

[33] In Beispiel 3.14 könnte z. B. der Punkt A eine höhere Priorität haben als der Punkt B, wodurch es zur ersten Verteilung kam (B wurde kein FG zugeordnet). Die nächste Verteilung stellt die Umverteilung eines Freiheitsgrades vom höher priorisierten A zum niedriger priorisierten B dar.

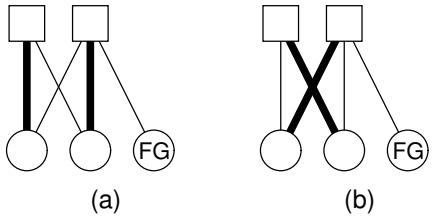


Abbildung 3.16: Unterschied zwischen der Zuordnung von Knoten und der Zuordnung von Freiheitsgraden. (a) und (b) zeigen zwei unterschiedliche Zuordnungen von Knoten, jedoch ist die Zuordnung der Freiheitsgrade in beiden Fällen gleich. Die Zuordnung von Knoten ist durch breitere Kanten zwischen ihnen dargestellt. Knoten, denen ein Freiheitsgrad zugeordnet wurde, sind mit „FG“ gekennzeichnet.

- Verringerung der Valenz eines Constraints

  Bei Verringerung der Valenz eines Constraints kann ein Objekt einen höheren Freiheitsgrad erhalten. Das muss nicht eines der vom Constraint referenzierten Objekte sein. Im Allgemeinen wird es auch über Umleitungspfade erreichbare und höherpriorisierte Objekte geben, bei denen der Freiheitsgrad steigen kann. Es wird also ein Umleitungspfad vom Constraint zu einem möglichst hoch priorisierten Objekt gesucht. Kann er gefunden werden, so wird die auf das Objekt über den Pfad wirkende Valenz im Objekt abgezogen, d. h. im Objekt erfolgt eine entsprechende Freiheitsgraderhöhung. Die Pfadsuche und das entsprechende Erhöhen der Freiheitsgrade erfolgt so lange, bis die Valenz nicht mehr auf die Objekte wirkt. Sollte schon zu Beginn der Suche kein Umleitungspfad gefunden werden, lag eine Überbestimmtheit vor und der Constraint wurde im Konstruktionsplan ggf. nur auf Konsistenz getestet.

In [Bar87, Abschnitt 4.3.4] sind inkrementelle Algorithmen zur Überprüfung der Konsistenz eines Constraint-Netzes beschrieben, die dem Nutzer nach Modifikationen des Modells insbesondere Überbestimmtheiten anzeigen sollen. Die Algorithmen sind jedoch nicht für die Beachtung von Prioritäten ausgelegt.

3.3.4 Vorgehensweisen zur Bestimmung von weiteren Zuordnungen

Im Kontext der Freiheitsgradanalyse ist es immer wieder notwendig, nach dem Finden einer ersten Zuordnung der Freiheitsgrade zu den Objekten eine oder mehrere alternative Zuordnungen zu bestimmen (vgl. Beispiel 3.14). Für eine ähnliche Problemstellung werden in [Uno97] Algorithmen gezeigt. Allerdings sind die Algorithmen nicht anwendbar, denn:

- sie berücksichtigen keine Prioritäten und
- sie suchen nach den Zuordnungen von der einen Knotenmenge zur anderen Knotenmenge. Bei der Freiheitsgradanalyse ist jedoch nur die Zuordnung der Freiheitsgrade von Interesse. Die Algorithmen würden zu viele Zuordnungen liefern (siehe Abbildung 3.16), was unnötigen Berechnungsaufwand verursacht.

Unterbestimmte Modelle weisen Freiheitsgrade auf. Diese können typischerweise unterschiedlich auf die einzelnen Objekte verteilt werden, jedoch gibt es auch Ausnahmen.

Beispiel 3.16: Abhängigkeit der Existenz weiterer Zuordnungen vom Interaktionspunkt

Es seien drei Punkte und zwei Abstände gegeben (siehe Abbildung 2.10), d. h. es verbleiben vier Freiheitsgrade. Beim Ziehen am äußeren Punkt C kann der mittlere Punkt B berechnet werden und A stehen bleiben bis zu weit gezogen wird. Dann müssen die zwei Freiheitsgrade auf B und A aufgeteilt werden. Analoges gilt für das Bewegen von A. Wird aber an B gezogen, kann keine weitere Zuordnung gefunden werden, bei der der bewegte Punkt zwei Freiheitsgrade hat. Erst wenn zugelassen wird, dass der bewegte Punkt weniger als zwei Freiheitsgrade hat, werden weitere Freiheitsgradverteilungen möglich. □

Vorgehensweisen

Generell sind unterschiedlichste Vorgehensweisen bei der Bestimmung der nächsten Zuordnung denkbar. Im Zusammenhang mit Freiheitsgradverteilungen wurden folgende drei Vorgehensweisen untersucht[34]:

- *MAP*: Versucht möglichst die hoch priorisierten Freiheitsgrade nicht zu verändern. Hierbei wird zugelassen, dass niedrig priorisierten Objekten frühzeitig (d. h. schon nach der Ermittlung einiger nächster Verteilungen) Freiheitsgrade zugeordnet werden. *MAP* ist z. B. gut um die Freiheitsgrade, die einem hoch priorisierten bewegten Objekt zugeordnet wurden, auch bei der Berechnung weiterer Freiheitsgradverteilungen möglichst lange für die Bewegung des Objektes zu behalten. *MAP* ist somit sehr wichtig für eine nutzerorientierte Interaktion.
- *MIP*: Vermeidet die Vergabe von Freiheitsgraden an niedrig priorisierte Objekte. Wird die Vergabe von Freiheitsgraden an niedrig priorisierte Objekte nicht verhindert (wie bei *MAP*), so impliziert das, dass höher priorisierte Objekte[35] (unnötigerweise) Freiheitsgrade verlieren. Dies könnte je nach Modell zu Unverständnis beim Nutzer führen.
- *MXP*: Hierbei handelt es sich um eine Kombination der beiden oben beschriebenen Vorgehensweisen.

Grundidee

Die drei im Constraint-Solver getesteten Vorgehensweisen basieren auf der gleichen Grundidee, die hier zum besseren Verständnis der drei in den folgenden Abschnitten beschriebenen Algorithmen erörtert werden soll. Sie besteht darin, dass jede der möglichen Zuordnungen eindeutig durch einen Binärcode repräsentiert ist, der ähnlich einer Binärzahl hochgezählt werden kann, wobei die Anzahl der Einsen in der Zahl erhalten bleiben muss[36].

Die Binärcoderepräsentation der Zuordnung ist wie folgt definiert:

- Die Stellenanzahl entspricht der Summe der Objektfreiheitsgradkardinalitäten, so dass für jeden möglicherweise in der Zuordnung benutzten Freiheitsgrad eine Binärstelle zur Verfügung steht.
- Die Notation der Binärstellen erfolgt von links nach rechts, wobei links die Freiheitsgrade mit den geringsten Prioritäten stehen.
- Ist ein Freiheitsgrad in der Zuordnung benutzt, so wird für die entsprechende Binärstelle eine „1“ notiert, ansonsten eine „0“. In Abbildungen wird zur besseren Sichtbarkeit mit den Symbolen • und ◦ gearbeitet.

---

[34] Motivation zur Namensgebung: ***M***atching focused on m***A***ximal/m***I***nimal/mi***X***ed ***P***riority

[35] Gemeint sind hier auch die Objekte mit mittlerer Priorität. Abbildung 3.18 und Abschnitt 3.3.5 veranschaulichen dies noch eingehender.

[36] z. B. bei 2 gesetzen Bits: 3 (11_b), 5 (101_b), 6 (110_b), 9 (1001_b), 10 (1010_b), 12(1100_b), usw.

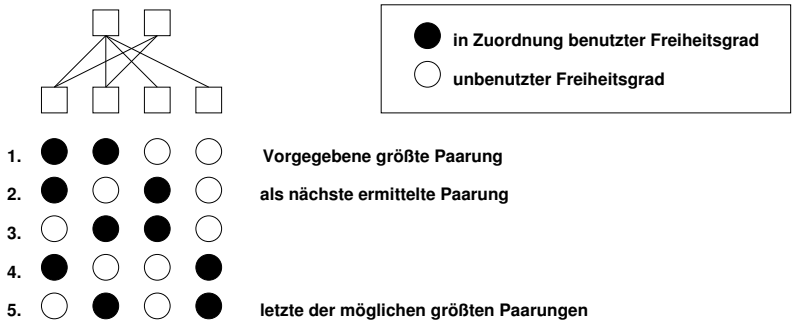


Abbildung 3.17: Ein Beispielgraph mit seinen fünf größten Paarungen.

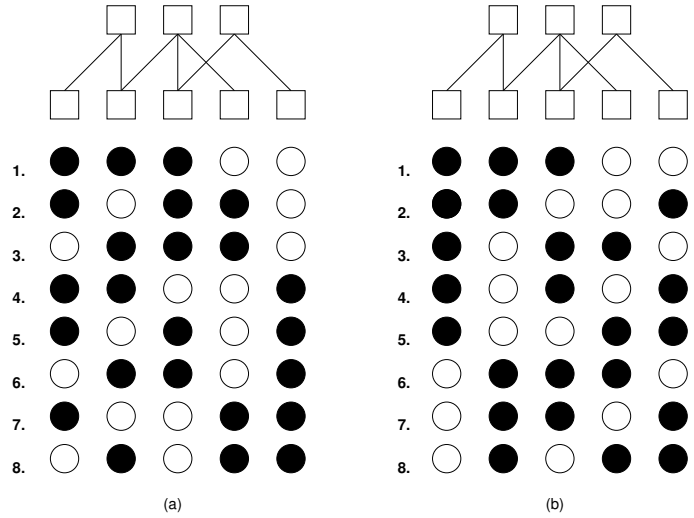


Abbildung 3.18: Ein Beispielgraph mit seinen acht größten Paarungen (a) sortiert mit dem Algorithmus MAP1 und (b) sortiert nach dem Algorithmus MIP1.

Beim Hochzählen müssen je nach Modell einige Binärcodes (Binärzahlen) übersprungen werden, da sie keine möglichen Zuordnungen darstellen. Für die unten beschriebenen Algorithmen muss der Binärcode stets größte Paarungen repräsentieren. In Abbildung 3.17 wird ein Graph gezeigt, bei dem zwei Valenzen auf vier Freiheitsgrade zu verteilen sind. Somit bleiben zwei Freiheitsgrade in den Zuordnungen (größte Paarungen) ungenutzt. Von den sechs Binärcodes mit zwei Einsen, sind jedoch nur fünf gültig. Aufgrund der Kanten im Graph kann ($\circ\circ\bullet\bullet$) nicht erzeugt werden.

3.3.5 Algorithmen MAP1 und MIP1

Von den unterschiedlichen Umsetzungen der Berechnungen von nächsten größten Paarungen sollen zunächst zwei einfache Vertreter (*MAP1* und *MIP1*) betrachtet werden. Eine Einteilung der Freiheitsgrade in Gruppen wird nicht vorgenommen. Die beiden Algorithmen starten bei einem gegebenen Binärcode für eine größte Paarung und geben im Erfolgsfall den Binärcode der nächsten größten Paarung aus. Ein Beispiel für die Arbeitsweisen wird in Abbildung 3.18 gezeigt. Wie dort zu sehen ist, vermeidet *MAP1* die Zuordnung der hochpriorisierten Freiheitsgrade, sie werden erst in den späteren Zuordnungen verwendet. *MIP1* verfährt hingegen nach der Strategie, niedrig priorisierte Freiheitsgrade, die bereits vergeben wurden, möglichst lange in der Zuordnung zu behalten.

In Abbildung 3.18b) ist dementsprechend zu sehen, wie der niedrigst priorisierte Freiheitsgrad in den ersten fünf Zuordnungen belegt bleibt.

Der Algorithmus *MAP1* (Abbildung 3.18a) arbeitet jeweils folgende Schritte ab:

1. Suche den ersten[37] unbenutzten Freiheitsgrad (○), dem eine Valenz zugeordnet werden kann, ohne dass bis jetzt benutzte höherpriorisierte Freiheitsgrade dadurch unbenutzt werden. Von niedriger priorisierten Freiheitsgraden kann die Valenz jedoch umgeleitet werden („*hochzählen*“).
2. Wenn kein derartiger Freiheitsgrad gefunden werden konnte, dann stoppe. Ansonsten markiere diesen Freiheitsgrad als benutzt und alle niedriger priorisierten Freiheitsgrade als unbenutzt.
3. Erstelle eine neue Zuordnung für die niedriger priorisierten Freiheitsgrade. Hierfür wird auf sie der Algorithmus *BG_ M0* (Seite 52) angewendet.

Zum besseren Verständnis von *MAP1* wird nun der Übergang von der ersten größten Paarung (●●●○○), die vorgegeben wurde, zur zweiten größten Paarung in Abbildung 3.18a erläutert. Der im Schritt 1 gesuchte unbenutzte Freiheitsgrad wird an der vierten Stelle (von vorn beginnend gezählt) gefunden, denn er ist der erste unbenutzte Freiheitsgrad und es gelingt einen Umleitungspfad zu einem niedriger priorisierten benutzten Freiheitsgrad zu finden (z. B. zweite Stelle oder dritte Stelle). In Schritt 2 wird die vierte Stelle als nun benutzt markiert (●) und die Stellen eins bis drei als unbenutzt (○). Im letzten Schritt wird dann für die Stellen eins bis drei eine neue Zuordnung gefunden. Das Resultat ist (●○●●○).

Der Algorithmus *MIP1* (Abbildung 3.18b) arbeitet jeweils folgende Schritte ab:

1. Suche den am höchsten priorisierten benutzten Freiheitsgrad h, dessen Valenz einem noch höher priorisierten (bisher noch ungenutzten) Freiheitsgrad i zugeordnet werden kann.
2. Wenn kein derartiger Freiheitsgrad h gefunden werden konnte, dann stoppe. Ansonsten markiere den Freiheitsgrad h sowie alle höher priorisierten Freiheitsgrade als unbenutzt.
3. Erstelle eine neue Zuordnung für die höher priorisierten Freiheitsgrade. Hierfür wird auf sie der Algorithmus *BG_ M0* (Seite 52) angewendet.

3.3.6 Algorithmen MAPG und MIPG

An dieser Stelle soll nun eine Weiterentwicklung von *MAP1* und *MIP1* vorgestellt werden. Die Algorithmen *MAPG* und *MIPG* verfolgen jeweils die gleiche Strategie wie *MAP1* bzw. *MIP1*, benutzen jedoch zusätzlich noch Gruppierungsinformationen, um nur sinnvolle nächste größte Paarungen auszugeben. D. h. die Gruppierungsinformation wird genutzt, um einige größte Paarungen, die die Algorithmen *MAP1* bzw. *MIP1* liefern würden, zu überspringen. Zur Motivation sei hier folgendes Beispiel gegeben.

Beispiel 3.17: Nutzung von Gruppierungsinformationen in MAPG und MIPG
Ein Punkt P (zwei Freiheitsgrade) in der Ebene sei auf einer Gerade fixiert. Er kann sich also noch auf ihr mit einem Freiheitsgrad bewegen. Ein Freiheitsgrad wird dem „Punkt auf Gerade“-Constraint (Valenz ist eins) zugeordnet. Um welchen der beiden Freiheitsgrade es sich dabei handelt, ist unerheblich. Dementsprechend führt eine Gruppierung der beiden Freiheitsgrade von P dazu, dass Algorithmen, die wie *MAPG* und *MIPG* auf Gruppierungsinformationen zugreifen, keine weiteren Zuordnungen vorschlagen. □

In *MAPG* und *MIPG* wird vorausgesetzt, dass sich zwischen den Freiheitsgraden einer Gruppe keine Freiheitsgrade anderer Gruppen befinden. Das wird stets erreicht, wenn die Freiheitsgrade

[37] von der niedrigsten Priorität aus gesehen

von jedem Objekt (z. B. Punkt, Gerade, Ebene usw.) in einer eigenen Gruppe zusammengefasst sind. Die Priorisierung der Objekte führt dann zu einer Sortierung der entsprechenden Gruppen.

Eingabe:

- Vektor von nach der Priorität sortierten Freiheitsgraden,
- Vektor mit Gruppierungsinformationen für die Freiheitsgrade,
- aktuelle Zuordnung und
- der Graph aus Freiheitsgraden und Valenzen.

Ausgabe:

- nächste Zuordnung oder
- Rückmeldung, dass keine nächste Zuordnung gefunden werden konnte.

benutzte Variablen:

- *f0*, der aktuelle Freiheitsgrad, für den ein Pfad zu einer Valenz gesucht wird.
- *g0*, die aktuelle Gruppe, in der *f0* enthalten ist.

Ablauf für *MAPG*:

1. Suche die erste Gruppe *g0*, in der die Anzahl benutzter Freiheitsgrade[38] erhöht werden kann, ohne Freiheitsgrade in höher priorisierten Gruppen freizugeben[39]. Kriterien für eine solche Gruppe sind:
   (a) Der letzte Versuch, die Anzahl der benutzten Freiheitsgrade zu vergrößern darf nicht fehlgeschlagen sein, ansonsten muss bei der nächsten Gruppe fortgefahren werden.
   (b) In der Gruppe muss noch ein unbenutzter Freiheitsgrad vorhanden sein, ansonsten ist bei der nächsten Gruppe fortzufahren.
   (c) Es existiert ein benutzter Freiheitsgrad in einer niedriger priorisierten Gruppe, der in dieser Gruppe genutzt werden kann. Diese Bedingung ist ab dem zweiten Durchlauf von Schritt 1 stets erfüllt.

   f0 ist dann der erste nicht benutzte Freiheitsgrad in *g0.*
2. Alle Zuordnungen zu Freiheitsgraden in niedriger priorisierten Gruppen (in den Darstellungen links von *g0*) werden aufgehoben. Die Zuordnungen in den höher priorisierten Gruppen bleiben jedoch entsprechend der Strategie, die MAPG umsetzt, erhalten.
3. Suche einen Pfad von *f0* zu einer nicht benutzten Valenz (eine der in Schritt 2 freigegebenen).
4. Wende Algorithmus *BG_M0* (Seite 52) auf die Freiheitsgrade aller niedriger priorisierten Gruppen an.

Ablauf für *MIPG*:

1. Suche die am höchsten priorisierte benutzte Gruppe *g0,* von der eine Zuordnung auf eine höher priorisierte Gruppe erfolgen kann.

   f0 ist dann ein benutzter Freiheitsgrad in *g0.*

[38] „benutzter Freiheitsgrad“ heißt: in Zuordnung benutzt, d. h. durch eine Valenz verbraucht
[39] freigeben impliziert, dass sie vorher in der Zuordnung benutzt wurden

2. Alle Zuordnungen zu Freiheitsgraden in höher priorisierten Gruppen (in den Darstellungen rechts von *g0*) werden aufgehoben. Die Zuordnungen in den niedriger priorisierten Gruppen bleiben jedoch entsprechend der Strategie, die MIPG umsetzt, erhalten.

3. Markiere *f0* als unbenutzt, hierdurch wird eine weitere Valenz „frei“ (d. h. sie ist dem Freiheitsgrad nicht mehr zugeordnet).

4. Wende Algorithmus *BG_ M0* (Seite 52) auf die Freiheitsgrade aller höher priorisierten Gruppen an. Hierdurch werden die noch nicht zugeordneten Valenzen zugeordnet.

Ab Schritt 3 wird keine Gruppeninformation mehr benötigt.

In der Implementierung der Algorithmen für den Constraint-Solver sind *f0* und *g0* Indizes für Vektoren, in denen Informationen zu den Freiheitsgraden und Gruppen gespeichert sind. Zudem konnten die Algorithmen in ihrer Laufzeit dadurch optimiert werden, dass:

- die Suche nach einer genutzten oder ungenutzten Valenz bzw. einem Freiheitsgrad nach dem „Breite zuerst“-Ansatz erfolgt (lange Suchpfade werden vermieden), wobei bereits erreichte Knoten markiert werden[40],
- die noch zuzuordnenden Valenzen mit der Anzahl der nicht zugeordneten Freiheitsgrade verglichen wurde - vorzeitiger Abbruch, wenn die Anzahl der nicht zugeordneten Freiheitsgrade zu klein ist,
- die Zuordnungen in den niedriger priorisierten Gruppen (Algorithmus *MAPG*) bzw. den höher priorisierten Gruppen (Algorithmus *MIPG*) nicht mehr explizit gelöscht werden, sondern eine Umverteilung zu niedriger priorisierten Freiheitsgraden erfolgte.

3.3.7 Der Algorithmus MXPGT

In der aktuellen Implementierung von *Ficucs* wird der Algorithmus *MXPGT* zur Ermittlung einer nächsten Zuordnung genutzt. Er setzt beide oben erörterten Strategien um:

1. die Vermeidung von Zuordnungen zu hoch priorisierten Freiheitsgraden und
2. das Beibehalten von Zuordnungen zu niedrig priorisierten Freiheitsgraden.

Wie in 3.18 zu sehen ist, bringen diese beiden Strategien sehr unterschiedliche Resultate hervor. Ihre gemeinsame Nutzung in einem Algorithmus wird durch die Beachtung von Typinformationen möglich. Solche Typinformation kann aus der jeweiligen Art der Objekte gewonnen werden. Es ist aber auch möglich, dass für einzelne Objekte problem- bzw. modellspezifische Typen definiert wurden, wie z. B. in Abschnitt 5.2.3.

Ziel ist es, die Anzahl der Zuordnungen zu den Freiheitsgraden eines Typs möglichst lange beizubehalten. Innerhalb der Freiheitsgrade der Typen wird dann entsprechend *MIPG* die Zuordnung zu gering priorisierten Freiheitsgraden möglichst lange beibehalten. Erst wenn diese Strategie zu keinen weiteren Zuordnungen führt, wird die zweite Strategie verwendet, die ähnlich *MAPG* arbeitet. In einem möglichst niedrig priorisierten Typ wird die Anzahl der Zuordnungen dadurch erhöht, dass in einem noch niedriger priorisierten Typ eine Zuordnung weggenommen wird. Dann werden die Zuordnungen wieder nur innerhalb der Typen geändert usw.

---

[40] Das erlaubt das Verwerfen von Suchpfaden - nur der erste Pfad, der einen Knoten erreicht, wird weiter verfolgt. Die Verwendung des „Breite zuerst“-Ansatzes und das Markieren von bereits erreichten Knoten ist häufig in der Literatur beschrieben, z. B. [MTYS94] und [ABMP91].

3.3.8 Behandlung alternativer Constraints

Wie in Abschnitt 2.6 gezeigt wurde, ist es durch den Einsatz von Hinweisen möglich, die Freiheitsgradanalyse, die Planerstellung und auch Berechnungen zu steuern. Die meisten Hinweise müssen nicht in der Freiheitsgradanalyse berücksichtigt werden. Hinweise zu alternativen Constraints beeinflussen jedoch die mögliche Verteilung der Freiheitsgrade. Hierzu umfasst jeder Hinweis Folgendes:

- Die Menge der zumindest teilweise redundanten Constraints, aus der verschiedene (also alternative) Untermengen gebildet werden können.
- Eine Angabe über die Valenz $V_{\dot{U}}$, die bei der Freiheitsgradanalyse nicht berücksichtigt werden darf.

Aus Sicht der Freiheitsgradanalyse erfordert die Behandlung von alternativen Constraints nur geringfügige Anpassungen der Algorithmen bzw. Datenstrukturen. Die wesentlichen Aspekte sind:

- Bei der Abbildung einer Menge von alternativen Constraints wird jeder der Constraints mit einem neuen Objekt[41] verbunden, das die überschüssige Valenz der Menge aufnimmt. In flussbasierten Ansätzen hat das Objekt eine Kardinalität von $V_{\dot{U}}$. In matching-basierten Ansätzen sind für das Objekt n Knoten zu modellieren mit $n = V_{\dot{U}}$.
- Dieses Objekt ist niedrigst priorisiert, so dass es beim Finden einer ersten Zuordnung die überschüssigen Valenzen $V_{\dot{U}}$ aufnimmt.
- Bei der Ermittlung einer nächsten Zuordnung muss die Zuordnung von $V_{\dot{U}}$ Valenzen zu diesem Objekt stets erhalten bleiben, ansonsten würden zu viele Constraints gleichzeitig auf die „normalen“ Objekte wirken und das Modell wäre überbestimmt.

3.4 Pläne

3.4.1 Struktur

Pläne können aus Einträgen unterschiedlicher Komplexität bestehen. So können sie einfache Operationen beinhalten, die zur Berechnung eines geometrischen Objektes oder eines Parameters dienen. Es gibt aber auch komplexe Einträge die Schleifen oder Verzweigungen beschreiben. Im folgenden Text soll der Begriff *Eintrag* verwendet werden, wenn der Fokus auf dem Enthaltensein in einem Plan liegt. Der Begriff *Operation* weist auf den Aspekt der Berechnung hin, zu der Planeinträge typischerweise dienen.

Einfache Operationen

Grundsätzlich kann ein Objekt eines bestimmten Typs auf unterschiedlichen Wegen berechnet werden. So ist es u. a. möglich, einen Punkt in *2D* als Schnittpunkt zweier Geraden (wenn die beiden Geraden bekannt sind und er per Constraint auf den beiden liegt) oder als Schnittpunkt eines Kreises und einer Geraden (z. B. wenn der Punkt per Constraint auf dem Kreis und auf der Geraden liegt) zu konstruieren. Ein wesentliches Merkmal des vorgestellten Constraint-Solvers besteht darin, dass er für die unterschiedlichen Wege zur Berechnung eines Objektes jeweils andere Codesequenzen beinhaltet. Das hat zum Vorteil, dass auf Spezialfälle gezielt eingegangen werden kann[42]. Problematisch wird der gewählte Ansatz jedoch, wenn die Anzahl der verwendeten

[41] Dieses Objekt repräsentiert den Hinweis über die alternativen Constraints.

[42] siehe z. B. die Ausführungen zu Hinweisen in Abschnitt 2.6 und die Heterarchie von Konstruktoren in Abschnitt 3.5.3

Objekt- sowie Constraint-Typen groß wird, denn die Anzahl der benötigten Operationen wächst kombinatorisch zur Anzahl der Constraint-Typen sowie den Freiheitsgraden der Objekte[43].

Für einen Objekttyp mit f Freiheitsgraden, der in n verschiedenen Constraint-Typen verwendet werden kann, sind *etwa*[44] K einfache Berechnungsoperationen zu implementieren, wobei

$$K = \begin{pmatrix} n+f-1 \\ f \end{pmatrix} \tag{3.3}$$

$$= \frac{(n+f-1)!}{f!(n-1)!} \tag{3.4}$$

gilt, denn K entspricht der Anzahl der Kombinationen von n Elementen zur f-ten Klasse mit Wiederholung (vgl. [Bar86]). Wenn z. B. in einem Constraint-Solver für einen Punkt in *2D* vier Constraint-Typen definiert sind, so werden entsprechend Gleichung 3.3 zehn Operationen benötigt. Angenommen für Punkte in *3D* sind auch vier Constraint-Typen definiert, dann sind aufgrund ihres höheren Freiheitsgrades bereits zwanzig Operationen für Punktberechnungen zu implementieren.

Unter Berücksichtigung von möglicher Unterbestimmtheit erhöht sich die Anzahl der Constraint-Typen quasi um eins, denn an Stelle jedes der Constraint-Typen[45] kann nun auch ein „dieser Freiheitsgrad wird nicht beeinflusst“ auftreten. Demzufolge ergibt sich für Constraint-Solver, die unterbestimmte Fälle zulassen, ein noch größeres K_u mit:

$$K_u = \begin{pmatrix} n+f \\ f \end{pmatrix} \tag{3.5}$$

$$= \frac{(n+f)!}{n! \cdot f!} \tag{3.6}$$

Die kombinatorische Vielfalt der benötigten Operationen wird auch in anderen Quellen beschrieben. Zum Beispiel geben Chung und Sachs in [CS94] folgende Abschätzung an:

$$K_{Chung} = \begin{pmatrix} N+2M-1 \\ 2M \end{pmatrix} \tag{3.7}$$

$$= \frac{(N+2M-1)!}{(2M)!(N-1)!} \tag{3.8}$$

Hierbei ist N die Anzahl der unterschiedlichen Constraint-Typen zwischen zwei geometrischen Objekten und M die Anzahl von gleichzeitig zu berechnenden Objekten. Die Gleichungen 3.3 und 3.7 können mit $n = N$ sowie $f = 2M$ ineinander überführt werden, dementsprechend lässt sich die Schlussfolgerung ziehen, dass Chung und Sachs in ihrer Gleichung für jedes der Objekte einen durchschnittlichen Freiheitsgrad von zwei angenommen haben.

Sequenz

Die typische Anordnung von Einträgen ist die Sequenz. Wenn die Abhängigkeiten der einzelnen zu konstruierenden Objekte beachtet werden, lässt sich die Reihenfolge der Einträge, die bei der Serialisierung des Abhängigkeitsgraphen aufgestellt wurde, verändern. Dies kann z. B. für die Optimierung von Plänen verwendet werden (siehe Optimierung auf Seite 64).

---

[43] Da in *2D* die Objekte typischerweise weniger Freiheitsgrade haben als in *3D*, fällt hier die kombinatorische Vielzahl benötigter Operationen im Allgemeinen nicht so drastisch aus.

[44] Die Anzahl ist in der Regel kleiner, da in Gleichung 3.3 angenommen wird, dass jeder Constraint eine Valenz von eins hat. Sie könnte jedoch auch größer sein, wenn z. B. in Abhängigkeit verschiedener Berechnungsmodi für die gleiche Constraint-Kombination unterschiedliche Berechnungen ablaufen sollen.

[45] wieder unter der Annahme, dass Constraints eine Valenz von eins aufweisen

Verzweigung

Verzweigungen werden bisher nur innerhalb der Operationen genutzt, insbesondere zur Behandlung von Spezialfällen (siehe Abschnitt 3.5.3). Als Planeinträge sind sie in *Ficucs* noch nicht realisiert worden. Deshalb ist es auch noch nicht möglich, Planeinträge zu überspringen bzw. bestimmte Abarbeitungspfade auszuwählen, so dass in der jetzigen Implementierung stets alle Einträge abzuarbeiten sind[46].

Beispiel 3.18: Plan mit Verzweigung zur Behandlung degenerierter Fälle
Ein möglicher Anwendungsfall für Verzweigungen im Plan ist in Beispiel 3.14 gezeigt worden, wo sich im Laufe interaktiver Modifikationen zwei Punkte soweit voneinander entfernten, dass sich ein dritter Punkt nicht mehr durch das Schneiden von Kreisen um die Punkte konstruieren ließ. Wenn Verzweigungen als Planeinträge zulässig wären, könnte ein Test erfolgen, ob die Punkte A und C (siehe Abbildung 3.14) nahe genug zusammen liegen. Ist das so, dann würden die beiden Kreise geschnitten werden. Ansonsten müsste in einem alternativen Zweig die Konstruktion von B aber auch die von A erfolgen. □

Wie Beispiel 3.18 zeigt, ließen sich in einem Plan unterschiedliche Freiheitsgradverteilungen berücksichtigen. Verzweigungen in Plänen könnten so die Notwendigkeit der Neuerstellung von Plänen vermeiden. Ein Plan würde dann für unterschiedliche Objektpositionen nutzbar sein. Insbesondere die degenerierten Fälle bei den Objektpositionen können bisher plötzliche Verzögerungen in der Interaktion verursachen, wenn der Erstellungsaufwand für den neuen (an den degenerierten Fall angepassten) Plan hoch ist. Enthalten Pläne Verzweigungen so wird die Interaktion in derartigen Fällen weiterhin gleichmäßig erfolgen. Ein weiterer Vorteil der geringeren Anzahl von Plänen, die für ein Modell benötigt werden, liegt darin, dass die Speicherung von Plänen (siehe Abschnitt 3.4.3) und die Suche nach ihnen effektiver gestaltet werden könnte. Es lassen sich zwei mögliche Umsetzungen unterscheiden:

1. „Pessimistisch/Vorausschauend“: Spezielle Tests führen in den richtigen Abarbeitungszweig.
2. „Optimistisch/Probleme behebend“: Es wird zunächst der allgemeine Fall angenommen. Die Operationen erkennen degenerierte Fälle so wie es auch jetzt implementiert ist. Das Fehlschlagen der Berechnungen führt jedoch nicht zum Verwerfen des Planes, sondern zum Aufruf einer alternativen Sequenz[47]. Unter Umständen müssen dann jedoch bisherige Konstruktionen verworfen werden.

Bis zu einer Umsetzung sind jedoch noch umfangreiche Untersuchungen nötig. Dies gilt insbesondere für:

- die Berücksichtigung des domänen-spezifischen Wissens über die Spezialfälle und die jeweils sinnvollen Reaktionen bei der automatischen Erstellung der Verzweigungen,
- die effiziente Integration der (inkrementellen) Freiheitsgradanalyse bei der Planerstellung entsprechend den gefundenen Spezialfällen,
- die Beherrschung der Komplexität der Pläne, denn die Berücksichtigung einer neuen Verzweigung (d. h. einer neuen Freiheitsgradverteilung) führt zu weiteren Konstruktionen bei denen im Allgemeinen neue Spezialfälle auftreten und
- die Berücksichtigung der Verzweigungen bei der konsistenten Auswahl aus Mehrfachlösungen.

[46] Es sei denn, eine Konstruktion schlägt fehl.

[47] Hierfür müssten die Operationen die degenerierten Fälle in Rückgabewerten beschreiben. Zudem könnte es auch notwendig sein, in Spezialfällen, wie z. B. dem Schneiden zweier *gleicher* Kreise, die Konstruktion nicht lokal im Planeintrag auszuführen, sondern in einen anderen Zweig des Planes zu wechseln. So könnte der Freiheitsgrad des Punktes (er kann sich auf den zwei gleichen Kreisen frei bewegen) entsprechend der gültigen Priorisierung anders genutzt werden.

Schleife

Iterative Konstruktionen führen zu Schleifen in der Abarbeitung. Eine ausführliche Behandlung von iterativen Konstruktionen, die letztlich Schleifen darstellen, wird im Abschnitt 3.6 vorgenommen.

Bei den in Ficucs implementierten iterativen Konstruktionen werden die einzelnen Berechnungen pro Durchlauf jeweils auf die gleichen Objekte angewendet. Grundsätzlich bestünde aber auch die Möglichkeit bei jedem Durchlauf eine andere Menge von Objekten zu bearbeiten. Die Anzahl der Objektmengen könnte hierbei durchaus variabel sein. Dies wäre wichtig für die Umsetzung existenzbestimmender Constraints (vgl. Abschnitt 2.3). In den einzelnen Konstruktionen müsste dann jedoch die Umschaltung auf jeweils andere Objekte vorgenommen werden.

Unterprogramme

Unterprogramme wurden noch nicht als spezielle Planeinträge realisiert. Die einzelnen Operationen können jedoch auch als Unterprogramme betrachtet werden, denn letztlich wird in ihnen jeweils eine C++-Codesequenz aufgerufen.

Lokale Objekte und Parameter

Insbesondere für Hilfskonstruktionen (in der jetzigen Version des Constraint-Solvers noch nicht realisiert) wäre die Verwaltung von Hilfsobjekten bzw. -parametern im Plan sinnvoll. Der Bedarf an derartigen Hilfsobjekten hängt grundsätzlich vom jeweiligen Plan ab. Beim Verwerfen eines Planes müssen die entsprechenden Hilfsobjekte auch verworfen (implementierungstechnisch: freigegeben) werden und andererseits sind die Hilfsobjekte eines neuen Planes wieder bereitzustellen. Die explizite Modellierung von Hilfsobjekten als Planeinträge würden den Wechsel zwischen verschiedenen Plänen vereinfachen, denn beim Speichern und Laden der Pläne käme es zu einer automatischen Berücksichtigung der Hilfsobjekte. Dies gilt insbesondere auch für die externe Verwaltung der Pläne in einer Anwendungssoftware, da hier interne Hilfsobjekte des Constraint-Solvers nicht bekannt sind.

3.4.2 Erstellung von Plänen

Suche nach einer geeigneten Berechnungssequenz

In Veröffentlichungen über die Erstellung von Plänen konstruktiver Constraint-Solver lassen sich zwei grundlegende Vorgehensweisen erkennen, die beide einen entgegengesetzten Ansatz verfolgen. Bereits in [Sut63a] wurde eine Vorgehensweise („*one pass method*“) beschrieben, mit der in Sketchpad eine Sequenz direkter Konstruktionsschritte ermittelt werden konnte. Die Grundidee lässt sich etwas allgemeiner wie folgt formulieren:

1. Suche ein Objekt o, dessen Freiheitsgrad gleich oder größer der Summe der angrenzenden Valenzen (d. h. der Summe der Valenzen der angrenzenden Constraints) ist.
2. Wenn kein solches Objekt gefunden wurde, dann stoppe die Suche. In Sketchpad wurde in einem solchen Fall für die Berechnung der Modelle auf ein Relaxationsverfahren zurückgegriffen.
3. Speichere o sowie die mit ihm verbundenen Constraints $c_1 \dots c_n$ auf einem Stack S.
4. Entferne o inklusive aller mit ihm verbundenen Constraints $c_1 \dots c_n$ aus dem zu untersuchenden Graphen.

5. Gehe so lange zu Schritt 1, bis der Graph leer ist.
6. Entnehme das zuletzt in S gespeicherte Objekt o sowie die entsprechenden Constraints und konstruiere o auf Basis dieser Constraints.[48]
7. Gehe zu Schritt 6 bis S leer ist.

Die gleiche Vorgehensweise wurde auch in weiteren Veröffentlichungen beschrieben, u. a. in [Tod89].

Im vorgestellten Constraint-Solver wird der Konstruktionsplan in umgekehrter Weise aufgestellt. Dies wird in der Literatur u. a. als *„Sequenzielle Rekonstruktion“* beschrieben ([Rol95]). Der Unterschied zum obigen Ansatz besteht darin, dass hier von bekannten Objekten (z. B. fixiert, durch die Freiheitsgradanalyse als frei wählbar erkannt oder in einem vorhergehenden Schritt konstruiert) ausgegangen wird. Von dort erfolgt über nutzbare Constraints (solche in denen nur ein Objekt unbekannt ist) die schrittweise Ermittlung eines als nächstes konstruierbaren Objektes. Dies entspricht einer Serialisierung des bei der Freiheitsgradanalyse ermittelten Abhängigkeitsgraphen ausgehend von den unabhängigen Objekten. Ein ausführliches Beispiel für die Erstellung einer direkten Konstruktionssequenz ist im Anhang A.4.1 gezeigt.

Auswahl eines Planeintrages bzw. Konstruktors

Für ein als konstruierbar eingestuftes Objekt o sei C die Menge der für die Konstruktion nutzbaren Constraints und Hinweise. Ein Constraint bzw. Hinweis gilt als nutzbar, wenn er nur noch ein nicht konstruiertes Objekt (also o) referenziert. Während das Finden von o und die Bildung der Menge C im *generischen* Teil des Constraint-Solvers abläuft, fällt die Entscheidung, welcher Konstruktor für o verwendet werden soll, im *aufgabenspezifischen* Teil (siehe Abbildung 2.14).

Bezüglich der Menge C kann o über-, unter- oder vollbestimmt sein. Bei Überbestimmtheit kann aus C eine Untermenge C_K ausgewählt werden. Der Rest $C_O = C - C_K$ wird im Konstruktor vermerkt (z. B. *XHS*-Hinweise) oder wirkt direkt auf die Auswahl der Constraints, auf denen die Konstruktion beruht (vgl. Abschnitt 2.6):

- alternative Constraints:

  Entsprechend der Freiheitsgradanalyse dürfen nicht alle Constraints einer Menge von alternativen Constraints benutzt werden. Belassene Freiheitsgrade sind bei der Konstruktion unbedingt zu beachten und auch die Auswahl der für die Konstruktion benutzten Constraints muss der Freiheitsgradanalyse entsprechen.

- Ebenen-Scharen:

  Bei der Ermittlung eines Konstruktors für einen Punkt p wird C darauf untersucht, ob ein Hinweis gegeben ist, dass „Punkt in Ebene“-Constraints aus C Ebenen e_i referenzieren, die einer Ebenen-Schar angehören. Gibt es einen solchen Hinweis, so werden höchstens zwei Ebenen aus einer Schar gewählt (siehe auch Abbildung 2.8).

Optimierung

Grundsätzlich besteht die Möglichkeit, einen Plan durch Umsortierung seiner Einträge zu optimieren:

- Iterative Konstruktionen sollten möglichst kurz sein. Deshalb werden bei einer Optimierung alle Operationen, die nicht für die Berechnung des Fehlerwertes nötig sind, hinter die iterative Konstruktion platziert.

[48] In Schritt 1 wurde sichergestellt, dass im allgemeinen Fall die Konstruktion erfolgreich ist.

- Eine Konstruktion k, die aufgrund von degenerierten Fällen fehlschlagen kann, sollte in einer Sequenz möglichst weit nach vorn gestellt werden[49]. Dies gilt insbesondere in iterativen Konstruktionen.

Neben einer automatischen Optimierung nach den oben vorgestellten Gesichtspunkten, kann die Optimierung auch durch den Nutzer erfolgen (siehe externe Modifikation in Abschnitt 3.4.3).

3.4.3 Speicherung von Plänen

In der aktuellen Realisierung des Constraint-Solver-Moduls wird die Möglichkeit der Speicherung von erstellten Plänen noch nicht genutzt. Erste Implementierungsarbeiten und Tests hierzu sind jedoch bereits vorgenommen worden. Es gibt verschiedene Fälle, in denen die Speicherung von Plänen sinnvoll sein kann. Sie sollen im folgenden Text erörtert werden.

Sparen von Erstellungszeit

Bei unterbestimmten Modellen kommt es grundsätzlich zu einer kombinatorischen Vielzahl von möglichen Freiheitsgradverteilungen. Wenn nun die ersten Freiheitsgradverteilungen zu Plänen führen, die nicht erfolgreich ausgeführt werden können (weil degenerierte Fälle auftreten - sich z. B. Kreise nicht schneiden) kann die Zeit bis zum Finden eines passenden Planes unerwünscht lang werden.

Unter Umständen wurde für das Modell (z. B. bei der Erstellung und den anschließenden Tests des Modells) bereits ein passender Plan erstellt. Wenn dieser wesentlich schneller gefunden werden kann, als die Erstellung dauert, dann wäre eine Speicherung sinnvoll.

Planexport in andere Formate

Denkbar ist z. B. der Export als:

- ein kompilierbares Programm, z. B. in C++ oder Java,
- ein Skript für einen Interpreter wie z. B. GML (Generative Modelling Language, siehe [Hav05]),
- eine Konstruktionssequenz in einem Format von history-basierter Geometriesoftware[50] wie z. B. Cinderella ([RGK00]) und Geometer's Sketchpad ([Ben03]), wobei die Unterstützung iterativer Konstruktionen hier grundsätzlich nicht zu erwarten ist, oder
- eine textuelle Beschreibung des Plans zur Analyse der Arbeitsweise des Constraint-Solvers.

Externe Modifikation

Eine externe Modifikation könnte in Spezialfällen zu den effektivsten Berechnungssequenzen führen. Die Vorgehensweise wäre die folgende:

1. Das Constraint-Modul erstellt nach seiner im Allgemeinen sinnvollen Heuristik einen Plan, der extern gespeichert wird (beim nächsten Mal könnte dieser Plan im Constraint-Modul genutzt werden).

[49] D. h. möglichst gleich nach dem Eintrag, durch den das letzte in k benötigte Objekt konstruiert wird.
[50] Derartige Software wird auch als Software für „*dynamische Geometrie*“ bezeichnet.

2. Aufgrund von speziellen Anforderungen wird jedoch eine andere (z. B. robustere) Berechnungssequenz gewünscht. Deshalb modifiziert der Ersteller des Modells den Plan nach seinen Vorstellungen, z. B. indem er andere Operationen verwendet und die Reihenfolge der Planeinträge vertauscht.
3. Er speichert den modifizierten Plan zum Modell wieder ab.
4. Beim nächsten Laden des Modells wird der Constraint-Solver den modifizierten Plan verwenden.

3.4.4 Umverteilung von Freiheitsgraden zur Iterationsvermeidung

In der Praxis gab es wiederholt Beispiele dafür, dass eine ungünstige Freiheitsgradverteilung zu Iterationen führte. In einigen von diesen Fällen lassen sich die Iterationen durch eine nachträgliche Umverteilung von Freiheitsgraden vermeiden. Der Ablauf ist hierbei wie folgt:

1. Eintreten des Problemfalls: Auf Basis einer Freiheitsgradverteilung wurde begonnen einen Plan zu erstellen. Dieser konnte jedoch nicht abgeschlossen werden, da unter Berücksichtigung der gegebenen Freiheitsgradverteilung kein Objekt konstruierbar[51] ist.
2. Es wird untersucht, ob es unter den noch nicht konstruierten[52] Objekten zumindest ein Objekt gibt, das entsprechend der Freiheitsgradverteilung einen Freiheitsgrad aufweist. Ist dies nicht der Fall, kann keine Umverteilung vorgenommen werden, d. h. entweder wird nun ein iterativer Plan erstellt oder der bisherige Plan wird verworfen[53] und die Planerstellung mit der nächsten Freiheitsgradverteilung (siehe Abschnitt 3.3.4 und folgende) gestartet.
3. Wenn ein Objekt mit einem Freiheitsgrad gefunden wird, so wird für alle noch nicht konstruierten Objekte und die sie referenzierenden Constraints ein bipartiter Graph erstellt. Innerhalb dieses Graphen kann nun versucht werden, eine Umverteilung des vorhandenen Freiheitsgrades vorzunehmen, so dass ein Objekt konstruierbar wird und somit die direkte Konstruktion fortgesetzt werden kann.

Diskussion des Ansatzes

Bei dem Ansatz handelt es sich um eine Heuristik, die in einigen Fällen zum Erfolg führt. Die Erstellung des bipartiten Graphen und die Umverteilung der Freiheitsgrade kostet aber Rechenzeit, so dass sich jedes Fehlschlagen negativ auf die Interaktivität auswirkt. Zudem wird die Priorisierung der Vermeidung von Iterationen untergeordnet, d. h. der Nutzer kann Änderungen am Modell ggf. sehr flüssig vornehmen (weil Iterationen vermieden worden sind). Aufgrund der nicht beachteten Priorisierung sind die Änderungen für ihn jedoch schlecht nachvollziehbar.

Um die genannten Nachteile zu vermeiden, kann es sinnvoll sein die Verteilung nur unter gleich priorisierten Objekten vorzunehmen oder Gruppen von Prioritäten zuzulassen, innerhalb denen eine Umverteilung zulässig sein soll. Hierdurch könnte zum einen die Nachvollziehbarkeit der Änderungen gewährleistet werden und zum anderen würde der Suchraum und damit die möglicherweise umsonst verbrauchte Suchzeit für eine Umverteilung begrenzt werden.

---

[51] Konstruierbar soll in diesem Kontext heißen, dass ein Konstruktor gefunden werden kann, der dann in den Plan aufgenommen (bzw. einfach an ihn angehängt) werden kann.

[52] Konstruiert heißt hier, dass sich ein Konstruktor für das Objekt bereits im Plan befindet. Nicht konstruierte Objekte haben dementsprechend noch keinen Konstruktor im Plan.

[53] Zum Beispiel weil keine (weitere) Iteration erlaubt ist. Diese Situation kann in interaktiven Anwendungen auftreten, denn zur Gewährleistung von interaktiven Berechnungsgeschwindigkeiten wird hier die Iterationstiefe sinnvollerweise begrenzt.

3.5 Direkte Konstruktionen

Den Kern aller Berechnungen bildet im vorgestellten Constraint-Modul die Ausführung direkter Konstruktionen. Wie in Abschnitt 3.4.1 erörtert ist die Anzahl der im Constraint-Modul zu implementierenden direkten Konstruktionen ein kombinatorisch Vielfaches bezüglich der Anzahl der Freiheitsgrade der zu konstruierenden Objekte sowie der Anzahl der unterstützten Constraints (Gleichung 3.5). Im *2D*- und *3D*-Modul ergeben sich insgesamt ca. 120 Konstruktoren. Hinzu kommen noch ca. 50 Berechnungen, die als Start bzw. Ende von iterativen Konstruktoren dienen.

In diesem Abschnitt sollen ausgesuchte direkte Konstruktionen näher erörtert werden. Hierbei werden beispielhaft Probleme der Behandlung von:

- Unterbestimmtheiten,
- Berechnungsrichtungen,
- degenerierten Fällen (bzw. den möglichen Spezialfällen) und
- Mehrfachlösungen

erörtert.

3.5.1 Ausgewählte Gleichungs-Constraints

Der Gleichungs-Constraint $y = m \cdot x + n$

Der Gleichungs-Constraint $y = m \cdot x + n$ ist wegen der Vielzahl der möglichen Berechnungen interessant, die er je nach zu berechnendem Parameter hervorrufen kann. Jeder der vier Parameter kann im Laufe der Planerstellung als zu berechnend erkannt werden. Wenn die Mehrfachreferenzierung von Parametern in Constraints verboten wäre, würden sich daraus vier Berechnungsrichtungen ergeben. Im vorgestellten Constraint-Solver ist die Mehrfachreferenzierung jedoch erlaubt und wird insbesondere beim Gleichungs-Constraint $y = m \cdot x + n$ auch bewusst zur Modellierung spezieller Abhängigkeiten eingesetzt (vgl. Modellierung der Quadratwurzel im Abschnitt 5.1.3). Entsprechend der möglichen Berechnungsrichtungen und den aus unterschiedlichen Mehrfachreferenzierungen resultierenden Referenzierungsschemen ergeben sich die in Tabelle 3.1 gezeigten fünfzehn Fälle. Für einige der Fälle sind ggf. noch Spezialfälle zu berücksichtigen, die aus den jeweils zur Berechnung dienenden Werten der Parameter resultieren können. Als Beispiel hierfür soll der Fall 12 dienen. In ihm gilt (nach Umformung) $0 = a \cdot b_x$. Das heißt, dass entweder $a = 0$ zu berechnen ist oder aber wenn $b_x = 0$ ist, a beliebig gewählt werden kann (z. B. unverändert gelassen wird).

Mehrfachlösungen bei Betragsberechnung und Normalisierung

Die Betragsbestimmung $a = abs(x)$ ist je nach Berechnungsrichtung eindeutig (Berechnung von a) oder zweideutig (Berechnung von x). Um das Vorzeichen von x zu bestimmen gibt es unterschiedliche Möglichkeiten:

- Beibehalten des alten Vorzeichens,
- Nähe zum vorhergesagten Wert (entsprechend den n letzten Werten und einer zu wählenden Extrapolationsstrategie) oder
- entsprechend gegebenen Hinweisen.

Tabelle 3.1: Fälle der Berechnung von Parametern im Gleichungs-Constraint $y = m \cdot x + n$ in Abhängigkeit vom zu berechnenden Parameter und dem Referenzierungsschema. Die Bezeichner in den Referenzierungsschemen wurden so gewählt, dass stets a zu berechnen ist. Die unabhängigen Variablen b_y, b_m, b_x und b_n können gegebenenfalls mit dem gleichen Parameter b belegt sein. Dies ist für die Berechnung ohne Bedeutung. Die Fälle 5 bis 15 ergeben sich durch Mehrfachreferenzierungen des zu berechnenden Parameters a.

Fall	Referenzierungsschema	
1	$a = b_m \cdot b_x + b_n$	
2	$b_y = a \cdot b_x + b_n$	
3	$b_y = b_m \cdot a + b_n$	
4	$b_y = b_m \cdot b_x + a$	
5	$a = a \cdot b_x + b_n$	
6	$a = b_m \cdot a + b_n$	
7	$a = b_m \cdot b_x + a$	
8	$b_y = a \cdot a + b_n$	(Quadratwurzel)
9	$b_y = a \cdot b_x + a$	
10	$b_y = b_m \cdot a + a$	
11	$a = a \cdot a + b_n$	(Quadratwurzel)
12	$a = a \cdot b_x + a$	
13	$a = b_m \cdot a + a$	
14	$b_y = a \cdot a + a$	(Quadratwurzel)
15	$a = a \cdot a + a$	(a ist stets 0)

Eine ähnliche Betrachtung ist für Normalisierungsoperationen nötig. Sie werden z. B. für die Modellierung von Rädergetrieben benutzt[54].

3.5.2 Ausgewählte 2D-Konstruktoren

Die wohl interessanteste Anwendung des *2D*-Constraint-Moduls ist MASP (vgl. Abschnitt 5.2.3). Bei der automatischen wie auch interaktiven Bewegung von Mechanismen und Getrieben stellt die korrekte Auswahl von Lösungen immer wieder ein nicht zu unterschätzendes Problem dar. Deshalb sollen hier exemplarisch einige Konstruktoren besprochen werden, deren sorgfältige Umsetzung für eine der Aufgabenstellung angepasste Lösungsfindung wichtig ist. Ein umfangreicherer historischer Abriss zum Umgang mit Mehrfachlösungen inklusive einer Diskussion der Ansätze ist in Kapitel 4 gegeben.

Initiale Konfigurationsbestimmung

Konstruktoren berücksichtigen bei ihren Berechnungen einen oder mehrere Constraints mit dem Ziel, dass nach der Berechnung des jeweiligen Objektes diese Constraints erfüllt sind. Spielen die in einem Constraint C referenzierten Objekte unterschiedliche Rollen[55], so müssen bei der Initialisierung eines Konstruktors K, der C berücksichtigen soll, diese Rollen festgestellt werden[56]. Geschieht das nicht, so ist die Konsistenz der Berechnungsergebnisse nicht mehr gewährleistet.

---

[54] Die Notwendigkeit ergibt sich z. B. aus der Berechnung von normalisierten Winkeln α_{norm} mit $-\Pi < \alpha_{norm} \leq +\Pi$ als aktueller Winkel zwischen zwei Geraden. Für kontinuierliche Bewegungen von verbundenen Zahnrädern werden jedoch akkumulierte Winkel α_{akku} benötigt, diese sind in ihrem Wertebereich nicht begrenzt.

[55] im Constraint

[56] Beispiele für derartige Constraints sind in *2D* „Anstieg von einem Punkt zu einem anderen“ sowie „paralleler Abstand zwischen zwei Geraden“ und in *3D* „paralleler Abstand zwischen zwei Ebenen“ sowie „Symmetrie zwischen drei Ebenen“ (hier spielt die Symmetrieebene eine spezielle Rolle).

Beispiel 3.19: Initialisierung einer Konstruktion, die auf einem 2D-Winkel-Constraint beruht

In *2D* ist ein Winkel α zwischen zwei Richtungen R_a und R_b vorzeichenbehaftet. Dies hat z. B. den Vorteil, dass sich bei bekanntem α die Richtung R_b eindeutig aus R_a bestimmen lässt[57]. Da $\alpha = Winkel(R_a, R_b) = -Winkel(R_b, R_a)$ gilt, muss das Vorzeichen des Wertes von α invertiert werden, falls R_b bekannt ist und R_a konstruiert werden soll. Diese Notwendigkeit der Vorzeichenumkehr wird im Konstruktor vermerkt und bei den späteren Berechnungen berücksichtigt. Dies gilt nicht nur für den Konstruktor einer Richtung R_a bzw. R_b, auch der Konstruktor eines Winkels α muss bei der Berechnung des aktuellen Wertes berücksichtigen, wie α in C benutzt wurde und das Vorzeichen entsprechend wählen. □

Sicherung der Starrheit

Ein wichtiges Mittel zur korrekten Auswahl aus Mehrfachlösungen ist die Definition bzw. die spätere Berücksichtigung von *XHS*-Hinweisen. In *2D* ist es so möglich, das Umschlagen von Dreiecken zu verhindern. Das Problem des Umschlagens erwächst dadurch, dass es interaktiv leicht möglich ist zwischen Punkten Abstands-Constraints zu definieren. Wurden von einem Nutzer drei Punkte zu einem Dreieck versteift, so erwartet dieser typischerweise nicht, dass der Constraint-Solver ihm im Zuge der Berechnungen das gespiegelte Dreieck als Lösung anbietet. Aus Sicht des Constraint-Solvers ist die gespiegelte Lösung jedoch gültig, denn auch sie hält die Abstands-Constraints ein. Hier ist es sinnvoll, dass die Applikation selbständig *XHS*-Hinweise erzeugt[58] oder aber der Nutzer hat die volle Kontrolle und vergibt diese Hinweise selbst.

Die nutzbaren Hinweise werden bei der Erstellung des Konstruktors in ihm vermerkt (siehe Abschnitt 3.4.2). In den Berechnungen haben die Konstruktoren die für sie gültigen Hinweise bei der Wahl aus Mehrfachlösungen nun auszuwerten, z. B. beim Schneiden von Kreis und Gerade, beim Schneiden von zwei Kreisen, aber auch wenn der Punkt einen Freiheitsgrad hat und sich frei auf einem Kreis oder einer Geraden bewegen kann[59]. Zu beachten ist hierbei, dass Hinweise im Allgemeinen nicht zwangsläufig zur Lösungsfindung beitragen, siehe Abbildung 3.19.

Simulation von Trägheit

Die Simulation von Trägheit bzw. kontinuierlicher Bewegung muss in Ficucs entsprechend geometrischen Kriterien erfolgen[60]. Wie das geschehen kann, soll anhand der Konstruktion eines Punktes P durch das Schneiden zweier Kreise $K_a$und K_b gezeigt werden. Die Konstruktion impliziert, dass die Mittelpunkte P_a und P_b der Kreise und die Radien (bzw. Abstände) D_a und D_b bekannt sind. Im Kontext von MASP (wo kontinuierliche Bewegungen von großer Bedeutung sind) könnte es sich bei den Kreismittelpunkten um Drehgelenke, bei den Radien um die Längen zweier Getriebeglieder und beim zu konstruierenden Punkt um ein weiteres Drehgelenk handeln. In einem typischen Anwendungsfall bewegen sich P_a und P_b, die Abstände D_a und D_b sind hingegen fix. Wenn P_a und P_b bei der Bewegung in der Ebene ihre relative Position zueinander (d. h. ihren Abstand D_{ab}) nicht ändern, könnte ein *XHS*-Kriterium zur Lösungsauswahl herangezogen werden (siehe vorheriger Abschnitt). Im Allgemeinen wird der Abstand D_{ab} jedoch nicht konstant bleiben. Zur Auswahl sind verschiedene Kriterien denkbar:

---

[57] Würden häufig Fälle auftreten, in denen explizit beide Winkel zulässig sind, das Vorzeichen also irrelevant ist, so müsste auch hier aus beiden Lösungen ausgewählt werden. In den bisher bearbeiteten Modellen war dies jedoch nicht erwünscht. Starrheit oder die Eindeutigkeit von Antriebswinkeln sind in den Applikationen von großer Bedeutung.

[58] Oder die Applikation weist den Constraint-Solver an dafür zu sorgen, dass das LHS bzw. RHS jeweils erhalten bleibt und auch nicht im Zuge einer Variantenbildung das Dreieck umschlagen lässt.

[59] Dies gilt auch, wenn der Punkt als Beginn einer iterativen Konstruktion testweise konstruiert wird.

[60] Denkbar ist jedoch, dass bestimmte Entscheidungen extern (z. B. physikbasiert) getroffen werden. Geometrisch formuliert könnte diese Entscheidung Ficucs für die Auswahl der Mehrfachlösungen bereitgestellt werden.

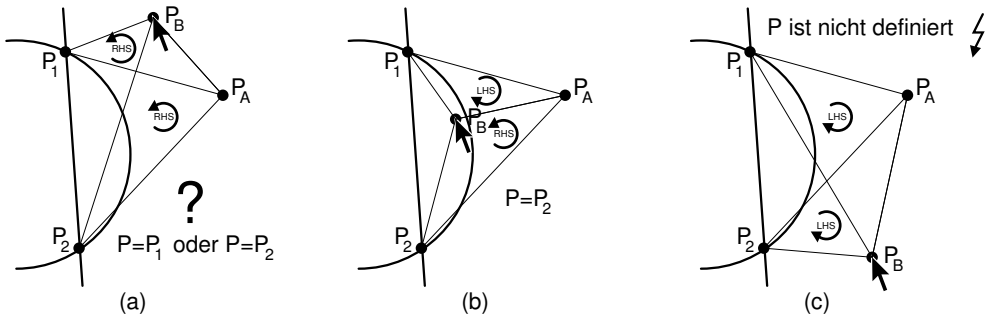


Abbildung 3.19: Fälle, die bei der Auswertung eines *RHS*-Hinweises während der Punktkonstruktion (hier das Schneiden von Gerade und Kreis) auftreten können. Der Punkt P_B wird für die drei Fälle jeweils an eine andere Position geschoben. Es ist ersichtlich, dass die getestete Punktmenge $\{P_A, P_B, P\}$ *nicht* starr ist. (a) Der Hinweis bringt keine Entscheidung. Beide Punkte sind möglich. (b) Nur ein Punkt (P_2) erfüllt den Hinweis. (c) Keiner der beiden Punkte erfüllt den Hinweis, denn es ergibt sich stets ein *LHS*.

- Abstand D_{ab} und XHS_{alt}
  Die Konstruktion könnte so lange die alte *XHS*-Eigenschaft beibehalten[61], bis der Abstand D_{ab} entsprechend den Dreiecksungleichungen zu groß bzw. zu klein wird und somit keine reellen Lösungen mehr gefunden werden können. Für die nächste konstruierbare Lösung wird die *XHS*-Eigenschaft invertiert, d. h. *LHS* wird zu *RHS* und umgekehrt. Dieses Verhalten ist in [End90] beschrieben (siehe auch Abschnitt 4.1, Wechsel zwischen zwei Lösungen nach inkonsistenten Zuständen). Dieses Verhalten ist gut für kleinere Modelle geeignet, bei denen der Antrieb im Falle eines inkonsistenten Antriebswinkels Konterbewegungen ausführt, d. h. seine Drehrichtung kehrt sich im Falle einer (Über-)Streckung um.

- XHS_{alt}
  Die alte (bzw. eine bestimmte) *XHS*-Eigenschaft könnte auch stets beibehalten werden. Dies würde z. B. Sinn machen, wenn das Drehgelenk ein Scharnier ist, dass sich nur in eine Richtung klappen lässt.

- Extrapolation aus den letzten n Werten des Abstandes D_{PG}
  D_{PG} sei der vorzeichenbehaftete Abstand von P zur gerichteten Kriteriumsgerade G, die durch P_a und P_b (in dieser Reihenfolge) verläuft. In einem *RHS* (P ist links von G) sei D_{PG} positiv und in einem *LHS* (P ist rechts von G) negativ. Speichert man die letzten n Werte von D_{PG}, so lässt sich aus ihnen der nächste Wert von D_{PG} abschätzen, z. B. durch lineare Extrapolation bei Verwendung der letzten zwei Werte. Für die Berechnung ist eine Annahme über die Ursache der letzten Abstandswerte zu machen. Wenn sie nicht von einem Antrieb konstanter Geschwindigkeit verursacht wurden, dann muss die Abhängigkeit der gemessenen Abstände von einer Antriebsvariablen (z. B. einem Antriebswinkel) berücksichtigt werden. Allgemein lautet der lineare Extrapolationsansatz:

$$\frac{y_k - y_{k-1}}{x_k - x_{k-1}} = \frac{y_{k-1} - y_{k-2}}{x_{k-1} - x_{k-2}} \tag{3.9}$$

  wobei y_k der zu schätzende Abstand und x_k der aktuelle Antriebsparameter ist. Die Indizes $k-1$ und $k-2$ kennzeichnen die letzten bzw. vorletzten festgestellten Werte. Bei konstanter Antriebsgeschwindigkeit gilt $x_k - x_{k-1} = x_{k-1} - x_{k-2}$. Hierdurch vereinfacht sich die Abschätzung zu $y_k = 2 * y_{k-1} - y_{k-2}$, d. h. ohne Kenntnis eines Antriebswinkels

[61] D. h. die Konstruktion wählt P so, dass $\{P_a, P_b, P\}$ in einem *LHS* oder *RHS* bleiben, je nachdem was initial vorlag.

und unter Annahme einer kontinuierlichen Bewegung lässt sich D_{PG} durch lineare Extrapolation als $D_{PG} = 2 * D_{PG\,k-1} - D_{PG\,k-2}$ abschätzen. Ist diese Abschätzung positiv (also $2 * D_{PG\,k-1} > D_{PG\,k-2}$), dann wird ein *RHS* angenommen, ansonsten ein *LHS*.

Diese Berücksichtigung der Annäherung an die Strecklage, ist eine sehr abstrakte Form der Simulation von Trägheit bzw. Kontinuität der Bewegungen. Da die Abschätzung jedoch nur für die Auswahl aus den beiden möglichen Punkten verantwortlich ist[62], ist das Resultat meist plausibel.

3.5.3 Ausgewählte 3D-Konstruktoren

Schwerpunkt der Beispiele für *3D*-Planeinträge soll die Behandlung von Mehrfachlösungen sowie der Umgang mit degenerierten Fällen sein.

Umgang mit degenerierten Fällen

Für eine Darstellung des Umgangs mit degenerierten Fällen werden hier Planeinträge betrachtet, die Ebenen berechnen. Eine Auswahl derartiger Berechnungen ist in Abbildung 3.20 zueinander in Beziehung gesetzt. Aus der Abbildung wird ersichtlich, wie sich degenerierte Fälle durch den Aufruf anderer Berechnungen behandeln lassen. Eine solche Aufrufheterarchie[63] senkt den Implementierungsaufwand für die einzelnen Konstruktionen erheblich und senkt somit die Gefahr, bestimmte Spezialfälle unberücksichtigt zu lassen.

Behandlung von Mehrfachlösungen

Zur Behandlung von Mehrfachlösungen für zu berechnende Ebenen werden den in Abbildung 3.20 gezeigten Konstruktoren Referenzrichtungen oder wo sinnvoll auch Referenzebenen[64] übergeben. Im folgenden Text soll allgemein nur von Referenzobjekten gesprochen werden. Standardmäßig werden Referenzobjekte aus den jeweils zuletzt gültigen Objektwerten gewonnen - im einfachsten Fall *kopiert*, denkbar ist aber auch eine *Extrapolation* (vgl. Seite 70). Weitaus interessanter (und deshalb auch an dieser Stelle näher behandelt) ist jedoch die Wahl von Referenzwerten auf Basis von *Hinweisen*. Als sehr wichtige Hinweise für die Animation von *3D*-Modellen haben sich die sogenannten *XHS*-Hinweise erwiesen. Wie das folgende Beispiel zeigt, können sie aber auch für statische (d. h. nicht animierte) Modelle bedeutsam sein.

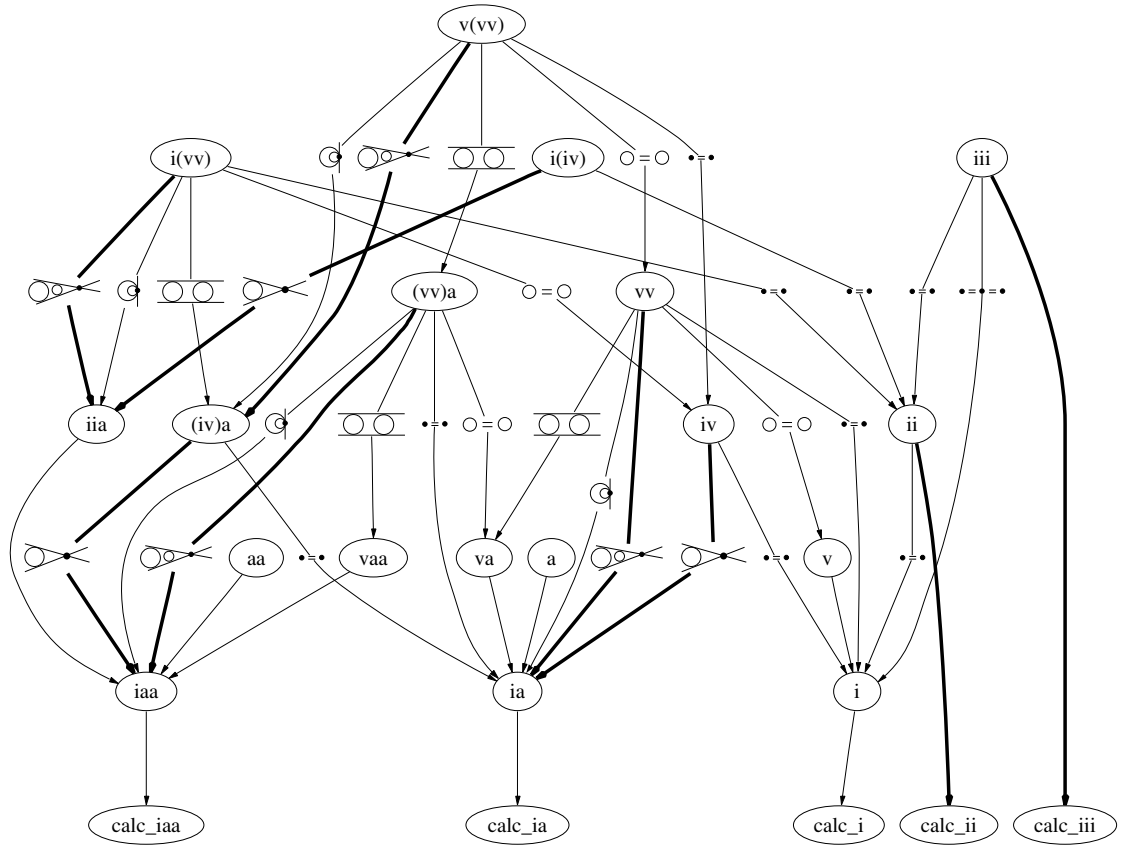
Beispiel 3.20: Einsatz von Hinweisen bei der Modellierung einer Wendeltreppe

Es soll eine Wendeltreppe modelliert werden, bei der die einzelnen Stufen aus Quadern bestehen. Die erste Stufe könnte in einer bestimmten Position fixiert werden, in Abbildung 3.21 wurde jeweils der untere Quader fixiert. Alle folgenden Stufen sind bei einer Wendeltreppe nun nicht parallel zu dieser ersten Stufe, sondern nehmen einen bestimmten Winkel α zur jeweils vorigen Stufe ein. In diesem Beispiel ist das der Winkel zwischen den Normalen der Vorderseiten der Stufenblöcke. Die Konstruktion der zweiten Normalen N_2 ist jedoch nicht eindeutig. Neben der in Abbildung 3.21(a) gezeigten Lösung existiert noch eine Normale N_2', die auch rechtwinklig zu N_0 ist und zu N_1 den Winkel α einnimmt (Abbildung 3.21(b)). Ohne die Angabe von Hinweisen, wählt der Constraint-Solver die Lösung jeweils nahe den alten Werten der einzelnen geometrischen Objekte. Typischerweise haben die geometrischen Objekte nach ihrer Erzeugung Standardwerte, für die Bestandteile der Quader könnte das entsprechend einer achsenparallelen Ausrichtung sein. Würde in einem solchen Fall für die Konstruktion der Normalen der Vorderseiten die Lösung

[62] Die Berechnung der möglichen Punkte erfolgt „exakt“ auf Basis der geometrischen Constraints.

[63] auch als DAG bezeichnet (engl.: *d*irected *a*cyclic *g*raph)

[64] Bei den meisten Konstruktionen liegen stets (auch in degenerierten Fällen) Informationen über den Offset der Ebene (ihr Positionsfreiheitsgrad) vor, d. h. die Angabe einer Referenzrichtung ist hier ausreichend. In Abbildung 3.20 wird die Referenzebene nur für den *aa*-Konstruktor benötigt.



nutzbare Relation		mögliche Fälle, resultierende Relationen			
i	Punkt in Ebene (die Ebene geht durch den Punkt)		Kegel aus 2 Kugeln, ia		Ebene, ia
v	Abstand Punkt/Ebene (die Ebene ist tangential zu einer Kugel)		Kegel aus Kugel und Punkt, ia	○ = ○	2 gleiche Kugeln, v
a	Winkel Ebene/Ebene (die Ebene wird tangential zu einem Kegel)			• = •	2 gleiche Punkte, i
(...)	verwendete Relationen		Zylinder, da 2 gleichgroße Kugeln, va	• = • = •	3 gleiche Punkte, i

Abbildung 3.20: Heterarchie von Konstruktoren für Ebenen. Aus ein bis drei nutzbaren Relationen (z. B. *vvv* für die Distanzen zu drei üblicherweise verschiedenen Punkten) werden schrittweise weitere Relationen abgeleitet (z. B. $vvv \rightarrow iva$). Hierbei wird bei einigen Konstruktoren zunächst nur ein Teil der Informationen genutzt, in der Abbildung ist der genutzte Teil in Klammern gesetzt (z. B. werden beim Übergang von $v(vv)$ zu *iva* nur zwei Kugeln betrachtet, zu denen die Ebene tangential ist). In den meisten Konstruktoren sind Spezialfälle zu berücksichtigen. Treten mehrere Fälle auf, so ist die normalerweise stattfindende Ableitung in der Abbildung hervorgehoben (z. B. wird bei $vv \rightarrow ia$ aus der Tangentialität zu zwei Kugeln typischerweise die Tangentialität zu einem Kegel). In den Endknoten findet schließlich die eigentliche Berechnung der gesuchten Ebene statt (z. B. bei *calc_iaa* aus zwei Kegeln, die eine gemeinsame Spitze haben und zu denen die Ebene tangential ist).

In der Abbildung sind nur die Fälle dargestellt, die zu einer gültigen Konstruktion führen, Fehlerfälle hingegen nicht. Ein solchen Fehlerfall liegt z. B. vor, wenn beim *vv*-Konstruktor (die Ebene ist tangential zu zwei Kugeln) sich eine Kugel vollständig in der anderen befindet.

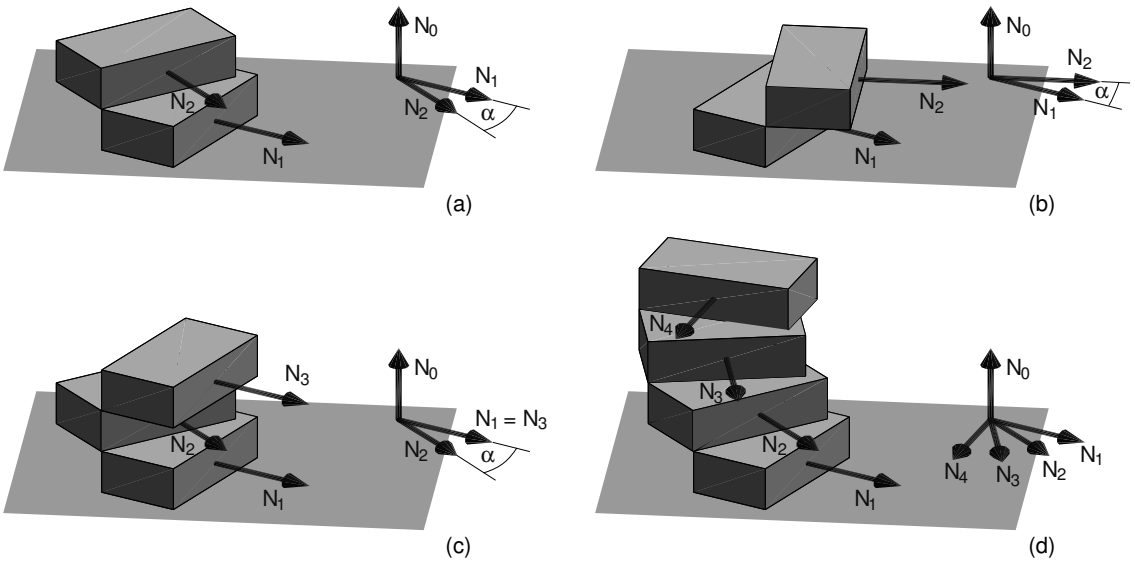


Abbildung 3.21: Ausrichtung von Stufen einer Wendeltreppe. (a) nach einem linkshändigen Koordinatensystem (*LHS*). (b) nach einem rechtshändigen Koordinatensystem (*RHS*). (c) typisches Resultat, wenn Mehrfachlösungen durch die „nahe der alten Lösung"-Regel ausgewählt werden. (d) typische Lösung, wenn bei der Modellierung Hinweise (hier *LHS*-Hinweise) gegeben und vom Constraint-Solver ausgewertet werden.

gewählt werden, die nahe der alten Lösung ist (d. h. hier nahe der initialen Lösung, die als Referenzobjekt dem Konstruktor übergeben wird), so wäre die in Abbildung 3.21(c) gezeigte Lösung das Resultat dieser unzulänglichen Modellierung. Erst wenn explizit der Hinweis auf das *LHS* zwischen den angrenzenden Normalen[65] gegeben wird, kann der Constraint-Solver die gewünschte Wendeltreppe berechnen (Abbildung 3.21(d)). □

3.6 Iterative Konstruktionen

Die wesentliche Stärke des in [Hsu96] und [HB97]vorgestellten hybriden Ansatzes ist die Verbindung von schnellen direkten Konstruktionen mit einem iterativen Ansatz zur Nullstellensuche. Die Grundidee besteht darin, zum einen möglichst nur direkte Konstruktionen zu verwenden, was zu einer schnelleren Planabarbeitung führt, und zum anderen auch während iterativen Konstruktionen auf direkte Konstruktionen zurückzugreifen, die dann allerdings mehrfach durchlaufen werden. Abbildung 3.22 zeigt die Grundstruktur eines hybriden Konstruktionsplanes. Vor und nach der iterativen Konstruktion werden n bzw. k Planeinträge ausgeführt. Die m Planeinträge der iterativen Konstruktion müssen mehrere Male durchlaufen werden, bis eine Nullstelle gefunden wurde[66].

Ähnliche Ansätze sind in [RBDH93] bzw. [RB98] (*late constraints*) sowie [HGY02][67] (*locus intersection*) beschrieben. Der Vorteil des von Hsu vorgestellten und in dieser Dissertation weiter entwickelten Ansatzes besteht in der zielgerichteten (d. h. zeitsparenden und robusten) Suche nach Lösungen in iterativen Plänen.

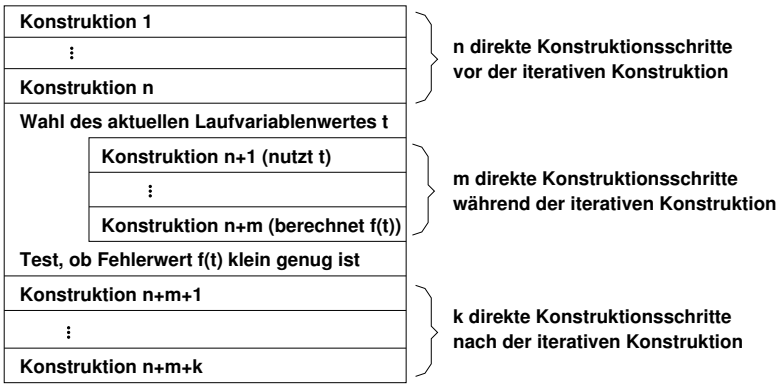


Abbildung 3.22: Schema eines hybriden Konstruktionsplanes

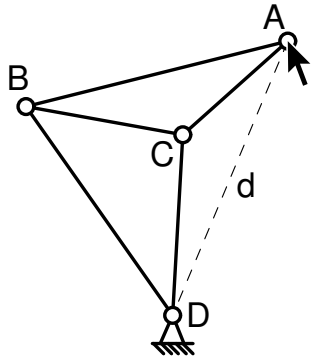


Abbildung 3.23: Wahl eines geeigneten Parameters d für einen iterativen Plan durch Hinzufügen eines neuen Constraints (mit freiem Abstandsparameter d, um das Modell nicht überbestimmt zu machen).

3.6.1 Aufstellung iterativer Pläne

Die Erstellung von Plänen mit ausschließlich direkten Konstruktionen bricht ab, wenn kein passendes Objekt (siehe Seite 63) oder anschließend kein passender Konstruktor (siehe Seite 64) mehr gefunden werden kann. Sind Iterationen erlaubt, beginnt nun die Suche nach einem Objekt, in dem genau ein Freiheitsgrad nicht bestimmt ist[68]. Für alle anderen Freiheitsgrade müssen Constraints nutzbar sein. Das Hauptproblem besteht in der Wahl eines geeigneten Objektes (bzw. des entsprechend zu testenden Parameters), denn bei ungünstiger Wahl sind u. U. unnötigerweise noch weitere Iterationen erforderlich[69]. Strategien zur Auswahl des Objektes könnten sein:

- Konstruiere das erste passende Objekt. Die Planerstellung geht so am schnellsten, allerdings kann sich der entsprechend geringere Aufwand in der Planerstellungsphase in einem viel höheren Berechnungsaufwand in der Planausführungsphase niederschlagen.
- Wähle das Objekt so, dass es durch möglichst viele Constraints referenziert wird. Nach seiner Konstruktion stehen oft sehr viele dieser Constraints für weitere Konstruktionen zur Verfügung[70] und die Wahrscheinlichkeit, dass dann direkte Konstruktionsschritte möglich sind, ist somit größer. Eine Garantie, so den besten Parameter zu finden, gibt es jedoch nicht. Gegebenenfalls kann es vorkommen, dass trotzdem ein Start weiterer Iterationen nötig ist, während die Wahl eines anderen Parameters diese vermieden hätte.
- Bevorzuge solche Objekte, für die Konstruktoren mit begrenztem Suchraum zur Verfügung stehen, z. B. solche, die ein Objekt über einen Winkelparameter auf einer Kreisbahn konstruieren. Demgegenüber sollten Konstruktoren mit unbegrenztem Suchraum gemieden werden, da hier das Erreichen eines Abbruchkriteriums für die erschöpfende Suche meist aufwändiger ist. Ein derartiger (im Fall eines Misserfolges ungünstigerer) Konstruktor könnte z. B. einen Abstandsparameter auf einer Gerade iterieren. Hier lassen sich die zulässigen Objektpositionen auf der Gerade ohne Zusatzinformationen grundsätzlich nicht einschränken. Mit Zusatzinformation ist das jedoch möglich, z. B. über Dreiecks- bzw. n-Ecks-Ungleichungen sowie über die Auswertung applikations- oder modellspezifischer Vorgaben für sinnvolle Objektpositionen.
- Wenn die Existenz von Clustern festgestellt werden kann, dann sollte möglichst ein Parameter in einem Cluster als Laufvariable für die Iteration gewählt werden. Der Vorteil bestünde darin, dass das Cluster bei sukzessiven Änderungen starr bleibt (zumindest wenn kein das Layout des Clusters bestimmender Parameter geändert wird). Somit würde sich auch der Parameter nicht ändern und die alte Lösung für die Laufvariable ist immer gleich der neuen, wodurch dann alle Objekte stets in der ersten Iteration korrekt konstruiert werden würden. Solche Parameter müssten aber eingeführt werden (interne Hilfsvariablen sowie entsprechende Hilfsconstraints, Abbildung 3.23), denn wenn sie schon da wären, hätte direkt konstruiert werden können (es sei denn es konnte wegen eines fehlenden Konstruktors nicht konstruiert werden). Eine Alternative wäre eine spezielle Berechnung des ersten zu testenden Wertes für die iterative Konstruktion (siehe Abschnitt 3.6.3).

Nach der Auswahl eines iterativen Konstruktors kann in der Regel wieder mit direkten Konstruktionsschritten weitergearbeitet werden, bis dann eine Überbestimmtheit für ein Objekt o festgestellt

---

[65] d. h. $\{N_0, N_1, N_2\}$, $\{N_0, N_2, N_3\}$ und $\{N_0, N_3, N_4\}$ bilden jeweils ein *LHS*.

[66] Ein Richtwert liegt bei etwa 20 Iterationen für kontinuierliche interaktive Änderungen.

[67] Zuvor wurde in [HY01] der Ansatz durch Hoffmann und Yuan kurz skizziert.

[68] Bisher sind alle in Ficucs implementierten Konstruktoren für Laufvariablen iterativer Konstruktionen univariat, d. h. sie starten stets eine univariate Nullstellensuche (siehe Abschnitt 3.6.3).

[69] Abbildung 3.24 zeigt eine derartige (nicht angestrebte) Verschachtelung der iterativen Konstruktionen.

[70] Das testweise zu konstruierende Objekt sei o. Ein Constraint c (der o referenziert) steht genau dann für die Konstruktion eines weiteren Objektes p zur Verfügung, wenn außer o und p alle in c referenzierten Objekte bereits konstruiert sind. Im einfachsten Fall ist c binär, d. h. o und p sind die einzigen Objekte, dann kann c bei bekanntem o für die Konstruktion von p benutzt werden. Häufig sind aber auch parametrisierte Constraints für Abstände und Winkel nach einer Konstruktion von o nutzbar, z. B. wenn der Parameter bereits bekannt ist.

wird. Sie zeigt das Ende einer iterativen Konstruktion an. Die Menge der für die Konstruktion von o nutzbaren Constraints C wird aufgeteilt in die Constraints C_K, die tatsächlich zur Konstruktion von o genutzt werden, einen Constraint c_T (vorzugsweise mit der Valenz von eins), der anschließend für den Test auf die Nullstelle ausgewertet wird und ggf. noch weitere Constraints $C_{\ddot{U}}$, deren Einhaltung zu testen ist. Wenn bei der Planaufstellung innerhalb eines iterativen Planes kein direkter Konstruktor mehr gefunden werden kann, so kommt es zu einer rekursiven Verschachtelung der iterativen Suche. Abbildung 3.24 zeigt das Schema eines entsprechenden Planes. Bei derartigen verschachtelten Plänen ist es möglich, dass Constraints aus $C_{\ddot{U}}$ auch weitere iterative Pläne (in der jeweils übergeordneten Rekursionsstufe) abschließen.

3.6.2 Abarbeitung iterativer Pläne

Die Abarbeitung eines einfachen iterativen Planes wurde bereits in Abschnitt 2.7 besprochen. Die gleiche Vorgehensweise gilt auch für (sequenziell) aufeinanderfolgende iterative Pläne. Hier sollen Strategien zur Abarbeitung rekursiv verschachtelter iterativer Pläne erörtert werden.

Eine einfache Abarbeitungsstrategie besteht darin, stets erst die innere Iteration bis zum Auffinden einer Nullstelle zu durchlaufen, bevor in der äußeren Schleife weiter konstruiert wird. Da dieses Vorgehen jeweils stets auf der Nullstellensuche für nur eine Variable t beruht, ist die Suche als *univariat* zu bezeichnen. Die in Abbildung 3.24 gezeigte Verschachtelung berührt die Suchstrategie nicht. Sie ist sowohl in der äußeren als auch in der inneren Schleife univariat. Die Schachtelung univariater Suchen führt allerdings schnell dazu, dass die Berechnungen nicht mehr in interaktiver Zeit vollzogen werden können, denn die Anzahl der nötigen Konstruktionsschritte wächst exponentiell mit der Schachtelungstiefe[71]. Das gilt insbesondere dann, wenn die innere Schleife relativ groß ist. In Abschnitt 3.4.2 sind einige Hinweise gegeben, wie die iterative Suche effizient gestaltet werden kann.

Eine weitere Abarbeitungsstrategie, die das bezüglich der Schachtelungstiefe exponentielle Anwachsen der Suchschritte vermeiden soll, unterscheidet nicht zwischen inneren und äußeren Schleifen. Statt den Fehler f_a in der äußeren Schleife erst zu bestimmen, wenn der Fehler f_i in der inneren Schleife als null angenommen werden kann, wird die Abarbeitung auch für Werte von f_i ungleich null bis zur Bestimmung von f_a fortgesetzt. Hier werden also zwei Variablen gleichzeitig iteriert (t_a und t_i) bzw. zwei Fehlerwerte gleichzeitig gemessen (f_i und f_a). Der allgemeine Fall der Iteration von n Variablen und dem Messen von n Fehlerwerten wird als *multivariate* Suche bezeichnet. Ansätze zur multivariaten Nullstellensuche sind im Anhang B beschrieben. Für die Zukunft ist die Integration des vielversprechenden multivariaten Suchverfahrens nach Färber ([FB09]) geplant.

3.6.3 Univariate Nullstellensuche

Eine zentrale Bedeutung für den Erfolg von iterativen Konstruktionen hat die Nullstellensuche. Entsprechend den in Abschnitt 3.1 diskutierten Anwendungsfällen lassen sich erforderliche Eigenschaften für den univariaten Nullstellensuchalgorithmus formulieren:

- **Abtastung der Funktion**

  Es gibt (im allgemeinen Fall) bei iterativen Konstruktionsplänen keine explizite Funktion, die die Abbildung des Eingangswertes t (der zu iterierende Parameter) auf den Fehler f (soll null werden) beschreibt. Demzufolge kann die implizit durch den Konstruktionsplan beschriebene Funktion $f(t)$ nur für (aus Sicht des Suchalgorithmus interessante) t Werte

[71] Wenn pro Iterationsstufe 20 Durchläufe angenommen werden, so ergeben sich bei einer Iterationstiefe von 2 (wie in Abbildung 3.24) $20 * 20$ Durchläufe für die innere Schleife, was bei kleineren inneren Schleifen durchaus noch zu einer brauchbaren Interaktivität führt. Bei einer angenommenen Iterationstiefe von t sind im allgemeinen aber 20^t Durchläufe zu erwarten.

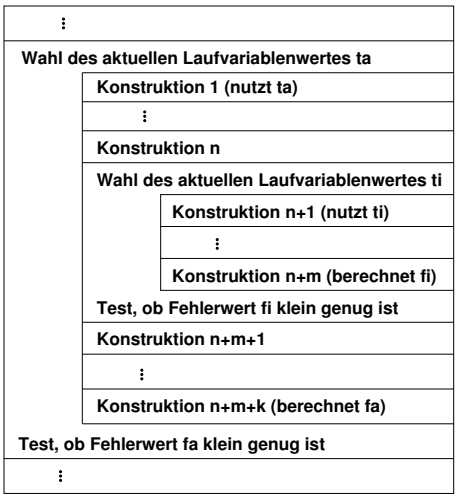


Abbildung 3.24: Schema eines zweistufigen hybriden Konstruktionsplanes

abgetastet werden. Im Laufe der Suche wird also eine Menge von Paaren (t, f), sogenannte *Samples*, bestimmt werden, die den aktuellen Wissensstand des Suchalgorithmus über die Funktion beschreibt.

- **Robustheit**

  Mit Robustheit ist insbesondere die Vermeidung von Stagnation im Suchverlauf gemeint. Das Hauptproblem besteht darin, dass die (implizit gegebene) Funktion im allgemeinen Fall als beliebig komplex angenommen werden muss. Das heißt, dass in ihr beliebige Muster (Kurvenverläufe oder sogar nicht definierte Bereiche) auftreten können. Eine „blinde Suche“ mit einem lokalen Verfahren, wie z. B. dem Newtonschen Näherungsverfahren könnte z. B. schnell dazu führen, dass viele Samples berechnet werden, ohne einen Suchfortschritt zu erreichen, siehe z. B. Abbildung 3.27. Ein anderes Problem besteht in der Unterscheidung von mehreren Nullstellen, um die im konkreten Fall richtige auswählen zu können. Ein unmotivierter Wechsel zwischen verschiedenen Nullstellen würde zu nicht nachvollziehbaren Veränderungen an den Modellen führen.

- **Schnelligkeit**

  Für die interaktive Anwendung ist ein schnelles Finden der Nullstelle wichtig. Aber auch die Lösung nicht interaktiver Aufgaben wie sie z. B. bei der Optimierung auftreten, kann durch schnelles Finden der Nullstellen an Qualität gewinnen.

- **Ausnutzung von Zusatzinformationen**, z. B.:

  - Insbesondere bei der Behandlung kontinuierlicher Änderungen, wie sie z. B. beim Verschieben eines Punktes oder beim Ziehen eines Parameterschiebereglers auftreten, kann der Hinweis: „Suche die Nullstelle nahe der alten Nullstelle“ sehr zur Effizienz und Robustheit beitragen.
  - Zur Sicherstellung der Schnelligkeit ist die Vorgabe von Abbruchbedingungen wichtig, diese können z. B. in folgender Form vorliegen:
    - eine maximale Anzahl von Samples zum Finden einer Nullstelle oder
    - Informationen um feststellen zu können, ob in einem bestimmten Intervall noch eine Nullstelle gesucht werden soll, z .B. Vorgabe einer Minimalbreite, die durchaus auch von den Funktionswerten an den Intervallrändern abhängig sein kann.

Ein schnelles Erkennen, dass die Nullstellensuche fehlschlägt, spart Rechenzeit, die für das Berechnen alternativer Varianten (Iteration über Mehrfachlösungen oder Freiheitsgradverteilungen) verwendet werden kann.

– Eine Einschränkung auf einen bestimmten Funktionstyp, der in speziellen Fällen der Applikation bekannt ist und der iterativen Suche als Zusatzinformation mitgeteilt werden kann, ermöglicht die Nutzung einer auf diesen Funktionstyp angepassten Suchstrategie, die ggf. schneller Nullstellen findet oder spezielle Abbruchbedingungen für die Suche berücksichtigen kann.

Zwei grundlegende Ansätze

In den verschiedenen Algorithmen zur Nullstellensuche fällt der Kompromiss zwischen Robustheit und Schnelligkeit jeweils anders aus. Schnelligkeit wird durch *„Tiefe zuerst“*-Strategien erreicht. Sie gehen davon aus, dass eine schrittweise Annäherung an eine Nullstelle möglich ist und speichern bzw. berücksichtigen für die Annäherung nur eine sehr begrenzte Anzahl von Informationen über den bisherigen Suchverlauf. Bei entsprechend gutartigem Funktionsverlauf zwischen dem Startpunkt der Suche und der Nullstelle wird diese auch (mehr oder weniger) schnell gefunden. Im Allgemeinen kann jedoch nicht von dieser Gutartigkeit ausgegangen werden, so dass eine reine „Tiefe zuerst“-Strategie nicht zufriedenstellend arbeitet.

Robustheit wird vor allem durch *„Breite zuerst“*-Strategien gewährleistet. Die Gleichmäßigkeit der Abtastung des Suchraumes steht hier im Vordergrund. Auf mehr oder weniger spekulative Annahmen über die Lage der Nullstelle, wie sie bei den „Tiefe zuerst“-Strategien getroffen werden, wird hier zugunsten eines gesicherten Fortschrittes bei der Erforschung des Suchraumes verzichtet. Damit ist jedoch die Schnelligkeit oft nicht zufriedenstellend.

Im nachfolgenden Abschnitt sollen einige Verfahren auch bezüglich dieser beiden Ansätze näher betrachtet werden.

Stärken und Schwächen einfacher Verfahren

Im Laufe der Zeit wurden eine ganze Reihe einfacher Verfahren zur Nullstellensuche entwickelt (siehe z. B. [TS88]). Diese haben jeweils ihre Vor- und Nachteile, auf die in diesem Abschnitt eingegangen werden soll. Komplexere Verfahren setzen meist spezielle Funktionstypen voraus (z. B. Polynome) und sollen zugunsten der Allgemeinheit des Verfahrens hier nicht betrachtet werden.

Die *Bisektion* eines bestimmten Intervalls ist eine sehr einfache und robuste Operation. Ohne Berücksichtigung von bekannten Funktionswerten wird in der Mitte des Intervalls ein neuer Funktionswert ermittelt, wodurch zwei Intervalle entstehen. Eine derartige Auswahl des nächsten zu untersuchenden Punktes unterstützt die Gleichmäßigkeit der Verteilung der Samples. Das geschieht insbesondere dann, wenn das jeweils im gesamten Suchraum größte Intervall geteilt wird („Breite zuerst“). Im sogenannten *Bisektionsverfahren* wird bei Feststellung eines Vorzeichenwechsels zwischen zwei Samples in dem Intervall, das zwischen den beiden Samples liegt, eine Nullstelle vermutet. In diesem Fall wird das entsprechende Intervall (und nicht das größte) zur Teilung ausgewählt („Tiefe zuerst“). Nach jeder Unterteilung wird die Bisektion stets in dem Teilintervall fortgesetzt, in dem der Vorzeichenwechsel vorliegt bis die Nullstelle mit hinreichender Genauigkeit ermittelt wurde. Durch die einfache Halbierung ist die Bisektion robuster als spekulative[72] (und damit möglicherweise schnellere) Verfahren, auf die noch eingegangen wird. Wie Abbildung 3.25 zeigt, muss die Verfolgung eines Vorzeichenwechsels jedoch nicht zwangsläufig zu einer Nullstelle führen.

[72] Man „spekuliert“ bei solchen Verfahren darauf, dass die Auswertung von Informationen (z. B. bekannten Funktionswerten) zu einem schnelleren Finden der Lösung führt und berechnet deshalb (verfahrensabhängig) eine Stelle, an der der nächste Funktionswert zu berechnen ist.

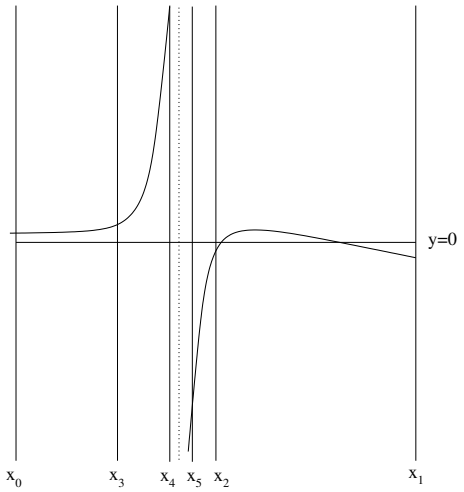


Abbildung 3.25: Beispiel, bei dem die von einem Vorzeichenwechsel gesteuerte Bisektion fehlschlägt. Die Indizes der x-Werte geben die Reihenfolge der Samples an.

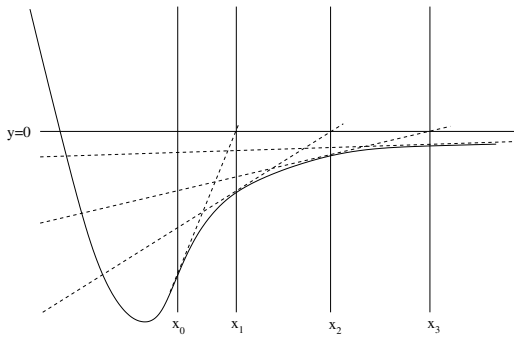


Abbildung 3.26: Beispiel, bei dem die Suche nach dem Newtonschen Verfahren divergiert.

Es gibt eine Reihe von Verfahren, die im Gegensatz zum Bisektionsverfahren keinerlei Breitensuche unterstützen. Derartige Verfahren arbeiten mit dem Anstieg der Funktion in einem bestimmten Punkt oder einer Approximation dieses Anstiegs[73]. Ein oft verwendeter Vertreter ist das *Newtonsche Verfahren*. In ihm wird an einer Stelle x_k eine Tangente an die zu untersuchende Funktion $f(x)$ angelegt. Der Schnittpunkt der Tangente mit der x-Achse liefert die nächste zu untersuchende Stelle x_{k+1}. Das Verfahren wird so lange fortgesetzt, bis eine Nullstelle mit hinreichender Genauigkeit gefunden wurde, oder die Suche als erfolglos gilt (z. B. wegen Überschreitung einer festgesetzten Anzahl von Suchschritten). Die Berücksichtigung eines aktuell untersuchten Intervalls oder gar aller bereits untersuchten Stellen ist im Verfahren nicht vorgesehen. Dementsprechend ist es möglich, dass das Verfahren divergiert (Abbildung 3.26) oder sich nur in einer Quasi-Endlosschleife bewegt (Abbildung 3.27 sowie 3.28).

Wenn die Funktion wie in den iterativen Plänen nur abgetastet wird, dann kann die Tangente nicht genau bestimmt werden, denn hierfür wäre die Ableitung $f'(x)$ zu bilden. Einen Ausweg bietet die Verwendung einer Sekante, die die Tangente beliebig genau annähert. Zur Bestimmung der Sekante ist eine zweite Abtastung nötig, die in einer Entfernung Δx von der Stelle x_k vorgenommen wird,

[73] multivariate Varianten solcher Verfahren sind im Anhang B beschrieben

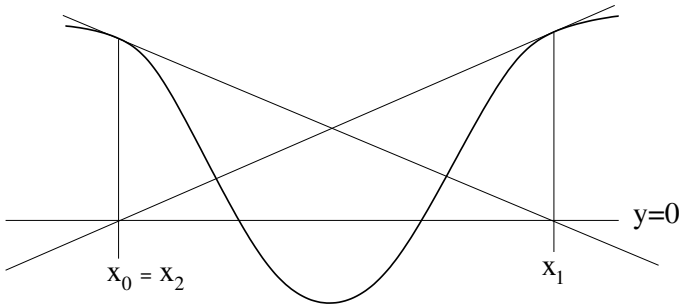


Abbildung 3.27: Beispiel, bei dem das Newtonsche Verfahren in eine (Quasi-) Endlosschleife laufen kann - symmetrische Anordnung

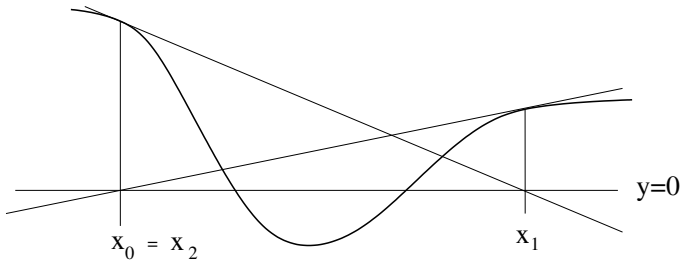


Abbildung 3.28: Weiteres Beispiel, bei dem das Newtonsche Näherungsverfahren in eine (Quasi-) Endlosschleife laufen kann - nicht symmetrische Anordnung

wobei Δx hinreichend klein zu wählen ist. Der Suchverlauf[74] dieses angenäherten Newtonschen Verfahrens weicht dann beliebig wenig von dem des originalen Newtonschen Verfahrens ab.

Das *Sekantenverfahren* verwendet die letzten zwei abgetasteten Stellen x_k und x_{k-1}, um die Sekante zu berechnen. Der Schnittpunkt x_{k+1} der Sekante mit der x-Achse wird als nächstes abgetastet. Der Unterschied zum oben beschriebenen angenäherten Newtonsche Verfahren besteht darin, dass keine Stelle quasi zweimal abgetastet wird (das zweite Mal jeweils um ein vorgegebenes Δx versetzt), sondern dass jede Abtastung zu einem Fortschreiten der Suche beiträgt. Wie Abbildung 3.29 zeigt, kann das Sekantenverfahren wie das Newtonsche Verfahren divergieren.

Die Berechnung von Sekanten ist auch Grundlage des als *regula falsi* bezeichneten Verfahrens, das auf ein bestimmtes, schrittweise verkleinertes Intervall angewendet wird. Es ist diesbezüglich mit dem Bisektionsverfahren vergleichbar. Der Unterschied zum Bisektionsverfahren besteht darin, dass die Teilung des Intervalls nicht einfach in seiner Mitte erfolgt, sondern an der Stelle, an der die Sekante die x-Achse schneidet. Die Sekante ergibt sich als Gerade durch die an den Intervallgrenzen ermittelten Funktionswerte. Um das Verfahren anwenden zu können, müssen sich wie auch beim Bisektionsverfahren die an den Intervallgrenzen ermittelten Funktionswerte im Vorzeichen unterscheiden, denn sonst würde der Schnittpunkt außerhalb des Intervalls liegen. Typischerweise konvergiert regula falsi schneller als das Bisektionsverfahren, in Abbildung 3.30 ist jedoch eine Funktion dargestellt, in der der Suchfortschritt bei jedem Schritt nur sehr klein ist. Bisektion hätte sich der Nullstelle schneller angenähert.

Bei iterativen Konstruktionen kann es immer wieder dazu kommen, dass sich die Konstruktion für gegebene Eingangswerte[75] nicht ausführen lässt. Hierdurch ergeben sich Bereiche, für die kein

[74] die Folge der Abtaststellen $x_0, x_1, x_2 ... x_n$

[75] Eingangswerte sind außer dem iterierten x-Wert auch sämtliche bei den einzelnen Konstruktionen verwendete Punktpositionen, Abstands- und Winkelparameter usw.

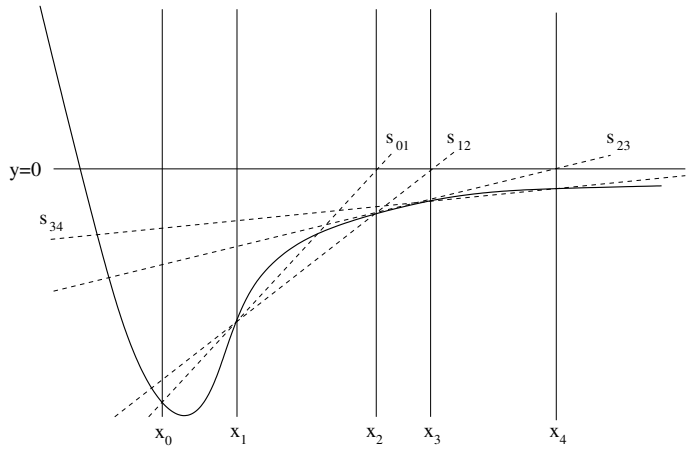


Abbildung 3.29: Divergierende Suche nach dem Sekantenverfahren.

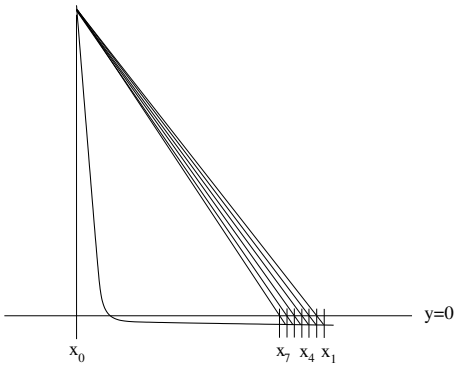


Abbildung 3.30: Beispiel, bei dem regula falsi sich nur langsam der Nullstelle nähert. Läge x_0 (die linke Begrenzung des initialen Intervalls) noch weiter links, so wären die Sehnen noch viel steiler und der Fortschritt pro Abtastung wäre noch wesentlich schlechter.

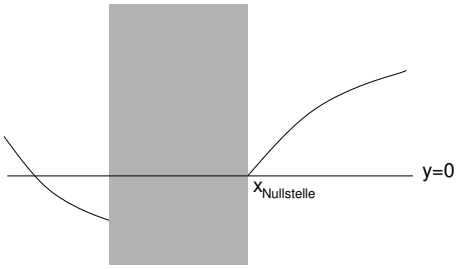


Abbildung 3.31: Beispiel, in dem sich der Nullstelle nur von einer Seite genähert werden kann

Fehlerwert bestimmt werden kann (siehe Abbildung 3.31). Die oben erörterten einfachen Suchverfahren müssen an einer solchen Stelle abbrechen. Der Algorithmus zur Nullstellensuche in iterativen Konstruktionen sollte also die Möglichkeit des Auftretens solcher Bereiche berücksichtigen können bzw. in der Lage sein, spezielle „nicht definiert"-Funktionswerte zu verarbeiten. [Hsu96] schlägt an dieser Stelle eine zusätzliche Fehlerfunktion $\delta(x)$ vor, die das Fehlschlagen der entsprechenden Konstruktionen beschreibt[76]. Die Funktion könnte genutzt werden, um den Bereich, in dem die Konstruktion nicht ausgeführt werden kann, zielgerichtet zu verlassen. Eine allgemeine Umsetzung dieser Strategie ist jedoch nicht trivial[77].

Ein hybrider Ansatz

Als Schlussfolgerung aus den oben beschriebenen Problemen der einfachen Verfahren wurde ein hybrider Ansatz entwickelt, der robust gegen Stagnation und trotzdem schnell ist. Hierfür vereint er Folgendes in sich:

1. Strategie: „Breite zuerst"

   Wegen der großen Bedeutung der Robustheit hat die Erforschung des Suchraumes im allgemeinen Fall eine höhere Priorität, als die lokale Nullstellensuche. Die Extremform dieses Ansatzes wäre das Bisektionsverfahren, das zu Ungunsten der Schnelligkeit den Suchraum sehr gleichmäßig erforscht.

2. Effiziente Speicherung der gesammelten Funktionswerte

   Hier wird auf spezielle Datenstrukturen zurückgegriffen werden. Sie dienen zur Speicherung von Funktionswerten und auch von interessanten Bereichen bzw. Intervallen[78]. Wichtige Operationen sind das Einfügen und Löschen aus diesen Datenstrukturen sowie das Auffinden von Vorgänger und Nachfolger.

3. Lokale „Tiefe zuerst"-Suche

   Um die Geschwindigkeit der Nullstellensuche gegenüber einer reinen (sehr robusten) Bisektion wesentlich zu verbessern, kann bei hinreichendem Fortschritt in der Annäherung an die Nullstelle auch nach der Strategie „Tiefe zuerst" vorgegangen werden. Wichtig ist hierbei jedoch stets die Erfolgsbewertung, um die Robustheit nicht zu gefährden.

---

[76] z. B. der vorzeichenbehaftete Abstand zweier Kreise, die sich nicht schneiden.

[77] Zum Beispiel könnte $\delta(x)$ beschreiben wie weit zwei Kreise, die sich nicht schneiden, voneinander entfernt sind. Es ist zu beachten, dass unterschiedliche Konstruktoren die Quelle des Fehlschlagens sein können. Dementsprechend ist ggf. eine Menge von derartigen Funktionen ($\delta_1(x)$,$\delta_2(x)$,...$\delta_k(x)$) separat zu betrachten. Zudem ist bei verschachtelten Iterationen das Bestimmen eines entsprechenden Fehlermaßes für innere iterative Konstruktionen, in denen keine Nullstelle gefunden werden kann, nicht trivial. Die Frage ist z. B., wie genau im Rahmen einer Extremwertermittlung der entsprechende kleinste gefundene Fehler bestimmt werden muss, um die Suche auf der entsprechenden Fehlerfunktion robust ablaufen lassen zu können.

[78] Zur Verwendung der Begriffe: Ein *Intervall* liegt genau zwischen zwei abgetasteten Punkten (und keinem weiteren Punkt dazwischen). Der Begriff *Bereich* wird im allgemeineren Sinn verwendet, so dass ein Bereich ein oder mehrere benachbarte Intervalle umfassen kann oder gar nicht an Intervallgrenzen (die Samples) gebunden ist.

4. Berücksichtigung von nicht definierten Bereichen

   Wegen der oben beschriebenen Schwierigkeiten wurde auf die Implementierung von $\delta(x)$ Funktionen verzichtet. Die nicht definierten Bereiche werden während der „Breite zuerst"-Suche erschlossen. Spezielle Berücksichtigung finden die Grenzen zu nicht definierten Bereichen, d. h. Bereiche zwischen einem bekannten Wert und einem nicht definierten Wert.

5. Möglichkeit zur Wichtung des jeweiligen Suchgebietes

   Um sich gezielter einer Nullstelle nähern zu können oder einfach den Suchraum zu erforschen, ist eine Klassifizierung des jeweils (lokal) zu untersuchenden Gebietes sinnvoll:

   - Je größer ein Bereich ist, umso wichtiger ist er für die „Breite zuerst"-Suche.
   - Je größer die Wahrscheinlichkeit ist, dass in ihm eine Nullstelle gefunden wird (z. B. durch das Vorhandensein eines Vorzeichenwechsels oder eines guten Suchfortschritts), desto wichtiger ist ein Bereich für die „Tiefe zuerst"-Suche. Dies gilt auch für die speziell behandelten Grenzen der nicht definierten Bereiche.

6. Begrenzung des Suchraumes

   Oft lassen sich sinnvolle Wertebereiche für die Wahl der Iterationsparameter bestimmen, z. B. $-\Pi \ldots + \Pi$ für die Richtung von Geraden oder Bauraumbegrenzungen für Punktkoordinaten[79]. Offene Suchräume werden in der aktuellen Implementierung durch große Grenzen emuliert, die von der Anwendungssoftware festzulegen sind. Günstiger wäre in solchen Fällen ein spezieller Ansatz, der Informationen über die Wahrscheinlichkeit der Lösungsfindung berücksichtigen kann. Dies könnte z. B. über eine Funktion geschehen, die nahe der alten Lösung ein Maximum besitzt und mit zunehmender Entfernung von der alten Lösung abnimmt. Durch die Art und Weise dieses Abnehmens ließe sich der vermutete Suchraum modellieren. Wenn die Suche auch außerhalb des vermuteten Suchraumes gewünscht würde, könnte auch dort die Funktion von null verschiedene Werte liefern (im Unendlichen gegen null gehend).

7. Optionale Berücksichtigung eines Startwertes

   Bei interaktiven Änderungen liegt die neue Nullstelle oft in der Nähe der alten Nullstelle. Wenn die Anwendungssoftware (z. B. solche, die das interaktive Bewegen von Mechanismen unterstützt) die Berücksichtigung fordert, dann wird zunächst an der Stelle der letzten Lösung abgetastet[80]. Von hier aus wird der Suchraum zunächst in kleinen und dann zunehmend größer werdenden Schritten abgesucht bis die Grenzen des Suchraumes erreicht sind.

   Auch bei der Konstruktion von starren Körpern kann sich eine gezielte Wahl des Startwertes wesentlich auf die Suche auswirken. Im besten Fall führt bereits der Startwert stets zu einer Nullstelle, so z. B. bei Abständen innerhalb eines starren Körpers (vgl. Abschnitt 3.6.6).

Implementierungsdetails

Zur Speicherung von Funktionswerten und interessanten Bereichen bzw. Intervallen werden bei der Suche folgende Listen aufgebaut:

- die Liste aller abgetasteten Samples, sortiert nach der x-Koordinate,
- die Liste aller existierenden Bereiche, sortiert nach Größe (optional gewichtet mit einer apriori Lösungswahrscheinlichkeit) für „Breite zuerst"-Suche,

[79] Implementierungsdetail: Durch eine entsprechende Normierung kann die Suche im Bereich von $+1 \ldots -1$ ablaufen.

[80] Ein spezieller Ansatz kann zudem die n letzten Nullstellen berücksichtigen und daraus die neue extrapolieren. Wichtig ist hierfür, dass die Änderungen kontinuierlich erfolgen und es sich stets um die gleiche Nullstelle handelt (eindeutige Wahl aus Mehrfachlösungen!).

- die Liste der Bereiche mit Vorzeichenwechsel, sortiert nach ihrer Interessantheit[81],
- die Liste der Bereiche mit einem nicht definierten Wert, sortiert nach ihrer Interessantheit[81] und
- die Liste der Extrema (Paare von benachbarten Bereichen, d. h. drei Samples, von denen der mittlere null am nächsten ist). Auch die Extrema sind nach ihrer Interessantheit[81] sortiert.

Es wird hier von „Listen“ gesprochen, um die Sortiertheit der Daten hervorzuheben. Implementiert ist jedoch eine komplexere Datenstruktur, die das Einfügen und Löschen in $O(\log n)$ garantiert[82]. Diese Datenstruktur basiert auf Patricia-Bäumen[83] (vgl. [Sed98]). Der Vorteil der Patricia-Bäume liegt darin, dass sie aufgrund der bitweisen Betrachtung der Suchschlüssel (hier Gleitkommawerte in Form des Datentyps *double*) keinen speziellen Aufwand zum Ausbalancieren benötigen[84]. In der Implementierung wurden die einzelnen Blattknoten zur effizienten Bestimmung von Vorgänger und Nachfolger in einer doppelt verketteten Liste eingebunden, um u. a. in konstanter Zeit Vorgänger und Nachfolger zu finden oder auf das erste Element zugreifen zu können. Zudem können Dubletten verwaltet werden, was für die Speicherung der Bereiche wichtig ist, denn es kann vorkommen, dass Bereiche mit gleicher Interessantheit einzufügen sind.

Der Algorithmus besteht im Wesentlichen aus folgenden zwei Teilen:

- Suche nach der ersten Nullstelle (Algorithmus *SEN*) und
- Suche nach einer weiteren Nullstelle (Algorithmus *SWN*).

Beide Algorithmen greifen bei ihrer Suche auf Algorithmen zur Bestimmung einer Nullstelle in einem bestimmten Intervall nach einer jeweils erfolgversprechenden Strategie zurück. Sie können wählen zwischen[85]:

- SNI_{vzw}
  Hierbei handelt es sich um die Suche nach einem Vorzeichenwechsel. Um Probleme wie in Abbildung 3.30 zu vermeiden, sollte die zu untersuchende Stelle x_{Test} als ein Mittelwert aus:
  - $x_{TestSek}$, dem Schnittpunkt der Sekante mit der x-Achse, und
  - $x_{TestBis}$, der Mitte des Intervalls,

  gebildet werden. Bei Verwendung eines gewichteten Mittels (z. B. des gewichteten arithmetischen Mittels) kann durch das Setzen der Gewichte die Robustheit (Bevorzugung von $x_{TestBis}$) oder der erhoffte schnelle Fortschritt (Bevorzugung von $x_{TestSek}$) begünstigt werden[86].
- SNI_{ext}
  Es wird nach einem Extremum in einem Bereich gesucht, der aus zwei benachbarten Intervallen besteht. Durch die drei Wertepaare kann z. B. eine Parabel gelegt werden, um die Position des Extremums abzuschätzen. Gegebenenfalls können auch weitere benachbarte Funktionswerte für die Abschätzung der zu untersuchenden Stelle herangezogen werden.

---

[81] siehe Seite 86: bereichsspezifische Kriterien für Interessantheit

[82] Bei einfachen Listen kann die Suche nach einem bestimmten Element dazu führen, dass die gesamte Liste durchlaufen werden muss. Man spricht deshalb von linearer Zeitkomplexität bezüglich n, der Anzahl der in der Liste der Elemente enthaltenen Elemente. Die entsprechende Notation ist $O(n)$. $O(\log n)$ weist dementsprechend auf eine wesentlich günstigere Zeitkomplexität hin.

[83] Patricia (Practical algorithm to retrieve information coded in alphanumeric), ein erstmals 1968 von D. R. Morrison beschriebener digitaler Baum.

[84] Im aktuellen C++-Standard werden für die Speicherung eines *double*-Wertes 64 Bit benötigt. Ein Patricia-Baum, der *double*-Werte als Suchschlüssel verwendet, kann deshalb im ungünstigsten Fall nur eine Höhe von 64 aufweisen. Das entspricht (selbst wenn theoretisch alle Werte der Domäne gleichzeitig in ihm verwendet worden sind) einer maximalen Anzahl von 64 Bittests, bis ein Schlüssel gefunden wird.

[85] Auf eine detailliertere Darstellung der vier *SNI*-Algorithmen wird verzichtet.

[86] Hierbei ist eine Adaption der Gewichte entsprechend dem Fortschritt sinnvoll.

- SNI_{bri}
  Durch Unterteilung eines Intervalls (typischerweise ist hier das breiteste am interessantesten) wird die gleichmäßige Abtastung des Suchraums umgesetzt. Am robustesten ist die Unterteilung in der Mitte.

- SNI_{gnd}
  Auch die Suche an der Grenze zwischen einem definierten und einem nicht definierten Bereich läuft bei Unterteilung in der Mitte des Intervalls am robustesten. Die Beachtung von bekannten Nachbarwerten, die die Annäherung der Fehlerfunktion an die Grenze beschreiben, ist aber sinnvoll (vgl. die Abbildung zu Punkt 4 auf Seite 87).

Algorithmus SEN:

1. Lösche die alten Suchergebnisse.
2. $n_{ges} := Anzahl\,der\,erlaubten\,Suchschritte$
3. $n_{vzw} := 12\,(erlaubte\,Anzahl\,von\,Suchschritten\,bei\,Vorzeichenwechseln)$
4. $n_{ext} := 12\,(erlaubte\,Anzahl\,von\,Suchschritten\,bei\,Extremen)$
5. $x_{Start} := vermuteter\,Lösungswert$
6. $x_{Test} := x_{Start}$
7. solange $n_{ges} > 0$ ist und die Bereichsgrenzen noch nicht erreicht worden sind:
   (a) Ermittle den Funktionswert y_{Test} an der Stelle x_{Test}. Auf Basis des gefundenen Wertepaars $(x_{Test,}\, y_{Test})$ werden die Listen aktualisiert. Gebe y_{Test} zurück, wenn x_{Test} eine Nullstelle ist.
   (b) Wenn ein Vorzeichenwechsel aufgetreten ist, dann suche im entsprechenden Intervall mit höchstens $\min(n_{vzw},\, n_{ges})$ Schritten nach einer Nullstelle mit dem Algorithmus SNI_{vzw}. Vermindere n_{ges} um die bei der Suche gemachten Suchschritte und falls eine Nullstelle gefunden wurde, liefere sie zurück.
   (c) Wenn ein Extremum (in Richtung Nullstelle) aufgetreten ist, dann suche im entsprechenden Intervall mit höchstens $\min(n_{ext},\, n_{ges})$ Schritten nach einer Nullstelle mit dem Algorithmus SNI_{ext}. Vermindere n_{ges} um die bei der Suche gemachten Suchschritte und falls eine Nullstelle gefunden wurde, liefere sie zurück.
   (d) Ermittle den nächsten Wert von x_{Test}. Hierbei wird in sich stetig vergrößernden Schritten abwechselnd links und rechts von x_{Start} gesucht.
8. Setze die Suche mit der Abarbeitung von SWN fort.

Auch die in den Algorithmen SNI_{vzw} und SNI_{ext} gefundenen Wertepaare werden in den Datenstrukturen gespeichert. Sollten die für den Algorithmus SEN festgelegten Werte n_{vzw} oder n_{ext} zu klein gewesen sein (d. h. eine Suche zu schnell abgebrochen worden sein), dann kann die entsprechende Nullstelle in SWN gefunden werden.

Algorithmus SWN:

1. Untersuche die Grenzen des Suchraumes, falls sie noch nicht untersucht wurden und falls $n_{ges} > 0$.
2. $n_{vzw} := 10\,(erlaubte\,Anzahl\,von\,Suchschritten\,bei\,Vorzeichenwechseln)$
3. $n_{ext} := 5\,(erlaubte\,Anzahl\,von\,Suchschritten\,bei\,Extremen)$

4. $n_{gnd} := 2$ (*erlaubte Anzahl von Suchschritten an Grenzen zu nicht definierten Bereichen*)
5. $n_{bri} := 1$ (*erlaubte Anzahl der zu teilenden breitesten Intervalle*)
6. solange $n_{ges} > 0$ ist und die Suche nicht ins Stocken gekommen ist :
   (a) Wenn es Intervalle mit einem Vorzeichenwechsel gibt, dann mache im Algorithmus SNI_{vzw} höchstens $\min(n_{vzw}, n_{ges})$ Suchschritte zur Unterteilung des jeweils interessantesten Intervalls. Vermindere n_{ges} um die bei der Suche gemachten Suchschritte und falls eine Nullstelle gefunden wurde, liefere sie zurück.
   (b) Wenn es Intervalle mit einem Extremum (in Richtung Nullstelle) gibt, dann mache im Algorithmus SNI_{ext} höchstens $\min(n_{ext}, n_{ges})$ Suchschritte zur Unterteilung des jeweils interessantesten Intervalls. Vermindere n_{ges} um die bei der Suche gemachten Suchschritte und falls eine Nullstelle gefunden wurde, liefere sie zurück.
   (c) Wenn es Intervalle mit *einem*[87] nicht definierten Grenzwert gibt, dann mache im Algorithmus SNI_{gnd} höchstens $\min(n_{vzw}, n_{gnd})$ Suchschritte zur Unterteilung des jeweils interessantesten Intervalls. Vermindere n_{ges} um die bei der Suche gemachten Suchschritte und falls eine Nullstelle gefunden wurde, liefere sie zurück.
   (d) Wenn es noch Intervalle mit einer Mindestbreite gibt, dann mache im Algorithmus SNI_{bri} höchstens $\min(n_{bri}, n_{ges})$ Suchschritte zur Unterteilung des jeweils interessantesten Intervalls. Vermindere n_{ges} um die bei der Suche gemachten Suchschritte und falls eine Nullstelle gefunden wurde, liefere sie zurück.
7. Teile mit, dass keine (weitere) Nullstelle gefunden werden konnte.

In Schritt 6 des Algorithmus *SWN* wurde die Reihenfolge der Teilschritte entsprechend der Wahrscheinlichkeit eines schnellen Erfolges gewählt. Da die „Tiefe zuerst"-Suche den schnellen Erfolg liefern kann, wird sie zuerst durchgeführt. Dann folgt die Untersuchung der Extrema. Sie ist insbesondere dann nötig, wenn die Suchschritte im Algorithmus *SEN* ungünstig zur Nullstelle lagen und kein Vorzeichenwechsel erkannt werden konnte. Die Reihenfolge der Anwendung der letzten beiden Suchstrategien ist unerheblich, zumindest, wenn n_{bri} und n_{gnd} klein sind.

Für die jeweils gespeicherten interessanten Intervalle sind Mindestbreiten definiert. Werden sie unterschritten, so wird das entsprechende Intervall nicht mehr gespeichert. Im Laufe der Abarbeitung des Algorithmus *SWN* kommt es dadurch typischerweise dazu, dass nur noch der Algorithmus SNI_{bri} aufgerufen wird (zumindest wenn alle Nullstellen im Suchraum gefunden worden sind).

Die Bestimmung der *Interessantheit* einer Unterteilung in einem Intervall (bzw. der Interessantheit des entsprechenden Bereiches) erfolgt für jede Suchstrategie nach eigenen Kriterien:

1. Für die Unterteilung der *größten Intervalle* ist die Interessantheit trivial. Es wird lediglich nach der Breite sortiert. Bei gleichen Breiten wäre allerdings auch eine Bevorzugung von Intervallen in der Nähe von x_{Start} denkbar[88]. Grundsätzlich könnte die Breite auch mit einem Maß für den Abstand zu x_{Start} gewichtet werden (je näher das Intervall an x_{Start} liegt, desto höher das Gewicht).
2. Die Interessantheit für die Untersuchung der *Intervalle mit Extrema* I_{ext} kann als $I_{ext} = \frac{y - y_{erwartet}}{y}$ berechnet werden, also als das Verhältnis von erwartetem Fortschritt $y - y_{erwartet}$ zum aktuellen Abstand y. Robuster ist jedoch eine Berücksichtigung des letzten relativen Fortschritts $\frac{y_{alt} - y}{y_{alt}}$, denn es ist denkbar, dass der erwartete Fortschritt stets sehr gut ist, jedoch die tatsächlich gefundenen Funktionswerte immer in der Nähe der alten Werte liegen oder sogar schlechter sind.

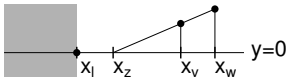
[87] Sind beide Werte nicht definiert, kann eine weitere Untersuchung nur in Schritt 6d erfolgen.

[88] Man beachte, dass die Gleichheit von Gleitkommazahlen numerisch problematisch ist. Hier wären Toleranzen zu berücksichtigen, evtl. in Form einer Quantisierung, die von der Genauigkeit der zu liefernden Lösung abhängen könnte.

3. Kriterien für die Interessantheit I_{vzw} der *Intervalle mit Vorzeichenwechseln* sind z. B.:
   (a) die Breite b,
   (b) die gefundene (vorzeichenfreie) Nähe h und
   (c) der Suchverlauf, wobei die vorher bestimmte Nähe h_{alt} bekannt sein muss.

   Breite und gefundene Nähe sollten immer relativ zueinander betrachtet werden, z. B. wie in $I_{vzw} = \frac{b}{h}$. Eine Berechnung, die den Suchverlauf berücksichtigt, ist die folgende $I_{vzw} = \frac{h_{alt}-h}{h_{alt}}$.
   Zur Unterstützung der „Tiefe zuerst"-Strategie, die insbesondere dann sinnvoll ist, wenn möglichst schnell (irgend-)eine Lösung gefunden werden soll, ist aber auch die Nutzung von Zeitstempeln denkbar. Das zuletzt gefundene Intervall hat den höchsten Zeitstempel (z. B. ein hochzuzählender Zähler) und wird jeweils als interessantestes Intervall ausgewählt. Für eine „Breite zuerst"-Strategie könnte analog das jeweils älteste Intervall ausgewählt werden.

4. Ein typisches Beispiel für die Untersuchung der *Intervalle mit einer Grenze zu nicht definierten Bereichen* könnte wie folgt aussehen:



   Hierbei liegt das zu untersuchende Intervall zwischen x_l (wo kein Funktionswert definiert ist) und x_v (hier ist der Funktionswert $y_v$definiert). Wenn ein Nachbarwertepaar (x_w, y_w) bekannt ist, kann ein Wert x_z linear interpoliert werden. Liegt x_z zwischen x_l und x_v (d. h. im zu untersuchenden Intervall), so ist das Intervall grundsätzlich interessant. Es scheint sinnvoll, die Interessantheit I_{gnd} von der Robustheit abhängig zu machen. Dies ist z. B. möglich mit $I_{gnd} = 2 - abs(\frac{x_v-x_z}{x_v-x_l})$. Bei einer derartigen Bestimmung sind Interessantheitsgrade von 1 (falls $x_z = x_l$) und 2 (falls $x_z = x_v$) erreichbar[89]. Liegt x_z außerhalb des Intervalls, so kann I_{gnd} ein beliebiger Wert kleiner 1 zugeordnet werden. Falls y_w auch nicht definiert ist und somit x_z nicht interpoliert werden kann, sollte $I_{gnd} = 1$ angenommen werden.
   Zusätzlich zur Robustheit könnte auch:
   (a) die Breite b des Intervalls (je breiter desto interessanter),
   (b) die gefundene (vorzeichenfreie) Nähe $h = abs(y_v)$ und,
   (c) die Nähe zu x_{Start}(je näher desto interessanter) sowie
   (d) der Suchverlauf

   mit in die Interessantheit einfließen.

Die Bestimmung der zu untersuchenden Werte x_{Test} ist sinnvollerweise ein Kompromiss zwischen Schnelligkeit und Robustheit, der bei „Tiefe zuerst"-Suchen mehr zugunsten der Schnelligkeit ausfällt. Um Probleme wie in Abbildung 3.30 zu vermeiden lässt sich z. B. im Algorithmus SNI_{vzw} der Startwert als $x_{Start} = m_{vertrauen} * x_0 + (1 - m_{vertrauen}) \frac{x_1+x_2}{2}$ bestimmen, wobei x_0 die durch lineare Interpolation vermutete Nullstelle und $\frac{x_1+x_2}{2}$ die Intervallmitte ist. Das Vertrauensmaß in die lineare Interpolation (bzw. die Sekante) könnte stets als $m_{vertrauen} = 0.9$ gewählt worden sein. Sinnvoll ist jedoch eine Abhängigkeit des Vertrauensmaßes von den Anstiegen in den benachbarten Intervallen. Je stärker die Anstiege von dem Anstieg im aktuellen Intervall abweichen, desto geringer sollte das Vertrauen in die lineare Interpolation sein.

[89] Das Finden von Funktionswerten ist robuster, da nur mit ihnen Nullstellen festgestellt werden können (die Erkenntnis, dass an einer bestimmten Stelle kein Wert definiert ist, kann nur indirekt zur Nullstellenfindung beitragen). Da die Wahrscheinlichkeit in der Nähe von x_v einen Funktionswert zu finden größer ist, wird die Suche dort bevorzugt.

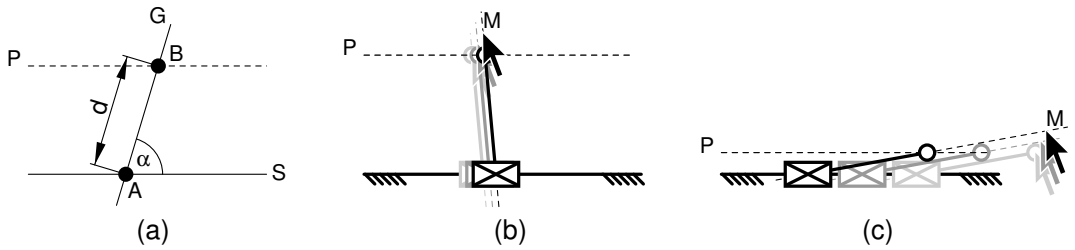


Abbildung 3.32: Bewegung eines Punktes auf einer implizit gegebenen Geraden. (a) Geometrische Beschreibung des Modells. Bei gegebenen S, α und d muss sich B auf einer (implizit gegebenen) Geraden P bewegen, die parallel zur Geraden S liegt. Für die beiden folgenden Teilabbildungen wurde angenommen, dass B mit dem Mauszeiger M bewegt werden soll und hierfür eine iterative Konstruktion erfolgt (siehe Text). (b) G steht nahezu rechtwinklig auf S. B kann gut mit M bewegt werden, denn vertikale Abweichungen von M bezüglich P haben kaum Einfluss auf die Position von B. (c) Ist α fast 0° oder 180°, so lässt sich B nur schlecht positionieren. Schon kleinste Abweichungen des Mauszeigers von P führen zu großen Bewegungen des Punktes B, was für den Nutzer nicht nachvollziehbar ist.

3.6.4 Suche nach Extremwerten

Die in dieser Arbeit beschriebenen Konstruktionen beruhen auf Constraints, die als *scharf* oder auch *hart* bezeichnet werden können. Insbesondere für die Lösung von Interaktions- und Optimierungsaufgaben ist die Verwendung *unscharfer* Constraints[90] sinnvoll. Denkbar ist z. B. ein „Nahe"-Constraint, um einen mit dem Mauszeiger zu bewegenden Punkt auch dann in der Nähe des Mauszeigers zu halten, wenn der Punkt durch Constraints an eine bestimmte Bewegungsbahn gebunden ist. Einige Konstruktionen setzen eine solche Suche nach der naheliegenden Lösung lokal um (u. a. Konstruktion eines Punktes auf einem Kreis und nahe dem Mauszeiger, wenn der Punkt durch den Mauszeiger bewegt werden soll). Wenn die Bewegungsbahn des Punktes jedoch nur implizit gegeben ist, dann ist die lokale Lösung u. U. nicht oder nur schlecht nachvollziehbar.

Beispiel 3.21: Bewegung eines Punktes auf einer implizit gegebenen Gerade
Abbildung 3.32 zeigt ein Bespiel aus MASP. Auf einem gestellfesten Getriebeglied (Hauptachse ist S) wird ein Schubgelenk (charakteristischer Punkt ist A) bewegt. An dem Schubgelenk ist ein weiteres Getriebeglied angebracht (Hauptachse ist G), wobei zwischen beiden ein fester Winkel α besteht. Der Endpunkt B des zweiten Getriebegliedes muss sich wegen der definierten Constraints auf einer (implizit gegebenen) Geraden P bewegen, die parallel zu S ist. P stellt somit die mögliche Bewegungsbahn von B dar. Soll B nun interaktiv mit dem Mauszeiger (M) bewegt werden, so wäre die Projektion von M auf die Bewegungsbahn (also P) die für den Nutzer plausible Lösung. Hierfür müsste MASP jedoch die Bewegungsbahn kennen, was durch eine spezielle Transformation des Constraint-Netzes möglich wäre (ähnlich den in Abschnitt 3.2 gezeigten Transformationen). In der aktuellen Implementierung von Ficucs wird jedoch iterativ konstruiert, was zu dem in Abbildung 3.32(c) gezeigten Problem führen kann[91]. Grund hierfür ist, dass bei der iterativen Konstruktion der Punkt A so lange auf S verschoben wird, bis G durch M geht (siehe Plan im Anhang A.3), was für spitze und stumpfe Winkel α zu dem sehr ungünstigen Verhalten führt. □

Zur Lösung des Problems ist es denkbar, die beschriebenen Heuristiken für eine Nullstellensuche auch in einer entsprechenden Suche nach Extremwerten zu verwenden. Die in Abbildung 4.4 gezeigte Bewegung des Punktes P könnte so als Suche des minimalen Abstandes des Punktes zur Bewegungsbahn umgesetzt werden. Andererseits zeigt dieses Beispiel auch, dass eine allgemein gültige Implementierung nicht trivial ist, denn in der Bewegungsbahn werden unterschiedliche Konfigurationen für die Wahl von Mehrfachlösungen benutzt. Das würde zu mehreren Fehlerfunktionen (bzw. hier Gütefunktionen) führen.

[90] auch als *weiche* Constraints oder *Soft*-Constraints bezeichnet
[91] In MASP wurde deshalb die in Abschnitt 5.2.3 beschriebene Vorgehensweise implementiert.

Eine mögliche Lösung des Problems auf der Ebene der Applikationssoftware ist in Abschnitt 5.2.3 gezeigt. Der dort beschriebene Ansatz basiert auf einer Analyse der Bewegungsbahnen mit Hilfe des Constraint-Solvers.

Auch die Optimierung bestimmter Modelleigenschaften wie ein Volumen oder der potentiellen Energie könnte durch eine Suche nach Extremwerten vorgenommen werden. Hierzu müssten jedoch alle freien (und den zu optimierenden Wert beieinflussenden) Variablen bei Bedarf variiert werden können. Dies ist in der aktuellen Umsetzung des Constraint-Solvers jedoch noch nicht unterstützt.

3.6.5 Wahl von Mehrfachlösungen

Mehrfachlösungen treten nicht nur im Zusammenhang mit direkten Konstruktionen auf (vgl. Abschnitt 3.5). Auch in iterativen Konstruktionen sind Mehrfachlösungen möglich. Sie äußern sich im Vorhandensein mehrerer Nullstellen der Fehlerfunktion. Die Problematik der Wahl geeigneter Kriterien für die Unterscheidung von Mehrfachlösungen wird im folgenden Text an einem Beispiel verdeutlicht. Eine ausführlichere Diskussion zum Problem der korrekten Wahl der Lösungen sowie vergleichende Betrachtungen zu anderen Ansätzen erfolgen in Kapitel 4.

Das im einführenden Kapitel in Abbildung 2.9 gezeigte Sechseck ist nicht eindeutig bestimmt. Die iterative Konstruktion kann zwei unterschiedliche Lösungen liefern, wobei angenommen wird, dass bei den direkten Konstruktionsschritten (Schneiden von zwei Kreisen) die jeweilige Wahl der Lösung beibehalten wird.

Für das in Abbildung 3.33 gezeigte Beispiel wurde angenommen, dass das Sechseck durch Bewegung des Mauszeigers, der in Punkt 1 angreift, verschoben wird. Es ergab sich folgender Plan:

1. Konstruiere p_2 im Abstand zu $p_1$und nahe der alten Lösung, also hier auf gleicher Höhe wie p_1.
2. Konstruiere p_3 testweise auf einem Kreis um p_2.
3. Konstruiere p_4 als Schnittpunkt zweier Kreise um p_1 und p_3.
4. Konstruiere p_6 als Schnittpunkt zweier Kreise um p_3 und p_1.
5. Konstruiere p_5 als Schnittpunkt zweier Kreise um p_6 und p_4.
6. Messe den Abstand zwischen p_2 und p_5. Stoppe, wenn er korrekt ist, ansonsten gehe zum Schritt 2, um eine andere Lösung zu finden.

Wie aus dem Plan zu ersehen ist, wurde in Punkt $p2$ ein Freiheitsgrad belassen (der Rotationsfreiheitsgrad des starren Objektes). Mit der Konstruktion von Punkt $p3$ startet der iterative Teil des Planes. Abbildung 3.33 zeigt die zwei möglichen Lösungen, wenn die oben angegebenen Konfigurationen für die Konstruktion der Punkte $p4$, $p5$ und $p6$ verwendet werden. Da für jeden der drei Punkte jeweils zwei Lösungen existieren, sind für eine vollständige Suche nach allen gültigen Lösungen des Modells $2^3 = 8$ verschiedene Fehlerfunktionen zu berechnen. Abbildung 3.33(a) zeigt eine von ihnen, in ihr sind die beiden Nullstellen enthalten, die zu den Lösungen in den Abbildungen 3.33(b) und 3.33(c) führen. In sechs weiteren Fehlerfunktionen ist keine Nullstelle enthalten. Die zu den in Abbildung 3.33 gezeigten gespiegelten Lösungen[92] werden mit den Nullstellen in der Fehlerfunktion aus Abbildung 3.34(a) gefunden. Hier sind sowohl der zur Konstruktion von Punkt $p3$ verwendete Winkel als auch die Konfigurationen der Konstruktionen für die Punkte $p4$, $p5$ und $p6$ invertiert.

Bei der Suche nach einer ersten Lösung, z. B. wenn ein neu definiertes Modell konsistent zu machen ist, muss keine spezielle Auswahl getroffen werden. Die zuerst gefundene Nullstelle (bzw. die

[92] z. B. an der Geraden durch 1 und 2

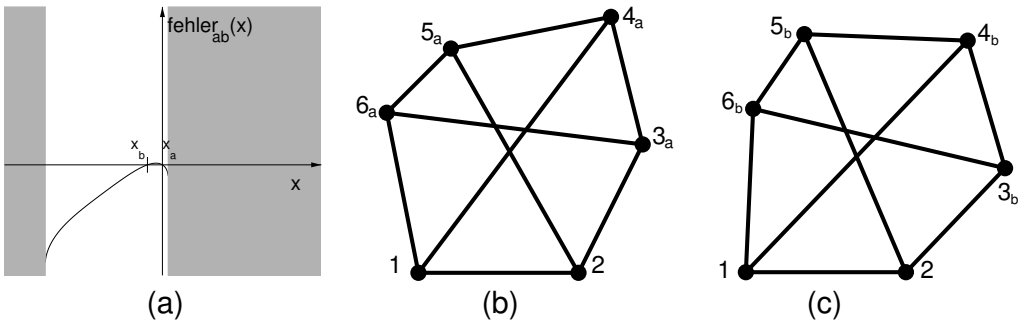


Abbildung 3.33: Mehrfachlösungen für ein Sechseck mit neun Abständen. (a) zeigt die Fehlerfunktion der iterativen Konstruktion, (b) die Lösung, die in der Nähe der alten Lösung liegt und (c) eine weitere mögliche Lösung.

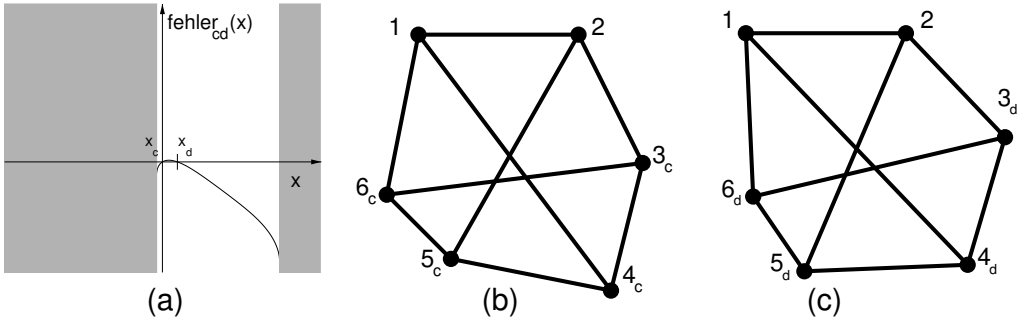


Abbildung 3.34: Die zu Abbildung 3.33 an der Geraden durch die Punkte p_1 und p_2 gespiegelten Lösungen.

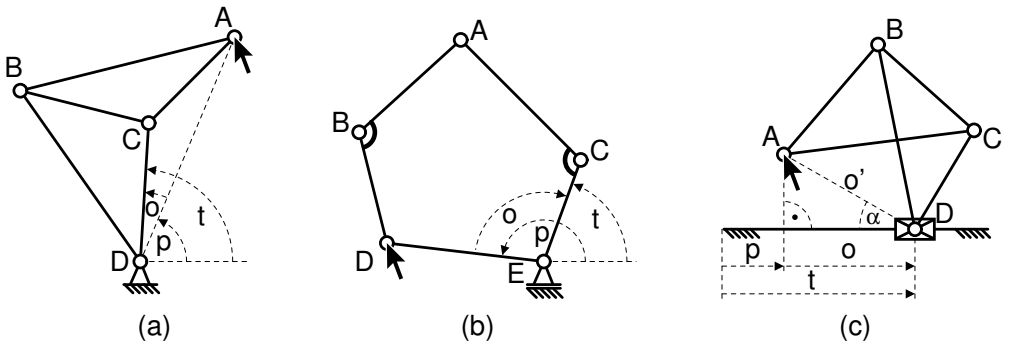


Abbildung 3.35: Berücksichtigung eines aktuellen Parameters p sowie eines zu speichernden Parameters o bei der Wahl der Startwerte mit dem Ziel, in iterativen Konstruktionen stets mit einem Schritt die Lösung zu finden. Für die in (a) und (c) gezeigten Modelle bestünde grundsätzlich die Möglichkeit einen Hilfs-Constraint (Abstand D zu A) einzuführen, um die iterative Konstruktion zu vermeiden. Hier soll aber gezeigt werden, wie man ohne derartige Hilfs-Constraints auskommen kann.

mit ihr verbundene Lösung), die alle Constraints und Hinweise erfüllt, kann verwendet werden. Gegebenenfalls ist über die Fehlerfunktionen zu iterieren (Algorithmus *SWN* auf Seite 85). Grundsätzlich ist auch eine Wahl der nächsten Lösung auf Basis des Konstruktionsplanes sowie der durch ihn implizit gegebenen Fehlerfunktionen möglich, insbesondere wenn die iterativ zu berechnenden Modelle starr bleiben. In diesem Fall ändert sich auch die Form[93] der Fehlerfunktion nicht. Es kann lediglich eine Verschiebung entlang der x-Achse der Fehlerfunktion auftreten, falls das Objekt rotiert wird[94]. Die konstante Form könnte ausgenutzt werden, um constraint-solver-intern die nächste korrekte Nullstelle auszuwählen, z. B. entsprechend dem Anstieg der Fehlerfunktion in der Nullstelle oder auf Basis der relativen Position der Nullstellen zueinander. In kritischen Fällen wäre auch die Kombination mehrerer Kriterien nötig. Trotzdem sind Fehlerfunktionen denkbar, in denen die eindeutige Auswahl der richtigen Nullstelle auf Basis der Form der Fehlerfunktion nicht eindeutig ist. Deshalb wurden entsprechende Ansätze nicht weiter verfolgt. Statt dessen wird die Auswahl der Nullstellen in der Anwendungssoftware vorgenommen. Entsprechende Vorgehensweisen sind in Kapitel 4 und 5 beschrieben.

3.6.6 Wahl der Startwerte

Entsprechend dem Ansatz, dass Lösungen zunächst nahe der zuletzt gefundenen Lösung gesucht werden, lässt sich der Startwert für die neue Suche leicht ermitteln. Hierfür werden bei der Initialisierung der Suche die letzten konsistenten Werte (oder ggf. die Werte des initialen Modells) verwendet. Wenn z. B. die Richtung eines Punktes P_1 zu einem Punkt P_2 zu iterieren ist, so wird der Startwert als Anstieg von P_1 nach P_2 ermittelt.

Für einige Fälle ist die Möglichkeit der speziellen Berechnung eines Startwertes möglich. Hierdurch lässt sich u. U. sogar bereits im ersten Suchschritt eine Lösung finden. In Abschnitt 3.6.1 wurde vorgeschlagen, dass Constraints und Hilfsparameter in das interne Modell eingeführt werden können um unnötige Suchschritte zu vermeiden. Wie dies sich auch durch eine geschickte Berechnung von Startwerten erreichen lässt, wird im Folgenden beschrieben.

---

[93] Gemeint ist ihr Verlauf. Die Fehlerfunktion wird als Kurve aufgefasst, bei der das Wiedererkennen von bestimmten lokalen Eigenschaften, z .B. Extrema, nicht berechenbarer Bereiche usw. die Identifikation der Nullstellen ermöglicht.

[94] Ursache ist, dass der Winkelparameter für die iterative Konstruktion gegenüber der x-Achse des *2D*-Koordinatensystems gemessen wird.

Beispiel 3.22: Vermeidung von Iterationsschritten durch geschickte Startwertewahl Angenommen im Modell aus Abbildung 3.35(a) würde der iterative Plan mit der Konstruktion von C beginnen. Dann wäre t der Parameter für die entsprechende Konstruktion. Wird der Startwert als $t = p+o$ berechnet[95], so benötigt die iterative Konstruktion jeweils nur einen Schritt. Voraussetzung hierfür ist, dass nach der ersten Berechnung des Modells (die mehrere Suchschritte braucht, denn sie kann noch nicht auf o zurückgreifen[96]) der Wert für o ermittelt ($o = t - p$) und gespeichert wird. Denkbar ist auch die laufende Ermittlung und Speicherung von o für den Fall, dass das Modell nicht starr ist (z. B. wenn während der Interaktion Abstände verändert werden).

Die Anwendung des gleichen Prinzips ist auch in Abbildung 3.35(b) veranschaulicht. Wieder ist $t = p + o$, wobei deutlich wird, dass die Winkel vorzeichenbehaftet sein müssen. Abbildung 3.35(c) zeigt ein Modell, in dem ein Abstand iteriert wird.

Um die obigen Beispiele umzusetzen, wäre folgender Ablauf denkbar:

- Bei Erstellung eines Planeintrages zum Start einer iterativen Konstruktion ist zu untersuchen:
  - ob das zu konstruierende Objekt Teil eines starren Körpers ist,
  - ob von diesem Körper schon Objekte berechnet wurden (in Abbildung 3.35(a) wäre das D und in Abbildung 3.35(b) sind das D und E) und
  - ob für ein Objekt dieses Körpers eine Vorzugsposition genutzt werden kann (das wäre A in Abbildung 3.35(a), D ist in Abbildung 3.35(b) bereits berechnet, deshalb ist die Nutzung der Vorzugsposition nicht mehr sinnvoll).
- Wurden keine solche bereits berechneten Objekte gefunden, so lassen sich keine speziellen Startwerte ermitteln. Ansonsten ist zu unterscheiden, welcher Art der Parameter t ist:
  - Ist t ein Winkel, so muss eine Referenzrichtung gefunden werden. Dies könnte die Richtung einer bekannten Geraden sein, oder der Anstieg zwischen zwei bekannten Punkten (in Abbildung 3.35(a) sind das D und die gewünschte Position von A, in Abbildung 3.35(b) sind es D und E).
  - Wenn t ein Abstand ist, führt die einfache Anwendung der Werteermittlung ($t = p + o$) leicht dazu, dass das ermittelte t nicht korrekt ist. Abbildung 3.35(c) zeigt einen solchen Fall. Das Problem besteht darin, dass der zu addierende Wert o abhängig vom Winkel α ist, den das gestellfeste Getriebeglied S und die Richtung von D nach A einschließen. t ist nur korrekt, wenn α sich nicht ändert (z. B. weil A beim Bewegen stets den gleichen Abstand zu S hat). Zur Behebung des Problems müsste ein o' gespeichert werden, das unabhängig von α ist. Es gilt dann $o = o' * cos\alpha$ bzw. bei der initialen Bestimmung $o' = \frac{t-p}{cos\alpha} = \frac{o}{cos\alpha}$, wobei auffällt, dass die Initialisierung fehlschlagen kann (z. B. $\alpha = 90°$).
- Die Art und Weise der Ermittlung des Wertes von p (und wo nötig der Werte o und o') ist so zu speichern, dass vom iterativen Konstruktor darauf zugegriffen werden kann. □

3.6.7 Zusammenfassende Betrachtungen zur Geschwindigkeit

Die Beispiele auf den Seiten 79 bis 82 verdeutlichen, wie wichtig eine globale Sichtweise auf die Fehlerfunktion ist. Nur so kann ein Suchfortschritt gesichert werden und nur so ist es möglich, alle relevanten Nullstellen zu finden. Das vorgestellte hybride Verfahren wird bei der Suche in Fehlerfunktionen zwar immer wieder etwas langsamer sein als z. B. das Newtonsche Verfahren. Durch die Kombination mehrerer Ansätze gelingt es jedoch, Robustheit und akzeptable Geschwindigkeit zu vereinen. Ein Vergleich verschiedener Verfahren anhand der Berechnung eines geometrischen Modells wird in Abschnitt 4.2.3 durchgeführt.

Trotz aller Erfolge bei der Verbesserung der Nullstellensuche bezüglich Robustheit und Geschwindigkeit ist an dieser Stelle jedoch festzustellen, dass die schnellste iterative Konstruktion diejenige

[95] Die Winkel sind vorzeichenbehaftet.

[96] Eine Initialisierung ist aber auch hier denkbar, o wäre der Winkel zwischen den Punkten A, D und C. Ist das Modell konsistent, so wäre o korrekt initialisiert. Demzufolge würde selbst die erste Suche nur einen Schritt benötigen.

ist, die vermieden werden konnte, sei es durch geeignete Transformation des Constraint-Netzes, so dass die direkte Konstruktion möglich wird - oder durch geeignete Wahl der Startwerte, so dass die Schleife nur einmal durchlaufen werden muss.

3.7 Möglichkeiten der Umsetzung von Clustering

Verallgemeinernd lässt sich über Clustering-Ansätze, wie sie z. B. in [Owe91], [BFH+95] oder [HLS97] beschrieben werden, Folgendes sagen:

- Zunächst **analysieren** sie ein Modell, um starre Teilmodelle zu finden und nehmen dann (zumindest implizit) eine entsprechende Zerlegung des Modells in diese Teilmodelle vor.
- Schließlich **bestimmen** sie auf Basis der Zerlegung geeignete Hilfsvariablen, die nach der Konstruktion eines Teilmodells A in diesem abgelesen werden und dann zur (vorzugsweise direkten) Konstruktion eines Teilmodells B herangezogen werden können.
- Die dann folgenden Berechnungen (konstruktiv oder in einem Gleichungslöser) sollen hier nicht mehr als für das Clustering charakteristisch angesehen werden.

In den berechneten Clustern (bzw. den starren Teilmodellen, die sie repräsentieren) wird die relative Position von Objekten zueinander explizit beschrieben. Objekte können hierbei sowohl einzelne Parameter sein (z. B. Abstände, die sich aus den Maßzahlen einer Kettenvermaßung ergeben) aber auch geometrische Elemente in *2D* oder *3D*. Eine Einteilung bezüglich der in den Clustern verwendeten Koordinatensysteme kann wie folgt vorgenommen werden.

- Die einfachste Form stellen sogenannte Äquivalenz-Cluster dar. Sie werden meist auf Basis von Gleichheits-Constraints gebildet. Derartige Cluster benötigen kein Koordinatensystem.
- Cluster von Parametern sind in einem Koordinatensystem mit einer Dimension definiert. Das Koordinatensystem kann geschlossen sein, z. B. wenn es sich bei den Parametern um Winkel handelt.
- Cluster von *2D*-Geometrieelementen werden in einem (bezüglich der Szene bzw. dem Modell lokalen) *2D*-Koordinatensystem definiert und
- analog hierzu *3D*-Geometrieelemente in einem *3D*-Koordinatensystem.

Die Benennung der Arten von Clustern (*0D* bis *nD*) erfolgt entsprechend der Anzahl der Parameter (in Implementierungen sind das typischerweise Gleitkomma-Werte), die zur Bestimmung der relativen Position der Objekte zueinander benötigt werden. Wie in den folgenden Ausführungen gezeigt wird, gibt es aber bei dieser Einteilung auch Ausnahme- bzw. Spezialfälle.

3.7.1 Äquivalenz-Cluster (0D)

Der hier verwendete Begriff *Äquivalenz-Cluster* ist in dem von Brüderlin [Brü87, Abschnitt 3.2.4] beschriebenen engeren Sinne zu sehen, wo Gleichheits-Constraints zwischen Parametern (oder auch Objekten) zu Äquivalenz-Clustern führen. Der in [Brü87, Seite 44] beschriebenen Erweiterung des Begriffs auf starre Anordnungen geometrischer Objekte (es werden Beispiele für Vektoren und Dreiecke gegeben) wird in der vorliegenden Arbeit nicht gefolgt. Derartige Cluster werden entsprechend dem Raum, in dem die Objekte eingebettet sind, eingeteilt.

Allgemein lässt sich feststellen, dass in Äquivalenz-Clustern Objekte mit gleichen Werten vereinigt sind. Parameter zur Beschreibung der relativen Position zueinander sind nicht nötig, deshalb

können sie auch als 0D-Cluster bezeichnet werden. Wie in Abschnitt 3.2.2 gezeigt, ist es jedoch sinnvoll bestimmte Abweichungen wie die Entgegengerichtetheit bei Richtungsvektoren zuzulassen. Diese Abweichungen können allerdings nur bestimmte Zustände annehmen, so dass kein Parameter zur Beschreibung dieses Spezialfalls nötig ist[97].

3.7.2 Parameter-Cluster (1D)

Ein einfacher Ansatz zur Bildung von Parameter-Clustern besteht in der Ausnutzung domänenspezifischer Constraints. Verroust, Schonek und Roller beschrieben eine derartige Vorgehensweise in [VSR92]. Dort sammelten sie Informationen über Winkel zwischen *2D*-Liniensegmenten in sogenannten *CA-Sets*, die als Winkel-Cluster angesehen werden können. Gleichmächtig sind z. B. auch die *Rewrite Rules* nach Brüderlin [Brü87, Brü93], mit denen sich die Eigenschaften wie Transitivität, Symmetrie und Reflexivität ($winkel(a, a, 0)$) modellieren lassen:

$$\begin{array}{ccc} [winkel(a,b,\alpha), winkel(b,c,\beta)] & \rightarrow & [winkel(a,b,\alpha), winkel(a,c,\alpha+\beta)] \\ [winkel(a,b,\alpha)] & \rightarrow & [winkel(b,a,-\alpha)] \end{array} \tag{3.10}$$

Eine sehr allgemeine Herangehensweise an Parameter-Cluster müsste beliebige Constraints, in denen Parameter referenziert sind, berücksichtigen, so auch Gleichungs-Constraints.

Beispiel 3.23: Parameter-Cluster basierend auf Gleichungen (Addition)
Definiert seien folgende Gleichungen, die aus einer Kettenvermaßung resultieren könnten:

$$\begin{array}{rcrcl} d_1 & = & d_0 & + & a \\ d_2 & = & d_1 & + & b \\ d_3 & = & d_2 & + & c \end{array} \tag{3.11}$$

Sind die Maße a, b und c fix, so bilden die Abstände d_0 bis d_3 ein Abstands-Cluster, z. B. als Abstände von Ebenen zu einer Referenz-Ebene oder als Abstände von Liniensegmenten zu einer Koordinatenachse. Eventuell sind die Gleichungen im Modell nur implizit vorhanden und wurden aus drei „paralleler Abstand"-Constraints durch Transformationen gewonnen (vgl. Abschnitt 3.2.3). a, b und c wären dann beispielsweise Abstandsparameter für parallele Ebenen. Die Ebenen bilden so eigentlich ein *3D*-Cluster, denn sie haben zueinander feste relative Positionen. Die Richtungsvektoren (die Normalen) der Ebenen sind in diesem Fall gleich und werden konsequenterweise in einem Äquivalenz-Cluster zusammengefasst. Dies ist somit auch ein Beispiel dafür, dass *3D*-Geometrieelemente in Clustern niedriger Dimension (Äquivalenz-Cluster, Parameter-Cluster usw.) eingeordnet werden können, wenn die *3D*-Geometrieelemente in geeignete Subelemente zerlegt wurden. □

Allgemein kann bei Parameter-Clustern von Starrheit im geometrischen Sinne jedoch nicht gesprochen werden, wie folgendes Beispiel zeigt.

Beispiel 3.24: Parameter-Cluster basierend auf Gleichungen (Multiplikation)
Definiert seien folgende Gleichungen:

$$\begin{array}{rcrcl} d_1 & = & d_0 & * & k_1 \\ d_2 & = & d_1 & * & k_2 \\ d_3 & = & d_2 & * & k_3 \end{array} \tag{3.12}$$

Sind die Faktoren k_1, k_2 und k_3 fix, so bilden die Parameter d_0 bis d_3 ein Cluster. Wenn es sich bei den Parametern d_0 bis d_3 wie in Beispiel 3.23 angenommen um Abstände von vier Ebenen zu einer

[97] Diese Betrachtungsweise ist vergleichbar mit der von Freiheitsgraden geometrischer Objekte. Um z. B. einen Punkt in *2D* zu repräsentieren werden zwei Parameter (zumeist Koordinaten in der euklidischen Ebene) benötigt. Dementsprechend hat der Punkt zwei Freiheitsgrade. Wird der Punkt als Schnitt zweier Kreise berechnet, gelten diese Freiheitsgrade als aufgebraucht bzw. die entsprechenden Koordinaten gelten als bestimmt - obwohl es eine diskrete Anzahl von Zuständen (die zwei möglichen Lösungen) gibt. Die möglichen Zustände werden *nicht* als neuer Freiheitsgrad gewertet.

Koordinaten-Ebene handeln würde, dann würde eine Verschiebung einer dieser Ebenen die Abstände zwischen den Ebenen ändern. Das aus den Ebenen bestehende Teilmodell wäre *nicht* starr[98]. □

Das Beispiel zeigt, dass eine generische Analyse (siehe z. B. [HLS98]) in derartigen Fällen scheitert. Folglich scheint die in Abschnitt 3.2 beschriebene Beschränkung auf spezielle im Constraint-Netz zu erkennende Muster sinnvoll.

3.7.3 Cluster in der Ebene (2D)

Die klassischen Clustering-Ansätze beziehen sich auf starre Objekte in *2D*. Die referenzierten Objekte (Geraden und Punkte) haben jeweils zwei Freiheitsgrade. Parameter wurden grundsätzlich als fix angenommen.

Owen veröffentlichte 1991 in [Owe91] seinen *Top-Down*-Ansatz, der im Wesentlichen auf der sukzessiven Zerlegung eines Graphen in Teilgraphen an sogenannten *Articulation Pairs* beruht. Hierbei handelt es sich um jeweils zwei Punkte, an denen sich der Graph in zwei oder mehr Teilgraphen zerlegen lässt. 2002 ist der Ansatz von Joan-Arinyo und anderen neu formalisiert worden ([JSVV02]).

Eine sinngemäß entgegengesetzte Richtung wurde von Verroust, Schonek und Roller 1992 in [VSR92] beschrieben. Die Cluster, in der Veröffentlichung auch als als *CD-Set* bezeichnet, wurden über bestimmte Regeln aus einzelnen Objekten bzw. kleineren Clustern gebildet. Deshalb wird hier auch von einem *Bottom-Up*-Ansatz gesprochen.

Bei dem in [BFH$^+$93, BFH$^+$95] von Hoffmann et al. beschriebenen Ansatz handelt es sich ebenfalls um einem *Bottom-Up*-Ansatz. Er wurde in diversen Veröffentlichungen noch tiefergehend erörtert und ausgebaut [FH96b, Fud95, BFH$^+$95, FH96a, FH97]. In jüngerer Zeit erfolgte auch eine weitere Vervollständigung der Beschreibung von Konstruktionen für Kreise mit variablem Radius ([HC02a, HC02b]).

Vergleichende Beispiele zu den Verfahren können [DM97] und [Bri01] entnommen werden. Die Gegenüberstellung zeigt u. a., dass Clustering-Verfahren und regelbasierte Verfahren wie die *Rewrite Rules* [Brü87, Brü93] grundsätzlich gleichmächtig sind. Unterschiede ergeben sich insbesondere in der Effizienz der Lösungsfindung und der Steuerbarkeit der Lösungssuche.

3.7.4 Cluster im Raum (3D)

1994 übertrugen Hoffmann und Vermeer in [HV95a, HV95b] den *2D*-Clusteringansatz aus [BFH$^+$93] auf *3D*. Hoffmann setzte die Untersuchungen in Zusammenarbeit mit verschiedenen Partnern fort, u. a.:

- Durand ([Dur98] und [DH00])
  Der Fokus der Arbeiten liegt auf der Berechnung räumlicher Cluster mittels symbolischer Umformungen und insbesondere auch auf dem anschließenden Einsatz von Homotopie für die numerische Lösung der aufgestellten Gleichungssysteme.
- Yuan ([HY01])
  Hier werden für ausgewählte Probleme der Berechnung räumlicher Cluster unterschiedliche Möglichkeiten der Lösungssuche diskutiert:
  - auf algebraischem Wege (Aufstellen algebraischer Gleichungen, Umformung des Gleichungssystems und anschließende direkte oder numerische Lösungsfindung),
  - durch geometrische Transformationen (z. B. die Einführung von Hilfsgeometrien oder die Ersetzung von Constraints) und

[98] Es könnte als Skalierungs-Cluster bezeichnet werden.

  - mittels einer Kombination von direkten Konstruktionen und numerischer Lösungsfindung[99].

- Gao und Yang ([HGY02] bzw. [GHY04])
  Gegenstand dieser Arbeiten ist die nähere Beschreibung eines iterativ-konstruktiven Ansatzes (*locus intersection*). Seine Anwendung im Programmsystem MMP/Geometer wird in [GL04] dargestellt.

Doch auch die im folgenden Abschnitt beschriebenen Ansätze ermöglichen die Behandlung von Clustern im Raum[100].

3.7.5 Cluster im nD

Die bisher besprochenen Ansätze gehen in der Regel von einer Analyse des Constraint-Netzes aus, die das Ziel hat, bestimmte Muster in ihm zu finden. Diese Muster stammen typischerweise aus der *2D* oder *3D* Domäne, wodurch die Ansätze nicht allgemein anwendbar sind.

Ansätze, die diese Einschränkung überwinden, nutzen bipartite Graphen für die Analyse des Constraint-Netzes. Hierbei ist sowohl die Nutzung einer matching-basierten Analyse ([LM98][101]) als auch die Nutzung einer flussbasierten Analyse (z. B. [HLS98]) beschrieben. Die Vereinheitlichung der unterschiedlichen Domänen erfolgt über die Berücksichtigung einer Konstante k, die die Anzahl der Freiheitsgrade eines starren Körpers im jeweiligen Raum angibt. Allgemein gilt $k = \frac{d \cdot (d+1)}{2}$, wobei d die Dimension des jeweiligen Raumes ist. Den Ansatz aus [HLS98] behandeln eine Reihe weiterer Veröffentlichungen, z. B. [HLS97], [HLS01a] und [HLS01b]. Den Kern des Ansatzes bildet ein sogenannter DR-Planer (*Decomposition-Recombination* Planer). Die Anwendung eines DR-Planers auf *2D*- und *3D*-Modelle im Programmsystem FRONTIER ist Gegenstand von [OSMA01] und [Oun01].

Wie in den vorherigen Abschnitten gezeigt wurde, können jedoch Cluster unterschiedlicher Dimensionen in ein und demselben Modell auftreten, wodurch die Festlegung einer Konstante k für ein Modell allgemein nicht möglich ist. Zudem bereiten degenerierte Fälle Probleme:

- Cluster können aufgrund von Rotationssymmetrie (z. B. zwei Punkte in *3D* mit einem fixen Abstand) oder aufgrund ihrer unendlichen Ausdehnung (z. B. zwei parallele Ebenen in *3D* mit einem fixen Abstand) weniger Freiheitsgrade als im allgemeinen Fall haben. Derartige Fälle sind als Muster im Constraint-Netz erkennbar.

- Ihre Bestandteile können zueinander spezielle relative Positionen einnehmen, wobei dieser Zustand

  - temporär durch eine entsprechende Interaktion verursacht ist oder
  - permanent durch Constraints (implizit, also nicht direkt erkennbar) erzwungen wird

  (z. B. Inzidenz einer Ebene zu drei Punkten des Clusters C, wobei die Punkte jedoch colinear sind).

---

[99] Der Ansatz wird später in Zusammenarbeit mit Gao und Yang als *locus intersection* weiterentwickelt.

[100] Beachte auch die grundsätzliche Gleichmächtigkeit regelbasierter Ansätze wie die 1987 von Brüderlin beschriebenen *Rewrite Rules*.

[101] In diesem Ansatz werden freie Parameter sogar explizit zugelassen. Der Ansatz beruht auf der Dekomposition nach Dulmage und Mendelsohn, mit der eine Zerlegung in einen unter-, voll- und überbestimmten Subgraphen möglich ist. Darauf folgt die weitere Zerlegung des vollbestimmten Teils in nicht weiter zerlegbare Subgraphen (laut [LP86] wurde auch eine derartige Vorgehensweise schon durch Dulmage und Mendelsohn beschrieben). Das Problem des Ansatzes von Lamure und Michelucci besteht darin, dass freie (d. h. nicht fixierte) Parameter und geometrische Objekte (z. B. auch die Freiheitsgrade eines starren Körpers) schnell dazu führen, dass der unterbestimmte Teil des Constraint-Netzes sehr groß wird. Dieser Teil wird jedoch nicht weiter zerlegt, was schlecht für die nachgeschalteten Berechnungsalgorithmen ([LM98] schlagen numerische oder symbolische Ansätze vor) ist, denn diese arbeiteten umso effizienter, je kleiner die zu lösenden Gleichungssysteme sind.

Die Anzahl der Freiheitsgrade k eines Clusters (bzw. die Anzahl der Freiheitsgrade des entsprechenden starren Körpers) kann also aus verschiedenen Gründen nicht als Konstante angesehen werden. Sie ist vielmehr entsprechend den geometrischen Gegebenheiten zu bestimmen. Die logische Schlussfolgerung ist ein Clustering-Ansatz, der dies berücksichtigt. Ein derartiger Ansatz wird in [JNT04] und [JNT03] vorgestellt[102]. Kerngedanke ist der Übergang von der sogenannten strukturellen Starrheit[103] zur erweiterten strukturellen Starrheit[104], die Spezialfälle berücksichtigt. Somit vereint der Ansatz die allgemeine bipartite Analyse mit der Suche nach speziellen Mustern[105].

3.7.6 Vergleich mit der iterativen Konstruktion

Der Einsatz des Clusterings im vorgestellten Solver wäre grundsätzlich wünschenswert. Es gibt viele Beispiele, bei denen Clustering-Ansätze helfen würden, den durch iteratives Konstruieren verursachten zeitlichen Aufwand sowie Nachteile in der Steuerbarkeit zu vermeiden. Dennoch ist es möglich, dass unter Berücksichtigung der Menge und Vielfalt der potentiellen Modelle die iterativen Konstruktionen Vorteile haben. Dies soll in den folgenden vergleichenden Betrachtungen verdeutlicht werden.

Die beim Clustering durch die Analyse gefundenen Berechnungsschritte können als Hilfskonstruktionen aufgefasst werden (siehe Abschnitt 2.9). Auch iterative Konstruktionen haben während der Nullstellensuche beim Durchrechnen der unterschiedlichen Varianten (also temporär) den Charakter von Hilfskonstruktionen. Im iterativen Plan wird jedes Objekt jedoch nur durch einen Konstruktor, d. h. an einer Stelle berechnet, während bei den Berechnungen in „klassischen“ Clusteransätzen ein Objekt oft mehrmals berechnet wird, jeweils im Koordinatensystem eines anderen Teilmodells - zunächst in einer oder mehreren Hilfskonstruktionen und dann in einer abschließenden Konstruktion. Für einige Probleme (z. B. die Konstruktion des in Abschnitt 2.9 gezeigten Fünfecks) sind die Clusteransätze effizienter, denn gegenüber der ausschließlich direkten Konstruktion in Clustering-Ansätzen wird bei einer iterativen Konstruktion je nach Entfernung der Startwerte von der Lösung und den Genauigkeitsanforderungen üblicherweise das zehn- bis zwanzigfache an Rechenzeit benötigt. Andererseits müssen die rein konstruktiven Clustering-Ansätze erweitert werden, um Probleme wie in Abbildung 3.33 lösen zu können, während iterative Konstruktionen hierfür bereits mächtig genug sind.

In starren Teilmodellen ist bei geeigneter Wahl des zu testenden Parameters (nach dem Finden der ersten Lösung) bei iterativen Lösungen die alte Lösung stets gleich der neuen Lösung (vgl. Abschnitt 3.6.6). Damit führt bereits die erste Konstruktion zum Erfolg und Hilfskonstruktionen in Form von Suchschritten werden nicht vorgenommen. Da beim Clustering die Hilfskonstruktionen stets ausgeführt werden müssen, ist in derartigen Fällen der auf ihm beruhende Berechnungsablauf stets langsamer als der beim iterativen Konstruieren.

Auch bei interaktiven Bewegungssimulationen von Mechanismen können die obigen Clustering-Ansätze nur zur Behandlung starrer Teilmodelle dienen. Für das Finden der optimalen Position kann Clustering nicht herangezogen werden. Hier sind Konzepte wie das iterative Konstruieren notwendig.

---

[102] Die notwendige Analyse der geometrischen Verhältnisse zur Auffindung der Spezialfälle ist in [Jer02] beschrieben. Bei der Erörterung des Ansatzes schließen die Autoren jedoch die Fixierung von Objekten und variable Parameter explizit aus. Zumindest die Fixierung ließe sich jedoch ansatzkonform realisieren, indem ein Koordinatensystem modelliert wird (per Constraint senkrecht aufeinander stehende Geraden (*2D*) oder Ebenen (*3D*)) und Constraints, die Fixierung bezüglich diesem Koordinatensystem beschreiben.

[103] *structural rigidity*, die Starrheit, die auf Basis der generischen Struktur des Constraint-Netzes abgeleitet wurde.

[104] *extendet structural rigidity*, die Starrheit, die auf Basis der generischen Struktur des Constraint-Netzes und der Berücksichtigung der Spezialfälle abgeleitet wurde.

[105] In [JNT03] wird darauf hingewiesen, dass auch im DR-Planer bestimmte Fälle erkannt und dann speziell behandelt werden. Die entsprechende grundsätzliche Vorgehensweise in Ficucs zeigt Abschnitt 3.2.

3.7.7 Zusammenfassende Betrachtung

Einfache Formen des Clusterings (Äquivalenz-Cluster und testweise auch *1D*-Cluster bei Ebenen mit „Paralleler Abstand"-Constraints) werden bereits jetzt in Ficucs umgesetzt. Die Implementierung eines Ansatzes, wie er in [JNT03] beschrieben wird, ist zur Vermeidung von Iterationen und zur Behandlung von degenerierten Fällen[106] sinnvoll. Jedoch muss hierbei eine umsichtige Integration erfolgen, d. h. andere Konzepte, wie die Priorisierung von Freiheitsgraden und die iterative Konstruktion dürfen hiervon nicht beeinträchtigt werden.

Für Ficucs wäre die Integration von Clustering insbesondere im Sinne von separaten[107] Hilfskonstruktionen zur *initialen*[108] Berechnung der Startwerte iterativer Konstruktionen denkbar. Cluster würden hier zur Vermeidung von Informationsverlust (und letztlich zur Vermeidung der Iterationen) beitragen, denn einmal gebildet bzw. berechnet lassen sich aus ihnen die benötigten Informationen über die Startwerte ableiten. Vor einer Entnahme der benötigten Information muss jedoch die Konsistenz des Clusters sichergestellt werden:

- Bei Hilfskonstruktionen, die selbst iterativ konstruiert wurden, ist:
  - die iterative Konstruktion durchzuführen (pessimistischer Ansatz) oder
  - zunächst eine Konsistenzüberprüfung (optimistischer Ansatz) und bei Inkonsistenz eine anschließende iterative Konstruktion durchzuführen.

  Da die Konsistenzprüfung schneller als die iterative Konstruktion ist[109], wäre der optimistische Ansatz vorzuziehen, es sei denn die Modelle sind in der Regel inkonsistent (nicht starr). Dann würde der erste Ansatz, in dem nicht einmal der Versuch der Konsistenzüberprüfung gemacht wird, Geschwindigkeitsvorteile bringen.
- Wenn (wie in Ficucs) die verwendeten Parameter variabel sein können, ist auch für nicht iterativ konstruierte Cluster eine Konsistenzsicherstellung nötig. Wie bei den iterativen Hilfskonstruktionen kann hier stets ohne vorheriges Testen konstruiert werden oder die Konstruktion erfolgt nur nach Fehlschlagen der Tests (d. h. bei Nichterfüllung mindestens eines Constraints oder bei erkannter Änderung von Parametern).

[106] In der aktuellen Implementierung erfolgt die Behandlung erst während der Ausführung des Konstruktionsplanes, wodurch degenerierte Fälle nicht in der Freiheitsgradanalyse berücksichtigt werden können.

[107] D. h. sie wird auf Basis von Hilfsobjekten durchgeführt, die speziell für das Cluster dem Modell hinzugefügt wurden.

[108] Bei starren Körpern und geeigneter Wahl der Iterationsparameter wird für alle folgenden Konstruktionen stets der erste Versuch erfolgreich sein (siehe Abschnitt 3.6.6).

[109] Insbesondere wenn die Invarianz der bei der letzten Konstruktion verwendeten Parameterwerte getestet werden kann. Ein Cluster, das nur mit Hilfe von parameterfreien Constraints und Constraints, die Konstanten referenzieren, gebildet wird, ist sogar stets konsistent.

Kapitel 4

Die Behandlung von Mehrfachlösungen

Ein zentrales Problem der Berechnung constraint-basierter Modelle, das in der vorliegenden Arbeit bisher noch nicht ausreichend erörtert wurde, ist die Behandlung von Mehrfachlösungen. In diesem Kapitel sollen deshalb Ausführungen zu zwei Hauptanwendungsgebieten des vorgestellten Constraint-Solvers gemacht werden: der Bewegungssimulation von Mechanismen und der Konstruktion starrer Körper. Diese Unterteilung spiegelt sich im Wesentlichen in der Wahl der diskutierten Beispiele wider. Im Laufe des Kapitels wird jedoch deutlich, dass in beiden Anwendungsgebieten sehr ähnliche Problemstellungen zu lösen sind, z. B. kann bei einem (an sich) starren Körper eine kontinuierliche Parameteränderung durchzuführen sein oder aus einer initial inkonsistenten Beschreibung eines Mechanismus ist eine erste Lösung zu berechnen. Dementsprechend kann es sinnvoll sein, einem Nutzer zunächst mögliche Lösungen zu präsentieren und die vom Nutzer ausgewählte als Start für die Bewegungssimulation zu nutzen.

Im Einzelnen geht es jeweils darum zu zeigen,

1. wie eine (bzw. genau die) für den Nutzer plausible Lösung ausgewählt werden kann[1] und
2. wie jede einzelne Lösung als mögliche Variante eines Modells gefunden werden kann.

Hierbei wird eine Diskussion der in der Literatur beschriebenen Ansätze in Bezug auf Mehrfachlösungen stattfinden und es werden vertiefende Beispiele für die eigene Umsetzung im Kontext direkter und iterativer Konstruktionen erörtert.

4.1 Mehrfachlösungen bei Bewegungssimulationen

4.1.1 Ansätze anderer constraint-basierter Systeme

Bereits Ivan Sutherland behandelte in den sechziger Jahren in Sketchpad die Darstellung und Bewegung von Mechanismen ([Sut63a] und [Sut63b]). Anfänglich stand die korrekte Auswahl aus Mehrfachlösungen jedoch noch im Hintergrund des Interesses. Aber auch Kramer behandelte 1990 Mehrfachlösungen noch als Nebenproblem[2] und bot keine praktikablen Lösungsmöglichkeiten an[3].

[1] Das muss gezielt geschehen, d. h. auch bei komplexeren Modellen soll möglichst schnell in der Menge der Lösungen (ihre Größe kann exponentiell zur Zahl der Objekte anwachsen) eine passende Lösung gefunden werden.

[2] in [Kra90] bzw. mit nahezu identischem Inhalt 1992 in [Kra92].

[3] Er schreibt in Bezug auf seinen konstruktiven Ansatz lediglich, dass es Sache des Nutzers sei, über ein grafisches Interface des TLA („*The Linkage Assistant*“) die richtige Lösung zu wählen.

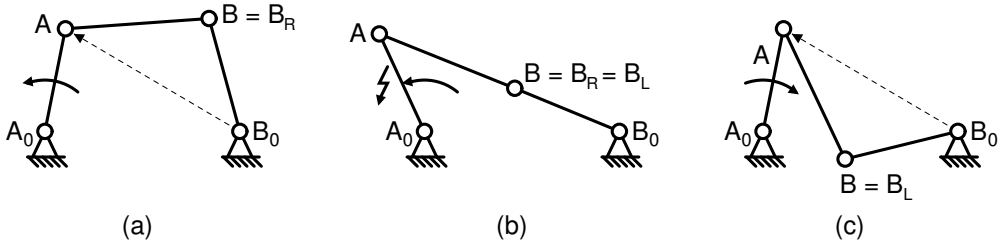


Abbildung 4.1: Bewegung einer Viergelenkkette bis zur Strecklage und Wechsel der Lösungsauswahl bei der Konterbewegung. (a) Zeigt die Bewegung des Getriebegliedes A_0A. Das Gelenk B liegt hier stets auf der rechten Seite der Kriteriumsgeraden durch B_0 und A. (b) Die Strecklage ist erreicht worden. Für B existiert in dieser Stellung nur eine Lösung. (c) Nach der Strecklage läuft der Antrieb in entgegengesetzter Richtung weiter. Für B wird nun stets die andere Lösung gewählt, die links von der Kriteriumsgeraden liegt.

Im gleichen Jahr stellte Enderton in [End90] Möglichkeiten zum Umgang mit Mehrfachlösungen (er verwendet den Begriff *branches*) vor. Seine Ansätze waren:

- Wahrung der Kontinuität der Bewegungen und
- Wechsel zwischen zwei Lösungen nach inkonsistenten Zuständen.

Für einfache Fälle konnte er zeigen, wie sich mit diesen Ansätzen Trägheit simulieren lässt und hierdurch das *Prinzip des geringsten Erstaunens*[4] für den Nutzer umgesetzt wird.

Beispiel 4.1: Bewegung einer Viergelenkkette nach Enderton

Gegeben sei die in Abbildung 4.1(a) gezeigte Viergelenkkette. Wenn sie am Getriebeglied A_0A angetrieben wird, kommt es im Laufe der Bewegungssimulation zu der in der Abbildung 4.1(b) angegeben Stellung, bei der die zwei Lösungen für B sehr dicht beieinander liegen bzw. aufeinander fallen und dann zu einer Stellung, in der die Konsistenz nicht mehr hergestellt werden kann. Nach Enderton sollte B während der vorhergehenden Bewegungen stets auf derselben Seite der (gerichteten) Geraden durch B_0 und A konstruiert werden, wodurch die Kontinuität der Bewegung von B gewährleistet ist. Wird der Antrieb nach der Strecklage in entgegengesetzter Richtung weiter rotiert, dann sollte die Lage von B bezüglich der Geraden durch B_0 und A gewechselt werden, da ein inkonsistenter Zustand erreicht worden war. Bei den folgenden Konstruktionen wird die geänderte Lage von B wiederum beibehalten (Abbildung 4.1(c)). □

Für größere Beispiele, in denen viele Mehrfachlösungen gleichzeitig zu betrachten sind, ist nach Endertons Aussage der Ansatz jedoch nicht geeignet und er schlägt für die Zukunft interaktive Eingriffsmöglichkeiten durch den Nutzer vor. Es liegt jedoch auf der Hand, dass sowohl bei automatischen wie auch interaktiven Bewegungssimulationen interaktive Eingriffe nur in Ausnahmefällen praktikabel sind. Ein mehr oder weniger komplexer Mechanismus mit entsprechend vielen (z. B. n) nicht eindeutigen Konstruktionen würde bei einer Bewegungssimulation mit s Schritten $n \cdot s$ Anfragen an den Nutzer stellen. Wenn hierbei iterative Konstruktionen erfolgen, dann könnte sich die Anzahl der Anfragen noch vervielfachen.

Brunkharts Arbeit [Bru94] aus 1994 basiert sowohl auf der von Enderton als auch der von Kramer. Sein Ansatz ist sehr interessant, denn er versucht eine Verbindung von direkten Konstruktionen

[4] vgl. [Bar87, Seite 59]: *„principle of least astonishment“*. Barford versuchte bei der Berechnung von Modellen die neue Lösung nahe der alten zu halten und verwendete hierzu den euklidischen Abstand als Kriterium. Allerdings war sein Ziel die plausible Berechnung unterbestimmter Modelle und nicht die Auswahl aus mehreren möglichen Lösungen.

und numerischen Berechnungen umzusetzen[5]. Bei seinen Erörterungen geht er jedoch nicht über die von Enderton diskutierten Ansätze zur Behandlung von Mehrfachlösungen hinaus.

1988 wurde von Hüsing die numerische Lösung von getriebetechnischen Simulationen in [Hüs88] behandelt. Hier tritt grundsätzlich das Problem der Wahl der Startwerte auf (vgl. Abschnitt B.3). Im praktischen Einsatz erwies sich die numerische Lösung der Bewegungssimulation meist als brauchbar, jedoch besteht stets die Gefahr des Springens der Lösung oder des Divergierens. Entsprechende Beispiele können Abschnitt 4.2.3 entnommen werden. Diese Beispiele stellen zwar starre Modelle dar, aber auch jeder Mechanismus kann bei gegebenem Antriebsparameter (bzw. Antriebsparametern) als starr betrachtet werden – die Untersuchungsergebnisse aus Abschnitt 4.2.3 sind mithin auf die hier betrachteten Bewegungssimulationen übertragbar, wenn zum Beispiel ein Antriebswinkel, wie er in Abbildung 4.1 angedeutet ist, als konstant angesehen wird .

Eine für die Simulation von Mechanismen sehr interessante Annahme formulieren Richter-Gebert und Kortenkamp in [RGK01][6]. Sinngemäß heißt es dort, dass Berechnungen bei *kontinuierlichen Änderungen der Eingangsvariablen* stets zu *kontinuierlichen Änderungen der Ausgangsvariablen* führen sollten. Die Formulierung ist deshalb so interessant, weil sie die Möglichkeit impliziert, dass die Eingangsvariablen auch nicht kontinuierlich sein können. Genau dies ist nämlich bei iterativer Lösungsfindung der Fall (siehe Ausführungen in Abschnitt 4.1.2).

Zusammenfassend lässt sich sagen, dass die Behandlung von Mehrfachlösungen bei constraint-basierten Bewegungssimulationen von Mechanismen bisher eher oberflächlich vorgenommen wurde. Dies hat mehrere Gründe:

- Typischerweise werden die Modelle bereits in einer konsistenten Position modelliert und dann in kleinen Schritten bewegt. Die Lösung ist also stets in der Nähe der alten Lösung zu finden. Die sowohl bei den numerischen wie auch bei den konstruktiven Lösungsansätzen auftretenden Probleme, wenn zwei Lösungen dicht beieinander liegen, werden oft verschwiegen. In der vorgestellten Literatur zur Bewegungssimulation von Mechanismen macht Enderton hierzu die ausführlichsten Betrachtungen.
- Die Position des Antriebes wird in der Regel sehr einfach gewählt, typischerweise der Winkel eines Getriebegliedes an einem Festlager. Interaktionen wie das Ziehen an einem beliebigen Punkt oder Getriebeglied werden nicht betrachtet. Dementsprechend treten Probleme mit Mehrfachlösungen, wie sie in Abbildung 4.8 gezeigt werden, nicht auf.
- Im Vordergrund steht immer nur die Suche nach einer bzw. der plausiblen Lösung. Ansätze um alle möglichen Lösungen aufzuzeigen, erscheinen den Autoren als unwichtig (da die Modelle typischerweise initial konsistent sind) und werden deshalb nicht betrachtet.

4.1.2 Realisierung der Ansätze in Ficucs

Bei den in MASP durchzuführenden Bewegungssimulationen kommt es in der Regel darauf an, eine für den Nutzer plausible Folge von Lösungen zu finden, wobei die Abfolge interaktiv (durch Maus, Stift usw.) oder automatisch (durch Parameteränderungen, z. B. der Position eines Schubgelenkes oder dem Anstieg eines Getriebegliedes) gesteuert werden kann. Zur Gewährleistung der Interaktivität bzw. einer flüssigen Animation müssen die Berechnungen zudem schnell plausible Resultate

[5] Allerdings nur in zwei Phasen, erst direkte Konstruktionen (*contraction*) und dann numerische Berechnungen, die laut den Ausführungen auf der *Singular Value Decomposition* (SVD) und *Newton-Raphson* beruhen. Lediglich im Ausblick weist er drauf hin, dass direkte Konstruktion und numerisches Lösen noch enger verknüpft werden müssten.

[6] In der zugehörigen Software für *Dynamische Geometrie* (Cinderella, siehe [RGK00] und für Interna [Kor99]) wird ein history-basierter Ansatz verfolgt, d. h. gegenüber constraint-basierten Systemen im engeren Sinne (die Konstruktionspläne je nach Berechnungsziel selbst erzeugen) muss in den entsprechenden Cinderella-Modellen der Konstruktionsplan vom Nutzer explizit vorgegeben werden. Da in der Berechnungsphase jedoch ähnliche Probleme zu lösen sind wie bei constraint-basierten Systemen, ist eine Betrachtung der dort beschriebenen Probleme und Lösungen sinnvoll (vgl. Abbildung 4.2).

liefern. Deshalb steht beim Einsatz von Ficucs innerhalb von MASP die lokale Lösungsfindung in den jeweiligen Konstruktoren im Vordergrund. Dass die lokale Lösungsauswahl nicht immer richtig getroffen wird belegen einige Beispiele im folgenden Text. Ein Ziel der weiteren Entwicklung von MASP und Ficucs ist es darum, Kriterien zu finden, wann die Korrektheit einer lokal gewählten Lösung unwahrscheinlich ist[7], um in solchen Fällen mit mehr Aufwand nach besseren Lösungen zu suchen.

Um Ficucs im Rahmen von MASP sinnvoll für Bewegungssimulationen einsetzen zu können war es nötig, unterschiedliche Ansätze in ihm zu realisieren. Bei den folgenden Betrachtungen werden zwei Schwerpunkte gesetzt, die direkte Konstruktion und die iterative Konstruktion. Hierbei ist jedoch zu beachten, dass eine iterative Konstruktion typischerweise direkte Konstruktionsschritte enthält und somit die Lösungsauswahl in den einzelnen direkten Konstruktionsschritten auch für die gesamte iterative Konstruktion relevant ist.

Direkte Konstruktion

In Abschnitt 3.5.2 (Beispiele für *2D*-Konstruktoren in Ficucs) und 4.1.1 (Beispiele aus der Literatur) wurden bereits einige mögliche Strategien vorgestellt. Zusammenfassend seien die in Ficucs realisierten verschiedenen Ansätze für einzelne direkte Konstruktionsschritte hier aufgeführt:

- die Beachtung der vergangenen Lösungen (Annahme von Trägheit),
- die Berücksichtigung eines bestimmten vom Constraint-Solver angenommenen Kriteriums (z. B. Konstruktion rechts oder links einer Hilfsgeraden[8]), das aus der gegebenen Geometrie gewonnen wird (vergangene Lösungen werden nicht berücksichtigt),
- eine Kombination aus den obigen beiden Ansätzen (siehe Extrapolationsansatz auf Seite 70) und
- die Auswertung von explizit vorgegebenen Hinweisen.

Ficucs unterstützt unterschiedliche Berechnungsmodi, die durch die Applikation zu setzen sind. Bei kontinuierlicher Parameteränderung an einem Antrieb gehen die Konstruktoren von einer Kontinuität der lokalen Lösungen aus. Interaktive Bewegungen sind zumeist sprunghafter als sie von den Nutzern eingeschätzt werden. Deshalb wird hier die Folge der bisherigen Lösungen nicht berücksichtigt sondern nur die Beibehaltung eines geometrischen Kriteriums. Die daraus möglicherweise resultierenden Unterschiede im Bewegungsverhalten sind im Anhang A.5.1 verdeutlicht (vgl. die Abbildungen A.8 und A.9).

Insbesondere die Auswertung von explizit vorgegebenen Hinweisen kann nicht immer in den Konstruktoren berücksichtigt werden. Derartige Hinweise lassen sich dann nur auf ihr Erfülltsein testen. Momentan bricht Ficucs bei einem Fehlschlagen des Tests ab und es ist Aufgabe der Applikation andere Auswahlen für die Mehrfachlösungen zu treffen, um den Hinweis zu erfüllen. Für eine Behandlung des Fehlschlagens innerhalb von Ficucs wäre ein Backtracking zu implementieren.

Viele der Konstruktoren arbeiten mit lokalen[9] Kriterien, z. B. lokalen *XHS*-Hinweisen oder der Lage bezüglich einer Kriteriumsgeraden. Ein Modell, bei dem die Arbeit mit einer Kriteriumsgeraden versagen kann, ist in Abbildung 4.2 dargestellt. Das Hauptproblem besteht darin, dass die Kriteriumsgerade ihre Richtung schlagartig ändert, wenn die Schubgelenke A und B in Teilabbildung

[7] Erste Ideen hierzu basieren auf einem Vertrauensmaß. Eine mögliche Umsetzung wird im folgenden Text erörtert.

[8] Dies entspricht dem *XHS*-Kriterium zwischen den Punkten A, B und C, wenn die gerichtete Kriteriumsgerade durch die Punkte A und B bestimmt wird und C der zu konstruierende Punkt ist.

[9] Lokal heißt hier, dass diese Kriterien nicht explizit vom Nutzer vorgegeben werden, sie sind vielmehr Bestandteil der Konstruktoren. Wenn im Konstruktor keine explizit durch die Applikation bzw. den Nutzer definierten Hinweise vorliegen, erfolgt die Auswertung der lokalen Kriterien, um ein für den jeweils festgelegten Berechnungsmodus plausibles Verhalten zu erreichen.

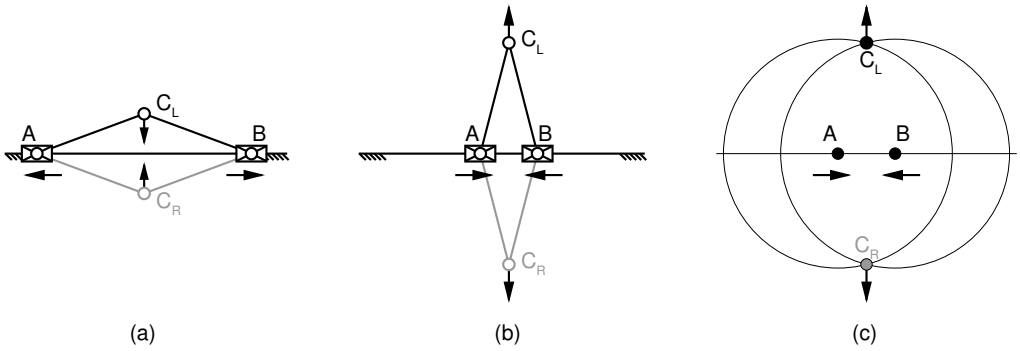


Abbildung 4.2: Problematisches Modell für eine korrekte Auswahl aus zwei möglichen Lösungen. (a) Günstiger Fall. Die Schubgelenke A und B bewegen sich voneinander weg. Die durch A und B verlaufende Kriteriumsgerade für die Auswahl des Drehgelenkes C (kann C_R oder C_L sein) ist wohl definiert. Momentan sei $C = C_L$. Die Auswahl kann entsprechend Abbildung 4.1 erfolgen (auch wenn eine Konterbewegung erfolgt). (b) Ungünstiger Fall. Da A und B sich aufeinander zubewegen besteht die Gefahr, dass die Kriteriumsgerade wegen Gleichheit von A und B nicht nutzbar ist. Zudem wird die bisher stets nach rechts zeigende Kriteriumsgerade nach dem Passieren der Schubgelenke plötzlich nach links zeigen. Dies hat zur Folge, dass die untere Lösung als C_L erkannt wird, denn nach dem Passieren ist die untere Lösung links von der Geraden durch A und B. Dies bedeutet aber, dass C einen für den Nutzer nicht nachvollziehbaren Sprung macht. (c) Geometrisches Modell für die in (b) gezeigte Stellung. Eine ähnliche Abbildung wird in [RGK02] gezeigt. Im dort beschriebenen Programm Cinderella erfolgt eine Nullstellenwahl auf Basis der Verfolgung der einzelnen Lösungen, wobei diese in komplexwertigen homogenen Koordinaten vorliegen.

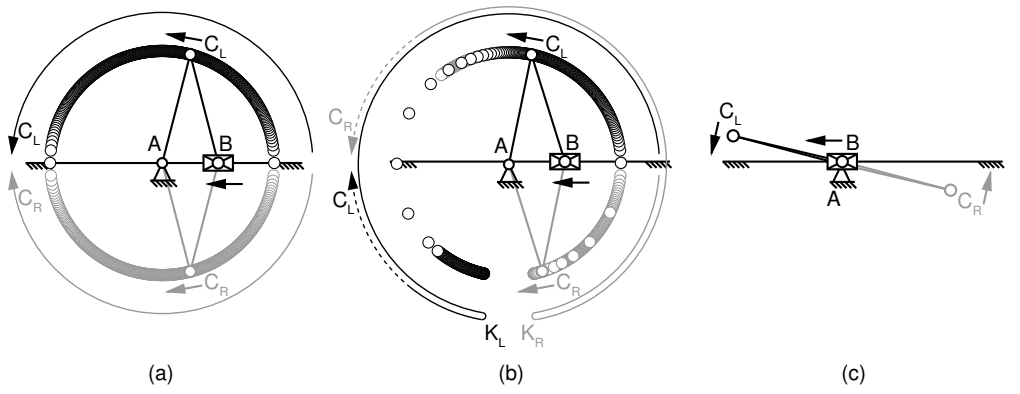


Abbildung 4.3: Grenzfall für die Zulässigkeit von sprunghaften Änderungen. A sei fix und B ändere sich mit konstanter Geschwindigkeit. (a) entspricht der in Abbildung 4.2(b) dargestellten Position und zeigt die kontinuierliche Änderung der beiden möglichen Lösungen. In (b) ist ein geringer Versatz modelliert worden, der dazu führt, dass sich C in einer korrekten Bewegungssimulation nahezu sprunghaft ändern muss. Der angedeutete Verlauf veranschaulicht die Konterbewegung der beiden Bewegungspfade im unteren Teil der Abbildung. Die Umkehrpunkte sind mit K_R und K_L bezeichnet. In (c) wird die Position gezeigt, bei der B nahe A ist. C_L bewegt sich sehr schnell nach unten und C_R nach oben.

(b) die Seiten wechseln. Hier wird das Prinzip *„kontinuierliche Änderungen der Eingangsvariablen sollen kontinuierliche Änderungen der Ausgangsvariablen hervorrufen“* verletzt. Das falsche Vertrauen in die Kriteriumsgerade führt dazu, dass die falsche Lösung für den Punkt C gewählt wird. Für den Nutzer äußert sich das in einem plötzlichen Springen von C. Um das Springen bei kontinuierlich geänderten Positionen von A und B zu vermeiden, könnte ein *Vertrauensmaß* für die Kriteriumsgerade berücksichtigt werden. Der sprunghafte Richtungswechsel (insbesondere nach vorangehender Konstantheit) würde das Vertrauensmaß stark herabsetzen. Statt der Kriteriumsgeraden sollte in einem solchen Fall ein anderes Kriterium für die Wahl von C herangezogen werden. Hier bietet sich ein Kriterium *„Nähe zur letzten Position“* bzw. *„Nähe zu einer extrapolierten Position“* (angewendet auf C) an. Im Anhang A.5.1 wird das durch Ficucs berechnete Bewegungsverhalten von Parallelkurbeln gezeigt. Auch die dort gegebenen Beispiele verdeutlichen die Notwendigkeit der Berücksichtigung eines Vertrauensmaßes für Hilfsgeometrien (z. B. Kriteriumsgeraden).

Die Ermittlung und Berücksichtigung eines Vertrauensmaßes ist momentan noch nicht in Ficucs implementiert. Eine angemessene Vorgehensweise zur Ermittlung eines Vertrauensmaßes muss unbedingt auch Fälle berücksichtigen, die nicht so eindeutig wie der aus Abbildung 4.2 sind. Ein entsprechendes Modell zeigt Abbildung 4.3(b)[10]. Die Schubgelenke befinden sich auf zwei leicht versetzten Getriebegliedern. Nähern sich A und B, so beschleunigt C stark (Abbildung 4.3(c)) und wechselt nach unten. Hier ist der Wechsel erwünscht und bei sehr langsamer Bewegung auch für den Nutzer nachvollziehbar. Wird der Versatz zwischen den beiden Getriebegliedern, auf denen sich die Schubgelenke befinden, kleiner als ein festzulegendes Toleranzmaß, dann sollten die Getriebeglieder jedoch wieder als gleich betrachtet werden, was zu dem in Abbildung 4.3(a) gezeigten Verhalten führt.

Iterative Konstruktionen

Trotz der mitunter auftretenden Probleme haben sich die obigen Ansätze bei Modellen, die durch direkte Konstruktionen berechenbar sind[11], vielfach bewährt. Das Auftreten iterativer Konstruktionen erfordert jedoch eine spezielle Herangehensweise. Zur Verdeutlichung wird nun ein Beispiel erörtert, das Probleme und deren Lösungsmöglichkeiten beim Einsatz von Ficucs für Bewegungssimulationen von Mechanismen mit einem Freiheitsgrad zeigt.

Beispiel 4.2: Problematik der Bewegungssimulation eines Punktes bei einem komplexeren Bewegungspfad

In Abbildung 4.4 ist eine Viergelenkkette dargestellt, wobei sich der Punkt P auf einem bestimmten Pfad bewegt. Dementsprechend kann interaktives Bewegen von P mit dem Mauszeiger nur mit einem Freiheitsgrad vorgenommen werden. Da die Bewegungsbahn nur implizit gegeben ist, lässt sich P nicht direkt durch Projektion des Mauszeigers auf die Bahn konstruieren. Statt dessen wird ein iterativer Plan[12] benutzt. Er beginnt mit der Wahl des Punktes A auf dem Kreis um A_0 und endet mit dem Test, ob P auf der Achse des Getriebegliedes CP liegt.

Jede der beiden durch die iterative Konstruktion entstehenden Fehlerfunktionen hat drei Nullstellen (siehe Abbildungen 4.5 und 4.6). Das mehrfache Auftreten resultiert daraus, dass auch für andere Stellungen der Gelenkkette der Mauszeiger Q auf der Geraden durch die Punkte C und P liegt. Insgesamt existieren sechs derartige Stellungen[13] (siehe Abbildung 4.8), bei der Iteration über den Winkel von A_0 nach A werden entsprechend der einen Fehlerfunktion die drei Stellungen $P = P_1$, $P = P_2$ und $P = P_6$ erreicht.

---

[10] Weitere Testfälle lassen sich durch Variation der Längen der bewegten Getriebeglieder sowie des Winkels der gestellfesten Getriebeglieder erstellen.

[11] Zum Beispiel sind Bewegungssimulationen oft direkt konstruierbar, wenn jeweils ein Antriebswinkel vorgegeben wird (in Abbildung 4.4 könnte das der Winkel des Getriebegliedes A_0A sein). Das Ziehen an beliebigen Punkten oder Getriebegliedern führt hingegen häufig zu iterativen Konstruktionen.

[12] Der Plan ist in Abschnitt A.3 aufgelistet.

[13] Die Lösungen bei denen sich P auf dem (an der Geraden durch A_0 und B_0) gespiegelten Pfad bewegt, kommen für die Analyse nicht in Betracht, denn bei ihnen bilden die drei (starr verbundenen) Punkte A, B und P ein Linkssystem. Das in MASP generierte Modell schreibt jedoch ein Rechtssystem vor.

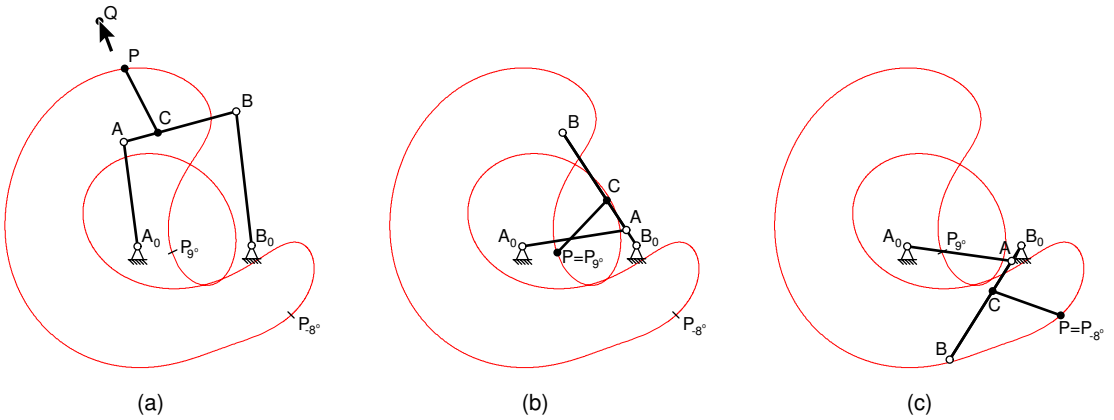


Abbildung 4.4: Viergelenkkette mit dem Pfad des Punktes P. (a) Die Position des Mauszeigers Q könnte zur Auswahl der gezeigten Stellung gedient haben. Die beiden Punkte P_{9° und P_{-8° markieren die Positionen von P, an denen zwischen den in Abbildungen 4.5 und 4.6 gezeigten Fehlerfunktionen gewechselt wird. (b) Stellung des Mechanismus bei einem Anstieg des Getriebegliedes A_0A von 9°. (c) Entsprechende Stellung für -8°.

An den Grenzen zum nicht definierten Bereich (d. h. bei einem Winkel von A_0 nach A gleich 9° bzw. -8°, siehe Abbildung 4.4(b) und (c)) findet ein Wechsel zwischen den beiden Funktionen statt. Hierbei wird für die Konstruktion des Punktes B eine andere Lösung gewählt. Während sich B bei der in Abbildung 4.5 gezeigten Fehlerfunktion stets rechts von der Geraden durch B_0 und A befindet, wird B in der Fehlerfunktion aus Abbildung 4.6 stets links von der Geraden durch B_0 und A konstruiert (Stellungen $P = P_3$, $P = P_4$ und $P = P_5$). □

Das Beispiel zeigt, dass bei Bewegungen nahe der in Abbildung 4.4 dargestellten Stellung die korrekte Nullstelle sicher gefunden werden kann, denn die einzelnen Nullstellen sind gut voneinander zu unterscheiden. Problematischer wird die Wahl jedoch, wenn der Punkt P weiter in Richtung der zweiten Lösung bewegt wird. Bei einem Anstieg A_0A von 50° sind die Nullstellen P_1' und P_2', wie in Abbildung 4.7 dargestellt, kaum noch voneinander zu unterscheiden. Hier kann es leicht passieren, dass die iterative Konstruktion der falschen Nullstelle folgt. Dies führt zu einer immer weiteren Entfernung[14] des gewählten Punktes P zum Mauszeiger Q. Die Wahl der Distanz von P zu Q wäre ein Kriterium, durch dessen Anwendung dieser Fehler vermieden werden könnte. Die Distanz von P zu Q kann jedoch nur zur Unterscheidung dicht beieinander liegender Nullstellen dienen. Würde diese Distanz stets zum Kriterium gemacht, dann könnte es an Stellen, wo sich der Pfad selbst schneidet, zu unerwünschten Sprüngen kommen!

Um den Constraint-Solver generisch zu halten wurde bisher auf eine Behebung des Problems innerhalb von Ficucs verzichtet, d. h. die Applikationssoftware[15] muss (ggf. durch einen Nutzer gesteuert) ein sinnvolles Auswahlkriterium in Form eines Hinweises vorgeben oder die kritischen Fälle gänzlich vermeiden. Die momentan in MASP gewählte Lösung ist in Abschnitt 5.2.3 erörtert[16]. Wie bei den direkten Konstruktionen ließe sich aber auch hier die Berücksichtigung eines Vertrauensmaßes in Ficucs einführen. Die Nähe von zwei oder mehreren Nullstellen zur Nullstelle der erwarteten Lösung ist ein generisches Kriterium, um ein gesenktes Vertrauen zu erkennen. Die Applikation müsste allerdings ein modellabhängiges Abstandsmaß bereitstellen, das durch Ficucs ausgewertet werden kann. Im Falle eines zu geringen Vertrauens in die Lösungsauswahl könnte das

[14] Nach einer einmaligen falschen Auswahl wird kontinuierlich diese falsche Nullstelle weiter verfolgt.
[15] z. B. MASP
[16] MASP kann mit Hilfe von Ficucs den Bewegungspfad analysieren, um dann z. B. bei interaktiven Bewegungen die für den Nutzer plausible Lösung auf diesem Pfad zu finden.

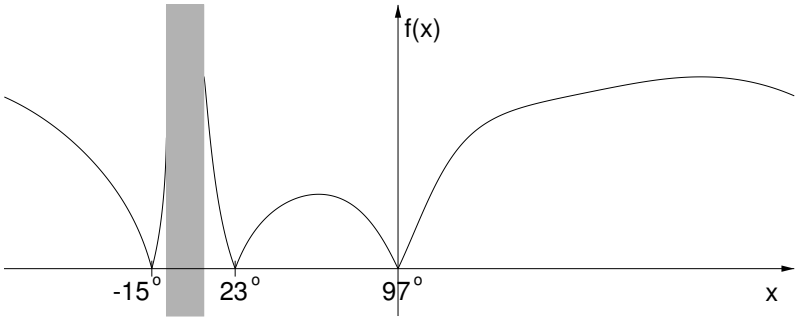


Abbildung 4.5: Fehlerfunktion für die in Abbildung 4.4 gezeigte Gelenkkette, wenn Punkt P bewegt wird und B rechts von der Geraden durch B_0 und A liegt. Die aktuelle Nullstelle liegt bei einem Anstieg des Getriebegliedes A_0A von 97°, das entspricht der in Abbildung 4.4 gezeigten Stellung. Im grau gefärbten Bereich kann die Fehlerfunktion nicht berechnet werden, weil sich hier die Kreise um A und B_0 nicht schneiden.

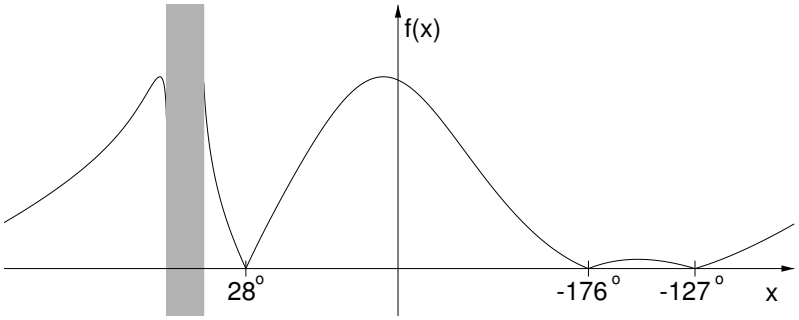


Abbildung 4.6: Fehlerfunktion für die in Abbildung 4.4 gezeigte Gelenkkette, wenn Punkt P bewegt wird und B sich links von der Geraden durch B_0 und A befindet. Die aktuelle Nullstelle liegt bei einem Anstieg des Getriebegliedes A_0A von 97° allerdings in einer anderen Fehlerfunktion (siehe Abbildung 4.5).

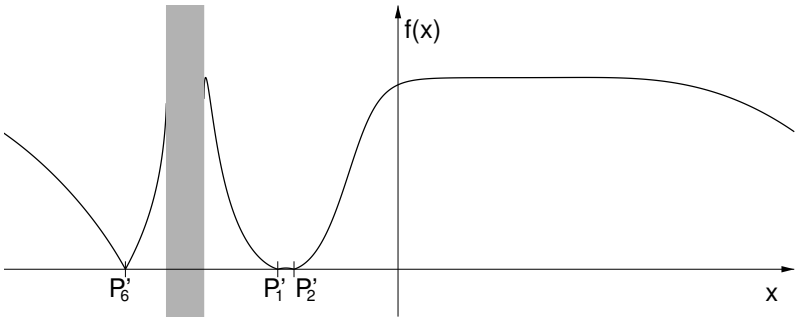


Abbildung 4.7: Fehlerfunktion für die in Abbildung 4.4 gezeigte Gelenkkette, wenn Punkt P bewegt wird und B rechts von der Geraden durch B_0 und A liegt. Gegenüber der Abbildung 4.4 wurde P weiter nach rechts bewegt, bis die Gerade durch P und C nahezu tangential zum Pfad ist. Wie der Fehlerfunktion zu entnehmen ist, liegen die zwei Nullstellen P_1' und P_2' jetzt dicht beieinander.

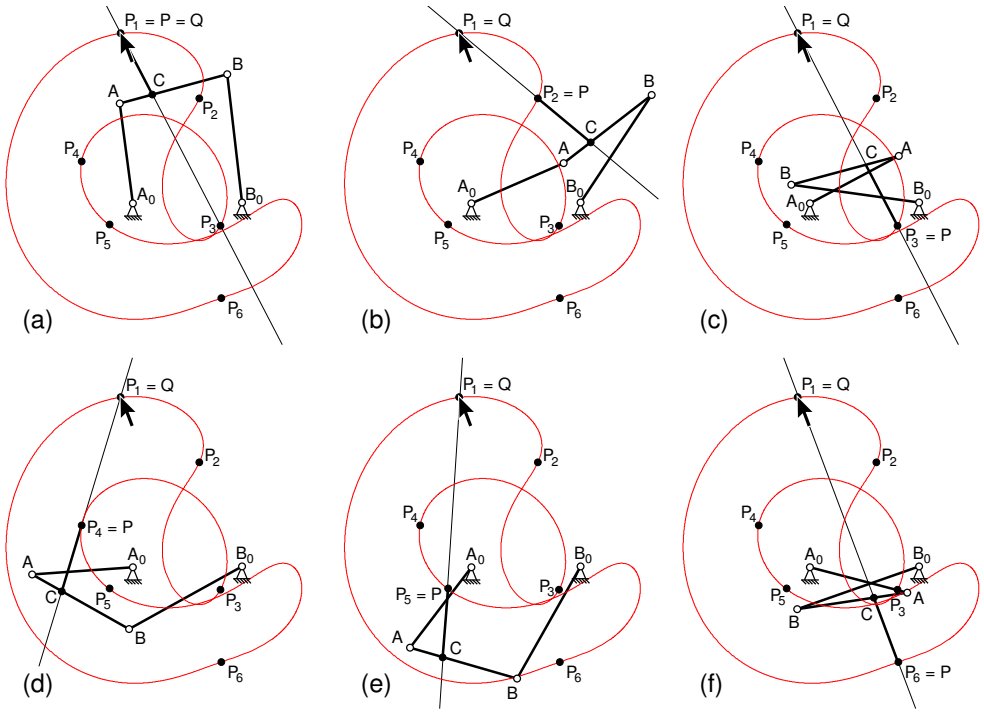


Abbildung 4.8: Sechs Stellungen der Gelenkkette aus Abbildung 4.4. Alle sechs Stellungen sind dadurch gekennzeichnet, dass der Punkt Q (im Beispiel der Mauszeiger) auf der Geraden durch die Punkte C und P liegt. Die möglichen Positionen von P sind mit P_1 bis P_6 bezeichnet. Die Nullstellen aus Abbildung 4.5 (97°, 23° und −15°) entsprechen den in (a), (b) sowie (f) dargestellten Lösungen. Die in Abbildung 4.6 gezeigten Nullstellen von 28°, −176° und −127° führen zu den Stellungen (c), (d) und (e).

Ziel der Interaktion (in Beispiel 4.2 ist das die Nähe zum Mauszeiger) als Entscheidungsgrundlage dienen.

Ist für die iterative Konstruktion der Modus für Bewegungskontinuität aktiv, wird für die Schätzung der nächsten Nullstelle eine Extrapolation vorgenommen, ansonsten wird lediglich die letzte Lösung als Startwert verwendet. Wesentlich für die erfolgreiche Lösungsfindung in iterativen Konstruktionen ist, dass in allen während der Iterationen aufgerufenen (zumeist direkten) Konstruktoren der Modus für Bewegungskontinuität deaktiviert wird, denn wie Abschnitt 3.6.2 verdeutlicht, ist die Nullstellensuche zumeist sehr diskontinuierlich. Lediglich beim ersten Berechnungsdurchlauf kann die Beachtung von Bewegungskontinuität für die einzelnen Konstruktoren sinnvoll sein, vorausgesetzt die Extrapolation des Startwertes der iterativen Konstruktion ist in der Nähe der korrekten Nullstelle. Um die Fehlerfunktion nicht zu verlassen wird bei allen folgenden Iterationen in den einzelnen Konstruktoren die Lösungsauswahl entsprechend einem lokalen Kriterium (z. B. *XHS*) beibehalten, denn wie oben erörtert arbeitet die Nullstellensuche nicht kontinuierlich, sondern sie springt zwar effektiv aber bildlich gesprochen „ziemlich wild im Suchraum umher“. Ein besonderes Problem stellt die mögliche Verschachtelung von iterativen Konstruktionen dar, denn auch hier müsste in den inneren iterativen Konstruktionen die Lösungsauswahl beibehalten werden. Dies ist jedoch wie in Abschnitt 3.6.5 besprochen problematisch.

Es wurde bereits diskutiert, dass MASP das Problem der Unterscheidung von Nullstellen durch eine spezielle Vorgehensweise behebt. Es gibt jedoch noch ein weiteres Problem, das bei der Nutzung von Ficucs im Beispiel 4.2 auftritt. Das für Ficucs typische Festhalten an einer bestimmten Fehlerfunktion führt bei komplexen Bewegungspfaden, die durch mehrere Fehlerfunktionen zu be-

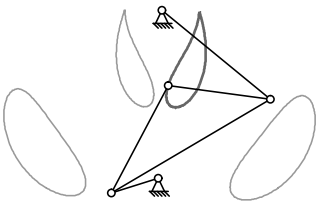


Abbildung 4.9: Mechanismus mit vier getrennten Bewegungspfaden.

rechnen sind, zu einem für den Nutzer unerwarteten Verhalten. An den Übergängen zwischen diesen Fehlerfunktionen (siehe z. B. Abbildung 4.4) stoppt die Bewegung oder es erfolgt sogar ein Rücksprung an eine andere Stelle des Bewegungspfades, die durch die gleiche Fehlerfunktion berechnet wird. Eine allgemein gültige Behandlung des Problems, die unabhängig von einer speziellen Lösung für die Bewegungssimulation von Mechanismen in MASP ist, wäre wünschenswert. Hierfür ist ein Ansatz zum gezielten Wechsel zwischen den Fehlerfunktionen zu finden. Basis dieses Ansatzes könnte die Feststellung eines nicht definierten Bereiches in der Fehlerfunktion sein sowie das Auffinden eines inneren Konstruktors, bei dem zwei mögliche Lösungen dicht beieinander liegen und somit der Wechsel zwischen diesen Lösungen den Übergang in die „benachbarte“ Fehlerfunktion ermöglicht.

Systematische Suche nach allen Lösungen

Wie zuvor beschrieben, wird bei iterativen Konstruktionen von Ficucs die Fehlerfunktion während der Nullstellensuche nicht gewechselt. Es ist jedoch möglich, dass eine Applikation in einem bestimmten Konstruktionsplan explizit die gewünschte Lösung (pro Konstruktor) auswählt. Somit lassen sich gezielt alle Fehlerfunktionen durchtesten[17]. Fehlerfunktionen, die nicht erwünscht sind, müssen über Hinweise ausgeschlossen werden. Dies bietet sich z. B. für das Modell in Abbildung 4.4 an, wo die Definition eines *RHS*-Hinweises bezüglich der Punkte A, B und P sinnvoll ist.

Abbildung 4.9 zeigt ein Modell mit seinen vier möglichen Bewegungspfaden. Zwischen den dargestellten Bewegungspfaden darf in einer korrekten Simulation jedoch nicht gewechselt werden. Um an einem körperlichen Modell einen Wechsel durchzuführen wäre es notwendig, den Mechanismus auseinanderzunehmen und neu zusammenzubauen. Es muss deshalb vom Nutzer entschieden werden, ob er nur den aktuellen Pfad gezeigt haben möchte, oder auch die anderen, die nicht durch Bewegungssimulation erreichbar sind.

4.2 Mehrfachlösungen bei der Berechnung starrer Körper

Allgemein lässt sich feststellen, dass eine Reihe von Ansätzen existiert, um aus einer Menge möglicher Lösungen eine vermutlich für den Nutzer plausible Lösung zu wählen. Es soll an dieser Stelle jedoch auch ausdrücklich darauf hingewiesen werden, dass unerwünschte Lösungen möglichst durch eine eindeutige Modellierung vermieden werden sollten. Dieser Mehraufwand bei der Modellierung wird sich durch eine erhöhte Robustheit gegenüber schnelleren Änderungen auszahlen. Zur Vermeidung von Mehrdeutigkeiten gibt es verschiedene Vorgehensweisen. Einige von ihnen sind die folgenden:

- Vermeidung der „Abstand von Punkt zu Punkt“-Constraints.
  Dies gilt insbesondere dann, wenn es sich quasi um einen horizontalen bzw. vertikalen Abstand in einem (lokalen) Koordinatensystem handelt. Hier ist ein vorzeichenbehafteter Con-

[17] Diese Möglichkeit wurde auch für die Erstellung der in dieser Arbeit beschriebenen Beispiele benutzt.

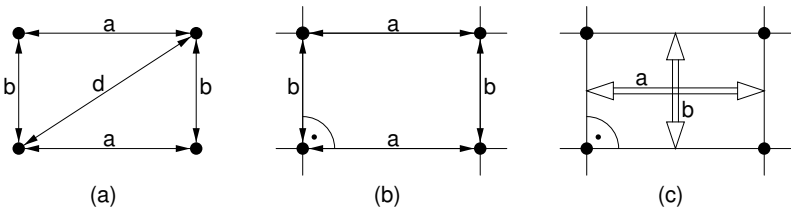


Abbildung 4.10: Varianten der Modellierung eines Rechtecks. (a) Es werden ausschließlich „Abstand von Punkt zu Punkt"-Constraints verwendet und ggf. ein Constraint zur Berechnung von d. *XHS*-Hinweise könnten Mehrfachlösungen verhindern. (b) Durch den Einsatz von gerichteten Geraden und eines Rechtwinkligkeits-Constraints wird d nicht mehr benötigt. Mehrfachlösungen sind noch möglich. (c) Zwei „paralleler Abstand"-Constraints verhindern Mehrfachlösungen.

straint wie der „Abstand Punkt zu Ebene"-Constraint wesentlich günstiger. Unter Umständen lohnt es sich Hilfsobjekte einzuführen.

Beispiel 4.3: Varianten der Modellierung eines Rechtecks
In *2D* sei ein Rechteck zu modellieren. Eine erste Variante könnte sich auf die vier Eckpunkte beziehen, die in rechteckiger Form zu versteifen sind. Die Realisierung einer solchen Versteifung ist in Abbildung 4.3(a) gezeigt. Als ungünstig erweist sich hierbei zum einen die notwendige Berechnung von d entsprechend a und b und zum anderen die Möglichkeit, dass z. B. für den oberen linken Punkt oder den unteren rechten Punkt eine andere als die dargestellte Lösung gewählt wird. Es werden Hinweise benötigt, um dies zu verhindern. In Abbildung 4.3(b) wurden vier Hilfsgeraden (und acht entsprechende „Punkt auf Gerade"-Constraints) in das Modell eingeführt. Der Rechtwinkligkeits-Constraint ersetzt den Abstands-Constraint für die Diagonale. Aber auch hier sind noch diverse unerwünschte Lösungen durch Hinweise auszuschließen. Erst das Modell in Abbildung 4.3(c) ist auch ohne Hinweise frei von Mehrfachlösungen, wenn sowohl im Winkel- als auch im „paralleler Abstand"-Constraint die Vorzeichen der Parameter beachtet werden.
Beim Einsatz eines Constraint-Solvers, der „Abstand von Punkt zu Gerade"-Constraints unterstützt, würde sich das Modell aus Abbildung 4.3(c) noch vereinfachen lassen. Zwei Geraden (z. B. die untere und die linke) könnten ein rechtwinkliges Koordinatensystem aufspannen, zu dem die entsprechenden „Abstand von Punkt zu Gerade"- bzw. „Punkt auf Gerade"-Constraints definiert werden. a und b würden so die Rolle lokaler Koordinaten übernehmen. □

- Explizites Fixieren der Hauptgeometrie
  Dies hilft zu vermeiden, dass die Suche nach der Form eines Körpers zu seiner Verschiebung bzw. Drehung im Raum führt. Hier bietet sich z. B. auch die Einführung eines (fixierten) Koordinatensystems an, auf das sich in entsprechenden „paralleler Abstand"-Constraints bezogen wird. Diese Vorgehensweise wurde u. a. auch für die in den Abbildungen 5.13 und 5.15 gezeigten Modelle genutzt.

- Nutzung von Hinweisen
  In Ficucs bieten sich verschiedene Formen an (vgl. Abschnitt 2.6):
  - die Einführung eines *XHS*-Hinweises[18] für relevante Dreiecke,
  - das Erzeugen einer konsistenten Überbestimmtheit durch zusätzliche Abstands- oder Winkel-Constraints und
  - die Nutzung von Ungleichungen.

Generell ist die Nutzung von Hinweisen als ungünstiger einzuschätzen, da die Gefahr besteht, dass sie nicht direkt in einem Konstruktor berücksichtigt werden können. In derartigen Fällen ist es

[18] Im Englischen auch als *chirality constraints* bezeichnet (siehe z. B. [Sch93] und [Hav97]), was mit „Händigkeits-Constraints" übersetzt werden kann (vgl. *RHS* und *LHS*).

möglich, dass sich die bisherige Konstruktion beim späteren Testen des Hinweises als falsch erweist und durch den Constraint-Solver oder die Applikation muss dann eine Variation der Konstruktionsannahmen vorgenommen werden, typischerweise eine Änderung vorheriger Lösungsauswahlen. Ein Algorithmus, der dazu dient hier allgemein die richtige Wahl zu treffen, ist jedoch nicht trivial und kann (falls eine solche Berechnungsstrategie in einem Constraint-Solver oder einer Applikation implementiert ist) u. U. erheblichen Berechnungsaufwand verursachen. Zudem ist es möglich, dass die gezielte Erzeugung von Überbestimmtheiten Probleme in der Freiheitsgradanalyse verursacht.

4.2.1 Erläuterungen zum gewählten Beispielmodell

Bei den Betrachtungen zu Mehrfachlösungen an starren Körpern soll die schon in den Abschnitten 2.7 und 3.6.5 verwendete Menge aus sechs Punkten als Beispiel dienen. Auch andere Autoren haben Berechnungen für diese Menge beschrieben, so dass im folgenden Text entsprechende Vergleiche gezogen werden können.

Grundsätzlich lässt sich die Punktmenge auf unterschiedliche Art und Weise zu einem starren Körper versteifen. In Abbildung 4.11 sind vier dieser Möglichkeiten gezeigt und mit Schema I bis IV benannt.

Im Allgemeinen können zwischen n Punkten a verschiedene Abstände definiert werden, mit

$$a = \frac{n \cdot (n-1)}{2}$$

Um n Punkte in *2D* (wo jeder Punkt zwei Freiheitsgrade besitzt) vollbestimmt zu versteifen, so dass nur noch die drei Freiheitsgrade eines starren Körpers in *2D* verbleiben, sind c Abstands-Constraints (jeder hat eine Valenz) nötig, mit

$$c = 2 \cdot n - 3$$

Das heißt zwischen den sechs Punkten stehen fünfzehn verschiedene Abstände für die Versteifung zur Auswahl von denen neun auszuwählen sind. Die Auswahl der neun Abstände[19] entscheidet mit darüber, wie viele Mehrfachlösungen sich ergeben können. Im Allgemeinen sind bei direkten Konstruktionen[20] für eine Menge aus n Punkten bis zu m Mehrfachlösungen möglich, mit

$$m = 2^{n-2}$$

Die Anzahl der reellen Lösungen (und nur solche sind hier von Interesse) ist kleiner als diese Maximalanzahl, wenn durch eine bestimmte Auswahl der Abstände sowie der entsprechenden Abstandsparameter degenerierte Fälle auftreten, z. B. falls Dreiecksungleichungen nicht erfüllt sind[21].

In Schema I wird angenommen, dass zwei Punkte (hier p_1 und p_2) bekannt sind, wobei der zwischen ihnen definierte Abstand eingehalten ist. Von diesen beiden Punkten existieren jeweils zwei Abstands-Constraints zu den übrigen vier Punkten. In einem solchen Schema sind alle $2^{6-2} = 16$

[19] Theoretisch lassen sich aus den 15 Abständen 5005 Kombinationen bilden. Hierunter sind aber auch solche, die lokal über- bzw. unterbestimmt sind. Außer der eigentlichen Auswahl ist für die Berechnung (und damit das Auftreten von Mehrfachlösungen) auch die Verteilung der Freiheitsgrade auf die Punkte von Bedeutung. Es ist z. B. möglich, dass die drei Freiheitsgrade des starren Körpers auf drei Punkte aufgeteilt sind.

[20] Vergleiche Abbildung 4.11. Hier sind die Schemen I und II direkt konstruierbar, III und IV hingegen nicht. Aber auch für das Schema III sind keine Punktmengen mit mehr als 16 Lösungen bekannt.

[21] z. B. ergeben sich in Abbildung 4.11 im Schema I bei bekannten Punkten p_1 und p_2 für die Berechnung von p_6 derartige degenerierte Fälle, wenn $abstand(p_6, p_1) + abstand(p_1, p_2) \geq abstand(p_6, p_2)$ ist. Bei Gleichheit existiert genau eine Lösung, ansonsten keine Lösung.

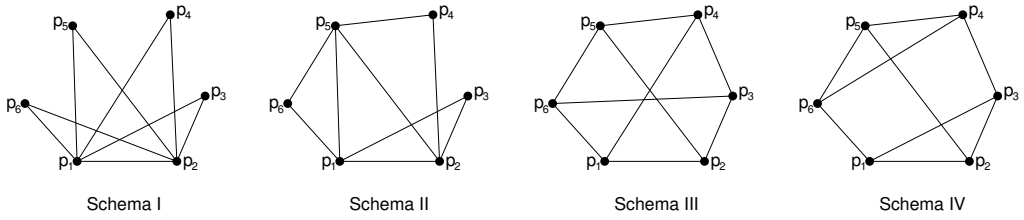


Abbildung 4.11: Vier Schemen, wie sechs Punkte in *2D* durch Abstände zu einem starren Körper versteift werden können.

Mehrfachlösungen möglich, denn die vier Punkte p_3 bis p_6 lassen sich jeweils unabhängig voneinander oberhalb oder unterhalb der Punkte p_1 und p_2 konstruieren. In Schema II ist diese Unabhängigkeit zwar nicht mehr gegeben, so dass die Reihenfolge der direkten Konstruktionen bzw. Berechnungen für die Punkte p_3 bis p_6 nicht mehr frei wählbar ist, trotzdem lassen sich auch hier grundsätzlich wieder sechzehn verschiedene Lösungen generieren. Zu beachten ist jedoch, dass die Lösungsmengen für Schema I und II im Allgemeinen nur zwei gemeinsame Lösungen besitzen, die Ausganspunktmenge (z. B. die in Abbildung 4.11 gezeigte) sowie die gespiegelte Lösung.

In den Schemen III und IV wurden durch die Wahl der Abstände zyklische Abhängigkeiten erzeugt. In derartigen Schemen finden sich typischerweise weniger Mehrfachlösungen. Das Beispiel in Abschnitt 3.6.5 hat z. B. nur vier Lösungen (die zwei in Abbildung 3.33 gezeigten Lösungen, sowie die beiden gespiegelten Lösungen in Abbildung 3.34).

Die für das Beispielmodell im nachfolgenden verwendeten Punktkoordinaten sind:

$$\begin{array}{rrr} p1 & (0.20, & 0.00) \\ p2 & (0.70, & 0.00) \\ p3 & (0.13, & -0.08) \\ p4 & (0.92, & 0.31) \\ p5 & (0.30, & 1.18) \\ p6 & (-1.11, & -0.19) \end{array}$$

Die Koordinaten wurden so gewählt, dass sich auch nach Schema III sechzehn verschiedene Lösungen ergeben. In den Betrachtungen wird davon ausgegangen, dass die Position von Punkt p_1 bekannt ist (die zwei Positionsfreiheitsgrade des starren Körpers sind aufgebraucht) und dass p_2 in beliebiger Richtung um p_1 positioniert werden kann (p_2 braucht so den Richtungsfreiheitsgrad des starren Körpers auf). In den folgenden Abschnitten wurde p_2 stets waagerecht zu p_1 gesetzt.

Dies ist jedoch nur eine der Möglichkeiten, wie die Freiheitsgrade verteilt sein können. Andere interessante Fälle, die hier jedoch nicht tiefergehend betrachtet werden können, sind z. B. solche, wo die drei Freiheitsgrade auf drei Punkte verteilt sind oder wo zwischen dem Punkt mit zwei Freiheitsgraden und dem mit einem Freiheitsgrad kein Abstand bekannt ist.

4.2.2 Mehrfachlösungen in der konstruktiven Berechnung

In diesem Abschnitt soll der Unterschied zwischen der Suche von Mehrfachlösungen in:

- nicht iterativen Plänen (nur direkte Konstruktionen) und
- iterativen Plänen (direkte und iterative Konstruktionen)

verdeutlicht werden, wobei der Fokus auf der schwierigeren Suche in iterativen Plänen liegt. Bezüglich Ficucs wird insbesondere gezeigt, wie alle Lösungen systematisch gefunden werden können. Im Kontext kontinuierlicher Veränderungen ist jedoch auch das schnelle Finden einer plausiblen nächsten Lösung wichtig. Die hierfür in Ficucs benutzten Ansätze wurden bereits in Abschnitt 4.1.2 erörtert.

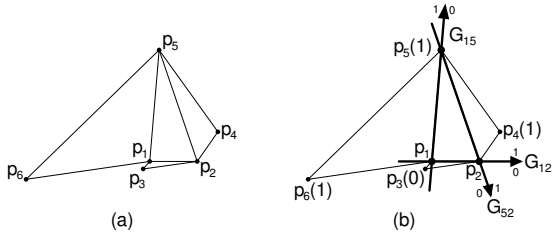


Abbildung 4.12: Die neun versteifenden Abstände zwischen den sechs Punkten wurden nach Schema II gewählt. Die direkte Konstruktion der Punkte ist möglich. In (b) sind die Kriteriengeraden G_{12}, G_{15} und G_{52} eingetragen. An der Pfeilspitze dieser gerichteten Geraden ist die Kodierung für die Konstruktion auf der jeweiligen Seite eingetragen. Beispielsweise wird p_6 als Schnitt zweier Kreise um p_1 und p_5 konstruiert. G_{15} ist somit die Kriteriengerade. Da p_6 links von G_{15} liegt, wird die aktuelle Lösung mit 1 kodiert.

Direkte Konstruktion

Abbildung 4.12 zeigt die Auswahl der Abstände nach Schema II, das die direkte Konstruktion der Punkte ermöglicht[22]. Unter der Annahme, dass die drei Freiheitsgrade des starren Körpers p_1 und p_2 zugeordnet wurden, könnte sich folgender Plan ergeben:

1. Konstruiere p_2 im Abstand zu p_1 und nahe der alten Lösung, also hier auf Höhe von p_1.
2. Konstruiere p_3 als Schnittpunkt zweier Kreise um p_1 und p_2.
3. Konstruiere p_5 als Schnittpunkt zweier Kreise um p_1 und p_2.
4. Konstruiere p_4 als Schnittpunkt zweier Kreise um p_5 und p_2.
5. Konstruiere p_6 als Schnittpunkt zweier Kreise um p_1 und p_5.

Für den 2. bis 5. Konstruktionsschritt (vier Konstruktionen) ergeben sich jeweils zwei mögliche Lösungen, so dass insgesamt $2^4 = 16$ verschiedene Lösungen gefunden werden. Zur Unterscheidung der Lösungen kann jeweils die Gerade durch die beiden Punkte dienen, auf denen die jeweilige Konstruktion beruht (d. h. die Mittelpunkte der beiden Kreise werden genutzt, siehe Abschnitt 3.5.2). Die Kodierung der einzelnen Lösungen kann in Form einer Binärzahl erfolgen, z. B. 1110_b für $p_6(1)$, $p_5(1)$, $p_4(1)$ und $p_3(0)$, wobei eine 1 für die Lösungen steht, die links von der gerichteten Kriteriumsgeraden liegen und eine 0 für alle anderen Lösungen. Auf diese Weise kodiert 1110_b die in der Abbildung 4.12 dargestellte Lösung. Zählt man die Binärzahl von 0 beginnend hoch und interpretiert die entsprechenden Binärstellen wieder als Hinweis auf die vorzunehmende Lösungsauswahl, so kann man nacheinander alle sechzehn Lösungen berechnen[23].

[Sch93] zeigt eine Triangulationsmethode (eigentlich Trilateration[24]), die aus der Suche nach einer Konstruktionssequenz (dort als *Henneberg 2-sequence* bezeichnet, vgl. [Hen11]) und einem

[22] Sollte aber in einer interaktiven Anwendung beispielsweise $p6$ bewegt werden (der Mauszeiger bestimmt die zwei Positionsfreiheitsgrade) und der Rotationsfreiheitsgrad des starren Körpers $p4$ zugeordnet sein, dann muss auch bei diesem Modell iterativ konstruiert werden. Zur direkten Konstruktion ist die Kenntnis des Abstandes zwischen p_6 und p_4 nötig. Allerdings ist auch die Möglichkeit der Umverteilung der Freiheitsgrade in Betracht zu ziehen (Abschnitt 3.4.4).

[23] Eine derartige Kodierung bietet sich an, wenn zwei, vier oder acht Lösungen pro Konstruktor möglich sind, bei vier und acht Lösungen werden dann zwei bzw. drei Bit pro Konstruktor verwendet. Allgemein ist die Kodierung jedoch komplizierter, denn wie in Abbildung 4.14 ersichtlich ist, kann z. B. die Anzahl der auszuwählenden Nullstellen einer bestimmten Fehlerfunktion nahezu beliebig sein.

[24] Bei der Berechnung von Punktpositionen in Dreiecksnetzen wird in einigen Quellen (z. B. [GKN81] oder [Neu95]) streng zwischen der Trilateration (von der Punktmenge sind einzelne Abstände bekannt) und der Triangulation (von der Punktmenge sind ein Abstand und sonst nur Winkel bekannt) unterschieden. In der Landesvermessung erlangte die Trilateration erst im 20. Jahrhundert Bedeutung, als genaue Entfernungsmessungen über große Strecken möglich wurden (z. B. mittels Laser).

einfachen *Backtracking*-Algorithmus besteht, um alle Lösungen zu finden. Hierbei werden auch Überbestimmtheiten berücksichtigt, die die Zahl der Lösungen einschränken.

In [Fud95] werden durch Fudos allgemeine Überlegungen für die Lösungsauswahl bei direkten Konstruktionen (nach einem Clustering) erörtert. Hierbei geht er insbesondere auf folgende Möglichkeiten ein:

- Berücksichtigung globaler topologischer Kriterien (geschlossene Konturen, Konvexität, Vermeidung von Selbstschneidungen),
- Hinzufügen von Constraints,
- Nähe zur alten Lösung beim Bewegen von Elementen,
- Berücksichtigung der Vorzeichen in bestimmten Abstands- und Winkel-Constraints und
- dialogbasierte Interaktion mit dem Nutzer.

Ähnliche Ideen sind auch in [BFH+93] Gegenstand der Diskussion.

Iterative Konstruktion

Auch bei iterativen Konstruktionen lassen sich für die gewählte Punktmenge sechzehn verschiedene Lösungen finden. Abbildung 4.13 zeigt die Punktmenge mit den nach Schema III gewählten Abständen, die stets eine iterative Konstruktion erzwingen (vgl. auch Abschnitt 2.7). Unter der Annahme, dass die drei Freiheitsgrade des starren Körpers p_1 und p_2 zugeordnet wurden, könnte sich folgender Plan ergeben:

1. Konstruiere p_2 im Abstand zu p_1 und nahe der alten Lösung, also hier auf Höhe von p_1.
2. Konstruiere p_3 testweise auf einem Kreis um p_2.
3. Konstruiere p_4 als Schnittpunkt zweier Kreise um p_1 und p_3 (allg. zwei Lösungen).
4. Konstruiere p_6 als Schnittpunkt zweier Kreise um p_3 und p_1 (allg. zwei Lösungen).
5. Konstruiere p_5 als Schnittpunkt zweier Kreise um p_6 und p_4 (allg. zwei Lösungen).
6. Messe den Abstand zwischen p_2 und p_5. Stoppe, wenn er korrekt ist. Ansonsten gehe zum Schritt 2, um eine andere Lösung zu finden (allg. mehrere Lösungen).

Für den 3. bis 5. Konstruktionsschritt (drei Konstruktionen) ergeben sich jeweils zwei mögliche Lösungen, so dass insgesamt $2^3 = 8$ verschiedene Fehlerfunktionen für den iterativen Teilplan (Schritte 2. bis 6.) existieren. In Abbildung 4.14 sind die Fehlerfunktionen und die entsprechenden Lösungen aufgeführt. Insgesamt wurden sechzehn Lösungen gefunden. Wie in der Abbildung zu sehen ist, können aber auch mehr als zwei Lösungen pro Fehlerfunktion auftreten. Bei ungünstiger Wahl der Abstandsparameter (bzw. bei Wahl anderer Punktpositionen, denn aus ihnen wurden die Abstandsparameter bestimmt) kommt es jedoch leicht zu weniger als sechzehn Lösungen. Am deutlichsten wird das bei der dritten und sechsten Fehlerfunktion in Abbildung 4.14. Hier ist die einzige Lösung sehr nah an einem Bereich, in dem keine Lösung mehr gefunden werden kann. Schon kleine Änderungen der Abstandsparameterwerte können dazu führen, dass die Fehlerfunktion keine Nullstelle mehr ausweist.

Wie bereits im Abschnitt 3.6.5 verdeutlicht wurde, ist eine eindeutige Zuordnung der Nullstellen von zwei Fehlerfunktionen (z. B. der letzten und der aktuellen) nur schwer möglich, insbesondere wenn sich in diesen Fehlerfunktionen mehrere Nullstellen dicht nebeneinander befinden oder eine

signifikante Änderung des Funktionsverlaufes vorliegt[25]. Auch für kontinuierliche Änderungen an starren Körpern sind deshalb Konzepte wie die Berücksichtigung von Vertrauensmaßen und das Umschalten zwischen Fehlerfunktionen von großer Bedeutung (vgl. Seite 105). Im Kontext der Berechnung starrer Körper steht jedoch typischerweise das systematische Auffinden aller Lösungen im Vordergrund. Hierfür bietet Ficucs eine Schnittstelle an, die folgende Methoden enthält[26]:

- Setze in allen Konstruktoren des aktuellen Planes, in denen Mehrfachlösungen auftreten können, den Index für die Auswahl zurück auf null. Das Zurücksetzen ist unabhängig davon, ob die Konstruktoren direkt oder iterativ arbeiten.
- Gehe über zur nächsten möglichen Auswahl bezüglich des Gesamtplanes. Hierfür wird beginnend vom Ende[27] des Planes nach einem Eintrag gesucht, in dem der Auswahlindex erhöht werden kann. Alle bis dahin überprüften (jedoch nicht erhöhbaren) Indizes werden auf null zurückgesetzt. Der Auswahlindex von Konstruktoren, die zwischen dem Anfang des Planes und dem erhöhten liegen, wird nicht beeinflusst. Bei Auftreten eines iterativen Konstruktors wird zunächst der Index in diesem Konstruktor erhöht. Ist dies nicht mehr möglich heißt das, dass keine weiteren Nullstellen in der aktuellen Fehlerfunktion gefunden werden können. Erst dann erfolgt das Hochzählen der Auswahlindizes der Konstruktionen innerhalb der iterativen Konstruktion, wodurch es zu einem Wechsel der Fehlerfunktion kommt.
- Setze den Auswahlindex für einen bestimmten Planeintrag auf einen gewünschten Wert. Während für die Nutzung der oberen beiden Methoden kein Wissen über das Modell bzw. den Konstruktionsplan nötig ist, müssen hier gezielt die Modellelemente bzw. die Planeinträge referenziert werden. Letzteres steht im engen Zusammenhang mit der Möglichkeit, von Ficucs erstellte Pläne zu exportieren oder extern erstellte/gespeicherte/modifizierte Pläne an Ficucs zu übergeben (siehe Abschnitt 3.4.3).

Die Zuordnung zwischen Index und zu wählender Lösung ist bei direkten Konstruktionen zumeist leicht möglich, für das Schneiden zweier Kreise z. B. entsprechend Abbildung 4.12. Um in einer iterativen Konstruktion den aktuellen Index einer bestimmten Nullstelle zuzuordnen, können verschiedene Strategien verwendet werden. In Abbildung 4.14 wurde die Reihenfolge des Auffindens gewählt. Die erste gefundene Nullstelle entspricht dem Index null, die zweite dem Index eins usw. Es wäre jedoch auch denkbar, die Sortierung entsprechend der festgestellten Lage der Nullstellen vorzunehmen. Dann könnte die, die in der Fehlerfunktion am weitesten links liegt, den Index null erhalten, die Nullstelle rechts von ihr den Index eins usw. Allerdings müsste hierzu zuerst die Gesamtheit der Nullstellen der aktuellen Fehlerfunktion gefunden werden.

Berechnungen zu den Modellen der Schemen III und IV sind auch Gegenstand der Arbeiten anderer Autoren. Borcea und Streinu liefern ein Beispiel für ein nach Schema IV versteiftes Modell, das sogar vierundzwanzig Lösungen aufweist. Die Ausführungen in [BS02] lassen vermuten, dass die Lösungen mit Hilfe von Cinderella (Software für dynamische Geometrie, siehe [RGK00]) gefunden wurden. Die iterativ-konstruktive Lösung des Beispiels mit Ficucs ist im Anhang auf Seite 166 erläutert.

Ficucs resultierte aus der Fortsetzung der Arbeiten von Hsu. In [Hsu96, Seite 22 ff.] und [HB97] zeigt Hsu, wie die iterativ-konstruktive Lösung eines Modells nach Schema IV mit der Vorgängerversion von Ficucs realisierbar war. Zur Nullstellenfindung wurde das Sekantenverfahren (siehe

[25] Zum Beispiel können die starke Verschiebung einer Nullstelle und die Änderung der Anzahl der Maxima oder der nicht definierten Bereiche signifikant sein. Unter Umständen reichen hierfür bereits kleine Änderungen am Interaktionspunkt aus.

[26] Diese Schnittstelle ermöglicht die direkte Steuerung der Lösungsauswahl im Plan. Ähnlich direkt können Hinweise wirken, wenn der Plan so gestaltet ist, dass sie sich direkt in einem Konstruktor berücksichtigen lassen. Die Nutzung von allgemeinen Auswahlkriterien wie die Nähe zur alten Lösung oder zu einer extrapolierten Lösung ist hingegen eine indirekte Steuerung der Lösungsauswahl.

[27] Wenn die Änderungen vornehmlich am Ende des Planes vorgenommen werden, ergibt sich die Möglichkeit Rechenzeit einzusparen, denn die Werte der Objekte am Anfang des Planes werden durch die Wahl einer anderen Lösung am Ende in der Regel nicht beeinflusst. Dies gilt jedoch nicht, wenn die entsprechenden Operatoren Bestandteil einer iterativen Konstruktion sind (Rückkopplung!).

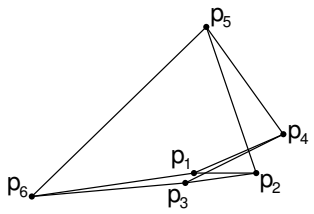


Abbildung 4.13: Die neun Abstände wurden nach Schema III gewählt (vgl. Abbildung 4.11), so dass iterativ konstruiert werden muss. Die aktuell ausgewählten Lösungen sind $p_6(1)$, $p_5(1)$ und $p_4(1)$, also stets links von der jeweiligen Kriteriumsgeraden. Hierbei ist zu beachten, dass entsprechend dem Plan auf Seite 113 die Geraden G_{13} und G_{31} zu unterscheiden sind. Die Wahl der Nullstelle in der iterativen Konstruktion erfolgt entsprechend den Ausführungen auf Seite 114.

Seite 80) eingesetzt. Nicht definierte Bereiche der Nullstellenfunktion sollten mit einer zusätzlichen Fehlerfunktion verlassen werden (siehe Seite 82). Eine gezielte Wahl der Nullstellen wurde durch Hsu als sehr wichtig eingeschätzt. Mehrfachlösungen direkter Konstruktionen wurden analog zu den bereits auf Seite 113 erörterten Ansätzen aus [Fud95] gewählt. Für die Beibehaltung der letzten Wahl aus den möglichen Fehlerfunktionsnullstellen schlägt Hsu kleine Schrittweiten sowie die Verwendung der letzten Lösung als Startwert der neuen Suche vor. Schema III wurde in [Hsu96, Seite 33 ff.] in einem Beispiel für die iterativ-konstruktive multivariate Suche benutzt.

Beschreibungen, wie andere iterativ-konstruktive Verfahren speziell zur Lösung der Beispiele nach Schema III und IV eingesetzt werden, sind nicht bekannt. In Arbeiten zum *locus intersection* Verfahren ([HGY02] und [GHY04]) beschreiben Hoffmann und seine Coautoren wie dieses Verfahren genutzt werden kann, um alle Lösungen für vollbestimmte Modelle zu finden, die aus bis zu sechs *3D*-Punkten, Ebenen und *3D*-Geraden bestehen. Wie in diesem Ansatz die Nullstellensuche umgesetzt ist, wird nicht näher erörtert. Da die Lösungsfindung nur einer grundsätzlichen Problemanalyse dient (Ermittlung der Nullstellenanzahl für ein bestimmtes Modell), besteht nicht die Notwendigkeit einer effektiven Suche. Vermutlich ist deshalb eine erschöpfende Suche implementiert worden, die relativ langsam den Suchraum in konstanter Schrittweite absucht. Zudem lassen die dargestellten Fehlerfunktionen vermuten, dass die genutzte Implementierung Probleme mit der Zuordnung der berechneten Fehlerwerte zu bestimmten Fehlerfunktionen hat. Dies scheint von den Autoren jedoch auch nicht angestrebt zu werden.

In [Ber96] erörtert Berling eine *strukturbasierte Methode,* die bei zyklischen Abhängigkeiten als Alternative zur multivariaten Lösung des entsprechenden Gleichungssystems (nach Newton-Raphson bzw. durch direkte Linearisierung, vgl. Anhang B.3) eingesetzt werden kann. Sie hat den Vorteil, dass hierbei nur eine kleinere Variablenmenge iteriert werden muss. Die für die Iteration relevanten Variablen entsprechen der sogenannten *Feedback-Vertex-Menge*[28], die möglichst klein gewählt wird. Auf diese Art lässt sich die Anzahl der Variablen im Allgemeinen nicht so wie beim in Abschnitt 3.6 beschriebenen Verfahren reduzieren. Jedoch kann für jede Variable, die nicht iteriert werden muss, eine Konstruktion ausgeführt werden, bei der die Lösungsauswahl nach einem jeweils geeigneten Kriterium möglich ist. Dadurch verbessert sich die Steuerbarkeit der Lösungsauswahl gegenüber der nur von Startwert und Schrittweite abhängigen Lösung des kompletten Gleichungssystems.

[28] Die Untermenge F der Knoten V eines zyklischen Graphen $G(V, E)$ ist nach [Ber96] genau dann eine *Feedback-Vertex-Menge*, wenn der resultierende Graph $G'(V \setminus F, E')$ keine Zyklen mehr enthält (E' enthält keine Kanten, die F referenzieren). Das in einer Implementierung zu lösende Problem liegt in der effizienten Bestimmung einer möglichst kleinen *Feedback-Vertex-Menge*.

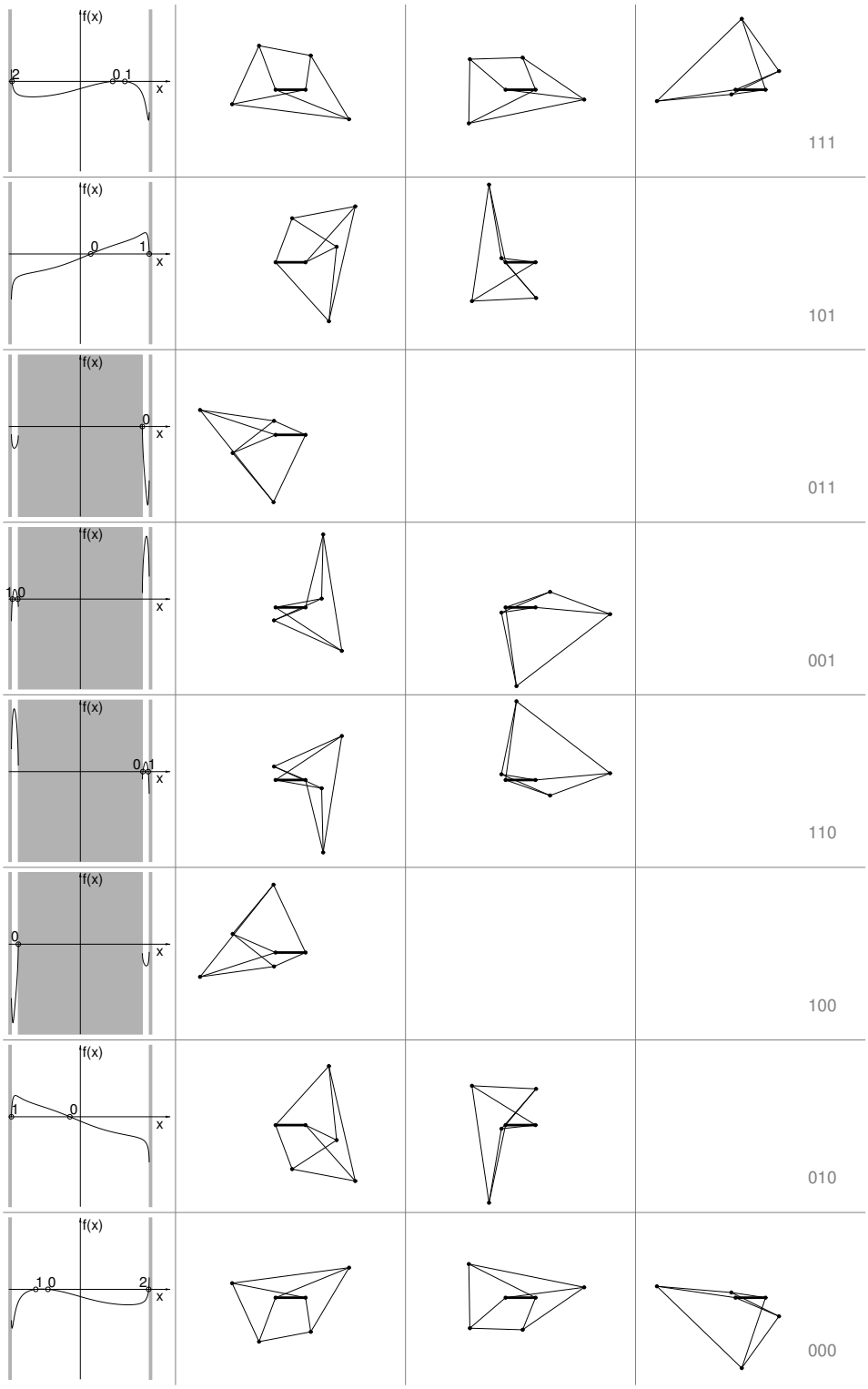


Abbildung 4.14: Sechzehn Lösungen für ein Modell aus sechs Punkten und neun Abständen (Schema III). Die Lösungen mussten iterativ berechnet werden. Jeweils zwei Lösungen sind zueinander spiegelsymmetrisch. In jeder Spalte ist rechts angegeben, welche Lösung in den direkten Konstruktoren für p_6, p_5 und p_4 gewählt wurde (siehe Plan auf Seite 113). Bei der Kodierung wurden die Stellen in der Reihenfolge p_6, p_5 und p_4 belegt, die erste Ziffer steht also für die aktuelle Lösung von p_6.

4.2.3 Vergleich mit anderen Ansätzen

Interessanterweise werden in der Literatur über nicht konstruktiv-iterative Ansätze zur Berechnung constraint-basierter Modelle immer wieder Modelle der Schemen III und IV als Beispiel herangezogen. An dieser Stelle soll nun ein Vergleich mit den dort beschriebenen Verfahren vorgenommen werden.

Algebraische Umformung

Grundsätzlich werden in allen Ansätzen Umformungen bzw. Transformationen vorgenommen, u. a. bei einer Normalisierung der Constraint-Menge oder bei der Aufstellung der Jakobimatrix. Derartige Transformationen lassen sich jeweils als mehr oder weniger komplexe algebraische Umformungen interpretieren.

Ein Ansatz, der explizit auf sehr umfangreichen algebraischen Umformungen basiert, ist in [Sch93] beschrieben. Dort wird das nach Schema IV (siehe Abbildung 4.11) versteifte Modell analysiert und gezeigt, dass sich aus der constraint-basierten Beschreibung ein Polynom zwölften Grades herleiten lässt. Dementsprechend könnten sich bis zu zwölf Lösungen ergeben. Da für die Herleitung in [Sch93] ein Dreieck als fix angenommen wurde, muss für den allgemeinen Fall die Zahl der möglichen Lösungen noch auf vierundzwanzig verdoppelt werden, denn durch die Fixierung wurden die gespiegelten Lösungen nicht berücksichtigt[29]. Für konkrete Modelle könnte das Polynom mit numerischen Verfahren gelöst werden. Hierbei sind auch insbesondere der Sturmsche Satz, die Descartsche Zeichenregel oder der Satz von Rolle anwendbar (vgl. z. B. [QN96], [TS88] und [Sch97]). Der Aufwand für algebraische Umformungen ist jedoch beträchtlich, so dass der Ansatz nur in einer Vorverarbeitungsphase praktikabel ist. Denkbar wäre z. B. die Analyse häufig auftretender Modelle und die Speicherung einer Berechnungsvorschrift für die jeweiligen Polynomkoeffizienten. Zur Laufzeit könnten die Koeffizienten aus den Modellparametern berechnet werden und anschließend die Bestimmung der Nullstellen der Polynome erfolgen. Ob sich aus einem derartigen Ansatz gegenüber dem in Abschnitt 3.6 vorgestellten Ansatz Vorteile ergeben ist jedoch ungewiss.

Numerische Ansätze

Im vorherigen Abschnitt wurde noch einmal sehr deutlich, wie wichtig im Allgemeinen numerische Berechnungen für die Lösungsfindung sind, selbst wenn vorher algebraisch umgeformt wurde. Das Gleiche gilt auch für Clustering-Verfahren (z. B. [HLS01a, HLS01b]) die für nicht direkt konstruierbare Modelle auf numerische Gleichungslöser zurückgreifen.

Ein zentrales Problem der numerischen Verfahren ist die Wahl der Startwerte, insbesondere dann, wenn die Suche nicht auf einem explizit gegebenen Polynom basiert[30] und alle möglichen Lösungen gefunden werden sollen. Hierzu lassen sich in der Literatur verschiedene Aussagen finden, z. B.:

- Kramer wies in [Kra90, Kra92] auf das chaotische Verhalten von numerisch-iterativen Ansätzen hin, wobei er das Newton-Raphson-Verfahren als Beispiel heranzog.
- [QN96]: Bei transzendenten Funktionen (insbesondere solche mit Winkelfunktionen) ist das Finden eines geeigneten Startwertes schwierig (der Autor sagt sinngemäß: „man muss Glück haben, um überhaupt einen Startpunkt zu finden, bei dem der Algorithmus konvergiert“).
- [LM96]: „Homotopie konvergiert viel öfter als Newton-Raphson“ und „Homotopie kann auch zu einer ungewollten Lösung konvergieren“.

[29] Wenn basierend auf einem Dreieck ABC zwölf Lösungen gefunden werden, dann können basierend auf dem zu ihm spiegelsymmetrischen Dreieck ABC' auch zwölf Lösungen gefunden werden. Jede von ihnen ist spiegelsymmetrisch zu einer der ersten zwölf Lösungen.

[30] Im Allgemeinen ist die Fehlerfunktion (bzw. die Fehlerfunktionen) weder explizit gegeben noch ein Polynom.

Diese Abhängigkeit des Sucherfolges (bzgl. der Divergenz oder der Konvergenz zu einer bestimmten Lösung) vom Startwert der Suche lässt sich sehr einfach visualisieren. Hierzu werden zwei Parameter des Startwertevektors variiert und zu jedem Parameterpaar das Resultat der Suche (die Nummer der Lösung) ermittelt. Wenn die beiden Parameter als x- bzw. y-Koordinate und die Nummer der Lösung als Farbwert interpretiert werden, ergeben sich Bilder, die zeigen, wie sehr ein bestimmtes Verfahren für ein bestimmtes Modell von den jeweiligen Startwerten abhängig ist. Auf diese Weise wird z. B. in [Var01] ein Überblick über das Verhalten verschiedenster Verfahren bei der Suche nach den drei Lösungen der Gleichung $z^3 - 1 = 0$ gegeben, wobei die resultierenden Bilder sehr gut den fraktalen Charakter der Suche widerspiegeln.

Im folgenden Text soll anhand entsprechender Suchraumanalysen gezeigt werden, wie problematisch die Suche nach Lösungen constraint-basierter Modelle mit Hilfe numerischer Methoden sein kann. Hierbei werden folgende Verfahren betrachtet:

- Newton-Raphson und
- Homotopie (die wiederum das Newton-Raphson-Verfahren mehrmals benutzt).

Der Vergleich ist jedoch nur grundsätzlicher Natur. Speziell das testweise implementierte Homotopie-Verfahren könnte durch den Einsatz eines Pfadverfolgungsansatzes noch bessere Ergebnisse liefern. Weitere Erläuterungen zu den beiden Verfahren befinden sich im Anhang B.

Für ein sehr einfaches Modell wie in Abbildung 4.15(a) ist es möglich, das fraktale Verhalten der newton-raphson-basierten Lösungsfindung (z. B. Abbildung 4.15(c) und (d)) durch den Einsatz von Homotopie zu vermeiden. In Abbildung 4.16(a) ist hierzu die Aufteilung des Suchraumes bei der Suche mit einem einfachen Homotopie-Verfahren dargestellt. Der schmale weiße Bereich in der Mitte verdeutlicht, dass auch dieser Ansatz divergiert, wenn der Startpunkt C_0 auf der Geraden durch A und B (d. h. auf der Grenze zwischen den Bereichen die zu verschiedenen Lösungen führen) liegt[31]. Wenn die Suche weit von der Lösung entfernt startet und die Anzahl der Zwischenschritte zu klein ist, dann stößt allerdings auch das genutzte Homotopie-Verfahren an seine Grenzen, siehe Abbildung 4.16(b). Abbildung 4.16(c) zeigt, dass nach dem iterativ-konstruktiven-Ansatz des Constraint-Solvers stets die gleiche Lösung gefunden werden kann[32], insbesondere wenn (wie hier) die direkte Konstruktion genügt[33].

Bei dem sehr einfachen Modell aus Abbildung 4.15 bzw. 4.16 konnte der Homotopie-Ansatz eine zielgerichtete Suche sehr gut unterstützen. Bei komplexeren Modellen ist es jedoch möglich, dass auch der verwendete Homotopie-Ansatz versagt. Ein Beispiel hierfür wird in Abbildung 4.18 gezeigt. Zwar kann durch den Homotopie-Ansatz erreicht werden, dass in größeren Bereichen die gleiche Lösung gefunden wird[34], aber es bleiben auch Bereiche, in denen schon kleine Startwertänderungen zu verschiedensten Lösungen führen oder die Suche nicht konvergiert.

Der im vorgestellten Constraint-Solver implementierte Ansatz führt bei der Berechnung des Modells zu einem wesentlich besseren Konvergenzverhalten:

1. Es wird stets eine Lösung gefunden. In Spezialfällen ist ggf. die erlaubte Anzahl der Abtastungen von Fehlerfunktionen zu erhöhen. Im obigen Beispiel waren stets 100 Abtastungen erlaubt.

---

[31] In einem solchen Fall schlägt [LM96] vor, den Startpunkt etwas zu verschieben und die Suche neu zu starten. Wie im nachfolgenden Text gezeigt wird, kann es aber auch Bereiche geben, wo Homotopie-Verfahren scheitern. Dann nützt auch eine derartige leichte Startpunktverschiebung nicht.

[32] Es wurde stets Lösung C_2 konstruiert, da C_2 die erste gefundene Lösung war (die erste zur Berechnung des Bildes verwendete Startposition für C_0 lag unten links) und dann diese Konfiguration („konstruiere auf der rechten Seite der Gerade von A nach B“) im Plan gespeichert wurde. Es wäre jedoch auch möglich, die Lösung stets dem aktuellen Startpunkt C_0 anzupassen, so dass eine Suchraumeinteilung wie in Abbildung 4.16(a) vorgenommen wird. Allerdings wird beim konstruktiven Ansatz stets eine Lösung gefunden.

[33] Zur Berechnung waren nur zwei Kreise zu schneiden.

[34] Vergleiche Abbildung 4.17, die die entsprechenden Resultate des Newton-Raphson-Verfahrens zeigt.

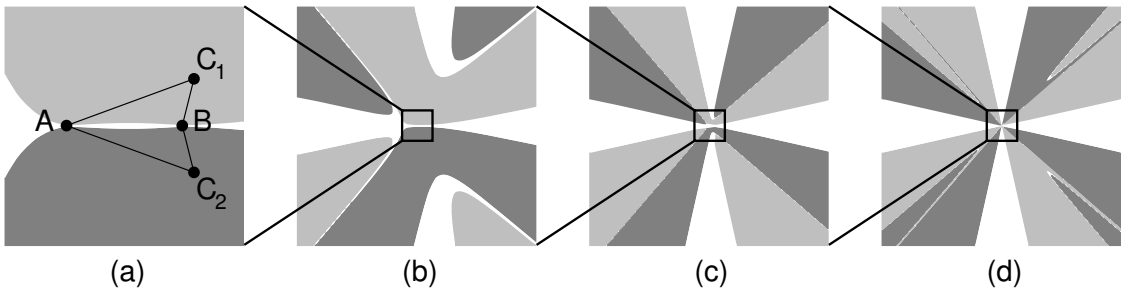


Abbildung 4.15: Startpunktabhängige Lösungsfindung bei der Berechnung eines Punktes C entsprechend zwei Abständen zu den Punkten A und B. Zur Berechnung wurde das Newton-Raphson-Verfahren angewendet. Lag der Startpunkt C_0 in einem weißen Bereich, so divergierte das Verfahren. Im hellgrauen Bereich wurde die Lösung C_1 und im dunkelgrauen Bereich die Lösung C_2 gefunden. In (a) ist der Suchraum in der Nähe des Modells gezeigt. (b), (c) und (d) zeigen einen jeweils um den Faktor acht größeren Suchraum. Die Linien zwischen den Teilbildern verdeutlichen, wo (a) in (b) usw. zu finden ist.

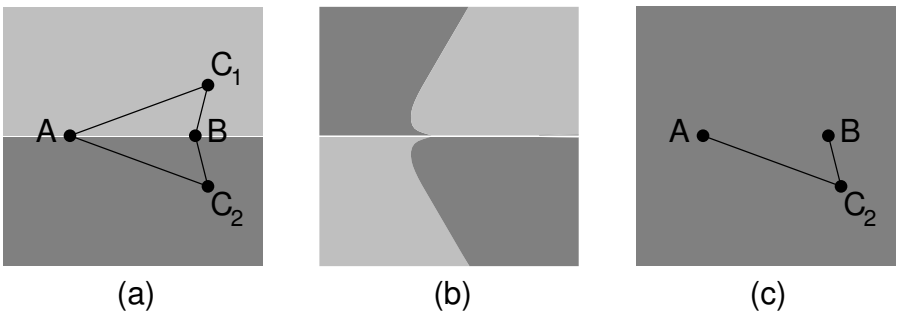


Abbildung 4.16: Startpunktabhängige Lösungsfindung wie in Abbildung 4.15. Zur Berechnung wurde in (a) und (b) das Homotopie-Verfahren eingesetzt, in (c) das vorgestellte iterativ-konstruktive Verfahren mit dem Modus „behalte letzte Lösungsauswahl bei“. In (a) wurden zwanzig Zwischenschritte bis zur Lösung berechnet, in (b) nur zehn, wobei der Suchraum in jeder Dimension 2^{14}-mal größer war. (b) zeigt, dass der Einsatz des Homotopie-Verfahrens grundsätzlich auch bei sehr einfachen Modellen fehlschlagen kann, allerdings musste das Fehlschlagen durch einen extrem großen Suchraum und eine größere Schrittweite erzwungen werden.

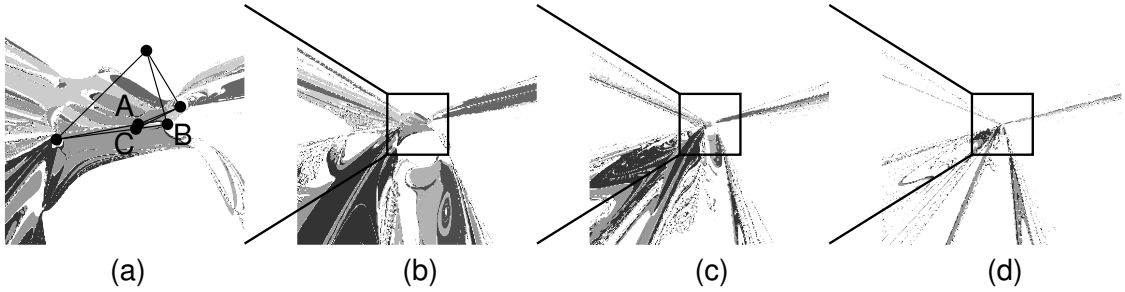


Abbildung 4.17: Startpunktabhängige Lösungsfindung bei Berechnung eines nach Schema III versteiften Modells (vgl. Abbildung 4.13). Zur Berechnung wurde das Newton-Raphson-Verfahren angewendet. A und B sind fix. Der Startpunkt C_0 wurde variiert. Die unterschiedlichen Grautöne entsprechen unterschiedlichen Lösungen, die von den jeweiligen Positionen aus gefunden wurden. Positionen, von denen aus das Verfahren divergierte, sind weiß. In (a) ist der Suchraum in der Nähe des Modells gezeigt. (b), (c) und (d) zeigen einen jeweils um den Faktor vier größeren Suchraum. Die Linien zwischen den Teilbildern verdeutlichen, wo (a) in (b) usw. zu finden ist.

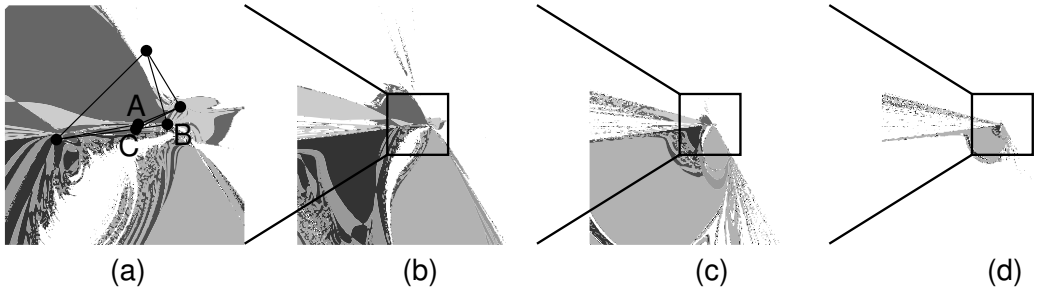


Abbildung 4.18: Startpunktabhängige Lösungsfindung bei Berechnung eines nach Schema III versteiften Modells (vgl. Abbildung 4.13). Zur Berechnung wurde das Homotopie-Verfahren angewendet. A und B sind fix. Der Startpunkt C_0 wurde variiert. Die unterschiedlichen Grautöne entsprechen unterschiedlichen Lösungen, die von den jeweiligen Positionen aus gefunden wurden. In (a) ist der Suchraum in der Nähe des Modells gezeigt. (b), (c) und (d) zeigen einen jeweils um den Faktor vier größeren Suchraum. Die Linien zwischen den Teilbildern verdeutlichen, wo (a) in (b) usw. zu finden ist.

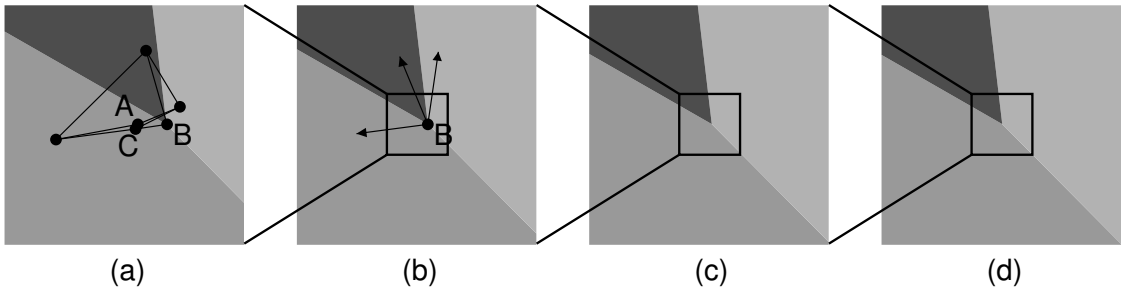


Abbildung 4.19: Startpunktabhängige Lösungsfindung bei Berechnung eines nach Schema III versteiften Modells (vgl. Abbildung 4.13). Zur Berechnung wurde Ficucs genutzt. A und B sind fix. Der Startpunkt C_0 wurde variiert. Die drei unterschiedlichen Grautöne entsprechen den drei unterschiedlichen Lösungen, die die Fehlerfunktion liefert (vgl. Abbildung 4.14, oberste Zeile). In (a) ist der Suchraum in der Nähe des Modells gezeigt. (b), (c) und (d) zeigen einen jeweils um den Faktor vier größeren Suchraum. Die Linien zwischen den Teilbildern verdeutlichen, wo (a) in (b) usw. zu finden ist. In (b) sind zudem die Richtungen vom fixen B zu den drei möglichen Lösungen für C eingetragen, die mit dieser Fehlerfunktion gefunden werden können. Es wird ersichtlich, dass die Winkelhalbierende zwischen zwei Richtungen jeweils die Begrenzung für das Gebiet darstellt, dem Ficucs eine bestimmte Lösung zuordnet. So wird z. B. die in (a) dargestellte Lösung immer dann gefunden, wenn C initial im mittelgrauen Bereich liegt.

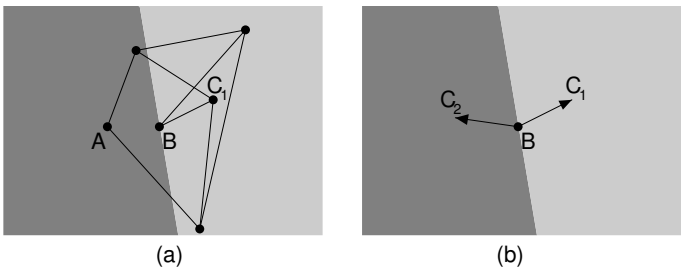


Abbildung 4.20: Der von Ficucs berechnete Suchraum entsprechend der Fehlerfunktion in Abbildung 4.14, zweite Zeile. (a) zeigt die erste der beiden möglichen Lösungen. Alle Startpositionen für C, die sich zu Beginn der iterativen Suche im hellgrauen Bereich befinden, führen zur Lösung $C = C_1$. Liegt die Startposition im dunkelgrauen Bereich, so ermittelt Ficucs $C = C_2$. In (b) wird die Suchraumaufteilung durch die Winkelhalbierenden der beiden Lösungsrichtungen von C deutlich.

2. Insbesondere die direkten Konstruktionsschritte ermöglichen eine gezielte Wahl der Lösung und führen so zu den homogenen Bereichen in Abbildung 4.19 und 4.20.

3. Ist Ficucs in einem Modus, in dem bei der iterativen Suche die Nullstelle der Fehlerfunktion zur Konstruktion genutzt wird, die am nächsten zur erwarteten Lösung liegt, so kann durch geeignete Wahl der Startwerte zwischen den Lösungen gewechselt werden. In Abbildung 4.19 wird die Richtung von Punkt B nach Punkt C iteriert. Die initiale Positionierung von C ist somit sehr gut zur Steuerung der Auswahl geeignet[35].

4. Unter Nutzung der auf Seite 114 beschriebenen Methoden zur direkten Steuerung der Lösungswahl ist das gezielte Durchsuchen der Fehlerfunktionen und somit das Finden aller Lösungen möglich.

Heuristiken

Zunächst ist festzustellen, dass keine einheitliche Festlegung existiert, wann bei einem Verfahren zur Berechnung constraint-basierter Modelle von einer Heuristik[36] zu sprechen ist. Es wäre durchaus plausibel, auch die Anwendung des Newton-Raphson-Verfahrens als Heuristik einzustufen, denn der Versuch, die Lösung basierend auf dem linearisierten Modell zu finden, beruht auf der Annahme, dass dies zu einem Erfolg führt. Wie die Untersuchungen in Abschnitt 4.2.3 zeigen, ist diese Annahme jedoch auch oft falsch.

Eine grundlegende Heuristik zur Berechnung von Punktmengen (z. B. den Schemen I bis IV) geht von der Vorstellung aus, dass für jeden Abstands-Constraint, der zwischen zwei Punkten im geometrischen Modell definiert ist, eine Feder in ein entsprechendes mechanisches System eingebracht werden kann. Die Länge der entspannten Feder entspricht dem per Constraint definierten Abstand. Nicht eingehaltene Abstände führen zu Zug- oder Druckkräften in den Federn, die bei einer Simulation des mechanischen Systems einen Ausgleich bewirken. Hierbei werden sich die Punkte so lange bewegen, bis ein Gleichgewichtszustand erreicht ist. Dies kann der gesuchte Zustand sein (alle Federn sind entspannt). Es ist jedoch auch möglich, dass sich das System nur in einem lokalen Minimum befindet. Dann sind noch Zug- und Druckkräfte vorhanden bzw. einige Abstands-Constraints sind nicht erfüllt. Schorn beschreibt in [Sch93] einen entsprechenden Ansatz. Eine genauere Erläuterung wird in Anhang B.8 gegeben. Das Resultat der Berechnungen (siehe Abbildung 4.21) bleibt bei Startpunkten nahe der Lösung nahezu konstant. Aber auch bei weiterer Entfernung bleiben die Bereiche wesentlich homogener als bei den numerischen Verfahren. Bei diesem Vergleich ist jedoch zu berücksichtigen, dass bei dieser Heuristik schon bei einem relativen Fehler von 10^{-2} abgebrochen wurde[37]. Zudem lag die Anzahl der erlaubten Iterationen bei 10.000, d. h. beim 10-fachen dessen, was bei den numerischen Verfahren zulässig war.

Auch Solano und Brunet zeigen ähnliche Algorithmen (siehe Anhang B.9). Zudem geben die beiden ein Beispiel, dass eine Heuristik unter bestimmten Voraussetzungen ähnlich einem herkömmlichen Verfahren arbeiten kann[38] (siehe Anhang B.7).

In [Luz01] und [JLS02] wird von Luzon eine Evolutionsstrategie beschrieben, die dazu dienen soll zulässige Lösungen für *2D*-Konturen zu finden. Es wird angenommen, dass durch die Art der Berechnung der Konturen zunächst exponentiell viele Lösungen auftreten[39]. Aus diesen Lösungen soll

---

[35] Die entsprechende Fehlerfunktion ist Abbildung 4.14 zu entnehmen (oberste Zeile). Die Bereichsbegrenzungen in Abbildung 4.19 ergeben sich als Winkelhalbierende der drei Richtungen BC_1, BC_2 und BC_3, wobei C_1 bis C_3 die jeweilige Lage des Punktes C ist. Die drei Richtungen wurden in Abbildung 4.19(b) eingetragen.

[36] [Kli54]: „heuristisches Prinzip: vorläufig, versuchsweise angenommener Grundsatz, der sich als falsch erweisen kann, aber die Auffindung von Erkenntnissen ermöglicht“

[37] Eine Fehlergrenze von 10^{-12}, wie sie bei den numerischen Verfahren gewählt worden war, ist für diese Heuristik völlig indiskutabel.

[38] Im Umkehrschluss lassen sich Heuristiken durch Abwandlung herkömmlicher mathematischer Verfahren gewinnen.

[39] Ein verwendeter Plan besteht aus direkten Konstruktionsschritten. Die Modelle enthalten viele „Abstand von Punkt zu Punkt“-Constraints, was zu der großen Anzahl von Konstruktoren mit Mehrfachlösungen führt.

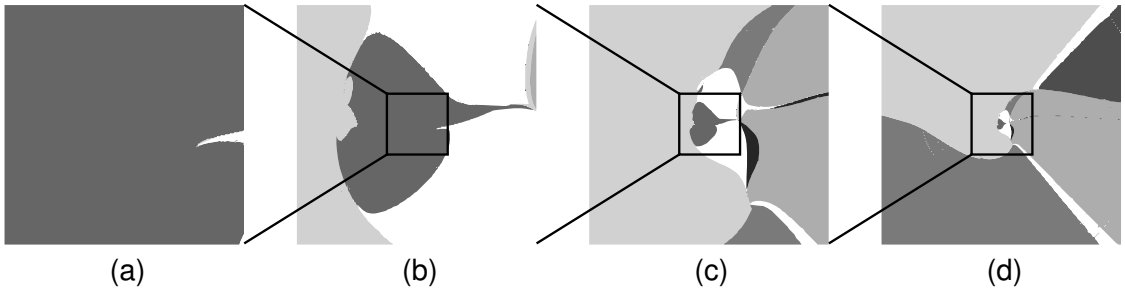


Abbildung 4.21: Startpunktabhängige Lösungsfindung bei Berechnung eines nach Schema III versteiften Modells (vgl. Abbildung 4.13). Zur Berechnung wurde die in Anhang B.8 näher beschriebene Heuristik nach Schorn angewendet. Für die Berechnungen waren p_1 und p_2 fix. Als Startpunkt wurde p_3 variiert. Die unterschiedlichen Grautöne entsprechen unterschiedlichen Lösungen, die von den jeweiligen Positionen aus gefunden wurden. In (a) ist der Suchraum in der Nähe des Modells gezeigt. (b), (c) und (d) zeigen einen jeweils um den Faktor vier größeren Suchraum. Die Linien zwischen den Teilbildern verdeutlichen, wo (a) in (b) usw. zu finden ist. Im weißen Bereich wurde nach 10.000 Iterationen die Suche als erfolglos abgebrochen. Als Fehlergrenze wurde 0.01 angenommen.

eine Lösung ausgewählt werden, die den zusätzlich gegebenen Hinweisen (in den Arbeiten als *extra constraints* bezeichnet) entspricht. Während der Suche nach einer richtigen Lösung wird die lokale Lösungsauswahl in den einzelnen Konstruktoren mit Hilfe genetischer Algorithmen variiert. Dies soll verhindern, dass der exponentiell große Lösungsraum erschöpfend durchsucht werden muss. Die genetischen Algorithmen werden auf zwei ausgewählte Beispiele angewendet und für gut befunden. Der allgemeine Nutzen des Einsatzes genetischer Algorithmen in einem Constraint-Solver ist allerdings insbesondere hinsichtlich der Robustheit der Lösungsfindung und der Berechnungsgeschwindigkeit fraglich[40]. Die Vermeidung von Mehrfachlösungen (siehe Seite 108) und die sofortige Berücksichtigung der Hinweise in den Konstruktoren gewährleisten einen weitaus effizienteren Umgang mit Mehrfachlösungen.

Vergleich der Berechnungsgeschwindigkeiten

Die Berechnungsgeschwindigkeiten von Newton-Raphson, Homotopie und iterativen Konstruktionen unterscheiden sich nicht wesentlich. Homotopie nutzt zwar mehrmals Newton-Raphson, um die einzelnen Zwischenschritte zu berechnen (Lamure und Michelucci schlagen in [LM96] ca. zwanzig vor), dafür wird die Lösung jeweils aber auch schneller gefunden. Ein allgemein gültiger Vergleich der Lösungsgeschwindigkeit der beiden Verfahren scheint nicht sinnvoll, denn es wären hierbei viele Aspekte zu berücksichtigen, u. a.:

- Wie wird Divergenz festgestellt, d. h. wie schnell wird die Suche abgebrochen? Da Homotopie öfter als Newton-Raphson konvergiert, würde ein spätes Abbrechen der Suche sich nachteilig auf die durchschnittliche Lösungsgeschwindigkeit für Newton-Raphson auswirken.
- Wie ist der Suchraum geartet bzw. wie weit liegen die Startwerte von der Lösung entfernt? Wenn oft in Bereichen gesucht wird, wo eines der beiden Verfahren (typischerweise eher Newton-Raphson) divergiert oder schlecht konvergiert, wird sich das Verhältnis der Lösungsgeschwindigkeiten entsprechend verschieben.

[40] Zudem ist nicht plausibel, warum neben der *Mutation* (zufällige Veränderung der lokalen Lösungsauswahl) ein sogenannter *Crossover* (Vertauschung von lokalen Lösungsauswahlen) sinnvoll sein sollte.

- Wie sollen Parameter gewählt werden? Bei Newton-Raphson lässt sich ein Faktor einführen, der die Schrittweite steuert. Faktoren kleiner eins wirken ähnlich dem Homotopie-Verfahren. Sie verlangsamen die Annäherung an die Lösung, senken aber gleichzeitig die Gefahr der Divergenz (zumindest wenn sich die Startpunkte in der Nähe einer Lösung befinden).

Verglichen mit den numerischen Ansätzen sind die in [Sch93] und [SB00] beschriebenen Heuristiken sehr langsam. Während das Newton-Raphson-Verfahren bei günstig gewählten Startwerten in etwa zehn Schritten eine Lösung mit einem Fehler von kleiner als 10^{-12} findet, sind bei den heuristischen Ansätzen nach ca. hundert Schritten erst Fehler von 10^{-2} erreicht.

Der Geschwindigkeitsvorteil des iterativ-konstruktiven Ansatzes erwächst aus seiner Robustheit und seiner Steuerbarkeit. Die Gefahr, dass numerische Verfahren keine Lösung finden, ist groß. Sie müssen dann u. U. immer wieder mit neuen Startwerten aufgerufen werden, insbesondere auch, wenn eine bestimmte Lösung aus mehreren möglichen zu finden ist.

Kapitel 5

Anwendungsbeispiele

Im Laufe der Arbeit am Constraint-Solver sind viele verschiedene Anwendungsbeispiele entwickelt worden. Die Entwicklung umfasst zum einen die Implementierung der eigentlichen Anwendungsprogramme und zum anderen die Modellierung der Beispiele. In diesem Kapitel soll die Vielfalt der Einsatzfälle von Constraints für das geometrische Modellieren im Allgemeinen und die Umsetzung mit dem vorgestellten Constraint-Solver im Besonderen gezeigt werden.

5.1 Lösen von Gleichungen und Gleichungssystemen

In CAx-Modellen sind neben geometrischen und topologischen Zusammenhängen in *2D* oder *3D* meist auch physikalisch-technische Zusammenhänge constraint-basiert abzubilden. Oft können diese als Gleichungen formuliert werden. Die nachfolgenden Abschnitte zeigen hierfür einige Beispiele, u. a. auf Seite 145, wo bei der Auslegung einer Tragflügelbox von geometrischen Größen auf die Materialbeanspruchung geschlossen wird. Es ist jedoch auch möglich, nur Parameter und Gleichungen zwischen den Parametern in einem Modell zu hinterlegen, z. B. folgendes Gleichungssystem:

$$\begin{array}{rcrcrcrcr} a & + & b & + & 3c & + & d & = & 6 \\ a & + & 2b & + & c & + & 3d & = & 19 \\ 3a & + & b & + & 2c & + & d & = & 11 \\ 2a & + & 2b & + & c & + & d & = & 13 \end{array} \tag{5.1}$$

Die Effizienz der Lösung derartiger Gleichungssysteme liegt jedoch hinter der von speziellen Gleichungslösern zurück (im zugehörigen Plan sind drei ineinander verschachtelte Iterationen enthalten, siehe Seite 157), es sei denn, das System weist von vornherein eine Dreiecksstruktur auf. Das Gleichungssystem:

$$\begin{array}{rcrcrcrcr} a & + & b & + & 3c & + & d & = & 6 \\ & - & 4b & + & c & - & d & = & -17 \\ & & & & c & + & 3d & = & 11 \\ & & & & & & 7d & = & 28 \end{array} \tag{5.2}$$

wird z. B. direkt (d. h. ohne Iterationen) gelöst. Der vom Constraint-Modul erstellte Plan sowie eine Beschreibungsdatei für das Gleichungssystem 5.2 kann im Anhang auf Seite 155 eingesehen werden.

Außer den oben gezeigten linearen Gleichungen werden jedoch auch andere Formen von Gleichungen bei der Modellierung benötigt. Der folgende Text geht auf die vom Constraint-Modul

direkt unterstützten Gleichungs-Constraints ein und zeigt, wie unterschiedliche Gleichungsformen mit den zur Verfügung gestellten Mitteln in Modellen umgesetzt werden können. Die aktuelle Umsetzung ist hierbei als Kompromiss zwischen Aufwand zum Aufbau der Modelle[1] und Implementierungsaufwand aufzufassen. In der Zukunft sind ggf. noch weitere Gleichungs-Constraints hinzuzufügen, wenn für bestimmte Projekte der Modellieraufwand zu groß wird.

Sämtliche in den folgenden Abschnitten gezeigten Gleichungs-Constraints sind sowohl im *2D*- als auch im *3D*-Modul nutzbar.

5.1.1 Modellierung mittels einem $y = m \cdot x + n$ Constraint

Die beim Modellieren am häufigsten verwendete Form stellen lineare Gleichungen dar. Oft sind Parameter durch lineare Gleichungen zueinander in Beziehung zu setzen, teilweise auch mittels sehr einfacher Constraints, wie z. B. zur Vorzeichenumkehr[2] ($Parameter_{positiv} = -1 \cdot Parameter_{negativ}$) oder zur Verdopplung ($Parameter_{ganz} = 2 \cdot Parameter_{halb}$) von Parameterwerten. Bei einfachen Beziehungen genügt der Constraint $y = m \cdot x + n$. Mitunter ist es nötig Konstanten zu referenzieren, z. B. $a_{pos} = -1 \cdot a_{neg} + 0$ für eine Vorzeichenumkehr. Zur Modellierung komplexerer linearer Gleichungen sind Hilfsvariablen einzuführen und mehrere Constraints der Form $y = m \cdot x + n$ zu verwenden. Beispiele sind im Anhang zu finden, u. a. auf Seite 155, wo die Modellierung des Gleichungssystems 5.2 gezeigt wird.

Die Gleichungssysteme 5.1 und 5.2 tragen eher akademischen Charakter. Es lassen sich jedoch auch physikalisch-technische Modelle finden, die nur Gleichungs-Constraints anwenden.

Beispiel 5.1: Gleichungssystem für Berechnungen an einem fest eingespannten Stab
In [CS94] wird von Chung und Sachs ein Beispiel für einen fest eingespannten Stab gezeigt. Das beschriebene Modell betrachtet nicht die geometrischen Zusammenhänge, sondern besteht nur aus elf Variablen, zwischen denen sechs Gleichungen (5.3 bis 5.8) definiert sind[3].

$$\sigma = \frac{M \cdot y}{I} \tag{5.3}$$

$$M = F \cdot l \tag{5.4}$$

$$I = \frac{b \cdot h^3}{12} \tag{5.5}$$

$$y = \frac{h}{2} \tag{5.6}$$

$$K = \frac{3 \cdot E \cdot I}{l^3} \tag{5.7}$$

$$\phi = \frac{F \cdot l^2}{2 \cdot E \cdot I} \tag{5.8}$$

Da für die elf Parameter (je ein Freiheitsgrad) nur sechs Gleichungen (sechs Valenzen) definiert sind, können $11 - 6 = 5$ Parameter frei gewählt werden. Bei einer Simulation wären das *b, h, l, F* und *E*. In diesem Fall sind die anderen sechs Parameter direkt berechenbar (siehe Plan auf Seite 156). Im Beispiel wurden jedoch *ϕ, E, M, K* und *σ* als gegeben angenommen, so dass die geometrischen Parameter *b, h, y* und *l* sowie die Kraft *F* und das Elastizitätsmodul *E* zu berechnen sind. Das Modell wird somit für die Synthese von Geometrie, Material und angreifender Kraft genutzt. Die Berechnung kann in dieser Richtung nicht mehr direkt (vgl. Abschnitt 2.7) erfolgen, denn es gibt keine Gleichung, in der nur ein Parameter unbekannt ist. In einem Constraint-Solver, der auf symbolischer Basis Umformungen der Gleichungen vornehmen kann, wäre es nun z. B. möglich, mit Hilfe der Gleichungen 5.4, 5.7 und 5.8 (drei Gleichungen,

[1] Nicht vorhandene Formen von Gleichungen lassen sich oft mittels anderer Formen und den Einsatz von Hilfsvariablen modellieren. Der Modellierungsaufwand kann jedoch hoch sein.

[2] Vergleiche z. B. vorzeichenbehaftete Abstandsmaße in Abbildung 3.5.

[3] Bedeutung der verwendeten Parameter nach [BK90]: σ Biegespannung, M Biegemoment, y halbe Höhe, I Flächenmoment zweiten Grades, F angreifende Kraft, l wirkende Hebellänge, b Querschnittbreite, h Querschnitthöhe, K Federkonstante, E Elastizitätsmodul, ϕ resultierender Neigungswinkel im Angriffspunkt.

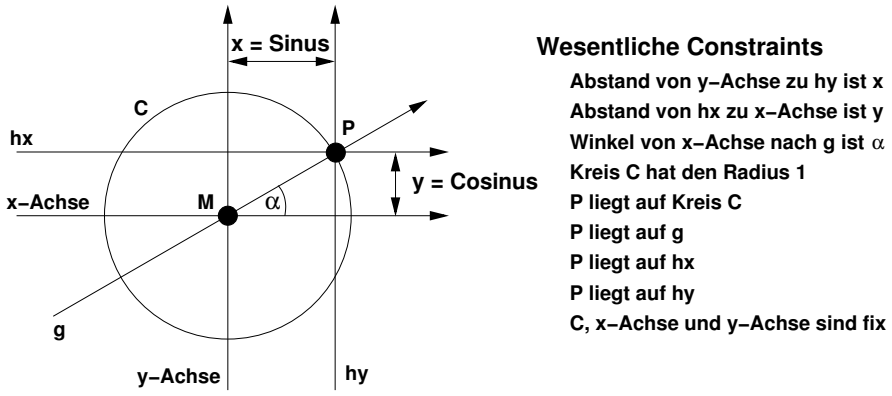


Abbildung 5.1: Modellierung von Sinus und Cosinus mittels geometrischer Constraints, wie sie in MASP zur Verfügung stehen

die die drei Unbekannten F, E und l beinhalten) eine Berechnungsvorschrift für F aufzustellen (Gleichung 5.9).

$$F = \sqrt{\frac{2}{3} \cdot \phi \cdot K \cdot M} \tag{5.9}$$

Auf Basis dieser Gleichung wären alle Unbekannten direkt ermittelbar. Das vorgestellte Constraint-Modul kann nicht auf einen Algorithmus zur Umformung von Gleichungen zurückgreifen. Es löst das Problem iterativ. Im Anhang A.1 sind der Plan für die iterative Konstruktion sowie eine Beschreibungsdatei aufgeführt. Es wird gezeigt, wie unter ausschließlicher Verwendung der einfachen Form $y = m \cdot x + n$ die obigen Gleichungen modelliert werden können. □

Um eine direkte Konstruktion zu ermöglichen, wäre es jedoch auch möglich, Constraints wie Gleichung 5.9 dem Modell hinzuzufügen. Derartige Constraints würden jedoch zu einer Überbestimmtheit führen. Sie kann durch die Bildung von Mengen *alternativer Constraints* (siehe Abschnitt 2.6 sowie 3.3.8) verhindert werden. In obigen Beispiel würden die Gleichungen 5.4, 5.7, 5.8 und 5.9 eine solche Menge bilden, aus der nur drei Constraints gleichzeitig wirken dürfen.

5.1.2 Einsatz von Trigonometrischen Constraints

Grundsätzlich lassen sich trigonometrische Constraints durch geometrische Constraints und wenn nötig einen Gleichungs-Constraint der Form $y = m \cdot x + n$ modellieren[4].

Beispiel 5.2: Geometrische Modellierung von trigonometrischen Constraints

Abbildung 5.1 zeigt eine Möglichkeit, wie die Gleichungen $x = sin(\alpha)$ und $y = cos(\alpha)$ mit geometrischen Constraints modelliert werden können. Wesentlich ist hierbei, dass das Constraint-Modul die Vorzeichen von Winkel und Sinus bzw. Cosinus beachtet, d. h. das Constraint-Modul muss für Konstruktionen sowohl beim Winkel-Constraint als auch bei den Abstands-Constraints die Vorzeichen der Parameter richtig auswerten, zudem muss es bei der Berechnung von Parameterwerten das Vorzeichen richtig bestimmen.□

Um den in Abbildung 5.1 verdeutlichten Aufwand[5] [6] zu sparen, stellt das Constraint-Modul einige trigonometrische Constraints direkt bereit. Im Einzelnen handelt es sich um:

---

[4]Wenn im entsprechenden Modell die Hypotenuse per Definition gleich eins ist, dann wird kein Gleichungs-Constraint benötigt. Ansonsten muss der Constraint für die Normierung mit der Hypotenusenlänge benutzt werden.

[5]Die Verwendung eines vorzeichenbehafteten „Abstand Punkt zu Gerade"-Constraints würde das Modell vereinfachen, denn die Hilfsobjekte hx und hy wären dann unnötig. Das vorgestellte Constraint-Modul bietet jedoch nur einen Constraint zur Definition eines Abstandes zwischen parallelen Geraden. Eine andere Vereinfachung ergibt sich, wenn direkt auf die Koordinaten von P zugegriffen werden kann, insbesondere wenn M im Ursprung liegt.

[6]Wenn trigonometrische Funktionen in komplexeren Modellen benötigt werden, dann sind Teile der in Abbildung

- $x = sin(\alpha)$,
- $y = cos(\alpha)$ und
- $\alpha = arctan2(y, x)$.

Auf die Semantik von Sinus- und Cosinus-Constraint muss hier nicht näher eingegangen werden. Der *arctan*2-Constraint hat drei Parameter. Es handelt sich hierbei um einen Winkel α und die Längen der zwei Katheten des entsprechenden rechtwinkligen Dreiecks. y ist die Länge der Gegenkathete und x die Länge der Ankathete. Der Vorteil von $arctan2(y, x)$ gegenüber einem herkömmlichen $arctan(y/x)$ liegt in der Vermeidung der Division durch null.

Da, wie bereits in Abschnitt 1.1.1 gezeigt wurde, die Berechnungsrichtung durch Constraints nicht festgelegt wird, lassen sich mit den drei oben gezeigten trigonometrischen Constraints auch die Beziehungen:

- $\alpha = arcsin(x)$,
- $\alpha = arccos(y)$,
- $y = tan(\alpha)$ sowie
- $x = cot(\alpha)$

direkt modellieren. Für *tan* und *cot* sind ggf. Hilfsvariablen bzw. Konstanten zu verwenden, z. B.:

- im Constraint $\alpha = arctan2(y, 1)$ zur Modellierung von $y = tan(\alpha)$ und
- im Constraint $\alpha = arctan2(1, x)$ zur Modellierung von $x = cot(\alpha)$.

5.1.3 Sonstige Gleichungs-Constraints

Außer den oben gezeigten Gleichungs-Constraints wurde noch ein Constraint

$$a = abs(x) \tag{5.10}$$

implementiert, mit dem Absolutwerte gebildet werden können. Die Bildung von Absolutwerten ist insbesondere deshalb wichtig, weil das Constraint-Modul Abstände und Winkel typischerweise vorzeichenbehaftet auswertet bzw. berechnet. Werden für Darstellungen in der Benutzungsschnittstelle oder für physikalisch-technische Berechnungen Absolutwerte benötigt, so ermöglicht der Constraint eine einfache Modellierung der Umformung zu einem stets positiven Parameter[7].

Die Bereitstellung eines Constraints der Art

$$w = sqrt(x) \tag{5.11}$$

zur Modellierung von Quadratwurzeln ist wegen der beliebigen Berechnungsrichtung des Constraints $y = m \cdot x + n$ nicht nötig. Jede Beziehung $a = \sqrt{b}$ ist durch den Constraint $b = a \cdot a + 0$ darstellbar.

5.1 gezeigten Geometrie oft schon im Modell enthalten, d. h. in solchen Fällen ist der Modellieraufwand geringer als in der Abbildung dargestellt.

[7] In der Regel soll der Nutzer nicht mit für ihn unverständlichen negativen Vorzeichen konfrontiert werden, so dass in den entsprechenden Fällen eine Vorzeichenumkehr sinnvoll ist. Eine aufwändige geometrische Alternative zur Verwendung einer Gleichung ist z. B. die Konstruktion eines Punktes P in dem vorzeichenbehafteten Abstand von einer Gerade (*2D*) oder Ebene (*3D*) sowie seiner Projektion P' auf die Gerade bzw. Ebene. Der Abstandsparameter im Abstands-Constraint von P zu P' ist dann stets positiv.

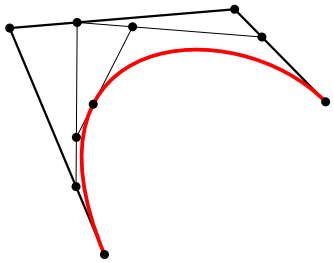


Abbildung 5.2: eine Bezierkurve mit Kontrollpolygon und das zugehörige geometrische Modell für die Konstruktionsvorschrift

Maximum und Minimum lassen sich mit Hilfe des Abs-Constraints (Entfernung des Vorzeichens) modellieren. Die Bestimmung des Maximums m aus den Parametern a und b

$$m = \max(a, b) \tag{5.12}$$

lässt sich wie folgt modellieren:

$$m = \frac{a + b + abs(a - b)}{2} \tag{5.13}$$

Analog lässt sich ein Minimum n

$$n = \min(a, b) \tag{5.14}$$

ermitteln. Es ist lediglich ein Vorzeichen umzukehren:

$$n = \frac{a + b - abs(a - b)}{2} \tag{5.15}$$

5.2 2D-Geometrie

Der Constraint-Solver wurde mit seinen Erweiterungen für Objekte in *2D* in unterschiedlichen Applikationen eingesetzt. Die wichtigsten sind:

- eine Testapplikation für *2D*-Constraints und entsprechende Objekte (Constraint-Modellierer),
- ein Modul zur Eingabe von Konturen (COSMOS verwendet diese Kontureingabe als Basis für Sweeps) und
- MASP, ein Werkzeug zur Modellierung und Analyse von Lösungsprinzipen.

5.2.1 Constraint-Modellierer (2D)

Der Constraint-Modellierer ist für die direkte Modellierung mit geometrischen Objekten (Punkte, Kreise, Geraden), Parametern und den entsprechenden Constraints und Hinweisen konzipiert. Er kann für verschiedene Zwecke eingesetzt werden, z. B.:

- zum interaktiven Testen von Konstruktionsbeschreibungen, so zeigt Abbildung 5.2 die im Constraint-Modellierer erstellte Konstruktionsbeschreibung einer Bezierkurve entsprechend dem Rekursionsschema von De Casteljau (siehe z. B. [BM01]),
- als Lehr-/Lernsoftware für den Umgang mit Constraints in *2D*,
- zum möglichst direkten interaktiven Testen von Ficucs sowie auch insbesondere
- zum Auffinden von geeigneten Modellierungsvarianten für unterschiedlichste Problemstellungen.

Auf Basis der interaktiv als geeignet ermittelten Modellierungsvarianten erfolgt die weitere Implementierung constraint-basierter Anwendungssoftware. Aber auch die Vorbereitung einer Erweiterung von Ficucs ist so möglich, z. B. wenn spezielle Transformationen Bestandteil von Ficucs werden sollen, um mit ihrer Hilfe Iterationen zu vermeiden. Diese Vorgehensweise hat sich u. a. bereits bei der Modellierung der in MASP unterstützten Prinzipelemente (sowohl der ebenen als auch der sphärischen) bewährt.

Beispiel 5.3: Modellierung einer einfachen Kollisionsvermeidung für MASP

Im Zuge der Arbeiten an MASP sollte eine Kollisionserkennung (bzw. -vermeidung) zwischen einem Punkt P und einem Getriebeglied G dem Nutzer zur Modellierung der Lösungsprinzipe bereitgestellt werden. P kann hierbei z. B. ein Drehgelenk, ein Schubgelenk oder das Ende eines anderen Getriebegliedes sein. Vor einer Implementierung dieser Funktionalität als C++-Code erfolgte die interaktive Untersuchung geeigneter Modellierungsvarianten.
Die Grundidee bestand darin, dass P stets dann nicht die Seite von G wechseln darf, wenn er sich „neben“ G befindet. Die entsprechende Logik sollte mit Conditional-Constraints realisiert werden. Varianten ergaben sich z. B.:

- bei der Modellierung des Tests auf die relative Position (wie soll „neben“ constraint-basiert formuliert werden) und
- bei der Verhinderung des Seitenwechsels, falls festgestellt wurde, dass sich P „neben“ G befindet.

Ziel der Untersuchungen war es, eine möglichst einfache und zugleich für den Nutzer nachvollziehbare Art der Modellierung zu finden. Abbildung 5.3(a) zeigt eine ungünstige Variante, denn hier werden unnötigerweise Hilfsobjekte und entsprechende Constraints eingeführt, um festzustellen, ob P neben G liegt. Die Hilfsgeraden H_1 und H_P führen zudem zu einer für praktische Belange unsinnigen unendlichen Ausdehnung des Bereiches. Für den Test, ob sich P „neben“ dem Getriebeglied befindet, wird bei Variante (a) der vorzeichenbehaftete Abstand von H_1 zu H_P mit 0 sowie der Länge des Getriebegliedes verglichen. Variante (b) aus Abbildung 5.3 vergleicht den Abstand von P zu G_1 sowie von P zu G_2 mit der Getriebegliedlänge und kommt hingegen gänzlich ohne Hilfsobjekte aus. In dieser Variante können die Bedingungen direkt formuliert werden. Zudem scheint die seitliche Begrenzung des Bereiches, der die Bedingung „neben“ erfüllt, einleuchtender als die unendliche Ausdehnung des Bereiches in Abbildung 5.3(a).

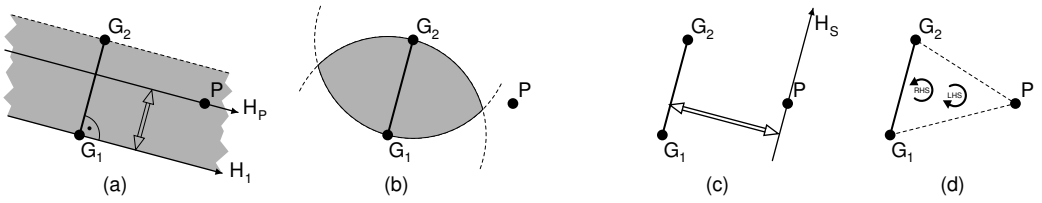


Abbildung 5.3: (a) und (b) zeigen Tests auf den Zustand „neben“, (c) und (d) können nur zur Beibehaltung der Seite dienen.

Auch bei der Verhinderung des Seitenwechsels sind unterschiedliche Varianten untersucht worden. Abbildung 5.3(c) zeigt die Modellierung über eine Hilfsgrade H_S und einen „Paralleler Abstand“-Constraint. Die Seite wird durch das Vorzeichen des Abstandes von H_S zur Hauptachse von G bestimmt. Das Modell in

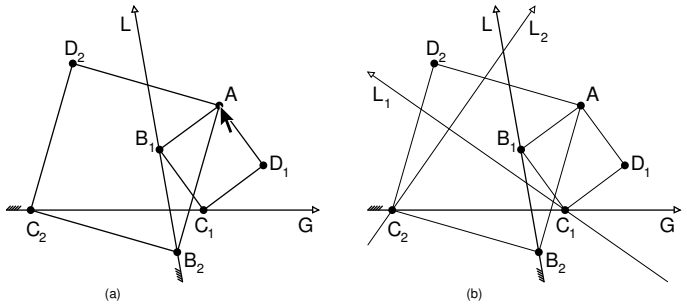


Abbildung 5.4: (a) zeigt zwei Lösungen für die Konstruktion eines Quadrates bei gegebenem Punkt A und der Inzidenz der Punkte B und C zu zwei gegebenen Geraden L und G. In (b) sind die per Drehstreckung transformierten Geraden L_1 und L_2 eingezeichnet, welche eine direkte Konstruktion der Quadrate ermöglichen.

Abbildung 5.3(d) nutzt *XHS*-Hinweise, so dass vorteilhafterweise auf Hilfsobjekte verzichtet werden kann. In der Abbildung wird der *RHS*-Hinweis erfüllt.

In Tabelle 5.1 (Seite 136) ist die für MASP gewählte Umsetzung der Kollisionserkennung zwischen einem Punkt und einem Getriebeglied beschrieben (Kombination der in (b) und (d) gezeigten Varianten). Auf Seite 158 kann die zum Testen benutzte Beschreibungsdatei der insgesamt günstigsten Modellierung eingesehen werden. Der Fokus lag hierbei auf einer einfachen Modellierung. Die sich aus ihr ergebende Begrenzung des Bereiches, in dem Seitenwechsel verhindert werden, wird „wohlwollend“ hingenommen, da dieses Verhalten typischerweise der Nutzerintention entspricht. □

Beispiel 5.4: Modellierung einer Drehstreckung

In Abbildung 5.4(a) sind die zwei möglichen Lösungen einer Aufgabe gezeigt, bei der ein Quadrat zu konstruieren ist. Ein Eckpunkt (A) sowie zwei Geraden (L und G), auf denen zwei weitere Eckpunkte des Quadrates (B und C) liegen, seien gegeben. Man beachte, dass in der Aufgabenstellung durch die Fixierung von A (oder das Bewegen von A mit dem Mauszeiger) nur die Position des Quadrates gegeben ist, nicht aber seine Ausrichtung und seine Größe. Diese werden nur implizit durch die zwei Inzidenz-Constraints festgelegt. Aus diesem Grund kann der Constraint-Solver die Lösungen nur iterativ berechnen.
Mit einer entsprechenden Hilfskonstruktion, lassen sich die beiden Lösungen jedoch auch direkt konstruieren. Hierfür ist eine Transformation der Geraden L oder G nötig. In Abbildung 5.4(b) ist ein Modell gezeigt, das auf der Transformation von L zu L_1 und L_2 basiert. Es handelt sich hierbei um eine sogenannte Drehstreckung[8], die auf die Gerade angewandt wird.
Im Constraint-Modellierer lassen sich beide Modelle (die ursprüngliche, iterativ zu konstruierende Beschreibung und die transformierte, direkt konstruierbare Beschreibung) untersuchen und Rückschlüsse für eine geeignete Modellierung ziehen. Später können die hierbei gewonnenen Erkenntnisse in eine Weiterentwicklung des Constraint-Solvers oder von Applikationen, die diesen nutzen, fließen. □

5.2.2 2D-CAD

Kommerzielle 2D-CAD Systeme sind auch bei der Unterstützung von Constraints schon weit fortgeschritten. Oft wird der *D-Cubed Constraint Manager* (DCM) der Firma *D-Cubed* eingesetzt, z. B. im AutoDesk Inventor. Andere Firmen, wie PTC, nutzen Eigenentwicklungen. Der Aufwand ein gleichmächtiges System aufzubauen wäre sehr groß, insbesondere bezüglich Benutzungsoberfläche (Menüs, Dialoge, Assistenten, Visualisierung usw.) sowie Import und Export. Deshalb wurden nur kleinere (akademische) Beispiele realisiert. Auf eine interessante Möglichkeit den Freiheitsgradverteilungsansatz von Ficucs für die Steuerung von Modifikationen feature-basierter Modelle einzusetzen wird im folgenden Text näher eingegangen.

[8] Diese Transformation wird in der Literatur zur Elementargeometrie beschrieben, z. B. in [BS67] oder [Adl06].

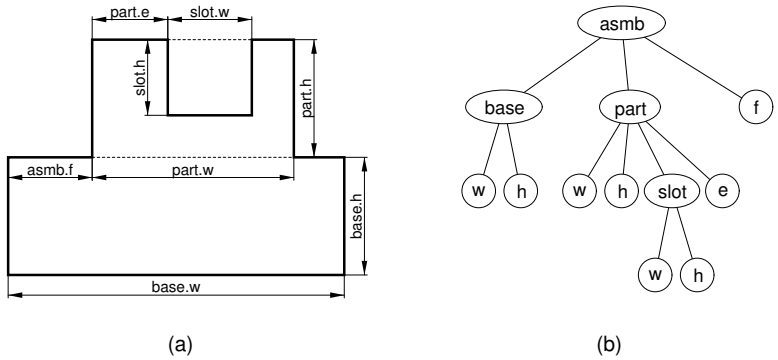


Abbildung 5.5: Featurehierarchie zur Strukturierung einer Kontur. (a) Vermaßte Kontur. (b) Hierarchische Anordnung der Features und Zuordnung der Parameter.

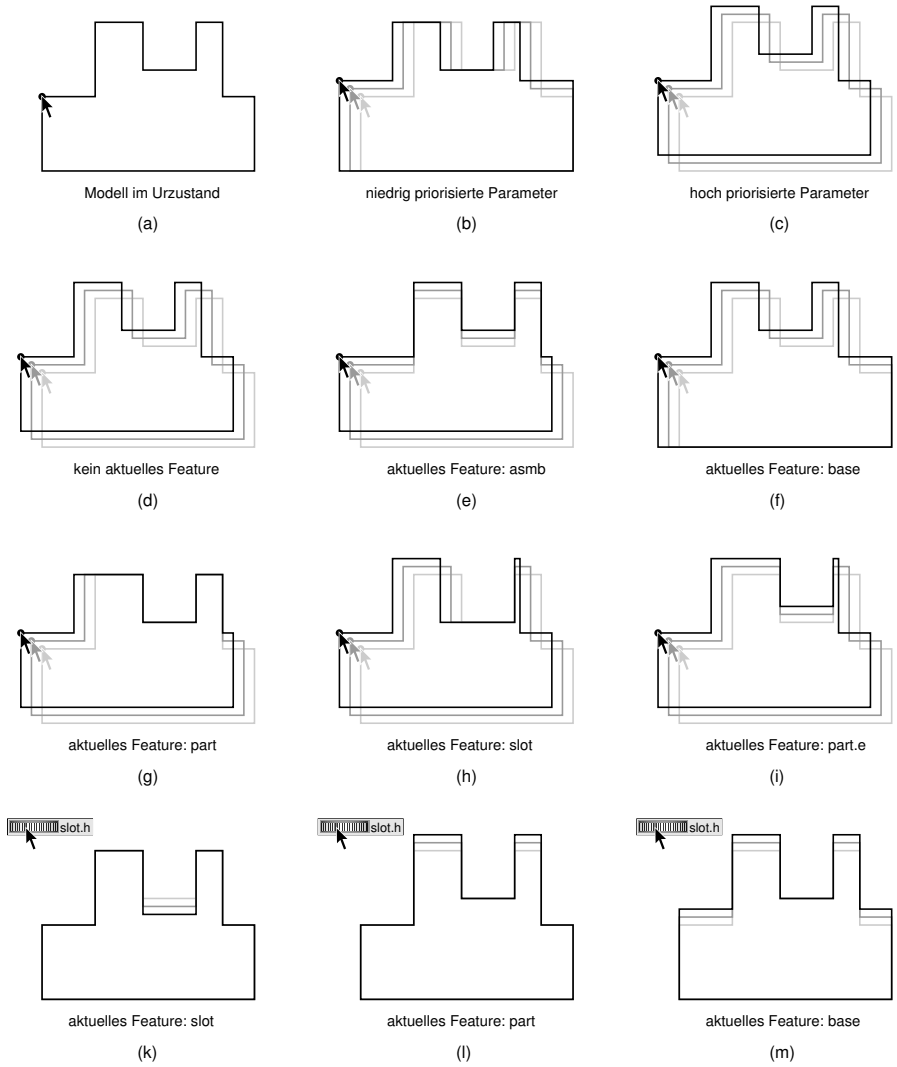


Abbildung 5.6: Interaktive Modifikation des Modells aus Abbildung 5.5. Detaillierte Erläuterungen werden im Beispiel 5.5 gegeben.

Feature-basierte Modellierung zur Steuerung der Auswirkung interaktiver Änderungen

In [BDKM00] werden die Möglichkeiten des Einsatzes einer Featurehierarchie in COSMOS erörtert. Die wichtigsten Aussagen sind folgende:

- Änderungen sollten möglichst lokal vorgenommen werden (dann sind sie für den Nutzer gut nachvollziehbar), wobei sich stets die Frage stellt, was der Nutzer gerade als lokal betrachtet.
- Geometrische Elemente und Parameter können gruppiert bzw. als logisch zusammengehörig beschrieben werden. Eine solche Gruppe wird hier als Feature bezeichnet. Die hierarchische Anordnung von Features impliziert „ist Teil von"-Beziehungen. Auch hierdurch wird Zusammengehörigkeit (also Nähe) beschrieben.
- Die Featurehierarchie wird deshalb typischerweise so aufgebaut, dass ein Feature seine Subfeatures und seine Parameter sowie geometrischen Elemente umfasst.
- Der Nutzer kann wählen, an welcher Stelle der Featurehierarchie sich seine Änderungen auswirken sollen. In der jetzigen Implementierung von Ficucs wird die Stelle durch genau ein *aktuelles Feature* beschrieben[9], das im folgenden Text mit F bezeichnet wird.
- Aus Sicht des Features F sind die Subfeatures grundsätzlich starr[10]. Änderungen im Feature sollen sich in der Repositionierung der geometrischen Elemente sowie Subfeatures auswirken, wobei die entsprechenden Parameter des Features angepasst werden.
- Innerhalb von F wird eine Priorisierung der Freiheitsgrade vorgenommen (wie in den Abschnitten 2.2.5 und 5.2.3 beschrieben), was insbesondere für Features mit vielen lokalen Freiheitsgraden wichtig ist.
- Bei unerwünschten Auswirkungen kann der Nutzer die nach den standardmäßigen Einstellungen gefundene Freiheitsgradverteilung durch temporäre Vergabe von Constraints und insbesondere auch von Fixierungen beeinflussen.

Typischerweise greift der Nutzer bei seiner Interaktion auf geometrische Elemente oder Parameter von F zu. Es besteht jedoch auch die Möglichkeit, die Änderung an einer anderen Stelle des Modells zu veranlassen. Hierdurch kann der Nutzer eine gezielte Änderung an einer Stelle S_1 des Modells vornehmen, wobei sich die Änderung an einer Stelle S_2 (den Elementen des vom Nutzer gewählten aktuellen Features) auswirkt. Der Einfachheit halber sind die nun folgenden Erläuterungen zur Nutzung einer Featurehierarchie in *2D* gegeben. Beispiel 5.5 beschreibt hierzu die constraintbasierte Definition einer Kontur. Derartige Konturen werden in COSMOS (siehe Abschnitt 5.3.1) zur Generierung von Volumenmodellen herangezogen. Der Schwerpunkt des Beispiels liegt darauf zu verdeutlichen, wie sich durch Modellierung und Berücksichtigung einer Featurehierarchie Modifikationen an unterbestimmten Modellen besser kontrollieren lassen.

Beispiel 5.5: Featurehierarchie in 2D
Abbildung 5.5(a) zeigt ein Modell, das sich aus drei rechteckigen Querschnitten zusammensetzt. Sie sind mit *base*, *part* und *slot* benannt. Constraints sichern die Einhaltung der sechs Parameter für Breite und Höhe in den drei Rechtecken. Außerdem sichern sie die Einhaltung der relativen Positionen zwischen *base* und *part* sowie *part* und *slot*. Da die insgesamt acht Parameter nicht fixiert sind (ebenso wie die Position des Modells), ist das Modell unterbestimmt. Dementsprechend können sich Probleme bei der Steuerung von Modifikationen ergeben. Aus Abbildung 5.5(b) wird die gewählte Anordnung der einzelnen Features in der Featurehierarchie ersichtlich. Von besonderer Bedeutung ist hierbei die Zuordnung der Parameter *asmb.f* und *part.e* (siehe Text zu den Teilabbildungen (e), (g) und (i).

[9] Denkbar wäre auch die gleichzeitige Wahl mehrerer Features (evtl. die priorisierte Menge der aktuellen Features), hierzu wurden jedoch noch keine Untersuchungen vorgenommen.
[10] Gemeinsam genutzte Parameter der Subfeatures könnten jedoch dem Feature zugeordnet werden.

Abbildung 5.6 zeigt die Reaktion des Modells auf Veränderungen, wobei deutlich wird, dass dieses unterbestimmte Modell auf sehr verschiedene Art reagieren kann. In den Teilabbildungen (a) bis (i) wird jeweils der obere linke Punkt des Features *base* bewegt. Zunächst findet die Featurehierarchie (siehe Abbildung 5.5(b)) bei der Bestimmung der Freiheitsgradverteilung jedoch keine Berücksichtigung. (b) verdeutlicht, wie sich die nächstliegenden Parameter ändern (*base.w*, *base.h* und *part.h*), wenn die Parameter niedrig priorisiert sind. Hoch priorisierte Parameter führen in (c) zu einer Verschiebung des starr bleibenden Modells.
Ab Abbildung 5.6(d) wird die Featurehierarchie in der Bestimmung der Freiheitsgradverteilung berücksichtigt[11]. Das Modell bleibt in (d) jedoch starr, da kein aktuelles Feature festgelegt wurde. Das aktuelle Feature gibt an, wo sich die Änderungen im Modell vorzugsweise auswirken sollen. In Abbildung 5.6(e) wird die Wirkung gezeigt. Die Bewegung des Mauszeigers in x-Richtung ändert *asmb.f* (den Parameter für die relative Position von *base* und *part*), denn nur *asmb.f* ist im aktuellen Feature *asmb* als Parameter enthalten. Dementsprechend ändert die Mausbewegung in y-Richtung nur Positionen, jedoch keine Parameter. In den Teilabbildungen (f) bis (h) wird deutlich, wie sich die Mauszeigerbewegung auf die Höhe und Breite des jeweils aktuellen Features auswirkt und die anderen Modellbestandteile starr bleiben. Aber auch einzelne Parameter können für Änderungen bevorzugt werden, wie (i) am Beispiel der relativen Positionierung von *part* und *slot* zeigt (Parameter *part.e*). Die Höhen und Breiten von *base*, *part* und *slot* bleiben hier konstant.
In den Teilabbildungen 5.6(k) bis (m) wird der Parameter für die Höhe des Features *slot* mittels Schieberegler vergrößert. Diese Änderung beeinflusst stets die Form von *slot*. In Teilabbildung (k) erfolgt die Änderung lokal in *slot* dort, weil es als das aktuelle Feature festgelegt wurde. Wenn wie in (l) und (m) aber andere Modellbestandteile als aktuelles Feature gekennzeichnet werden, wirken sich Änderungen auch dort aus. In (l) vergrößert sich die Höhe von *part* und in (m) die Höhe von *base*. □

5.2.3 Lösungsprinzipe (MASP)

MASP erwuchs aus einer schrittweisen Erweiterung des in Abschnitt 5.2.1 beschriebenen *2D*-Constraint-Modellierers. Ziel war und ist es, eine Applikation zu schaffen, die die Arbeit in den frühen Entwurfsphasen für Mechanismen und Getriebe unterstützt. Neben einer Anpassung der Oberfläche an die speziellen Modellierungs- und Analyseaufgaben wurden hierbei auch diverse Erweiterungen an Ficucs vorgenommen. Diese Erweiterungen waren zum einen durch die speziellen Anforderungen in MASP motiviert, zum anderen aber auch allgemein genug, um sie für andere Projekte einsetzen zu können. Die Arbeiten sind in [Bri01] und vielen anderen Veröffentlichungen, die im Zusammenhang mit MASP entstanden sind ([BBDH98, BBD99, BBDH01, BBDH02, BDR03, BD03]) beschrieben. Einige der in MASP verwendeten Techniken werden in den folgenden Abschnitten erörtert, insbesondere auch solche, die eine Motivation für die Entwicklung der in den vorangegangenen Kapiteln beschriebenen Algorithmen gaben und deshalb als Beispiele für den Einsatz der Algorithmen dienen sollen.

Vorteil constraint-basierter Modellierung

MASP verknüpft die Vorteile einer effizienten interaktiven Eingabe, wie sie z. B. von Vektorgrafikprogrammen bekannt ist, mit der Möglichkeit die Lösungsprinzipe für Simulationen und Analysen zu nutzen. Der Vorteil einer interaktiven Eingabe wird insbesondere dann deutlich, wenn man MASP mit Systemen vergleicht, die nur spezielle Strukturen handhaben können. Dort ist Spezialwissen über diese unterstützten Strukturen und ihre Bezeichnung nötig, um Lösungsprinzipe eingeben zu können (siehe z. B. [RL00] und [VDI95]). Mitunter wird für die Analyse auch eine Zerlegung in Assursche Gruppen gefordert (siehe [LM95]). Ziel derartiger Vorgehensweisen ist es den Nutzer zu zwingen, die Lösungsprinzipe so zu definieren, dass für die Analysen auf vordefinierte Berechnungsalgorithmen zugegriffen werden kann. Darunter leidet zwangsläufig die Allgemeinheit dieser Ansätze. Ein freies experimentelles Modellieren der Lösungsprinzipe, eine testweise kinematische Umkehr[12] oder auch das Ziehen an einem beliebigen Punkt oder Getriebeglied ist nicht

[11] Der Modus wurde in der Applikation über einen Schalter interaktiv geändert.
[12] siehe Beispiel 5.6

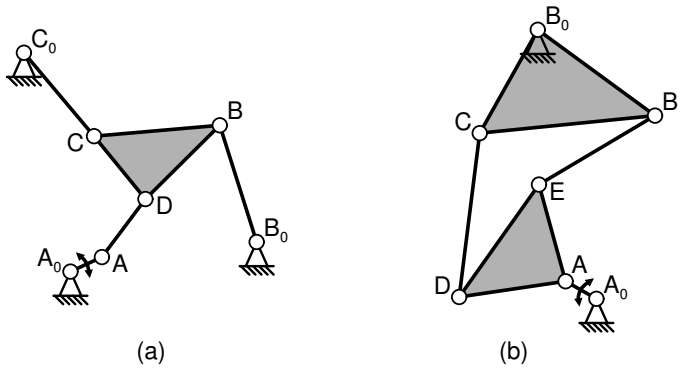


Abbildung 5.7: Koppelgetriebe, die für die Animation von Ficucs iterativ zu berechnen sind, wenn sie in A_0 angetrieben werden. (a) mit einer Assurschen Gruppe III. Ordnung und (b) mit einer Assurschen Gruppe IV. Ordnung.

möglich. Erst durch einen allgemeinen Ansatz, wie ihn die constraint-basierte Modellierung im Zusammenspiel mit einem Constraint-Solver bietet, wird ein wichtiger Schritt hin zu einem kreativen Entwurf auf der Basis von Lösungsprinzipen getan.

Modellierung von Prinzipsymbolen

Für die constraint-basierte Modellierung technischer Prinzipe kommt in MASP eine symbolische Darstellung zum Einsatz, die bekannte Sinnbilder aus der Mechanismen- und Getriebetechnik nutzt. In Tabelle 5.1 sind einige Symbole für die Modellierung planarer[13] Prinzipe und deren constraint-basierte Beschreibung aufgeführt. Eine umfangreichere Zusammenstellung, die u.a. auch constraint-basierte Repräsentationen von Räder-, Zugmittel-, Schrauben- und Schrittgetrieben enthält, ist in [Bri01] zu finden.

Ausgewählte Beispiele für die Modellierung

Neben vielen kleineren Beispielen für die Modellierung von Lösungsprinzipen mit MASP, die an unterschiedlichsten Stellen dieser Arbeit beschrieben wurden, seien hier noch die folgenden beiden gegeben.

Beispiel 5.6: Modellierung von Gelenkketten mit Assurgruppen höherer Ordnung

Wie bereits erwähnt, spielen in einigen in der Literatur beschriebenen Ansätzen oder auch in Softwareprogrammen zur Analyse von Lösungsprinzipen Assursche Gruppen eine Rolle. Durch das freie Modellieren ist es in MASP möglich, beliebig komplexe Gelenkketten zu modellieren - darunter auch solche, die Assursche Gruppen höherer Ordnung beinhalten. Abbildung 5.7(a) zeigt ein Modell, das im Wesentlichen aus einer Assurschen Gruppe III. Ordnung besteht, lediglich das gestellfeste Drehgelenk A_0 und das angetriebene Getriebeglied A_0A bilden eine separate Gruppe I. Ordnung. In Abbildung 5.7(b) ist eine Assursche Gruppe IV. Ordnung dargestellt. Auch hier bilden lediglich das gestellfeste Drehgelenk A_0 und das angetriebene Getriebeglied A_0A eine separate Gruppe I. Ordnung. MASP kann mit Hilfe von Ficucs die Bewegungen der Modelle darstellen. Sitzt der Antrieb wie dargestellt jeweils in A_0, so muss Ficucs auf eine iterative Konstruktion zurückgreifen. Der Nutzer kann den Antrieb jedoch auch an einer anderen Stelle platzieren[14]. Die constraint-basierte Modellierung ermöglicht auch in einem solchen Falle die entsprechenden Berechnungen. Iterative Konstruktionen sind dann oft nicht nötig. □

[13] Planare Prinzipe lassen sich durch geometrische Objekte und Constraints in der euklidischen Ebene abbilden. Demgegenüber sind sphärische Prinzipe auf eine Kugeloberfläche abbildbar (siehe Abschnitt 5.3.4) und zur Beschreibung allgemeiner räumlicher Getriebe werden *3D*-Geometrieobjekte und Constraints benutzt (Abschnitt 5.3.5).

[14] kinematische Umkehr

Tabelle 5.1: Modellierung einiger Prinzipbestandteile für planare Mechanismen und Getriebe

Prinzipbestandteile und ihre Verknüpfung	Constraint-basierte Repräsentation
Drehgelenk ○	Ein Drehgelenk wird als Punkt abstrahiert (x- und y-Koordinatenparameter).
Gestellfestes Drehgelenk 	Ein fixierter Punkt.
Schubgelenk 	Ein Punkt auf einer Gerade.
Getriebeglied 	Ein Getriebeglied wird beschrieben durch: • eine Hauptachse (eine Gerade), • zwei Punkte (Anfangs- und Endpunkt), • zwei „Punkt auf Gerade“-Constraints, um die Punkte auf der Geraden zu halten, • einen Längenparameter für die Getriebegliedlänge, • einen Abstands-Constraint (zwischen dem Anfangs- und dem Endpunkt) um die Länge sicherzustellen.
Verknüpfungen zwischen Gelenken und Getriebegliedern	Um einen Anfangs- oder Endpunkt eines Getriebegliedes mit einem Drehgelenk zu verbinden wird ein Gleichheits-Constraint zwischen den entsprechenden Punkten definiert. Um ein Schubgelenk auf einem Getriebeglied zu platzieren werden die beiden Hauptachsen per Constraint gleichgesetzt. Zwei Ungleichungs-Constraints verhindern das „Herunterfallen“ des Schubgelenks vom Getriebeglied.
Kollisionsvermeidung zwischen einem Punkt P und einem Getriebeglied G	Entsprechend der Beschreibungsdatei auf Seite 158 wird die Vermeidung der Kollision beschrieben durch: • die Bedingung `P_nahe_G1`, die als inaktive Ungleichung für den Abstand von P zum Endpunkt G_1 definiert ist, • die Bedingung `P_nahe_G2`, die als inaktive Ungleichung für den Abstand von P zum Endpunkt G_2 definiert ist, • den zunächst inaktiven *RHS*-Hinweis `P_ist_rechts` für G_1, P und G_2, • den zunächst inaktiven *LHS*-Hinweis `P_ist_links` für G_1, P und G_2(später wird höchstens einer der beiden Hinweise aktiv sein), • den Conditional-Constraint, der `P_ist_rechts` aktiviert, wenn die Bedingungen `P_nahe_G1`, `P_nahe_G2` und `P_ist_rechts` zutreffen sowie • den Conditional-Constraint, der `P_ist_links` aktiviert, wenn die Bedingungen `P_nahe_G1`, `P_nahe_G2` und `P_ist_links` zutreffen.

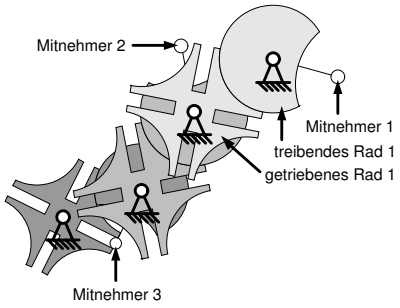


Abbildung 5.8: Zählwerk aus Malteserkreuz-Getrieben.

Beispiel 5.7: Modellierung eines Zählwerkes

Um die Modellierung von Schrittgetrieben durch Ficucs besser unterstützen zu können wurde das Konzept der *Conditional Constraints* in den Constraint-Solver eingeführt. Abbildung 5.8 zeigt ein Zählwerk, das aus drei einzelnen Malteserkreuz-Getrieben zusammengesetzt ist. Da die treibenden Räder nicht ständig mit dem jeweils getriebenen Rädern verbunden sind, müssen die Constraints, die die Verbindungen realisieren, ein- und ausschaltbar sein. Bei Gelenkketten sind die Constraints hingegen stets aktiv. Die Bedingungen für das Ein- und Ausschalten werden über geometrische Kriterien definiert, im Fall des Malteserkreuzes ist das z. B. der Abstand vom Mitnehmer am treibenden Rad zum Mittelpunkt des jeweils getriebenen Rades. Eine ausführliche Beschreibung der Modellierung kann [BBD99] entnommen werden. Die Anwendung der Conditional Constraints für eine Kollisionsvermeidung innerhalb von Lösungsprinzipen ist auf den Seiten 130 und 158 erläutert. □

Umsetzung geometrischer Lösungsverfahren

Für eine Vielzahl von Berechnungen in Wissenschaft und Technik besteht die Möglichkeit, eine Lösung durch die Ausführung von geometrischen Konstruktionen zu finden. Die entsprechenden Verfahren wurden über viele Jahre von Ingenieuren auf dem Papier umgesetzt. Heute ist eine derartige Vorgehensweise jedoch weitgehend durch den Einsatz von Berechnungsprogrammen verdrängt, welche vornehmlich mit numerischen Gleichungslösern arbeiten. Zwar werden die geometrischen Verfahren noch gelehrt, in der Praxis kommen sie aber kaum noch zum Einsatz. In Zusammenhang mit Constraint-Solvern könnten die Verfahren jedoch eine Renaissance erleben. Die geometrischen Konstruktionen, die sie beschreiben, lassen sich als Menge von Objekten in der Ebene (Punkte, Geraden, Kreise, Abstände, Winkel) und entsprechende Constraints beschreiben. Im folgenden Abschnitt soll dies anhand der Umsetzung eines Verfahrens zur Analyse von Kräften in Stabwerken verdeutlicht werden[15].

Beispiel 5.8: Analyse von Stabkräften in einem Tragwerk

Die Kraftanalyse in Stabwerken kann mit Hilfe von Methoden der Graphischen Statik (siehe z. B. [Hen11], [Bla44] oder [Dre53]) erfolgen. Eine dieser Methoden basiert auf dem sogenannten Cremonaplan (engl.: Maxwell diagram). Abbildung 5.9(a) zeigt das Stabwerk eines Dachträgers mit dem dazugehörigen Kraftdiagramm in Form eines Cremonaplanes. Es gibt eine angreifende Kraft F und aus ihr resultierende Kräfte in den Stäben (fünf Stück) sowie in den Lagern[16], wobei das Lager B nur vertikale Kräfte aufnehmen kann. Mit Hilfe der Kraftanalyse können die Kräfte in den Stäben und den Lagern bestimmt werden. Von den

[15] In [Sha02] und [Sha03] zeigt Shai die Dualität von Kraftanalyse in Stabwerken und Geschwindigkeitsanalyse in Mechanismen. Der im folgenden Text vorgestellte Ansatz der constraint-basierten Analyse von Kräften ist dementsprechend auf Geschwindigkeitsanalysen übertragbar.

[16] Der Dachträger ist starr und hat somit drei Freiheitsgrade. Diese werden durch die Lager fixiert. Lager A fixiert die Position und Lager B den verbleibenden Richtungsfreiheitsgrad. Eine mögliche Überbestimmtheit wird durch die Verwendung eines Gleitlagers für B vermieden, da Gleitlager nur eine Valenz von eins haben (sie erlauben die Bewegung auf einer fixen Geraden).

Kräften in den Stäben und in Lager B ist die Wirkungsrichtung bekannt. Die Richtung der Kraft F_A in Lager A muss erst im Laufe der Berechnungen ermittelt werden.

Das Stabwerk und die angreifenden Kräfte sind interaktiv modelliert worden. Die Struktur des Cremonaplanes ergibt sich aus dem Stabwerk und den definierten Kräften. Bei der automatischen Generierung des Cremonaplanes werden folgende Objekte:

- fünf Geraden für die Wirkungslinien der Stabkräfte,
- zwei Geraden für die Wirkungslinien der Lagerkräfte,
- eine Gerade für die Wirkungslinie der abgreifenden Kraft und
- fünf Punkte, die die Maschen im Tragwerk symbolisieren (siehe Abbildung 5.9(b))

sowie Constraints:

- Parallelität[17] der Stäbe im Dachträger mit den zugehörigen Geraden im Diagramm (fünf Stück),
- Parallelität[17] der an den Dachträger angreifenden Kräfte (bzw. deren Wirkungslinien) mit den zugehörigen Geraden im Diagramm (drei Stück) und
- „Punkt in Gerade"-Constraints entsprechend der Struktur des Cremonaplanes (fünfzehn Stück)

erzeugt. Jeder der oben beschriebenen Constraints hat eine Valenz von eins. Es ergibt sich somit eine Gesamtvalenz von $5 + 3 + 15 = 23$. Da der Cremonaplan aus fünf Punkten und acht Geraden mit jeweils zwei Freiheitsgraden besteht, ergibt sich ein Gesamtfreiheitsgrad[18] von $(5 + 8) \cdot 2 - 23 = 3$. Dies spiegelt die Möglichkeit der freien Positionierung (zwei Freiheitsgrade) und der freien Skalierung (1 Freiheitsgrad) wider. Eine Rotation ist bei Modellierung mit Parallelitäts-Constraints nicht möglich!

Die Skalierung gibt das Verhältnis der Kraftpfeillängen bezüglich Stabwerk und Diagramm wieder. Zur besseren Übersichtlichkeit des Diagrammes wurde in Abbildung 5.9(a) der Skalierungsfaktor S auf einen Wert von (konstant) 4 gesetzt. Die Einbindung des entsprechenden Parameters in das Modell erfolgt durch folgende Constraints:

- $f.len = S * F.len$
- $f_A.len = S * F_A.len$
- $f_B.len = S * F_B.len$

wobei $X.len$ jeweils die Länge des Kraftpfeiles X im Stabwerk referenziert und $x.len$ sich auf die entsprechende Länge im Cremonaplan bezieht. Die Wirkung der Gleichungen wird im später aufgeführten Berechnungsablauf deutlich.

Interessanterweise lässt sich der in Abbildung 5.9(a) dargestellte Cremonaplan nicht mit Zirkel und Lineal konstruieren, da die Kraft F an einem Knoten mit drei Stäben angreift und so nicht eindeutig zerlegbar ist[19]. In den Lehrbüchern wird deshalb auf andere (z. B. das Seileck-) Verfahren verwiesen. Ficucs ist jedoch nicht auf die ausschließlich konstruktive Lösbarkeit angewiesen. Er kann die Lösung iterativ ermitteln, z. B. indem er die Wirkungsrichtung der Lagerkraft A oder den Betrag der Lagerkraft B variiert bis eine konsistente Lösung gefunden wurde.

Ein typischer Berechnungsablauf ist der folgende:

1. An der Spitze des Kraftpfeiles F wird interaktiv gezogen. Seine Endpunkte sind somit bekannt, der eine z. B. durch die Mausposition, der andere wegen des starren und in seiner Lage fixierten Stabwerkes[20]. Länge und Richtung von F ergeben sich als Abstand der beiden Punkte bzw. aus dem Anstieg zwischen ihnen.

---

[17] Das entspricht einem Winkel von 0° zwischen den jeweiligen Geraden. Es könnten statt der Parallelität auch Winkel-Constraints verwendet werden, die alle den selben Winkelparameter referenzieren, dem dann beliebige Werte zugewiesen werden können. Dieser Winkel repräsentiert die Ausrichtung bzw. den Rotationsfreiheitsgrad des Diagrammes.

[18] Die Rechnung geht davon aus, dass die Richtungen des Stabwerkes fixiert sind und die Parallelitäts- (bzw. Winkel-) Constraints in Richtung des Cremonaplanes wirken.

[19] Bei einem Knoten mit zwei Stäben lassen sich die entsprechenden zwei Stabkräfte aus einem Dreieck ablesen, von dem eine Seite (die bekannte Kraft, oder auch die Summe der bekannten Kräfte) und zwei Winkel (jeweils zwischen Kraftrichtung und Richtung der Stäbe) bekannt ist.

[20] Wir nehmen an dieser Stelle an, dass das Stabwerk bereits fertig konstruiert ist. Anmerkung: Auch das Stabwerk ist nicht ohne weiteres direkt konstruierbar. Ficucs geht hierbei iterativ vor. Durch Clustering könnte dies vermieden werden.

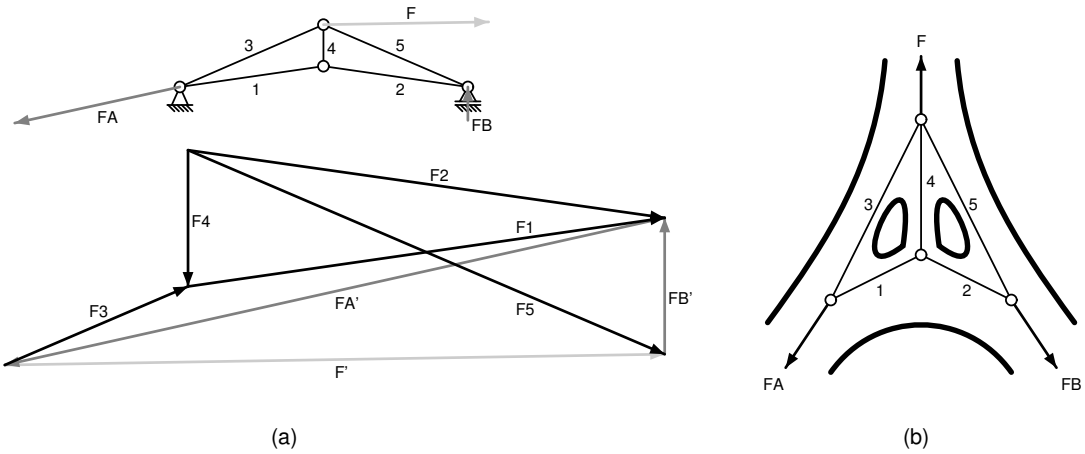


Abbildung 5.9: Cremonaplan für einen Dachträger. (a) Darstellung des Dachträgers und des zugehörigen Cremonaplanes in MASP. Ficucs kann das Modell iterativ berechnen (siehe Beispiel 5.8). (b) Schematische Darstellung des Dachträgers mit den fünf Maschen, welche durch die fünf (Knoten-)Punkte des Cremonaplanes repräsentiert werden. Maschen mit externen Kräften sind offen, z. B. die Masche, die durch F, Stabkraft 3 und F_A gebildet wird. Alle drei Kräfte kommen im Cremonaplan in einem Punkt (der, der die Masche repräsentiert) zusammen.

2. Durch einen Parallelitäts-Constraint und die oben beschriebene Skalierungsgleichung können die Länge und die Richtung von f (die Entsprechung von F im Cremonaplan) bestimmt werden. Ein Endpunkt von f kann frei positioniert werden[21]. Der andere wird entsprechend Richtung und Länge von f konstruiert.
3. Die Wirkungsgerade von f_B kann eingezeichnet werden, da ihre Richtung und ein Punkt auf ihr (der Endpunkt von f) bekannt sind. Allerdings ist die Länge von f_B noch unbekannt.
4. Im Cremonaplan kann nicht mehr direkt konstruiert werden. Es sind zwar sämtliche Richtungen bekannt, jedoch können keine Punkte gefunden werden, in denen sie anzutragen sind. Die direkte Konstruktion ist hier beendet. Der Constraint-Solver wählt nun einen Parameter aus, der ihm ein Weiterkonstruieren ermöglicht, z. B. die Länge von f_B.
5. Basierend auf der jeweils „erratenen“ Länge von f_B kann der Endpunkt von f_B bestimmt werden. Gleichzeitig ist nun F_B entsprechend dem Skalierungsfaktor im Stabwerk konstruierbar.
6. Die Endpunkte von f_A sind bekannt und legen dessen Länge und Richtung fest. Unter Berücksichtigung des Skalierungsfaktors lässt sich nun auch F_A zeichnen.
7. Aus bekannten Punkten und Richtungen können nun sukzessive Geraden und weitere Punkte (als Schnittpunkte der Geraden) konstruiert werden.
8. Am Ende der Konstruktion bleibt ein nichtberücksichtigter „Punkt in Gerade“-Constraint übrig. Durch Messen des Abstandes vom Punkt zur Geraden kann die Güte des in Schritt 4 „geratenen“ Wertes eingeschätzt werden. Der Wert wird in Schritt 4 so lange variiert (vgl. Abschnitt 3.6.3), bis sich nach entsprechender Konstruktion ein Abstand von 0 ergibt. In diesem Fall ist der passende Cremonaplan gefunden und die Kräfte in den Stäben sowie in den Lagern sind bestimmt.
9. Für weitere Berechnungen oder Anzeigen lassen sich die Längen, die die Beträge der Kräfte widerspiegeln, über einen (in entsprechenden Gleichungen referenzierten) Umrechnungsfaktor in Zahlenwerte für Kräfte, z. B. mit der Einheit Newton, umrechnen.

□

[21] Der Cremonaplan hat (wie oben beschrieben) beide Positionierungsfreiheitsgrade. Es ist sinnvoll sie an dieser Stelle gezielt einzusetzen, z. B. durch eine entsprechende Priorisierung oder Fixierung des Endpunktes.

Kontextsensitive Priorisierung

Die Priorisierungsreihenfolge muss für die Interaktionen des Nutzers mit einem bestimmten Modell nicht zwangsläufig konstant sein. Wie das folgende Beispiel zeigt, bietet MASP mit der Möglichkeit einer kontextabhängigen (bzw. kontextsensitiven) Priorisierung einen mächtigen Ansatz zur zielgerichteten Steuerung der Berechnungen in unterbestimmten Modellen.

Beispiel 5.9: Nutzung unterschiedlicher Priorisierungen bei der Tragwerkanalyse

Am Beispiel aus Abbildung 5.9(a) lässt sich sehr gut die Mächtigkeit kontextsensitiver Priorisierung demonstrieren. Die Grundidee ist, dass für unterschiedliche Aktionen unterschiedliche Prioritäten vergeben werden. Die Identifizierung der Aktion und damit der aktuell gültigen Priorität der einzelnen Typen ist über den vom Nutzer gewählten Interaktionspunkt möglich. Der Interaktionspunkt ist hierbei das Objekt, das manipuliert wird, zum Beispiel der Endpunkt eines Kraftpfeiles oder ein Parameter. Alternativ ist aber auch die Abhängigkeit der aktuellen Priorisierung von bestimmten Zuständen in der Applikation denkbar. Zum Beispiel könnte der Nutzer einen Modus wählen, in dem auch (normalerweise fixierte) Festlager verschiebbar sind.

Beispiele für Interaktionen sind folgende:

- Änderung von Größe und Richtung der angreifenden Kraft
  Dies ist der offensichtliche Normalfall für eine Interaktion mit dem in Abbildung 5.9(a) gezeigten Modell. Der Nutzer kann hierfür den freien Endpunkt des entsprechenden Kraftpfeiles verschieben. Seine relative Lage zum Angriffspunkt bestimmt die Größe der Kraft (repräsentiert durch die Länge des Kraftpfeiles) sowie die Angriffsrichtung. Das Stabwerk selbst kann fix sein. Der Cremonaplan sollte sich in Lage, Ausrichtung und Skalierung nicht ändern. Die zugehörigen Berechnungsabläufe sind oben beschrieben.
- Verschieben von Knoten
  Eine sehr interessante Interaktion ist die Verschiebung von Knoten im Stabwerk, denn hierdurch ändern sich diverse Richtungen im Cremonaplan sowie die zugehörigen Stabkräfte. Um Knoten verschieben zu können müssen Stablängen änderbar sein, d. h. sie sind nicht fixiert und haben eine geringere Priorität als die Knoten. Sinnvollerweise sollte die Verschiebung des Knotens, an dem eine äußere Kraft angreift, nicht zu deren Änderung führen. Wichtig ist es also die Richtung und die Länge der äußeren Kräfte höher zu priorisieren als die Endpunkte der Kräfte.
- Manipulation von Lagerkräften
  Selbstverständlich lässt sich die Untersuchung auch umkehren, so dass sich z. B. ausgehend von bestimmten Lagerkräften die Richtung einer angreifenden Kraft F oder die Länge eines bestimmten Stabes ermittelt wird. Das Berechnungsziel wird durch die Vergabe der kleinsten Priorität gewählt. Unter Umständen kann es passieren, dass das Berechnungsziel nur auf Basis von höher priorisierten Objekten berechnet wird, nicht aber in Abhängigkeit des Interaktionspunktes (hier die Lagerkraft). In solchen Fällen wäre eine automatische Priorisierung wünschenswert.
- Sonstige Interaktionen
  Weiterhin sind u. a. auch folgende Interaktionen durchaus sinnvoll und sollten ggf. durch eine entsprechende Priorisierung unterstützt werden: Verschieben, Skalieren oder Rotieren des Cremonaplanes sowie Manipulation einer Stabkraft.

□

Analyse von Bewegungspfaden

In diesem Abschnitt wird die für MASP gewählte Lösung des in den Abschnitten 3.6.5 und 4.1 diskutierten Problems der Wahl von Mehrfachlösungen bei Bewegungssimulationen erörtert werden. Dort wurde darauf hingewiesen, dass für spezielle Anwendungsgebiete eine Lösungsauswahl sinnvollerweise auf Applikationsebene erfolgt. Auch im Kontext kontinuierlicher Bewegungen mit einem Freiheitsgrad (d. h. auf einer implizit gegebenen Bewegungsbahn), wie sie für MASP typisch sind, trifft dies zu. Die besonderen Schwierigkeiten, die das Finden einer allgemeinen Lösungsstrategie erschweren bzw. unmöglich machen, bestehen darin:

- dass oft iterativ konstruiert werden muss,

- dass sich hierbei der Bewegungspfad typischerweise aus mehreren Teilpfaden zusammensetzt, die jeweils auf einer anderen Fehlerfunktion basieren,
- dass der Wechsel zwischen diesen Teilpfaden jeweils durch ein „Trägheitskriterium“ zu steuern ist und
- insbesondere auch, dass es in den Pfaden Bereiche geben kann, die sehr sorgfältig zu untersuchen sind, um die richtige Wahl treffen zu können (z. B. wenn dort trotz konstanter Antriebsgeschwindgkeit stark beschleunigte Bewegungen auftreten).

Allgemein reichen lokale Entscheidungen[22] für eine korrekte Auswahl nicht aus. Die möglichen Bewegungen von Punkten werden in MASP deshalb unter Berücksichtigung globaler Kriterien vorberechnet. Hierzu erfolgt quasi eine Abtastung jedes interessierenden Bewegungspfades. Das macht in MASP insbesondere dadurch Sinn, weil sich eine explizite Speicherung des Pfades auch für die Untersuchung der Geschwindigkeit und der Beschleunigung der Punkte nutzen lässt[23].

Der Lösungsansatz zur Berechnung eines Bewegungspfades beinhaltet folgende Schritte:

- Festlegung einer Menge M der Punkte, für die der Bewegungspfad zu bestimmen ist[24].
- Auswahl von geeigneten Parametern aus dem Modell. Sie sind nicht fixiert, über das Constraint-Netz mit zu analysierenden Punkten verbunden und auch nicht redundant (wie z. B. die Richtungen in einem starren Subobjekt). Die nach diesen Kriterien ausgewählten Parameter seien in der Menge P zusammengefasst.
- Nach diesen Festlegungen erfolgt eine kontinuierliche Iteration entlang der Bewegungspfade der Punkte in M. Ein einzelner Iterationsschritt lässt sich wie folgt beschreiben:
  - Auswahl des Parameters $p_t \in P$ entsprechend der Sensitivität[25] in der aktuellen Stellung (initial der Anfangsposition). p_t ist der Parameter, der sich bei einer Bewegung in dieser Stellung am meisten ändert und somit selbst am wenigsten sensitiv ist. Eine derartige Wahl vermeidet starke Änderungen anderer Parameter bzw. entsprechende sprunghafte Bewegungen von Getriebeteilen, denn eine Änderung des am wenigsten sensitiven Parameters p_t wird nur kleinere Änderungen anderer Parameter verursachen.
  - Für den Parameter p_t wird ein neuer Wert $p'_t = p_t + \Delta p$ berechnet. Die Änderung entspricht einer bestimmten Schrittweite, die von der Art des Parameters und dem Modell abhängt, z. B. ist die Schrittweite von Abstandsparametern von den Abmessungen des Mechanismus abhängig.
  - Es folgt die Berechnung des Modells mit dem aktuellen Parameter p'_t durch Ficucs. Um Trägheit zu simulieren laufen die lokalen Konstruktionen entsprechend[26] den Konfigurationen der Konstruktionen der letzten Bewegungen ab (vgl. *2D*-Konstruktoren in Abschnitt 3.5.2). Für ein korrektes Ergebnis musste die Schrittweite klein genug gewählt sein, u. U. ist eine zu große Schrittweite (z. B. zunächst wegen des Versuches der Rechenzeiteinsparung gewählt) interaktiv anzupassen.
  - Nach der Berechnung erfolgt die Speicherung der Positionen für alle Punkte $v \in M$, sowie der Information, wie sich dieses Modell in der aktuellen Stellung wieder berechnen lässt. Benötigt werden die Werte der treibenden Parameter und die aktuelle Konfigurationsinformation zur Wahl der lokalen Mehrfachlösungen.

---

[22] Die z. B. schon für das ungünstige Verhalten in Beispiel 3.21 verantwortlich waren.

[23] Hierfür werden die Bahnpositionen der Punkte synchronisiert ermittelt bzw. abgespeichert, wodurch für jede Punktbewegung auch die jeweilige Bewegung des Antriebs ermittelt werden kann.

[24] Ohne spezielle Einschränkung durch Nutzervorgaben sind das z. B. alle nicht fixierten charakteristischen Punkte des Modells.

[25] Ein Parameter ist umso sensitiver, je größer die durch seine Änderungen hervorgerufenen Änderungen anderer Parameter sind bzw. je stärker die durch die Änderung hervorgerufenen Bewegungen des Mechanismus sind.

[26] „entsprechend“ heißt nicht, dass die Konfigurationen gleich bleiben müssen. Es kann auch auf Basis der bisherigen Bewegungen extrapoliert werden!

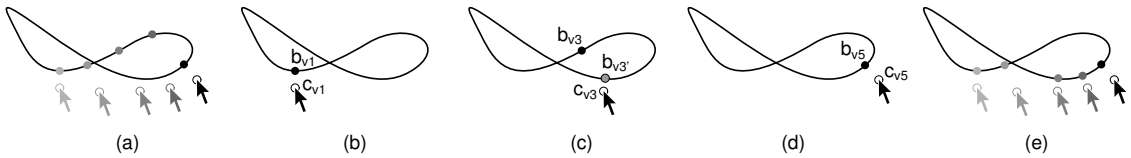


Abbildung 5.10: Bewegung eines Punktes auf einer vorgeschriebenen Bahn. (a) Angenommener (plausibler) Bewegungsfluss. (b) Anfangsposition des Bewegungsflusses, in der die Mausposition c_{v1} auf die Bahnposition b_{v1} projiziert wird. (c) Zwischenposition, die verdeutlicht, dass außer der plausiblen Projektion auf b_{v3} noch eine Projektion auf b'_{v3} möglich wäre, die sogar wesentlich dichter an c_{v3} liegt als der gewählte Punkt b_{v3}. (d) Endposition des Bewegungsflusses. (e) Fehlerhafter Bewegungsfluss.

Wenn ein Punkt $v \in M$ interaktiv mit der Maus bewegt wird, kann eine plausible Pfadposition b_v ermittelt werden, die der aktuellen Mausposition c_v entspricht. Diese ist im Kontext der Bewegung von Mechanismen jedoch nicht unbedingt diejenige Position auf dem Pfad, die den kleinsten euklidischen Abstand zur Mausposition hat. Deshalb wird bei der Projektion des zu bewegenden Punktes auf den gespeicherten Pfad die letzte Position mit berücksichtigt.

Beispiel 5.10: Projektion eines Punktes auf seinen Bewegungspfad

Abbildung 5.10 zeigt das interaktive Bewegen eines Punktes v auf einer festgelegten Bahn. Während der Interaktion kann an jedem Zeitpunkt i zur Steuerung des Punktes v auf die Mausposition c_{vi} zugegriffen werden. Die entsprechend c_{vi} beste Lösung b_{vi} führt zu einem lokalen Minimum in der Funktion des Abstandes von c_{vi} zu allen Pfadpunkten. Das Minimum ist jedoch so zu wählen, dass zwischen der letzten Lösung $b_{v,i-1}$und der aktuell gewählten besten Lösung b_{vi} kein lokales Maximum auftritt[27]. Würde dies nicht beachtet werden, so wäre in Abbildung 5.10(c) der Punkt b'_{v3} als beste Projektion gewählt worden, was jedoch im Kontext der Bewegung von Mechanismen unerwünscht ist, siehe Abbildung 5.10(e). □

Für die gefundene Position werden nun alle Informationen ausgelesen, die eine (Re-)Konstruktion der aktuellen Stellung des Getriebes ermöglicht, d. h.:

- der Wert des treibenden Parameters,
- die Konfiguration zur Steuerung der Wahl von Mehrfachlösungen und
- die Parameterwerte für iterative Konstruktionen (da die Eindeutigkeit einer Angabe wie „3. Nullstelle“ allgemein nicht gegeben ist).

Alternativ hierzu hätten auch die Daten aller geometrischen Objekte sowie Parameter gespeichert werden können, wodurch der Einsatz eines Constraint-Solvers für die Bewegungsberechnungen verzichtbar wird. Die Rekonstruktion könnte dann sehr schnell vorgenommen werden. Allerdings würden sich erhebliche Nachteile ergeben.

- Der Speicheraufwand ist um ein Vielfaches höher. Zum Vergleich:
  - In jedem Konstruktionsschritt wird zwar meist ein Objekt berechnet, aber die Konfiguration muss nur für solche Konstruktionen gespeichert werden, für die auch Mehrfachlösungen auftreten und auch hier werden sie nur gespeichert, wenn sie von null abweichen (bei Konstruktionen mit zwei Lösungsmöglichkeiten also statistisch gesehen nur die Hälfte).

[27] Man kann sich das Verhalten so vorstellen, dass zwischen dem projizierten Punkt b_v auf der Bahn (der sich dort frei bewegen kann) und dem zu projizierenden Punkt c_v ein Gummiband gespannt ist. b_v wird sich stets so bewegen, dass das Band möglichst entspannt ist. Die Existenz eines Maximums zwischen der letzten Lösung und der neuen Lösung würde bedeuten, dass sich das Band zwischenzeitlich quasi von selbst stärker gestrafft hat.

  - Die für die Objekte und Parameter zu speichernden Werte sind hingegen typischerweise Gleitkommazahlen (interner Datentyp *double,* der implementierungsabhängig mehr Speicher belegt). Die Anzahl der für die Speicherung benötigten Gleitkommazahlen richtet sich nach den Freiheitsgraden der Objekte bzw. ist gleich eins für jeden Parameter. Für per Definition gleiche Objekte (z. B. für die Endpunkte von Getriebegliedern und den Punkt, der das Drehgelenk repräsentiert, an das die Getriebeglieder gekoppelt sind) sind auch jeweils alle Werte zu speichern. Die Konsistenz ist sonst ohne Einsatz des Constraint-Solvers nicht gewährleistet.

  Dieser Nachteil kommt insbesondere dann zum Tragen, wenn der Pfad sehr fein abgetastet wurde und dementsprechend viele Stellungen in ihm gespeichert sind.

- Insbesondere bei grob abgetasteten Pfaden oder bei entsprechend starker Vergrößerung eines Teils des Bewegungspfades ist eine Feinpositionierung zwischen den abgetasteten und gespeicherten Stellungen unbedingt erforderlich, ansonsten könnte das Getriebe nur zwischen den abgespeicherten Stellungen springen (bei feiner Abtastung oder verkleinerter Darstellung des Modells wäre das kein Problem). Ohne Konstruktionsplan (einschließlich der korrekten Wahl von Mehrfachlösungen) könnte zur Feinpositionierung nur eine Interpolation erfolgen, z. B. ähnlich dem *Key-Frame*-Ansatz aus der Computeranimation (vgl. z. B. [FvDFH97]). Dabei würden jedoch unnötigerweise Fehler gemacht werden, die bei der Berechnung mittels Konstruktionsplan nicht auftreten.

Auf Basis der gespeicherten Werte erfolgt nun die Berechnung der Getriebestellung für die nächstliegende im Bewegungspfad gespeicherte Position. Daran schließt sich dann die Feinpositionierung an. Hierbei werden zwischen den beiden am nächsten gelegenen gespeicherten Positionen iterativ so lange Stellungen berechnet, bis ein minimaler Abstand des zu bewegenden Punktes vom Mauszeiger gefunden wurde oder eine bestimmte Anzahl von Konstruktionen erreicht wurde. Letztere Beschränkung dient der Sicherstellung der Interaktivität und kann bei Bedarf vom Nutzer an die Ziele seiner Interaktion (schnelle Grob- oder aufwändige Feinpositionierung) eingestellt werden.

Darstellung von planaren Mechanismen in 3D

MASP wurde für die Modellierung und Analyse planarer Mechanismen entwickelt. Die zunächst ausschließlich symbolische Darstellung wurde später durch räumliche Darstellungen ergänzt. Die Abbildungen 5.11 und 5.12 zeigen Beispiele für den Übergang vom Lösungsprinzip zu einer ersten Grobgestalt. Die softwaretechnische Unterstützung des Überganges im Rahmen von MASP ist Gegenstand verschiedener Forschungsarbeiten ([BBH+02], [BDR03], [RBD06], [BRD07], [RDB07]), wobei sich der Forschungsschwerpunkt immer weiter in Richtung des Entwurfes heterogener Systeme verschiebt.

Die für das Bewegen der Mechanismen notwendigen Berechnungen werden aber auch hier durch das *2D*-Modul von Ficucs durchgeführt. Erst die plausible Visualisierung der *3D*-Objekte ist Aufgabe der Applikation. Hierbei wird u. a. Folgendes berücksichtigt:

- Für die bewegten *3D*-Teile, wie z. B. die Polster in Abbildung 5.11, sind in Abhängigkeit von den *2D*-Positionsberechnungen *3D*-Transformationen zu ermitteln, so dass die Teile an den richtigen Positionen visualisiert werden.
- Die einzelnen Objekte sind zur Kollisionsvermeidung entsprechend der Bauraumanalyse auf unterschiedliche Ebenen aufzuteilen.
- Auch die Wahl der passenden Typen für die Koppelstellen muss zur Kollisionsvermeidung beitragen, indem die bei den Analysen ermittelte Umlauffähigkeit berücksichtigt wird.

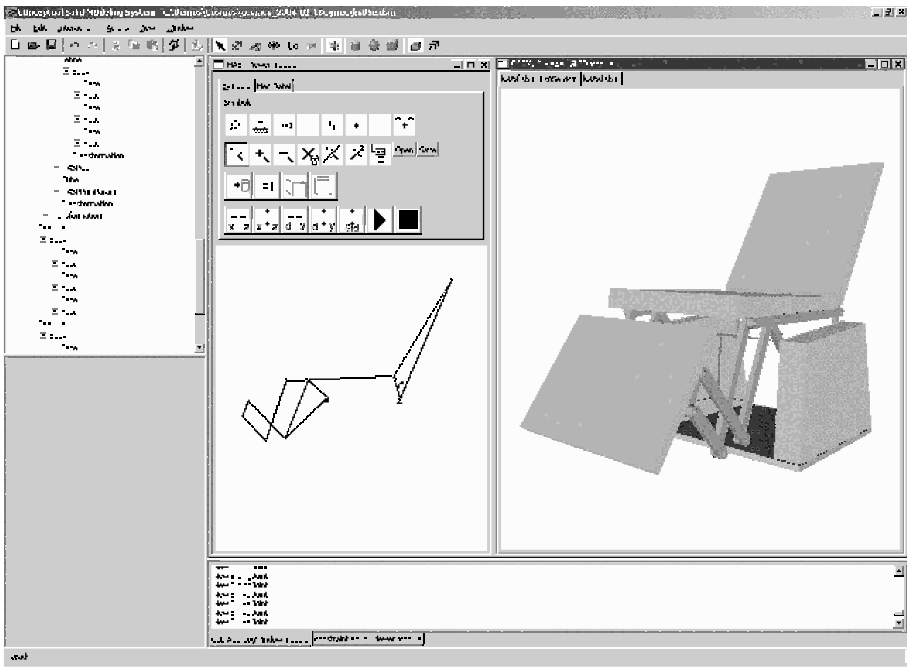


Abbildung 5.11: Modellierung eines Möbels. Die verwendeten Mechanismen sind planar und können somit durch MASP bearbeitet werden. Die Visualisierung erfolgt in *3D*.

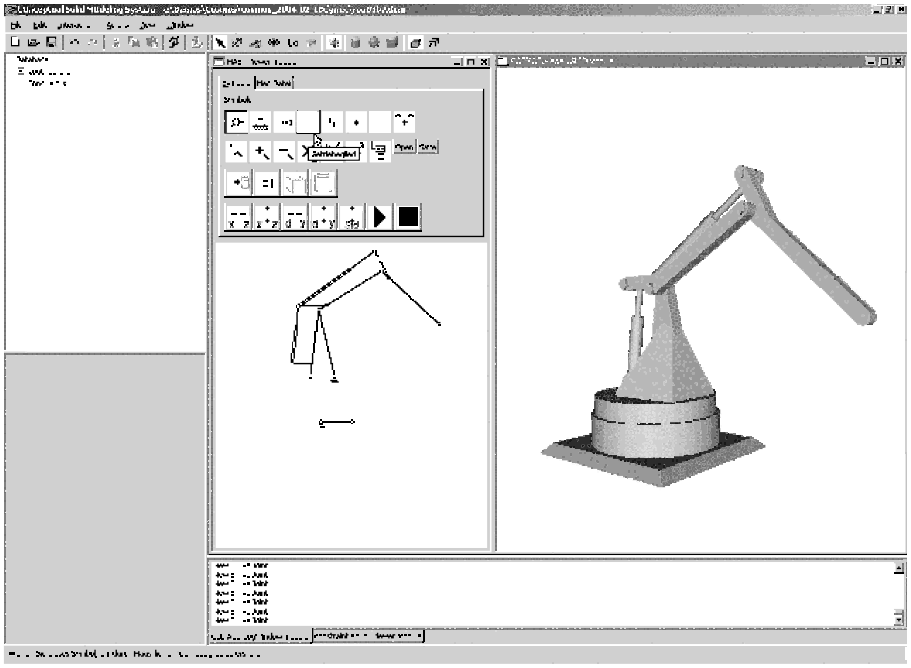


Abbildung 5.12: Modellierung eines Roboterarms. Die verwendeten Mechanismen sind planar und können somit durch MASP bearbeitet werden. Die Visualisierung erfolgt in *3D*.

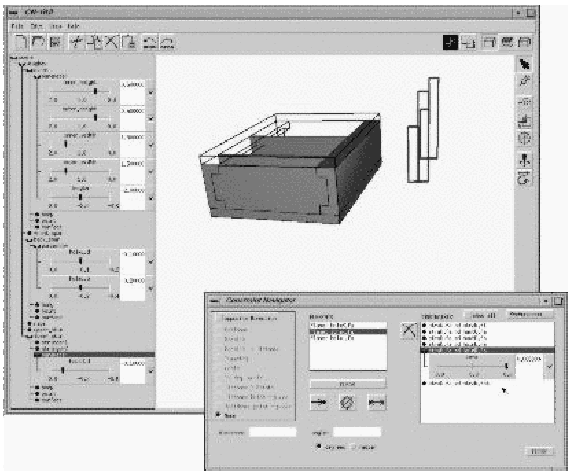


Abbildung 5.13: Das Tragflügelbox-Modell in COSMOS

5.3 3D-Geometrie

Auch das *3D*-Constraint-Modul ist im Laufe der Jahre in den verschiedensten Applikationen bzw. Projekten zum Einsatz gekommen und wurde dabei ständig weiterentwickelt. Die wichtigsten Anwendungen sind:

- COSMOS, ein 3D-Modellierwerkzeug,
- KONNI, ein Projekt in dem die constraint-basierte Modellierung von Architekturmodellen bearbeitet wurde,
- KfzKonzept, eine Applikation, die eine mögliche Anwendung von Constraints zur Modellierung der konzeptuellen Gestalt von Kraftfahrzeugen demonstriert,
- SphereMech, eine Testapplikation anhand der die Modellierung von sphärischen Mechanismen und Getrieben untersucht wurde und
- T3D-Mech, ein Modellierwerkzeug für räumliche Mechanismen und Getriebe, das auch zur Modellierung sphärischer Modelle verwendet werden kann.

5.3.1 COSMOS

Aufbauend auf den Arbeiten von Hsu wurde die Entwicklung des *3D*-Constraint-Moduls kontinuierlich vorangetrieben. In COSMOS, einem Werkzeug für das konzeptuelle Modellieren, fand das Modul seine bisher wichtigste Anwendung. Diverse Veröffentlichungen dokumentieren die Entstehung von COSMOS, u. a. [DMB98, BDKM00, BBDO00, BBDO02].

Im Rahmen der Entwicklung von COSMOS erfolgte die Schaffung von Steuerungsmöglichkeiten wie die typisierte Priorisierung von Freiheitsgraden und die Definition spezieller Freiheitsgradtypen. Ausführlichere Anwendungsbeispiele sind im Abschnitt 5.2.3 sowie im Abschnitt 5.3.2 gegeben.

Beispiel 5.11: Die Tragflügelbox, Modellierung in COSMOS

Eine Tragflügelbox (engl. Wingbox) ist das konzeptuelle Modell der tragenden Konstruktion eines Flugzeugflügels. Aus Sicht der constraint-basierten Modellierung ist sehr interessant, dass bei der Dimensionierung der Tragflügelbox physikalische Randbedingungen eine große Rolle spielen. Dementsprechend wurden Parameter in das Modell eingeführt, wie z. B. die Kraft, die auf den gesamten Tragflügel wirkt, oder Kräfte,

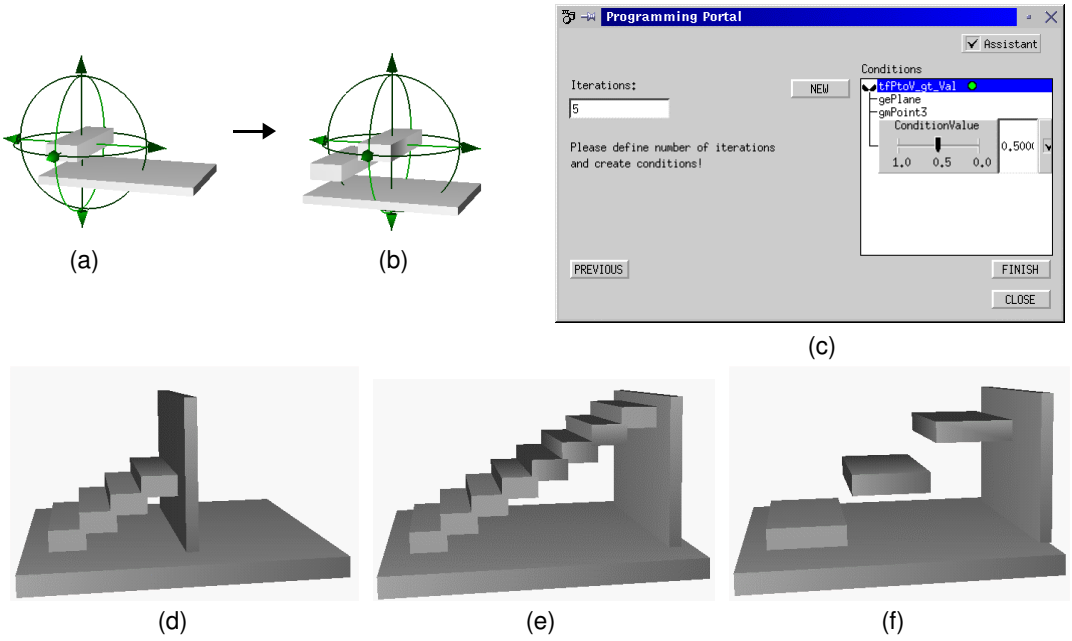


Abbildung 5.14: Assistentenbasierte Modellierung einer Treppe als rekursive Anordnung von einzelnen Stufen

die in ausgewählten Strukturelementen auftreten. Gleichungen verknüpfen diese physikalischen Parameter über Konstanten mit geometrischen Parametern, wie z. B. mit dem Abstand zwischen den versteifenden Elementen, mit ihrer Höhe usw. Zur Visualisierung der auftretenden Kräfte wurden neben ausgewählten Strukturelementen der Tragflügelbox auch quaderförmige Balken modelliert, deren Höhe die aktuelle und die kritische Kraft für die entsprechenden Elemente anzeigt. Diese Kräfte sind über die Constraints abhängig von der äußeren Kraft sowie der aktuellen Geometrie. Da bei einem Costraint-System der Rechenweg umkehrbar ist, lässt sich aber auch für eine bestimmte Geometrie die zulässige Gesamtkraft abschätzen oder auch eine Geometrie ermitteln, bei der vorgegebene Kraftverhältnisse auftreten. □

Bereits in Abschnitt 2.3 wurde darauf hingewiesen, dass Ficucs nur wertebestimmende Constraints behandeln kann. In COSMOS ist deshalb das Konzept der *programmierbaren Features* implementiert worden, wodurch sowohl die Objekt- als auch die Constraint-Menge eines Modells nicht von vornherein festgelegt sind, vielmehr werden die beiden Mengen während der Lösungsfindung ermittelt [BBDO00, BBDO02]. Ein ähnliches Konzept wird durch Rosendahl und Berling in [RB98] vorgestellt, wobei die dort erörterten Segmente den Templates der programmierbaren Features entsprechen.

Beispiel 5.12: Assistenten-basierte Modellierung in COSMOS

Abbildung 5.14 zeigt die assistentenbasierte Modellierung einer Treppe in COSMOS. (a) zeigt die Definition der ersten Stufe und (b) die anschließende Erzeugung eines Duplikates sowie seine Verschiebung. Mittels Constraints wird das Duplikat in der gewünschten Position gehalten. Es folgt in (c) die Definition von Abbruchbedingungen für die Rekursion (hier in Form einer Maximalanzahl oder das Erreichen einer Referenzebene). (d) und (e) zeigen die automatische Erzeugung von Stufen, wenn die Referenzebene verschoben wird. In (f) wurden die Parameter für die Stufenbreite sowie die Höhendifferenz zwischen zwei Stufen geändert. Es wäre theoretisch möglich, dieses einfache Beispiel mit einem *1D*-Solver zu berechnen. Durch den Einsatz von Ficucs stellt aber auch die Modellierung von Wendeltreppen (siehe Abbildung 3.21) o. Ä. mittels assistentenbasierter Modellierung kein Problem dar. □

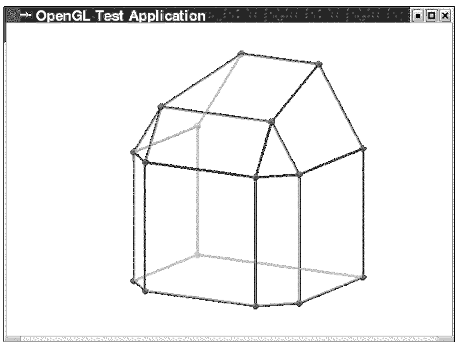


Abbildung 5.15: Modell eines Wintergartens

5.3.2 Architekturmodelle

Im KONNI-Projekt ist der Constraint-Solver zur Berechnung von Architekturmodellen eingesetzt worden. Hervorzuheben ist hierbei, dass es sich bei den einzelnen geometrischen Objekten nicht nur um achsenparallele Quader handelt. Vielmehr können alle Ebenen beliebig im Raum positioniert werden (siehe Abbildung 5.15). Im Allgemeinen gibt es auch Einsatzfälle, in denen z. B. Architektur- oder auch Inneneinrichtungsmodelle (die letztlich dreidimensionale Räume beschreiben) durch einen simplen *1D*-Constraint-Solver berechnet werden können. Bei derartigen Modellen handelt es sich typischerweise um Mengen quaderartiger Objekte, die achsenparallel ausgerichtet sind, sowie Constraints, die diese Objekte in x-, y- oder z-Richtung zueinander in Beziehung setzen. Da ein Constraint immer nur bezüglich einer Koordinatenachse wirkt, muss ein entsprechender Constraint-Solver nur die Auswertung linearer Gleichungen bzw. Ungleichungen beherrschen. *ORANOS*, ein Vertreter dieser Art von Constraint-Solvern, sowie seine Anwendung auf Inneneinrichtungsmodelle sind in [KGC97] beschrieben.

Im Speziellen handelt es sich bei den hier betrachteten Modellen um Wintergärten. Die verwendeten konzeptuellen Modelle sind jedoch oft eins zu eins auch auf Wohn-, Geschäfts- oder Industriebauten übertragbar. Die Modelle sind durch eine Parametermenge beschrieben, die typischerweise redundant ist. Dies hat zur Folge, dass sich bei Änderung eines Parameters auch andere Parameter ändern müssen. Wie das geschehen soll, wird von einzelnen Anbietern[28] sehr unterschiedlich gewünscht. Statt nun für die Änderung jedes Parameters von jedem Modell eines jeden Anbieters eine Berechnungsroutine in einer bestimmten Programmiersprache zu implementieren ist es sinnvoll, die Modelle constraint-basiert zu beschreiben. Die entsprechenden Berechnungsabläufe werden dann jeweils automatisch durch den Constraint-Solver ermittelt. Die Wünsche der Anbieter hinsichtlich der Auswirkungen von Parameteränderungen auf andere Parameter lassen sich sehr gut mittels typbasierter Priorisierung beschreiben.

Beispiel 5.13: Interaktive Änderungen an einem Wintergartenmodell
Gegeben sei das in Abbildung 5.16 gezeigte konzeptuelle Modell eines Wintergartens. Es wird wie in (a) gezeigt durch fünf Parameter beschrieben, wobei von den vier Parametern T_g, H_v, H_h und N_d jeweils nur drei frei gewählt werden können, der vierte ist in Abhängigkeit von den anderen dreien zu berechnen. Welcher Parameter berechnet wird, hängt davon ab,

- welcher Parameter vom Nutzer interaktiv geändert (d. h. vorgegeben) wird, dieser Parameter hat automatisch die höchste Priorität[29],
- ob Fixierungen vorliegen, denn fixierte Parameter können nicht berechnet werden und

[28] Gemeint sind hier Anbieter von zu berechnenden/auszulegenden Produkten. Die Einschätzung der erwünschten Interaktionsmöglichkeiten und der Erwartungshaltungen von potentiellen Käufern erfolgt durch die Anbieter oft sehr unterschiedlich.

[29] In Abschnitt 5.2.3 ist ein Beispiel gegeben, wo die Priorität eines interaktiv geänderten geometrischen Objektes nicht stets die höchste Priorität aufweist.

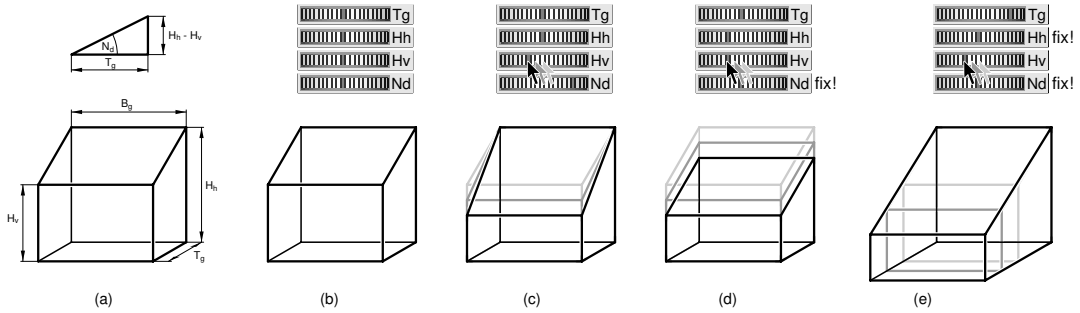


Abbildung 5.16: Beschreibung eines einfachen Wintergartenmodells mit seinen Parametern in (a) und (b) sowie der interaktiven Verkleinerung des Parameters H_v in (c) bis (e).

- welche Priorisierungsreihenfolge im Modell festgelegt wurde.

Durch einen Hersteller sei folgende Priorisierungsreihenfolge der Parameter festgelegt worden: B_g, T_g, H_v, H_h und N_d, d. h. dass B_g die höchste Priorität besitzt und N_d die kleinste. Punkte und Ebenen seien noch niedriger als diese Parameter priorisiert. Zudem sei zunächst keiner der Parameter fixiert. Änderungen von Parametern führen deshalb stets zu einer Änderung der Dachneigung N_d (siehe Abbildung 5.16(c)), lediglich die Änderung der Breite B_g hat keine Auswirkung, da keine Abhängigkeit zwischen B_g und N_d besteht. Wünscht der Nutzer, dass eine bestimmte Neigung beibehalten wird, so kann er den entsprechenden Parameter N_d fixieren. Die Freiheitsgradverteilung erfolgt nun derart, dass z. B. die Änderung der vorderen Höhe H_v eine Änderung der hinteren Höhe H_h verursacht, siehe Abbildung 5.16(c). Der Nutzer kann aber auch H_h fixieren. In diesem Fall ändert sich die Tiefe T_g des Modells in Abhängigkeit vom gewählten H_v sowie den fixierten Parametern H_h und N_d, siehe Abbildung 5.16(e). Erst wenn auch der Parameter für die Tiefe fixiert wird, kann keine Freiheitsgradverteilung gefunden werden und die Änderung von H_v wird abgewiesen.
Hätte nun ein Hersteller eine andere Priorisierung gewählt, z. B. B_g, T_g, N_d, H_v und H_h, so würde eine Änderung der vorderen Höhe zunächst die (nun höher priorisierte) Dachneigung konstant lassen und sich auf die hintere Höhe auswirken. Dennoch hätte der Nutzer auch hier die Möglichkeit, über Fixierungen das Verhalten des Modells an seine Vorstellungen anzupassen. □

Wie im obigen Beispiel gezeigt wurde, werden dem Nutzer nicht bestimmte Berechnungs- bzw. Änderungsabläufe aufgezwungen. Zwar gibt es bevorzugte Parameteränderungen, jedoch hat der Nutzer stets die Möglichkeit seine Vorstellungen über die Auswirkungen von Parameteränderungen der Applikation mitzuteilen. Dies ist für die Nutzerakzeptanz von großer Bedeutung.

5.3.3 Konzeptuelle Kfz-Modelle

Ein weiteres Beispiel dafür, wie Constraints zur Modifikation von Modellen hoher Abstraktheit eingesetzt werden können, stammt aus dem Automobilbau. Aufbauend auf COSMOS wurde ein Demonstrator implementiert, der zeigt, wie sich konzeptuelle Kfz-Modelle constraint-basiert modellieren und modifizieren lassen.

Einsatzfall

Die prinzipielle Form einer Fahrzeugkarosserie wird über sogenannte Einflusspunkte festgelegt. Die Grundidee ist, dass bereits die Lage der Einflusspunkte wesentliche Eigenschaften eines Fahrzeuges bestimmt[30]. Diese Einflusspunkte werden in Gruppen zusammengefasst, z. B. Dach, Frontscheibe,

[30] Anhand der Einflusspunkte können einige Eigenschaften in Form von Kennwerten abgeschätzt werden. Andere Eigenschaften lassen sich z. B. durch Simulationsrechnungen für die sich ergebende Karosserie ermitteln. Die

Motorhaube usw. Damit wird dem Nutzer die Möglichkeit gegeben, Mengen von Punkten gleichzeitig zu manipulieren. Die Manipulation der Gruppe könnte stets so auf die Punkte wirken, dass sich deren relative Positionen in der Gruppe nicht ändern. Denkbare Manipulationen sind u.a.:

- die Verschiebung der Dachpunkte nach unten,
- das Drehen der Frontscheibe, so dass sie windschnittiger wird, und
- das Hochziehen des Kofferraumes.

Die beschriebene Funktionalität lässt sich recht einfach auch ohne Constraints implementieren. Dadurch, dass Punkte typischerweise in mehreren Gruppen enthalten sind, würde es jedoch bei Manipulation der einen Gruppe zu relativen Verschiebungen der Punkte innerhalb einer anderen Gruppe kommen, was aufwändige Nachpositionierungen der einzelnen Punkte zur Folge hätte.

Constraint-basierte Modellierung

Zur Vermeidung von derartigen Nachpositionierungen und für eine bessere Steuerbarkeit der Änderungen ist es möglich, Einflusspunkte und ihre Gruppierungen als Constraint-Netz zu modellieren.

- Jede Gruppe ist hierbei als lokales Koordinatensystem definiert, welches aus drei rechtwinklig zueinander stehenden Ebenen aufgebaut ist.
- Lokale Koordinaten werden als Abstände der Punkte zu den Koordinatenebenen modelliert. Die entsprechenden Abstandsparameter werden in das Modell eingeführt.
- Relative Positionen in der Gruppe können dadurch modelliert werden, dass Skalierungsfaktoren eingeführt werden (für die drei Hauptrichtungen der Gruppe) und ausgewählte Parameter (typischerweise solche, die lokale Koordinaten darstellen) von einer linearen Gleichung abhängig gemacht werden (z. B. $x_1 = s_x * x_{1R}$, wobei x_1 die lokale x-Koordinate ist, s_x der Skalierungsfaktor in der lokalen x-Richtung und x_{1R} die für x_1 definierte relative lokale x-Position).
- Je nach Ziel der Manipulation werden den Objekten im Modell andere Prioritäten zugeordnet, z. B.:
  - niedrigste Priorität für die Abstandsparameter (z. B. das oben erwähnte x_{1R}) bei Verschiebung innerhalb der Gruppen,
  - höchste Priorität, wenn die Gruppen starr bleiben sollen (auch die Skalierungsfaktoren sind hoch priorisiert oder fix) bzw. lediglich skaliert werden sollen (die Skalierungsfaktoren haben eine geringe Priorität).
  - Sinnvoll ist auch die Vermeidung von Rotationen durch hohe Priorisierung der Normalen. Erst wenn eine Ebene explizit durch den Nutzer rotiert wird (oder Fixierungen von Punkten Rotationen erzwingen) ändert sich die entsprechende Normale.

Für Änderungen am Modell, das typischerweise stark unterbestimmt ist, wird zur Steuerung der Berechnungen zunächst ein Modus gewählt, wobei bestimmten Modellteilen (hier Gruppen oder einzelnen Punkten) Eigenschaften zugeordnet sein können (z. B. fix, streckbar oder rotierbar). Die Applikation übermittelt diese Eigenschaften dem Constraint-Solver, der dann das Modell entsprechend den vom Nutzer getätigten Modifikationen und unter Beachtung der gewünschten Auswirkungen jeweils neu berechnet.

In den Abbildungen 5.17, 5.18 und 5.19 sind Beispiele für die Arbeit an den konzeptuellen Kfz-Modellen dargestellt.

Erzeugung der Karosserie erfolgt hierfür automatisch unter Zuhilfenahme von zugeordneten Profilen in einer Art history-basiertem Ansatz.

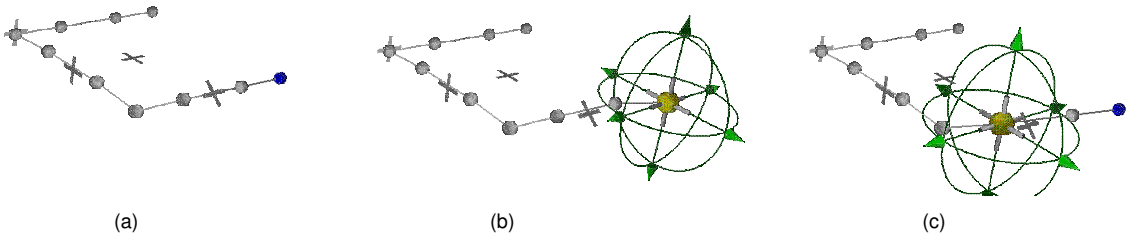


Abbildung 5.17: Konzeptuelle Modellierung eines Kfz-Daches. (a) Zehn Punkte sind in der Dach-Gruppe vereinigt. Die dargestellten Kreuze markieren Referenzebenen. (b) Ein Punkt wird interaktiv in der Gruppe verschoben, seine relativen Koordinaten in den lokalen Koordinatensystemen haben niedrigste Priorität. (c) Auch die anderen Punkte der Vorderkante des Daches werden an eine gewünschte Position verschoben.

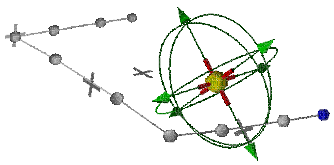


Abbildung 5.18: Konzeptuelle Modellierung eines Kfz-Daches. Die Verschiebung der Frontscheibengruppe (zu der auch die vier vorderen Punkte der Dachgruppe gehören) führt zu einer Streckung der Dachgruppe. Die Verschiebung erfolgt durch Bewegen der entsprechenden Koordinatenebene.

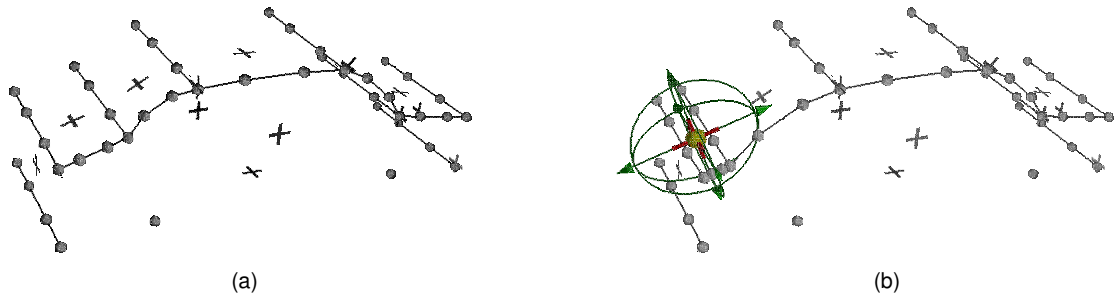


Abbildung 5.19: Konzeptuelle Modellierung eines Kfzs. (a) Der Ausgangszustand in Form eines Halbautos, wie es für derartige Modelle üblich ist. (b) Das Absenken der Haube am Heck hat (aufgrund der konstanten Neigungen und Positionen der Heckscheibe und des Hecks) eine Verkleinerung der Heckhaube zur Folge. Dabei werden die Punkte in der entsprechenden seitlichen Gruppe skaliert, d.h. näher zusammengerückt. Die Seite der Heckscheibe wird hingegen gestreckt.

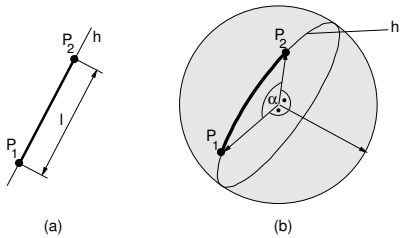


Abbildung 5.20: Dualität der constraint-basierten Modelle eines Getriebegliedes. (a) in der Ebene und (b) auf einer Kugel.

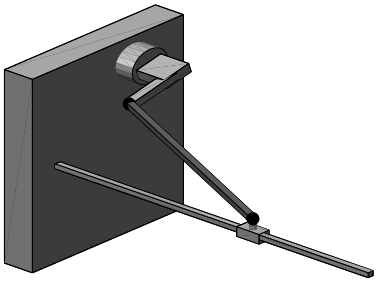


Abbildung 5.21: Ein einfaches räumliches Getriebe.

5.3.4 Sphärische Getriebe

Für die Modellierung von Prinzipsymbolen auf einer Kugel sind die Möglichkeiten von Constraints, die für die Ebene definiert sind, nicht ausreichend. Statt dessen müssen *3D*-Objekte und entsprechende Constraints definiert werden. Hierbei ergibt sich eine interessante Dualität zwischen planaren und sphärischen Modellen. Abbildung 5.20 zeigt diese Dualität am Beispiel eines Getriebegliedes. Beide Modelle nutzen zwei Punkte P_1 und P_2, eine Hauptachse bzw. einen Großkreis h, einen Längenparameter l bzw. Winkelparameter α, zwei Constraints, um die Punkte P_1 und P_2 auf der Hauptachse bzw. dem Großkreis zu halten, und einen Constraint, um den Abstand l bzw. α der Punkte P_1 und P_2 zueinander sicherzustellen. In der Tabelle 5.2 sind einige Aspekte der constraint-basierten Modellierung von sphärischen Getrieben aufgelistet.

5.3.5 Räumliche Getriebe

Die planaren und sphärischen Getriebe der vorhergehenden Abschnitte lassen sich jeweils auf ein *2D*-Koordinatensystem (Ebene bzw. Kugel) abbilden, auch wenn sehr anschauliche *räumliche Darstellungen* der Modelle erzeugt werden. Ist ein Getriebe (oder zumindest ein Teil von ihm) auf die euklidische Ebene abbildbar, so lässt sich grundsätzlich das *2D*-Modul des Constraint-Solvers für die entsprechenden Berechnungen einbinden, wie es auch für die in den Abbildungen 5.11 und 5.12 gezeigten Modelle geschehen ist. Für räumliche Getriebe, die eben nicht nur räumliche Darstellungen ebener Getriebe sind, muss hingegen stets das *3D*-Modul genutzt werden.

Im Rahmen der vorliegenden Arbeit konnten erste einfache Beispiele modelliert werden. Abbildung 5.21 zeigt die Kopplung eines rotatorischen Antriebes mit einem Schubgelenk, wobei die durch die Drehbewegung aufgespannte Ebene und die Gerade, auf der sich das Schubgelenk bewegt, nicht parallel zueinander sind, denn ansonsten wäre die Abbildung auf eine Ebene möglich gewesen.

Tabelle 5.2: Modellierung einiger Prinzipbestandteile für sphärische Getriebe

Prinzipbestandteile und ihre Verknüpfung	Constraint-basierte Repräsentation
Parameter	Parameterwerte werden durch Gleitkomma-Zahlen (wie in *2D*-Modellen) beschrieben. Abstände zwischen Punkten werden durch Angabe einer Bogenlänge oder eines Winkels modelliert, da beide Repräsentationen bei bekanntem Radius äquivalent sind.
Punkt	Jeder Punkt wird durch die Spitze eines Ortsvektors, der vom Kugelmittelpunkt ausgeht, repräsentiert. Für technische Prinzipe, bei denen Kollisionen nicht von Interesse sind, entsprechen die Beträge aller Vektoren zur Definition von Punktpositionen dem Kugelradius.
Gerade	Eine Gerade ist ein Großkreis der Kugel. Dadurch entspricht die kürzeste Verbindung zweier Punkte auf einer Geraden stets einer Strecke auf dieser Geraden. Eine Gerade wird durch den Normalenvektor des Großkreises repräsentiert. Die Richtung der Geraden ist invertierbar.
Kreis	Ein Kreis kann durch seinen Normalenvektor (geht durch seinen Mittelpunkt, wenn er als Ortsvektor im Kugelmittelpunkt angetragen wird) und einen Radiusparameter (z. B. ein Winkel, s. o.) dargestellt werden.
Punkt auf Gerade	Ein „Punkt auf Gerade“-Constraint wird durch einen Rechtwinkligkeits-Constraint zwischen Vektoren repräsentiert. Jeder Punkt auf dem Großkreis besitzt einen Vektor, der senkrecht zur Normalen des Großkreises ausgerichtet ist. Siehe auch Abbildung 5.20.
Drehgelenk	Ein Drehgelenk wird durch einen Punkt abstrahiert und lässt sich deshalb wie dieser als Vektor modellieren (s. o.).
Getriebeglied	Ein Getriebeglied (siehe Abbildung 5.20b) wird beschrieben durch: • eine Hauptachse (eine Gerade, repräsentiert durch einen Vektor), • zwei Punkte (Anfangs- und Endpunkt, repräsentiert durch je einen Vektor), • zwei Rechtwinkligkeits-Constraints, um die Punkte auf der Geraden zu halten, • einen Winkel- bzw. Längen-Parameter für die Getriebegliedlänge sowie • einen Winkel-Constraint (zwischen den Vektoren, die den Anfangs- und Endpunkt repräsentieren) um die Getriebegliedlänge sicherzustellen.
Verknüpfung	Zum Verbinden des Anfangs- oder Endpunktes eines Getriebegliedes mit einem Drehgelenk dient ein Gleichheits-Constraint zwischen den entsprechenden Vektoren.

Kapitel 6

Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurden unterschiedliche Sichtweisen auf das Thema geometrische Constraints im Allgemeinen und Ficucs sowie sein Einsatz in diversen Applikationen im Besonderen dargestellt. Neben einer Erörterung der Grundlagen für den besprochenen konstruktiv-iterativen Ansatz erfolgte eine detaillierte Diskussion der in Ficucs umgesetzten Konzepte sowie ihres Zusammenspiels. Schwerpunkte bildeten hierbei die Freiheitsgradanalyse, die Umsetzung der univariaten Nullstellensuche sowie die Behandlung von Mehrfachlösungen. In Vergleichen mit Clustering-Ansätzen und numerischen Methoden wurden Stärken und Schwächen des vorgestellten Constraint-Solvers deutlich gemacht. Durch eine umfassende Darstellung des Einsatzes von Ficucs in unterschiedlichsten Applikationen konnte schließlich sein breites Einsatzspektrum unter Beweis gestellt werden.

Zur besseren Nachvollziehbarkeit der Ausführungen im Hauptteil dienen einige Gleichungssysteme, Modelle und Pläne, die beispielhaft in den Anhang aufgenommen wurden. Für ein besseres Verständnis des Lesers, wie das Zusammenspiel der einzelnen Algorithmen beim Constraint-Solving erfolgt, sind zudem durchgängige Beispiele für direkte und indirekte Konstruktion im Anhang enthalten. Weitere ausgewählte Beispiele zur Problematik der Mehrfachlösungen können als Anregungen für Tests von Geometrie- oder Kinematikberechnungssoftware dienen. Eine Betrachtung verschiedenster in der Literatur über geometrische Constraints beschriebener numerischer Ansätze zur Lösungsfindung bildet den Abschluss der Arbeit. Hier werden die einzelnen Verfahren verglichen und Gründe für die im Hauptteil beschriebenen Probleme näher erörtert.

In der Zukunft werden die Schwerpunkte der Forschungs- und Entwicklungsarbeiten an Ficucs hauptsächlich auf einer robusten multivariaten Suche, einer verbesserten Behandlung degenerierter Fälle und einer verbesserten Diagnosefähigkeit liegen. Für die multivariate Suche bietet sich die Integration des Ansatzes nach Färber an. Dieser Ansatz ist grundsätzlich allgemein einsetzbar, berücksichtigt aber auch speziell die Belange des Lösens geometrischer Constraints. Für die bessere Behandlung degenerierter Fälle ist das Zusammenspiel von Constraint-Netztransformation, Freiheitsgradanalyse und Planerstellung sowie Planberechnung entsprechend den Ausführungen im Hauptteil zu überarbeiten. Die Erweiterung der Anzahl der unterstützten Konstruktionsoperationen für geometrische Objekte sowie deren Fähigkeiten bezüglich der Berücksichtigung von Hinweisen sowie der Behandlung von Spezialfällen bildet eine wichtige Ausgangsbasis für die Verbreiterung der Anwendungsfelder des Constraint-Solvers.

Ein weiteres Ziel könnte die Integration eines komplexeren Clustering-Ansatzes darstellen. Hierbei sollen die Cluster aber eher im Sinne von Hilfskonstruktionen eingeführt werden (geclusterte Objekte werden mehrfach instanziiert), denn man kann nicht davon ausgehen, dass Cluster stets starr sind. Besser ist es, sie als Hilfskonstruktionen zu betrachten, aus denen sich bestimmte Werte ablesen lassen, z. B. zur optimalen Bestimmung der Startwerte für iterative Konstruktionen.

In manchen Anwendungen ist es nicht sinnvoll oder sogar unmöglich sämtliche Modellierungsaspekte in Ficucs berechnen zu lassen. Hier kann die Kombination des Constraint-Solvers mit verschiedenartigen Berechnungsmodulen innerhalb einer Blackboard-Architektur helfen. Eine Fortführung der laufenden Forschungs- und Entwicklungsarbeiten zum Einsatz von geometrischen Constraints im Rahmen des Entwurfs heterogener Systeme verspricht weitere Erkenntnisse bezüglich einer derartigen Synchronisation verschiedener Solver bzw. Berechnungsmodule. Für MASP wird die Weiterentwicklung in Richtung sphärische und räumliche Getriebe erfolgen. In Hinblick auf die Entwicklung von Layout-Applikationen für Architekturmodelle oder einfache Bauteile (polyedrische Modelle) sowie die jeweilige constraint-basierte Modellierung wäre zudem die automatische Generierung von geometrischen und topologischen Constraints interessant.

Im Rahmen des DMG-Lib-Projektes besteht die Aufgabe, Mechanismenbeschreibungen in verschiedenste Formate zu exportieren. Neben den deklarativen Modellbeschreibungen sind die vom Constraint-Solver erstellten Berechnungspläne eine wichtige Basis für den Getriebemodellexport. Erweiterte und besser steuerbare Exportmöglichkeiten für die Pläne erlauben eine bessere Integration des Constraint-Solvers in Applikationen, die aus constraint-basierten Modellen Berechnungsbeschreibungen (kompilierbare Programme, Skripte, spezielle Beschreibungsformate) erzeugen. In diesem Zusammenhang ist auch eine Bereitstellung der Konstruktoren in separaten Berechnungsbibliotheken sinnvoll, denn so können von anderen Applikationen Modellberechnungen (basierend auf den exportieren Plänen) durchgeführt werden.

Anhang A

Vertiefende Beispiele

A.1 Gleichungen

Gleichungssystem 5.2

Beschreibungsdatei für das Gleichungssystem 5.2 auf Seite 125:

```
# initiale Werte
arg a -6
arg b  8
arg c  6
arg d -3

# Gleichung 1
arg  h11
LnEq G11 h11 = a       + b
arg  h12
LnEq G12 h12 = 3   * c + h11
LnEq G13 6   = h12     + d

# Gleichung 2
arg  h21
LnEq G21 h21 = -4  * b + c
LnEq G23 -17 = -1  * d + h21

# Gleichung 3
LnEq G31 11  = 3   * d + c


# Gleichung 4
LnEq G41 28  = 7   * d


# Start der Berechnungen
Calc
```

Plan zur Lösung des Gleichungssystems 5.2 auf Seite 125:

```
ev_arg_lnEq(   d )   d   = (28 - 0) / 7
ev_arg_lnEq( h12 )   h12 = ( 6 - d) / 1
ev_arg_lnEq( h21 )   h21 = -17 - -1 * d
ev_arg_lnEq(   c )   c   =  11 - 3 * d
ev_arg_lnEq( h11 )   h11 = h12 - 3 * c
ev_arg_lnEq(   b )   b   = (h21 - c) / -4
ev_arg_lnEq(   a )   a   = (h11 - b) / 1
```

Gleichungen 5.3 bis 5.8

Beschreibungsdatei für die Gleichungen 5.3 bis 5.8:

```
# initiale Werte
arg S
arg P
arg I
arg K
arg E 1
arg F 1
arg M
arg y
arg l 1
arg h 0.1
arg b 0.1

# Festlegung der Fixierung. Nicht fixierte Werte werden berechnet
Fix F
Fix l
Fix h
Fix b
Fix E

# Gleichung 1
arg  lS
LnEq G11   lS = l * S
LnEq G12   lS = M * y

# Gleichung 2
LnEq G21    M = F * l

# Gleichung 3
arg  h2
LnEq G31   h2 = h * h
arg  h3
LnEq G32   h3 = h * h2
arg  I12
LnEq G33  I12 = I * 12
LnEq G34  I12 = b * h3

# Gleichung 4
LnEq G41    h = y * 2

# Gleichung 5
arg  l2
LnEq G42   l2 = l * l
arg  l3
LnEq G43   l3 = l * l2
arg  EI
LnEq G44   EI = E * I
arg  l3K
LnEq G45  l3K = l3 * K
LnEq G46  l3K = EI * 3

# Gleichung 6
arg  P2
LnEq G61   P2 = P * 2
arg  P2EI
LnEq G62 P2EI = P2 * EI
LnEq G63 P2EI = l2 * F

# Start der Berechnungen
Calc
```

Plan zur Lösung der Gleichungen 5.3 bis 5.8:

```
ev_arg_lnEq(    M )   M    = F * l + 0
ev_arg_lnEq(    y )   y    = (h - 0) / 2
ev_arg_lnEq(   lS )   lS   = M * y + 0
ev_arg_lnEq(    S )   S    = (lS - 0) / l
ev_arg_lnEq(   h2 )   h2   = h * h + 0
ev_arg_lnEq(   h3 )   h3   = h * h2 + 0
ev_arg_lnEq(  I12 )   I12  = b * h3 + 0
ev_arg_lnEq(    I )   I    = (I12 - 0) / 12
ev_arg_lnEq(   EI )   EI   = E * I + 0
```

```
ev_arg_lnEq(  l3K )   l3K  = 3 * EI + 0
ev_arg_lnEq(   l2 )   l2   = l * l + 0
ev_arg_lnEq(   l3 )   l3   = l * l2 + 0
ev_arg_lnEq( P2EI )   P2EI = F * l2 + 0
ev_arg_lnEq(    K )   K    = (l3K - 0) / l3
ev_arg_lnEq(   P2 )   P2   = (P2EI - 0) / EI
ev_arg_lnEq(    P )   P    = (P2 - 0) / 2
```

Gleichungssystem 5.1

Beschreibungsdatei für das Gleichungssystem 5.1 auf Seite 125:

```
# initiale Werte
arg a  1
arg b -1
arg c  3
arg d  2

# Gleichung 1
arg  h11
LnEq G11 h11 = 3   * a + b
arg  h12
LnEq G12 h12 = 2   * c + h11
LnEq G13 11  = h12     + d

# Gleichung 2
arg  h21
LnEq G21 h21 = 2   * b + a
arg  h22
LnEq G22 h22 = c       + h21
LnEq G23 19  = 3   * d + h22

# Gleichung 3
arg  h31
LnEq G31 h31 = a       + b
arg  h32
LnEq G32 h32 = 3   * c + h31
LnEq G33 6   = h32     + d

# Gleichung 4
arg  h41
LnEq G41 h41 = 2   * a + c
arg  h42
LnEq G42 h42 = 2   * b + h41
LnEq G43 13  = h42     + d

# Start der Berechnungen
Calc
```

Plan zur Lösung des Gleichungssystems 5.1 auf Seite 125:

```
gs_arg( a )

    gs_arg( b )

        ev_arg_lnEq( h11 ) h11 = 3 * a + b
        ev_arg_lnEq( h21 ) h21 = 2 * b + a
        ev_arg_lnEq( h31 ) h31 = 1 * a + b

        gs_arg( c )

            ev_arg_lnEq( h12 ) h12 =  2 * c + h11
            ev_arg_lnEq( h22 ) h22 =  1 * c + h21
            ev_arg_lnEq(   d )   d = 11 - 1 * h12

        ms_lnEq( 19 = 3 * d + h22 )

        ev_arg_lnEq( h32 ) h32 = 3 * c + h31

    ms_lnEq( 6 = h32 + d )

    ev_arg_lnEq( h41 ) h41 = 2 * a + c
    ev_arg_lnEq( h42 ) h42 = 2 * b + h41

ms_lnEq( 13 = h42 + d )
```

A.2 Modelle aus 2D

Kollisionserkennung

Die folgende Beschreibungsdatei zeigt die Kollisionserkennung bzw. -vermeidung zwischen einem Punkt und einem Getriebeglied auf Basis von Conditional Constraints. Das Modell verhindert, dass der Punkt P auf die jeweils andere Seite des Getriebegliedes G wechselt, falls P sich „neben“ dem Getriebeglied befindet. Der in den Bezeichnern für Ungleichungen und Hinweise verwendete Präfix `PG` erlaubt die gleichzeitige Modellierung mehrerer zu überwachender Paare von Punkten und Geraden (dort wird dann ein anderer Präfix verwendet).

Die Beziehung „neben“ wurde durch die Ungleichungen `P_nahe_G1` und `P_nahe_G2` modelliert, wobei eine Abstandsmessung und ein Vergleich mit der Getriebegliedlänge vorgenommen wird. Abbildung 5.3(b) zeigt die Form des dadurch entstehenden Bereiches, in dem angenommen wird, dass sich P „neben“ G befindet. Da die Ungleichungen nur als auszuwertende Bedingung dienen, müssen sie inaktiv sein (`CNInactive`), ansonsten würde der Constraint-Solver fälschlicherweise versuchen, beide Ungleichungen zu erfüllen.

Zur Feststellung der Seite, auf der sich P gerade befindet, werden der *LHS*-Hinweis `P_ist_links` und der *RHS*-Hinweis `P_ist_rechts` ausgewertet, vgl. 5.3(d). Zunächst sind beide Hinweise inaktiv, so dass sie vom Constraint-Solver bei der Herstellung der Konsistenz des Modells nicht berücksichtigt werden. Die Aktivierung der beiden *XHS*-Hinweise hängt jeweils von drei Bedingungen ab, zwei für die Kontrolle, ob Seitenwechsel verhindert werden sollen (`P_nahe_G1` und `P_nahe_G2` müssen wahr sein), sowie eine Bedingung die kontrolliert, ob P sich gerade links oder rechts von G befindet.

Eine gegenüber der originalen Beschreibungsdatei besser lesbare Notation der Kollisionserkennung sieht wie folgt aus:

```
P_nahe_G1   : der Abstand von P zu G1 ist kleiner als die Getriebegliedlänge
P_nahe_G2   : der Abstand von P zu G2 ist kleiner als die Getriebegliedlänge
P_ist_rechts: G1, P und G2 bilden ein RHS
P_ist_links : G1, P und G2 bilden ein LHS
P_neben_G   : P_nahe_G1 UND P_nahe_G2
WENN P_neben_G UND P_ist_rechts DANN aktiviere P_ist_rechts
WENN P_neben_G UND P_ist_links  DANN aktiviere P_ist_links
```

Treffen die jeweiligen Bedingungen nicht zu, dann bleiben `P_nahe_G1` bzw. `P_nahe_G2` inaktiv oder werden deaktiviert.

Die originale Beschreibungsdatei für MASP ist:

```
# Das urspruengliche Modell
Bar G 0.2 -0.3 0.7 0.5
Swj P 0.6 0.4

# Kollisionsvermeidung zwischen P und G
CNInactive Modifier < VtoV KV_PG.PnaheG1 G.pt1 P G.len
CNInactive Modifier < VtoV KV_PG.PnaheG2 G.pt2 P G.len
CNInactive XHS KV_PG.pRechts R G.pt1 P G.pt2
CNInactive XHS KV_PG.pLinks  L G.pt1 P G.pt2

CNIfCond KV_PG.PnaheG1 KV_PG.PnaheG2 KV_PG.pRechts CNThenCnss KV_PG.pRechts
CNIfCond KV_PG.PnaheG1 KV_PG.PnaheG2 KV_PG.pLinks  CNThenCnss KV_PG.pLinkss
```

A.3 Pläne zu bestimmten Modellen

Modell aus Abbildung 4.4

Der entsprechende Plan lautet:

```
gs2_pt_vtov ( A ) A0

    ev2_pt_dd    ( B             ) B0 A
    ev2_ln_ii    ( AB.axis       ) B  A
    ev2_pt_di    ( C             ) A  C.axis
    ev_chk       (  C_to_B       )
    ev_chk       (  C_to_A       )
    ev2_ln_ia    ( CP.axis       ) C  AB.axis.Dir
    ev2_pt_vtov  ( P             ) C
    ev_chk       ( RHS(A, B, P) )

ms2_vinl( P_in_CP.axis )

ev2_ln_ii    ( A0A.axis ) A A0
ev2_ln_ii    ( B0B.axis ) B0 B
```

Modell aus Beispiel 3.21 (Seite 88)

Der entsprechende Plan lautet:

```
gs2_pt_vinl      ( A       ) S

    ev_chk       ( A to S.pt2,  A to S.pt1 )
    ev2_pt_vtov  ( B       ) A
    ev_chk       ( RHS(S.pt1, S.pt2, B)    )
    ev2_ln_dirI  ( G       ) A G.D

ms2_vinl         ( B in G )
```

A.4 Durchgängige Beispiele zur Arbeitsweise von Ficucs

A.4.1 Viergliedriges Koppelgetriebe und dessen rechnerinterne Modellierung

Die Arbeitsweise von Ficucs wird in diesem Abschnitt an einem einfachen Mechanismus erläutert, der auf einer Viergelenkkette beruht (eine Kurbelschwinge, siehe Abbildung A.1). Das Modell ist in MASP erstellt worden. Dementsprechend wird das *2D*-Constraint-Modul für die Berechnungen eingesetzt. Eine iterative Konstruktion ist hierbei nicht nötig. Der in Abbildung A.1 gezeigte Mechanismus besitzt einen Freiheitsgrad, d. h., dass sich alle geometrischen Pole jeweils nur auf einer bestimmten Bahn bewegen.

Zur Ermittlung unterschiedlicher Lagen des Mechanismus muss die momentane Position eines beweglichen Teils bekannt sein. In Abbildung A.1 dient die gewünschte Position des Drehgelenks A zur Auswahl einer bestimmten Getriebestellung. Denkbar wäre aber auch die Vorgabe des Anstiegswinkels des Getriebegliedes A_0A. Da sich das Gelenk A nur auf einer Kreisbahn um das gestellfeste Drehgelenk A_0 bewegen kann, liefert der Constraint-Solver die Position, bei der der Abstand zwischen Gelenk A und der Mauszeigerposition minimal wird. Das Ergebnis der Berechnung ist in Abbildung A.1(b) zu sehen. Abbildung A.2(a) zeigt die Kurbelschwinge aus Abbildung A.1 mit ergänzenden Bezeichnungen. In Abbildung A.2(b) sind die verwendeten Symbole deutlicher zu erkennen. Das aus der symbolischen Beschreibung nutzerunabhängig entstehende constraint-basierte Modell ist in Abbildung A.3(a) zu sehen (vgl. auch Tabelle 5.1). Es liegt in dieser Form als rechnerinternes Modell vor und enthält redundante Informationen. Diese äußern sich in Abbildung A.3(a) durch das dreifache Vorhandensein der Gelenkpunkte A und B sowie

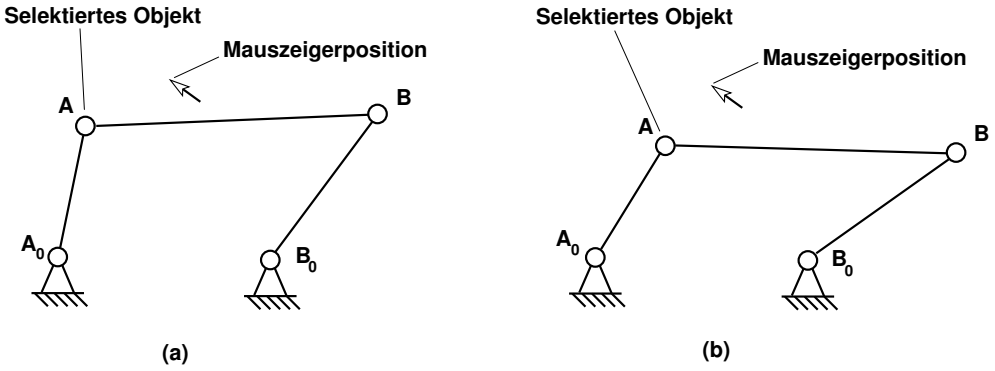


Abbildung A.1: Kurbelschwinge in unterschiedlichen Lagen. (a) zeigt die Stellung der Kurbelschwinge vor einer interaktiven Lageänderung des Gelenkes A in Richtung der Position des Mauszeigers und (b) das Ergebnis der interaktiven Lageänderung.

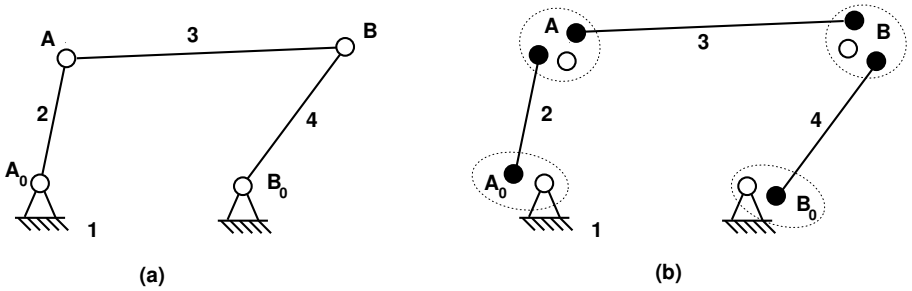


Abbildung A.2: Symbolische Beschreibung einer Kurbelschwinge im Entwurfssystem MASP. Es wird gezeigt (a) das Koppelgetriebe aus Nutzersicht, (b) die vorhandenen Einzelsymbole.

von vier Gleichheits-Constraints zwischen den Gelenkpunkten zur Gewährleistung der Modellkonsistenz. Diese Redundanz bietet Vorteile bei der Durchführung von Modellmodifikationen, wie z. B. beim Löschen und Einfügen. Für einen effizienten Ablauf der Analyse- und Berechnungsschritte im Constraint-Solver wird bei der Übertragung des constraint-basierten Modells an den Constraint-Solver eine Transformation zur Reduzierung von redundanten Constraints und geometrischen Objekten vorgenommen. Im Beispiel aus Abbildung A.3(a) werden die drei Gelenkpunkte für A und B jeweils durch einen einzigen Referenzpunkt ersetzt. Das reduzierte Modell ist in Abbildung A.3(b) veranschaulicht.

Die Verarbeitung der an den Constraint-Solver übergebenen Daten erfolgt in zwei Phasen. Zunächst wird eine Modellanalyse durchgeführt, die aus einer vorgeschalteten Freiheitsgradanalyse und dem Generieren des Konstruktionsplanes besteht. Dieser Plan enthält alle Konstruktionsschritte, die zur Berechnung der Geometrie des Modells benötigt werden. In der zweiten Phase erfolgt dann die eigentliche Berechnung mit konkreten Parameterwerten unter Nutzung des erstellten Konstruktionsplanes. Die Zweiteilung des Ablaufs hat zum Vorteil, dass für aufeinander folgende Änderungen, wie sie z. B. bei einer interaktiven Bewegungssimulation auftreten, kein neuer Konstruktionsplan ermittelt werden muss. Die Speicherung des in Abbildung A.3(b) gezeigten internen Modells erfolgt im Constraint-Solver als Graph (Abbildung A.4(a)). Objekte sind dort mit einem Kreis und Constraints mit einem Quadrat dargestellt. Für die Freiheitsgradanalyse wird daraus ein bipartiter - d. h. in zwei Knotenmengen geteilter - Graph generiert (Abbildung A.4(b)). Die eine Menge beschreibt die Objekte mit ihren Freiheitsgraden und die andere Menge beschreibt die Constraints mit ihren Valenzen. Diese „negativen Freiheitsgrade“ schränken den Wertebereich der Geometrieparameter ein. Verknüpfungen existieren nur zwischen Knoten aus unterschiedlichen

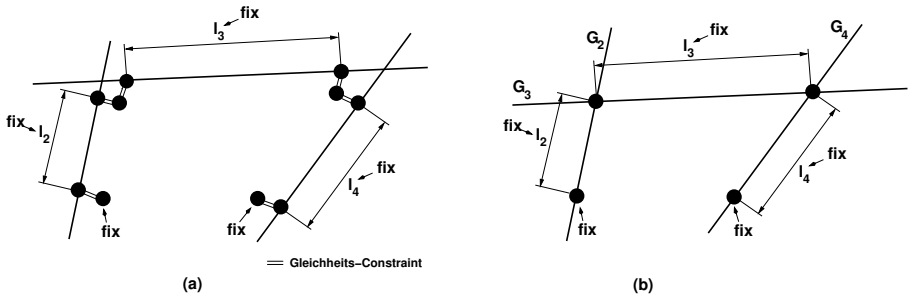


Abbildung A.3: Constraint-basierte Beschreibung der Kurbelschwinge aus Abbildung A.2. Es wird gezeigt (a) das constraint-basierte Modell in MASP, das auf der symbolischen Beschreibung basiert und (b) das vereinfachte Modell, mit dem der Constraint-Solver intern arbeitet.

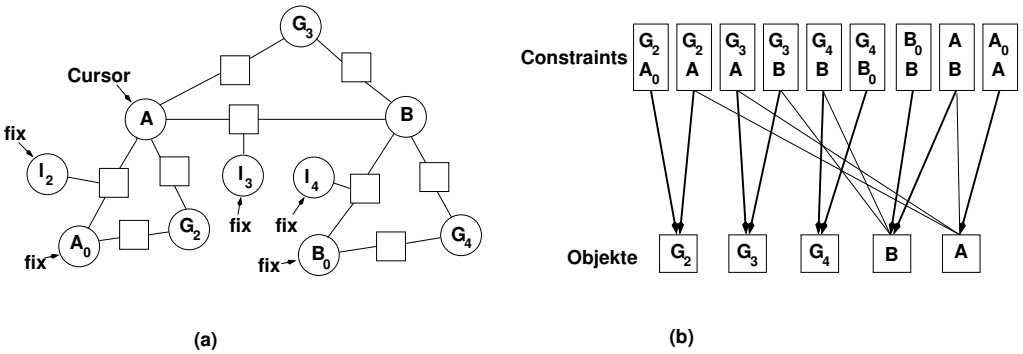


Abbildung A.4: Constraint-Netz für das Beispiel aus Abbildung A.2. (a) zeigt das Constraint-Netz zur Repräsentation des Modells im Constraint-Solver und (b) den bipartiten Constraint-Graphen mit den definierten Constraints „Abstand von Punkt zu Punkt“ und „Punkt auf Gerade“ sowie den nicht-fixierten Objekten für die Freiheitsgradanalyse

Knotenmengen. Fixierte Objekte, die keinen Freiheitsgrad besitzen, haben keinen Einfluss auf die Freiheitsgradanalyse und brauchen deshalb nicht berücksichtigt werden.

Ziel der Freiheitsgradanalyse ist es, eine Zuordnung der Valenzen zu den Freiheitsgraden der Objekte zu finden. In der englischsprachigen Literatur wird diese Zuordnung auch als „maximum matching“ oder „maximum flow“ bezeichnet (vgl. Abschnitt 3.3.1). In Abbildung A.4(b) wird eine gefundene Zuordnung auf der Basis des Beispiels aus A.2 gezeigt. Kanten, die in der Zuordnung enthalten sind, wurden fett gezeichnet. Alle im Beispiel benutzten Constraints haben eine Valenz von eins, d. h. dass sie von dem Objekt, dem sie zugeordnet werden, einen Freiheitsgrad binden. Im Allgemeinen können jedoch auch Constraints mit höheren Valenzen auftreten. So besitzt z. B. ein Symmetrie-Constraint zwischen zwei Geraden bezogen auf eine Spiegelgerade in der euklidischen Ebene eine Valenz von zwei und ein Symmetrie-Constraint zwischen zwei Ebenen bezüglich einer Spiegelebene im euklidischen Raum eine Valenz von drei. Die gefundene Zuordnung von Valenzen zu Freiheitsgraden bietet folgende Informationen:

- Constraints, deren Valenzen keinen Objekten zugeordnet werden können, müssen nach der Berechnung getestet werden, ob sie erfüllt sind (überbestimmter Fall).
- Objekte, deren Freiheitsgrade durch zugeordnete Constraints nicht vollständig gebunden wurden, sind unterbestimmt. In Abbildung A.4 ist das der Gelenkpunkt A. Wie die Analyse zeigt, besitzt A noch einen Freiheitsgrad, der für die Positionierung benutzt werden kann. Im Allgemeinen werden die noch vorhandenen Objektfreiheitsgrade dazu benutzt, ein für den

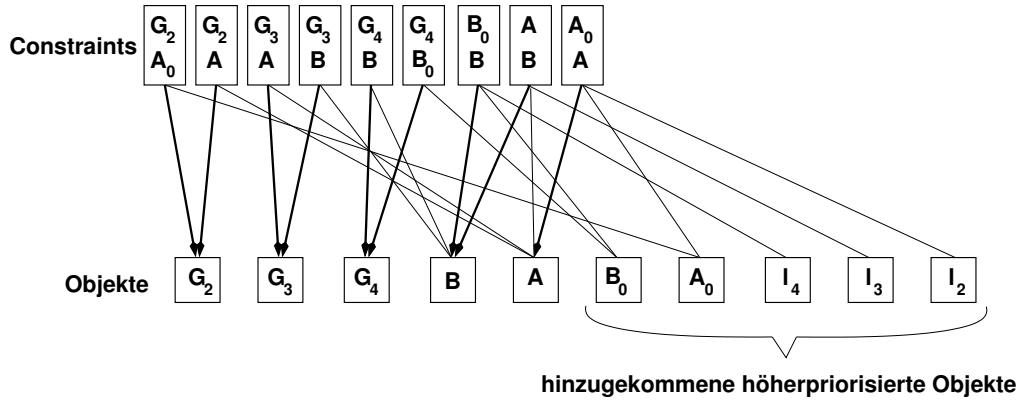


Abbildung A.5: Zuordnung von Constraints zu den Objekten im bipartiten Graphen nach der Freiheitsgradanalyse, wenn Getriebegliedlängen und Drehgelenke aus Abbildung A.2 nicht fixiert sind und eine hohe Priorität besitzen.

> Nutzer möglichst plausibles Verhalten zu erreichen, beispielsweise zum Erhalt der Position von Objekten oder für eine kontinuierliche Bewegung (unterbestimmter Fall).

Eine wichtige Eigenschaft der im Constraint-Solver implementierten Freiheitsgradanalyse ist die Berücksichtigung von Prioritäten für Objekte bzw. deren Freiheitsgrade. Solche Prioritäten beeinflussen maßgeblich die Konstruktionsplanerzeugung. Im Beispiel aus Abbildung A.1 besitzt der Gelenkpunkt A eine hohe Priorität, da er zum Anpassen an die momentane Cursorposition den maximal möglichen Objektfreiheitsgrad haben sollte. Da der Gelenkpunkt A über Constraints mit anderen Objekten verknüpft ist, wird er entsprechend der hohen Priorität bei der Zuordnung der Valenzen zu den einzelnen Objekten als letztes Objekt berücksichtigt (Abbildung A.4(b)), so dass der Gelenkpunkt noch einen Freiheitsgrad besitzt. Wäre der Gelenkpunkt B bewegt worden, hätte B eine gegenüber den anderen Objekten höhere Priorität erhalten. Durch die gezielte Vergabe von unterschiedlichen Prioritäten kann sogar auf Fixierungen verzichtet werden. So ist es möglich die konstanten Abstandsparameter zur Definition von Getriebegliedlängen oder die Koordinatenparameter von gestellfesten Gelenkpunkten in Variable zu überführen, ohne dass sich das Bewegungsverhalten ändert. Dazu ist nur eine höhere Priorisierung dieser Parameter gegenüber allen anderen Objekten notwendig (Abbildung A.5). Neben dem schon beschriebenen Verschieben des Gelenkpunktes A (Abbildung A.1) oder des Gelenkpunktes B ergeben sich neue Möglichkeiten der Interaktion, die es z. B. erlauben die Länge von Getriebegliedern oder die Position von gestellfesten Gelenkpunkten zu modifizieren. Die interaktionsabhängige Priorisierung ermöglicht es, auch bei mehrfach unterbestimmten Modellen für den Nutzer plausible Lösungen zu berechnen.

Nach der Freiheitsgradanalyse wird der Konstruktionsplan erstellt. Unter Beachtung der fixierten Objekte und der noch vorhandenen Objektfreiheitsgrade lassen sich die Constraints ermitteln, die für die Konstruktion weiterer Objekte herangezogen werden können. Es sind stets die Constraints, die nur noch mit einem Objekt verknüpft sind, dessen Konsistenz im Gesamtmodell noch nicht sichergestellt ist. Im Beispiel aus Abbildung A.4 haben vier Constraints diese Eigenschaft („$A_0 auf\ G_2$“, „$B_0 auf\ G_4$“, „$A_0 zu\ A$“ und „$B_0 zu\ B$“). Es kann jedoch nur für den Gelenkpunkt A eine Konstruktionsvorschrift ermittelt werden, für andere Objekte reicht die Summe der verbliebenen Freiheitsgrade und der nutzbaren Valenzen nicht aus. Entsprechend der Art des Constraints und der Information, dass der Gelenkpunkt A nahe der Cursorposition platziert werden soll, wird im Konstruktionsplan der erste Eintrag vorgenommen. Nachdem A bekannt ist, stehen auch die Constraints „$A\,zu\,B$“ und „$A\,auf\,G_2$“ zur Verfügung. Dementsprechend könnten nun Gelenkpunkt B oder Gerade G_2 konstruiert werden. Nach der Konstruktion von B sind dann auch Geraden G_3 und G_4 konstruierbar. Der fertige Konstruktionsplan kann wie folgt aussehen:

1. Konstruiere den Punkt A als Projektion der Cursorposition auf den Kreis um A_0 mit dem Radius l_2,
2. konstruiere den Punkt B als Schnittpunkt des Kreises um A mit dem Radius l_3 und des Kreises um B_0 mit dem Radius l_4,
3. konstruiere die Gerade G_1 durch die Punkte A_0 und A,
4. konstruiere die Gerade G_2 durch die Punkte A und B,
5. konstruiere die Gerade G_3 durch die Punkte B und B_0.

Die Werte der unabhängigen Parameter (die Koordinaten von A_0, B_0 und der Cursorposition sowie die Abstände l_2, l_3 und l_4) im aufgestellten Konstruktionsplan sind, wie auch bei den history-basierten Ansätzen, durch den Nutzer wählbar.

A.4.2 Mechanismus mit iterativer Berechnung

Im Beispiel aus Anhang A.4.1 konnte bei der Aufstellung des Konstruktionsplanes eine iterative Berechnung vermieden werden. Abbildung A.6(a) zeigt einen Mechanismus, bei dem nicht ausschließlich direkt konstruiert werden kann, wenn der Gelenkpunkt E bewegt wird. Die Geraden G_1 und G_2, der Punkt E sowie die sechs Punktabstände l_1 bis l_6 in Abbildung A.6(b) sind bekannt. Keiner der Punkte A bis D ist jedoch konstruierbar, denn bei jedem Punkt ist nur ein Constraint für die Konstruktion nutzbar. An dieser Stelle wird der in Abschnitt 3.6 beschriebene Ansatz verwendet und ein Plan für eine iterative Konstruktion erstellt, der wie folgt aussehen könnte:

1. Konstruiere B auf dem Kreis um E und unter dem Anstiegswinkel α der Geraden durch die Punkte E und B,
2. konstruiere A als Schnittpunkt von G_1 und dem entsprechenden Kreis um B,
3. konstruiere C als Schnittpunkt von G_2 und dem entsprechenden Kreis um B,
4. konstruiere D als Schnitt zweier Kreise um C und E,
5. vergleiche den sich aus der Konstruktion ergebenden Abstand zwischen A und D mit dem definierten Abstand und wenn eine Abweichung $f(\alpha) \neq 0$ auftritt, gehe zu Schritt 1 und variiere α.

Mit diesem Vorgehen lässt sich die Fehlerfunktion $f(\alpha)$ abtasten, bis eine Nullstelle gefunden wird. Die Nullstellensuche verwendet eine Kombination aus Bisektions- und Sekantenverfahren sowie eine Untersuchung von Extremwerten wie sie in Abschnitt 3.6 erläutert werden.

A.5 Behandlung von Mehrfachlösungen

A.5.1 Parallelkurbeln

In Abschnitt 4.1.2 wurden unterschiedliche Berechnungsmodi diskutiert, die Ficucs unterstützt. An dieser Stelle sollen Unterschiede im Bewegungsverhalten anhand der Bewegung einer Parallelkurbel (Abbildung A.7) erläutert werden.

Interaktive Bewegungen erfolgen zumeist sprunghafter als sie von den Nutzern eingeschätzt werden, weshalb die Folge der bisherigen Lösungen bei der Berechnung der nächsten Lösung nicht

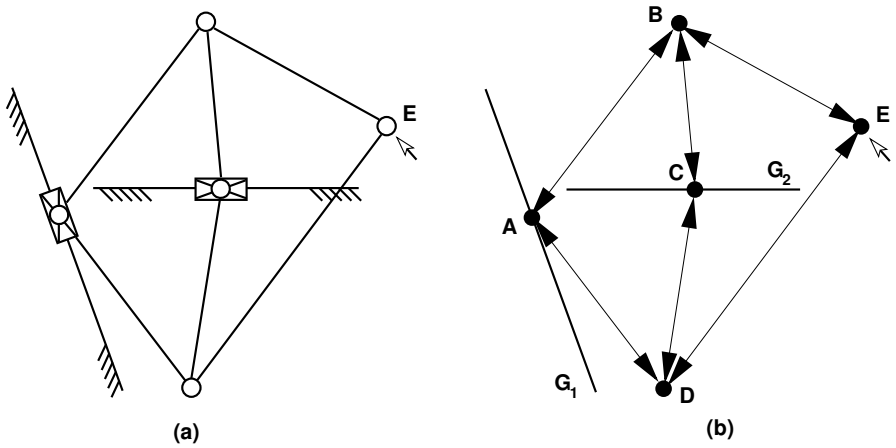


Abbildung A.6: Mechanismus als Beispiel für die Anwendung numerischer Verfahren. (a) Der Mechanismus, der bei Bewegung des Gelenkpunktes E iterativ berechnet wird und (b) das vereinfachte Constraint-Modell (es enthält nur die für die iterative Konstruktion wichtigen Elemente).

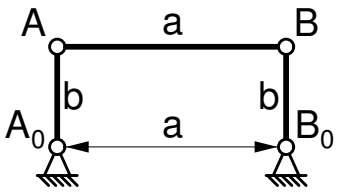


Abbildung A.7: Parallelkurbel

berücksichtigt wird. Abbildung A.8 zeigt die Nutzung eines *RHS*-Hinweises, der die Konstruktion von B stets so steuert, dass B rechts von der Geraden durch B_0 und A (sie wird auch als Kriteriumsgerade bezeichnet) liegt. Die Bewegungsfolge (a_1) bis (a_3) wird von vielen Nutzern als plausibel eingeschätzt. Das Ergebnis (c_3) der Bewegungsfolge (c_1) bis (c_3) wird jedoch von weniger Nutzern erwartet[1]. Das Hauptproblem besteht aber darin, dass die Kriteriumsgerade ihre Richtung schlagartig ändern kann. Dies ist in Bewegungsfolge (b_1) bis (b_3) dargestellt. Bei dem Durchgang $(A = B_0)$ ändert sich die Richtung extrem, unabhängig davon, wie kontinuierlich der Mauszeiger bewegt wurde. Wenn diese Situation nicht speziell behandelt wird (siehe Vorschlag in Abschnitt 4.1.2 auf Seite 104), dann kommt es zu einer falschen Lösungsauswahl, wie sie in Abbildung A.8(b_3) dargestellt ist.

Bei kontinuierlicher Parameteränderung an einem Antrieb gehen die in Ficucs implementierten Konstruktoren von einer Kontinuität der lokalen Lösungen aus. Statt der *„Beibehaltung der relativen Position“* zur Kriteriumsgeraden wird hier ein anderes Kriterium für die Wahl von B herangezogen, die *„Nähe zur extrapolierten relativen Position“* (Abbildung A.9). Es wird hierbei beobachtet, wie sich B relativ zur Geraden durch A_0 und A bewegt. Auf diese Art und Weise kann ein Seitenwechsel vollzogen werden. Dies führt zu einem meist als plausibel empfundenen Verhalten, wie es die Bewegungsfolgen in Abbildung A.9 (a_1) bis (a_3) sowie (c_1) bis (c_3) zeigen. Im Falle $a = b$ treten jedoch auch hier wieder Probleme auf, denn die relative Position von B hat sich gegenüber der Kriteriumsgeraden nicht wesentlich geändert. Nach dem abrupten Richtungswechsel beim Durchgang $(A = B_0)$ wird B folglich wieder auf der gleichen Seite der Geraden konstruiert. Dies stellt jedoch ein fehlerhaftes Verhalten dar. Für eine korrekte Lösungsauswahl bietet sich wieder die Nutzung eines Vertrauensmaßes an (Seite 104).

[1] Es scheint nicht dem Erfahrungsschatz dieser Nutzer zu entsprechen.

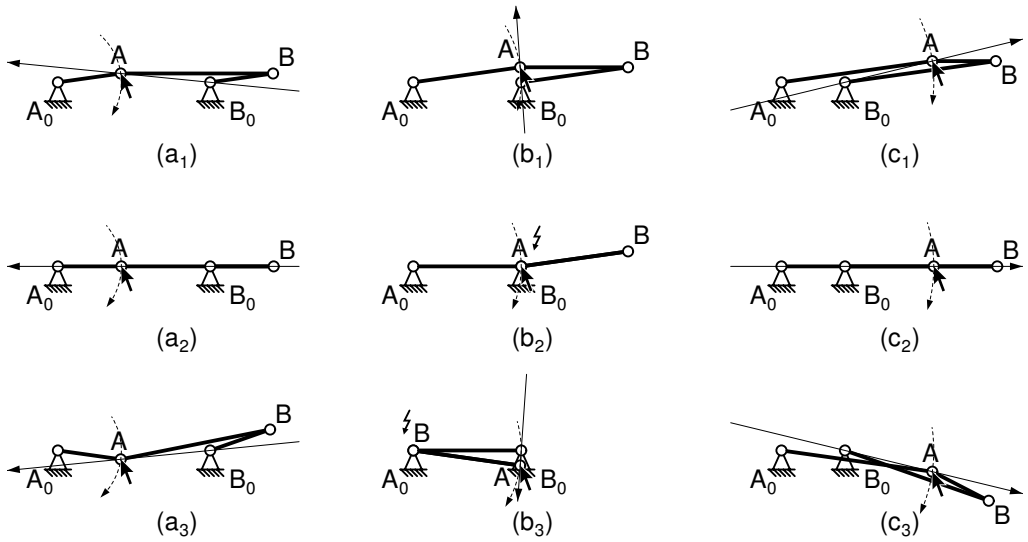


Abbildung A.8: Interaktive Änderungen durch Bewegen des Drehgelenkes A mit dem Mauszeiger. B wird **stets rechts** von der eingezeichneten Kriteriumsgeraden durch B_0 und A konstruiert. In den Teilabbildungen (a_1) bis (a_3) ist $a > b$, in (b_1) bis (b_3) ist $a = b$ und in (c_1) bis (c_3) ist $a < b$. Das Hauptproblem tritt beim Übergang von (b_2) nach (b_3) auf.

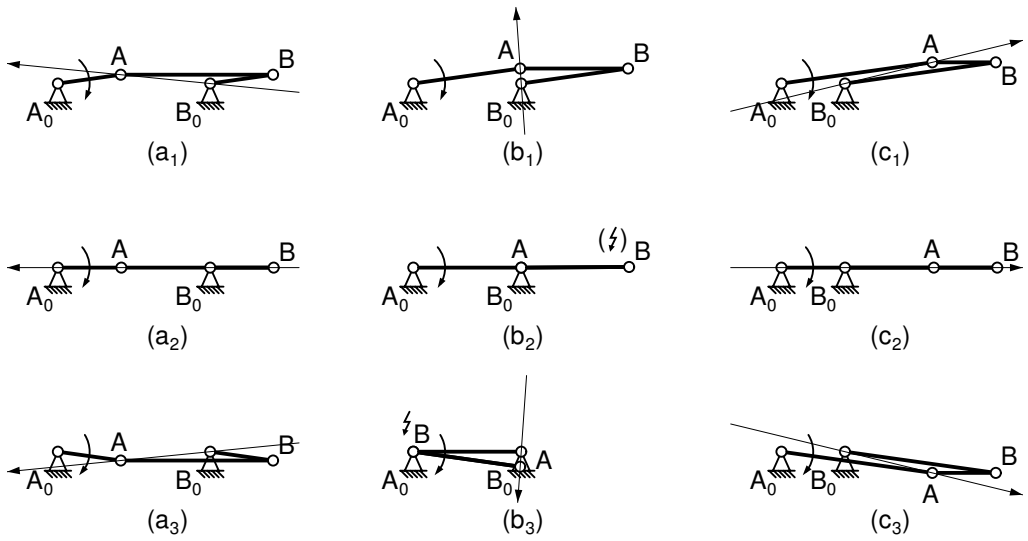


Abbildung A.9: Animation durch konstante Winkeländerung des Getriebegliedes A_0A. Die bisherige Bewegung von B bezüglich der Kriteriumsgeraden wird berücksichtigt. Man sieht wie in Abbildung (a_1) sich B der Geraden genähert hat, dann in (a_2) auf der Geraden liegt und schließlich in (a_3) die Seite gewechselt hat. Das Verhalten in den Abbildungen (c_1) bis (c_3) ist analog. Auch hier werden die Lösungen so gewählt, dass die Parallelität erhalten bleibt. Nur wenn $a = b$ ist und es dazu kommt, dass die Kriteriumsgerade nicht konstruiert werden kann bzw. sich ihre Richtung schnell ändert, schlägt die Nutzung der Kriteriumsgeraden fehl. Zwar kann in (b_2) die Position von B noch extrapoliert werden (aktuell wird eine lineare Extrapolation genutzt, deren Resultat auf den Kreis um B_0 projiziert wird), doch in (b_3) wird die falsche Seite gewählt. B hat sich nicht an die Kriteriumsgerade angenähert, denn es hatte stets etwa einen Abstand von a (bzw. b) von ihr. Folgerichtig bleibt B fälschlicherweise auf der rechten Seite und macht einen Sprung.

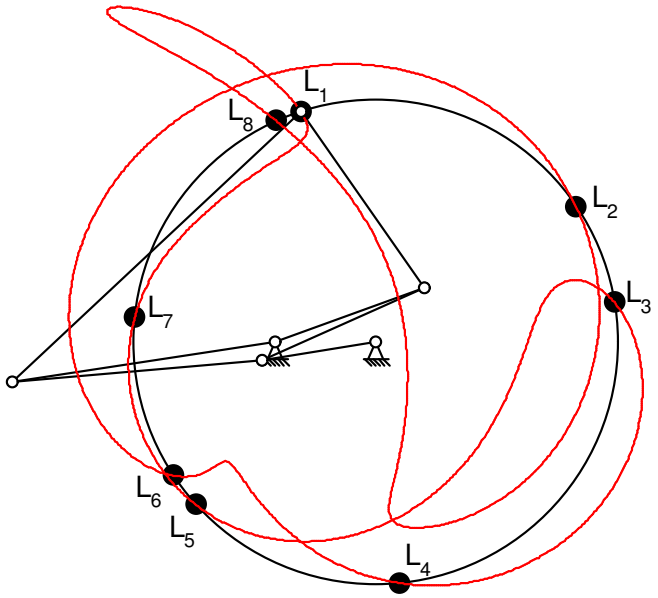


Abbildung A.10: Transformation des starren Körpers aus Abbildung 4.13 in einen Mechanismus mit einem Freiheitsgrad durch Aufheben des Abstands-Constraints zwischen p_2 und p_5. Die Punkte L_1 bis L_8 sind die Orte, an denen sich Punktbahn und Kreis schneiden. Sie repräsentieren acht Lösungen für den starren Körper. In der aktuellen Lösung ist $p_5 = L_1$.

A.5.2 Überführung des Modells aus Abbildung 4.13 in einen Mechanismus

Abbildung A.10 zeigt, wie sich das Prinzip der Transformation von einem starren Körper in einen Mechanismus auf das in Abbildung 4.13 gezeigte Modell übertragen lässt. Es ergeben sich die acht gezeigten Lösungen sowie weitere acht Lösungen aus den Schnittpunkten des Kreises mit der gespiegelten Bewegungsbahn.

A.5.3 Modell aus sechs Punkten mit mehr als 2^4 Lösungen

In Abschnitt 4.2 wurden vier mögliche Schemen gezeigt, die zur Versteifung von sechs Punkten in der Ebene dienen können. Borcea und Streinu beschreiben in [BS02] ein nach Schema IV versteiftes Modell (vgl. Abbildung 4.11). Die Besonderheit dieses Modells besteht darin, dass es alle entsprechend [Sch93] möglichen vierundzwanzig Lösungen aufweist. Abbildung A.11 zeigt die in MASP gefundenen Lösungen sowie die entsprechenden Fehlerfunktionen der iterativen Konstruktionen.

Zur Veranschaulichung der Lösungen wurde in [BS02] eine interessante Problemtransformation vorgenommen, die die Analyse von starren Körpern auf die Analyse von Mechanismen zurückführt. Auf Grund der Ausführungen lässt sich vermuten, dass zum Auffinden der Lösungen auf die Geometriesoftware Cinderella zurückgegriffen wurde. Abbildung A.12 verdeutlicht die in [BS02] zur Darstellung der Lösungen benutzte Problemtransformation. Abbildung A.13 zeigt schließlich die Lage der vierundzwanzig möglichen Lösungen in vier Konfigurationen mit jeweils sechs Schnitten von Kreisen und Bewegungspfaden.

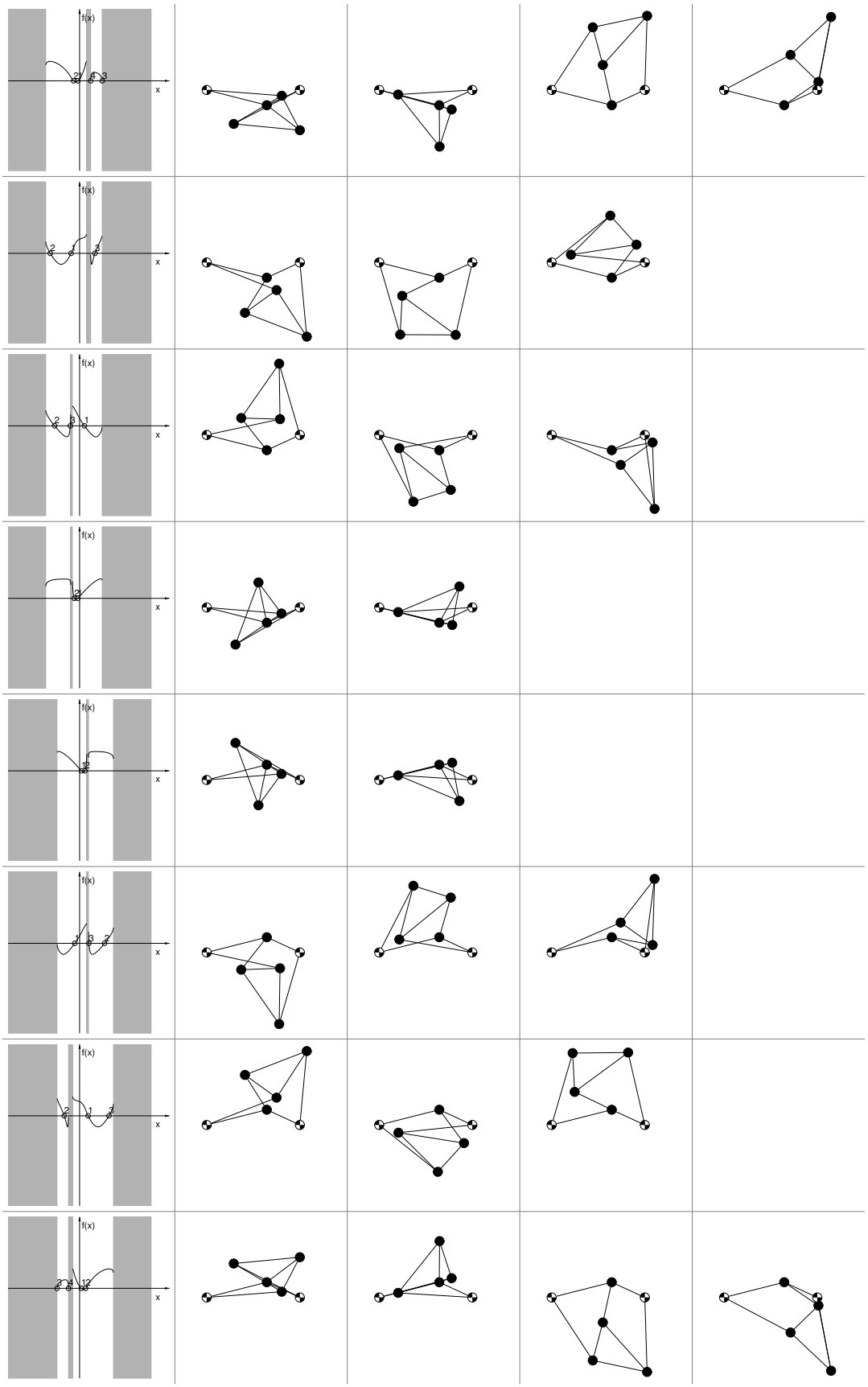


Abbildung A.11: Alle vierundzwanzig Lösungen einer nach Schema IV versteiften Menge von sechs Punkten.

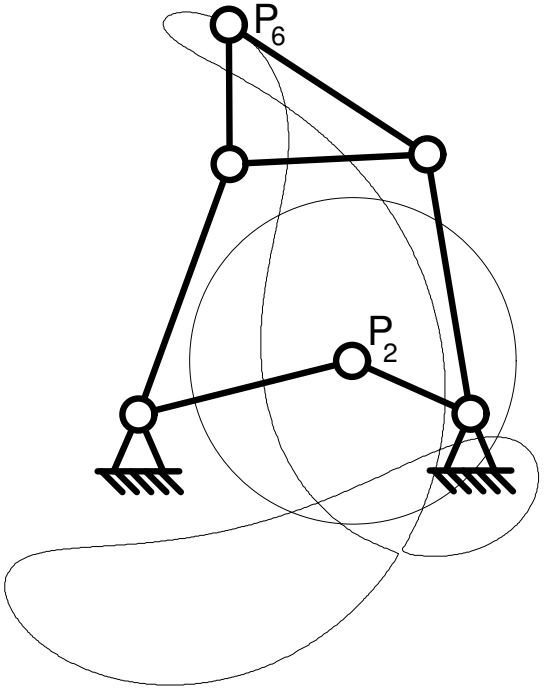


Abbildung A.12: Transformation des starren Körpers in einen Mechanismus mit einem Freiheitsgrad durch Aufheben eines Abstands-Constraints (hier zwischen P_2 und P_6). Die Orte, an denen sich Punktbahn und Kreis schneiden, bilden die Menge der Lösungen für die Lage des Punktes P_6 des starren Körpers.

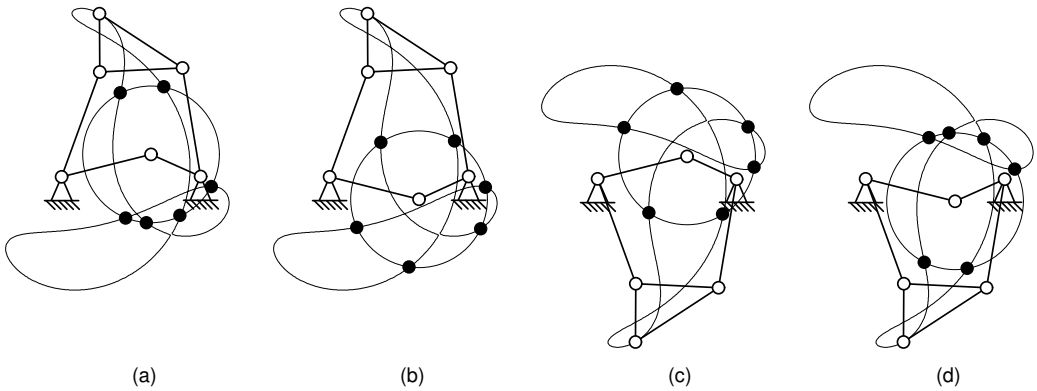


Abbildung A.13: Lage der vierundzwanzig Lösungen für Punkt P_6, die sich aus den zwei Pfaden und den zwei möglichen Kreispositionen ergeben.

Anhang B

Diskussion verschiedener numerischer Verfahren

An dieser Stelle sollen numerische Verfahren näher diskutiert werden. Ziel der Diskussion ist es, Gründe für die in Kapitel 4 beschriebenen Probleme numerischer Verfahren aufzuzeigen.

B.1 Übersicht über die erläuterten Verfahren

Die in Kapitel 4 beschriebenen Beispiele basieren auf *2D*-Punkten und Abständen, die zwischen diesen Punkten definiert sind. Grundsätzlich lassen sich auch andere Modelle (z. B. solche mit Winkel-Constraints) so transformieren, dass in ihnen nur noch Abstands-Constraints enthalten sind[1]. Das durch die Abstand-Constraints beschriebene Gleichungssystem ist zwar typischerweise vollbestimmt, jedoch sind die einzelnen Gleichungen nichtlinear (quadratisch), so dass Ansätze für das Lösen linearer Gleichungssysteme (z. B. die Gauß-Elimination oder die QR-Zerlegung) im Allgemeinen nicht direkt einsetzbar sind. Statt dessen wird in den Verfahren zur Lösung des nichtlinearen Gleichungssystems iterativ vorgegangen. Ausgehend von einer fehlerbehafteten initialen Lösung (sie wird durch die sogenannten Startwerte der Variablen beschrieben) erfolgt eine schrittweise Annäherung an die gesuchte fehlerfreie Lösung. Diese ist gefunden, wenn die Werte der Variablen, die die jeweils aktuelle Lösung beschreiben, das Gleichungssystem erfüllen. Die einzelnen Verfahren unterscheiden sich darin, wie (ausgehend von der aktuellen fehlerbehafteten Lösung) die nächste bessere[2] (d. h. insgesamt mit einem geringeren Fehler behaftete) Lösung bestimmt wird. In der Literatur speziell zu geometrischen Constraints sowie allgemein zur Numerik kann eine Vielzahl von grundlegenden Lösungsansätzen sowie Vorschlägen zu ihrer Erweiterung und auch Kombination gefunden werden. Die in dieser Arbeit erläuterten Verfahren lassen sich bezüglich ihrer Berechnungen pro Iterationsschritt folgendermaßen einteilen:

- Verschiebung von jeweils nur einem Punkt,
- Verschiebung von jeweils zwei Punkten und
- gleichzeitige Verschiebung aller Punkte.

[1] engl.: *d-reduction*, siehe z. B. [SB98b] Problematisch kann hierbei jedoch das Beibehalten der ursprünglichen Mehrfachlösungen sein. Zum einen ist es möglich, dass Mehrfachlösungen, die im Ausgangsmodell vorhanden waren, nach der Transformation nicht mehr existieren und zum anderen können nach der Transformation neue Mehrfachlösungen auftreten.

[2] Teilweise wird auch nicht darauf geachtet, dass eine Verbesserung stattfindet. Wenn es dadurch zu einer stetigen Vergrößerung des Gesamtfehlers kommt, spricht man von der Divergenz der Lösungssuche. Vgl. Abbildung 3.29 als Beispiel für Divergenz in der univariaten Suche.

Zum Finden einer nächsten besseren Lösung durch Verschiebung von jeweils einem Punkt oder zwei Punkten lassen sich entsprechende Heuristiken[3] anwenden. Die besprochenen Verfahren betrachten bestimmte Variablen (z. B. die Koordinaten von Punkten) und suchen nach einer (aus geometrischer Sicht) lokal besseren Lösung für diese Variablen. Wichtig für den Erfolg der entsprechenden Verfahren ist, dass die einzelnen lokalen Verbesserungen auch insgesamt zu einer Verbesserung der Lösung führen. Die besprochenen Verfahren konvergieren eher sehr langsam.

Die gleichzeitige Verschiebung aller Punkte lässt sich gut mit etablierten Methoden der Mathematik lösen. Die gemeinsame Grundidee besteht darin das Gleichungssystem des Modells zu linearisieren. Auf die folgenden Varianten wird näher eingegangen:

1. Direkte Linearisierung des Gleichungssystems

   Das Verfahren arbeitet analog zum auf Seite 79 beschriebenen Newtonschen Verfahren, welches auf eine Funktion mit einer Variablen anzuwenden ist.[4] Es nutzt *direkt* die partiellen Ableitungen[5] der einzelnen Gleichungen nach den Variablen des Gleichungssystems (entsprechend dem Satz von Taylor bei mehreren Veränderlichen). Hierdurch werden die im Allgemeinen nichtlinearen Gleichungen durch je eine lineare Gleichung ersetzt. Geometrisch gesehen beschreibt jede lineare Gleichung eine Hyperebene, die in der aktuellen Position tangential an der Hyperfläche (sie repräsentiert die nichtlineare Gleichung) anliegt[6]. Die Lösung des linearisierten Gleichungssystems ergibt Punktverschiebungen, die oft zu einer verbesserten Lösung führen. Die Konvergenzgeschwindigkeit ist dann quadratisch. Jedoch kommt es ebenso oft vor, dass das Verfahren divergiert, insbesondere wenn die Startwerte nicht nahe der Lösung liegen. Durch Schrittweitensteuerung oder die Nutzung eines Homotopie-Ansatzes lässt sich dieses Problem zum Teil lösen.

2. Minimierung der Summe der Fehlerquadrate[7]

   Dieser Ansatz wurde durch Gauß und Legendre zur Ausgleichsrechnung (u. a. bei der Berechnung von Dreiecksnetzen in der Landesvermessung) entwickelt. Der Ansatz bietet den Vorteil, dass mit seiner Hilfe grundsätzlich auch überbestimmte Gleichungssysteme gelöst werden können[8]. Die Idee besteht darin, dass die Funktionswerte der nicht erfüllten Gleichungen als Fehlerwerte angesehen werden, die in ihrer Gesamtheit möglichst minimiert werden sollen. Diese Gesamtheit ist als Summe der quadrierten Fehler definiert. Um das Minimum der Summe der Fehlerquadrate zu finden, werden die partiellen Ableitungen der Summe nach den einzelnen Variablen gleich null gesetzt[9]. So ergibt sich ein vollbestimmtes Gleichungssystem, in dem die Anzahl der Variablen gleich der Anzahl der Gleichungen

---

[3] Der Begriff Heuristik lässt sich nur schwer fassen (vgl. Seite 122). In dieser Arbeit wird grundsätzlich zwischen in der Mathematik etablierten Verfahren und Heuristiken unterschieden.

[4] In der deutschsprachigen Literatur wird typischerweise auch bei einer Anwendung auf Gleichungssysteme die Bezeichnung *Newtonsches Verfahren* verwendet. In der englischsprachigen Literatur ist hierfür eher die Bezeichnung *Newton-Raphson-Verfahren* üblich. Zum Teil lassen sich auch andere in der Literatur beschriebene Verfahren auf die direkte Linearisierung zurückführen (z. B. [SB98b], siehe Anhang B.7).

[5] Die Werte der partiellen Ableitungen an der aktuellen Position bilden die Jacobimatrix bzw. die Systemmatrix.

[6] Im einfachsten Fall (eine Gleichung, eine Variable) liegt eine Gerade als Tangente an der Fehlerkurve an (siehe z. B. Abbildung 3.26). Der Schnittpunkt der Tangente mit der x-Achse bildet jeweils die nächste zu untersuchende Position.

Bei zwei Variablen sind typischerweise zwei Gleichungen gegeben (wegen der Vollbestimmtheit der Modelle). Jede Gleichung kann man sich als eine Fläche vorstellen, die durch die Funktion $f_i(x, y)$ mit $i = 1$ *bzw.* 2 beschrieben wird. Evtl. ergeben sich auch mehrere Flächen, z. B. bei $a \pm \sqrt{x + y} = 0$. Im folgenden Text wird jedoch nur von einer ausgegangen. Die Lösung der Gleichung $f_i(x, y) = 0$ entspricht der (nichtlinearen) Schnittkurve n_i der Fläche mit der xy-Ebene. Die Lösung des Gleichungssystems besteht schließlich aus den gemeinsamen Punkten der Schnittkurven n_i. Zur Linearisierung wird an die Flächen $f_i(x, y)$ in der jeweils aktuellen Position (x_k, y_k) je eine Tangentialebene $t_i(x, y)$ gelegt. Als Schnittpunkte der Tangentialebenen mit der xy-Ebene ergeben sich zwei Geraden $t_i(x, y) = 0$. Die Lösung des linearen Gleichungssystems $t_1(x, y) = t_2(x, y) = 0$ wird oft als nächste zu untersuchende Position (x_{k+1}, y_{k+1}) verwendet.

[7] eine Anwendung der Methode der kleinsten Quadrate

[8] Es ist jedoch nicht sicher, ob der bei einer widersprüchlichen Überbestimmtheit gefundene Kompromiss vom Nutzer akzeptiert wird.

[9] Problematisch hieran ist, dass diese Bedingung auch Sattelpunkte und Maxima erfüllen. In den Iterationen ist deshalb sicherzustellen, dass bei der Suche die Summe der Fehlerquadrate gegen null konvergiert.

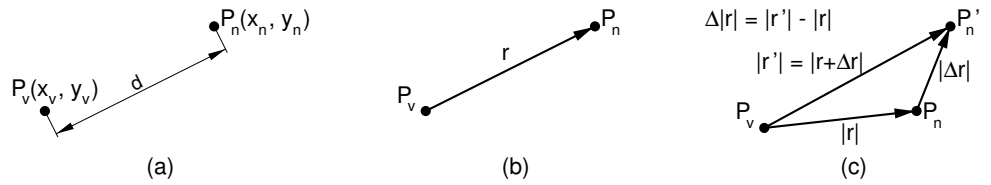


Abbildung B.1: Modellierung eines Abstands-Constraints zwischen P_n und P_v. (a) Verwendete Variablen bzw. Konstanten. (b) Ausrichtung des Abstands-Constraints durch Einführung eines Vektors. (c) Notation für die Längen der Vektoren r, Δr und r' sowie die Längenänderung zum nächsten Iterationsschritt $\Delta|r|$.

ist. Im Allgemeinen ist das resultierende Gleichungssystem jedoch nichtlinear. Deshalb wird es typischerweise mittels einer nachgeschalteten Linearisierung gelöst. Das Gesamtverfahren aus Minimierung der Summe der Fehlerquadrate und iterativer Lösungssuche im linearisierten Gleichungssystem wird auch als Gauß-Newton-Methode bezeichnet (z. B. [Sch97]). Einen einfacheren Ansatz stellt die Gradientenmethode ([RS89]) dar. Hier werden die Werte der partiellen Ableitungen der Fehlerquadratsumme nach den einzelnen Variablen direkt als die Richtung des nächsten Suchschrittes verwendet.

3. Approximative Linearisierung des Gleichungssystems

   Für die quadratischen Abstands-Gleichungen der diskutierten Modelle sind die Ableitungen leicht zu bestimmen, so dass für die Berechnung dieser Modelle die direkte Linearisierung verwendet werden kann. Allgemeinere Ansätze gehen jedoch davon aus, dass die Ableitungen nicht bekannt sind und somit approximiert werden müssen. Zum anderen ist die Berechnung der Jacobi-Matrix für große Gleichungssysteme möglicherweise sehr aufwändig, so dass mit einer Approximation der Jacobi-Matrix Rechenzeit gespart werden kann. Die entsprechenden Verfahren werden auch als Quasi-Newton-Verfahren bezeichnet. Eines von ihnen ist das Broyden-Verfahren (z. B. [TS88]).

Zum besseren Verständnis werden die Verfahren auf die in Kapitel 4 beschriebene Menge von sechs Punkten mit neun Abständen (versteift nach Schema III) angewendet.

Zunächst erfolgt jedoch eine allgemeine Betrachtung zur Linearisierung, die vielfach die Grundlage der Verfahren bildet.

B.2 Allgemeine Betrachtungen zur Linearisierung des Beispiels

Zur Auffindung einer Möglichkeit, wie die aktuelle Lösung verbessert werden kann, wird oft eine Linearisierung des entsprechenden nichtlinearen Modells vorgenommen. Im Beispielmodell kommen nur Abstands-Constraints bzw. entsprechende quadratische Gleichungen vor. Angenommen das ursprüngliche Modell enthält (wie in Abbildung B.1(a) dargestellt) eine Gleichung für den Abstand d zwischen einem fixierten[10] Punkt $P_v(x_v, y_v)$ und einem freien Punkt $P_n(x_n, y_n)$. In diesem Fall ist es üblich, den eigentlich ungerichteten Abstand intern für die Berechnungen durch Einführung eines Richtungsvektors r auszurichten[11] (Abbildung B.1(b)). Die aktuelle Länge des

[10] Die Fixierung erlaubt hier einfachere Betrachtungen. In den folgenden Abschnitten sind grundsätzlich beide Punkte nicht fixiert.

[11] Die Ausrichtung steht in keinem Zusammenhang mit der Propagation von Werten während der Berechnung. Die Einführung des Vektors dient vielmehr der eindeutigen Beschreibung der relativen Lage, die die beiden Punkte zueinander einnehmen.

Richtungsvektors kann dann in den Berechnungen als aktueller Abstand genutzt werden. Die Abstandsgleichung lautet:

$$f(x_n, y_n) = |r| - d = \sqrt{(x_n - x_v)^2 + (y_n - y_v)^2} - d = 0 \tag{B.1}$$

Zur Linearisierung lässt sich der Satz von Taylor für zwei Veränderliche anwenden[12]. Hierfür ist nach den beiden Veränderlichen x_n und y_n partiell abzuleiten:

$$f(x_n + \Delta x_n, y_n + \Delta y_n) = f(x_n, y_n) + \Delta x_n \cdot \frac{\partial f}{\partial x_n}(x_n, y_n) + \Delta y_n \cdot \frac{\partial f}{\partial y_n}(x_n, y_n) \tag{B.2}$$

Zum Zeitpunkt der Linearisierung ist $f(x_n, y_n) = |r| - d$ typischerweise ungleich null. Der Wert von $f(x_n, y_n)$ wird oft direkt als vorzeichenbehaftetes Maß für den aktuellen Fehler δ verwendet.

$$f(x_n, y_n) = |r| - d = \delta \tag{B.3}$$

Durch die Gleichung

$$f(x_n + \Delta x_n, y_n + \Delta y_n) = \delta' = 0 \tag{B.4}$$

beschreibt man eine Verschiebung $\Delta r(\Delta x_n, \Delta y_n)$ von P_n in x- bzw. y-Richtung, die dazu führen soll, dass der Constraint nach der Verschiebung erfüllt ist[13] ($\delta' = 0$). Gleichung B.2 kann unter Beachtung von Gleichung B.4 umgestellt (B.5) und vereinfacht werden (B.6 und B.7):

$$-f(x_n, y_n) = \Delta x_n \cdot \frac{\partial f}{\partial x_n}(x_n, y_n) + \Delta y_n \cdot \frac{\partial f}{\partial y_n}(x_n, y_n) \tag{B.5}$$

$$-\delta = d - |r| = \Delta x_n \cdot \frac{x_n - x_v}{\sqrt{(x_n - x_v)^2 + (y_n - y_v)^2}} + \Delta y_n \cdot \frac{y_n - y_v}{\sqrt{(x_n - x_v)^2 + (y_n - y_v)^2}} \tag{B.6}$$

$$-\delta = \Delta x_n \cdot \frac{x_n - x_v}{|r|} + \Delta y_n \cdot \frac{y_n - y_v}{|r|} \tag{B.7}$$

Die Einführung der Variablen $c = \frac{x_n - x_v}{|r|}$ und $s = \frac{y_n - y_v}{|r|}$ für den Cosinus bzw. Sinus von r vereinfacht die Notation nochmals:

$$-\delta = \Delta x_n \cdot c + \Delta y_n \cdot s \tag{B.8}$$

Gleichung B.8 beschreibt wie die zulässige Änderung $\Delta r(\Delta x_n, \Delta y_n)$ vom Fehler δ abhängt, wenn die Linearisierung nach Gleichung B.2 erfolgte. Zur Veranschaulichung, was Gleichung B.8 geometrisch bedeutet, soll an dieser Stelle das Skalarprodukt $r \cdot \Delta r$ betrachtet werden[14]. Es ist abhängig vom Winkel α, den die beiden Vektoren einnehmen, sowie ihren Längen[15].

$$r \cdot \Delta r = \cos\alpha \cdot |r| \cdot |\Delta r| = k \tag{B.9}$$

---

[12] Der Abbruch der Reihenentwicklung erfolgt nach dem Glied erster Ordnung.

[13] Wegen des Linearisierungsfehlers wird der Constraint im Allgemeinen nur näherungsweise erfüllt sein. Der Fehler soll durch die Wiederholung der Berechnungen stetig verkleinert werden. Die Größe des Linearisierungsfehlers hängt von den konkreten Gegebenheiten im Modell ab. Ein Beispiel ist in Abbildung B.2(c) gezeigt.

[14] Ein im Umgang mit Linearer Algebra und Analytischer Geometrie geübter Leser dürfte bereits erkannt haben, dass die rechte Seite der Gleichung B.8 als ausmultipliziertes Skalarprodukt aufgefasst werden kann. Er mag bei Gleichung B.13 weiterlesen.

[15] An dieser Stelle sei nochmals auf den Unterschied zwischen $|\Delta r|$, der Länge des Verschiebungsvektors, und $\Delta|r|$, der Längenänderung von r zwischen den Iterationsschritten, hingewiesen.

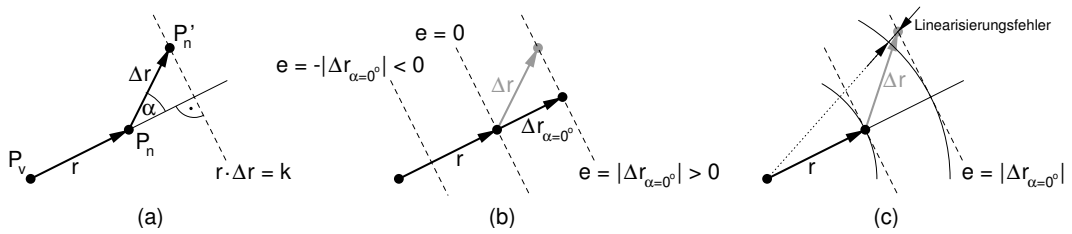


Abbildung B.2: Nutzung des Skalarproduktes $r \cdot \Delta r$ zur Beschreibung der Linearisierung eines euklidischen Abstandes. (a) Implizite Definition einer Gerade, auf der ein Punkt $P_n^{'}$ liegen kann, wenn r (der alte Vektor von P_v nach P_n) und k (ein skalarer Wert) als fix angenommen werden. (b) Bedeutung des Wertes der Konstanten e unter der Annahme, dass r normalisiert wurde. (c) Linearisierungsfehler für ein bestimmtes r sowie Δr.

Wird der (skalare) Wert des Skalarproduktes auf eine Konstante k festgelegt und r als gegeben angenommen, so können noch unendlich viele verschiedene Δr die Gleichung B.9 erfüllen. Eine geometrische Interpretation ist in Abbildung B.2(a) gezeigt. Alle durch die Verschiebung Δr erzeugbaren Punkte $P_n^{'}$ befinden sich auf einer Geraden, die senkrecht zu r steht. Die Wahl von k legt fest, wo die Gerade sich bezüglich P_n befindet (Abbildung B.2(b)). Geht die Gerade durch P_n, so ist $\alpha = 90°$. Nach Gleichung B.9 entspricht dies $k = 0$. Für Werte mit $k \neq 0$, ist es günstiger eine Konstante e zu nutzen, die sich durch Normierung von k bestimmen lässt.

$$e = \frac{k}{|r|} = \frac{r}{|r|} \cdot \Delta r = \begin{pmatrix} c \\ s \end{pmatrix} \cdot \begin{pmatrix} \Delta x_n \\ \Delta y_n \end{pmatrix} \tag{B.10}$$

Durch Ausmultiplizieren des Skalarproduktes ergibt sich:

$$e = c \cdot \Delta x_n + s \cdot \Delta y_n \tag{B.11}$$

In die Gleichung $k = \cos\alpha \cdot |r| \cdot |\Delta r|$ lässt sich statt dem allgemeinen Δr auch eine der möglichen speziellen Richtungen einsetzen. Der einfachste Fall liegt für $\alpha = 0°$ vor. Da $\cos 0° = 1$ ist erhält man:

$$k = |r| \cdot |\Delta r_{\alpha=0°}| \tag{B.12}$$

Unter Berücksichtigung von $k = |r| \cdot e$ zeigt sich, dass e gleich dem Abstand des Punktes P_n von der Geraden, auf der sich die neuen Lösungen $P_n^{'}$ befinden müssen, ist.

Eine Gegenüberstellung der Gleichungen B.8 und B.11 zeigt zudem, dass bei der oben beschriebenen Linearisierung nach Taylor e aus dem negierten Fehler (also der benötigten Änderung) gewonnen wird.

$$e = -\delta = d - |r| \tag{B.13}$$

Bei zu großem aktuellen Abstand wird e negativ, bei zu kleinem Abstand positiv. Angenommen es liegt ein in Abbildung B.3(a) bzw. (b) gezeigter Fall vor, wo $|r| > d$ bzw. $|r| < d$ ist. Gleichung B.13 zeigt, dass die Linearisierung nach Taylor den neuen Punkt $P_n^{'}$ auf die Gerade setzen würde, die den Kreis (um P_v mit dem Radius d) in Richtung von r tangiert (Abbildung B.3(c)). Ist der Winkel α zwischen dem gegeben r und dem berechneten Δr gleich 0°, so wird kein Linearisierungsfehler gemacht. Der nächste Fehler δ' wird dadurch zu null. Aber auch bei Werten von $\alpha \neq 0°$ kann der Absolutwert des Fehlers verkleinert werden ($\delta' = |r'| - d < \delta$). Der entsprechende Bereich auf der Geraden wurde in Abbildung B.3(c) hervorgehoben. Außerhalb dieses Bereiches besteht die Gefahr der Divergenz, denn der absolute Fehler vergrößert sich hier.

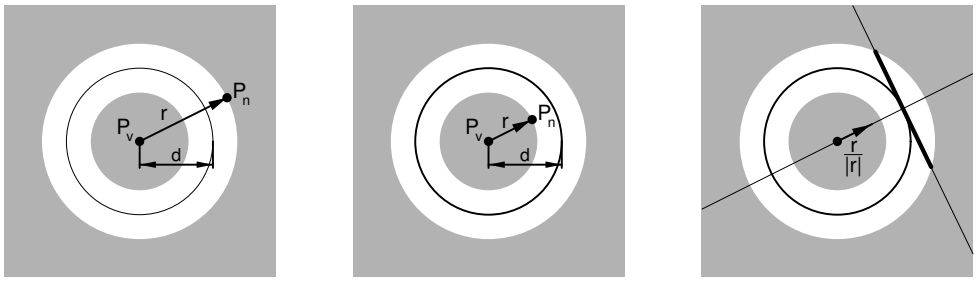


Abbildung B.3: Konvergenzbereich für Verschiebungen des Endpunktes eines Vektors, wenn d der per Constraint definierte Abstand zwischen zwei Punkten P_v und P_n ist. Alle Verschiebungen von P_n in den weißen Bereich führen zu einer Verkleinerung des Fehlers. Erfolgt eine Verschiebung in den grauen Bereich, so besteht die Gefahr der Divergenz. (a) Der Vektor ist zu lang bzw. der Abstand zwischen den Punkten ist zu groß. (b) Der Vektor ist zu kurz bzw. der Abstand zwischen den Punkten ist zu klein. (c) Konvergenzbereich auf einer Geraden, die im Zuge der Linearisierung als Menge der möglichen Orte für $P_n^{'}$ angenommen wurde.

B.3 Direkte Linearisierung[16]

Der Ansatz

Gesucht ist ein Lösungsvektor X für ein nichtlineares Gleichungssystem F, so dass $F(X) = 0$. Zur iterativen Lösung des Gleichungssystems wird ein Startwertvektor X_0 herangezogen, der z. B. aus dem aktuellen Zustand des Modells ermittelt werden kann. Wesentlich bei der direkten Linearisierung ist, dass für die Funktionen $F_i(x_1, \ldots x_m)$ mit $i = 1 \ldots n$ (und $m = n$ in vollbestimmten Modellen) die Werte der partiellen Ableitungen $\frac{\partial F_i}{\partial x_j}$ an der jeweils aktuellen Stelle X_k zur Verfügung stehen. Die Werte der partiellen Ableitungen werden in der Jacobi-Matrix[17] J zusammengefasst.

$$J(X_k) = \begin{pmatrix} \frac{\partial F_1}{\partial x_1}(X_k) & \cdots & \frac{\partial F_1}{\partial x_n}(X_k) \\ \vdots & & \vdots \\ \frac{\partial F_n}{\partial x_1}(X_k) & \cdots & \frac{\partial F_n}{\partial x_n}(X_k) \end{pmatrix} \tag{B.14}$$

Im Laufe der Suche werden Suchschritte ΔX_k ausgeführt, die von einer Stelle X_k zur Stelle X_{k+1} führen.

$$X_{k+1} = X_k + \Delta X_k \tag{B.15}$$

Der Ansatz beruht auf der Annahme, dass die Änderung der Funktionswerte ΔF_k näherungsweise linear vom Suchschritt abhängt, wobei die Jacobi-Matrix die entsprechende lineare Abhängigkeit an der Stelle X_k beschreibt[18]. Da das Ziel der Suche $F(X) = 0$ ist, wird die im Suchschritt gewünschte Änderung ΔF_k gleich dem im Vorzeichen negierten aktuellen Fehler $F(X_k)$ gesetzt:

$$\Delta F_k = J(X_k) \cdot \Delta X_k = -F(X_k) \tag{B.16}$$

[16] bzw. Newtonsches Verfahren oder Newton-Raphson-Verfahren, siehe Anmerkung auf Seite 170

[17] Sie wird in der Literatur auch als Funktionalmatrix oder Systemmatrix bezeichnet.

[18] Die Verwendung der Jacobi-Matrix resultiert aus der Reihenentwicklung nach dem Satz von Taylor für mehrere Veränderliche, wobei die Reihenentwicklung nach dem linearen Glied abgebrochen wird.

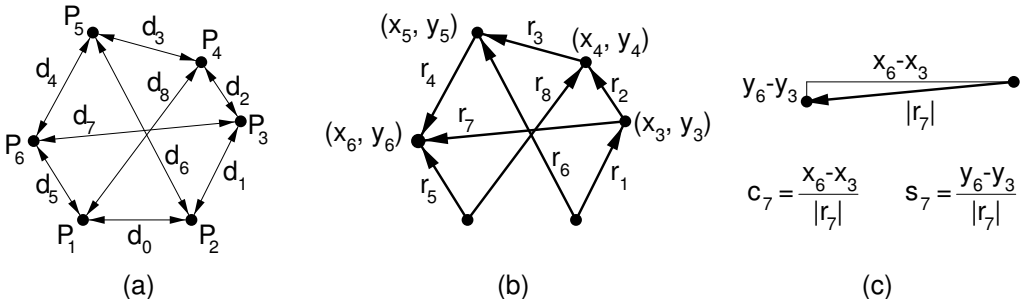


Abbildung B.4: Erläuterungen zur direkten Linearisierung des Schemas III. (a) Ausgangsmodell. (b) Transformiertes Modell mit den relevanten Richtungsvektoren sowie den zu berechnenden Koordinaten. (c) Beispiel für die Hilfsvariablen s_i und c_i, die die aktuellen Werte für den Sinus und den Cosinus eines Richtungsvektors r_i aufnehmen.

Basierend auf Gleichung B.16 lässt sich die Vorschrift für die Bestimmung der Schrittweite ΔX_k ermitteln, die in Gleichung B.15 benötigt wird:

$$\Delta X_k = -J^{-1}(X_k) \cdot F(X_k) \tag{B.17}$$

Zur Berechnung von ΔX_k kann zwar (wie formal in Gleichung B.17 beschrieben) die invertierte Jacobi-Matrix (z. B. mittels Gauß-Jordan-Verfahren, siehe [TS88]) ermittelt werden, allgemein stehen jedoch eine ganze Reihe weiterer Algorithmen für die Lösung des linearen Gleichungssystems zur Verfügung, u. a. der Gaußsche Algorithmus oder die QR-Zerlegung.

Das Beispiel

Zunächst wird das Modell aus den sechs Punkten und neun Abständen (Abbildung B.4(a)) zur Vereinfachung so transformiert, dass P_1 und P_2 im richtigen Abstand d_0 zueinander positioniert sind (Abbildung B.4(b)). Das so aufgestellte Modell hat acht Variablen (die Koordinaten der Punkte P_3 bis P_6). Diese Variablen müssen acht quadratischen Gleichungen genügen, die aus den acht verbleibenden Abstands-Constraints resultieren.

Die direkte Linearisierung (angewendet z. B. in [Hüs88]) geht davon aus, dass für kleine Koordinatenänderungen der Zusammenhang zwischen den Änderungen der Koordinaten und den Änderungen der Abstände linear ist. Zur Bestimmung des jeweiligen Einflussfaktors werden die partiellen Ableitungen gebildet. Dies soll anhand von r_7 erläutert werden (vgl. Abbildung B.4(c)). Die Länge von r_7 ist von den vier Variablen x_6, y_6, x_3 und y_3 (nicht linear!) abhängig:

$$|r_7| = \sqrt{(x_6 - x_3)^2 + (y_6 - y_3)^2} \tag{B.18}$$

Als partielle Ableitung der Länge von r_7 nach x_3 ergibt sich:

$$\frac{\partial |r_7|}{\partial x_3} = \frac{2 \cdot (x_6 - x_3) \cdot -1}{2 \cdot \sqrt{(x_6 - x_3)^2 + (y_6 - y_3)^2}} = -\frac{(x_6 - x_3)}{|r_7|} = -c_7 \tag{B.19}$$

Analog lassen sich die partiellen Ableitungen nach x_6, y_3 und y_6 bestimmen:

$$\frac{\partial |r_7|}{\partial x_6} = \frac{2 \cdot (x_6 - x_3) \cdot 1}{2 \cdot \sqrt{(x_6 - x_3)^2 + (y_6 - y_3)^2}} = \frac{(x_6 - x_3)}{|r_7|} = c_7 \tag{B.20}$$

$$\frac{\partial|r_7|}{\partial y_3} = \frac{2\cdot(y_6-y_3)\cdot -1}{2\cdot\sqrt{(x_6-x_3)^2+(y_6-y_3)^2}} = -\frac{(y_6-y_3)}{|r_7|} = -s_7 \tag{B.21}$$

$$\frac{\partial|r_7|}{\partial y_6} = \frac{2\cdot(y_6-y_3)\cdot 1}{2\cdot\sqrt{(x_6-x_3)^2+(y_6-y_3)^2}} = \frac{(y_6-y_3)}{|r_7|} = s_7 \tag{B.22}$$

Für kleine Änderungen kann also angenommen werden, dass:

$$\Delta|r_7| = -c_7\cdot\Delta x_3 - s_7\cdot\Delta y_3 + c_7\cdot\Delta x_6 + s_7\cdot\Delta y_6 \tag{B.23}$$

Da y_6 nahe y_3 liegt und somit s_7 fast 0 ist, haben Δy_6 und Δy_3 bei den in Abbildung B.4 gezeigten Punktpositionen kaum Einfluss auf $\Delta|r_7|$. Die Änderungen der x-Koordinaten haben hingegen großen Einfluss[19].

Nach der für r_7 gezeigten Vorgehensweise wird ein lineares Gleichungssystem aufgestellt, das beschreibt, wie sich die Verschiebung der Punkte P_3 bis P_6 auf die Längenänderungen $\Delta|r_1|$ bis $\Delta|r_8|$ auswirkt:

$$\begin{array}{lcrrrr}
\Delta|r_1| &=& c_1\cdot\Delta x_3 & +s_1\cdot\Delta y_3 & & \\
\Delta|r_2| &=& -c_2\cdot\Delta x_3 & -s_2\cdot\Delta y_3 & +c_2\cdot\Delta x_4 & +s_2\cdot\Delta y_4 \\
\Delta|r_3| &=& -c_3\cdot\Delta x_4 & -s_3\cdot\Delta y_4 & +c_3\cdot\Delta x_5 & +s_3\cdot\Delta y_5 \\
\Delta|r_4| &=& -c_4\cdot\Delta x_5 & -s_4\cdot\Delta y_5 & +c_4\cdot\Delta x_6 & +s_4\cdot\Delta y_6 \\
\Delta|r_5| &=& c_5\cdot\Delta x_6 & +s_5\cdot\Delta y_6 & & \\
\Delta|r_6| &=& c_6\cdot\Delta x_5 & +s_6\cdot\Delta y_5 & & \\
\Delta|r_7| &=& -c_7\cdot\Delta x_3 & -s_7\cdot\Delta y_3 & +c_7\cdot\Delta x_6 & +s_7\cdot\Delta y_6 \\
\Delta|r_8| &=& c_8\cdot\Delta x_4 & +s_8\cdot\Delta y_4 & &
\end{array} \tag{B.24}$$

Die gewünschten Längenänderungen $\Delta|r_1|$ bis $\Delta|r_8|$ sind bekannt. Sie ergeben sich aus den Differenzen der messbaren aktuellen Abstände zu den per Constraint definierten Abständen d_1 bis d_8, d. h. $\Delta|r_i| = d_i - |r_i| = -\delta_i$ für $i = 1\ldots 8$. Auch die aktuellen Werte von c_1 bis c_8 sowie s_1 bis s_8 können gemessen werden. Durch Lösen des Gleichungssystems B.24 lassen sich deshalb die vorzunehmenden Koordinatenänderungen Δx_3 bis Δx_6 und Δy_3 bis Δy_6 bestimmen. In Matrixform sieht das Gleichungssystem wie folgt aus:

$$\begin{bmatrix} -\delta_1 \\ -\delta_2 \\ -\delta_3 \\ -\delta_4 \\ -\delta_5 \\ -\delta_6 \\ -\delta_7 \\ -\delta_8 \end{bmatrix} = \begin{bmatrix} c_1 & s_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -c_2 & -s_2 & c_2 & s_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -c_3 & -s_3 & c_3 & s_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & -c_4 & -s_4 & c_4 & s_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & c_5 & s_5 \\ 0 & 0 & 0 & 0 & c_6 & s_6 & 0 & 0 \\ -c_7 & -s_7 & 0 & 0 & 0 & 0 & c_7 & s_7 \\ 0 & 0 & c_8 & s_8 & 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \Delta x_3 \\ \Delta y_3 \\ \Delta x_4 \\ \Delta y_4 \\ \Delta x_5 \\ \Delta y_5 \\ \Delta x_6 \\ \Delta y_6 \end{bmatrix} \tag{B.25}$$

Das selbe Gleichungssystem ergibt sich, wenn die Abstandsgleichungen nach dem in Anhang B.2 beschriebenen Formalismus (Reihenentwicklung nach Taylor und Abbruch nach der Ableitung

[19] Man kann an diesem Beispiel gut abschätzen, wann Änderungen „klein“ sind bzw. wann nicht. Änderungen, die sich in Richtung von r_i abspielen, wirken sich tatsächlich linear in $\Delta|r|$ aus. Auch wenn die Änderung in dieser Richtung ein Vielfaches von $|r_i|$ beträgt. Bezüglich der Korrektheit der Linearisierung sind solche Änderungen „klein“. Eine Änderung, die jedoch in senkrechter Richtung zu einem r_i ein Vielfaches von $|r_i|$ beträgt, ist sicher als nicht mehr „klein“ zu bezeichnen. Der Fehler zwischen dem (linearisiert) geschätzten $\Delta|r|$ und dem realen $\Delta|r|$ ist zu groß. Derartige Betrachtungen ermöglichen eine Schrittweitensteuerung, um so die Gefahr einer Divergenz der Nullstellensuche zu vermindern.

erster Ordnung) linearisiert werden. Als ausführlicheres Beispiel hierfür kann die in Anhang B.5 durchgeführte Linearisierung dienen. Eine Kurzform der Notation des Gleichungssystems lautet:

$$\Delta R = S \cdot \Delta X \tag{B.26}$$

ΔR umfasst die noch notwendigen Längenänderungen, S ist die Systemmatrix (bzw. die Jacobimatrix), die den aktuellen Zustand des Modells beschreibt, und ΔX ist der zu bestimmende Vektor der Punktverschiebungen, der sich z. B. als $\Delta X = S^{-1} \cdot \Delta R$ berechnen lässt.

Da die Linearisierung typischerweise nicht das eigentliche Modell repräsentiert, werden auch nach der Verschiebung der Punkte P_3 bis P_6 entsprechend der berechneten (bzw. geschätzten) notwendigen Offsets noch Fehler verbleiben. Deshalb ist die Berechnung der notwendigen Punktverschiebungen (entsprechend den aktuellen Werten von $\Delta|r_i|$, s_i und c_i) so lange durchzuführen, bis alle Fehler unter einer bestimmten Toleranzgrenze liegen oder die Berechnung als gescheitert abgebrochen wird.

Zur besseren Vermeidung von Divergenz ist es möglich, den auf Basis der Linearisierung berechneten Vektor ΔX nicht direkt als Änderungsvektor für die aktuelle Lösung zu verwenden. Vielmehr kann er als Hinweis für die lokal günstige Änderungsrichtung aufgefasst werden. Die Güte der sich auf Basis der Linearisierung ergebenden Schrittweite (Größe der Gesamtänderung $|\Delta X|$) sollte überprüft werden, um sicherzustellen, dass durch sie auch tatsächlich eine bessere Zwischenlösung erreicht wird. Es ist denkbar, dass *verschiedene Schrittweiten* in die ermittelte Richtung untersucht werden. Die Schrittweite mit der besten Lösung ist dann als Start für die nächste Iteration zu wählen. Der Mehraufwand hierfür ist vertretbar, denn zur Ermittlung der Güte einer bestimmten Schrittweite muss lediglich für jeden Constraint der aktuelle Fehler berechnet und zu einem Gesamtfehler aufsummiert werden[20]. Die Anzahl der sinnvollerweise zu untersuchenden Schrittweiten hängt vom Aufwand der Lösung des linearen Gleichungssystems ab. Je mehr Variablen es enthält, umso mehr Untersuchungen sind sinnvoll. Zur Steuerung der Wahl einer nächsten (auf ihre Güte zu untersuchenden) Schrittweite ist bei hinreichend vielen erlaubten Untersuchungen die Anwendung einer speziellen Heuristik sinnvoll. Diese Heuristik könnte die Schrittweitenwahl z. B. ähnlich dem in Abschnitt 3.6.3 beschriebenen hybriden Ansatz durchführen.

B.4 Homotopie

Der Ansatz

Beim Homotopie-Ansatz ist man bestrebt ein vorhandenes konsistentes Modell, z. B. den letzten konsistenten Zustand, über mehrere Homotopieschritte in das Modell zu überführen, das durch das zu lösende Gleichungssystem $F(X) = 0$ beschrieben wird. Hierbei wird ausgehend von der initialen Lösung X_0 über mehrere Homotopie-Schritte mit den Zwischenlösungen $X^1, X^2 \ldots X^h$ eine kontinuierliche Modelltransformation vorgenommen bis $F(X^h) = 0$ gefunden ist. Die Grundidee des Ansatzes beruht auf der Feststellung, dass Verfahren wie das Newtonsche Verfahren umso besser konvergieren, je dichter sie bereits an der Lösung liegen. Wenn also initial $F(X_0) = F_0$ gilt, so wird davon ausgegangen, dass die Lösungsfindung für $F(X^1) = H_1$ eine gute Konvergenz aufweist, falls H_1 nahe F_0 liegt. Die Modelltransformation kann z. B. linear erfolgen. Dann gilt bei h Homotopieschritten:

$$H_i = F_0 \cdot \frac{h - i}{h} \tag{B.27}$$

mit $i = 1 \ldots h$. Im letzten Schritt ist $H_h = 0$, wodurch die ursprünglich gesuchte Lösung beschrieben wird. Die Folge der Zwischenlösungen $X^1, X^2 \ldots X^h$ ergibt den sogenannten Homotopie-Pfad.

[20] Es kann z. B. auch die Summe der Fehlerquadrate verwendet werden.

Zur Berechnung der einzelnen Zwischenlösungen lässt sich z. B. wie in [LM96] das Newtonsche Verfahren einsetzen. Dort werden ca. zwanzig Homotopieschritte vorgeschlagen. Grundsätzlich können jedoch auch andere Verfahren zur Ermittlung von $X^1, X^2 \dots X^h$ herangezogen werden.

Das Beispiel

Zur Berechnung der Zwischenlösungen wurde auf das Newtonsche Verfahren zurückgegriffen, siehe Anhang B.3. Die zwanzig Homotopieschritte ($h = 20$) erfolgten für ein linear transformiertes Modell. Da das Modell durch die Werte der Abstände d_j mit $j = 1 \dots 8$ beschrieben wird, waren diese Werte kontinuierlich von den initial vorliegenden Werten $|r_j| = d_j^0$ in die per Constraint geforderten Werte $d_j = d_j^h$ zu überführen:

$$d_j^i = d_j^0 \cdot \frac{h-i}{h} + d_j \cdot \frac{i}{h}, i = 1...h \tag{B.28}$$

Die entsprechenden Werte wurden jeweils zur Berechnung der $\delta_1 \dots \delta_8$ benutzt, welche in Gleichung B.25 benötigt werden. Die Robustheit des Verfahrens kann im Allgemeinen durch die gezielte Verfolgung der Homotopiepfade noch verbessert werden, siehe z. B. [Fär09].

B.5 Minimierung der Summe der Fehlerquadrate

Der Ansatz

Ein Gleichungssystem $F(X) = 0$ wird in ein Optimierungsproblem umformuliert:

$$S = \sum_{i=1}^{n} (F_i(X))^2 \to Min \tag{B.29}$$

Für die Suche des Minimums werden die m partiellen Ableitungen $\frac{\partial S}{\partial x_i}$ von S nach den m Variablen $x_1 \dots x_m$ gebildet. Hier wird ersichtlich, dass auch ein initial überbestimmtes Gleichungssystem mit $n > m$ Gleichungen in ein vollbestimmtes Gleichungssystem mit m Gleichungen überführt wird. Zudem ist es möglich, auch bei Modellen ohne reelle Lösungen (z. B. wenn sich zwei Kreise nicht schneiden können) Näherungslösungen anzubieten. Die m Gleichungen:

$$\frac{\partial S}{\partial x_i}(x_1 \dots x_m) = 0 \tag{B.30}$$

sind wiederum typischerweise nichtlinear und müssen mit einem entsprechenden Verfahren, z. B. dem Newtonschen Verfahren, gelöst werden.

Das Beispiel

Für einen einzelnen Abstands-Constraint ist der aktuelle Fehler $\delta = |r| - d$ (bzw. die notwendige Abstandsänderung $\Delta|r| = d - |r| = -\delta$) abhängig von der per Constraint festgelegten Länge d und der aktuellen Länge (bzw. dem aktuellen Abstand) $|r|$:

$$\delta = |r| - d = \sqrt{(x_n - x_v)^2 + (y_n - y_v)^2} - d$$

Die Indizes n und v stehen für *„nach“* und *„von“* entsprechend der für den Abstand gewählten Ausrichtung (vgl. Abbildung B.4(b)). Die Unterscheidung der Indizes n und v ist wichtig, um in

den weiteren Berechnungen die Vorzeichen richtig zu wählen. Zur Minimierung der quadrierten Fehler δ^2 müssen die partiellen Ableitungen nach den vier Variablen x_n, x_v, y_n und y_v gebildet werden.

$$\delta^2 = (|r| - d)^2 = (\sqrt{(x_n - x_v)^2 + (y_n - y_v)^2} - d)^2$$

$$\frac{\partial(\delta^2)}{\partial x_n} = 2 \cdot (\sqrt{(x_n - x_v)^2 + (y_n - y_v)^2} - d) \cdot \frac{2 \cdot (x_n - x_v)}{2 \cdot \sqrt{(x_n - x_v)^2 + (y_n - y_v)^2}} = 2 \cdot (|r| - d) \cdot c = \delta_x \cdot 2$$

$$\frac{\partial(\delta^2)}{\partial y_n} = 2 \cdot (\sqrt{(x_n - x_v)^2 + (y_n - y_v)^2} - d) \cdot \frac{2 \cdot (y_n - y_v)}{2 \cdot \sqrt{(x_n - x_v)^2 + (y_n - y_v)^2}} = 2 \cdot (|r| - d) \cdot s = \delta_y \cdot 2$$

$$\frac{\partial(\delta^2)}{\partial x_v} = 2 \cdot (|r| - d) \cdot (-c) = -2 \cdot \delta \cdot c = -\delta_x \cdot 2$$

$$\frac{\partial(\delta^2)}{\partial y_v} = 2 \cdot (|r| - d) \cdot (-s) = -2 \cdot \delta \cdot s = -\delta_y \cdot 2$$

Da die partiellen Ableitungen in den folgenden Schritten gleich null gesetzt werden, sind die in allen Termen vorhandenen konstanten Faktoren 2 vernachlässigbar und wurden deshalb zur Vereinfachung ausgeklammert. Die Summe der acht Fehlerquadrate $S = \sum_{i=1}^{8} \delta_i^2$ wird partiell nach den acht Variablen abgeleitet und jede partielle Ableitung gleich null gesetzt. Die resultierenden Gleichungen bestehen jeweils aus einer Summe nichtlinearer Terme[21]:

$$\begin{array}{rcccccccccc}
\frac{\partial S}{\partial x_3} & = & \delta_{1x} & - & \delta_{2x} & - & \delta_{7x} & = & 0 & = & f_1 \\
\frac{\partial S}{\partial y_3} & = & \delta_{1y} & - & \delta_{2y} & - & \delta_{7y} & = & 0 & = & f_2 \\
\frac{\partial S}{\partial x_4} & = & \delta_{2x} & - & \delta_{3x} & + & \delta_{8x} & = & 0 & = & f_3 \\
\frac{\partial S}{\partial y_4} & = & \delta_{2y} & - & \delta_{3y} & + & \delta_{8y} & = & 0 & = & f_4 \\
\frac{\partial S}{\partial x_5} & = & \delta_{3x} & - & \delta_{4x} & + & \delta_{6x} & = & 0 & = & f_5 \\
\frac{\partial S}{\partial y_5} & = & \delta_{3y} & - & \delta_{4y} & + & \delta_{6y} & = & 0 & = & f_6 \\
\frac{\partial S}{\partial x_6} & = & \delta_{4x} & + & \delta_{5x} & + & \delta_{7x} & = & 0 & = & f_7 \\
\frac{\partial S}{\partial y_6} & = & \delta_{4y} & + & \delta_{5y} & + & \delta_{7y} & = & 0 & = & f_8
\end{array} \tag{B.31}$$

[21] Die partielle Ableitung nach x_3 lässt sich z. B. folgendermaßen beschreiben:
x_3 hat nur Einfluss auf die Richtungen r_1, r_2 und r_7. Deshalb sind nur die Ableitungen der entsprechenden Fehlerterme δ_1^2, δ_2^2 und δ_7^2 von Interesse. Die fünf anderen Fehlerterme werden bei der partiellen Ableitung nach x_3 zu null. Die Vorzeichen der Terme sind abhängig von der Ausrichtung der jeweiligen r_i. In r_1 entspricht x_3 dem x_n, deshalb ist $+\delta_{1x} = \delta_1 \cdot c_1$ zu verwenden. In r_2 und r_7 entspricht x_3 dem x_v, so dass $-\delta_{2x} = \delta_2 \cdot c_2$ und $-\delta_{7x} = \delta_7 \cdot c_7$ zu verwenden sind.

Zur iterativen Lösung des nichtlinearen Gleichungssystems erfolgt nun seine Linearisierung.[22] Entsprechend dem Satz von Taylor für mehrere Veränderliche v_i (hier $i = 1 \dots 8$) und Abbruch der Reihenbildung nach den Gliedern erster Ordnung gilt für jede der Gleichungen f_j mit $j = 1 \dots 8$:

$$f_j(v_1 + \Delta v_1, \dots v_8 + \Delta v_8) = f_j(v_1, \dots v_8) + \sum_{i=1}^{8} \Delta v_i \cdot \frac{\partial f_j}{\partial v_i}(v_1, \dots v_8) \tag{B.32}$$

Da $f_j = 0$ für alle $j = 1 \dots 8$ ist, kann die Gleichung B.32 umgestellt werden nach:

$$-\delta_j = -f_j(v_1, \dots v_8) = \sum_{i=1}^{8} \Delta v_i \cdot \frac{\partial f_j}{\partial v_i}(v_1, \dots v_8) \tag{B.33}$$

Hieraus ergibt sich die Matrixform des linearisierten Gleichungssystems:

$$\begin{bmatrix} -\delta_{1x} + \delta_{2x} + \delta_{7x} \\ -\delta_{1y} + \delta_{2y} + \delta_{7y} \\ -\delta_{2x} + \delta_{3x} - \delta_{8x} \\ -\delta_{2y} + \delta_{3y} - \delta_{8y} \\ -\delta_{3x} + \delta_{4x} - \delta_{6x} \\ -\delta_{3y} + \delta_{4y} - \delta_{6y} \\ -\delta_{4x} - \delta_{5x} - \delta_{7x} \\ -\delta_{4y} - \delta_{5y} - \delta_{7y} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_3}(x_3, \dots y_6) & \cdots & \frac{\partial f_1}{\partial y_6}(x_3, \dots y_6) \\ \vdots & & \vdots \\ \frac{\partial f_8}{\partial x_3}(x_3, \dots y_6) & \cdots & \frac{\partial f_8}{\partial y_6}(x_3, \dots y_6) \end{bmatrix} \cdot \begin{bmatrix} \Delta x_3 \\ \Delta y_3 \\ \Delta x_4 \\ \Delta y_4 \\ \Delta x_5 \\ \Delta y_5 \\ \Delta x_6 \\ \Delta y_6 \end{bmatrix} \tag{B.34}$$

Zur Bestimmung der Jacobimatrix sind die in den Gleichungen vorkommenden Terme der Form δ_x und δ_y also nochmals nach den acht Variablen partiell abzuleiten[23], was zu folgenden Termen führt:

$$\begin{array}{lllllll} \frac{\partial \delta_x}{\partial x_n} & = & \frac{\partial(\delta \cdot c)}{\partial x_n} & = & c \cdot c + \delta \cdot \frac{1 - c \cdot c}{|r|} & = & \delta_{xx} \\ \frac{\partial \delta_x}{\partial y_n} & = & \frac{\partial(\delta \cdot c)}{\partial y_n} & = & s \cdot c - \delta \cdot \frac{s \cdot c}{|r|} & = & \delta_{xy} \\ \frac{\partial \delta_y}{\partial x_n} & = & \frac{\partial(\delta \cdot s)}{\partial x_n} & = & c \cdot s - \delta \cdot \frac{c \cdot s}{|r|} & = & \delta_{yx} \quad = \quad \delta_{xy} \\ \frac{\partial \delta_y}{\partial y_n} & = & \frac{\partial(\delta \cdot s)}{\partial y_n} & = & s \cdot s + \delta \cdot \frac{1 - s \cdot s}{|r|} & = & \delta_{yy} \end{array} \tag{B.35}$$

Partielle Ableitungen nach x_v und y_v haben lediglich ein negatives Vorzeichen und werden deshalb hier nicht extra aufgeführt. Die Jacobimatrix hat die in Gleichung B.34 gezeigte Struktur. Entsprechend dieser Struktur sind die partiellen Ableitungen der Gleichungen B.31 in die Matrix

---

[22] Im hierzu alternativ einsetzbaren Gradientenverfahren ([RS89]) werden die partiellen Ableitungen direkt zur Berechnung des nächsten Suchschrittes eingesetzt. Für eine Variable v_i ergibt sich $v_{i+1} = v_i + \Delta v_i$ mit $\Delta v_i = -\frac{\partial S}{\partial v_i}(v_1, \dots v_8) \cdot \frac{S}{l}$ und $l = \sum_{k=1}^{8} (\frac{\partial S}{\partial v_k}(v_1, \dots v_8))^2$. Allerdings konvergiert dieses Verfahren nur linear.

[23] Anmerkung für diejenigen, die die Bildung der zweiten Ableitungen nachvollziehen wollen:
$\frac{\partial(|r|)}{\partial x_n} = c$ (entsprechend Gleichung B.20), $\frac{\partial(|r|)}{\partial y_n} = s$ (entsprechend Gleichung B.22)
$\frac{\partial \delta}{\partial x_n} = \frac{\partial(|r| - d)}{\partial x_n} = c$, $\frac{\partial \delta}{\partial y_n} = \frac{\partial(|r| - d)}{\partial y_n} = s$, $\frac{\partial(\frac{1}{|r|})}{\partial x_n} = -\frac{1}{|r| \cdot |r|} \cdot c$, $\frac{\partial(\frac{1}{|r|})}{\partial y_n} = -\frac{1}{|r| \cdot |r|} \cdot s$
$\frac{\partial c}{\partial x_n} = \frac{\partial((x_n - x_v) \cdot \frac{1}{|r|})}{\partial x_n} = \frac{1}{|r|} + (-\frac{c}{|r| \cdot |r|} \cdot (x_n - x_v)) = \frac{1}{|r|} - \frac{c \cdot c}{|r|} = \frac{1 - c \cdot c}{|r|}$
$\frac{\partial s}{\partial x_n} = \frac{\partial((y_n - y_v) \cdot \frac{1}{|r|})}{\partial x_n} = 0 + (-\frac{c}{|r| \cdot |r|} \cdot (y_n - y_v)) = -\frac{c \cdot s}{|r|}$
$\frac{\partial c}{\partial y_n} = \frac{\partial((x_n - x_v) \cdot \frac{1}{|r|})}{\partial y_n} = 0 + (-\frac{s}{|r| \cdot |r|} \cdot (x_n - x_v)) = -\frac{s \cdot c}{|r|} = \frac{\partial s}{\partial x_n}$
$\frac{\partial s}{\partial y_n} = \frac{\partial((y_n - y_v) \cdot \frac{1}{|r|})}{\partial y_n} = \frac{1}{|r|} + (-\frac{s}{|r| \cdot |r|} \cdot (y_n - y_v)) = \frac{1}{|r|} - \frac{s \cdot s}{|r|} = \frac{1 - s \cdot s}{|r|}$

einzusetzen. Die Vorzeichen der Terme müssen die gewählte Ausrichtung der Vektoren berücksichtigen (siehe Abbildung B.4). Die Jacobimatrix ergibt sich dann als:

$$\begin{bmatrix}
\begin{matrix}\delta_{1xx}\\ +\delta_{2xx}\\ +\delta_{7xx}\end{matrix} & \begin{matrix}\delta_{1xy}\\ +\delta_{2xy}\\ +\delta_{7xy}\end{matrix} & -\delta_{2xx} & -\delta_{2xy} & 0 & 0 & -\delta_{7xx} & -\delta_{7xy} \\
\begin{matrix}\delta_{1xy}\\ +\delta_{2xy}\\ +\delta_{7xy}\end{matrix} & \begin{matrix}\delta_{1yy}\\ +\delta_{2yy}\\ +\delta_{7yy}\end{matrix} & -\delta_{2xy} & -\delta_{2yy} & 0 & 0 & -\delta_{7xy} & -\delta_{7yy} \\
-\delta_{2xx} & -\delta_{2xy} & \begin{matrix}\delta_{2xx}\\ +\delta_{3xx}\\ +\delta_{8xx}\end{matrix} & \begin{matrix}\delta_{2xy}\\ +\delta_{3xy}\\ +\delta_{8xy}\end{matrix} & -\delta_{3xx} & -\delta_{3xy} & 0 & 0 \\
-\delta_{2xy} & -\delta_{2yy} & \begin{matrix}\delta_{2xy}\\ +\delta_{3xy}\\ +\delta_{8xy}\end{matrix} & \begin{matrix}\delta_{2yy}\\ +\delta_{3yy}\\ +\delta_{8yy}\end{matrix} & -\delta_{3xy} & -\delta_{3yy} & 0 & 0 \\
0 & 0 & -\delta_{3xx} & -\delta_{3xy} & \begin{matrix}\delta_{3xx}\\ +\delta_{4xx}\\ +\delta_{6xx}\end{matrix} & \begin{matrix}\delta_{3xy}\\ +\delta_{4xy}\\ +\delta_{6xy}\end{matrix} & -\delta_{4xx} & -\delta_{4xy} \\
0 & 0 & -\delta_{3xy} & -\delta_{3yy} & \begin{matrix}\delta_{3xy}\\ +\delta_{4xy}\\ +\delta_{6xy}\end{matrix} & \begin{matrix}\delta_{3yy}\\ +\delta_{4yy}\\ +\delta_{6yy}\end{matrix} & -\delta_{4xy} & -\delta_{4yy} \\
-\delta_{7xx} & -\delta_{7xy} & 0 & 0 & -\delta_{4xx} & -\delta_{4xy} & \begin{matrix}\delta_{4xx}\\ +\delta_{5xx}\\ +\delta_{7xx}\end{matrix} & \begin{matrix}\delta_{4xy}\\ +\delta_{5xy}\\ +\delta_{7xy}\end{matrix} \\
-\delta_{7xy} & -\delta_{7yy} & 0 & 0 & -\delta_{4xy} & -\delta_{4xy} & \begin{matrix}\delta_{4xy}\\ +\delta_{5xy}\\ +\delta_{7xy}\end{matrix} & \begin{matrix}\delta_{4yy}\\ +\delta_{5yy}\\ +\delta_{7yy}\end{matrix}
\end{bmatrix} \tag{B.36}$$

Konvergenz vorausgesetzt kann das Gleichungssystem B.31 iterativ z. B. über die jeweilige Linearisierung in Form eines Gleichungssystems entsprechend B.34 gelöst werden. Zur Vermeidung der Divergenz ist auch hier eine Schrittweitensteuerung (siehe Anhang B.3) sinnvoll.

B.6 Approximative Linearisierung

Der Ansatz

Die approximative Linearisierung arbeitet ähnlich der direkten Linearisierung. Der Unterschied besteht in der verwendeten Systemmatrix. Es werden nicht die partiellen Ableitungen der Jacobimatrix genutzt, sondern nur geschätzte Werte für diese Ableitungen. Der Algorithmus startet mit einer möglichst genauen Schätzung S_0 der Systemmatrix. Wenn vorhanden, lässt sich hier die Jacobi-Matrix verwenden. In [TS88] wird als anderes Extrem auch die Verwendung der Einheitsmatrix als S_0 gezeigt. Wie bei der direkten Linearisierung erfolgt zunächst die Ermittlung:

1. der aktuellen Funktions- bzw. Fehlerwerte F_0 an der Stelle der Startwerte X_0,

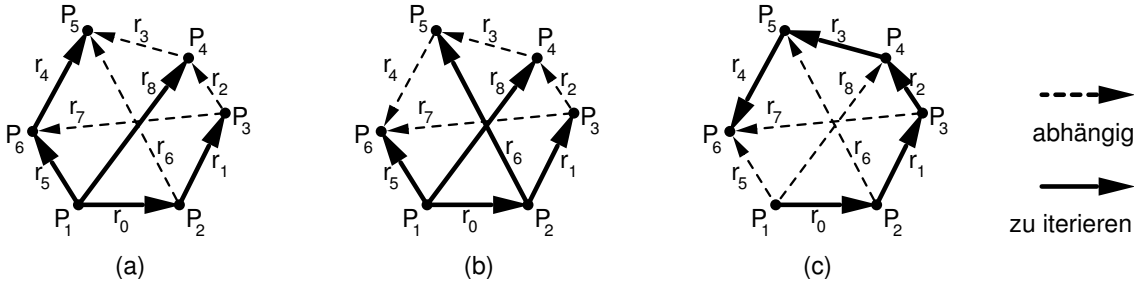


Abbildung B.5: Einige Möglichkeiten der Wahl von zu iterierenden Richtungsvektoren bei der Verschiebung der Punktmenge nach Solano. (a) Entsprechend den Veröffentlichungen von Solano und Brunet. (b) So, dass eine Konvergenz wie bei der direkten Linearisierung möglich ist. (c) Eine der weiteren Alternativen.

2. der im Suchraum auszuführenden Bewegung D_0 durch Lösung des linearen Gleichungssystems $-F_0 = S_0 \cdot D_0$,
3. der neuen Stelle $X_1 = X_0 + D_0$ und
4. der Funktionswerte F_1 an der Stelle X_1.

Die Schätzung der nächsten anzuwendenden Matrix S_1 bzw. allgemein S_{t+1} erfolgt in jedem Iterationsschritt neu auf Basis der jeweils aktuellen Matrix S_t, des aktuellen Fehlervektors F_{t+1} und der aktuellen Schrittweite D_t:

$$S_{t+1} = S_t + \frac{F_{t+1} \cdot D_t}{D_t^T \cdot D_t} \tag{B.37}$$

Dies entspricht dem Sekantenverfahren (siehe Seite 80).[24] Nach Ermittlung von S_{t+1} kann die Iteration mit der Berechnung des neuen Suchschrittes D_{t+1} fortgesetzt werden. Diese multivariate Version des Sekantenverfahrens wird auch als Broyden-Verfahren bezeichnet.

Das Beispiel

Das Aufstellen der Gleichungen für das Beispiel erfolgt wie in Anhang B.3 beschrieben. Als initiale Systemmatrix konnte die Jacobi-Matrix verwendet werden. In den weiteren Suchschritten wurde die Systemmatrix nach Gleichung B.37 berechnet.

B.7 Verschiebung der Punktmenge nach Solano

Die Grundidee des Ansatzes

Solano und Brunet beschreiben in einer Reihe von Veröffentlichungen (u. a. [SB98b], [SB98a] und [SB00]) einen Ansatz, der nicht auf Punktpositionen P_i sondern auf Richtungsvektoren r_i bzw.

[24] Zur Erläuterung sei Gleichung B.37 für die univariate Nullstellensuche umgeformt, wobei s_{t+1} der skalare Wert für die Schätzung des nächsten Anstieges ist:

$$S_{t+1} = s_{t+1} = s_t + \frac{f_{t+1} \cdot d_t}{d_t \cdot d_t} = s_t + \frac{f_{t+1}}{d_t} = -\frac{f_t}{d_t} + \frac{f_{t+1}}{d_t} = \frac{f_{t+1} - f_t}{d_t} \tag{B.38}$$

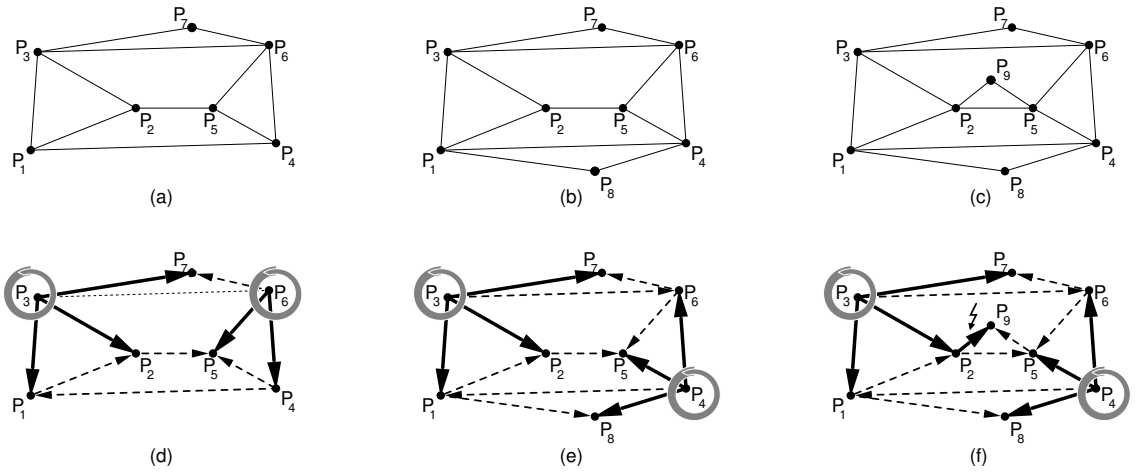


Abbildung B.6: Modelle (a), (b) und (c) sowie jeweils eine mögliche Auswahl (d), (e) und (f). In (d) können zwei Punkte gefunden werden, die verbunden sind und von denen aus sich alle anderen Punkte erreichen lassen. In (e) kann ein solches Paar nicht gefunden werden, aber es ist z. B. möglich von P_3 und P_4 aus die anderen Punkte zu erreichen. Für das Modell in (c) ist auch das nicht mehr möglich.

pseudo-normiert[25] $R_i = \frac{r_i}{d_i}$ zwischen den Punkten P_i basiert. Neben der Nutzung von Richtungsvektoren ist für den Ansatz von Solano und Brunet die Berücksichtigung von Skalarprodukten $b_i = R_i \cdot R_i$ zwischen diesen Richtungsvektoren grundlegend. Die Beschreibung des Ansatzes ist jedoch nicht vollständig, so wird insbesondere nicht gezeigt:

1. wie die Menge der Richtungsvektoren in die zu iterierenden und die abhängigen Richtungsvektoren aufzuteilen ist (Abbildung B.5 verdeutlicht unterschiedliche Möglichkeiten) und
2. wie die b_i zu wählen sind. Solano und Brunet gehen lediglich auf das Vorzeichen der b_i näher ein[26].

Für einige Modelle lässt sich zeigen, dass eine bestimmte Auswahl der zu iterierenden Richtungsvektoren sowie der Werte für die b_i zum selben Gleichungssystem führt, welches im Zuge einer direkten Linearisierung aufgestellt werden würde. Andererseits kann der Algorithmus aber auch so implementiert werden, dass die Suche nur langsam konvergiert, wodurch er eher den Charakter einer Heuristik trägt.

Umsetzung des Ansatzes ähnlich der direkten Linearisierung

Die unklare Beschreibung des Ansatzes von Solano und Brunet soll nun so ergänzt werden, dass er für bestimmte Modelle gleich der direkten Linearisierung bei iterierten Punktverschiebungen (Anhang B.3) arbeitet. Zugleich soll die in [SB98b] für ein Beispiel gezeigte Gleichungssystemstruktur hergeleitet werden.

---

[25] Eine herkömmliche Normierung ist $\frac{r_i}{|r_i|}$. Es wäre zu untersuchen, welche Vorteile die von den Autoren gewählte Art der Normierung bringt.

[26] Solano und Brunet gehen in ihren Ausführungen nur auf das Vorzeichen von b ein und versuchen zu beweisen, dass ihr Algorithmus bei richtiger Wahl des Vorzeichens stets konvergiert. Das Hauptproblem ihrer Beweisskizze liegt darin, dass sie es nicht schaffen, die Einhaltung der zulässigen Schrittweite bei zu großem Abstand formal zu beschreiben. Es wird lediglich ein „*must be scaled to guarantee small values*“ angedeutet. Ein Beweis für die Konvergenz wird auf diese Art und Weise nicht erbracht. Auch die Wahl der Werte für die einzelnen b_i wird durch die Autoren nicht näher erläutert, in [SB98b] und [SB00] heißt es nur „*with these values it is possible to tune the different targets*“. Unter Berücksichtigung der Überlegungen aus Abschnitt B.2 lassen sich jedoch leicht günstige Werte für die einzelnen b_i finden.

Zunächst erfolgt eine Betrachtung, wann sich die Struktur der Gleichungen B.24 beim Iterieren von Richtungen erreichen lässt. Jede Gleichung repräsentiert einen Abstands-Constraint. Dieser ist grundsätzlich abhängig von den beiden Punktpositionen zwischen denen der Abstand definiert wurde. Einer von ihnen kann fixiert sein, so dass nur der andere Punkt in die Gleichung eingeht. Werden wie in Anhang B.3 die Punktverschiebungen iteriert, ist dementsprechend jede Gleichung von ein oder zwei Punkten abhängig. Sollen nun Änderungen von Richtungsvektoren iteriert werden, so kann immer dann die gleiche Struktur der Gleichungen erreicht werden, wenn jeder nicht konstante Punkt höchstens von einem zu iterierenden Richtungsvektor abhängt. Dies trifft auf Abbildung B.5(b) zu, auf (a) und (c) jedoch nicht. Um eine Auswahl der zu iterierenden Richtungsvektoren entsprechend Abbildung B.5(b) zu finden, kann auf eine „Breite zuerst"-Suchstrategie zurückgegriffen werden. Sie muss von den fixen Punkten ausgehen (in der Abbildung sind das P_1 und P_2). Für jeden der anderen Punkte wird nach einem Abstands-Constraint c_q gesucht, durch den der Punkt mit einem fixen Punkt verbunden ist. Die entsprechenden Richtungen r_q sind zu iterieren. Falls es keine direkte Verbindung von einem Punkt zu einem fixen Punkt gibt, so hängt seine Position von mehreren zu iterierenden Richtungsvektoren ab und es ergibt sich eine andere Struktur der Gleichungen. Entsprechende Beispiele sind in Abbildung B.6 gezeigt. Wichtig ist hierbei, dass nur in Abbildung B.6(a) bzw. (d) zwischen den fixen Punkten ein Abstand definiert ist. Dies ermöglicht die beiden Punkte entsprechend dem ursprünglichen Modell zueinander zu positionieren. Für (b) kann ein solches Paar nicht gefunden werden. Zwar sind alle Punkte von P_3 und P_4 erreichbar, jedoch ist nicht bekannt, in welchem Abstand P_3 und P_4 voneinander fixiert werden müssen. Deshalb haben die Gleichungen für die in den Abbildungen B.6(b) und (c) gezeigten Modelle bei einer Iteration von Richtungen mehr Terme als bei einer Iteration von Punktverschiebungen.

Für die weiteren Überlegungen soll angenommen werden, dass im Modell zwei Punkte gefunden werden können, deren Abstand bekannt ist und von denen aus alle anderen erreicht werden können. Die entsprechenden Richtungsvektoren werden für die Iteration genutzt, jedoch nicht direkt, sondern nach [SB98b] als pseudo-normierte Richtungsvektoren $R_i = \frac{r_i}{d_i}$. Entsprechend sind die Skalarprodukte zu bilden[27]:

$$b_i = R_i \cdot \Delta R_i = x_{Ri} \cdot x_{\Delta Ri} + y_{Ri} \cdot y_{\Delta Ri} \tag{B.39}$$

Eine Umformung von Gleichung B.39 führt zu:

$$b_i = \frac{x_{ri}}{d_i} \cdot \frac{x_{\Delta ri}}{d_i} + \frac{y_{ri}}{d_i} \cdot \frac{y_{\Delta ri}}{d_i} \tag{B.40}$$

$$b_i \cdot d_i \cdot d_i = x_{ri} \cdot x_{\Delta ri} + y_{ri} \cdot y_{\Delta ri} \tag{B.41}$$

Andererseits lässt sich aus Gleichung B.11 herleiten:

$$e_i \cdot |r_i| = c_i \cdot |r_i| \cdot x_{\Delta ri} + s_i \cdot |r_i| \cdot y_{\Delta ri} = x_{ri} \cdot x_{\Delta ri} + y_{ri} \cdot y_{\Delta ri} \tag{B.42}$$

Die Gegenüberstellung der Gleichungen B.41 und B.42 zeigt, dass

$$b_i \cdot d_i \cdot d_i = e_i \cdot |r_i| \tag{B.43}$$

Somit sind entsprechend der direkten Linearisierung die b_i folgendermaßen zu wählen:

[27] Es sei darauf hingewiesen, dass $x_{\Delta r}$ (die x-Komponente des Richtungsvektors Δr) stets gleich Δx_r (der Änderung der x-Komponente des Richtungsvektors r) ist. Analog gilt $y_{\Delta r} = \Delta y_r$. Hier wird $(x_{\Delta r}, y_{\Delta r})$ als Notation verwendet, um entsprechend Abbildung B.2 explizit auf die Verwendung des Richtungsvektors Δr bei der Bildung des Skalarproduktes zu verweisen.

$$b_i = \frac{e_i \cdot |r_i|}{d_i \cdot d_i} \tag{B.44}$$

Die Pseudo-Normierung lässt sich demzufolge für die iterierten Richtungsvektoren derartig in den b_i berücksichtigen, dass sich Skalarprodukte wie bei der direkten Linearisierung ergeben. Auch bei einem abhängigen Richtungsvektor r_i^* ist das möglich, wenn r_i^* von einem zu iterierenden r_{iv} und einem zu iterierenden r_{in} abhängt, wobei r_{iv} von einem fixen Punkt zum Anfang von r_i^* geht und r_{in} zum Ende[28]. Es gilt:

$$r_i^* = r_{in} - r_{iv} \tag{B.45}$$

beziehungsweise wegen der Separierbarkeit in x- und y-Anteile:

$$\begin{array}{rcl} x_i^* & = & x_{in} - x_{iv} \\ y_i^* & = & y_{in} - y_{iv} \end{array} \tag{B.46}$$

Hierbei wird davon ausgegangen, dass r_{in} und r_{iv} vom fixen Punkt weggerichtet sind, ansonsten wären die Vorzeichen umzukehren. Angewendet auf Gleichung B.41 folgt:

$$b_i \cdot d_i \cdot d_i = x_{ri}^* \cdot (x_{\Delta rin} - x_{\Delta riv}) + x_{ri}^* \cdot (y_{\Delta rin} - y_{\Delta riv}) \tag{B.47}$$

$$b_i = \frac{x_{ri}^*}{d_i \cdot d_i} \cdot (x_{\Delta rin} - x_{\Delta riv}) + \frac{y_{ri}^*}{d_i \cdot d_i} \cdot (y_{\Delta rin} - y_{\Delta riv}) \tag{B.48}$$

$$b_i = \frac{x_{Ri}^*}{d_i} \cdot (x_{\Delta rin} - x_{\Delta riv}) + \frac{y_{Ri}^*}{d_i} \cdot (y_{\Delta rin} - y_{\Delta riv}) \tag{B.49}$$

Dies führt zu Gleichungen, wie sie von Solano und Brunet in ihrem Beispiel angegeben werden:

$$b_i = x_{Ri}^* \cdot (x_{\Delta Rin} \cdot \frac{d_{in}}{d_i} - x_{\Delta Riv} \cdot \frac{d_{iv}}{d_i}) + y_{Ri}^* \cdot (y_{\Delta Rin} \cdot \frac{d_{in}}{d_i} - y_{\Delta Riv} \cdot \frac{d_{iv}}{d_i}) \tag{B.50}$$

Das Beispiel

Im Beispiel der nach Schema III versteiften Menge von sechs Punkten werden statt der Koordinaten der Punkte P_3 bis P_6 die Richtungen der Vektoren r_1, r_5, r_6 und r_8 (bzw. jeweils die x- und y-Koordinaten der pseudo-normierten Vektoren) iteriert[29]. Die aktuellen Richtungen r_2, r_3, r_4 und r_7 lassen sich aus den vier iterierten Richtungen nach folgendem Schema ableiten (vgl. auch Abbildung B.7(b)):

$$\begin{array}{rcrcrcr} r_2 & = & r_8 & - & r_0 & - & r_1 \\ r_3 & = & r_0 & + & r_6 & - & r_8 \\ r_4 & = & r_5 & - & r_0 & - & r_6 \\ r_7 & = & r_5 & - & r_0 & - & r_1 \end{array} \tag{B.51}$$

[28] Wäre der Anfangs- oder der Endpunkt fix so würde bei diesem Ansatz (die Anzahl der zu iterierenden Richtungsvektoren ist gleich der Anzahl der zu berechnenden Punkte) dieser Richtungsvektor sinnvollerweise selbst iteriert werden.

[29] In ihren Veröffentlichungen iterieren Solano und Brunet r_0, r_1, r_4, r_5 und r_8, also zehn Variablen, wobei eine „beliebig auf einen kleinen Wert gesetzt werden kann“ [SB98b, Seite 14]. Wegen der besseren Vergleichbarkeit mit den anderen Verfahren wird auch in diesem Abschnitt das einfachere Modell mit fixem P_2 angenommen, so dass r_0 nicht iteriert werden muss.

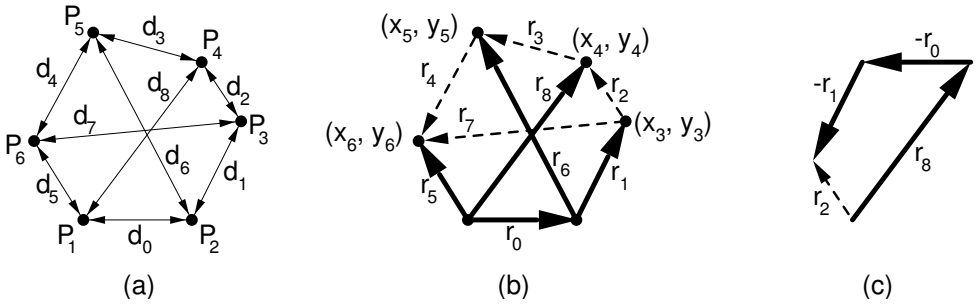


Abbildung B.7: Auswahl der zu iterierenden Richtungen für das Schema III

P_1 und P_2 werden wieder als gegeben angenommen, wobei P_2 auf gleicher Höhe zu P_1 im korrekten Abstand d_o positioniert ist. r_0 ist somit konstant und $x_{\Delta r0} = y_{\Delta r0} = 0$. Für die Änderungen ergibt sich:

$$\begin{array}{rcl} \Delta r_2 &=& \Delta r_8 - \Delta r_1 \\ \Delta r_3 &=& \Delta r_6 - \Delta r_8 \\ \Delta r_4 &=& \Delta r_5 - \Delta r_6 \\ \Delta r_7 &=& \Delta r_5 - \Delta r_1 \end{array} \tag{B.52}$$

Durch Berücksichtigung von $\Delta r_i = \Delta R_i \cdot d_i$ lassen sich die Gleichungen B.52 umformen:

$$\begin{array}{rcl} \Delta R_2 &=& \Delta R_8 \cdot \frac{d_8}{d_2} - \Delta R_1 \cdot \frac{d_1}{d_2} \\ \Delta R_3 &=& \Delta R_6 \cdot \frac{d_6}{d_3} - \Delta R_8 \cdot \frac{d_8}{d_3} \\ \Delta R_4 &=& \Delta R_5 \cdot \frac{d_5}{d_4} - \Delta R_6 \cdot \frac{d_6}{d_4} \\ \Delta R_7 &=& \Delta R_5 \cdot \frac{d_5}{d_7} - \Delta R_1 \cdot \frac{d_1}{d_7} \end{array} \tag{B.53}$$

Entsprechend den Gleichungen B.39, B.50 und B.44 sieht das zu lösende Gleichungssystem wie folgt aus:

$$\begin{bmatrix} \frac{e_1 \cdot |r_1|}{d_1 \cdot d_1} \\ \frac{e_2 \cdot |r_2|}{d_2 \cdot d_2} \\ \frac{e_3 \cdot |r_3|}{d_3 \cdot d_3} \\ \frac{e_4 \cdot |r_4|}{d_4 \cdot d_4} \\ \frac{e_5 \cdot |r_5|}{d_5 \cdot d_5} \\ \frac{e_6 \cdot |r_6|}{d_6 \cdot d_6} \\ \frac{e_7 \cdot |r_7|}{d_7 \cdot d_7} \\ \frac{e_8 \cdot |r_8|}{d_8 \cdot d_8} \end{bmatrix} = S \cdot \begin{bmatrix} x_{\Delta R1} \\ y_{\Delta R1} \\ x_{\Delta R8} \\ y_{\Delta R8} \\ x_{\Delta R6} \\ y_{\Delta R6} \\ x_{\Delta R5} \\ y_{\Delta R5} \end{bmatrix} \tag{B.54}$$

mit

$$S = \begin{bmatrix} x_{R1} & y_{R1} & 0 & 0 & 0 & 0 & 0 & 0 \\ -x_{R2} \cdot \frac{d_1}{d_2} & -y_{R2} \cdot \frac{d_1}{d_2} & x_{R2} \cdot \frac{d_8}{d_2} & y_{R2} \cdot \frac{d_8}{d_2} & 0 & 0 & 0 & 0 \\ 0 & 0 & -x_{R3} \cdot \frac{d_8}{d_3} & -y_{R3} \cdot \frac{d_8}{d_3} & x_{R3} \cdot \frac{d_6}{d_3} & y_{R3} \cdot \frac{d_6}{d_3} & 0 & 0 \\ 0 & 0 & 0 & 0 & -x_{R4} \cdot \frac{d_6}{d_4} & -y_{R4} \cdot \frac{d_6}{d_4} & x_{R4} \cdot \frac{d_5}{d_4} & y_{R4} \cdot \frac{d_5}{d_4} \\ 0 & 0 & 0 & 0 & 0 & 0 & x_{R5} & y_{R5} \\ 0 & 0 & 0 & 0 & x_{R6} & y_{R6} & 0 & 0 \\ -x_{R7} \cdot \frac{d_1}{d_7} & -y_{R7} \cdot \frac{d_1}{d_7} & 0 & 0 & 0 & 0 & x_{R7} \cdot \frac{d_5}{d_7} & y_{R7} \cdot \frac{d_5}{d_7} \\ 0 & 0 & x_{R8} & y_{R8} & 0 & 0 & 0 & 0 \end{bmatrix} \tag{B.55}$$

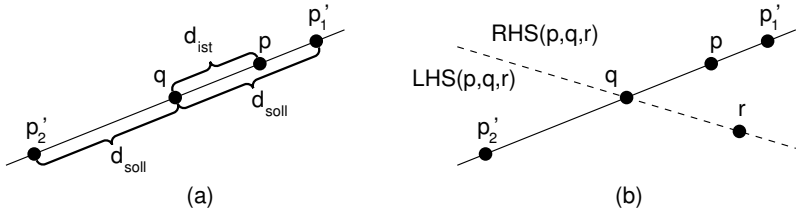


Abbildung B.8: Verschiebung eines Punktes von der aktuellen Position p nach p', um einen Abstands-Constraint zu erfüllen. In (a) wird der aktuelle Abstand d_{ist} zwischen den Positionen p und q gezeigt sowie zwei Positionen p'_1 und p'_2, die auch auf der Geraden liegen und den geforderten Abstand d_{soll} erfüllen. In (b) ist zudem eine Position r zu sehen. Um einen *RHS*-Hinweis zu erfüllen, sollte p'_1 gewählt werden und bei einem *LHS*-Hinweis p'_2.

Abschließende Bemerkung

Bisher wurden von Solano und Brunet nur einfache Modelle betrachtet und auch in der vorliegenden Arbeit konnten noch keine weiteren Untersuchungen der Grundidee von Solano und Brunet an komplexeren Modellen vorgenommen werden. Ein ausführlicher Vergleich der Iteration von Punktverschiebungen und Richtungsvektoränderungen für komplexere Modelle steht noch aus, wie auch die Untersuchung des Einflusses der Normalisierungsart auf die Konvergenzeigenschaften.

Zusammenfassend lässt sich bisher sagen, dass die Beschreibung des Ansatzes unterschiedlichste Realisierungsmöglichkeiten zulässt, die von einer langsam konvergierenden Heuristik (bei der Wahl der b_i wird nur auf das Vorzeichen geachtet) bis zu einem sehr schnell konvergierenden Algorithmus (die Wahl der b_i erfolgt analog zur direkten Linearisierung auf Basis der Punktverschiebungen) gehen. Interessant ist an dieser Stelle ein Vergleich der in den Veröffentlichungen beschriebenen Resultate. Während in [SB98b, Abbildung 20] noch über 100 Iterationsschritte bis zum Erreichen einer Genauigkeit von 10^{-1} angegeben wurden (die Anfangsfehler lagen in einer Größenordnung von ± 1), sind es in [SB00, Abbildung 7] schon weniger als zehn Iterationsschritte. Möglicherweise ist der Algorithmus, der in [SB00, Abbildung 7] zur Berechnung benutzt wurde, äquivalent zur direkten Linearisierung. Es wäre dann der Wechsel von einer Heuristik (man beachte die sehr langsame Konvergenz) zu einem etablierten mathematischen Verfahren vollzogen worden.

Die von Solano und Brunet in ihren Arbeiten gegebene Beweisskizze zur Konvergenz besitzt keinen praktischen Nutzen. Wie gezeigt werden konnte, lassen sich durch geeignete Wahl der iterierten Richtungen sowie der Werte der b_i für einige Modelle Konvergenzeigenschaften erreichen, die für Verfahren typisch sind, welche auf direkter Linearisierung beruhen. Dies beinhaltet neben der schnellen Konvergenz auch die Möglichkeit der Divergenz. Letztlich ist es Aufgabe der Schrittweitensteuerung einen geeigneten Kompromiss zwischen Geschwindigkeit und Robustheit zu finden.

B.8 Verschiebung je eines Punktes nach Schorn

Schorn beschreibt in [Sch93] einen Ansatz, der das Einstellen eines Gleichgewichtszustandes in einem Federsystem nachempfindet, um so iterativ die initial falschen Abstände den per Constraint geforderten Abständen anzunähern (vgl. Erläuterungen auf Seite 122). Ist für einen Punkt P, der sich aktuell an der Position p befindet, nur ein Abstand definiert (und aktuell nicht erfüllt), so wird seine neue Position p' auf der durch P und den anderen Punkt Q definierten Geraden gesucht. $d_{soll}(p,q)$ sei der per Constraint definierte Abstand zwischen den zwei Punkten und $d_{ist}(p,q)$ der aktuell gemessene Abstand. Die Berechnung von p' erfolgt nach Schorn mit:

$$p' = p + (p - q) \cdot (\frac{d_{soll}(p,q)}{d_{ist}(p,q)} - 1) \tag{B.56}$$

Allgemein hat P insgesamt n Abstands-Constraints. Zur Berechnung der Gesamtbewegung von P verwendet Schorn das arithmetische Mittel der Bewegungen, die aus den nicht erfüllten Abstands-Constraints zu den Punkten q_i resultieren. Es gilt:

$$p' = p + \lambda \cdot \frac{\sum_{i=1}^{n} (p - q_i) \cdot (\frac{d_{soll}(p,q_i)}{d_{ist}(p,q_i)} - 1)}{k} \tag{B.57}$$

wobei k die Anzahl der nicht erfüllten Abstände (bzw. der nicht entspannten angrenzenden Federn[30]) angibt. Mittels λ lässt sich die Konvergenzgeschwindigkeit steuern. In der Arbeit wird $\lambda = 2,7$ als günstig genannt. Es ist zu beachten, dass ein zu großes λ schnell zur Divergenz führt.

Die Auswahl von Mehrfachlösungen wird in [Sch93] durch die Vergabe von *XHS*-Hinweisen (dort *chirality constraints* genannt) gesteuert. Zur Berücksichtigung der Hinweise nutzt Schorn aus, dass es auf der Geraden durch P und Q jeweils zwei Positionen gibt, die den Abstands-Constraint erfüllen. Wenn zwischen den Punkten P, Q und R ein *XHS*-Hinweis definiert ist, dann erfüllt grundsätzlich eine der beiden Positionen diesen *XHS*-Hinweis (siehe Abbildung B.8(b)). Allerdings kann es vorkommen, dass für P und Q mehrere *XHS*-Hinweise definiert sind (zu verschiedenen anderen Punkten). Für einen solchen Fall wurden von Schorn verschiedene Strategien untersucht. Bewährt hat sich ein Seitenwechsel, falls auf der aktuellen Seite kein Hinweis erfüllt ist oder falls auf der aktuellen Seite weniger Hinweise erfüllt sind als auf der anderen Seite.

Wie erläutert wird jede Punktverschiebung lokal vorgenommen, d. h. unter Betrachtung des Punktes und seiner Constraints. Unabhängig davon lassen sich pro Iterationsschritt jedoch auch mehrere Punkte gleichzeitig verschieben. Grundsätzlich kann die Iteration jeweils erfolgen:

- über die Verschiebung nur eines Punktes (z. B. der, an den die größte resultierende Kraft angreift),
- über die Verschiebung einer Menge von ausgewählten Punkten (z. B. solche mit überdurchschnittlich[31] großen resultierenden Kräften) oder
- über die gleichzeitige Verschiebung aller Punkte.

In [Sch93] werden pro Iterationsschritt alle Punkte gleichzeitig entsprechend den lokal berechneten Kräften (bzw. Positionsänderungen) verschoben.

B.9 Verschiebung von je zwei Punkten

Die Betrachtung jeweils eines Abstands-Constraints führt zur Verschiebung der beiden durch ihn referenzierten Punkte P_a und P_b. Sie werden typischerweise auf der durch sie definierten Gerade G_{ab} so verschoben, dass der Constraint teilweise oder vollständig erfüllt ist. Wenn beide Punkte gleichmäßig verschoben werden sollen, wird der Punkt $P_m = \frac{P_a + P_b}{2}$ durch die Verschiebungen nicht geändert.

Solano und Brunet beschreiben in [SB98b], [SB98a] und [SB00] einen entsprechenden *one-constraint relaxation solver*. Je nach Implementierung entspricht das resultierende Verhalten mehr oder weniger dem der Verschiebung eines Punktes nach Schorn (siehe Anhang B.8). Der Unterschied liegt also eher in der Herleitung der einzelnen Punktverschiebungen als im erreichbaren Resultat.

---

[30] Die Festlegung, ab wann eine Feder als entspannt gilt, sollte mit dem Abbruchkriterium für die Simulation harmonieren. Schorn gibt einen tolerierten Fehler von durchschnittlich einem Prozent an. In einer Implementierung muss sichergestellt werden, dass stets $k \geq 1$ ist.

[31] Hier besteht die Möglichkeit der Wahl des Operators zur Mittelwertberechnung. Neben arithmetischem und geometrischem Mittel bieten sich z. B. auch parametrisierte Operatoren wie der Min-Max-Operator oder der algebraische Produkt-Summe-Operator an (siehe z. B. [MMSW93]).

Literaturverzeichnis

[ABMP91] H. Alt, N. Blum, K. Mehlhorn und M. Paul: *Computing a Maximum cardinality Matching in a Bipartite Graph in Time* $O(n^{1.5}\sqrt{m/log\ n})$. Information Processing Letters, 37(4):237–240, Februar 1991.

[ADH98] Armen S. Asratian, Tristan M. J. Denley und Roland Häggkvist: *Bipartite Graphs and their Applications*. Cambridge University Press, 1998.

[Adl06] August Adler: *Theorie der geometrischen Konstruktionen*. G. J. Göschensche Verlagshandlung, Leipzig, 1906.

[Bar86] Hans-Jochen Bartsch: *Mathematische Formeln*. VEB Fachbuchverlag Leipzig, 21. Auflage, 1986.

[Bar87] Lee Alton Barford: *A Graphical, Language-based Editor for Generic Solid Models represented by Constraints*. Doktorarbeit, Cornell University, Department of Computer Science, März 1987.

[BBD99] Torsten Brix, Beat D. Brüderlin und Ulf Döring: *Conditional Constraints in Conceptual Design*. In: *Proceedings of 32st ISATA*, Seiten 441–448, Wien, Juni 1999.

[BBDH98] Torsten Brix, Beat D. Brüderlin, Ulf Döring und Günter Höhne: *Analysis of solution principles in mechanical engineering by means of geometric constraint solving*. In: *Proceedings of 31st ISATA*, Seiten 387–393, Düsseldorf, Juni 1998.

[BBDH01] Torsten Brix, Beat D. Brüderlin, Ulf Döring und Günter Höhne: *Feature- and constraint-based design of solution principles*. In: *Proceedings of the International Conference on Engineering Design, ICED 01*, Seiten 613–620, Glasgow, August 2001.

[BBDH02] Torsten Brix, Beat D. Brüderlin, Ulf Döring und Günter Höhne: *Representation and Analysis of Solution Principles*. In: *Proceedings of CAD 2002*, Seiten 45–55, Dresden, März 2002.

[BBDO00] David Beier, Beat D. Brüderlin, Ulf Döring und Holger Oestreich: *Programmable features for geometric modeling*. In: *Proceedings of 33st ISATA*, Dublin, September 2000.

[BBDO02] David Beier, Beat D. Brüderlin, Ulf Döring und Holger Oestreich: *Programmable features for CAD modeling*. In: *Proceedings of CAD 2002*, Seiten 161–170, Dresden, März 2002.

[BBH+02] Torsten Brix, Beat D. Brüderlin, Günter Höhne, Michael Reeßing und Gerhard Wolf: *Feature- und constraint-basierte Modellierung auf der Funktions-, Prinzip- und Gestaltebene*. In: Heinrich Kern (Herausgeber): *Tagungsband zum 47. Internationalen Wissenschaftlichen Kolloquium*, Ilmenau, 2002.

[BD03] Torsten Brix und Ulf Döring: *Constraint-basierte Berechnung kinematischer Geometrien.* In: *Proceedings of Dresden Symposium Geometry: constructive & kinematic, DSG 03*, Dresden, Februar 2003.

[BDKM00] Beat D. Brüderlin, Ulf Döring, Rüdiger Klein und Paul Michalik: *Declarative Geometric Modeling with Constraints.* In: A. Iwainsky (Herausgeber): *Proceedings of CAD 2000*, Seiten 377–393, Berlin, März 2000.

[BDR03] Torsten Brix, Ulf Döring und Michael Reeßing: *Multi-Stage Modeling in the Early Phases of Design.* In: *Proceedings of the International Conference on Engineering Design, ICED 03*, Seiten 279–280, Stockholm, August 2003. Eine Kurzversion (2 Seiten) wurde gedruckt. Die Vollversion (10 Seiten) ist auf CD beigefügt (Artikel-Nr.: 1076).

[Ben03] Dan Bennett: *Exploring Geometry With the Geometer's Sketchpad.* Key Curriculum Press, Emeryville, California, USA, März 2003. www.keypress.com/sketchpad/index.html.

[Ber96] Roland Berling: *Eine Constraint-basierte Modellierung für Geometrische Objekte.* Doktorarbeit, Universität Koblenz-Landau, Koblenz, 1996. ISBN 3-923532-52-0.

[BFH+93] William Bouma, Ioannis Fudos, Christoph M. Hoffmann, Jiazhen Cai und Robert Paige: *A geometric constraint solver.* Technical Report CSD-TR-93-054, Purdue University, Department of Computer Sciences, 1398 CS Building, West Lafayette, IN 47907-1398, 1993.

[BFH+95] William Bouma, Ioannis Fudos, Christoph M. Hoffmann, Jiazhen Cai und Robert Paige: *A Geometric constraint solver.* Computer-aided Design, 27(6):487–501, 1995. http://www.cs.prurdue.edu/homes/cmh/distribution/papers/Constraints/CAD95.pdf.

[BK90] Wolfgang Beitz und K.-H. Küttner (Herausgeber): *Taschenbuch für den Maschinenbau.* Springer-Verlag, 17. Auflage, 1990.

[Bla44] Heinrich Blasius: *Statik*, Band 1 der Reihe *Mechanik, Physikalische Grundlagen vom technischen Standpunkt.* Boysen + Maasch Verlag, Hamburg, 3. Auflage, 1944.

[BM01] Beat D. Brüderlin und Andreas Meier: *Computergrafik und Geometrisches Modellieren.* Teubner, 2001. ISBN 3-519-02948-0.

[BR98] Beat D. Brüderlin und Dieter Roller (Herausgeber): *Geometric Constraint Solving and Applications.* Springer Verlag, 1998. ISBN 3-540-64416-4.

[BRD07] Torsten Brix, Michael Reeßing und Ulf Döring: *Ganzheitlicher Entwurf von Bewegungssystemen.* In: *7. Kolloquium Getriebetechnik*, Seiten 176–186, Siegen, 2007.

[Bri01] Torsten Brix: *Feature- und constraint-basierter Entwurf technischer Prinzipe.* Doktorarbeit, Technische Universität Ilmenau, Fakultät für Maschinenbau, 2001.

[Brü87] Beat D. Brüderlin: *Rule-Based Geometric Modelling.* Doktorarbeit, ETH Zürich, Switzerland, Zürich, 1987. ISBN 3-7281163-8.

[Brü93] Beat D. Brüderlin: *Using geometric rewrite rules for solving geometric problems symbolically.* Theoretical Computer Science, 116:291–303, 1993.

[Bru94] Mark W. Brunkhart: *Interactive Geometric Constraint Systems.* Diplomarbeit, University of Carlifornia, Berkeley, Computer Science Division, Mai 1994.

[BS67] Walter Breidenbach und Wilhelm Süss: *Geometrische Konstruktionen.* In: H. Behnke, F. Bachmann und K. Fladt (Herausgeber): *Grundlagen der Geometrie, Elementargeometrie*, Band II, Teil A der Reihe *Grundzüge der Mathematik*, Seiten 214–252. Vandenhoeck & Ruprecht, Göttingen, 1967.

[BS02] Ciprian Borcea und Ileana Streinu: *On the Number of Embeddings of Minimally Rigid Graphs.* In: *SoCG'02*, Barcelona, Spain, Juni 2002.

[CS94] J. C. H. Chung und K.-H. Sachs: *Constraint-based Variational Technology with I-DEAS Master Series.* In: Josef Hoschek und Werner Dankwort [HD94], Seiten 103–122.

[DH00] Cassiano Durand und Christoph M. Hoffmann: *A Systematic Framework for Solving Geometric Constraints Analytically.* Journal of Symbolic Computation, 30(5):493–519, 2000. http://www.cs.purdue.edu/homes/cmh/distribution/papers/Constraints/JSC00.pdf.

[DM97] Ulf Döring und Paul Michalik: *Parametrischer Entwurf - Einführung und ausgewählte Algorithmen*, Mai 1997. 1. GI-Seminar Effiziente Methoden der geometrischen Modellierung und der wissenschaftlichen Visualisierung, IBFI Schloß Dagstuhl, Deutschland.

[DMB98] Ulf Döring, Paul Michalik und Beat D. Brüderlin: *A constraint-based shape modeling system.* In: Beat D. Brüderlin und Dieter Roller [BR98], Seiten 127–147.

[Dre53] Georg Dreyer: *Graphostatik.* Fachbuchverlag GmbH, Leipzig, 1953.

[Dur98] Cassiano Durand: *Symbolic and Numerical Techniques for Constraint Solving.* Doktorarbeit, Purdue University, West Lafayette, Indiana, Dezember 1998.

[End90] Eric Enderton: *Interactive Type Synthesis of Mechanisms.* Masterthesis, University of Carlifornia, Berkeley, Computer Science Division, März 1990.

[Fär09] Markus Färber: *A Note on Homotopy Continuation for Geometric Constraint Solving.* Technischer Bericht, Technische Universität Ilmenau, 2009.

[FB09] Markus Färber und Beat Brüderlin: *Multivariate root finding with search space decomposition and randomisation.* In: *Proceedings of the 2009 ACM symposium on Applied Computing*, Seiten 1142–1143, Honolulu, Hawaii, März 2009.

[Fed32] Karl Federhofer: *Graphische Kinematik und Kinetostatik*, Band 1, Heft 2 der Reihe *Ergebnisse der Mathematik und ihrer Grenzgebiete.* Julius Springer, Berlin, 1932.

[FH96a] Ioannis Fudos und Christoph M. Hoffmann: *Constraint-Based Parametric Conics for CAD.* Computer-aided Design, 28(2):91–100, 1996. http://www.cs.prurdue.edu/homes/cmh/distribution/papers/Constraints/CAD96.pdf.

[FH96b] Ioannis Fudos und Christoph M. Hoffmann: *Correctness proof of a geometric constraint solver.* International Journal of Computational Geometry and Applications, 6(4):405–420, 1996. http://www.cs.purdue.edu/homes/cmh/distribution/papers/Constraints/IJCGA96.pdf.

[FH97] Ioannis Fudos und Christoph M. Hoffmann: *A Graph-constructive Approach to Solving Systems of Geometric Constraints.* ACM Transactions on Graphics, 16(2):179–216, 1997.

[Fud95] Ioannis Fudos: *Constraint solving for computer aided design.* Doktorarbeit, Purdue University, Computer Science Department, August 1995.

[FvDFH97] James D. Foley, Andries van Dam, Steven K. Feiner und John F. Hughes: *Computer Graphics, Principles and Practice, second edition in C.* Addison-Wesley, 1997. ISBN 0201848406.

[GHY04] Xiao-Shan Gao, Christoph M. Hoffmann und Wei-Qiang Yang: *Solving Spatial Basic Geometric Constraint Configurations with Locus Intersection.* Computer-aided Design, 36(2):111–122, 2004.

[GKN81] Walter Gellert, Herbert Kästner und Siegfried Neuber (Herausgeber): *Lexikon der Mathematik.* VEB Bibliographisches Institut Leipzig, 3. Auflage, 1981.

[GL04] Xiao-Shan Gao und Qiang Lin: *MMP/Geometer - A Software Package for Automated Geometric Reasoning.* In: Franz Winkler [Win04], Seiten 44–66.

[Hav97] Timothy F. Havel: *Distance Geometry: Theory, Algorithms, and Chemical Applications.* Technical Report, Harvard Medical School, Boston, MA, USA, 1997.

[Hav05] Sven Havemann: *Generative Mesh Modeling.* Doktorarbeit, Technische Universität Braunschweig, Braunschweig, November 2005.

[HB97] Ching-yao Hsu und Beat D. Brüderlin: *A hybrid constraint solver using exact and iterative geometric constructions.* In: Dieter Roller und Pere Brunet (Herausgeber): *CAD Systems Development. Tools and Methods.* Springer Verlag, 1997.

[HC02a] Christoph M. Hoffmann und Ching-Shoei Chiang: *Variable-radius circles in cluster merging, Part I: Translational Clusters.* Computer-aided Design, 34(11):787–797, 2002. http://www.cs.prurdue.edu/homes/cmh/distribution/papers/Constraints/CAD01a.pdf.

[HC02b] Christoph M. Hoffmann und Ching-Shoei Chiang: *Variable-radius circles in cluster merging, Part II: Rotational Clusters.* Computer-aided Design, 34(11):799–805, 2002. http://www.cs.purdue.edu/homes/cmh/distribution/papers/Constraints/CAD01b.pdf.

[HD94] Josef Hoschek und Werner Dankwort (Herausgeber): *Parametric and Variational Design.* Teubner, 1994. ISBN 3-519-02632-5.

[Hen11] Lebrecht Henneberg: *Die graphische Statik der starren Systeme.* B. G. Teubner, Leipzig, 1911.

[HGY02] Christoph M. Hoffmann, Xiao-Shan Gao und Wei-Qiang Yang: *Solving Spatial Basic Geometric Constraint Configurations with Locus Intersection.* In: *ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications*, Seiten 95–104, Saarbrücken, Germany, Juni 2002.

[HK73] John E. Hopcroft und Richard M. Karp: *An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs.* SIAM J. Comput., 2(4):225–231, Dezember 1973.

[HLS97] Christoph M. Hoffmann, Andrew Lomonosov und Meera Sitharam: *Finding Solvable Subsets of Constraint Graphs.* In: Gert Smolka (Herausgeber): *CP*, Band 1330 der Reihe *Lecture Notes in Computer Science*, Seiten 463–477. Springer, 1997. http://www.cs.purdue.edu/homes/cmh/distribution/papers/Constraints/CP97.pdf.

[HLS98] Christoph M. Hoffmann, Andrew Lomonosov und Meera Sitharam: *Geometric constraint decomposition.* In: Beat D. Brüderlin und Dieter Roller [BR98], Seiten 170–195.

[HLS01a] Christoph M. Hoffmann, Andrew Lomonosov und Meera Sitharam: *Decomposition Plans for Geometric Constraint Problems, Part I: Performance Measures for CAD.* Journal of Symbolic Computation, 31(4):367–408, 2001. http://www.cs.purdue.edu/homes/cmh/distribution/papers/Constraints/JSC01a.pdf.

[HLS01b] Christoph M. Hoffmann, Andrew Lomonosov und Meera Sitharam: *Decomposition Plans for Geometric Constraint Problems, Part II: New Algorithms.* Journal of Symbolic Computation, 31(4):409–428, 2001. http://www.cs.purdue.edu/homes/cmh/distribution/papers/Constraints/JSC01b.pdf.

[Hof94] Christoph M. Hoffmann: *Semantic Problems of Generative, Constraint-Based Design.* In: Josef Hoschek und Werner Dankwort [HD94], Seiten 37–46.

[Hsu96] Ching-yao Hsu: *Graph-based approach for solving geometric constraint problems.* Doktorarbeit, University of Utah, Computer Science Department, 1996.

[Hüs88] Mathias Hüsing: *Erstellung eines iterativen Programmsystems zur kinematischen Analyse ebener Koppel- und Koppel-Räder-Getriebe.* Diplomarbeit, Universität Hannover, Institut für Getriebetechnik im Maschinenbau, September 1988.

[HV95a] Christoph M. Hoffmann und Pamela J. Vermeer: *Geometric constraint solving in R^2 and R^3.* In: D. Z. Du und F. Hwang (Herausgeber): *Computing in Euclidean Geometry*, Seiten 266–298. World Scientific Publishing, Singapore, 2. Auflage, 1995.

[HV95b] Christoph M. Hoffmann und Pamela J. Vermeer: *A spatial constraint problem.* In: Jean-Pierre Merlet und Bahram Ravani (Herausgeber): *Computational Kinematics '95*, Seiten 83–92. Kluwer Academic Publishers, Dordrecht, 1995. http://www.cs.prurdue.edu/homes/cmh/distribution/papers/Constraints/Nice95.pdf. Proceedings of the Second Workshop on Computational Kinematics held in Sophia Antipolis, France, September 4–6, 1995.

[HY01] Christoph M. Hoffmann und Bo Yuan: *On Spatial Constraint Solving Approaches.* In: Jürgen Richter-Gebert und Dongming Wang (Herausgeber): *Automated Deduction in Geometry*, Band 2061 der Reihe *Lecture Notes in Computer Science*, Seiten 1–15. Springer, 2001.

[Jer02] Christophe Jermann: *Résolution de constraintes géométriques par rigidification récursive et propagation d'intervalles.* Doktorarbeit, Université de Nice - Sophia Antipolis, 2002.

[JLS02] Robert Joan-Arinyo, Maria Victoria Luzón Garcia und A. Soto-Riera: *Searching the Solution Space in Constructive Geometric Constraint Solving with Geometric Constraints.* Technical Report R02-42, Universidad Politècnica de Cataluna, 2002. www.lsi.upc.es/dept/techreps/ps/R02-42.ps.gz.

[JNT03] Christophe Jermann, Bertrand Neveu und Gilles Trombettoni: *Algorithms for Identifying Rigid Subsystems in Geometric Constraint Systems.* In: Georg Gottlob und Toby Walsh (Herausgeber): *IJCIA*, Seiten 233–238. Morgan Kaufmann, 2003.

[JNT04] Christophe Jermann, Bertrand Neveu und Gilles Trombettoni: *A New Structural Rigidity for Geometric Constraint Systems.* In: Franz Winkler [Win04], Seiten 87–105.

[JSVV02] Robert Joan-Arinyo, A. Soto-Riera, S. Vila-Marta und J. Vilaplana-Pastó: *Revisiting Decomposition Analysis of Geometric Constraint Graphs.* In: *ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications*, Seiten 105–115, Saarbrücken, Germany, Juni 2002.

[KGC97] Ghassan Kwaiter, Véronique Gaildrat und René Caubet: *DEM^2ONS: A High Level Declarative Modeler for 3D Graphics Applications.* Technischer Bericht, I.R.I.T, Université Paul Sabatier, 1997.

[Kli54] Horst Klien: *Fremdwörterbuch.* VEB Bibliographisches Institut Leipzig, Leipzig, August 1954.

[Kor99] Ulrich Kortenkamp: *Foundations of Dynamic Geometry.* Doktorarbeit, ETH Zürich, Switzerland, Zürich, November 1999.

[Kra90] Glenn Andrew Kramer: *Geometric Reasoning in the Kinematic Analysis of Mechanisms.* Technical Report TR-91-02, Schlumberger Laboratory for Computer Science, Oktober 1990.

[Kra92] Glenn Andrew Kramer: *Solving Geometric Constraint Systems : A Case Study in Kinematics.* The MIT Press, 1992. ISBN 0-262-11164-0.

[LIJ97] Hon-Wai Leong, Hiroshi Imai und Sanjay Jain (Herausgeber): *Algorithms and Computation, 8th International Symposium, ISAAC '97, Singapore, December 17-19, 1997, Proceedings*, Band 1350 der Reihe *Lecture Notes in Computer Science.* Springer, 1997. ISBN 3-540-63890-3.

[LM95] Kurt Luck und Karl-Heinz Modler: *Getriebetechnik: Analyse, Synthese, Optimierung.* Springer-Verlag, 2. Auflage, 1995.

[LM96] Hervé Lamure und Dominique Michelucci: *Solving Geometric Constraints by Homotopy.* IEEE Transactions on Visualization and Computer Graphics, 2(1):28–34, März 1996.

[LM98] Hervé Lamure und Dominique Michelucci: *Qualitative study of geometric constraints.* In: Beat D. Brüderlin und Dieter Roller [BR98], Seiten 234–258.

[LP86] L. Lovász und M. D. Plummer: *Matching Theory.* NorthHolland Publ. Co., 1986.

[Luz01] Maria Victoria Luzón Garcia: *Resolución de Restricciones Geométricas. Selección de la Solución Deseada.* Doktorarbeit, Universidad de Vigo, Dept. Informática, Dezember 2001.

[McH90] James A. McHugh: *Algorithmic Graph Theory.* Prentice-Hall, 1990. ISBN 0-13-023615-2.

[MMSW93] Andreas Mayer, Bernhard Mechler, Andreas Schlindwein und Rainer Wolke: *Fuzzy Logic.* Addison-Wesley, 1993.

[MTYS94] Oleg Melnikov, Regina Tyshkevich, Vladimir Yemelichev und Vladimir Saranov: *Lectures of Graph Theory.* BI Wissenschaftsverlag, 1994.

[MV80] Silvio Micali und Vijay V. Vazirani: *An $O(\sqrt{|V|}\cdot|E|)$ Algorithm for Finding Maximum Matching in General Graphs.* In: *21st Annual Symposium on Foundations of Computer Science*, Seiten 17–23, Syracuse, New York, Oktober 1980.

[Neu95] Wolfram Neutsch: *Koordinaten: Theorie und Anwendungen.* Spektrum, Akademischer Verlag GmbH, 1995.

[NS63] Allen Newell und Herbert Simon: *GPS, A Program that Simulates Human Thought.* In: *Computers and Thought*, Seiten 279–293. McGraw-Hill, 1963.

[OSMA01] Jianjun Oung, Meera Sitharam, Brandon Moro und Adam Arbree: *FRONTIER: fully enabling geometric constraints for feature-based modeling and assembly.* In: *Sixth ACM Symposium on Solid Modeling and Applications*, Seiten 307–308, Ann Arbor, Michigan, USA, Juni 2001. http://www.cise.ufl.edu/∼sitharam/full-frontier.ps.

[Oun01] Jianjun Oung: *Design and implementation of an object-oriented geometric constraint solver.* Diplomarbeit, University of Florida, Department of Computer and Information Science and Engineering, Mai 2001. http://www.cise.ufl.edu/∼joung/thesis/thesis.pdf.

[Owe91] J. C. Owen: *Algebraic solution for geometry from dimensional constraints.* In: *ACM/-SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications*, Seiten 397–407, Austin, Texas, 1991.

[QN96] Trân Quôc-Nam: *A Hybrid Symbolic-Numerical Approach in Computer Aided Geometric Design (CAGD) and Visualization.* Doktorarbeit, Johannes Kepler Universität Linz, Linz, Austria, Oktober 1996.

[RB98] Manfred Rosendahl und Roland Berling: *Modelling of geometric constraints in CAD-Applications.* In: Beat D. Brüderlin und Dieter Roller [BR98], Seiten 151–169.

[RBD06] Michael Reeßing, Torsten Brix und Ulf Döring: *Entwurf heterogener Systeme in frühen Phasen der Produktentwicklung.* In: *4. Gemeinsames Kolloquium Konstruktionstechnik*, Kühlungsborn, September 2006.

[RBDH93] Manfred Rosendahl, Roland Berling, Chun Du und Walter Hower: *Modellierung geometrischer Constraints für CAD-Anwendungen.* In: Klaus Kansy und Peter Wißkirchen (Herausgeber): *Proceedings des GI-Workshops "Neue Architekturkonzepte zur Gestaltung graphischer Systeme*, Band 223 der Reihe *GMD-Studien*, Seiten 19–29, November 1993.

[RDB07] Michael Reeßing, Ulf Döring und Torsten Brix: *Modeling of Heterogeneous Systems in Early Design Phases.* In: Frank-Lothar Krause (Herausgeber): *The Future of Product Development, Proceedings of the 17th CIRP Design Conference*, Seiten 247–258, Berlin, 2007. Springer. http://www.springerlink.com/content/l185r3n7175vk2g2.

[RGK00] Jürgen Richter-Gebert und Ulrich Kortenkamp: *Cinderella - die interaktive Geometriesoftware.* HEUREKA-Klett Softwareverlag, Stuttgart, Januar 2000. www.cinderella.de.

[RGK01] Jürgen Richter-Gebert und Ulrich Kortenkamp: *A dynamic setup for elementary geometry.* In: *Proceedings of MTCM 2000.* Springer-Verlag, 2001.

[RGK02] Jürgen Richter-Gebert und Ulrich Kortenkamp: *Dynamische Geometrie: Grundlagen und Möglichkeiten.* In: Thomas Weth (Herausgeber): *Tagungsband zum Nürnberger Kolloquium zur Didaktik der Mathematik 2002*, 2002.

[RL00] Willi Rehwald und Kurt Luck: *KOSIM - Koppelgetriebesimulation*, Band 332 der Reihe *Fortschritt-Berichte VDI: Reihe 1, Konstruktionstechnik, Maschinenelemente.* VDI-Verlag, Düsseldorf, 2000.

[Rol91] Dieter Roller: *An Approach to Computer Aided Parametric Design.* Computer Aided Design, 23(5):385–391, Juni 1991.

[Rol95] Dieter Roller: *CAD. Effiziente Anpassungs- und Variantenkonstruktion.* Springer Verlag, 1995.

[RS89] Hendrik Rothe und Manfred Steinbach: *BASIC-Programmbausteine.* VEB Verlag Technik, Berlin, 1989.

[SB98a] Lluis Solano Albajes und Pere Brunet Crosa: *Geometric relaxation for solving constraint-based models.* In: Beat D. Brüderlin und Dieter Roller [BR98], Seiten 259–270.

[SB98b] Lluis Solano Albajes und Pere Brunet Crosa: *A Geometric Relaxation Solver for Parametric Constraint-Based Models.* Technical Report LSI-98-40-R, Universidad Politècnica de Cataluna, November 1998.

[SB00] Lluis Solano Albajes und Pere Brunet Crosa: *A Geometric Relaxation Solver for Constraint-Based Models.* In: Pere Brunet, Christoph M. Hoffmann und Dieter Roller (Herausgeber): *CAD Tools and Algorithms for Product Design*, Seiten 251–268. Springer Verlag, 2000.

[Sch93] Peter Schorn: *Exact and approximate solutions to the embedding problem in Distance Geometry.* Technischer Bericht, Institut für Theoretische Informatik, ETH Zürich, 1993.

[Sch97] Hans Rudolf Schwarz: *Numerische Mathematik.* Teubner, Stuttgart, 4. Auflage, 1997.

[Sed98] Robert Sedgewick: *Algorithms in C++.* Addison-Wesley, 3. Auflage, 1998.

[Sha02] Offer Shai: *Utilization of the Dualism between Determinate Truses and Mechanisms.* Mechanism and Machine Theory, 37(11):1307–1323, November 2002.

[Sha03] Offer Shai: *Transforming Engineering Problems through Graph Representations.* Advanced Engineering Informatics, 17(2):77–93, April 2003.

[Sut63a] Ivan E. Sutherland: *Sketchpad–A Man-Machine Graphical Communication System.* In: *Proceedings of the Spring Joint Computer Conference*, Seiten 329–346, Detroit, Michigan, Mai 1963.

[Sut63b] Ivan Edward Sutherland: *Sketchpad: A man-machine graphical communication system.* Doktorarbeit, Massachusetts Institute of Technology, 1963. http://www.cl.cam.ac.uk/TechReports. UCAM-CL-TR-574.

[Tod89] Philip Todd: *A k-Tree Generalization that characterizes Consistency of Dimensioned Engineering Drawings.* SIAM J. Disc. Math., 2(2):255–261, Mai 1989.

[TS88] Willi Törning und Peter Spellucci: *Numerische Methoden der Algebra*, Band 1 der Reihe *Numerische Mathematik für Ingenieure und Physiker.* Springer-Verlag, 2. Auflage, 1988.

[Uno97] Takeaki Uno: *Algorithms for Enumerating all Perfect, Maximum and Maximal Matchings in Bipartite Graphs.* In: Hon-Wai Leong et al. [LIJ97]. http://research.nii.ac.jp/∼uno/papers/isaac97web.pdf.

[Var01] Juan L. Varona: *Graphic and Numerical Comparison between Iterative Methods.* Technical Report, Universidad de La Rioja, Logroño, Spain, 2001.

[VDI95] VDI-Gesellschaft Entwicklung Konstruktion Vertrieb, Ausschuß Ebene Gelenkgetriebe: *Modulare kinematische Analyse ebener Gelenkgetriebe mit Dreh- und Schubgelenken.* Verein Deutscher Ingenieure, Düsseldorf, April 1995. VDI-Richtlinie 2729.

[VSR92] A. Verroust, F. Schonek und Dieter Roller: *Rule-oriented method for parameterized computer-aided design.* Computer Aided Design, 24(10):531–540, Oktober 1992.

[Win04] Franz Winkler (Herausgeber): *Automated Deduction in Geometry, 4th International Workshop, ADG 2002, Hagenberg Castle, Austria, September 4-6, 2002, Revised Papers*, Band 2930 der Reihe *Lecture Notes in Computer Science.* Springer, 2004. ISBN 3-540-20927-1.

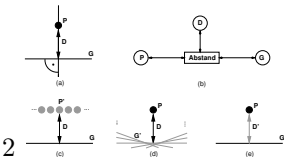
[Wol90] Rudolf Wolf: *Handbuch der Astronomie, ihrer Geschichte und Litteratur.* Verlag F. Schulthess, 1890. Nachdruck von Meridian Publishing Co. aus dem Jahr 1973.

[Zus48] Konrad Zuse: *Über Theorie und Anwendungen logistischer Rechengeräte.* Technical Report ZIANr 0301, ZuP 037/017, ZUSE-Ingenieurbuero, Hopferau bei Füssen, 1948. www.zib.de/zuse/Inhalt/Texte/Chrono/40er/Pdf/0301.pdf.

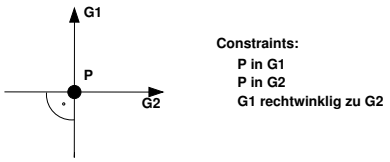
Abbildungsverzeichnis

Textuelle Auflistung

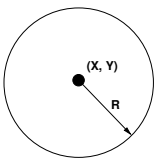
Visuelle Auflistung



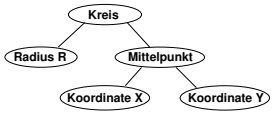
2

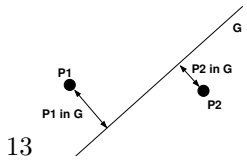


10

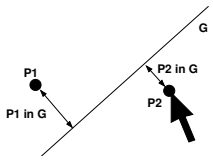


11

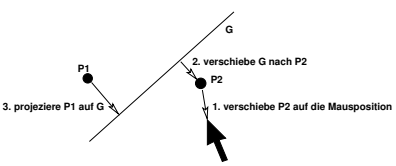




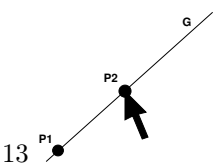
13



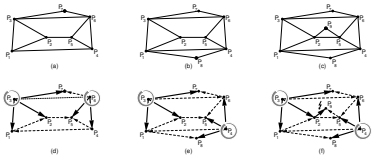
13

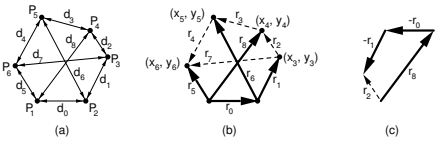


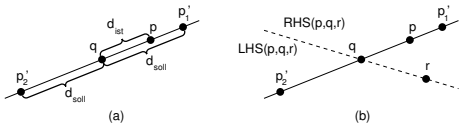
13



13

183 

186 

187 

Liste der Beispiele

Erklärung

Ich versichere, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Andere Personen waren an der inhaltlich-materiellen Erstellung der vorliegenden Arbeit nicht beteiligt (weder entgeltlich noch unentgeltlich). Insbesondere habe ich hierfür nicht die entgeltliche Hilfe von Vermittlungs- bzw. Beratungsdiensten (Promotionsberater oder anderer Personen) in Anspruch genommen. Niemand hat von mir unmittelbar oder mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form einer Prüfungsbehörde vorgelegt.

Ich bin darauf hingewiesen worden, dass die Unrichtigkeit der vorstehenden Erklärung als Täuschungsversuch angesehen wird und den erfolglosen Abbruch des Promotionsverfahrens zur Folge hat.

Ilmenau, den 12. Januar 2010

Ulf Döring

Thesen

1. Geometrische Verfahren finden bei der Lösung ingenieurtechnischer Probleme seit jeher eine breite Anwendung. Heutzutage ist jedoch durch den Einsatz von Software, welche Berechnungen zumeist mittels numerischer Gleichungslöser durchführt, eine Verdrängung zu beobachten.

2. Die Arbeit mit interaktiv anpassbaren Modellen hilft Ingenieuren wesentlich bei der Ideenfindung in frühen Phasen der Produktentwicklung. Für eine zufriedenstellende Interaktivität muss ein eingesetzter Constraint-Solver bezüglich Robustheit, Geschwindigkeit und Steuerbarkeit der Lösungsfindung domänenspezifische Anforderungen erfüllen. Eine Nutzerakzeptanz kann oft nur unter Berücksichtigung von entsprechendem ingenieurtechnischen Kontextwissen erreicht werden. Die plausible Behandlung von Über- und Unterbestimmtheit stellt hierbei eine besondere Herausforderung dar.

3. Ein weiteres zentrales Problem der Steuerbarkeit beim Constraint-Lösen ist die plausible Auswahl aus Mehrfachlösungen. Numerische Verfahren zeigen hier Schwächen, denn im Gegensatz zu geometrischen Verfahren sind ihre Möglichkeiten, Kontextwissen in die Lösungsfindung einzubeziehen, begrenzt. Insbesondere Aussagen wie „bei geeigneter Startpunktwahl“ zeigen den praxisfernen Charakter entsprechender Publikationen. Suchstrategien müssen sowohl Breiten- als auch Tiefensuche unterstützen.

4. Die Kombination von direkten und iterativen Konstruktionsschritten stellt einen Kompromiss dar, der grundsätzlich die Lösungsgeschwindigkeit steigert und Probleme mit der Auswahl von Mehrfachlösungen minimiert, aber letztlich nicht beseitigt.

5. Um verschiedensten Einsatzszenarien gerecht zu werden, muss die Implementierung einer geometrischen Konstruktionsvorschrift für eine bestimmte Regelgeometrie die möglichen Spezialfälle sowie Hinweise zur Lösungsauswahl berücksichtigen können.

6. Die schnellste iterative Konstruktion ist diejenige, die vermieden werden kann – sei es durch geeignete Transformation des Constraint-Netzes, so dass die direkte Konstruktion möglich wird, oder durch geeignete Wahl der Startwerte, so dass die Schleife nur einmal durchlaufen werden muss.

7. Der Einsatz des vorgestellten Constraint-Solvers erfordert typischerweise Expertenwissen, das sowohl solver- als auch domänenabhängig ist. Dieses Expertenwissen kann in den domänenspezifischen Applikationen oder in vordefinierten Modellen hinterlegt werden, so dass der Endnutzer nicht gezwungenermaßen ein Experte sein muss.

Lebenslauf

Zur Person

Ulf Döring
Goetheallee 4
98693 Ilmenau

geboren am 4. Mai 1969
in Potsdam
verheiratet, drei Kinder

Schulische und Berufliche Ausbildung

09/75 – 08/85 Polytechnische Oberschule Groß Kreutz

09/85 – 07/88 Geräte- und Reglerwerke Teltow
Berufsausbildung zum Facharbeiter für BMSR-Technik mit Abitur

09/89 – 12/94 TU Ilmenau
Studiengang Informatik, Nebenfach: Automatisierung
Vertiefungsgebiete: Datenbanken und Grafische Datenverarbeitung
Abschluss als Diplom-Informatiker, Thema der Diplomarbeit:
„Inferenzmechanismen unter Berücksichtigung von Unschärfe und Unsicherheit“

Beruflicher/Wissenschaftlicher Werdegang

01/95 – 12/95 TU Ilmenau
Wissenschaftlicher Mitarbeiter in einem DFG-Projekt mit den
Schwerpunkten: Behandlung großer Regelbasen und Fuzzy-Operatoren

01/96 – 09/96 R3M Softwarebüro Ilmenau
Tätigkeiten als Entwicklungsingenieur bei der Implementierung /
Portierung eines Fuzzy-Entwicklungssystems sowie bei der Software-
qualitätssicherung zu einem verteilten Funküberwachungssystem

seit 11/96 TU-Ilmenau
Wissenschaftlicher Mitarbeiter im Fachgebiet Computergrafik
Lehrtätigkeit sowie Mitarbeit an / Leitung von Forschungs- und
Industrieprojekten auf den Gebieten:

- constraintbasierte Modellierung,
- intelligentes CAD,
- Visualisierung,
- Bilderkennung,
- Nutzerinteraktion,
- Content-Management in Digitalen Bibliotheken

Aktuelle Projekte:

- „DMG-Lib“ (Digitale Mechanismen- und Getriebebibliothek) sowie
- „thinkMOTION“ (DMG-Lib goes Europeana)

Tätigkeiten im Bereich der Soft- und Hardwarebasis der DMG-Lib, u. a.:

- Content-Management, Webdesign, Datenaustausch,
- Generierung von Beschreibungsformaten für Texte, Getriebe u. a. Objektarten,
- semantische Analyse von Bildern, Texten und Getriebebeschreibungen,
- Modellierung der den Getrieben zugrundeliegenden technischen Prinzipe