Transformation und ontologische Formulierung multikriterieller Problemstellungen für die Lösung mit Verfahren der nichtlinearen Parameter-Optimierung

Der Fakultät für Elektrotechnik, Informationstechnik und Medientechnik der
Bergischen Universität Wuppertal
zur Erlangung des akademischen Grades eines
Dr.-Ing.

eingereichte Dissertation

von
Herrn Christian John M.Sc.

Datum der Einreichung: 15.10.2015

Referent: Prof. Dr.-Ing. Dietmar Tutsch
Korreferent: Prof. Dr.-Ing. Bernd Tibken

Tag der mündlichen Prüfung: 15.01.2016

Abstract

While in many cases algorithmic problem solving can be achieved by using closed deterministic procedures, recurrent tasks arise the need for an iterative and probabilistic approach. Non-linear parameter optimization is the most generalized way for iteratively solving those probabilistic problems.
While parameter-oriented optimization is based on unified structures of input and output data, the data is an abstract of semantic information given in the original task formulation. Semantic aspects are submitted to reduction from particular criteria-bound task scenarios, in order to be transformed into the numeric data structures needed for optimization. Result data is interpreted and re-used independently from time and place by reversal onto the scenario's semantic informational level.
A standardized transformation process for generation of abstract optimizational data structures is presented, serialization and structuring requirements are derived with regard to interpretational needs for result data. Conclusively, a standardized pre- and post-process mechanism is presented for automatic processing, interpretation, and representation of optimization result data, including an interface definition proposal.

Kurzfassung

Programmgestützte Problemlösungsverfahren können Aufgabenstellungen oft nicht mit einem geschlossen-deterministischen, sondern nur mit einem iterativ-probabilistischen Ansatz lösen. Die typischen Lösungsansätze sind als Verfahren der Optimierung beschrieben, unter denen die Verfahren der nichtlinearen Parameter-Optimierung eine Generalisierung unter den Optimierungsverfahren darstellen.
Parameter-Optimierungsverfahren lassen sich auf eine vereinheitlichte Struktur von Eingabe- und Ausgabe-Daten zurückführen, die jedoch von semantischen Informationen der eigentlichen Aufgabenstellung abstrahiert: Das Szenario einer konkreten, kriteriengebundenen Aufgabenstellung muss von semantischen Aspekten befreit und in die Datenstruktur der Optimierung transformiert werden; die Interpretation und zeitlich-räumlich getrennte Weiterverwendung der Ergebnisdaten nach erfolgter Optimierungsrechnung entspricht dann einer Rückführung in die semantische Informationsebene der ursprünglichen Aufgabenstellung.
Diese Arbeit beschreibt zunächst einen standardisierten Transformationsprozess zur Gewinnung der abstrahierten Datenstrukturen für die Optimierung und leitet daraus die notwendigen Serialisierungs- und Informationsstruktur-Anforderungen für die interpretatorische Rückführung der Ergebnisdaten her. Abschließend wird ein standardisierter Prä- und Post-Prozess-Mechanismus zur automatisierten Verarbeitung, Interpretation und Darstellung der Ergebnisdaten von Optimierungsverfahren zusammen mit dem Vorschlag einer Schnittstellendefinition beschrieben.

Inhaltsverzeichnis

Kapitel 1

Einleitung

„Die ungelösten Probleme halten einen Geist lebendig und nicht die gelösten.“ [Kol12]

Probleme lösen zu wollen liegt in der Natur des Menschen. Wer ein Problem als Aufgabenstellung annimmt und lösen will, muss zunächst die Ziele und Bedingungen in der Aufgabe erkennen, dann eine konkrete Vorgehensweise auswählen und schließlich eine Lösung des Problems als Antwort auf die Aufgabenstellung herbeiführen.

1.1 Einführung

Unter allen denkbaren Aufgabenstellungen werden in dieser Arbeit solche betrachtet, die durch identifizierbare Kriterien gegeben sind und durch Softwaregestützte Algorithmen einer Lösung zugeführt werden können. Die Suche nach der Lösung für ein solches algorithmisch-formalisierbares Problem ist primär die Suche nach dem geeigneten Lösungsalgorithmus.

Ein Algorithmus ist nach allgemeiner Auffassung „eine aus endlich vielen Schritten bestehende, eindeutige und ausführbare Handlungsvorschrift zur Lösung eines Problems oder einer Klasse von Problemen.“ [OW02]

Die naheliegende Vorgehensweise ist die Suche nach einem auf die konkrete Problemstellung angepassten Lösungsalgorithmus. Während für viele Standardaufgaben geschlossen-deterministische Algorithmen existieren, kann sich das Formulieren eines neuen Lösungsalgorithmus für viele Aufgaben aus der Praxis schwierig gestalten oder sich sogar als unmöglich erweisen. Gleichzeitig

können aber die problembeschreibenden Kriterien offensichtlich sein und es kann auch eine Vorstellung über einen gangbaren Lösungsweg vorliegen, oder sie sind – zumindest verbal – formulierbar. In diesen Fällen kann ein iterativ-probabilistischer Ansatz zum Erfolg führen.

Typische bekannte Lösungsverfahren für diesen Ansatz sind als Verfahren der Optimierung beschrieben. Optimierung in diesem Sinne ist jede Verbesserung eines System-Ist-Zustands im Hinblick auf ein konkretes Ziel, mit der Absicht, einen Optimalzustand zu erreichen oder sich diesem zumindest anzunähern. Optimierungsverfahren überführen die Kriterien der Aufgabenstellung in Variationskriterien, Beschränkungskriterien und Zielkriterien und unterwerfen diese einer numerischen Iteration zur Annäherung an den optimalen Systemzustand. Die numerischen Ergebnisse einer solchen Optimierung beschreiben den erreichten Optimalzustand und können für die Beantwortung der ursprünglichen Aufgabenstellung herangezogen werden.

1.2 Motivation

Wenn man Optimierung anwenden will, entstehen zwei Probleme:

1. Die zielbeschreibenden Kriterien in der Aufgabenstellung müssen erkannt und für das Optimierungsverfahren vorbereitet werden.

2. Die numerische Lösung muss nach der Optimierung im Sinne der Aufgabenstellung als Antwort auf die zugrunde liegende Frage formuliert werden.

Um die Kriterien der Aufgabenstellung für das Optimierungsverfahren zu erkennen, müssen die Kriterien identifiziert und anschließend auf die für das Optimierungsverfahren nutzbaren Daten reduziert werden. Bei der Reduktion gehen semantische Informationen verloren. Die semantischen Informationen werden jedoch nach der Optimierungsrechnung wieder benötigt, damit die numerische Lösung als Antwort auf die Frage im Sinne der Aufgabenstellung interpretiert und präsentiert werden kann.

Insbesondere bei automatisierten Abläufen zur Lösung probabilistischer Aufgabenstellungen erhält diese Problematik besondere Relevanz, da manuelle

Eingriffe und Rückgriffe auf individuelle Humanintelligenz dann nicht anwendbar sind.

1.3 Ziel der Arbeit

Die aufgezeigten Probleme der Kriterien-Transformation und der Ergebnis-Interpretation werden in dieser Arbeit analysiert und diskutiert.

Für die Kriterien-Transformation soll ein standardisiertes Phasenmodell zur gesicherten Überführung der Kriterien einer Aufgabenstellung in die Datenmodelle der Optimierung aufgestellt werden. Die in der Transformation abstrahierten semantischen Informationen sollen dabei einer strukturierten Persistierung zugeführt werden.

Aus den Datenstrukturen der semantischen Persistenz soll ein Rückführungsprozess zur semantischen Interpretation und Präsentation der Ergebnisdaten hergeleitet werden. Für die Semantik-Persistenz soll eine strukturierte Schnittstellen- und Dokument-Beschreibung aufgestellt werden.

Abschließend soll ein Modell für die Integration von Prä- und Post-Prozessoren zur automatischen, individualisierten Ergebnisverarbeitung vorgestellt werden.

Ausgehend von existierenden Ansätzen zu Problemlösungsstrategien und zum Erkennen von Problemzielen wird zunächst ein generalisiertes Optimierungsmodell formuliert und strukturierte Vorgehensweisen zur Datengewinnung und Ergebnisinterpretation aufgezeigt. Die Transformationsverfahren werden durch eine Ontologiebeschreibung und ein Klassenmodell konkretisiert und anhand von Beispielen verifiziert. Jedes Kapitel beginnt mit einem motivierenden Satz in allgemeinsprachlicher Formulierung.

Kapitel 2

Grundlagen

2.1 Problemstellungen

Weil viele Problemstellungen durch Kriterien beschrieben und durch Berechnen gelöst werden können, müssen die Kriterien erkannt und ein Rechenweg festgelegt werden.

Die Informationstechnologie beschäftigt sich mit konkretisierbaren und formalisierbaren Problemstellungen mit dem Ziel, eine Lösung zu finden. Präzisierend werden automatisierbare, berechenbare und mathematisch-algorithmisch beschreibbare Lösungen betrachtet. Diese seien im Folgenden als programmierbare Lösungen bezeichnet.

Problemstellungen sind im Hinblick auf die zu findende Lösung durch Kriterien beschrieben. Eine Lösung kann als gültig im Sinne der gegebenen Problemstellung angesehen werden, wenn sie die Kriterien vollständig oder zumindest in hinreichendem Maße erfüllt. Kriterien sind alle Merkmale, die eine Problemstellung charakterisieren. Kriterien im engeren Sinne sind diejenigen Merkmale, die für die Lösung einer Problemstellung zielführend und relevant sind. Nicht-triviale Problemstellungen sind im Allgemeinen durch eine Kriterienmenge mit mehreren oder vielen Kriterien beschrieben.

Diese Arbeit beschäftigt sich mit multikriteriellen Problemstellungen und Möglichkeiten der automatisierten Lösungssuche.

Für Problemstellungen gibt es grundlegend zwei Herangehensweisen zur Lösung: Einerseits sind deterministische, geschlossene Verfahren durch eine jederzeit nachvollziehbare Berechnungsvorschrift gekennzeichnet, andererseits

können nichtdeterministische, offene Probleme einen probabilistisch-iterativen Ansatz erfordern, der sich mit stochastischen, wahrscheinlichkeitstheoretischen Vorgehensweisen einer Lösung nähert (vertiefend s. Abschnitt 4.1.1).

Für die Lösung mithilfe eines probabilistisch-iterativen Vorgehens werden vorteilhaft Verfahren der Optimierung verwendet, da diese einen strukturierten und wohlbekannten [Kal13, SM09] Ansatz verfolgen.

Diese Arbeit beschäftigt sich mit nichtdeterministisch-offenen Problemstellungen und diskutiert die Überführung multikriterieller Aufgabenstellungen in Datenstrukturen zur Lösung mit Optimierungsverfahren. Eine solche Aufgabenstellung wird auch als Szenario bezeichnet.

2.2 Nichtlineare Parameter-Optimierung

Unter Optimierung versteht man im allgemeinen Sprachgebrauch die Verbesserung eines gegebenen Systemzustands mit dem Ziel, einen bestmöglichen Systemzustand aufzufinden. In Abwandlung und Abschwächung dieser Formulierung kann man Optimierung auch als Maßnahmen verstehen, die einen gegebenen Systemzustand in einen besseren Zustand überführen, ohne dass dieser neue Zustand zugleich das ideale Optimum darstellen muss. Jede Annäherung an ein Optimum wäre dann ebenfalls eine Optimierung.

2.2.1 Verfahrensweisen

Eine Optimierung im konkreten, informationstechnischen Sinn ist eine numerische Optimierung. Die Kriterien sind dann durch numerische Werte definiert, und auch das Ergebnis der Optimierung ist durch numerische Werte repräsentiert.

Der Zweck einer Optimierung ist somit das Auffinden oder die Annäherung an ein Optimum. Die Kriterien der zugrunde liegenden Problemstellung definieren einen Werteraum, der durch Randbedingungen begrenzt ist und den Suchraum für das Optimum darstellt. Die Kriterien, die den Suchraum definieren, werden als Parameter bezeichnet; die Parameter sind die in der Optimierung variierbaren Merkmale der Problemstellung. Das Optimum ist dann die global beste Lösung im Suchraum und damit die Menge der Parameterwerte, die den optimalen Systemzustand repräsentieren.

Optimierung ist damit die Suche nach dem Optimum im Suchraum. Prinzipiell würde ein erschöpfendes Durchsuchen [OW02] des Suchraums das Optimum sicher finden. Dieser Lösungsweg ist jedoch praktisch nicht durchführbar, da man – auch bei restringiertem Suchraum – mit einer potenziell unendlichen Anzahl von zu durchsuchenden Lösungskandidaten konfrontiert ist. Optimierungsverfahren versuchen daher, den Suchraum, ausgehend von einem aktuellen Systemzustand, sinnvoll und gezielt zu durchsuchen, um mit dem Testen einer endlichen Anzahl von Lösungskandidaten ein Optimum zu finden [Kal12]. Sie unterscheiden sich in der Art, wie die Lösungskandidaten ermittelt werden und wie der Suchweg bestimmt wird.

Ein Optimierungsverfahren kann entweder einen Minimal- oder einen Maximalwert als Optimum anstreben. Der Minimal- oder Maximalwert ist ein skalarer Zahlenwert, der sich aus der Berechnung einer Bewertungsfunktion in Abhängigkeit von den aktuellen Parameterwerten ergibt. Beide Ansätze eignen sich gleichermaßen gut für das Auffinden eines Optimums, häufig wird jedoch der Minimierungsansatz verfolgt. Das Optimum ist dann der Ort im Suchraum, in dem die Bewertungsfunktion minimal wird [GK02].

Optimierung wird damit auf die Identifizierung von variierbaren Parametern P und die Aufstellung und Minimierung einer Bewertungsfunktion zurückgeführt:

$$X_p = \{\overline{x}_p \in \Omega | \forall x' \in \Omega : f(\overline{x}_p) \succ f(\overline{x}'_p)\} \tag{2.1}$$

mit

X_p	Menge aller gleichwertigen Optima
Ω	Suchraum
$f : \Omega \to \mathbb{R}$	Zielfunktion
$\succ \in \{<, >\}$	Vergleichsoperator
$\overline{x}_p \in \Omega$	Parameterkombination eines Optimums

Diese verallgemeinerte Grundgleichung (2.1) der Parameteroptimierung gilt sowohl für die Minimum- als auch für die Maximum-Betrachtung der Optimierung. Die im weiteren Verlauf dieser Arbeit zugrunde gelegte Minimum-Betrachtung wird dann von dem Vergleichsoperator $<$ abgedeckt.

Ein Optimierungsverfahren arbeitet iterativ, es variiert in jedem Iterationsschritt alle aktuellen Parameterwerte und bewertet die neue Wertekombination

anhand der Bewertungsfunktion, deren Funktionswert über den weiteren Verlauf und schließlich den Abbruch der Iteration bestimmt [PS00].

Nichtlineare Parameter-Optimierung lässt nichtlineare Zusammenhänge zwischen den Parametern, den Randbedingungen und insbesondere in der Bewertungsfunktion zu. Dies ist der allgemeinste Fall und deckt ein breites Spektrum von realen Optimierungsproblemen ab. Durch die Nichtlinearität der Bewertungsfunktion wird allerdings die Bestimmung des Minimums und damit des Optimums im Suchraum zu einer nicht-trivialen Aufgabe. Insbesondere Bewertungsfunktionen, die nicht stetig differenzierbar sind, also Unstetigkeiten, Löcher etc. enthalten können, erzwingen einen iterativen Lösungsverlauf, anstatt auf einfache Methoden der Analysis zurückgreifen zu können.

In jedem Fall besteht die Aufgabe, für einen gegebenen Ort der Bewertungsfunktion eine Abstiegsrichtung zu finden, um sich auf das Minimum zuzubewegen. Dafür werden sowohl deterministische als auch stochastische Suchverfahren eingesetzt. Ein deterministisches Verfahren versucht zumindest in einer lokalen Umgebung der Bewertungsfunktion, die Richtung des schnellsten Abstiegs zu finden. Diese kann man finden, indem man den negativen Gradienten am aktuellen Ort der Bewertungsfunktion bestimmt; dazu muss die Bewertungsfunktion zumindest lokal stetig differenzierbar sein. Stochastische Suchverfahren versuchen demgegenüber, eine Abstiegsrichtung durch zufallsbestimmtes Ausprobieren zu finden. Gradientenverfahren versprechen im Allgemeinen ein besseres Laufzeitverhalten, die stochastischen Verfahren können andererseits auf Bewertungsfunktionen angewendet werden, die nicht (auch nicht lokal) stetig differenzierbar sind.

Im Folgenden werden für beide Verfahrensklassen typische Vertreter beschrieben, anschließend wird zusätzlich ein Mischverfahren dargestellt.

Gradientenverfahren

Gradientenverfahren sind Vertreter der deterministischen Verfahrensklasse, die durch Differentiation der Bewertungsfunktion analytisch den Weg des steilsten Abstiegs zu finden versuchen. In einer mehrdimensionalen Bewertungsfunktion, resultierend aus einer Parameteranzahl > 1, entspricht die lokale Richtung des steilsten Aufstiegs dem Gradientenvektor. Die Richtungsumkehr des Gradientenvektors zeigt dann in die lokal steilste Abstiegsrichtung. Eine Parameter-

variation wird mit einer gegebenen Schrittweite in dieser Richtung vorgenommen. An jedem neuen Ort wird erneut der negative Gradient bestimmt und der Abstiegsrichtung weiter gefolgt, bis ein Abbruchkriterium erreicht ist; als Abbruchkriterium wird üblicherweise eine Konvergenzüberprüfung auf aufeinander folgende Bewertungsfunktionswerte angewendet. Mit anderen Worten: Der lokalen Abstiegsrichtung wird solange gefolgt, bis keine relevante Verbesserung mehr erreicht werden kann. Wie bereits beschrieben, sind Gradientenverfahren auf die stetige Differenzierbarkeit der Bewertungsfunktion über dem gesamten Suchraum angewiesen. Weiterhin entsteht das Problem der Neben-Minima, das aber an dieser Stelle nicht weiter verfolgt werden soll [Mei05].

Evolutionsverfahren

Evolutionsverfahren bilden die Prinzipien der Mutation und Selektion in der natürlichen Entwicklung der biologischen Arten nach. Sie sind Vertreter der stochastischen Verfahrensklasse. Nach dem Prinzip des Zufalls wird am aktuellen Ort eine Änderung der Parameterwerte innerhalb eines Schrittweitenradius vorgenommen. Am Zielort wird anhand der Bewertungsfunktion entschieden, ob der neue Ort eine Verbesserung gegenüber dem bisherigen Zustand darstellt. Verschlechterungen werden sofort verworfen, Verbesserungen werden beibehalten und dienen als Ausgangspunkt für die nächste Parametervariation. Da in diesem Ablauf eine echte Konvergenz nicht sinnvoll erkannt werden kann, ist das Abbruchkriterium im Allgemeinen eine vorgegebene Schrittanzahl [Wei15, SKK10].

Hooke-Jeeves-Verfahren

Als Vertreter eines Mischverfahrens mit erfahrungsgemäß gutem Laufzeitverhalten sei hier zusätzlich das Verfahren von Hooke-Jeeves vorgestellt. Es ist ein Zwei-Schritte-Optimierungsverfahren: Im ersten Schritt (Exploration) wird stochastisch eine Abstiegsrichtung erkundet. Im Erfolgsfall schließt sich ein weiterer Schritt (Extrapolation) in die gefundene Richtung an; bei Misserfolg der Exploration wird der Variationsradius kontrahiert und erneut exploriert. Das Verfahren erfordert keine stetige Differenzierbarkeit der Bewertungsfunktion, kann aber durch den Extrapolationsschritt trotzdem einer gefundenen lokalen Abstriegsrichtung folgen, sodass das Verfahren im Allgemeinen sehr

schnell konvergiert [Woh05, Fun13].

Zu jeder der vorgestellten Verfahrensklassen gibt es zahlreiche weitere Vertreter und Abwandlungen, die jeweils ihre eigenen spezifischen Vor- und Nachteile haben. Diese sollen hier aber nicht weiter diskutiert werden [HL97].

2.2.2 Konstituierende Elemente

Die konstituierenden Elemente eines Verfahrens der Parameter-Optimierung sind die Parameter, die Randbedingungen, die Zielfunktion und steuernde Attribute. Diese werden im Folgenden erklärt, soweit es für die weitere Diskussion im Rahmen dieser Arbeit erforderlich ist. Dabei werden die Randbedingungen in Mussbedingungen (Beschränkungen) und Wunschbedingungen (Restriktionen) unterteilt. Die Zielfunktion setzt sich aus der eigentlichen Bewertungsfunktion und den Straffunktionen der Restriktionen zusammen.

Parameter

Die Parameter sind die in der Optimierung veränderlichen Zahlenwerte. Ein aktueller Systemzustand ist die Menge aller aktuellen Parameterwerte, zusammengefasst zu einer Parameterkombination. Ein Optimierungsverfahren variiert die Parameter nach Maßgabe seiner spezifischen Funktionsweise, mit dem Ziel, die Bewertungsfunktion zu minimieren.

Randbedingungen

Randbedingungen beschränken den Suchraum, in dem das Optimierungsverfahren gültige Parameterwert-Kombinationen auffinden kann. Dabei kann man zwischen zwangsweise einzuhaltenden Bedingungen und wünschenswerten Bedingungen unterscheiden.

Beschränkungen

Eine Beschränkung (engl. *constraint*) ist eine Muss-Bedingung. Sie definiert einen gültigen Wertebereich für Parameter. Die Notwendigkeit, eine Bedingung als absolut zu erfüllende Muss-Bedingung zu implementieren, kann entweder aus der Logik der Aufgabenstellung abzuleiten oder numerischen Ursprungs

sein. Insbesondere darf durch unzulässige Parameterwerte die numerische Berechenbarkeit und Stabilität nicht gefährdet werden, man denke an eine Division durch Null und ähnliche Beschränkungen.

Beschränkungen wirken sich also als Reduktion des Suchraumes der Parameter aus. Unter bestimmten Voraussetzungen kann es sich als vorteilhaft erweisen, die zulässigen Werte einzelner Parameter bereits vor dem eigentlichen Optimierungslauf zu bestimmen. Im allgemeinen werden jedoch die durch eine Variation in einem Iterationsschritt entstehenden Parameterwerte individuell auf Gültigkeit überprüft und gegebenenfalls verworfen.

Restriktionen

Wunschbedingungen sind solche Bedingungen, die in der Optimierung so gut wie möglich erfüllt werden sollen, ohne dass dieser Wunsch Teil des in der Bewertungsfunktion ausgedrückten Optimierungsziels ist. Wunschbedingungen werden hier als Restriktionen (engl. *restrictions*) bezeichnet. Eine Wunschbedingung führt nicht zu einem Verwerfen eines Parameterwertes, verschlechtert aber über eine Straffunktion den Zielfunktionswert, sodass unerwünschte Systemzustände das Konvergenzverhalten von sich weglenken.

Eine Restriktion berechnet den Wert einer Straffunktion in Abhängigkeit von der aktuellen Parameterkombination. Die Straffunktionswerte aller Restriktionen werden der Bewertungsfunktion superponiert. Restriktionen können unterschiedlich gewichtet werden, indem man ihnen einen Priorisierungsfaktor zuweist.

Bewertungsfunktion

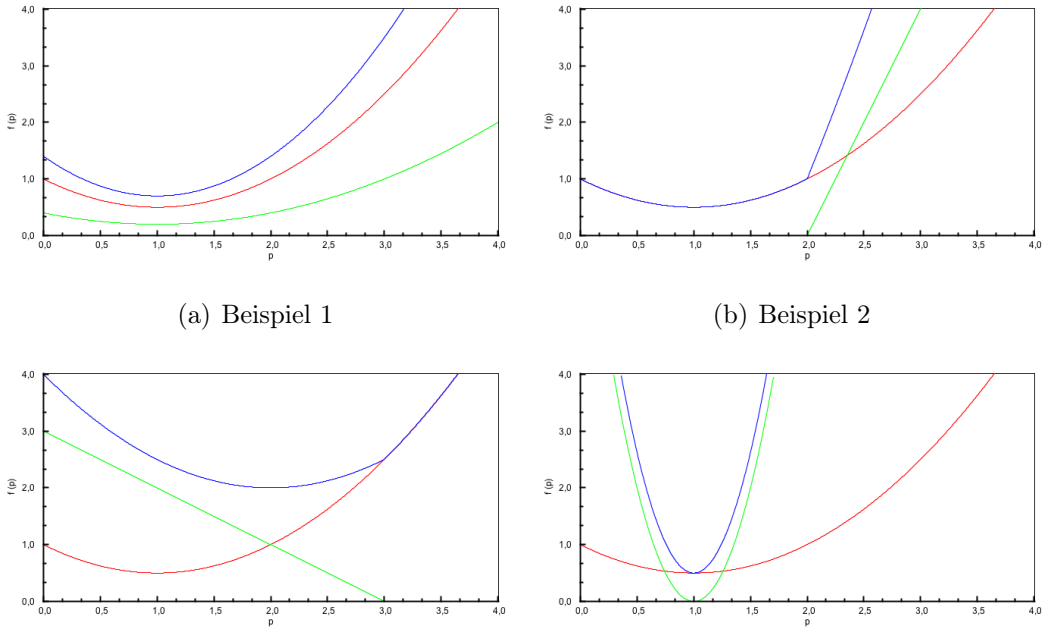
Die Bewertungsfunktion berechnet in Abhängigkeit von den aktuellen Parameterwerten einen Gütewert. Unterschiedliche Parameterkombinationen können dadurch miteinander verglichen werden, dass ihre Gütewerte berechnet und verglichen werden. Ein Optimierungsverfahren nutzt dies, um Verbesserungen und Verschlechterungen erkennen zu können, und leitet daraus in jeweils spezifischer Weise den weiteren Suchweg ab.

Die Gütewerte von Start-Parameterkombination und Ergebnis-Parameterkombination sind ein Maß für den Grad der Verbesserung durch die Optimierung.

Zielfunktion

Der tatsächlich im Optimierungsverfahren angewendete Vergleichswert entsteht aus dem Wert der Bewertungsfunktion, superponiert mit den Straffunktionswerten aller verletzten Restriktionen.

Verschiedene typische Konstellationen sind in den Abbildungen 2.1(a) bis 2.1(d) dargestellt. Hierin ist die Bewertungsfunktion (rot) zusammen mit einer Straffunktion (grün) zu einer resultierenden Zielfunktion (blau) zusammengeführt.



(a) Beispiel 1

(b) Beispiel 2

(c) Beispiel 3

(d) Beispiel 4

Abbildung 2.1: Zielfunktion (blau) mit überlagerter Gütewertfunktion (rot) und Restriktion (grün)

Variation

Das Optimierungsverfahren variiert die Parameter in verfahrensspezifischer Weise, die Art der Parametervariation bestimmt den Suchweg und ist abhängig von einer Schrittweite. Um einzelne Parameter in der Variation nicht zu bevorzugen oder zu benachteiligen, können Parameter durch eine Normalisierung auf eine einheitliche Wertebasis überführt werden. Der von außen erkennbare Wert des Parameters und sein tatsächlich im Optimierungsverfahren variierter Wert unterscheiden sich dann. Dies entspricht einer Unterscheidung in einen

(nach außen erkennbaren) Phänotyp und einen (im Inneren wirksamen) Genotyp eines Parameters.

Ein Standardmodell der Optimierung, basierend auf den hier vorgestellten konstituierenden Elementen, wird in Kap. 3 formuliert.

2.3 Kriterien einer Aufgabenstellung

Eine Aufgabenstellung ist eine Problemstellung, die einer Lösung zugeführt werden soll. Die Aspekte und Merkmale, die die angestrebte Problemlösung erkennbar machen, seien hier als *Kriterien* bezeichnet. Da im Allgemeinen mehrere Kriterien auftreten, wird die Aufgabenstellung als *multi-kriterielles* Problem bezeichnet.

Der Begriff *multi-kriteriell* (engl. *multi-criteria*) wird in der Optimierung häufig mit zwei voneinander zu unterscheidenden Hauptrichtungen [Gel14] assoziiert:

- MADM: Multi Attribute Decision Making, Multi-attributives Entscheidungsmodell
- MODM: Multi Objective Decision Making, Multi-Ziel-orientiertes Entscheidungsmodell

MADM: Multi-attributives Entscheidungsmodell

Ein MADM konzentriert sich auf die Betrachtung der zahlreichen Attribute der Problemstellung aus dem Blickwinkel der Ausgangssituation und versucht, damit ein Ziel zu erfüllen. Zu diesem Zweck wird eine für die Zielerreichung geeignete Kriterien-Auswahl herbeigeführt: durch eine Selektion der best-geeigneten Kriterien, eine Sortierung in Bedeutungsklassen und ein Ordnen aller untersuchten Alternativen [RB93].

MODM: Multi-Ziel-orientiertes Entscheidungsmodell

Ein MODM [Jah04b, Mie99] beschreibt eine Optimierung, die mehrere Ziele gleichzeitig verfolgt. Diese Ziele stehen meist in Konkurrenz zueinander oder weisen ein gegenläufiges Verhalten auf. Das Problem besteht also letztlich darin, unter mehreren Optima das insgesamt bestgeeignete auszuwählen. Eine

Lösung kann mithilfe von Pareto-Modellierungen gefunden werden, hier werden ineffiziente Lösungen eliminiert, es findet eine Reduktion auf die effizienten Lösungen statt [ZG91].

Multikriterielle Aufgabenstellung

Im Gegensatz zu den genannten Hauptrichtungen der Optimierung soll in dieser Arbeit die Attributierung *multi-kriteriell* nicht im unmittelbaren Zusammenhang mit der Optimierung im engeren Sinne verstanden werden, sondern im erweiterten, semantischen Sinn aus den identifizierbaren Lösungsanforderungen der Aufgabenstellung hergeleitet werden. Im weiteren Verlauf werden die semantischen Kriterien der Aufgabenstellung in die Kernelemente der Optimierungsverfahren transformiert und so der Optimierung zugänglich gemacht.

In diesem Rahmen können die Kriterien der Aufgabenstellung unterschieden werden in

- Variationskriterien (= Parameter)
- Beschränkungskriterien (= Beschränkungen und Restriktionen)
- Zielkriterien (= Zielfunktion)
- Konvergenzkriterien (= Abbruchbedingungen)

2.4 Standard-Modellierungssprachen

Algebraische Modellierungssprachen (AML, Algebraic Modeling Languages) können verwendet werden, um Optimierungsprobleme zu formulieren und einem Lösungsverfahren zuzuführen. AML sind um 1980 [Kal12] zum ersten Mal in Erscheinung getreten, zunächst durch die Varianten GAMS [Dre03, BKM10, Kal12], LINGO [CS04, Sch13, Sch14, Sch15] und MP-Model [GHPS02, Kal12]; letztere ist mittlerweile durch den Nachfolger Mosel abgelöst worden. Zuvor wurden Probleme maschinennah oder vorzugsweise in Fortran implementiert. Zwei häufig eingesetzte Modellierungssprachen sollen im Folgenden kurz diskutiert werden [Kal12, Kal04, GPS02, FGK02]:

AMPL

AMPL (A Mathematical Programming Language) wurde 1985 von Robert Fourer, David Gay und Brian Kernighan als reine Modellierungsspache an den Bell Laboratories entwickelt mit dem Ziel, eine Formulierung mathematischer Aufgabenstellungen zu ermöglichen. Daten und Problemparameter werden separat dem Löser (engl. *solver*) zugeführt; diese Entkopplung erlaubt eine Wiederverwendung des Problemmodells. Das Lösen durch ein Optimierungsverfahren bedeutet aber zusätzlich den Einsatz eines passenden Lösers, der jeweils individuell für die Problemklasse angepasst gewählt werden muss. AMPL ist an vielen Universitäten verbreitet und war im ersten Halbjahr 2015 mit einem Nutzungsanteil von ca. 75 % (vergleiche Abb. 2.2) die am häufigsten eingesetzte Modellierungssprache für den NEOS-Onlinelöser [FGK02, Kal12, GG95].

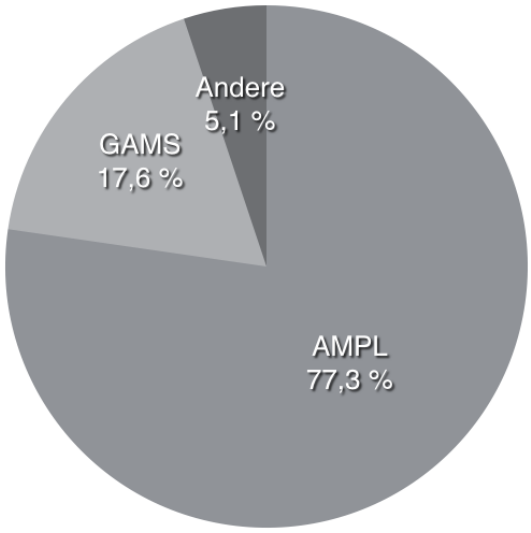


Abbildung 2.2: Übersicht der verwendeten Modellierungssprachen im NEOS-Onlinelöser von 1.1.2015 - 1.7.2015 [NEO]

GAMS

Das General Algebraic Modeling System (GAMS) wurde Ende 1970 von Alexander Meeraus und Jan Bisschop als erste AML entwickelt. Seit 1987 wird es von der GAMS Development Corporation vertrieben. Zunächst für die Weltbank [Dre03] konzipiert, liegt der heutige Schwerpunkt in der klassischen Optimierung. GAMS war im ersten Halbjahr 2015 mit knapp 20 % Nutzungsanteil die zweithäufigste Modellierungssprache für den NEOS-Onlinelöser. Der Schwerpunkt liegt bei GAMS in der Erstellung von Problemmodellierungen,

die dann an Löser übergeben werden können [BKM10, And03, Kal12].

NEOS [NEO] ist eine online verfügbare Sammlung von Solvern. Abb. 2.2 zeigt für einen Vier-Monats-Zeitraum im Jahre 2015 eine Übersicht der am meisten verwendeten Modellierungssprachen.

2.5 Informationstechnologische Ontologien

Als Ontologie wird im allgemeinen philosophischen Verständnis die „Lehre vom Seienden“ verstanden. Im Kontext von Informationstechnologien ist Ontologie seit den 90er Jahren als technischer Terminus [SSS09, Stu08] spezifischer gefasst und bezeichnet hier ein formal modelliertes Domänenmodell [HL97].

> *Definition: Ontologie*
> „Eine [informationstechnische] Ontologie ist eine explizite, formale Spezifikation der Konzeptualisierung eines abgegrenzten Diskursbereichs zu einem definierten Zweck, auf die sich eine Gruppe von Akteuren geeinigt hat.“ [Stu08]

Konzeptionell basiert eine Ontologie auf einer Taxonomie im Sinne einer hierarchischen Ableitung von Begriffen, welche sich durch Attribute und Beziehungen beschreiben und strukturieren lassen. Die Benennungen Konzepte und Klassen werden im Umfeld der Ontologie synonym zu Begriffen verwendet; Attribute sind die Ausprägungen von Eigenschaften konkreter Begriffe. Dabei verfolgt eine explizite formale Spezifikation der Begriffe die Absicht, eine fehl-interpretierbare Form der Konzeptualisierung während einer Ontologie-Erstellung zu vermeiden [Stu08, Stu11, Gua98].

Der in der obigen Definition enthaltene Einigungsaspekt erzeugt eine gemeinsame Verständigungsbasis der beteiligten Akteure: Nachrichtenaustausch und Informationsintegration funktioniert damit über die gemeinsame Sprache der Ontologie [Stu11, ES07].

2.6 Stand der Forschung

2.6.1 Semantische Textanalyse

Nicht nur im Rahmen des semantischen Web [HKRS07] ist es notwendig, Texte auf semantischen Gehalt zu untersuchen, zu interpretieren und semantische Strukturen interpretatorisch zu extrahieren. Eine Möglichkeit, Semantiken zu definieren und zu extrahieren, ist die Anwendung regelbasierter Systeme [Den12].

Der Begriff *Text Mining* wurde 1995 von Feldman und Dagan [FD95] als Textanalyse-Konzept eingeführt. Als Analogie zum Data Mining [MW05] funktioniert Text Mining ohne eine strukturierte Vorlage, sondern unterstützt Verfahren wie Information Retrieval [WIZD05] zum Dokumenten-Clustering [Atk07]. Der Schwerpunkt liegt auf dem Erkennen von implizit vorhandenen Informationen in einem Text und der Relationserhaltung zwischen Informationen [HNP05].

RuleML ist ein Vertreter XML-basierter Beschreibungssprachen zur semantischen Darstellung von Daten. Ausgehend von einem zu untersuchenden Text müssen RuleML-Regeln über eine Rule-Engine (typische Vertreter: Prova, DR-DEVICE, NxBRE [Rul15]) definiert werden, um den Text semantisch in das RuleML zu übersetzen. RuleML arbeitet als semantische Beschreibungssprache, die über definierbare Regeln einen Text in eine semantische Struktur transformiert, jedoch ist keine mathematisch-algebraische Modellierung möglich [Bik14, Rul15].

Das Software-Produkt *Docs2DB* von info-key konzentriert sich auf das Identifizieren von technischen Maßeinheiten und den jeweils zugeordneten Werten. Die zu suchenden Einheiten stammen aus einer Datenbank, die die Einheiten als Suchmuster für eine Textanalyse vorhält. Nach dem Textdurchlauf und dem Identifizieren der Daten werden diese vor Übergabe an ein Rechenverfahren auf Plausibilität untersucht, da Fehler wie Kommaverschiebungen oder falsche Einheitenzuweisungen im Text vorhanden sein können, die korrigiert werden müssen, bevor die Daten maschinell weiterverarbeitet werden [MG15].

Künstliche Intelligenz (KI) wird bereits seit geraumer Zeit zur Analyse von Texten herangezogen. Ein KI-Ansatz zur Textanalyse wurde schon 1964 von Bobrow [Bob64] unter dem Namen *student* entwickelt. Die Suche basierte auf

Schlüsselwörtern in wenig komplexen Aufgabenstellungen, die zu einer Lösungsentwicklung beitragen können. Sobald ein Schlüsselwort gefunden wurde, wurde in dem Umfeld nach einem Zahlenwert gesucht. Nach dem Durchlauf versucht student die Zahlenwerte in einer Gleichung auszuwerten. Die KI verstand die Aufgabe jedoch nicht semantisch, sie versuchte, Aufgaben, die einer ähnlichen Grundstruktur folgen, zu lösen [Bob94].

IBM entwickelte mit *WATSON* [WAT15] eine semantische Suchmaschine [KH13] auf Basis von Konzepten der Künstlichen Intelligenz, die auf eine formfrei gestellte Frage eine passende Antwort findet. Basierend auf einem umfangreichen Datenbestand wird die gestellte Frage analysiert, die Semantik der Fragestellung durch Information Retrieval [WIZD05] ermittelt und semantisch passend beantwortet.

WolframAlpha [WOL15] wurde 2009 von Wolfram Research auf Basis der Software Mathematica entwickelt; es handelt sich dabei um eine semantische Suchmaschine für mathematische Aufgabenstellungen. Aus einer gegebenen regelformulierten Aufgabenstellung sucht WolframAlpha basierend auf einer großen Datenbank die passende semantische Antwort aufgrund mathematischer Algorithmen. Die Frage wird nicht verstanden, sondern durch Regelelemente in der Frage interpretiert und darauf basierend startet eine semantische Suche.

Die Universität Michigan arbeitet seit 1980 an *Soar* (State Operator Apply Result) [Lai12], einer kognitiven Architektur, die unter Einsatz der Künstlichen Intelligenz auf Themen menschlichen Verstehens wie Kopfrechnen und Sprachverarbeitung angewendet wird. Aufgabenstellungen werden in der gleichnamigen Sprache Soar geschrieben und bauen sich axiomatisch auf, d. h. alle Prozesse werden auf einfache Prinzipien zurückgeführt, von deren Basis aus die Lösung bestimmt werden kann [SOA15].

Während die vorgestellten Verfahren und Konzepte in jeweils spezifischen Problemfeldern Stärken in der Analyse von textlichen Informationen aufweisen, ist keines der Verfahren speziell für den Aspekt der Kriterienextraktion im Umfeld der Optimierung (oder in einem allgemeineren Umfeld numerischer Berechnungen) entwickelt worden. Teile der angewendeten Verfahren und Ideen könnten aber für eine automatisierte Aufgabenanalyse angepasst und verwendet werden.

2.6.2 Ontologien im semantischen Web

Der Begriff der Ontologie wird seit einiger Zeit im Zusammenhang mit Bestrebungen, Informationen im World Wide Web semantisch analysierbar zu machen, diskutiert. Während es hier bereits viele Ansätze [AH11] zur Beschreibung und Analyse von Semantiken gibt, insbesondere unter Hinzuziehung von Taxonomien, ist die Zielsetzung hier nicht das Erkennen von Kriterien in zu lösenden Aufgabenstellungen. Auch eine semantische Suche kann nicht als Analyse einer komplexen, multikriteriellen Aufgabe verstanden werden. Taxonomiesysteme im semantischen Web sind daher wenig relevant für die in dieser Arbeit behandelte Thematik [RT06, HLMS08].

2.6.3 Automatische Verarbeitung natürlicher Sprache

Auch in der Medizin besteht der Bedarf, natürliche Sprache (aus Krankenakten und Dokumentationen) in strukturierte Daten für Fachdatenbanken, Abrechnungs- und Controlling-Systeme umzuwandeln. Schulz [Sch05] beschreibt diese Übertragung als bidirektional, wobei die Hürden der jeweiligen Fachsprachen und Strukturen zu überwinden sind: Während die natürliche Sprache für die Kommunikation und Dokumentation notwendig ist, erfordern strukturierte Daten ein medizinisches Terminologiesystem für Studien, Reporting, Controlling und Retrieval. Der Übertrag von natürlicher Sprache in eine medizinische Datenstruktur soll sich auf die wesentlichen Informationen konzentrieren. Als Übertragungsverfahren werden die bereits beschriebenen Verfahren des Text Mining und des Information Retrieval eingesetzt.

Schulz erweitert die Textanalyse-Aspekte um die Frage nach der bidirektionalen Überbrückung des Grabens zwischen Freitext-Formulierung und Datenstrukturen. Während der Übergang von Freitext zur Datenstruktur aus Gründen der automatisierten Weiterverarbeitung offensichtlich ist, kommt nun die Anforderung hinzu, aus den strukturierten Daten verständliche medizinische Aussagen auf unterschiedlichen Sprachniveaus (Fachsprache des Arztes, Laiensprache des Patienten) zu gewinnen. Ziel ist es, dank strukturierter Daten medizinische Fragen qualifiziert und verständlich beantworten zu können.

Die diskutierten Zusammenhänge kommen der in dieser Arbeit intendierten bidirektionalen Überbrückung einer interpretatorischen Lücke zwischen textu-

ellen Informationen und strukturierten Daten sehr nahe, bewegen sich aber im Kontext medizinischer Interpretation und beziehen numerische Berechnungen nicht mit ein.

2.6.4 Vor- und Nachverarbeitung von numerischen Berechnungen

Die Finite-Elemente-Methode (FEM) ist ein numerisches Verfahren zur diskretisierten Lösung technisch-physikalischer Problemstellungen. Ausgehend von einer Bauteilgeometrie wird eine Diskretisierung in ein Netz Finiter Elemente durchgeführt, Randbedingungen und äußere Belastungen werden aufgebracht. Anschließend erfolgt eine numerische Lösung mit einem Solver, dessen Ergebnisse interpretiert, visualisiert und weiterverarbeitet werden. Während prinzipiell das FE-Netz manuell aufgestellt werden kann, werden üblicherweise Netzgeneratoren eingesetzt, die aus Geometriedaten, insbesondere aus CAD-Modellen, die numerischen Element- und Netzdaten generieren. Auch die Ergebnisinterpretation und Visualisierung erfolgt heute nicht mehr manuell, sondern durch entsprechende Software-Werkzeuge. Die Netzgeneratoren (vor der Berechnung) und die Analysierer und Visualisierer (nach der Berechnung) werden in integrierten FEM-Paketen (ANSYS [ANS15], NASTRAN [NAS15] u. a.) als Prä- und Post-Prozessoren bereitgestellt.

Die für FEM eingesetzten Prä- und Post-Prozessoren stellen ein typisches Beispiel für die Vor- und Nachverarbeitung von Problemstellungen aus der Praxis für einen numerischen Solver dar, sind aber sehr dediziert für die effiziente Verarbeitung technisch-physikalischer Modelle abgestimmt. Obwohl sie eine Ergebnisvisualisierung erzeugen, sind sie nicht Semantik-getrieben und überlassen die Interpretation der Ergebnisse dem Anwender.

Kapitel 3

Generalisiertes Modell der Parameteroptimierung

Weil Parameteroptimierungsverfahren trotz einiger Unterschiede in Details auf wenige einheitliche Konzepte zurückgeführt werden können und eine einheitliche Handhabung solcher Konzepte für die Übertragung von Kriterien einer Aufgabenstellung in eine berechenbare Optimierung nützlich ist, wird ein abstraktes, generalisiertes Optimierungsmodell mit seinen Grundbestandteilen beschrieben und eingeführt.

Nachdem in Kap. 2 die prinzipiellen Konzepte der nichtlinearen Parameteroptimierung eingeführt und verschiedene konkrete Verfahrenskonzepte dargestellt und gegeneinander abgegrenzt worden sind, soll für den weiteren Verlauf der Betrachtung ein generalisiertes Optimierungsverfahren formuliert und verwendet werden. Zunächst werden die Komponenten und Schnittstellen des generalisierten Optimierungsverfahrens beschrieben und zu einem verallgemeinerten Gesamtaufbau zusammengesetzt; das generalisierte Optimierungsverfahren wird im Weiteren als agnostisch gegenüber dem tatsächlich eingesetzten Optimierungsalgorithmus angesehen. Anschließend werden typische Test- und Anwendungsfälle für die weitere Untersuchung aufgestellt.

3.1 Grundelemente der Optimierung

Die Grundelemente für eine nichtlineare Parameter-Optimierung sind

- Parameter

- Bedingungen (Randbedingungen, Nebenbedingungen)
- Zielfunktion

Diese drei Elemente müssen aufgestellt und als Datenstruktur dem Optimierungsverfahren zugeführt werden. Die Elemente sollen hier im Vorgriff als die „Begriffe“ einer Optimierungs-Ontologie verstanden werden; zum Verständnis der Ontologie s. Kap. 6.

3.2 Parameter

Die Parameter sind die wesentlichen problembeschreibenden Variablen. Sie bilden mit Anfangswerten den Ausgangspunkt eines Optimierungslaufs, werden während der Optimierung variiert und beschreiben schließlich auch die ermittelte Lösung der Aufgabenstellung. Für die Durchführung einer Optimierung müssen die Parameter vollständig identifiziert und als Gesamtheit dem Optimierungsverfahren bekannt gegeben werden (Laufindex i):

$$\overline{p} = [p_i] \tag{3.1}$$

Sinnvollerweise werden die Parameter in einer listenartigen Datenstruktur verwaltet und aufbereitet (3.1). Die Anfangswerte der Parameter werden als $\overline{p}_0$ dem Optimierungsverfahren übergeben. Das Optimierungsverfahren liefert die Problemlösung im Rahmen seines iterativen Algorithmus als $\overline{p}_L$.

3.3 Bedingungen

Das allgemeine Optimierungsverfahren begrenzt Parameterwerte aufgrund von Bedingungen (synonym: Randbedingungen oder Nebenbedingungen). Wie in Kap. 2 dargestellt, werden Muss-Bedingungen (Beschränkungen) und Wunsch-Bedingungen (Restriktionen) unterschieden. Diese beiden Arten von Bedingungen werden im Optimierungsverfahren unterschiedlich verarbeitet und daher auch unterschiedlich formuliert.

3.3.1 Muss-Bedingungen: Beschränkungen

Es ist notwendig, für das Optimierungsverfahren nur semantisch und numerisch gültige Parameterkombinationen zuzulassen. Ungültige Parameterkombinationen werden im Rahmen der Parametervariation grundsätzlich ausgeschlossen, sie werden dem Optimierungsverfahren nicht zugeführt, sondern sofort verworfen. Eine gültige Parameterkombination setzt voraus, dass jeder einzelne Parameter gültig ist.

Um gültige Parameterwerte zu erhalten, muss die Definitionsmenge jedes Parameters auf einen gültigen Wertebereich beschränkt werden. Dies geschieht durch Einführen individueller Grenzwerte für jeden Parameter im Sinne eines Definitionsbereichs.

Die Beschränkungen werden in einer Listenstruktur (3.2) verwaltet (Laufindex j):

$$\overline{b} = [b_j] \tag{3.2}$$

3.3.2 Wunsch-Bedingungen: Restriktionen

Wunsch-Bedingungen, die eine Parameterkombination zwar nicht als ungültig, aber als unerwünscht kennzeichnen, sollen als Restriktionen (3.3) in das Optimierungsverfahren eingebracht werden (Laufindex k):

$$\overline{r} = [r_k] \tag{3.3}$$

Restriktionen werden als Straffunktionen f_{r_k} in Abhängigkeit der aktuellen Parameterwerte (3.4) formuliert:

$$r_k = f_{r_k}(\overline{p}) \tag{3.4}$$

Um eine Priorisierung von Restriktionen zu ermöglichen, wird der berechnete Wert der Straffunktion zusätzlich mit einem Faktor $F_r > 1,0$ beaufschlagt: je höher die Wertigkeit der Wunschbedingung, desto höher der Faktor. Damit wird eine feingranulare Differenzierung von Restriktionen nach ihrer Bedeutung ermöglicht:

$$r_k = F_{r_k} \cdot f_{r_k}(\overline{p}) \tag{3.5}$$

Die faktorisierten Straffunktionen (3.5) werden in die Zielfunktion eingebracht (s. Abschnitt 3.4.2) und unterstützen die Konvergenz der variierten Parameterwerte auf das Optimierungsziel.

3.4 Gütewert und Zielfunktion

3.4.1 Gütewert

Das Optimierungsverfahren benötigt eine Instanz, die für die aktuelle Parameterkombination einen Gütewert berechnet:

$$f_G = f(\overline{p}_{akt}) \tag{3.6}$$

Der Gütewert (3.6) repräsentiert und bewertet das momentan erreichte Ergebnis aufgrund der aktuellen Parameterwerte. Anhand der Gütewerte können unterschiedliche Parameterkombinationen bezüglich ihrer Erreichung des Optimums verglichen werden.

3.4.2 Zielfunktion

Die Zielfunktion superponiert den Gütewert mit den faktorisierten Straffunktionswerten (3.7) der Restriktionen:

$$f_Z = f(\overline{p}_{akt}) + \sum (F_{r_k} \cdot f_{r_k}(\overline{p}_{akt})) = f_G + \sum r_k \tag{3.7}$$

Der jeweils günstigste erreichte Zielfunktionswert wird mit der zugehörigen Parameterkombination gespeichert.

3.5 Variator

Das Optimierungsverfahren benötigt eine Instanz, die die Parameterwerte geeignet variiert. Dabei werden, ausgehend von der aktuellen Parameterkombination, neue Parameterwerte ermittelt. Die Parameterwerte werden mithilfe der Beschränkungen auf Gültigkeit getestet.

Die Variatorinstanz und ihr Verhalten ist eng an den tatsächlich verwendeten Optimierungsalgorithmus gekoppelt und ist prinzipieller Bestandteil des Algorithmus. Im Rahmen des generalisierten Optimierungsverfahrens wird nur sichergestellt, dass es eine solche Variatorinstanz gibt.

3.6 Abbruchkriterium

Das Optimierungsverfahren benötigt eine Instanz, die den Abbruch des Verfahrens steuert. Als Abbruch gilt sowohl eine willkürliche Unterbrechung als auch die erfolgreiche Beendigung.

Die Abbruchinstanz wird als eine Programm-Funktion realisiert, deren Rückgabewert ein Wahrheitswert (boolean) ist. Der Rückgabewert bestimmt, ob das Optimierungsverfahren beendet oder mit dem nächsten Iterationsschritt fortgesetzt werden soll. Für das generalisierte Optimierungsverfahren wird festgelegt, dass der Rückgabewert *true* als Abbruch interpretiert wird.

3.7 Gesamtaufbau

Der resultierende prinzipielle Ablauf eines allgemeinen Optimierungsverfahrens ist in Abb. 3.1 dargestellt. Hierin sind die einzelnen Bestandteile die Informations-Entitäten des Szenarios.

> *Definition: Informations-Entität*
> Eine Informations-Entität ist ein atomarer Bestandteil des Szenario-Datenmodells. Parameter, Beschränkungen, Restriktionen und die Zielfunktion sind Informations-Entitäten.

Die in Abb. 3.1 grau hinterlegten Strukturen müssen vor dem Lösungsstart dem Optimierungsverfahren zur Verfügung gestellt werden; ihre Gewinnung wird im folgenden Kap. 4 im Rahmen der Transformation diskutiert und erläutert.

3.8 Testszenarien und Aufgabenstellungen

Das generalisierte Optimierungsverfahren soll sowohl mit der in Kap. 4 beschriebenen Szenario-Transformation als auch mit dem in Kap. 8 eingeführten OOP-Klassenmodell anhand typischer Testszenarien und ausgewählter Aufgabenstellungen überprüft werden. Die verwendeten Szenarien werden im Folgenden beschrieben.

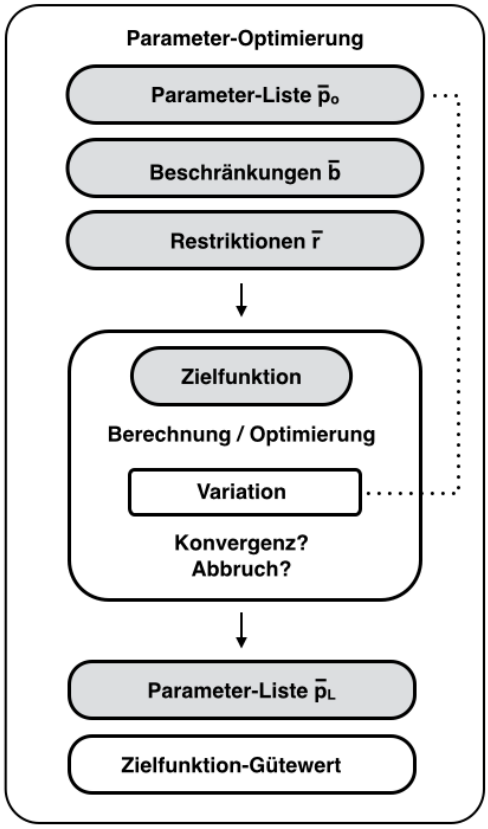


Abbildung 3.1: Generalisiertes Modell der Parameteroptimierung

3.8.1 Rosenbrock-Testfunktion

Eine typische und häufig verwendete Testfunktion für Optimierungsverfahren ist die Funktion von Rosenbrock [Ros60](3.8), auch als Banana-Valley-Funktion bekannt. Die Funktion weist ein gekrümmtes Tal mit steilen Flanken und geringem Gefälle im Talgrund auf, sodass Parameteroptimierungsverfahren vor charakteristische Herausforderungen gestellt werden und in ihrem iterativen Verhalten besonders gut beobachtet werden können:

$$f(x, y) = (x - a)^2 + b(y - x^2)^2 \tag{3.8}$$

Die Rosenbrock-Testfunktion soll als quasi-triviales Einstiegsbeispiel einer zwei-parametrigen Optimierungsaufgabe „Rosenbrock-Funktion“ die prinzipielle Funktionsfähigkeit der eingesetzten Optimierungsverfahren zeigen. Besondere Transformationen oder Semantikinterpretationen treten hier nicht auf und sind daher nicht Teil dieses Testszenarios.

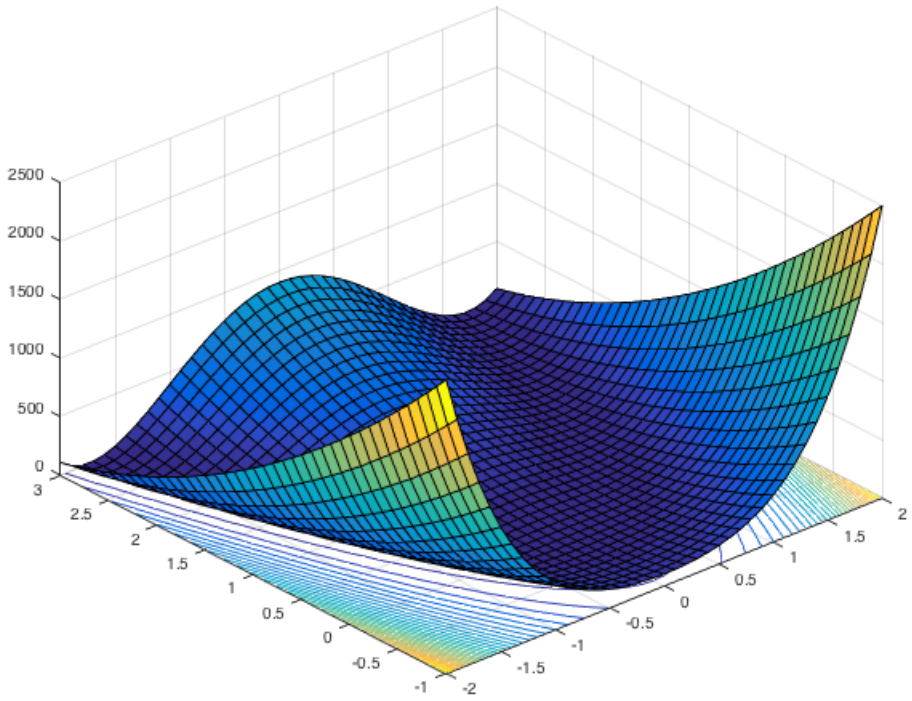


Abbildung 3.2: Rosenbrock-Testfunktion (für: a=1, b=100)

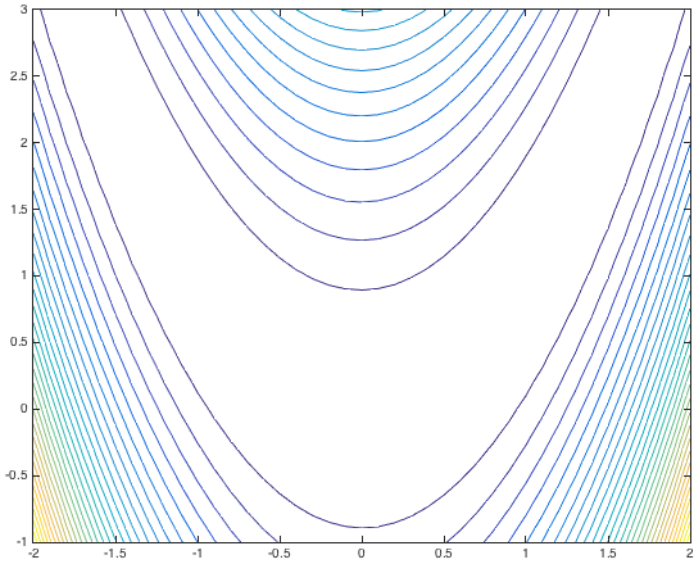


Abbildung 3.3: Rosenbrock-Testfunktion (für: a=1, b=100)

3.8.2 Kreis-Aufgabe

Gesucht sei der Radius eines Halbkreises. Gegeben ist ein Halbkreis mit unbekanntem Durchmesser, zusammen mit zwei ausgezeichneten Strecken (Abb. 3.4). Die geschlossene Lösung führt unter Verwendung des Satzes von

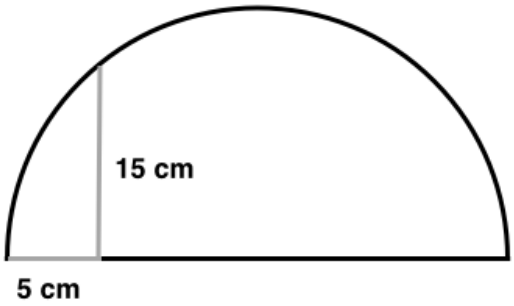


Abbildung 3.4: Kreis-Problem

Thales schnell auf den gesuchten Radius des Halbkreises. Als Optimierungsproblem aufgefasst, läuft die Parametervariation über drei Parameter: der Radius und die Position des Mittelpunktes (x- und y-Koordinate). Dies ist ein Beispiel für eine einfach formulierte Aufgabenstellung und einen Vergleich der Optimierungslösung mit der geschlossenen Lösung.

3.8.3 Leiter-Aufgabe

Die in der Mathematik bekannte Leiter-Aufgabe nach Simpson [Sim45] fragt nach der maximal erreichbaren Höhe einer Leiter, die an eine Wand gelehnt ist und dabei ein Hindernis – meist eine quadratische Box – berühren soll. Gegeben sind die Leiterlänge und die Abmessungen der Box (siehe Abb. 3.5).

Für diese Aufgabenstellung existiert eine geschlossene Lösung, die allerdings auf ein Problem 4. Grades führt. Die Aufgabenstellung soll hier als Optimierungsproblem formuliert werden, dessen Ergebnis man mit der bekannten geschlossenen Lösung vergleichen kann. Es handelt sich um ein 1-parametriges Problem, bei dem man intuitiv zunächst den Anstellwinkel der Leiter als Parameter wählt.

3.8.4 Schaf-Aufgabe

Die Schaf-Aufgabe fragt nach der Länge des Seils, mit dem ein Schaf am Rand einer kreisrunden Weidefläche angebunden werden soll, dergestalt dass das

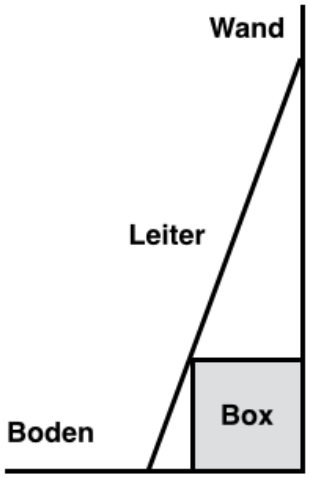


Abbildung 3.5: Leiter mit Hindernis

Schaf genau die Hälfte der Fläche abgrasen kann (siehe Abb. 3.6). Für diese

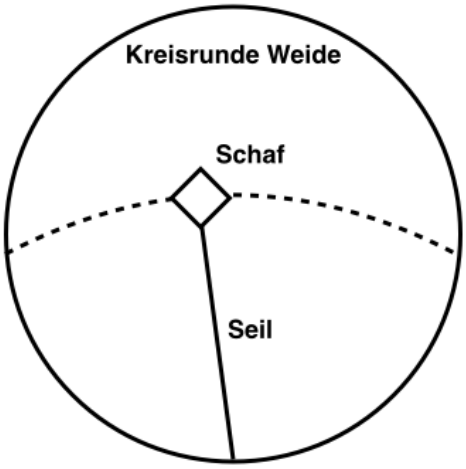


Abbildung 3.6: Schaf-Aufgabe

Aufgabenstellung existiert keine geschlossene Lösung, für ausgewählte Geometrien werden aber in der Literatur [MW14] iterativ ermittelte Lösungen genannt. Die Aufgabenstellung soll hier als Optimierungsproblem formuliert werden, dessen Ergebnis man mit bekannten iterativen Lösungen vergleichen kann. Es handelt sich um ein 1-parametriges Problem mit der Seillänge als Parameter.

3.8.5 Time Tabling

Die Verteilung von Veranstaltungen über einen definierten Zeitraum unter Nebenbedingungen wird als Time Tabling-Problematik verstanden [JTR+14]. Ei-

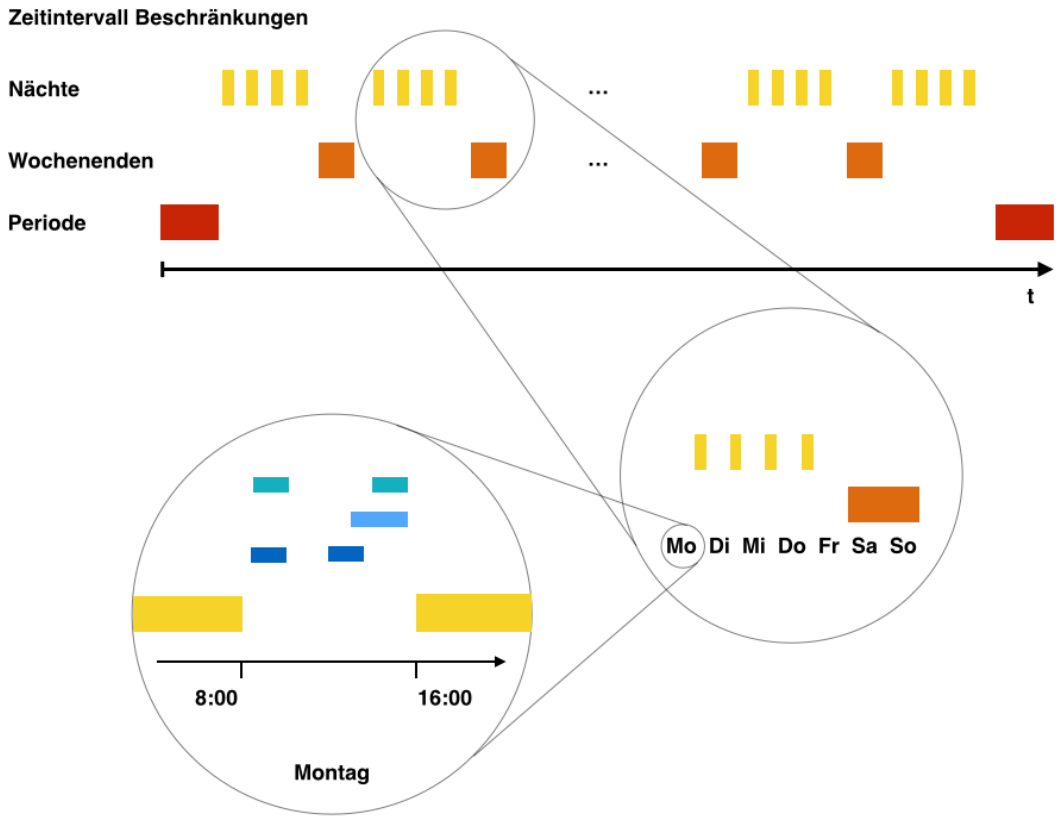


Abbildung 3.7: Time Tabling-Elemente: Sperrblöcke und geplante Ereignisse

ne feste Menge von Veranstaltungen mit jeweils bekannter Dauer soll unter Einhaltung von Bedingungen geeignet in einem Zeitintervall verteilt werden. Ein typisches Beispiel ist die Planung von Klausurterminen in einer Prüfungsperiode; Nebenbedingungen sind hier gegenseitige Ausschlüsse und Abstände, Prioritäten und die Vermeidung von Nachtzeiten und Wochenendterminen (Abb. 3.7).

Die Aufgabenstellung ist charakteristisch für Probleme mit Zielfunktionen mit vielen annähernd gleichwertigen Neben-Minima und soll überprüfen, wie in diesen Fällen Parameteroptimierung für die Lösung formuliert oder modifiziert werden muss: Im ersten Ansatz ist die Anzahl der Parameter identisch mit der Anzahl der Veranstaltungen, wenn man jeden Anfangszeitpunkt als variierbaren Parameter auffasst ($n_{Parameter} = n_{Events}$), ggf. muss man hier aber zu einer anderen Formulierung übergehen.

3.8.6 Lochwand-Platzierung

Die Verteilung und Anordnung einer definierten Anzahl von unterschiedlich großen Einzelhandelsprodukten auf einer Regal-Lochwand unterliegt einem festen Platzierungsraster und dem Ziel einer möglichst großen Platzierungsdichte bzw. Flächenfüllung (Abb. 3.8).

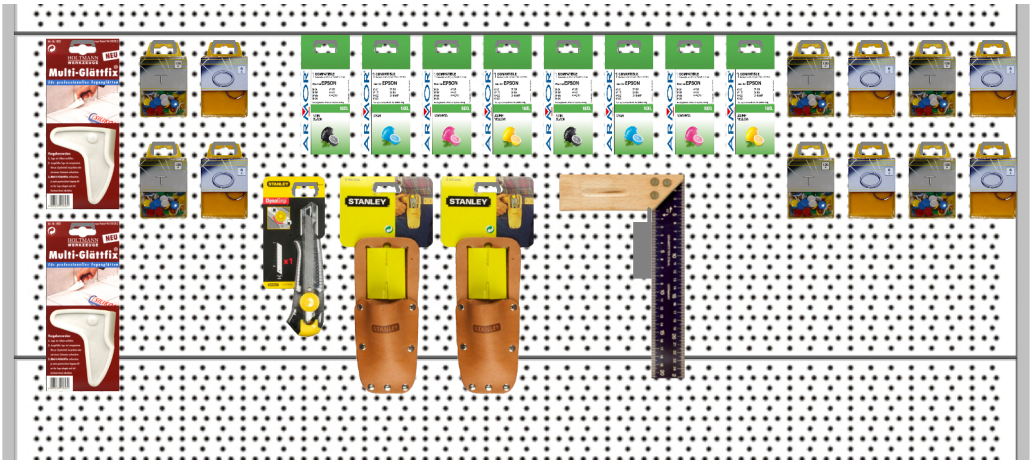


Abbildung 3.8: Lochwand-Platzierung

Es handelt sich um ein vielparametriges Problem, wenn man die x- und y-Koordinate jedes Artikels als variierbaren Parameter auffasst, da die Pro-

dukte auf einer zweidimensionalen Fläche positioniert werden ($n_{Parameter} = 2 \cdot n_{Artikel}$).

3.8.7 Paletten-Aufgabe

Aus der Logistik für Materialfluss mit Paletten oder Containern entstammt das Problem der Beladung von Paletten mit unterschiedlich großen Gütern, im allgemeinen Kartons. Logistisches Ziel ist ein Materialfluss der Güter unter Einsatz minimal eingesetzter Paletten [Sch08, PUZ15]. Gesucht ist also eine optimale Anordnung einer Menge von Kartons auf einer Palette bekannter Größe (Abb. 3.9).

Abbildung 3.9: Paletten-Aufgabe

Die Anzahl der Parameter ist linear abhängig von der Anzahl zu platzierender Kartons, da die Kartons in einem dreidimensionalen Raum platziert werden, wenn man die x-, y- und z-Koordinate jedes Artikels als variierbaren Parameter auffasst: ($n_{Parameter} = 3 \cdot n_{Karton}$).

3.8.8 Pattern Matching

Als Pattern Matching wird die Erkennung von Mustern in Bildern bezeichnet; die Anwendungsfälle und Begriffsausprägungen sind vielfältig. Im Kontext dieser Arbeit wird die strukturelle Mustererkennung [HSD01] in 2D-Bildern betrachtet. Dabei wird ein zweidimensionales Bild (Abb. 3.10) nach einem definierten Muster durchsucht, wobei Clipping und Ähnlichkeiten berücksichtigt werden sollen [JM13, JLB+14, JTL+15].

Ohne Berücksichtigung von Rotation und Farbe beschränken sich die Parameter auf die durch eine x- und y-Koordinate gegebene Position des Suchmusters:

$$n_{Parameter} = 2$$

Soll Rotation des Suchmusters mitberücksichtigt werden, wird diese als dritter Parameter eingeführt:

$$n_{Parameter} = 3$$

Soll zusätzlich auch die Farbe des Suchmusters mitberücksichtigt werden, wird diese als vierter Parameter eingeführt:

$$n_{Parameter} = 4$$

Interessant an dieser Problemstellung im Rahmen dieser Arbeit ist die potenzielle Unschärfe in der Szenariobeschreibung, die Trennung und Identifikation von Parametern und Bedingungen, die formale Beschreibung der Muster-Form sowie die Berücksichtigung des auf den ersten Blick nicht-numerischen Parameters *Farbe*.

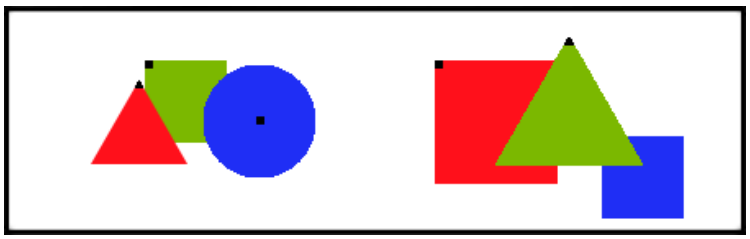


Abbildung 3.10: Beispiel für Mustererkennung: Gesucht ist ein grünes Quadrat

3.8.9 Lemniskaten-Aufgabe

Doppellenker-Wippkrane (Abb. 3.11) sollen aus Energie- und Handhabungsgründen einen möglichst horizontalen Weg der Auslegerspitze besitzen.

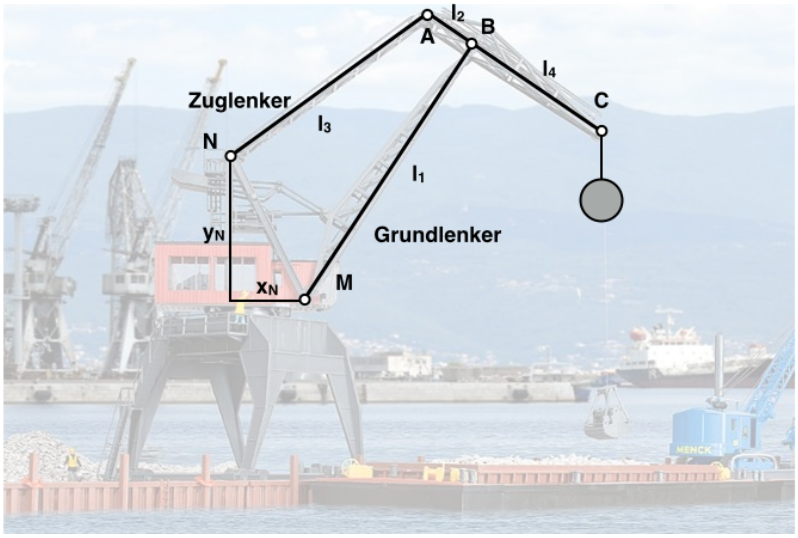


Abbildung 3.11: Doppellenker-Kran

Das zugrunde liegende Viergelenkgetriebe führt auf eine Lemniskatenkurve, die durch die Geometrie der Lenkeranordnung beschrieben wird [Kur65, May81, Vol73, Kon41]. Die Längen der Lenker ($l_1 = 10800$, $l_2 = 3000$, $l_3 = 9900$ und $l_4 = 5000$) und die Position des Zuglenker-Stützpunktes N ($x_N = -4000$, $y_N = 5000$) sind so zu wählen, dass beim Ein- und Ausziehen des Wippwerks ein möglichst horizontaler Lastweg gegeben ist, sodass keine Hubkorrekturen notwendig sind und der Energieaufwand minimal ist.

Für sechs Parameter x_N, y_N, l_1, l_2, l_3 und l_4 ist die bestmögliche Parameterkombination zu finden. Punkt M ist gegeben bzw. wird als Nullpunkt definiert. Der zu überstreichende Wippweg (Abb. 3.12) ist mit einem Minimal- und einem Maximal-Auslegerwert gegeben. Der Weg des Punktes C soll optimiert werden.

Das Lemniskaten-Problem ist ein reales Optimierungsproblem in mehreren Parametern mit einer typischen Aufgabenstellung aus dem Maschinenbau.

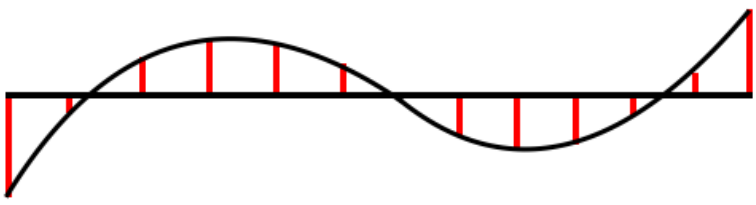


Abbildung 3.12: Lemniskaten-Wippweg des Punktes C (qualitativ, überhöht)

Kapitel 4

Phasenbasiertes Transformationsmodell

Weil eine realweltliche Optimierungs-Aufgabenstellung zu unscharf ist, um sie direkt berechnen zu lassen, müssen die relevanten Kriterien in der Aufgabenstellung aufgefunden und so aufbereitet und umformuliert werden, dass sie von einem Optimierungsverfahren richtig übernommen und die Optimierungsergebnisse wieder in den Sinnzusammenhang der Aufgabenstellung zurückgeführt werden können.

Das generalisierte Optimierungsverfahren setzt bestimmte numerische Konzepte voraus, die für die Bearbeitung einer konkreten Aufgabenstellung erfüllt werden müssen. Dafür müssen die expliziten und impliziten textlichen Aussagen der Aufgabenstellung in diese Konzepte überführt werden. Die Interpretation der vom Optimierungsverfahren erhaltenen numerischen Ergebnisse im Sinne der ursprünglichen Aufgabenstellung entspricht einer Rücküberführung in den semantischen Kontext der Aufgabenstellung.

Dieses Kapitel führt einen phasenbasierten Transformationsprozess ein, mit dem die Hin- und Rück-Überführung zwischen Aufgabenstellung und Optimierungsverfahren standardisiert durchgeführt werden kann. Die Anwendung einer solchen kontrollierten Transformation formalisiert die Anwendung von Optimierung, gestaltet sie leichter anwendbar, überprüfbar und führt zu reproduzierbaren Ergebnisinterpretationen.

4.1 Aufgabenstellung

In den verschiedensten Fachgebieten stellen sich regelmäßig zu lösende Probleme, die je nach Gebiet und Kontext unterschiedlichen Charakter haben. Während sich in der Geisteswissenschaft oder gesellschaftlichen Problematik die Probleme eher interpretativ äußern und deren Lösung von der Interpretation abhängen, konzentriert sich der hier eingeführte Begriff der Aufgabenstellung auf solche, die nicht zu interpretieren sind, sondern automatisiert und programmierbar lösbar sind. Das wieder impliziert noch nicht, dass die Aufgabenstellung direkt formal beschrieben ist. Aufgaben dieser Art können durchaus mit Worten formuliert und ausgeschmückt werden, die nach genauerer Betrachtung wenig oder nichts zum Informationsgehalt beitragen.

4.1.1 Probleme und ihre Charakterisierung

Als Problemstellung sollen hier solche ergebnisoffenen Fragestellungen verstanden werden, für die eine Lösungsstrategie prinzipiell möglich erscheint. Dazu gehören naturgemäß auch die Problemstellungen, für die Lösungsstrategien bereits bekannt sind.

Eine Problemstellung ist im allgemeinen Fall als formelle oder informelle Erzählung formuliert, die einen eigenen Kontext hat und einen nicht-trivialen Komplexitätsgrad aufweist. Ziel der Problemstellung ist die Anwendung einer Lösungsstrategie, die die zugrunde liegende Fragestellung befriedigend beantworten soll. Die der Problemstellung zugrunde liegende Erzählung besteht aus Details, die die Bedingungen, Ausgangs- und Zielwerte der Fragestellung definieren. Nicht alle angegebenen Details sind für die Zielerreichung notwendig oder hilfreich, ihre Relevanz ist aber nicht immer unmittelbar erkennbar. Im Hinblick auf eine angewendete Lösungsstrategie ist jedoch das Erkennen des informationstragenden Kerns der Problemstellung unverzichtbar und elementar.

Einer Problemstellung liegt damit immer die Intention einer Lösung zugrunde. Konzentriert und beschränkt man sich auf automatisiert lösbare oder auf darauf rückführbare Problemstellungen, dann zielt die Lösung evidenterweise auf automatisierbare und programmierbare Lösungsstrategien ab, die prinzipiell in zwei Kategorien unterschieden werden können (vgl. Abschnitt 2.1):

1. Deterministische Lösungsstrategien
   Die Verfahren, die deterministisch eine Lösung bestimmen, liefern für jeden Durchgang mit gleichem Aufwand – unter gleichen Bedingungen – eine eindeutig beschriebene und reproduzierbare Lösung, d.h. ein geschlossener Algorithmus ist bekannt bzw. kann gefunden und formuliert werden.

2. Probabilistische Lösungsstrategien
   Für viele Problemstellungen existieren keine geschlossenen deterministischen Algorithmen, oder solche Algorithmen sind zum gegebenen Zeitpunkt nicht bekannt. Jedoch lassen sich für hinreichend bestimmte Problemstellungen immer bewertende, annähernde oder wahrscheinlichkeitstheoretische Aussagen über bestimmte – zur Lösung beitragende – Sachverhalte finden. Dieser stochastische Ansatz bedient mit seinen Aussagen die Prinzipien der Optimierungsverfahren und führt über schrittweise Näherungslösungen zu einer Zielmenge mit problemlösenden Elementen.

Im Fall, dass ein geeigneter geschlossener Algorithmus für eine Lösungsstrategie bekannt und verfügbar ist, sollte dieser bevorzugt angewendet werden, da er im Allgemeinen eine effektive Performanz verspricht. Anderenfalls muss zu einem probabilistischen Verfahren gegriffen werden, dessen Vorgehen nach dem Prinzip des Versuch-und-Irrtum tendenziell eine schlechtere Performanz erwarten lässt, aber für eine Lösungserreichung häufig unvermeidlich ist. Während das vollständige Durchsuchen des Lösungsraums sicher auf eine geeignete Lösung führt (sofern eine Lösung überhaupt existiert), verbietet sich ein solcher Brute-Force-Ansatz in der Regel aus Performanzgründen; stattdessen versuchen Optimierungsverfahren einen „guten“ Weg durch den Lösungsraum zu beschreiten und dabei auf eine möglichst gute Lösung zu kommen.

Im Folgenden sollen nur automatisiert lösbare Problemstellungen, auf die probabilistische Lösungsstrategien angewendet werden können, betrachtet werden. Solche Problemstellungen werden hier *Aufgabenstellung* genannt. Einer Aufgabenstellung liegt eine Erzählung (engl. *story*) zugrunde, die die Bedingungen und Werte der Problemstellung beschreibt. Diese Erzählung wird im Folgenden *Szenario* genannt. Die *Transformation* ist dann die Überführung des Szenarios in eine formale Modellierung aller Bedingungen und Werte, die

der probabilistischen Lösungsstrategie zugeführt werden kann.

Während die Entscheidung über die automatisiert-programmatische Lösbarkeit einer Problemstellung relativ leicht erscheint, ist die Entscheidung zwischen deterministischer Lösung, also der Anwendbarkeit eines geschlossenen Lösungsalgorithmus, und der Anwendung eines probabilistischen Lösungsverfahrens, nicht so einfach zu treffen.

Ein erster Hinweis für die Entscheidung zwischen einer geschlossen-deterministischen oder probabilistischen Lösungsstrategie kann die Abschätzung der Kardinalität der auffindbaren Lösungen sein. Folgende Fälle können unterschieden werden:

1. Es gibt genau eine Lösung: $|\overline{p}_L| = n_L = 1$

2. Es gibt endlich viele Lösungen: $|\overline{p}_L| = n_L > 1$

3. Es gibt unendliche viele Lösungen: $|\overline{p}_L| = n_L \to \infty$

4. Keine Lösung: $|\overline{p}_L| = n_L = 0$

Anhand der Fälle kann eine Präferenz für eine Lösbarkeitskategorie abgeleitet werden:

1. Der Fall genau einer erwartbaren Lösung ist ein Hinweis auf eine deterministische Lösbarkeit.

2. Sind mehrere oder viele Lösungsalternativen denkbar und liegt eine Bewertbarkeit der Alternativen vor, ist das ein Hinweis auf die Anwendbarkeit einer probabilistischen Lösungsstrategie.

3. Lässt eine Lösungsstrategie unendlich viele Lösungen erwarten, kommt das einer Nicht-Lösbarkeit der Problemstellung nahe, wenn keine eindeutige Entscheidung für eine Variante getroffen werden kann; ersatzweise kann man versuchen, die Lösungsmenge zu beschränken. Eine probabilistische Lösungsstrategie, die eine Bewertungsfunktion beinhaltet, kann aber eine geeignete Lösung identifizieren.

4. Dies ist der triviale Fall der Nicht-Lösbarkeit.

Ein weiterer Hinweis auf die Erfordernis eines probabilistischen Lösungsverfahrens besteht dann, wenn in einer kombinatorischen Fragestellung offenbar nur ein iterativer Ansatz erfolgversprechend ist, insbesondere wenn die iterativen Lösungsalternativen mit einer Gütefunktion bewertbar sind. Dies ist in der Regel bei Problemstellungen, die Nichtlinearitäten enthalten, der Fall.

Ein eindeutiger Hinweis auf die Anwendbarkeit eines deterministischen Lösungsverfahrens ist selbstverständlich, wenn ein geschlossener Algorithmus bekannt und tatsächlich physisch verfügbar ist.

Die Unterscheidung zwischen deterministischen und probabilistischen Lösungsstrategien ist nicht immer eindeutig und nur für jeden Einzelfall entscheidbar. Während man häufig eine deterministisch lösbare Problemstellung auch probabilistisch angehen kann, gilt dies umgekehrt nicht unbedingt. Generell kann man daher probabilistische Lösungsstrategien als universeller einsetzbar ansehen.

4.1.2 Daten und Informationen

Im Rahmen der Informationstechnologie werden die Begriffe Daten und Informationen einerseits oft synonym [LM94, Dat95] verwendet, andererseits aber auch gegeneinander abgegrenzt [Gem93, Wit59]. Für das Thema dieser Arbeit erscheint es zielführend, diese Begriffe semantisch zu unterscheiden.

Daten werden je nach Sichtweise in der Literatur unterschiedlich definiert. Die ISO/IEC 2382-1 (1993) beschreibt Daten wie folgt:

> „a reinterpretable representation of information in a formalized manner, suitable for communication, interpretation, or processing“.

Hier wird also auf die Interpretierbarkeit von Daten und ihre maschinelle Verarbeitbarkeit abgehoben. Diese Vorstellung findet sich generell in der Informatik und Datenverarbeitung wieder. Daten können darüber hinaus nach Beständigkeitsgrad und Strukturierungsgrad kategorisiert werden.

Im Gegensatz zu diesem Daten-Begriff sind Informationen dadurch gekennzeichnet, dass sie der menschlichen Interpretation – oder allgemeiner: der Interpretation des Rezipienten – bedürfen. Der Rezipient fügt den Daten einen semantischen Bedeutungskontext hinzu, der die passende Interpretation erst

sinnvoll macht. Information entsteht also durch Aggregation und Rekonstruktion aus Daten unter Hinzuziehung eines ontologischen Bedeutungskontexts: Wenn man die Begriffe Information und Daten als Antagonisten in einem interpretatorischen Kontext betrachtet, ist der Übergang von Information zu Daten eine De-Konstruktion, während der Übergang von Daten zu Information eine Re-Konstruktion ist (Abb. 4.1). Die De-Konstruktion überführt die Ontologie der Informationen in ein technisches Kondensat maschinenverarbeitbarer Daten und damit in eine durch die Datenstrukturen definierte Ontologie eigener Art.

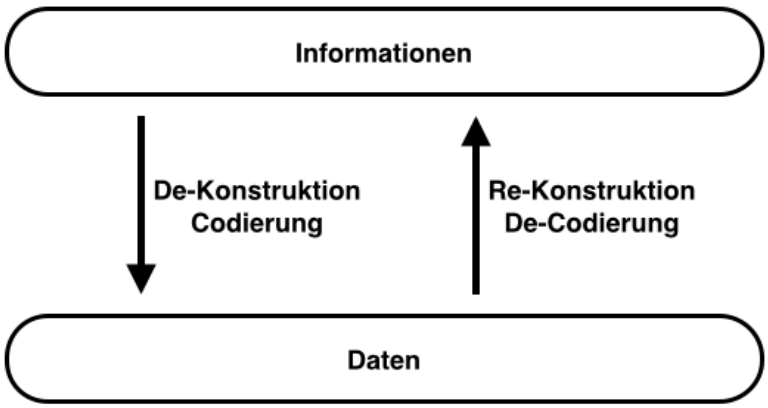


Abbildung 4.1: Informationen und Daten

4.1.3 Interpretatorische Lücke

Die Informationen-Daten-Ambivalenz findet sich in konkreter Ausprägung bei der Behandlung von probabilistischen Aufgabenstellungen. Hier sind die Informationen in informeller, semantischer Formulierung in der Aufgabenstellung enthalten, während das probabilistische Lösungsverfahren eine technische, maschinennahe Implementierung der Bedingungen in Form von Daten und Datenstrukturen benötigt.

Abb. 4.2 zeigt den Gesamtzusammenhang: Ein Initiator verfasst ein *Szenario*, welches eine Problemstellung der Realwelt beinhaltet, die durch Sachverhalte und Aussagen (*Informationen*) beschrieben ist. Diese Informationen müssen nun in für das Lösungsverfahren geeignete, maschinenverarbeitbare *Daten* überführt werden. Die resultierenden Daten und Datenstrukturen stehen nun für die Verarbeitung durch das *Lösungsverfahren* bereit. Nach der Berechnung

der Lösung liegen Ergebnisdaten vor, die wieder mit Semantik entsprechend dem Bedeutungskontext angereichert und auf das Szenario rücktransformiert werden müssen, um eine der Problemstellung angemessene Lösung in der Realwelt liefern zu können.

Die Transformation *Informationen* → *Daten* und die Rücktransformation *Daten* → *Informationen* überbrücken die interpretatorische Lücke zwischen der Aufgabenformulierung in der Realwelt und der Lösungsimplementierung in der Maschinenwelt. Die Transformation ist bidirektional. Somit entspricht die Transformation der zuvor beschriebenen De-Konstruktion und die Rücktransformation der Re-Konstruktion.

Dieser bidirektionale Charakter der Transformation unterscheidet an der interpretatorischen Lücke zwei Seiten: diejenige, die semantisch und kontextbezogen eine Problemstellung erklärt, und diejenige, die maschinennahe Datenstrukturen dem Lösungsverfahren zuführt, welches dann eine Lösung produziert. Die Transformation befreit einerseits die Informationen von den realweltlichen Semantiken und erhöht andererseits den Strukturierungsgrad der Daten. Um mit der abschließenden Rücktransformation der Lösungsdaten semantisch angemessen eine Antwort auf die Problemstellung im Sinne des Szenarios zu ermöglichen, können die semantischen Informationsteile vor der Transformation von den erzeugten Datenstrukturen referenziert werden, sodass die Rücktransformation auf diese semantischen Informationsteile zurückgreifen kann. Ansonsten ist eine Rücktransformation der Daten in den Kontext der Aufgabenstellung nicht automatisiert möglich.

Hin- und Rück-Transformation unterscheiden sich dahingehend, dass die Hin-Transformation alle relevanten Informationen verarbeiten muss, während die Rück-Transformation nur die reinen Ergebnisdaten betrachten muss. Nicht jeder Transformationsschritt muss also invertierbar sein, sondern nur ausgewählte Daten-Teilmengen: Nur die Ergebnisdaten werden unter Einbeziehung von Szenario-relevantem Wissen mit Bedeutungskontext angereichert und auf das semantische Niveau des Szenarios gehoben.
Eine Überwindung der interpretatorischen Lücke erfordert somit eine sorgfältige Betrachtung und Gestaltung dieses bidirektionalen Übergangs im Sinne einer implementierbaren Schnittstelle.

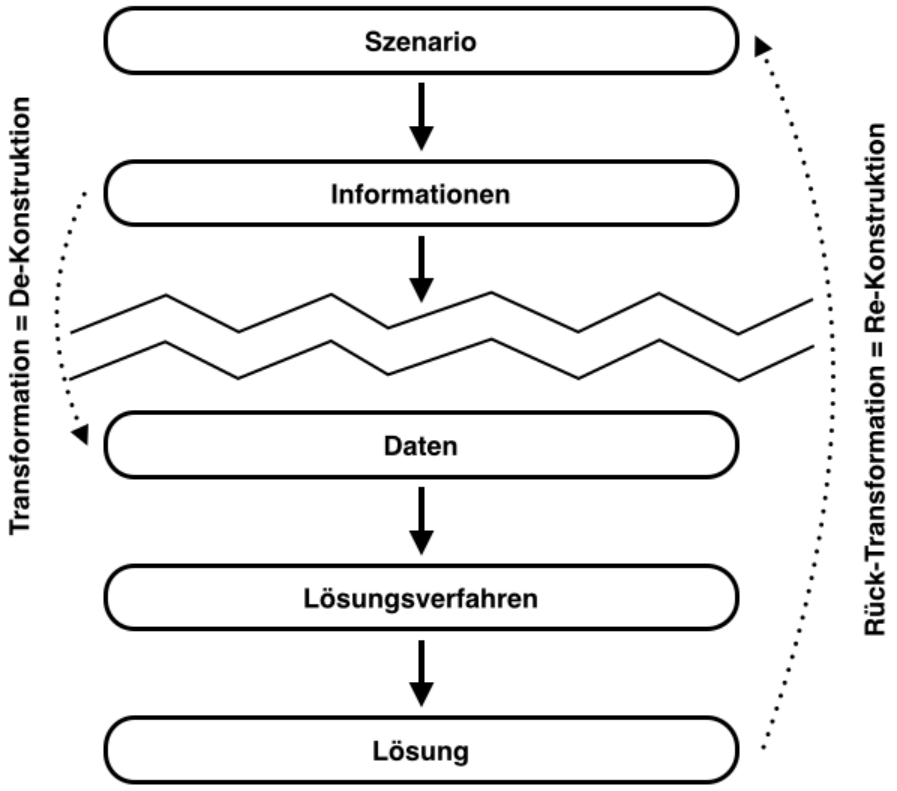


Abbildung 4.2: Transformation und Rück-Transformation entlang der interpretatorischen Lücke

4.2 Ziele und Anforderungen

4.2.1 Ziel der Transformation

Ausgangspunkt für den Transformationsprozess ist die gestellte Aufgabe. Jede probabilistische Aufgabenstellung hat zunächst einen nicht-formalen Charakter, meist in einer natürlichsprachlichen Form. Die Transformation generiert daraus ein für das Lösungsverfahren geeignetes transformiertes Datenmodell.

Die Aufgabe der Transformation ist somit das Schließen der Lücke zwischen der Aufgabenstellung und dem Beginn des Lösungsverfahrens im Sinne des generalisierten Optimierungsverfahrens gemäß Kap. 3 und damit die Übertragung der Kriterien der Aufgabenstellung in die Ontologie der Optimierung. Zu beachten ist, dass diese Transformation bidirektional ist, da die Ergebnisdaten der Optimierung wieder in die Informationsebene der Aufgabenstellung rücküberführt werden müssen.

4.2.2 Anforderungen an die Transformation

Innerhalb der Aufgabenstellung muss mindestens ein Szenario enthalten sein. Dieses muss veränderbare Werte und begrenzende Bedingungen enthalten, um dem fundamentalen Anspruch an Lösbarkeit zu genügen. Für das intendierte probabilistische Lösungsverfahren sollen aus der Aufgabenstellung alle relevanten Kriterien extrahiert werden. Dabei ist die prinzipielle Anforderung an die Informationstransformation die angemessene Bereitstellung der Eingangswerte für das generalisierte Optimierungsverfahren.

Im Detail bedeutet das für den Transformationsprozess, dass folgende Integritätsbedingungen erfüllt werden müssen:

1. Korrektheit:
   Die in dem Ausgangs-Szenario enthaltenen Tatsachen müssen korrekt in das transformierte Datenmodell übernommen werden.

2. Vollständigkeit:
   Die Transformation darf keine relevanten Informationen unterdrücken oder unberücksichtigt lassen.

3. Berechenbarkeit:
   Die Transformation muss die Lösbarkeit gewährleisten und die Berechenbarkeit herbeiführen.

4. Determiniertheit:
   Die Transformation muss deterministisch sein.

5. Stabilität:
   Die Transformation muss stabil und robust sein.

6. Umkehrbarkeit:
   Die Transformation muss bidirektional anwendbar sein.

Eine Verletzung einer der aufgeführten Voraussetzungen führt zu folgenden Konsequenzen:

1. Eine falsch übernommene, verzerrte, nur teilweise oder falsch interpretierte Tatsache beeinträchtigt eventuell nicht die Lösbarkeit, widerspricht aber dem Anspruch auf eine richtige Lösung.

2. Die Nichtberücksichtigung oder Unterdrückung von relevanten Informationen führt zu einer Lösung, die relevante Parameter und Bedingungen nicht beinhaltet. Das Ergebnis erfüllt somit nicht den Anspruch an eine Lösung gemäß der Aufgabenstellung.

3. Die Transformation muss ein Datenmodell für das gewählte Lösungsverfahren bereitstellen, das die Lösbarkeit gewährleistet. Andernfalls kann das Lösungsverfahren nicht durchlaufen werden und liefert dadurch kein Ergebnis.

4. Eine nicht reproduzierbare deterministische Transformation liefert bei mehrfachem Durchlaufen kein identisches Datenmodell und gefährdet die Nachvollziehbarkeit der Ergebnisgewinnung und die Reinterpretation der Ergebnisse im Sinne der Aufgabenstellung.

5. Eine instabile Transformation würde einerseits Satz 4 verletzen, andererseits kann eine nicht stabile und nicht robuste Transformation nicht durchlaufen werden.

6. Eine Transformation ohne bidirektionale Eigenschaft lässt sich nicht in das Informationsmodell der Aufgabenstellung zurückführen.

4.2.3 Katalog validierbarer Anforderungen

In dieser Detaillierung sind die Anforderungen aus 4.2.2 nicht exakt verifizierbar, daher wird ein summarischer Katalog von Anforderungen an eine Transformationsvorschrift formuliert, die überprüfbar sind und erfüllt sein müssen, damit die Transformationsvorschrift als valide gelten kann:

1. Die Transformation muss auf eine Datenstruktur führen, die mit einem Parameteroptimierungsverfahren lösbar ist. Konkret muss die Transformation die Elemente der Optimierungs-Ontologie Parameter, Nebenbedingungen und Zielfunktion enthalten. Anderenfalls wäre die Aufgabenstellung nicht mithilfe eines Optimierungsverfahrens lösbar.

2. Die Transformationsvorschrift muss eine vollständig invertierbare Datenstruktur erzeugen. Das nach Optimierung erwartete Ergebnis unterscheidet sich nicht formal, sondern nur wertemäßig von der Eingabe-

Datenstruktur der Optimierung. Die Ergebnis-Datenstruktur muss auf die semantische Ebene der Informationen im Sinne der Aufgabenstellung rücktransformierbar sein. Diese Rücktransformation kann man aber testweise bereits mit der Datenstruktur nach Transformation durchführen: wenn diese tatsächlich invertierbar ist, war die Transformation offensichtlich umkehrbar und daher strukturell korrekt.

4.3 Risiken der Transformation

Die probabilistische Aufgabenstellung liegt zunächst in einer mehr oder weniger natürlichsprachlichen Form vor, ggf. ergänzt durch formelhafte Angaben und numerische Vorgaben. Es wird angenommen, dass die Aufgabe von einem *Initiator* eingebracht und von einem *Bearbeiter* der Lösung zugeführt wird.

Für eine geeignete Überführung der Anforderungen aus der Problemstellung in die Begrifflichkeiten des angewendeten Lösungsverfahrens muss sichergestellt sein, dass die Anforderungen angemessen erkannt, interpretiert und überführt werden können. Missverständnisse und Unklarheiten semantischer und syntaktischer Art in der formulierten Aufgabenstellung sind allerdings im allgemeinen Fall zu erwarten. Diese Fehlinformationen müssen erkannt und aufgelöst werden. Gelingt dies nicht, muss die Aufgabe als unlösbar betrachtet werden.

4.3.1 Fehlformulierungen

Hier können *syntaktische* und *semantische* Fehlformulierungen unterschieden werden:

Unter der Annahme, dass die Aufgabenstellung in einer natürlichen Sprache formuliert ist, sind Syntaxfehler Rechtschreib- und Grammatik-Fehler, wobei hier auch falsche Satzzeichen subsumiert sein sollen. Ein typisches Beispiel für eine syntaktische Fehlformulierung aufgrund falscher Zeichensetzung ist:

Komm, wir essen, Opa! vs. *Komm, wir essen Opa!*

Weitere linguistische Problemstellungen sollen hier nicht weiter diskutiert werden, vielmehr wird im Folgenden von einer syntaktisch korrekten Verwendung der Sprache ausgegangen. Da aber auch eine natürlichsprachlich formulierte Aufgabenstellung formalisierte Bestandteile – Formeln und allgemeine

mathematische Terme – haben kann, kann die Diskussion der syntaktischen Korrektheit auf solche formalen Bestandteile beschränkt werden. Ein Syntaxfehler in einer formalen Beschreibung impliziert entweder eine absichtlich oder unabsichtlich falsche Aussage oder eine nicht-brauchbare, unsinnige Aussage.

Im Folgenden wird also davon ausgegangen, dass die Aufgabenstellung sprachlich und formal korrekt im Sinne einer Syntax ist. Im Falle, dass diese Voraussetzung nicht erfüllt ist, wäre die Aufgabenstellung an dieser Stelle bereits als nicht – jedenfalls nicht-problemadäquat – lösbar identifiziert. Eine Übertragung der Aufgabe in einen formalisierten Lösungsansatz ist dann naturgemäß nicht mehr sinnvoll.

Semantische Fehlinformationen basieren demgegenüber nicht auf formalen Kriterien, sondern auf einer Untersuchung des Sinngehalts der Aussagen. Falsche Aussagen sind zu vermeiden, da sie die Lösbarkeit der Aufgabe grundsätzlich in Frage stellen. Falsch in diesem Sinne sind alle Aussagen, die wahrheitswidrig oder widersprüchlich sind. Redundanzen können, müssen aber nicht zu Widersprüchen führen. Unter- oder Überbestimmtheit der Aufgabe können ebenfalls als semantische Fehlinformation betrachtet werden.

Konsequenz: Fehlformulierungen liegen im Verantwortungsbereich des Initiators der Aufgabe. Der Initiator hat die Aufgabe so zu formulieren, dass sie vollständig und fehlerfrei ist.

4.3.2 Fehlinterpretationen

Auch bei Annahme, dass die Aufgabenstellung in syntaktisch und semantisch fehlerfreier Form vorliegt, besteht die Möglichkeit einer fehlerhaften Interpretation durch den Bearbeiter.

Eine Fehlinterpretation ist ein Missverstehen einer an sich korrekten Formulierung der Aufgabenstellung durch den Bearbeiter. Neben schlichtem Irrtum und mangelndem Kontextwissen des Bearbeiters kann dafür eine nicht-eindeutige Nomenklatur, also ein divergierendes Verständnis verwendeter (Fach-)Begriffe sein.

Konsequenz: Eine Fehlinterpretation liegt im Verantwortungsbereich des Bearbeiters.

4.3.3 Unschärfe

Es sind Unklarheiten in der Aufgabenstellung vorstellbar, bei denen nicht eindeutig entscheidbar ist, ob es sich dabei um eine Fehlformulierung oder um eine Fehlinterpretation handelt. Initiator und Bearbeiter können darüber streiten, ob eine ungenaue Formulierung in der Aufgabenstellung vorliegt oder ob die Formulierung falsch interpretiert worden ist.

Konsequenz: Die Unschärfe impliziert, dass die Verantwortung nicht eindeutig dem Initiator oder dem Bearbeiter zugeschrieben werden kann. Hingegen hat der Initiator die Pflicht, interpretationsfrei zu formulieren.

4.3.4 Unter- und Überbestimmtheit

Werden in der Aufgabenstellung fehlende Angaben erkannt, liegt *Unterbestimmtheit* vor. Fehlende Angaben bedeuten hier fehlende Bedingungen und Werte, sodass angenommen werden muss, dass die Transformation dem Lösungsverfahren nicht die erforderlichen Daten liefern kann.

Enthält die Aufgabenstellung nach Elimination von Redundanzen zu viele Informationen, können vier Fälle unterschieden werden:

1. Es gibt Bedingungen, die in Widerspruch zueinander stehen.
2. Es gibt konkurrierende Bedingungen, ohne dass ein unmittelbarer Widerspruch erkennbar wäre.
3. Es gibt transitive Bedingungen, die auseinander hervorgehen oder aufeinander aufbauen und daher keine neue Information beisteuern.
4. Die Bedingungen engen die Lösungsmöglichkeiten zu sehr ein, sodass keine geeignete Lösung mehr gefunden werden kann.

In allen diesen Fällen liegt *Überbestimmtheit* vor. Ihr Auftreten führt zu folgenden Konsequenzen:

1. Widersprüchliche Informationen lassen keine Lösung zu, ohne Bedingungen zu verletzen.
2. Konkurrierende Bedingungen lassen sich nur separat auf Kosten divergenter Bedingungen erfüllen, es können nicht alle Bedingungen zur gleichen Zeit erfolgreich erfüllt werden.

3. Transitive Bedingungen verdecken durch ihr Auftreten in mehreren separaten Bedingungen eine zusammenfassbare Bedingung.

4. Bedingungen, die jeweils disjunkte Zielgebiete bestimmen, lassen kein gemeinsames Zielgebiet zu. Ebenso kann das Zielgebiet komplett verschwinden.

Eine Untersuchung auf Vorliegen von Über- oder Unterbestimmtheit kann sinnvoll erst erfolgen, nachdem Redundanzen aus den Bedingungen der Aufgabenstellung eliminiert worden sind. Informationen sind redundant, wenn sie in ihrer inhaltlichen Aussage ganz oder teilweise identisch sind. Eine scheinbare Überbestimmtheit kann sich nach Elimination der Redundanzen als tatsächliche Unterbestimmtheit herausstellen.

Über- und Unterbestimmtheiten könnten als Teil semantischer Fehlformulierung verstanden werden. Allerdings handelt es sich hier nicht um fehlerhafte Formulierungen im eigentlichen Sinne, sondern um ein Zuviel oder Zuwenig an Information, unabhängig von der Frage, ob der Rest der Aufgabenstellung missverständlich oder fehlerhaft formuliert sein könnte.

Konsequenz: Der Initiator hat die Verantwortung für die angemessene Auswahl der Bedingungen in der Aufgabenstellung.

4.3.5 Uneinheitliche Wertebasis

Iterative Lösungsverfahren für probabilistische Aufgabenstellungen basieren in der Regel – und insbesondere im Zusammenhang mit Parameteroptimierungsverfahren – auf dem Vergleich von Lösungsvarianten. Um die Vergleichbarkeit herzustellen, müssen die variierbaren numerischen Werte einer einheitlichen, vergleichbaren Wertebasis entstammen.

Sollte das nicht der Fall sein und eine Konsolidierung oder Normalisierung der Wertebasen nicht möglich sein, ist die Vergleichbarkeit und damit die Ermittlung eines Gütewertes jeder Lösungsvariante nicht gegeben.

Konsequenz: Aufgabenstellungen, die sich einer Konsolidierung der Werte entziehen, sind nicht lösbar.

4.3.6 Ex falso quodlibet

Divergierende und konkurrierende Formulierungen müssen im Sinne der Information zu einer eindeutigen Aussage führen.

Die Interpretation aufgrund einer Fehlformulierung impliziert nach dem Prinzip des „ex falso quodlibet“ – unter Berücksichtigung nicht nur tatsächlicher Gegebenheiten, sondern auch logisch falsch basierter Fakten – eine beliebige Aussage.

Konsequenz: Der Initiator ist für die faktische Korrektheit der Angaben der Aufgabenstellung verantwortlich.

4.4 Informationsmodellierung der Transformation

Nachdem die Anforderungen – in Form eines Anforderungskataloges – und Ziele für die Transformation bekannt sind und die Risiken aufgezeigt wurden, wird nun eine Informationsmodellierung der Transformation in Form von einzelnen Phasen vorgeschlagen. Diese Phasen erfüllen die Anforderungen nach Abschnitt 4.2.2 und sollen die Risiken nach Abschnitt 4.3 minimieren.

Der prinzipielle Ablauf eines allgemeinen Optimierungsverfahrens wird in Abb. 3.1 dargestellt. Die grau hinterlegten Strukturen repräsentieren die Optimierungs-Ontologie; sie müssen vor dem Lösungsstart dem Optimierungsverfahren zur Verfügung gestellt werden. Dazu werden sie im nachfolgenden Phasenmodell der Transformation erkannt und bereitgestellt.

Das vorgeschlagene Phasenmodell soll in einem strukturierten Vorgehen sicherstellen, dass alle aufgezeigten Ziele erfüllt werden, unter möglichster Vermeidung der Risiken. Abb. 4.3 zeigt die Transformationsphasen im Gesamtablauf, die Details jeder einzelnen Phase werden im Folgenden beschrieben.

4.4.1 Identifikation

Das Phasenmodell der Transformation beginnt in der Identifikationsphase mit der semantischen Analyse des Szenarios; hier werden die Kriterien identifiziert und kategorisiert. Die Unterteilung erfolgt zunächst möglichst atomar unter Klassifizierung nach Variablen, Bedingungen und einer Zielbeschreibung.

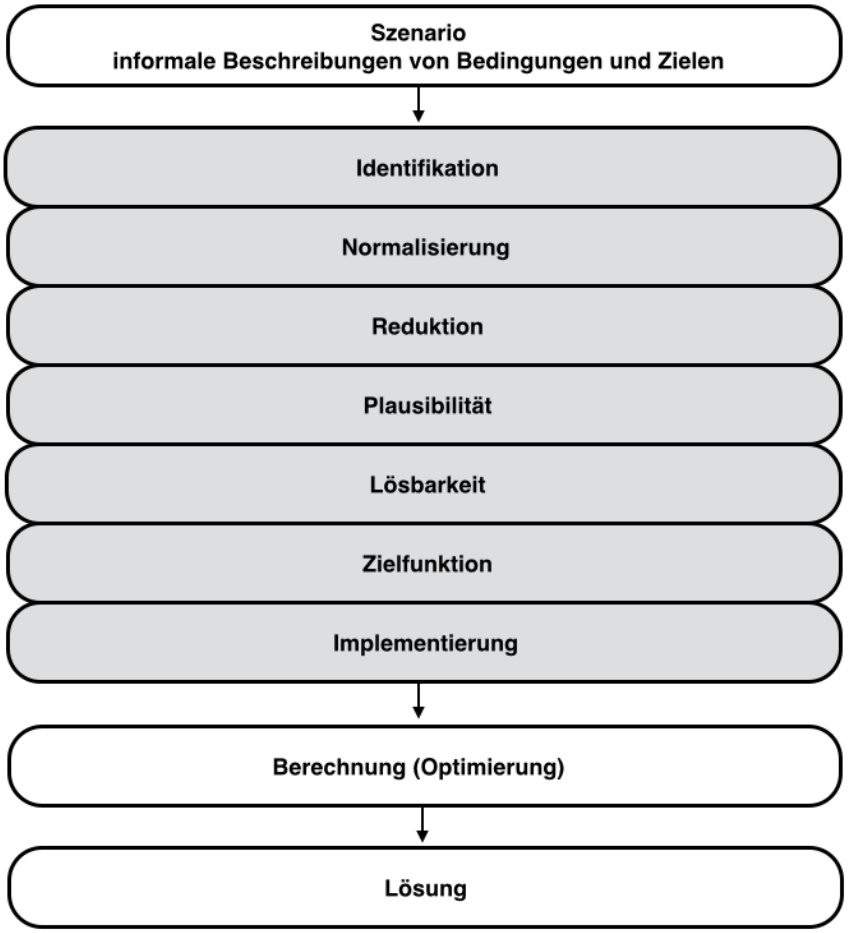


Abbildung 4.3: Transformationsphasen

Die Parameter des Szenarios sind als diejenigen skalaren Werte identifizierbar, die unabhängig variierbar sind und durch ihre Variation eine Annäherung an die gesuchte Lösung erwarten lassen. Wie bereits zuvor dargestellt, ist die Lösung der Aufgabenstellung eine Kombination von Parameterwerten, die eine gültige und bestmögliche Wertekombination darstellen. Formal werden die Parameter in einer Parameterliste verwaltet.

Eine geeignete Anfangsbelegung für die Parameter lässt sich oftmals aus dem Szenario herleiten und als Startwerte in die Parameterliste eintragen. Im Zweifelsfall müssen Startwerte geeignet ermittelt und gesetzt werden; mögliche Ansätze sind eine Trivialbelegung (0-Werte), Zufallswerte oder eine gute Abschätzung aus dem Wertebereich der Beschränkungen.

Die in dem Szenario identifizierbaren Bedingungen werden danach unterschieden, ob sie die möglichen Parameterwert-Kombinationen auf zulässige („gültige“) Wertekombinationen beschränken oder einen restringierenden Bei-

trag zur Bewertung einer gültigen Wertekombination im Sinne von „erwünscht“ oder „zu vermeiden“ liefern. Der Unterschied zwischen „Beschränkung“ und „Restriktion“ ist also die Relevanz: Beschränkungen sind Muss-Bedingungen (Zulässigkeitsbedingungen), Restriktionen sind mit einem Faktor beaufschlagte Wunsch-Bedingungen (s. Abschnit 3.3.2). Der Faktor einer Restriktion geht als Gewichtung des Verletzungsgrades der Restriktion im Sinne einer Straffunktion in die Bewertungsfunktion ein (s. Abschnitt 3.4.2).

Das Ziel der Aufgabenstellung steht in der Identifikationsphase noch nicht im Fokus des Interesses, es wird erst in der Zielfunktion-Phase konkret formuliert. Dennoch muss in der Identifikationsphase bereits ein Verständnis für das Ziel vorhanden sein, damit die Parameter und Bedingungen angemessen erkannt werden können.

Parameter

$$\overline{p} = [p_i] \tag{4.1}$$

mit $i \in \{1 \dots i_{max}\}$ und i_{max} = Anzahl unabhängiger Parameter

Beschränkungen

$$\begin{aligned} \overline{b} &= [b_j] \\ b_j &= f_{b_j}(\overline{p}) \end{aligned} \tag{4.2}$$

mit $j \in \{1 \dots j_{max}\}$ und j_{max} = Anzahl Beschränkungen

Restriktionen

$$\begin{aligned} \overline{r} &= [r_k] \\ r_k &= f_{r_k}(\overline{p}, \text{penalty}_k) \end{aligned} \tag{4.3}$$

mit $k \in \{1 \dots k_{max}\}$ und k_{max} = Anzahl Restriktionen

4.4.2 Normalisierung

Die variierbaren Parameter, die in der Identifikationsphase erkannt worden sind, werden in der eigentlichen Lösungsberechnung im Sinne eines Parameteroptimierungsverfahrens einer schrittweisen Variation unterworfen. Es ist

nun sicherzustellen, dass in jedem Variationsschritt die Variation auf alle Parameter gleich einwirkt, dass also durch die Variation der Parameterwerte kein Parameter bevorzugt oder benachteiligt wird. Dies entspricht einer Unterscheidung der Realwelt-Interpretation der Parameter – auch als Phänotyp (s. Abschnitt 2.2.2) bezeichnet – und ihrer verfahrenstechnischen Interpretation – analog als Genotyp bezeichnet.

Wie in Kap. 2 dargestellt, arbeiten nichtlineare Parameteroptimierungsverfahren im Allgemeinen mit kontinuierlichen Parametern. Die Parameterwerte repräsentieren also jeweils dimensionslose reelle Zahlen auf einem linearen Wertekontinuum. Sie entstammen aber entsprechend dem jeweiligen Szenario unterschiedlichen Wertedimensionen und haben unterschiedliche Definitionsbereiche. Die – ebenfalls schon in der Identifikationsphase erkannten – Beschränkungen erlauben eine Abschätzung der Definitionsbereiche für jeden der auftretenden Parameter.

Das Zwei-Parameter-Problem in Abb. 4.4 zeigt das durch Beschränkungen definierte Zielgebiet Z, aus dem die Definitionsbereiche $P_{def,1}$ und $P_{def,2}$ als

$$P_{def,1} := [P_{1A}, P_{1E}] \tag{4.4}$$

$$P_{def,2} := [P_{2A}, P_{2E}] \tag{4.5}$$

abgeschätzt werden können. Die Abschätzung kann anhand der minimalen und maximalen zulässigen Werte jedes Parameters erfolgen. Die Variation der Parameter in einem beliebigen Schritt der Parameteroptimierung erfolgt zunächst innerhalb eines dimensionslosen Variationsradius R_V. Die unterschiedliche Größe der Definitionsbereiche erfordert die Skalierung der Variationsbreite zu $R_{V,N}$, um der Anisotropie der Parameter P_1 und P_2 gerecht zu werden. Aus der abgeschätzten Größe des zu einem Parameter P_i gehörenden Definitionsbereichs kann ein Normierungsfaktor $F_{N,i}$ abgeleitet werden. Dieser ist zusammen mit dem Parameter zu speichern und stellt die Übergangsbedingung vom phänotypischen zum genotypischen Wert des Parameters dar. Der Normierungsfaktor wird entweder in jedem Variationsschritt der Optimierung bei der Wertevariation des Parameters herangezogen oder die Variation erfolgt ausschließlich auf dem genotypischen Parameterwert; im letzteren Fall findet der Normierungsfaktor nur für die Übergänge Phänotyp $\rightarrow$ Genotyp und der Rücktransformation Genotyp $\rightarrow$ Phänotyp Anwendung.

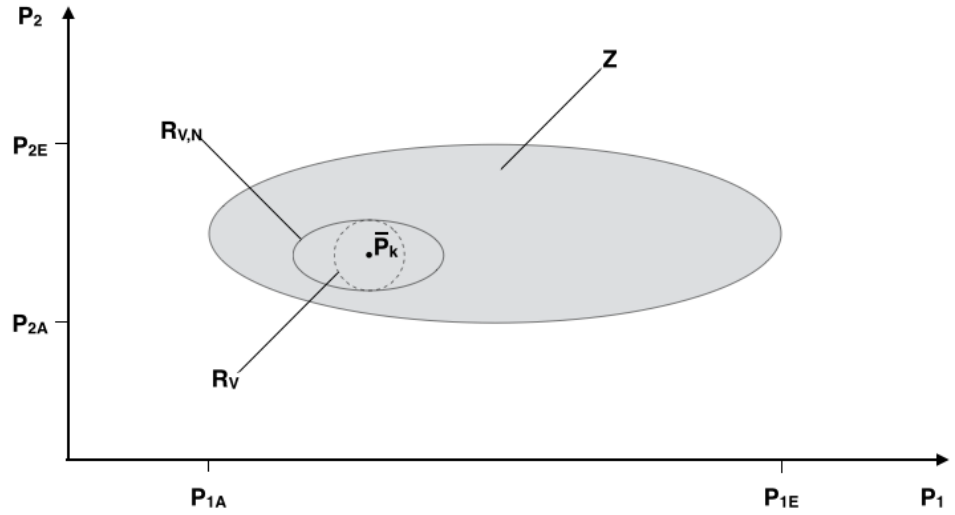


Abbildung 4.4: Anisotropes Zwei-Parameter-Problem

In einem beliebigen Variationsschritt V_s (4.6) der Optimierung wird jeder Parameterwert P_i in einen variierten Parameterwert P_i' überführt:

$$V_s : P_i \rightarrow P_i' \tag{4.6}$$

Dabei wird die aktuelle Schrittweite W_S angewendet. Der tatsächliche variierte Parameterwert P_i' (4.7) bestimmt sich also unter Anwendung des zugeordneten Normierungsfaktors $F_{N,i}$:

$$P_i' := P_i \cdot W_S \cdot F_{N,i} \tag{4.7}$$

Die Liste der Parameter wird somit begleitet und ergänzt von einer Liste der zugeordneten Normierungsfaktoren (4.8):

$$\overline{F}_N = [F_{N,i}] \tag{4.8}$$

mit $i \in \{1 \ldots i_{max}\}$ und i_{max} = Anzahl unabhängiger Parameter

Uneinheitliche Wertebasis

Ein weiterer Aspekt der Normalisierung kann in der uneinheitlichen Wertebasis von Bedingungen oder Parametern gesehen werden (s. Abschnitt 4.3.5). Parameter, die derselben Maßdimension entstammen, können vorteilhaft auf eine einheitliche Wertebasis zurückgeführt werden, um die Vergleichbarkeit innerhalb der Zielfunktionsauswertung zu erleichtern. Ein Beispiel wäre die

Zurückführung von Längenangaben auf eine Basiseinheit wie [m] oder [mm]. Diese Umrechnung ist aber im Grunde trivial und offensichtlich und bedarf im Rahmen der Normalisierungsphase keiner gesonderten Berücksichtigung, solange Fragen der Rechengenauigkeit [KKL+93] keine spezielle Rolle spielen. In der Formulierung der Zielfunktion ist ggf. auf die uneinheitlichen Maßeinheiten zu achten.

4.4.3 Reduktion

Innerhalb der Reduktionsphase werden Bedingungen und Parameter auf *Redundanz* und *Irrelevanz* untersucht.

Technische *Redundanz* „ist das zusätzliche Vorhandensein funktional gleicher oder vergleichbarer Ressourcen [...], wenn diese bei einem störungsfreien Betrieb im Normalfall nicht benötigt werden. [...] In der Regel dienen diese zusätzlichen Ressourcen zur Erhöhung der Ausfall-, Funktions- und Betriebssicherheit.“ [LD13]

Die Sprachtheorie versteht *Redundanz* als mehrfache Nennung von Informationen, die für das Verständnis des Gesamtkontexts nicht notwendig sind [BV06]. In Anlehnung an diese Definition sei Redundanz im Kontext der hier behandelten Kriterientransformation die mehrfache Angabe von Bedingungen und Parametern, die für die Erreichung des Lösungsziels nicht notwendig sind.

Irrelevanz im Rahmen der Kriterientransformation ist die erkennbare Bedeutungslosigkeit von Bedingungen und Parametern.

Ziel der Reduktionsphase ist es, Redundanzen und Irrelevanzen solange zu eliminieren, bis ein redundanz- und irrelevanzfreier Zustand erreicht ist. Die Eliminierung erfolgt durch Streichen aller Parameter und Bedingungen (Beschränkungen und Restriktionen), deren Irrelevanz oder Redundanz erkennbar ist. In der Identifikationsphase ist sichergestellt worden, dass Bedingungen sich auf durch das Szenario erkannte oder durch die Bedingung selbst erkannte Parameter beziehen, sodass Parameter, die nicht in einer Bedingung behandelt werden, nicht vorkommen.

Für die Reduktionsphase eignet sich ein Drei-Schritte-Reduktions-Modell für die Reduzierung der Parameter, Beschränkungen und Restriktionen. Das Modell beginnt mit der Untersuchung von Reduktion auf Parameter; falls hier Redundanz festgestellt wird, kann dies Einfluss auf die Beschränkungen und

Restriktionen haben.

Parameter-Reduktion

Die Untersuchung auf Reduktion der Variationsparameter begrenzt sich auf die Analyse von Redundanz, denn irrelevante Parameter schließt bereits die Identifikationsphase aus: Sobald ein Parameter in einer Bedingung oder einer Zielbeschreibung Erwähnung findet, handelt es sich um einen relevanten Parameter; insbesondere hängt die Zielfunktion

$$f_Z = f(\overline{p}) \tag{4.9}$$

von den Parametern ab. Ein Parameter, der in der Zielfunktion und damit auch in der Zielbeschreibung nicht vorkommt bzw. diese nicht beeinflusst, ist kein systembestimmender Parameter. Dies sollte in der Identifikationsphase bereits erkannt worden sein.

Die Untersuchung auf Parameter-Redundanz sucht nach Parametern, die semantisch für ein Gleiches stehen, die also dieselbe Variations-Dimension repräsentieren. Im Szenario können diese Parameter durchaus unterschiedlich benannt sein (Beispiel: „Abstand“/„Strecke“/„Differenz“). Sie können sogar linear abhängige Konzepte repräsentieren (Beispiel: „Radius“/„Durchmesser“), was ihr Erkennen erschwert.

Nach der Reduktion liegt die Menge der Parameter fest und wird sich in der bearbeiteten Aufgabenstellung nicht mehr ändern.

Beschränkungs-Reduktion

Die Reduktion der Beschränkungen erfolgt zunächst durch Untersuchung auf Redundanz. Zwei Beschränkungen $b_M \in \overline{b}$ und $b_N \in \overline{b}$ sind zueinander redundant, wenn eine (irgendeine) der beiden Beschränkungen b_I entfernt werden kann, ohne dass sich die Größe des Zielgebiets Ω verändert. Dies ist gleichbedeutend mit der Beobachtung, dass die zwei Beschränkungen die Parameter in der gleichen Art und Weise beschränken. Eine der beiden Beschränkungen kann aus der Liste der Beschränkungen entfernt werden, wobei es egal ist, für welche der beiden man sich entscheidet.

$$\Omega = \Omega(\overline{b}) = \Omega(\overline{b} \setminus b_I) \Rightarrow b_I \; ist \; redundant \tag{4.10}$$

Eine Beschränkung ist irrelevant, wenn sie entfernt werden kann, ohne dass dadurch die Größe des Zielgebiets verändert wird. Dies liegt offensichtlich dann vor, wenn eine Beschränkung $b_{irr} \in \overline{b}$ vollständig von einer anderen Beschränkung b^* abgedeckt wird (4.11) oder anderweitig keinen Einfluss auf die Beschränkung von Parameterkombinationen hat. Eine als irrelevant erkannte Beschränkung (4.12) kann aus dem System gestrichen werden.

$$b_{irr} \subset b^* \tag{4.11}$$

$$\Omega = \Omega(\overline{b}) = \Omega(\overline{b} \setminus b_{irr}) \tag{4.12}$$

Restriktions-Reduktion

Das Prinzip der Eliminierbarkeit kann auf Restriktionen nicht in derselben Art und Weise angewendet werden wie auf Parameter und Beschränkungen. Einerseits wären Eliminierbarkeitskriterien nicht systematisch, sondern höchstens in ausgewählten Einzelfällen identifizierbar, andererseits wirken die Restriktions-Straffunktionen immer kumulativ auf den Wert der Zielfunktion.

	Redundanz	Irrelevanz
Parameter	•	
Beschränkung	•	•
Restriktion	○	

Tabelle 4.1: Eliminationsauswirkungen

Auch die reine Irrelevanz von Restriktionen bedarf hier keiner weiteren Betrachtung, denn irrelevante Restriktionen sind außerhalb des zulässigen Zielgebiets angesiedelt und werden bei der Lösungsberechnung ohnehin nicht mitberechnet. Ihre Nicht-Elimination ist also auch performanztechnisch nicht nachteilig. Dabei ist zu beachten, dass das Eliminieren von Redundanzen keine Auswirkungen auf die Lösung aufweist (vgl. Tab. 4.1), jedoch die Menge von Bedingungen und Parametern auf eine notwendige Menge reduziert.

Demgegenüber ermöglicht die Eliminierung (4.13) von Irrelevanzen gegebenfalls erst die Lösbarkeit, da irrelevante Parameter und Bedingungen ein nicht lösbares System erzeugen können:

$$\overline{p} \rightarrow \overline{p}_{red} \tag{4.13}$$

mit $|\overline{p}_{red}| \leq |\overline{p}|$.

4.4.4 Plausibilität

Die nun vorliegenden normalisierten und reduzierten Bedingungen werden in der Plausibilitäts-Phase auf Logik und Verständnis untersucht. Ziel dieser Phase ist die Sicherstellung von plausiblen Bedingungen für die nachfolgenden Phasen. Wie bereits in Abschnitt 4.3 dargestellt, können Bedingungen auf unterschiedliche Weise als nicht-plausibel eingestuft werden. In einem solchen Fall muss entschieden werden, ob die Bedingung nachgebessert werden muss oder ignoriert werden kann. Folgende nicht-plausible Fälle von Bedingungen sind möglich:

Fehlformulierung – *Die Bedingung wurde vom Initiator falsch formuliert.*
: Die Fehlformulierung kann nicht von dem Bearbeiter aufgelöst werden, sie muss an den Initiator zwecks Nachbesserung und Konkretisierung zurückgegeben werden.

Fehlinterpretation – *Die Bedingung wurde vom Bearbeiter falsch verstanden.*
: Eine Fehlinterpretation unterliegt der Prämisse, dass sie auch bemerkt wird. Die Plausibilitätsphase prüft Bedingungen innerhalb der Logik- und Verständnisanalyse ebenfalls auf einen vorhandenden Kontext. Wird festgestellt, dass die Bedingung hier falsch verstanden wurde, muss ein Vermerk zu dieser Bedingung an das Szenario angefügt werden.

Unschärfe – *Bedingungen weisen eine irgendwie geartete Unschärfe auf.*
: Bedingungen, die als unscharf identifiziert worden sind, müssen durch den Bearbeiter geprüft werden und unter einem Vermerk der Bedingung an das Szenario dem Initiator zur Prüfung und Behebung der Unschärfe vorgelegt werden.

Unter- und Überbestimmtheit – *Bedingungen sind widersprüchlich.*
: Bedingungen, die zu sich selbst im Widerspruch stehen – sei es durch Unter- oder Überbestimmtheit – müssen an den Initiator zurückgegeben werden. Er ist in der Lage, die Bedingungen, die bereits reduziert worden sind, formal richtig zu definieren.

Sobald mindestens einer der oben aufgezählten Fälle eintritt, muss die Transformation mit der Identifikationsphase erneut gestartet werden. Wird

die Plausibilitätsphase ohne einen erkennbaren Fall durchlaufen, bildet die Sicherstellung der Plausibilität für alle Bedingungen die Grundlage für die nachfolgende Lösbarkeitsphase.

4.4.5 Lösbarkeit

Lösbarkeit der Aufgabenstellung ist gegeben, wenn mindestens eine Kombination von Parameterwerten existiert, die keine der Beschränkungen verletzt. Dies ist äquivalent zu der Aussage, dass das Zielgebiet unter Einwirkung aller Beschränkungen nicht verschwindet. Dies ist wiederum gegeben, wenn es keinen Parameter gibt, dessen Definitionsbereich unter Einwirkung aller Beschränkungen verschwindet.

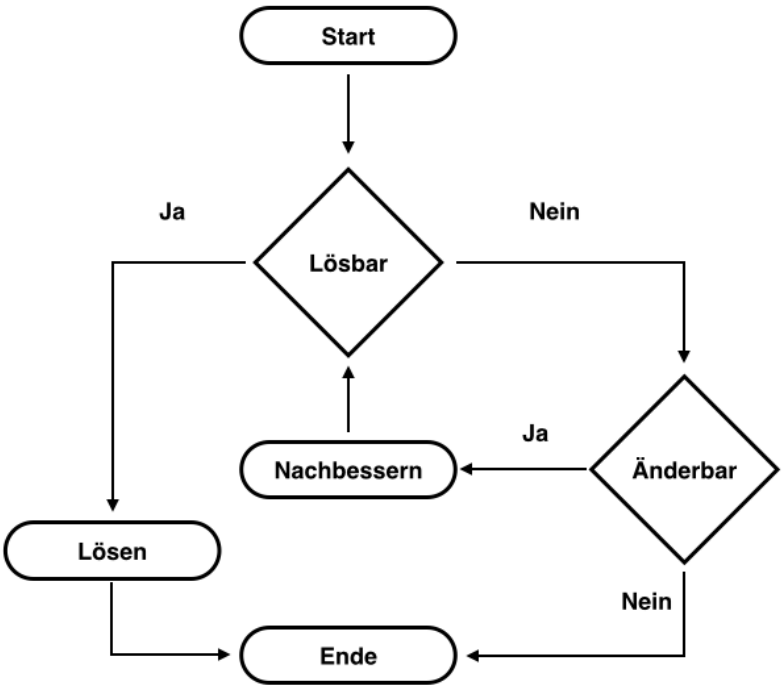


Abbildung 4.5: Abstrakte Lösungsbestimmung

Daraus folgt, dass mindestens eine Lösung existiert, wenn das Zielgebiet nicht verschwindet. Die Prüfung darauf kann dadurch erfolgen, dass für jeden einzelnen Parameter nach Anwendung aller Beschränkungen sein Definitionsbereich nicht leer wird. Wenn unter den gegebenen Bedingungen keine Lösung existiert, endet die Bearbeitung der Aufgabenstellung. Wenn aber die Aufgabenstellung als prinzipiell lösbar angesehen wird, ist die Nicht-Lösbarkeit ein Hinweis auf eine fehlerhafte Formulierung des Szenarios (Abb. 4.5).

Die Existenz einer Lösung ist noch keine Aussage über die Qualität der Lösung. Die Bewertung der Qualität einer Lösung wird mithilfe der Zielfunktion bestimmt.

4.4.6 Zielfunktion

Nachdem die vorherigen Phasen erfolgreich durchlaufen worden sind, liegt ein lösbares System mit normalisierten, reduzierten und plausiblen Parametern und Bedingungen vor. Die inhärente Aufgabe der Zielfunktionsphase liegt in der Definierung der Zielfunktion aus den gegebenen Parametern und Bedingungen und erfolgt am Ziel orientiert im Kontext der Aufgabenstellung.

Wie in Kap. 2 beschrieben, ist die Zielfunktion die Bewertungsfunktion, die die Güte der momentan erreichten Parameterwertkombination (4.14) berechnet. Eingangswerte sind die Parameter mit ihren aktuellen Werten:

$$f_Z = f(\overline{p}) \tag{4.14}$$

Die Zielfunktion orientiert sich in ihrer Formulierung also am primären Ziel der Aufgabenstellung. Es werden zwei Ausprägungen von Zielfunktionen unterschieden:

Explizite Zielfunktion

Eine explizite Formulierung der Zielfunktion liegt vor, wenn die Funktion $f(\overline{p})$ in einer geschlossenen mathematischen Form erfolgen kann. Dann liegt eine Funktion in klassischer Art vor. Dies ist insbesondere von Bedeutung, wenn für die Anwendung von Gradientenverfahren eine stetige Differenzierbarkeit der Zielfunktion gefordert ist.

Implizite Zielfunktion

Eine implizite Zielfunktion basiert demgegenüber auf arbiträren algorithmischen Kalkulationen komplexerer Art, die jede Form der Gütewertberechnung (Fallunterscheidungen, Datenbankabfragen, Anstoßen von Finite-Elemente-Berechnungen u. ä.) beinhalten kann. Eine stetige Differenzierbarkeit ist hier nicht mehr erreichbar, sodass Gradientenverfahren in diesen Fällen nicht eingesetzt werden können.

Der tatsächliche Wert (4.15) der Zielfunktionsauswertung bestimmt sich aus dem eigentlichen Gütewert und der Superposition der aus den Restriktionen resultierenden Straffunktionswerte:

$$f_{Z,res} = f(\overline{p}) + \sum (F_{r_k} \cdot f_{r_k}(\overline{p})) \tag{4.15}$$

4.4.7 Implementierung

Die Implementierungsphase steht für den Übertrag der in den vorher durchlaufenen Phasen gewonnenen, separierten Kriterien in für das Lösungsverfahren nutzbare Strukturen.

Die erste zu implementierende Struktur ist die Liste der Parameter. Die Menge der Parameter liegt nach Durchlaufen der zurückliegenden Phasen fest. Insbesondere die Phasen Identifikation und Reduktion ergeben die Menge der tatsächlich anzuwendenden Parameter. Jeder Parameter repräsentiert einen skalaren Wert aus $\mathbb{R}$. Sinnvollerweise werden die Parameter für die Berechnung in einer listenartigen Datenstruktur abgelegt. Hierbei ist zu beachten, dass die Menge der Parameter fest ist, sodass für die listenartige Datenstruktur eine Implementierung gewählt werden sollte, die für eine feste Anzahl der Elemente geeignet ist (Vektor $\overline{p}$ in Abb. 4.6). Als Datentyp für die Elemente ist ein geeigneter Fließkomma-Zahlentyp zu wählen. Aus Gründen der Performanz und der numerischen Stabilität bietet es sich an, einen maschinennahen Typ mit hoher Genauigkeit zu wählen, z. B. den Typ *double* in Java und C.

Aus Gründen der Praktikabilität und der Berechenbarkeit liegen innerhalb der Datenstruktur die Parameter an konstanten Stellen, d. h. die Reihenfolge der Parameter in der Listenstruktur ist stabil: Obwohl die eigentliche Optimierungsberechnung auf semantikfreien Zahlenwerten basiert, ist sowohl für die Berechnung aller Zwischenschritte als auch für die spätere Reinterpretation der erhaltenen Ergebnisse im Rahmen der Rücktransformation die Zuordnung je einer Semantik für jeden Parameter erforderlich. Spätestens für die Interpretation der Ergebnisse wird das offensichtlich: Ohne Semantik kann aus dem berechneten Zahlenwert eines Parameters keine sinnvolle Schlussfolgerung für die Aufgabenstellung gezogen werden. Im Sinne der möglichen Reinterpretation können die Semantiken der Parameter in einem *Semantikvektor* $\overline{s}$ (Abb. 4.6) abgelegt werden, typischerweise in einem Vektor für Zeichenketten (*String*).

Die Elemente von $\overline{s}$ beziehen sich jeweils indexgenau auf die zugehörigen Parameterwerte des Vektors $\overline{p}$, um den semantischen Bezug zu gewährleisten; $\overline{s}$ hat daher auch dieselbe Dimension n wie $\overline{p}$. Es erfolgt also eine strikte Trennung der identifizierten Parameter nach den eigentlichen skalaren Parameterwerten in einen Parametervektor und der zugeordneten Semantik in einen Semantikvektor.

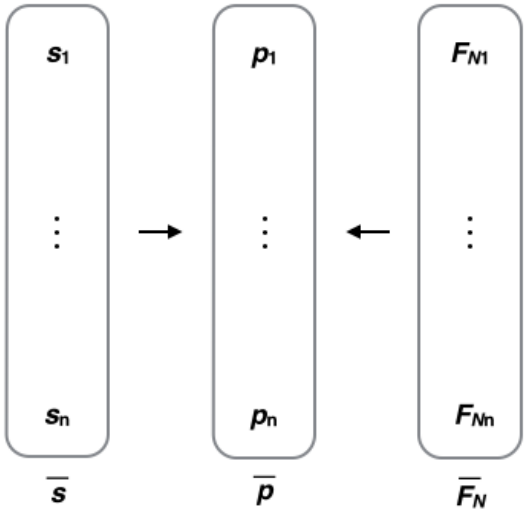


Abbildung 4.6: Parametervektor mit Semantiken und Normalisierungsvektor

Im Anschluss an die Phase der Identifikation der Parameter wurden innerhalb der Normalisierungsphase die Parameter für die isotrope Variation mit Normalisierungsfaktoren versehen. Daraus ergibt sich für alle Parameter ein zugeordneter Normalisierungsfaktor F_{N_i} von einem Fließkommatyp (*double*). Auch diese F_{N_i} stehen in einem stabilen direkten Bezug zu ihrem Parameter. Die Normalisierungsfaktoren ($\overline{F}_N$ in Abb. 4.6) werden ebenfalls in einem Vektor der Dimension n abgelegt. Die Elemente von $\overline{F}_N$ stehen genauso in direktem Bezug zum Parametervektor wie die Elemente des Semantikvektors.

Die zweite zu implementierende Struktur ist die der Beschränkungen. Auch hier lässt sich, ähnlich wie schon bei den Parametern, als Resultat der vorherigen Phasen eine konkrete Anzahl an Beschränkungen identifizieren. Im Gegensatz zu den Parametern ist eine Beschränkung allerdings nicht durch einen skalaren Wert repräsentiert, sondern stellt eine Grenzwertüberprüfung dar, sodass jede Beschränkung in Form von algorithmischem Quellcode umzusetzen ist. Dies läuft üblicherweise auf die Auswertung von Vergleichsoperatoren hinaus und ergibt einen Boole'schen Ergebniswert.

Eine Beschränkung ist formal eine Funktion der aktuellen Parameter:

$$f_b = f(\overline{p}) \tag{4.16}$$

Jede eine Beschränkung repräsentierende Beschränkungsfunktion f_{b_j} (4.16) liefert einen Boole'schen Rückgabewert, der die Zulässigkeit der aktuellen Parameterwertkombination unter den Bedingungen der jeweiligen Beschränkung darstellt. Eine gültige Parameterkombination setzt voraus, dass alle Beschränkungen erfüllt sind. Es müssen also in jedem Lösungsschritt alle Beschränkungen durchlaufen werden und ein positives Funktionsergebnis (true) ergeben. Sobald eine einzige Beschränkung ein negatives Ergebnis (false) liefert, ist die aktuelle Parameterkombination ungültig und muss verworfen werden.

Die Reihenfolge, in der man die Beschränkungen durchläuft, ist prinzipiell zunächst ohne Bedeutung. Unter dem Aspekt, dass jedoch eine erste auftauchende Beschränkung die aktuelle Parameterkombination als ungültig markiert und dadurch weitere Beschränkungen sofort obsolet werden, bietet es sich an, Beschränkungen, die eine hohe oder höhere Wahrscheinlichkeit auf Nicht-Erfüllung – negatives Ergebnis – haben, bevorzugt (also in der Reihenfolge früher) auszuführen. Diese Vorgehensweise lässt eine allgemeine, aber nicht quantifizierbare Performanzsteigerung – ausschließlich beim Auffinden und Identifizieren von ungültigen Parameterkombinationen – erwarten, ist aber für das Ergebnis des Lösungsverfahrens letztlich nicht relevant.

Die Zielfunktion stellt die dritte zu implementierende Struktur dar, sie ergibt sich aus dem im Szenario beschriebenen angestrebten Zielzustand und wird in Abhängigkeit von den Parametern formuliert. Daraus lässt sich eine algorithmische, parameterorientierte Zielfunktion definieren. Die Zielfunktion muss so implementiert werden, dass sie einen Gütewert für jede aktuelle Parameterkombination liefert und dabei berücksichtigt, ob das Optimierungsverfahren nach dem Prinzip der Minimumsuche oder der Maximumsuche arbeitet.

Die konkrete programmiertechnische Implementierung der Zielfunktion muss in der Weise erfolgen, dass aus der Liste der aktuellen Parameterwerte ein Gütewert berechnet wird, der als Maß für die Verbesserung oder Verschlechterung der gefundenen Parameterkombination im Rahmen der Optimierung herangezogen werden kann. Sinnvollerweise erfolgt die Implementierung in Form

einer Function (4.17), die als Funktionsparameter ausschließlich die Parameterliste übernimmt und als Rückgabewert einen Fließkommawert (double) liefert.

$$f_Z = f(\overline{p}) \tag{4.17}$$

```
double zielfunktion(double[] params) { ... }
```

Restriktionen sind Bedingungen, die Wunsch- oder Soll-Kriterien im Szenario repräsentieren. Eine Restriktion schließt also eine aktuelle Parameterwertkombination nicht grundsätzlich aus, sondern soll eine ungünstige Parameterkombination schlechter bewerten als eine günstige. Weiterhin wird gefordert, dass eine nicht-erfüllte Restriktion das Optimierungsverfahren in die Richtung erwünschter Kombinationen führt. Insofern stehen die Restriktionen in engem Zusammenhang zur Zielfunktion. Die Realisierung erfolgt geeignet in Form von Straffunktionen, die durch Superposition den zunächst berechneten Wert der Zielfunktion beeinflussen. Ähnlich der Zielfunktion sind Straffunktionen Bewertungsfunktionen f_r (4.18) in Abhängigkeit von den Parametern $\overline{p}$ sowie einem Straffaktor F_{r_k}:

$$f_{r_k} = f(\overline{p}, F_{r_k}) \tag{4.18}$$

```
double straffunktion(double[] params, double factor) { ... }
```

Zu jeder identifizierten Restriktion wird eine Straffunktion aufgestellt. Diese liefert ähnlich wie eine Zielfunktion einen Rückgabewert als Fließkommawert (double), der unerwünschte Parameterkombinationen schlecht bewertet. „Nichtunerwünschte“ Parameterkombinationen werden zu 0 ausgewertet, eine Restriktion kann also den Zielfunktionswert nicht verbessern. Eingangswerte sind die Liste der aktuellen Parameterwerte und ein Straffaktor. Der Straffaktor wird im einfachsten Fall als neutrales Element der Multiplikation vorbelegt (vulgo: 1) und kann im Übrigen für eine Priorisierung oder Gewichtung mehrerer Restriktionen untereinander und gegenüber der Zielfunktion benutzt werden.

Die Berechnung des endgültigen Gütewertes erfolgt durch die Berechnung der Zielfunktion und Superposition der Gesamtheit aller Restriktionen in Abhängigkeit der aktuellen Parameterwertkombination (im Folgenden als resul-

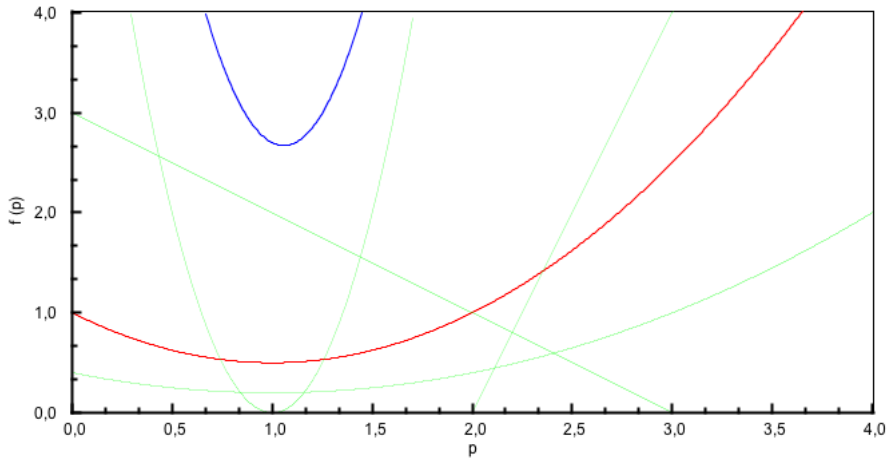


Abbildung 4.7: Beispiel einer resultierenden Zielfunktion (blau) aus Gütewertfunktion (rot) mit vier Restriktionen (grün)

tierende Zielfunktion $f_{Z,res}$ (4.19) bezeichnet):

$$f_{Z,res} = f_Z + \sum(f_r) \tag{4.19}$$

Passend zur Zielfunktion muss eine Abbruchbedingung definiert werden, die das iterative Optimierungsverfahren im geeigneten Moment beendet. Bei Erfüllung der Abbruchbedingung wird der aktuell beste Zielfunktionswert als Gütewert zurückgeliefert; die zu diesem Zeitpunkt erreichte zugehörige Parameterwertkombination wird als Liste der Ergebniswerte ausgegeben. Eine Abbruchbedingung kann in Form eines der folgenden absoluten und relativen Abbruchkriterien oder einer Kombination davon implementiert werden:

Manueller Abbruch

Im trivialen Falle eines manuellen Abbruches beendet der Benutzer aus individuellen Gründen das Verfahren. Ggf. wird der aktuelle Gütewert zurückgeliefert, ist jedoch wenig aussagekräftig.

Definierte Anzahl von Schritten

Das Verfahren bricht nach Erreichen von einer vorher definierten Anzahl von Durchläufen ab und liefert das beste bis dahin erreichte Gütemaß zurück.

Definierte Laufzeit

Für das Verfahren wird ein Zeitintervall angegeben, in welchem solan-

ge Gütewerte berechnet werden, bis die Laufzeitgrenze erreicht ist. Das beste Gütemaß wird nach Ablauf der Zeit zurückgeliefert.

Die genannten absoluten Abbruchkriterien garantieren in jedem Fall die Beendigung des Verfahrens, garantieren aber nicht das Erreichen irgendeines Optimums. Sie werden angewendet, wenn keines der nachfolgenden relativen Abbruchkriterien anwendbar ist.

Definition eines zu erreichenden Zielwertes für den Gütewert
Sobald durch die Berechnung der resultierenden Zielfunktion der vorab definierte Zielwert erreicht oder unterschritten wird, gilt das Verfahren als erfolgreich durchlaufen und wird beendet. Der Zielwert muss vorab bekannt sein.

Erreichen einer vorgegebenen Konvergenz des Gütewerts
Sobald durch die Berechnung Zielfunktionswerte aufeinander folgender Iterationen einen definierten Differenzwert ϵ unterschreiten, ist eine weitere Verbesserung der Lösung nicht mehr zu erwarten und es erfolgt eine Beendigung des Verfahrens. Der erreichte Gütewert wird zurückgeliefert.

Unterschreitung einer minimalen Variationsbreite
Unterschreitet innerhalb des Verfahrens die Variationsbreite einen gewählten Grenzwert, ist eine weitere Verbesserung der Lösung im Rahmen der zugrunde gelegten Genauigkeit nicht mehr zu erwarten und es erfolgt eine Beendigung des Verfahrens. Der erreichte Gütewert wird zurückgeliefert.

Die genannten relativen Abbruchkriterien heben im Gegensatz zu den absoluten Kriterien auf eine tatsächliche Erreichung oder eine hinreichende Annäherung an das gesuchte Optimum ab. Die Problematik der Unterscheidung eines echten Optimums von Nebenminima tritt auf, wird hier aber nicht weiter betrachtet.

Die Implementierung der Abbruchbedingung (4.20) erfordert mindestens ein Abbruchkriterium. Für das generalisierte Optimimierungsverfahren wurde festgelegt, dass ein Rückgabewert *true* als Abbruch interpretiert wird. Eine allgemeine Implementierung einer Abbruchbedingungsfunktion übernimmt als

Funktionsparameter den aktuellen Parametersatz, den aktuellen Zielfunktionswert und Konvergenzinformationen sowie zusätzliche Systemparameter wie Anzahl der durchlaufenen Iterationsschritte oder die Laufzeit:

$$f_A = f(\overline{p}, f_Z, \text{Konvergenzdaten}, \text{Systemdaten}) \tag{4.20}$$

4.5 Berechnung und Lösung

Die Transformation hat den Input für das generalisierte Optimierungsverfahren in geeigneter Weise vorbereitet. Gemäß Kap. 3 ist das generalisierte Verfahren agnostisch gegenüber dem tatsächlich eingesetzten Verfahren. Der Input ist durch die Transformation jedoch hinreichend verallgemeinert und unabhängig von der konkreten Verfahrensauswahl.

Nachdem auch die Abbruchkriterien für das Optimierungsverfahren festliegen, werden die aufgestellten Datenstrukturen dem Optimierungsverfahren zugeführt und der Start des Verfahrens ausgelöst.

Je nach gesetzten Abbruchkriterien wird das Optimierungsverfahren nach endlicher Laufzeit zu einem Ergebnis kommen. Betrachtet man nur den echten Konvergenzfall, also das erkennbare Erreichen eines Minimums, als effektive Lösung, liegen die Lösungswerte numerisch als Endwerte der günstigsten Parameterkombination vor. Zusätzlich wird der damit erreichte Zielfunktionswert ausgeworfen. Weitere Systemparameter wie Anzahl der benötigten Variationsschritte, Konvergenzwerte und Laufzeit können ebenfalls als Resultate anfallen.

Die Lösung ist an dieser Stelle rein numerisch und uninterpretiert im Kontext der Aufgabenstellung. Sie muss daher der Rück-Transformation zugeführt werden.

Sollte eine echte Konvergenz nicht erreicht worden sein, z. B. nach einem manuellen oder laufzeitbedingten Abbruch, liegt keine effektive Lösung vor. In diesem Fall kann man versuchen, die Optimierung mit einer geänderten Startbelegung der Parameterwerte, anderen Kontrollwerten und ggf. geänderten Abbruchbedingungen erneut auszulösen. Auch der Wechsel zu einem anderen Optimierungsverfahren ist hier denkbar.

4.6 Szenario, Transformation und Konzentrat

Die Transformation soll Informationen der Aufgabenstellung in eine Datenstruktur überführen, die für die Lösung mit einem Optimierungsverfahren gemäß Abschnitt 2.2.1 geeignet ist. Dazu werden die Informations-Entitäten aus dem Szenario extrahiert und durch die Transformation in eine optimierungsfähige Datenstruktur überführt.

> *Definition: Konzentrat*
> Die als Ergebnis einer Transformation erhaltene Datenstruktur wird als Konzentrat bezeichnet.

Das Konzentrat wird dem Optimierungsverfahren übergeben. Das vom Optimierungsverfahren berechnete Ergebnis hat dieselbe Struktur wie das Konzentrat, enthält aber erwartbar unterschiedliche numerische Werte, insbesondere für die Parameter.

> *Definition: Strukturelle Kongruenz*
> Zwei Konzentrate sind strukturell kongruent, wenn ihre Datenstrukturen identisch sind. Numerische Unterschiede von Datenwerten stehen der strukturellen Kongruenz nicht entgegen.

Da die Eingangs- und Ausgangs-Datenstrukturen einer Optimierung in diesem Sinne strukturell kongruent sind, kann das Ergebnis-Konzentrat einer Optimierungsrechnung wieder als Eingangs-Konzentrat einer weiteren Optimierung (mit demselben oder einem anderen Optimierungsalgorithmus) eingesetzt werden (Abb. 4.8). Die strukturelle Kongruenz der Konzentrate vor und nach der Optimierung kann vorteilhaft für die Validierung der Transformation genutzt werden (s. Kap. 5).

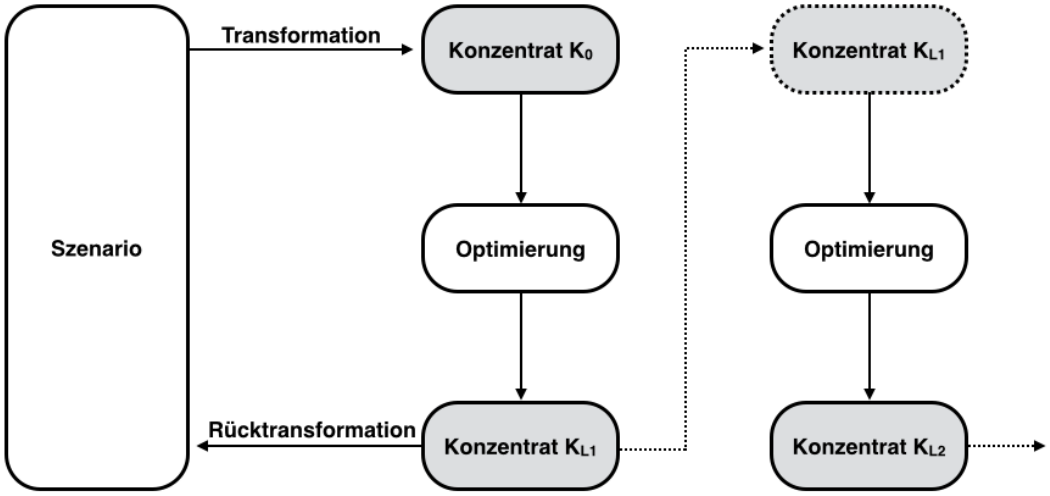


Abbildung 4.8: Konzentrat als Input und Output einer Optimierung

4.7 Rück-Transformation der numerischen Ergebnisse

Um dem Szenario der Aufgabenstellung und dem Initiator eine adäquate semantikbezogene Lösung zu präsentieren, müssen die numerischen Ergebnisse gemäß Abb. 4.3 rücktransformiert werden. Die numerische Lösung liegt in Form von Daten als Parameterkombination vor und muss zur Überbrückung der interpretatorischen Lücke (Abschnitt 4.1.3) auf die Informationssemantik der Aufgabenstellung rekonstruiert, d.h. rücktransformiert werden. Sie ist mit einem Normalisierungs- und einem Semantikvektor bijektiv verknüpft (Abb. 4.6).

Die Rück-Transformation besteht aus einem Zwei-Phasen-Modell: zunächst wird die Normalisierung aus der Normalisierungsphase rückgängig gemacht, bevor schließlich die Parameterwerte wieder mit ihren Semantiken angereichert werden.

4.7.1 Umkehrung der Normalisierung

Um die Lösung in die in der Aufgabenstellung beschriebenen Wertebereiche zu überführen, erfolgt in dieser Phase die Umkehrung der Normalisierung aller Parameterwerte.

Für die Umkehrung wird der in der Normalisierungsphase (4.4.2) erzeugte

Vektor auf die Lösung invers angewendet:

$$\overline{p}_{DN} = \frac{\overline{p}}{\overline{F}_N} \tag{4.21}$$

Die Parameterkombination liegt nach der Umkehrung in denormalisierter Form (4.21) vor, die Parameter repräsentieren nun Werte aus ihrem jeweiligen realweltlichen Wertebereich.

4.7.2 Semantikanreicherung

Aus dem berechneten Zahlenwert eines Parameters kann mithilfe des zugeordneten Semantikwertes eine sinnvolle Schlussfolgerung für die Aufgabenstellung hergeleitet werden. Gemäß Abb. 4.6 wurden jedem Parameter für die Reinterpretation die Semantiken der Parameter in einem Semantikvektor $\overline{s}$ (Abb. 4.6) als beschreibende Zeichenkette zugewiesen.

Praktisch bedeutet dies, dass die denormalisierten Parameterwerte zusammen mit ihrem Beschreibungstext in für den Initiator geeigneter Weise ausgegeben werden. Dies kann im Rahmen eines konfigurierbaren Reportings geschehen.

4.8 Anforderungen an eine Strukturierung der Semantik

Aus den dargestellten Einzelaspekten kann man die Anforderungen, die eine Strukturierung der Semantik erfüllen muss, herleiten.

Die oberste Maßgabe ist immer das Ziel, die Ergebnisse der Optimierung im Sinne der ursprünglichen Aufgabenstellung rekonstruieren zu können. Es ist mit der Strukturierung also die Frage zu beantworten, welche semantischen Informationen strukturiert aufbewahrt, also persistiert werden müssen, um eine vollständige Rekonstruktion der Kriterien der Aufgabenstellung herbeiführen zu können. Abschnitt 6.5 wird dazu typische Rückführungsszenarien aufstellen und die daraus resultierenden Rekonstruktionsanforderungen konkretisieren.

Klar ist, dass eine unvollständig erfasste Semantik zwangsläufig eine nur unvollständige Ergebnisinterpretation nach sich zieht. Die aufwendig in der Optimierung berechneten Ergebnisse lassen sich dann nicht mehr angemessen

als Ergebnisse im Sinne der Aufgabenstellung nutzen. Im schlimmsten Fall hat man viele Zahlenwerte, aber keine tatsächliche Lösung im Sinne der Aufgabenstellung.

Kapitel 5

Transformationsvalidierung

Weil eine Kriterientransformation über mehrere Phasen nur dann eine sinnvolle Optimierungsrechnung ergeben kann, wenn sie richtig und der Aufgabenstellung angemessen ausgeführt wurde, werden für die Gültigkeitsüberprüfung des Transformationsergebnisses Aspekte für eine Validierung betrachtet.

Nachdem in Kap. 4 ein vollständiges Phasenmodell der Transformation von Szenario-Beschreibungen eingeführt wurde, sollen nun Kriterien für die Überprüfung einer Transformation auf Validität aufgestellt und diskutiert werden. Eine mögliche Prüfung besteht darin, das durch die Transformation entstandene Konzentrat mit der ursprünglichen Szenario-Beschreibung im Sinne der strukturellen Kongruenz gemäß Abschnitt 4.6 zu vergleichen. Dafür wird auf Methoden der algebraischen Mengenabbildung zurückgegriffen. Eine zweite Prüfung versucht, die Validierung auf die Semantik zu erweitern, indem eine semantische Rücktransformation vom Konzentrat auf das Szenario versucht wird.

Für die Validierung werden die formalen Benennungen S für das Szenario und K für das Konzentrat eingeführt.

5.1 Validierung der Transformation

Die Transformation

$$S \rightarrow K \tag{5.1}$$

überführt die Informationen des Szenarios S der Aufgabenstellung möglichst informationserhaltend, aber semantikfrei, in das Konzentrat K für die Lösung

durch Optimierung. Das Konzentrat K als Ergebnis der Transformation ist also eine stark strukturierte Beschreibung der im Szenario unscharf beschriebenen Sachverhalte. Bevor man die Optimierung startet, kann es sinnvoll sein, die Inhalte des Konzentrats auf Gültigkeit und Angemessenheit im Sinne der Aufgabenstellung zu überprüfen. Nach der Lösung durch Optimierung wird dann das Konzentrat wieder mit Semantik angereichert und auf die Ebene des Szenarios rück-transformiert.

Die Validierung der Transformation soll also die Zurückführbarkeit der optimierten Ergebnisdaten über die Rück-Transformation $K \rightarrow S$ sicherstellen. Die Informationen des Szenarios sollen daher eindeutig, vollständig und korrekt im Konzentrat präsent sein.

Da die Konzentrat-Strukturen vor und nach der Optimierung, wie in Abschnitt 4.6 dargestellt, kongruent sind, kann man die Validierung so durchführen, dass man die Rück-Transformation versuchsweise bereits vor dem Start der Optimierung, also unmittelbar nach der Transformation, durchführt und die Angemessenheit der Lösung im Sinne der Aufgabenstellung untersucht (Abb. 5.1). Ein weiteres Überprüfungskriterium neben der semantischen Angemessenheit ist, dass das Konzentrat in diesem (prä-optimierten) numerischen Zustand wieder exakt auf das Szenario zurückführen muss.

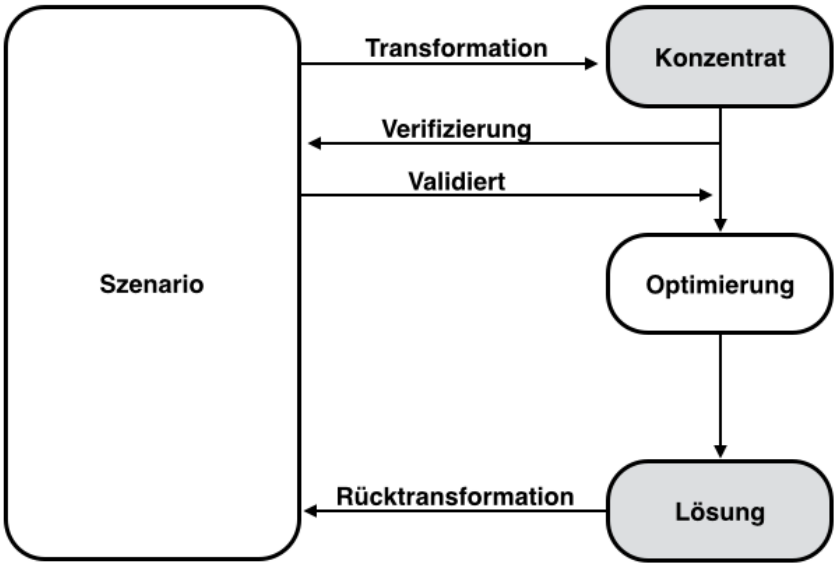


Abbildung 5.1: Validierungsprozess

Im Idealfall kann die Transformation $S \rightarrow K$ als mengentheoretische Abbildung von Szenario-Informationsentitäten auf Konzentrat-Informations-

Entitäten dargestellt werden. Dann kann auch die Rück-Transformation $K \rightarrow S$ als Abbildung von Informations-Entitäten beschrieben werden. In diesem formalen Fall kann man die Validierung durch formale Betrachtung der Abbildungen realisieren.

Im allgemeinen Fall ist jedoch eine solche streng formale Abbildung nicht darstellbar. Dies ist insbesondere in der bereits diskutierten Unschärfe der Informationen im Szenario begründet, die eine Identifikation eindeutiger Informations-Entitäten auf Szenario-Seite unmöglich macht. Es ist ja gerade erst die Identifikationsphase der Transformation, die – zusammen mit den weiteren Transformationsphasen – diese Unschärfe eliminieren soll.

Trotz dieser Problematik im allgemeinen Fall kann man zumindest für Spezialfälle eine formale Validierung anstreben. In allen anderen Fällen ist eine alternative Vorgehensweise unumgänglich. Die beiden Vorgehensweisen sollen daher im Folgenden diskutiert und ihre Anwendbarkeit untersucht werden.

5.1.1 Transformation als formale Abbildung

Die Transformation überführt die Informationen des Szenarios in Datenstrukturen des Konzentrats: $S \rightarrow K$. Unter der Annahme, dass Szenario und Konzentrat als Mengen mit den jeweiligen Informationsentitäten als Elemente dargestellt werden können, ist die Transformation eine Abbildung des Szenarios auf das Konzentrat. Die Rück-Transformation ist dann eine Abbildung des Konzentrats auf das Szenario: $K \rightarrow S$ (Abb. 5.2).

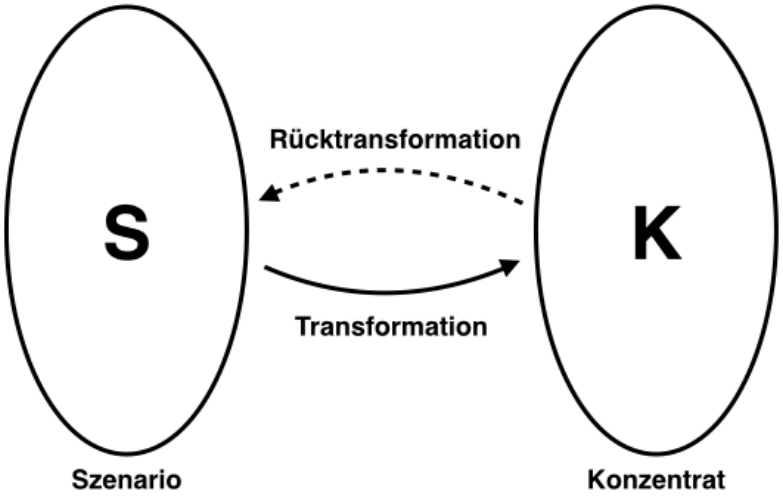


Abbildung 5.2: Abbildung $S \rightarrow K$ und $K \rightarrow S$

Führt man die Abbildungen $S \rightarrow K$ und $K \rightarrow S$ nacheinander aus und betrachtet die elementweisen Kardinalitäten der Abbildungen, kann man daraus auf die quantitative Validität der Transformation rückschließen. Eine qualitative Aussage über die inhaltliche Korrektheit jeder einzelnen elementweisen Abbildung erfolgt dadurch allerdings nicht.

Dabei kann man folgende drei Fälle unterscheiden:

„Beidseitig bijektive“ Abbildung

Jedes Element der Szenariobeschreibung wird eindeutig auf genau ein Element des Konzentrats abgebildet (Abb. 5.3). Diesem Fall liegt ein redundanzfreies Szenario zu Grunde, d. h. nach Durchlaufen der Transformationsphasen wird jede Szenario-Informationsentität auf genau eine Konzentrat-Informationsentität abgebildet. Wenn die Transformation keine Entitäten hinzufügt, ist auch die Rücktransformation bijektiv.

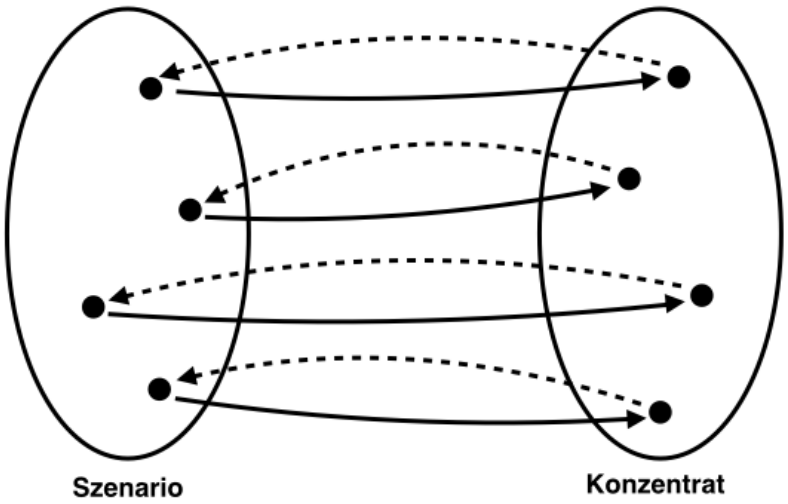


Abbildung 5.3: Transformation als bijektive Abbildung

Wenn $S \rightarrow K$ bijektiv und $K \rightarrow S$ bijektiv, dann besteht eine eineindeutige Beziehung der Informationsentitäten. Daraus folgt zumindest, dass alle Kriterien berücksichtigt wurden und nichts hinzugefügt wurde. Damit ist die quantitative Validität auf der Basis der Kardinalitäten gegeben.

„Surjektiv-injektive“ Abbildung

Eine Szenariobeschreibung, die Redundanzen beinhaltet, kann zwar Teilmengen von Informationsentitäten bijektiv in das Konzentrat abbilden,

aber nicht diejenigen, die redundanzbehaftet vorliegen. Diese redundanzbehafteten Elemente bilden Teilmengen, die surjektiv auf Teilmengen des (prinzipbedingt redundanzfreien) Konzentrats abgebildet werden (Abb. 5.4). Die Rückabbildung vom Konzentrat zum Szenario $K \rightarrow S$ ist dann injektiv.

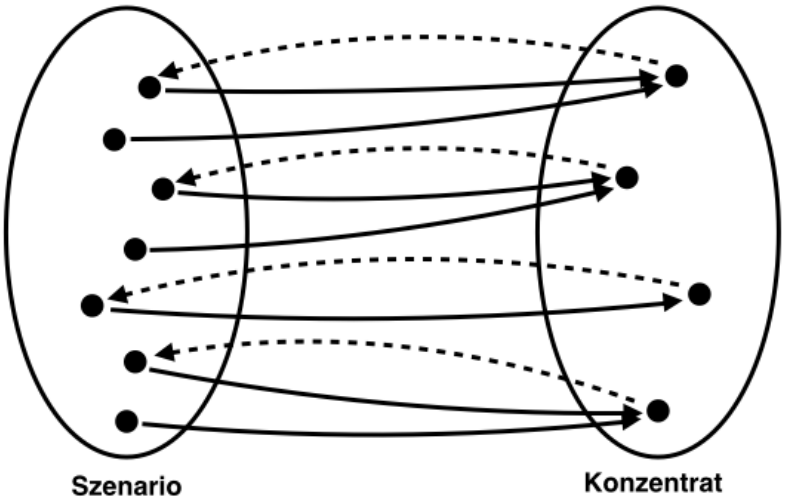


Abbildung 5.4: Surjektiv-injektive Abbildung

Wenn $S \rightarrow K$ surjektiv und $K \rightarrow S$ injektiv, dann besteht keine eineindeutige Beziehung der Informationsentitäten. Daraus folgt, dass alle Kriterien, von denen einige redundant sind, als Konzentrats-Informationsentitäten berücksichtigt wurden und nichts hinzugefügt wurde. Die Rückabbildung geschieht redundanzfrei, also injektiv, auf Szenario-Informationsentitäten. Damit ist die quantitative Validität auf der Basis der Kardinalität des Konzentrats gegeben.

Irreversible Abbildung

Es kann die Situation eintreten, dass im Rahmen der Transformation vom Bearbeiter zusätzliche Entitäten eingefügt werden, die nicht der ursprünglichen Szenariobeschreibung entnommen werden können. Gründe dafür können in numerischen und verfahrenstechnischen Notwendigkeiten liegen, z. B. wenn ein Parameter durch eine Beschränkung auf einen numerischen Bereich eingeschränkt werden muss („nur positive Werte für Abstände“) und dies dem Initiator mangels Verfahrenskenntnis nicht bekannt war. Dann entstehen auf der Seite des Konzentrats neue Infor-

mationsentitäten, die dazu führen, dass die Rücktransformation keine überprüfbare Abbildung mehr darstellt. In diesem irreversiblen Fall ist keine sinnvolle Aussage über die Validität der Transformation anhand der Abbildungskardinalitäten möglich (Abb. 5.5).

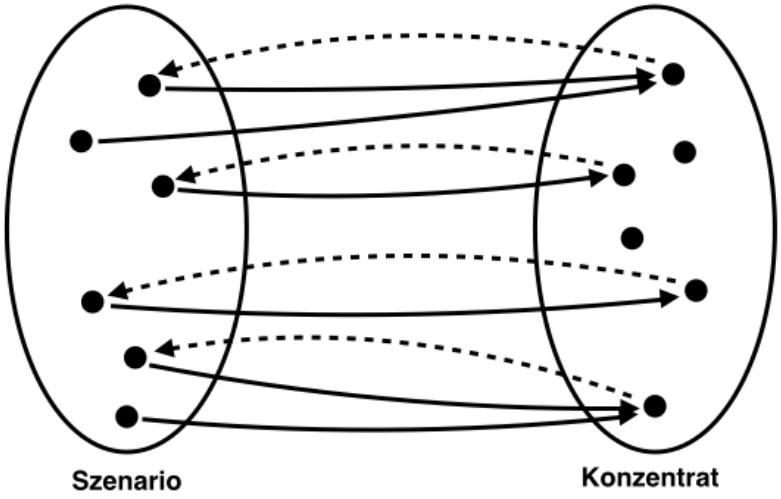


Abbildung 5.5: Irreversible Abbildung

5.1.2 Semantische Rücktransformation

Für den allgemeinen Fall, dass eine streng formale Validierung der Transformation nicht möglich ist, kann die Validität der Transformation durch einen semantischen Abgleich des Transformations-Konzentrats mit dem Szenario abgeschätzt werden. Wie bereits dargestellt, ist das Konzentrat, das nach Durchführung der Transformation entstanden ist, strukturell-kongruent mit dem Ergebnisdatensatz, der nach Durchführung der Optimierung erwartet wird. Man kann daher bereits unmittelbar nach der Transformation das Konzentrat einer Rücktransformation unterziehen und das Ergebnis mit dem Szenario inhaltlich vergleichen (Abb. 5.1). Da das Konzentrat vor der Optimierung zwangsläufig die numerischen Werte des Szenarios widerspiegelt, muss diese Rücktransformation exakt auf das Szenario zurückführen. Man erhält also im validen Fall eine inhaltlich identische Rückabbildung. Sind an dieser Stelle Abweichungen in den Aussagen oder den Zahlenwerten erkennbar, kann daraus gefolgert werden, dass die Transformation Fehler enthält.

Als Nebeneffekt ist dadurch sichergestellt, dass die aus dem Optimierungs-

verfahren erhaltene Lösung ebenfalls valide zurücktransformiert werden kann. Die Verifizierung des Konzentrats vor der Optimierung ist insbesondere dann gewinnbringend, wenn das Optimierungsverfahren oder die Zielfunktion komplex ist und eine lange Laufzeit erwarten lässt.

5.1.3 Isomorphismus als Idealtransformation

Die Idealtransformation basiert auf einer bijektiven Abbildung (Abb. 5.3): Jede Informations-Entität wird per Transformations-Abbildung von dem Szenario eindeutig auf das Konzentrat abgebildet und per Rücktransformations-Abbildung vom Konzentrat eindeutig wieder zurück in das Szenario überführt. Die Bilder der jeweiligen Abbildungen sind informationserhaltend und bilden damit einen Homomorphismus [Fis00, Wal92]. Die Bijektivität der Abbildungen und der Homomorphismus der Bilder führen auf den für die Idealtransformation geforderten Isomorphismus [Fis00].

Eine Transformation gilt als eindeutig, wenn einerseits deterministisch aus einer gegebenen Problembeschreibung mittels mehreren Transformationen die gleichen Konzentrate entstehen und andererseits ein entstandenes Konzentrat über eine Rücktransformation eindeutig in das Szenario überführt werden kann.

5.2 Defekterkennung durch Validierung

Zu identifizierende Defekte in der Transformation können einerseits bereits in der Szenariobeschreibung begründet sein oder andererseits in der Transformation selbst entstehen. Beide Möglichkeiten beinflussen die Validität der Transformation negativ.

Wenn die Szenariobeschreibung den Anforderungen der Transformation gemäß Abschnitt 4.2.2 nicht genügt und ein Defekt zu vermuten ist, wird die Szenariobeschreibung aufgrund von Untersuchungen nach Abschnitt 4.3.2 geprüft und der Defekt, wenn möglich, korrigiert oder im Fall eines nicht korrigierbaren Defektes an den Initiator zur Nachbesserung übergeben. Ist eine Nachbesserung erfolgt, beginnt eine erneute Prüfung der Szenariobeschreibung.

Ist ein Defekt erkennbar, obwohl die Szenariobeschreibung selbst den Anforderungen an die Transformation genügt, muss zwangsläufig der Defekt in

der Transformation lokalisiert sein. Dann ist die Transformation zu korrigieren oder erneut durchzuführen.

5.3 Ergebnisdaten

Wenn sichergestellt ist, dass man in die Optimierung mit verifizierten Daten eintritt, werden die Ergebnisdaten zumindest semantisch richtig sein; eine Aussage über die Güte der Optimierungsrechnung ist damit noch nicht verbunden.

5.3.1 Valide Ergebnisdaten

Für eine Weiterverarbeitung der Ergebnisdaten, insbesondere im Rahmen einer Rücktransformation, muss sichergestellt sein, dass die Ergebnisdaten eine gültige Parameterkombination repräsentieren. Dies ist aber bereits durch die zwangsweise Erfüllung der Beschränkungen (*constraints*) erfüllt: die Beschränkungen wurden ja genau mit dem Ziel formuliert, nur gültige Parameterkombinationen zuzulassen. Unter der Voraussetzung, dass die formulierten Beschränkungen richtig und vollständig sind, sind auch die Ergebnisdaten in jedem Falle zulässig.

Weiterhin ist zu untersuchen, ob und inwieweit die Ergebnisdaten die Wunsch-Bedingungen (*restrictions*) erfüllen. Während ein „hoher“ Wert der letzten Zielfunktionsberechnung ein Hinweis auf Verletzung von Restriktionen sein kann, wird daraus aber nicht klar, welche Restriktionen verletzt wurden und in welchem Maße. Dies kann man nur anhand der einzelnen Ergebniswerte jeder Restriktion bewerten. Idealziel wäre, dass alle Restriktionen erfüllt wurden (Anzahl nicht-erfüllter Restriktionen gleich 0), allerdings impliziert eine „kleine“ Anzahl (> 0) verletzter Restriktionen allein nicht bereits eine „gute“ Lösung, vielmehr kommt es auf die Gesamtbetrachtung aller Restriktionswerte an. Im Rahmen einer Rücktransformation sind daher neben dem (superponierten) Zielfunktionswert auch die Einzelwerte jeder Restriktion zu übergeben; dabei ist zu beachten, dass diese Zahlenwerte bereits mit dem jeweiligen Gewichtungsfaktor der Restriktion beaufschlagt sind.

5.3.2 Güte der Lösung

Die Validität der Ergebnisdaten impliziert, dass die Ergebnisse „richtig“ sind im Sinne der aufgestellten Bedingungen. Die Frage, ob die Daten „richtig“ sind im Sinne einer Optimierung, ist differenzierter zu betrachten. Einerseits kann man untersuchen, inwieweit man sich gegenüber dem Ursprungszustand durch die Optimierungsrechnung verbessert hat, andererseits kann man fragen, bis zu welchem Grad man das tatsächliche, absolute Optimum erreicht hat. Das Bewertungskriterium ist der Optimierungsrechnung inhärent, denn die Zielfunktion ist per definitionem identisch mit der Berechnung des Gütewerts einer konkreten Parameterkombination. Die Frage der erreichten Verbesserung beantwortet sich also durch den Vergleich der Zielfunktionswerte von Start-Parametern und Ergebnis-Parametern. Die zweite Frage, die die Erreichung des tatsächlich besten Optimums beantworten soll, ist hingegen nur in speziellen Testfällen sinnvoll, da das Optimum ja prinzipbedingt nicht bekannt ist. Vergleichsrechnungen mit unterschiedlichen Verfahren oder Grenzwerten können hier Anhaltspunkte geben. In jedem Fall ist dafür Sorge zu tragen, dass alle diese Werte

- Start-Parameterwerte
- Ergebnis-Parameterwerte
- Zielfunktionswerte von Start und Ergebnis
- alle Ergebnis-Restriktionswerte

einer Nachbearbeitung zwecks Rücktransformation vollständig zur Verfügung stehen.

Kapitel 6

Ontologische Formulierung der Ergebnisinterpretation

Weil die Ergebnisse einer Optimierung möglichst schematisiert und automatisiert im Sinne der Aufgabenstellung interpretiert und dargestellt werden sollen, ist eine ontologische Formulierung der semantischen Bestandteile eines Optimierungsverfahrens als formales Wissensmodell notwendig.

Die auf das eigentliche Optimierungsverfahren eingeengte Betrachtung der Begriffe Parameter, Randbedingungen und Zielfunktion sind zwar für die Optimierungsrechnung selbst notwendig und hinreichend, für die Betrachtung der Gesamtaufgabe aber zu eng (vgl. Abschnitt 4.8).

Daher sollen nun diese Begrifflichkeiten erweitert werden, um eine Diskussion und Anwendbarkeit der in Kap. 3 und Kap. 4 erlangten Erkenntnisse für eine Ergebnisinterpretation im Sinne der Aufgabenstellung zu ermöglichen. Als formaler Rahmen einer solchen Diskussion kann eine streng mathematische Darstellung nicht gefunden werden, daher wird hier für die Diskussion das Ontologie-Konzept als formaler Rahmen in Ansatz gebracht.

Das Konzept der Ontologieformulierung wird hier gewählt, weil es durch folgende, vorteilhaft nutzbare Elemente charakterisiert ist:

- Abgrenzung eines Diskursbereichs
- Klare Definition der auftretenden Konzepte und Ideen als Begriffe
- Klare Definition der Beziehungen zwischen den Begriffen

- Diskussion im Rahmen einer abstrakten, generalisierten Sicht auf das Problem

Damit kann das Konzept der Formulierung der ergebnisinterpretatorischen Aspekte als Ontologie als zulässige und angemessene Methodik gewählt werden, um die methodische Vollständigkeit und Folgerichtigkeit der dargestellten Lösung sicherzustellen.

Ausgehend von einer Formulierung der Optimierung im engen Sinne als eine „Ontologie des Lösers“ werden in diesem Kapitel die Begriffe und ihre Relationen identifiziert, die für eine Formulierung der automatisierten Ergebnisinterpretation notwendig sind. Zunächst werden Begriffskandidaten und Relationskandidaten zusammengetragen und definiert. Anschließend werden alle Kandidaten auf Nützlichkeit und Notwendigkeit für das Ziel der automatisierten Ergebnisinterpretation untersucht. Schließlich werden die verbleibenden Begriffskandidaten in tatsächliche Begriffe der aufzustellenden Ontologie überführt und mit den verbleibenden Relationskandidaten miteinander in Beziehung gesetzt.

Im Anschluss wird die aufgestellte Ontologie auf Tauglichkeit bezüglich der automatischen semantischen Ergebnisinterpretation untersucht.

6.1 Ontologie des Lösers

Das in Kap. 3 eingeführte allgemeine Optimierungsproblem ist charakterisiert durch die drei Kernelemente Parameter, Randbedingungen und Bewertungsfunktion. Fasst man diese drei Elemente als wohldefinierte und aufeinander bezogene Begriffe auf, hat man damit bereits eine erste Optimierungsontologie, die sich allerdings konkret auf die Notwendigkeiten eines Optimierungs-Lösers verengen. Sie sollen daher hier als „Ontologie des Lösers“ beschrieben werden. Im technischen Sinne ist die Ontologie des Lösers die Begriffswelt mit den Datenstrukturen, die der algorithmische Rechenkern definiert und erfordert.

Abb. 6.1 stellt die Ontologie des Lösers als Diagramm dar: Die Bedingungen nehmen Einfluss auf den Wertebereich der Parameter; die Zielfunktion berechnet für die aktuellen Werte der Parameter einen Zielfunktionswert, der zur Bewertung der aktuellen Lösung und iterativen Herleitung einer neuen Parameterkombination herangezogen wird.

Abbildung 6.1: Ontologie des Lösers

Während mit diesen Begriffen ein Durchführen einer Optimierung möglich ist, kann man in Betrachtung dieser Ontologie die Frage aufwerfen, ob die Begriffsmenge durch den Begriff der Lösung (im Sinne eines Ergebnisses oder eines Optimums) erweitert werden muss. Während technisch betrachtet die Optimierungs-Lösung nur eine konkrete Wertepräsentation der Parameter ist, kann man inhaltlich das Optimum (oder jedenfalls das als Optimum geltende Ergebnis der Optimierung) als eigenständigen Begriff auffassen. Eine erste Erweiterung der Ontologie des Lösers ergibt damit die Ontologie der Optimierung gemäß Abb. 6.2. Zu beachten ist hier der bereits diskutierte Sachverhalt, dass das Ergebnis – Optimum – eine bestmögliche Lösung im Rahmen des konkreten Optimierungsverfahrens ist.

Weitere Begriffe in diesem Zusammenhang sind Verfahren, Variator, Steuerdaten, Abbruchbedingungen etc. Diese sollen hier aber nicht in die Optimierungsontologie aufgenommen werden, da sie nur verfahrensinterne Sachverhalte widerspiegeln.

Im Übrigen stellen die in Abschnitt 2.4 vorgestellten Standard-Modellierungssprachen in dieser Sichtweise konkrete ontologische Formulierungen der Input- und Output-Daten der Optimierung dar.

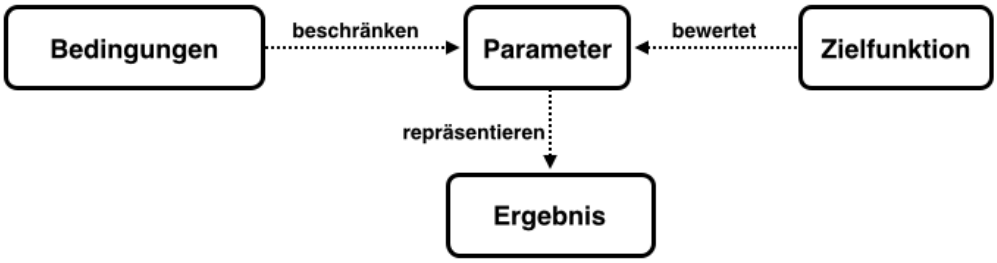


Abbildung 6.2: Ontologie des Lösers um Ergebnis erweitert

6.2 Erweiterung der Ontologie des Lösers um semantische Elemente

Wie die Diskussion der einer Optimierung zugrunde liegenden Aufgabenstellung und ihre Transformation in Kap. 4 gezeigt hat, ist die tatsächliche Anwendung einer Optimierung auf eine Aufgabenstellung komplexer als die Darstellung der Optimierung im engeren Sinne gemäß Abschnitt 6.1. Die hier erkennbare höhere Komplexität lässt sich pauschalisierend auf die Problematik der De-Konstruktion der Informationen zu Daten und der späteren Re-Konstruktion der Daten zu Informationen zurückführen, wird aber im weiteren Verlauf dieses Kapitels noch weiter differenziert.

Die Ontologie des Lösers nach Abschnitt 6.1 muss somit erweitert werden, um die erkennbaren, zusätzlichen Anforderungen erfüllen zu können.

Da nicht von vornherein klar ist, welche Begriffe und Beziehungen für eine solche erweiterte Ontologie in Frage kommen, sollen zunächst kursorisch Aspekte gesammelt werden, die für die Ontologieerweiterung nützlich erscheinen. Anschließend sollen diese Aspekte in allgemeine Kategorien generalisiert und in vorläufige Begriffe (Proto-Begriffe) und vorläufige Beziehungen (Proto-Relationen) eingeteilt werden. Anschließend können alle Proto-Begriffe auf eine begründbare Zugehörigkeit zur angestrebten Ontologie untersucht werden. Nicht begründbare Zugehörigkeiten führen zum Verwerfen des Begriffs, übrigbleibende Begriffe gehen dann in die endgültige Ontologie ein. Diese wird, unter Hinzuziehung der endgültig festgestellten Relationen zwischen den (verbleibenden) Begriffen anschließend als erweiterte Ontologie festgeschrieben.

Die Ontologiebildung vollzieht sich also in folgenden Schritten:

1. Aspekte sammeln
2. Aspekte in Kategorien zusammenfassen
3. Aspekte unterscheiden in Proto-Begriffe und Proto-Relationen
4. Alle Proto-Begriffe auf Notwendigkeit im Rahmen einer automatisierten Ergebnisinterpretation testen; unbegründbare eliminieren
5. Verbleibende Proto-Relationen identifizieren

Den Erörterungen in Kap. 3 und Kap. 4 folgend, insbesondere auch durch die Transformationsvorschrift, lassen sich die Aspekte gemäß Tab. 6.1 als Ausgangspunkt für die Ontologieerweiterung herleiten.

Aspekte	Erläuterung
Normalisierung	s. Abschnitt 4.4.2
Reduktion	s. Abschnitt 4.4.3
Plausibilität	s. Abschnitt 4.4.4
Lösbarkeit	s. Abschnitt 4.4.5
Aufgabe	einzelne Optimierungsaufgabe und zugehörige Zielsetzung
Aufgabenstellung	s. Abschnitt 4.1
Kommentar	kommentierender Freitext zur Aufgabenstellung
Maßeinheiten	Kriterien können in unterschiedlichen Maßeinheiten definiert sein
Semantik	generelle Vorstellung über den Bedeutungsgehalt irgendeines Elements gemäß Abschnitt 4.1.2 und 4.7.2
Beschreibungstexte der Parameter	bedeutungstragende Texte zu den einzelnen Parametern
Beschreibungstexte der Bedingungen	bedeutungstragende Texte zu den einzelnen Bedingungen
Gewichtungsfaktoren der Bedingungen	s. Abschnitt 3.3.2
Steuerdaten	Daten zum Steuern einzelner Berechnungsläufe
Abbruchbedingungen	s. Abschnitt 3.6
Variator	s. Abschnitt 3.5
Schrittweite	Variationsradius der iterativen Änderung der Parameter
Verfahren	verwendete Optimierungsverfahren

Aspekte	Erläuterung
Metadaten	Beschreibungsdaten zu Berechnungsläufen, die den Vorgang als Ganzes betreffen und die nicht zu den eigentlichen Problemdaten gehören
Startwerte von Parametern	Vorbelegung der Parameter
Berechnungsläufe	mehrere unabhängige Optimierungsläufe
Einzelergebniswerte	Endwerte einzelner Parameter am Ende einer Optimierung
Ergebnissätze	mehrere Gruppen von Einzelergebnissen resultierend aus mehreren unabhängigen Optimierungsläufen
Implementierungsklassen	s. Abschnitt 4.4.7
Jeder Berechnungslauf erzeugt eigene Ergebnisse	
Ein Ergebnis eines Berechnungslaufs hat Ergebniswerte für alle Parameter	
Jeder Berechnungslauf besitzt Metadaten	
Eine Aufgabe beginnt mit der Aufgabenstellung	
Eine Aufgabe kann kommentiert sein	

Tabelle 6.1: Aspektsammlung

Eine erste grobe Kategorisierung fasst die Aspekte gemäß Tab. 6.2 zusammen.

Im nächsten Schritt sollen nun die erwartbaren Beziehungen zwischen den Aspekten diskutiert werden. Dazu werden die Relationen in den aufgefundenen Aspekten zunächst verallgemeinert aus der Sicht der Kategorien dargestellt (Abb. 6.3).

Um daraus nun die tatsächlichen Begriffe der angestrebten erweiterten Ontologie herleiten zu können, sollen alle Begriffskandidaten und Relationskandidaten auf ihre Begründbarkeit im Sinne der angestrebten Ontologieverwendung

untersucht werden. Dies entspricht einem Vorgehen gemäß dem Ockham'schen Prinzip, das hier folgendermaßen verstanden werden soll [Ock11]:

1. Von mehreren möglichen Erklärungen desselben Sachverhalts ist die einfachste Theorie allen anderen vorzuziehen.

2. Eine Theorie ist einfach, wenn sie möglichst wenige Variablen und Hypothesen enthält, die in klaren logischen Beziehungen zueinander stehen, aus denen der zu erklärende Sachverhalt logisch folgt.

Clauberg [Cla54] hat dieses Prinzip formuliert als: „Entia non sunt multiplicanda praeter necessitatem" – Entitäten sollen nicht über das notwendige Maß hinaus vermehrt werden. Eine populäre Formulierung stammt von Antoine de Saint-Exupéry [SE39]: „Vollkommenheit entsteht offensichtlich nicht dann, wenn man nichts mehr hinzuzufügen hat, sondern wenn man nichts mehr weglassen kann."

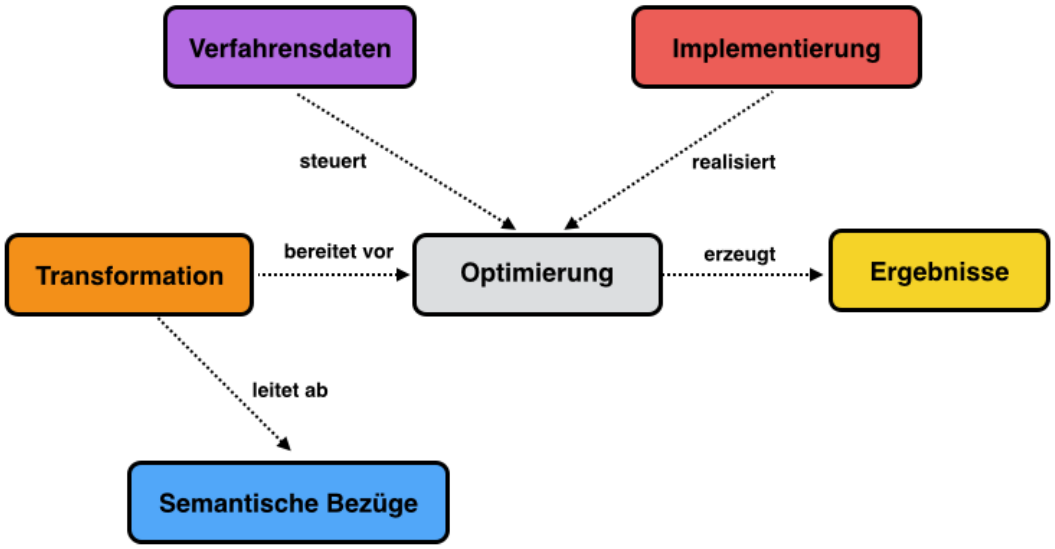


Abbildung 6.3: Aspekt-Relationen, verallgemeinert auf der Ebene von Kategorien

Aspekte	Kategorie
Normalisierung	Transformtion
Reduktion	
Plausibilität	
Lösbarkeit	
Aufgaben	Semantische Bezüge
Aufgabenstellung	
Kommentar	
Maßeinheiten	
Semantik	
Beschreibungstexte der Parameter	
Beschreibungstexte der Bedingungen	
Gewichtungsfaktoren der Bedingungen	
Steuerdaten	Verfahrensdaten
Abbruchbedingungen	
Variator	
Schrittweite	
Verfahren	
Metadaten	Ergebnisse
Startwerte	
Berechnungsläufe	
Einzelergebnisse	
Implementierungsklassen	Implementierung
Jeder Berechnungslauf erzeugt eigene Ergebnisse	Relationen
Ein Ergebnis hat Ergebniswerte für alle Parameter	
Jeder Berechnungslauf besitzt Metadaten	
Eine Aufgabe beginnt mit der Aufgabenstellung	
Eine Aufgabe kann kommentiert sein	

Tabelle 6.2: Kategorienzuordnung

6.3 Ziele einer automatisierten Ergebnisinterpretation

Um aus den aufgeführten Begriffskandidaten diejenigen zu isolieren, die für die Ontologie der automatisierten Ergebnisinterpretation benötigt werden, ist es sinnvoll und notwendig, von den denkbaren Zielen und Anwendungsweisen der Ergebnisinterpretation auszugehen.

Betrachtet man grundsätzliche Kategorien von Zielen einer (manuellen oder automatisierten) Ergebnisinterpretation, kann man daraus die notwendig zu persistierenden Datenelemente herleiten, die eine Modelldefinition der Ergebnisinterpretation beinhalten muss.

6.3.1 Vorher-Nachher-Vergleich

Die offensichtliche Frage nach Durchführung einer Optimierung ist die nach der erreichten Ergebnisgüte. Als Maßstab liegt die Formulierung der Zielfunktion vor. Der Vorher-Nachher-Vergleich stellt dann die Zielfunktionswerte der Start-Parameterkombination und der Ergebnis-Parameterkombination gegenüber. Die zugehörigen Zielfunktionswerte (ohne Straffunktions-Anteile durch Randbedingungen) sind die unmittelbar vergleichbaren Gütewerte.

Für die interpretatorische Persistenz werden daher benötigt: Alle Parameterwerte der Start-Kombination, ein vorab berechneter Zielfunktionswert der Start-Kombination, alle Parameterwerte der Ergebnis-Kombination und der Zielfunktionswert der Ergebnis-Kombination.

6.3.2 Ergebnissatz-Vergleich

Da unterschiedliche Optimierungsverfahren zu mehr oder weniger stark abweichenden Ergebnissen führen können, entsteht der Wunsch, Ergebnisse aus mehreren Optimierungsläufen miteinander zu vergleichen. In ähnlicher Weise sind unterschiedliche Ergebnisse auch bei gleichem Optimierungsverfahren zu erwarten, wenn man mit unterschiedlichen Steuerparametern (Schrittweiten, Abbruchbedingungen) und Startwerten rechnet. Es geht hier also um den Vergleich mehrerer Ergebnissätze untereinander.

Für die interpretatorische Persistenz folgt: Mehrere Ergebnissätze müssen mit allen Ergebniswerten und den zugehörigen Zielfunktionswerten (ohne Penalty-Anteile) gespeichert werden. Um eine Aussage über die Güte der unterschiedlichen Optimierungsläufe treffen zu können, sind außerdem alle zugehörigen Steuerparameter und beschreibenden Metadaten jedes Optimierungslaufs zu speichern. Wegen der Abhängigkeit der Ergebnisse von den Startwerten sind die Startwerte jedes einzelnen Optimierungslaufs zu speichern.

6.3.3 Ergebnis-Markierung

Unabhängig von einer Formulierung als Bedingung können Parameter-Korrelationen schon vor der Optimierung als kritisch oder interessant, jedenfalls als einer besonderen Beachtung wert, definiert werden. Andererseits können auch nach der Optimierung Parameter-Korrelationen als beachtenswert oder eines Hinweises bedürftig gekennzeichnet werden.

Für die interpretatorische Persistenz folgt: Referenzen auf einzelne Parameter und auf Parameter-Mengen müssen gespeichert werden können.

6.3.4 Ergebnis-Reporting

Eine einfache Ergebnisauswertung kann in Form eines standardisierten Reportings erfolgen. Dies ist im einfachsten Fall eine Ausgabe aller Ergebnisse in Listenform. Weitergehende, interpretierende Reportfunktionen werden durch das Post-Processing gemäß Abschnitt 6.3.6 abgedeckt.

Für die interpretatorische Persistenz folgen keine weiteren Anforderungen, da ein einfaches Ergebnis-Reporting bereits mit den zuvor beschriebenen Persistenzaspekten erreichbar ist.

6.3.5 Metadaten-Reporting

Alle textlichen Beschreibungen zur Aufgabenstellung als Ganzes, zu den einzelnen Parametern und den Bedingungen können optional in die Modelldaten eingefügt und für eine Ergebnisweiterverarbeitung geeignet herangezogen werden. Die Texte sind jedoch immer als optional anzunehmen. Das Modell der interpretatorischen Persistenz kann daher keine Textbeschreibungen erzwingen, sollte aber entsprechende Datenfelder auf allen Ebenen vorsehen.

Für die interpretatorische Persistenz folgt: Eine Aufgabenbeschreibung, die die Aufgabe textuell beschreibt, sollte optional enthalten sein. Kommentare, die auf die Aufgabenstellung Bezug nehmen, können ebenfalls gespeichert werden. Die Parameter sollten mit einem Parameternamen und einer semantischen Beschreibung versehen werden.

6.3.6 Ergebnis-Folgeverarbeitung

Alle weitergehenden Verarbeitungen der unmittelbaren Optimierungsergebnisse sind stark kontextabhängig: die konkrete Aufgabenstellung definiert, wie die (genotypischen) Ergebniszahlenwerte der Optimierungsrechnung verarbeitet werden müssen.

Die Ergebnis-Folgeverarbeitung ist also fallweise im jeweiligen Kontext zu lösen. Dazu bedarf es einer interpretierenden Ausführungsinstanz im Sinne eines Post-Prozessors. Alle Aspekte einer Ergebnisaufarbeitung und Ergebnisdeutung werden dann von der Post-Prozessor-Instanz abgewickelt.

Für die interpretatorische Persistenz folgt: Eine Referenz auf einen Post-Prozessor muss gespeichert werden. Die Referenz ist abhängig vom gewählten Laufzeitsystem; im Rahmen einer Java-VM wäre dies die Angabe einer Java-Klasse inkl. des zugehörigen Klassenpfads. Weitere mitzuführende Attribute können nicht mehr allgemeinverbindlich definiert werden, sondern sind ebenfalls kontextabhängig und vom Post-Prozessor definiert. Hier bleibt nur die Möglichkeit, entweder durch Namespaces abgegrenzte Datenbereiche einzuführen oder alle Zusatzdaten in Form von referenzierten Dateien mitzuführen.

6.4 Begriffsanalyse

Die vorgeschlagenen Aspekte werden nun einzeln auf Unverzichtbarkeit im Rahmen der automatisierten Ergebnisinterpretation in der angestrebten Ontologie untersucht. Dabei wird konkret auf folgende Kriterien und Ziele einer automatisierten Ergebnisinterpretation geachtet:

- Wenn der Aspekt erkennbar keinen Beitrag zur automatisierten Interpretation der Ergebnisse leistet, kann er entfallen.
- Aspekte, die nicht sinnvoll persistiert werden können, entfallen.

- Abhängige Aspekte, also solche Aspekte, die sich indirekt aus anderen Aspekten herleiten lassen, können entfallen.
- Redundante Aspekte, die von anderen Aspekten mitabgedeckt werden, können entfallen.
- Optimierungsverfahrenstechnische Aspekte sind für eine Ergebnisinterpretation irrelevant.

Im Gegenzug sind alle interpretatorischen Elemente, die im Rahmen der Transformation, Rücktransformation, des Semantik-Stripping und der Semantik-Wiederanreicherung benötigt werden, in die Ontologie aufzunehmen.

Normalisierung

Jedem variierbaren Parameter wird ein Normalisierungsfaktor zugeordnet, der eine einheitliche Variation aller Parameterwerte sicherstellt. Der Normalisierungsfaktor überführt also den phänotypischen in einen genotypischen Wert. Da im Rahmen der Rücktransformation die berechneten genotypischen Ergebniswerte wieder in die phänotypische Ebene der Aufgabenstellung zurückgeführt werden sollen, müssen alle Normalisierungsfaktoren und ihre Beziehungen zu ihrem jeweiligen Parameter notiert werden. Die Normalisierungsfaktoren sind also für die erweiterte Ontologie unverzichtbar.

Proto-Begriff: *Normalisierungsfaktor*
Proto-Relation: *Zugeordnet zu je einem Parameter*

Reduktion

Bezugnehmend auf die elementaren Begriffe *Parameter* und *Bedingungen* wirkt der Begriff Reduktion als eine Art Filter für das Optimierungsverfahren. Es werden nur die Parameter und Bedingungen für das Verfahren verfügbar gemacht, die für die Aufgabenstellung relevant sind. Die Reduktion leistet keinen nutzbaren Beitrag zur Interpretation der Ergebnisse.

Entfällt

Plausibilität

Die Plausibilität ist ein Aspekt, der nur in der Hinführung der Aufga-

benstellung zur Optimierung auftritt und Aussagen über die prinzipielle Berechenbarkeit macht. Diese Aussagen sind aber zum Zeitpunkt der Ergebnisinterpretation nicht mehr relevant, daher ist der Aspekt der Plausibilität obsolet.

Entfällt

Lösbarkeit

Der Lösbarkeitsaspekt ist für die Optimierung ein notwendiges Kriterium. Zum Zeitpunkt der Ergebnisinterpretation ist der Aspekt implizit herleitbar: Wenn Ergebnissätze existieren, ist der Aspekt der Lösbarkeit erfüllt, wenn hingegen keine Ergebnisse existieren, ist die Lösbarkeit erkennbar nicht gegeben. Für beide Fälle bietet der Aspekt der Lösbarkeit für eine Ergebnisinterpretation daher keinen Beitrag.

Entfällt

Aufgabe

Aus der Aufgabenstellung lässt sich als wesentlicher Kern die zu optimierende Aufgabe formulieren. Frei von zusätzlichen Informationen konzentriert sich die Aufgabe auf genau eine Zieldefinition und ist damit ein unverzichtbarer Begriff der erweiterten Ontologie.

Proto-Begriff: *Aufgabe*
Proto-Relatione: *Enthält eine Aufgabenstellung*

Aufgabenstellung

Eine gegebene Problembeschreibung stellt Parameter unter Einfluss von Bedingungen und einer Zielfunktion in einem semantischen Kontext dar. Der Aspekt der Aufgabenstellung bedingt daher seine Existenz in der Ontologie.

Proto-Begriff: *Aufgabenstellung*
Proto-Relation: *zugeordnet zu einer Aufgabe*

Kommentar

Zusätzlich zu einer Aufgabenstellung können Hinweise aller Art in Form eines Kommentars einfließen. Ein Kommentar stellt eine optionale Informationsquelle dar und bedingt seine mögliche Einführung in die Ontologie.

Proto-Begriff: *Kommentar*
Proto-Relation: *zugeordnet zu einer Aufgabe*

Maßeinheiten

Parameter können in verschiedenen Dimensionen und Maßeinheiten in der Aufgabenstellung auftreten. Damit Maßeinheiten, obwohl sie verfahrensintern keine Berücksichtigung finden, der Lösung wieder zugeführt werden können, muss dieser Aspekt in der Ontologie Berücksichtigung finden.

Proto-Begriff: *Maßeinheiten*
Proto-Relation: *zugeordnet zu je einem Parameter*

Semantik

Der Aspekt Semantik beschreibt Informationen explikatorischer Art zu den Parametern und Bedingungen. Die Semantiken entstammen der Aufgabenstellung und enthalten eine kontextabhängige Bedeutungsrepräsentation der Parameter und Bedingungen. Semantiken sind insofern unverzichtbar für die Ontologie.

Proto-Begriff: *Semantik*
Proto-Relation: *zugeordnet zu je einem Parameter oder einer Bedingung*

Beschreibungstexte der Parameter

In einer verbal formulierten Aufgabenstellung können Parameter in einem kontextabhängigen Textabschnitt enthalten sein, der diese Parameter beschreibt und in der Aufgabenstellung gegenüber weiteren Begriffen in einen Kontext stellt. Der Beschreibungstext der Parameter ist aber jederzeit aus der Aufgabenstellung und der Semantik ableitbar.

Entfällt

Beschreibungstexte der Bedingungen

In einer verbal formulierten Aufgabenstellung können Bedingungen in einem kontextabhängigen Textabschnitt enthalten sein, der diese Bedingungen beschreibt und in der Aufgabenstellung gegenüber weiteren Begriffen in einen Kontext stellt. Der Beschreibungstext der Bedingungen ist aber jederzeit aus der Aufgabenstellung und der Semantik ableitbar.

Entfällt

Gewichtungsfaktoren der Bedingungen

Innerhalb der Aufgabenstellung können Bedingungen priorisiert sein. Um eine Gewichtung von Bedingungen verarbeiten zu können, muss der Gewichtungsfaktor in die Ontologie eingeführt werden.

Proto-Begriff: *Gewichtungsfaktoren der Bedingungen*
Proto-Relation: *zugeordnet zu je einer Bedingung*

Steuerdaten

Ein Optimierungsverfahren kann Steuerdaten für die interne Berechnung erfordern. Nachdem ein Optimierungsverfahren Ergebnisse produziert hat, sind die Steuerdaten obsolet, jedoch für eine erweiterte Ontologie im Sinne der Vergleichbarkeit von Ergebnissätzen nach Abschnitt 6.3.3 sind diese Steuerdaten erforderlich.

Proto-Begriff: *Steuerdaten*
Proto-Relation: *zugeordnet zu Berechnungsläufen*

Abbruchbedingungen

Falls ein Optimierungsverfahren nicht inhärent determiniert, können definierte Abbruchbedingungen das Verfahren beenden. Nachdem ein Optimierungsverfahren Ergebnisse erzeugt hat, sind Abbruchbedingungen für eine Lösungsdarstellung im Rahmen von Vergleichen nützlich. Sie sind Bestandteil der Steuerdaten.

Proto-Begriff: *Steuerdaten*
Proto-Relation: *zugeordnet zu Berechnungsläufen*

Variator

Der Variator definiert die Art und Weise, wie Parameter variiert werden und welche Schrittweitensteuerung gewählt wird. Der Variator ist Bestandteil eines Optimierungsverfahrens und nicht Teil der erweiterten Ontologie, da diese agnostisch gegenüber dem tatsächlich eingesetzten Optimierungsverfahren ist.

Entfällt

Schrittweite

Die Schrittweite ist Teil des Variators und entfällt aus demselben Grund.

Entfällt

Verfahren

Der Aspekt des tatsächlich angewendeten Optimierungsverfahrens ist für Zwecke der Vergleichsauswertung und für eine reproduzierbare Lösung notwendig. Die Angabe des Verfahrens ist daher optimierungsintern und für die Ontologie unverzichtbar.

Proto-Begriff: *Verfahren*
Proto-Relation: *zugeordnet zu Berechnungsläufen*

Metadaten

Zusätzliche Informationen über Parameter, Bedingungen oder weitere Kriterien, die eine meta-technische Beschreibung erhalten können, werden in dem Aspekt der Metadaten verwaltet und über das Optimierungsverfahren mitgeführt.

Proto-Begriff: *Metadaten*
Proto-Relation: *Aufgabe und Berechnungsläufe*

Startwerte

Die Startwerte der Parameter sind die Ausgangswerte vor Beginn der Optimierung. Da im Rahmen von mehrfach angewendeter Optimierung der Ergebnis-Parametersatz einer Optimierung zum Start-Parametersatz einer Folge-Optimierung werden kann, werden die Startwerte der Parameter sinnnvollerweise nicht gesondert, sondern im Rahmen der Einzelergebniswerte der Parameter gespeichert.

Proto-Begriff: *Einzelergebniswerte*
Proto-Relation: *zugeordnet zu Parametern*

Berechnungsläufe

Die Berechnungsläufe bilden einen Container für Ergebnissätze. In Abschnitt 6.3.2 wird die Möglichkeit gefordert, mehrere Ergebnissätze mit unterschiedlichen Steuerdaten und Optimierungsverfahren zu vergleichen.

Ein Optimierungsverfahren kann durch mehrfache Anwendung gegebenfalls eine bessere Lösung produzieren. Um einen Verlauf über mehrere Berechnungen nachvollziehen zu können, müssen diese Berechnungsläufe erfasst werden.

Proto-Begriff: *Berechnungslauf*
Proto-Relation: *zugeordnet zu der Ontologie des Lösers*

Einzelergebniswerte

Nach Abschnitt 6.3.2, dem Ergebnissatz-Vergleich, sollen aus zwei Ergebnissätzen von jeweils zwei identischen (semantisch gleichen) Parametern die Einzelergebniswerte miteinander verglichen werden. Die Ergebnis-Markierung gemäß Abschnitt 6.3.3 fordert die Referenzierung auf Parameter für eine besondere Beachtung und für eine Markierung in einer Ausgabe.

Proto-Begriff: *Einzelergebniswerte*
Proto-Relation: *zugeordnet zu je einer Parameterreferenz in einem Ergebnissatz*

Ergebnissätze

Für das Ziel einer automatischen Ergebnisinterpretation nach Abschnitt 6.3.1, dem Vorher-Nacher-Vergleich, benötigt man zwei Ergebnissätze: einer, der die Startwerte der Aufgabe enthält, und derjenige, der nach der Optimierung bessere Werte im Sinne der Zielfunktion repräsentiert. Diese werden einander gegenübergestellt. Zudem wird in Abschnitt 6.3.2 die Möglichkeit gefordert, mehrere Ergebnissätze mit unterschiedlichen Steuerdaten und Optimierungsverfahren zu vergleichen.

Proto-Begriff: *Ergebnissätze*
Proto-Relation: *Berechnungslauf und Einzelergebniswerte*

Implementierungsklassen

Bedingungen und Zielfunktionsdefinitionen erfordern für ein Optimierungsverfahren eine Implementierung, die jedoch für die Ergebnisinterpretation nicht mehr relevant ist.

Entfällt

Damit stehen die Begriffe für die Ontologie fest. Implizit stehen damit auch die zugehörigen Relationen fest, da nur die Relationen in die Ontologie übernommen werden, die zwischen den verbleibenden Begriffen existieren.

6.5 Ontologie der Interpretation

Die Dekonstruktion (Informationen → Daten) entfernt Semantik. Diese soll im Rahmen der Ontologie der Interpretation persistiert werden, um nach der Optimierung den Ergebnisdaten wieder hinzugefügt werden zu können. Abb. 6.4 verdeutlicht die Aufspaltung der Informationen in Daten und Semantik.

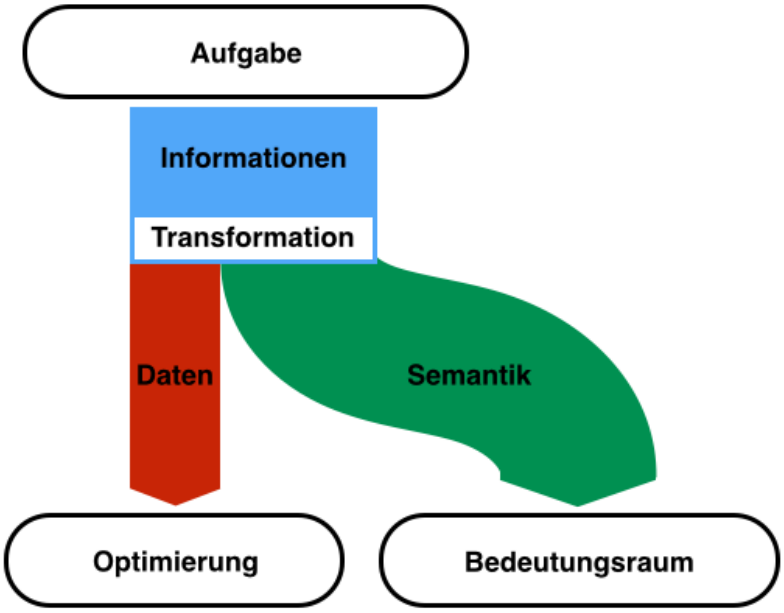


Abbildung 6.4: Informationsfluss und Teilung der Daten und Semantik

Nachdem in Abschnitt 6.4 die Begriffe der Ontologie der Interpretation begründet wurden, erfolgt durch Abb. 6.5 eine Gesamtdarstellung der Begriffe als Baumstruktur und ausgehend von der gestellten Aufgabe. Die farblich gekennzeichneten Begriffe entsprechen der Bedeutung und Kategorisierung der Informationsflüsse aus Abb. 6.4.

Abb. 6.5 erweitert die Ontologie des Lösers (rot) um die Begriffe der interpretatorischen Ontologie (grün und blau). Während die blau gekennzeichneten Begriffe eine allgemeine Aussage zu der Aufgabe liefern, werden durch die grün dargestellten Begriffe die semantik-spezifischen Teile der Aufgabe beschrieben.

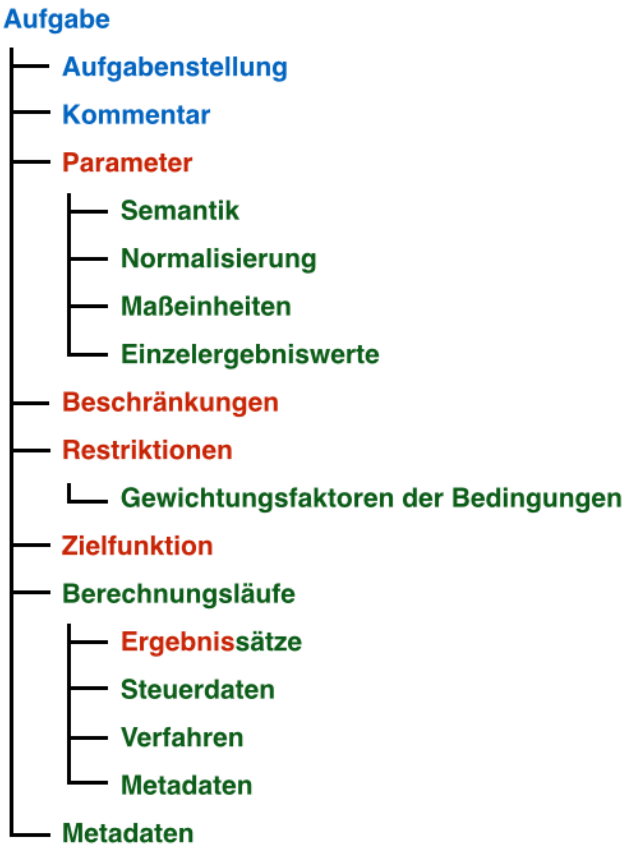


Abbildung 6.5: Ontologie der Interpretation

Kapitel 7

Interpretatorische Modellierungssprache

Weil eine Optimierung strukturierte Daten erwartet, um daraus Parameterkombinationen und Ergebnisse zu produzieren, müssen die Daten in einer geeigneten Datenstruktur beschrieben und gespeichert werden können.

Durch Anwenden eines Optimierungsverfahrens auf eine konkrete Problemstellung werden Startwerte von Parametern in (hoffentlich bessere) Endwerte überführt. Diese Start- und Endwerte sind zunächst semantikfreie reine Zahlenwerte im Sinne der Optimierungsrechnung. Weiterhin ist eine wesentliche Fragestellung, ob und inwiefern die erzielten Optimierungsergebnisse die Problemstellung nun tatsächlich „besser“ lösen als die ursprünglichen Startwerte. Da man in der Praxis darüber hinaus nicht selten mehrere Optimierungsläufe, ggf. mit unterschiedlichen Optimierungsverfahren, anstößt, kommt der Speicherung, dem Vergleich und der Verifizierung der Ergebniswerte eine hohe Bedeutung zu. Insbesondere Verifizierung und Vergleich sind nicht ohne Beachtung der aufgabenbezogenen Semantik möglich.

Optimierungsergebnisse sind also immer im Sinne von Abschnitt 4.4 auf die semantische Ebene der Aufgabenstellung rückzutransformieren, um sie interpretieren und vergleichen zu können. Dies ist gleichbedeutend mit dem Überspringen der semantischen Lücke (Abschnitt 4.7) in Hin- und Rück-Richtung.

Nachdem in Kap. 6 die konzeptionellen Elemente einer semantischen Ergebnisinterpretation als Ontologie formuliert wurden, sollen diese nun als nutzbare Modellierungssprache implementiert werden.

7.1 Anforderungen an eine Modellierungssprache für Optimierung

Die Meta-Anforderungen an die zu wählende Sprachimplementierung sind die üblicherweise in solchen Fällen gesetzten Anforderungen; die Sprache soll:

- vollständig und strukturiert
- exakt
- relationserhaltend
- flexibel
- standardisiert verarbeitbar

sein. Zur Wahl stehen Realisierungen als Datenbank-Modell (in der Regel als relationales Datenmodell) und eine strukturiert-textuelle, dokumentbasierte Implementierung (z. B. als XML oder JSON).

Die Entscheidung für oder gegen eines dieser Persistenzverfahren ist für die weitere Diskussion nicht relevant und kann fallweise getroffen werden. Wesentlich ist, dass die Anforderungen an Strukturierung und Abbildbarkeit von Referenzen erfüllt sind. Für die weitere Diskussion wird beispielhaft XML als Sprachstruktur verwendet, da es sich leicht textlich darstellen lässt und keine besonderen technischen Anforderungen stellt. Eine Implementierung mit XML bietet sich auch an, da XML die Kriterien

- systemunabhängig
- Validierbarkeit und leichte Erweiterbarkeit
- lesbares Dateiformat
- transportierbares Datenformat
- Standard-Implementierung durch vorhandene Bibliotheken

erfüllt. XML besitzt eine weite Unterstützung zur Verarbeitung in vielen Kontexten, Systemen und Anwendungen. Im Rahmen der Optimierungsproblematik sind insbesondere die Aspekte der hierarchischen Strukturierung, der Erweiterbarkeit und der Kommunizierbarkeit über Netzwerkschnittstellen, z. B. im Rahmen von Webservices, hervorzuheben.

7.2 Verwendung existierender Modellierungssprachen

In Abschnitt 2.4 wurden algebraische Modellierungssprachen wie GAMS und AMPL eingeführt und ihre gängige Verwendung im Rahmen von Optimierungsrechnungen dargestellt. Sie basieren auf proprietären Textformaten, die auf ihren Anwendungszweck der Modellierung algebraischer Probleme angepasst sind. Es ist daher zu prüfen, ob diese Sprachen für den angestrebten Zweck der automatisierten Ergebnisinterpretation eingesetzt werden können, da sie ja bereits die Modellierung der Optimierungsrechnung erlauben.

Die Modellierung der Optimierungsrechnung im engeren Sinne entspricht der im Abschnitt 6.1 beschriebenen Ontologie des Lösers. Am Beispiel der „Kreis“-Aufgabe (Abschnitt 3.8.2) soll im Folgenden die Formulierung der Optimierungsrechnung mit zwei bekannten algebraischen Modellierungssprachen dargestellt werden.

7.2.1 Formulierung mit GAMS

Die Testaufgabe „Kreis-Aufgabe“ (Abschnitt 3.8.2) kann mithilfe von GAMS in der folgenden Weise als drei-parametriges Problem (mit den x- und y-Koordinaten der Parameter P_1, P_2, P_3) formuliert werden:

```
SETS
P    points    / P1, P2, P3 / ;
PARAMETERS
PX(P)     / P1    0.0
            P2    5.0
            P3    5.0 /
PY(P)     / P1    0.0
            P2   15.0
            P3 -15.0 / ;
VARIABLES
X         x-koordinate des mittelpunkts
Y         y-koordinate des mittelpunkts
R         radius
Z         zielfunktionswert;
POSITIVE VARIABLE R;
EQUATIONS
```

```
ZFK     zielfunktion definieren ;
ZFK ..        Z =E= SUM((P), POWER(R -- SQRT(POWER(PX(P) - X, 2) +
                          POWER(PY(P) - Y, 2)), 2) ;) ;
MODEL CIRCLEPROBLEM /ALL/ ;
SOLVE CIRCLEPROBLEM USING NLP MINIMIZING Z ;
```

Der GAMS-basierende Löser kann das Ergebnis in folgender Form auswerfen:

```
VAR X          -INF          25.0000     +INF     EPS
VAR Y          -INF            .         +INF     EPS
VAR R            .           25.0000     +INF     EPS
VAR Z          -INF     5.260756E-26     +INF      .

X     x-koordinate des mittelpunkts
Y     y-koordinate des mittelpunkts
R     radius
```

Am gegebenen Beispiel kann man erkennen, dass GAMS die Problem-Modellbildung sehr kompakt hält. Als Ergebnis wird die Mittelpunktskoordinate (25, 0) und der Radius (25) ausgegeben sowie der Zielfunktionswert mit $5,26 \cdot 10^{-26}$. Als Ausgabe werden für die Variablen X, Y und R der Definitionsbereich $[-\infty \cdots +\infty]$ bzw. $[0.0 \cdots +\infty]$ aufgrund der gegebenen Randbedingungen und der Ergebniswert ausgegeben (mit dem einfachen Punkt als Kürzel für 0.0). Die Variable Z repräsentiert den Zielfunktionswert; ihr Wert von annähernd 0 zeigt als Gütewert, dass die Optimierungsrechnung dem theoretischen Ergebniswert extrem nahe gekommen ist. Die letzte Spalte repräsentiert die Sensitivitätswerte pro Variable, wobei die Angabe EPS eine sehr kleine Sensitivität der Werte gegenüber Wertänderungen anzeigt.

Die komplette Berechnungsausgabe ist in Anhang 11.2.2 gegeben.

7.2.2 Formulierung mit AMPL

Dieselbe Aufgabe in einer AMPL-Fomulierung sieht folgendermaßen aus:

```
var radius;          ## Radius
set pointNames;      ## Gegebene Punkte
var mx;      ## Mittelpunkt
var my;      ## Mittelpunkt
```

```
param n;
param pointsXi in 1..n;
param pointsYi in 1..n;

minimize zfkvalue: sumi in 1..n (radius - sqrt( (pointsX[i]-mx)^2
   + (pointsY[i]-my)^2)  )^2;
subject to Xrange: 0 <= mx <= 120;
subject to Yrange: 0 <= my <= 120;
subject to Radius: 0 <= radius <= 120;
```

mit der zusätzlichen Beschreibung der Anfangswerte:

```
param radius := 60;
param n:= 3;
param mx:= 10;
param my:= 10;

param pointsX:= 1   0
                2   5
                3   5;
param pointsY:= 1   0
                2 -15
                3  15;
```

An der beispielhaften Modellierung kann man zeigen, dass AMPL die Trennung von Modell-Ebene und Daten-Ebene ermöglicht. Eine Entkopplung von Modell und Daten erscheint dann sinnvoll, wenn das Modell stabil ist und keine Anpassungen vor einem Berechnungslauf benötigt, die Aufgabenbeschreibung also statisch ist.

```
ampl: reset;
ampl: option solver minox;
ampl: model circle.mod;
ampl: data circle.dat;
ampl: solve;
MINOS 5.51: optimal solution found.
11 iterations, objective 1.184022629e-13
Nonlin evals: obj = 23, grad = 22.
ampl: display radius > circle.out;
radius = 25
```

Diese Optimierungsrechnung kommt erwartungsgemäß zum prinzipiell glei-

chen Ergebnis. Abweichungen in der Genauigkeit des Ergebnisses (repräsentiert durch den Zielfunktionswert von $1,184 \cdot 10^{-13}$) sind der Tatsache geschuldet, dass hier ein anderer Löser eingesetzt wurde. In der praktischen Interpretation des Ergebnisses ist der Unterschied in der Größenordnung 10^{-26} (GAMS) gegenüber 10^{-13} (AMPL) vernachlässigbar. Die Ausgabe ist vom Benutzer auf den Radius begrenzt worden.

Die komplette Berechnungsausgabe ist in Anhang 11.2.3 gegeben.

7.2.3 Anwendbarkeit algebraischer Modellierungssprachen

Die beiden vorherigen Abschnitte haben an einem einfachen Beispiel gezeigt, wie eine Problemstellung für die Optimierungsrechnung abgebildet werden kann. Andere, hier nicht ausgeführte algebraische Modellierungssprachen führen auf ähnliche Formulierungen.

An den dargestellten Beispielen wird sofort die schwache Strukturierung der Sprachbeschreibung deutlich. Insbesondere die Tatsache, dass die existierenden Sprachimplementierungen keinen Mechanismus für die Abbildung von Referenzen auf andere semantische Elemente besitzen, muss negativ bewertet werden. Wie bereits diskutiert und in der Ontologie in Abschnitt 6.1 dargelegt, ist gerade diese Referenzbildung für eine automatisierte Ergebnisinterpretation unverzichtbar.

Dem Vorteil der Verwendung einer Standard-Modellierungssprache steht also der Nachteil gegenüber, dass sich Standard-Modellierungssprachen schlecht um referenzierende Attribute erweitern lassen. Für den hier diskutierten Einsatzzweck würden Felder benötigt, die nicht in den beschriebenen Modellierungssprachen enthalten sind und auch nicht oder nur unzulänglich eingebracht werden könnten, nämlich höchstens als unstrukturierte textliche Anmerkungen.

Der Tatsache geschuldet, dass bekannte Modellierungssprachen nicht geeignet erweiterbar sind, die benötigten zusätzlichen Datenfelder aber abgedeckt werden müssen, wird nunmehr XML als Modellierungssprache ausgewählt.

7.3 Eigene XML-Sprachimplementierung

Die grundsätzliche Forderung an eine Modellierungssprache im Umfeld der Optimierung ist die Erfassung aller für die Optimierung notwendigen Informationen. Dabei bedürfen die Angaben zum Optimierungsverfahren weiterer Erläuterung: Zum einen kann es sich hier um die Angabe eines genauen Verfahrens handeln, zum anderen ist aber auch die allgemeine Angabe einer Löserklasse (linear, nichtlinear, etc.) denkbar. Die Steuerdaten können Schrittweitensteuerungen, Abbruchbedingungen, Variationsgrenzen u. ä. sein.

Für das Ziel der automatischen Ergebnisinterpretation und Weiterverarbeitung benötigt man ein vereinheitlichtes Modellierungsprachenformat, welches neben der Ontologie des Lösers auch die erweiterte Ontologie der Interpretation in einem Datenformat kapselt. Wenn XML als Sprachformat festgelegt ist, kann eine beschreibende Schemadatei erstellt werden, um eine Validierung zu ermöglichen und damit die sichere Persistenz, den Transport und die Weiterverarbeitung aller Daten zu gewährleisten.

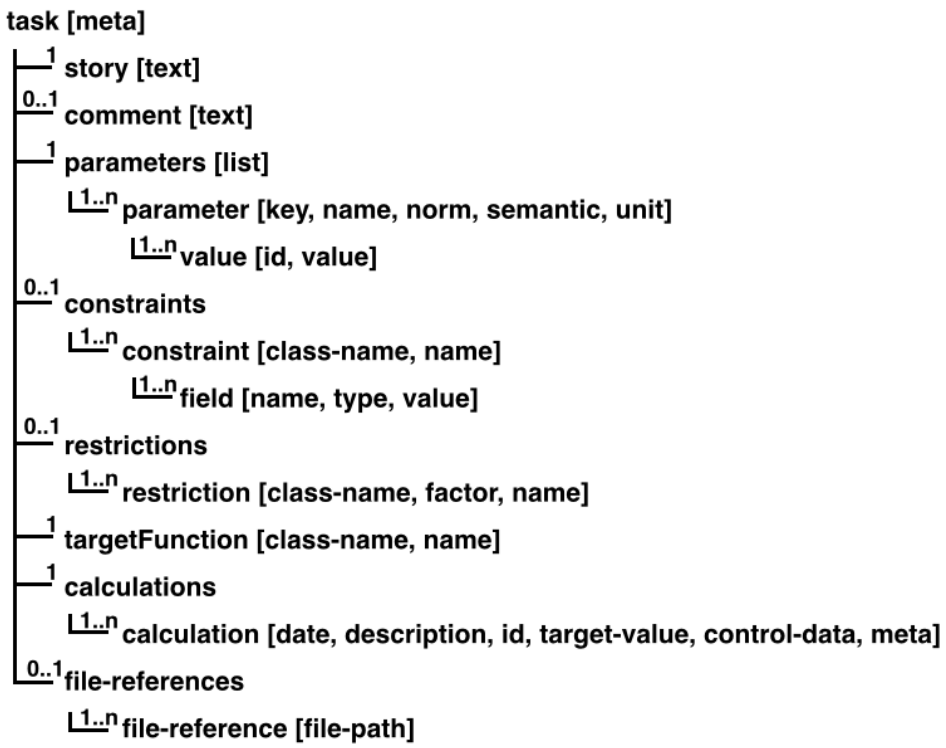


Abbildung 7.1: Struktur für eine XML-Schema-Datei der Ontologie der Interpretation

Aus der in Abschnitt 6.5 aufgestellten Ontologie und nach der Baumstruktur in Abb. 6.5 werden die Ontologie-Begriffe in ein XML-Schema überführt, dabei werden die Begriffe internationalisierend in englischsprachige Bezeichner übersetzt. Einige der Begriffe werden im Rahmen dieser XML-Schema-Definition als Elemente, andere als Attribute der Elemente dargestellt. Im Rahmen einer Relationen-Zuordnung wird die Darstellung aus Abb. 6.5 mit Kardinalitäten zu Abb. 7.1 erweitert; diese dient dann als Vorlage der endgültigen XML-Schema-Definition. Die vollständige Schema-Definition findet sich im Anhang in 11.1.1.

Die XML-Schema-Definition erlaubt es, Aufgabenstellungen als validen, überprüfbaren XML-Code zu modellieren. Die XML-Schema-Definition lässt sich flexibel bei Bedarf um weitere Elemente und Attribute erweitern, sollte eine Aufgabenstellung dies fordern.

Im folgenden Abschnitt wird die XML-Schema-Definition anhand eines konkreten Beispiels vorgestellt und diskutiert.

7.3.1 Beispielformulierung im eigenen XML-Format

Das bereits verwendete „Kreis“-Beispiel aus Abschnitt 3.8.2 formuliert sich in dem aufgestellten XML-Schema wie folgt:

```
<?xml version="1.0" encoding="UTF-8" ?>
<task name="DynamicCircle">
  <story>
    Kreisproblem: gegeben sei ein Halbkreis mit unbekanntem
        Durchmesser mit zwei ausgezeichneten orthogonalen Strecken.
        Wie groß ist der Radius?
  </story>
  <comment/>
    Die geschlossene Lösung führt unter Verwendung des Satzes von
        Thales schnell auf den gesuchten Radius des Halbkreises.
        Als Optimierungsproblem aufgefasst, läuft die
        Parametervariation über drei Parameter: der Radius sowie
        die Position des Mittelpunktes (x- und y-Koordinate).
  </comment>
  <parameters>
    <parameter key="x" name="x" norm="1.0" semantic="x-Wert"
      semantic-unit="mm" />
      <value id="1" value="20.0"/>
```

```
    </parameter>
    <parameter key="y" name="y" norm="1.0" semantic="y-Wert"
      semantic-unit="mm" />
      <value id="1" value="0.0"/>
    </parameter>
    <parameter key="R" name="R" norm="1.0" semantic="Radius"
      semantic-unit="mm" />
      <value id="1" value="15.0"/>
    </parameter>
  </parameters>
  <constraints>
    <constraint class-name="RangeConstraint" name="Radius">
      <field name="Minimum" type="min" value="0.0"/>
      <field name="Parameter" type="ref" key-ref="R"/>
    </constraint>
  </constraints>
  <restrictions/>
  <target-function class-name="CircleTarget" name="Zfk"/>
  <calculations>
    <calculation date="13.08.2015;14:45" description="Startwerte"
      id="1" target-value="100000" control-data="" meta=""/>
    <calculation date="13.08.2015;14:47" description="NLPO_1"
      id="2" target-value="1.3142892" control-data="" meta=""/>
  </calculations>
  <file-references>
    < file-reference name="" />
  </file-references>
</task>
```

Eingabeformat

Dieses Schema-Modell geht davon aus, dass alle algorithmisch arbeitenden Subjekte als Klassen im Sinne einer Java-Implementierung realisiert werden. Dies betrifft z. B. auszuwertende Beschränkungen und die Zielfunktion. Es muss betont werden, dass dies eine Implementierungsentscheidung ist, die auch anders gefällt werden kann. Einerseits können solche auszuwertenden Subjekte in anderen Laufzeitumgebungen implementiert werden, andererseits wäre es auch denkbar, die Implementierungen in Webservices zu kapseln. Für die Diskussion der Ergebnisinterpretation ist die Art der konkreten Implementierung

allerdings unerheblich, sodass die hier gewählte Darstellung als Java-Klassen als exemplarisch zulässig angesehen werden kann. Andere konkrete Implementierungen, z. B. als Webservice, würden an diesen Stellen anstatt eines Klassenbezugs eine URL eines Webservice verwenden.

Bei der Entwicklung der Modellierungssprache wurde wesentlich der Aspekt berücksichtigt, neben der Aufnahme einer strukturierten transformierten Beschreibung der Aufgabe insbesondere auch die ursprünglich formulierte Aufgabenstellung selbst zu beinhalten. Diese konzeptionelle Doppelung der Informationen ist für die spätere Ergebnisinterpretation insofern förderlich, als sich jederzeit Ergebnisse mit den transformierten Daten im Abgleich der ursprünglichen Aufgabenstellung vergleichen und verifizieren lassen. Zwar werden Semantik-Informationen in der Transformation in der Weise behandelt, dass sie an die entsprechenden Elemente gehängt werden. Dies ist jedoch nur element-lokal semantikerhaltend, ein globaler Zusammenhang zwischen den Elementinformationen ist dadurch nicht hinreichend gegeben. Dieser lässt sich später über die ursprüngliche Aufgabenstellung wiederherstellen.

Das Wurzelelement der XML-Struktur stellt die Aufgabe <task> mit zusätzlichen Informationen (meta) dar, in der alle Informationen, im Sinne von Daten, Semantik und Berechnungsläufen enthalten sind:

```
<task meta="" >
  ...
</task>
```

Die eigentliche XML-Beschreibungssprache beginnt damit, die ursprüngliche Aufgabenbeschreibung in einem Element <story> vollständig zu erfassen. Optionale zusätzliche Informationen können im nicht maschinell auswertbaren Kommentar-Element <comment> aufgenommen werden.

Für die transformierte Beschreibung der Optimierungsaufgabe ist offensichtlich die Angabe aller auftretenden Parameter, aller Beschränkungen und Restriktionen sowie der Zielfunktion erforderlich. Da Parameter, Beschränkungen und Restriktionen mehrfach und damit als Liste auftreten können, werden diese Elemente in sogenannte Container-Elemente, bspw. dem Container-Element <parameters> eingebettet, um die mehrfach auftretenden Elemente sinnvoll zu verwalten.

Ein Parameter (<parameter>) benötigt als Angabe eine Benennung (name)

und einen Startwert, der abstrakt in einem Element <value> mit Angabe einer Identifikationsnummer gespeichert wird. Die Speicherung als Element erlaubt das Hinterlegen mehrerer Startwerte und ermöglicht das Abspeichern von Parameterergebniswerten, die ihrerseits als Startwerte für weitere Berechnungsläufe dienen können. Aus der Kriterientransformation folgt, dass eine Semantikinformation (semantic) und ein Normierungsfaktor (norm) angegeben werden müssen. Dimensionsbehaftete Werte können in Erweiterung der Semantikinformation mit einer Maßeinheit (unit) versehen werden. Jeder Parameter wird mit einem eindeutigen Identifikator (key) für Referenzen versehen.

```
<parameter key="R" name="R" norm="1.0" semantic="Radius" unit="mm"
  <value id="1" value="15.0" />
</parameter>
```

Ein Beschränkungs-Element (<constraint>) benötigt neben der Angabe eines Namens (name) ein Attribut (class-name), das den Klassennamen mit der Implementierung der Beschränkung enthält. Damit diese Klasse die betroffenen Parameter und die beschränkenden Werte in Relation setzen kann, werden diese als Feld-Elemente (<field>) ebenfalls mit Namen (name) versehen, wobei ein Verwendungstyp (type) beschreibt, wie das Feld angewendet werden soll: Angaben wie min oder max kennzeichnen limitierende Zahlenwerte, die Angabe ref kennzeichnet einen Bezug zu einem Parameter. Das Attribut value gibt den konkreten Zahlenwert an, während im Falle type="ref" der bezogene Parameter über seinen Schlüssel im Attribut key-ref angegeben wird:

```
<constraint name="Radius" class-name="RangeConstraint"
  <field name="Minimum" type="min" value="0.0" />
  <field name="Parameter" type="ref" key-ref="R" />
</constraint>
```

Alternativ zur Angabe einer Klasse kann die Implementierung der Beschränkung auch explizit unter Verwendung eines Ausdrucks in einer Scriptsprache, z. B. JavaScript, erfolgen:

```
<constraint name="Radius">
  <code language="JavaScript">
    parameter["R"] > 0.0;
  </code>
</constraint>
```

Die Definition einer Restriktion folgt einem ähnlichen Prinzip. Zusätzlich zu dem Attribut name benötigt eine Restriktion ein bewertendes Attribut factor, welches nach Verletzungsgrad den Ergebniswert der jeweiligen Restriktion gewichtet. Die Implementierung geschieht analog zu den Beschränkungen unter Angabe einer Klasse oder unter Verwendung eines Ausdrucks in einer Scriptsprache:

```
<restriction name="[Name]" factor="[Wert]"
  class-name="[RestriktionKlasse]" />
```

Parameter, Beschränkungen und Restriktionen werden in der Zielfunktion benötigt, um den Zielfunktionswert zu ermitteln. Die Zielfunktion muss ebenfalls in das XML-Modell eingefügt werden. Analog zu Beschränkungen und Restriktionen erfolgt die Implementierung der Zielfunktion als Klasse oder alternativ als Angabe durch eine Scriptsprache (Beispiel: s. Anhang 11.2.1):

```
<target-function class-name="CircleTarget" name="ZfK" />
```

Ergebnisformat

Nach der Berechnung werden die Ergebnisse an das XML-Dokument in einem eigenen Container-Element angehängt.

Um die Ergebnisse nach der Optimierung zusammen mit der Startwert-Zielfunktionsauswertung zu persistieren, beinhaltet das Container-Element <calculations> die Parameterkombinationen, die vorliegen oder berechnet wurden, in dem Element <calculation>. Neben einer Beschreibung des Berechnungslaufes (description), was eine semantische Aussage zu den Werten ermöglicht, stellt das Attribut target-value den berechneten Zielfunktionswert anhand der aktuellen Parameterkombination dar und ermöglicht somit innerhalb des Container-Elements die Auswahl auf den Berechnungslauf mit dem kleinsten Wert. Eine Datumsangabe und Steuerdateninformationen sowie weitere Meta-Informationen geben dem Berechnungslauf eine zeitliche Einordnung, eine Beschreibung der verwendeten Steuerdaten zum Optimierungsverfahren und Zusatzdaten.

Im konkreten Beispiel wird als erste Berechnung (id=1) der Zielfunktionswert für den Anfangszustand (target-value=100000, description=Startwerte) und als zweite Berechnung (id=2) der Zielfunktionswert nach Optimierung

mit einem bestimmten Löser (target-value=1.3142892, description=NLPO_1) eingetragen.

```
<calculations>
  <calculation date="13.08.2015;14:45" description="Startwerte"
    id="1" target-value="100000" control-data="" meta=""/>
  <calculation date="13.08.2015;14:47" description="NLPO_1"
    id="2" target-value="1.3142892" control-data="" meta=""/>
</calculations>
```

Aufgabenstellungen können innerhalb einer formulierten Aufgabe direkte Verweise auf Dateien oder Bilder enthalten. Damit insbesondere Beschränkungen oder Restriktionen auf diese Dateien Bezug nehmen und in eine mögliche Berechnung einfließen lassen können, erlaubt das Container-Element file-references die Referenz auf Dateien:

```
<file-references>
  < file-reference name="" />
</file-references>
```

7.4 Automatisierte Vor- und Nachbereitung

Das aufgestellte XML-Format für die automatisierte Ergebnisinterpretation verfolgt den Zweck, den kompletten Problemlösungsweg von der Aufgabenstellung über die Datenextraktion, die Optimierungsrechnung, die Ergebnisentgegennahme und die semantische Rücktransformation zu begleiten. Dabei ist das XML-Format der Persistenz-Container, der alle semantischen und numerischen Werte speichert und über die verschiedenen Stationen weiterreicht.

Die Stationen, die der Container anläuft, liegen vor der Optimierung, während der Optimierung und im Anschluss an die Optimierung. Vor der Optimierung besteht die Notwendigkeit der De-Konstruktion der Informationen aus der Aufgabe. Dies ist einerseits natürlich das Durchlaufen der Transformationsphasen gemäß Abschnitt 4.4, aber weitere Vorabverarbeitungen, z. B. zu Dokumentationszwecken, können hinzukommen. Notwendig ist selbstverständlich die Verarbeitung der Daten während der eigentlichen Optimierung. Nach der Optimierung kommen alle denkbaren Arten der Weiterverarbeitung, Dokumentation und Ergebnispräsentation hinzu; die wesentlichen Verarbei-

tungsaspekte wurden bereits in Abschnitt 6.3 dargestellt.

7.4.1 Ausführende Instanzen der Verarbeitung

Offensichtlich ist, dass die eigentliche Berechnung durch den Optimierungslöser erfolgt. Es verbleiben die ausführenden Instanzen der Vorverarbeitung und der Nachverarbeitung. Diese sollen hier verallgemeinernd als

- Prä-Prozessor für die Vorverarbeitung
- Post-Prozessor für die Nachverarbeitung

aufgefasst werden. Beide Elemente können als Schnittstellen (Java-Interfaces) implementiert werden. Als Generalisierung kann man ein Interface *Processor* in Ansatz bringen.

Die ausführenden Instanzen der Vor- und Nachverarbeitung sind dann Klassen, die die Interfaces *PreProcessor* und *PostProcessor*, beide abgeleitet von *Processor*, implementieren. Methoden zum Lesen und Schreiben der Daten des Persistenz-Containers sind in *Processor* enthalten.

Alternativ können statt Java-Klassen WebServices angesprochen werden, die den jeweiligen Prozessor-Typ in Form einer Web-Schnittstelle, z. B. als REST-Service [TESW15], implementieren. Die Referenzierung erfolgt dann nicht als Angabe einer Klasse, sondern als URL.

7.4.2 Verarbeitungsmodell

Man erhält also ein Verarbeitungsmodell, das PreProcessor-Instanzen, den Löser und PostProcessor-Instanzen umfasst.

Dabei ist es für den allgemeinen Fall sinnvoll, von jeweils mehreren PreProcessor- und PostProcessor-Instanzen auszugehen. Das Processing besteht also im Durchlaufen mehrerer PreProcessoren, der Optimierungsrechnung und dem Durchlaufen mehrerer PostProcessoren (Abb. 7.2). Sowohl die PreProcessoren als auch die PostProcessoren haben dabei eine Aufrufreihenfolge.

Man benötigt also eine Steuerdatei, die die Abfolge des Processing beschreibt. Sie muss alle aufzurufenden PreProcessoren, den Optimierungslöser

und alle aufzurufenden PostProcessoren aufführen und dabei die Aufrufreihenfolge definieren.

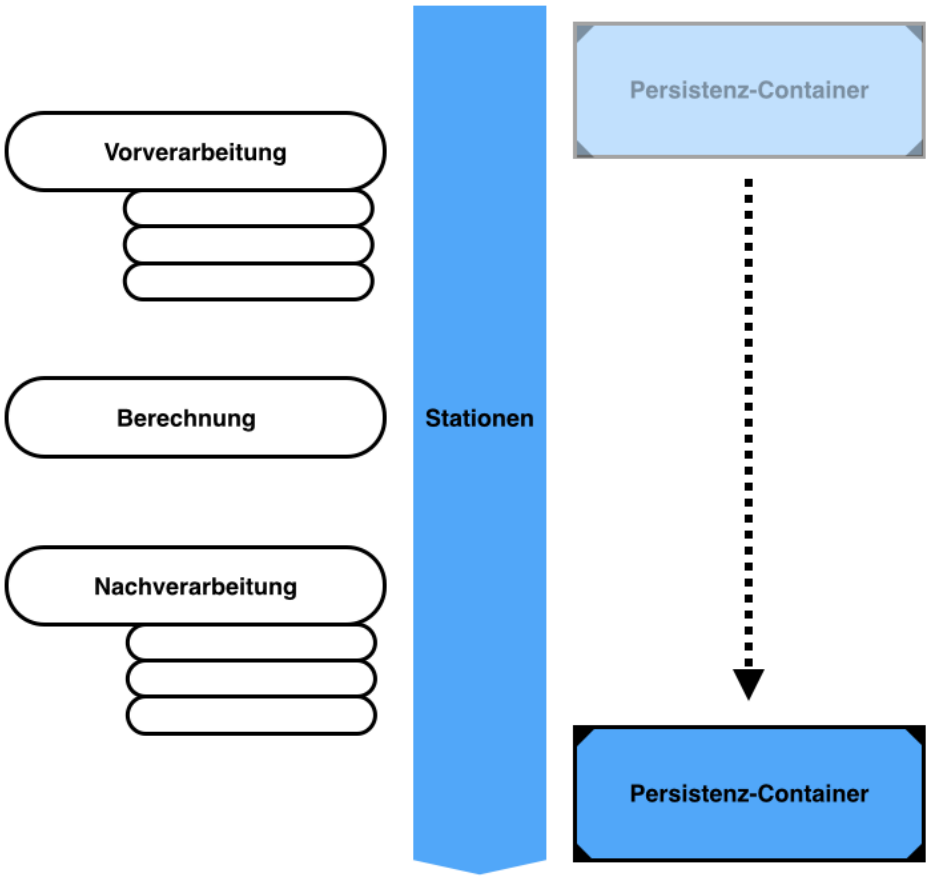


Abbildung 7.2: Stationen des Persistenz-Containers

Im Rahmen der gewählten Modellierungssprache auf XML-Basis erscheint es sinnvoll, die Steuerdatei ebenfalls in XML mit passender XML-Schema-Definition vorzuhalten. Im Anhang 11.1.2 findet sich die vollständige XML-Schema-Definition für die Prozessoren. Eine beispielhafte Steuerdatei nach diesem XML-Schema könnte folgendermaßen aufgebaut sein:

```
<processors>
  <pre-processor name="1" id="1" export-path="" >
    <pre-processor-class class-name="pre-example-class" />
  </pre-processor>
  <post-processor name="1" id="1" export-path="" >
    <post-processor-class class-name="post-example-class" />
  </post-processor>
  <post-processor name="2" id="2" export-path="" >
    <post-processor-service url="post-example-service" />
```

```
  </post-processor>
</processors>
```

Die Aufrufreihenfolge der abzuarbeitenden Prozessoren basiert auf der eingetragenen Reihenfolge.

7.4.3 Automatisierte Ergebnisinterpretation

Die Verknüpfung einer Prozessor-Ablauf-Definition mit den Daten in einem Persistenz-Container ist dann die Basis einer automatisierten Verarbeitung mithilfe eines implementierenden Prozessor-Managers.

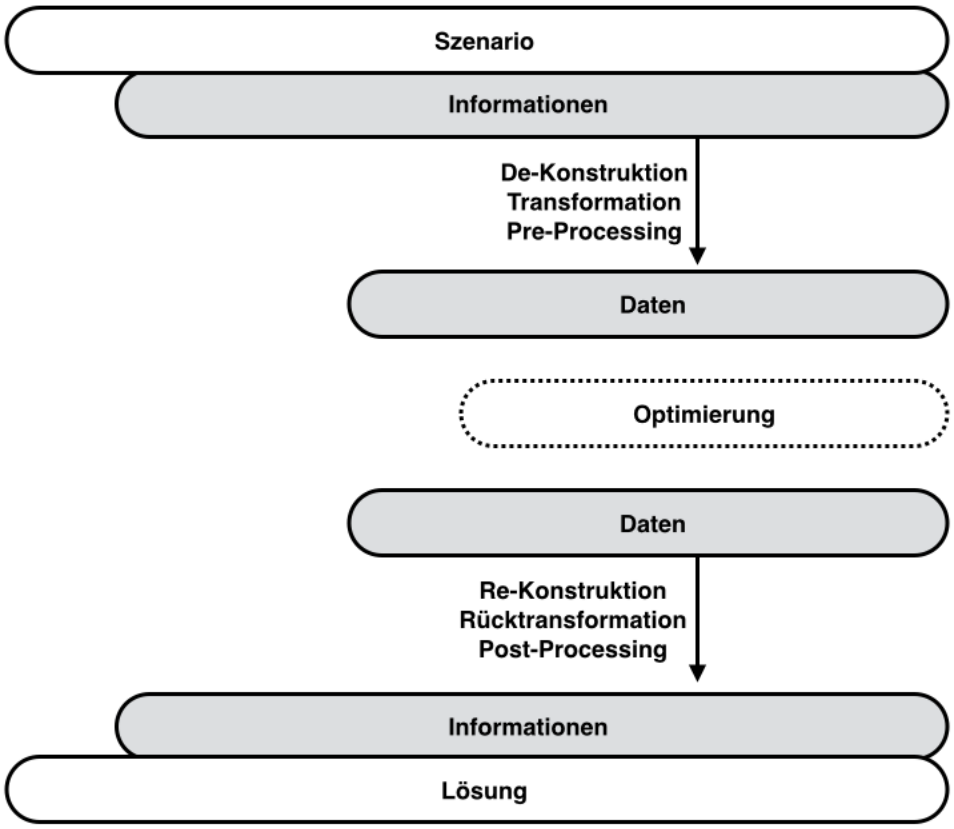


Abbildung 7.3: Automatisierte Ergebnisinterpretation

Der Prozessor-Manager verantwortet auf Grundlage der Steuerdatei den gesamten Ablauf der Lösung der Aufgabe in semantischer Hinsicht (Abb. 7.3).

7.5 Verallgemeinerung der Vor- und Nachbearbeitung

Während die Probleme der Hin- und Rücktransformation und die Erfordernisse und Vorteile einer automatisierten Ergebnisinterpretation in dieser Arbeit im Kontext der Optimierung betrachtet wurden, taucht diese Problematik in anderen, ähnlich gelagerten Kontexten ebenfalls auf. Hier sei auf Beispiele wie Finite-Elemente-Berechnungen und Big-Data-Auswertungen hingewiesen. Alle diese Problembereiche sind gekennzeichnet durch einen Ablauf, bei dem die Kriterien einer Problemstellung zu berechenbaren Daten kondensiert, eine Berechnung durchgeführt und die Ergebnisdaten wieder zu nutzbaren Lösungsaussagen rücktransformiert und interpretiert werden müssen.

Die Überlegungen zur Automatisierung der Transformation und Ergebnisinterpretation beschreiben auch in diesen verallgemeinerten Problembereichen einen effektiven Weg der Anwendung solcher Verfahren. In der Tat sind die Aspekte des Prä- und Post-Prozessings insbesondere im Rahmen von Finite-Elemente-Systemen seit langem bekannt, allerdings nicht allgemeingültig, sondern jeweils proprietär formuliert [ANS15, NAS15].

Alle hier diskutierten Modellierungsschritte und Verfahrensabläufe des Prä- und Post-Prozessings sind auf die genannten Problembereiche in gleicher oder analoger Form ebenfalls anwendbar. Das in dieser Arbeit vorgestellte Modell der Kriterientransformation und Ergebnisinterpretation ist somit für alle gleich gelagerten Fälle ebenso relevant.

Kapitel 8

Framework und Implementierung

Weil eine theoretische Modellierung erst in einer konkreten Implementierung ausgeführt werden kann und nach dem objektorientierten Ansatz dafür Basisklassen und Interfaces implementiert werden müssen, wird ein Klassenmodell aufgestellt, das die erforderlichen Basisklassen und Interfaces beinhaltet und durch eine integrierte Ablaufsteuerung die Aufgabe eines Frameworks übernimmt.

8.1 Klassenmodell der Optimierung

Im Folgenden werden die für die Optimierung relevanten Klassen des aufgestellten Frameworks vorgestellt. Die Implementierung basiert auf Java-Interfaces, die die notwendigen Methoden definieren. Wo es sinnvoll erscheint, werden abstrakte Basisklassen mit geeigneten Attributen bereitgestellt, um für die die tatsächliche Logik implementierenden Klassen Standardeigenschaften einzuführen und einheitlich zu benennen.

IParameter, Parameter, ParameterSet, Value

Gemäß Abschnitt 3.2 stellen Parameter die in der Optimierung veränderlichen Größen dar. Ein Optimierungsparameter (Klasse Parameter) implementiert das Interface IParameter; die Ableitung einer konkreten Unterklasse für Parameter ist nicht notwendig (Abb. 8.1). Der aktuelle Wert eines Parameters wird als Fließkommawert gespeichert. Alle Parameter einer Optimierung werden in einem ParameterSet zusammengefasst. Diese Zusammenfassung ist nicht zuletzt

der Tatsache geschuldet, dass in einem Schritt des Optimierungsverfahrens alle Parameter gleichzeitig variiert werden. Parameter speichern den aktuellen Wert im Sinne eines Genotyp-Werts sowie den Normierungsfaktor, der sicherstellt, dass auch bei unterschiedlichen Größenordnungen der Parameter eine gleichmäßige Variation aller Parameter erfolgen kann. Der Genotyp-Wert ist also der Parameterwert im inneren numerischen Sinne der Optimierung. Der Phänotyp-Wert, also der äußerlich sichtbare Parameterwert im Sinne der Szenario-Semantik, wird aus dem Genotyp-Wert und dem Normierungsfaktor berechnet. Weiterhin wird eine Objektreferenz als Semantik-Bezug für die spätere Rücktransformation an den Parameter angehängt.

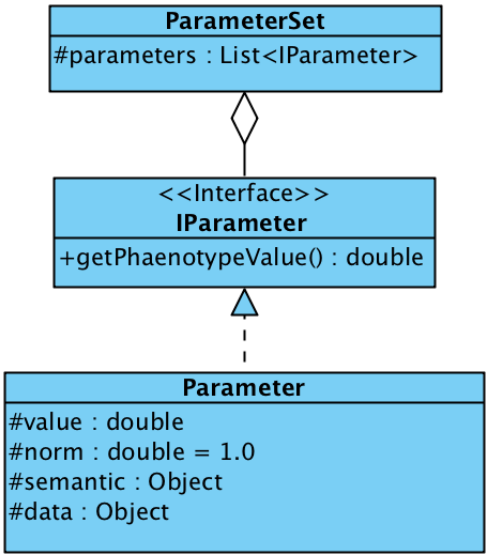


Abbildung 8.1: Parameter

IRestriction, Restriction

Gemäß Abschnitt 3.3.2 repräsentiert Restriction eine Wunschbedingung, die möglichst gut zu erfüllen ist; Abweichungen vom Wunschkriterium werden durch einen Strafwert negativ bewertet. Eine Restriction implementiert das Interface IRestriction, das eine Methode zur Strafwertberechnung bei Verletzung des betreffenden Kriteriums in Abhängigkeit des aktuellen ParameterSet vorgibt. Eine konkrete Restriction-Klasse erbt von der abstrakten Klasse Restriction und muss in erster Linie die Implementierung der Strafwertberechnungs-Methode liefern. Ein Prioritätsfaktor kann das Ergebnis entsprechend seiner Priorisierung beeinflussen; der Standardwert für diesen Faktor ist das neutra-

le Element der Multiplikation (1.0). Jedes Wunschkriterium muss als eigene Restriction-Klasse implementiert werden.

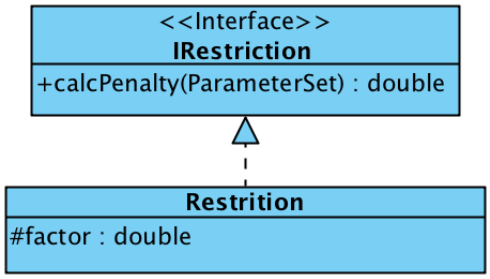


Abbildung 8.2: Restriction

IConstraint, Constraint

Gemäß Abschnitt 3.3.1 liefert die Klasse Constraint mit der Methode isValid (ParameterSet) eine Aussage über die Gültigkeit des aktuellen ParameterSets. Darüber hinaus hat die Schnittstelle IConstraint einen Methodenrumpf für eine Methode, die optional einen ungültigen ParameterSet in einen gültigen ParameterSet überführen kann. Eine konkrete Unterklasse von Constraint muss entsprechend der Spezialisierung mindestens die Methode isValid (ParameterSet) implementieren. Jedes Musskriterium muss als eigene Constraint-Klasse implementiert werden.

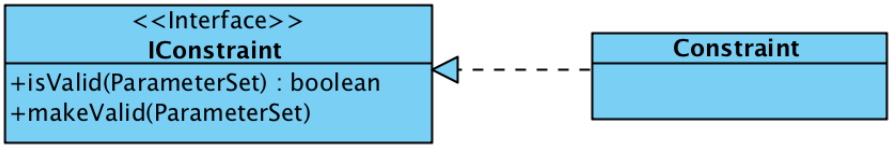


Abbildung 8.3: Constraint

ITarget

Der Darstellung in Abschnitt 3.4.2 folgend, wird die Zielfunktion der Optimierung über das Interface ITarget realisiert, welches zwei Methoden vorgibt. Die wichtigste Methode ist die Berechnung eines Zielfunktionswerts in Abhängigkeit von den aktuellen Parameterwerten; da die Strafwerte der Restrictions auf den Zielfunktionswert superponiert werden, werden die Restrictions ebenfalls dieser Methode übergeben, der Ergebniswert ist der strafwertbeaufschlagte Gesamt-Zielfunktionswert, da dieser in der Optimierung für die Gütebewertung der aktuellen Lösung herangezogen wird. ITarget definiert zusätzlich eine

Methode, die den zuletzt berechneten Zielfunktionswert ohne Neuberechnung zurückliefert. Die konkrete Zielfunktion muss in jedem Einzelfall als eigene ITarget-Klasse implementiert werden.

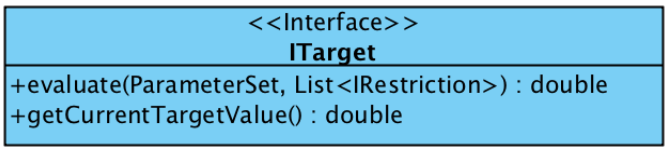


Abbildung 8.4: ITarget

IOptimizer, Optimizer, BasicOptimizer

Für die konkrete Realisierung von Optimierungsverfahren dient das Interface IOptimizer und die abstrakte Klasse Optimizer, die den Ablauf eines Optimierungsverfahrens folgendermaßen strukturiert abbilden:

- Die Initialisierung des Optimierungsverfahrens mit den Parametern, Beschränkungen, Restriktionen und der Zielfunktion wird durch die Methode initialize (ParameterSet, List<IConstraint> List<IRestriction>, ITarget) vorgegeben.
- Zum Starten der Optimierung dient die run()-Methode, die das Ergebnis in Form einer Instanz der Klasse Result zurückliefert.
- Innerhalb der Methode run() wird durch eine Schablonen-Methode evaluate(ParameterSet, List<IRestriction>, ITarget) auf die Zielfunktion zugegriffen und der Zielfunktionswert in verfahrensspezifischer Art ausgewertet und verarbeitet.

Weitere Methoden können Informationen zu Ablaufdetails liefern, z. B. ein permanentes Reporting der Schrittanzahl und des bisher erreichten Gütewertes. Erweiternd kann das Optimierungsverfahren statt von Optimizer von der Unterklasse BasicOptimizer abgeleitet werden; ein BasicOptimizer fügt Infrastrukturmethoden und Metadaten insbesondere für die Zeitmessung und das Sammeln von auflaufenden Meldungen zum Verfahrensablauf hinzu.

Ergänzend kann eine IOptimizer-Klasse als Fassade für das Ansprechen externer Löser mit Hilfe von algebraischen Modellierungssprachen implementiert werden. Die run-Methode deligiert dann die eigentliche Optimierung an den externen Löser.

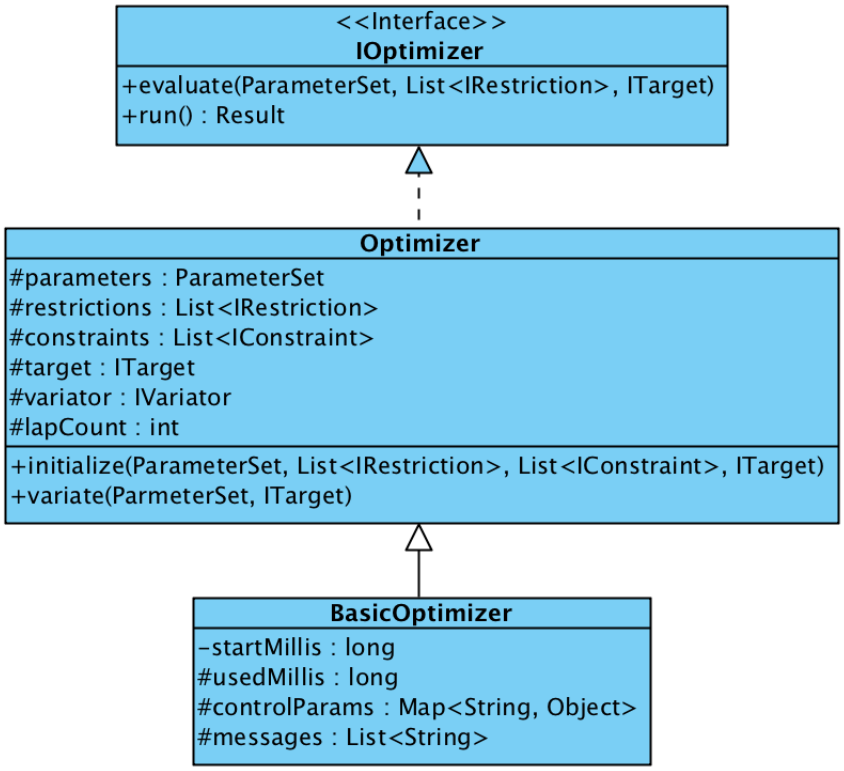


Abbildung 8.5: Optimizer

IVariator

Die konkrete Implementierung eines Optimierungsverfahrens bedient sich innerhalb der von Optimizer abgeleiteten Klasse eines Objekts, das für die verfahrensspezifische Variation der Parameter verantwortlich ist. Gemäß Abschnitt 3.5 definiert das Interface IVariator die Methoden zur Steuerung der Schrittweite und der Modifikation der Parameterwerte in jedem Einzelschritt der Optimierung.

Im Fall der Verwendung eines externen Lösers entfällt die Notwendigkeit, einen IVariator zu implementieren, da dieser Teil des externen Lösers ist.

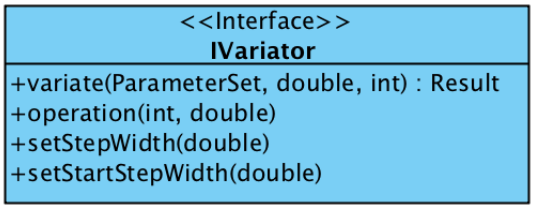


Abbildung 8.6: IVariator

Result

Bei erkannter Konvergenz liefert das implementierte Optimierungsverfahren sein Ergebnis in Form einer Instanz der Klasse Result zurück. Das Ergebnis besteht aus dem final erreichten Zielfunktionswert (als Fließkommawert), den Parametern (in Form eines ParameterSet mit den endgültigen Parameterwerten) und eine Referenz auf den benutzten Optimizer (als Instanz von IOptimizer). Im Fall des externen Lösers wird das Ergebnis von dem externen Löser zurückgeliefert und in die Result-Instanz überführt.

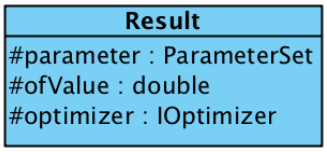


Abbildung 8.7: Result

ITask, Task

Die abstrakte Klasse Task implementiert ITask und damit die Methode solve(), die den Optimierungsvorgang startet und ein Result zurückliefert. Die konkrete Klasse GeneralizedTask ist von Task abgeleitet; sie wickelt ein standardisiertes Vorgehen bei der Optimierung ab und fügt Infrastrukturelemente für die Dokumentation der Optimierung hinzu.

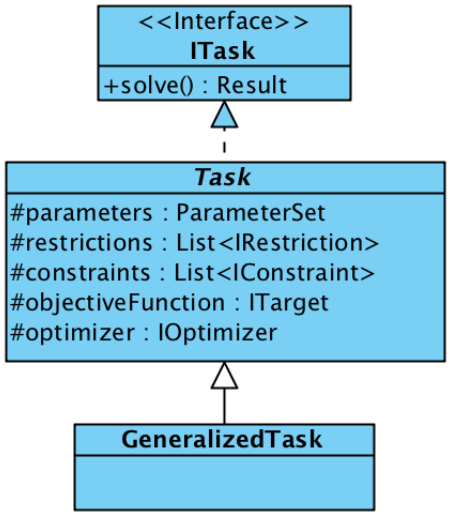


Abbildung 8.8: Task

Der Gesamtzusammenhang aller Klassen, die ein optimierungsfähiges System bilden, ist in Abb. 8.9 dargestellt. Das zusammenfassende Objekt ist in

der Klasse Task repräsentiert. Eine Task hält Referenzen auf alle Constraints und Restrictions, auf den ParameterSet, die Zielfunktion ITarget und den eigentlichen Optimierungsalgorithmus IOptimizer. Der ParameterSet enthält alle variierbaren Parameter mit ihren Startwerten. Der IOptimizer wird über die Task gestartet und liefert im Anschluss an die Optimierung seine Ergebnisse in Form einer Result-Instanz.

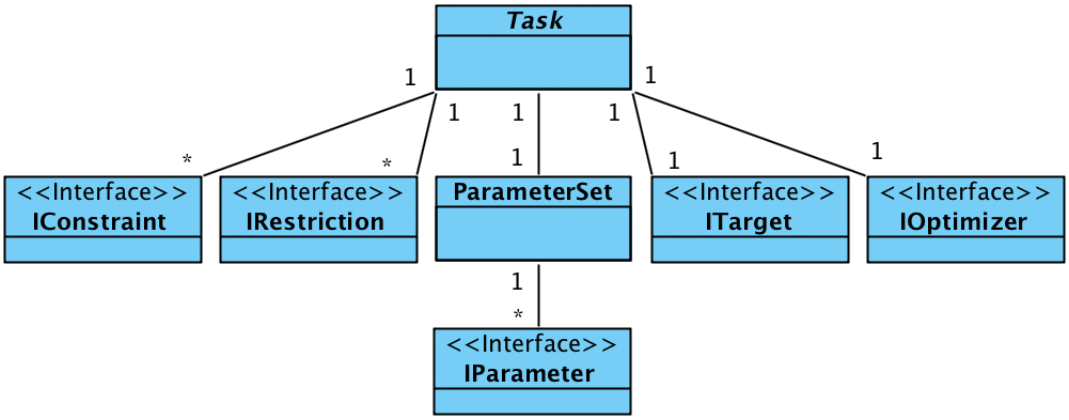


Abbildung 8.9: Task und ihre Bestandteile

8.2 Klassenmodell der Transformation

Die Transformation des Szenarios in eine Konzentrat-Datenstruktur gemäß dem Phasenmodell in Abschnitt 4.4 bildet die im Szenario auffindbaren Kriterien auf Datenelemente ab, die allerdings nicht unmittelbar in eine Optimierung einfließen können. Dies ist in der Tatsache begründet, dass in der Transformation mit dem Durchlaufen der verschiedenen Phasen zahlreiche semantikgetriebene Umwandlungen stattfinden, die zunächst nicht an einem Optimierungsalgorithmus orientiert sind, sondern informationsbezogen und kommunikationsorientiert die Kriterien des Szenarios in überprüfbare, identifizierbare und mit dem Szenario abgleichbare und dem Auftraggeber diskutierbare Bestandteile zerlegen.

Praktisch geschieht dies durch Bedienung eines Editors, in dem das Szenario in Textform, alle Kriterien, implementierende Klassen, Bedingungen und Zahlenwerte erfasst und persistent gemacht werden. In der vorliegenden Form

verwaltet der Editor ein TaskModel und erzeugt eine XML-Datei entsprechend der Ontologie der Interpretation nach Abschnitt 6.5, die alle vorbereiteten Elemente einer berechenbaren Task enthält. Die Implementierung erfolgt mit Co-Klassen zu jeder der in Abschnitt 8.1 vorgestellten Berechnungsklassen; die Benennung erfolgt mit dem Namenszusatz ...Model (z. B. Parameter → ParameterModel).

Für die tatsächliche Berechnung werden die erfassten Elemente in Berechnungsklassen überführt. Dazu existiert die Klasse Realizer. Nach der Berechnung werden die Ergebnisse von der Algorithmus-Ebene wieder auf die Modell-Ebene zurücktransformiert (Klasse Unrealizer). Es existiert damit eine klare Trennung zwischen Transformationsmodell und Berechnungsmodell.

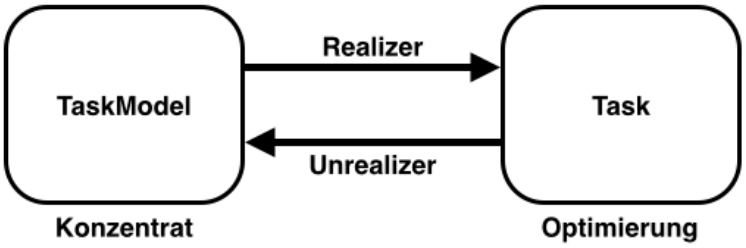


Abbildung 8.10: Realizer und Unrealizer

Das TaskModel beschreibt also die aus dem Szenario entnehmbare Aufgabenstellung in editierbarer und persistierbarer Form, während in einer Task die Algorithmus-orientierten Strukturen vorliegen. Das TaskModel ist also die Datenrepräsentation des Szenarios und des strukturierten Transformationsergebnisses als Konzentrat in die XML-Struktur der Ontologie der Interpretation nach Abschnitt 6.5.

TaskModel

Die zentrale Klasse im Kontext der Transformation ist das TaskModel. Es ist der Container für die Modelle aller Parameter, Restriktionen, Beschränkungen, die Zielfunktionsdefinition und für die Aufnahme von Berechnungsergebnissen (Abb. 8.11). Auch die ursprüngliche Aufgabenstellung wird als Szenario-Text im TaskModel abgelegt. Parameter, Restriktionen, Beschränkungen und Zielfunktion werden beim Start der Optimierung vom Realizer in Berechnungsdaten gemäß Abschnitt 8.1 überführt. Die Berechnungsergebnisse werden nach

der Optimierung vom Unrealizer in das TaskModel zurückgeführt.

Die textliche Beschreibung des Szenarios wird als Zeichenkette (story) mit Kommentarmöglichkeit (comment) seitens des Initiators oder Bearbeiters unter einem eindeutigen Namen (taskName) persistiert. Die elementaren Ergebnisse nach Durchlaufen der Transformationsphasen werden in den Listen für ParameterModel, RestrictionModel und ConstraintModel abgelegt, für die Zielfunktion wird ein vollqualifizierter Klassenname als ITargetFunctionModel angegeben (Abb 8.11).

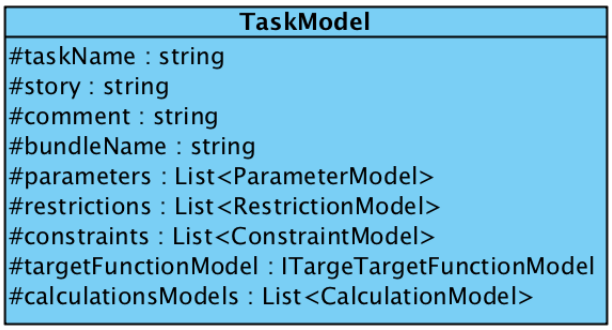


Abbildung 8.11: TaskModel

ParameterModel, ParameterResultModel

Instanzen der Klasse ParameterModel repräsentieren die identifizierten veränderlichen Größen des Szenarios. Ein ParameterModel speichert den Parameter-Wert als nicht-normalisierten, phänotypischen Zahlenwert. Dieser Wert entstammt unmittelbar dem Szenario. Während der Normalisierungsphase werden alle identifizierten Parameter miteinander in Beziehung gesetzt und daraus ein Normalisierungsfaktor für jeden Parameter hergeleitet. Zusätzlich zu dem Wert wird der Semantik-Bezug des Parameters zusammen mit der Maßeinheit des Parameterwertes im ParameterModel gespeichert, um im Anschluss an eine Berechnung die Ergebniswerte kontextbezogen darstellen zu können. Nach einer Berechnung wird der Wert aus der Optimierung durch eine Klasse ParameterResultModel persistiert und entsprechend als Instanz in einer Liste innerhalb von ParameterModel geführt. Somit wird ermöglicht, dass für einen Parameter mehrere Ergebniswerte gespeichert und ausgewertet werden können. Das ID-Feld in ParameterResultModel referenziert den generierenden Optimierungslauf (CalculationModel).

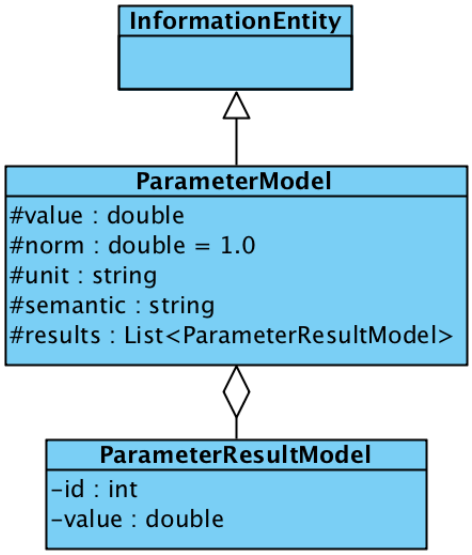


Abbildung 8.12: ParameterModel

RestrictionModel

RestrictionModel ist die Repräsentation einer Wunsch-Bedingung aus dem Szenario. Die Implementierung einer mathematischen Straffunktion, deren Wirkungsweise auf der im Szenario beschriebenen Wunsch-Bedingung basiert, liefert einen Strafwert, der je nach Priorisierung der Restriktion mit einem vorgegebenen Faktor versehen wird und Einfluss auf den Zielfunktionswert nimmt. Für ein RestrictionModel muss eine konkrete, das Interface IRestriction implementierende Klasse in Form eines vollqualifizierten Klassennamens angegeben werden.

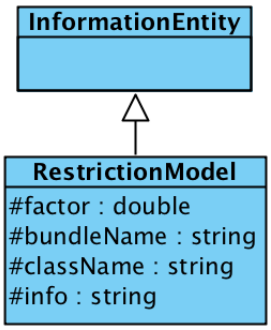


Abbildung 8.13: RestrictionModel

ConstraintModel

Die Klasse ConstraintModel setzt eine Muss-Bedingung des Szenarios um. Sie prüft alle aktuellen Parameterwerte im Sinne der Beschränkung auf Gültigkeit.

Für ein ConstraintModel muss eine konkrete, das Interface IConstraint implementierende Klasse in Form eines vollqualifizierten Klassennamens angegeben werden.

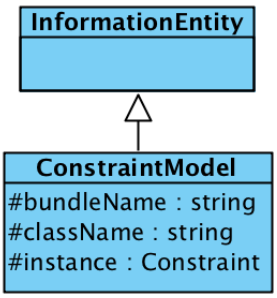


Abbildung 8.14: ConstraintModel

TargetFunctionModel

Das TargetFunctionModel erfordert analog zum RestrictionModel und ConstraintModel die Angabe einer das Interface ITarget implementierenden Klasse über ihren vollqualifizierten Klassennamen.

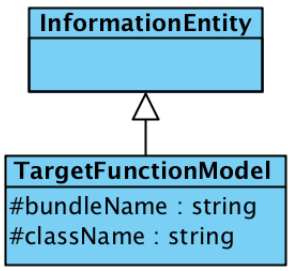


Abbildung 8.15: TargetFunctionModel

CalculationModel

Die Klasse CalculationModel ermöglicht als Referenzklasse auf Modellseite den Zugriff auf die in einer konkreten Optimierung berechneten Ergebnisse. Über eine Referenz-ID besteht eine eindeutige Assoziation zu den Parameterwerten in ParameterResultModel, die in ParameterModel als Liste geführt werden. Zusätzlich speichert das CalculationModel das Datum der Optimierung und den Zielfunktionswert, um einen historischen Verlauf bei mehreren Ergebnissen zu erhalten.

Als Gesamtzusammenhang und in Erweiterung von Abb. 8.11 zeigt Abb. 8.17 die Klasse TaskModel in Abhängigkeit zu ihren Bestandteilen. Die Klas-

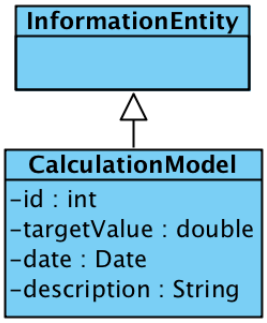


Abbildung 8.16: CalculationModel

se bedient auf Optimierungsseite die Klasse Task mit dem Konzentrat und speichert die von der Klasse Task erhaltenen Ergebnisse in den Datenstrukturen ParameterResultModel und CalculationModel, die beide über eine eindeutige ID referenziert werden.

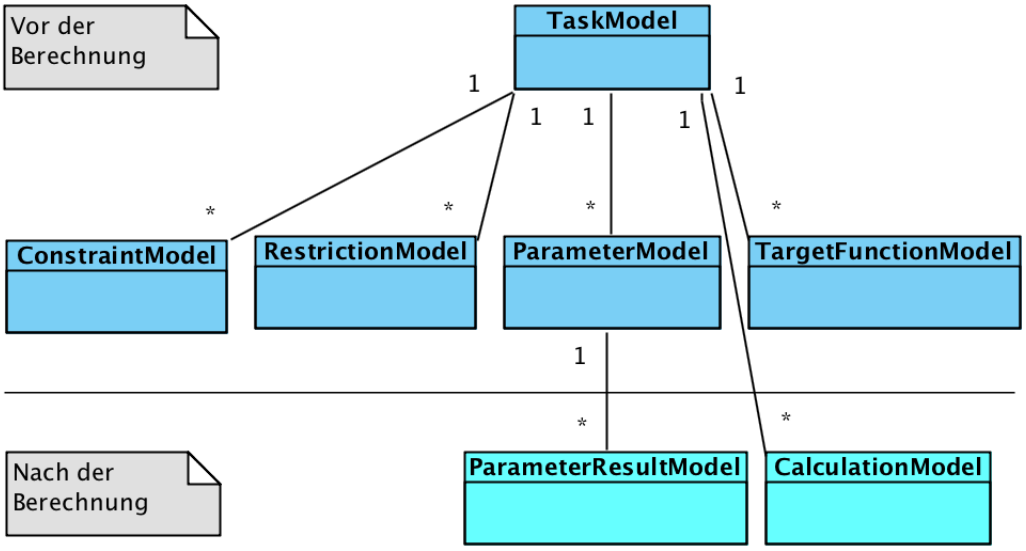


Abbildung 8.17: TaskModel und seine Bestandteile

8.3 Klassenmodell des Pre-Post-Processing

Vor einer Optimierung kann es sich als sinnvoll herausstellen, vorab die Aufgabe einem Prä-Prozessor für Dokumentationszwecke zuzuführen.

Das Ergebnis eines Optimierungslaufs ist eine Liste von reinen Zahlenwerten. Dies sind die durch die Optimierung ermittelten Endwerte der Parameter. Der sich daran anschließende Schritt ist die Interpretation der Zahlenwerte im

Sinne der Aufgabenstellung. Im Kontext der hier diskutierten Kriterientransformation ist dies die Rücktransformation der Ergebnisse und die Wiederanreicherung der Ergebnisdaten mit Semantikinformationen gemäß Kap. 4 und Kap. 6.

Während die reine Rückführung der Ergebnis-Parameterwerte von der Task-Ebene auf die TaskModel-Ebene durch den Unrealiser erledigt wird, können die Ergebnisse darüber hinaus in einem problemspezifischen Kontext weiterverarbeitet und ausgewertet werden. Dies ist die Aufgabe der Post-Processing-Komponente (Abb. 8.18).

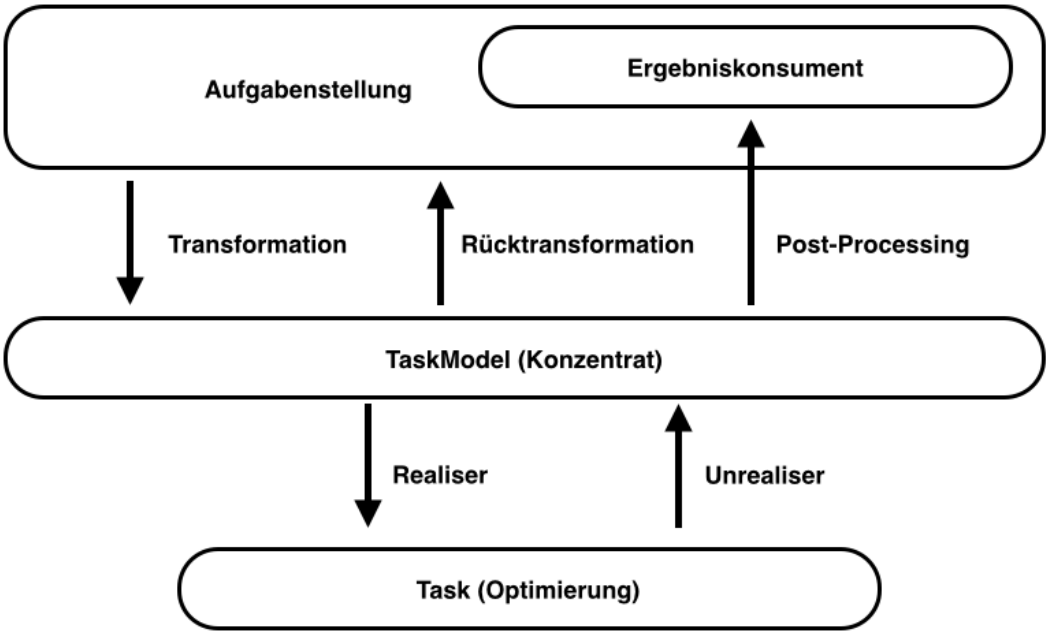


Abbildung 8.18: Gesamtübersicht

IPrePostProcessor

Das Interface IPrePostProcessor führt eine Methode postProcess() ein, über die eine Nachverarbeitung der Ergebnisdaten ermöglicht wird. Die Art der Nachverarbeitung ist an dieser Stelle noch nicht definiert und wird erst von einer das Interface implementierenden Klasse bestimmt. Hier sind alle Arten von Weiterverarbeitung der Ergebnisdaten denkbar, insbesondere auch Verarbeitungen hinsichtlich Vergleich und Gütebewertung mit dem Ursprungszustand vor der Optimierung. Um diesen Vergleich zu ermöglichen, führt das Inter-

face auch die Methode preProcess() ein, die eine beliebige Vorabbearbeitung der Eingangsdaten ermöglicht. Ein PrePostProcessor kann also sowohl eine Vorverarbeitung (PreProcessing) als auch eine Nachverarbeitung (PostProcessing) durchführen. Da bei einer solchen Verarbeitung generell Ergebnisse in Form von Dateien entstehen können, ermöglicht die dritte Methode des Interface mit getProcessedFiles() einen Zugriff auf während des Pre- oder PostProcessing entstehende Dateien.

TaskModelPrePostProcessor

Die allgemeine abstrakte Klasse TaskModelPrePostProcessor liefert Zugriffsmöglichkeiten auf die Ergebnismenge eines wählbaren Parameters aus einem TaskModel.

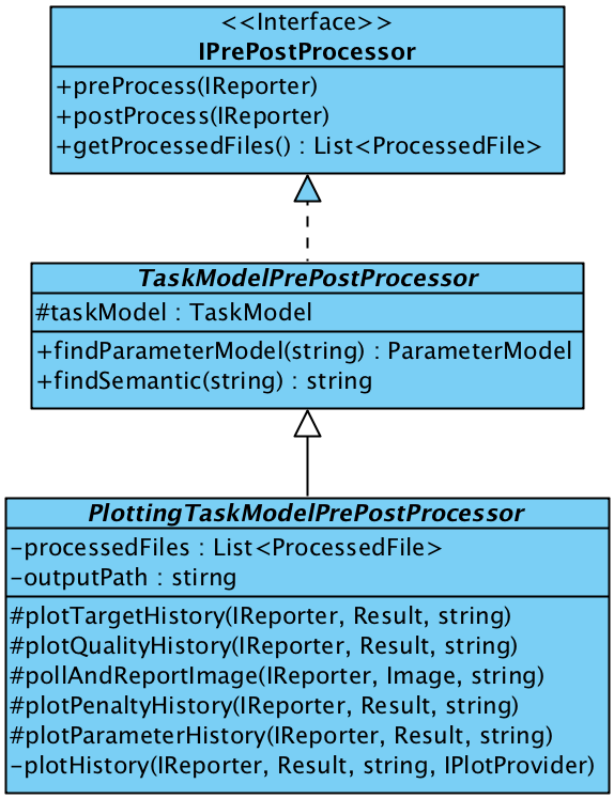


Abbildung 8.19: PrePostProcessor

PlottingTaskModelPrePostProcessor

Die von TaskModelPrePostProcessor abgeleitete abstrakte Klasse PlottingTaskModelPrePostProcessor fügt Fähigkeiten hinzu, Ergebnisdaten in grafische Auswer-

tungen zu überführen. Beispielhaft werden in den Abbildungen 8.21(a) bis 8.21(d) die Ergebnisdaten in jedem Iterationsschritt der Optimierung, sowohl der Parameter (a) als auch der Zielfunktionsauswertung, getrennt nach reinem Gütewert (c), Straffunktionswert (b) und superponierter Zielfunktion (d) als Funktionsgraph geplottet.

Ein konkreter PostProcessor wird für jedes TaskModel erzeugt und beschreibt in einer konkreten Klasse eine angepasste Ausgabe und Formatierung in spezieller Anlehnung an die Aufgabenbeschreibung. Diese konkrete Klasse erbt die Standard-Methoden der Klasse PlottingTaskModelPrePostProcessor und kann um weitere Spezialisierungen erweitert werden.

Die Aufbereitung und grafische Darstellung der Ergebnisse geschieht auf Modellseite (Abschnitt 8.2). Aus einer Tabelle lassen sich chronologisch (Datum und Uhrzeit) sortiert die Ergebnisse nach Zielfunktionswerten und verwendetem Optimierungsverfahren auswählen und Parameter sortiert nebeneinander zur Anzeige bringen.

Konstruktionsdarstellungen sind als grafische Darstellung im Bereich von konstruktiven Optimierungsaufgaben sinnvoll. Ein Beispiel zeigt Abb. 8.20: hier werden die Lenkerpositionen und die Grenzwerte des Hakenwegs für eine Optimierung der Lemniskaten-Aufgabe (Abschnitt 3.8.9) skizziert. Die konkrete Aufgabenstellung definiert, welche grafischen Ausgaben sinnnvoll sind und welche Aussagen damit getroffen werden.

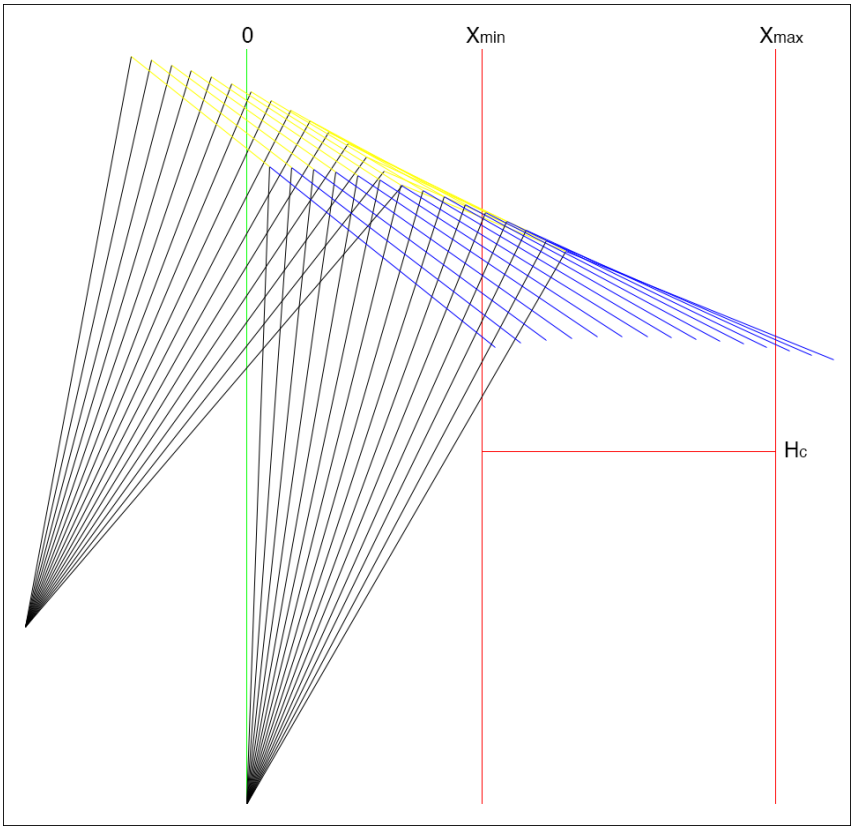
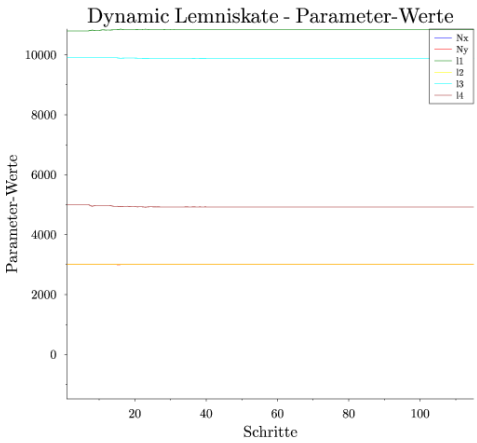
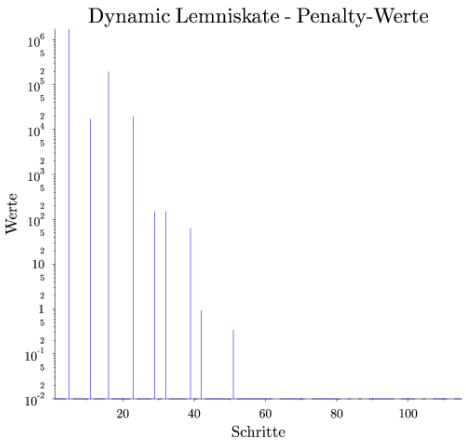


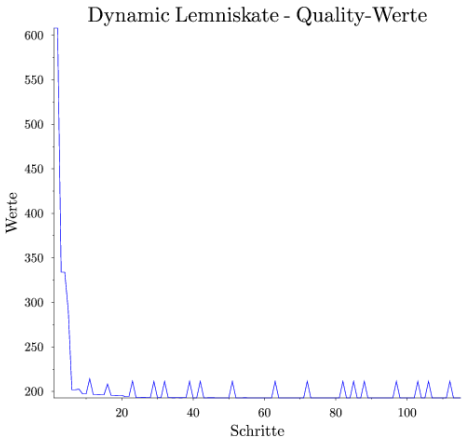
Abbildung 8.20: Beispiel eines Kinematik-Plots: Lenker-Positionen für diskrete Wippwinkel-Werte und resultierender Wippweg für einen optimierten Lemniskatenkran (vergleiche Abschnitte 3.8.9 und 9.7)



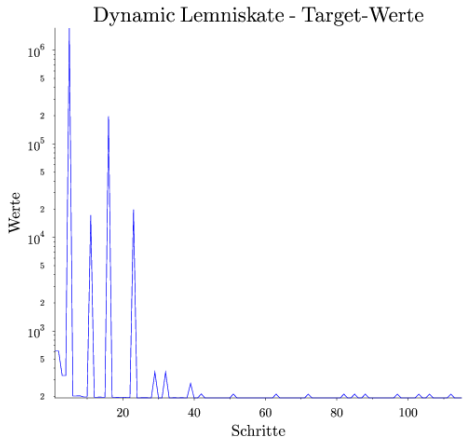
(a) Parameter-Werte



(b) Straf-Werte



(c) Güte-Werte



(d) Zielfunktions-Werte

Abbildung 8.21: Beispiel für grafische Darstellungen

8.4 Steuerkomponente des Pre- und Post-Processing

Der ProcessorManager verwaltet die der Aufgabe zugewiesenen Prozessoren. Die Verwaltung der Prozessoren erfolgt in zwei Listen: Der Liste für Prä-Prozessoren und der Liste für Post-Prozessoren. Die Prozessoren werden in der Reihenfolge ausgeführt, in der sie in der Liste definiert wurden. Falls ein Prozess nicht erfolgreich ausgeführt wurde, beendet der ProcessorManager die Steuerung und alle anschließenden Prozessoren kommen nicht mehr zur Ausführung.

Der Persistenz-Container wird von dem ProcessorManager an die Prozessoren weitergereicht, um dorthin für die jeweilige Ergebnisdarstellung Informationen und Semantik liefern zu können.

Die Festschreibung der Prozessoren kann beliebig erfolgen. Im Rahmen dieser Arbeit in Abschnitt 7.4.1 bereits erwähnt, liefert hier exemplarisch eine Java-Klasse die ergebnisdarstellende Implementierung.

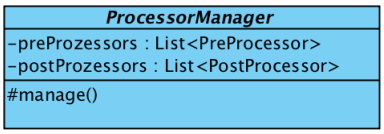


Abbildung 8.22: ProcessorManager

Kapitel 9

Anwendungsbeispiele

Weil die Einsatzmöglichkeiten der Kriterien-Transformation und der semantischen Ergebnisinterpretation zahlreich sind, soll anhand konkreter Beispiele die Implementierung von Prä- und Post-Prozessoren im Kontext spezieller Anwendungsfälle gezeigt werden.

In Kap. 3 wurden typische Aufgabenstellungen der Optimierung vorgestellt. Diese sollen nun mithilfe der erarbeiteten Prinzipien der Kriterientransformation, Vorverarbeitung, Optimierung und Nachverarbeitung sowie unter Einbeziehung der Semantik-Abstraktion und Semantik-Wiederanreicherung beispielhaft gelöst werden. Das Hauptaugenmerk liegt dabei nicht auf den Details der Optimierung selbst oder auf den konkreten Ergebnissen, sondern in der Anwendung der genannten Transformations- und Verarbeitungsverfahren.

Die relevanten Ergebnisse der verschiedenen Optimierungsläufe werden jeweils dokumentiert und diskutiert, ohne alle Details und Steuerparameter aufzuführen. Details sind im Anhang dokumentiert.

Es wird angenommen, dass in allen Optimierungsbeispielen mindestens ein einfacher Standard-Post-Prozessor zum Einsatz kommt, der jeweils die numerischen Ergebniswerte ausgibt. Die Ergebnis-Zahlenwerte werden von dem Standard-Post-Prozessor mit voller numerischer Genauigkeit, also mit allen errechneten Nachkommastellen ausgegeben. Es sei darauf hingewiesen, dass für eine praktische Verwendung der Zahlenwerte diese Ergebnisgenauigkeit eher als fiktiv, jedenfalls als rein numerisch bedingt, anzusehen ist und im konkreten Einzelfall zu entscheiden ist, mit welcher Genauigkeit die Ergebniswerte tatsächlich genutzt werden können. Die Notwendigkeit, Zahlenwerte im Sinne

der Aufgaben-Semantik geeignet zu runden und zu interpretieren, fällt in den Verantwortungsbereich der angewendeten Post-Prozessoren.

Anhand von einfachen Testbeispielen wird zunächst die prinzipielle Funktionsfähigkeit und Anwendbarkeit des Optimierungsprinzips aufgezeigt. Jedes der Beispiele begründet sich mit jeweils einem spezifischen Teilaspekt der Optimierungsrechnung oder der Semantik-Transformation. Im abschließenden Abschnitt 9.7 wird dann ein praxisorientiertes Optimierungsproblem vollständig unter Anwendung der in dieser Arbeit diskutierten Vorgehensweisen aufbereitet und durchlaufen.

9.1 Rosenbrock-Testfunktion

Die in Abschnitt 3.8.1 beschriebene Rosenbrock-Testfunktion erfüllt für die in dieser Arbeit dargestellten Transformations- und Optimierungsverfahren den Zweck, den Optimierungskern und die Übergabe-Schnittstellen auf prinzipielle Funktionsfähigkeit hin zu überprüfen.

Die Rosenbrock-Testfunktion hat ein bekanntes Ergebnis mit einem Optimum an der Stelle

$$\begin{aligned} x &= 1 \\ y &= 1 \end{aligned}$$

mit einem Funktionswert

$$f(1;1) = 0$$

Ausgehend von einem gewähltem Startpunkt

$$\begin{aligned} x &= -2 \\ y &= 2 \end{aligned}$$

und unter Verwendung des in Kap. 2 beschriebenen Optimierungsverfahrens von Hooke und Jeeves berechnet die Optimierung folgende Ergebniswerte:

```
READY (Optimum found)
steps = 65
best Zfk = 5.458064883761188E-7
  x = 1.0007362365722656
  y = 1.001479148864746
```

Die zu minimierende Zielfunktion ist definiert als die absolute Abweichung vom bekannten Funktionswert 0 am Ort des Optimums. Nach 65 Schritten (im Sinne von Parameter-Variationen und Zielfunktionsauswertungen) hat das Hooke-Jeeves-Optimierungsverfahren den angestrebten Optimalpunkt mit einer maximalen Abweichung von ca. 0,001 erreicht. Der Zielfunktionswert beträgt $5,458 \cdot 10^{-7}$. Bei einer engeren Wahl der Abbruchbedingung wären – allerdings nach einer höheren Schrittanzahl – noch höhere Genauigkeiten im Ergebnis erreichbar. Das Ziel dieses ersten Tests, das Optimierungsverfahren und seine Schnittstellen zu verifizieren, wurde damit erfüllt.

Den Verlauf der Parameter- und Zielfunktionswerte über den Optimierungsfortschritt zeigen die Abbildungen 9.1 und 9.2. Die Formulierung mithilfe des in Kap. 8 beschriebenen Klassenmodells wird im Anhang 11.3.1 dokumentiert.

Obwohl dieses Beispiel nur dem Test der eigentlichen Optimierung dienen soll, ist die Ergebnisauswertung in Form der beiden Plots aus den Abbildungen 9.1 und 9.2 bereits eine Demonstration eines grafischen Post-Prozessings.

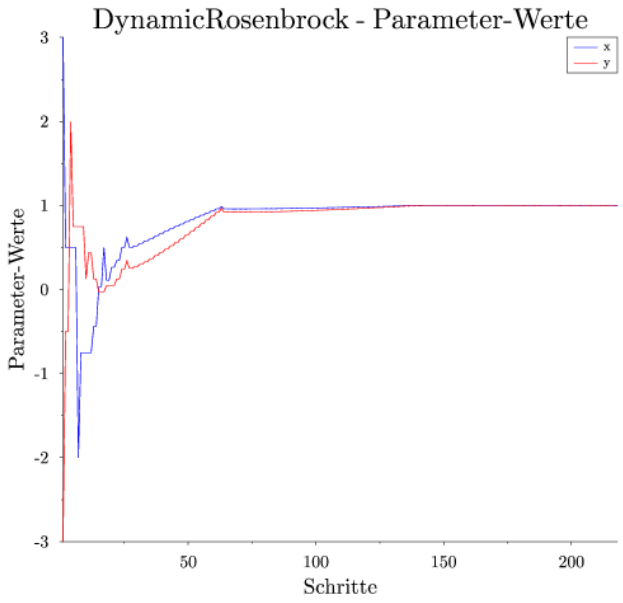


Abbildung 9.1: Parameter-Werte

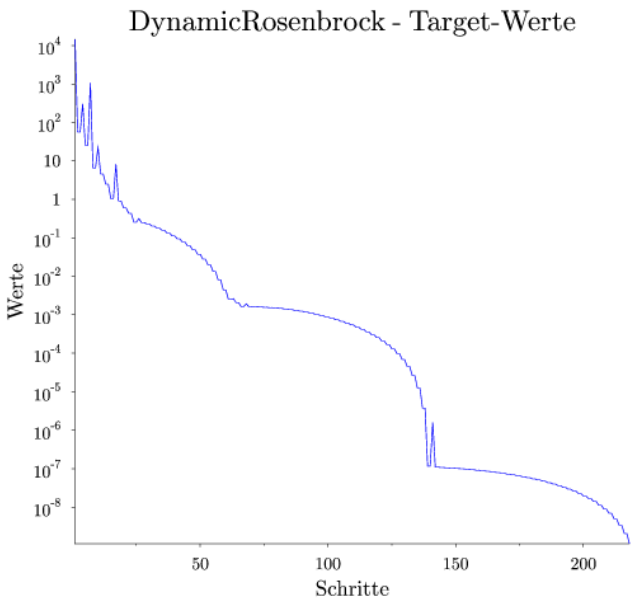


Abbildung 9.2: Zielfunktionswert

9.2 Kreis-Aufgabe

Die in Abschnitt 3.8.2 beschriebene Kreis-Aufgabe wurde in Abschnitt 7.2 bereits mit existierenden Modellierungssprachen (GAMS, AMPL) formuliert und gelöst. In Abschnitt 7.3.1 diente die Aufgabe als Beispiel für die Problemformulierung mit dem in dieser Arbeit aufgestellten XML-Format. Es sei an dieser Stelle noch einmal aufgegriffen, um das Format im Gesamtkontext einer kompletten Optimierungsrechnung anzuwenden. Angesichts der Offensichtlichkeit der Aufgabenstellung und ihrer Bestandteile wird hier noch keine Semantik-Transformation durchlaufen, sondern die Parameter, Restriktionen, Beschränkungen und die Zielfunktion sowie Semantiktexte werden manuell in das XML-Format eingetragen.

In der Aufgabe sind zwei Punkte gegeben:

$$Punkt_1 : (5; 15)$$

$$Punkt_2 : (0; 0)$$

Aus der Kreissymmetrie ergibt sich evident folgender dritter Punkt:

$$Punkt_3 : (5; -15)$$

Gesucht wird der Mittelpunkt und der Radius des Kreises, auf dem die genannten Punkte liegen. Es handelt sich also um ein Drei-Parameter-Problem mit den Parametern M_x, M_y und R.

Als Startwerte werden willkürlich gewählt:

$$M_x = 20$$

$$M_y = 0$$

$$R = 15$$

Das Problem lässt sich ohne Optimierung geschlossen durch Anwendung des Satzes von Thales lösen. Die so ermittelbaren exakten Ergebniswerte sind:

$$M_x = 25$$

$$M_y = 0$$

$$R = 25$$

Für die Lösung mithilfe von Optimierung wird als Zielfunktion die Summe der Abstandsquadrate der Punkte von der Kreislinie des aktuellen Kreises berechnet und minimiert. Die Berechnung erfolgt unter Anwendung des Verfahrens von Hooke und Jeeves. Die Problemformulierung zeigt der folgende XML-Quelltext:

```
<?xml version="1.0" encoding="UTF-8" ?>
<task name="DynamicCircle">
  <story>
    Kreisproblem: gegeben sei ein Halbkreis mit unbekanntem
        Durchmesser mit zwei ausgezeichneten orthogonalen Strecken.
        Wie groß ist der Radius?
  </story>
  <comment/>
    Die geschlossene Lösung führt unter Verwendung des Satzes von
        Thales schnell auf den gesuchten Radius des Halbkreises.
        Als Optimierungsproblem aufgefasst, läuft die
        Parametervariation über drei Parameter: der Radius sowie
        die Position des Mittelpunktes (x- und y-Koordinate).
  </comment>
  <parameters>
    <parameter key="x" name="x" norm="1.0" semantic="x-Wert"
```

```
      semantic-unit="mm" />
      <value id="1" value="20.0"/>
    </parameter>
    <parameter key="y" name="y" norm="1.0" semantic="y-Wert"
      semantic-unit="mm" />
      <value id="1" value="0.0"/>
    </parameter>
    <parameter key="R" name="R" norm="1.0" semantic="Radius"
      semantic-unit="mm" />
      <value id="1" value="15.0"/>
    </parameter>
  </parameters>
  <constraints>
    <constraint class-name="RangeConstraint" name="Radius">
      <field name="Minimum" type="min" value="0.0"/>
      <field name="Parameter" type="ref" key-ref="R"/>
    </constraint>
  </constraints>
  <restrictions/>
  <target-function bundle-name="" class-name="CircleTarget"
    name="Zfk"/>
</task>
```

Das Optimierungsverfahren berechnet:

```
READY (Optimum found)
steps = 90
best Zfk = 0.0
  x = 25
  y = 0
  R = 25
```

und trägt die Start- und Ergebniswerte in den <parameters>- und <calculations>-Abschnitt der XML-Beschreibung ein:

```
<parameters>
  <parameter key="x" name="x" norm="1.0" semantic="x-Wert"
    semantic-unit="mm" />
    <value id="1" value="20.0"/>
    <result id="2" value="25.0"/>
  </parameter>
  <parameter key="y" name="y" norm="1.0" semantic="y-Wert"
    semantic-unit="mm" />
```

```
    <value id="1" value="0.0"/>
    <result id="2" value="0.0"/>
  </parameter>
  <parameter key="R" name="R" norm="1.0" semantic="Radius"
    semantic-unit="mm" />
    <value id="1" value="15.0"/>
    <result id="2" value="25.0"/>
  </parameter>
</parameters>
...
<calculations>
  <calculation date="13.08.2015;15:19" description="Startwerte"
    id="1" target-value="100" control-data="" meta=""/>
  <calculation date="13.08.2015;15:22" description="HookeJeeves"
    id="2" target-value="0.0" control-data="" meta=""/>
</calculations>
```

Nach 90 Schritten wird das angestrebte Ergebnis mit vollständiger Genauigkeit erreicht. Als Anwendungsbeispiel für ein grafisches Post-Prozessing werden die Verläufe der Parameterwerte und der Zielfunktionswerte, aufgetragen über die Iterationsschritte, in Abbildung 9.3 und 9.4 dargestellt. Obwohl diese Abbildungen nur ein Plot-Beispiel im Rahmen eines Post-Prozessing sein sollen, kann man am Verlauf der Zielfunktionswerte die iterative Erreichung des Optimums nachvollziehen.

Implementierungsdetails befinden sich im Anhang 11.3.2.

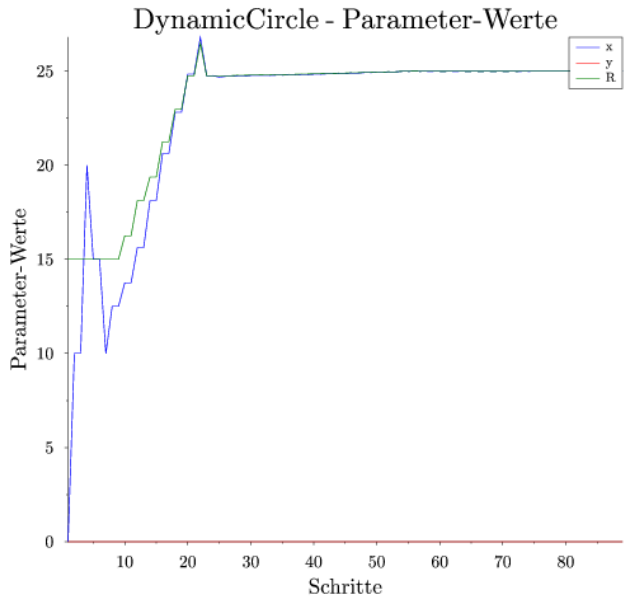


Abbildung 9.3: Parameter-Werte

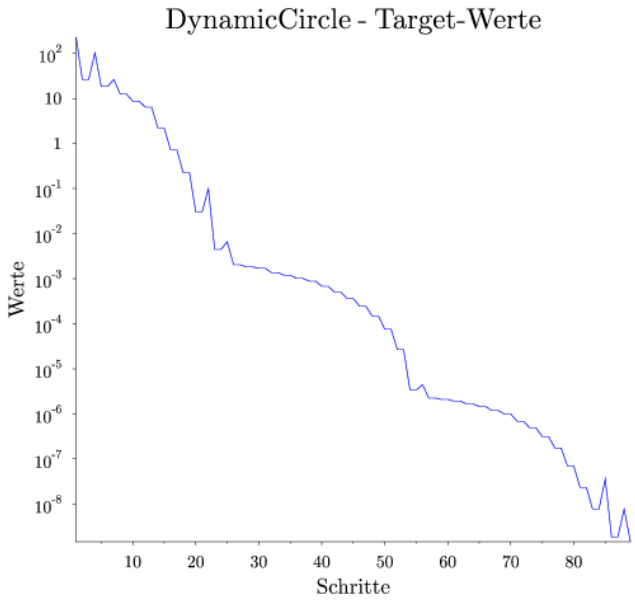


Abbildung 9.4: Zielfunktionswert

9.3 Leiter-Aufgabe

Die in Abschnitt 3.8.3 eingeführte Aufgabenstellung beschreibt ein einparametriges Problem, das hier mithilfe von Optimierung gelöst wird: Eine Leiter mit gegebener Länge 7 m soll an einer Wand aufgestellt werden, an der sich unten vor dieser Wand eine quadratische Box mit gegebenener Seitenlänge 1 m befindet. Gefragt ist die maximale Anstellhöhe der Leiter. Der intuitive Ansatz wäre, die Anstellhöhe als variierbaren Parameter in der Optimierung anzusetzen. Wählt man hingegen den Anstellwinkel der Leiter als variierbaren Parameter, verspricht dies wesentliche Vorteile in der Formulierung der Zielfunktion.

Die Variation erfolgt in der Optimierung hier also zunächst nicht über den in der Aufgabenstellung angefragten Wert, sondern über einen anderen, leichter zu implementierenden Parameter, der die Formulierung der Zielfunktion stark vereinfacht. Dies ist ein typisches Beispiel für die Umformulierung der Kriterien der Aufgabe im Sinne einer leichten numerischen Berechenbarkeit. Diese Umformulierung findet während der Transformation statt und muss dokumentiert, also im Rahmen der Semantikabstraktion persistiert werden. Für die anschließende Ergebnisdarstellung muss hingegen sichergestellt sein, dass die ursprüngliche Fragestellung der Aufgabe beantwortet wird. Dies wird durch eine Rücktransformation in einem Post-Prozessor durchgeführt, der in den Optimierungsablauf eingebunden wird und von dem Anstellwinkel auf die Anstellhöhe zurückrechnet.

Als Startwert wurde der Anstellwinkel willkürlich gewählt zu:

$$\alpha_A = 50,0^\circ$$

Nach Ausführung des Hooke-Jeeves-Optimierungsverfahrens ergibt sich ein Anstellwinkel von

$$\alpha_A = 80,3828681295272^\circ$$

und daraus folgend eine Anstellhöhe an der Wand von

$$h_A = 6,901622895164711 \text{ m}$$

Nach der Berechnung wird in einer Nachverarbeitung der berechnete Zahlenwert in den ursprünglich geforderten semantischen Ergebniswert zurückgerechnet. Diese Umrechung erfolgt in einem eingebundenen Post-Prozessor.

Weitere Post-Prozessoren erzeugen die schon aus den vorhergegangenen Beispielen bekannten Verlaufsplots.

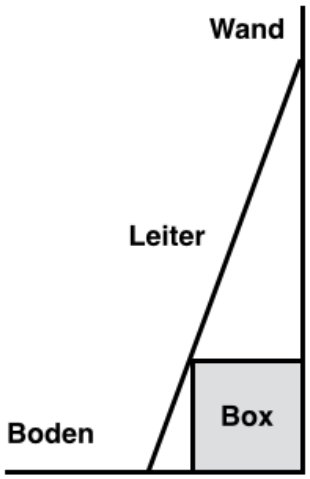


Abbildung 9.5: Leiter mit Hindernis

Das Ergebnis wurde in 55 Iterationsschritten errechnet. Implementierungsdetails sind im Anhang 11.3.3 dokumentiert.

```
READY (Optimum found)
steps = 55
best Zfk = 6.790878970264203E-10
  Winkel = 80.3828681295272°
  Anstellhöhe = 6.901622895164711 m
```

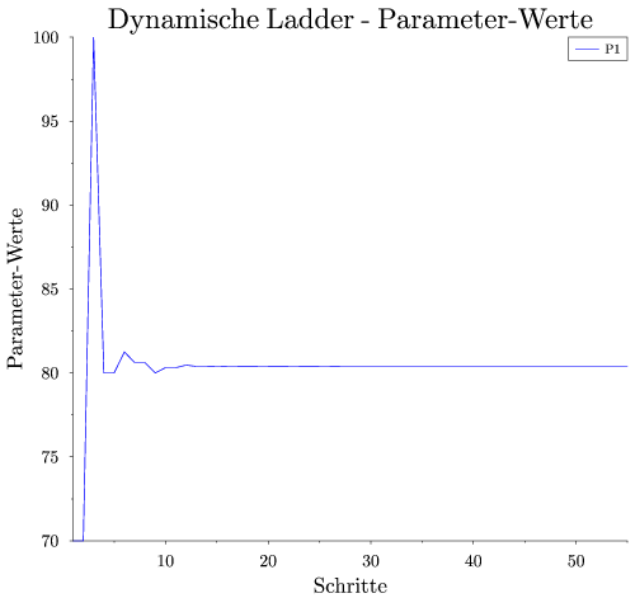


Abbildung 9.6: Anstellwinkel

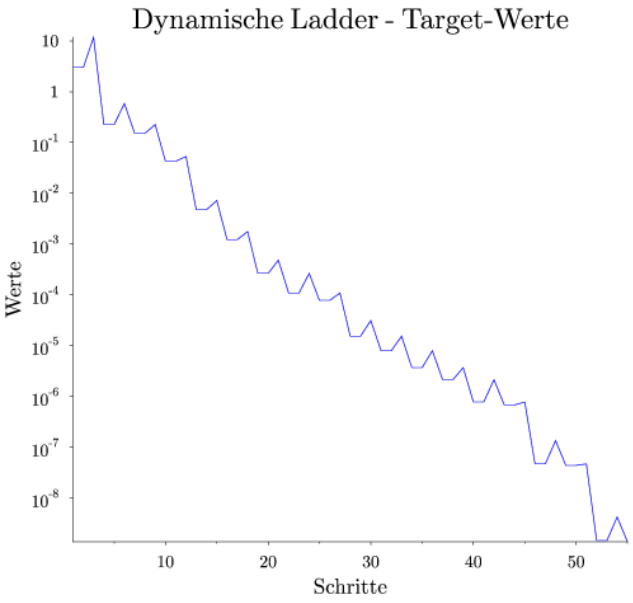


Abbildung 9.7: Zielfunktionswert

9.4 Schaf-Aufgabe

Die in Abschnitt 3.8.4 beschriebene Schaf-Aufgabe beinhaltet genau einen variierbaren Parameter, der unmittelbar als Seillänge für das Schaf aus der Aufgabenstellung hervorgeht. Bei einer gegebenen kreisrunden Weidefläche mit einem Radius von 50 m lässt sich abschätzen, dass die Hälfte der abzugrasenden Fläche eine Seillänge von mehr als 50 m erfordert.

Aus der Aufgabenstellung wäre die gefragte Seillänge das zu ermittelnde Kriterium, jedoch lässt sich hier die Zielfunktion dadurch vereinfachen, dass man einen alternativen Zwischenwert variiert. Die Berechnung der Seillänge basiert im Inneren der Zielfunktion auf der Addition von zwei Kreissegmenten, die an einer Linie aneinanderliegen. Die Entfernung vom Mittelpunkt zu dieser Linie wird als der variierbare Parameter gewählt, der zu optimieren ist.

Auch hier folgt aus der Transformation der Kriterien, dass nicht der ursprünglich offensichtliche Wert als Parameter eingesetzt werden sollte, sondern daraus eine alternative Größe abgeleitet wird. Wiederum muss diese Tatsache in der Vorverarbeitung erkannt und persistiert werden, um in der Ergebnisnachverarbeitung wieder rücktransformierbar zu sein.

Das Hooke-Jeeves-Optimierungsverfahren liefert:

Distanz zum Mittelpunkt: 16,433708145050332 m

Dieser Wert wird in der Nachverarbeitung geeignet auf die Seillänge zurückgerechnet.

```
READY (Optimum found)
steps = 47
best Zfk = 3.5921402741223574E-8
  Parameter = 16.433708145050332 m
  Seillänge = 57.936423651231434 m
```

An diesem Beispiel ist zusätzlich zu erkennen, dass die Ausgabe der Seillänge mit 15 Nachkommastellen in der Maßeinheit [m] nicht praxisgerecht ist. In konstruktiven Optimierungsaufgaben tritt das Problem häufig auf, dass ein numerisches Ergebnis in der Praxis nicht mit der angegebenen Genauigkeit herzustellen oder einzuhalten ist. Es wird also eine geeignete Rundung des numerischen Ergebniswerts auf einen praxisgerechten Wert erforderlich. Dies

ist ein Beispiel für die Einbringung eines weiteren Post-Prozessors: der Post-Prozessor für die Umrechnung des Variationsparameters in die Seillänge und der Post-Prozessor für die Rundung werden sequenziell ausgeführt. Im vorliegenden Fall ergibt sich eine Seillänge von ca. 57,94 m für das Schaf.

9.5 Time Tabling

Time Tabling-Aufgabenstellungen gemäß Abschnitt 3.8.5 stellen typische Multi-Kriterien-Probleme dar. Für jedes zu planende Ereignis in einem Time Tabling-Szenario wird ein Parameter benötigt, der die Startzeit des Ereignisses repräsentiert. Zu jedem Ereignis gibt es Zusatzinformationen, die Berücksichtigung finden müssen; dies sind im wesentlichen die Ereignisdauer und die Raumbelegung. Diese semantischen Informationen sind in der Transformationsphase zu erkennen und in der Vorverarbeitung zu persistieren.

Weiterhin existieren Beschränkungen der Form

- „Wochenenden müssen frei bleiben“
- „Ereignisse dürfen nur zwischen 8 und 18 Uhr stattfinden“

und Restriktionen der Art

- „Folgende Ereignisse dürfen nicht zur gleichen Zeit stattfinden oder überlappen“
- „Folgende Ereignisse sollen einen Abstand von mindestens x Tagen einhalten“

Insbesondere die Restriktionen beschreiben Beziehungen zwischen jeweils zwei Ereignissen E_i und E_j, sodass ein mittlerer Aufwand der Ordnungsklasse $\mathcal{O}(n^2)$ bei der Behandlung der Restriktionen zu erwarten ist.

Time Tabling ist somit ein Beispiel für eine multikriterielle Problemstellung [JTR+14] mit expliziten Beziehungen zwischen Parametern. Der folgende Quelltextausschnitt zeigt die Implementierung dieser Aspekte in XML: Ein Szenario mit zwei exemplarischen Ereignissen, die Definition einer Beschränkung für das Freihalten von Wochenenden für beide Ereignisse sowie eine Restriktion, die die Überschneidung der beiden Ereignisse mit einer Gewichtung von 500 vermeiden soll.

```
<parameters>
  <parameter key="E1" name="Ereignis_1" norm="1.0"
    semantic="Ereignis_Name" semantic-unit="hh:mm" />
  </parameter>
  <parameter key="E2" name="Ereignis_2" norm="1.0"
    semantic="Ereignis_Name" semantic-unit="hh:mm" />
  </parameter>
  ...
</parameters>
<constraints>
  <constraint class-name="WeekendConstraint" name="Wochenenden">
    <field name="Parameter" type="ref" key-ref="E1"/>
    <field name="Parameter" type="ref" key-ref="E2"/>
  </constraint>
  ...
</constraints>
<restrictions>
  <restriction class-name="NoCollisionRestriction" factor="500.0"
    name="Kollisionsvermeidung">
    <field name="Parameter" type="ref" key-ref="E1"/>
    <field name="Parameter" type="ref" key-ref="E2"/>
  </restriction>
  ...
</restrictions>
```

Die Optimierung eines solchen Szenarios wirft als Ergebnis die numerischen Werte der errechneten Startzeiten aller Ereignisse aus. Dies ist zunächst eine nur bedingt nützliche Reihe von Zahlen.

Um diese Ergebniswerte im Sinne der Aufgabenstellung nutzbar zu machen, müssen die Zahlenwerte mindestens in einer strukturierten Tabellenform ausgegeben werden, wobei die Rechenwerte der Ereignis-Startzeiten in Datums- und Zeitwerte umgerechnet und formatiert werden müssen. Weiterhin bietet es sich an, die Ergebniswerte für den Export in Kalenderformate aufzubereiten, damit sie in gängigen Kalenderdarstellungen angezeigt werden können. Dies ist Aufgabe geeigneter Post-Prozessoren. Die Darstellung in einem Kalender zeigt beispielhaft Abb. 9.8. Anhang 11.3.5 zeigt ausschnittweise den Quellcode einer möglichen Implementierung.

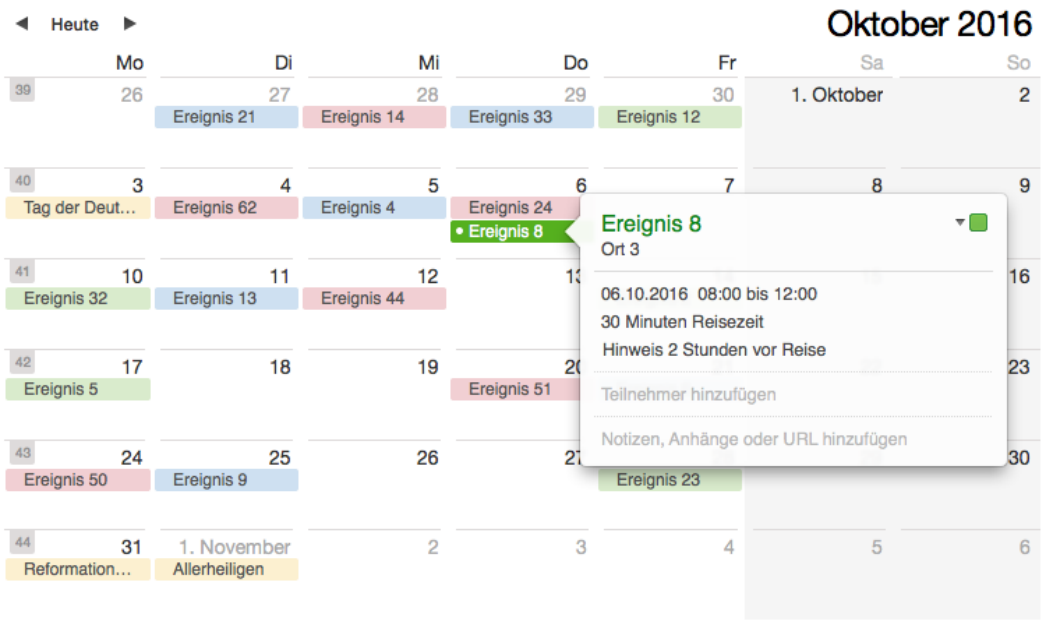


Abbildung 9.8: Post-Prozessor-Import in einen Kalender

9.6 Pattern-Matching

Die Mustererkennung gemäß Abschnitt 3.8.8 zielt auf die Erkennung und Positionsbestimmung von farbigen Formobjekten in einer Menge beliebiger anderer Formobjekte [JM13]; die Objekte können sich überdecken. In Abb. 9.9 ist ein solches Szenario exemplarisch gegeben. Hier sei das grüne Quadrat aufzufinden.

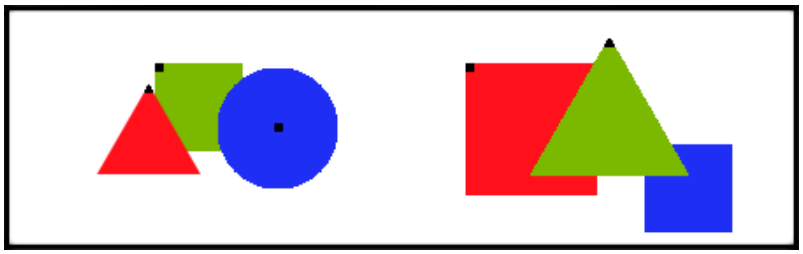


Abbildung 9.9: Parameter gleicher Dimension in der Mustererkennung

„Erkennen“ bedeutet, dass herausgefunden werden soll, ob die gesuchte Form überhaupt in der Objektmenge enthalten ist. „Positionsbestimmung“ bedeutet die Lokalisierung des erkannten Objekts in der zweidimensionalen Objektebene.

Die Identifikationsphase der Kriterientransformation findet als variierbare

Parameter die zweidimensionalen Koordinaten $(x; y)$ des gesuchten Objekts. Die Aufgabe wird dadurch durch Optimierung lösbar, dass man eine Zielfunktion und eine Variation formuliert, die ein Testobjekt über die Bildfläche wandern lässt und den Grad der Überdeckung mit dem gesuchten Muster (grünes Quadrat) als Zielwert definiert. Das Muster wurde erkannt, wenn die Zielfunktion einen bestimmten Grenzwert über- oder unterschreitet. Die Position des Musters wurde erkannt, wenn der Grad der Überdeckung maximal wird.

Die Transformation muss also neben der Interpretation der Eckkoordinaten des Objekts $(x; y)$ die Objektgröße, seine Farbe sowie den Erkennungs-Grenzwert identifizieren und persistieren. Vom Original abweichende Farbwerte können akzeptiert werden, wenn man die Farberkennung unscharf als gewichtete Restriktion (mithilfe des Gewichtungsfaktors der Straffunktion) formuliert.

Dieses multikriterielle Szenario beleuchtet den Aspekt von Optimierungsparametern, die aus unterschiedlichen Wertedomänen (Koordinaten vs. Farben) stammen. Ein weiterer Aspekt ist hier die denkbare Erweiterung der betrachteten Kriterienmenge, wenn man z. B. eine Rotation oder Stauchung des Musters zulässt [JLB+14, JTL+15].

Im konkreten Beispiel hat das gesuchte grüne Quadrat einen RGB-Wert von (123;184;0). Das Suchmuster hat einen reinen Grünwert (0;255;0). Als Startwert kann sinnvollerweise der Mittelpunkt des Suchbilds angenommen werden.

Die Aufgabe wurde mithilfe des Evolutionsverfahrens durch Optimierung gelöst. Das Verfahren konvergiert auf das gesuchte Muster, auch wenn es teilweise (jedoch nicht vollständig!) von anderen Objekten verdeckt ist.

Die Optimierung liefert für das Beispiel folgende Ausgabe:

```
READY (Optimum found)
steps = 24
best Zfk = 317.0
  X-Koords = 88 Pixel
  Y-Koords = 35 Pixel
```

Das Zahlenwert-Ergebnis muss in der Nachverarbeitungsphase anzeigen, ob das Suchmuster erkannt werden konnte, sowie seine Lokalisierung in Pixel-Koordinaten des Bildes ausgeben. Der Zielfunktionswert kann für eine Interpretation der Güte oder Sicherheit der Erkennung herangezogen werden; hierbei

kann unterschieden werden zwischen der Sicherheit der Formerkennung (beeinflusst durch Überdeckungen oder Formabweichungen) und der Sicherheit der Farberkennung.

Ein weiterer Post-Prozessor kann das aufgefundene Muster in ein Freistellungsbild überführen und ausgeben (Abb. 9.10).

Abbildung 9.10: Parameter gleicher Dimension in der Mustererkennung

9.7 Lemniskatenkran

Abschließend soll die in Abschnitt 3.8.9 beschriebene Aufgabenstellung zur Wippweg-Optimierung eines Lemniskatenkrans als vollständig ausgeführtes Anwendungsbeispiel für die in dieser Arbeit diskutierten Verfahrensweisen dienen. Als Optimierungsverfahren wird der Algorithmus von Hooke-Jeeves angewendet, wobei die Startwerte der gegebenen Konstruktion entnommen werden.

Transformationsphase 1: Identifikation

Aus der Aufgabenstellung lassen sich die variierbaren Parameter $\overline{p} = [p_1, \cdots, p_6]$ wie folgt (mit den Bezeichnungen gemäß Abb. 3.11) identifizieren:

$\overline{p}$	Name	Phänotypischer Startwert $\overline{p}_0$	Einheit	Semantik $\overline{s}$
p_1	N_x	-4000	mm	Stützpunkt-Koord-X
p_2	N_y	3000	mm	Stützpunkt-Koord-Y
p_3	l_1	10800	mm	GrundlenkerLänge
p_4	l_2	3000	mm	linkes oberes Teilsegment
p_5	l_3	9900	mm	Zuglenker
p_6	l_4	5000	mm	rechtes oberes Teilsegment

Tabelle 9.1: Parameter

Bezugnehmend auf Abb. 3.11 soll anhand eines gegebenen Wippweg-Intervalls [4000; 9000] die Lemniskatenkurve einen möglichst horizontalen Verlauf nehmen. Der minimale Hakenraum unter Punkt C darf 6000 mm (bezogen auf den Anlenkpunkt M) nicht unterschreiten.

Das kinematische System wird folgenden Restriktionen $\overline{r} = [r_1, r_2]$ für den Wippweg unterworfen: Für den geforderten minimalen Hakenraum, der nicht

$\overline{r}$	Name	Minimum	Maximum	Straffaktor F_{r_k}
r_1	X_{min}	4000		1000
r_2	X_{max}		9000	1000

Tabelle 9.2: Restriktionen

unterschritten werden darf, wird das System einer Zwangsbedingung in Form einer Beschränkung $\overline{b} = [b_1]$ unterworfen:

$\overline{b}$	Name	Minimum
b_1	H_C	6000

Tabelle 9.3: Beschränkung

Transformationsphase 2: Normalisierung

Alle Parameter werden auf den größten vorkommenden phänotypischen Wert (Parameter l_1) normiert. Daraus ergeben sich folgende Normalisierungsfaktoren:

$\overline{F}_N$	Name	Norm
$F_{N,1}$	N_x	2,7
$F_{N,2}$	N_y	3,6
$F_{N,3}$	l_1	1,0
$F_{N,4}$	l_2	3,6
$F_{N,5}$	l_3	1,090909
$F_{N,6}$	l_4	2,16

Tabelle 9.4: Normalisierungsfaktoren

Transformationsphase 3: Reduktion

Die Überprüfung der Parameter und Beschränkungen ergibt, dass keine Redundanzen oder Irrelevanzen in der Aufgabe auftreten.

Transformationsphase 4: Plausibilität

Die Überprüfung der Formulierung der Aufgabenstellung ergibt, dass keine Widersprüche in der Aufgabe auftreten.

Transformationsphase 5: Lösbarkeit

Die Lösbarkeitsphase ist erfolgreich durchlaufen, wenn die Beschränkungen das Zielgebiet nicht verschwinden lassen. In der vorliegenden Aufgabe wird als Beschränkung die Zwangsbedingung eingeführt, dass der Hakenraum unter Punkt C einen bestimmten Minimalwert nicht unterschreiten darf.

Ohne die kinematische Berechnung hier im Einzelnen darzustellen, ist zu beachten, dass diese Beschränkung für alle momentanen Punkte C_i über dem gesamten Wippweg gelten muss; die Beschränkung muss entsprechend implementiert sein. Unter Anwendung dieser Beschränkung verschwindet das Zielgebiet nicht, sodass Plausibilität gegeben ist.

Transformationsphase 6: Zielfunktion

Die Lenker müssen so dimensioniert werden, dass die Kinematik des Viergelenk-Getriebes den Punkt C vertikal nur minimal bewegt. Die Zielfunktion wird so implementiert, dass sie unter Anwendung der kinematischen Beziehungen im Viergelenkgetriebe aus den aktuellen Konstruktionsparametern die über dem relevanten Wippweg auftretende maximale Höhendifferenz $\Delta_{H,max}$ untersucht (Anhang 11.3.6); sie wird so formuliert, dass sie die Summe der Abstandsquadrate von Punkten auf dem Wippweg berechnet. Die Berechnung erfolgt an diskreten Punkten, die sich kinematisch aus diskreten Werten des Winkels ϕ_{20} ergeben.

Transformationsphase 7: Implementierung

Das Optimierungsverfahren variiert die Lenkerlängen und berechnet die auftretenden Höhendifferenzen $\Delta_{H,i}$ über dem Wippweg. Dazu wird in jedem Ite-

rationsschritt der Wippweg überstrichen, indem der Grundlenkerwinkel ϕ_{20} zwischen zwei Grenzwerten durchlaufen wird.

Die Quelltexte der implementierten Klassen sind im Anhang 11.3.6 dokumentiert.

$f(\overline{p})$	Name	Implementierungsklasse
$f_{b_1}(\overline{p})$	Höhen-Beschränkung	de.buw.lfa.combo.lemniskate.HeightConstraint
$f_{r_1}(\overline{p})$	Xmax-Restriktion	de.buw.lfa.combo.lemniskate.XMaxRestriction
$f_{r_2}(\overline{p})$	Xmin-Restriktion	de.buw.lfa.combo.lemniskate.XMinRestriction
$f_Z(\overline{p})$	Zielfunktion	de.buw.lfa.combo.lemniskate.LemniskateTarget

Tabelle 9.5: Implementierungsklassen

Optimierungsrechnung

Im Anschluss an die sieben Phasen der Transformation liegen die Daten für die Optimierungsrechnung geeignet vor. Die Optimierung wird mit dem Verfahren von Hooke-Jeeves durchgeführt. Als Anfangswert für die Variationsbreite wird 2 % vom maximal auftretenden Parameterwert angesetzt. Gemäß dem Hooke-Jeeves-Verfahren wird diese Variationsbreite im Laufe des Verfahrens schrittweise bis zu einer Abbruchgrenze reduziert. Testweise wurde der Anfangswert der Variationsbreite auf den Wert 5 % gesetzt; dies führte abgesehen von einer geringfügig höheren Schrittanzahl letztlich zu demselben Endergebnis.

Die Berechnung (Anhang 11.3.6) erzeugt nach 36 Optimierungsschritten die folgenden Ergebnisse:

$\overline{p}$	Name	Startwert $\overline{p}_0$	Endwert $\overline{p}_L$	Semantik $\overline{s}$
p_1	N_x	-4000	-4054,681529845111	Stützpunkt-Koord-X
p_2	N_y	3000	2958,9889674063684	Stützpunkt-Koord-Y
p_3	l_1	10800	10652,35986927056	Grundlenker-Länge
p_4	l_2	3000	2958,9889674063684	linkes oberes Teilsegment
p_5	l_3	9900	9764,66320267107	Zuglenker
p_6	l_4	5000	4931,648088062711	rechtes oberes Teilsegment

Tabelle 9.6: Parameterwerte

Die Wippweg-Höhendifferenz der Startsituation von

$$f_G(\overline{p}_0) = \Delta_{H,Start} = 377,35605167375616$$

wird durch die Optimierung zu einer Höhendifferenz von

$$f_G(\overline{p}_L) = \Delta_{H,Opt} = 301,13125340110764$$

Die Optimierung erzielt somit eine Verbesserung von 20,2 %. Da man davon ausgehen kann, dass die konstruktive Ursprungsgeometrie des Krans bereits einen guten kinematischen Zustand, basierend auf den Erfahrungen der Konstrukteure, repräsentiert hat, ist die rechnerische Verbesserung um über 20 % bemerkenswert.

Die Transformation speichert im Persistenz-Container die semantischen Informationen der Parameter, insbesondere die Zuordnung zu konstruktiven Elementen und Maßeinheiten, die nach der Berechnung durch die Post-Prozessoren den Zahlenwerten wieder zugeführt werden. Weiterhin werden Normierungsfaktoren für die Parameter eingeführt und ebenfalls im Container persistiert.

In diesem der konstruktiven Praxis entnommenen Beispiel kommt der Weiterverarbeitung der Ergebniswerte, insbesondere der Weitergabe der Werte an andere, externe Software-Systeme hohe Bedeutung zu. Zunächst kommt auch hier der bereits in Abschnitt 9.4 eingeführte Rundungs-Post-Prozessor für praxisgerechte konstruktive Maße zum Einsatz. Weiterhin wird die Optimierung üblicherweise in einen Konstruktionsprozess eingebunden sein, der es erfordert, konstruktive Varianten in CAD-Zeichnungen und andere Engineering-Dokumente zu übernehmen.

Dieser Ablauf soll hier dadurch exemplarisch aufgezeigt werden, dass die Endergebnisse der Optimierung in Form eines kinematischen Plots der Viergelenk-Geometrie ausgegeben werden (Abb. 9.11): Ein geeigneter Post-Prozessor erzeugt für diskrete Wippwinkel eine grafische Ausgabe der Kinematik-Hauptlinien. Zusätzlich ist der geforderte Wippweg mit roten Linien abgegrenzt. Der Post-Prozessor verwendet eine eigene Grafik-Engine mit grafischen Bibliotheken zur Erzeugung seiner Ausgabe.

Diese Aufgabe zeigt, basierend auf einer gegebenen konkreten Aufgabenstellung, die vollständige Durchführung aller Transformationsphasen, die Übergabe an ein Optimierungsverfahren, die Optimierungsrechnung und die Nachverarbeitung der Ergebnisse. Neben einem Standard-Post-Prozessor stellt ein individualisierter Post-Prozessor die Ergebnisse grafisch als Teil eines Konstruktionsprozesses dar. Damit konnte die Funktionsfähigkeit und Praxistauglichkeit

der in dieser Arbeit vorgestellten Vorgehensweise zur Vor- und Nachverarbeitung multikriterieller Aufgabenstellungen für die Optimierung aufgezeigt werden.

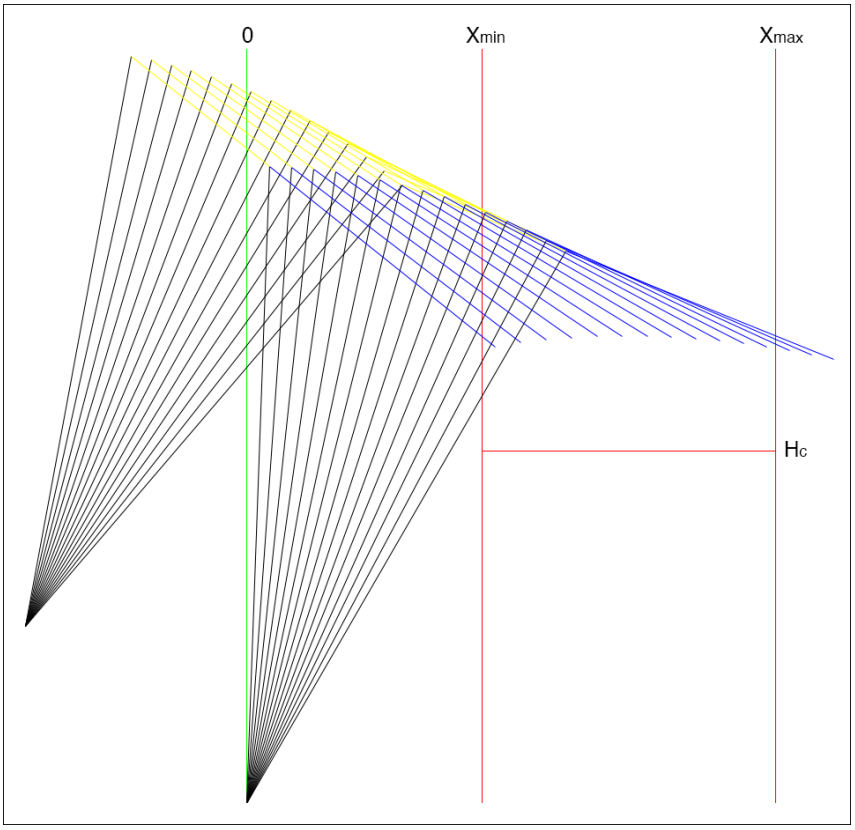


Abbildung 9.11: Kinematik-Plot der Lenker und des Wippweges

Kapitel 10

Zusammenfassung und Ausblick

10.1 Zusammenfassung

Diese Arbeit identifiziert und diskutiert Fragestellungen im Kontext semantischer Kriterieninterpretationen und Ergebnisdarstellungen am Beispiel von nichtlinearen Optimierungsverfahren.

Parameter-Optimierungsverfahren lassen sich auf eine vereinheitlichte Struktur von Eingabe- und Ausgabe-Daten zurückführen, die jedoch von semantischen Informationen der eigentlichen Aufgabenstellung abstrahieren. Das Szenario einer konkreten, kriteriengebundenen Aufgabenstellung wird von semantischen Aspekten befreit und in die Datenstruktur der Optimierung transformiert. Die Interpretation und zeitlich-räumlich getrennte Weiterverwendung der Ergebnisdaten der Optimierung wird dann wieder in die semantische Informationsebene der ursprünglichen Aufgabenstellung zurückgeführt.

Zunächst wird ein standardisierter Transformationsprozess zur Gewinnung abstrahierter Datenstrukturen für die Optimierung beschrieben. Im Anschluss werden die notwendigen Serialisierungs- und Informationsstruktur-Anforderungen für die interpretatorische Rückführung der Ergebnisdaten hergeleitet.

Um Optimierungsergebnisse wieder mit der in der Transformationsphase abstrahierten Informationsbedeutung zu versehen, wurde die Ontologie eines Semantik-Containers konzipiert, der die semantischen Anteile der Anforderungskriterien vor einem Lösungsverfahren von den Optimierungsdaten löst und in einer aus der Ontologie abgeleiteten, strukturierten Sprache persistiert.

Auf Basis der entwickelten Semantiksprache wird aufgezeigt, wie eine Automatisierung der Vor- und Nachbearbeitung kriteriengetriebener Aufgabenstellungen unter Einbeziehung eines numerischen Rechenkerns formuliert werden kann. Dazu werden Prä- und Post-Prozessoren in einen gesteuerten Ablauf eingebunden. Insbesondere Implementierungen der Post-Prozessor-Komponenten sind frei in der Art der Ergebnis-Interpretation und Ergebnis-Präsentation und können zeitlich-räumlich getrennte Darstellungen in tabellarischer, grafischer oder exportierender Art produzieren.

Die Anwendbarkeit der vorgestellten Transformationen und Persistierungen werden anhand praxisgerechter Beispiele gezeigt. Die diskutierten Transformations- und Interpretationsprozesse werden schließlich auf andere numerische Kontexte jenseits der Parameteroptimierung verallgemeinert.

Mithilfe dieser Arbeit wird es möglich, auf strukturierte Weise aus einer allgemein formulierten textuellen Aufgabenstellung die Kriterien für eine Optimierungsrechnung zu extrahieren, Semantik und Daten voneinander zu trennen, die numerische Berechnung durchzuführen und abschließend Ergebnisdaten und Semantik wieder zusammenzuführen, sodass eine angemessene Antwort auf die gestellte Aufgabe gegeben werden kann. Damit werden multikriterielle Problemstellungen und ihre berechneten Lösungen von der Ebene der rein numerischen Berechnung auf die Ebene des semantischen Verständnisses der gestellten Aufgabe gehoben und die kontextuellen Verständnisebenen von Aufgabe, Berechnung und Lösung miteinander versöhnt.

10.2 Ausblick

Der beschriebene Transformationsprozess basiert bisher noch auf manueller und kognitiver Identifikationsarbeit durch einen Bearbeiter. Eine volle Automatisierung würde demgegenüber auf Konzepten der Künstlichen Intelligenz basieren und eine automatische Erkennung von Kriterien und ihre Überführung in Parameter, Beschränkungen und Restriktionen ermöglichen; im Idealfall würde auch die Zielfunktion algorithmisch erfasst und formuliert werden können, jedoch erscheint dies vom heutigen Standpunkt aus gesehen problematisch.

Die beschriebene Verallgemeinerung des Transformations- und Ergebnis-

interpretations-Verfahrens auf andere, gleichgelagerte Problemstellungen des Übergangs Informationen → Daten → Informationen erfordert eine Verallgemeinerung der ontologischen Datenstrukturen, da diese dann nicht mehr auf die Optimierungsproblematik begrenzt sein dürften. Im Beispiel der Finite-Elemente-Berechnung wäre beispielsweise eine Verwaltung von Knoten, Elementen und Kräften erforderlich. Eine tatsächliche Verallgemeinerung der Datenstrukturen würde eine Abstraktion der für den Rechenkern erforderten und vom Rechenkern gelieferten Datenelemente benötigen.

Kapitel 11

Anhang

11.1 XML-Quellcode

11.1.1 XML-Schema der Optimierungs-Modellierung

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xml:lang="de" elementFormDefault="qualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="www.opppml.de"
  version="1.0">

  <xs:annotation>
    <xs:appinfo>Task-Infos</xs:appinfo>
    <xs:documentation>Strukturierung fuer die Task</xs:documentation>
  </xs:annotation>

  <xs:element name="task">
    <xs:annotation>
      <xs:documentation>Wurzelelement der Task</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="story" minOccurs="1" maxOccurs="1" />
        <xs:element ref="comment" minOccurs="0" maxOccurs="1" />
        <xs:element ref="parameters" minOccurs="1" maxOccurs="1" />
        <xs:element ref="constraints" minOccurs="0" maxOccurs="1" />
        <xs:element ref="restrictions" minOccurs="0" maxOccurs="1" />
        <xs:element ref="calculations" minOccurs="1" maxOccurs="1" />
        <xs:element ref="post-processors" minOccurs="1" maxOccurs="1" />
        <xs:element ref="file-references" minOccurs="0" maxOccurs="1" />
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" />
      <xs:attribute name="meta" type="xs:string" />
    </xs:complexType>
  </xs:element>

  <xs:element name="story" type="xs:string">
    <xs:annotation>
      <xs:documentation>Aufgabenstellung der Task</xs:documentation>
    </xs:annotation>
  </xs:element>

  <xs:element name="comment" type="xs:string">
    <xs:annotation>
      <xs:documentation>Kommentar zur Task</xs:documentation>
    </xs:annotation>
```

```
  </xs:element>

  <xs:element name="parameters">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="parameter" minOccurs="1" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="parameter">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="value" minOccurs="1" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="key" type="xs:integer" />
      <xs:attribute name="name" type="xs:string" />
      <xs:attribute name="norm" type="xs:double" />
      <xs:attribute name="semantic" type="xs:string" />
      <xs:attribute name="unit" type="xs:string" />
    </xs:complexType>
  </xs:element>

  <xs:element name="value">
    <xs:complexType>
      <xs:attribute name="id" type="xs:integer" />
      <xs:attribute name="value" type="xs:double" />
    </xs:complexType>
  </xs:element>

  <xs:element name="constraints">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="constraint" minOccurs="1" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="constraint">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="field" minOccurs="1" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="class-name" type="xs:string" />
      <xs:attribute name="name" type="xs:string" />
    </xs:complexType>
  </xs:element>

  <xs:element name="field">
    <xs:complexType>
      <xs:attribute name="name" type="xs:string" />
      <xs:attribute name="type" type="xs:string" />
      <xs:attribute name="value" type="xs:double" />
      <xs:attribute name="key-ref" type="xs:string" />
    </xs:complexType>
  </xs:element>

  <xs:element name="restrictions">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="restriction" minOccurs="1" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="restriction">
    <xs:complexType>
      <xs:attribute name="class-name" type="xs:string" />
      <xs:attribute name="name" type="xs:string" />
      <xs:attribute name="factor" type="xs:double" />
```

```
    </xs:complexType>
  </xs:element>

  <xs:element name="target-function">
    <xs:complexType>
      <xs:attribute name="class-name" type="xs:string" />
      <xs:attribute name="name" type="xs:string" />
    </xs:complexType>
  </xs:element>

  <xs:element name="calculations">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="calculation" minOccurs="1" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="calculation">
    <xs:complexType>
      <xs:attribute name="date" type="xs:date" />
      <xs:attribute name="description" type="xs:string" />
      <xs:attribute name="id" type="xs:integer" />
      <xs:attribute name="target-value" type="xs:double" />
      <xs:attribute name="control-data" type="xs:string" />
      <xs:attribute name="meta" type="xs:string" />
    </xs:complexType>
  </xs:element>

  <xs:element name="file-references">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="file-reference" minOccurs="1" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="file-reference">
    <xs:complexType>
      <xs:attribute name="file-path" type="xs:string" />
    </xs:complexType>
  </xs:element>
</xs:schema>
```

11.1.2 XML-Schema der automatischen Ergebnisinterpretation

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xml:lang="de" elementFormDefault="qualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" targetNamespace="www.opppml.de"
  version="1.0">

  <xs:annotation>
    <xs:appinfo>Prozessoren</xs:appinfo>
    <xs:documentation>Strukturierung von automatischen Ergebnisinterpretationen</
        xs:documentation>
  </xs:annotation>

  <xs:element name="processors">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="pre-processor" minOccurs="0" maxOccurs="unbounded" />
        <xs:element ref="post-processor" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
```

```
  </xs:element>

  <xs:element name="pre-processor">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="pre-processor-class" minOccurs="0" maxOccurs="unbounded" />
        <xs:element ref="ppre-processor-service" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" />
      <xs:attribute name="id" type="xs:integer" />
      <xs:attribute name="export-path" type="xs:string" />
    </xs:complexType>
  </xs:element>

<xs:element name="pre-processor-class">
    <xs:complexType>
      <xs:attribute name="class-name" type="xs:string" />
    </xs:complexType>
  </xs:element>

<xs:element name="pre-processor-service">
    <xs:complexType>
      <xs:attribute name="url" type="xs:string" />
    </xs:complexType>
  </xs:element>

  <xs:element name="post-processor">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="post-processor-class" minOccurs="0" maxOccurs="unbounded" />
        <xs:element ref="post-processor-service" minOccurs="0" maxOccurs="unbounded" />
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" />
      <xs:attribute name="id" type="xs:integer" />
      <xs:attribute name="export-path" type="xs:string" />
    </xs:complexType>
  </xs:element>

<xs:element name="post-processor-class">
    <xs:complexType>
      <xs:attribute name="class-name" type="xs:string" />
    </xs:complexType>
  </xs:element>

<xs:element name="post-processor-service">
    <xs:complexType>
      <xs:attribute name="url" type="xs:string" />
    </xs:complexType>
  </xs:element>

</xs:schema>
```

11.2 Modellierungssprachen-Quellcode

11.2.1 CircleProblem in Java

```
package de.buw.lfa.circle.model;
import java.util.ArrayList;
import java.util.List;

import de.buw.lfa.combo.model.*;
import de.buw.lfa.combo.model.exceptions.OptimizationException;
import static java.lang.Math.*;
```

```
public class CircleTarget implements ITarget {
    protected List<Point> points;
    protected double radius;

    public CircleTarget() {
      points = new ArrayList<Point>();
      points.add(new Point(  5.0, 15.0));
      points.add(new Point(  5.0, -15.0));
      points.add(new Point(  0.0, 0.0));
      radius = 60;
    }

    public CircleTarget(List<Point> points, double radius) {
      this.points = points;
      this.radius = radius;
    }

    @Override
    public double evaluate(ParameterSet parameters, List<IRestriction> restrictions)
                throws OptimizationException {
      Point m = new Point(parameters.getValueByKey("x"),                  parameters.
          getValueByKey("y"));
      double r = parameters.getValueByKey("R");
      double res = sum(points, r, m);
      return res;
    }

    @Override
    public double getCurrentTargetValue() { return 0; }

    private double sum(List<Point> points, double radius, Point m) {
      double res = 0.0;
      for (Point p : points) {
        res += pow(radius - sqrt(pow(p.x-m.x, 2) + pow(p.y-m.y, 2)), 2);
      }
      return res;
    }
    public List<Point> getPoints() { return points; }

    public void setPoints(List<Point> points) { this.points = points; }

    public double getRadius() { return radius;   }

    public void setRadius(double radius) { this.radius = radius; }
}
```

11.2.2 CircleProblem in GAMS

Befehlsausführung

```
user\$ gams circle.gms
```

circle.gms

```
SETS
 P   points   / P1, P2, P3 / ;
PARAMETERS
 PX(P)     / P1   0.0
             P2   5.0
             P3   5.0 /
 PY(P)    / P1   0.0
             P2  15.0
             P3 -15.0 / ;
  VARIABLES
       X       x-koordinate des mittelpunkts
```

```
      Y       y-koordinate des mittelpunkts
      R       radius
      Z       zielfunktionswert ;
  POSITIVE VARIABLE R ;
  EQUATIONS
      ZFK       zielfunktion definieren ;
  ZFK ..  Z  =E=  SUM((P), POWER(R - SQRT(POWER(PX(P)-X, 2)
                                  + POWER(PY(P)-Y, 2)), 2)) ;
  MODEL CIRCLEPROBLEM /ALL/ ;
  SOLVE CIRCLEPROBLEM USING NLP MINIMIZING Z ;

  Display R.l, R.m ;
```

circle.gms

```
GAMS 24.4.3  r51699 Released Apr  2, 2015 DEX-DEG x86 64bit/MacOS X
05/18/15 17:55:37 Page 1
General  Algebraic  Modeling  System
Compilation

   1  SETS
   2   P   points   / P1, P2, P3 / ;
   3  PARAMETERS
   4   PX(P)   / P1   0.0
   5             P2   5.0
   6             P3   5.0 /
   7   PY(P)   / P1   0.0
   8             P2  15.0
   9             P3 -15.0 / ;
  10    VARIABLES
  11         X       x-koordinate des mittelpunkts
  12         Y       y-koordinate des mittelpunkts
  13         R       radius
  14         Z       zielfunktionswert ;
  15    POSITIVE VARIABLE R ;
  16    EQUATIONS
  17         ZFK          zielfunktion definieren ;
  18    ZFK ..  Z =E= SUM((P), POWER(R - SQRT(POWER(PX(P)-X, 2)
                                  + POWER(PY(P)-Y, 2)) ,2));
  19    MODEL CIRCLEPROBLEM /ALL/ ;
  20    SOLVE CIRCLEPROBLEM USING NLP MINIMIZING Z ;
  21
  22    Display R.l, R.m ;

COMPILATION TIME     =        0.001 SECONDS      3 MB  24.4.3 r51699
DEX-DEG
GAMS 24.4.3  r51699 Released Apr  2, 2015 DEX-DEG x86 64bit/MacOS 05/18/15 17:55:37 Page 2
General  Algebraic  Modeling  System
Equation Listing    SOLVE CIRCLEPROBLEM Using NLP From line 20

---- ZFK  =E=  zielfunktion definieren

ZFK..  (20)*X + (0)*Y + (63.2455532033676)*R + Z =E= 0 ; (LHS = -500, INFES = 500 ****)

GAMS 24.4.3  r51699 Released Apr  2, 2015 DEX-DEG x86 64bit/MacOS X
05/18/15 17:55:37 Page 3
General  Algebraic  Modeling  System
Column Listing      SOLVE CIRCLEPROBLEM Using NLP From line 20

---- X  x-koordinate des mittelpunkts

X
                (.LO, .L, .UP, .M = -INF, 0, +INF, 0)
      (20)      ZFK

---- Y  y-koordinate des mittelpunkts

Y
                (.LO, .L, .UP, .M = -INF, 0, +INF, 0)
       (0)      ZFK
```

```
---- R  radius

R
                (.LO, .L, .UP, .M = 0, 0, +INF, 0)
      (63.2456) ZFK

---- Z  zielfunktionswert

Z
                (.LO, .L, .UP, .M = -INF, 0, +INF, 0)
        1       ZFK

GAMS 24.4.3  r51699 Released Apr  2, 2015 DEX-DEG x86 64bit/MacOS X
05/18/15 17:55:37 Page 4
G e n e r a l  A l g e b r a i c  M o d e l i n g  S y s t e m
Model Statistics    SOLVE CIRCLEPROBLEM Using NLP From line 20

MODEL STATISTICS

BLOCKS OF EQUATIONS           1     SINGLE EQUATIONS            1
BLOCKS OF VARIABLES           4     SINGLE VARIABLES            4
NON ZERO ELEMENTS             4     NON LINEAR N-Z              3
DERIVATIVE POOL              20     CONSTANT POOL              18
CODE LENGTH                  37

GENERATION TIME      =        0.001 SECONDS      4 MB  24.4.3 r51699 DEX-DEG

EXECUTION TIME       =        0.001 SECONDS      4 MB  24.4.3 r51699 DEX-DEG
GAMS 24.4.3  r51699 Released Apr  2, 2015 DEX-DEG x86 64bit/MacOS X
05/18/15 17:55:37 Page 5
G e n e r a l  A l g e b r a i c  M o d e l i n g  S y s t e m
Solution Report     SOLVE CIRCLEPROBLEM Using NLP From line 20

          S O L V E      S U M M A R Y

     MODEL   CIRCLEPROBLEM       OBJECTIVE  Z
     TYPE    NLP                 DIRECTION  MINIMIZE
     SOLVER  CONOPT              FROM LINE  20

**** SOLVER STATUS     1 Normal Completion
**** MODEL STATUS      2 Locally Optimal
**** OBJECTIVE VALUE                0.0000

 RESOURCE USAGE, LIMIT          0.002       1000.000
 ITERATION COUNT, LIMIT        14     2000000000
 EVALUATION ERRORS              0              0
CONOPT 3          24.4.3 r51699 Released Apr  2, 2015 DEG x86 64bit/MacOS X

    C O N O P T 3   version 3.16D
    Copyright (C)   ARKI Consulting and Development A/S
                    Bagsvaerdvej 246 A
                    DK-2880 Bagsvaerd, Denmark

    The model has 4 variables and 1 constraints
    with 4 Jacobian elements, 3 of which are nonlinear.
    The Hessian of the Lagrangian has 3 elements on the diagonal,
    3 elements below the diagonal, and 3 nonlinear variables.

 ** Optimal solution. Reduced gradient less than tolerance.

  CONOPT time Total                            0.001 seconds
   of which: Function evaluations             0.000 =  0.0%
             1st Derivative evaluations       0.000 =  0.0%
             2nd Derivative evaluations       0.000 =  0.0%
             Directional 2nd Derivative       0.000 =  0.0%

     LOWER          LEVEL          UPPER         MARGINAL

---- EQU ZFK          .              .             .    1.0000
```

```
  ZFK  zielfunktion definieren

     LOWER          LEVEL          UPPER         MARGINAL

---- VAR X    -INF      25.0000          +INF        EPS
---- VAR Y    -INF        .              +INF        EPS
---- VAR R      .       25.0000          +INF        EPS
---- VAR Z    -INF       5.260756E-26    +INF         .
  X  x-koordinate des mittelpunkts
  Y  y-koordinate des mittelpunkts
  R  radius

GAMS 24.4.3  r51699 Released Apr  2, 2015 DEX-DEG x86 64bit/MacOS X
05/18/15 17:55:37 Page 6
G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m
Solution Report     SOLVE CIRCLEPROBLEM Using NLP From line 20

  Z  zielfunktionswert

**** REPORT SUMMARY :        0     NONOPT
                             0  INFEASIBLE
                             0  UNBOUNDED
                             0     ERRORS
GAMS 24.4.3  r51699 Released Apr  2, 2015 DEX-DEG x86 64bit/MacOS X
05/18/15 17:55:37 Page 7
G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m
E x e c u t i o n


----     22 VARIABLE R.L                  =       25.000  radius
            VARIABLE R.M                  =         EPS   radius

EXECUTION TIME  =  0.001 SECONDS      3 MB  24.4.3 r51699 DEX-DEG

USER: GAMS Development Corporation, Washington, DC   G871201/0000CA-ANY
 Free Demo,  202-342-0180,  sales@gams.com,  www.gams.com   DC0000

**** FILE SUMMARY
Input      /Applications/GAMS/gams/circle.gms
Output     /Applications/GAMS/gams/circle.lst
```

11.2.3 CircleProblem in AMPL

Befehlsausführung:

```
ampl: reset;
ampl: option solver minox;
ampl: model circle.mod;
ampl: data circle.dat;
ampl: solve;
MINOS 5.51: optimal solution found.
11 iterations, objective 1.184022629e-13
Nonlin evals: obj = 23, grad = 22.
ampl: display radius > circle.out;
radius = 25
```

circle.mod:

```
var radius;          ## Radius
set pointNames;      ## Gegebene Punkte
var mx;      ## Mittelpunkt
var my;      ## Mittelpunkt

param n;
```

```
param pointsX{i in 1..n};
param pointsY{i in 1..n};

minimize zfkvalue: sum{i in 1..n} (radius - sqrt( (pointsX[i]-mx)^2 + (pointsY[i]-my)^2)
    )^2;
subject to Xrange: 0 <= mx <= 120;
subject to Yrange: 0 <= my <= 120;
subject to Radius: 0 <= radius <= 120;
```

circle.dat:

```
param radius := 60;
param n:= 3;
param mx:= 10;
param my:= 10;

param pointsX:= 1   0
                2   5
                3   5;
param pointsY:= 1   0
                2 -15
                3  15;
```

11.3 Implementierung-Code

11.3.1 Rosenbrock

```
public class RosenbrockProcessor extends PlottingTaskModelPrePostProcessor {

  public RosenbrockProcessor() {
    super();
  }

  @Override
  public void preProcess(IReporter reporter) {
    if (reporter != null) {
      reporter.write("Starten...");
    }
  }

  @Override
  public void postProcess(IReporter reporter, Result result) {

    if (reporter != null) {
      reporter.write("Nachbearbeitung:");
      for (IParameter p : result.getParameters()) {
        String sem = findSemantic(p.getKey());
        reporter.write("Parameter: "+p.getName()+" ("+ sem +")");
      }
      plotTargetHistory(reporter, result, outputPath + "/RosTarget.png");

      plotParameterHistory(reporter, result, outputPath + "/RosParameters.png");
    }}}
```

```
public class RosenbrockTarget implements ITarget {

  @Override
  public double evaluate(ParameterSet parameters, List<IRestriction> restrictions) throws
      OptimizationException {
    double x = parameters.getValueByKey("x");
    double y = parameters.getValueByKey("y");
```

```
    return Math.pow(1.0-x, 2) + 100 * Math.pow(y-x*x, 2);
  }

  @Override
  public double getCurrentTargetValue() {
    return 0;
  }}
```

11.3.2 Kreis-Aufgabe

```
public class CircleProcessor extends PlottingTaskModelPrePostProcessor {

  public CircleProcessor() {
    super();
  }

  @Override
  public void preProcess(IReporter reporter) {
    if (reporter != null) {
      reporter.write("Starten...");
    }
  }

  @Override
  public void postProcess(IReporter reporter, Result result) {

    if (reporter != null) {
      reporter.write("Nachbearbeitung:");
      for (IParameter p : result.getParameters()) {
        String sem = findSemantic(p.getKey());
        reporter.write("Parameter:␣"+p.getName()+"␣("+ sem +")");
      }

      plotTargetHistory(reporter, result, outputPath + "/CirTarget.png");

      plotParameterHistory(reporter, result, outputPath + "/CirParameters.png");
   }}}
```

```
public class CircleTarget implements ITarget {
  protected List<Point> points;
  protected double radius;

  public CircleTarget() {
    points = new ArrayList<Point>();
    points.add(new Point(  5.0, 15.0));
    points.add(new Point(  5.0, -15.0));
    points.add(new Point(  0.0, 0.0));
    radius = 60;
  }

  public CircleTarget(List<Point> points, double radius) {
    this.points = points;
    this.radius = radius;
  }
  @Override
  public double evaluate(ParameterSet parameters, List<IRestriction> restrictions) throws
      OptimizationException {

    Point m = new Point(parameters.getValueByKey("x"), parameters.getValueByKey("y"));
    double r = parameters.getValueByKey("R");

    return sum(points, r, m);
  }

  @Override
  public double getCurrentTargetValue() {
```

```
    return 0;
  }

  private double sum(List<Point> points, double radius, Point m) {
    double res = 0.0;
    for (Point p : points) {
      res += pow(radius - sqrt(pow(p.x-m.x, 2) + pow(p.y-m.y, 2)), 2);
    }
    return res;
  }

  public List<Point> getPoints() {
    return points;
  }

  public void setPoints(List<Point> points) {
    this.points = points;
  }

  public double getRadius() {
    return radius;
  }

  public void setRadius(double radius) {
    this.radius = radius;
  }
}
```

11.3.3 Leiter-Aufgabe

```
package de.buw.lfa.combo.ladder;

import java.util.List;

import de.buw.lfa.combo.model.*;
import de.buw.lfa.combo.model.exceptions.OptimizationException;

public class LadderTarget implements ITarget { // Keine Restriktionen,
  double length = 7.0;       // Laenge der Leiter
  double qWidth = 1.0;       // Breite des Quaders
  double qHeight = 1.0;       // Hoehe des Quaders

  double currentQualityValue;

  @Override
  public double evaluate(ParameterSet parameters, List<IRestriction> restrictions) throws
      OptimizationException {

    double alpha = parameters.getValue(0);
    alpha = Math.toRadians(alpha);         // Bogenma"s

    double l1 = qHeight / Math.sin(alpha);
    double l2 = qWidth / Math.cos(alpha);

    currentQualityValue =  Math.abs(length - (l1 + l2));

    return currentQualityValue;
  }

  /**
   * Zfk-Wert aus dem letzten evaluate()
   */
  @Override
  public double getCurrentTargetValue() {
    return currentQualityValue;
  }
```

```
}
```

11.3.4 Schaf-Aufgabe

```
public class SheepTarget implements ITarget {

  double R = 50.0;       // Radius der Weideflaeche

  @Override
  public double evaluate(ParameterSet parameters, List<IRestriction> restrictions) throws
      OptimizationException {

    double distance = parameters.getValue(0);
    double sHalf = Math.sqrt(R*R - distance*distance);
    double r =  Math.sqrt(sHalf*sHalf + (R-distance)*(R-distance));     // aktuelle
        Leinenlaenge

    double alphaHalf = Math.asin(sHalf/r);
    double alpha = 2 * alphaHalf;
    double A1 = r * r / 2.0 * (alpha - Math.sin(alpha));

    double betaHalf = Math.asin(sHalf/R);
    double beta = 2 * betaHalf;
    double A2 = R * R / 2.0 * (beta - Math.sin(beta));

    double A = A1 + A2;                          // aktuelle Schaf-Flaeche
    double Ak = 0.5 * Math.PI * R * R;      // halbe Gesamtweideflaeche

    return Math.abs(Ak - A);                    // Schaf soll die Haelfte der
        Gesamtweideflaeche erreichen koennen
  }

  @Override
  public double getCurrentTargetValue() {
    return 0;
  }
}
```

```
public class SheepTarget implements ITarget {

  double R = 50.0;       // Radius der Weideflaeche

  @Override
  public double evaluate(ParameterSet parameters, List<IRestriction> restrictions) throws
      OptimizationException {

    double distance = parameters.getValue(0);
    double sHalf = Math.sqrt(R*R - distance*distance);
    double r =  Math.sqrt(sHalf*sHalf + (R-distance)*(R-distance));     // aktuelle
        Leinenlaenge

    double alphaHalf = Math.asin(sHalf/r);
    double alpha = 2 * alphaHalf;
    double A1 = r * r / 2.0 * (alpha - Math.sin(alpha));

    double betaHalf = Math.asin(sHalf/R);
    double beta = 2 * betaHalf;
    double A2 = R * R / 2.0 * (beta - Math.sin(beta));

    double A = A1 + A2;                          // aktuelle Schaf-Flaeche
    double Ak = 0.5 * Math.PI * R * R;                  // halbe Gesamtweideflaeche

    return Math.abs(Ak - A);                     // Schaf soll die Haelfte der
        Gesamtweideflaeche erreichen koennen
  }
```

```
  @Override
  public double getCurrentTargetValue() {
    return 0;
  }
}
```

```
public class SheepProcessor extends PlottingTaskModelPrePostProcessor {

  public SheepProcessor() {
    super();
  }

  @Override
  public void preProcess(IReporter reporter) {
    if (reporter != null) {
      reporter.write("Starten...");
    }
  }

  @Override
  public void postProcess(IReporter reporter, Result result) {

    if (reporter != null) {
      reporter.write("Nachbearbeitung:");
      for (IParameter p : result.getParameters()) {
        String sem = findSemantic(p.getKey());
        reporter.write("Parameter:␣"+p.getName()+"␣("+ sem +")");
      }

      plotTargetHistory(reporter, result, outputPath + "/SheTarget.png");

      plotParameterHistory(reporter, result, outputPath + "/SheParameters.png");
    }
  }
}
```

11.3.5 Time Tabling

```
public class DayOfWeekConstraint extends TimeConstraint {
  private int dayOfWeek;
  private TimeLine timeLine;

public class DayOfWeekConstraint extends TimeConstraint {
  private int dayOfWeek;
private TimeLine timeLine;
public DayOfWeekConstraint(String name, int dayOfWeek, TimeLine timeLine) {
    super(name, 0, 0);
    this.dayOfWeek = dayOfWeek;
    this.timeLine = timeLine;

    System.out.println("day␣of␣week:␣"+dayOfWeek);
  }

  public boolean isTimeValid(int time) {
    GregorianCalendar cal = timeLine.calendarFromHour(time);
    int dow = cal.get(GregorianCalendar.DAY_OF_WEEK);          // 1=Sunday...
    System.out.println("time:␣"+time+"␣␣␣␣␣dow:␣"+dow);
    return dow != dayOfWeek;
  }
}
```

```
public class DaylightConstraint extends TimeConstraint {


  public DaylightConstraint(String name, int start, int end) {
```

```
    super(name, start, end);
  }

  public boolean isTimeValid(int time) {
    time = time % 24;
    return time >= start && time <= end;
  }
}
```

```
public class WeekendConstraint extends TimeConstraint {


  public WeekendConstraint(String name, int start, int end) {
    super(name, start, end);
  }

  public boolean isTimeValid(int time) {
    time = time % 168;
    return time < start;
  }
}
```

```
public class NoCollisionRestriction extends TimeRestriction {

  protected TimeSpanList timeSpans;

  public NoCollisionRestriction(String name, double factor, TimeSpanList timeSpans) {
    super(name, factor, 0, 0);
    this.timeSpans = timeSpans;
  }

  @Override
  public double calcPenalty(ParameterSet parameters) throws OptimizationException {
    double overlap = 0.0;
    for (int i = 0; i<timeSpans.spans.size(); i++) {
      TimeSpan ts1 = timeSpans.spans.get(i);

      for (int j=i+1; j<timeSpans.spans.size(); j++) {
        TimeSpan ts2 = timeSpans.spans.get(j);
        overlap += ts1.getOverlap(ts2);
      }
    }
    return overlap * factor;
  }

  @Override
  public boolean isTimeValid(int time) {
    return false;
  }
}
```

```
public class DistanceRestriction extends TimeRestriction {

  private TimeSpanList timeSpans;

  private int distance;

  public DistanceRestriction(String name, double factor, TimeSpanList timeSpans, int
      distance) {
    super(name, factor, 0, 0);
    this.timeSpans = timeSpans;
    this.distance = distance;
  }

  @Override
  public double calcPenalty(ParameterSet parameters) throws OptimizationException {
    double overlap = 0.0;
    for (int i = 0; i<timeSpans.spans.size(); i++) {
```

```
      TimeSpan ts1 = timeSpans.spans.get(i);

      for (int j=i+1; j<timeSpans.spans.size(); j++) {
        TimeSpan ts2 = timeSpans.spans.get(j);
        int dist = ts1.getDistance(ts2);
        if (dist < distance)
          overlap += (distance - dist);    // Ueberlappung wird bestraft
      }
    }

    return overlap * factor;
  }

  class TimeSpanDistance {
    TimeSpan ts1;
    TimeSpan ts2;
    double distance;

    public TimeSpanDistance(TimeSpan ts1, TimeSpan ts2, double distance) {
      this.ts1 = ts1;
      this.ts2 = ts2;
      this.distance = distance;
    }

    public double getDiff() {
      return Math.abs(ts1.getStart() - ts2.getStart());
    }

    public boolean isDistanceOk() {
      return getDiff() >= distance;
    }
  }

  @Override
  public boolean isTimeValid(int time) {
    return false;
  }
}
```

```
  @Override
  public Result solve() {
    TimeLine timeLine = new TimeLine(0, 8 * HOURS_PER_WEEK);        // 8 Wochen

    List<TimeSpan> timeSpans = new ArrayList<TimeSpan>();
    timeSpans.add(new TimeSpan(1, "Experimentalphysik (NK)", 0, 3));    // Start, Dauer
    timeSpans.add(new TimeSpan(2, "Mess- und Schaltungstechnik", 0, 4));
    timeSpans.add(new TimeSpan(3, "Energiespeicher (NK)", 0, 3));
    timeSpans.add(new TimeSpan(4, "Grundlagen der Hochfrequenztechnik", 0, 3));
    timeSpans.add(new TimeSpan(5, "Werkstoffe und Grundschaltungen (NK)", 0, 3));
    timeSpans.add(new TimeSpan(6, "Duennschichttechnologie (NK)", 0, 4));
    timeSpans.add(new TimeSpan(7, "Elektronische Bauelemente (NK)", 0, 3));
    timeSpans.add(new TimeSpan(8, "Mathematik B", 0, 3));
    timeSpans.add(new TimeSpan(9, "Theoretische Elektrotechnik I", 0, 3));
    timeSpans.add(new TimeSpan(10, "Regelungstechnik (NK)", 0, 4));
    timeSpans.add(new TimeSpan(11, "Elektromobilitaet", 0, 3));
    timeSpans.add(new TimeSpan(12, "Prozessinformatik", 85000, 180));
    timeSpans.add(new TimeSpan(13, "Mathematik 3 (NK)", 50000, 180));
    timeSpans.add(new TimeSpan(14, "Rechnernetze und Datenbanken", 33000, 180));
    timeSpans.add(new TimeSpan(15, "Grundlagen der Elektrotechnik B (NK)", 20000, 180));
    timeSpans.add(new TimeSpan(16, "Theoretische Nachrichtentechnik (NK)", 2100, 240));
    timeSpans.add(new TimeSpan(17, "Energiesysteme (NK)", 15000, 180));
    timeSpans.add(new TimeSpan(18, "Technische Mechanik", 85000, 180));
    timeSpans.add(new TimeSpan(19, "Mathematik A (NK)", 50000, 180));
    timeSpans.add(new TimeSpan(20, "Grundzuege der Informatik (NK)", 33000, 180));
    timeSpans.add(new TimeSpan(21, "Signale und Systeme", 20000, 180));
    timeSpans.add(new TimeSpan(22, "Opto- und Nanoelektronik", 2100, 240));
    timeSpans.add(new TimeSpan(23, "Theoretische Elektrotechnik II (NK)", 15000, 180));
    timeSpans.add(new TimeSpan(24, "Kommunikationstechnik (NK)", 85000, 180));
    timeSpans.add(new TimeSpan(25, "Analoge und digitale Schaltungen", 50000, 180));
    timeSpans.add(new TimeSpan(26, "Physikalische Grundlagen drahtloser
```

```
        Kommunikationstechnologie", 33000, 180));
    timeSpans.add(new TimeSpan(27, "Regelungstheorie", 20000, 180));
    timeSpans.add(new TimeSpan(28, "Signal- und Mikroprozessortechnik (NK)", 20000, 180));

    restrictions.add(new NoCollisionRestriction("Kollisionen vermeiden", 1000, new
        TimeSpanList(timeSpans, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)));
    restrictions.add(new DistanceRestriction("Abstand mindestens 4 Tage", 10, new
        TimeSpanList(timeSpans, 1, 2, 3), 4*24));

    constraints.add(new WeekendConstraint("Wochenenden", 120, 168));
    constraints.add(new DaylightConstraint("Tagsueber", 8, 18));

    TimeTablingParameterFactory factory = new TimeTablingParameterFactory(timeLine,
        constraints);
    for (TimeSpan ts : timeSpans) {
      TimeTablingParameter param = factory.createParameter(ts);
      ts.setParameter(param);
      parameters.add(param);
    }

    objectiveFunction = new TimeTableObjectiveFunction();

    EvolutionOptimizer optimizer = new EvolutionOptimizer();
    optimizer.initialize(parameters, constraints, restrictions, objectiveFunction);

    Map<String,Object> controlParams = new HashMap<String,Object>();
    controlParams.put(EvolutionOptimizer.PARAM_MAXSTEPS, new Integer(10));

    Result result = optimizer.run(controlParams);

    StringBuffer buf = new StringBuffer("Endergebnis:"+'\n');

    Collections.sort(timeSpans, new Comparator<TimeSpan>() {
      @Override
      public int compare(TimeSpan o1, TimeSpan o2) {
        double d1 = o1.getCurrentStart();
        double d2 = o2.getCurrentStart();
        return Double.compare(d1, d2);
      }
    });
    for (TimeSpan ts : timeSpans) {
      buf.append(hourToDate(((TimeTablingParameter)ts.getParameter()).getTimeValue()) + "
          " + ts.getId() + "  " + ts.getName() + '\n');
  }

    buf.append("    Zfk-Wert = " + result.getOfValue() + '\n');
    buf.append("    Schritte: " + result.getOptimizer().getLapCount() + '\n');
    if (reporter != null)
      reporter.write(buf.toString());

    return result;
  }
```

11.3.6 Lemniskate

```
public class LemniskateProcessor extends PlottingTaskModelPrePostProcessor {

  public LemniskateProcessor() {
    super();
  }

  @Override
  public void preProcess(IReporter reporter) {
    if (reporter != null) {
      reporter.write("Starten...");
    }
```

```
  }

  @Override
  public void postProcess(final IReporter reporter, Result result) {
    processedFiles.clear();

    if (reporter != null) {
      reporter.write("Nachbearbeitung:");
      for (IParameter p : result.getParameters()) {
        String sem = findSemantic(p.getKey());
        reporter.write("Parameter: "+p.getName()+" ("+ sem +")");
      }

      Lemniskate lem = Lemniskate.getInstance();
      double hdiff = lem.getHeightDiff(Math.toRadians(Lemniskate.PHIA), Math.toRadians(
          Lemniskate.PHIE), Math.toRadians(Lemniskate.DPHI));
      double hmin = lem.getMinimumHeight(Math.toRadians(Lemniskate.PHIA), Math.toRadians(
          Lemniskate.PHIE), Math.toRadians(Lemniskate.DPHI));
      double minXC = lem.getMinimumX(Math.toRadians(Lemniskate.PHIA), Math.toRadians(
          Lemniskate.PHIE), Math.toRadians(Lemniskate.DPHI));
      double maxXC = lem.getMaximumX(Math.toRadians(Lemniskate.PHIA), Math.toRadians(
          Lemniskate.PHIE), Math.toRadians(Lemniskate.DPHI));
      reporter.write("");
      reporter.write("Hoehendifferenz: " + hdiff);
      reporter.write("Minimale Hoehe: " + hmin);
      reporter.write("min X (XA)    : " + minXC);
      reporter.write("max X (XE)    : " + maxXC);

      final String lemniskateFile = outputPath + "/Lemniskate.png";
      plot(lemniskateFile);
      pollAndReportImage(reporter, lemniskateFile, "Konstruktion");
      processedFiles.add(new ProcessedFile(outputPath + "/Lemniskate.png", ProcessedFile.
          TYPE_IMAGE));

      reporter.write("Optimizer: Steps: " + result.getOptimizer().getLapCount() + "    Zfk
          -Historie: " + result.getOptimizer().getHistoryTargetValues().size());

      plotTargetHistory(reporter, result, outputPath + "/LemTarget.png");
      processedFiles.add(new ProcessedFile(outputPath + "/LemTarget.png", ProcessedFile.
          TYPE_IMAGE));

      plotPenaltyHistory(reporter, result, outputPath + "/LemPenalty.png");
      processedFiles.add(new ProcessedFile(outputPath + "/LemPenalty.png", ProcessedFile.
          TYPE_IMAGE));

      plotQualityHistory(reporter, result, outputPath + "/LemQuality.png");
      processedFiles.add(new ProcessedFile(outputPath + "/LemQuality.png", ProcessedFile.
          TYPE_IMAGE));

      plotParameterHistory(reporter, result, outputPath + "/LemParameters.png");
      processedFiles.add(new ProcessedFile(outputPath + "/LemParameters.png",
          ProcessedFile.TYPE_IMAGE));
    }
  }

  private void plot(String fileName) {
    File file = new File(fileName);
    if (file.exists())
      file.delete();

    if (!file.exists()) {
      Plotter plotter = new Plotter(Lemniskate.getInstance(), 1000, 1000);
      plotter.plot(Display.getDefault(), fileName);
    }
  }}
```

```
public class LemniskateTarget implements ITargetEx {

  double currentQualityValue;
  double currentPenaltyValue;
```

```
  @Override
  public double evaluate(ParameterSet parameters, List<IRestriction> restrictions) throws
      OptimizationException {
    Lemniskate lem = Lemniskate.getInstance();
    lem.setParameters(parameters);

    // Berechnung des G"utewerts:
    currentQualityValue = lem.getHeightDiff(Math.toRadians(Lemniskate.PHIA), Math.
        toRadians(Lemniskate.PHIE), Math.toRadians(Lemniskate.DPHI));  // Guetewert

    // Summe der Penaltywerte aller Restriktionen:
    currentPenaltyValue = 0.0;
    for (IRestriction r : restrictions) {
      currentPenaltyValue += r.calcPenalty(parameters);  // Penaltywert
    }

    // Superposition:
    return currentQualityValue + currentPenaltyValue;
  }

  /**
   * Zfk-Wert aus dem letzten evaluate()
   */
  @Override
  public double getCurrentTargetValue() {
    return currentQualityValue + currentPenaltyValue;
  }

  /**
   * Guete-Wert aus dem letzten evaluate()
   */
  @Override
  public double getCurrentQualityValue() {
    return currentQualityValue;
  }

  /**
   * Penalty-Wert aus dem letzten evaluate()
   */
  @Override
  public double getCurrentPenaltyValue() {
    return currentPenaltyValue;
  }}
```

```
public class HeightConstraint extends Constraint {

  public HeightConstraint() {}

  public HeightConstraint(String name, double factor) {
    super(name, factor);
  }

  @Override
  public double calcPenalty(ParameterSet parameters) throws OptimizationException {
    Lemniskate lem = Lemniskate.getInstance();
    lem.setParameters(parameters);
    double hMin = lem.getMinimumHeight(Math.toRadians(Lemniskate.PHIA), Math.toRadians(
        Lemniskate.PHIE), Math.toRadians(Lemniskate.DPHI));

    if (hMin < Lemniskate.HMIN) {
      return (factor * Math.pow(Lemniskate.HMIN-hMin, 2.0));
    }
    if (hMin > Lemniskate.HMAX) {
      return (factor * Math.pow(Lemniskate.HMAX-hMin, 2.0));
    }
    return 0.0;
  }}
```

```
/**
 * Berechnungen an Lemniskatenlenker.
 * <p>
 * Der fixe Fusspunkt ist M im Ko-Ursprung. Die Gelenke in Umlaufrichtung sind
 * dann M-A-B-N. Der Auslegerpunkt ist C.
 * <p>
 * Der Abstand zwischen den Fundamentpunkten M und N ist L0, gegeben durch
 * die Koordinaten Nx und Ny. Die Grund-/Drucklenkerlaenge ist L1, die Zuglenkerlaenge ist
      L3,
 * der Abstand zwischen A und B ist L2, der Abstand von A zum Auslegerpunkt C ist L4.
 * <p>
 * Der Winkel phi24 zwischen den Strecken L2 und L4 wird als 180 Grad angenommen
 * (gestreckter, nicht-gekroepfter Ausleger).
 * <p>
 * Der Basiswinkel ist der Winkel phi10 des Drucklenkers L0 am Punkt M gegen die
     Horizontale.
 */
public class Lemniskate implements IDrawer {
  public static final double XA = 4000.0;
  public static final double XE = 9000.0;
  public static final double HMIN = 6000.0;
  public static final double HMAX = 9000.0;

  public static final double PHIA = 40.0;//70.0;
  public static final double PHIE = 89.0;//85.0;
  public static final double DPHI = 0.5;

  protected double nx, ny, l0, l1, l2, l3, l4;

  private double fx = 1.0;
  private double fy = 1.0;
  private double ox = 0.0;
  private double oy = 0.0;

  public static Lemniskate instance;

  private Lemniskate() {
  }

  public static Lemniskate getInstance() {
    if (instance == null)
      instance = new Lemniskate();
    return instance;
  }

  public void setParameters(ParameterSet parameters) {
    nx = parameters.getPhaenotypeValueByKey("Nx");
    ny = parameters.getPhaenotypeValueByKey("Ny");
    l1 = parameters.getPhaenotypeValueByKey("l1");
    l2 = parameters.getPhaenotypeValueByKey("l2");
    l3 = parameters.getPhaenotypeValueByKey("l3");
    l4 = parameters.getPhaenotypeValueByKey("l4");

    l0 = sqrt(sqr(nx) + sqr(ny));
  }

  /**
   * Koord. des Basispunkts
   * <p>
   * Trivial: (0;0)
   *
   * @return
   */
  public Vertex getPointM() {
    return new Vertex();       // immer (0;0)
  }

  /**
   * Zweiter Fundamentpunkt hinten
```

```
   *
   * @return
   */
  public Vertex getPointN() {
    return new Vertex(nx, ny, 0);  // nx ist immer negativ
  }

  /**
   * Lemniskatenpunkt A aufgrund von Grundlenkerwinkel phi10
   * @param phi10   Grundwinkel in Bogenma"s
   * @return
   */
  public Vertex getPointA(double phi10) {
    double ax = l1 * Math.cos(phi10);
    double ay = l1 * Math.sin(phi10);
    return new Vertex(ax, ay, 0);
  }

  /**
   * Lemniskatenpunkt B aufgrund von Grundlenkerwinkel phi10
   * @param phi10   Grundwinkel in Bogenma"s
   * @return
   */
  public Vertex getPointB(double phi10) {
    double phi20 = getPhi20(phi10);
    double bx = l1 * Math.cos(phi10) + l2 * Math.cos(phi20);
    double by = l1 * Math.sin(phi10) + l2 * Math.sin(phi20);
    return new Vertex(bx, by, 0);
  }

  /**
   * Lemniskatenpunkt C aufgrund von Grundlenkerwinkel phi10
   * @param phi10   Grundwinkel in Bogenma"s
   * @return
   */
  public Vertex getPointC(double phi10) {
    double phi20 = getPhi20(phi10);
    double cx = l1 * Math.cos(phi10) + l4 * Math.cos(phi20 - AngleMath.PI);
    double cy = l1 * Math.sin(phi10) + l4 * Math.sin(phi20 - AngleMath.PI);
    return new Vertex(cx, cy, 0.0);
  }

  /**
   * Winkel von L0 gegen"uber der pos. x-Richtung
   * @return
   */
  public double getPhi00() {
    if (Math.abs(nx) > 1E-3)
      return Math.atan2(ny, nx) - AngleMath.PI;
    return Double.POSITIVE_INFINITY;
  }

  /**
   * Winkel von L2 gegen"uber der pos. x-Richtung
   * @param phi10   Grundwinkel in Bogenma"s
   * @return
   */
  public double getPhi20(double phi10) {
    double phi00 = getPhi00();          // < 0

    double a = l0 * Math.cos(phi00) + l1 * Math.cos(phi10);
    double b = l0 * Math.sin(phi00) + l1 * Math.sin(phi10);

    double w1 = sqrt(4 * sqr(l3) * (sqr(a) + sqr(b)) - sqr(sqr(l2) - sqr(l3) - sqr(a) -
        sqr(b)));

    double cosphi20_1 = (-a * (sqr(a) + sqr(b) + sqr(l2) - sqr(l3)) + b * w1) / (2 * l2 *
        (sqr(a) + sqr(b)));
    double cosphi20_2 = (-a * (sqr(a) + sqr(b) + sqr(l2) - sqr(l3)) - b * w1) / (2 * l2 *
        (sqr(a) + sqr(b)));
```

```
    double sinphi20_1 = (-b * (sqr(a) + sqr(b) + sqr(l2) - sqr(l3)) + a * w1) / (2 * l2 *
        (sqr(a) + sqr(b)));
    double sinphi20_2 = (-b * (sqr(a) + sqr(b) + sqr(l2) - sqr(l3)) - a * w1) / (2 * l2 *
        (sqr(a) + sqr(b)));

    double[] phi20_candidates = new double[4];
    phi20_candidates[0] = AngleMath.asincos(sinphi20_1, cosphi20_1);
    phi20_candidates[1] = AngleMath.asincos(sinphi20_2, cosphi20_1);
    phi20_candidates[2] = AngleMath.asincos(sinphi20_1, cosphi20_2);
    phi20_candidates[3] = AngleMath.asincos(sinphi20_2, cosphi20_2);

    for (int i=0; i<4; i++) {
      if (phi20_candidates[i] > AngleMath.PI_HALF && phi20_candidates[i] < AngleMath.PI)
        return phi20_candidates[i];
    }
    return Double.NaN;
  }
  /**
   * Winkel von L3 gegenueber der pos. x-Richtung
   * @param phi10   Grundwinkel in Bogenmass
   * @return
   */
  public double getPhi30(double phi10) {
    // TODO
    return Double.POSITIVE_INFINITY;
  }

  /**
   * Hoehendifferenz der Lemniskatenbahn zwischen XA und XE
   *
   * @param startPhi10    Start-Grundwinkel in Bogenmass
   * @param endPhi10      End-Grundwinkel in Bogenmass
   * @param deltaPhi      Deltawinkel in Bogenma"s
   *
   */
  public double getHeightDiff(double startPhi10, double endPhi10, double deltaPhi) {
    List<Vertex> points = new ArrayList<Vertex>();

    for (double phi10 = startPhi10; phi10 < endPhi10; phi10 += deltaPhi) {
      Vertex cpoint = getPointC(phi10);
      if (cpoint.x >= XA && cpoint.x <= XE)  // nur Punkte innerhalb des Wippbereichs
        points.add(cpoint);
    }

    if (points.size() > 2) {
      double yMin = Double.MAX_VALUE;
      double yMax = -yMin;

      for (Vertex p : points) {
        if (p.y < yMin)
          yMin = p.y;
        if (p.y > yMax)
          yMax = p.y;
      }

      return Math.abs(yMax - yMin);
    } else {
      System.out.println("WARNING: not enough points!");
      return Double.POSITIVE_INFINITY;
    }
  }

  /**
   * Minimale Hoehe von Punkt C zwischen XA und XE
   *
   * @param startPhi10    Start-Grundwinkel in Bogenma"s
   * @param endPhi10      End-Grundwinkel in Bogenma"s
   * @param deltaPhi      Deltawinkel in Bogenma"s
   *
   */
```

```
  public double getMinimumHeight(double startPhi10, double endPhi10, double deltaPhi) {
    double hMin = Double.MAX_VALUE;
    for (double phi10=startPhi10; phi10<=endPhi10; phi10+=deltaPhi) {
      Vertex c = getPointC(phi10);
      if (c.x >= XA && c.x <= XE)
        if (c.y < hMin)
          hMin = c.y;
    }
    return hMin;
  }

  /**
   * Minimaler Wipp-Punkt von Punkt C
   *
   * @param startPhi10   Start-Grundwinkel in Bogenma"s
   * @param endPhi10     End-Grundwinkel in Bogenma"s
   * @param deltaPhi     Deltawinkel in Bogenma"s
   *
   */
  public double getMinimumX(double startPhi10, double endPhi10, double deltaPhi) {
    double xMin = Double.MAX_VALUE;
    for (double phi10=startPhi10; phi10<=endPhi10; phi10+=deltaPhi) {
      Vertex c = getPointC(phi10);
      if (c.x < xMin)
        xMin = c.x;
    }
    return xMin;
  }

  /**
   * Maximaler Wipp-Punkt von Punkt C
   *
   * @param startPhi10   Start-Grundwinkel in Bogenma"s
   * @param endPhi10     End-Grundwinkel in Bogenma"s
   * @param deltaPhi     Deltawinkel in Bogenma"s
   *
   */
  public double getMaximumX(double startPhi10, double endPhi10, double deltaPhi) {
    double xMax = -Double.MAX_VALUE;
    for (double phi10=startPhi10; phi10<=endPhi10; phi10+=deltaPhi) {
      Vertex c = getPointC(phi10);
      if (c.x > xMax)
        xMax = c.x;
    }
    return xMax;
  }

  /**
   * Quadrat
   * @param d
   * @return
   */
  protected double sqr(double d) {
    return d * d;
  }

  /**
   * Quadratwurzel
   * @param d
   * @return
   */
  protected double sqrt(double d) {
    return Math.sqrt(d);
  }

  @Override
  public void draw(Display display, GC gc, Rectangle bounds) {
    Color black = new Color(display, 0, 0, 0);
    Color red = new Color(display, 255, 0, 0);
    Color green = new Color(display, 0, 255, 0);
```

```
    Color blue = new Color(display, 0, 0, 255);
    Color yellow = new Color(display, 255, 255, 0);

    // Skalierung:
    double factor = 15.0;
    scaleDrawing(-nx, bounds.height*factor, 1.0/factor, 1.0/factor);

    // Wippbereich:
    gc.setForeground(green);
    drawLine(gc, 0, 0, 0, oy);

    gc.setForeground(red);
    drawLine(gc, XA, 0, XA, oy);
    drawLine(gc, XE, 0, XE, oy);
    drawLine(gc, XA, HMIN, XE, HMIN);

    // Kinematik:
    gc.setForeground(black);
    for (double phi=60.0; phi<90.0; phi+=2.0) {
      double phi10 = Math.toRadians(phi);
      gc.setForeground(black);
      drawLine(gc, 0, 0, getPointA(phi10).x, getPointA(phi10).y); // M - A
      drawLine(gc, nx, ny, getPointB(phi10).x, getPointB(phi10).y); // N - B
      gc.setForeground(blue);
      drawLine(gc, getPointA(phi10).x, getPointA(phi10).y, getPointC(phi10).x, getPointC(
          phi10).y); // A - C
      gc.setForeground(yellow);
      drawLine(gc, getPointB(phi10).x, getPointB(phi10).y, getPointA(phi10).x, getPointA(
          phi10).y); // B - A
    }

    green.dispose();
    red.dispose();
    black.dispose();
    yellow.dispose();
    blue.dispose();
  }

  private void drawLine(GC gc, double x1, double y1, double x2, double y2) {
    x1 = (x1 + ox) * fx;
    y1 = (-y1 + oy) * fy;
    x2 = (x2 + ox) * fx;
    y2 = (-y2 + oy) * fy;
    gc.drawLine((int) x1, (int) y1, (int) x2, (int) y2);
  }

  private void scaleDrawing(double ox, double oy, double fx, double fy) {
    this.ox = ox;
    this.oy = oy;
    this.fx = fx;
    this.fy = fy;
  }
}
```

Literaturverzeichnis

[AH11] Allemang, Dean ; Hendler, Jim: *Semantic Web for the Working Ontologist*. Morgan Kaufmann, 2011. – 335–338 S. – ISBN 9780123859655

[Alt11] Alt, Walter: Nichtlineare Optimierung: Eine Einführung in Theorie, Verfahren und Anwendungen. In: *Nichtlineare Optimierung: Eine Einführung in Theorie, Verfahren und Anwendungen*. Wiesbaden : Vieweg+Teubner, 2011. – ISBN 3834815586

[And03] Andrei, Neculai: *Nonlinear Optimization Applications Using the GAMS Technology*. Springer, 2003. – ISBN 9781461467977. – http://www.springer.com/series/7393 (02.07.15)

[ANS15] ANSYS: *ANSYS FEM Software*. Website, 2015. – http://www.ansys.com/ (29.08.2015)

[Atk07] Atkinson, John: Evolving explanatory novel patterns for semantically-based text mining. In: *Natural Language Processing and Text Mining*. Springer, 2007. – ISBN 184628175X, S. 145–169

[Bay68] Bayer, Rudolf: *MPL: Mathematical Programming Language*. Verlag PN, 1968

[Bik14] Bikakis, Antonis: *Rules on the Web: From Theory to Applications: 8th International Symposium, RuleML 2014*. Springer, 2014. – ISBN 3–31909–869–1

[Bis06] Bishop, Johannes: *AIMMS - Optimization Modeling*. Paragon Decision Technology, 2006. – 316 S. – ISBN 9781847539120.

– http://download.aimms.com/aimms/download/manuals/AIMMS3_OM.pdf (29.08.2015)

[Bis08] Bishop, Judith: *C# 3.0 Entwurfsmuster*. Köln : O'Reilly, 2008. – ISBN 3897218674

[Bis09] Bishop, Johannes: *AIMMS 3 – Optimization Modeling*. Harlem : Paragon Decision Technology, 2009. – http://www.aimms.com/downloads/manuals/user-s-guide (27.08.15)

[Bis10] Bishop, Christopher: Pattern Recognition and Machine Learning. In: *Pattern Recognition and Machine Learning*, Springer, 2010. – ISBN 9780387310732

[BKM10] Brooke, Anthony ; Kendrick, David ; Meeraus, Alexander: *Gams: A User's Guide*. Washington, DC : Verlag GAMS Development Corporation, 2010

[Bob64] Bobrow, Daniel G.: *Natural language input for a computer problem solving system*. Massachusetts : Massachusetts Institute of Technology, 1964

[Bob94] Bobrow, Daniel G.: *Artificial Intelligence in Perspective, Special Issues of Artificial Intelligence*. Massachusetts Institute of Technology, 1994. – ISBN 0–262–52186–5

[BSS05] Burke, Edmund. K. ; Silva, J. Dario L. ; Soubeiga, Eric: Multi-Objective Hyper-Heuristic approaches for space allocation and timetabling. In: *Metaheuristics: Progress as Real Problem Solvers*, 2005. – ISSN 1387666X, S. 129–158

[BV06] Bartsch, Renate ; Vennemann, Theo: *Grundzüge der Sprachtheorie: eine linguistiche Einführung*. Michigan : Niemeyer Max Verlag GmbH, 2006. – ISBN 3–484–22032–5

[CDM98] Colorni, Alberto ; Dorigo, Marco ; Maniezzo, Vittorio: Metaheuristics for high school timetabling. In: *Computational Optimization and Applications*, 1998. – ISBN 0926–6003, S. 275–298

[CG13] Clausen, Uwe ; Geiger, Christian: *Verkehrs- und Transportlogistik*. Berlin : Springer Vieweg, 2013. – ISBN 978–3–540–34299–1

[Cla54] Clauberg, Johannes: *JElementa Philosophiae sive Ontosophia*. Universität Groningen, 1654

[CS04] Cunningham, Kevin ; Schrage: *The Lingo Algebraic ModelingLanguage*. Norwell MA USA : Kluwer Academic Publishers, 2004

[Dam14] Dambeck, Holger: *Das Kreuz mit dem Kreis*. Spiegel-Verlag Rudolf Augstein, 2014. – http://www.spiegel.de/wissenschaft/mensch/raetsel-der-woche-kreis-unbekannter-groesse-a-1005539.html (30.11.2014)

[Dat95] Date, Christopher J.: An Introduction to Database Systems. In: *An Introduction to Database Systems* 1 (1995), Nr. c, S. 839. – ISBN 0201142015. – http://books.google.fr/books?id=dv2wQgAACAAJ (12.10.14)

[DD05] Domschke, Wolfgang ; Drexl, Andreas: *Einführung in Operations Research*. Berlin : Springer, 2005. – ISBN 3–540–23431–4

[Deb10] Deb, Kalyanmoy: *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, 2010

[Den12] Dengel, Andreas: Semantische Technologien. In: *SpringerLink Bücher* (2012), 231 – 256. `http://link.springer.com/10.1007/978-3-8274-2664-2`. ISBN 978–3–8274–2663–5

[Dre03] Dreher, Axel: *Die Kreditvergabe von IWF und Weltbank: Ursachen und Wirkungen aus politisch-ökonomischer Sicht*. Berlin : wvb Berlin, 2003. – ISBN 3–936846–54–5

[EBL03] Ellinger, Theodor ; Beuermann, Günter ; Leisten, Rainer: *Operations Research*. 6.Auflage. Berlin : Springer, 2003. – ISBN 978–3–540–64847–5

[Ehr10] Ehrgott, Matthias: Multicriteria Optimization. In: *Multicriteria Optimization*. Berlin Heidelberg : Springer, Januar 2010. – ISBN 3642059759, S. 340

[ES07] Euzenat, Jérôme ; Shvaiko, Pavel: *Ontology matching*. Berlin : Springer, 2007. – 333 S. – ISBN 3540496114. – http://book.ontologymatching.org/ (02.04.2015)

[FD95] Feldman, Ronen ; Dagan, Ido: Knowledge Discovery in Texts. In: *First International Conference on Knowledge Discovery (KDD)*, 1995

[FF08] Freeman, Eric ; Freeman, Elisabeth: *Entwurfsmuster von Kopf bis Fuß*. Köln : O'Reilly, 2008. – 672 S. – ISBN 978–3–89721–421–7

[FGK02] Fourer, Robert ; Gay, David ; Kernighan, Brian: *AMPL: a modeling language for mathematical programming*. Duxbury Resource Center : Learning, Cengage, 2002. – 540 S. – ISBN 0–534–38809–4

[Fis00] Fischer, Gerd: *Lineare Algebra*. 12. Auflag. Wiesbaden : Vieweg+Teubner Verlag, 2000. – ISBN 9–783–52887217–5

[Fun13] Funke, Stefan: Numerik III (Numerische Lineare Algebra und Optimierung) / Universität Ulm. Ulm, 2013. – Forschungsbericht

[Gaw11] Gawrychowski, Pawel: Optimal pattern matching in LZW compressed strings. In: *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, 2011, 362–372

[Gel14] Geldermann, Jutta: *Multikriterielle Optimierung*. 2014. – http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/technologien-methoden/Operations-Research/Mathematische-Optimierung/Multikriterielle-Optimierung (26.08.2015)

[Gem93] Gemünden, Hans G.: *Information: Bedarf, Analyse und Verhalten*. Stuttgart : Schäffer-Pöschel Verlag, 1993

[GG95] Grunow, Martin ; Günther, Hans-Otto: AMPL. In: *OR Spektrum* 17 (1995), Nr. 1, S. 1–3. http://dx.doi.org/10.1007/BF01719723. – DOI 10.1007/BF01719723. – ISSN 01716468

[GHJV96] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*. Addison-Wesley Verlag, 1996. – 504 S. – ISBN 3827321999

[GHPS02] Guéret, Christelle ; Heipcke, S ; Prins, Christian ; Sevaux, Marc: *Applications of Optimization with Xpress-MP*. 2002. – http://examples.xpress.fico.com/example.pl#mosel_model (19.05.2015)

[GK02] Geiger, Carl ; Kanzow, Christian: *Theorie und Numerik restringierter Optimierungsaufgaben*. Berlin : Springer, 2002. – ISBN 3–540–42790–2

[GPS02] Guéret, Christelle ; Prins, Christian ; Sevaux, Marc: *Applications of Optimization with Xpress-MP*. Sevaux : Dash Optimization Ltd., 2002. – ISBN 0–9543503–0–8

[Gre04] Greco, Salvatore: Multiple Criteria Decision Analysis - State of Art Surveys. In: *Multiple Criteria Decision Analysis - State of Art Surveys*, Springer, 2004. – ISBN 038723067X

[Gru95] Gruber, Thomas: Toward principles for the design of ontologies used for knowledge sharing. In: *International Journal of Human-Computer Studies* 43 (1995), Nr. 5-6, 907–928. http://dx.doi.org/10.1006/ijhc.1995.1081. – ISBN 10715819

[Gua98] Guarino, Nicola: Formal Ontology and Information Systems. In: *FOIS'98*, 1998. – ISBN 9051993994, 3–15

[HKHM13] Hussain, Iftikhar ; Kausar, Samina ; Hussain, Liaqat ; Muhammad, Asif K.: Improved Approach for Exaxt Pattern Matching. In: *Journal of Computer Science Issues*, 2013. – http://ijcsi.org/papers/IJCSI-10-3-1-59-65.pdf (05.02.2014)

[HKRS07] Hitzler, Pascal ; Krötzsch, Markus ; Rudolph, Sebastian ; Sure, York: *Semantic Web: Grundlagen.* Berlin : Springer, 2007. – ISBN 3–540–339930

[HL97] Hillier, Frederick S. ; Liebermann, Gerald J.: *Operations Research.* München Wien : Oldenbourg Verlag, 1997. – ISBN 3–48623987–2

[HLMS08] Hepp, Martin ; Leenheer, Pieter D. ; Moor, Aldo D. ; Sure, York: *Ontology Management, Semantic Web, Semantic Web Services, and Business Applications.* Bd. 7. Springer, 2008. – xix, 293 p. S. – ISBN 038769899X. – http://www.springer.com/computer/database+management+& +information+retrieval/book/978-0-387-69899-1 (01.03.2015)

[HNP05] Hotho, Andreas ; Nürnberger, Andreas ; Paass, Gerhard: A Brief Survey of Text Mining. In: *LDV Forum - GLDV Journal for Computational Linguistics and Language Technology* 20 (2005), S. 19–62. – ISBN 0175–1336. – http://www.kde.cs.uni-kassel.de/hotho/pub/2005/hotho05TextMining.pdf (20.11.2014)

[Hro04] Hromkovič, Juraj: Randomisierte Algorithmen: Methoden zum Entwurf von zufallsgesteuerten Systemen für Einsteiger. In: *Randomisierte Algorithmen: Methoden zum Entwurf von zufallsgesteuerten Systemen für Einsteiger*, Vieweg+Teubner, Juni 2004. – ISBN 3519004704

[HSD01] Hart, Richard O. ; Stork, Peter E. ; Duda, David G.: Pattern Classification. In: *Pattern Classification.* New York : Wiley, November 2001. – ISBN 0–471–05669–3

[HSW97] Heijst, Gertjan van ; Schreiber, Guus ; Wielinga, Bob: Using explicit ontologies in KBS development. In: *International Journal of Human-Computer Studies* 46 (1997), Nr. 2-3, S. 183–292. – ISBN 1071–5819. – http://www.sciencedirect.com/science/article/B6WGR-45M91MP-V/1/0a5ed7f5c878f5c4b332de9c453fead7 (17.06.2014)

[Jah04a] Jahn, Johannes: Vector Optimization: Theory, applications and extensions. In: *Vector Optimization: Theory, applications and extensions*. Heidelberg : Springer, 2004

[Jah04b] Jahn, Johannes: *Vector Optimization: Theory, applications, and extensions Title*. Heidelberg : Springer, 2004

[JLB+14] John, Christian ; Lepich, Thomas ; Beitz, Bernard ; Moller, Reinhard ; Tutsch, Dietmar: A probabilistic approach to pattern-matching based on non-linear parameter optimization. In: *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, 2014, 1–5

[JM13] John, Christian ; Moller, Reinhard: A probabilistic approach to pattern-matching based on a dynamic rule-driven system. In: *2013 IEEE Global High Tech Congress on Electronics*, 2013. – ISBN 978–1–4799–3209–2, 112–113

[JTL+15] John, Christian ; Tutsch, Dietmar ; Lepich, Thomas ; Beitz, Bernard ; Möller, Reinhard: A Criteria Transformation Approach to Pattern Matching based on Non-Linear Parameter Optimization. In: *Journal of Intelligent Systems* 24 (2015), Nr. 2, S. 1–15. – ISSN 2191–026X. – http://www.degruyter.com/view/j/jisys.2015.24.issue-2/jisys-2014-0114/jisys-2014-0114.xml (30.04.2014)

[JTR+14] John, Christian ; Tutsch, Dietmar ; Reinhard, M ; Lepich, Thomas ; Beitz, Bernard: A Criteria Transformation Approach to Timetabling based on Non-Linear Parameter Optimization. In: *Proceedings of the 10th International Conference on the Practice and Theory of Automated Timetabling*, 2014, 252–268

[Juk12] Jukna, Stasys: *Boolean function complexity. Advances and frontiers*. Springer, 2012. – ISBN 978–3–642–24507–7

[Kal04] Kallrath, Josef: *Modeling Languages in Mathematical Optimization*. Kluwer Academic Publishers, 2004. – 440 S. – ISBN

1402075472. – http://books.google.com/books?hl=en&lr=&id=wJYART7VYe8C&pgis=1 (17.05.2015)

[Kal12] Kallrath, Josef: *Algebraic Modeling Systems – Modeling and Solving Real World Optimization Problems*. Bd. 104. Springer, 2012. – ISBN 978–3–642–23591–7. – http://link.springer.com/10.1007/978-3-642-23592-4 (18.05.2015)

[Kal13] Kallrath, Josef: *Gemischt-ganzzahlige Optimierung: Modellierung in der Praxis: Mit Fallstudien aus Chemie, Energiewirtschaft, Papierindustrie, Metallgewerbe, Produktion und Logistik*. Berlin Heidelberg : Springer Spektrum, 2013. – ISBN 978–3–658–00690–7

[KH13] Kelly, John E. ; Hamm, Steve: *IBM's Watson and the Era of Cognitive Computing*. Columbia Business School Publishing, 2013. – ISBN 0–231–16856–X

[Kis23] Kissinger, Henry: *Zitat: Kondensat*. 1923

[KKL+93] Klatte, R. ; Kulisch, U. ; Lawo, C. ; Rauch, M. ; Wiethoff, A.: *C-XSC, A C++ Class Library for Extended Scientific Computing*. Berlin : Springer-Verlag, 1993

[KN09] Krumke, Sven O. ; Noltemeier, Hartmut: *Graphentheoretische Konzepte und Algorithmen*. Wiesbaden : Vieweg Teubner, 2009. – 444 S. – ISBN 3834818496

[Kol12] Kolbenheyer, Erwin G.: *Amor Dei: Ein Spinoza-Roman*. Ulan Press, 2012. – ISBN B009F7SZKG

[Kon41] Konietschke, W.: Über Last- und Eigengewichtsausgleich an Wippkranen. In: *Ingenieur-Archiv* 12 (1941), Nr. 3, S. 133–157. http://dx.doi.org/10.1007/BF02084692. – DOI 10.1007/BF02084692. – ISSN 00201154

[Kur65] Kurth, F.: *Unstetigförderer I*. Berlin : VEB Verlag Technik, 1965

[Lai12] Laird, John E.: The Soar Cognitive Architecture. In: *AISB Quarterly* 171 (2012), Nr. 134, S. 224–235. ISBN 0262122960. – http://web.eecs.umich.edu/ soar/sitemaker/docs/-pubs/AISB_quarterly_134.pdf (20.03.2015)

[LD13] Lieser, Jürgen ; Dijkzeul, Dennis: *Handbuch Humanitäre Hilfe*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2013. – ISBN 978–3–642–32289–1. – http://link.springer.com/10.1007/978-3-642-32290-7 (26.01.2015)

[LM94] Lehner, Franz ; Maier, Ronald: *Information in Betriebswirtschaftslehre, Informatik und Wirtschaftsinformatik*. Koblenz : Wissenschaftliche Hochschule für Unternehmensführung Koblenz, 1994

[LPBM12] Lapègue, Tanguy ; Prot, Damien ; Bellenguez-Morineau, Odile: A Tour Scheduling Problem with Fixed Jobs: use of Constraint Programming. In: *Practice and Theory of Automated Timetabling*, 2012

[MAK07] Michiels, Wil ; Aarts, Emile ; Korst, Jan: Theoretical Aspects of Local Search. In: *Theoretical Aspects of Local Search*, 2007. – ISBN 9783540358534, S. 237

[Mat03] Matheplanet: *Das Ziegenproblem*. 2003. – http://matheplanet.com/matheplanet/nuke/html/dl.php?id=17&1139827476 (13.12.2014)

[May81] Mayer, S.: *Die Entwicklung von Hafen- und Werkkranen*. Berlin : Hebezeuge und Fördermittel, 1981. – 102–104 S.

[Mei05] Meister, Andreas: *Numerik linearer Gleichungssysteme*. Wiesbaden : Vieweg, 2005. – ISBN 3–528–13135–7

[MG15] Milonia, Eliseo ; Gülcan, Ahmet: *Docs2DB*. Wuppertal, 2015

[Mie98] Miettinen, Kaisa: *Nonlinear Multiobjective Optimization*. Springer, 1998. – ISBN 978–1–4615–5563–6

[Mie99] Miettinen, Kaisa: *Nonlinear multiobjective optimization.* Bd. 12. Springer, 1999. – 320 S. – ISBN 978–0–7923–8278–2. – http://books.google.com/books?id=ha_zLdNtXSMC (23.10.2014)

[MR12] Müller, Tomáš ; Rudová, Hanna: Real-life Real-life Curriculum-based Timetabling. In: *Proceedings of the 9th international conference on the Practice And Theory of Automated Timetabling*, 2012

[Mül14] Müller, Tomáš: Real-life examination timetabling. In: *Journal of Scheduling*, 2014. – ISSN 10946136

[MW05] Mehler, Alexander ; Wolff, Christian: Einleitung: Perspektiven und Positionen des Text Mining [Einf{ü}hrung in das Themenheft Text Mining des LDV-Forum]. In: *LDV-Forum* 20 (2005), Nr. 1, S. 1–18. ISBN 0175–1336. – http://epub.uni-regensburg.de/6844/ (09.07.2014)

[MW14] Mühlenthaler, Moritz ; Wanka, Rolf: Fairness in academic course timetabling. In: *Annals of Operations Research*, 2014. – ISSN 02545330, S. 1–18

[NAS15] NASTRAN: *MSCSoftware FEM-Programm.* Website, 2015. – http://www.mscsoftware.com/de/product/msc-nastran (29.08.2015)

[NEO] NEOS: *NEOS Server.* Website, . – http://www.neos-server.org/neos/report.html (2015-05-27)

[Nou45] Nourse, John: *A Treatise of Algebra.* 1745. – http://www.mathematische-basteleien.de/ladder.htm#References (13.12.2014)

[Ock11] Ockham, William von: *Ockhams Rasiermesser.* Prinzip der Parsimonie, 1311. – ISBN 0268019401. – http://plato.stanford.edu/entries/ockham/ (25.02.2014)

[Ott98] Ottmann, Thomas: *Prinzipien des Algorithmenentwurfs.* Heidelberg Berlin : Spektrum Akademischer Verlag, 1998. – 229 S. – ISBN 3827402387

[OW02] Ottmann, Thomas ; Widmayer, Peter: *Algorithmen und Datenstrukturen.* Heidelberg Berlin : Spektrum Akademischer Verlag, 2002. – ISBN 3827428033

[PD08] Pomberger, Gustav ; Dobler, Heinz: *Algorithmen und Datenstrukturen – Eine systematische Einführung in die Programmierung.* München : Pearson Studium, 2008. – ISBN 3827372682

[Pes12] Pesant, Gilles: A Constraint Programming Approach to the Traveling Tournament Problem with Predefined Venues. In: *Practice and Theory of Automated Timetabling.* Son, 2012, S. 303–315

[Pru82] Pruefer, H.-P.: *Parameteroptimierung - Ein Werkzeug des rechnerunterstützten Konstruierens.* Bochum : Ruhr-Univ. Bochum, 1982. – ISBN 3–89194–019–X

[PS00] Papadimitriou, Christos H. ; Steiglitz, Kenneth: Combinatorial Optimization - Algorithms and Complexity. In: *Combinatorial Optimization - Algorithms and Complexity.* Dover, Februar 2000

[PUZ15] PUZZLE: *PUZZLE.* 2015. – http://www.iml.fraunhofer.de/content/dam/iml/de/documents/OE120/PUZZLE.pdf (13.12.2014)

[Raf03] Raffensperger, Fritz: *User's Manual.* 2003. – http://faculty.oxy.edu/lengyel/M372/LindoUsersManual.pdf (26.09.2014)

[RB93] Roy, Bernard ; Bouyssou, Denis: *Aide multicritère à la décision.* Paris : Economica, 1993

[RB12a] Roelofs, Marcel ; Bishop, Johannes: *AIMMS 3.13 – Language Reference.* 2012. – http://www.aimms.com/aimms/download/manuals/aimms_ref.pdf (02.06.2015)

[RB12b] Roelofs, Marcel ; Bishop, Johannes: *AIMMS 3.13 – User's Guide*. 2012. – http://www.aimms.com/downloads/manuals/users-guide/ (09.04.2015)

[REEB13] Rui, Li ; Emmerich, T.M. M. ; Eggermont, Jeroen ; Bäck, Thomas: Mixed Integer Evolution Strategies for Parameter Optimizations. In: *EVOLUTIONARY COMPUTATION* Bd. 21, 2013, S. 29–64

[Rie97] Riehle, Dirk: *Entwurfsmuster für Softwarewerkzeuge*. Bonn : Addison Wesley, 1997. – ISBN 3827311470

[Ros60] Rosenbrock, HH: An automatic method for finding the greatest or least value of a function. In: *The Computer Journal* 3 (1960), Nr. 3, S. 175–184. – ISSN 0010–4620. – http://comjnl.oxfordjournals.org/content/3/3/175.short (12.01.2014)

[RS13] Raphael, Benny ; Smith, Ian F.: *Engineering Informatics: Fundamentals of Computer-Aided Engineering*. Wiley, 2013. – ISBN 9781119953418

[RT06] Rahayu, Johanna W. ; Taniar, David: *Web Semantics Ontology*. Idea Group Publishing, 2006. – 2006 S. – ISBN 1–591–40905–5

[Rul15] RuleML: *RuleML*. Website, 2015. – http://wiki.ruleml.org (2015-08-27)

[Sch05] Schulz, Stefan: Automatische Verarbeitung medizinischer Sprache. In: *Symposium*. Köln : Deutsche Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie e.V., 2005

[Sch08] Scheithauer, Guntram: *Zuschnitt- und Packungsoptimierung: Problemstellungen, Modellierungstechniken, Lösungsmethoden*. Vieweg+Teubner Verlag, 2008. – 191–230 S. – ISBN 383510215X

[Sch13] Schrage, Linus: *The Modeling Language and Optimizer*. 2013. – http://www.lindo.com/index.php?option=com_content&view=article&id=38&Itemid=24 (02.06.2015)

[Sch14] Schrage, Linus: *API 9.0 User Manual*. 2014. – http://www.lindo.com/index.php?option=com_content&view=article&id=38&Itemid=24 (02.06.2015)

[Sch15] Schrage, Linus: *Optimization Modeling with LINGO*. Verlag LINDO, 2015. – ISBN 1–893355–00–4

[SE39] Saint-Exupéry, Antoine D.: *Wind, Sand und Sterne*. Dessau : Durst Karl Rauch Verlag, 1939

[Sed02] Sedgewick, Robert: *Algorithmen*. Addison-Wesley Verlag, 2002. – ISBN 3827370329

[Sim45] Simpson, Thomas: *A Treatise of Algebra*. printed for John Nourse, 1745

[Sip12] Sipser, Michael: *Introduction to the Theory of Computation*. Course Technology, 2012. – ISBN 053494728X

[SKK10] Schmidt, Jörn ; Klüver, Christina ; Klüver, Juergen: Programmierung naturanaloger Verfahren: Soft Computing und verwandte Methoden. In: *Programmierung naturanaloger Verfahren: Soft Computing und verwandte Methoden*, Vieweg+Teubner, 2010. – ISBN 3834808229

[SM09] Suhl, Leena ; Mellouli, Taïeb: *Optimierungssysteme – Modelle, Verfahren, Software, Anwendungen*. Dordrecht : Springer, 2009. – ISBN 978–3–642–01580–9

[SOA15] SOAR: *Soar*. Website, 2015. – http://soar.eecs.umich.edu/ (29.08.2015)

[SS02] Saake, Gunter ; Sattler, Kai-Uwe: *Algorithmen und Datenstrukturen – Eine Einführung mit Java*. Heidelberg : dpunkt, 2002. – ISBN 3864901367

[SSS09] Sure, York ; Staab, Steffen ; Studer, Rudi: Handbook on Ontologies. In: *Handbook on Ontologies*. Springer, 2009. – ISBN 978–3–540–70999–2, S. 135–152. – http://link.springer.com/10.1007/978-3-540-92673-3 (28.09.2014)

[Stu08] Studer, Rudi: Ontologien. In: *Enzyklopädie der Wirtschaftsinformatik, Online-Lexikon*. 2008. – http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/daten-wissen/Wissensmanagement/Wissensmodellierung/Wissensreprasentation/Semantisches-Netz/Ontologien (19.04.2015)

[Stu11] Stuckenschmidt, Heiner: *Ontologien: Konzepte, Technologien und Anwendungen*. 2. Auflage. Berlin Heidelberg : Springer, 2011. – ISBN 3–642–05403–X

[TESW15] Tilkov, Stefan ; Eigenbrodt, Martin ; Schreier, Silvia ; Wolf, Oliver: *REST und HTTP: Entwicklung und Integration nach dem Architekturstil des Web*. Dordrecht : dpunkt.verlag, 2015. – ISBN 978–3–864–90120–1

[Vol73] Volmer, Johannes: *Getriebetechnik Lehrbuch*. Berlin : VEB Verlag Technik Berlin, 1973

[Vol99] Vollmer, Heribert: *Introduction to Circuit Complexity: a Uniform Approach*. Springer Verlag, 1999. – ISBN 3–540–64310–9

[Wal92] Walter, Wolfgang: *Analysis 2*. 3. Auflage. Berlin : Springer, 1992. – ISBN 9–783–54055385–4

[Wan06] Wanka, Rolf: *Approximationsalgorithmen – Eine Einführung*. Wiesbaden : Teubner, 2006. – ISBN 3519004445

[WAT15] WATSON: *WATSON*. Website, 2015. – http://www-05.ibm.com/de/watson/ (29.08.2015)

[Wei15] Weicker, Karsten: Evolutionäre Algorithmen. In: *Evolutionäre Algorithmen*, Springer Vieweg, Juli 2015. – ISBN 3658099577

[Wit59] Wittmann, Waldemar: *Unternehmung und Unvollkommene Information*. Wiesbaden : VS Verlag für Sozialwissenschaften, 1959. – ISBN 978–3–322–98247–6. – http://link.springer.com/10.1007/978-3-322-98938-3 (16.05.2014)

[WIZD05] Weiss, Sholom M. ; Indurkhya, Nitin ; Zhang, Tong ; Damerau, Fred J.: *Text mining: Predictive methods for analyzing unstructured information*. Springer, 2005. – 1–237 S. – ISBN 0387954333

[Woh05] Wohlers, Wolfgang: *Mehrstufige Optimierung Komplexer Struktur-Mechanischer Probleme*. Göttingen : Cuvillier Verlag, 2005. – ISBN 3–86537–479–4

[Wol11] Wolff, Matthias: Akustische Mustererkennung. In: *Akustische Mustererkennung*, TUDpress Verlag der Wissenschaften, 2011. – ISBN 3942710145

[WOL15] WOLFRAM: *Wolfram Alpha*. Wolfram Research. Website, 2015. – http://www.wolframalpha.com/ (29.08.2015)

[ZG91] Zimmermann, H.J. ; Gutsche, L.: *Multi-Criteria Analyse*. Heidelberg : Springer, 1991

[Zim08] Zimmermann, Hans-Jürgen: *Operations Research – Methoden und Modelle. Für Wirtschaftsingenieure, Betriebswirte und Informatiker*. Vieweg, 2008. – ISBN 978–3–8348–0455–6