Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik
Institut für Simulation und Graphik

Diplomarbeit

Design und Implementation einer Optimierungsschnittstelle für stochastische Modelle

Verfasser:

Marcus Müller-Dornieden

30. Oktober 2005

Betreuer:

Prof. Dr. Graham Horton (Universität Magdeburg)

Dipl.-Inf. Stefan Heller (DaimlerChrysler AG)

Universität Magdeburg
Fakultät für Informatik
Postfach 4120, D–39016 Magdeburg
Germany

Müller-Dornieden, Marcus:
Design und Implementation einer Optimierungsschnittstelle für stochastische Modelle
Diplomarbeit, Otto-von-Guericke-Universität
Magdeburg, 2005.

Danksagung

An dieser Stelle möchte ich mich bei den Menschen bedanken, die mich in den letzten Monaten unterstützt und zum Gelingen dieser Arbeit beigetragen haben:

Prof. Dr. Graham Horton und Stefan Heller für die interessante Themenstellung, sehr gute Betreuung, hilfreiche Ratschläge und konstruktive Kritik.

Dr. Stefan Greiner, der sich bereit erklärt hat, als Zweitgutachter dieser Arbeit zur Verfügung zu stehen.

Meinen Eltern, die immer für mich da waren und sind, zum Beispiel bei geduldigem Korrekturlesen.

Meiner Freundin für ihre Unterstützung und das Ertragen meiner Ab- sowie Anwesenheit während meiner Diplomarbeit.

Christian Eggers insbesondere dafür, dass er immer ein offenes Ohr für meine LaTeX-Probleme hatte.

Allen Mitarbeitern in der Abteilung REI/AA bei DaimlerChrysler für die angenehme und anregende Arbeitsatmosphäre während meiner Zeit in Stuttgart.

Abstrakt

Auf der Suche nach optimalen Einstellungen für komplexe Systeme ist die Optimierung mit stochastischen Modellen eine Möglichkeit, die in den letzten Jahren immer mehr an Bedeutung gewinnt. Dies hängt unter anderem mit der ständig steigenden Leistungsfähigkeit moderner Computer zusammen.

In dieser Diplomarbeit wird ein Einblick in Aufbau und Bestandteile sowie zusätzlich ein Überblick über verfügbare Algorithmen und Metaheuristiken der Optimierung stochastischer Modelle gegeben. Darauf aufbauend wurde eine Optimierungsschnittstelle für die Scripting-schnittstellen des Tools Expect implementiert. Hier wurde besonderes Augenmerk auf die Erweiterbarkeit gelegt. Für diese Schnittstelle wurden der Partikel Schwarm Algorithmus, der Genetische Algorithmus und ein lokaler Algorithmus auf Basis der FDSA- und SPSA-Gradientenschätzer umgesetzt.

Bei Experimenten mit drei verschiedenen stochastischen Modellen haben insbesondere die globalen Algorithmen, also der Genetische und der Partikel Schwarm Algorithmus, gute und robuste Ergebnisse geliefert.

Abstract

Simulation optimization is an efficient way of finding optimal parameter sets for complex systems, that cannot be optimized with other methods. In recent years this has been a field of research. One reason is the steady increase of computing power of modern PC's.

This thesis will identify parts and process of optimizing stochastic models and an overview of optimization algorithms, that can be used in simulation optimization, will be presented. With this knowledge an interface for optimizing stochastic models is created. Extendability was an important criteria for this interface. In this interface three algorithms a particle swarm algorithm, a genetic algorithm and a local algorithm based on FDSA and SPSA gradient estimators were implemented.

With different stochastic models especially the particle swarm and the genetic algorithm showed promising results. These algorithms were able to find optimal parameter regions with very naive configurations, while results of the local algorithm depended heavily on its configuration.

Inhaltsverzeichnis

Abbildungsverzeichnis **vii**

Tabellenverzeichnis **vii**

1 Einleitung **1**

1.1 Motivation 1

1.2 Rahmenbedingungen 2

1.3 Ziele und Aufgabenstellung 3

1.4 Vorgehensweise 4

2 Grundlagen **6**

2.1 Stochastische Modelle am Beispiel Petri-Netze 6

2.2 Expect – Ein integriertes Analyse- und Modellierungsframework 10

2.3 Struktur und Funktionalität der Expect-Skriptingschnittstelle 11

2.4 Optimierung stochastischer Modelle – Bestandteile und Ablauf 15

3 Optimierungsalgorithmen **19**

3.1 Auswahlkriterien 19

3.2 Lokale Optimierung – Hill Climbing 21

3.3 Globale Optimierungsalgorithmen 24

3.4 Metaheuristiken 28

3.5 Auswahl der Algorithmen 30

4 Implementierung der Optimierungsschnittstelle **32**

4.1 Das Optimierungsskript – Struktur und Bestandteile der Optimierung 32
4.1.1 Einbinden des Skriptings 33
4.1.2 Algorithmen – Kern der Optimierung 34
4.1.3 Zielfunktion & Abbruchkriterien 37
4.2 Der Optimierer – Koordination der Optimierung 41
4.3 Benutzerschnittstelle und Ergebnisausgabe 41

5 Experimente / Optimierung 45
5.1 Einfaches Petri-Netz-Modell 45
5.1.1 Modellbeschreibung und Parametrisierung 45
5.1.2 Ergebnisse 47
5.2 Life-Cycle-Modell 48
5.2.1 Modellbeschreibung und Parametrisierung 48
5.2.2 Ergebnisse 50
5.3 Logistik-Modell 51
5.3.1 Modellbeschreibung und Parametrisierung 51
5.3.2 Ergebnisse 53
5.4 Deterministisches Modell – Globale und Lokale Optimierung 54
5.4.1 Modellbeschreibung und Parametrisierung 54
5.4.2 Ergebnisse 55
5.5 Bewertung 55

6 Bewertung und Ausblick 57
6.1 Zusammenfassung 57
6.2 Bewertung 57
6.3 Ausblick 59
6.4 Schlusswort 60

A Screenshots 61

B Algorithmenparameter der Algorithmen 69

Literaturverzeichnis **73**

Abbildungsverzeichnis

2.1 Zwei Zustände eines Systems modelliert mit einem Petri-Netz 9
2.2 Überblick Expect-Funktionalität und DaVincy-Plattform 10
2.3 Bestandteile des Expect-Skriptings 12
2.4 Ablauf des Skriptings 14
2.5 Schematischer Ablauf und Bestandteile der Optimierung 16

3.1 Einfachster lokaler Suchalgorithmus 21
3.2 Ameisen-Algorithmus (siehe [DCG99] S. 3) 27

4.1 Trennung der Algorithmusfunktionalität von Konfigurationsdaten und Benutzeroberfläche 35
4.2 Zielfunktion und auslesbare Variable 38
4.3 Baumstrukturen für die Variablen der Zielfunktion 39
4.4 Definition einer Variablen in der Optimierung 42

5.1 Petri-Netz-Modell 1 46
5.2 Life-Cycle-Modell 48
5.3 Petri-Netz eines Logistik-Centers, in einer einfachen und der originalen Ausführung 52
5.4 Ablauf der Kommissionierung eines Teiles 52
5.5 Verlauf der Gleichung 5.4 von x = -100 bis x = 100 54

A.1 Screenshot Register Übersicht 62
A.2 Screenshot Register Scripting 63

A.3 Screenshot Register Nebenbedingungen 64
A.4 Screenshot Register Zielfunktion 65
A.5 Screenshot Register Optimierungsparameter 66
A.6 Screenshot Register Optimierung 67
A.7 Screenshot Register Ergebnisse 68

Tabellenverzeichnis

2.1 Vor- und Nachteile stochastischer Modelle 8
2.2 Permutationen 15

3.1 Eigenschaften von Algorithmen 20
3.2 Modellauswertungen lokale Suche (Einfache Nachbarschaftssuche, FDSA und SPSA) 23

4.1 Bit- und Graykodierung von Integer-Zahlen 36

5.1 Übersicht Ergebnisse des einfachen Petri-Netz-Modells 47
5.2 Zuordnung der Garantie- und Kulanzstufen im Hinblick auf Antriebsstrang- und Normalteile zu Fahrzeugalter und gefahrenen Kilometern 49
5.3 Ergebnisse des Life-Cycle-Modell 51
5.4 Ergebnisse des Logistik-Modells 53
5.5 Ergebnisse der Optimierung des deterministischen Modells 55

B.1 Anforderungen der Routen im Logistik-Modell 72

Kapitel 1

Einleitung

Dieses Kapitel gibt einen ersten Einblick in die Diplomarbeit. Es werden Motivation und Ziele beschrieben. Die Rahmenbedingungen und verfügbaren Werkzeuge bei DaimlerChrysler werden dargelegt. Abschließend wird die Vorgehensweise der Arbeit geschildert.

1.1 Motivation

Die Entwicklung neuer Produkte findet in immer kürzeren Zeiträumen statt. Dies trifft insbesondere auf die Automobilindustrie zu. Gleichzeitig steigen auch die Ansprüche an Qualität und Sicherheit der Produkte. Dadurch, dass ständig neue Sicherheits- und Fahrerassistenzsysteme entwickelt werden, steigt die Komplexität der Abläufe im Fahrzeug ständig. Um Fehler und damit Unfälle zu minimieren, müssten die vorhandenen Maßnahmen zur Qualitätssicherung beziehungsweise Gefahrenanalyse (z. B. Fahrversuche, Crash-Tests etc.) ausgebaut werden. Die Ziele, möglichst schnell, dabei aber mindestens genauso sicher, hochwertig und günstig neue Produkte zu entwickeln, widersprechen sich. Mit herkömmlichen Technologien sind sie nicht zu erreichen. Um dennoch möglichst schnell Produkte zu entwickeln, ist es unerlässlich die Leistungsfähigkeit moderner Computer auszunutzen.

Eine gute Möglichkeit dies zu erreichen ist die Analyse oder auch Simulation[1] von stochastischen Modellen. Mit diesen können auch schwierige und komplexe Gegebenheiten nachgebildet werden. Es lassen sich schnell und kostengünstig neue Ideen ausprobieren, die früher einen teuren Prototypen benötigt hätten. Versuche mit Prototypen lassen sich zwar nicht ganz eliminieren, aber die Anzahl der benötigten Versuche lässt sich auf ein Minimum reduzieren.

Je größer ein Modell ist und je mehr veränderbare Eigenschaften es hat, desto komplexer wird es. Damit steigen auch die Schwierigkeiten für Experten alle Abhängigkeiten zu überblicken die sich ergeben und somit gute Parametersätze zu finden. Ab einer bestimmten Größe wird dies praktisch unmöglich. Gerade bei diesen Modellen ist es interessant optimale oder gute Parameter zu finden. Hier können auf Grund der Komplexität durch Verbesserungen oft große Einsparungen bewirkt werden. Ohne eine systematische Auswertung, also durch reines

[1] Vgl. auch Kapitel 2.1

„Ausprobieren“, lassen sich diese aber nicht finden. Das Auswerten aller möglichen Kombinationen von Parameterausprägungen ist ab einer bestimmten Größe nicht mehr machbar, da die Summe aller Kombinationen gegen unendlich wächst.

Ein Lösungsansatz wäre eine große Anzahl an Parametersätzen zufällig auszuwählen und diese auszuwerten. Wenn genügend Einstellungen ausprobiert werden, findet sich mit an Sicherheit grenzender Wahrscheinlichkeit eine gute Lösung (Sicherheit besteht dann, wenn alle Parametersätze ausprobiert werden). Dieses Vorgehen beinhaltet im Grunde genommen die unsystematische Auswertung einer großen Menge an Parametersätzen. Hier bietet die Optimierung stochastischer Modelle mit ihren Optimierungsalgorithmen deutliche Vorteile. Mit Optimierungsalgorithmen lassen sich darüber hinaus auch große Parameterräume strukturiert durchsuchen.

Mit verschiedenen Algorithmen lassen sich diverse Problemstellungen unterschiedlich gut optimieren. Einige liefern sehr schnell gute Ergebnisse, andere konvergieren bewiesenermaßen gegen ein globales Optimum. Weitere beschränken sich auf spezielle Anwendungsgebiete und wieder andere sind praktisch universell einsetzbar. Mit diesen Algorithmen lassen sich im Vergleich zur kompletten Auswertung des Suchraums schnell und gezielt gute, beziehungsweise optimale Parametersätze finden. Diese sind mit zufälligem Ausprobieren in der Regel nicht oder nur durch überproportionalen Mehraufwand zu finden.

Ein Überblick über diese Algorithmen findet sich in Kapitel 3. Eine vollständige Auflistung ist auf Grund der Vielzahl an Algorithmen praktisch unmöglich und wird auch nicht angestrebt. Obwohl an dem Gebiet der Optimierung stochastischer Systeme noch nicht lange gearbeitet wird[2], gibt es im Internet bereits unzählige Informationen zu diesem Thema. Viele Algorithmen, besonders solche für kontinuierliche Parameter, stammen aus der Mathematik und wurden teilweise an die Optimierung mit stochastischen Modellen angepasst.

Durch Optimierungsalgorithmen und die sich rasant entwickelnde Computertechnik ergeben sich also Möglichkeiten, die vor einigen Jahren noch völlig utopisch waren. Da sich das Gebiet der Optimierung ständig weiterentwickelt, ist anzunehmen, dass in der Zukunft immer komplexere Modelle sich mit steigender Computerleistung und ausgereifteren Algorithmen relativ schnell und qualitativ hochwertig optimieren lassen.

1.2 Rahmenbedingungen

In der DaimlerChrysler Forschung wird ein Tool zum Modellieren, Parametrisieren und Analysieren von stochastischen Modellen entwickelt. Das Tool Expect[3] bietet zurzeit die Modellierung und Analyse von Petri-Netzen, Fehlerbäumen und Zustandsräumen an. Für diese stochastischen Modelle werden verschiedene Analysemöglichkeiten angeboten. Für die Optimierung besonders interessant ist, dass Modelle auch parallel, also auf mehreren PCs gleichzeitig berechnet werden können. Da die Optimierung besonders rechenintensiv ist, können so auch größere Modelle oder Modelle mit vielen Einstellungsmöglichkeiten in relativ kurzer

[2] Vgl. [Fu02] S. 1

[3] Siehe auch Kapitel 2.2 und 2.3

Zeit berechnet werden.

Zusätzlich gibt es eine Skriptingschnittstelle, mit der ein stochastisches Modell mit mehreren Parametersätzen analysiert werden kann. Damit kann abstrakt auf verschiedene Modellarten zugegriffen werden. Diese Schnittstelle bietet sich somit an, um zum Beispiel mehrere Parametersätze auszuwerten. Diese Analyseergebnisse werden für die Optimierung benötigt. Durch den Zugriff der Optimierung auf das Skripting lässt sich die Optimierung praktisch vollständig von dem zu optimierenden Modell abkoppeln. So lässt sich die einmal programmierte Optimierungsschnittstelle leicht auf neue Modell- und Analysearten anwenden.

Weil Expect komplett auf der Programmiersprache Java basiert, ist es plattformunabhängig und kann in verschiedenen Umgebungen eingesetzt werden. Dadurch, dass Java objektorientiert ist, lassen sich abstrakte Elemente nutzen und die Voraussetzung für die Erweiterbarkeit der Optimierungsschnittstelle ist gegeben.

Die Modelle, die bei DaimlerChrysler modelliert werden, beschränken sich nicht auf eine bestimmte Art der Modellierung. Es gibt Modelle mit diskreten, mit kontinuierlichen und mit gemischten Parametern. Durch Expect und die Skriptingschnittstelle sind hier kaum Grenzen bei der Modellierung gesetzt.

1.3 Ziele und Aufgabenstellung

Ziel der Arbeit ist es eine Optimierungsschnittstelle zu entwerfen, die ein Framework für Optimierungsalgorithmen, stochastische Modelle und andere Optimierungsbestandteile bildet. Dadurch sollen Erweiterungen und Verbesserungen auf allen Ebenen erleichtert werden. Dazu gehören neue Modellarten, Algorithmen oder auch eine neue Benutzerführung. Die Möglichkeit der Erweiterbarkeit ist insofern essentiell, als dass das Feld der Optimierung stochastischer Modelle noch jung ist und sich ständig weiterentwickelt. Zudem werden sich während einer Diplomarbeit nicht alle Möglichkeiten, die eine Optimierungsschnittstelle bieten kann, umsetzen lassen.

Zu diesem Zweck soll eine Optimierungsschnittstelle in Expect geschaffen werden, die erweiterbar ist und gute Optimierungsergebnisse liefert.

Die Optimierungsergebnisse sollen folgende Eigenschaften bestmöglichst erfüllen:

- **Optimale Ergebnisse**

  Das Ergebnis der Optimierung soll das (globale) Optimum sein.

- **Mit allen Modellen**

  Die Optimierung soll, soweit möglich, auf alle Modelle im Skripting anwendbar sein. Die Modelle sollen ohne weitere Anpassung optimierbar sein.

- **Möglichst schnell**

  Die Optimierung soll die Ergebnisse so schnell wie möglich liefern.

- **Einfach**

  Die Nutzung der Optimierung soll für den Anwender so einfach wie möglich sein. Je weniger Parameter/Einstellungen ein Anwender für die Optimierung anpassen muss, um ein gutes Ergebnis zu erreichen, desto einfacher ist sie.

Dieses sind idealisierte Ziele. Soll ein Ziel vollständig erfüllt sein, muss mindestens eines der anderen vernachlässigt werden. Beispielsweise ließe sich das globale Optimum finden, indem man den Suchraum vollständig durchläuft. Dies würde mit allen Modellen funktionieren und einfach realisierbar sein. Aber wie schon vorher erwähnt, explodiert damit die Zeitdauer und es wird praktisch kein Ergebnis zurückgegeben.

Eine gute Kombination der Ziele wäre zum Beispiel eine Optimierung, die eine gute Lösung (nahe am Optimum) in relativ kurzer Zeit (je nach Komplexität des Modells in einigen Minuten bis einigen Stunden/Tagen) mit einem Modell liefert, ohne dass der Anwender sich in die Optimierungs-Literatur vertiefen muss. Einem erfahrenen Anwender sollten andererseits umfangreiche Möglichkeiten der Parametrisierung zur Verfügung stehen, zum Beispiel um Optimierungsläufe zu beschleunigen.

1.4 Vorgehensweise

In Kapitel 2 werden die Grundlagen dieser Arbeit vorgestellt und das theoretische Fundament der Arbeit gelegt. In Abschnitt 2.1 werden die stochastischen Modelle, die optimiert werden sollen, am Beispiel von Petri-Netzen vorgestellt. Anschließend wird das Programm Expect und insbesondere dessen Skriptingschnittstelle erläutert. Mit der Skriptingschnittstelle können einzelne Parametersätze eines Modells für die Optimierung ausgewertet werden. Abschließend werden Ablauf und Elemente der Optimierung analysiert und identifiziert.

In Kapitel 3 wird ein Überblick über die verfügbaren Optimierungsalgorithmen gegeben. Die zu implementierenden Algorithmen werden ausgewählt. Zuerst werden die Auswahlkriterien, anhand derer die Auswahl der Optimierungsalgorithmen getroffen wird, in Anlehnung an die Ziele aus Abschnitt 1.3 definiert. Darauf folgt ein Überblick über Algorithmen der lokalen und globalen Optimierung sowie von Metaheuristiken, die zum Beispiel globale und lokale Optimierungsalgorithmen kombinieren. Abschließend werden die zu implementierenden Algorithmen anhand der Auswahlkriterien ausgewählt.

In Kapitel 4 wird die Implementation der identifizierten Optimierungsbestandteile in Expect beschrieben. Zuerst wird die Umsetzung beziehungsweise Einbindung der einzelnen Bestandteile in einem Optimierungsskript beschrieben, das alle zur Optimierung benötigten Bestandteile zusammenfasst. Anschließend wird der eigentliche Optimierungsablauf beschrieben und abschließend wird die Benutzerschnittstelle und Ergebnisausgabe vorgestellt.

In Kapitel 5 werden die implementierten Algorithmen anhand von drei Petri-Netzen mit ansteigender Komplexität verglichen. Das erste Netz ist ein frei erfundenes, sehr einfaches und übersichtliches, also einführendes Modell. Das zweite Modell ist ein Life-Cycle-Modell, das immer noch relativ einfach ist, aber schon eine praktische Anwendung darstellt. Das dritte

ist eine vereinfachte Version des sehr umfangreichen und komplexen Modells eines Logistik-Centers von DaimlerChrysler. Die Vorgehensweise verläuft bei allen drei Experimenten nach dem gleichen Schema. Erst werden Modell und Stellschrauben für die Optimierung (änderbare Parameter) sowie Algorithmusparameter vorgestellt. Danach werden die Ergebnisse und Laufeigenschaften der verschiedenen Algorithmen verglichen. Abschließend werden die Ergebnisse bewertet.

Das 6. Kapitel beginnt mit einer Zusammenfassung. Darauf folgt eine Bewertung der gesamten Arbeit. Abschließend wird ein Ausblick auf zukünftige Weiterentwicklungsmöglichkeiten gegeben.

Kapitel 2

Grundlagen

In diesem Kapitel werden die Grundlagen zum Verständnis dieser Arbeit erläutert. Die zu optimierenden stochastischen Modelle werden kurz am Beispiel von Petri-Netzen vorgestellt. Zu den Modellen und verschiedenen Analysearten gibt es umfangreiche Fachliteratur und auch im Internet finden sich zahllose Arbeiten beziehungsweise Einführungen in die Materie. Da Expect kein öffentliches Werkzeug ist, wird in Abschnitt 2.2 ein Überblick über die Grundidee, die derzeitige und die geplante Funktionalität von Expect gegeben. In Abschnitt 2.3 wird die Skripting-Schnittstelle von Expect genauer vorgestellt, da sie die Grundlage der Optimierung bildet. Abschließend werden die Bestandteile und Abläufe der Optimierung identifiziert und erläutert, bevor im nächsten Kapitel konkret auf Optimierungsalgorithmen eingegangen wird.

Grundkenntnisse in Java (Elemente und Strukturen) und Statistik (insbesondere Wahrscheinlichkeitsverteilungen) werden vorausgesetzt, sind aber für das grundsätzliche Verständnis nicht unbedingt notwendig.

2.1 Stochastische Modelle am Beispiel Petri-Netze

Vor allem die Geschwindigkeit und die Häufigkeit, mit der sich stochastische Modelle auswerten lassen, eröffnen Möglichkeiten, die mit dem „Ausprobieren“ in der Wirklichkeit nie möglich wären. Zum Beispiel wären Versuche mit neuen Arbeitszeitmodellen in einem Logistik-Center[1] sehr kostspielig und würden Wochen dauern. Ist aber zum Beispiel ein Petri-Netz-Modell des Centers erstellt, lassen sich durch Variationen der Parameter des Netzes Daten sammeln, die auf die Realität übertragen werden können, wenn das Netz sorgsam erstellt wurde. Hier ist es dann zum Beispiel einem Fachmann möglich, neue Modelle zunächst am Computermodell zu testen. Mit den Ergebnissen kann er die erfolgversprechendsten Modelle in der Realität ausprobieren beziehungsweise umsetzen.

Zunächst werden die Begriffe System, Modell und Analyse beziehungsweise Simulation eines Systems definiert:

System (siehe [Bro01]):

[1] Vgl. Petri-Netz-Modell eines Logistik Centers Kapitel 5.3

> „... fundamentaler Begriff, der die Zusammenfassung mehrerer, im Allgemeinen untereinander in Wechselwirkung stehender Komponenten zu einer als Ganzes aufzufassenden Einheit bezeichnet...“

Modell (siehe [Bro93] S. 301):

> „... ein Abbild der Natur unter Hervorhebung für wesentlich erachteter Eigenschaften und Außerachtlassen als nebensächlich angesehener Aspekte...“

Analyse dynamischer Systeme oder auch **Simulation** (siehe [Wik05]):

> „Simulation ist eine Vorgehensweise überwiegend zur Analyse dynamischer Systeme. Bei der Simulation werden Experimente an einem Modell durchgeführt, um Erkenntnisse über das reale System zu gewinnen.“

Ein Modell ist also ein vereinfachtes auf das Wesentliche reduzierte Abbild eines Systems oder einer Menge in Wechselwirkung stehender Komponenten. Durch das Weglassen nicht als wichtig angesehener Teile lassen sich mit Modellen komplexe Zusammenhänge vereinfacht darstellen. Diese Zusammenhänge lassen sich dann, unter Umständen ohne große Abstriche in der Genauigkeit, analysieren und auf das ursprüngliche System zurückführen. Dabei ist zu beachten, dass das System abstrahiert und vereinfacht worden ist. Wenn wichtige Komponenten nicht modelliert werden, ist dies für die Ergebnisse fatal. Ein Modell muss daher verifiziert und validiert werden.

Das Besondere an einem stochastischen Modell ist nun, dass bestimmte Vorgänge stochastischer Natur abgebildet sind. Dies kann entweder dadurch bedingt sein, dass die Vorgänge auch im Modell zufällig sind, oder die Abläufe in der Natur zu kompliziert oder zu umfangreich sind, um sie deterministisch nachzubilden. Dadurch entstehen neue Fehlerquellen, die beachtet werden müssen. Der große Vorteil eines deterministischen Modells ist, dass es nur einmal ausgewertet werden muss.

Es ist nicht immer sinnvoll, ein stochastisches Modell eines Systems zu verwenden. Vor- und Nachteile, die es gilt abzuwägen, sind in Tabelle 2.1 abgebildet. Der Entwickler muss die Entscheidung treffen, ob ein stochastisches Modell sinnvoller ist als ein deterministisches Modell oder ein Prototyp.

Beispiele für stochastische Modelle sind Petri-Netze, Zustandsräume und Fehlerbäume. Nachfolgend sollen als Beispiel für ein stochastisches Modell Petri-Netze vorgestellt werden, da in dieser Arbeit vor allem Petri-Netze als Beispiel für stochastische Modelle genutzt werden. Petri-Netze sind in der Wirtschaft und in der Informatik weit verbreitet.

Vorteile	Nachteile
• Kostengünstig – Stochastische Modelle lassen sich erstellen ohne aufwendig Prototypen zu bauen. Ist ein Modell gebaut, lassen sich meist in vergleichsweise kurzer Zeit viele Versuche durchführen. • Auch wirtschaftlich, rechtlich und physikalisch „Unmögliches“ ist simulierbar. Es besteht kein Risiko für Menschen. Es können Gegebenheiten wie zum Beispiel Unfälle oder Gefahrensituationen millionenfach wiederholt werden. • Die erstellten Modelle ermöglichen ein leichteres Verstehen des zugrunde liegenden Systems.	• Kostenintensiv – Die Modellierung ist aufwendig und die Modelle müssen sorgsam verifiziert und validiert werden, um gute Ergebnisse zu gewährleisten. • Die Ergebnisse sind sehr stark abhängig von der Qualität der Modellierung. Ist das Modell nicht gut oder ist ein falscher Sachverhalt nachgebildet, sind die Ergebnisse wertlos und eventuell irreführend. • Da die Eingangswerte stochastisch sind, sind auch die Ergebnisse zufallsbehaftet. Es gibt keine deterministischen Ergebnisse.

Tabelle 2.1: Vor- und Nachteile stochastischer Modelle

Petri-Netze werden im Wesentlichen aus vier Grundkomponenten aufgebaut[2]:

1. **Marken/Token**

   Mit Marken wird der derzeitige Zustand des Petri-Netz-Modells gekennzeichnet. Die Marken existieren in Stellen und symbolisieren Objekte oder einen Zustand. Im Fall von Objekten wird die Anzahl der existierenden Objekte durch die Anzahl der Marken dargestellt. Soll ein Zustand dargestellt werden, bedeutet die Anwesenheit eines Tokens, dass der Zustand aktiv ist.

   Marken können mit Transitionen erzeugt, von einer Stelle in eine andere transportiert und wieder zerstört werden. Die Marken sind nicht unterscheidbar.

2. **Stellen**

   Mit einer Stelle kann ein Zustand oder ein Ort dargestellt werden. Im Zusammenspiel mit Marken kann einem Ort (z. B. einer Lagerstätte) eine Menge von Objekten zugewiesen werden oder es kann ein Zustand als aktiviert oder nicht aktiviert markiert werden.

3. **Transitionen**
   Es gibt

   zeitbehaftete und

   zeitlose Transitionen.

[2] Vgl. http://wwwisg.cs.uni-magdeburg.de/sim/its/ws0405/Lectures/10-PetriNets.pdf

Transitionen verändern den Zustand des Netzes. Sie transportieren Marken von einer Stelle zu einer anderen, erzeugen Marken und vernichten sie wieder.

Ohne Transitionen wäre ein Petri-Netz immer nur der Zustand eines Systems zu einem bestimmten Zeitpunkt. Durch Transitionen lassen sich stochastische Zusammenhänge modellieren. Zeitlose Transitionen transportieren Marken (feuern) weiter, sobald sie aktiviert sind (Aktive Transition, siehe unten). Zeitbehaftete Transitionen transportieren Marken nach einer gewissen Zeit weiter. Diese kann über eine Wahrscheinlichkeitsfunktion zufällig bestimmt werden oder diskret sein.

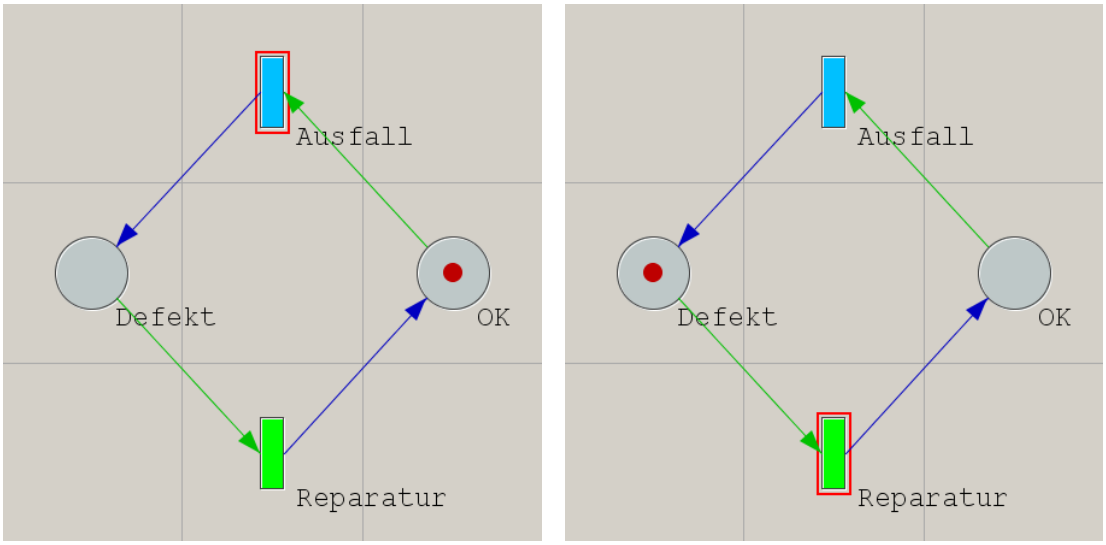
4. **Kanten**
   Es gibt

   → Eingangskanten: Eine Verbindung von einer Stelle zu einer Transition.

   ← Ausgangskanten: Eine Verbindung von einer Transition zu einer Stelle.

   Mit Kanten werden die Stellen und Transitionen eines Petri-Netzes verbunden. Durch einen Pfeil wird die Richtung angezeigt. Eingangskanten werden die Kanten genannt, die eine Stelle mit einer Transition verbinden. Ausgangskanten sind die, die aus einer Transition auf eine Stelle zeigen.

   Wenn in allen Stellen der Eingangskanten einer Transition eine Marke liegt, dann ist die Transition aktiviert. Feuert eine Transition, dann werden die aktivierenden Marken entfernt und an allen Stellen der Ausgangskanten werden Marken erzeugt. Die Anzahl der Marken muss nicht übereinstimmen.



(a) Transition Ausfall aktiv (b) Transition Reparatur aktiv

Abbildung 2.1: Zwei Zustände eines Systems modelliert mit einem Petri-Netz

In Abbildung 2.1(a) und 2.1(b) ist das Zusammenspiel von Tokens, Stellen, Kanten und Transitionen abgebildet. Das abgebildete Netz kann in zwei verschiedenen Zuständen sein.

Entweder es ist im Zustand *OK* (Abb. 2.1(a)) oder im Zustand *Defekt* (Abb. 2.1(b)). Der Zustand des Modells wird durch einen Token markiert. Ist das Modell im Zustand *OK*, ist die Transition Ausfall aktiv, erkennbar an der roten Umrandung der Transition in Abb. 2.1(a). Die Transition Reparatur ist in Abb. 2.1(b) aktiv, wenn das modellierte System im Zustand *Defekt* ist.

Zusätzlich gibt es noch zahlreiche Details beziehungsweise Erweiterungen, wie Alterungsstrategien, Guard-Funktionen, Kostenfunktionen, eindeutig identifizierbare Marken usw.. Diese sind aber für das Verständnis der Arbeit nicht erforderlich. Ein gründlicherer Einstieg in das Thema Petri-Netze (beziehungsweise stochastische Modelle) kann in der einschlägigen Fachliteratur und auch im Internet (z. B. unter `http://www.tfh-berlin.de/~grude/Petrinetze.pdf`) gefunden werden.

2.2 Expect – Ein integriertes Analyse- und Modellierungsframework

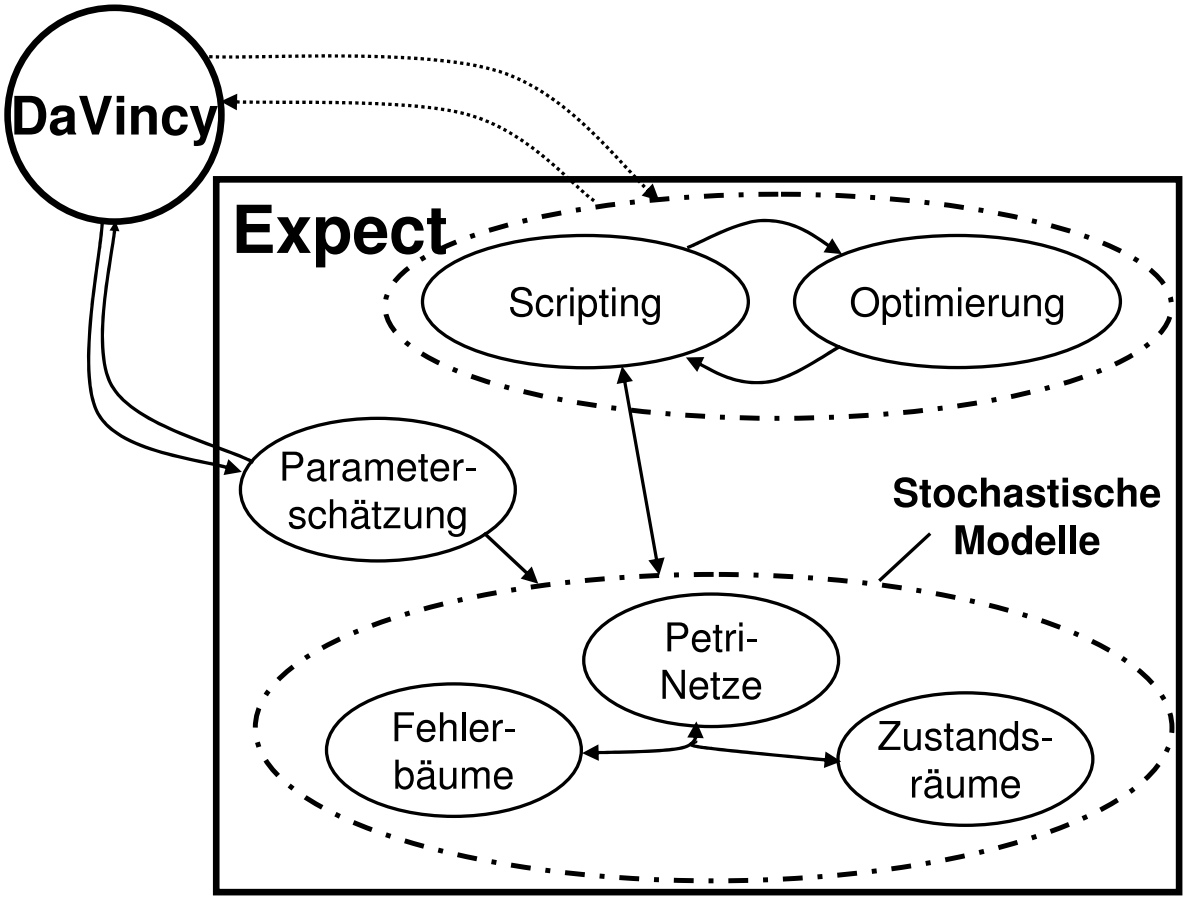


Abbildung 2.2: Überblick Expect-Funktionalität und DaVincy-Plattform

Der Kerngedanke hinter Expect ist, dass alle Daten, die bei der Modellierung eines Systems anfallen und verwendet werden, mit **einem** Werkzeug bearbeitet und erstellt werden. Dadurch sind die Daten und Modelle, die im Laufe der Modellierung erstellt wurden, eindeutig identifizierbar und lassen sich nach Monaten noch genau zuordnen. Darüber hinaus soll die Arbeit mit Expect möglichst vielen Nutzern ermöglicht werden, darunter auch Anwendern, die keine Experten in der Erstellung und Analyse von stochastischen Modellen sind. Durch die vollständige Implementation in Java ist eine weitgehende Plattformunabhängigkeit gegeben. Eine schematische Darstellung der Expect-Funktionalität ist in Abbildung 2.2 zu sehen.

Zu diesem Zweck und um eine Datenbasis für Expect zur Verfügung zu stellen, wurde in Expect eine Schnittstelle zur Datenbasis und Netzwerk-Plattform DaVincy geschaffen. Zum einen kann so auf die großen Datenmengen (zum Beispiel Schadfalldaten von Fahrzeugen) zugegriffen werden, die im DaVincy vorhanden sind. Aus diesen können zum Beispiel Verteilungsfunktionen für die stochastischen Modelle errechnet werden (siehe Parameterschätzung). Zum anderen kann über die Skriptingschnittstelle (siehe Abschnitt 2.3) aus dem DaVincy auf die Expect-Funktionalität zugegriffen werden. Dadurch, dass im Skripting genau definiert werden kann, welche Parameter eines Netzes verändert werden dürfen, besteht im DaVincy die Möglichkeit, über diese Parameter Modelle zu parametrisieren und zu analysieren. Sind die änderbaren Parameter durch einen Modellexperten definiert und beschrieben, dann ist eine Kenntnis des Modells für die Parametrisierung nicht unbedingt erforderlich und somit über das DaVincy auch Nicht-Experten möglich.

Zurzeit bietet Expect drei verschiedene stochastische Modellarten an. Dieses sind Petri-Netze, Zustandsräume und Fehlerbäume. Für jede Modellart ist ein graphischer Editor und eine oder mehrere Analysearten vorhanden. Davon sind die Möglichkeiten zur Modellierung und Analyse von Petri-Netzen zurzeit am weitesten fortgeschritten. Im Rahmen der Integration und Datendurchgängigkeit sollen die einzelnen Modellarten unter- und ineinander eingesetzt werden können. Zum Beispiel ließe sich ein Petri-Netz als Basis-Event in einen Fehlerbaum einsetzen oder ein Fehlerbaum in einem Petri-Netz nutzen.

Modellübergreifend lässt sich die Parameterschätzung einsetzen, um aus Daten (zum Beispiel aus dem DaVincy) Verteilungsfunktionen zu schätzen.

Auch eine Verteilung der Berechnung stochastischer Modelle auf mehrere Rechner ist möglich. Dies hat besonders bei großen Modellen deutliche Vorteile/Zeitersparnisse gegenüber der Berechnung auf einem Rechner.

Die Grundlage der Optimierung bildet das bereits erwähnte Skripting, das im nächsten Abschnitt ausführlich vorgestellt wird.

2.3 Struktur und Funktionalität der Expect-Skriptingschnittstelle

Ist ein Modell erstellt, verifiziert und validiert, sollen aus dem Modell neue Erkenntnisse gewonnen werden. Dazu ist es notwendig, das Modell mit verschiedenen Parametrisierungen zu analysieren. Die Skriptingschnittstelle ermöglicht genau dies. Sie bietet die Möglichkeit, Para-

metern Wertebereiche oder -listen zuzuordnen und automatisiert Kombinationen dieser Werte zu bilden. Diese Kombinationen können dann mit verschiedenen Analysemethoden ausgewertet werden.

Die gesamte Skriptingschnittstelle ist abstrakt implementiert. Das hat den Vorteil, dass sie für jede Modellart benutzt werden kann. Dadurch stehen dann auch die Dialoge, die automatische Analysefunktion und die Ergebnisdarstellung des Skriptings zur Verfügung. Die Modelle sind aber meist grundverschieden oder zumindest im Detail anders aufgebaut. Deshalb muss für jede Modellart eine entsprechende Schnittstelle implementiert werden. Für jedes Teil des Skriptings ist ein entsprechendes Interface vorgegeben, das die benötigten Methoden vorgibt. Dazu gehören zum Beispiel das Setzen der neuen Variablenwerte oder das Starten der Analyse.

Ist die Modellseite der Skriptingschnittstelle umgesetzt, dann steht der volle Umfang des Skriptings zur Verfügung. Dadurch kann jede Art von Modell in das Skripting eingebaut werden, ohne in die bereits implementierten Modelle einzugreifen. Auch Erweiterungen des Skriptings (die keine neuen Schnittstellen definieren) sind sofort für alle implementierten Modellarten anwendbar, ohne dass weitere Anpassungen nötig sind.

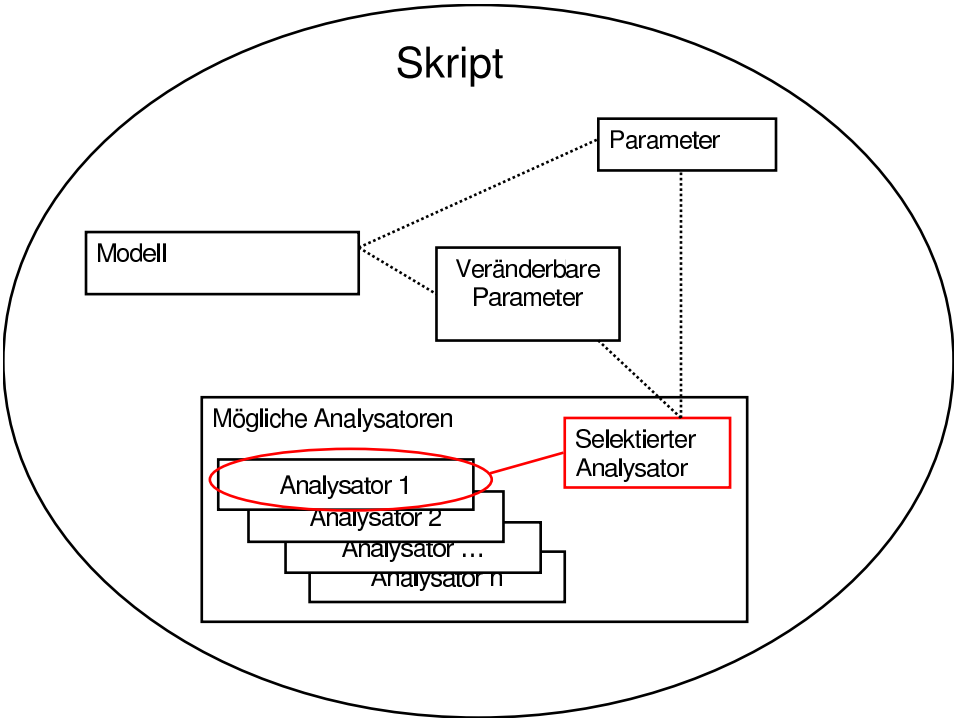


Abbildung 2.3: Bestandteile des Expect-Skriptings

Alle für das Skripting benötigten Daten werden in einem Skript zusammengefasst. Abbildung 2.3 zeigt die wichtigsten Teile des Skriptings. Um eine größere Menge verschiedener Parameter auszuwerten, werden folgende Objekte benötigt:

- **Das Modell**
  Das eigentliche (stochastische) Modell, das analysiert werden soll. Das Modell hat be-

reits Voreinstellungen für alle Parameter (Urzustand). Dadurch müssen beim Skripting nicht alle Parameter neu eingestellt werden, sondern nur die veränderten.

- **Die möglichen Analysatoren und der ausgewählte Analysator**
  Eine Modellart kann mehrere verschiedene Möglichkeiten der Analyse anbieten. Eine Möglichkeit der Analyse muss auf jeden Fall gegeben sein. Bei der Auswertung des Modells wird dann mit dem ausgewählten Analysator analysiert.

- **Parameter**
  Sowohl für das Modell als auch für den Analysator werden die veränderten Parameter gespeichert. Es werden nur die Einstellungen gespeichert, die verändert wurden. Die neuen Werte der Parameter können einzelne Werte, Wertelisten oder Wertebereiche sein.

- **Veränderbare Parameter**
  Die veränderbaren Parameter sind eine Besonderheit. Auch sie existieren für Analysator und Modell und stellen eine Schnittstelle zu anderen Programmteilen oder Programmen her. Wird nun zum Beispiel ein Modell im DaVincy bearbeitet[3], können hier nur die veränderbaren Parameter modifiziert werden. Dadurch kann zwischen den Parametern unterschieden werden, die nur der Modell-Experte modifizieren darf, und denen, die auch der Endanwender ändern kann.

Hat der Anwender alle Eingaben getätigt und die Analyse gestartet, beginnt der eigentliche Ablauf des Skriptings (eine schematische Übersicht ist in Abbildung 2.4 zu sehen). Aus allen geänderten Parametern werden jetzt die Parametersätze gebildet, die analysiert werden sollen. Die Menge der Parametersätze wird mit dem Kreuzprodukt aller angegebenen Parametermengen beziehungsweise -wertebereiche gebildet.

Als Beispiel sei ein Modell mit drei geänderten Parametern angenommen. Der erste Parameter wird nur in seinem Wert geändert, dem zweiten wird eine Menge von Werten zugewiesen und dem dritten ein Wertebereich.

- Parameter 1 : Einzelner Wert 5
- Parameter 2 : Wertemenge: 1,2,3
- Parameter 3 : Wertebereich Von 2,5 bis 3,5 Schrittweite 0,5

Daraus werden die 9 Parametersätze (Permutationen) in Tabelle2.2 gebildet:

Der Ablauf des Skriptings wird von der „Skripting-Session“ gesteuert. In der Skripting-Session werden die Analyseläufe erstellt und initialisiert. Nach der Analyse werden hier in einem Result-Report die gesammelten Daten gespeichert. In Abbildung 2.4 ist der Zusammenhang zwischen Skript, Skripting-Session und Result-Report dargestellt. Die Session fragt die verschiedenen Permutationen eine nach der anderen vom Skript ab. Für jede Permutation wird ein Analyselauf (AnalysisRun) erstellt und initialisiert. Die Initialisierung beinhaltet das

[3] Siehe Abschnitt 2.2

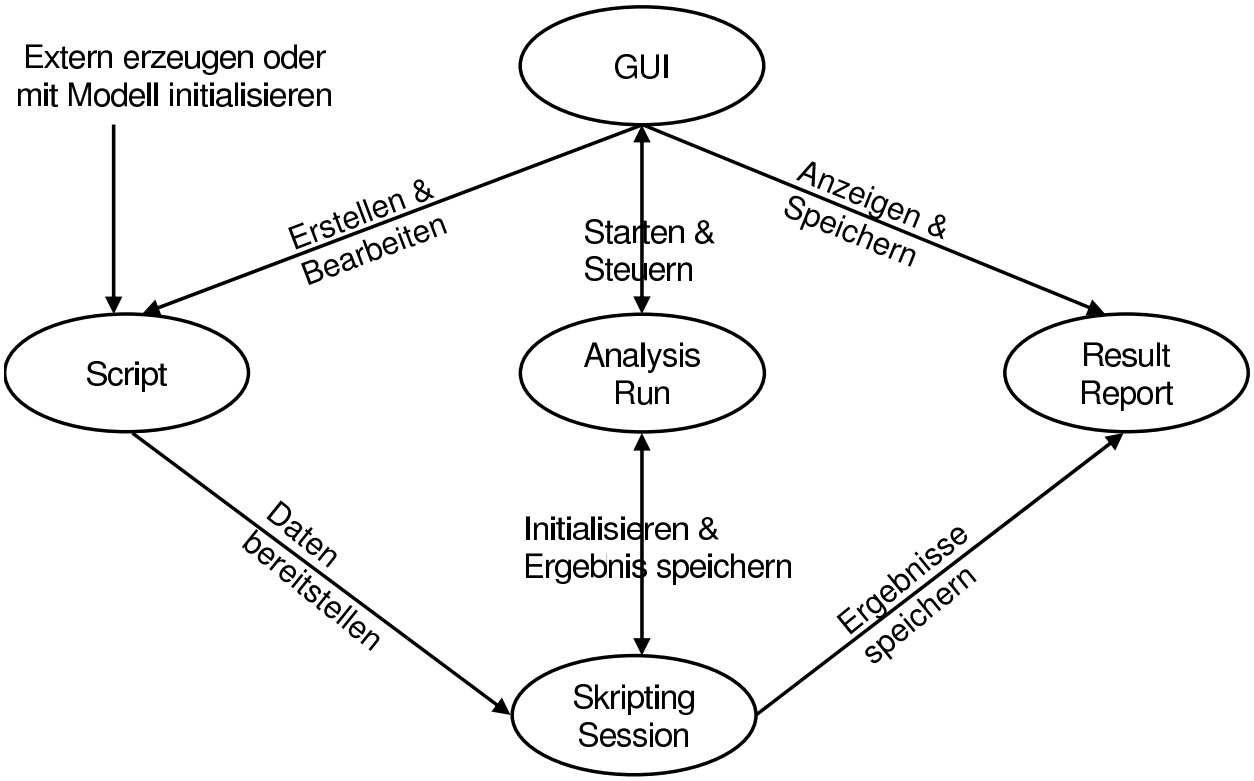


Abbildung 2.4: Ablauf des Skriptings

Erzeugen eines Modells und eines Analysators mit den veränderten Werten. Der Lauf wird nun von der GUI gestartet und nach abgeschlossener Analyse werden die Ergebnisse an die Skripting-Session zurückgegeben. Diese Werte werden dann in einem Result-Report gespeichert. Auf Basis des Result-Reports werden die Ergebnisse dann in der GUI dargestellt.

Zurzeit können die Ergebnisse nur im Programm verarbeitet und als Diagramm gespeichert werden. Eine Speicherung im XML-Format, in dem auch die Skripte gespeichert werden, ist aber geplant. Für die Optimierung können die internen Daten verwendet werden, die in einer XML-Baumstruktur vorliegen.

Durch das XML-Format ist es auch möglich, Skripte in anderen Programmen zu erzeugen beziehungsweise die Ergebnisse zu verarbeiten. Dies soll insbesondere an der Schnittstelle zwischen DaVincy und Expect ausgenutzt werden. Im DaVincy sollen Parameter für ein Modell gesetzt werden, die dann als XML-Datei an einen Expect-Server geschickt werden. Dort werden die Daten mit dem entsprechenden Modell ausgewertet. Die Ergebnisse werden dann wieder an das DaVincy zurückgeschickt und dort dargestellt.

Permutation	Par. 1	Par. 2	Par. 3	Permutation	Par. 1	Par. 2	Par. 3
1	5	1	2,5	**6**	5	2	3,5
2	5	1	3,0	**7**	5	3	2,5
3	5	1	3,5	**8**	5	3	3,0
4	5	2	2,5	**9**	5	3	3,5
5	5	2	3,0				

Tabelle 2.2: Permutationen

2.4 Optimierung stochastischer Modelle – Bestandteile und Ablauf

Optimierung allgemein ist, (siehe [Bro01]):

> „... insbesondere die Suche des kleinsten (Minimierung) oder größten (Maximierung) Wertes einer mathematischen Funktion mehrerer Veränderlicher (Ziel-, Objektfunktion) in einem bestimmten, durch Nebenbedingungen festgelegten und in Form von Gleichungen oder Ungleichungen beschriebenen (zulässigen) Bereich."

Die Optimierung stochastischer Modelle ist eng angelehnt an die allgemeine Optimierung. Der Unterschied ist, dass anstelle einer mathematisch auswertbaren Funktion hier ein stochastisches Modell[4] analysiert wird und aus den Ergebnissen dieser Auswertungen der Zielfunktion ein Wert zugewiesen wird. Die Zielfunktion ist also immer noch eine Funktion mehrerer Veränderlicher. Die veränderlichen Variablen sind in diesem Fall aber Eigenschaften des stochastischen Modells. Die Nebenbedingungen entsprechen hier genauso den Grenzen der Variablen. Auch Abhängigkeiten können definiert werden.

Für das Design und die Implementation einer Optimierungsschnittstelle ist es wichtig, die Bestandteile und den Ablauf der Optimierung genau zu definieren. Besonders wenn Bestandteile austauschbar sein sollen, und für die Erweiterbarkeit ist dies essentiell, müssen Schnittstellen zwischen den einzelnen Elementen gefunden werden.

Schon in der Definition der Optimierung können einige wichtige Bestandteile der Optimierung identifiziert werden: Die Zielfunktion, die Veränderlichen und der Bereich, in dem sich die Variablen bewegen. Zusätzlich wurde für die Optimierung stochastischer Modelle bereits das Modell identifiziert (Ablauf und Bestandteile der Optimierung sind in Abbildung 2.5 zu sehen). Sämtliche Bestandteile der Optimierung sind in der folgenden Liste aufgelistet:

- **Ein Modell**

  Um ein beliebiges System im Computer zu optimieren, muss es zuallererst in ein Modell übersetzt werden, welches ausgewertet werden kann.

[4] Auch eine mathematische Funktion ist ein Modell.

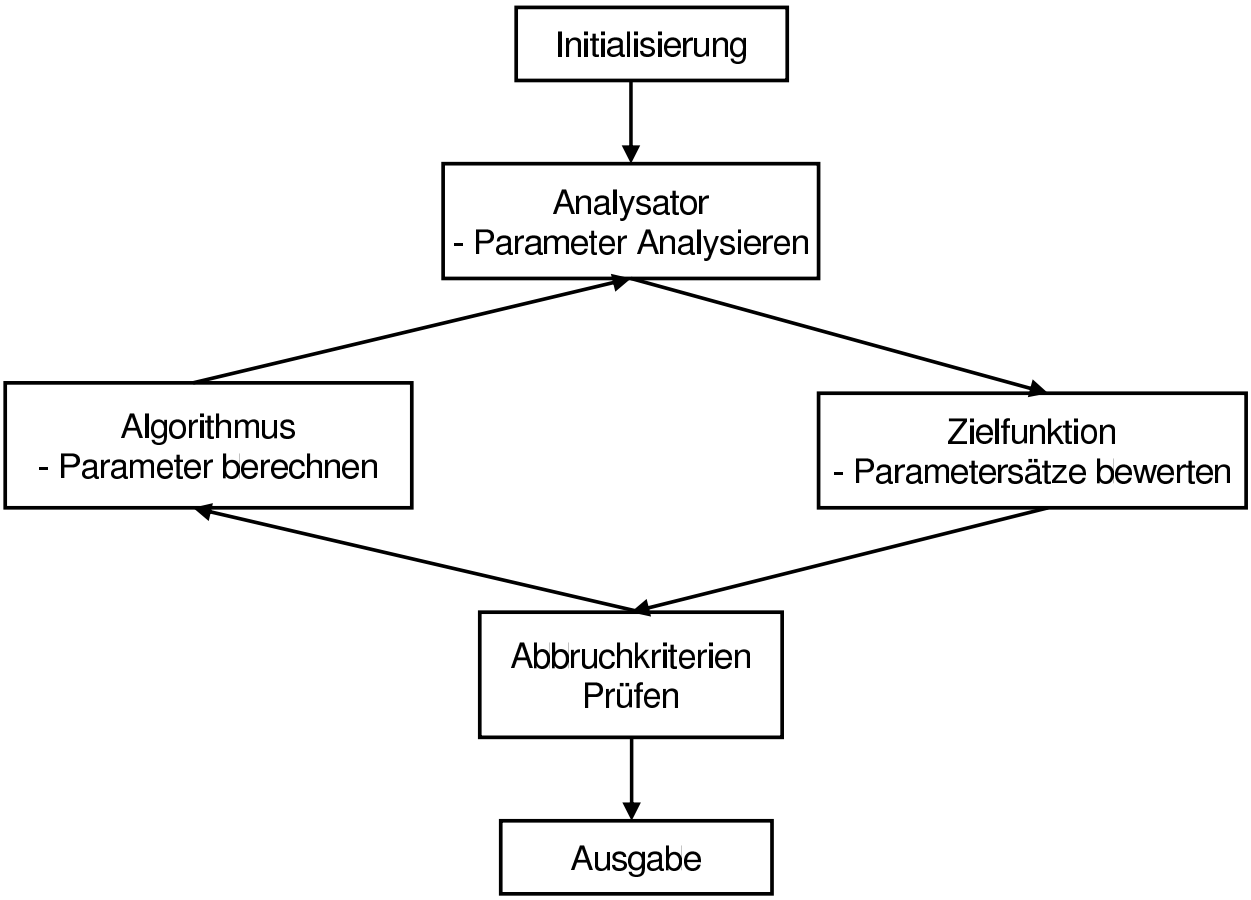


Abbildung 2.5: Schematischer Ablauf und Bestandteile der Optimierung

- **Ein Analysator für das Modell**

  Es muss eine Möglichkeit geben, das Modell auszuwerten. Um das Modell zu bewerten (ohne Bewertung keine Optimierung), muss es in irgendeiner Weise analysiert werden können. Eine Art ein Modell auszuwerten ist die Simulation eines Petri-Netzes.

- **Die veränderbaren Variablen des Modells**

  Damit die Optimierung eines Modells interessant ist, muss es Parameter an dem Modell geben, die veränderbar sind. Gibt es keine, dann gibt es nur eine mögliche Modellausprägung und die Optimierung ist überflüssig/trivial. Diese variablen Größen sind meist nur in Grenzen zu verändern. Hier spielen zum Beispiel Geld, physikalische Gesetze usw. eine Rolle. Beispielsweise würde eine KFZ-Wartung alle 100 km die Anzahl der Autopannen sicherlich verringern, wäre aber unökonomisch.

- **Eine Zielfunktion**

  Die Zielfunktion drückt den „Wert“ des Modells in einer Zahl aus. Sie weist dem Modell mit einer Parameterausprägung einen bestimmten Wert (die Fitness) zu, der aus den Kennzahlen des analysierten Modells berechnet wird. Das Ergebnis der Zielfunktion ist

die eigentliche Optimierungsgröße. Die Definition der Zielfunktion ist einer der wichtigsten Teile der Optimierung. Ist die Berechnung, die hier definiert wird, nicht korrekt beziehungsweise beschreibt sie nicht den gewünschten Tatbestand, werden auch die Ergebnisse unbrauchbar sein.

- **Ein Optimierungsalgorithmus**

  Die einfachste Art, eine optimale Parameterausprägung für ein Modell zu finden, wäre die komplette Enumeration/eine Brute-Force Iteration über alle Parameterausprägungen, die möglich sind. Da die Anzahl der Parameterausprägungen sehr schnell so groß wird, dass die komplette Enumeration nicht mehr möglich ist (Zum Beispiel ergeben sich bei 10 änderbaren Werten mit je 100 Ausprägungen $100^{10} = 10^{20}$ verschiedene Kombinationen möglicher Parametersätze.). Bei einem großen Modell wie dem Logistik-Modell aus Kapitel 5.3, dessen Simulation zwei Stunden pro Replikation dauert, ist dies keine sinnvolle Möglichkeit. Aber auch bei einem einfachen Modell, das in einer Sekunde simuliert ist, dauert die komplette Enumeration 3 170 979 198 376 Jahre.
  Mit geeigneten Verfahren lassen sich aber in relativ kurzer Zeit gute Parametersätze oder interessante Parameterregionen finden. Die verschiedenen Arten von Optimierungsalgorithmen werden im nächsten Kapitel vorgestellt.

- **Die Abbruchkriterien**

  Die Optimierungsalgorithmen haben meist kein natürliches Ende oder es dauert sehr lange, bis das Ende erreicht wird. Das heißt, dass die Optimierung ohne zusätzliche Kriterien, die den Abbruch eines Optimierungsalgorithmus bestimmen, nicht einsetzbar ist. Dazu kommt, dass die relative Verbesserung meist nachlässt, je länger ein Optimierungsalgorithmus läuft. Deswegen werden Abbruchkriterien definiert.

  Gebräuchliche Abbruchkriterien[5] sind die Zeit, die Anzahl der Wiederholungen oder Generationen des Algorithmus oder die Anzahl der Generationen, in denen nicht eine Mindestverbesserung erreicht wurde. Die maximale Zeit oder die maximale Anzahl an Generationen eines Algorithmus sind insofern gute Abbruchkriterien, als dass schon bei Beginn genau beziehungsweise annäherungsweise feststeht, wann die Optimierung terminiert.

  Zusätzlich kann ein Algorithmus terminieren, wenn er keine neuen Parameterwerte beziehungsweise Parameterregionen erforscht. Informationen dazu und weitere Abbruchkriterien sind in [Poh00] S.63ff zu finden.

Sind die obigen Teile der Optimierung gegeben, kann eine Optimierung durchgeführt werden. Bei der Optimierung kann ein bestimmter Ablauf identifiziert werden, in dem die bereits definierten Teile verwendet werden.

1. **Die Initialisierung**

[5] Vgl. S.63ff [Poh00]

Bevor die Optimierung beginnen kann, müssen Modell und Optimierungsbestandteile initialisiert werden. Sind bereits Lösungen für das Optimierungsproblem bekannt, können diese als Startposition oder Startpopulation der Optimierung übergeben werden. Andernfalls wird die Startposition zufällig generiert.

2. **Die Optimierung**

   Während der Optimierung werden die folgenden Schritte solange wiederholt, bis das Ergebnis die gewünschte Qualität hat oder ein anderes Abbruchkriterium (zum Beispiel die maximale Laufzeit) erreicht wurde[6].

   (a) Die ermittelten Parametersätze werden im Modell gesetzt und das Modell wird vom Analysator ausgewertet.

   (b) Die berechneten Werte werden analysiert und mit der Fitnessfunktion bewertet.

   (c) Die bewerteten Parametersätze und Lösungen werden an den Optimierungsalgorithmus übergeben.

   (d) Der Optimierungsalgorithmus sucht viel versprechende Lösungen und gibt sie an den Optimierer zur Analyse zurück.

3. **Die Ausgabe**

   Nach der Optimierung werden die Ergebnisse der Optimierung an den Nutzer zurückgegeben. Auf Grund der stochastischen Natur der Bewertung der Lösungen ist es sicherlich sinnvoll, die Gruppe der besten Lösungen noch einmal gründlich zu untersuchen.

[6] Vgl. [Fu02] S.199

Kapitel 3

Optimierungsalgorithmen

Im ersten Abschnitt dieses Kapitels werden die Ziele, die mit den Optimierungsalgorithmen erreicht werden sollen, von den Zielen aus Kapitel 1.3 abgeleitet. Im nächsten Abschnitt werden lokale Optimierungsalgorithmen vorgestellt. In Abschnitt 3.3 werden dann einige globale Algorithmen vorgestellt. Darauf folgt ein kurzer Einblick in verschiedene Metaheuristiken, bevor die Auswahl der zu implementierenden Algorithmen das Kapitel schließt.

Bei der Optimierung wird aus der Menge der möglichen Parametersätze für ein Optimierungsproblem der Parametersatz gesucht, der den besten Zielfunktionswert beschreibt. Die Menge der Parametersätze ist meist zu groß um jede mögliche Kombination zu testen. Ein Optimierungsalgorithmus stellt also eine Strategie dar, mit der der Raum der möglichen Parametersätze durchsucht wird.

3.1 Auswahlkriterien

Da es eine praktisch unüberschaubare Vielzahl von Algorithmen gibt und nicht alle (während dieser Diplomarbeit) implementiert werden können, sollen hier einige Auswahlkriterien definiert werden. Diese werden aus den Zielen aus Kapitel 1.3 abgeleitet. Verschiedene Algorithmen sollen dann in den anschließenden Abschnitten ohne Anspruch auf Vollständigkeit vorgestellt werden.

Die Optimierung soll einfach und schnell sein sowie für alle denkbaren Modelle ein optimales Ergebnis liefern. Deswegen werden folgende idealisierte Kriterien für die Auswahl der Algorithmen definiert:

1. **Gute Ergebnisse**

   Die Algorithmen sollen optimale Ergebnisse liefern.

2. **Robust**

   Die Algorithmen sollen möglichst robust auf stochastische Schwankungen bei der Auswertung von Modellen reagieren.

3. **Vielseitig**

   Die Algorithmen sollen mit allen möglichen Parameterausprägungen arbeiten können, also zum Beispiel mit realen und natürlichen Zahlen.

4. **Schnell**

   Die Algorithmen sollen schnell einen optimalen Wert finden.

5. **Einfache Parametrisierung**

   Die Algorithmen sollen möglichst ohne zusätzliche Parametereinstellungen gute Ergebnisse liefern, beziehungsweise es soll einfach sein, die Algorithmen zu parametrisieren. Ist der Algorithmus weit verbreitet oder gut dokumentiert, ist diese Forderung eher sekundär.

6. **Wiederverwendbarkeit**

   Algorithmen, die in anderen Meta- beziehungsweise einfachen Heuristiken Wiederverwendung finden können, werden positiv bewertet.

POSITIVE EIGENSCHAFTEN	NEGATIVE EIGENSCHAFTEN
• Globale Konvergenz bewiesen • Schnelle Ergebnisse, die zum Teil schon in einem Simulationslauf gefunden werden • Auf alle Parameterarten anwendbar, also auf kontinuierliche, auf diskrete und auf gemischte Parametersätze • Robust gegenüber stochastischen Schwankungen	• Optimierung ist lokal • Hoher Anpassungsaufwand an jedes Modell, zum Teil sind Eingriffe in den Analysator notwendig • Eingeschränkte Anwendbarkeit, beispielsweise nur auf kontinuierliche Parameter • Sehr empfindlich • Lange Laufzeit, im Extremfall sogar höher als bei der kompletten Enumeration

Tabelle 3.1: Eigenschaften von Algorithmen

Es gibt jedoch keinen Algorithmus, der alle diese Ziele vollständig erfüllen kann. Verschiedene Algorithmen haben unterschiedliche positive Eigenschaften, die meist mit negativen Eigenschaften erkauft werden. In Tabelle 3.1 sind verschiedene positive und negative Eigenschaften von Optimierungsalgorithmen aufgezählt.

Im Folgenden sollen einige lokale und globale Optimierungsalgorithmen sowie Metaheuristiken vorgestellt werden.

3.2 Lokale Optimierung – Hill Climbing

Eine grundsätzliche Eigenschaft von Optimierungsalgorithmen ist, dass sie zuverlässig entweder ein globales Optimum (oder einen Punkt in dessen Nähe) oder konstruktionsbedingt nur lokale Optima finden. Es existieren allerdings verschiedene Ansätze, einen lokalen Algorithmus als Grundlage für einen globalen zu benutzen[1].

Ein lokales Optimum zu finden ist im Prinzip relativ einfach. Der einfachste Algorithmus wertet alle Punkte in der Nachbarschaft der derzeitigen Position aus. Der beste Punkt wird als neue Position ausgewählt. Dieser Algorithmus terminiert, wenn kein besserer Punkt als die aktuelle Position gefunden wurde.

Ein Pseudo-Code für die Suche eines Maximums kann so aussehen (Abb. 3.1):

```
Initialisiere Algorithmus mit Parametern und Anfangslösung
Werte Anfangslösung aus
Während ( Bessere Lösung gefunden )
{
   Bestimme Nachbarschaft
   Wähle beste Lösung in der Nachbarschaft
}
Beste gefundene Lösung zurückgeben
```

Abbildung 3.1: Einfachster lokaler Suchalgorithmus

Für dieses Verfahren ist es aber notwendig, die komplette Nachbarschaft auszuwerten.

Ein anderes Verfahren, STOCHASTIC APPROXIMATION[2], das an die mathematische Optimierung angelehnt ist, versucht die Steigung (oder auch den Gradienten) am derzeitigen Punkt zu ermitteln. Um ein Optimum zu finden, bewegt sich der Algorithmus nun solange in Richtung der stärksten Steigung (Maximierung), bis ein (lokales) Maximum erreicht ist. Formel 3.1 zeigt die Berechnung einer neuen Position Θ_{k+1} mit Hilfe des alten Parametersatzes Θ_k, der Schrittweite a_k und dem approximierten Gradienten $\widehat{g}_k(\widehat{\Theta}_k)$.

$$\widehat{\Theta}_{k+1} = \widehat{\Theta}_{k+1} - a_k \widehat{g}_k(\widehat{\Theta}_k) \tag{3.1}$$

Da eine genaue Ableitung der Zielfunktion (und somit die mathematische Ermittlung des Gradienten) nicht möglich ist, kann der Gradient nur geschätzt werden. Dafür gibt es unter anderem die *Finite Difference Stochastic Approximation* (**FDSA**) und das *Simultaneous Perturbation Stochastic Approximation* (**SPSA**) Verfahren, die auf einer Schätzung des Gradienten basieren. Damit die lokale Optimierung ein gutes Verhältnis von Laufzeit und Ergebnis hat, werden so genannte „Gain"-Sequenzen oder Schrittweitensequenzen eingesetzt. Diese sind stetig fallend. Dadurch werden am Anfang größere Schritte gemacht als am Ende, wo mit kleineren Schritten eine höhere Genauigkeit erreicht wird. Für die beiden Verfahren (FDSA

[1] Siehe dazu die nächsten Abschnitte insbesondere Metaheuristiken

[2] Vgl. [Spa] Abschnitt 6.3.1ff und [Spa98] S. 818ff

und SPSA) werden zwei Gain-Sequenzen benötigt. Eine für die Ermittlung der Nachbarschaft (c_k) und eine weitere für die Berechnung der Schrittweite (a_k), dabei ist k der Zähler der derzeitigen Wiederholung. Für die Sequenzen gilt $a > 0, c > 0, a \to 0, c \to 0, \sum_{k=0}^{\infty} a_k = \infty$ und $\sum_{k=0}^{\infty} a_k^2/c_k^2 < \infty$[3].

Zwei in der Literatur beliebte Gain-Sequenzen sind für a Formel 3.2 und für c Formel 3.3. Die Wahl der Variablen ist dabei alles andere als einfach, Hilfestellung kann in [Spa98] S. 819 gefunden werden.

$$a(k) = \frac{a}{(k+1+A)^{\alpha}} \tag{3.2}$$

$$c(k) = \frac{c}{(k+1)^{\gamma}} \tag{3.3}$$

Bei dem **FDSA**-Verfahren wird der Gradient für jede Dimension des Parametersatzes einzeln bestimmt. Dazu werden für jeden Teil des Parametersatzes zwei neue Parametersätze[4] analysiert, in denen jeweils nur der zu analysierende Teil des Parametersatzes um $\pm$ eine Schrittweite c (Die durch die Schrittweitenfunktion c vorgegeben wird.) verändert wird. Mit diesen neuen Parametersätzen wird anhand der Fitnessfunktionswerte dann der Gradient für den aktuellen Parametersatz (die derzeitige Position) approximiert. Die zugehörige Funktion ist:

$$\widehat{g}_k(\widehat{\Theta}_k) = \frac{y(\widehat{\Theta}_k + c_k) - y(\widehat{\Theta}_k - c_k)}{2c_k} \tag{3.4}$$

Dabei ist y die Zielfunktion, θ der Parametersatz und $c > 0$ die derzeitige Schrittweite[5].

So müssen für jede Iteration des Algorithmus zweimal die Anzahl der Parameter berechnet werden. Die Bestimmung des Gradienten kann auch einseitig erfolgen, dann wird die derzeitige Position als zweiter Punkt der Gradientenbestimmung verwendet. Damit sinkt die Anzahl der nötigen Funktionsauswertungen um die Hälfte.

Bei dem **SPSA**-Verfahren werden die Gradienten für alle Dimensionen des Parametersatzes gleichzeitig ermittelt. Hier werden mit Hilfe eines „Perturbationsvektors“ Δk zwei Parametersätze[4] gebildet, in denen alle Teile des Parametersatzes verändert sind. Die Elemente des Perturbationsvektors werden mit einer symmetrischen Bernoulliverteilung gebildet. Dazu wird jedes Element mit gleicher Wahrscheinlichkeit auf $+1$ oder -1 gesetzt[6].

Die beiden auszuwertenden Parametersätze $\Theta_{\pm}$ werden nach folgender Funktion gebildet:

$$\Theta_{+} = \Theta_k + \Delta_k * c_k \tag{3.5}$$

$$\Theta_{-} = \Theta_k - \Delta_k * c_k \tag{3.6}$$

[3] Vgl. [Spa] Abschnitt 6.3.2
[4] Siehe auch Tabelle 3.2
[5] Vgl. [Spa] Kapitel 6.3.2
[6] Vgl. [Spa98] S. 820

Aus den ermittelten Zielfunktionswerten werden jetzt die Gradienten mit der folgenden Formel ermittelt[7].

$$\widehat{g}_k(\widehat{\Theta}_k) = \frac{y(\widehat{\Theta}_k + c_k * \Delta_k) - y(\widehat{\Theta}_k - c_k * \Delta_k)}{2c_k} * \Delta_k \tag{3.7}$$

ANZAHL PARAMETER	EINFACHE NACHBARSCHAFTSSUCHE	FDSA	SPSA
1	2	2	2
2	8	4	2
3	26	6	2
4	80	8	2
5	242	10	2
.	.	.	.
.	.	.	.
.	.	.	.
n	$n^3 - 1$	$2 * n$	2

Tabelle 3.2: Modellauswertungen lokale Suche (Einfache Nachbarschaftssuche, FDSA und SPSA)

Schon der relativ einfache FDSA Algorithmus spart gegenüber der kompletten Nachbarschaftsbildung bei höherdimensionalen Problemen eine erhebliche Anzahl an Modellauswertungen (siehe Tabelle 3.2). Diese Einsparung ist beim Vergleich vom FDSA zum SPSA Verfahren wiederum erheblich. Es hat sich gezeigt, dass die Qualität des durch den SPSA-Schätzer ermittelten Gradienten der des FDSA-Gradienten meist ebenbürtig ist, insbesondere bei hochdimensionalen Problemen[8].

Nachteile von SPSA und FDSA sind, dass sie für kontinuierliche Parameter gedacht beziehungsweise entworfen sind und dass die Parametrisierung recht kompliziert ist[9].

Vorteil der lokalen Suche und insbesondere des SPSA-Algorithmus ist, dass sie relativ schnell ist. Die Suche mit SPSA und FDSA kann unter bestimmten Umständen auch als globaler Optimierer eingesetzt werden[10]. Die Erweiterungsmöglichkeiten dieses Verfahrens sind zahlreich und die lokale Suche kann in einigen Metaheuristiken eingesetzt werden.

Nachteil insbesondere der gradientenbasierten Suche ist, dass sie nur für kontinuierliche Parameter gedacht ist. Es gibt aber auch Ansätze für die Optimierung mit diskreten Parametersätzen[11].

[7] Vgl. [Spa98] S. 819f
[8] Vgl. [Spa92] S. 340 und [Spa99] S.536
[9] Siehe hierzu [Spa98] S. 819
[10] Vgl. [DCC02]
[11] Siehe [GHV99] S. 466ff

3.3 Globale Optimierungsalgorithmen

Für die Optimierung besonders interessant sind Algorithmen, die gezielt das globale Optimum suchen. Davon sollen einige im folgenden Abschnitt vorgestellt werden.

Infinitesimal Perturbation Analysis (IPA)

Bei der IPA werden die Gradienten für den gesamten Suchraum mit einem „Simulationslauf“ ermittelt. Dies wird erreicht durch geringe (infinitesimale) Veränderungen der Modellparameter während des Laufes. Deren Auswirkungen im System werden verfolgt und durch diese Veränderungen wird der Gradient approximiert. Der erhebliche Geschwindigkeitsvorteil, der dadurch erreicht wird, dass das gesamte Modell nur einmal analysiert werden muss, wird aber durch einige starke Einschränkungen erkauft.

Zum Beispiel darf sich bei einem Simulationsmodell die Reihenfolge der Ereignisse während der Analyse nicht durch die minimale Änderung der Parameter verschieben. Zusätzlich müssen genaue Kenntnisse über das zu optimierende Modell vorliegen, um die Auswirkungen der Veränderungen verfolgen zu können[12]. Dafür wäre ein tiefer Eingriff in die Analyse-Methoden notwendig.

Genetischer Algorithmus (GA)

Der genetische Algorithmus ist, wie schon der Name verrät, aus der Idee entstanden die natürliche Evolution nachzubilden. Eine Anfangspopulation wird zufällig bestimmt oder definiert und entwickelt sich durch Rekombination und Mutation ihrer Chromosomen fort. Die Nomenklatur ist eng an die Genetik angelehnt. Die Menge der Parametersätze wird als Population (oder auch Generation) bezeichnet. Die einzelnen Parametersätze heißen Chromosomen und die kleinsten Teile der Parametersätze (einzelne Parameter oder auch Teile von Parametern) werden Allele genannt.

Ein Allel, also der kleinste Teil eines Chromosoms, kann unterschiedlich lang sein. Dies beruht darauf, dass die einzelnen Elemente auf Basis von realen Zahlen oder als Bitwerte gespeichert werden. In einem Fall ist ein Bit ein Allel, im anderen wäre eine Zahl, also ein Parameterwert, der kleinste Teil eines Chromosoms.

Die Optimierung der Population geschieht von Generation zu Generation. Die alte Population bildet hier die Grundlage (den Gen-Pool) für die neue Generation. Die Chromosomen werden zufällig aus der alten Population gezogen, hierbei werden solche mit einem hohen Fitnesswert bevorzugt.

Dafür gibt es unter anderem die folgenden Verfahren[13]:

- Fitnessproportionale Selektion

[12] Vgl. [Aza99] S. 95
[13] Vgl. [Luk00] S. 9

Die Wahrscheinlichkeit, dass ein Chromosom gezogen wird, ist proportional zu seinem Fitnesswert.

- Rangbasierte Selektion

  Die Wahrscheinlichkeit, dass ein Chromosom gezogen wird, ist proportional zu seinem Rang in einer nach Fitness sortierten Liste der Chromosomen.

- Turnierselektion

  Bei der Turnierselektion werden mit gleicher Wahrscheinlichkeit mehrere Chromosomen gezogen. Das selektierte Chromosom, das den besten Fitnesswert hat, wird gewählt.

- Trunkationsselektion

  Bei der Trunkationsselektion wird eine Schwelle definiert. Die Individuen oberhalb dieser Schwelle werden mit gleicher Wahrscheinlichkeit selektiert, die anderen nicht.

Sind zwei Chromosomen gezogen, so werden aus diesen neue Chromosomen gebildet, dies geschieht mit dem so genannten Crossover oder auch der Kreuzung. Auch für diese Kreuzung gibt es mehrere Verfahren[14].

- 1-Punkt Kreuzung

  Hier werden aus zwei Chromosomen ein oder zwei neue Chromosomen erzeugt. Dies geschieht, indem zufällig ein Punkt zwischen zwei Allelen ausgewählt wird. Wird ein neues Chromosom erzeugt, werden ihm bis zu dem gewählten Punkt alle Allele des ersten Elters, danach die Allele des zweiten Elters zugeordnet.

- Uniform oder Multi-Punkt Kreuzung

  Hier werden mehrere Punkte, an denen die Übergabe an die neuen Chromosomen jeweils wechselt, ausgewählt. Bei der uniformen Kreuzung wird für jedes Allel des neuen Chromosoms zufällig ausgewählt, welches Chromosom Elter ist.

- „Headless Chicken“ Kreuzung

  Dies ist dasselbe Verfahren wie die beiden vorherigen, nur wird hier einer der beiden Eltern zufällig erzeugt.

- Reproduktion

  Gezogene Eltern werden ohne Veränderung in die neue Population übernommen.

- Mutation

  Die Mutation ist mit jeder anderen Art der Erzeugung kombinierbar. Meist wird mit einer niedrigen Wahrscheinlichkeit jedes Allel der Chromosomen verändert. Im Falle der Bitkodierung zeigt sich dies in einem Bitflip.

[14] Vgl. [Luk00] S. 8

Durch die Bevorzugung besserer Individuen werden die Eigenschaften der fitteren Chromosomen eher in eine neue Generation übernommen. Dadurch, dass auch schlechtere Chromosomen eine Chance haben gezogen zu werden, sind keine Regionen des Suchraums von vornherein ausgeschlossen. Mutation wiederum hilft zusätzlich neue Regionen des Suchraums zu erforschen.

Zu den Vorteilen des genetischen Algorithmus gehört, dass er sehr robust und praktisch auf alle Problemarten anwendbar ist. Zusätzlich ist er gut dokumentiert und in zahlreichen Varianten verfügbar, so dass diverse Erweiterungen möglich sind. Der Nachteil ist, dass er im Vergleich zu anderen Algorithmen relativ langsam ist[15].

Population Based Incremental Learning (PBIL)

Der PBIL Algorithmus ist wie der genetische Algorithmus ein evolutionärer Algorithmus. Der PBIL-Algorithmus speichert seinen Fortschritt aber nicht in einer Population, sondern in einem Wahrscheinlichkeitsvektor. Die Entwicklung findet auch über eine Population statt. Die Population wird über den Wahrscheinlichkeitsvektor gebildet. Mit den besten oder schlechtesten Parametersätzen wird der Wahrscheinlichkeitsvektor dann aktualisiert. Die Aktualisierung kann über eine Lernrate (α) gesteuert werden[16].

Die Ergebnisse des PBIL-Algorithmus sind nicht so gut erforscht wie die des GA. Der PBIL-Algorithmus ist dennoch viel versprechend, hat unter Umständen eine schnellere Konvergenz als der genetische Algorithmus, erbringt dabei aber etwas schlechtere Ergebnisse[17].

Ant Colony Optimization (ANT)

Der ANT-Algorithmus oder auch Ameisenalgorithmus ist der Beobachtung entsprungen, dass Ameisen beim Transport von Nahrung intelligent gute beziehungsweise optimale Wege finden. Ein einfaches Beispiel ist in Abb. 3.2 zu sehen. Hier bewegen sich die Ameisen anfänglich (Abb. 3.2 Teil a) ohne Präferenz über die verschiedenen Wege von ihrem Nest zu dem Nahrungsgebiet. In Abb. 3.2 Teil (b) ist das Verhalten der Ameisen nach einiger Zeit zu beobachten. Die Präferenz des direkten Weges ist erkennbar. Diese Wegfindung ist durch eine Pheromonspur erklärbar, die jede Ameise legt. Ameisen wählen unter verschiedenen Wegen, wobei sie Wege mit einer stärkeren Pheromonspur bevorzugen. Die kürzeren Wege werden schneller und dadurch von einer höheren Anzahl von Ameisen durchquert. Dadurch ist hier die Pheromonspur stärker und mehr Ameisen folgen dieser Spur.

Dieses Verhalten wird bei dem ANT-Algorithmus nachgebildet. Die Pheromon-Spuren werden über einen „Pheromonvektor“, der nach jeder Iteration aktualisiert wird, nachgebildet. In jeder Iteration sucht eine künstliche Ameise oder eine Menge von Ameisen einen Weg. Für jede Abzweigung gibt es einen Pheromonwert. Aus diesem wird eine Wahrscheinlichkeit ermittelt, die bestimmt, mit welcher Wahrscheinlichkeit die Ameise welchen Weg wählt.

[15] Vgl. [CM97] S. 121 und [Ras05] S. 13f
[16] Vgl. [YG03]
[17] Vgl. [TJTG04] S. 7

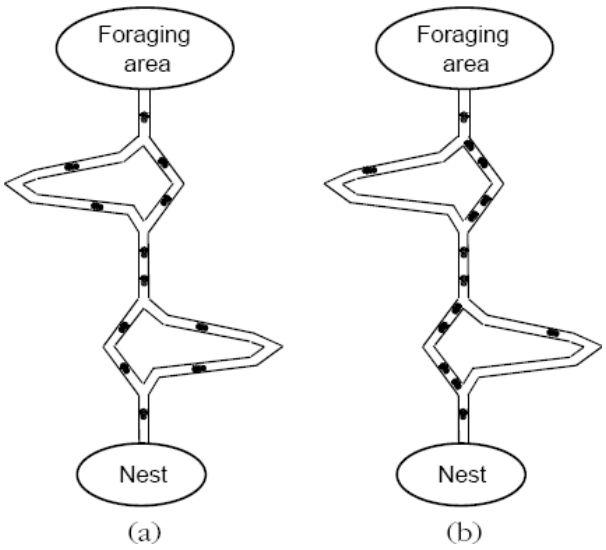


Abbildung 3.2: Ameisen-Algorithmus (siehe [DCG99] S. 3)

Je nach Länge des Weges beziehungsweise der Fitness der Lösung wird der Pheromonwert aktualisiert[18].

Der Ameisen-Algorithmus wird vor allem für Travelling-Salesman-Probleme und andere Probleme des Operation Research eingesetzt[19].

Partikel Schwarm Algorithmus (PSA)

Der Partikel Schwarm Algorithmus ist ein populationsbasierter Optimierungsalgorithmus, der dem Verhalten von Vogel- beziehungsweise Fischschwärmen bei der Nahrungssuche nachempfunden ist. Hat zum Beispiel ein Vogel Nahrung gefunden, bewegen sich die restlichen Mitglieder des Vogelschwarms in Richtung dieser Stelle.

Der Partikel Schwarm Algorithmus wird mit einer zufälligen oder gewählten Menge an Parametersätzen initialisiert. Anschließend wird die Fitness dieser Punkte berechnet. Die Position des besten Individuums (auch Partikel) ist der gesamten Population bekannt. Nach Berechnung der Fitness wird jedem Partikel eine Geschwindigkeit zugeordnet. Die Geschwindigkeit (v) wird berechnet aus der Geschwindigkeit des Partikels in der letzten Generation, der Distanz zum derzeitigen globalen Maximum (g_{best}) und der besten gefunden Position dieses Partikels (p_{best}). Mit dieser Geschwindigkeit bewegen sich die Partikel in jeder Generation erneut durch den Suchraum[20]. Zusätzlich wird die jeweilige Bewegung durch einen Zufallsfaktor ($rand()$ gleichverteilt zwischen 0 und 1) angepasst. Die Geschwindigkeit wird mit Formel 3.8

[18] Vgl. [DCG99] S. 5f
[19] Vgl. [MGL03] S.9ff
[20] Vgl. [Hu] und [KE95] S. 1942ff

berechnet.

$$v_{i+1} = v_i + g * rand() * (g_{best} - pos_i) + p * rand() * (p_{best} - pos_i). \tag{3.8}$$

Die Gewichtung der beiden Positionen wird durch einen Gruppen- (g) und einen persönlichen Faktor (p) gesteuert. Nach Berechnung der Geschwindigkeit wird die aktuelle Position (pos_i) mit Formel 3.9 neu berechnet. Ist die Geschwindigkeit größer als eine definierte Maximalgeschwindigkeit, dann wird sie auf die Maximalgeschwindigkeit begrenzt.

$$pos_{i+1} = pos_i + v_i \tag{3.9}$$

Vorteile des Partikel Schwarm Algorithmus sind, dass er leicht zu implementieren ist und wenige Parametereinstellungen benötigt um gute Ergebnisse zu liefern. Zusätzlich liefert er schon mit relativ kleinen Populationen (10) gute Werte.

Nachteile sind, dass Partikel weit über das Ziel hinausschießen können. Um neue Regionen des Lösungsraumes zu erkunden, ist dies zwar gewollt, führt aber teilweise dazu, dass die nähere Region des Optimums nur sehr grob durchsucht wird[21]. Hier schafft zum Beispiel eine adaptive Schrittweitenregelung Abhilfe.

3.4 Metaheuristiken

Aufbauend auf andere Algorithmen gibt es die Metaheuristiken. Hier werden entweder Algorithmen vereint, um deren Vorzüge zu kombinieren, oder Algorithmen werden um Teile erweitert, die die Leistung der Algorithmen verbessern.

Adaptive Memory Programming (AMP) - Framework

Die Grundidee des AMP-Frameworks ist, Vorzüge lokaler und globaler Algorithmen in einer strukturierten Umgebung zu aggregieren. Die Idee dieses Frameworks basiert auf der Tatsache, dass viele globale Optimierungsalgorithmen eine Art von Gedächtnis haben[22]. Oft stellt eine Population das Gedächtnis dar (zum Beispiel PSO, GA, ANT, ...). Der globale Algorithmus sucht hier global nach neuen interessanten Lösungsregionen, während der lokale Algorithmus die gefundenen Lösungen lokal verbessert.

Der Algorithmus hat folgenden Ablauf[23]:

- Initialisiere die Population.
- Wiederhole (bis Abbruchkriterium erreicht){
    1. Diversifikation: Erzeuge neue Population mit dem globalen Algorithmus

[21] Vgl. [Lø02] S. 15
[22] Vgl. [TGGP01] S. 2f
[23] Vgl. [MK05] S. 3

2. Intensifikation: Verbessere die einzelnen Individuen mit dem lokalen Algorithmus
3. Aktualisiere den globalen Algorithmus mit den verbesserten Lösungen

}

Die Diversifikationsphase nutzt die Vorteile der globalen Algorithmen, die schnell interessante Positionen im Suchraum finden. Die Intensifikationsphase bringt die Vorteile der lokalen Optimierung mit sich, die intensiv in der lokalen Nachbarschaft nach besseren Punkten sucht.

Dadurch, dass das Framework für Algorithmen mit gleicher beziehungsweise ähnlicher Struktur implementiert werden kann, können mehrere beziehungsweise alle implementierten Algorithmen von dieser Struktur profitieren.

Die Vorteile des AMP-Frameworks sind offensichtlich. Durch die Kombination globaler und lokaler Algorithmen werden deren Vorteile kombiniert.

Der Nachteil ist, dass durch die Intensifikation teilweise erheblich mehr Auswertungen der Zielfunktion und damit des Modells nötig sind. Auch müssen zusätzlich zur Implementation des Frameworks einige andere Algorithmen umgesetzt werden um das Framework zu gebrauchen.

Tabu Suche (TS)

Die Tabu-Suche erweitert die lokale Suche (oder einen anderen Algorithmus[24]) um eine Tabu-Liste und damit um die Fähigkeit lokalen Optima zu „entkommen“. Die lokale Suche terminiert hier nicht, wenn sich in der Nachbarschaft keine bessere Lösung findet. Der Algorithmus wählt immer die beste Lösung in der Nachbarschaft. Damit bei Verlassen eines lokalen Optimums der Algorithmus nicht in eine Endlosschleife verfällt, werden in einer Liste die bereits besuchten Parametersätze gespeichert. Diese werden in Zukunft nicht mehr besucht, sie sind tabu[25].

Die Tabu-Suche lässt sich relativ leicht in einer simplen Version implementieren. Für die Tabu-Suche existieren aber auch diverse Erweiterungen, die die einfache Version verbessern und erweitern[26].

Auch die Tabu-Suche hat wie alle Metaheuristiken den Vorteil, dass sie mehreren verschiedenen Algorithmen helfen kann lokale Optima zu verlassen beziehungsweise zu ignorieren und das globale Optimum zu finden. Andererseits kann sich die Laufzeit stark verlängern.

Simulated Annealing

Das SIMULATED ANNEALING ist in der Anlehnung an die Härtung von Stahl entstanden. Dort wird geschmolzenes Metall langsam abgekühlt. Dabei sammeln sich die einzelnen Atome in Positionen, die die potentielle Energie zueinander minimieren. Anfangs ist die Bewegung der

[24] Vgl. [Glo90] S. 74f
[25] Vgl. [Glo89] S.192
[26] Vgl. dazu [GL97] S. 6ff

Atome noch relativ hoch, später mit sinkender Temperatur bewegen sich die Atome langsamer. Wird das Abkühlen im richtigen Tempo vollzogen, entwickelt das Metall die gewünschte Festigkeit.

Das Simulated Annealing kann nun zum Beispiel auf die lokale Optimierung angewandt werden. Dort wird immer nur ein besserer Punkt als neue Lösung akzeptiert. Hier wird zusätzlich eine stochastische Komponente eingeführt, die auch schlechtere Lösungen mit einer Wahrscheinlichkeit größer 0 akzeptiert. Diese Wahrscheinlichkeit ist proportional zu einer „Temperatur“, die im Laufe der Optimierung sinkt. So werden anfangs relativ viele schlechte Lösungen akzeptiert. So entkommt der Algorithmus lokalen Optima, aber gegen Ende der Optimierung werden bei niedriger Temperatur praktisch keine schlechteren Lösungen mehr akzeptiert[27].

Simulated Annealing ist eine relativ alte und an vielen Problemen getestete Metaheuristik, die teilweise recht langsam ist, aber gute Ergebnisse liefert[28].

3.5 Auswahl der Algorithmen

Auf Grund der beschränkten Zeit während einer Diplomarbeit konnten nur drei Algorithmen implementiert. Deshalb musste eine Auswahl getroffen werden. In der engeren Wahl standen:

- Die Lokale Suche (FDSA und SPSA)

  Die lokale Suche basierend auf den Gradientenschätzern FDSA und SPSA hat den Vorteil, dass sie besonders mit kontinuierlichen Parametern relativ schnell lokale Optima findet. Zusätzlich können die Bestandteile als Grundlage für andere Algorithmen/Heuristiken verwendet werden. Zum Beispiel als Grundlage für das Simulated Annealing oder die Tabu-Suche.

- Der Genetische Algorithmus

  Der Genetische Algorithmus ist interessant, weil er bei verschiedensten Parameterräumen gute Ergebnisse liefern kann. Er stellt eine globale Suche dar. Er hat seine Fähigkeiten in zahlreichen Studien bewiesen[29] und ist allgemein in der Literatur gut dokumentiert.

- Der Partikel Schwarm Algorithmus

  Der Partikel Schwarm Algorithmus ist interessant, weil er relativ leicht zu implementieren ist und relativ schnell und mit wenigen Zielfunktionsauswertungen gute Ergebnisse für Optimierungsprobleme liefert[30].

[27] Vgl [Ana97] S.623 und [JW04] S.502
[28] Vgl. [Ing93] S. 9ff und [JW04] S. 506
[29] Vgl. [MC96] S. 15 und [GK04] S. 13
[30] Vgl. [KE95] S. 1947f

- Der Population Based Incremental Learning Algorithmus

  Das Population Based Incremental Learning bietet eine gute Alternative zum genetischen Algorithmus, da es eine schnellere Konvergenz bei ähnlich guten Resultaten verspricht.

- Das AMP-Framework

  Das AMP-Framework ist interessant, da Stärken verschiedener Algorithmen kombiniert werden können.

Die Wahl fiel auf einen lokalen Algorithmus mit FDSA- und SPSA-Gradientenschätzer, den Genetischen und den Partikel Schwarm Algorithmus.

Die Auswahl des lokalen Algorithmus wurde einerseits wegen seiner Schnelligkeit, andererseits, weil er im Zusammenspiel mit einer oder mehreren Metaheuristiken auch als globaler Optimierer eingesetzt werden kann, getroffen. Dieser Algorithmus ist eher für kontinuierliche Parameterräume konzipiert, soll aber mit einer Erweiterung für diskrete Parameter implementiert werden[31]. Von den beiden Gradientenschätzern überzeugt am ehesten der SPSA-Schätzer, da sich die beiden aber ähneln, sollen beide implementiert werden.

Der genetische Algorithmus wurde ausgewählt, da er bewährt ist und in praktisch allen Gebieten der Optimierung Anwendung findet. Zusätzlich gibt es zahlreiche Varianten, die nach Implementation einer Basisversion relativ leicht zu implementieren sind.

Der Partikel Schwarm Algorithmus ist mit relativ wenig Aufwand zu implementieren und verspricht gute Ergebnisse. Insbesondere die Eigenschaft, dass auch Populationen mit 10–20 Partikeln gute Ergebnisse liefern, ist für die Optimierung mit stochastischen Modellen gut geeignet.

Das AMP-Framework soll nicht implementiert werden, aber die drei zu implementierenden Algorithmen sollen die spätere Umsetzung womöglich begünstigen.

[31] Siehe [GHV99] S. 466ff

Kapitel 4

Implementierung der Optimierungsschnittstelle

Dieses Kapitel beschreibt, wie die im vorherigen Kapitel identifizierten Komponenten und der Ablauf der Optimierung umgesetzt wurden. In Abschnitt 4.1 wird die Umsetzung der bereits in Abschnitt 2.4 beschriebenen Elemente der Optimierung erläutert sowie die Kommunikation zwischen Optimierung und Skripting vorgestellt. Abschnitt 4.2 zeigt den Ablauf der Optimierung und dessen Umsetzung auf. Das Kapitel schließt mit einem Einblick in die Ergebnisausgabe und Benutzerschnittstelle.

4.1 Das Optimierungsskript – Struktur und Bestandteile der Optimierung

Die zuvor in Kapitel 2.4 identifizierten Bestandteile der Optimierung werden in einem Optimierungsskript zusammengefasst. Da alle Daten in einer Struktur gespeichert sind, können Redundanzen vermieden werden.

Damit Weiterentwicklungen möglich sind, wurden alle Teile der Optimierung als eigenständige, strikt getrennte Objekte implementiert. Dadurch entsteht zum Beispiel die Möglichkeit leicht neue Algorithmen zu implementieren, ohne dass andere Teile der Optimierungsschnittstelle geändert werden müssen.

Einige Teile, die für die Optimierung benötigt werden, konnten von der Skriptingschnittstelle übernommen werden. Wie in Kapitel 4.1.1 beschrieben, werden das Modell, der Analysator und die veränderbaren Variablen bereits im „Skripting“-Skript gespeichert. Für die Optimierung wurde das Skript um die Zielfunktion, die Abbruchbedingungen und den Optimierungsalgorithmus erweitert.

4.1.1 Einbinden des Skriptings

Wie bereits erwähnt wird das Skripting als Teil der Optimierung verwendet. Praktisch alle Teile des Skriptings konnten wieder verwertet oder können in Zukunft wieder verwendet werden. Dies ermöglichte die Übernahme großer Teile der Funktionalität und der Benutzeroberfläche. Das hatte nicht nur den Vorteil, dass diese nicht neu entworfen werden mussten, sondern beinhaltet auch, dass neue Funktionalität (andere Modelle, neue Variablen, etc.) ohne weitere oder mit minimaler Anpassung in der Optimierung genutzt werden können.

Im Skripting wird eine Klasse von änderbaren Variablen definiert. Diese bilden die Eingangsvariablen für die Optimierung. Das bedeutet, dass sie die „Stellschrauben“ sind, mit denen das optimale Ergebnis gesucht werden kann. Wenn nur über „änderbare“ Variablen optimiert werden kann, entsteht der Vorteil, dass bei der Erzeugung eines Skripts oder Optimierungsskripts bereits definiert werden kann, welche Variablen verändert werden dürfen. So kann zum Beispiel eine Arbeitsteilung realisiert werden, wie sie auch bei Kommunikation zwischen DaVincy und Expect existiert. Hat ein Experte die änderbaren Variablen definiert, ist es für den Optimierenden einfacher auf deren Basis einen guten Suchraum und eine Suchfunktion zu definieren. Da es im Skripting bereits möglich ist Mengen beziehungsweise Wertebereiche für Parameter vorzugeben, werden diese als Einschränkungen für den Suchraum genutzt.

Die Definition des Suchraums ist dabei sicherlich das erste Problem, das bei der Optimierung eines Modells auftritt. Hier sind grundsätzlich alle Parameterarten nutzbar, die im Skripting verwendet werden können. Die meisten Optimierungsalgorithmen benötigen Parameter oder funktionieren am besten mit Parametern, die in einer logischen Reihenfolge geordnet sind. Dabei sollten ähnliche Werte nahe beieinander liegen. Bei Wertebereichen ist die natürliche Ordnung der Zahlen vorgegeben. Bei Mengen von Einzelwerten, Vektoren oder Matrizen muss diese aber vom Nutzer bedacht werden.

Um überhaupt eine abstrakte Optimierung über alle Arten von Mengen und Wertebereichen zu ermöglichen, musste der Zugriff auf die verschiedenen Variablen vereinfacht werden. Da das Skripting intern mit Indizes auf seine Variablen zugreift, wurde die Optimierung über diese Indizes realisiert. Das hat den Vorteil, dass über jegliche Art von Parametertyp optimiert werden kann. Der Nachteil ist, dass der Nutzer bei der Definition von Mengen benachbarte Parameterausprägungen nebeneinander definieren muss. Zum Beispiel müssen die Zahlen 1 bis 5 als Menge in der Reihenfolge {1,2,3,4,5} und nicht gemischt {1,4,2,3,5} definiert werden, da sonst einige Algorithmen schlechte beziehungsweise zufällige Ergebnisse liefern. Die Mächtigkeit der Optimierung wächst aber durch die Möglichkeit der Optimierung mit Mengen von Vektoren und Matrizen stark an. Hier besteht zusätzlich Potenzial zur Weiterentwicklung. Zum Beispiel ist eine automatische Iteration oder gar Optimierung über Vektoren und Matrizen denkbar.

Das Skripting ist darauf ausgelegt, alle möglichen Permutationen der Parameter zu bilden und damit den Suchraum komplett zu analysieren. Für die Optimierung ist es aber wichtig, eine Menge einzelner Parametersätze auszuwerten. Um dies zu ermöglichen wurde eine OPTIMIZATIONSOLVER-Klasse definiert. An diese werden die zu analysierenden Parametersätze übergeben. Dadurch wird eine Entkoppelung von Optimierungsalgorithmus und Berechnung des Modells mit verschiedenen Parametersätzen erreicht. Die Koordination der Berechnung

erfolgt vollständig aus dieser Klasse. Das heißt, der OptimizationSolver wird von der Optimierung als Black Box betrachtet. So ist zum Beispiel ein Parametersatz-Cache möglich, in dem bereits ausgewertete Parametersätze gespeichert werden. Dem OptimizationSolver werden die zu berechnenden Parameter übergeben, intern berechnet und dann als fertige Ergebnisse zurückgegeben. Dadurch können Erweiterungen oder Änderungen der Berechnung (zum Beispiel paralleler Berechnung) einfacher umgesetzt werden. Im OptimizationSolver wird zu der Berechnung der Parametersätze wiederum auf das Skripting zugegriffen.

4.1.2 Algorithmen – Kern der Optimierung

Damit verschiedene Algorithmen implementiert werden können ohne den Ablauf der kompletten Optimierung zu verändern, müssen die Schnittstellen zwischen Optimierungsalgorithmen und dem allgemeineren Ablauf der Optimierung identifiziert werden. Die meisten Optimierungsalgorithmen (der Infinitesimal Perturbation Analysis Algorithmus und andere Algorithmen, die in die Analyse eingreifen, ausgenommen[1]) haben vier wichtige Schnittstellen, an denen sie Daten benötigen beziehungsweise bereitstellen. Diese sind:

- Die Initialisierung des Algorithmus (Eingabe einer Startposition).
- Die Rückgabe der vom Algorithmus gefundenen Parametersätze zur Auswertung des Modells und der Zielfunktion.
- Das Einlesen der durch Auswertung des Modells gewonnenen Werte.
- Die Rückgabe des besten gefundenen Parametersatzes am Ende der Optimierung.

Anhand dieser Schnittstellen wurden die Algorithmen in die Optimierung integriert[2]. Zusätzlich wurden verschiedene Standardelemente in Form abstrakter Klassen vorgefertigt (zum Beispiel Rückgabe des besten Ergebnisses) und können von allen Algorithmen verwendet werden. Dies führt zu einer weiteren Erleichterung bei der Implementierung neuer Algorithmen. Ist durch die komplette Optimierungsschnittstelle ein Rahmen geschaffen, dann ist der Aufwand neue Algorithmen zu implementieren relativ klein.

Von der Optimierungsschnittstelle losgelöst wurden die Algorithmen in drei logische Teile geteilt. Es wurden zum einen die Ablauflogik (den eigentlichen Algorithmus (siehe oben)), zum anderen die Konfiguration oder die Parametrisierung des Algorithmus und zuletzt die GUI, also die graphische Benutzeroberfläche (siehe Abb. 4.1), getrennt implementiert. Durch die Trennung von Funktionalität und GUI wird vor allem die Übersichtlichkeit der einzelnen Programmteile gefördert. Zudem ist die Benutzerschnittstelle so einfacher austauschbar. Durch die zusätzliche Trennung von Algorithmus und Parameterdaten könnten zum Beispiel mehrere verschiedene vordefinierte Defaulteinstellungen für einen Algorithmus bereitgestellt und dadurch die Parametrisierung vereinfacht werden.

[1] Siehe Kapitel 3
[2] Siehe Abschnitt 4.2

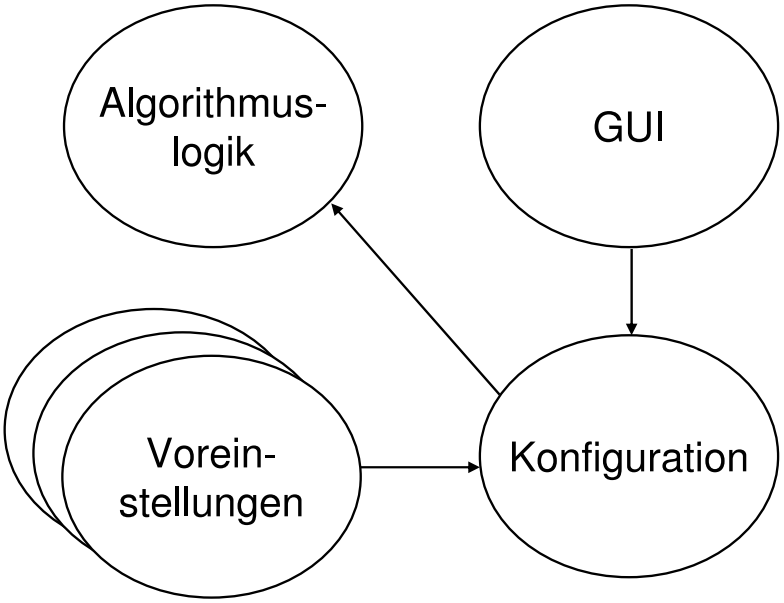


Abbildung 4.1: Trennung der Algorithmusfunktionalität von Konfigurationsdaten und Benutzeroberfläche

Das Arbeiten mit der XML-Struktur, in der die Variablen im Skripting gespeichert sind, ist für die Optimierungsalgorithmen nicht ideal. Hier werden zusätzliche Informationen gespeichert und das einfache Ändern der einzelnen Werte ist nur schwer möglich. Da in der Optimierung aber ständig die Werte einzelner Parametersätze geändert werden müssen, um zum Beispiel bei einem Partikel Schwarm Algorithmus die Position zu verändern, wurde hier eine eigene Struktur zur Speicherung der einzelnen Parametersätze implementiert. Zusätzlich werden für die verschiedenen Algorithmen auch einige Informationen gebraucht, die in dieser Struktur gespeichert werden können. Die Skriptingschnittstelle ermöglicht aber auch, wie schon in 4.1.1 erwähnt, über Indices den Zugriff auf die einzelnen Variablenausprägungen. Über die Indices wurden die Variablen im Skripting abstrahiert in die Optimierung übernommen.

Hierfür wurde die Schnittstelle OPTIMIZATIONPARAMETERSET implementiert. Das OptimizationParameterSet repräsentiert also eine Position im Suchraum oder einen Parametersatz. Da das Skripting aber auf diese Parametersätze nicht zugreifen kann, so wie umgekehrt die Optimierung nicht mit den Parametersätzen des Skriptings arbeiten kann, wurde hier eine OPTIMIZATIONPARAMETERSETFACTORY geschaffen. Diese bietet eine Schnittstelle, die Parametersätze übersetzen kann. Dadurch ist bei der Übergabe der Parametersätze an die Skriptingschnittstelle (zur Auswertung) eine einfache Transformation für eine Auswertung möglich.

Durch im Grundsatz (also in der Speicherung) gleiche Parametersätze wird auch gewährleistet, dass in einer Metaheuristik Daten zwischen verschiedenen Algorithmen ausgetauscht werden können. Im Hinblick auf eine spätere Implementation eines AMP-Frameworks wurde darauf geachtet, dass die lokalen Algorithmen, die mit nur einem Parametersatz arbeiten, auch mit den Parametersätzen der globalen Algorithmen arbeiten können.

In diesem Rahmen wurden die in Kapitel 3.5 ausgewählten Algorithmen implementiert.

Partikel Schwarm Algorithmus

Der Partikel Schwarm Algorithmus wurde nach der Beschreibung in Abschnitt 3.3 umgesetzt.

Das OptimizationParameterSet wurde als ein PARTIKEL implementiert. Das Partikel speichert als statische Variable die beste bisher gefundene Position des Schwarms. Dadurch steht diese Information wie gefordert allen Partikeln zur Verfügung. Eigenschaften, die nur die einzelnen Partikel betreffen (Position, beste eigene Position und Geschwindigkeit), werden lokal gespeichert.

Nach Berechnung der Fitness mit Modell und Zielfunktion werden den Partikeln die derzeitigen Fitnesswerte zugewiesen und wenn nötig die besten gefundenen Positionen aktualisiert. Sind alle Werte aktualisiert wird für alle Partikel die neue Geschwindigkeit mit Formel 3.9 berechnet und die Position wird aktualisiert. So stehen in den Partikeln alle Daten zur Verfügung, die für die Berechnung der neuen Position notwendig sind.

Genetischer Algorithmus

Zum Genetischen Algorithmus gehören fünf Elemente. Der Aufbau der Chromosomen, der Gen-Pool (oder die Population), das Ziehen der Eltern aus dem Gen-Pool, die Bildung neuer Chromosomen aus den gezogenen Chromosomen und die Mutation der neu gebildeten Elemente.

Integerwert	Bitkodierung	Graykodierung
0	0	0
1	1	1
2	10	11
3	11	10
4	100	110
5	101	111
6	110	101
7	111	111

Tabelle 4.1: Bit- und Graykodierung von Integer-Zahlen

Die Chromosomen wurden wieder mit dem OptimizationParameterSet implementiert. Die Eigenschaften eines Chromosomes beschränken sich auf die derzeitige Position und Fitness. Die Positionen der Chromosomen werden zwar auch auf Basis der Indices gespeichert, die genetischen Operatoren arbeiten aber auf Bitbasis. Dadurch wird die Verwendung allgemeiner genetischer Operatoren vereinfacht[3]. Die normale Bitkodierung hat den Nachteil, dass teilweise mehrere Bits einer kodierten Zahl verändert werden müssen, um die Zahl um eins zu

[3]Vgl. [Jak04] S. 27f

erhöhen (siehe Tabelle 4.1). Deshalb haben bei der einfachen Bitkodierung fortlaufende Werte teilweise eine große Hamming-Distanz, das heißt, dass sich von Zahl zu Zahl mehr als eine Ziffer ändert. Darum wurde zusätzlich eine Codierung im Gray-Code implementiert. Bitzahlen im Gray-Code haben die Eigenschaft, dass sich benachbarte Werte nur in einem Bit unterscheiden. Dies hat insbesondere Vorteile bei der Mutation, da hier meist nur ein Bit oder Allel verändert wird[4]. So erzeugt die Mutation eher Werte, die nah am ursprünglichen Wert liegen.

Um die Selektion und Erzeugung der Chromosomen möglichst variabel zu gestalten, wurden die in Kapitel 3.3 beschriebenen Operatoren so implementiert, dass diese kombiniert werden können. Hierfür wird bei der Parametrisierung des Algorithmus für jeden Operator angegeben, zu wie viel Prozent ein Algorithmus in Selektion oder Generation der Chromosomen verwendet werden soll.

Lokaler Algorithmus

Der lokale Algorithmus mit den SPSA und FDSA Gradientenschätzern (siehe Kapitel 3.2) wurde als Greedy Algorithmus implementiert. Das heißt, dass eine Änderung der derzeitigen Position nur akzeptiert wird, wenn die Fitness der neuen Position besser ist als die der alten.

Für lokale Algorithmen wurde eine minimale Implementation des OptimizationParameterSets umgesetzt. Hier werden nur die Positionsinformationen gespeichert. Diese Algorithmen können so auch mit anderen Implementationen des OptimizationParameterSets (zum Beispiel Chromosomen oder Partikeln) in einer Metaheuristik arbeiten.

Da der Ablauf von SPSA und FDSA Gradientenschätzer, beziehungsweise deren Integration in einen Algorithmus, grundsätzlich gleich sind, wurde für die Gradientenschätzer zusätzlich eine Schnittstelle implementiert. So können diese entweder in anderen Algorithmen wiederverwendet oder in demselben Algorithmus durch einen anderen Gradientenschätzer ersetzt werden.

Sowohl FDSA als auch SPSA wurden wie in Kapitel 3.2 beschrieben implementiert. Um eine Diskretisierung zu erreichen, werden die Schrittweiten der beiden Schrittweitenfunktionen a und c gerundet. Um die Unterschiede von Parametern verschiedener Größenordnung zu berücksichtigen, wurde ein „Größenordnungsmultiplikator“ eingeführt. Hier kann ein relativer Multiplikator für die Parameter angegeben werden. Ist dieser zum Beispiel 0.01 (1%), dann bezieht sich die Schrittgröße 1 auf ein Prozent der Größenordnung jedes Parameters. Ist ein Parameter variierbar von 0–100 mit Schrittgröße 1, dann entspricht die relative Schrittgröße 1 auch einer absoluten Schrittgröße 1. Variiert der Parameter aber von 0–1000, dann entspricht die relative Schrittgröße 1 der absoluten Schrittgröße 10.

4.1.3 Zielfunktion & Abbruchkriterien

Die Zielfunktion, über die ein Modell optimiert werden soll, ist für jedes Modell anders. Eine allgemeingültige Zielfunktion ist auf Grund der vielfältigen Modelle, die denkbar sind, un-

[4] Vgl. [Rot] S. 7ff

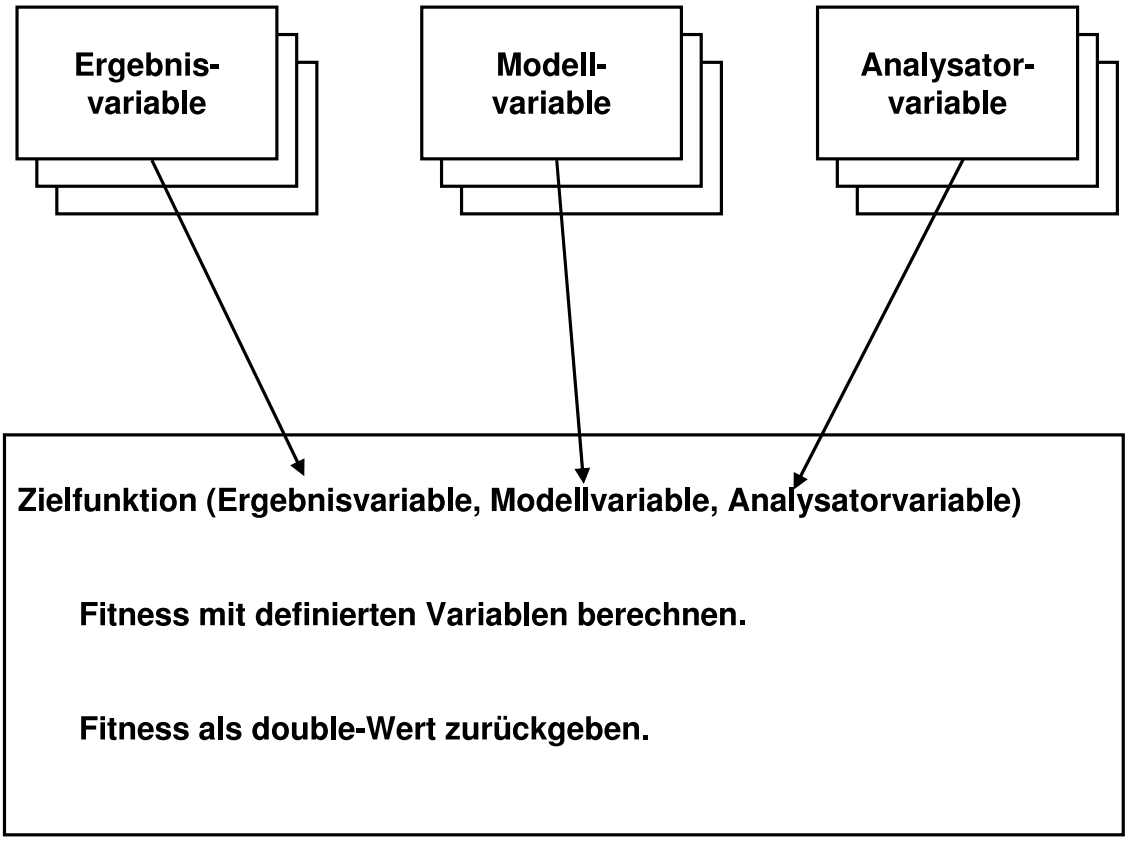
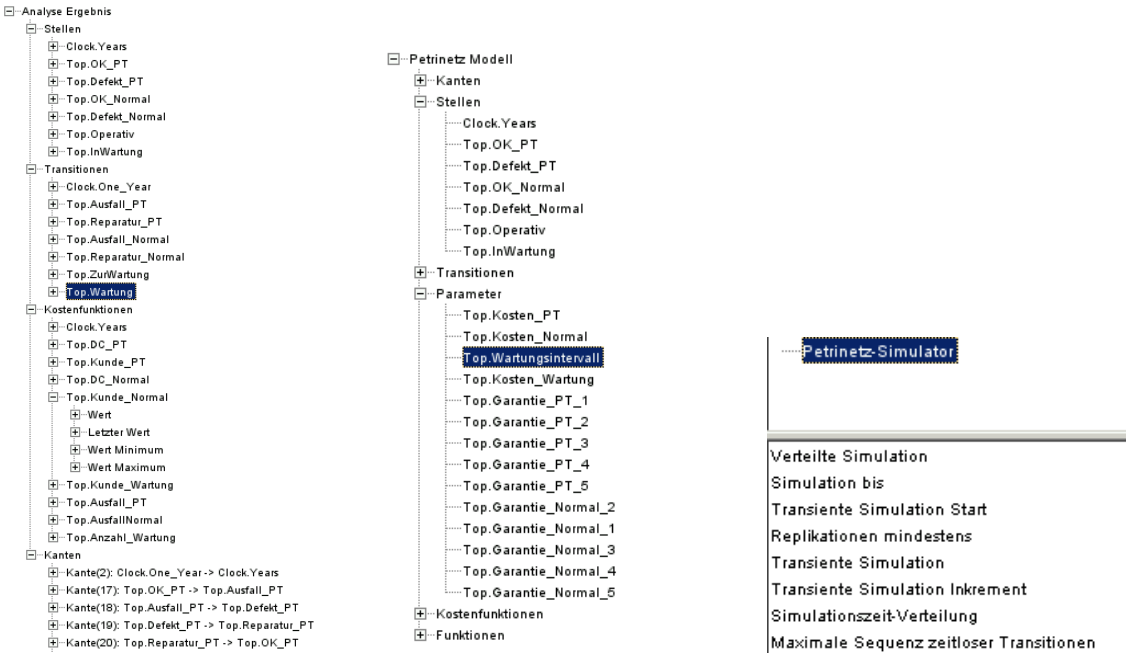


Abbildung 4.2: Zielfunktion und auslesbare Variable

möglich. Eine gute Definition der Zielfunktion ist aber für die Optimierung, wie schon vorher angemerkt, sehr wichtig. Um dem Nutzer alle Möglichkeiten bei der Berechnung der Zielfunktion zu lassen, soll die Zielfunktion als Java-Quellcode eingegeben werden. Die Definition einer Zielfunktion, die auf mathematischen Grundlagen (Addieren, Subtrahieren, Potenzieren, etc.) beruht, sollte auch Java-Unerfahrenen nach einer kurzen Einweisung möglich sein. Durch die Definition der Zielfunktion in Java steht dem Nutzer theoretisch die gesamte Mächtigkeit von Java zur Verfügung.

Zusätzlich stand in Expect eine Umgebung für das Kompilieren dynamischen Codes zur Verfügung. Diese wurde für die Elemente von Petri-Netzen genutzt (Definition von eigenen Verteilungsfunktionen, ...). Die Umgebung wurde für die Optimierung so angepasst, dass Funktionen mit Eingabeparametern kompiliert und genutzt werden können. Dies ist für die Zielfunktion wichtig, da sie immer wieder mit den verschiedenen, durch Analyse des Modells ermittelten Parameterwerten aufgerufen werden muss.

Auch die Eingangsgrößen für die Zielfunktion sind von Modell zu Modell, das optimiert werden soll, verschieden. Zu den Eingangsvariablen für die Zielfunktion gehören Ergebnis-, Modell- und Analysatorvariable. Ergebnisvariable werden bei der Analyse des Modells berechnet. Sie sind Kenngrößen, wie zum Beispiel die durchschnittliche Markenanzahl in einer Stelle eines Petri-Netzes (zum Beispiel in einem Puffer), die durch Analyse des Modells ge-



(a) Ergebnisvariable (b) Modellvariable (c) Analysatorvariable

Abbildung 4.3: Baumstrukturen für die Variablen der Zielfunktion

wonnen wurden. Modellvariable sind alle Ausprägungen oder Parameter des zu optimierenden Modells. Besonders interessant für die Optimierung sind die Modellparameter, die veränderbar sind (die veränderbaren Variablen siehe Abschnitt 4.1.1). Aber nicht nur änderbare Parameter können eingelesen werden. Durch Einlesen verschiedener Parameter des Netzes kann so die Optimierung auf Netzebene neu parametrisiert werden. Ist zum Beispiel in einem Modell eine Anzahl an Arbeitern variabel, so könnte jeder Arbeiter Kosten von x verursachen. Diese werden dann mit den Ergebnisvariablen in der Zielfunktion vereint.

Zusätzlich können Variable aus dem Analysator in der Zielfunktion verwendet werden. Eine Analysatorvariable ist zum Beispiel die Länge der Simulation eines Petri-Netzes. Analysatorvariable sind, im Gegensatz zu den anderen Variablenarten, für die Zielfunktion nicht unbedingt notwendig. Die Analysatorvariablen verändern sich während einer Optimierung nicht, deshalb könnten sie auch per Hand in die Zielfunktion eingegeben werden. Durch die Kodierung als Variable ergibt sich der Vorteil, dass für eine Optimierung, bei der in der Zielfunktion verwendete Analyseparameter (zum Beispiel die Simulationszeit) verändert wurden, die Zielfunktion nicht angepasst werden muss.

Die Variablen sind im Skripting als Baumstruktur vorhanden. Sie werden skriptingintern mit einem Pfad identifiziert. Die Verwendung eines Pfades im Java-Quellcode ist fehlerträchtig und für den Nutzer umständlich, da er den genauen und ganzen Pfad zu einer Variablen angeben muss. Deshalb sollen die Variablen als normale Java-Variablen definiert werden. Zur Definition wählt der Nutzer die Variablen in einem Dialog aus und vergibt dann jeder Variablen einen Variablennamen, der dann in der Java-Zielfunktion (siehe auch Abbildung 4.3) verwen-

det werden kann. Damit besteht jede Variable aus einem **Pfad** und einem **Namen**. So können die Ergebnis-, Modell- und Analysatorvariablen vor der Berechnung der Zielfunktion aus den Skriptingergebnissen ausgelesen und dann durch den Namen eindeutig in der Zielfunktion zugeordnet werden.

Wie schon in Kapitel 2.4 erwähnt, muss es für die Optimierung gewisse Abbruchkriterien geben. Zum einen kosten Zeit sowie Rechenzeit Geld und zum anderen sinkt mit immer längerer Laufzeit die relative Verbesserung, die mit der Optimierung erreicht wird.

Folgende Abbruchkriterien wurden implementiert:

- Die maximale Zeit

  Die maximale Zeit, die die Optimierung laufen soll. Hier wird die verstrichene Zeit vom Start der Optimierung berechnet.

- Die maximale Rechenzeit

  Die maximale Rechenzeit, die von der Optimierung gebraucht werden soll. Hier wird die Zeit begrenzt, die bei der Analyse des Modells verwendet wird. Die Optimierung terminiert, wenn die maximale Analysezeit durch Auswertungen erreicht wurde. Dies ist zum Beispiel bei der parallelen Berechnung interessant. Hier könnte die Zeit aufsummiert werden, die von mehreren Rechnern zur Analyse verwendet wurde.

- Die maximale Anzahl an Generationen/Wiederholungen

  Die Zahl an Generationen beziehungsweise Wiederholungen (zum Beispiel im Fall der lokalen Suche), nach der die Optimierung abgebrochen wird.

- Die maximale Anzahl an Auswertungen der Zielfunktion

  Da es ein Cache für die Auswertungen des Modells gibt, können zeitlich kostspielige Auswertungen des Modells teilweise unterbleiben. Dieses Abbruchkriterium beschränkt die Anzahl der realen Auswertungen eines Modells.

- Der relative Unterschied zwischen der Fitness der jetzigen und der letzten Generation

  Die relative Verbesserung der Fitness ist ein guter Wert um den Fortschritt der Optimierung zu überwachen. Dieses Abbruchkriterium bricht die Optimierung dann ab, wenn die relative Verbesserung über eine gewisse Anzahl von Iterationen nicht größer als ein einzugebender Schwellwert ist.

Auch für die Abbruchkriterien wurde ein Interface entworfen. Wie bei den Algorithmen ist so das Einbinden neuer Abbruchkriterien erleichtert. Es wurden zwei Arten von Abbruchkriterien identifiziert. Dazu gehören einfache, die nur einen Wert überprüfen (zum Beispiel die Zeit), und solche, die einen Schwellwert und einen Zähler haben, der besagt, wie oft der Schwellwert schon verletzt wurde. Die wesentliche Funktion der Abbruchkriterien ist die Methode $isStopCriteriaSatisfied(currentValue)$, der der derzeit relevante Wert übergeben wird. Dies ist zum Beispiel die bis jetzt verstrichene Zeit. Die Methode gibt zurück, ob das Abbruchkriterium erfüllt ist.

4.2 Der Optimierer – Koordination der Optimierung

Der Optimierer vereint und koordiniert die einzelnen Teile der Optimierung. Eingaben vom Nutzer (zum Beispiel Starten, Pausieren und Stoppen der Optimierung) werden an den Optimierer weitergeleitet, der die Befehle dann umsetzt. Dadurch wird eine Trennung von GUI und der Ablauflogik der Optimierung geschaffen. So könnte zum Beispiel eine Optimierung auch über ein XML-Skript angestoßen werden und die Ergebnisse als ein weiteres XML-Skript zurückgeben werden. Dies wäre zum Beispiel sinnvoll, wenn die Optimierung aus der DaVinci-Plattform gestartet werden würde (siehe Kapitel 2.2).

Der Optimierer führt den Rahmenalgorithmus für die Optimierung durch (siehe Kapitel 2.4, Abbildung 2.5). Das Optimierungsskript wird an den Optimierer übergeben und die einzelnen Objekte der Optimierung werden initialisiert. Anschließend läuft der Optimierungsalgorithmus bis der Nutzer die Optimierung abbricht oder bis ein Abbruchkriterium erreicht wird.

Die bei der Optimierung auftretenden Ereignisse werden über die Event-Listener Struktur von Java ausgegeben und weiterverarbeitet. Die Ereignisse (Events), die während der Optimierung auftreten, werden mit den dabei anfallenden Informationen an die registrierten Listener weitergereicht. Die Menge der Listener ist dabei nicht begrenzt. Dadurch ist es möglich, die Information über eine Schnittstelle an mehrere Stellen zu verteilen und diese Verteilung auch leicht zu erweitern. Zu den Listenern gehören zum Beispiel die direkte Ausgabe an den Nutzer und die Speicherung für die Ergebnisausgabe.

Die wichtigsten bei der Optimierung auftretenden Ereignisse sind:

- Der Start, das Pausieren, das Abbrechen und das Ende der Optimierung.
- Das Finden einer neuen besten Lösung.
- Der Abschluss einer Wiederholung beziehungsweise Generation.
- Ein bei der Optimierung auftretender Fehler.

Durch die Struktur von Events und Listener können auch neue Programmteile relativ leicht eingebunden werden.

4.3 Benutzerschnittstelle und Ergebnisausgabe

Die Benutzerschnittstelle wurde wie die GUI des Skriptings als Wizard aufgebaut. Dadurch lassen sich zum einen Teile der Oberfläche übernehmen, zum anderen wird die Benutzerführung vereinfacht. Der Nutzer wird der Reihe nach durch einige Registerkarten geführt, die die nötigen Elemente der Optimierung parametrisieren.

Die Registerkarten sind:

- **Übersicht** (siehe Abbildung A.1):

  Alle Informationen werden hier auf einen Blick angezeigt. Fehlt etwas zum Starten der Optimierung, wird es rot markiert. Dadurch kann der Benutzer sich einen Überblick verschaffen, welche Einstellungen er schon vorgenommen hat und welche noch fehlen.

- **Skripting** (siehe Abbildung A.2):

  Aus dem Skripting übernommen wurde die Auswahl der veränderbaren Variablen. Aus diesem Register können veränderbare Variablen von Modell und Analysator ausgewählt werden. Aus Gründen der Konsistenz können Variable hier nur hinzugefügt aber nicht verändert werden. Das Ändern von Variablen ist nur in den Nebenbedingungen möglich. So kann auch von anderen Tools (siehe Kapitel 2.2) auf diese Variablen zugegriffen werden.

- **Nebenbedingungen**(siehe Abbildung A.3):

  Hier können für die vorher definierten änderbaren Modellvariablen Werte beziehungsweise Wertebereiche angegeben werden. Die Wertebereiche bilden dann im Anschluss den gültigen Suchraum für die Optimierung. Es sind grundsätzlich alle Werte änderbar, die im Skripting verfügbar sind. Durch die Trennung von Auswahl und Parametrisierung wird der Nutzer gezwungen sich zuerst zu überlegen, welche Werte veränderbar sind, und anschließend, welchen Wertebereich sie einnehmen können. Variable aus dem Analysator können nur einzelne Werte und keine Wertebereiche annehmen.

- **Zielfunktion** (siehe Abbildung A.4):

  Hier kann die Zielfunktion für die Optimierung definiert werden. Diese Registerkarte besteht aus zwei Teilen. Im oberen Teil können „ähnlich“ den Java-Konventionen Variable definiert werden, die im unteren Teil in der Zielfunktion verwendet werden. Es können die drei Variablenarten, die in Abschnitt 4.1.3 vorgestellt wurden, definiert werden. Die Auswahl funktioniert über einen Dialog mit der im gleichen Abschnitt bereits erwähnten Baumstruktur (Abb. 4.3). Da die Struktur für Ergebnisse, Modell und Analysator verschieden ist, gibt es verschiedene Dialoge. Die ausgewählten Variablen werden

Herkunft	Typ	Variablen Name	Variablen Pfad
Modell	int	interval	/parameters/Top.Wartungsintervall/iValue

Abbildung 4.4: Definition einer Variablen in der Optimierung

  in einer Tabelle (siehe Abbildung 4.4) in einer Reihenfolge ähnlich der Java-Definition von Variablen angezeigt. Die Herkunft entspricht der Art der Variablen, also den Ergebnissen, dem Modell oder dem Analysator. Der Typ wird automatisch ausgelesen und ist der Java-Typ, der auch in der Zielfunktion zur Verfügung steht. Der Name ist der einzige Teil, der direkt veränderbar ist und wird selbst gewählt. Alle anderen Variablen können nur über Dialoge verändert werden. Der Pfad entspricht dem Pfad in der

Skripting-Struktur. Dadurch wird ein einfacher und relativ sicherer Zugriff auf die im Skripting verfügbaren Werte gewährleistet.

Sind die Parameter definiert, kann die Zielfunktion im unteren Teil eingegeben werden. Durch eine Checkbox lässt sich die Zielfunktion während der Optimierung entweder minimieren oder maximieren. Wäre nur eine Maximierung möglich, müsste der Nutzer die Funktion bei der Eingabe umstellen, um zum Beispiel eine Zielfunktion mit positiven Werten zu minimieren. Hier würde eine weitere Fehlerquelle entstehen.

Die Zielfunktion wird sofort nach Eingabe kompiliert und eventuelle Fehler werden dem Nutzer angezeigt. Dadurch tritt der Fehler nicht erst bei der eigentlichen Optimierung auf und lässt sich sofort identifizieren und beheben.

- **Optimierungsparameter** (siehe Abbildung A.5):

  Auf diesem Dialog können Optimierungsparameter verändert werden. Der Algorithmus kann eingestellt und parametriert werden. Für die verschiedenen Algorithmen könnten durch die Trennung von Logik, GUI und Konfiguration Standard- beziehungsweise Defaultkonfigurationen vorgegeben werden. Dadurch ist es relativ leicht, die Möglichkeit, mehrere verschiedene Konfigurationen für einen Algorithmus vorzugeben, nachzurüsten[5]. Dadurch würde dem Anwender eine zusätzliche Hilfestellung bei den nicht trivialen Einstellungen der Algorithmenparameter gegeben. Zurzeit ist nur die Vorgabe einer Einstellung möglich. Auch die Startparameter des Algorithmus, also entweder eine Position oder auch eine ganze Population, können eingegeben werden. Sind bereits gute Lösungen bekannt, kann die Optimierung verkürzt werden. Mit den Endwerten einer vorhergegangenen Optimierung kann diese fortgesetzt werden.

  Zusätzlich können hier auch die Abbruchkriterien gesetzt werden, die bestimmen, wann die Optimierung terminiert. Diese sind wichtig, wie schon in Abschnitt 4.1.3 erläutert.

- **Optimierung** (siehe Abbildung A.6):

  Dieses Register dient zur Steuerung und mit der Ergebnisausgabe zur Überwachung der laufenden Optimierung. Hier kann die Optimierung gestartet, unterbrochen, fortgesetzt und abgebrochen werden. Dadurch ist es dem Anwender auch möglich, die Optimierung zu beenden, wenn das Ergebnis die gewünschte Güte erreicht hat, ohne ein Abbruchkriterium zu erreichen. Da die Optimierung sehr lange dauern kann, werden Fortschrittsinformationen in einem Nachrichtenfeld angezeigt. Weiterhin werden die derzeit besten gefundenen Parameter in einer Tabelle angezeigt. Über die Buttons „Detaillierte Informationen“ und „Zeige jetzige Parameter“ können zusätzliche Informationen angezeigt werden, die bei der Suche nach geeigneten Parametern für Algorithmen und der Suche nach Fehlern besonders interessant sind.

- **Ergebnisse** (siehe Abbildung A.7):

  Auf dem letzten Dialog werden die ermittelten Ergebnisse der Optimierung angezeigt. Hierzu gehört der beste Parametersatz mit dazugehöriger Fitness. Für jede Generation

[5] Siehe dazu den Ausblick Kapitel 6.3

oder Wiederholung wird das beste Ergebnis und die durchschnittliche Fitness bei populationsbasierten Algorithmen angezeigt. Dadurch ist eine Überwachung der Konvergenz des laufenden Algorithmus möglich. Zusätzlich können die Ergebnisse als Datei gespeichert werden, was eine Weiterbearbeitung beziehungsweise Aufbereitung ermöglicht.

Da die Ergebnisse nicht nur während der Optimierung zur Verfügung stehen sollen, muss es Möglichkeiten geben diese zu speichern. Zusätzlich ist oftmals nicht **nur** der ermittelte optimale Wert interessant.

Die Daten, die während der Optimierung entstehen und im Ergebnis-Register angezeigt werden, können nach Abschluss der Optimierung gespeichert werden. Dazu wurde ein OPTIMIZATIONWRITER als Interface definiert, der zurzeit für eine Ausgabe in *MS Excel* implementiert ist. Die Daten werden über Schnittstellen an den Writer übergeben, der sie dann in diesem Fall in eine Excel-Datei schreibt. Dort können dann zum Beispiel Diagramme über den Optimierungsverlauf erstellt werden. Schnittstellen existieren zum Beispiel für Daten, die in jeder Generation, und für solche, die am Ende der Optimierung ermittelt werden. Die Implementation über ein Interface hat hier den großen Vorteil, dass leicht andere Speicherungsarten für die Optimierungsdaten implementiert werden können. Zum Beispiel könnten Daten auch als HTML-Datei oder im Expect-Skriptingschnittstellenformat gespeichert werden.

Kapitel 5

Experimente / Optimierung

In diesem Kapitel werden einige mit den bereits vorgestellten Algorithmen durchgeführte Experimente beschriebenen. Im ersten Abschnitt wird an einem einfachen nachvollziehbaren Beispiel der Ablauf vorgestellt. In Abschnitt 5.2 wird ein Life-Cycle-Modell für LKW optimiert. Anschließend wird in Abschnitt 5.3 ein Teil des sehr umfangreichen Modells eines Logistikcenters optimiert. Der Unterschied zwischen lokalen und globalen Algorithmen soll in Abschnitt 5.4 verdeutlicht werden. Abschließend werden die Ergebnisse der Algorithmen an den drei Modellen kommentiert und bewertet. Die Experimente werden verteilt auf vier Bladeservern mit je zwei Prozessoren durchgeführt.

5.1 Einfaches Petri-Netz-Modell

5.1.1 Modellbeschreibung und Parametrisierung

Das erste Modell (siehe Abbildung 5.1) ist ein einfaches Petri-Netz, welches ein Bauteil modelliert, das drei Zustände (OK, Fehler und Wartung) annehmen kann. Das Netz ist im Anfangszeitpunkt im Zustand OK. In regelmäßigen Zeitabständen (Wartung) kann das Bauteil ausgetauscht werden. Das Bauteil fällt in unregelmäßigen Abständen aus (Fehler).

Die Lebenserwartung des Bauteils wird durch eine Normalverteilung mit einem Mittelwert $\mu = 5000$ und einer Standardabweichung $\sigma = 100$ modelliert. Die Dauer der Reparatur des Bauteils ist ebenfalls normalverteilt mit $\mu = 100$ Zeiteinheiten mit $\sigma = 10$. Eine Wartung wird in deterministischen Zeitabständen durchgeführt, wobei hier das Optimum zu ermitteln ist. Der Austausch bei einer Wartung dauert 1 Zeiteinheit.

Es soll die Verfügbarkeit, also die Zeit im Zustand OK, optimiert werden. Dabei wird die Zeit, die das Bauteil nicht im Zustand OK ist, mit 5 000 multipliziert und jede Wartung mit 10 000 negativ bewertet.

Das Wartungsintervall kann zwischen 1 000 und 11 000 Zeiteinheiten jeden Wert in Einerschritten annehmen. Die einzige veränderbare Variable ist das Wartungsintervall.

Die zu optimierende Zielfunktion setzt sich zusammen aus der Zeit, in der das Bauteil nicht

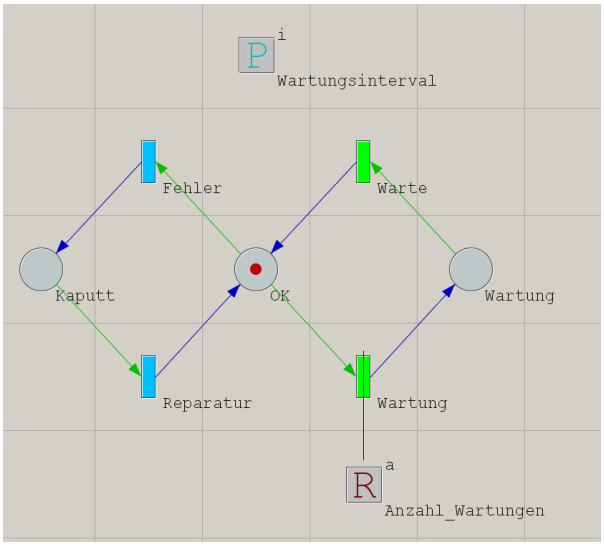


Abbildung 5.1: Petri-Netz-Modell 1

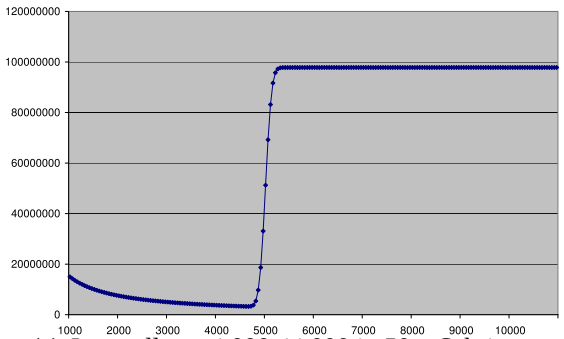
im Zustand OK ist, und der Anzahl der Wartungen, die zusätzlich negativ bewertet werden.

Um diese Werte zu ermitteln, werden aus den Ergebnissen der Modellauswertungen die Erwartungswerte der Wahrscheinlichkeit gebraucht, dass das Netz im Zustand Wartung ($p(W)$) und Fehler ($p(A)$) ist, sowie die erwartete Anzahl an Wartungen (n_W,) die durchgeführt wird. Um die Zeit, die in den Zuständen Fehler und Wartung verbracht wurde, zu errechnen, wird zusätzlich die Simulationszeit t gebraucht.

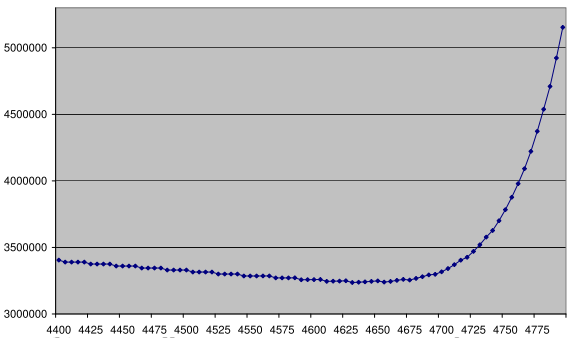
Für die Analyse des Petri-Netzes wurde eine Replikationsanzahl von 50 000 gewählt. Bei einem Wartungsintervall von 4 666 und einer Stichprobe von 10 Modellauswertungen ergibt sich daraus eine Durchschnittswert μ von 3 253 430 und eine Standardabweichung σ von 16.25 für den Zielfunktionswert.

Daraus ergibt sich die vom Wartungsintervall WI abhängige Zielfunktion 5.1:

$$ZF(WI) = 5000t\ (E(p(A)) + E(p(W))) + 10000E(n_W) \tag{5.1}$$



(a) Intervall von 1 000-11 000 in 50er Schritten



(b) Intervall von 4 400-4 800 in 5er Schritten

Da sich das Verhältnis zwischen Zielfunktion und Parameter für dieses Modell gut darstellen und relativ schnell auswerten lässt, wurde das Modell zusätzlich mit Hilfe der kompletten Enumeration ausgewertet. Die entstehenden Graphen sind in den Abbildungen 5.2(a) und 5.2(b) zu sehen.

Das optimale Wartungsintervall liegt ungefähr bei 4 630 mit einem Fitnesswert von circa 3 250 000. Ab einem Wartungsintervall von circa 5 500 bleibt die Zielfunktion knapp unter einem Wert von 100 Millionen.

Die Erwartung an die Optimierungsalgorithmen ist, dass alle relativ schnell zum Optimum hin konvergieren. Für die lokalen Algorithmen wird die Startposition entscheidend sein. Ist das Wartungsintervall am Anfang deutlich höher als 5500 werden die lokalen Algorithmen dort stagnieren.

5.1.2 Ergebnisse

ALGORITHMUS	FITNESS	ERGEBNIS	LAUFZEIT	AUSWERTUNGEN	% DES SUCHRAUMS
PSA	3 235 967	4630	ca. 46 min	155	1,55
GA	3 237 697	4633	ca. 72 min	242	2,42
LA(2 270)	3 300 086	4526	ca. 32 min	107	1,07
LA(7 166)	97 808 075	6867	ca. 9 min	31	0,31

Tabelle 5.1: Übersicht Ergebnisse des einfachen Petri-Netz-Modells

Wie erwartet haben die beiden globalen Algorithmen das globale Optimum gefunden. Der lokale Algorithmus scheiterte wahrscheinlich daran, dass die Fitnessfunktion nach dem Optimum stark abfiel. Dadurch wurden die Schrittgrößen und damit auch der Gradient schnell sehr klein. Hier würde eine intelligente adaptive Schrittweitenänderung der lokalen Suche zugute kommen. Alternativ würde sich eine gröbere Unterteilung des Suchraums positiv auf die Performanz der lokalen Verfahren auswirken.

Im Vergleich zu einer kompletten Auswertung aller möglichen Parameter für das Wartungsintervall bringen die globalen Algorithmen einen deutlichen Vorteil. Es wurden hier weniger als 3% der für eine komplette Enumeration benötigten Auswertungen durchgeführt. Allerdings wurde bei den beiden globalen Algorithmen ein nahezu optimaler Wert schon in der Startposition gefunden.

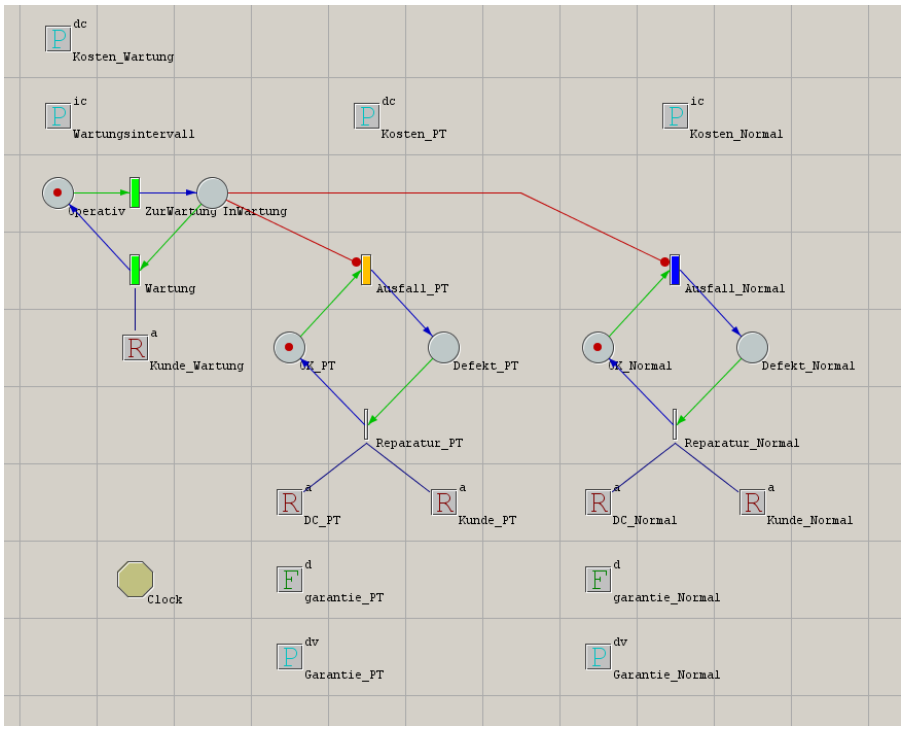


Abbildung 5.2: Life-Cycle-Modell

5.2 Life-Cycle-Modell

5.2.1 Modellbeschreibung und Parametrisierung

Das zweite Modell ist ein Life-Cycle-Modell der DaimlerChrysler Forschung, in dem eine mögliche Garantie- und Kulanzstrategie für LKW modelliert ist. Es werden zwei verschiedene Bauteiltypen modelliert, die unterschiedliches Ausfallverhalten zeigen. Für dieses Modell soll das optimale Wartungsintervall gefunden werden, das eine hohe Verfügbarkeit der LKW ermöglicht. Zusätzlich soll eine optimale Garantie- und Kulanzstrategie gefunden werden. Hier sollen insbesondere die geleisteten Garantie- und Kulanzzahlungen sowie die Kosten für den Kunden in die Zielfunktion einfließen. Dabei soll aber auch die Marketingwirkung einer hohen Garantie berücksichtigt werden.

Die Gewährleistung beziehungsweise Kulanz wird in fünf Stufen je nach Alter und Fahrleistung der LKW gewährt. Die Aufteilung in Stufen ist in Tabelle 5.2 aufgelistet. Dabei wird zum Beispiel eine Gewährleistung der Stufe 1 für Normalteile erbracht, wenn das Fahrzeug weniger als 2 Jahre alt und weniger als 200 000 Kilometer gefahren ist. Hierbei werden zum Beispiel in der ersten Stufe 100% der Kosten übernommen, in der zweiten 80% etc.. Dabei kostet eine Reparatur der Teile des Antriebsstrangs (Power Train = PT) 512 und die Reparatur eines normalen Teils 250 Geldeinheiten. Eine Wartung kostet den Kunden 750 Geldeinheiten.

Die Ausfallwahrscheinlichkeit der Teile des Antriebsstrangs wurde mit einer 3-parametrigen Weibullverteilung (ausfallfreie Zeit 0, Formparameter 1.7, charakteristische Le-

	Antriebsstrangteile(PT)		Normalteile	
STUFE	ALTER (JAHRE)	GEFAHRENE KM	ALTER (JAHRE)	GEFAHRENE KM
1	2	250 000	1	200 000
2	3	350 000	2	300 000
3	4	400 000	3	400 000
4	4	450 000	4	500 000
5	5	600 000	5	600 000

Tabelle 5.2: Zuordnung der Garantie- und Kulanzstufen im Hinblick auf Antriebsstrang- und Normalteile zu Fahrzeugalter und gefahrenen Kilometern

bensdauer 280 000 km) modelliert. Normalteile fallen exponentialverteilt (Feuerrate 0.42×10^{-6} oder durchschnittlich etwa alle 238 000 km) aus. Das Modell soll über 1 500 000 km simuliert werden. Dabei wird die jährliche Kilometerleistung der LKW mit einer Lognormalverteilung ($\mu = 150\,000$ km und σ = 35 000 km) approximiert.

Die **veränderbaren Parameter** sind das Wartungsintervall und die beiden Garantiestaffeln. Hier wurde jede Staffel mit 5 Parametern modelliert. Jede Garantiestufe hat also einen entsprechenden Parameter, wobei der erste Parameter immer 1 (entspricht der gesetzlichen Gewährleistung, also 100%) und der fünfte immer 0 (entspricht dem Ablauf der Garantiebeziehungsweise Kulanzleistungen) ist. Die drei Zwischenstufen können in 5% Schritten frei zwischen 0% und 100% schwanken. Das Wartungsintervall ist von 25 000 bis 250 000 km in 1 000 km Schritten frei einstellbar. So entstehen 19 383 143 346 verschiedene mögliche Parametersätze.

Zur Bildung der **Zielfunktion** werden die Kosten, die für die DaimlerChrysler AG entstehen addiert. Auch die Kosten, die den Kunden entstehen, fließen in die Zielfunktion ein, aber nur zu 10%. Wartungen und Ausfälle werden respektive mit 30 und 300 Geldeinheiten zusätzlich negativ bewertet, da Kunden durch Ausfälle und Wartungen zusätzliche Kosten entstehen. Da hohe Garantieleistungen erwünscht sind, werden diese zusätzlich positiv bewertet. Jeder 10 prozentige Teil der variablen Garantieleistung wird mit 50 Geldeinheiten multipliziert und dazugerechnet.

$$ZF(WI, G_N, G_{PT}) = k_{PT} + k_N + \frac{k_{cPT} + k_{cN} + k_W}{10} + m_A + m_W - b_G \qquad (5.2)$$

Die aus dem Ergebnis des Modells ausgelesenen Werte sind die Kosten der DaimlerChrysler AG für Normalteile und für Teile des Antriebsstrangs (k_N und k_{PT}), die entsprechenden Kosten für den Kunden (k_{cN} und k_{cPT}) und die Kosten für die Wartungen (k_W). Der Malus für Ausfälle (m_A) und Wartungen (m_W) wird durch Multiplikation des Erwartungswerts der Ausfälle mit der oben beschriebenen Bewertung errechnet. Der Bonus für hohe Garantieleistungen (b_G) errechnet sich aus der Summe der sechs variablen Garantieparameter multipliziert mit der Bewertung von 50 Geldeinheiten. Die verwendete Zielfunktion in Abhängigkeit von Wartungsintervall WI, sowie Garantie- und Kulanzstrategie für Normalteile G_N und Antriebsstrangteile G_{PT} ist in Gleichung 5.2 zu sehen.

Die voreingestellten (gegebenen) Parameter waren:

- Wartungsintervall: 100 000 Kilometer.
- Garantiestaffelung Normal: 1.0, 0.5, 0.4, 0.2 , 0
- Garantiestaffelung PT: 1.0, 0.8, 0.6, 0.4 , 0

Mit der Voreinstellung wurde eine Fitness von 4 792 ermittelt. Die Erwartungen an die Optimierung sind, dass gute Ergebnisse erzielt werden. Die lokale Suche mit FDSA- und SPSA-Gradientenschätzern könnte durch die relativ kleinen Dimensionen der Garantieparameter gestört werden.

Für die Modellauswertungen wurde eine Replikationszahl von 200 000 gewählt, die auch von der DaimlerChrysler AG zur Auswertung dieses Modells verwendet wurde. Mit den voreingestellten Parametern ergab sich hier (bei einer Stichprobe mit 10 Modellauswertungen) ein Mittelwert von $\mu = 4792.13$ bei einer Standardabweichung σ von 1.19.

5.2.2 Ergebnisse

Mit den ursprünglichen Werten verglichen, ergibt sich damit eine Verbesserung um ca. 7%. Das Wartungsintervall hat auf den Ausfall der Normalteile auf Grund der Exponentialverteilung keinen Einfluss. Da aber trotzdem Kosten für die Wartung dieser Teile entstehen, haben die Normalteile einen negativen Einfluss auf die Länge des Wartungsintervalls: Das heißt das Intervall verlängert sich. Diese Teile sollten gar nicht gewartet werden, da die Wartung hier (insbesondere für die Kunden) die Kosten erhöht, aber keinen positiven Einfluss auf das Ausfallverhalten hat. Die Optimierung des Wartungsintervalls hängt somit größtenteils vom Ausfall der Antriebsstrangteile ab.

Insgesamt sind die Ergebnisse der Algorithmen relativ ähnlich. In der vierten Garantiekategorie der Antriebsstrangteile scheinen praktisch keine Kosten zu entstehen. Dies ist auch das kleinste Intervall. Die zusätzlichen Kosten der Garantieleistung scheinen im Vergleich zur positiven Bewertung klein zu sein. Die Bewertung der Garantien beziehungsweise der Ausfälle hat hier einen großen Einfluss auf das Ergebnis. Hier wären weitere Versuche interessant, die zum Beispiel die positive Bewertung der Garantieleistung höher ansetzen oder aber auch die exponentialverteilten Normalteile nicht berücksichtigen.

Der letzte Eintrag in der Tabelle analysiert das Modell in einer Variante, die für die Teile des Antriebsstrangs sämtliche Kosten übernimmt. Die Mehrkosten sind relativ klein. Das lässt wiederum darauf schließen, dass das Wartungsintervall für die Teile des Antriebsstrangs nahezu ideal ist. Eine Erhöhung der Garantie für die Normalteile ist allerdings mit erheblichen Mehrkosten verbunden.

ALGORITHMUS	FITNESS	ERGEBNIS (WARTUNGSINTERVALL, GARANTIE NORMAL, GARANTIE PT)	LAUFZEIT	AUSWERTUNGEN	% DES SUCHRAUMS
Voreinstellung	4 792	100 000, [1.0,0.5,0.4,0.2,0.0], [1.0,0.8,0.6,0.4,0.0]	20 s	1	5×10^{-9}
GA	4 421	199 000, [1.0,0.0,0.0,0.0,0.0], [1.0,0.0,0.05,0.75,0.0]	ca. $8\frac{1}{2}$ h	1 623	8×10^{-6}
PSA	4 413	188 000, [1.0,0.0,0.0,0.0,0.0], [1.0,0.0,0.0,0.4,0.0]	ca. 7 h	1 355	7×10^{-6}
LA (SPSA)	4 414	194 000, [1.0,0.0,0.05,0.0,0.0], [1.0,0.0,0.0,0.85,0.0]	ca. 45 m	151	8×10^{-7}
LA (FDSA)	4 421	191 000, [1.0,0.0,0.0,0.1,0.0], [1.0,0.0,0.3,0.7,0.0]	ca. 50 m	176	9×10^{-7}
Vom Autor gewählt	4 492	188 000, [1.0,0.0,0.0,0.0,0.0], [1.0,1.0,1.0,1.0,0.0]	20 s	1	5×10^{-9}

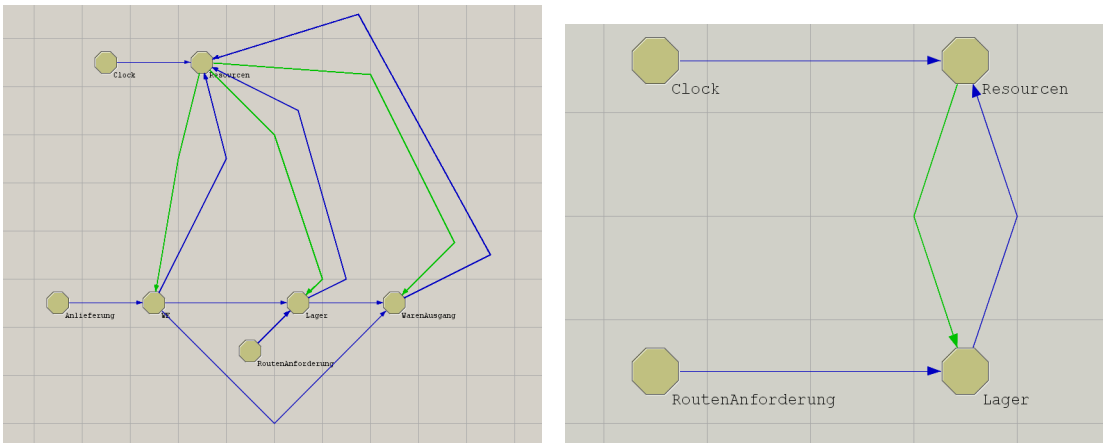
Tabelle 5.3: Ergebnisse des Life-Cycle-Modell

5.3 Logistik-Modell

5.3.1 Modellbeschreibung und Parametrisierung

Das dritte Modell ist die Modellierung eines Logistikcenters der DaimlerChrysler AG. Die modellierte Zeitdauer beträgt eine Woche. In dieser Zeit werden mehr als 20 000 Teile (in 7 verschiedenen Größen) angeliefert und mehr als 100 000 Teile (von 19 verschiedenen Orten) angefordert beziehungsweise kommissioniert. Der Ansatzpunkt für die Optimierung ist das modellierte Arbeitszeitmodell. Arbeitskräfte sind hier in verschiedenen Pools zusammengefasst, die jeweils für ein Lager beziehungsweise einen Wareneingang und -ausgang zuständig sind. Das resultierende, sehr umfangreiche Petri-Netz hat mehr als 5 000 Objekte, eine Replikation dauert (auf einem Pentium 4 mit 1GB RAM) circa 2 Stunden.

Da eine Replikation des ursprünglichen Modells soviel Zeit erfordert, wurde hier nur ein Ausschnitt und dieser auch nur für einen „Simulationstag“ optimiert. Die Optimierung hätte sonst zu viel Zeit in Anspruch genommen. Es werden nur die Anforderungen und Kommissionierungen für zwei Teilearten betrachtet. Jede Art von Teilen hat, wie das ursprüngliche Modell, einen eigenen Ressourcenpool.



(a) Das Original, ein Logistik-Modell (b) Das vereinfachte Logistik-Modell

Abbildung 5.3: Petri-Netz eines Logistik-Centers, in einer einfachen und der originalen Ausführung

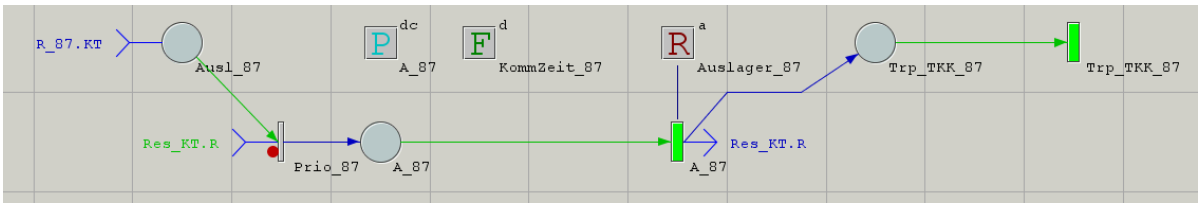


Abbildung 5.4: Ablauf der Kommissionierung eines Teiles

Der Ablauf der Kommissionierung ist in Abbildung 5.4 zu sehen. Für die Kommissionierung eines Teils wird eine Arbeitskraft gebraucht. Diese wird am Anfang aus dem entsprechenden Resourcenpool abgerufen und ist erst nach Abschluss der Kommissionierung wieder verfügbar. Die Zeitdauer der Kommissionierung beträgt für mittelsperrige Teile 4 Minuten und für kleine Teile 1 Minute. Aufträge, die nicht sofort bearbeitet werden können, da keine Arbeitskräfte verfügbar sind, warten in einem Puffer.

Für dieses Modell soll nun die ideale Anzahl an Arbeitskräften gefunden werden. Dabei sollen möglichst wenige Teile in einem Puffer zwischengespeichert werden und alle Teile am Ende eines Tages abgearbeitet sein.

Die **veränderbaren Variablen** sind die Anzahl der Arbeitskräfte pro Schicht. Es gibt zwei Resourcenpools (Kleinteile (AK_{KT}) und mittelsperrige Teile (AK_{MS})) und jeweils vier Schichten (Frühschicht, Teilzeitfrühschicht, Spätschicht und Teilzeitspätschicht). Die Arbeitszeiten für die Schichten sind fest vorgegeben. Es gibt also insgesamt 8 änderbare Variable. Es können bis zu 20 Arbeitskräfte in einer Schicht eingesetzt werden. Daraus ergeben sich bis zu $21^8 = 37\,822\,859\,361$ verschiedene Kombinationen.

$$ZF(AK_{KT}, AK_{MS}) = 500 * \Sigma_{AK} + 1000 P_{max} + 100000 T_{LG} \tag{5.3}$$

Für die **Zielfunktion** werden die Kosten pro Arbeiter mit 500 Geldeinheiten angesetzt

(für Arbeiter in Teilzeit mit 250). Die Kosten pro Pufferplatz betragen 1 000, liegengebliebene Teile werden mit 100 000 Geldeinheiten bewertet. Aus dem Ergebnis der Modellauswertung werden die maximale Puffergröße (P_{max}) und die liegen gebliebenen Teile (T_{LG}) ausgelesen. Das Modell liefert die Summe der Arbeitskräfte ($\Sigma_{AK} = \Sigma_{AKVollzeit} + 1/2 * \Sigma_{AKTeilzeit}$). Daraus ergibt sich die Zielfunktion 5.3.

Das Logistikmodell soll für die Modellauswertungen mit 500 Replikationen simuliert werden. Der sich daraus ergebende Mittelwert μ für eine Stichprobe von 10 Modellauswertungen beträgt 1707660.8, die Standardabweichung σ 8975.4.

5.3.2 Ergebnisse

ALGORITHMUS	FITNESS	ERGEBNIS(KT[FR,TEFR,SP,TESP], MS[FR,TEFR,SP,TESP])	LAUFZEIT	AUSWERTUNGEN	% DES SUCHRAUMS
Voreinstellung	1 707 661	[8,9,6,8], [4,0,4,0]	$3m$	1	2.64×10^{-11}
GA	1 678 252	[15,11,15,13], [16,11,9,3]	$54\ 1/2$ h	789	2.09×10^{-8}
PSA	1 656 199	[15,11,19,7], [11,7,16,3]	$29\ 1/2$ h	423	1.12×10^{-8}
LA (SPSA)	1 703 882	[6,20,6,15], [4,3,13,9]	$1\ 1/3$ h	21	5.55×10^{-10}
LA (FDSA)	1 700 776	[10,19,3,10], [5,7,12,0]	51 m	18	4.76×10^{-10}

Tabelle 5.4: Ergebnisse des Logistik-Modells

Die Optimierung liefert wiederum ähnliche Ergebnisse für die vier Algorithmen. Allerdings war die Parametrisierung beziehungsweise der Startpunkt der FDSA und SPSA Algorithmen nicht optimal. Der FDSA Algorithmus hat sich gar nicht von seiner Anfangsposition fortbewegt. Das positive Ergebnis liegt daran, dass hier die zufällig ausgewählte Startposition sehr gut war. Dieses Modell ist für die lokalen Algorithmen sicher auch nicht optimal. Weil nur kleine Intervalle für jeden Parameter gegeben sind, kann hier kein guter Gradient ermittelt werden. Trotzdem hat der lokale Algorithmus mit SPSA-Gradientenschätzer von einer relativ schlechten Ausgangsposition (mit einem Fitnesswert größer als 2 Millionen) ein relativ gutes Ergebnis erzielt.

Mit Genetischem und Partikel Schwarm Algorithmus wurden gute Ergebnisse erzielt. Im Gegensatz zu einer kompletten Auswertung des Suchraums ergibt sich eine erhebliche Einsparung.

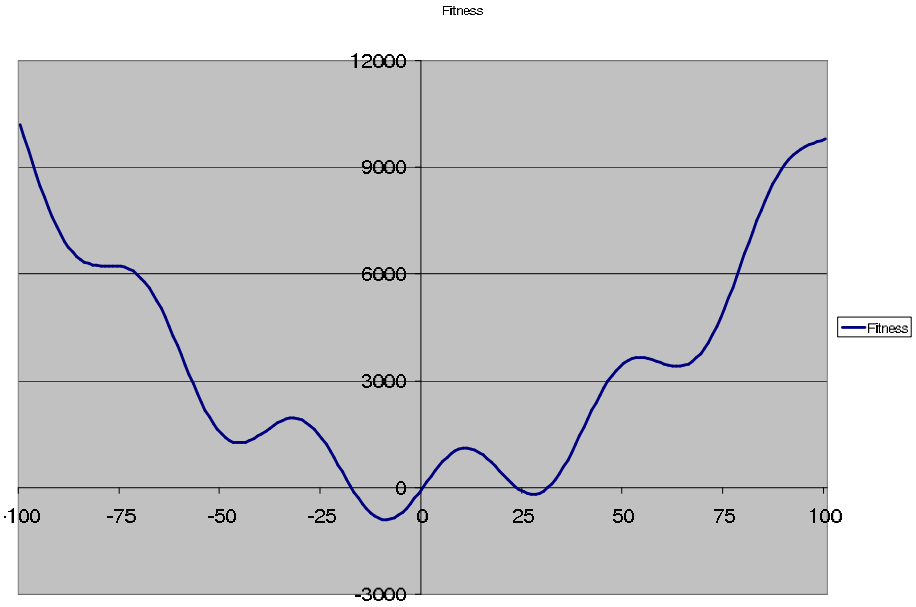


Abbildung 5.5: Verlauf der Gleichung 5.4 von x = -100 bis x = 100

5.4 Deterministisches Modell – Globale und Lokale Optimierung

5.4.1 Modellbeschreibung und Parametrisierung

Um die Unterschiede zwischen globalen und lokalen Algorithmen zu verdeutlichen, soll hier eine mathematische Funktion mit einigen lokalen Optima und einem globalen Optimum optimiert werden. Es wurde ein Dummy-Modell erstellt, das lediglich eine Variable zur Verfügung stellt, die verändert werden kann. Die zu optimierende Funktion wird dann in der Zielfunktion modelliert. Dieses deterministische Modell kann im Gegensatz zu den anderen stochastischen Modellen sehr schnell ausgewertet werden. Das liegt daran, dass **eine** Auswertung des Modells für eine exakte Bestimmung des Zielfunktionswerts ausreicht.

Der Verlauf der Funktion 5.4 ist in Abbildung 5.5 zu sehen. Das globale Minimum der Funktion liegt bei $x \approx -9.15$. Weitere Optima liegen zum Beispiel bei $x \approx 27$ und $x \approx -45$.

$$1000 \sin(\frac{x}{2\pi}) + x^2 \tag{5.4}$$

Die Versuche werden jeweils 10 mal mit Genetischem, Partikel Schwarm und lokalem Algorithmus auf SPSA-Basis durchgeführt[1].

[1]Da bei einem Modell mit einem Parameter zwischen SPSA- und FDSA-Verfahren kein Unterschied besteht wurde hier nur der SPSA-Algorithmus ausgeführt.

5.4.2 Ergebnisse

ALGORITHMUS	FITNESS	ERGEBNISSE UM OPTIMUM	OPTIMIERUNGSLÄUFE
PSA(Durchschnitt)	-836	9	10
GA(Durchschnitt)	-700	8	10
LA(Durchschnitt)	-2	2	10
LA(-81)	1255	-	1
LA(-30)	-909	-	1

Tabelle 5.5: Ergebnisse der Optimierung des deterministischen Modells

In Tabelle 5.5 sind die Ergebnisse der Optimierungsläufe dargestellt. Da die Optimierungsläufe praktisch keine Zeit verbrauchten, wurden für jeden Algorithmus 10 Optimierungsläufe durchgeführt. Die Startpopulationen oder -positionen wurden zufällig gewählt. Zusätzlich wurde der lokale Algorithmus mit zwei bewusst gewählten Startpositionen (-81 und -31) gestartet. Die günstige Startposition (-31) führte dazu, dass der lokale Algorithmus das globale Optimum gefunden hat. Mit der schlechten Startposition (-81) wurde aber nur das lokale Optimum x = -45 gefunden.

Dieses Verhalten zeigte sich auch in den anderen Versuchsläufen. Während der Partikel Schwarm und der Genetische Algorithmus in 9 beziehungsweise 8 von 10 Auswertungen das globale Optimum fanden, war dies bei dem lokalen Algorithmus nur zweimal der Fall. Bei den globalen Algorithmen wurde teilweise nur ein **Punkt** in der Nähe des globalen Minimums gefunden, während der lokale Algorithmus hier immer die Optima punktgenau fand. Hier kann eine Metaheuristik mit einer Kombination aus globalem und lokalem Algorithmus ansetzen.

5.5 Bewertung

Da die Optimierung der verschiedenen Modelle viel Zeit in Anspruch nahm, konnten pro Modell und Algorithmus jeweils nur ein Optimierungslauf durchgeführt werden[2]. Die gewonnenen Ergebnisse sind besonders für den Genetischen und den Partikel Schwarm Algorithmus durchweg positiv, aber auch die lokale Suche konnte gute Parametersätze (und diese insbesondere in kurzer Zeit) finden. Weitere Versuche sind aber unerlässlich, um die Funktionalität genauer beurteilen zu können. Der SPSA- beziehungsweise FDSA-basierte Algorithmus konnte auf Grund der durchweg diskreten und dabei kleinen Parameterdimensionen seine Wirkung nicht voll entfalten. Speziell in einer hybriden Variante aus einem globalen Algorithmus mit anschließender lokaler Optimierung würden sie bei Modellen mit einigen kontinuierlichen Parametern gute Ergebnisse liefern. Auch eine simple lokale Suche mit Schrittweite 1, die den erstbesten gefundenen Wert als neue Position akzeptiert, ist für Parameter mit wenigen Ausprägungen als Verbesserungsmöglichkeit der globalen Ergebnisse denkbar.

[2] Das deterministische Modell ausgenommen.

Die beiden globalen Algorithmen haben sich robust gezeigt und konnten auch mit Standardparametereinstellungen gute Ergebnisse liefern. Die Ergebnisse der lokalen Algorithmen waren von der Wahl der Algorithmenparameter stark abhängig. Eine adaptive Regelung der Algorithmenparameter würde hier sicherlich große Vorteile gegenüber der derzeitigen statischen Parametereinstellung bringen.

Kapitel 6

Bewertung und Ausblick

6.1 Zusammenfassung

In dieser Arbeit wurde eine Optimierungsschnittstelle für stochastische Modelle entwickelt. In Kapitel 2 und 3 wurden die Grundlagen für die Konzeption einer Optimierungsschnittstelle dargelegt. Schwerpunkt bildete hier die Identifikation von Ablauf und Bestandteilen der Optimierung. Es wurde ein Überblick über das Gebiet der Optimierungsalgorithmen gegeben. Zudem wurde im 3. Kapitel die Auswahl der zu implementierenden Algorithmen beschrieben.

In Kapitel 4 wurde die implementierte Optimierungsschnittstelle sowie deren Anbindung an das Tool Expect erläutert. Vorgestellt wurde die Schnittstellen, die ermöglichen, dass Bestandteile der Optimierung abstrakt in die Schnittstelle eingebaut und dann verwendet werden können.

Im fünften Kapitel wurden die implementierten Algorithmen an drei verschiedenen stochastischen Modellen getestet. Diese sind ein einfaches Petri-Netz-Modell, sowie ein Life-Cycle- und ein Logistik-Modell der DaimlerChrysler AG. Zusätzlich wurde der Unterschied zwischen globalen und lokalen Algorithmen anhand eines deterministischen Modells hevorgehoben. Insbesondere mit den beiden globalen Algorithmen konnten gute Resultate erzielt werden.

6.2 Bewertung

Die in Kapitel 1.2 und 1.3 beschriebenen Aufgabenstellungen und Ziele wurden erfüllt. Eine abstrakte Optimierungsschnittstelle wurde geschaffen, die erweiterbar ist und insbesondere mit austauschbaren stochastischen Modellen und Optimierungsalgorithmen gute Optimierungsergebnisse liefert.

Die Erweiterbarkeit ist gerade durch die Schnittstellen in der Optimierung gut gewährleistet. Praktisch jeder Teil der Optimierung ist ohne Anpassung anderer Teile austauschbar. Durch Anbinden der Skriptingschnittstelle sind auch Modell, Analysator und Modellvariable abstrakt eingebunden und damit austauschbar.

Optimale Ergebnisse

Die Optimierungsschnittstelle erzielt gute Optimierungsergebnisse. Globale Optima werden durch Heuristiken nicht garantiert, sie sind aber sonst ohne unermesslichen Aufwand nicht zu finden. Hierbei liefern gerade der Partikel Schwarm und der Genetische Algorithmus gute Ergebnisse.

Mit allen Modellen

Alle Modelle des Skriptings können optimiert werden. Hier ist eventuell nur die Anzahl der Algorithmen eingeschränkt, die effizient optimale Ergebnisse finden. Die Auswahl eines passenden Algorithmus ist in diesem Zusammenhang wichtig.

Möglichst schnell

Besonders der Partikel Schwarm Algorithmus und die lokale Suche mit dem SPSA Gradientenschätzer liefern schnell gute Ergebnisse. Dies trifft insbesondere auf Modelle zu, bei denen eine lange Analysezeit vorliegt. Da beide mit relativ wenigen Modellauswertungen gute Ergebnisse erzielen, sind sie hierfür gut geeignet. Die Qualität der Ergebnisse lässt sich durch eine Kombination dieser Algorithmen, also zum Beispiel durch ein AMP-Framework oder in einer einfachen Hybridisierung sicherlich noch verbessern.

Einfach

Der Benutzer wird durch einige Hilfestellungen, wie Voreinstellungen und Auswahl der Zielfunktionsvariablen per Dialog, unterstützt. Für Partikel Schwarm und Genetischen Algorithmus sind auch mit den Voreinstellungen gute Ergebnisse erzielbar, die aber eventuell (beim Genetischen Algorithmus) eine lange Laufzeit beanspruchen. Insbesondere bei den Parametereinstellungen bleiben einige Fragen offen, die zum Beispiel durch Heuristiken zum Anpassen von Algorithmenparametern lösbar sind.

Generell stellt sich die Frage, wann eine Optimierung mit einem stochastischen Modell sinnvoll ist. Sind andere exaktere oder schnellere Verfahren verfügbar, die Modelle analysieren und optimieren können, ist diese Art der Optimierung sicherlich vorzuziehen. Die meisten stochastischen Modelle beschreiben aber Systeme, die sich nicht ohne weiteres mit mathematischen Formeln beschreiben lassen oder bei denen es keine andere Möglichkeit der Beschreibung gibt. Ist sehr schnell ein Ergebnis gefragt oder ein Modell zu groß, um alle denkbaren Parameterausprägungen auszuwerten, so liefert die Optimierung eine gute Möglichkeit Parameterwerte zu ermitteln. Bei kleinen Modellen ist eventuell eine vollständige Auswertung vorzuziehen.

Auch die vollständige Auswertung und vor allem Bewertung aller Parameterausprägungen eines Modells ist durch die Optimierungsschnittstelle deutlich erleichtert beziehungsweise erst ermöglicht worden. Zusammenhänge in Modellen lassen sich so einfacher analysieren und

darstellen. Als einfaches Beispiel sei hier die Eigenschaft der Gedächtnislosigkeit von Exponentialverteilungen genannt, sich mit einem einfachen Wartungsmodell demonstrieren lässt.

Die Implementation der Optimierungsschnittstelle mit Expect und der darin bestehenden Scriptingschnittstelle hatte den großen Vorteil, dass hier die Auswertung der für die Optimierung benötigten Daten leicht umzusetzen war. Hier mussten keine Schnittstellen geschaffen werden, wie es bei anderen Programmen zur Auswertung nötig wäre. Die Integration in das Tool Expect gewährleistet auch eine Datendurchgängigkeit, die bei einer getrennten Implementation von Tool und Optimierung sicherlich nicht möglich wäre. Allerdings besteht so eine hohe Abhängigkeit von Expect. Expect ist ideal für die Optimierung, weil die Anzahl der zur Verfügung stehenden Modelle und Analysearten groß ist und ständig wächst. Über die Schnittstelle zu DaVincy stehen auch umfangreiche Datenbestände zur Nutzung zur Verfügung.

6.3 Ausblick

Für die Erweiterung der Optimierung gibt es sehr viele interessante und vielfältige Möglichkeiten, deren Zahl in Zukunft noch wachsen wird. Die Möglichkeiten der Erweiterung wurden der Übersichtlichkeit halber in einige Kategorien aufgeteilt.

- **Nutzerfreundlichkeit**
  - Die Zielfunktion könnte als graphischer Editor implementiert werden. Hier könnten einfache mathematische Operationen, sowie die aus Ergebnissen, Modell und Analysator ausgewählten Variablen durch Drag und Drop zu einer Zielfunktion kombiniert werden. Dies würde Nutzern ohne Java-Erfahrung die Definition von Zielfunktionen erleichtern.
  - Die Plausibilität der eingegebenen Daten könnte überprüft werden. Dadurch können Fehleingaben früh erkannt und somit nutzlose Optimierungsläufe vermieden werden. Die Überprüfung von Variablen ist bisher nur rudimentär implementiert.
  - Durch Einführung adaptiver Algorithmenparameter könnte die Parametrisierung verschiedener Algorithmen vereinfacht werden.
  - Die Ausgabe der Ergebnisse könnte über die Visualisierung der Ergebnisschnittstelle des Skriptings realisiert werden. Dann können sofort nach, beziehungsweise während der Optimierung Graphiken zum Verlauf der Optimierung ausgegeben werden, ohne den Umweg über MS Excel zu gehen.
- **Implementation weiterer Algorithmen und Metaheuristiken**
  - Durch Implementation weiterer Algorithmen könnten neue Verfahren eingeführt werden, die für verschiedene Modellarten bessere Ergebnisse liefern als die bereits umgesetzten.

  - Durch Umsetzung eines AMP-Algorithmus oder anderer Metaheuristiken könnten mehrere Algorithmen kombiniert und damit Stärken ausgenutzt, sowie Schwächen der Algorithmen beseitigt werden.

- **Erweiterung der Skripting- und Optimierungsschnittstelle**

  - Durch Definition von Abhängigkeiten zwischen verschiedenen Optimierungsvariablen könnten ungültige Parametersätze schon während der Erzeugung erkannt und durch gültige ersetzt werden. Jetzt ist diese Modellierung nur über eine schlechte Bewertung der Zielfunktion zu erreichen.
  - Im Skripting könnten weitere Modellarten und Möglichkeiten der Analyse implementiert werden. Dadurch steigt die Zahl der optimierbaren Modelle.
  - Durch andere Analyseverfahren könnten einige Modelle schneller ausgewertet und damit auch schneller oder genauer optimiert werden.

6.4 Schlusswort

Durch die Optimierung stochastischer Modelle ist es möglich Informationen über komplexe Systeme zu gewinnen, die mit herkömmlichen Methoden nicht zu finden sind. Es können leicht Zusammenhänge verdeutlicht und gezielt optimale Parametersätze für Systeme gewonnen werden, die ohne den Einsatz der Optimierung nur unter größtem zeitlichen und finanziellen Aufwand möglich sind. Ist ein Modell erstellt, gestaltet sich die Optimierung relativ einfach.

Dabei ist die Implementation einer Optimierungsschnittstelle in das Tool Expect ein wesentlicher Schritt zu der oben genannten Informationsgewinnung. Die Möglichkeiten der Optimierung stehen sicherlich noch am Anfang. Durch die Weiterentwicklung von Hardware können Modelle immer schneller ausgewertet werden und bieten so Platz für neue Algorithmen, die optimale Parametersätze zielgenauer finden. Auch die automatische Selektion von Algorithmenparametern oder sogar des optimierenden Algorithmus sind durch Mehraufwand machbar und lassen noch besseren Ergebnisse erwarten.

Das Tool Expect eignet sich auf Grund seiner vielfältigen Möglichkeiten und insbesondere durch die geplanten Erweiterungen sehr gut für die Optimierung stochastischer Modelle.

Anhang A

Screenshots

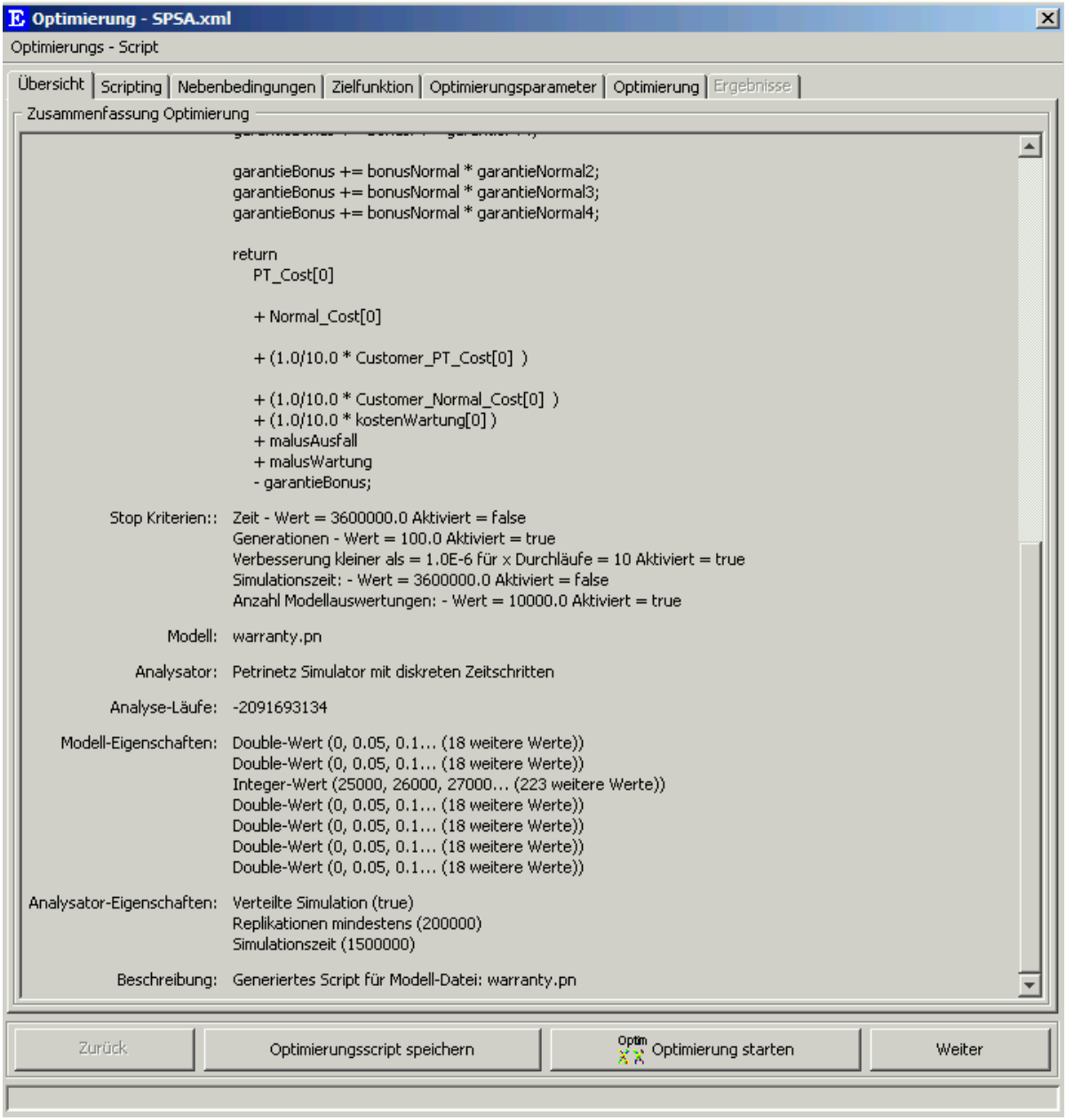


Abbildung A.1: Screenshot Register Übersicht

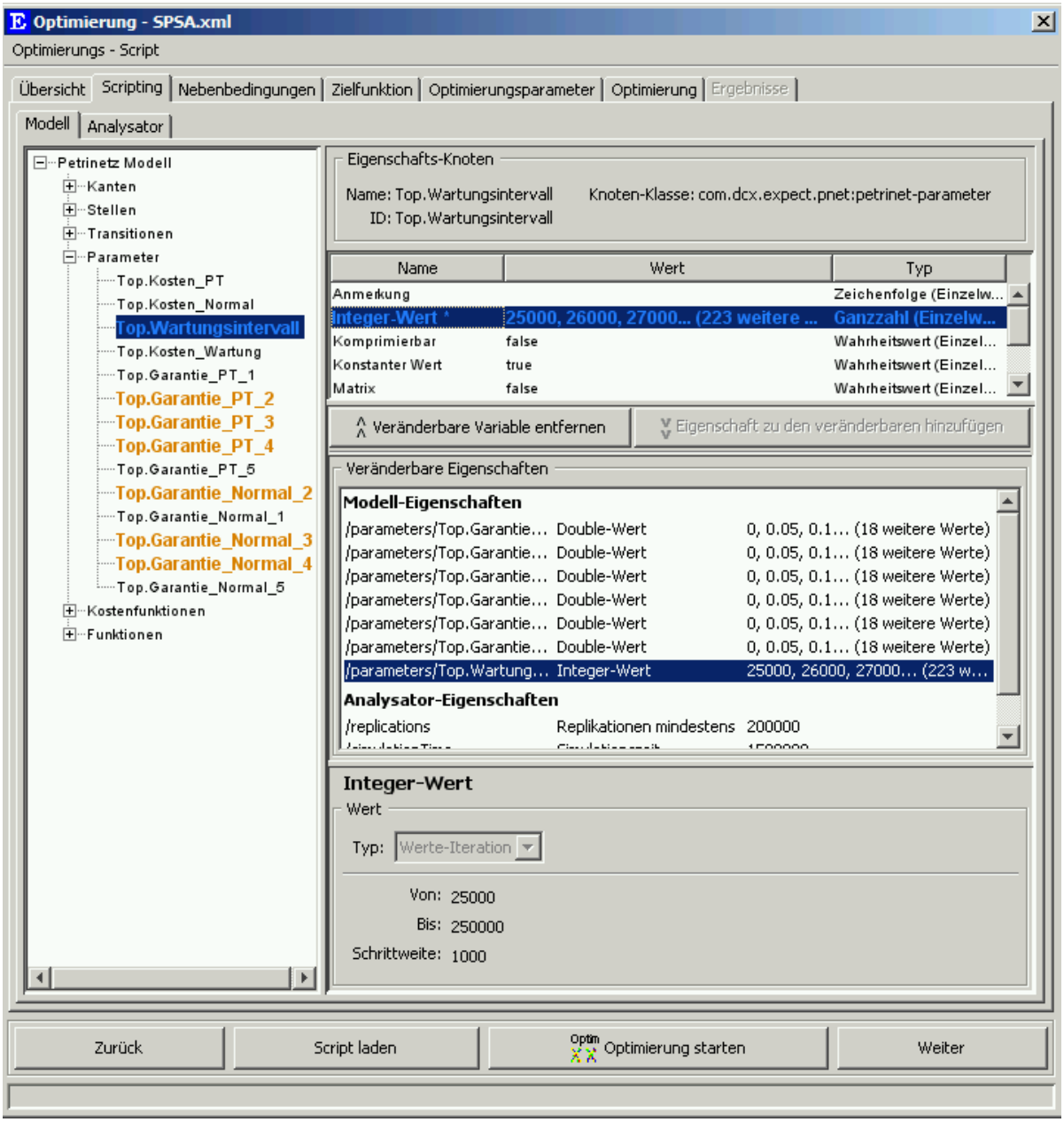


Abbildung A.2: Screenshot Register Scripting

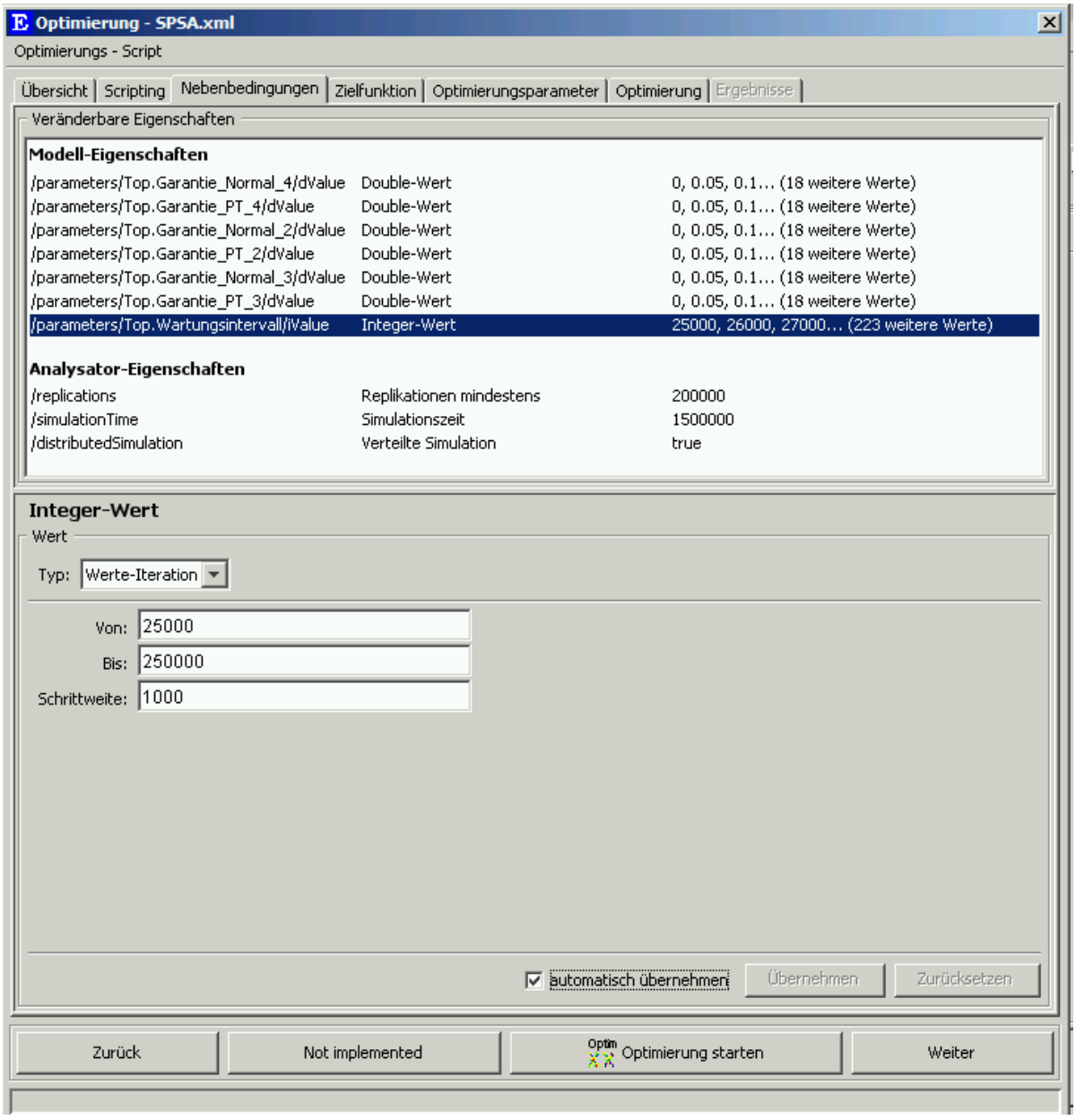


Abbildung A.3: Screenshot Register Nebenbedingungen

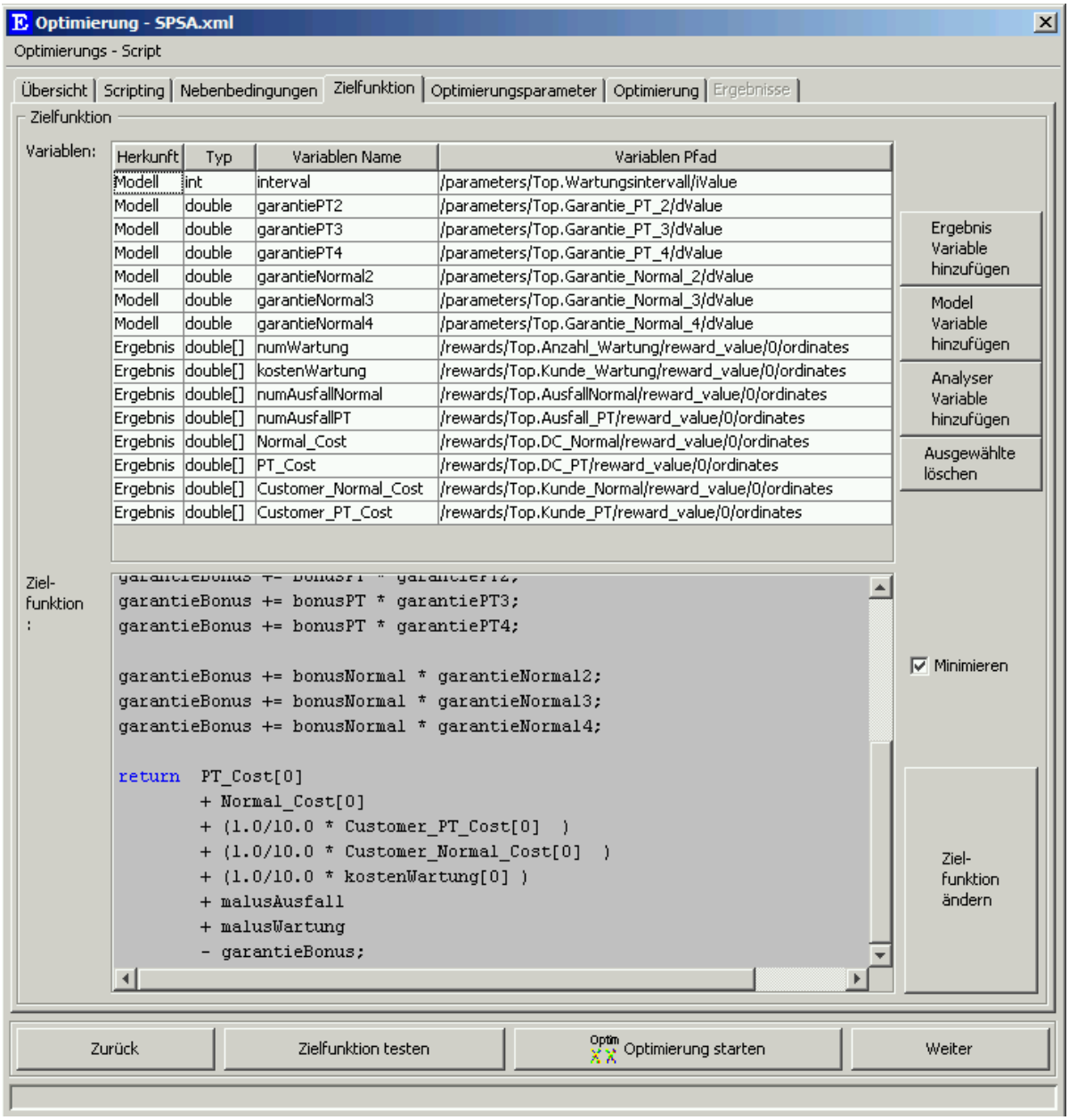


Abbildung A.4: Screenshot Register Zielfunktion

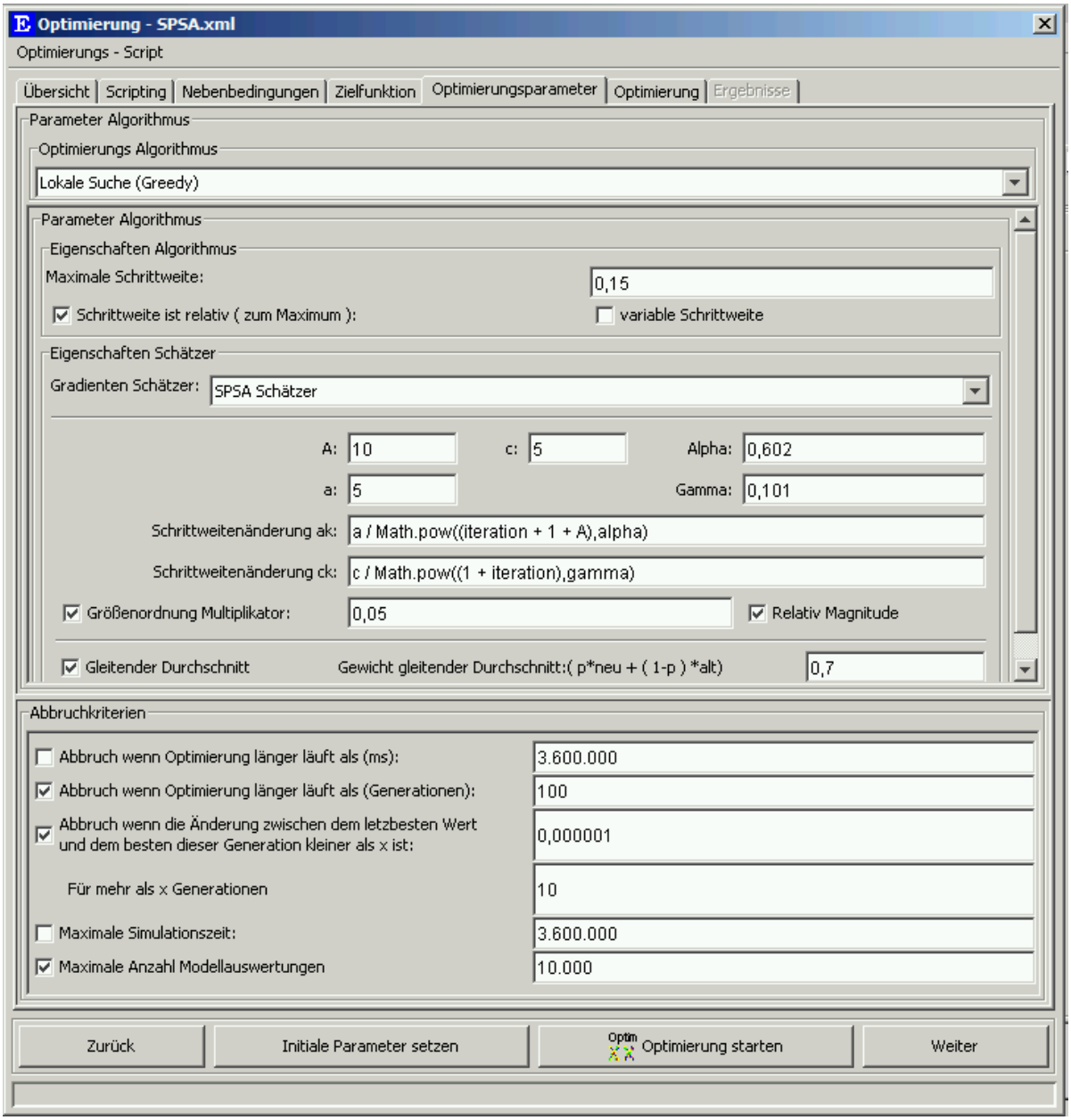


Abbildung A.5: Screenshot Register Optimierungsparameter

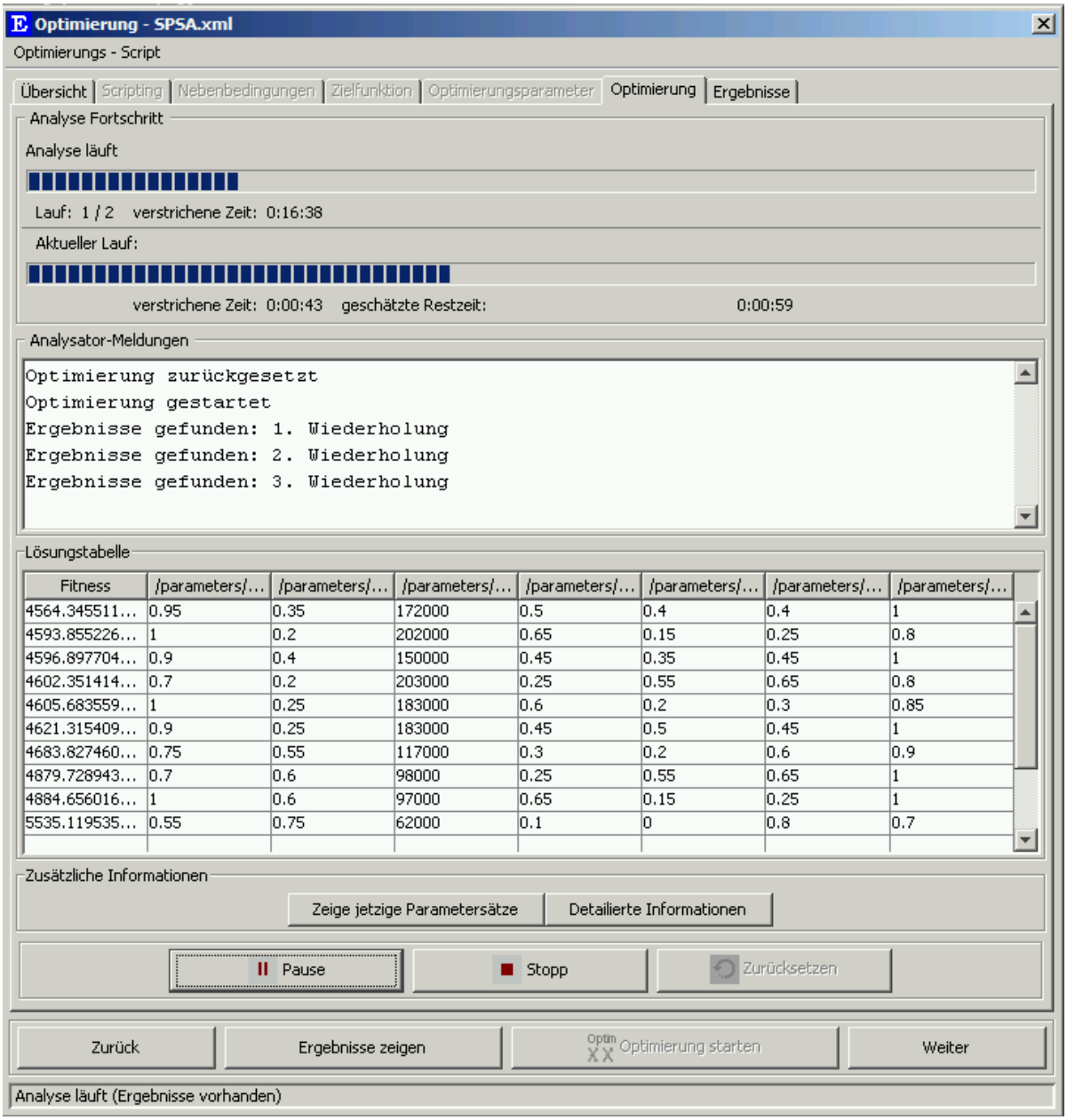


Abbildung A.6: Screenshot Register Optimierung

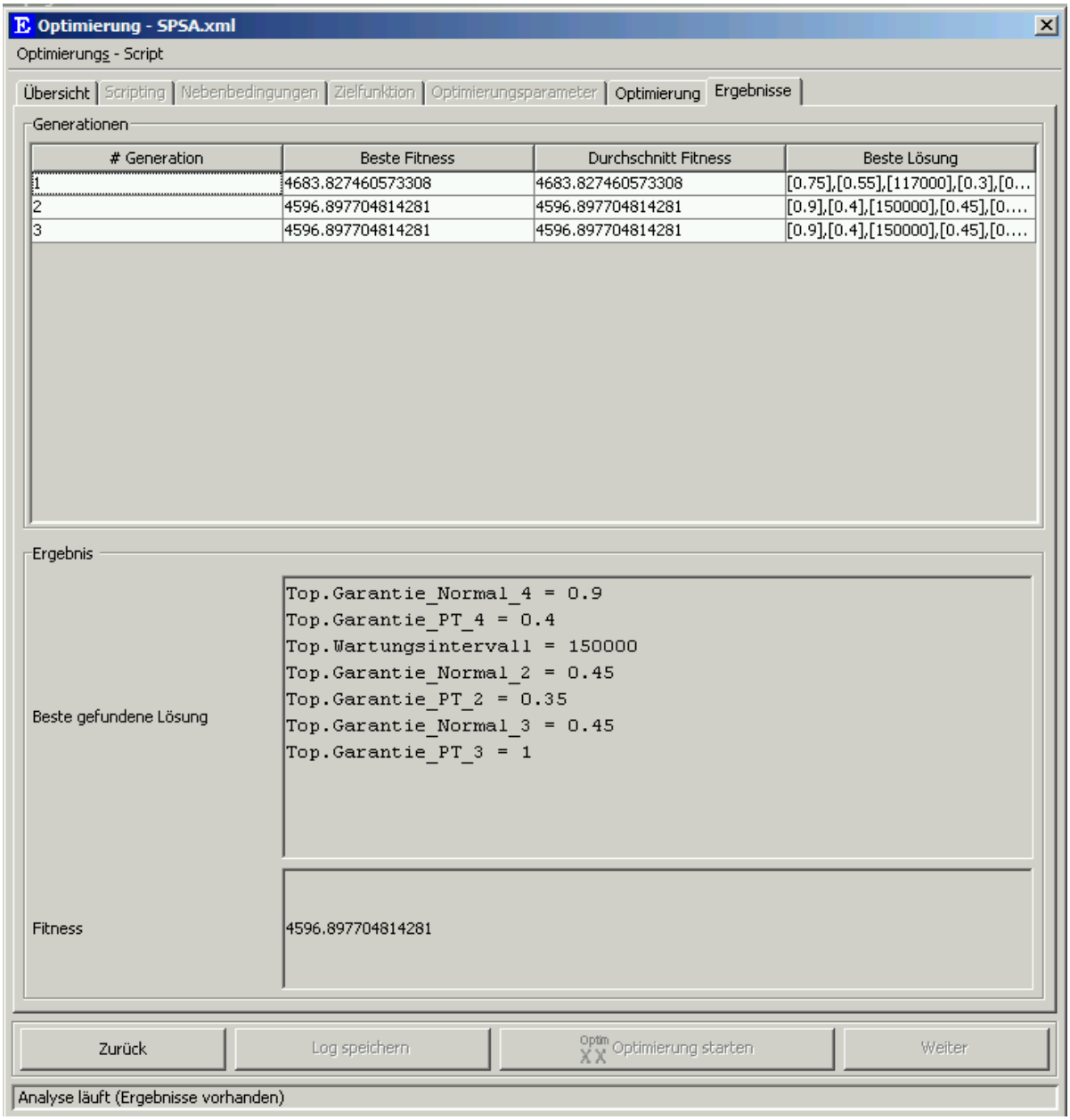


Abbildung A.7: Screenshot Register Ergebnisse

Anhang B

Algorithmenparameter der Algorithmen

Modell 1

Anzahl der Replikationen pro Simulationslauf: 50 000

- **Partikel Schwarm Algorithmus**
  Größe der Population: 20
  Faktor Gruppe: 2
  Faktor Partikel: 2
  Maximale Schrittweite: 10% der Reichweite
- **Genetischer Algorithmus**
  Größe der Population: 35
  Selektion der Eltern: 100% Rangselektion
  Bias der Selektion:1.5
  Erzeugung der Chromosomen: 50% Kreuzung 1-Kind, 50% Uniform-Kreuzung
  Mutationswahrscheinlichkeit = 0.1%
- **Lokale Suche (FDSA & SPSA)**
  A: 10
  a: 5
  alpha: 0.602
  c: 5
  gamma:0.101
  Gain-Funktion ak: a / Math.pow((iteration + 1 + A),alpha)
  Gain-Funktion ck: a / Math.pow((iteration + 1 + A),alpha)
  Relative Magnitudenanpassung = 0.05

Life-Cycle-Modell

Anzahl der Replikationen pro Simulationslauf: 200 000

- **Partikel Schwarm Algorithmus**

  Größe der Population: 25

  Faktor Gruppe: 2

  Faktor Partikel: 2

  Maximale Schrittweite: 10% der Reichweite

- **Genetischer Algorithmus**

  Größe der Population: 70

  Selektion der Eltern: 100% Rangselektion

  Bias der Selektion:1.5

  Erzeugung der Chromosomen: 50% Kreuzung 1-Kind, 50% Uniform-Kreuzung

  Mutationswahrscheinlichkeit = 0,1%

- **Lokale Suche (FDSA & SPSA)**

  A: 10

  a: 5

  alpha: 0.602

  c: 5

  gamma:0.101

  Gain-Funktion ak: a / Math.pow((iteration + 1 + A),alpha)

  Gain-Funktion ck: a / Math.pow((iteration + 1 + A),alpha)

  Relative Magnitudenanpassung: 0.05

  Maximale relative Schrittweite: 0.15

Logistik-Modell

Anzahl der Replikationen pro Simulationslauf: 500

- **Partikel Schwarm Algorithmus**

  Größe der Population: 25

  Faktor Gruppe: 2

  Faktor Partikel: 2

  Maximale Schrittweite: 10% der Reichweite

- **Genetischer Algorithmus**

  Größe der Population: 100

  Selektion der Eltern: 100% Rangselektion

  Bias der Selektion:1.5

  Erzeugung der Chromosomen: 50% Kreuzung 1-Kind, 50% Uniform-Kreuzung

  Mutationswahrscheinlichkeit = 0,1%

- **Lokale Suche (FDSA & SPSA)**

  A: 2

  a: 5

  alpha: 0.602

  c: 5

  gamma:0.101

  Gain-Funktion ak: a / Math.pow((iteration + 1 + A),alpha)

  Gain-Funktion ck: a / Math.pow((iteration + 1 + A),alpha)

  Relative Magnitudenanpassung: 0.05

  Maximale relative Schrittweite: 0.15

Deterministisches Modell

Anzahl der Replikationen pro Simulationslauf: 1

- **Partikel Schwarm Algorithmus**

  Größe der Population: 5

  Faktor Gruppe: 2

  Faktor Partikel: 2

  Maximale Schrittweite: 10% der Reichweite

- **Genetischer Algorithmus**

  Größe der Population: 10

  Selektion der Eltern: 100% Rangselektion

  Bias der Selektion:1.5

  Erzeugung der Chromosomen: 50% Kreuzung 1-Kind, 50% Uniform-Kreuzung

  Mutationswahrscheinlichkeit = 1%

- **Lokale Suche (FDSA & SPSA)**

  A: 5

  a: 0.5

  alpha: 0.602

  c: 5

  gamma:0.101

  Gain-Funktion ak: a / Math.pow((iteration + 1 + A),alpha)

  Gain-Funktion ck: a / Math.pow((iteration + 1 + A),alpha)

  Relative Magnitudenanpassung: 0.01

  Maximale relative Schrittweite: 0.1

	Anforderungen pro Viertelstunde			
	Anforderungen Route 1		Anforderungen Route 2	
UHRZEIT	KLEINTEILE	MITTELSPERRIGE	KLEINTEILE	MITTELSPERRIGE
00:00 - 06:00	-	-	-	-
06:00 - 09:00	300	30	1000	150
09:00 - 12:00	700	70	600	300
12:00 - 13:00	50	5	300	100
13:00 - 15:00	200	20	-	-
15:00 - 17:00	700	70	-	-
17:00 - 18:30	100	10	-	-
18:30 - 00:00	-	-	-	-
Kommissionierzeit	1 Min.	4 Min.	1 Min.	4 Min.

Tabelle B.1: Anforderungen der Routen im Logistik-Modell

Literaturverzeichnis

[Ana97] ANANDALINGAM, G.: Simulated Annealing. In: GASS, S. I. (Hrsg.) ; HARRIS, C. M. (Hrsg.): *Encyclopedia of Operations Research and Management Science*. Boston/Dordrecht/London : Kluwer Academic Publishers, 1997, S. 623–625

[Aza99] AZADIVAR, Farhad: Simulation optimization methodologies. (1999), 93–100. http://www.informs-sim.org/wsc99papers/012.pdf. – Online Ressource, Abruf: 14. 9. 2005

[Bro93] *Brockhaus Enzyklopädie 19. Auflage, Mag-Mod*. Mannheim, 1993

[Bro01] *Der Brockhaus Multimedial 2001 – 1 DVD-ROM*. Mannheim, 2001. – Elektronische Ressource

[CM97] CARSON, Yolanda ; MARIA, Anu: Simulation Optimization: Methods and Applications. (1997), 118–126. http://www.informs-sim.org/wsc97papers/0118.PDF. – Online Ressource, Abruf: 27. 9. 2005

[DCC02] DANIEL C. CHIN, John L. M.: *Global Optimization via SPSA*. Version: 2002. http://www.isd.mel.nist.gov/research_areas/research_engineering/Performance_Metrics/PerMIS_2002_Proceedings/Maryak_Chin.pdf. – Online–Ressource, Abruf: 13. 9. 2005

[DCG99] DORIGO, Marco ; CARO, Gianni D. ; GAMBARDELLA, Luca M.: Ant Algorithms for Discrete Optimization. In: *Artificial Life* 5 (1999), Nr. 2, 137–172. http://www.idsia.ch/~luca/ij_23-alife99.pdf

[Fu02] FU, Michael C.: Feature Article: Optimization for simulation: Theory vs. Practice. In: *INFORMS Journal on Computing* 14 (2002), Nr. 3, 192–215. http://www.ece.northwestern.edu/~nocedal/acnw/read/JOCsum02.pdf. – Online Ressource, Abruf: 11. 10. 2005

[GHV99] GERENCSÉR, László ; HILL, Stacy D. ; VÁGÓ, Zsuzsanna: Optimization over discrete sets via SPSA. (1999), 466–470. http://www.informs-sim.org/wsc99papers/065.PDF. – Online Ressource, Abruf: 10. 10. 2005

[GK04] GODEFROID, Patrice ; KHURSHID, Sarfraz: Exploring very large state spaces using genetic algorithms. In: *STTT* 6 (2004), Nr. 2, 117–127. http://sdg.csail.mit.edu/pubs/2002/GA_TACAS02.pdf. – Online Ressource, Abruf: 28. 09. 2005

[GL97] GLOVER, Fred ; LAGUNA, M.: *Tabu Search*. Version: 1997. http://www.dei.unipd.it/~fisch/ricop/tabu_search_glover_laguna.pdf. – Online–Ressource, Abruf: 26. 09. 2005

[Glo89] GLOVER, Fred: Tabu Search–Part I. In: *ORSA Journal on Computing* 1 (1989), Nr. 3, 190–206. http://leeds-faculty.colorado.edu/glover/TS-PartI-ORSA.pdf. – Online Ressource, Abruf: 26. 09. 2005

[Glo90] GLOVER, Fred: Tabu Search: A Tutorial. In: *Interfaces* 20 (1990), Nr. 4, 74–94. http://leeds-faculty.colorado.edu/glover/TS-Interfaces.pdf. – Online Ressource, Abruf: 26. 09. 2005

[Hu] HU, Dr. X.: *PSO Tutorial*. http://www.swarmintelligence.org/tutorials.php. – Online–Ressource, Abruf: 26. 09. 2005

[Ing93] INGBER, L.: Simulated annealing: Practice versus theory. (1993). http://www.ingber.com/asa93_sapvt.pdf. – Online Ressource, Abruf: 10. 10. 2005

[Jak04] JAKOB, Wilfried: *Eine neue Methodik zur Erhöung der Leistungsfähigkeit Evolutionärer Algorithmen durch die Integration lokaler Suchverfahren*. Karlsruhe, Universität Karlsruhe, Diss., März 2004. http://bibliothek.fzk.de/zb/berichte/FZKA6965.pdf. – Online–Ressource, Abruf: 27. 09. 2005

[JW04] JONES, Matthew H. ; WHITE, K. P.: Stochastic Approximation with Simulated Annealing as an Approach to Global Discrete-Event Simulation Optimization. (2004), 500–507. http://www.informs-sim.org/wsc04papers/060.pdf. – Online Ressource, Abruf: 10. 10. 2005

[KE95] KENNEDY, J. ; EBERHART, R. C.: Particle swarm optimization. In: *Proc. of the IEEE Int. Conf. on Neural Networks*. Piscataway, NJ : IEEE Service Center, 1995, S. 1942–1948

[Lø02] LØVBJERG, Morten: *Improving Particle Swarm Optimization by hybridization of stochastic search heuristics and Self-Organized Criticality*, EVALife, Dept. of Computer Science, Aarhus Universitet, Diplomarbeit, 2002. http://www.evalife.dk/publications/ML_thesis_2002.pdf. – Online–Ressource, Abruf: 26. 09. 2005

[Luk00] LUKE, Sean: *Issues in scaling genetic programming :–breeding strategies, tree generation, and code bloat*, research directed by Dept. of Computer Science, Diss., 2000. http://cs.gmu.edu/~sean/papers/thesis2p.pdf. – Online–Ressource, Abruf: 11. 10. 2005

[MC96] MANIKAS, Theodore W. ; CAIN, James T.: *Genetic Algorithms vs. Simulated Annealing: Comparison of Approaches for Solving the Circuit Partitioning Problem*. Version: May 1996. http://www.ee.utulsa.edu/~tmanikas/Pubs/gasa-TR-96-101.pdf. Technical Report. – Online–Ressource, Abruf: 28. 09. 2005

[MGL03] MANIEZZO, Vittorio ; GAMBARDELLA, Luca M. ; LUIGI, F. de ; ONWUBOLU, G. C. (Hrsg.) ; BABU, B. V. (Hrsg.): *Ant Colony Optimization*. Version: 2003. http://www.cs.unibo.it/bison/publications/ACO.pdf. – Online–Ressource, Abruf: 26. 09. 2005

[MK05] MARCIN KADLUCZKA, Peter C. Nelson und Thomas M. T.: *Adaptive Memory Programming: A Parameterized Framework for Meta-search*. Version: 2005. http://www.cs.uic.edu/~mkadlucz/AMPv11.pdf. – Online–Ressource, Abruf: 08. 04. 2005

[Poh00] POHLHEIM, Dr.-Ing. H.: *Evolutionäre Algorithmen [Medienkombination]: Verfahren, Operatoren und Hinweise für die Praxis*. Berlin, Heidelberg : Springer Verlag, 2000. – ISBN 3–540–66413–0

[Ras05] RASHEED, Khaled: *Lecture: Evolutionary Computation and Its Applications*. Version: 2005. http://www.cs.uga.edu/~khaled/ECcourse/EC.pdf. – Online–Ressource, Abruf: 27. 09. 2005

[Rot] ROTHLAUF, Franz: *The Influence of Binary Representations of Integers on the Performance of Selectorecombinative Genetic Algorithms*. http://www.bwl.uni-mannheim.de/wifo1/publications/working_paper_2002_1.pdf. Working Paper 1/2002. – Online–Ressource, Abruf: 12. 10. 2005

[Spa] SPALL, James C. ; J.E. GENTLE, Y. M. (Hrsg.): *Stochastic Approximation*. http://www.quantlet.com/mdstat/scripts/csa/html/node52.html. – Online–Ressource, Abruf: 26. 09. 2005

[Spa92] SPALL, J. C.: Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. In: *IEEE Transactions on Automatic Control* 37 (1992), 332–341. http://www.jhuapl.edu/SPSA/PDF-SPSA/Spall_TAC92.pdf. – Online Ressource, Abruf: 22. 09. 2005

[Spa98] SPALL, J. C.: Implementation of the simultaneous perturbation algorithm for stochastic optimization. In: *IEEE Transactions on Aerospace and Electronic Systems* 34 (1998), 817–823. http://www.jhuapl.edu/SPSA/PDF-SPSA/Spall_Implementation_Of_The_Simultaneous.pdf. – Article, Abruf: 12. 9. 2005

[Spa99] SPALL, James C.: Stochastic Optimization: Stochastic Approximation and Simulated Annealing. (1999), 529-542. http://www.jhuapl.edu/SPSA/PDF-SPSA/Spall_Stochastic_Optimization.PDF. – Online Ressource, Abruf: 22. 09. 2005

[TGGP01] TAILLARD, E. D. ; GAMBARDELLA, Luca M. ; GENDREAU, Michel ; POTVIN, Jean-Yves: Adaptive Memory Programming: A Unified View of Metaheuristics. In: *European Journal of Operational Research* 135 (2001), Nr. 1, 1–16. http://www.idsia.ch/~luca/ejor135_1_1_16.pdf. – Online Ressource, Abruf: 26. 09. 2005

[TJTG04] TSANG, C Sq England E. ; JIN, Nanlin ; TSANG, Prof E. ; GOSLING, Timothy: Population Based Incremental Learning Versus Genetic Algorithms: Iterated Prisoners Dilemma. Version: März 29 2004. http://cswww.essex.ac.uk/technical-reports/2004/csm401.pdf. – Forschungsbericht. – Online–Ressource, Abruf: 27. 09. 2005

[Wik05] WIKIPEDIA: *Simulation — Wikipedia, Die freie Enzyklopädie*. Version: 2005. http://de.wikipedia.org/wiki/Simulation. – Online–Ressource, Abruf: 10. 10. 2005

[YG03] YUAN, Bo ; GALLAGHER, Marcus: Playing in continuous spaces: Some analysis and extension of population-based incremental learning. (2003), 8-12 Dezember, S. 443–450. ISBN 0–7803–7804–0

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit selbstständig, ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Magdeburg, den 30. Oktober 2005 Marcus Müller-Dornieden