Technische Universität Ilmenau
Fakultät Informatik und Automatisierung
Fachgebiet System- und Software-Engineering

Dissertation

Zur Erlangung des akademischen
Grades Doktoringenieur (Dr.-Ing.)

Effiziente simulationsbasierte Optimierung farbiger stochastischer Petri-Netze

vorgelegt von: Dipl.-Inf. Christoph Bodenstein
geboren am: 18.04.1980 in Eisenach

Verantwortlicher Professor: Prof. Dr.-Ing. habil. Armin Zimmermann (TU-Ilmenau)
2. Gutachter: Prof. Dr.-Ing. habil. Andreas Mitschele-Thiel (TU-Ilmenau)
3. Gutachter: Prof. Dr. Katinka Wolter (FU-Berlin)

Tag der Einreichung: 08.01.2018
Tag der wissenschaftlichen Aussprache 05.07.2018

urn:nbn:de:gbv:ilm1-2018000264

Danksagung

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Arbeit unterstützt und motiviert haben.

Zusammenfassung

Modelle erleichtern das Verstehen und Verbesserung technischer Systeme. Dabei werden durch Abstraktion komplexer Systeme nur noch wesentliche Bestandteile von Design und Verhalten nachgebildet, das Modell ist damit deutlich leichter handhabbar und verständlicher als das reale System. Durch Anpassung des Modells bzw. seiner Konfiguration wird eine Optimierung des Systems erleichtert oder überhaupt erst ermöglicht.
Optimierung eines Modells bedeutet dabei, aus der Menge aller Systemkonfigurationen diejenige(n) zu bestimmen, für die sich das Modell - und damit später auch das reale System - hinsichtlich bestimmter Bewertungskriterien bestmöglich verhält.
Aufgrund zufälliger Einflussgrößen wird das Finden einer optimalen Systemkonfiguration auf konventionellem Wege unmöglich oder zumindest unrealistisch schwer. Hier setzt die indirekte Optimierung durch Simulation an. Ein großes Problem ist dabei der enorme Zeitbedarf von Simulationen.
Thema der Arbeit ist die Frage, wie die Effizienz simulationsbasierter Optimierung durch Kombination bekannter und neuer Verfahren erhöht werden kann. Dafür wurde ein neues Verfahren der adaptiven Genauigkeitssteuerung mittels Multiphasen-Optimierung entwickelt.
Für die Beantwortung der Frage wurde zunächst ein Analysewerkzeug erstellt, mit dem die verschiedenen Verfahren zur simulationsbasierten Optimierung untersucht werden können. Um auf bisherige Vorarbeiten und Veröffentlichungen am Fachgebiet aufzubauen, wurde für diese Arbeit das Simulationssystem TimeNET verwendet. Als formales Modell für komplexe, diskrete Systeme kommen farbige, stochastische Petri-Netze (**S**tochastic **C**olored **P**etri **N**ets) zum Einsatz.
Typische Probleme simulationsbasierter Optimierung werden betrachtet. Es werden bekannte Verfahren verglichen und ein neues Verfahren vorgestellt, welches den Simulationszeitbedarf in Betracht zieht und damit auf die Anwendung für simulationsbasierte Optimierung zugeschnitten ist.
Abschließend werden die Verfahren anhand von SCPN-Simulationen und Benchmarkfunktionen bewertet.

Abstract

Models facilitate the understanding and improvement of technical systems.
By abstracting complex systems, only essential components of design and behavior are reproduced, making the model much easier to handle and more understandable than the real system.
By adapting the model or its configuration, an optimization of the system is made easier or even possible. Optimization of a model means to determine from the set of all system configurations the one for which the model - and thus later also the real system - behaves best in terms of certain evaluation criteria.
Due to random factors, finding an optimal system configuration by conventional means, e.g. through (Mixed Integer) Linear Programming often is impossible or at least unrealistic hard. This is where indirect optimization through simulation comes into play. A big challenge is the amount of time required by simulations.
Topic of this thesis is increasing the efficiency of simulation-based optimization by combining well known and new methods. For this purpose, a new method of adaptive accuracy control using multi-phase optimization has been developed and integrated into a prototype software tool.
To build on previous work and publications, the simulation system TimeNET was used for this work. Therefor (**S**tochastic **C**olored **P**etri **N**ets) are used as a formal model for complex, discrete systems.
Typical problems of simulation-based optimization are considered. Known methods are compared and a new method is presented, which takes into account the required simulation time and thus is tailored to simulation-based optimization.
Finally, the presented methods are evaluated using SCPN simulations and benchmark functions.

Inhaltsverzeichnis

1 Einleitung

Um das Verhalten von Systemen verstehen und verbessern zu können, werden seit jeher Modelle eingesetzt. Durch eine Abstraktion komplexer Systeme werden nur noch wesentliche Bestandteile von Design und Verhalten nachgebildet, das Modell ist damit deutlich leichter handhabbar und verständlicher als das reale System. Durch Anpassung des Modells bzw. seiner Konfiguration wird eine Optimierung des Systems erleichtert oder überhaupt erst ermöglicht. Dies ist das wichtigste Ziel des Systementwurfs.
Optimierung eines Modells bedeutet dabei, aus der Menge aller Systemkonfigurationen diejenige(n) zu bestimmen, für die sich das Modell -und damit später auch das reale System- hinsichtlich bestimmter Bewertungskriterien bestmöglich verhält. I.d.R. bedeutet es, das Maximum oder Minimum für ein Leistungsmaß, wie z.B. die Betriebskosten eines Systems, zu finden.
Da realitätsnahe Modelle häufig zufällige Einflussgrößen haben, wird das Finden einer optimalen Systemkonfiguration auf konventionellem Wege, z.B. durch (Mixed Integer) Linear Programming unmöglich oder zumindest unrealistisch schwer.
Aufgrund dessen sowie der stetig steigenden Komplexität der Systeme und ihrer Modelle, ist die indirekte Optimierung mittels Simulation das einzige Mittel, optimale Systeme mit vertretbarem Aufwand zu finden. Dies wird bereits so lange untersucht, wie es Simulationen gibt [47].
Zum Teil werden dabei Techniken der klassischen Optimierung verwendet und an die besonderen Umstände simulationsbasierter Optimierung angepasst.
Ein großes Problem ist dabei der enorme Zeitbedarf von Simulationen. Die Berechnung eines einzelnen Kostenfunktionswertes kann Sekunden bis Minuten oder gar Stunden in Anspruch nehmen. Es wurden im Rahmen dieser Arbeit Verfahren zur effizienteren simulationsbasierten Optimierung entwickelt und analysiert.
Das zentrale Thema der Arbeit ist die Frage, wie die Effizienz simulationsbasierter Optimierung durch Kombination bekannter und neuer Verfahren erhöht werden

kann. Dadurch sollten vergleichbar gute Optimierungsergebnisse in kürzerer Zeit erreicht werden. Dafür wurde ein neues Verfahren der adaptiven Genauigkeitssteuerung mittels Multiphasen-Optimierung entwickelt.
Für die Beantwortung der Frage wurde zunächst ein Analysewerkzeug erstellt, mit dem die verschiedenen Verfahren zur simulationsbasierten Optimierung untersucht werden können.
Um auf bisherige Vorarbeiten und Veröffentlichungen am Fachgebiet aufzubauen, wurde für diese Arbeit das Simulationssystem TimeNET verwendet. Als formales Modell für komplexe, diskrete Systeme kommen farbige, stochastische Petri-Netze (**S**tochastic **C**olored **P**etri **N**ets) zum Einsatz. Deren Spezifika werden in Kapitel 2 und 4 besprochen. Die vorgestellten Verfahren lassen sich jedoch auch auf andere Simulationssysteme übertragen.
In der Arbeit erfolgt nach einem Rückblick auf den Stand der Technik simulationsbasierter Optimierung eine Analyse bekannter Verfahren/Heuristiken hinsichtlich ihrer Tauglichkeit zur Anwendung auf farbige Petri-Netze.
Typische Probleme simulationsbasierter Optimierung, aber auch solche, die ausschließlich für SCPNs gelten, werden betrachtet. Es werden Konfigurationen bekannter Verfahren verglichen und ein neues Verfahren vorgestellt, welches den Simulationszeitbedarf in Betracht zieht, indirekt steuert und damit auf die Anwendung für simulationsbasierte Optimierung zugeschnitten ist. Es stellt eine erste Umsetzung der adaptiven Genauigkeitssteuerung simulationsbasierter Optimierung dar, deren Funktionsweise und Effizienz in vorherigen Publikationen gezeigt wurde [69]. Anschließend werden die Verfahren anhand von SCPN-Simulationen und Benchmarkfunktionen bewertet.

1.1 Aufbau der Arbeit und wissenschaftlicher Beitrag

In der Arbeit werden bekannte und neu entwickelte Verfahren zur effizienten simulationsbasierten Optimierung vorgestellt und verglichen. Dabei wird auf Einschränkungen beim Einsatz dieser Verfahren zur Optimierung von farbigen Petri-Netzen eingegangen. Es werden Möglichkeiten zur effizienten simulationsbasierten Optimierung farbiger Petri-Netze vorgestellt und diese im Vergleich zum Stand der Technik untersucht.
Die Arbeit ist wie folgt aufgebaut: Zunächst werden die Grundlagen von Petri-Netzen erläutert, zum allgemeinen Verständnis der zugrunde liegenden Modelle.
Anschließend werden bekannte Verfahren zur effizienten simulationsbasierten Optimierung erläutert. Dies ist Ausgangspunkt der Forschungen im Rahmen dieser Arbeit.
Die ermittelten Einflussfaktoren auf den Zeitbedarf von SCPNs-Simulationen werden erläutert.
Neue Verfahren zur Effizienzsteigerung simulationsbasierter Optimierung werden vorgestellt. Hervorzuheben ist dabei die mehrphasige Optimierung mit adaptiver Genauigkeitssteuerung, welche im Rahmen eines Forschungsprojektes bereits veröffentlicht wurde [5].
Die Umsetzung bekannter und neuer Verfahren im Rahmen der Arbeit als prototypisches Optimierungswerkzeug wird erläutert. Ebenfalls werden die Gründe für die Notwendigkeit eines neuen Werkzeugs dargelegt.
Ein komplexes SCPN zur Modellierung kontinuierlicher Systeme mit variabler Genauigkeit wird vorgestellt. Es dient nachfolgend bei der Bewertung umgesetzter Optimierungsverfahren.
Bekannte und neue Verfahren werden mit Hilfe des entwickelten Werkzeugs anhand von Benchmark-Funktionen und der Simulation von SCPNs bewertet. Dabei wird insbesondere das Potential des neu entwickelten Verfahrens der adaptiven Genauigkeitssteuerung mittels Multiphasen-Optimierung untersucht.
Die wichtigsten Ergebnisse der Arbeit und Ausgangspunkte für künftige Forschungsprojekte sind die nachfolgend erläuterten Punkte:

Kernprinzipien effizienter simulationsbasierter Optimierung

Bekannte Verfahren für effiziente simulationsbasierte Optimierung und ihre Kernprinzipien werden untersucht und eingeordnet.

Neue Verfahren zur effizienteren Optimierung

Es werden neue Verfahren zur effizienten simulationsbasierten Optimierung –nicht nur von SCPNs– vorgestellt. Schwerpunkt liegt dabei auf der Multiphasen-Optimierung mit adaptiver Genauigkeitssteuerung.

Einflussfaktoren für den Zeitbedarf von SCPN-Simulationen

Einflussfaktoren auf die Laufzeit von SCPN-Simulationen werden diskutiert. Sie wurden im Rahmen der Arbeit experimentell ermittelt. In nachfolgenden Projekten wird dies den Entwurf geeigneter SCPNs unterstützen.

Genauigkeitsvariable Modellierung von kontinuierlichen Systemen mit SCPNs

Es wird anhand eines Beispiel-SCPN gezeigt, wie kontinuierliche Systeme (Energieflüsse) mit Hilfe von farbigen Petri-Netzen genauigkeitsvariabel modelliert werden können. Dieser Ansatz wird in der Arbeit genutzt, um die entwickelten Optimierungsverfahren zu untersuchen.

Analysewerkzeug für Verfahren zur simulationsbasierten Optimierung

Ein prototypisches Optimierungswerkzeug mit Anbindung an das Simulations- und Modellierungstool TimeNET wurde entwickelt und wird im Rahmen dieser Arbeit vorgestellt. Es diente der Durchführung sämtlicher Versuche und steht als freie Software zur Nutzung und Anpassung an andere Simulationswerkzeuge zur Verfügung.

Vergleich bekannter und neuer Verfahren zur effizienten Optimierung

Existierende und neu entwickelte Verfahren zur effizienten simulationsbasierten Optimierung werden ausführlich experimentell untersucht und bewertet, sowohl anhand von Benchmark-Funktionen als auch mit Simulation von SCPNs. Die Ergebnisse geben Hinweise auf die Wahl der bestmöglichen Optimierungsheuristik und ihrer Konfiguration.

Das Ziel der Arbeit wurde insofern erreicht, als dass ein Nachweis der Effizienzsteigerung durch Nutzung adaptiver Genauigkeitssteuerung bei mehrphasiger, simulationsbasierter Optimierung für einige Basisheuristiken erbracht werden konnte. Abhängig vom zugrundeliegenden Modell und der erwarteten Kostenfunktion eignen sich nicht alle untersuchten Verfahren für die Einbettung in ein mehrphasiges Optimierungsverfahren.

2 Stand der Technik, verwendete Technologien

Dieses Kapitel soll einen Überblick über die eingesetzten Modellierungs- und Simulationstechnologien geben. Es wird zunächst auf die Entwicklung farbiger Petri-Netze eingegangen und anschließend das verwendete Simulationswerkzeug sowie bekannte Ansätze zur Optimierung eingegangen.
Petri-Netze eignen sich sehr gut zur Modellierung von nebenläufigen Systemen. Viele reale Systeme lassen sich damit bereits abbilden. Mit ihren Weiterentwicklungen in Form von „Farben“, Hierarchien und Schaltwahrscheinlichkeiten ermöglichen Sie das präzise Modellieren komplexer Systeme.
Petri-Netze wurden Anfang der 1960er Jahre von Carl Adam Petri entwickelt, um nebenläufige Schaltvorgänge darstellen zu können. Ausgehend von endlichen Automaten stellte er dieses Mittel zur Modellierung sowie grundlegende Prinzipien in seiner Dissertation “Kommunikation mit Automaten” 1962 vor [46]. Einige dieser Prinzipien fanden Eingang in diverse Modellierungsmethoden in der Informatik, Biotechnologie und auch für Geschäftsprozessmodellierung.
Ein typisches Beispiel ist in Bild 2.1 zu sehen. Zwei konkurrierende Computersysteme (*PC_1 / PC_2*) teilen sich den Zugang zu einer begrenzten Ressource, in diesem Fall ein Drucker (*Drucker_frei*). Mit wenigen Modellelementen lässt sich das Gesamtsystem in Hinblick auf die Problematik des gegenseitigen Ausschlusses bei der Nutzung des Druckers darstellen.

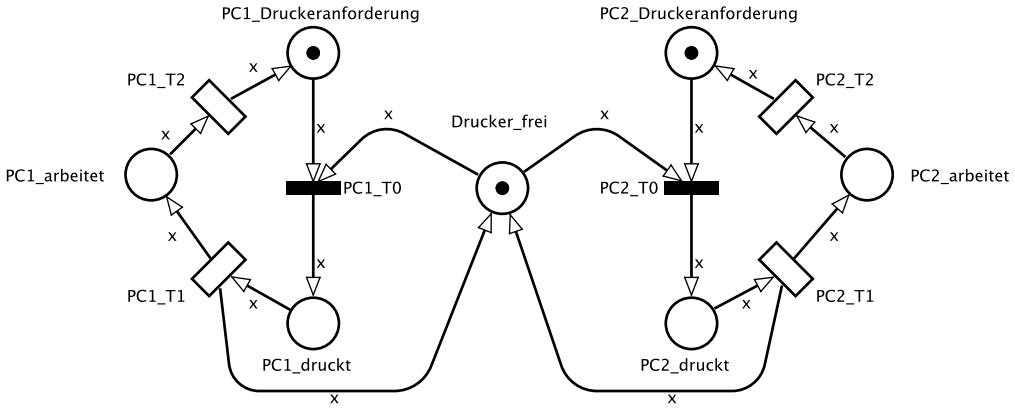


Abbildung 2.1: Gegenseitiger Ausschluss bei der Nutzung eines gemeinsamen Druckers

Im Allgemeinen lassen sich Petri-Netze als ein Quintupel (P, T, F, W, m_0) darstellen. Dabei gilt:

- $P \cap T = \emptyset$ und $P \cup T \neq \emptyset$
- P ist die Menge an Stellen, auch Plätze genannt *(PC1_arbeitet, Drucker_frei, ...)*
- T ist die Menge an Transitionen *(PC1_T2, PC1_T0, ...)*
- $F \subseteq (P \times T) \cup (T \times P)$ ist die Menge der Kanten zwischen Transitionen und Plätzen bzw. Plätzen und Transitionen
- $W : F \to \mathbb{N}$ die Kantengewichtungsfunktion, also wie viele Marken von einer Transition T über eine Kante F konsumiert bzw. erzeugt werden. *(x für alle Kanten des Modells)*
- $m_0 :\to \mathbb{N}_0$ die Anfangsmarkierung *(PC1_Druckeranforderung = 1, PC2_Druckeranforderung = 1)*

Im Laufe der Zeit wurden Petri-Netze von verschiedenen Forschergruppen weiterentwickelt. Die bedeutendsten Errungenschaften sind dabei die Einführung von Farben, Hierarchien sowie ein zeitabhängiges Schaltverhalten der Transitionen.
Bei einfachen farbigen Petri-Netzen kann jeder Marke im Netz eine Eigenschaft aus einer endlichen Menge von Eigenschaften (Farben) zugewiesen werden. Die Transitionen können abhängig von diesen Eigenschaften schalten. Solche Netze lassen sich

in jedem Fall durch Entfaltung wieder in ein einfarbiges Petri-Netz überführen. Farbige Petri-Netze eignen sich damit bereits sehr gut, um komplexe Zusammenhänge mit weniger Modellelementen und damit übersichtlicher darzustellen [30].
Mit Beginn der computergestützten Analyse von Petri-Netzen wurden die erstellten Modelle immer komplexer. Ein logischer Schritt war dabei die Einführung von Hierarchien. Eine Teilmenge zusammenhängender Kanten und Plätze wird dabei in ein Subnetz ausgelagert. Dies erhöht die Übersichtlichkeit, während für die Analyse und später auch die Simulation der Netze diese wieder komplett entfaltet werden. Basierend auf verschiedenen Petri-Netzvariationen und mit Blick auf unterschiedliche Einsatzzwecke wurden solche hierarchischen Ansätze von diversen Autoren beschrieben [10, 19, 11]. Dabei dient die Verwendung von Subnetzen nicht nur der Übersichtlichkeit, sondern kann auch im Sinne der effizienteren Optimierung ausgenutzt werden, wie in späteren Kapiteln erläutert wird.

2.1 Farbige stochastische Petri-Netze (SCPN)

SCPNs können als universelle Klasse von Petri-Netzen betrachtet werden, denn sie vereinen viele wichtige Eigenschaften anderer Petri-Netzklassen –unter anderem der **compound Petri nets** [4]– , die bei der Modellierung von komplexen Systemen sowie deren Optimierung wichtig sind.
Im Rahmen dieser Arbeit wird dabei von der konkreten Implementierung farbiger stochastischer Petri-Netze im bekannten Simulationstool TimeNET ausgegangen. Abweichend davon existieren auch weniger mächtige Umsetzungen dieses Konzeptes. TimeNET, ein Simulationstool für unterschiedliche Arten von Petri-Netzen wurde bereits bei früheren Forschungsprojekten zum Thema simulationsbasierter Optimierung genutzt und befindet sich in stetiger Weiterentwicklung am Fachgebiet SSE. Details zum Einsatz des Werkzeuges und ggf. Einschränkungen finden sich u.A. in „Stochastic Discrete Event Systems - Modeling, Evaluation, Applications“ sowie im Handbuch zur Software [67, 62]. Die nachfolgend beschriebenen Eigenschaften farbiger Petri-Netze beziehen sich auf die Umsetzung in TimeNET. Alternative Modellierungs- und Simulationswerkzeuge haben ggf. andere Einschränkungen in Bezug auf Datentypen und Modellierbarkeit von Systemverhalten.

Zu nennen sind unter anderem die folgenden Kerneigenschaften:

Farbigkeit von Token Token können grundsätzlich eine Eigenschaft des Typs String, Boolean, Integer oder Float (Real) haben.

Abstrakte Datentypen als Farbe Für Token lassen sich eigene Eigenschaften als Zusammensetzung von Basisdatentypen definieren. Besonders komplexe Zusammenhänge wie Modelle von Produktionsstätten lassen sich damit vergleichsweise kompakt, übersichtlich und trotzdem nachvollziehbar modellieren.

Hierarchie Spezielle Transitionen können Subnetze beliebiger Größe enthalten, welche wiederum derartige **Substitutionstransitionen** enthalten können. Grundsätzlich ist damit das Konzept hierarchischer Petri-Netze abgedeckt, wird jedoch sogar noch erweitert, wie im folgenden Punkt genannt.

Implementierungsalternativen für Subnetze Substitutionstransitionen können nicht nur eine, sondern beliebig viele Implementierungen (Petri-Netze) auf gleicher Hierarchieebene enthalten. Während der Simulation wird im Normalfall eine der schaltfähigen Implementierungen genutzt und simuliert. Durch ein Konstrukt im Subnetz lässt sich jedoch kontrollieren, welche konkrete Implementierung verwendet wird.

Zeitabhängiges Schaltverhalten Unabhängig von der grundsätzlichen Schaltfähigkeit einer Transition können Transitionen zeitverzögert schalten. Diese Zeitverzögerung wird in Simulationszeitschritten berechnet und kann über mannigfaltige Funktionen definiert werden. Dies ist ein wesentlicher Punkt bei der Modellierung und Simulation natürlicher Prozesse.

Bedingte Schaltfähigkeit von Transitionen Die Schaltfähigkeit von Transitionen hängt nicht nur von ihren Ein- und Ausgangsbedingungen ab, sondern kann zusätzlich über Aktivierungsfunktionen gesteuert werden; ein wichtiges Vehikel für die Untersuchung alternativer Subnetzarchitekturen.

Bedingte Aktivierung von Kanten Aktivierungsbedingungen von Transitionen, die an Ausgangskanten gekoppelt sind, können ebenfalls diverse Teilbedingungen enthalten. So lassen sich Token oder auch nur Teile ihrer Datenstrukturen zum Schaltzeitpunkt ändern.

Parametrisierbarkeit Für die simulationsbasierte Optimierung ist es erforderlich, ein Modell von außen konfigurieren zu können. Dies lässt sich bei SCPN sehr gut über globale Parameter realisieren. Die Parameter sind überall im Modell lesbar und eröffnen damit viele Möglichkeiten der flexiblen Modellierung.

Da die Schaltfähigkeit einer Transition auch von global verwendeten Variablen und Parametern in TimeNET abhängig sein kann, lassen sich hierarchische Petri-Netze entwickeln, die eine von außen gesteuerte Analyse verschiedener Netzarchitekturen ermöglichen. Die Möglichkeit der Architekturoptimierung ist dadurch mit dem später vorgestellten Optimierungswerkzeug gegeben, aber nicht Gegenstand dieser Arbeit.
Zusammenfassend ist dieser Überblick keinesfalls vollständig, sondern soll nur einen Eindruck von der Flexibilität farbiger Petri-Netze, insbesondere deren Implementierung in TimeNET, geben. Eine ausführliche Darstellung der Möglichkeiten und Beispiele für den Einsatz von SCPNs findet sich in „Stochastic Discrete Event Systems - Modeling, Evaluation, Applications“ [67]. Die Summe ihrer Eigenschaften, zusammen mit den Möglichkeiten des genannten Modellierungs- und Simulationswerkzeuges, machen es für den Einsatz zur Modellierung komplexer Systeme sowie für deren Optimierung sehr interessant. In dieser Arbeit kommt nur der Simulationstyp **stationär** zur Anwendung. Dabei werden so viele Ereignisse bzw. dafür notwendige Zeitschritte simuliert, bis die Werte aller Leistungsmaße innerhalb des geforderte Konfidenzintervalls liegen und damit die geforderte Genauigkeit erreicht ist. TimeNET bietet als Alternative für Modelle, deren Leistungsmaße nicht konvergieren, den Simulationstyp **transient**. Dies bedeutet, dass die Simulation immer wieder von vorn begonnen und zu einem definierten Zeitpunkt abgebrochen wird. Der Nutzer definiert eine Anzahl Messpunkte innerhalb dieses Intervalls, zu denen der gemessene Wert des Leistungsmaßes auf ausgewertet wird.

2.2 Simulationsbasierte Optimierung

2.2.1 Begriffsdefinition

Für das Verständnis der Arbeit sollen an dieser Stelle die wichtigsten Begriffe in Bezug auf simulationsbasierte Optimierung erläutert werden. Sie tauchen in dieser oder ähnlicher Form auch in der untersuchten Fachliteratur zum Thema auf.

Ein **Parameter** ist ein Übergabewert, der entweder intern (im Modell) das Modell und damit das Simulationsergebnis beeinflusst oder extern an das Simulationswerkzeug übergeben wird und damit primär die Art der Simulation beeinflusst. Dies hat ebenfalls Einfluss auf das Simulationsergebnis und die benötigte Simulationszeit. In Kapitel 4 wird dieser Zusammenhang näher erläutert.

Die Summe aller Parameter mit ihren konkreten Werten für eine Simulation wird als **Parametersatz** bezeichnet.

Der **Definitionsraum** ist die Menge aller möglichen Parametersätze. Bei kontinuierlichen Parametern wäre der Definitionsraum unendlich groß. Er wird durch geeignete Diskretisierung eingeschränkt.

Eine **Kostenfunktion**, auch Antwortfunktion genannt ist die Funktion, die den Rückgabewert für ein Leistungsmaß bestimmt. Typischerweise soll das Optimum dieser Funktion gefunden werden. Verallgemeinert ist es die Abbildung aller Werte des Definitionsbereichs auf Ergebnisse der Simulation.

Die **Wertelandschaft** bezeichnet die Summe aller Simulationsergebnisse, zugehörig zu den Werten des Definitionsbereichs. „Landschaft“ impliziert dabei, dass sich dies grafisch darstellen lässt, was aber nicht zwingend erforderlich ist. Bei der Abbildung von Ergebnisdaten zweidimensionaler Wertebereiche ergeben sich sogenannte **Antwortfunktionsgebirge**.

2.2.2 Simulationsbasierte Optimierung

Bei der Optimierung von Systemen kann man grundsätzlich zwischen zwei Arten unterscheiden: der direkten Optimierung und der indirekten. Direkte Optimierung bedeutet dabei, dass sich aus den Eigenschaften eines Systems bzw. eines Modells direkt Maßnahmen oder Parameterwerte schlussfolgern lassen, die ein optimales Systemverhalten (i.d.R. minimale oder maximale Werte für ein Leistungsmaß) versprechen. Schlussfolgern kann in diesem Fall ein, mit viel Rechenaufwand verbundener, Prozess sein. Dies ist jedoch nur möglich, wenn der Einfluss aller variablen Parameter auf das Systemverhalten bekannt und berechenbar ist. Bekanntester Vertreter der direkten Optimierung ist **Linear Programming** sowie dessen Abwandlung, **Mixed Integer Linear Programming**. Sie werden in der Literatur häufig erläutert und insbesondere für die lokale Optimierung erfolgreich eingesetzt. Jedoch ist ihr Einsatz auf Probleme beschränkt, die durch eine (beliebig große) Anzahl Funktionen beschrieben sind [36].
Bei komplexen Systemen wie Netzwerken für Kommunikation, Energie etc. ist der Einsatz direkter Optimierungsmethoden typischerweise nicht möglich. Da solche Systeme aus vielen Teilsystemen bestehen, deren jeweiliges Verhalten nur durch stochastische Funktionen modelliert werden kann, ist man gezwungen, solche Systeme zu simulieren und anhand der Simulationsergebnisse Schlüsse für die Optimierung zu ziehen.
In der Literatur werden dafür verschiedene Begriffe geprägt. **Indirekte Optimierung** kann dabei auch bedeuten, dass der Optimierungssteuerung gewisse Kenntnisse zu Architektur oder Verhalten des zu optimierenden Systems vorliegen. Wenn überhaupt keine Kenntnisse über das zu optimierende System vorliegen, abgesehen vom Definitionsbereich der Eingangsparameter, spricht man von einer **Black-Box Optimierung**. Der generelle Ablauf einer solchen ist in Bild 2.2 dargestellt.
Wenn für die Erzeugung der jeweiligen Ergebnisse eine Simulation notwendig ist (und das ist i.d.R. so) so spricht man von **simulationsbasierter Optimierung**. Wie dargestellt, erzeugt die Optimierungssteuerung jeweils einen Parametersatz als Eingabe für das Simulationssystem, startet die Simulation des Modells und wertet anschließend die Ergebnisse aus. Dabei wird entweder ein weiterer Parametersatz berechnet oder, sofern eine der Abbruchbedingungen erfüllt ist, das gefundene Optimum ausgegeben.

Ein anderer Aspekt ist die Tatsache, dass eine einzelne Simulation eines Parametersatzes nicht ausreicht, selbst wenn das Optimum gefunden scheint. Um dem Simulationsergebnis vertrauen zu können, muss der Erwartungswert des Simulationsergebnisses eine definierte maximale Abweichung erreichen. Anders formuliert, jeder Parametersatz muss „ausreichend“ häufig simuliert werden, da das Modell selbst beliebig viele stochastische Komponenten enthalten kann. Bei der Simulation mit TimeNET wird dieser Problematik bereits Rechnung getragen. Der Nutzer kann hier bereits vor der Simulation das geforderte Vertrauen in die Ergebnisse bzw. das Konfidenzniveau festlegen.
Mit diesen typischen Problemen simulationsbasierter Optimierung beschäftigten sich bereits viele Forschergruppen lange bevor Simulationswerkzeuge allgemein verbreitet waren [53, 52]. Später folgten viele Ansätze, Simulation und Optimierung optimal zu koppeln. Einerseits existieren einige Optimierungsalgorithmen bereits länger als Computer allgemein verfügbar sind [41]. Andererseits gibt es stets konzeptuelle und praktische Hürden zu überwinden, wenn es darum geht, die simulationsbasierte Optimierung möglichst effizient zu gestalten und dennoch eine hohe Qualität zu erreichen [47, 23]. Qualität bedeutet hier die Nähe eines Gefundenen zum tatsächlichen Optimum bzw. die empirisch bestimmte Wahrscheinlichkeit, tatsächlich das Optimum gefunden zu haben [40].

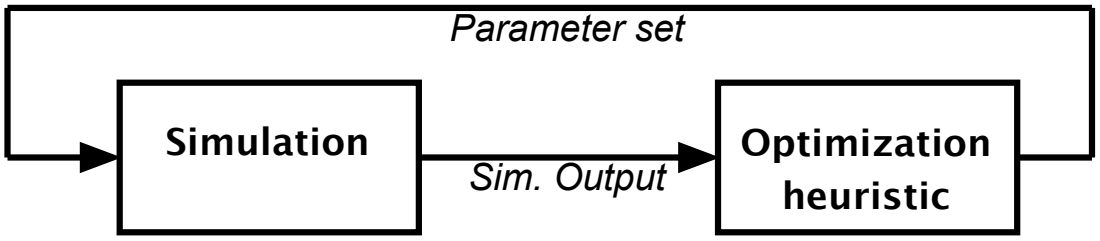


Abbildung 2.2: Blackbox Optimierung, Schema [13, 53, 39]

SCPNs, wie auch andere parametrisierte Petri-Netze eignen sich prinzipbedingt sehr gut für die simulationsbasierte Optimierung, ähnlich anderen Modellen mit diskreter Ereignissimulation [4]. Einer der Hauptvorteile der Nutzung von SCPN bei stationärer Simulation mit TimeNET ist die Kontrolle der erwarteten Simulationsgenauigkeit durch Variation des zu erreichenden Konfidenzniveaus und der maximalen Ergebnisabweichung. Die Auswirkungen und der Nutzen dessen wird in Kapitel 4 näher erläutert.

2.3 Allgemeine Ansätze zur Optimierung

Abhängig vom Modell bzw. der Problemstellung und der Eigenschaften der Parameter wurden im Laufe der Zeit diverse Ansätze zur simulationsbasierten Optimierung diskutiert. Basierend auf den Arbeiten von Yolanda Carson und Anu Maria [13] sowie Micheal C. Fu [23] soll hier ein aktueller Überblick über Klassen von Optimierungsalgorithmen und ihre jeweiligen Einsatzgebiete gegeben werden.
Die erste Klasse an Algorithmen bildet die **Gradienten-basierte Suche**. Hierbei werden zunächst einige Parameterwerte bzw. Kombinationen simuliert. Anhand der Simulationsergebnisse, die man auch als Antwortfunktion verstehen kann, wird der jeweilige Anstieg dieser Funktion bestimmt. Diese Information wird nun genutzt, um eine (vermutlich) bessere Kombination von Parameterwerten zu bestimmen. Wiederholt wird dieser Vorgang bis der Anstieg einen entsprechenden Schwellwert unterschreitet, d.h. neu errechnete Parameterwerte zu keiner wesentlichen Verbesserung des Funktionswertes/Simulationsergebnisses führen.
Ähnlich dieser Klasse verhalten sich **Response Surface Methoden**. Diese versuchen aufgrund der Simulationsergebnisse eine mögliche Antwortfunktion zu bestimmen, indem bekannte Funktionen wie Polynome durch Regression Schritt für Schritt angepasst werden. Abhängig von den verwendeten Approximationsfunktionen und der Anzahl an durch Simulation bestimmten Stützstellen entsteht dadurch eine synthetische Antwortfunktion, deren Optimum symbolisch berechnet werden kann. Oft werden aber auch einfache Funktionen verwendet und das Verfahren wiederholt, wenn die Abweichung der Simulationsergebnisse von den geschätzten Werten zu groß wird oder ein scheinbares Optimum gefunden wurde[53]. Ein ähnliches Verfahren wird in der Klasse der **Sample Path Methoden** verwendet. Auch hier werden aus den Simulationsergebnissen Näherungsfunktionen abgeleitet und für die Bestimmung der folgenden zu testenden Parametersätze genutzt [25]. Die Idee, anstatt einfacher Funktionen gleich andere simulierbare Modelle –praktisch Metamodelle– zu verwenden, wurde ebenfalls untersucht und kann eine vielversprechende Alternative zur Einbettung des Simulationswerkzeuges in eine externe Optimierungsumgebung darstellen [28].
Statistische Methoden gibt es in vielen Untervarianten. Wesentliche Gemeinsamkeit ist, dass entweder das Optimum selbst oder eine Menge an möglichen Parameter-Konfigurationen, welche das Optimum enthält, mit einer vorgegebenen Wahrschein-

lichkeit gefunden wird. Insbesondere Methoden des Ranking & Selection fallen in diese Kategorie [23].
Unter **A-Teams** (Asynchronous Teams) werden Methoden verstanden, bei denen mehrere unterschiedliche Strategien zum Finden des Optimums kombiniert werden. Sei es gleichzeitig oder durch aufeinanderfolgende Anwendung. Ergebnisse und Verhalten eines Algorithmus fließen dabei als Steuerparameter in den jeweils nächsten Algorithmus ein. In der Literatur werden solche und ähnliche Verfahren teilweise auch als Hyper- oder Metaheuristiken bezeichnet, da hier die Wahl und Konfiguration einer Optimierungsheuristik durch eine andere Heuristik bestimmt wird [12, 20].

2.3.1 Abgrenzung Algorithmus / Heuristik

Die Begriffe Algorithmus und Heuristik wurden und werden in vielen Fällen synonym verwendet. Die Umsetzung einer Heuristik als Ablauf von Arbeitsschritten oder Computerprogramm ist durchaus auch ein Algorithmus, daher wurde bis hierher auch nicht unterschieden. Im Allgemeinen gilt für das Gebiet der Optimierung, dass ein Optimierungsalgorithmus zwangsläufig und nach endlich langer Zeit die optimale Lösung für ein gegebenes Problem findet, sofern es existiert. Dafür sind i.d.R. Kenntnisse über die Antwortfunktion notwendig. Linear Programming und MILP sind typische Vertreter dieser Klasse.
Im Gegensatz dazu sind **Heuristiken** Methoden, die mit minimalen oder gänzlich fehlenden Informationen über zugrunde liegende Verteilungen und mögliche Antwortfunktionen arbeiten [64]. Sie werden im Allgemeinen als eine Teilmenge von Algorithmen zur Lösung von (Optimierungs-) problemen betrachtet. Oft arbeiten sie mit Versuch und Irrtum und sammeln erst während ihrer Laufzeit Wissen über das Problem an, um dies für seine Lösung nutzen zu können. Daher ist ihr grundsätzliches Verhalten in vielen Fällen nicht vollständig vorgegeben. Stattdessen fließen meist auch Zufallsgrößen in die Suche ein [55, 50].
Eine sehr passende Definition bzw. Abgrenzung von Optimierungsalgorithmen und Heuristiken findet sich in „Naturanaloge Verfahren: Metaheuristiken zur Reihenfolgeplanung“.

> [...] dass eine Heuristik als ein Verfahren zu kennzeichnen ist, das im Mittel gute Lösungen mit akzeptablem Aufwand generieren kann, ohne aber das Erreichen einer optimalen Lösung garantieren zu können [20].

Im Sinne dieser Definition ist jeder Algorithmus zur simulationsbasierten Optimierung eine Art Heuristik, denn die Simulation und damit die scheinbare Antwortfunktion selbst ist stets zufallsbehaftet. Die bekannten Verfahren wurden jedoch üblicherweise für eine entsprechende Problemklasse ohne Anwendung zur simulationsbasierten Optimierung entwickelt. Daher soll hier ein kurzer Überblick aktuell üblicher Einteilungen gegeben werden.
Im Weiteren wird auf den Versuch der Trennung zwischen Algorithmus und Heuristik verzichtet und Algorithmen stets als terminierende Folge von Anweisungen betrachtet, egal welche Idee damit umgesetzt werden soll.
In der Literatur werden verschiedene Systematiken zur Einteilung von Optimierungsverfahren diskutiert. Heiner Müller-Merbachs Systematik von 1981 stellt eine bis heute geltende Basis hierfür. Sie ist in Bild 2.3 gezeigt. Die grobe Unterscheidung zwischen direkten und iterativen Verfahren gilt im Wesentlichen noch immer.
Eine alternative Darstellung zeigt Bild 2.4. Insbesondere sind hier nur die indirekten Verfahren betrachtet, da nur diese für die simulationsbasierte Optimierung in Frage kommen. Markiert sind im Bild diejenigen Verfahren, die im Rahmen dieser Arbeit implementiert und validiert wurden.

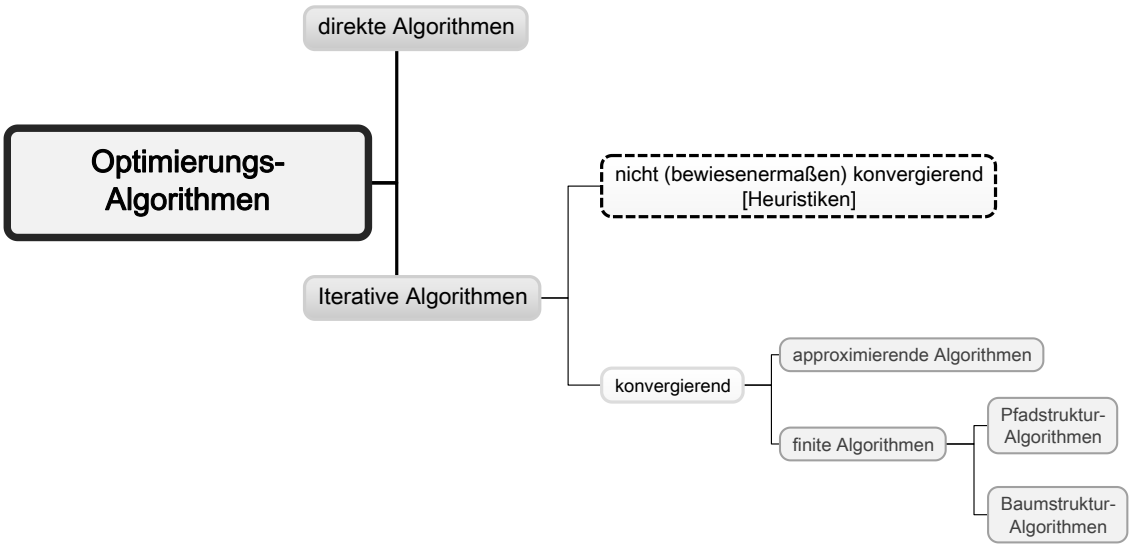


Abbildung 2.3: Einordnung von Heuristiken nach Müller-Merbach [43]

Die Tatsache, dass diese Herangehensweisen prinzipiell für die Optimierung bei Unkenntnis der Antwortfunktionsgebirge entwickelt wurden, macht sie für die simula-

tionsbasierte Optimierung besonders interessant. Beim realen Einsatz muss jedoch beachtet werden, dass die Anzahl an benötigten Simulationsläufen und die jeweilige Simulationsdauer großen Einfluss auf die Gesamtzeit für die Optimierung haben und dies bei der Entwicklung der meisten Heuristiken noch nicht als Kernproblem eingeflossen ist.
Schwerpunkt dieser Arbeit ist die Untersuchung von Heuristiken im Zusammenhang mit der Simulation von farbigen stochastischen Petri-Netzen. Die bekanntesten und verbreitetsten sind dabei Hill-Climbing und Simulated Annealing sowie ihre Unterarten bzw. Konfigurationen.
Anders als Carson und Maria es beschreiben, sind **stochastische Optimierungsverfahren** auch eher bei den Heuristiken angesiedelt [13]. Es sind Verfahren, bei denen die Suche nach dem Optimum zufallsbehaftete Elemente enthält oder die Antwortfunktion selbst nicht deterministisch aus der gleichen Eingabe wiederkehrend gleiche Ausgaben erzeugt, was eine statistische Auswertung der Antwortfunktionen notwendig macht. Im Bereich der **Ranking & Selection**-Verfahren, die z.B. bei [45, 56] beschrieben sind, werden solche Algorithmen eingesetzt. Derartige Verfahren sind prinzipbedingt für Probleme mit relativ eingeschränktem Definitionsraum geeignet, da jeder Parametersatz (Eingangsvektor) häufig simuliert werden muss, um mit entsprechender Sicherheit eine Aussage über das gefundene Optimum treffen zu können [44, 25].
Bei der hier verwendeten **stationären Simulation** von SCPNs mit TimeNET treten viele dieser Probleme für den Nutzer nicht auf, da die „Vertrauenswürdigkeit“ der Ergebnisse direkt gesteuert werden kann (siehe Kapitel 4). Für die transiente Analyse von SCPN unter Vorgabe von Grenzen in Simulationszeit oder -schritten können diese Verfahren in der Zukunft jedoch Bedeutung gewinnen.
Die meisten Verfahren können um zufälliges Verhalten bei der Parameterbestimmung ergänzt werden, um lokale Optima zu überwinden. Verfahren, die sich an natürlichen Prozessen orientieren, wie genetische Algorithmen, sowie auch Simulated Annealing fallen ebenso in diese Kategorie, da auch dort die Wahl des nächsten Parametersatzes teilweise zufallsbehaftet ist [49, 74].

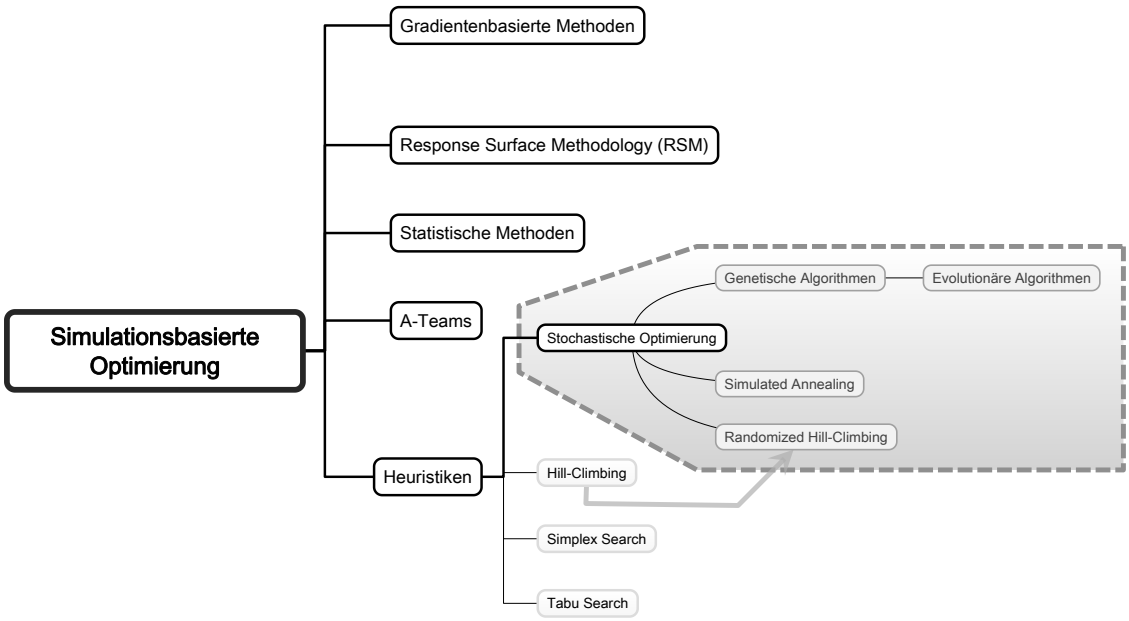


Abbildung 2.4: Klassifikation simulationsbasierter Optimierungsverfahren

2.4 Etablierte Heuristiken

Schwerpunkt der Untersuchung in dieser Arbeit sind verschiedene etablierte Heuristiken, die in unterschiedlichen Konfigurationen untersucht und in einer Metaheuristik eingebettet an künstlichen und praktischen Beispielen getestet werden. Im Folgenden sollen diese Heuristiken in ihrer hier eingesetzten Form vorgestellt werden. Grundsätzliche Einschränkung für die Verwendung der genannten Heuristiken ist, dass der jeweilige Definitionsbereich des Problems bzw. der Antwortfunktion endlich groß und diskret ist. Die Diskretisierung kann dabei beliebig -aber endlich- klein sein.

2.4.1 Hill-Climbing

Hill-Climbing, auch Bergsteigeralgorithmus genannt, ist die bekannteste und am weitesten verbreitete Heuristik, in vielen Fällen jedoch auch die ineffizienteste. Das Vorgehen ist durch den Namen bereits gut beschrieben. Vergleichbar ist diese Heuristik mit dem Besteigen eines Berges mit verbundenen Augen. Beginnend an einem festgelegten oder zufälligen Punkt im Definitionsraum werden ein oder mehrere Parameter schrittweise verändert. Ist das (Simulations-)Ergebnis des neuen Parameter-

satzes besser, so wird dieser übernommen. Ist es schlechter, wird davon ausgegangen, dass das Optimum bereits überschritten wurde und der letzte Parametersatz wird als Optimum akzeptiert.
Dieses Vorgehen wird u.A. auch als „gierig“ bezeichnet. Dies sind Algorithmen, die konsequent eine neue Konfiguration (Parametersatz) akzeptieren, wenn sie besser als die vorhergehende ist. Ist sie schlechter, wird abgebrochen. Dadurch eignet sich Hill-Climbing sehr gut für das Finden lokaler Optima. Ein globales Optimum kann leicht vom Algorithmus übersehen werden, weshalb im Laufe der Zeit diverse Varianten entwickelt wurden. Im Folgenden werden die Varianten beschrieben, die im Rahmen der Arbeit umgesetzt und untersucht worden sind.
Ausgehend davon, dass der Definitionsbereich aller Parameter diskretisierbar ist bzw. das Modell eine solche Sicht erlaubt, gibt es verschiedene Strategien zur Bestimmung des nächsten Parametersatzes für die Simulation, also zur Bestimmung des Antwortfunktionswertes. Es wurden die folgenden Varianten untersucht:

- *Diskret, aufwärts*
- *Diskret, auf/ab*
- *Zufällig, diskret, auf/ab*
- *Zufällig, diskret in Nachbarschaft*
- *Zufällig, kontinuierlich in Nachbarschaft*
- *Zufällig, diskret im Definitionsraum*

Diskret, aufwärts kann als Test für Teile der Optimierungssteuerung verstanden werden. Dabei wird jeder variable Parameter mit seinem Minimalwert initialisiert und dann schrittweise erhöht. Die Schrittweite, auch Diskretisierung genannt, wird hierbei und im Folgenden durch den Nutzer der Optimierungssteuerung festgelegt. Die Optimierung gilt als beendet, sobald die Erhöhung der Parameterwerte keine verbesserten Antwortfunktionswerte bzw. Simulationsergebnisse liefert.
Als wichtigste Strategie bei der Bestimmung neuer Parametersätze wurde *Diskret, auf/ab* gewählt. Grundsätzlich wird dabei im Round Robin Verfahren jeweils ein Parameter gewählt, schrittweise verändert und eine Simulation durchgeführt bzw. der Antwortfunktionswert berechnet. Ist dieser Wert besser als der Vergleichswert,

so wird der neue Wert und der zugehörige Parametersatz als Vergleich übernommen. Um lokale Optima überbrücken zu können, wurden zwei Steuerparameter für den Algorithmus eingeführt. Zum einen kann ein einzelner Parameter mehrmals inkrementiert werden und dennoch zu schlechteren Antwortfunktionsergebnisssen führen. Dies wird im realisierten Programm *WRONG_SOLUTION_PER_DIRECTION* genannt. Wird diese Anzahl an schlechteren Resultaten zum ersten Mal erreicht, startet die Prozedur mit dem letzten guten Parameterwert und geänderter Richtung, er wird also von dort aus dekrementiert. Führt dies nach der gleichen Anzahl an Simulationen nicht zu einem besseren Resultat, so wird der Parameter abermals zurück gesetzt und der nächste mögliche Modellparameter wird inkrementiert. Der Algorithmus terminiert, nachdem *WRONG_SOLUTIONS_IN_A_ROW* viele Simulationsläufe nacheinander kein besseres Ergebnis hervorbrachten. Durch Variation der beiden genannten Steuerungsparameter des Optimierungsalgorithmus kann dieser an das jeweilige Problem und dessen vermutetes Antwortfunktionsgebirge angepasst werden.
Viele bekannte Heuristiken zur Systemoptimierung enthalten zufallsbehaftete Komponenten, um neue Parametersätze zu berechnen. Der vergleichsweise einfache Hill-Cimbing-Algorithmus wurde deshalb im Rahmen dieser Arbeit ebenfalls um derartiges Verhalten erweitert, um ggf. lokale Optima überwinden zu können. Die Konfiguration Zufällig, diskret, auf/ab erweitert dabei die beschriebene Variante *Diskret, auf/ab*, wobei hier durch einen Zufallsgenerator entschieden wird, ob der gewählte Parameter inkrementiert oder dekrementiert wird. Folglich hat *WRONG_SOLUTION_PER_DIRECTION* hier keinen Effekt.
Bei *Zufällig, diskret in Nachbarschaft* wird der nächste Wert eines Parameters zufällig innerhalb eines definierten Bereichs um den bisherigen Wert gewählt. Dabei wird die Diskretisierung des Definitionsraumes beachtet, also der zufällig bestimmte Parameterwert entsprechend gerundet. Die Größe der Nachbarschaft wird in % des jeweiligen Definitionsbereichs angegeben. Neben der diskreten Variante kann es unter Umständen sinnvoll sein, Parameterwerte kontinuierlich zu wählen, also ohne festgelegte Diskretisierung. Dies wurde mit *Zufällig, kontinuierlich in Nachbarschaft* umgesetzt. Kontinuierlich bedeutet dabei, dass die Parameterwerte Fließpunktzahlen sind. Dadurch ergibt sich eine Art Pseudokontinuität, da die zufällige Wahl von Fließpunktzahlen von verschiedenen Faktoren abhängig ist und i.d.R. eine sehr feine Diskretisierung aufweist.

Lediglich als Vervollständigung, denn die Größe der Nachbarschaft kann beliebig zwischen 0..100 % sein, gibt es die Variante *Zufällig, diskret im Definitionsraum*, bei der jeder neue Parameterwert –analog der Brute-Force-Methode– zufällig aus dem gesamten Definitionsbereich gewählt wird.

2.4.2 Simulated Annealing

Simulated Annealing kann als Spezialfall von Hill-Climbing betrachtet werden, was sich auch in der umgesetzten Softwarearchitektur (siehe 6.3) widerspiegelt. Es wurde bereits auf einem der ersten Computer in abgewandelter Form von N. Metropolis untersucht [41]. Der dort vorgestellte Algorithmus kann als Basis aller späteren Varianten von Simulated Annealing angesehen werden. Die Namensgebung geht allerdings auf Kirkpatricks Veröffentlichung aus dem Jahre 1983 zurück [64].
Das Grundprinzip von Simulated Annealing ist das Nachbilden eines natürlichen Abkühlungsprozesses thermoplastisch verformbarer Materialien. Zentrale Elemente dieser Heuristik sind dabei die zufallsbehaftete Bestimmung des jeweils nächsten Parametersatzes und die ebenfalls zufallsbehaftete Akzeptanz eines schlechteren Wertes der Antwortfunktion. Die „Temperaturen“, welche die Entfernung zum bisherigen Parametersatz und die Akzeptanz eines schlechteren bestimmen, sowie ihre Reduktionsgeschwindigkeit (Abkühlungsfunktion) sind die wichtigsten Konfigurationsmöglichkeiten des Algorithmus.
Der folgende Pseudocode veranschaulicht den prinzipiellen Ablauf:
Für das Verständnis des Algorithmus sind einige Erläuterungen notwendig. p ist der zu modifizierende Parameter, dessen optimaler Wert gefunden werden soll. Er wird zunächst mit einem Zufallswert innerhalb seines Definitionsbereichs initialisiert. p_{best} beinhaltet jeweils den besten Parameterwert und wird mit dem gleichen Wert initialisiert. $p_{current}$ ist der zuletzt gefundene beste Parameterwert. T^{para} ist die „Temperatur“ für die Änderung der Parameter. Je höher sie ist, desto größer kann die Änderung eines Parameters pro Schleifendurchlauf sein. T^{cost} geht in die Akzeptanzwahrscheinlichkeit einer Lösung ein, die schlechter ist als die bisher beste. Beide werden typischerweise mit 1 initialisiert und mit jedem Schleifendurchlauf verringert. Somit sinkt mit jedem Durchlauf die Änderung der Parameterwerte und die Wahrscheinlichkeit, eine schlechtere Lösung als die bisher gefundene zu akzeptieren. Eines der wichtigsten Elemente bei der Implementierung von Simulated Anne-

$p := Random(p_{max}, p_{min})$
$p_{best} := p$
$p_{current} := p$
$accepted := 0$
$generated := 0$
$T^{para} := T_0^{para}$
$T^{cost} := T_0^{cost}$
while $(T^{para} \leq T_{min}^{para})$ **or** $(T^{cost} \leq T_{min}^{cost})$ **do**
  $generated := generated + 1$
  repeat
    $p := p + T^{para} * maxStep * signRandom(-1, 1)$
  until $p_{min} \leq p_{max}$
  if $cost(p) < cost(p_{best})$ **then**
    $p_{best} := p$
  end if
  $\delta := cost(p_{best}) - cost(p_{current})$
  if $\delta < 0$ **or** $random[0..1] < e^{\frac{-\delta}{T^{cost}}}$ **then**
    $p_{current} := p$
    $accepted := accepted + 1$
  end if
  $T^{cost} := cool(T^{cost})$
  $T^{para} := cool(T^{para})$
end while
print p_{best}

Algorithmus 1 : Prinzipieller Ablauf von Simulated Annealing mit einem Parameter

aling ist die verwendete Abkühlungsfunktion zur Reduzierung von T^{cost} und T^{para}, $cool(T)$. Hierbei versucht man stets eine optimale Balance zu finden aus schnellem Konvergieren und der Fähigkeit, lokale Optima zu überwinden. Die Optimierung wird abgebrochen, sobald T^{cost} oder T^{para} einen Grenzwert ϵ unterschreitet (im Algorithmus hier $T_{min}^{cost}/T_{min}^{para}$ genannt). Daher lässt sich die maximale Anzahl benötigter Simulationen a priori bestimmen: ein großer Vorteil gegenüber anderen Heuristiken. Die erste verwendete Funktion war die Boltzmannfunktion[67, 64], wie in Formel 2.1 gezeigt. Sie orientiert sich an natürlichen Abkühlungsprozessen. T_k ist dabei die jeweilige Temperatur zum Zeitpunkt bzw. Durchlauf k und T_0 ist die Starttemperatur. Die Namen T_k und T_0 stehen dabei in den Formeln synonym für $T_k^{cost/para}$ bzw. $T_0^{cost/para}$. Da dieser Ansatz jedoch zu viele Simulationsläufe benötigt und somit

schlicht zu langsam ist, obwohl das Optimum mit großer Wahrscheinlichkeit gefunden wird, wurden verschiedene alternative Abkühlungsfunktionen entwickelt, welche im Rahmen dieser Arbeit ebenfalls untersucht wurden. **Fast annealing** und **Very fast annealing** sind zwei solche Alternativen, welche im Rahmen dieser Arbeit ebenfalls untersucht wurden.
Bei der Verwendung der Boltzmannfunktion sollte die Anzahl der benötigten Simulationsläufe unbedingt vorher abgeschätzt werden. So benötigt diese bei typischen Standardeinstellungen von $T_0^{para} := 1$ und $T_{min}^{para} := 0.1$ mindestens 22026 Simulationsläufe, während Fast Annealing nur ca. 10. Very Fast Annealing benötigt, bei den verwendeten Standardparametern für $TRatioScale := 0.00001$ und $TAnnealScale := 100$ zur Berechnung von c ca. 20 Simulationsläufe bei einem veränderlichen Parameter bzw. 100 Simulationsläufe bei zwei Parametern.

$$T_k = \frac{T_0}{\ln k} \tag{2.1}$$

Boltzmann Annealing [67]

$$T_k = \frac{T_0}{k} \tag{2.2}$$

Fast Annealing [63, 67, 60]

$$T_k = T_0 e^{-ck^{1/D}} \tag{2.3}$$

Very fast Annealing [29, 63, 67]

Fast annealing basiert auf einer Cauchy-Verteilung bzw. Weiterentwicklungen dieser, wie von Szu und Hartley beschrieben [60]. In die letzte Abkühlungsfunktion (2.3) fließt die Anzahl an variierbaren Parametern D sowie eine daraus berechnete Konstante c ein. Die konkrete Implementierung durch den Autor dieser Arbeit erfolgt nach dem Algorithmus von Lester Ingber [29], wobei die Werte für $TRatioScale$ und $TAnnealScale$ zur Berechnung von c nach Formel 2.4 aus dem Buch „Stochastic Discrete Event Systems - Modeling, Evaluation, Applications“ [67], Seite 227/228

stammen.

$$x = -ln * TRatioScale * e^{(-ln\frac{TAnnealScale}{D})} \tag{2.4}$$

[67]

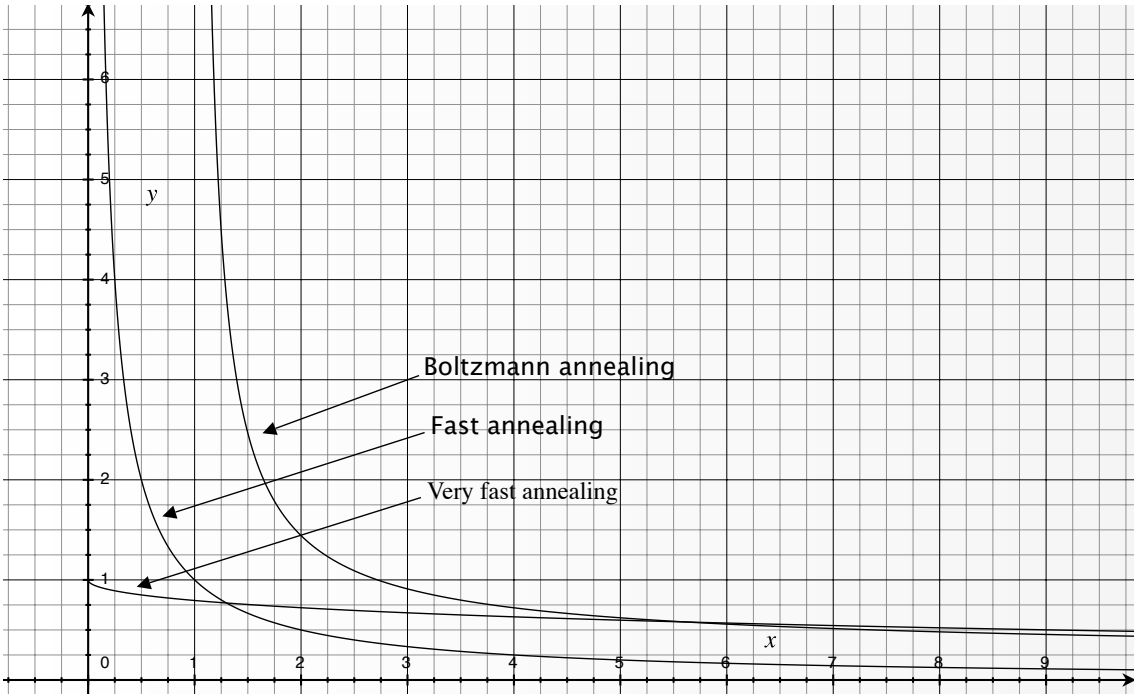


Abbildung 2.5: Übliche Abkühlungsfunktionen von Simulated Annealing [a]

[a] D=2 für Very Fast Annealing

Abbildung 2.5 zeigt den Verlauf der genannten Funktionen und lässt erkennen, wie wichtig die Wahl der Abkühlungsfunktion für die benötigte Zeit zur Optimierung ist. Die Boltzmannfunktion hat sich in vielen Fällen als unpraktisch erwiesen, wie anhand der Ergebnisse in späteren Kapiteln deutlich wird.
Ein großer Vorteil von Simulated Annealing gegenüber anderen Heuristiken ist die Fähigkeit, lokale Optima zu überwinden und somit die größere Wahrscheinlichkeit, dass das gefundene Optimum tatsächlich das globale ist. Nachteil dabei ist die unter Umständen lange Laufzeit sowie das Risiko, nicht exakt das Optimum zu treffen, sondern lediglich einen Wert in dessen Nähe. Aus diesem Grund wurden verschie-

dene mehrstufige Optimierungsheuristiken entwickelt. In der Regel wird dabei in einer ersten Phase der Definitionsbereich grob mit einem Algorithmus untersucht, der lokale Optima gut überwinden kann und in der zweiten Phase mit einem lokalen Suchalgorithmus wie Hill-Climbing die exakte Position des Optimum bestimmt. Teilweise wird dabei auch beide Male der gleiche Algorithmus, aber mit unterschiedlicher Konfiguration, genutzt [67, 73, 27, 45].

2.4.3 Naturanaloge Optimierungsverfahren

Viele erfolgreiche Optimierungsalgorithmen orientieren sich an überlebenswichtigen Strategien aus der Natur. Die Wahl der besten Heuristik ist stark vom Problem bzw. der vermuteten Verteilung der Kostenfunktionswerte abhängig. Zusammenfassende Veröffentlichungen wie die Dissertationsschrift von Sibylle Müller geben dabei Hilfestellung [42].
Die meisten dieser Verfahren lassen sich in zwei Gruppen einteilen. Die erste Gruppe versucht, das Verhalten von Pflanzen und Tieren bzw. von Schwärmen zu imitieren. So ist z.B. das Verhalten von Bienenvölkern bei der Nahrungssuche Vorbild für den Artificial Bee Colony Algorithmus [49]. Ein anderes Beispiel ist der Ant Colony Algorithmus, der die Nahrungssuche einer Ameisenkolonie nachbildet [65].
Die zweite Gruppe umfasst alle Verfahren, welche die Evolution selbst als Vorbild haben. Typische Einflüsse der Evolution wie Vererbung, Mutation und Selektion werden hierbei künstlich nachgebildet bzw. auf Optimierungsprobleme übertragen. Zumeist werden bestehende Lösungen (Parametersätze) neu kombiniert und/oder zufallsbehaftet variiert. Die Auswertung des Kostenfunktionswertes beeinflusst dann die Bestimmung der folgenden Parametersätze durch Mutationen und Rekombination. Dies ähnelt der Evolution von Lebewesen, wobei sich jeweils die besten Mitglieder einer Generation fortpflanzen, also zufallsbehaftet weiterentwickeln und damit der gesamten Population das Überleben durch Anpassung an ihre Umwelt sichern. Es spiegelt sich auch im Namen für diese Kategorie wider: die **Populationsbasierten Optimierungsheuristiken**, zu denen auch jene gehören, die nicht auf Mechanismen der Evolution, sondern des Verhaltens von Tierpopulationen basieren [50, 14, 38].
Im Rahmen dieser Arbeit wird ein genetisches Verfahren untersucht. Der grundsätzliche Algorithmus läuft für alle Verfahren gleich bzw. ähnlich ab. Zunächst werden

für eine Population aus Parametersätzen die Kostenfunktionswerte durch Simulation ermittelt. Anschließend wird eine Auswahl getroffen, diese dient als sogenannte Elterngeneration. Aus den Eltern, also diesen Parametersätzen, wird anschließend die Folgegeneration von Kindern berechnet und die Optimierungsschleife beginnt von vorn. Sobald keine Verbesserung der Kostenfunktionswerte mehr feststellbar ist oder die maximale Anzahl an Optimierungsdurchläufen erreicht ist, wird abgebrochen. Das verwendete Kreuzungsverfahren ist in 6.6 näher erläutert [16].

3 Strategien zur effizienteren, simulationsbasierten Optimierung

Strategien zur effizienteren Optimierung von Systemen mit vielen Konfigurationsmöglichkeiten sind Gegenstand diverser Forschungsprojekte. Kernproblem ist dabei die Größe des Definitionsraumes, des sogenannten Design spaces. Die Untersuchung dessen (engl. design space exploration) und Ansätze für das effiziente Finden der optimalen Werte für die Eingangsgrößen ist Teil dieses Kapitels. Zunächst erfolgt ein Rückblick auf bekannte Strategien und spezifische Probleme simulationsbasierter Optimierung. Die verschiedenen Strategien, welche in der Vergangenheit von diversen Forschergruppen hervorgebracht bzw. untersucht wurden, werden in Bild 3.1 in einer Übersicht reflektiert. Dies soll dem Leser einen Überblick über den Stand der Technik bieten. Dabei werden auch einige Überschneidungen mit grundlegenden Optimierungsverfahren ersichtlich, denn auch diese haben –wenn auch nicht primär– das Ziel, das Optimum mit möglichst wenigen Schritten/Simulationsläufen zu erreichen.
Anschließend werden Einschränkungen und Anpassungen der Heuristiken für die Optimierung von SCPNs vorgestellt, mit denen der Ressourcen- bzw. Zeitbedarf reduziert werden kann. Zur Überprüfung und Bewertung der vorgestellten Algorithmen und Strategien wurden diese im Rahmen eines prototypischen Software-Werkzeuges umgesetzt. Dabei wurden die Algorithmen weitestgehend konfigurierbar gehalten und insbesondere die neuartige Strategie bzw. Hyperheuristik der Multiphasen-Optimierung mit Steuerung der Simulationspräzision erstmals implementiert. Dies wird in Kapitel 6 vorgestellt. Basis der Experimente waren dabei ein akademisches Beispiel-Petri-Netz, verschiedene Benchmark-Funktionen sowie ein SCPN zur Modellierung von Energieflüssen im Haushalt. Die Ergebnisse der Experimente finden sich in Kapitel 8.

3.1 Allgemeine Ansätze

Optimierung von Systemen bzw. ihren Modellen bedeutet, dass für eine gegebene Menge veränderbarer Parameter jeweils der Wert gefunden werden soll, für den die Kostenfunktion, auch Leistungsmaß genannt, über alle Parameter einen optimalen Wert zurückgibt. Typischerweise ist dies als Minimierungsproblem definiert [23]. Im einfachsten Fall werden dabei alle Möglichkeiten durchprobiert, die Resultate geordnet und der beste Parametersatz zurück gegeben. Diese Methode stößt aber sehr schnell an ihre Grenzen, denn abhängig von der Größe des Definitionsraumes, also der Anzahl der Parameter sowie ihrer Diskretisierung, ergeben sich sehr viele Möglichkeiten. Tabelle 3.1 verdeutlicht dieses Problem, wobei jeder genannte Parameter nur 1000 verschiedene Werte annehmen kann.
Bei der simulationsbasierten Optimierung kommen noch zwei weitere Probleme hinzu. Die Berechnung des Funktionswertes für einen Parametersatz, also eine Simulation kann sehr zeitaufwendig sein. Für die minimale Simulationszeit des einfachen SCPNs aus Kapitel 4.1 von 114 Sekunden auf einem aktuellen Standard-PC zeigt Tabelle 3.1 wie schnell das Problem bei steigender Anzahl an Parametern unlösbar wird, zumindest nicht in realistischer Zeit. Hinzu kommt, dass Simulationen oft kein eindeutiges Ergebnis für ein Leistungsmaß liefern, sondern nur dessen Erwartungswert mit entsprechender Unsicherheit. Für eine zuverlässige Bestimmung des Optimums müssten also viele Simulationen mit dem gleichen Parametersatz durchgeführt werden [44]. Dies kann bei SCPN-Simulation implizit durch Vorgabe der Genauigkeitsparameter, wie in Kapitel 4.2 dargestellt, gesteuert werden.

Parameterzahl	Kombinationsmöglichkeiten	Simulationszeitbedarf
1	1 Tsd	32 Stunden
2	1 Mio	3,6 Jahre
3	1 Mrd	3,6 Jtsd.

Tabelle 3.1: Größe des Definitionsbereiches / Zeitbedarf

3.2 Systematik bekannter Ansätze zur beschleunigten Optimierung

Die Menge der bekannten Versuche, die simulationsbasierte Optimierung von Petri-Netzen zu beschleunigen, macht es unmöglich, im Rahmen dieser Arbeit sämtliche Ansätze zu untersuchen. Jedoch lassen sich Gemeinsamkeiten zwischen verschiedenen Strategien finden. In diesem Kapitel wird der Versuch unternommen, bekannte Verfahren anhand ihrer Gemeinsamkeiten zusammenzufassen und ihre Vor- und Nachteile zu beleuchten. Einige der genannten Quellen beschreiben dabei Verfahren, welche nicht primär für die simulationsbasierte Optimierung bzw. deren Beschleunigung entwickelt wurden, in diesem Bereich jedoch anwendbar sind und teilweise praktisch eingesetzt werden.

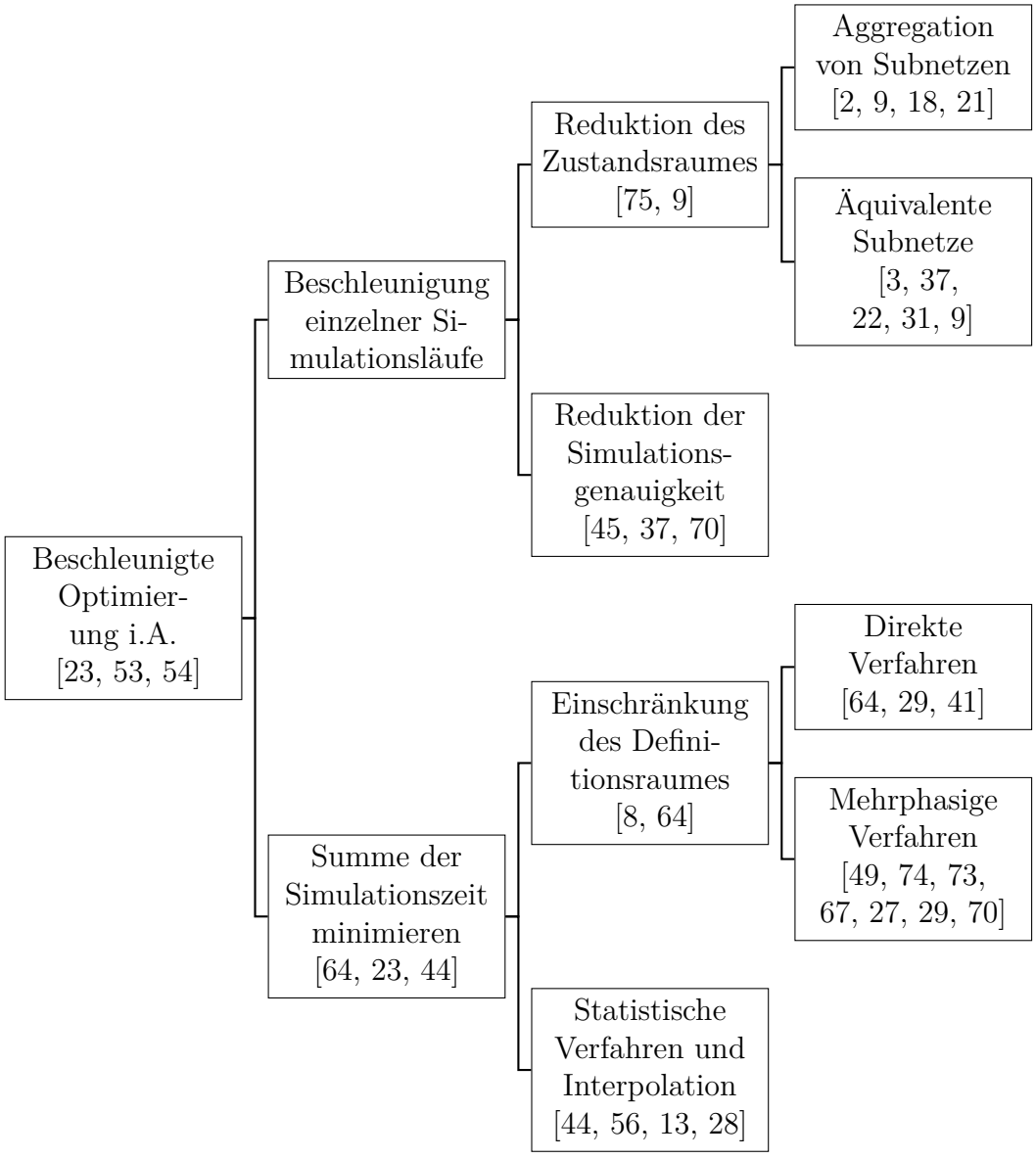


Abbildung 3.1: Ansätze zur Beschleunigung simulationsbasierter Optimierung

Beschleunigte Optimierung im Allgemeinen

Die Grundlagen der simulationsbasierten Optimierung und implizit damit auch die Grundlagen ihrer Beschleunigung wurden bereits in den 1970ern gelegt. Z.B. Dennis E. Smith führte aus, welche Besonderheiten eine simulationsbasierte Optimierung mit sich bringt und welche Verfahren zu ihrer Verbesserung genutzt werden könnten. In seinem Artikel von 1973 („Requirements of an "optimizer" for computer simulations“ [53]) prägte er als zentrale Methoden zur Beschleunigung bereits die Verkleinerung des Zustandsraumes eines Modells sowie die Verkleinerung des Definitionsraumes. Ersteres wurde von ihm als „interne Methoden“ zusammengefasst. Letzteres als „externe Methoden“. Auch wurden hier bereits Vorläufer aktueller Methoden wie HillClimbing und auch RSM erläutert. Selbst die Vorteile einer mehrstufigen Analyse des Definitionsraumes wurden hier bereits erkannt.
Spätere Zusammenfassungen, wie die von Michael C. Fu, lassen die Abstammung der Verfahren von den ersten Ideen erkennen, auch wenn sie im Laufe der Zeit verfeinert wurden. Dies wurde nicht zuletzt durch die breite Verfügbarkeit von Rechentechnik sowie Modellierungs- und Simulationswerkzeugen ermöglicht.
Unabhängig vom Simulationswerkzeug und verwendetem Modellierungsverfahren sind grundsätzlich zwei Eigenschaften der simulationsbasierten Optimierung verantwortlich für den Zeitbedarf. Einerseits die sogenannte **design space explosion**, also die wachsende Größe des Definitionsraumes bei kleinerer Diskretisierung der Parameter oder Erhöhung ihrer Anzahl. Ein Problem, das auch bei anderen Optimierungsverfahren, also nicht nur simulationsbasierten beherrscht werden muss. Es wirkt sich hauptsächlich auf die benötigte Anzahl an Simulationsläufen und damit die Summe der benötigten Simulationszeit aus. Diversen Versuchen, diese Anzahl zu reduzieren ist der Grundgedanke des **Teile & Herrsche** gemein, sodass nicht der gesamte Definitionsbereich untersucht werden muss.
Auf der anderen Seite steht das Problem der **state space explosion**, welche nur bei der simulationsbasierten Optimierung auftritt bzw. problematisch ist. Es beschreibt den Anstieg der möglichen Modellzustände während der Simulation bei Erhöhung der Anzahl verwendeter Modellelemente. Mit komplexer werdenden Modellen kann die Simulationszeit unbeherrschbar stark steigen, abhängig vom Simulationsverfahren und konkreten Modelleigenschaften. Simulationszustände, die niemals erreicht werden, haben dabei keinen Einfluss. Zustände, die sehr selten erreicht werden, stel-

len ein besonderes Problem dar, da ihre Analyse –ohne separate Behandlung– eine Erhöhung der Simulationszeit nach sich ziehen würde [66, 32, 68].

Beschleunigung einzelner Simulationsläufe

Historisch betrachtet wurde bei der Verbesserung der Optimierungsverfahren zunächst der Schwerpunkt auf die Beschleunigung einzelner Simulationsläufe gelegt, wobei zunächst die Verkleinerung des Zustandsraumes im Mittelpunkt stand, da dies nicht nur Rechenzeit, sondern auch Speicher in Anspruch nimmt.

Reduktion des Zustandsraumes

Der Zustandsraum eines Petri-Netzes wird durch einen Erreichbarkeitsgraphen dargestellt. Die Minimierung dieses Graphen und somit auch die Beschleunigung einer möglichen Simulation versucht man durch Aggregation zu erreichen. Dabei werden Teile eines Netzes durch Vereinfachungen ersetzt, die einen kleineren Zustandsraum aufweisen, als die ersetzten Teile. Erforderlich dafür sind Petri-Netze, die eine gewisse Hierarchie erlauben. Zurawski beschrieb dies am Beispiel eines GSPN-Modells einer komplexen Produktionsanlage. Dabei wurden Subnetze durch Vereinfachungen ersetzt, die das gleiche Interface, also das gleiche äußere Verhalten, aufwiesen [75]. Ähnliches beschrieb bereits einige Jahre zuvor Buchholz, insbesondere die Versuche, den Erreichbarkeitsgraphen zu minimieren, denn dies erlaubt auch ohne Simulation verschiedene Analysen. So lässt sich die Auftrittswahrscheinlichkeit einzelner Zustände auch symbolisch berechnen [9]. Auch wird hier bereits vorausgegriffen und die Möglichkeit der Ersetzung von Teilnetzen durch approximierte Aggregation. Denn die größte Schwierigkeit bei Aggregationsverfahren, egal für welche Modellklasse, ist der Nachweis, dass sich ein aggregiertes Teilmodell tatsächlich exakt so verhält wie das zu ersetzende Teilmodell.

Aggregation von Subnetzen

Für viele Teilmodelle kann dies nicht bewiesen werden, weshalb die meisten derartigen Verfahren für die Ausgangsmodelle entsprechende Bedingungen stellen. Nur wenn das Petri-Netz bestimmte Eigenschaften wie Symmetrie aufweist, lassen sich Teile davon vereinfachen und verhalten sich beweisbar exakt wie ihr Vorbild. Für viele Probleme sind diese Verfahren daher nicht praktikabel, da es einerseits schwer

ist, ein komplexes Modell zu erstellen und dabei bestimmte Entwurfsvorgaben wie Symmetrie einzuhalten. Andererseits müssen die Ersetzungen manuell erstellt und der Nachweis des gleichen Verhaltens jeweils geführt werden [2, 9, 18, 21].

Äquivalente Subnetze

Um diese Einschränkungen zu umgehen bzw. die Erstellung von alternativen Subnetzen zu vereinfachen, lag die Idee äquivalenter Subnetze nahe. Das Netz muss prinzipiell keine besondere Form haben. Jedoch ist es hilfreich, die Schnittstellen für die Einbindung der Subnetze zu vereinheitlichen. Diese Ansätze wurden ebenfalls vielfach untersucht und sind in ähnlicher Form unter verschiedenen Namen wie „approximate aggregation“ und „flow-aquivalent subnets“ zu finden. Besonders der Nachweis, dass ein Substitut sich in den „entscheidenen“ Aspekten der Simulation so verhält wie das ersetzte Originalnetz, trotz reduziertem Zustandsraum und damit verringerter Simulationszeit, beschäftigt die Forschung. So konzentrieren sich einige Ansätze darauf, das zeitliche Verhalten der Subnetze korrekt nachzubilden, während andere die korrekte Berechnung der Ergebnisse in den Vordergrund stellen und damit die zeitlichen Aspekte vernachlässigen. Indirekt verschlechtert sich damit die Genauigkeit der Simulation, es ist also stets vor der Simulation abzuwägen, welche Aspekte Schwerpunkt der Untersuchung sein sollen [3, 37, 22, 31, 9].

Reduktion der Simulationsgenauigkeit

Anstatt den Zustandsraum zu reduzieren, um einzelne Simulationen zu verkürzen, zielen andere Verfahren darauf ab, die Genauigkeit einer Simulation direkt zu beeinflussen. Bei Simulationsverfahren, die eine empirische Bestimmung des Erwartungswertes der Messgrößen erfordern, kann dabei einfach die Anzahl der durchgeführten Simulationsläufe reduziert werden. Zwar ist das einzelne Ergebnis damit weniger vertrauenswürdig, in einem großen Definitionsbereich lassen sich damit aber rasch interessante Regionen für die weitere Analyse bestimmen [45, 37]. Andere Verfahren erlauben eine direkte Kontrolle der Simulationsgenauigkeit durch Simulationsparameter. Die stationäre Analyse von SCPNs mit TimeNET ist ein Beispiel dafür, das in dieser Arbeit auch genutzt wird. Ebenso kann die Möglichkeit vereinfachter Subnetze herangezogen werden [69]. Die Verfahren überschneiden sich teilweise mit dem Verwenden äquivalenter Subnetze, jedoch werden die vereinfachten Subnetze nicht

für die gesamte Optimierung verwendet, sondern vom Optimierungssystem gesteuert eingesetzt. Mehrere Stufen der Vereinfachung und Simulationszeitreduktion sind damit möglich.

Summe der Simulationszeit minimieren

Alternative Herangehensweisen versuchen die Summe der Simulationszeit für den gesamten Optimierungsvorgang zu verringern. Dies kann durch Kombination oben genannter Verfahren geschehen, meist ist es aber auf etwas höherer Abstraktionsebene angesiedelt bzw. unabhängig vom konkreten Modellierungs- und Simulationsverfahren. Es geht vielmehr primär darum, mit möglichst wenigen Simulationsläufen das Optimum zu finden [64, 23, 44].

Statistische Verfahren

Die Wahrscheinlichkeit, das globale Optimum gefunden zu haben und nicht ein lokales ist umso geringer, je weniger Simulationen durchgeführt wurden und je kleiner der untersuchte Teil des Definitionsraumes ist. Insbesondere die Notwendigkeit, Simulationen mehrfach durchführen zu müssen, um eine gewisse Mindestwahrscheinlichkeit bzw. Vertrauen in die Ergebnisse zu erlangen, wird in Verfahren des Ranking & Selection behandelt [44, 56]. Diese Problematik muss bei der stationären Simulation von SCPNs mit TimeNET nicht separat betrachtet werden, wie bereits erwähnt. Hier kann die Abweichung der Leistungsmaße, also des Wertes der Antwortfunktion, vor der Simulation festgelegt werden und muss nicht separat berechnet werden. Dies hat zwar ebenfalls Einfluss auf die benötigte Simulationszeit, jedoch lassen sich dadurch die Ansätze des R&S nicht anwenden. Interessanter sind in diesem Zusammenhang Verfahren, die auf Basis der Simulationsergebnisse näherungsweise synthetische Antwortfunktionen bestimmen, um damit mögliche Parameterwerte für die weitere Analyse zu berechnen bzw. uninteressante Bereiche auszuschließen. Bekannteste Vertreter dieser Gruppe sind Response Surface Methoden und ihre Abwandlungen [13, 28].

Einschränkung des Definitionsraumes

Indirekt wird bei diesen Verfahren der Definitionsraum für die Analyse eingeschränkt. Es gibt aber auch Verfahren, bei denen diese Vorgehensweise als Haupt-

merkmal betrachtet werden kann. Simulated Annealing ist einer der bekanntesten Vertreter dieser Gattung. Aber auch Hill Climbing kann als solches Verfahren verstanden werden[8, 64].

Direkte Verfahren

Simulated Annealing geht auf den Metropolisalgorithmus zurück, der schon 1953 für die Teilchensimulation entwickelt wurde. Simulated Annealing bildet algorithmisch das Abkühlen eines thermoplastisch verformbaren Materials nach. Es wird unter anderem beim Leiterplattendesign eingesetzt, um aus Milliarden Routingalternativen die besten Alternativen zu finden. Dabei kommt es nicht zwingend darauf an, das Optimum zu finden. Vielmehr müssen nur gewisse (Kosten-) Schranken eingehalten werden. Ein ideales Einsatzgebiet für Heuristiken[64, 41]. Im Laufe der Zeit wurde Simulated Annealing in diverse Richtungen weiterentwickelt. Einerseits kamen verschiedene Abkühlungsfunktionen hinzu, das Herzstück des Verfahrens. Andererseits wurde mit dem Re-Annealing das Konzept des temporären Wiederaufwärmens eingeführt, um lokalen Optima zu entkommen [29]. Insbesondere der Einsatz bei der simulationsbasierten Optimierung mit diskreter Ereignissimulation wurde untersucht, so z.B. von Biel [4] und Zimmermann [67]. Diese Verfahren beschränken die Größe des Definitionsraumes direkt, ohne mehrmals durchlaufen zu werden. Alternativ dazu gibt es die mehrphasigen Ansätze, welche in jeder Phase einen anderen (größer, kleiner, andere Diskretisierung) Definitionsraum durchsuchen.

Mehrphasige Verfahren

Die Kombination verschiedener Verfahren, insbesondere die Anwendung in mehreren Phasen, wird in verschiedensten Veröffentlichungen diskutiert. Sowohl naturanaloge Verfahren, die das Schwarmverhalten von Tieren nachbilden, als auch solche, die Aspekte der Evolution zum Vorbild haben, laufen Prinzip bedingt immer in mehreren Phasen ab [49, 15]. Diese lassen sich ebenfalls noch in eine Metaheuristik mit mehreren Phasen einbetten [74]. Aber selbst das genannte Re-Annleaing, bei dem Simulated Annealing um Phasen der Erwärmung, also Vergrößerung von Sprungweite und Akzeptanz schlechterer Ergebnisse, erweitert wird, ist im Grunde ein mehrphasiges Vorgehen [29].
Der Einsatz von Simulated Annealing in mehreren Phasen wird u.a. von Zimmer-

mann und Rodriguez beschrieben. In beiden Fällen mit Blick auf die Optimierung von Produktionssystemen bzw. Modellen davon [73, 67, 27].
Bei diesen Kombinationen werden jedoch kaum die Besonderheiten der simulationsbasierten Gewinnung von Antwortfunktionswerten berücksichtigt. Zieht man die Kontrolle über die Simulationsgenauigkeit und damit den Zeitbedarf einzelner Simulationen hinzu, lassen sich weitere Verfahren entwerfen, die zum Teil im Rahmen dieser Arbeit untersucht wurden. Der Austausch von Subnetzen durch exakte oder ähnliche Substitute ist eine Variante davon. Insbesondere die Möglichkeit, Teile des Gesamtmodells mit unterschiedlicher Genauigkeit simulieren zu können, stand dabei im Vordergrund. Die Erstellung der Substitute und der Nachweis ihrer Ähnlichkeit stellen aber nach wie vor eine Hürde dar [70, 69].
Da die Simulation von SCPNs mit TimeNET die Kontrolle der Simulationsgenauigkeit anbietet, wurde zunächst der Ansatz verfolgt, die Genauigkeit der gesamten Simulation zu reduzieren. In mehreren Phasen wird mit zunehmender Nähe zum Optimum die Simulationsgenauigkeit erhöht. Dass dieser Ansatz in Kombination mit bekannten Optimierungsverfahren einen Vorteil hinsichtlich des Zeitbedarfs und der Qualität des gefundenen Optimum bietet, wird in späteren Kapiteln gezeigt [7, 5].

3.3 Besonderheiten bei der Optimierung von SCPN

Systeme als SCPN zu modellieren und zu optimieren bietet einige Einschränkungen, aber auch Vorteile. Grundsätzlich wirken sich die gleichen Faktoren, wie Größe des Definitionsraums und Zeit pro Simulationslauf auf den Gesamtzeitbedarf aus. Die scheinbare Antwortfunktion bzw. das Antwortfunktionsgebirge, welches sich aus den jeweiligen Werten der Leistungsmaße ergibt, ist in den meisten untersuchten Fällen jedoch stetig und zeigt wenige lokale Extrema. Ein Vorteil für Optimierungsalgorithmen, die ihre Schrittweite der Parametervariation anpassen können.
SCPN haben zwei Arten von Parametern. Interne Parameter werden direkt im Modell festgelegt und externe Parameter werden dem Simulationssystem zum Start übergeben. Derzeit wird zur Optimierung jeweils das Originalnetz kopiert und dem Simulationssystem mit veränderten Parametern übergeben. Dies ist im später beschriebenen Optimierungswerkzeug automatisiert (Kapitel 6).
Die Präzision der Simulation kann direkt über externe Parameter gesteuert werden.

Bild 4.5 zeigt den Zusammenhang zwischen gefordertem maximalen relativen Fehler, Konfidenzniveau und benötigter CPU-Zeit für je einen Simulationslauf. Des weiteren lässt sich die Präzision von Teilen des Netzes durch Verwendung alternativer Subnetzimplementierungen umsetzen. Diese grundsätzliche Idee wurde in verschiedener Form in der Literatur genannt [31]. Die Anwendung für SCPN, insbesondere mit Blick auf eine kontinuierliche Anpassung der Modellgenauigkeit während der Optimierung, wurde 2011 vorgestellt [69].
Wenn alternative Teilnetze z.B. zur Architekturoptimierung von komplexen SCPNs herangezogen werden, so zeigen sich leicht Unstetigkeiten und lokale Extrema im Antwortfunktionsgebirge. Dies muss bei der Wahl des Optimierungsverfahrens in Betracht gezogen werden.
Bevor in Kapitel 5 auf die konkret untersuchten Beschleunigungsstrategien bei simulationsbasierter Optimierung auf Basis von SCPNs eingegangen wird, werden im folgenden Kapitel die Ursachen für den unterschiedlichen Zeitbedarf der SCPN-Simulation untersucht.

4 Einflüsse auf den Zeitbedarf stationärer SCPN-Simulationen

Neben der Anzahl an Simulationsläufen ist die benötigte Zeit pro Simulation der wichtigste Einflussfaktor für die Gesamtzeit einer simulationsbasierten Optimierung. Hier soll erläutert werden, welche Faktoren den größten Einfluss auf die benötigte Zeit für die stationäre Simulation eines SCPN haben. Um die unterschiedlichen Einflüsse genauer untersuchen zu können und auch für die späteren Tests der Optimierungsalgorithmen, wird zunächst ein einfaches farbiges Petri-Netz vorgestellt, anhand dessen die Einflussfaktoren erläutert werden. Auch werden Hinweise gegeben, die künftigen Nutzern helfen sollen, Fallstricke bzgl. der Simulation von SCPNs zu vermeiden.

4.1 Typisches Beispielnetz, SCPN

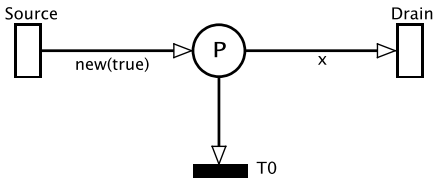


Abbildung 4.1: Einfaches Netz (SimpleOpti)

Die Simulation von SCPNs mit TimeNET bietet unter anderem die Möglichkeit, Durchsätze von Transitionen, durchschnittliche Platzbelegungen und damit gekoppelte Wahrscheinlichkeiten zu untersuchen.
Bild 4.1 zeigt ein einfaches farbiges Petri-Netz mit einer Markenquelle und zwei Senken. Um das Verhalten abschätzen zu können, ist jedoch genauere Kenntnis der

Schaltbedingungen, insbesondere der globalen Guard (Schaltbedingungen) notwendig. Das Netz besteht im wesentlichen aus einem zentralen Platz P mit unbegrenzter Kapazität, einer Transition *Source* zum Füllen des Platzes und 2 Transitionen zum Entleeren.
Dieses Netz wurde für ausgewählt, da es trotz seines simplen Aufbaues viele Möglichkeiten für Untersuchungen im Zusammenhang mit simulationsbasierter Optimierung bietet. Es kann als einfaches Modell einer Verarbeitungsstation innerhalb einer Produktionskette verstanden werden. Wobei für realistischere Ergebnisse die Platzkapazität eingeschränkt werden sollte.
Die Feuerrate von Transition *Source* wird festgelegt durch die Zeitfunktion $F = EXP(\frac{1.0}{Tsource})$, wobei *Tsource* ein Fließkomma-Parameter ist. *Drain* feuert mit einer, durch $F = EXP(\frac{1.0}{Tdrain})$ bestimmten, konfigurierbaren Rate. Auch *Tdrain* ist ein Fließkomma-Parameter, der vom Optimierungssystem verändert werden kann. *T0* ist eine immediate Transition, d.h. sie feuert, sobald ihre Vor- und Nachbedingungen erfüllt sind, ohne zeitliche Bedingungen. Eingeschränkt wird dies nur von ihrem globalen Guard $\#P > 70$. Das bedeutet, die Transition schaltet nur, wenn in Platz P mehr als 70 Marken vorhanden sind.
Während der Simulation sammelt sich eine durchschnittliche Anzahl an Marken im Platz P, welche hauptsächlich von den Parametern *Tdrain* und *TSource* sowie von einer zufälligen Komponente der exponentiellen Zeitfunktionen der Transitionen *Source* und *Drain* abhängt. Diese Markenanzahl in P wird im primären Leistungsmaß (Formel 4.1) verwendet.

$$Measure = 1 - ((40 \leq \#P)\&(50 \geq \#P)) \tag{4.1}$$

Dieses primäre Leistungsmaß gibt die Wahrscheinlichkeit an, mit der sich im Platz P nicht 40 bis 50 Marken befinden. Das Leistungsmaß wurde absichtlich in dieser Weise konstruiert, um ein sogenanntes zweidimensionales Optimierungsgebirge zu generieren wie in Bild 4.2 gezeigt. Dies wird im späteren Verlauf für den Test der Optimierungsheuristiken weiter verwendet, wobei der Wert des Leistungsmaßes als Kostenfunktion dient und minimiert werden soll.
Dieses Netz dient als Grundlage für alle Untersuchungen in Bezug auf den CPU-

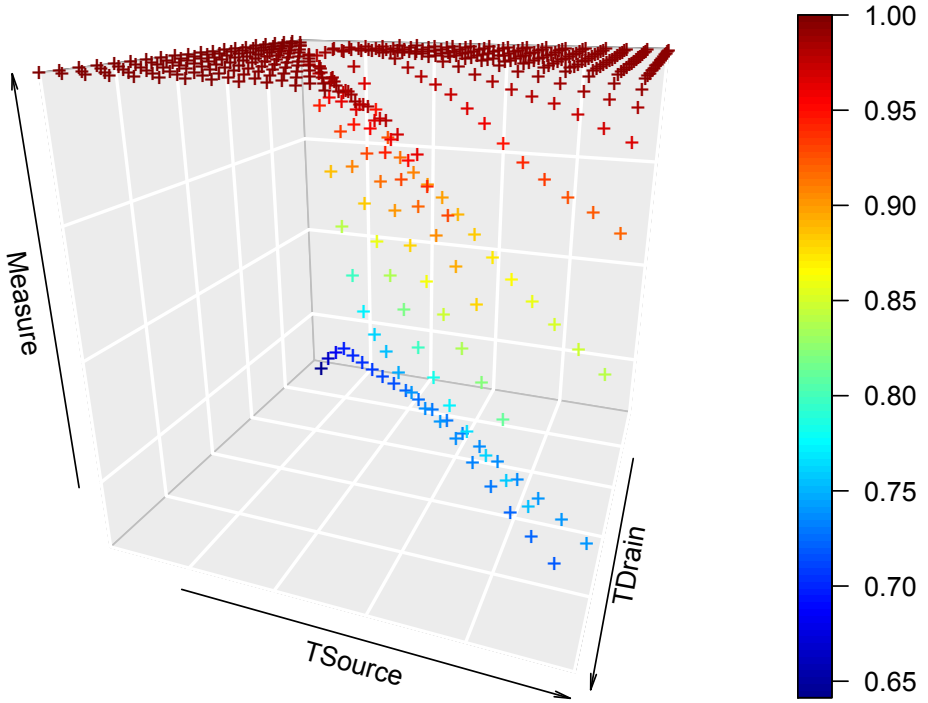


Abbildung 4.2: Ergebnismeasure, Wertelandschaft

Zeit-Bedarf von Simulationsläufen in dieser Arbeit.

4.2 Externe Einflüsse bzw. Simulationssteuerungsparameter

Abhängig von der Art der Simulation beeinflussen verschiedene Faktoren die benötigte Rechenzeit. Ein kurzer Einblick in die Simulation von farbigen stochastischen Petri-Netzen soll dies verdeutlichen. Weitere Details sind in der Fachliteratur zu finden [67].
Grundsätzlich kann zwischen 2 Arten unterschieden werden, der transienten und stationären Simulation.
Bei der **transienten** Simulation bzw. Analyse wird von Anfang an der Endzeitpunkt (in Simulationszeitschritten) für die Simulation festgelegt. Die Simulation wird wiederholt bis zum Endzeitpunkt durchgeführt und dabei die Werte der gewünschten Leistungsmaße zu definierten Zeitpunkten gemessen. Die Analyse wird beendet, sobald alle Leistungsmaße an jedem Messzeitpunkt die gewünschte Genauigkeit erreicht haben.
Genauigkeit wird im Zusammenhang mit der SCPN-Simulation mit TimeNET folgendermaßen beschrieben. Die Werte eines Leistungsmaßes liegen mit einer bestimmten Wahrscheinlichkeit, dem Konfidenzlevel bzw. Konfidenzniveau, innerhalb eines verlangten Intervalls um ihren Erwartungswert. Die Größe dieses Intervalls wird durch den maximalen relativen Fehler festgelegt.
Bei **stationärer** Simulation eines Petri-Netzes ist der Endzeitpunkt für die Simulation nicht vor Beginn festgelegt. Stattdessen wird das Netz so lange simuliert und die Werte der Leistungsmaße gemessen, bis diese ihre geforderte Genauigkeit erreicht haben. Die Werte der Leistungsmaße werden dabei nicht nur zu einigen wenigen Zeitpunkten, sondern kontinuierlich aufgenommen. Die Simulation wird dabei auch nicht wiederholt begonnen, sondern nur einmalig durchgeführt, wobei die Anzahl der Simulationszeitschritte ebenfalls nicht beschränkt ist. Diese Art der Simulation bedingt, dass Petri-Netze die analysiert werden sollen, zyklisch aufgebaut sein sollten[62].
Wenn bestimmte Transitionen nur sehr selten oder einmalig schaltfähig sind, dies aber in ein Leistungsmaß eingeht, kann die geforderte Genauigkeit nur mit sehr langer Simulationszeit erkauft werden. Dies wird näher in 4.3 behandelt.

4.2.1 Präzisionssteuerung

Das Festlegen, mit welcher Präzision eine Simulation durchgeführt wird, bedeutet im stationären Fall, die Abbruchbedingung der Simulation an die erreichte Ergebnisgüte zu koppeln. D.h. der Nutzer oder die künftige Optimierungssteuerung legt im Fall einer stationären SCPN-Simulation das geforderte **Konfidenzniveau** und den **maximalen relativen Fehler** fest.
Ein kleinerer maximaler relativer Fehler hat demnach zur Folge, dass mehr CPU-Zeit für die Simulation benötigt wird. Gleiches gilt für die Erhöhung des geforderten Konfidenzniveaus. Ein wesentlicher Grund dafür ist der Aufbau und die Art der Simulation von Petri-Netzen. Typischerweise werden zyklische Vorgänge simuliert, die nach endlicher Zeit ein Einschwingverhalten zeigen. Folglich ist die Zeit in Simulationsschritten umso höher, je kleiner der Bereich ist, innerhalb dem die Messwerte/Leistungsmaße eingeschwungen sein sollten. Die Anzahl der Simulationsschritte sowie die Berechnung der Leistungsmaße bedingt dabei jedoch mehr CPU-Zeit für die Simulation.
Das geforderte Konfidenzniveau und der maximale relative Fehler von Leistungsmaßen stehen in direktem Zusammenhang mit der notwendigen CPU-Zeit für die Simulation eines SCPN. Je höher das Konfidenzniveau und je kleiner der maximal erlaubte relative Fehler der Leistungsmaße, desto mehr Rechenzeit ist für eine Simulation notwendig.
Um dies zu zeigen, wurden verschiedene Versuche durchgeführt. Ausgangspunkt war das erwähnte SCPN in Bild 4.1 mit dem Leistungsmaß aus Formel 4.1. Das geforderte Konfidenzniveau wurde von 85 % (dem minimal einstellbaren Wert im verwendeten Simulationstool) bis 99 % erhöht und der maximale relative Fehler von 15 % bis auf 1 % reduziert. Insgesamt ergaben sich dadurch 225 Simulationsläufe. Um die Vergleichbarkeit zu gewährleisten, wurden alle Simulationen auf dem gleichen PC und mit dem gleichen Seed für den Zufallsgenerator durchgeführt. Das Ergebnis ist in Bild 4.3 zu sehen.
Das Bild, wie auch eine weitergehende Analyse zeigten keinen signifikanten Zusammenhang zwischen Konfidenzniveau, maximalem relativen Fehler und benötigter CPU-Zeit. Mit Ausnahme im Grenzbereich von minimalen Werten für relativen Fehler bei maximalem Konfidenzniveau. Auch die wiederholte Durchführung des Versuchs zeigte (abgesehen von zwei Ausreißern) mit 12,5 % maximalem Unterschied

an benötigter CPU-Zeit keinen klaren Zusammenhang. Dies widerspricht jedoch den Erfahrungen im bisherigen Einsatz des Simulationstools TimeNET. Daher wurde der gleiche Versuch mit komplexeren SCPNs wiederholt. Dabei handelte es sich um das Netz zur Modellierung von Energieflüssen im Haushalt aus Kapitel 7. Es zeigte sich ein vergleichsweise einfacher Zusammenhang von Konfidenzniveau, maximalem relativen Fehler und benötigter CPU-Zeit, wie in Bild 4.5 zu sehen ist. Anschaulich auch hier der Zusammenhang mit der benötigten Anzahl an Simulationszeitschritten (Bild 4.6).
Verallgemeinert kann dieser Zusammenhang mit Formel 4.2 nachgebildet werden. Ein einfaches Polynom, welches die gezeigten CPU-Zeit-Messwerte mit minimaler Abweichung abschätzt. Die Approximation des CPU-Zeit-Bedarfs war notwendig, um genauigkeitssteuernde Optimierungsalgorithmen anhand von Benchmarkfunktionen entwickeln und testen zu können (siehe Kapitel 8).

$$CPUTime = \left(\frac{cc * me}{a}\right)^3 \cdot b + c \tag{4.2}$$

Zur Erläuterung der Formel:

cc bezeichnet das gewählte Konfidenzniveau, in TimeNET auch **Confidence Interval** genannt.

me gibt den maximalen relativen Fehler (max. rel. error) an.

Die verwendeten Koeffizienten sind
$a = 49.74$, $b = 23.91$, $c = 154.294$. Die verwendeten Koeffizienten ergeben Werte zwischen 155 und 2367 Sekunden für die geschätzte CPU-Zeit im Vergleich zu gemessenen Werten von 114 bis 1990. Das Verhältnis zwischen verlangter Simulationsgenauigkeit und benötigter Zeit ist damit ausreichend gut darstellbar, was eine Bewertung von Heuristiken hinsichtlich ihres Zeitbedarfs bei simulationsbasierter Optimierung von SCPNs auch anhand von Benchmarkfunktionen erlaubt.
Neben der CPU-Zeit, welche für eine Simulation benötigt wird, scheint die benötigte Anzahl an Simulationszeitschritten in sehr ähnlicher Weise von der geforderten Simulationsgenauigkeit abhängig zu sein. Bild 4.4 zeigt diesen Zusammenhang für das Beispiel-SCPN.

Mit veränderten Parametern lässt sich somit die gleiche Formel nutzen, um den Bedarf an Simulationszeitschritten abzuschätzen. Für eine beschleunigte Simulation und Optimierung ist die Abschätzung der benötigten Simulationszeitschritte jedoch nicht relevant und wird hier nur exemplarisch gezeigt.

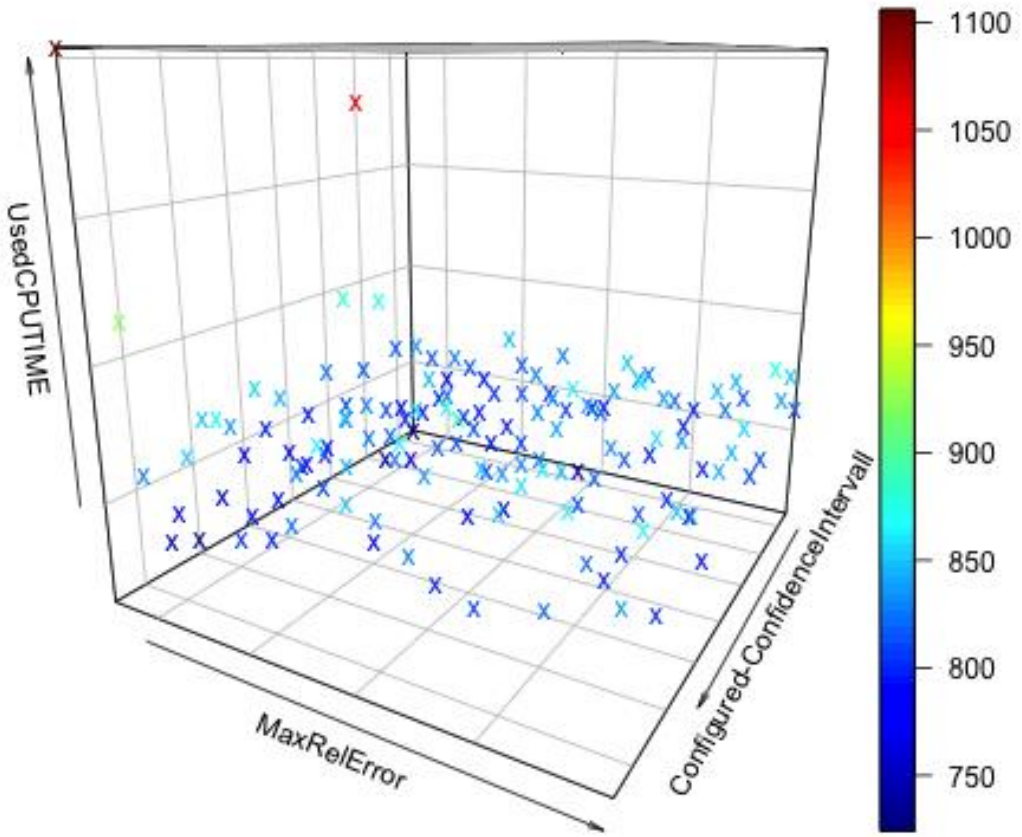


Abbildung 4.3: CPU-Zeitbedarf im Verhältnis zu maximalem relativen Fehler und Konfidenzniveau (einfaches SCPN)

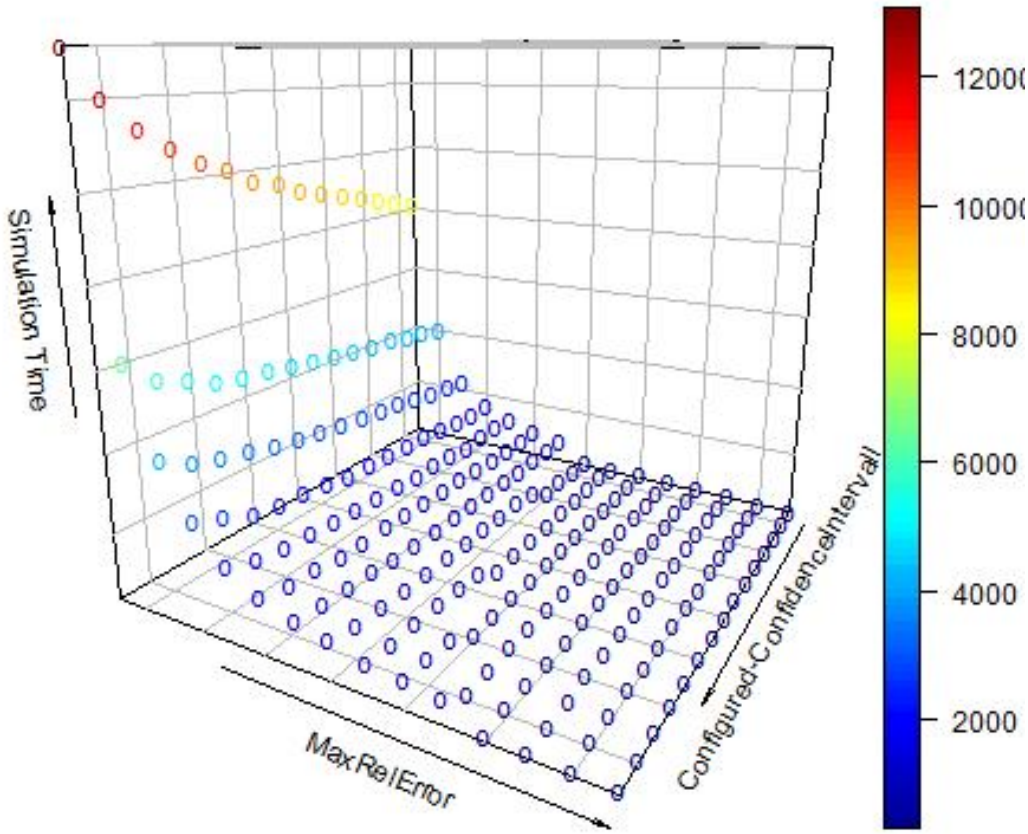


Abbildung 4.4: Benötigte Simulationszeitschritte im Verhältnis zu maximalem relativen Fehler und Konfidenzniveau (einfaches SCPN)

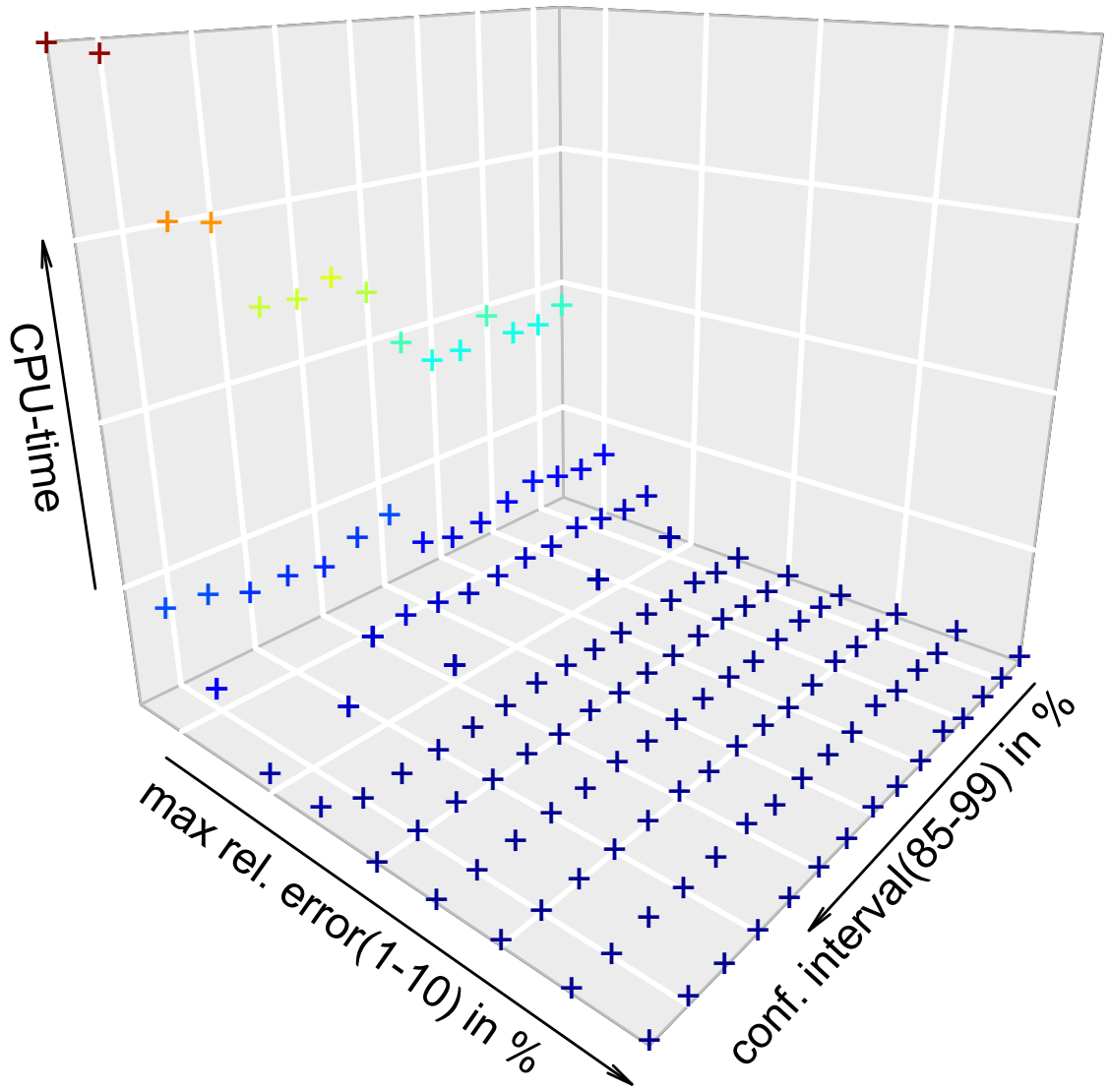


Abbildung 4.5: CPU-Zeitbedarf (Sekunden) im Verhältnis zum maximalen relativen Fehler und Konfidenzniveau (komplexes SCPN / Energiemodell)

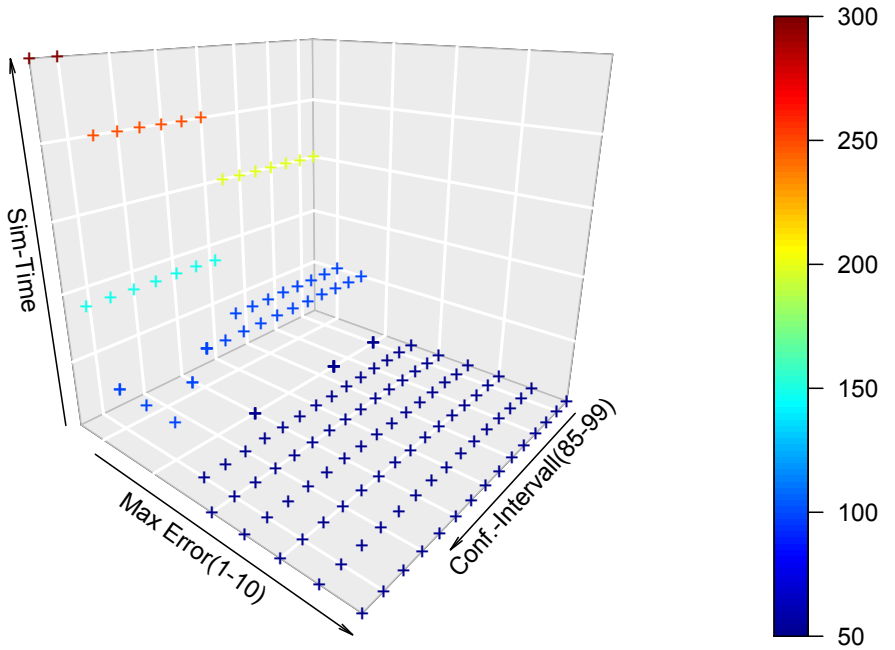


Abbildung 4.6: Benötigte Simulationszeitschritte im Verhältnis zum maximalen relativen Fehler und Konfidenzniveau (komplexes SCPN / Energiemodell)

4.2.2 Abbruchkriterien für die Simulation

Zusätzlich zur minimal erreichten Güte der Leistungsmaße können weitere Abbruchbedingungen für eine Simulation festgelegt werden. Dies ist einerseits die verwendete **CPU-Zeit** (Max. Running Time) des Simulators und anderseits die maximale **Anzahl an Simulationszeitschritten** (Max Model Time). Der Zusammenhang zwischen beidem ist stark abhängig vom Netz, von den verwendeten Leistungsmaßen und vielen weiteren Bedingungen.
Grundsätzlich sollte immer die verwendete CPU-Zeit beschränkt werden. Die Beschränkung der Simulationszeitschritte wird im transienten Fall immer vorgenommen und ist für die stationäre Simulation nicht zwingend notwendig. Gleichwohl dient es als zusätzliche Absicherung, falls die Simulation niemals oder erst sehr spät konvergiert.
Das Beenden der Simulation durch eine dieser Bedingungen bedeutet in der Regel, dass mindestens ein Leistungsmaß noch nicht das geforderte Konfidenzniveau erreicht hat. Anhand der Simulationsergebnisse muss anschließend entschieden werden, inwiefern die Berechnung des Leistungsmaßes geändert werden kann, um ein Konvergieren zu ermöglichen, oder ob die Simulation mit veränderten Abbruchbedingungen wiederholt werden muss.
Während der simulationsbasierten Optimierung dienen diese Parameter bzw. Beschränkungen lediglich als Sicherungsmaßnahme, um ein Terminieren der Simulation und später des Optimierungsalgorithmus sicherzustellen. Dabei besteht stets die Gefahr, durch vorzeitigen Abbruch einzelner Simulationen wichtige Messwerte bzw. das globale Optimum zu überspringen.

4.2.3 Weitere externe Einflüsse

Unabhängig von den genannten Einflüssen und Beschränkungen gibt es noch diverse Faktoren, welche die benötigte Zeit pro Simulationslauf beeinflussen. Diese sollen hier kurz erläutert werden.

Grafische Ergebnisdarstellung

TimeNET bietet die Möglichkeit, diverse Informationen während der Simulation grafisch darzustellen. Im Hintergrund werden dabei sämtliche darzustellende Daten

über einen Netzwerksocket (zumeist lokal) an eine weitere Java-GUI übertragen. Es können dabei nicht nur die aktuellen Werte der Leistungsmaße, sondern auch Markenanzahl von Plätzen, Schaltvorgänge von Transitionen und andere Daten übertragen werden.
Die Datenmenge kann, abhängig von verschiedenen Faktoren, schnell viele Megabytes betragen. Da die Daten innerhalb der Java-GUI permanent vorgehalten werden müssen, steigt der Speicherbedarf der JVM stark an. Es kann auch dazu führen, dass die JVM und damit TimeNET beendet wird. Die Simulation wird dennoch weiter ausgeführt, jedoch ist der Status des Simulators dann gänzlich unbekannt. Typischerweise muss die Simulation über einen Taskmanager abgebrochen werden.
Während der simulationsbasierten Optimierung ist die grafische Darstellung einzelner Leistungsmaße oder Schaltereignisse ohnehin irrelevant und kann abgeschaltet werden. Dies muss jedoch vor der Optimierung in TimeNET geschehen.

Initialisierung des Zufallsgenerators

TimeNET verwendet intern einen Pseudozufallsgenerator, dessen Initialwert (auch Seed genannt) vom Nutzer vorgegeben werden kann. Alternativ wird jeweils die aktuelle Systemzeit in ms zur Initialisierung verwendet.
Bei gleichem Seed berechnet der Zufallsgenerator stets exakt die gleiche Folge von Zahlen, was in exakt gleichen Simulationsergebnissen resultiert. Aus diesem Grund sollten Simulationen immer mehrfach mit unterschiedlichen Seed-Werten durchgeführt werden.
Unabhängig von den Simulationsergebnissen für Leistungsmaße hat die Initialisierung des Zufallsgenerators einen großen Einfluss auf die benötigte CPU-Zeit und die berechneten Simulationszeitschritte. Abbildung 4.7 zeigt den Einfluss des Seed-Wertes auf die benötigte CPU-Zeit. Der Seed kann beliebige Integerwerte annehmen, aber schon bei Untersuchung der ersten 256 Werte ist das Problem ersichtlich. Die CPU-Zeit liegt im Minimum bei 24 Sekunden und im Maximum bei 83, also dem 3,4-fachen. Die Anzahl der berechneten Simulationszeitschritte liegt zwischen 750 und 2700. Auch die berechneten Werte der Leistungsmaße variieren, jedoch in allen Untersuchungen nur maximal 35 %, was bei einem zufallsbehaftetem Prozess nicht ungewöhnlich ist.
Dieses Verhalten sollte bedacht werden, wenn Simulationen und Optimierungen

durchgeführt werden. Es ist ratsam, jedes Netz zunächst mit verschiedenen Seed-Werten hinsichtlich der benötigten Zeit für eine Simulation zu untersuchen und ggf. die Optimierung mit verschiedenen Seed-Werten durchzuführen.

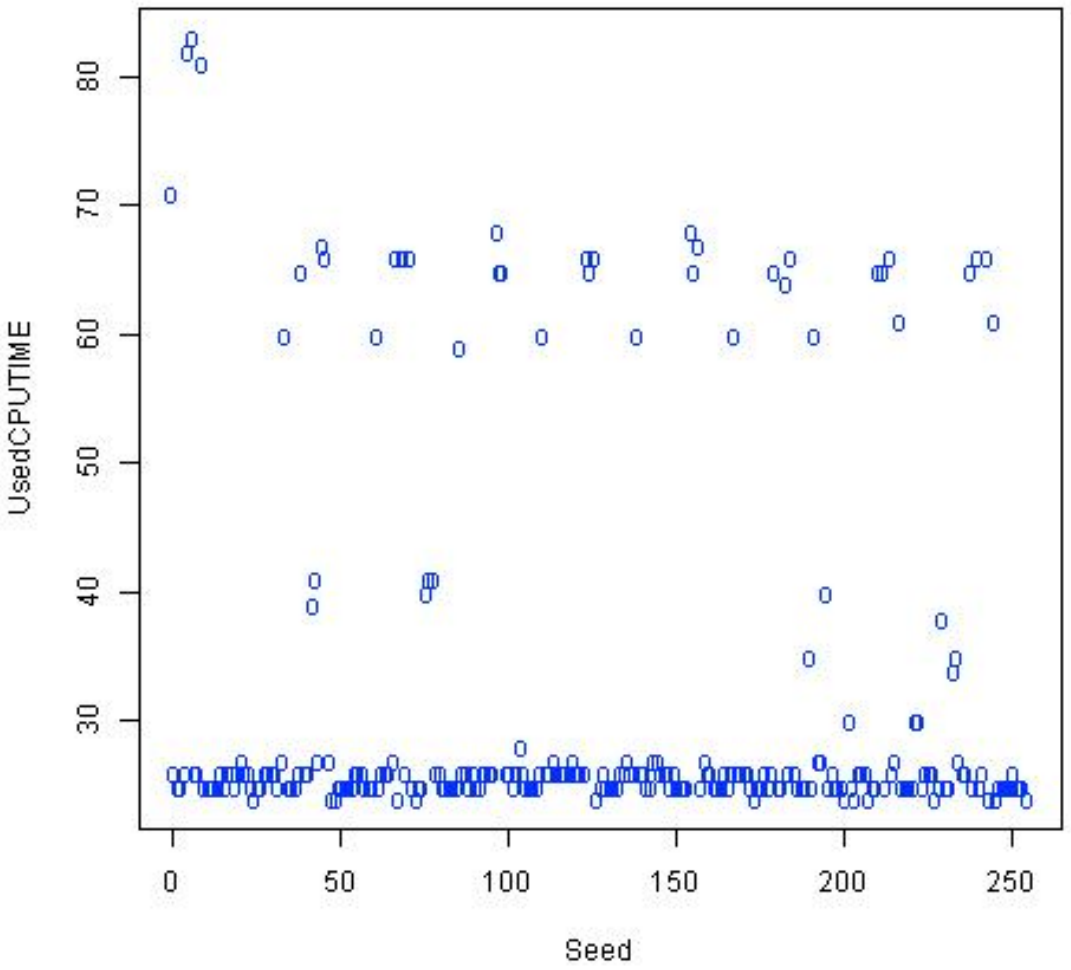


Abbildung 4.7: Benötigte CPU-Zeit in Abhängigkeit vom Seed (0..255)

Verwendeter Simulationsrechner

Die verwendete Hardware zur Durchführung von Simulationen hat einen untergeordneten Einfluss auf die Simulationszeit. TimeNET wurde entwickelt, um komplexe Simulationen auf relativ leistungsschwachen Computern der 1990er Jahre durchführen zu können. Anders als viele alternative Simulationswerkzeuge generiert TimeNET für jede Simulation den C-Quelltext für ein separates Simulationsprogramm. Dieser wird mit einem installierten Compiler für die verwendete Hardware übersetzt und dabei entsprechend optimiert. Das resultierende Programm sendet die Ergebnisse über einen Netzwerksocket an ein grafisches Anzeigefenster und speichert die Ergebnisse in log-Dateien.
Durch diese Vorgehensweise können Eigenheiten und insb. Vorteile der verwendeten Hardware ausgenutzt werden. Wenngleich aktuelle Versionen von TimeNET noch keine Multiprozessorunterstützung haben, nutzen die entstehenden Simulationsprogramme die verwendete Hardware optimal, sofern der installierte Compiler dies unterstützt. Die Gesamtlaufzeit inkl. Übersetzung der C-Quelltexte ist hingegen stark beeinflusst vom verwendeten Betriebssystem und dessen Auslastung mit anderen Tasks.
Alle Versuche zur Bewertung der Optimierungsheuristiken (Kapitel 8) in dieser Arbeit wurden auf virtuellen Maschinen durchgeführt, welche die gleiche Konfiguration aufweisen:

- **OS** Ubuntu 12.04 LTS
- **RAM** 4 Gb
- **HDD** 40 Gb
- **Prozessor** Pentium 4, Single Core, 2.4 GHz, 64 Bit

Tabelle 4.1 zeigt die Simulationsdauer in Abhängigkeit von System und Auslastung anhand einiger beispielhafter Computer.
Leerlauf bedeutet, dass der Computer keine weiteren rechenintensiven Tasks parallel erledigt. Im Zustand **Ausgelastet** wird der Simulationsrechner vor und während der Simulation mit niederprioren Aufgaben ausgelastet[1]. Dabei wird ersichtlich, wie

[1] Zur Auslastung wurde prime95, ein Programm für die Suche nach Mersenne-Primzahlen, im „Blend“-Modus genutzt.

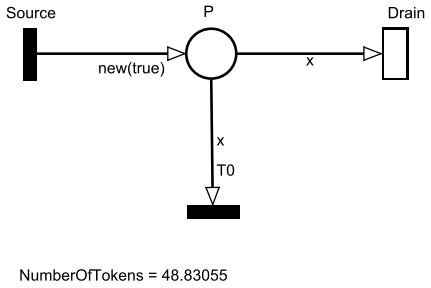
groß der Einfluss des Übersetzungsvorgangs auf die gesamte Simulationszeit ist. Die Tabelle soll dem Leser einen kurzen Überblick und Anstöße für die Wahl eines Simulationsrechners geben. Bei der verteilten Simulation in einer heterogenen Rechnerumgebung müssen die teilweise sehr großen Unterschiede der Rechenleistung beachtet werden. Demzufolge liefern die Angaben zur Simulationszeit in diese Fall keine verlässlichen Vergleichsdaten.
Die Betrachtung der benötigten Simulationszeit bei ausgelastetem Rechner kann für bestimmte Situationen wichtige Hinweise zur Einschätzung der Modellkomplexität oder Leistungsfähigkeit des Simulationsrechners liefern. Auch kann es aufgrund der Architektur von TimeNET vorkommen, dass Simulationen niemals beendet werden, auch wenn keine Ergebnisdaten angezeigt werden oder gar das Hauptprogramm beendet wurde. In diesem Fall muss der Simulationsprozess manuell beendet werden.

Typ	Situation	CPU-Zeit (sec.)	
OSX 10.9.5 2.8 GHz Core i7 8GB RAM	Leerlauf	Übersetzung Simulation	19 46
	Ausgelastet	Übersetzung Simulation	117 195
Ubuntu 12.04 Pentium 4, 2.4 GHz 4 GB RAM	Leerlauf	Übersetzung Simulation	18 22
	Ausgelastet	Übersetzung Simulation	56 44
Windows 7 Atom N270, 1.6 GHz 2 GB RAM	Leerlauf	Übersetzung Simulation	248 94
	Ausgelastet	Übersetzung Simulation	308 124
Windows 7 Core 2, 2.4 GHz 4 GB RAM	Leerlauf	Übersetzung Simulation	88 19
	Ausgelastet	Übersetzung Simulation	82 25
Windows 7 Core 2 Duo, 3 GHz 4 GB RAM 128 GB SSD	Leerlauf	Übersetzung Simulation	28 12
	Ausgelastet	Übersetzung Simulation	41 12

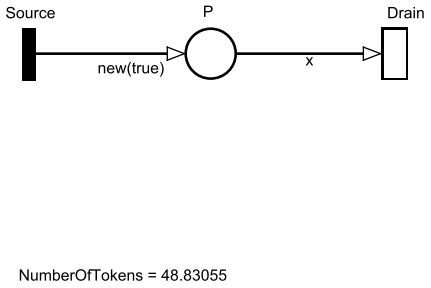
Tabelle 4.1: Art des Simulationsrechners vs. notwendige Simulationszeit

4.3 Interne Einflüsse / Eigenschaften des Netzes

Unabhängig von externen Einflüssen gibt es diverse netzinterne Faktoren, welche die Simulationszeit beeinflussen. Dies wirkt sich folglich auch auf die simulationsbasierte Optimierung von SCPNs aus.
Für die Untersuchung des Einflusses typischer Netzeigenschaften auf die benötigte Simulationszeit wurde das SCPN 4.1 in verschiedener Art und Weise modifiziert. Ziel der Modifikationen war, verschiedene interne Einflüsse isoliert betrachten zu können.
Ein interessanter Aspekt bei der Analyse und Simulation von Petri-Netzen sind **seltene Ereignisse (Rare Events)**. I.d.R. sind damit Schaltvorgänge einzelner Transitionen gemeint, die sehr selten im Vergleich zu anderen Schaltvorgänge sind. Insbesondere seltene Events, die direkten Einfluss auf Leistungsmaße haben, verursachen inakzeptabel lange Simulationen. Dies tritt nahezu unabhängig von der geforderten Simulationsgenauigkeit auf. Der Umgang bzw. die Lösung dieses Problems ist unter dem Namen **Rare Event Simulation** Gegenstand vergangener und aktueller Forschungsvorhaben [68, 35, 51].
Umgehen kann man dieses Problem zum Beispiel, indem man Leistungsmaße so definiert, dass sie nicht mehr direkt von selten schaltenden Transitionen abhängen, sofern dies möglich ist.
Andererseits können auch **sehr kurze Schaltzeiten** von Transitionen eine Ursache für lange Simulationen sein. Zumeist führt dies dazu, dass alle Leistungsmaße niemals konvergieren und die Simulation in einer Art Endlosschleife verharrt. Ein Beispiel ist in Bild 4.8(a) zu sehen. *Source* füllt den Platz P unendlich schnell. Dessen Kapazität ist unbegrenzt. *T0* schaltet auch unendlich schnell, jedoch ist dies an die globale Bedingung $\#P < \$capacity$ gebunden. *Drain* hingegen schaltet mit einer exponential verteilten Verzögerung von 1 Simulationszeiteinheit. Das Leistungsmaß $NumberOfTokens = \#P$ beschreibt lediglich die durchschnittliche Anzahl an Marken in P. Sie entspricht zu fast jedem Zeitpunkt der angegebenen Kapazität von P. Dennoch lässt sich das Netz nicht stationär simulieren.
Im Gegenbeispiel 4.8(b) wird P nur von der zeitabhängig aktivierten Transition *Drain* entleert. Seine Kapazität ist direkt beschränkt und nicht indirekt über eine zeitunabhängige Transition. Schon nach kurzer Simulationszeit konvergiert das Leistungsmaß beim erwarteten Wert von $\$capacity$.



(a) Zeitbasierter und zeitunabhängiger Markenabfluss

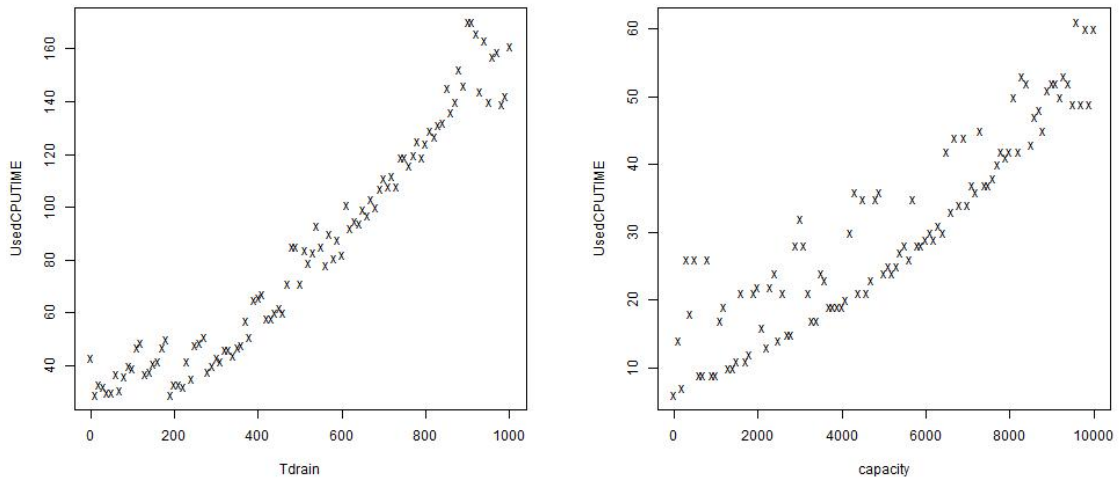


(b) Nur zeitbasierter Markenabfluss

Abbildung 4.8: Modifiziertes Beispielnetz zur Analyse der Ursachen für CPU-Zeit-Bedarf

Der Unterschied zwischen langen und kurzen Schaltzeiten wird in Bild 4.9(a) verdeutlicht. Hierfür wurde das Netz 4.8(b) genutzt und statt der Kapazität von *P* die Schaltzeit von *Drain* schrittweise von $1..\frac{1}{1000}$ reduziert. Im Ergebnis ist zu sehen, dass die simple Verringerung der Schaltzeit zu einer dramatischen Erhöhung der benötigten CPU-Zeit führte. Erklärbar ist dies mit der entsprechend steigenden Zahl an simulierten Events. Das Ergebnis des Leistungsmaßes ist derweil stets gleich, denn *P* wird unendlich schnell durch *Source* gefüllt, unabhängig von *Drain*.
Ein anderer interessanter Faktor für den Bedarf an Ressourcen (CPU-Zeit, Speicher) stationärer SCPN-Simulationen ist die **durchschnittliche Anzahl an Marken im Netz**. Mithilfe des Netzes aus 4.8(b) wurde dies untersucht, indem die Kapazität von P schrittweise erhöht wurde. Das Ergebnis ist in Bild 4.9(b) zu sehen. Mit steigender Zahl durchschnittlich existierender Marken in P bzw. im gesamten Netz steigt auch erkennbar die benötigte Zeit für eine Simulation. Ein Grund hierfür ist der benötigte Arbeitsspeicher der Simulation, welcher u.a. stark von der durchschnittlichen Markenanzahl abhängt.
Damit zusammen hängt auch die Art der zeitbehafteten Transitionen bzw. ihres sogenannten ServerTypes. Es stehen zwei Möglichkeiten zur Verfügung, deren Funktionsweise für die Modellierung wie auch für die Simulation von großer Bedeutung sind. Wenn eine Transition vom Typ **Exclusive Server** ist, wird, sobald die Vorbedingungen erfüllt sind, für eine Marke[2] aus dem jeweiligen Vorplatz ein Timer

[2]Die Wahl bzw. Reihenfolge der Marken kann im Modell je Platz festgelegt werden. FIFO, LIFO oder zufällige Auswahl sind möglich.



(a) Schaltzeit vs. benötigte CPU-Zeit (b) Markenanzahl im Netz vs. CPU-Zeit

Abbildung 4.9: Einfluss von Markenanzahl und Timing einzelner Transitionen vs. CPU-Zeit

gestartet. Nach Ablauf der definierten Zeit (unter Beachtung der gewählten Zeitfunktion) schaltet die Transition und die Marke wird konsumiert. Anders ist der Ablauf beim Typ **Infinite Server**. In diesem Fall wird für jede Marke, welche die Transition schaltfähig werden ließe, ein Timer gemäß der festgelegten Zeitfunktion gestartet. Jeder abgelaufene Timer führt zum Schalten der Transition. Dieses Verhalten ist in einigen Fällen erforderlich, führt jedoch zu einem großen Ressourcenbedarf, insbesondere wenn viele Marken im Netz sind.
Sofern die Logik des Modells bzw. des Systems es nicht zwingend erfordert, sollte der Einsatz von Infinite Server Transitionen vermieden werden um Ressourcen (insb. Zeit) zu sparen.
Eine weitere offensichtliche Ursache bzw. ein Einflussfaktor für den Ressourcenbedarf einer Simulation ist die **Anzahl an Modellelementen** und somit die Komplexität des Netzes. Dies gilt nicht nur für die Simulation von SCPNs. Verschiedene Forschungsvorhaben beschäftigten sich bereits in der Vergangenheit mit der Reduktion der Komplexität von Petri-Netzen. Ein Ansatz ist die Aggregation von Teilnetzen, wobei die Anzahl an Modellelementen sowie der Zustandsraum reduziert wird, das Verhalten aber exakt gleich bleibt. Entsprechende Verfahren setzen allerdings Petri-Netze mit speziellen Eigenschaften wie Symmetrie voraus. Alternativ können Teil-

netze durch vereinfachte Varianten ersetzt werden, deren äußeres Verhalten gleich oder ähnlich dem Original ist. Dies ist auch Teil des Forschungsansatzes zur adaptiven Genauigkeitssteuerung wie in [69] bereits vorgestellt.
Die Untersuchung und Reduktion des Zeitbedarfs von Simulationen war und ist Gegenstand verschiedener Forschungsprojekte. So lassen sich die Leistungsmaße von stochastischen Petri-Netzen auch durch numerische Analyse bestimmen, sofern der Zustandsraum (nicht Definitionsraum der Optimierung) klein genug ist. Ist der Zustandsraum zu groß, können die Leistungsmaße nur noch per Simulation in sinnvoller Zeit ermittelt werden. Dem Anwender die Entscheidung zwischen Simulation und statischer, numerischer Analyse zu erleichtern bzw. überflüssig zu machen, ist Ziel aktueller Forschung [71].
Obwohl ohne detaillierte Analyse des Petri-Netzes eine a priori Berechnung des Zeitbedarfs nicht möglich ist, so gibt es dennoch Verfahren, während einer Simulation die verbleibende Anzahl Simulationsschritte bis zum Erreichen der geforderten Konvergenzkriterien abzuschätzen[33]. Für den Einsatz in der simulationsbasierten Optimierung ist dies relevant, sobald eine enge Koppelung von Optimierungs- und Simulationssystem gelingt. Zum Zeitpunkt dieser Arbeit ist dies jedoch noch nicht der Fall.

4.4 Best practices für minimale Simulationsdauer

An dieser Stelle sollen einige Hinweise gegeben werden, um die Simulationszeit von SCPNs zu minimieren. Einerseits soll dies helfen, die simulationsbasierte Optimierung zeiteffizienter zu nutzen, andererseits sollen die Hinweise künftigen Anwendern bei der Suche nach Ursachen für ungewöhnlich lange Simulationen helfen.

Lebendigkeit Für eine stationäre Simulation muss ein SCPN zwingend zyklisch aufgebaut und lebendig sein, denn es muss so lange simuliert werden, bis alle Leistungsmaße konvergieren bzw. ihre Werte innerhalb der verlangten Grenzen von Konfidenzniveau und maximalem relativen Fehler liegen. Ist dies nicht möglich, kann das Netz dennoch transient simuliert werden. Mit dem derzeitigen Stand des entwickelten Optimierungs-Werkzeuges (Siehe Kapitel 6) ist damit jedoch keine Optimierung möglich.

Markenanzahl Die Anzahl der durchschnittlich existierenden Marken innerhalb eines Netzes wirkt sich indirekt auf die benötigte Zeit für die Berechnung eines Simulationszeitschrittes aus. Je mehr Marken gleichzeitig im Modell existieren, desto länger dauert somit die gesamte Simulation. Schon beim Entwurf eines Modells sollte dies berücksichtigt werden.

Seltene Ereignisse Da alle Leistungsmaße für eine stationäre Simulation konvergieren müssen, sollte unbedingt vermieden werden, dass Leistungsmaßberechnungen allein auf seltenen Ereignissen basieren, wenn nicht unbedingt notwendig. In der Praxis bedeutet dies, Leistungsmaße zu vermeiden, deren Ergebnis nahe 0 erwartet wird.

Anzahl der Leistungsmaße Je mehr Leistungsmaße definiert sind, desto wahrscheinlicher ist es, dass eines nicht konvergiert. Aus diesem Grund sollten stets noch weitere Abbruchbedingungen für eine Simulation verwendet werden.

Unterschiedliche Transitionsschaltzeiten Die Seltenheit von Ereignissen bezieht sich stets auf die relative Häufigkeit anderer untersuchter Ereignisse eines Netzes. Es sollte daher vermieden werden, dass einzelne Transitionen Schaltzeiten aufweisen, die stark von allen anderen abweichen, insbesondere wenn diese alleinig zur Berechnung eines Leistungsmaßes herangezogen werden. Ein Extremfall hierfür ist Mischung von zeitbehafteten und zeitlosen Transitionen.

Abbruchkriterien Auch auf die Gefahr hin, dass ein kompletter Optimierungslauf fehl schlägt, sollte immer das Abbruchkriterium maximaler CPU-Zeit festgelegt werden. Selbst wenn die Simulation nicht erfolgreich ist, lässt sich somit dennoch nach endlicher Zeit der Grund dafür (zumeist ein nicht konvergiertes Leistungsmaß) untersuchen.

Simulationsgenauigkeit Konfidenzniveau und maximaler relativer Fehler haben nur im Grenzbereich (99 %, 1 %) signifikanten Einfluss auf die Genauigkeit vieler Simulationen. Für erste Ergebnisse, insbesondere bei Netzen, die sich noch in der Entwicklung befinden, ist ein Konfidenzniveau von 90 % bei maximalen relativen Fehler von 5 % vollkommen ausreichend.

Zufällige Einflüsse Da der Einfluss des gewählten Seed-Wertes auf die benötigte

CPU-Zeit nicht zuverlässig vorhersagbar ist, sollte jedes Netz mit verschiedenen Seed-Werten simuliert werden, um diesen Einfluss ausschließen zu können.

Anzahl der Leistungsmaße Da eine stationäre Simulation erst beendet ist, wenn alle Leistungsmaße konvergiert sind bzw. ihre Werte innerhalb der geforderten Genauigkeitsgrenzen liegen, sollten alle Leistungsmaße, die nicht zwingend benötigt werden, entfernt werden. Insbesondere bei der simulationsbasierten Optimierung mit sehr vielen Simulationsläufen ist dies ein entscheidender Faktor.

Berechenbarkeit von Leistungsmaßen Leistungsmaße können unter Umständen bei ihrer Evaluation zu Divisionen durch 0 führen. Dies wird zum Zeitpunkt des Simulationsstartes meist nicht erkannt und führt während der Simulation zu unvorhergesehenem Verhalten bzw. Endlosschleifen. Abhilfe kann hier durch Hinzufügen von Konstanten zur Leistungsmaßberechnung geschaffen werden.

Grafische Darstellung Für Testzwecke ist die grafische Darstellung zwar sehr hilfreich. Während der Optimierung ist diese jedoch überflüssig und kostet unnötig CPU-Zeit.

Logging Beim logging von SCPN-Simulationen kann jeder Schaltvorgang und viele weitere Informationen in entsprechende Log-Dateien geschrieben werden. Für erste Untersuchungen von SCPNs ist dies praktisch, ähnlich wie die grafische Darstellung. Jedoch benötigt dies ebenfalls –im Zusammenhang mit den genannten anderen Faktoren– viel Ressourcen. Daher sollte während der Optimierung auf ein logging der einzelnen Simulationsläufe verzichtet werden.

5 Untersuchte Strategien zur Effizienzsteigerung

In diesem Kapitel werden die entwickelten Strategien vorgestellt, welche im Rahmen des prototypischen Software-Werkzeugs TOE umgesetzt wurden. In Kombination mit den implementierten Heuristiken werden diese später auf Benchmark-Funktionen sowie reale SCPN-Simulationen angewendet und deren Ergebnisse im Kapitel 8 diskutiert.

5.1 Teile und Herrsche

Der verbreitetste und offensichtlichste Ansatz ist es, den Definitionsraum aufzuteilen und somit die Anzahl der maximal notwendigen Simulationen zu reduzieren. Bei stetigen Antwortfunktionsgebirgen ohne lokale Optima sind derartige Strategien sehr erfolgreich. Benchmarks wie die Sphere- und Matya-Funktion (Bild 8.1) zeigen ein entsprechendes Verhalten, aber auch viele Leistungsmaße von SCPNs ähneln dem.
Auf verschiedene Arten lässt sich dieser Ansatz umsetzen und ist in mehr oder weniger ausgeprägter Form in vielen Heuristiken zu finden. So kann man den gesamten Definitionsbereich in Teilmengen zerlegen und diese anhand der Simulation einiger Vertreter für die weitere Suche ein- oder ausschließen. Auch Simulated Annealing setzt dieses Prinzip teilweise um, indem das theoretisch erreichbare Gebiet im Definitionsraum einer Abkühlungsfunktion folgend stetig reduziert wird(vgl. 2.5). Auch wird es als einer der Grundgedanken von Simulated Annealing von Kirkpatrick bereits in den ersten Veröffentlichungen hierzu genannt [64].
Populationsbasierte Heuristiken nutzen dieses Prinzip auf andere Weise. Es werden stets mehrere Parametersätze ausgewertet. Somit fließt das Wissen über mehrere Teilgebiete des Definitionsraumes in die Berechnung eines oder mehrerer weiterer

Parametersätze ein.
Andere Heuristiken machen sich den Teile und Herrsche-Ansatz ebenfalls zu Nutze. Es handelt sich um sogenannte Meta-Heuristiken, die verschiedene bekannte Strategien nacheinander anwenden, um zunächst den Definitionsraum grob nach interessanten Gebieten abzusuchen. Anschließend wird mit einer lokal arbeitenden Heuristik wie Hill-Climbing das eingegrenzte Gebiet genauer untersucht. Dadurch wird das Risiko reduziert, dass der Algorithmus auf einem lokalen Optimum konvergiert und gleichzeitig wird durch Ausschluss großer uninteressanter Gebiete des Definitionsraumes die Gesamtkomplexität reduziert. Derartige Ansätze wurden u.a. von A. Zimmermann und M. Syrjakow vorgestellt [59, 73, 25].
In dem umgesetzten und verwendeten Optimierungswerkzeug wird das Paradigma von Teile und Herrsche an verschiedenen Stellen deutlich. Simulated Annealing und populationsbasierte Heuristiken nutzen das Prinzip ohnehin auf die beschriebenen Arten aus. Hinzu kommt noch die Implementierung des zwei-phasigen Optimierungsansatzes nach [73]. Mit der Entwicklung der frei konfigurierbaren mehrphasigen Optimierung (siehe Kap. 6.6.5) können nun aber auch einfache Heuristiken wie Hill-Climbing von den Vorteilen dieses Prinzips profitieren [7].

5.2 Caching

Da Simulationen, abhängig von verschiedenen Faktoren, sehr viel Zeit in Anspruch nehmen können, ist die Vermeidung mehrfacher Simulationen einer der wichtigsten Punkte bei der Entwicklung von Optimierungsalgorithmen. Praktisch alle derzeit bekannten Softwaresysteme und theoretischen Ansätze zur simulationsbasierten Optimierung beinhalten daher solche Mechanismen. Um die mehrfache Simulation von Modellen bzw. Parametersätzen zu vermeiden, muss grundsätzlich jede Konfiguration aus Modell und Parametersatz eindeutig identifizierbar sein. Ist dies gegeben, kann während der Optimierung eine Datenbank gefüllt werden, aus der mehrfach angeforderte Simulationsergebnisse ausgelesen werden.
Interessant ist dabei die Betrachtung der Wahrscheinlichkeit, dass ein angefordertes Simulationsergebnis bereits im Cache vorliegt. Dies ist abhängig von der verwendeten Heuristik und der minimalen Diskretisierung des Definitionsbereichs. Je mehr zufällige Einflüsse die Berechnung des jeweils nächsten Parametersatzes beeinflus-

sen und je feiner die Diskretisierung gewählt wird, desto mehr Simulationen müssen tatsächlich durchgeführt werden. In der statistischen Auswertung der Versuche wird dies als Cache-Ratio angegeben.
Zwar sollte ein Optimierungsalgorithmus von sich aus nicht mehrfach die gleiche Simulation starten, jedoch ist dies bei zufallsbehafteten Heuristiken nicht immer vermeidbar. Dass der Einsatz eines entsprechenden Cache bzw. einer Ergebnisdatenbank für Verfahren wie Simulated Annealing großen Nutzen bringt, wurde unter anderem bereits 1999 nachgewiesen. Im konkreten Fall ergab sich bei einer Trefferrate von bis zu 90 % eine Verringerung der Optimierungszeit um 13 % [72].

5.2.1 Cache als Basisdaten

Für die Untersuchung und den Vergleich verschiedener Optimierungsheuristiken kann es sinnvoll sein, ausschließlich Simulationsdaten zu nutzen, die bereits in einer Datenbank bzw. im Cache vorliegen. Das bietet verschiedene Vorteile. Zum einen ist das absolut erreichbare Optimum bekannt, da es mit dem Auslesen der Datenbank bestimmt werden kann. Zum anderen haben zufällige Effekte im Simulationssystem keine Auswirkungen auf das Ergebnis. Denn die wiederholte Simulation eines Modells mit exakt gleichen Parametern kann bei SCPNs zu unterschiedlichen Ergebnissen führen. Außerdem kann die Funktionsweise der Heuristiken auf diese Art sehr schnell überprüft werden, da die Datenbankabfrage nur einen Bruchteil der Zeit einer vollständigen Simulation benötigt.
Neben dem Aufbau einer Ergebnisdatenbasis zur Laufzeit der Optimierung wurde, wie auch bei anderen Systemen üblich, wurde daher auch die Erzeugung einer solchen Datenbasis unabhängig von Optimierungen umgesetzt und untersucht. Sofern entsprechende Rechenkapzität zur Verfügung steht, kann vor einer Optimierung bereits eine entsprechende Menge an Parametersätzen simuliert werden. Die Ergebnisse werden gespeichert und können zu einem späteren Zeitpunkt für verschiedene Zwecke genutzt werden.

5.2.2 Chancen und Risiken vorausberechneter Simulationsergebnisse

Aus der Verwendung vorausberechneter Simulationsergebnisse ergeben sich verschiedene Vorteile, welche sich auf die Gesamtlaufzeit der Optimierung auswirken.

Antwortfunktionsplot / Heuristikauswahl Die gesammelten Ergebnisse können mit verschiedenen Softwarewerkzeugen grafisch dargestellt werden. Das erleichtert die Auswahl einer an das Problem angepassten Heuristik enorm. Insbesondere da viele Leistungsmaße von SCPNs sehr einfache stetige Antwortfunktionen liefern, kann in für einige SCPNs ganz auf eine computergestützte Optimierung verzichtet werden.

Beschleunigung jeder Heuristik Im Prinzip profitiert jede verwendete Heuristik von der Verwendung vorausberechneter Simulationsergebnisse. Die Beschleunigung hängt dabei stark von der verwendeten Diskretisierung der Parameter während der Optimierung ab. Ist diese gleich oder ein Vielfaches derjenigen aus der Datenbasis, so ist die Trefferwahrscheinlichkeit vielfach höher als bei wesentlich kleinerer Diskretisierung oder gar der Verwendung von Fließkommawerten für Parameter.

Festlegung von Startwerten Ein wichtiger Punkt bei der simulationsbasierten Optimierung durch Parametervariation ist die passende Wahl von Startwerten für die jeweiligen Parameter. Wenn möglich sollte eine Optimierung mehrfach mit zufällig gewählten Startwerten durchgeführt werden, um aussagekräftige Daten über die Qualität der verwendeten Heuristik und die Lage des globalen Optimums zu generieren. Ist die wiederholte Ausführung nicht möglich, erlaubt die grobe Kenntnis des Antwortfunktionsgebirges die Auswahl passender Startwerte, um mit großer Wahrscheinlichkeit das Optimum im ersten Versuch zu finden.

Beurteilung der Optimierungsergebnisse Heuristiken finden nicht zwingend das absolute Optimum. Deshalb ist eine Bewertung der gefunden Optima hilfreich. Wenn Heuristiken auf Benchmark-Funktionen angewendet werden, ist dies sehr einfach zu realisieren, da das absolute Optimum bekannt ist. Bei der simulationsbasierten Optimierung ist das nicht der Fall. Das tatsächliche

Optimum ist nicht bekannt und somit die Beurteilung der Optimierungsergebnisse erschwert. Mit Hilfe vorausberechneter Simulationsergebnisse ist bereits ein –scheinbar– absolutes Optimum bekannt und erlaubt so die Beurteilung der Ergebnisse.

Neben vielen Vorteilen bestehen auch gewisse Risiken. Wird der Cache mit einer sehr groben Diskretisierung des Definitionsbereichs erstellt, so besteht die Gefahr, dass beim Plot des Antwortfunktionsgebirges lokale Optima übersehen werden. Dadurch werden möglicherweise ungünstige Startwerte verwendet oder gar keine Optimierung durchgeführt.

5.2.3 Cache-miss Strategien

Wird ein Parametersatz und damit das entsprechende Simulationsergebnis nicht im Cache gefunden, so kann auf verschiede Art damit umgegangen werden. Im Normalfall wird dann eine reale Simulation des Modells mit diesem Parametersatz durchgeführt und das Ergebnis dem Cache hinzugefügt. Für die schnelle Untersuchung verschiedener Heuristiken und bei Vorhandensein einer entsprechend großen Ergebnisdatenbasis kann die Beschränkung auf bestehende Simulationsergebnisse sinnvoll sein. Dabei wird das Ergebnis des nächstgelegenen Parametersatzes im Cache gesucht und dem Optimierungssystem zurück gegeben, ohne dabei die tatsächlichen Parameterwerte, die zu diesem Ergebnis führten, zu übergeben. Mit dieser Strategie wurden entsprechende Untersuchungen durchgeführt, um Heuristiken anhand von realen Ergebnissen von SCPN-Simulationen zu testen.

5.2.4 Künftige Entwicklung

Im Bereich des Caching von Simulationsergebnissen sind, angepasst an die jeweilige Klasse von Optimierungsproblemen, noch viele Verbesserungen denkbar. So bleibt zu untersuchen, inwieweit ein Simulationsergebnis aus der Datenbank zurück gegeben wird, wenn die internen Modellparameter gleich sind, die Präzisionsparameter aber abweichen. In einigen Bereichen scheint eine Abweichung der Werte für Konfidenzniveau und maximalem relativen Fehler kaum Auswirkungen auf die Simulationsergebnisse und ihre Qualität zu haben. Dies legt den Schluss nahe, dass hier die Nähe der angeforderten Simulationsergebnisse zu den vorhandenen anders be-

rechnet werden kann, sodass weniger Simulationen tatsächlich durchgeführt werden müssen, ohne dass die Qualität des gefundenen Optimums darunter leidet. Momentan arbeitet der Cache als eine Art virtueller Simulator. Das Optimierungssystem ist davon getrennt und *weiß* nicht, dass die Ergebnisse möglicherweise nicht realen Simulationsergebnissen entsprechen bzw. dass sie ggf. durch Simulation abweichender Parametersätze entstanden. Das Wissen um diese Abweichungen in der Optimierungsheuristik zu nutzen ist Ziel zukünftiger Untersuchungen.

5.3 Verteilte Berechnung

Da eine einzelne Simulation von SCPNs mitunter mehrere Minuten bis Stunden in Anspruch nehmen kann, wurden schon früh Versuche unternommen, dies auf mehrere CPUs oder Computer zu verteilen. Typischerweise wird dabei ein bestehendes Netz zerlegt und dessen Teile auf unterschiedlichen Prozessoren simuliert. Damit das Ergebnis korrekt ist, müssen diese Teilsimulationen jedoch stets synchronisiert werden, es sei denn, sie sind nachweislich funktional unabhängig voneinander. Vereinfacht wird diese Zerlegung und verteilte Simulation, wenn bei der Modellierung bestimmte Regeln bzgl. der Architektur des Netzes eingehalten werden. Die Probleme bei der Zerlegung und verteilten Simulation wurden von verschiedenen Forschern untersucht [61] und bieten durchaus Möglichkeiten, die simulationsbasierte Optimierung auf Basis von Petri-Netzen zu beschleunigen. Wie in [34] gezeigt, eignet sich diese Art der verteilten Simulation gut dafür, alternative Teilarchitekturen von Modellen zu untersuchen. All diese Ansätze zielen darauf ab, einzelne aber komplexe Petri-Netze durch Zerlegung und –wo nötig– synchronisierte Simulation auf getrennten CPUs oder Rechnern zu beschleunigen.
Ausgehend davon, dass einzelne Simulationen auf aktuellen Rechnergenerationen eine vertretbare Zeit von wenigen Minuten benötigt, für eine Optimierung jedoch sehr viele Simulationen durchgeführt werden müssen, wurde im Rahmen dieses Forschungsprojektes ein Ansatz nach dem bekannten Projekt Seti@home [1] verfolgt. Mehrere Simulationen werden dabei parallel, aber auf je einem Rechenknoten durchgeführt. Dadurch ergeben sich zwei wesentliche Vorteile:

Die Petri-Netze müssen nicht zerlegt bzw. dekomponiert werden, was in den meisten Fällen viel Know-How erfordert und nur unter bestimmten Bedingungen

vollautomatisch erfolgen kann.

Es ist keine Synchronisation zwischen den Rechnerknoten notwendig. Die Netzwerkverbindung zwischen den Knoten wäre sonst ein Single Point of Failure, sodass die komplette Simulation leicht scheitern oder verzögert werden könnte.

Wesentliche Nachteile sind die fehlende Beschleunigung einzelner Simulationen und die Tatsache, dass nicht alle Optimierungsheuristiken direkt davon profitieren können.

5.3.1 Nutzbarer Speedup durch Parallelisierung

Heuristiken, die jeweils eine neue Parameterkombination erzeugen und deren Simulationsergebnis für das weitere Vorgehen benötigen, profitieren nicht. Populationsbasierte Heuristiken nutzen die Möglichkeiten der parallelen Simulation jedoch aus, da hier je Iterationsschritt mehrere Simulationen notwendig sind und –abhängig von den zur Verfügung stehenden Ressourcen– ein linearer Speedup erreichbar ist.
Praktisch nutzbar, auch für andere Heuristiken, ist die Nutzung verteilter Rechenkapazität zur Erstellung einer Ergebnisdatenbank. Abhängig von der Diskretisierung dieses Caches und der Heuristik bzw. deren gewählter Diskretisierung kann damit die Optimierung stark beschleunigt werden.

5.3.2 Künftige Entwicklung

Damit auch nicht-populationsbasierte Heuristiken von paralleler Simulation ohne vorausberechnete Ergebnisse profitieren können, ist die Vorausberechnung aller nächstmöglichen Parametersätze umsetzbar. Am Bild 5.1 zeigt sich dieses Vorgehen für ein Modell mit 2 Parametern x & y. Noch während die Simulation des Modells mit einem Parametersatz läuft, sind bereits alle erreichbaren Parametersätze bekannt und können im voraus simuliert werden. Der erreichbare Speedup ist nicht mehr linear sondern stark von der Anzahl und Diskretisierung der Simulationsparameter abhängig.
Es spannt sich ein Baum mit allen möglichen Konfigurationen, ausgehend von den aktuellen Parameterwerten auf. Bei genügend großer Rechenkapazität ist auch vorstellbar, bei der Analyse eine Ebene des Baumes zu überspringen.

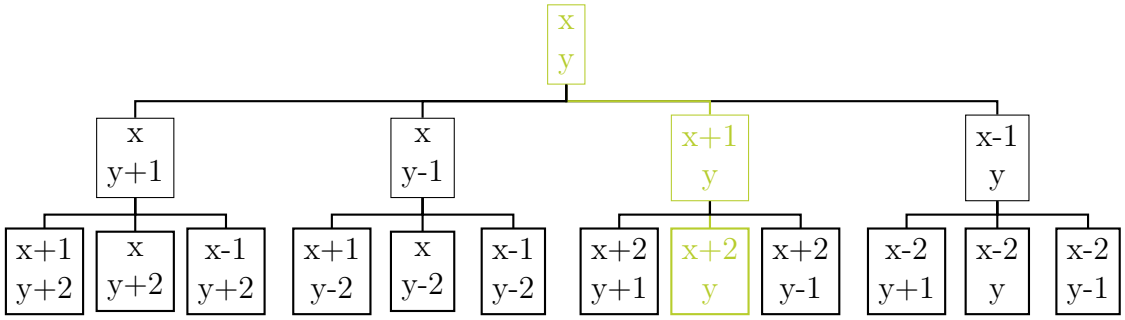


Abbildung 5.1: Beschleunigung durch Vorausberechnung mit Verteilter Simulation

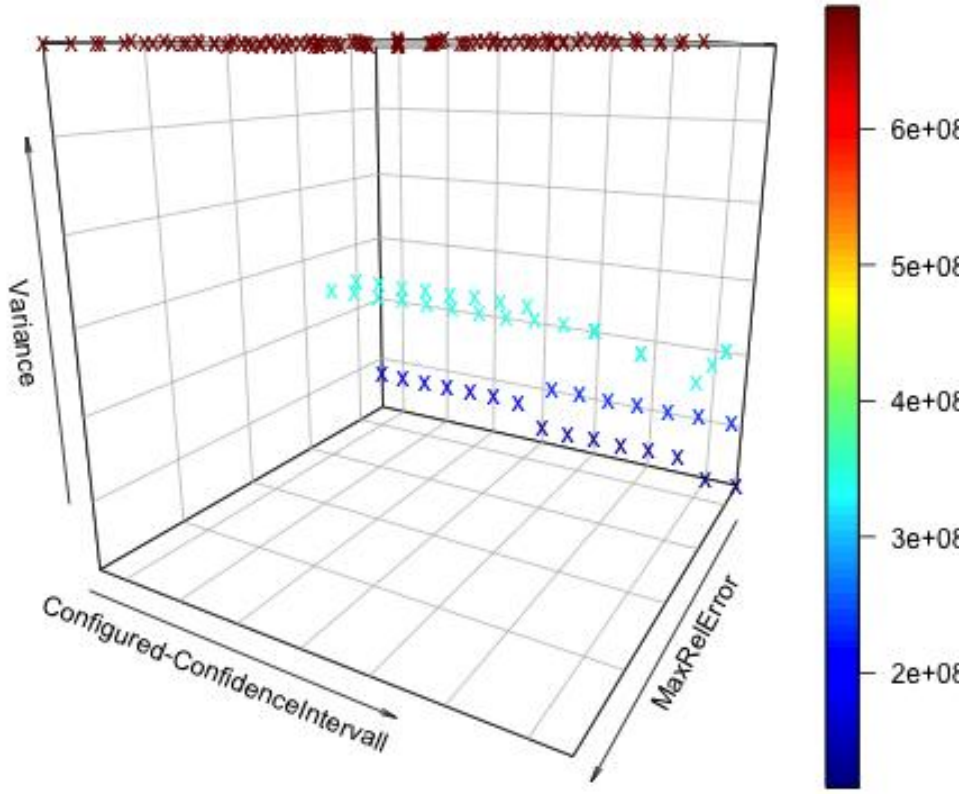


Abbildung 5.2: Varianz eines Leistungsmaßes im Verhältnis zum maximalen relativen Fehler und Konfidenzniveau (komplexes SCPN / Energiemodell)

5.4 Genauigkeitssteuerung

Während bisher genannte Strategien zur beschleunigten Optimierung primär die Anzahl an Simulationsläufen bzw. Antwortfunktionsberechnungen reduzieren, bietet die simulationsbasierte Optimierung noch eine weitere Möglichkeit: die Reduktion der Simulationsgenauigkeit. Wie bereits in Kapitel 4 beschrieben, hängt der Zeitbedarf für eine stationäre SCPN-Simulation von vielen Faktoren ab. Einige davon lassen sich gezielt manipulieren, um vergleichsweise schnell Simulationsergebnisse zu bekommen, wenn eine geringere Präzision erlaubt ist. Dieses Vorgehen lässt sich mit allen bekannten Heuristiken kombinieren, sofern diese nicht explizit über die verwendete Simulationsgenauigkeit informiert sein müssen.

5.4.1 Externe Parameter

Als externe Parameter einer Simulation werden in dieser Arbeit jene Parameter bezeichnet, welche die Art und Weise der Simulation selbst bzw. das Verhalten des Simulationssystems ändern, wobei das Modell unverändert bleibt. Hierbei geht es jedoch primär um externe Parameter, die Einfluss auf das Zeitverhalten der Simulation bzw. der Simulationsgenauigkeit haben.
Die wichtigsten Parameter zur Kontrolle der Simulationsgenauigkeit von SCPN-Simulationen sind das geforderte Konfidenzniveau und der maximale relative Fehler. Der allgemeine Zusammenhang zwischen diesen beiden Faktoren und der benötigten CPU-Zeit für eine stationäre Simulation ist in Bild 4.5 dargestellt. Abhängig von vielen Faktoren des Netzes und der Leistungsmaße, variieren die konkreten Werte der CPU-Zeit. Grundsätzlich zeigen aber alle untersuchten Netze ein ähnliches Zeitverhalten. Für die Untersuchung der Optimierungsheuristiken auf Basis von Benchmarkfunktionen wurde deshalb das Zeitverhalten durch Formel 4.2 nachgebildet. Vorläufige Ergebnisse der Untersuchungen mit diesem Vorgehen finden sich in [7].
Der Ansatz der Genauigkeitssteuerung besagt, dass während der Optimierung die Genauigkeit der Simulation schrittweise erhöht wird. Wie aus Bild 4.5 ersichtlich, kann sehr viel CPU-Zeit eingespart werden, wenn möglichst viele notwendige Simulationen mit vergleichsweise geringer Genauigkeit durchgeführt werden. Da aber auch die Ergebnisse bei Simulationen mit geringerer Präzision größeren Schwankungen unterliegen, wie anhand der Ergebnisvarianz in Bild 5.2 zu sehen ist, wurde unter-

sucht, inwieweit den Optimierungsergebnissen vertraut werden kann. Eine größere Varianz der Simulationsergebnisse erhöht das Risiko, ein globales Optimum zugunsten eines lokalen zu überspringen.

5.4.2 Interne Parameter

Unabhängig von externen Parametern zur Steuerung der Simulationspräzision lassen sich in den meisten Modellen auch interne Parameter etablieren, die Einfluss auf die Präzision und damit auch die benötigte CPU-Zeit haben. Anhand des Beispielnetzes in Kapitel 7 wird gezeigt, wie ein solcher Parameter in SCPNs genutzt werden kann. Der Wert dieses Parameters kann während der Optimierung anstatt oder in Kombination mit externen Präzisionsparametern variiert werden, um schrittweise die Präzision der Simulationsergebnisse zu erhöhen. Implementiert wurde diese Art der Präzisionssteuerung im mehrphasigen Optimierungsalgorithmus in Kapitel 6.6.5. Dass die schrittweise Erhöhung der Präzision CPU-Zeit einspart, ist unbestritten. Ähnlich wie die Variation der externen Präzisionsparameter wurde jedoch untersucht, inwieweit dem jeweils gefundene Optimum „vertraut“ werden kann, also wie groß das Risiko ist, nur ein lokales Optimum gefunden zu haben.

5.4.3 Künftige Entwicklung

Die weiteren Ziele bei der Entwicklung der Optimierung mit variabler Genauigkeit wurde bereits 2011 veröffentlicht [69]. Ziel ist es, Optimierungssteuerung und Simulationssystem enger zu koppeln, sodass ein Optimierungsalgorithmus in der Lage ist, die Simulationsgenauigkeit einzelner Modellkomponenten zu kontrollieren.
Auf diese Art werden flexibel dort Ressourcen eingespart, wo aufgrund des Modells oder des betrachteten Leistungsmaßes keine große Präzision gefordert ist. Die Grundidee ist in Bild 5.3 dargestellt. Alle Komponenten des Systems sind auf mehreren (im Idealfall auf allen) Abstraktionsebenen modelliert. Die Optimierungssteuerung bestimmt in einer ersten Phase, welche Komponenten mehr oder weniger Einfluss auf das aktuell betrachtete Leistungsmaß haben. Komponenten mit wenig Einfluss auf das Leistungsmaß können dann mit geringerer Präzision simuliert werden, indem für sie eine Implementierung auf höherem Abstraktionsniveau gewählt wird.
Eng verbunden mit dieser Thematik ist die Frage, ob und wie das Verhalten eines

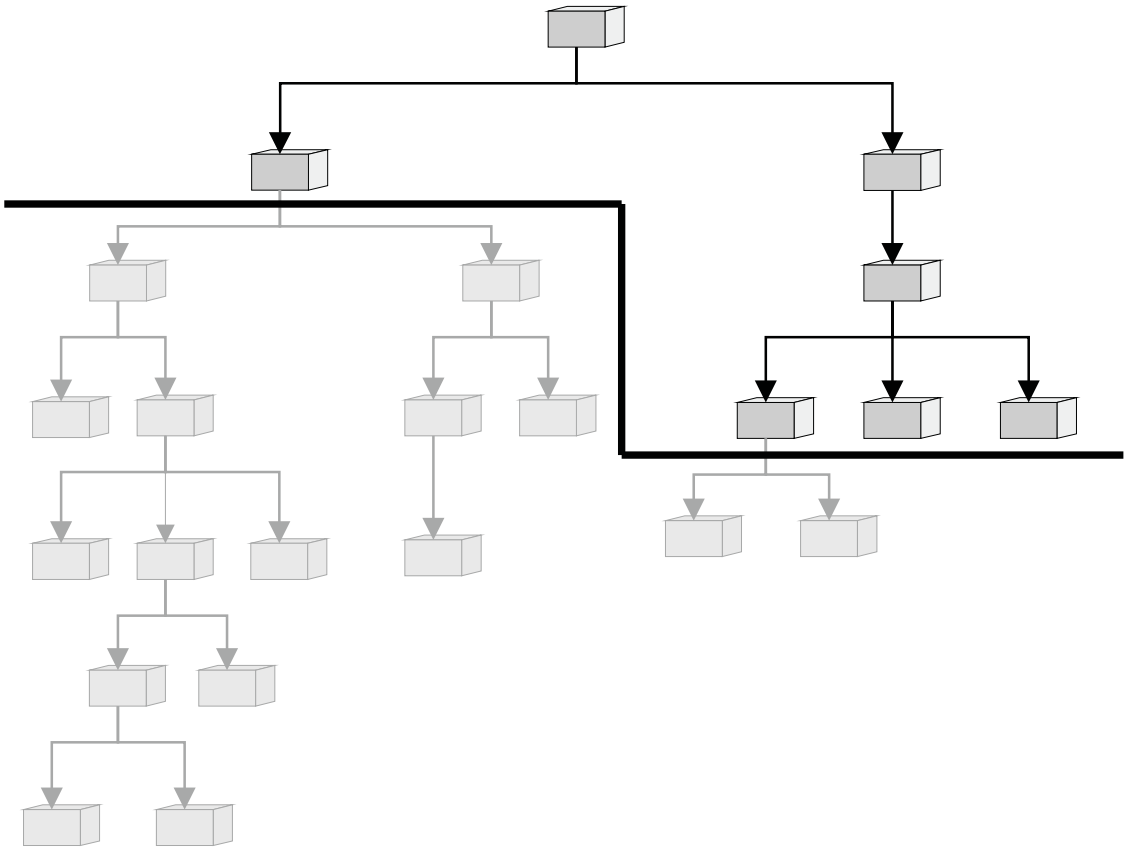


Abbildung 5.3: Komponentenweise Variation des Abstraktionsniveaus für die Simulation

Modells durch ein abstrakteres Modell nachgebildet werden kann. Viele Autoren beschäftigten sich bereits mit der Zusammenfassung von Petri-Netzen durch Aggregation, welche ein Netz mit kleinerem Zustandsraum aber beweisbar gleichem Verhalten aus einem komplexeren Netz erzeugt. Dabei müssen die Ausgangsnetze zumeist spezielle Anforderungen wie Symmetrie erfüllen [21, 9, 18].
Für die simulationsbasierte Optimierung ist es jedoch nicht erforderlich, dass abstraktere Modelle von Komponenten deren Verhalten exakt nachbilden. Zu untersuchen ist vielmehr die Abwägung zwischen Zeitersparnis durch abstraktere Modellierung von Komponenten und dem Risiko, dadurch das Verhalten des Gesamtmodells derart zu verfälschen, dass eine erfolgreiche Optimierung nicht mehr möglich ist.

6 TimeNET Optimization Environment (TOE)

Für viele bekannte Modellierungs- und Simulationswerkzeuge gibt es bereits Zusatzprogramme oder Plugins, um eine Optimierung des jeweiligen Modells durchführen zu können. Eine gute Übersicht dazu findet sich in Carsons Abhandlung zur simulationsbasierten Optimierung [13]. Diese Zusammenfassung ist inzwischen nicht mehr aktuell, einige genannte Werkzeuge werden nicht mehr weiterentwickelt oder wurden ersetzt. Neue Werkzeuge kamen auf den Markt. Die Anbindung an eine eigene Simulationssoftware wie z.B. TimeNET ist aus verschiedenen Gründen nur schwer bis gar nicht möglich. Die wenigsten Optimierungs-Werkzeuge sind quelloffen, sodass sie beliebig angepasst werden können. Andere sind von vornherein auf die Verwendung mit nur einer Simulationssoftware festgelegt. Ein kurze Übersicht bekannter Optimierungswerkzeuge ist in Tabelle 6.1.
Syrjakow entwickelte und veröffentlichte eine Optimierungssoftware, welche ähnliche Ansätze wie die hier Untersuchten (2- und Multiphasen-Optimierung) unterstützte. Das Programm war in SUN Pascal programmiert und ist auf Sparcstations mit OS 4.1.3 sowie Solaris 2.5.x lauffähig [59]. Diese und andere Einschränkungen wie die notwendige Kopplung an TimeNET mit Steuerung der Simulationsgenauigkeit machten eine Neuentwicklung notwendig.
Die Architektur, Dokumentation und Lizenz des entstandenen Programms soll später auch anderen Wissenschaftlern als Grundlage für die Untersuchung von Optimierungsalgorithmen in Zusammenarbeit mit verschiedenen Simulationswerkzeugen dienen.

Name	**Entwickler**	**Simulationssystem**
SimRunner	ProModel Corporation	ProModel
OptQuest96	Optimization Technologies, Inc.	Arena, Crystal Ball, OptSim, etc.
WITNESS Optimizer	Optimizer Lanner Group, Inc.	WITNESS
AutoStat	AutoSimulations, Inc.	AutoMod
OPTIMIZ	Visual Thinking International Ltd.	

Tabelle 6.1: Auswahl existierender Werkzeuge zur simulationsbasierten Optimierung

6.1 Existierende Werkzeuge zur simulationsbasierten Optimierung

Tabelle 6.1 gibt einen kleinen Überblick über aktuell verfügbare Werkzeuge zur simulationsbasierten Optimierung. Verschiedene Hersteller haben im Laufe der Entwicklung jeweils unterschiedliche Schwerpunkte bei der Implementierung von Optimierungsalgorithmen gesetzt. Je nach ursprünglichem Einsatzszenario unterstützen die Werkzeuge je eine entsprechende Auswahl von Algorithmen. Die meisten Optimierungswerkzeuge sind strikt an ein entsprechendes Simulationssystem gebunden, was den Einsatz für die Optimierung von SCPNs unmöglich macht. Dennoch ergibt sich ein praktikabler Marktüberblick, der bei der Auswahl alternativer Optimierungswerkzeuge helfen kann. Zum großen Teil entstammt der Überblick der Arbeit von Michael C. Fu ([24, 25]).

Neben vielen weiteren SW-Werkzeugen zur Simulation und Optimierung ist insbesondere die Arbeit von Syrjakow und Szczerbicka zu nennen. Sie beschreiben in ihren Arbeiten bereits früh, wie die Kombination bzw. ihre aufeinander folgende Anwendung verschiedener Optimierungsalgorithmen dabei helfen kann, das Optimum von Funktionen zu finden, die sehr viele lokale Extremstellen haben [57]. Diese, multimodal genannten, Kostenfunktionen werden in der Bewertungsphase dieser Arbeit durch das Hausenergiemodell sowie die Benchmarkfunktionen Ackley und Schwefel repräsentiert.

Ebenso wurden u.a. in den Arbeiten von Syrjakow und Szczerbicka die grundsätzlichen praktischen Probleme simulationsbasierter Optimierung genannt und Lösungs-

vorschläge veröffentlicht. So entstand eine Architektur für mögliche Werkzeuge zur simulationsbasierten Optimierung wie auch eine Implementierung dessen [59].

6.2 Anforderungen an das Werkzeug

Vor Beginn der Entwicklung wurden die Anforderungen an das SW-Werkzeug zusammengetragen. Die wichtigsten sollen hier kurz genannt und erläutert werden. Während der Entwicklung ergaben sich stets weitere Anforderungen, weshalb diese Liste keinesfalls vollständig ist. Die Umsetzung verschiedener Optimierungsheuristiken und Strategien zur Beschleunigung dieser standen im Vordergrund der Entwicklung.

Anbindung an TimeNET Die Anbindung an TimeNET ist eine der wichtigsten Forderungen und die Basis für SCPN-Optimierung. Dabei müssen neue Petri-Netze im kompatiblen Format erzeugt werden. Die Simulation muss gestartet und das Ergebnis ausgelesen/gespeichert werden.

Batch-Simulation Nicht nur als Test für die Anbindung an das Simulationswerkzeug, sondern auch zur Gewinnung von Daten für einen Ergebniscache ist die Simulation vieler Parameterkombinationen notwendig.

Heuristiken für die simulationsbasierte Optimierung Über eine entsprechend dokumentierte Schnittstelle sollten zunächst Basisverfahren zur simulationsbasierten Optimierung mit diversen Ausprägungen und anschließend erweiterte/kombinierte Verfahren umgesetzt werden.

Strategien zur beschleunigten Optimierung Die im Kapitel 3 genannten Strategien zur Beschleunigung von Optimierungen sollten umgesetzt werden. Dabei ist es notwendig, dass alte und verbesserte Verfahren für Vergleichstests verfügbar, also implementiert, sind.

Verteilte Simulation von Petri-Netzen Die verteilte Simulation verschiedener SCPNs zur gleichen Zeit erlaubt einerseits den schnelleren Aufbau eines Ergebniscaches, welcher einige Heuristiken indirekt beschleunigt, andererseits können populationsbasierte Heuristiken direkt von der parallelen Simulation profitieren.

Benchmarkfunktionen Optimierungsalgorithmen werden allgemein anhand von Benchmarkfunktionen getestet. Im Vergleich zur echten SCPN-Simulation ist eine Überprüfung und Bewertung der grundsätzlichen Funktion sehr zeitsparend möglich. Die Benchmarkfunktionen sollten möglichst transparent eingebettet und später um weitere Vertreter ergänzt werden können. Die jeweils verwendete Optimierungsheuristik soll keine Information darüber haben, ob tatsächlich Simulationen durchgeführt werden oder eine Benchmarkfunktion verwendet wird.

Bewertung von Optimierungsverfahren Um Optimierungsverfahren bewerten zu können, sollte jedes Verfahren beliebig oft automatisch ausgeführt werden können. Auf diese Weise lassen sich statistische Daten über die Qualität des Verfahrens sammeln. Dies umfasst die Entfernung zum tatsächlichen Optimum (falls bekannt) als auch zur Anzahl der Simulationsläufe, CPU-Zeit etc..

Graphische Darstellung der Ergebnisse Um die Funktionsweise von Optimierungsalgorithmen sowie die grundsätzliche Form der Antwortfunktion zu verstehen ist es hilfreich, eine grafische Darstellung der Funktionswerte zu haben. Im Vergleich mit der Darstellung der Benchmarkfunktionswerte lässt sich so später leicht entscheiden, welche Optimierungsheuristik für das verwendete SCPN am geeignetsten ist.

6.3 Software-Architektur

Da das Programm im Laufe seiner Entwicklung mit 47 Klassen und ca. 18k LOC relativ komplex geworden ist, sollen hier nur die wichtigsten Aspekte der Architektur erläutert werden. Grundsätzlich ist das Projekt in 6 Pakete eingeteilt.
Das Basispaket *toe* beinhaltet zentrale Steuerklassen, Fenster, Parser und die Klasse *Support*. Diese stellt alle Programmparameter und ihre Standardwerte zur Verfügung, mit Ausnahme der Heuristik-Konfigurationen. In *datamodel* sind die 3 zentralen Datenstrukturen enthalten, sie werden im folgenden Abschnitt noch genauer beschrieben. Die grafische Darstellung der Ergebnisse wird im Paket *plot* realisiert. Es beinhaltet ein Nutzerinterface und die Ansteuerung des, zusätzlich zu installierenden, Statistikwerkzeugs **R**.

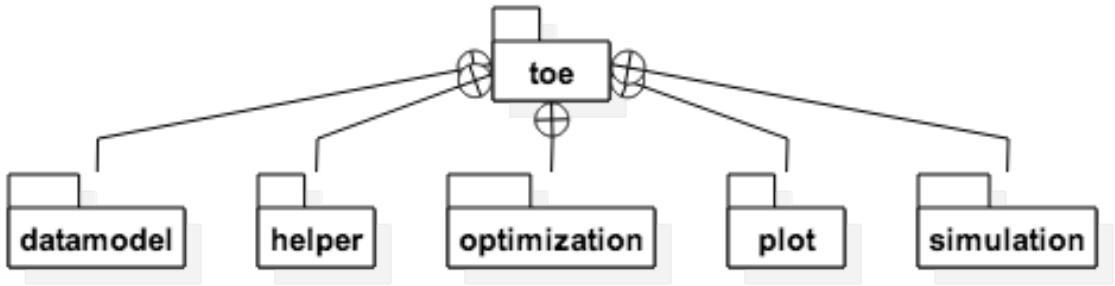


Abbildung 6.1: Überblick der Softwarearchitektur, Paketdiagramm

Wichtigste Pakete für die Simulation und Optimierung sind jedoch *simulation* und *optimization*.
Im Paket *simulation* wurde das Interface *Simulator* eingeführt. Dies erleichtert die spätere Integration von weiteren Simulatoren. Aus Sicht des Optimierungsalgorithmus ist es egal, welcher Simulator benutzt wird. Der Nutzer kann dies in der grafischen Nutzeroberfläche entscheiden.
Zur Verfügung stehen 6 Typen von Simulatoren, die im folgenden anhand ihrer Repräsentation in der Nutzeroberfläche identifiziert und beschrieben werden. Die tatsächlichen Klassen haben davon abweichende Namen.

Local Dies ist der Standardsimulator. Es wird die lokal verfügbare TimeNET-Installation verwendet, um eine Simulation zu starten und die Ergebnisse aus der Log-Datei zu lesen.

Distributed Verteilte Simulation. Alle geforderten Simulationen werden als xml-Datei zu einem Server hochgeladen und an Clients verteilt. Die Ergebnisse werden anschließend wieder heruntergeladen. Das Programm wartet, bis alle Ergebnisse erfolgreich heruntergeladen werden konnten. Auf den Clients wird dabei jeweils eine lokale Simulation ausgeführt.

Cache_only Es wird keine echte Simulation durchgeführt. Stattdessen wird die angeforderte Simulation bzw. deren Ergebnis in einer Ergebnisdatenbank gesucht. Wird kein passendes Ergebnis gefunden, so gibt der Simulator das nächstgelegene zurück. Dabei wird die Summe der absoluten Entfernungen aller Parameter der angeforderten Simulation zu allen vorhandenen Datensätzen verglichen.

Cached_Local Grundsätzlich liefert dieser Simulator ebenfalls nur Simulationser-

gebnisse aus einer zuvor erstellten Datenbasis, wie auch *Cache_only*. Falls jedoch ein angefordertes Simulationsergebnis nicht gefunden wurde, so wird eine lokale Simulation gestartet und das Ergebnis anschließend zurück gegeben, sowie in die temporäre Ergebnisdatenbank gespeichert.

Cached_Distributed Dieser Simulator arbeitet wie *Cached_Local*. Nur wird hier die Simulation nicht lokal ausgeführt, sondern verteilt auf andere Clients. Im Klassendiagramm 6.2 ist dieser Zusammenhang ersichtlich. Die Klasse erweitert den Simulator *Cached_Local* und verwendet statt dem lokalen Simulator eine Instanz der *Distributed*-Klasse.

Benchmark Diese Klasse implementiert ebenfalls das *simulator*-Interface, verhält sich also gegenüber den implementierten Optimierungsheuristiken wie ein echter Simulator. Tatsächlich werden die Ergebniswerte für alle Leistungsmaße des zu optimierenden SCPN anhand bekannter Benchmarkfunktionen berechnet. Jedem veränderbaren Parameter des SCPN wird eine Eingangsvariable der Benchmarkfunktion zugeordnet, wobei der Definitionsbereich des Parameters an den der Benchmarkfunktion angepasst/umgerechnet wird. Dieser ist für jede Benchmarkfunktion verschieden und in der Klasse *support* definiert. Vorteile der Verwendung von Benchmarkfunktionen sind u.a. die Geschwindigkeit, eine echte Simulation dauert um ein Vielfaches länger. Außerdem können auf diese Weise künstlich verschiedene Wertelandschaften erzeugt werden, um das Verhalten der Optimierungsalgorithmen zu testen.

Größter Vorteil ist aber die Tatsache, dass das globale Optimum bekannt ist bzw. berechnet werden kann. Damit ist eine Beurteilung der Optimierungsergebnisse einfach und automatisiert möglich. Aus der großen Menge bekannter Benchmarkfunktionen wurden jeweils 2 zur Umsetzung ausgewählt, welche sowohl viele als auch wenige lokale Optima aufweisen, auf 2 Dimensionen definiert sind und deren Optimum eindeutig berechenbar ist.

Es wurden folgende Benchmarkfunktionen umgesetzt:

Sphere Eine einfache Funktion, definiert für beliebig viele Eingangsvariablen.

Der Definitionsbereich ist $x_i \in [-5; 5]$ [48].

$$f(\overrightarrow{x}) = \sum_{i=1}^{D} x_i^2 \tag{6.1}$$

Matya Die Funktion ist nur für zwei Eingangsvariablen definiert. Ihr Optimum liegt bei $x = 0, y = 0$. Sie hat keine lokalen Optima, jedoch steigen die Funktionswerte in zwei Richtungen hinweg vom Optimum nur sehr langsam an. Dies führt zu interessanten Ergebnissen, wenn man die gefundenen Optima in Bezug auf ihren Abstand vom tatsächlichen Optimum im Definitionsbereich betrachtet [7, 48].

$$f(x, y) = 0.26(x^2 + y^2) - 0.48xy \tag{6.2}$$

Ackley Im Zweidimensionalen hat diese Funktion eine trichterförmige Gestalt mit ihrem Optimum bei $x = 0, y = 0$, also im Koordinatenursprung. Sie ist definiert im Bereich $x_i \in [-5; 5]$ und hat sehr viele lokale Optima [48].

$$\begin{aligned} f(\overrightarrow{x}) = -20 \exp\left\{-0,2 \cdot \sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right\} \\ - \exp\left\{-0,2 \cdot \sqrt{\frac{1}{D}\sum_{i=1}^{D} \cos(2\pi x_i)}\right\} + 20 + e \end{aligned} \tag{6.3}$$

Schwefel Diese Funktion hat ihr Optimum bei $f(\overrightarrow{x_i = 420.9687}) = 0$. Der Definitionsbereich ist im implementierten Fall $x_i \in [-500; 500]$. Sie hat ebenfalls viele lokale Optima und wird oft zur Evaluierung von Optimierungsalgorithmen herangezogen [48].

$$f(\overrightarrow{x}) = 418.9829 \cdot n - \sum_{i=1}^{n}(x_i \sin(\sqrt{|x_i|})) \tag{6.4}$$

Cached_Benchmark Analog zu *Cached_Distributed* und *Cached_Local* arbeitet dieser Simulator. Die Ergebnisdaten werden jedoch von der gewählten Benchmarkfunktion berechnet.

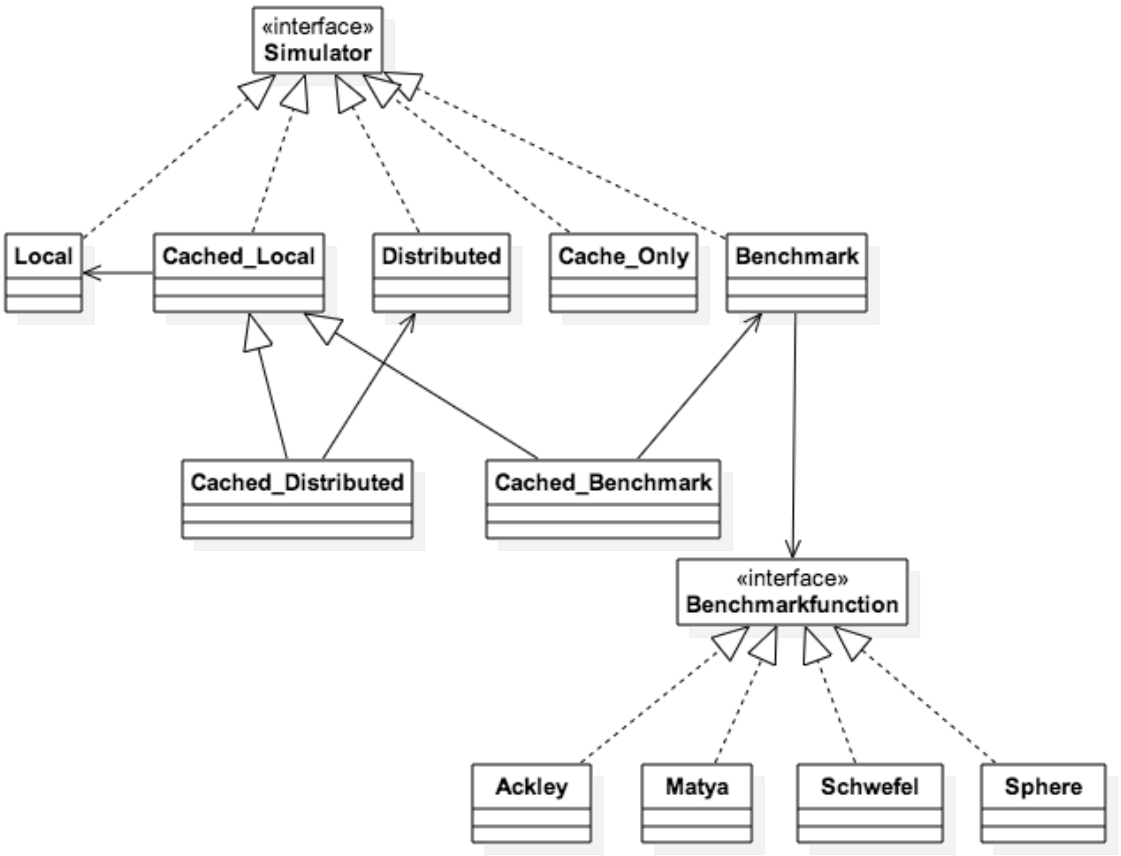


Abbildung 6.2: Implementierte Simulatoren, Klassendiagramm

6.4 Datenstrukturen

An dieser Stelle sollen nur die Klassen behandelt werden, die simulationsrelevante Daten halten oder manipulieren. Sie sind im Paket *datamodel* zu finden (Abbildung 6.1).

parameter Jeder Parameter einer Simulation wird intern durch ein Objekt vom Typ *parameter* repräsentiert. Es wird unterschieden zwischen externen Parametern, die für die Simulationssteuerung zuständig sind (Seed, Maximaler relativer Fehler, Konfidenzniveau, Maximale CPU-Zeit pro Simulation, Maximale Anzahl an Simulationsschritten) und netzinternen Parametern. Die internen Parameter werden durch Auslesen der xml-Datei des SCPN bestimmt. Die Klasse *parameter* beinhaltet unter anderem: **Start- und Endwert**, **Aktueller Wert**, **Schrittweite**, **Namen**. Aus diesen Informationen wird abgeleitet,

ob es sich um einen intern oder externen Parameter handelt und ob dieser iterierbar ist, oder aufgrund dieser Daten keinen anderen als den aktuellen Wert annehmen kann.

MeasureType Ein Objekt vom Typ *MeasureType* repräsentiert ein Leistungsmaß des SCPN. Prinzipiell können pro Netz beliebig viele Leistungsmaße definiert werden, dies geschieht in TimeNET. Sie werden von TOE aus der xml-Datei des Netzes gelesen und stehen zur Wahl für die Optimierung. Ein MeasureType-Objekt hat neben Namen und aktuellem Wert (Fließkommawert) auch die Eigenschaften **Optimierungsziel** und Zielwert. Das Optimierungsziel, oder auch *target* im Programm genannt, gibt an, ob die Optimierungssteuerung Parameterwerte suchen soll, für die dieses Leistungsmaß minimal oder maximal wird. Alternativ kann auch ein Zielwert angegeben werden, an den das Leistungsmaß angenähert werden soll. Die Methode getDistancefromTarget() liefert stets den aktuellen absoluten Abstand vom jeweiligen Zielwert zurück und wird von der Optimierungssteuerung bzw. den Heuristiken genutzt. Welches Leistungsmaß Ziel der Optimierung ist, wird vom Nutzer vorgegeben. Es wird nicht im Measure-Objekt selbst gespeichert.

Jedem Objekt dieses Typs ist eine Liste von Parametern zugeordnet, um es in der Ergebnisdatenbank eindeutig identifizieren zu können, wie im nächsten Abschnitt beschrieben.

SimulationType Objekte dieses Types werden vom Ergebnisparser erzeugt. Sie halten die Daten einer kompletten Simulation. Dies beinhaltet Namen des SCPN, sämtliche Parameter mit ihren verwendeten Werten und ggf. weiteren Eigenschaften sowie Leistungsmaße und deren Werte/Zielwerte. Außerdem noch einige statistische Daten über die abgelaufene Simulation. Dazu gehören CPU-Zeit, Anzahl an Simulationsschritten, Art der Simulation, etc..

Für eine Simulation wird stets der Dateiname und eine Liste von Parameterlisten an den internen Simulator übergeben. Der Simulator meldet über einen Statuswert, wie viele der übergebenen Parameterlisten -also parametrisierte Modelle- abgearbeitet bzw. simuliert wurden und kann danach oder auch schon während der Simulation die bisherigen Ergebnisse zurück geben.

6.5 Caching

Die Speicherung von Simulationsergebnissen ist wichtiger Bestandteil jedes Optimierungssystems, insbesondere bei simulationsbasierter Optimierung. Da die Bestimmung der Leistungsmaße durch Simulation u.U. viel Zeit in Anspruch nimmt, kann durch den geschickten Einsatz von Ergebnisdatenbanken bzw. Caches viel Zeit bei der Optimierung gespart werden, selbst wenn die Anzahl der notwendigen Simulationsläufe bzw. Berechnungen der Leistungsmaße nicht verringert wird.
Um die Optimierung nachvollziehbar zu machen und als Basis der Batch-Simulation, werden alle Ergebnisse von Simulationen, die von TOE angefordert werden, in einer csv-Datei gespeichert. Zusätzlich werden alle Simulationsergebnisse intern in einem globalen Cache (Ergebnisliste) abgelegt.
In die csv-Datei wird für jedes Leistungsmaß jeder Simulation eine Zeile geschrieben. In jede Zeile werden dazu sämtliche genutzten Parameterwerte und Ergebnisse eingetragen. Dadurch können diese Daten leicht in anderen Programmen wie MatLab, MS Excel, etc. weiter verarbeitet werden. Es lässt sich damit schnell ein grober Funktionsplot erzeugen oder der Ablauf einer Optimierung nachvollziehen.
Jede derartige csv-Datei kann ebenfalls vom Programm geladen und für die Beschleunigung der Optimierung oder den offline-Test von Heuristiken genutzt werden. Hierzu wird die Datei eingelesen und geprüft, ob die geladenen Daten mit dem gewählten SCPN übereinstimmen (Art und Anzahl der Parameter, Namen der Leistungsmaße).
In Bezug auf den Umgang mit Simulationsergebnissen gibt es TOE-intern drei Cache-Strategien. Sie werden durch die Art des gewählten Simulators bestimmt.

Kein explizites Cache-Verhalten Dies ist das Standardverhalten. Es werden keine Ergebnisse zwischengespeichert. Es ist hauptsächlich für Benchmarksimulationen und Heuristiken mit eingebauter Cache-Verwaltung sinnvoll.

Cache_only Hierfür muss zunächst eine passende csv-Datei mit Simulationsergebnissen geladen werden. Es wird nie eine echte Simulation durchgeführt. Der Optimierungsalgorithmus greift auf die Daten zu als würden diese live berechnet. Für diese Funktionalität wird zeitweise der lokale Simulator durch eine Ergebnisliste ersetzt. Wird eine Parameterkonfiguration nicht in der Liste gefunden, so wird die nächstgelegene gesucht und deren Ergebnis zurück

gegeben.

Cache_local / Cache_distributed Diese Strategie ist fast identisch mit **Cache_-only**. Falls jedoch eine angeforderte Parameterkonfiguration nicht in der Ergebnisliste gefunden wird, so startet eine lokale oder verteilte Simulation, deren Ergebnis anschließend ausgewertet und der Ergebnisliste angehängt wird.

6.6 Umgesetzte Optimierungsalgorithmen

Kern der Arbeit ist die Untersuchung bekannter Optimierungsheuristiken sowie die Implementierung und Analyse von Maßnahmen zur Beschleunigung. Nachfolgend werden die implementierten Heuristiken und ihre möglichen Konfigurationen erläutert.
Analog zur Umsetzung der Simulatortypen wurde auch hier ein Interface definiert, welches durch jede entsprechende Heuristikklasse implementiert werden musste. Wie in Bild 6.3 zu sehen, gibt es dabei nur zwei Besonderheiten. Die Klasse *Simulated-Annealing* ist abgeleitet von *HillClimbing*; die beiden Klassen *TwoPhase* und *Multiphase* sind die einzigen, welche das Interface nicht nur nutzen, sondern auch Referenzen auf Objekte dieses Typs halten. Die Gründe dafür erschließen sich bei näherer Betrachtung dieser Heuristiken bzw. ihrer Umsetzung.
Allen Heuristiken gemeinsam ist die Wahl der Startwerte für alle Parameter. 4 Möglichkeiten stehen zur Verfügung. So kann vor Beginn der Optimierung entschieden werden, ob alle veränderbaren Parameter des SCPN auf ihren niedrigsten, höchsten, mittleren oder einen zufälligen Wert gesetzt werden. Innerhalb des Programms besteht außerdem die Möglichkeit, einen frei gewählten Wert festzulegen, dies ist insb. bei später beschriebenen mehrphasigen Optimierung wichtig.
Für eine wiederholte Analyse der Heuristiken wurde stets ein zufälliger Anfangswert genutzt. Mithilfe des Java-eigenen Pseudozufallszahlengenerators wird dabei ein Wert zwischen Start und Endwert des jeweiligen Parameters gewählt, der dem definierten Raster (Schrittweite) entspricht, sofern die jeweilige Heuristik das erfordert.
Die Möglichkeit, die Optimierung mit mini- bzw. maximalem Wert für alle Parameter zu beginnen, ist eher für die grundsätzliche Funktionsprüfung der Algorithmen sinnvoll. Gleiches gilt für den Start mit mittlerem Parameterwert.

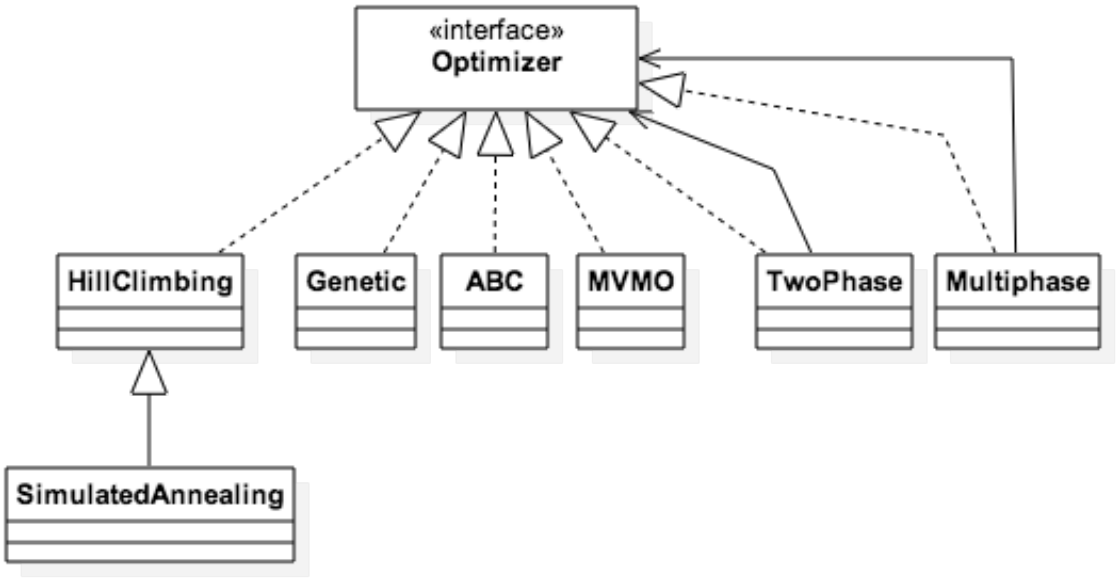


Abbildung 6.3: Implementierte Optimierungsalgorithmen, Klassendiagramm

6.6.1 Hill-Climbing

Hill-Climbing ist das bekannteste Verfahren, vergleichsweise einfach umzusetzen und nachvollziehbar. Der grundsätzliche Algorithmus bei Verwendung mehrerer Parameter ist in Listing 2 beschrieben. Die Umsetzung ist angelehnt an die Single-Factor-Methode, 1973 beschrieben von Smith [53]. Namensgebend ist die Tatsache, dass die Parameter nacheinander und nicht gleichzeitig variiert werden.
Das Listing bildet nur eine mögliche Konfiguration des implementierten Algorithmus ab. In der Initialisierungsphase werden zunächst alle Parameter auf zufällige Werte innerhalb ihres Definitionsbereichs gesetzt. Wichtige Einstellungen des Algorithmus sind die Anzahl der „schlechteren“ Lösungen ($MaxWrong = WRONG\_SOLUTIONS\_IN\_A\_ROW$) sowie die Anzahl der „schlechteren“ Lösungen pro Richtung ($MaxWrongPerDirection = WRONG\_SOLUTION\_PER\_DIRECTION$). So lange nicht $WRONG\_SOLUTIONS\_IN\_A\_ROW$ viele Parametersätze in Folge getestet wurden, deren Lösung bzw. deren Simulationsergebnis schlechter als das Gespeicherte ist, wird die Optimierungsschleife durchlaufen. Dabei wird stets ein Parameter um seine Schrittweite erhöht. Falls dies $WRONG\_SOLUTION\_PER\_DIRECTION$ mal zu einer schlechteren

Lösung führt, wird zunächst die Richtung gewechselt. Folgen dann wieder $WRONG\_SOLUTION\_PER\_DIRECTION$ viele schlechtere Lösungen, so wird der Parameter zurück gesetzt und der nächste wird verändert. Der wesentliche Punkt, die Berechnung eines neuen Wertes für einen Parameter kann, konfiguriert werden. Es wurden die in 2.4.1 beschriebenen Strategien komplett umgesetzt, wobei hauptsächlich *Diskret, auf/ab* im Rahmen der Untersuchungen eingesetzt wurde.

$p[0] := Random(p_{max}, p_{min})$
$p[1] := Random(p_{max}, p_{min})$
$p_{best} := p$
$MaxWrongPerDirection := WRONG\_SOLUTION\_PER\_DIRECTION$
$MaxWrong := WRONG\_SOLUTIONS\_IN\_A\_ROW$
$index := 0$
$dir := 1$
while $MaxWrong > 0$ **do**
  $p[index] := p[index] + stepSize(p[index]) * dir$
  if $cost(p) < cost(p_{best})$ **then**
    $p_{best} := p$
    $MaxWrongPerDirection :=$
    $WRONG\_SOLUTION\_PER\_DIRECTION$
    $MaxWrong := WRONG\_SOLUTIONS\_IN\_A\_ROW$
  else
    $MaxWrongPerDirection - -$
    $MaxWrong - -$
    if $MaxWrongPerDirection \leq 0$ **then**
      if $dir == 1$ **then**
        $dir = -1$
        $MaxWrongPerDirection :=$
        $WRONG\_SOLUTION\_PER\_DIRECTION$
      else
        $dir := 1$
        $index := (index + 1) \mod length(p)$
      end if
    end if
  end if
end while
print p_{best}

Algorithmus 2 : Prinzipieller Ablauf von Hill-Climbing mit zwei Parametern

6.6.2 Simulated Annealing

Der Ablauf von Simulated Annealing ist bereits in Abschnitt 2.4.2 erläutert worden. Es wurde analog zu Listing 1 implementiert. Dabei werden in jedem Optimierungsschritt stets alle veränderbaren Modellparameter geändert und nicht nur einer, wie bei Hill-Climbing. Für die wichtigsten Untersuchungen wurde die Konfiguration des Algorithmus grundsätzlich so gewählt wie in [67] vorgestellt. Als Abkühlungsfunktionen stehen Boltzmann - , Fast - und Very Fast - Annealing zur Verfügung.
Für die Berechnung des jeweils nächsten Parametersatzes wurden ebenfalls verschiedene Möglichkeiten umgesetzt, die insbesondere Einfluss auf die Beschleunigung durch caching haben.

Normal Hier wird für jeden Parameter der nächste Wert nach dem Algorithmus aus [67] berechnet bis auf eine kleine Abweichung im Exponenten $|2r| - 1$.
$r := Random(-1, 1)$
$p := p + T^{para} * \left[(1 + \frac{1}{T^{para}})^{|2r|-1} - 1\right] * d * sign(r)$
d entspricht hier der Dimension des Definitionsraumes, also der Anzahl an variablen Parametern.

Simple Dies ist eine vereinfachte Berechnung ohne Berücksichtigung der Anzahl an variablen Parametern.
$p := p + T^{para} * range * signRandom(-1, 1)$
$range$ entspricht hier der Größe des Definitionsbereichs des jeweiligen Parameters.

In beiden Fällen können zufällig auch Werte für p errechnet werden, die außerhalb des entsprechenden Definitionsbereichs liegen. Wenn dies vorkommt, wird die Berechnung so lange wiederholt, bis ein gültiger Wert für p vorliegt.
Zu jeder Variante existiert noch eine Modifikation mit Namenszusatz „Stepwise“. Dabei wird der neue Parameterwert zunächst berechnet und dann auf den nächsten Wert gerundet, welcher der vorgegebenen minimalen Schrittweite entspricht. Offensichtlich ist, dass die Wahrscheinlichkeit einen vorausberechneten Wert aus einer entsprechenden Datenbank anzufordern größer ist, wenn die Parameter auf eine Schrittweite ähnlich derer aus der Datenbank gerundet werden. Wird in diesem Zusammenhang als Simulator die Variante „Cache_only“ verwendet, so werden

dennoch keine echten Simulationen gestartet, sondern nur der nächstgelegene Wert aus der Datenbank gelesen.
Unabhängig von der konkreten Implementierung wird das Verhalten von Simulated Annealing im wesentlichen durch 3 Konfigurationsparameter bestimmt. Dies sind zum einen die beiden Startwerte der Temperaturen T_0^{para}, T_0^{cost} sowie die Abbruchtemperatur ϵ, im Listing 1 auch T_{min}^{para} und T_{min}^{cost} genannt. In der verwendeten Implementierung wird die gleiche Abbruchtemperatur für T^{para} und T^{cost} genutzt.
Für die Startwerte der Temperaturen wird standardmäßig 1 gewählt. Dies bedeutet, dass jedes Simulationsergebnis akzeptiert wird (maximale Akzeptanzwahrscheinlichkeit). Prinzipiell können auch Werte $\geqslant 1$ gewählt werden. Allerdings steigt damit nur die Wahrscheinlichkeit, dass der neu bestimmte Parameterwert außerhalb des Definitionsbereichs liegt und die Berechnung wiederholt werden muss. Die geschätzte Anzahl an Simulationsläufen wird im Programm angezeigt und unterstützt bei der Wahl der passenden Werte für die Konfigurationsparameter.

6.6.3 Genetische Optimierungsalgorithmen

Allen genetischen Verfahren gemein ist der prinzipielle Ablauf (Bild 6.4) bestehend aus 3 Elementen ausgehend von einer zufälligen Basispopulation aus Parametersätzen, deren Kostenfunktionswert bestimmt wurde. Parametersätze werden in diesem Zusammenhang oft auch als Vektoren bezeichnet.

1. **Selektion / Evaluierung** Anhand der berechneten Werte für die Kostenfunktion wird ein Ranking der Paramatersätze erstellt, die Population wird damit sortiert.

2. **Rekombination** Es werden i.d.R. aus zwei Elternvektoren mehrere neue Kindvektoren erzeugt.

3. **Mutation** Dabei werden zufällige Einflüsse in die nächsten Parametersätze, also die Kindvektoren einberechnet.

Diese Elemente können je nach Implementierung nacheinander ablaufen oder ineinander verschachtelt. Genetische Algorithmen unterscheiden sich im wesentlichen in der Erzeugung der Kindvektoren und dem verwendeten Mutationsverfahren.

In TOE wurden verschiedene Kreuzungsverfahren implementiert, von denen eines im Rahmen dieser Arbeit als Vergleich zu nicht-populationsbasierten Heuristiken herangezogen wird. Dabei handelt es sich um Simulated Binary Crossover (SBX), vorgestellt in [16]. Für jeden Parameter wird dabei aus der Elterngeneration pro Kind ein neuer Wert errechnet, unter Berücksichtigung zufälliger Einflüsse in Form eines sogenannten Spreizfaktors. Außerdem werden alle Mitglieder einer erzeugten Generation, wie bei allen anderen Verfahren, gemäß der Mutationswahrscheinlichkeit mutiert, also ihre Parameter ggf. zufällig verändert.
Die Wahl des Kreuzungsverfahrens und dessen Konfiguration ist einer der Forschungsschwerpunkte im Bereich der genetischen Optimierungsalgorithmen. In der Fachliteratur werden diverse alternative Kreuzungsverfahren beschrieben [17].

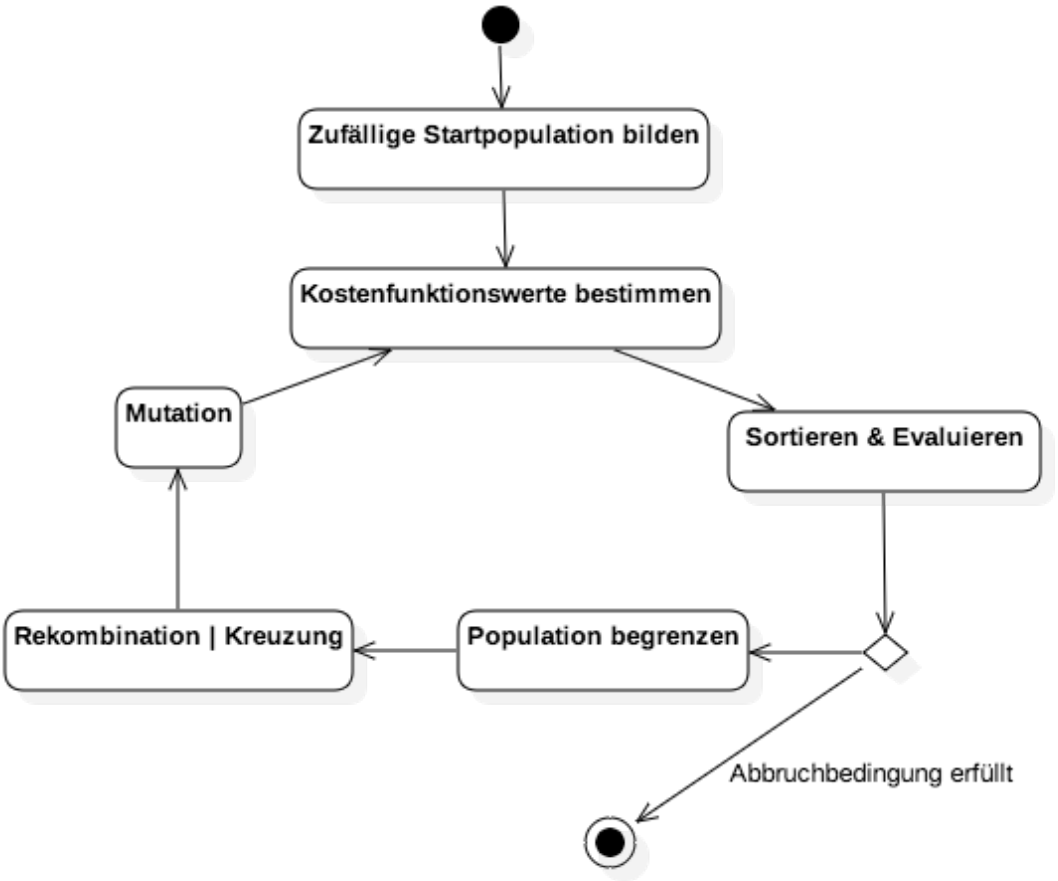


Abbildung 6.4: Ablauf genetischer Algorithmen

Als Parameter für die Anpassung der umgesetzten genetischen Optimierung stehen momentan zur Verfügung:

Populationsgröße Dies ist die Anzahl an zufällig erzeugten Parametersätzen bzw. Ausgangsvektoren zu Beginn der Optimierung. Nachdem je einige Kindvektoren erzeugt und ihre Kostenfunktionswerte bestimmt wurden, wird die Population wieder auf diese Größe beschränkt, d.h. die schlechtesten Mitglieder der Population werden entfernt.

Maximale Anzahl erfolgloser Optimierungsrunden bzw. Generationen Dies gibt an, wie viele Generationen erzeugt und ausgewertet werden, ohne dass eine Verbesserung des Kostenfunktionswertes festgestellt wurde. Je größer die Zahl ist, desto wahrscheinlicher kann ein lokales Optimum überwunden werden.

Anzahl der Kreuzungen pro Runde Pro Generation bzw. Optimierungsrunde werden ausgewählte Mitglieder der Population verwendet und eine Rekombination mit Erzeugung von mehreren Kindvektoren durchgeführt. Es können beliebig viele Rekombinationen je Runde durchgeführt werden, was zu einer entsprechend großen temporären Population führt. Da für alle Mitglieder der Population der Kostenfunktionswert ermittelt (also ggf. eine Simulation ausgeführt) wird sollte diese Anzahl mit Bedacht gewählt werden.

Mutationswahrscheinlichkeit Die Wahrscheinlichkeit, dass ein Gen bzw. ein Parameter mutiert, also zufällig geändert wird.

Als Abbruchkriterien dienen in der konkreten Implementierung einerseits eine absolute Zahl maximaler Optimierungs- bzw. Mutations-/Rekombinationsrunden, andererseits die variable Anzahl Optimierungsrunden ohne Verbesserung des Ergebnisses.

6.6.4 Zwei-Phasen-Optimierung

Analog zum Vorschlag in [73] wurde die Idee der zweiphasigen Optimierung umgesetzt. Ziel ist es, in einer ersten Voroptimierung ein „interessantes“ Gebiet im Definitionsbereich der Modellparameter zu finden und hiernach mit einer anderen Heuristik das Optimum in diesem Bereich zu finden.

Der wesentliche Unterschied zur, im Buch vorgestellten, Vorgehensweise liegt in der Verwendung der gleichen Kostenfunktion für beide Phasen. Um Zeit zu sparen, wird in der ursprünglichen Variante und den dort zitierten Quellen eine Approximation der Kostenfunktion bzw. Simulationsergebnisse gebildet. Diese wird anschließend für die Voroptimierung herangezogen und erst in der zweiten Phase werden tatsächlich Simulationen durchgeführt.
Die Beschleunigung im konkret implementierten Fall wird durch unterschiedliche Konfigurationen des Simulated Annealing Algorithmus erreicht, auf dem diese Heuristik basiert. Als Beispiel kann in der ersten Phase ein sehr schnelles Abkühlungsverfahren verwendet werden, um einen Bereich zu finden, der erst in der zweiten Phase genauer untersucht wird. Die Wahl einer passenden Konfiguration für die jeweilige Phase kann als klassische Kosten-Nutzen-Abwägung bzw. Exploration/Exploitation-Tradeoff verstanden werden. In einigen Punkten ähnelt die Zwei-Phasen-Optimierung mit Simulated Annealing einem Re-Annealing-Verfahren, wie in [29] vorgestellt.
Als Parameter stehen die in Kapitel 2.4.2 genannten $TRatioScale$ und $TAnnealScale$ zur Berechnung von c zur Verfügung sowie ϵ als Abbruchkriterium.

6.6.5 Mehr-Phasen-Optimierung

Die zweiphasige Optimierung bietet unabhängig davon, ob eine Approximation oder die tatsächliche Kostenfunktion in der ersten Phase verwendet wird, diverse Vorteile. Jedoch drängt sich die Überlegung auf, inwieweit die Verwendung weiterer Phasen Vorteile hinsichtlich der Zeit für eine Optimierung und der Qualität der gefundenen Optima birgt. Schon in den 1970er Jahren wurde dies in Betracht gezogen, um die simulationsbasierte Optimierung zu verbessern [52]. Die Ideen wurde immer wieder, –z.B. von Syrjakow– aufgegriffen und auch in prototypischen Werkzeugen umgesetzt [59, 58, 40].
In der hier beschriebenen konkreten Implementierung wurde der bekannte Ansatz in verschiedenen Punkten erweitert. So wird die Simulationsgenauigkeit phasenabhängig schrittweise erhöht und nur die letzte Optimierungsphase verlangt vom Simulationssystem maximale Genauigkeit. Dies geschieht durch Festlegung der externen Präzisionsparameter Konfidenzniveau und maximalem relativen Fehler (siehe 4.2.1). Zusätzlich kann ein interner Parameter phasenweise geändert werden. Falls

vom Modell unterstützt, kann damit ebenfalls die Präzision der Simulation modifiziert bzw. zwischen alternativen Implementierungen von Modellkomponenten umgeschaltet werden. Dies ist ein erster Schritt zur Umsetzung der genauigkeitsvariablen Optimierung, wobei das Optimierungssystem gezielt die Simulationsgenauigkeit einzelner Modellkomponenten steuern kann [69].
Darüber hinaus werden die Auflösung bzw. Diskretisierung sowie die Grenzen des Definitionsbereichs eines jeden Parameters pro Phase geändert. Beginnend mit einer sehr groben Diskretisierung unter kompletter Ausnutzung des Definitionsbereichs wird erst in der letzten Phase die festgelegte minimale Schrittweite erreicht, wobei dann mit entsprechend kleineren Grenzwerten der Parameter gearbeitet wird.
Allerdings gibt es auch Einschränkungen gegenüber anderen prototypischen Optimierungswerkzeugen mit mehrphasigen Heuristiken: So wird in der aktuellen Umsetzung die gleiche Basisheuristik in jeder Phase verwendet. Andere Umsetzungen erlauben die Verwendung verschiedener Heuristiken in jeder Phase.
Der konkrete Ablauf einer Mehr-Phasen-Optimierung eines Modells mit einem Parameter ist in Listing 3 dargestellt und soll hier kurz erläutert werden.
Die Startwerte für alle Modellparameter werden zufällig gesetzt. Prinzipiell ist auch hier die Festlegung auf Minimal-, Maximal-, oder Mittelwert möglich. Die Grenzen des Definitionsbereichs werden bestimmt und zusammen mit der Anzahl der Optimierungsphasen zur Berechnung der initialen Diskretisierung bzw. Parameterschrittweite genutzt. Nach jeder Phase wird später diese Schrittweite verdoppelt und die Ausdehnung des Definitionsbereichs halbiert. Jede neue Phase startet außerdem mit dem gefundenen Optimum aus der vorhergehenden Phase als Startwert der Modellparameter. Die ursprünglichen Grenzen des Definitionsbereichs (p_{min}, p_{max}) werden jedoch nie überschritten.
In der letzten Phase entspricht die Schrittweite jedes Parameters wieder der, die vom Nutzer vorgegeben wurde. Die Größe des Definitionsbereichs ist dabei jedoch nur ein Bruchteil des ursprünglich Gewählten, konkret $\frac{Orginialgroe„e}{2^{Phasenanzahl-1}}$.

In jeder Phase wird die gleiche Optimierungsheuristik angewendet. Unter anderem kann damit untersucht werden, wie Heuristiken mit theoretisch unbeschränkter Anzahl an Simulationsschritten von einer Beschränkung des Definitionsraumes profitieren.

$p := Random(p_{max}, p_{min})$
$p_{best} := p$
$phaseIndex := phaseCount - 1$
$range_p := p_{max} - p_{min}$
$stepSize_p := stepSize_p / 2^{phaseCount-1}$
while $phaseIndex \geq 0$ **do**
  $p_{best} := optimize(p_{best}, resolution_p, p_{min}, p_{max})$
  $range_p := \frac{range_p}{2}$
  $p_{min} := p_{best} - \frac{range_p}{2}$
  $p_{max} := p_{best} + \frac{range_p}{2}$
  $stepSize_p := \frac{stepSize_p}{2}$
end while
print p_{best}

Algorithmus 3 : Ablauf der mehrphasigen Optimierung

6.7 Kopplung an das Simulationstool

Damit die Optimierungssteuerung und später auch andere Programme eine stationäre SCPN-Simulation automatisch starten können, war es notwendig einige Änderungen an TimeNET durchzuführen. Diese sollten hier kurz erläutert werden, sie sind ab Version 4.2 enthalten und auf allen Plattformen verfügbar.
TimeNET selbst besteht aus diversen Komponenten, von denen nur eine die grafische Nutzeroberfläche ist. Diese ist in Java implementiert und wird entweder über eine mitgelieferte Batch-Datei gestartet oder mit dem üblichen „java -jar TimeNET.jar“-Befehl in der Kommandozeile des jeweiligen Betriebssystems. Nach dem Start zeigt die Nutzeroberfläche keine netzspezifischen Informationen und Steuermöglichkeiten an. Erst nachdem ein Netz geladen oder neu angelegt wurde, werden die Bedienelemente der Nutzeroberfläche angepasst. Der Editor und die Steuerung der Simulation ist für jede Netzklasse separat –ebenfalls in Java– implementiert. Für die automatisierte Simulation mussten somit 2 Komponenten geändert werden. Das Hauptpro-

gramm bzw. die GUI und die Netzklasse SCPN. Mit Hauptprogramm ist die Klasse *Peng* gemeint, welche die main-Methode enthält und beim Aufruf von TimeNET.jar gestartet wird. Hier wurde ein Parser für Kommandozeilenparameter hinzugefügt, welcher die folgenden Optionen verarbeitet:

Dateiname Der Name des zu simulierenden SCPN bzw. der xml-Datei inkl. dem vollständigen Pfad.

Zulässig sind Pfadangaben zu einem gültigen SCPN (xml-Datei) mit Systemspezifischen Seperatorzeichen.

autostart Die Option gibt an, ob das übergebene Netz sofort simuliert oder nur geöffnet werden soll.

Zulässige Werte sind „true“/„false“.

autostop Die Option gibt an, ob TimeNET nach der Simulation wieder geschlossen werden soll.

Zulässige Werte sind „true“/„false“.

secmax Die maximale Simulationslaufzeit in Sekunden. Nach dieser Zeit wird die Simulation abgebrochen, auch wenn nicht alle Leistungsmaße die geforderte Genauigkeit erreicht haben. „0“ bedeutet unbegrenzte Simulationslaufzeit[1].

Zulässig sind positive Integerwerte.

endtime Die maximalen Simulationszeitschritte, die von TimeNET simuliert werden bevor abgebrochen wird, auch wenn nicht alle Leistungsmaße die geforderte Genauigkeit erreicht haben. „0“ bedeutet unbegrenzte Anzahl an Simulationsschritten.

Zulässig sind positive Integerwerte.

seed Der zu verwendende Seedwert für den Zufallsgenerator. Falls 0 übergeben wird, verwendet TimeNET die aktuelle Systemzeit in Sekunden als Seed.

Zulässig sind positive Integerwerte.

[1] Es wird dringend empfohlen hier immer einen Wert größer 0 anzugeben.

confidence Das geforderte Konfidenzniveau für alle Leistungsmaße.

Zulässig sind positive Integerwerte wobei gilt:$(85 \leq confidence \leq 99)$.

epsmax Der erlaubte maximale relative Fehler für alle Leistungsmaße.

Zulässig sind positive Integerwerte wobei gilt:$(1 \leq epsmax \leq 15)$.

Unterstützt wird derzeit nur die stationäre Simulation von SCPNs. Nach Aufruf von TimeNET mit entsprechenden Parametern wird die Netzklasse geladen und startet selbstständig die Simulation. Hierfür waren kleinere Änderungen an der Netzklasse notwendig.
Während einer Simulation von SCPNs erzeugt TimeNET stets ein log-file mit allen Ergebnisdaten. Falls die Simulation fehl schlägt, ist die Datei leer. Diese Datei wird vom Optimierungswerkzeug ausgelesen. Für das erfolgreiche parsen der Log-Datei und die Übernahme aller relevanten Daten muss dem Optimierungsprogramm die Originaldatei des simulierten SCPN vorliegen. Die Daten über Parameterwerte und Namen von Leistungsmaßen werden mit den Ergebnisdaten aus der Logdatei zusammengeführt und intern in Objekten des Typs *SimulationType* abgelegt. Zusätzlich werden alle Ergebnisdaten in eine oder mehrere .csv-Dateien gespeichert. Dies erlaubt später die grafische Darstellung der Daten sowie das Einlesen als Ergebniscache und die Auswertung mit anderer Software.
Anschließend werden sämtliche Dateien, die im Zusammenhang mit der Simulation standen, gelöscht. Das betrifft die Log-Datei mit den Ergebnisdaten sowie alle Dateien, die von TimeNET für die Simulation angelegt wurden (c-Quelltexte, object-Dateien, ausführbares Simulationsprogramm). Prinzipiell kann dies auch unterbunden werden. Da pro Simulation jedoch sehr viele Dateien erzeugt werden (abhängig von der Anzahl der Modellelemente des simulierten SCPN) ist es sinnvoll, diese Dateien regelmäßig zu löschen. Der Speicher des Simulationsrechners wird ansonsten früher oder später überlastet.

6.8 Zeitverhalten

Die Messung des Zeitbedarfs einer Simulation ist von großer Bedeutung bei der Weiterentwicklung von Optimierungsheuristiken. Abhängig von geforderter Genauigkeit und weiteren Parametern wird unterschiedlich viel Zeit pro Simulationslauf

benötigt, wie in Kapitel 4 erläutert. Das Ziel einer Heuristik für simulationsbasierte Optimierung ist die Minimierung des Gesamtzeitbedarfs bei mindestens gleichbleibender Wahrscheinlichkeit, das globale Optimum zu finden.
Die Messung des Zeitbedarfs pro Simulation erfolgt durch TOE sekundengenau auf Basis der lokalen Systemzeit. Bedingt durch die Softwarearchitektur beinhaltet diese Zeit jedoch nicht nur die reine Simulation, sondern auch die Zeit, welche für das Erzeugen der C-Quelltexte aus dem jeweiligen Modell und deren Übersetzung notwendig ist.
Nach einer Optimierung wird die Summe der benötigten CPU-Zeit gebildet und kann zum Vergleich der Heuristiken herangezogen werden. Hierfür werden entsprechende statistische Daten gesammelt, insbesondere bei der wiederholten Optimierung. Dabei wird auch berechnet, wie viele der Simulationsergebnisse live berechnet wurden oder aus dem lokalen Ergebniscache geladen wurden. Daraus lassen sich ebenfalls interessante Erkenntnisse über das Verhalten der Heuristiken gewinnen. Eine Beispielausgabe nach einer Heuristikanalyse mit 100 Optimierungen ist in Listing 6.8 zu sehen.

```
++++ Start of Optimization Statistics ++++
Number of Optimization runs: 100
Average distance to target value (0.0/min): 12.343499999999988
Average number of Simulations: 28.13 ( 21.41 | 0.0 | 6.72 )  (Local|Web|Cache)
Average CPU Time used: 28889.51 ( 21988.07 | 0.0 | 6901.44 )  (Local|Web|Cache)
Average Simulation Steps: 281.3 ( 214.1 | 0.0 | 67.2 )  (Local|Web|Cache)
Average Cache Ratio (All/Total): 0.1737737195388394
Average relative distance to calculated optimum in value range: 1.9749599999999985 %
Average relative (EUKLID) distance to calculated optimum in definition range 21.507304673015007 %
++++ End of Optimization Statistics ++++
```

Um Heuristiken schneller analysieren zu können, sind Benchmarkfunktionen als alternative Simulatoren implementiert. Damit der Bedarf an CPU-Zeit für eine Optimierung abgeschätzt werden kann und somit die Heuristiken bzgl. ihres Gesamtzeitverhaltens vergleichen zu können, auch wenn keine echte SCPN-Simulation durchgeführt wurde, wird das Zeitverhalten echter Simulationen nachgebildet.
Versuche mit verschiedenen SCPNs zeigen, dass die benötigte CPU-Zeit für einen Simulationslauf mit einem Polynom abgeschätzt werden kann. Die Ergebnisse der Versuche sind in den Bildern 4.5 und 4.3 zu sehen. Bei relativ einfachen SCPNs, deren Leistungsmaße sehr schnell konvergieren, hat die verlangte Genauigkeit keinen messbaren Einfluss auf die Zeit pro Simulation. Lediglich die Anzahl simulierter Zeitschritte ist, wie in Bild 4.4 zu sehen, abhängig von der geforderten Simulationspräzision. Dies ist jedoch für die Beschleunigung der simulationsbasierten Optimierung

nicht relevant.
Bei komplexeren Netzen wie in Kapitel 7 vorgestellt, haben Parameter wie Konfidenzniveau und maximaler relativer Fehler einen größeren Einfluss auf die benötigte CPU-Zeit pro Simulation, insbesondere im jeweiligen Grenzbereich (99 % & 1 %). Das Polynom zur näherungsweisen Berechnung der benötigten CPU-Zeit ist in Formel 4.2 zu sehen. Die Parameter können derzeit nur im Quelltext der Optimierungssteuerung verändert werden. Sie basieren auf einer numerischen Annäherung im Vergleich zur Messung des CPU-Zeitbedarfs des Energiemodells aus Kapitel 7 und können anhand der Datei **messungen/time/functionsbestimmung.ods** auf der beiliegenden CD nachvollzogen werden.

6.9 Verteilte Simulation mit TOE

Die verteilte Simulation von SCPNs wurde entwickelt, um die Erstellung von Ergebnisdatenbanken zu vereinfachen und um Optimierungsverfahren zu beschleunigen, die je Iteration mehrere Parameterkonfigurationen simulieren, wie z.B. populationsbasierte Heuristiken. Das Gesamtsystem ist im Verteilungsdiagramm 6.5 dargestellt und besteht aus 3 Komponenten.

Simulation Master Eine Instanz von **TimeNET Optimization Environment**. Diese wird vom Nutzer gesteuert und startet eine verteilte Simulation durch Übertragung einer entsprechenden xml-Datei an den Server.

Simulation Client Eine Instanz von **TimeNET Optimization Environment**. Sie läuft ohne Nutzereingaben auf einem entfernten Rechner und meldet sich zyklisch beim Simulationsserver.

Simulationsserver Eine Webapplikation, die Simulationsdateien entgegen nimmt, alles koordiniert und statistische Daten bereit stellt.

Sämtliche Kommunikation zwischen Simulation Master, - Server und - Client erfolgt über HTTP-(POST/GET)-Requests. Der Ablauf einer verteilten Simulation ist dabei wie folgt:
Ein Simulation Master startet eine verteilte Simulation durch Übertragung beliebig vieler XML-Dateien von SCPNs an den Server. Alle Simulation Clients fragen zyklisch beim Server an, ob SCPNs zur Simulation vorliegen. Ist dies der Fall, lädt ein

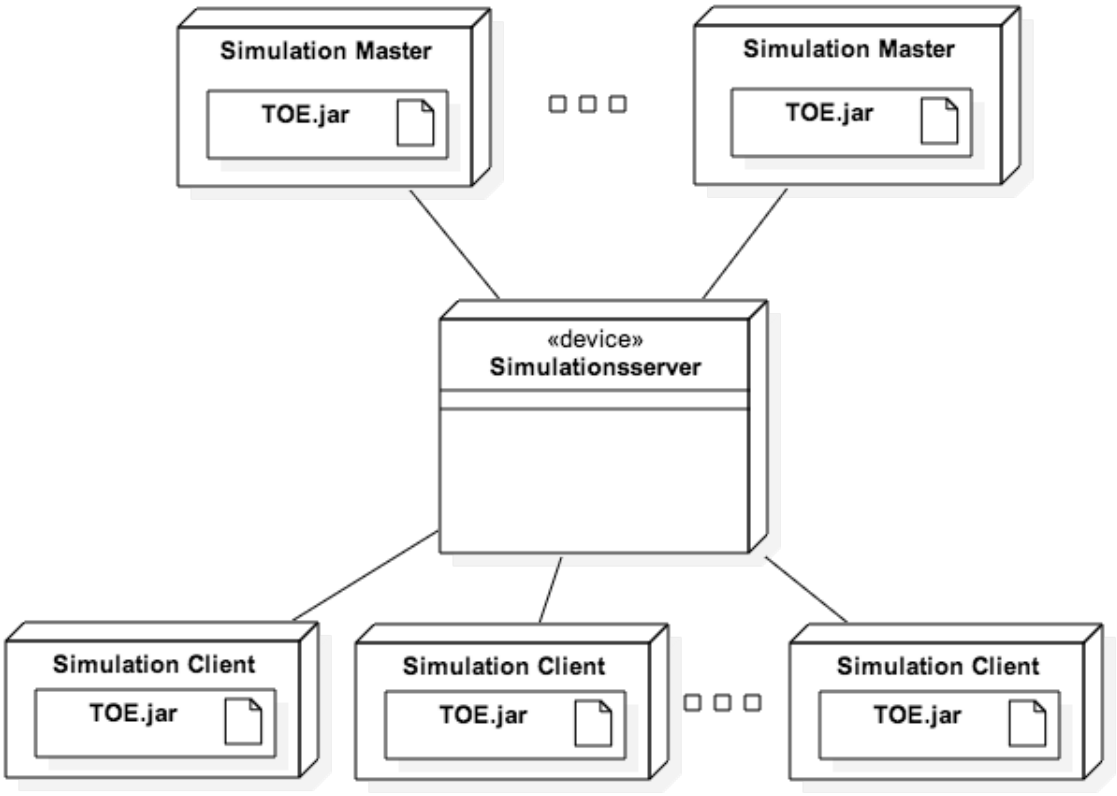


Abbildung 6.5: Verteilte Simulation mit TOE und Simulationsserver, Verteilungsdiagramm

Client eine xml-Datei herunter, startet die Simulation und lädt nach deren Abschluss die resultierende Log-Datei wieder hoch. Währenddessen erfragt der Simulation Master zyklisch den Status der hochgeladenen Simulationen bzw. xml-Dateien. Liegen erste Simulationsergebnisse vor, werden diese heruntergeladen und anschließend vom Server gelöscht.

Sicherheitsaspekte standen bei der Entwicklung nicht im Vordergrund. Weder Client noch Simulation Master müssen sich gegenüber dem Server authentifizieren. Die gesamte Kommunikation ist unverschlüsselt. Lediglich ein minimaler Schutz gegen versehentliches Löschen von Simulationsdateien oder Ergebnissen wurde implementiert. So kann jedem Client eine Zeichenkette zur Identifikation zugewiesen werden und die Simulationen dieses Clients können nur mit Kenntnis dieser Zeichenkette gelöscht werden. Ein ähnliches Verfahren wird für das Zurücksetzen des Servers und damit Löschen aller anstehenden Simulationen verwendet.

6.10 Zusammenfassung und Ausblick der Entwicklung von TOE

Im Laufe der Entwicklung von TOE entstand ein Werkzeug zur Analyse und Optimierung farbiger Petri-Netze durch Simulation mit TimeNET. Verschiedene Optimierungsheuristiken können hiermit untersucht werden, sowohl anhand von Benchmarkfunktionen als auch an echten Simulationen. Dennoch steht die Entwicklung hin zu einem universell nutzbaren Optimierungswerkzeug noch am Anfang. Dies sollen einige Beispiele für geplante zukünftige Erweiterungen deutlich machen.

Netzklassen / Simulationstypen Derzeit wird nur die stationäre Simulation von SCPNs unterstützt. TimeNET bietet jedoch neben weiteren Netzklassen auch alternative Simulations- und Analysemöglichkeiten. Deren Unterstützung ist für die Zukunft angestrebt.

Simulationstools Die Anbindung von TimeNET ist derzeit relativ strikt im Programm verankert. Eine Erweiterung um die Ansteuerung alternativer Simulationstools wie MLDesigner oder MatLab Simulink befindet sich in der Entwicklung.

Verteilte Simulation Diese Art der Simulation bietet großes Potential für die schnelle Analyse von Modellen durch Simulation. Jedoch sind noch viele Verbesserungen denkbar und teilweise dringend erforderlich. Dies umfasst Sicherheitsaspekte wie auch Kontrolle durch den Nutzer. Momentan kann eine vorsätzliche als auch fahrlässige Störung des gesamten Simulationssystems inkl. aller Client-Computer nicht ausgeschlossen werden.

Benutzerfreundlichkeit Eine leicht verständliche Bedienung stand stets mit im Fokus der Entwicklung. Nach Möglichkeit sollte jeder, der bereits SCPNs mit TimeNET simuliert hat, mit dem Optimierungswerkzeug umgehen können. Dennoch können noch viele Punkte verbessert werden. Insbesondere der Umgang mit Fehlern, die Einrichtung der Software und das Feedback der Optimierungsheuristiken bedarf noch weiterer Überarbeitung.

Neben den genannten Punkten sind noch viele weitere Verbesserungen denkbar, wie alternative Analysemöglichkeiten der Optimierungsheuristiken, eine verbesserte Zeitmessung, etc...

Um dies jedem interessierten Anwender zu ermöglichen, wurde der Quelltext entsprechend dokumentiert und auf einer bekannten Plattform unter der Adresse `https://github.com/ChristophBodenstein/TimeNETOptimizationEnvironment` veröffentlicht.

7 Modellierung von Energieflüssen mit SCPNs

Farbige stochastische Petri-Netze werden vielfach zur Abbildung von Fertigungs- und Logistikprozessen genutzt. Aber auch bei der Simulation von Wasserversorgungssystemen in Katastrophensituationen konnten sie wertvolle Informationen liefern [26, 72].
In diesem Kapitel wird erläutert, wie SCPNs zur Modellierung von kontinuierlichen Systemen wie Energienetzen verwendet werden können. Anhand eines Netzes wird beschrieben, wie kontinuierliche und zeitabhängige Vorgänge simuliert werden. Anschließend werden die beschriebenen Techniken verwendet, um eine optimale Konfiguration des Modells zu bestimmen.
Das gezeigte Modell eignet sich ebenso für die spätere Weiterentwicklung, z.B. um Optimierungsheuristiken umsetzen zu können, die ihrerseits spezifische Anforderungen hinsichtlich Architektur und Modellelementen haben.
Es zeigt es den praxisnahen Einsatz von Petri-Netzen zur Gewinnung von Erkenntnissen über reale Systeme bzw. deren Konfigurationen.

7.1 Ziel des Modells

Energieflüsse und die Einsparung von Energie ist ein aktuelles Thema. Neue Technologien und eine dezentrale Gewinnung elektrischer Energie stellen eine große Herausforderung für die zukünftige Entwicklung von Energieversorgungsnetzen, individueller Mobilität, Lebensgewohnheiten sowie Wohnsituationen dar. Vor diesem Hintergrund soll ein typisches Niedrigenergiehaus hinsichtlich einer optimalen Dimensionierung von Photovoltaikanlage und Speicherbatterie untersucht werden. Ziel einer simulationsbasierten Optimierung ist das Finden der kostengünstigsten Konfi-

guration bei gegebenem Durchschnittsverbrauch. Nachdem verschiedentlich gezeigt wurde, dass Petri-Netze sehr gut geeignet sind, um Produktionsabläufe, Warenverkehr und ähnliche Prozesse zu modellieren, verdeutlicht dieses Modell nun, wie man das Verhalten von kontinuierlichen Systemen modelliert und die Genauigkeit der Simulation anhand der gewählten Diskretisierung festlegen kann. Eine Übertragung dieser Vorgehensweise auf andere Systeme wie Wasserkreisläufe ist angestrebt.

7.2 Modellelemente und gewählte Abstraktionen

Jedes Modell stellt nur einige Aspekte des zu optimierenden realen Systems dar. Im Fall von Energieflüssen stellt sich zunächst die Frage, wie der kontinuierliche Fluss von elektrischer Energie mit Petri-Netzen dargestellt werden kann. Ein erster Versuch, jeder Marke eine Farbe in Form eines repräsentativen Energiewertes zuzuordnen, stellte sich als unzureichend heraus, da es die Evaluation von entsprechenden Leistungsmaßen durch das Simulationssystem erschwerte.
Stattdessen wurde eine Abstraktion umgesetzt, bei der jeder Marke im Netz ein Kilowatt (KW) bzw. ein Bruchteil dessen zugeordnet ist. Die Produktion und der Konsum der Marken pro Zeiteinheit (durch Transitionen) entspricht dabei der Erzeugung und dem Verbrauch von Energie pro Zeit, im Normalfall gemessen in Kilowattstunden (KWh). Alle Berechnungen im Modell sind so ausgelegt, dass Modellparameter in üblichen Einheiten, also KWh, angegeben werden und auch die Ergebnisse in diesen Einheiten vorliegen.
Zu lösende Fragestellungen waren die konkreten Umrechnungsvorschriften, die Modellierung zeitlicher Abläufe sowie die Darstellung von Messdaten in Form von evaluierbaren Leistungsmaßen.
Das Gesamtmodell ist in Bild 7.1 zu sehen. Es ist ein hierarchisches Netz, wobei nur der Haushalt und die Batterie tatsächlich als Subnetz implementiert sind. Die Bezeichnungen sind einheitlich in Englisch gehalten.
Wichtigste Komponente ist der zentrale Platz *MainEnergy*. Er symbolisiert die elektrische Verbindung aller anderen Komponenten. Wird Energie erzeugt, so werden Marken in diesem Platz erzeugt. Energieverbrauch wird durch Konsum von Marken aus diesem Platz dargestellt. Die Wahrscheinlichkeit, dass die Markenanzahl in diesem Platz 0 ist –$P(\#MainEngery \leq 0)$– entspricht demnach der Wahrscheinlichkeit

einer Überlastung des Stromnetzes (Kurzschluss).

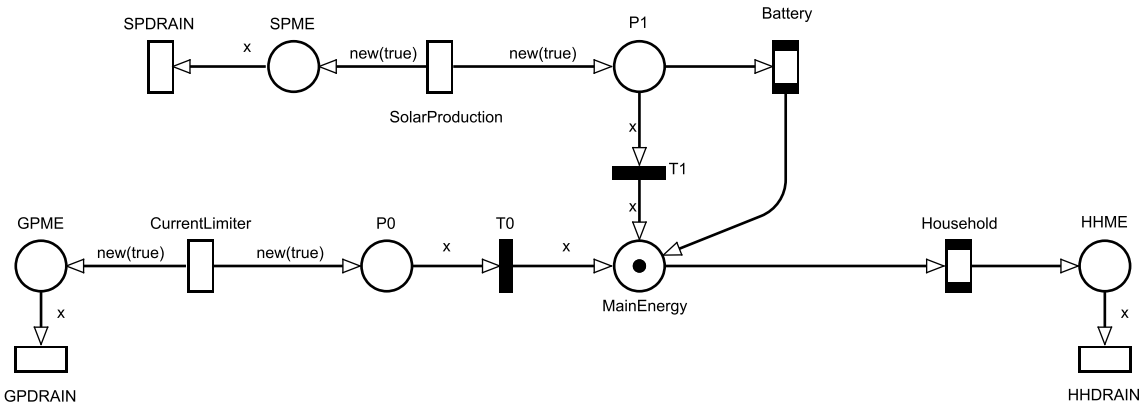


Abbildung 7.1: Übersicht des Modells eines Niedrigenergiehauses

7.2.1 Energieerzeugung

Die Energieerzeugung erfolgt an zwei Stellen, repräsentiert durch Transitionen. Die Produktion durch die Photovoltaikanlage wird durch die Transition *SolarProduction* modelliert.

Sie schaltet ohne Vorbedingungen mit exponentieller Schaltzeitverteilung, wobei die Schaltzeit nach Formel 7.1 anhand der gewählten Kapazität und des Effizienzfaktors berechnet wird. Pro Schaltvorgang wird jeweils eine Marke in *P1* und in *SPME* erzeugt. *P1* hat eine Kapazität von 1 und dient als Verbindung zwischen Stromerzeugung und -verbrauch, ähnlich wie *MainEnergy*. Es stellt einen sehr kleinen Zwischenpuffer dar. Ist hierin eine Marke enthalten, kann sie sofort von *T1* für den Hausenergiebedarf konsumiert werden. Alternativ kann sie im Inneren von *Battery* konsumiert werden, um damit einen Akkumulator in Form des Platzes *Tank* (nicht mit abgebildet) zu laden.

Die Produktion von Strom, der aus dem öffentlichen Netz in den Haushalt eingespeist wird, ist nicht direkt im Modell integriert. Stattdessen wird der Zufluss bzw. dessen Begrenzung durch die Transition *CurrentLimiter* umgesetzt. Abhängig von der konfigurierten Systemspannung und der maximalen Stromstärke am Hausanschluss wird ihre (deterministische) Schaltzeit berechnet. Für die Versuche wurde eine Spannung von 220V bei maximal 16A angenommen. Die konkrete Schaltzeitberechnung von *CurrentLimiter* ist in Formel 7.2 zu finden.

CurrentLimiter und *SolarProduction* schalten nur dann, wenn zuvor Energie von ihnen „angefordert“ wurde, d.h. wenn der korrespondierende Platz *P0/P1* leer ist. Ist dies nicht der Fall, so wird die Solaranlage keinen Strom, also keine Marken produzieren, was in der Realität einer Effizienzminderung der Anlage entspricht.
Zusammengenommen kann damit Strom aus drei möglichen Quellen ins lokale Netz, also in *MainEnergy* eingespeist und konsumiert werden. Über Schaltprioritäten kann festgelegt werden, welche Quelle bevorzugt wird. Höchste Priorität hat im verwendeten Modell die Transition *T1*. Dies bedeutet, wann immer Strom vom lokalen Netz angefordert wird bzw. *MainEnergy* leer ist und Strom von der Solaranlage zur Verfügung steht, wird dieser auch genutzt. Liefert die Solaranlage in diesem Moment keine Energie, wird der Bedarf aus der Batterie gedeckt. Erst wenn auch dies nicht möglich ist, schaltet *T0* und es wird Strom aus dem öffentlichen Netz genutzt.
Um die durchschnittliche Menge Strom zu bestimmen, die jährlich durch die Solaranlage oder aus dem öffentlichen Netz eingespeist wird, ist ein Hilfskonstrukt notwendig. Jeder Schaltvorgang von *CurrentLimiter* oder *SolarProduction* bewirkt auch die Produktion einer Marke in einem der verbundenen Plätze *SPME* bzw. *GPME*. Jeder dieser Plätze wird mit einer definierten Frequenz geleert, die nur abhängig vom –später näher beleuchteten– Präzisionsfaktor ist. Aus der durchschnittlichen Markenanzahl in diesen Plätzen kann daher die jährliche Produktion der Solaranlage und der Verbrauch aus dem öffentlichen Netz nach Formel 7.3 und 7.4 errechnet werden.

$$t_{solar} = \frac{365 * 24}{(SolarCapacity * SolarEfficiencyFactor * 1000 * Precision)} \tag{7.1}$$

Berechnung der Schaltzeit für *SolarProduction*

$$t_{solar} = \frac{365 * 24}{(GridVoltage * GridCurrent * 1000 * Precision)} \tag{7.2}$$

Berechnung der Schaltzeit für *CurrentLimiter*

$$SPPY = 365 * 24 * \#GPME \tag{7.3}$$

Berechnung des jährlichen Stromkonsums aus *GPME*

$$SPPY = 365 * 24 * \#SPME \tag{7.4}$$

Berechnung der jährlichen Solarstromproduktion aus *SPME*

7.2.2 Energieverbrauch / Haushalt

Der Konsum von Energie durch elektrische Verbraucher im Haushalt wurde auf ähnliche Weise abstrahiert wie die Stromerzeugung. Grundsätzlich werden die Marken aus *MainEnergy* durch die Transition *HouseHold* konsumiert und jeweils eine neue Marke in *HHME* erzeugt. Dies dient wiederum als Hilfskonstrukt zur Messung des Durchflusses, also des Stromverbrauchs durch den Haushalt. *HHME* wird durch *HHDRAIN* mit konstanter Schaltgeschwindigkeit geleert, weshalb sich der jährliche Verbrauch aus der durchschnittlichen Markenanzahl in HHME errechnen lässt (Formel 7.5).

$$HHCPY = 365 * 24 * \#HHME \tag{7.5}$$

Berechnung des jährlichen Stromverbrauchs aus *HHME*

Der Haushalt bzw. dessen Verbrauch ist in eine Substitutionstransition eingebettet, analog zur Batterie-Transition. Dies dient nicht nur der Übersichtlichkeit, sondern ermöglicht die Verwendung verschiedener Implementierungen der gleichen Komponente.
Die SCPN Netzklasse in TimeNET bietet entsprechende Möglichkeiten, welche von verschiedenen Autoren zum Thema Hierarchische Petri-Netze vorgeschlagen wurden. Eine Substitutionstransition kann beliebig viele Implementierungen beherbergen. Besonderheit bei TimeNET ist, dass jede dieser Implementierungen direkt über eine Eingangskante mit dem Platz in der nächsthöheren Hierarchieebene verbunden

ist. Das bedeutet, falls keine weiteren Bedingungen erfüllt sein müssen, sind alle Implementierungen gleichzeitig schaltfähig. Dies kann zu ungewollten Konfliktsituationen führen. Um Konflikte zu verhindern und verschiedene Implementierungen vergleichen zu können, wurde im vorgestellten Modell eine weitere Bedingung in der Eingangstransition jeder Implementierung eingeführt. Auf diese Weise konnte auch die prinzipielle Realisierbarkeit der in 5.4.3 vorgestellten Ideen sowie jene zur Architekturoptimierung mit TimeNET gezeigt werden.
Auf diese Weise wurden zwei Modellvarianten eines Haushalts in dieser Substitutionstransition realisiert. Variante 1 (Bild 7.3) zeigt eine stark vereinfachte Annäherung, eine Transition, deren Schaltzeit aufgrund des angegebenen Jahresdurchschnittsverbrauchs berechnet wird und während der Simulation konstant ist (Formel 7.6). Von dieser Transition abhängige Leistungsmaße konvergieren, abhängig von der berechneten Schaltzeit, vergleichsweise schnell. Gemäß den in Kapitel 4 erläuterten Bedingungen kann diese Variante daher mit wenig Rechenaufwand simuliert werden.

$$t_{house} = \frac{365 * 24}{(HouseholdYearlyUsage * Precision)} \tag{7.6}$$

Berechnung der Schaltzeit für *Consumption0*

Variante 2 (Bild 7.2) bildet eine etwas detailliertere Sichtweise auf den Haushaltsstromverbrauch ab. Ausgehend vom gleichen Durchschnittsverbrauch wird hier dem unterschiedlichen Nutzerverhalten in Abhängigkeit von der Tageszeit Rechnung getragen. Grundsätzlich wurde in Tag- und Nachtzeit unterschieden und dies mit zwei Transitionen realisiert. Es kann im Modell jeweils nur eine Transition aktiv sein, die andere wird über eine globale Schaltbedingung (global guard) blockiert. Mit der Definition **daytimeconsumption** kann festgelegt werden, welcher Anteil des Gesamtverbrauchs am Tag konsumiert wird. Der Rest entfällt automatisch auf die Nachtzeit. Die Schaltzeiten der Transitionen werden abhängig davon mit den Formeln 7.7 berechnet und bleiben während der Simulation konstant. Die beiden Transitionen werden über ihre Schaltbedingungen abhängig von der simulierten Uhrzeit wechselseitig aktiviert. Beginn und Ende eines Tages werden über 2 Definitionen festgelegt.
Erwähnenswert in diesem Zusammenhang ist die Implementierung der Uhr. Zwar

gibt es in TimeNET durchaus das Konzept einer Simulationszeit, die Auswertung dieser bzw. Nutzung als Schaltbedingung gestaltet sich jedoch im Einzelfall schwierig. Um eine möglichst universell verwendbare Uhr nutzen zu können, ist eben diese als Substitutionstransition in der Variante 2 des Haushaltsverbrauchs implementiert (Bild 7.4). Zentrale Transition dieser Uhr ist *HouseDayTime*, deren Markenanzahl die aktuelle Uhrzeit im Modell wieder spiegelt. Als Zeitbasis für das gesamte Modell wurden Stunden gewählt. *HouseClock* schaltet somit mit einer deterministischen Frequenz von 1. Hat *HouseDayTime* die Markenanzahl 24 erreicht, wird *T4* über eine globale Schaltbedingung aktiviert und *T4* setzt die Uhrzeit zurück auf 0. Dadurch wird *T5* aktiviert und die Stundenzählung beginnt von vorn.

$$dayTime = DayEnd - DayStart$$

$$t_{houseDay} = \frac{365 * dayTime}{(HouseholdYearlyUsage * Precision * DayTimeConsumption)}$$

$$t_{houseNight} = \frac{365 * (24 - dayTime)}{(HouseholdYearlyUsage * Precision * (1 - DayTimeConsumption))} \tag{7.7}$$

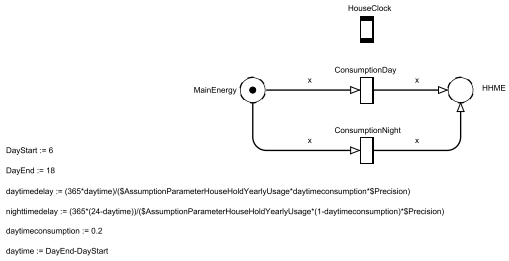


Abbildung 7.2: Exaktere Implementierung des Energieverbrauchs eines Haushaltes

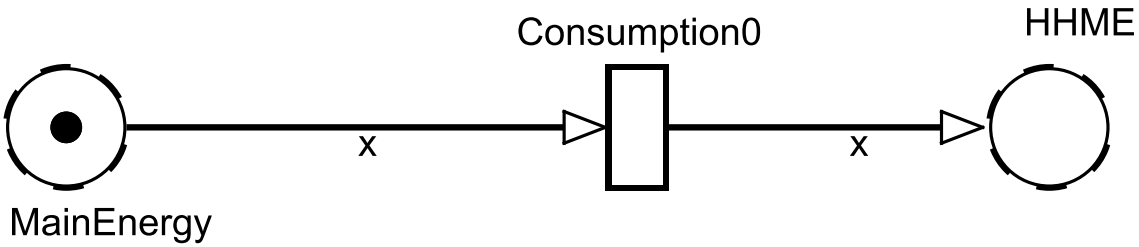


Abbildung 7.3: Vereinfachte Implementierung des Energieverbrauchs eines Haushaltes

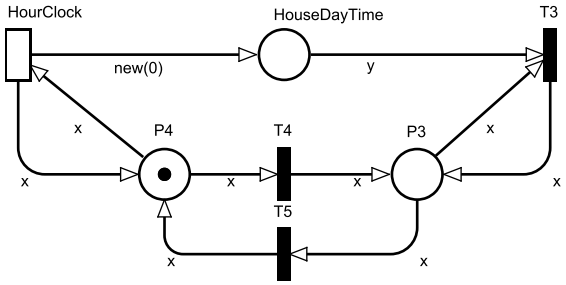


Abbildung 7.4: Modell zur Uhrzeitbestimmung in SCPNs

Leistungsmaße

Das offensichtliche Leistungsmaß für die Optimierung sind die Gesamtkosten des Systems, welche sich aus Kosten für Strom aus öffentlichem Netz und Anschaffungskosten für Solaranlage sowie Batterie zusammensetzen. Laufende Kosten in Form von Reparaturen und Verschleiß wurden im verwendeten Modell nicht nachgebildet. Die Gesamtkosten sind im Leistungsmaß *CostSum* erfasst, welches im Rahmen der späteren Optimierung minimiert werden soll.
Die Kosten für die Batterie wie auch für die Solaranlage sind gestaffelt je nach Größe. Die jeweiligen Grenzwerte und Kosten pro KWh sind im Modell als Variablen sichtbar und können demzufolge stets aktuell gehalten werden. Die Berechnung der jeweiligen Kosten erfolgt gemäß der betroffenen Preisgruppe zu Beginn jeder Simulation.
Der aktuelle Verbrauch aus dem öffentlichen Netz führt typischerweise zu einer Eingruppierung in Stromtarife oder zu entsprechenden Rabatten. Je mehr ein Teilnehmer verbraucht, desto weniger muss er pro KWh zahlen. Da der Verbrauch erst zur Laufzeit der Simulation auf einen Wert konvergiert bzw. auch stark schwanken kann, muss eine entsprechende Berechnung des aktuellen Stromtarifs ebenfalls zu Laufzeit

erfolgen. Um dies zu ermöglichen, ist ein kleines Hilfsnetz integriert, welches in Bild 7.5 zu sehen ist. Überschreitet der aktuell berechnete jährliche Verbrauch an Strom aus dem öffentlichen Netz einen bestimmten Schwellwert, schaltet *TGP12*. Wird der Wert unterschritten, so schaltet *TGP21*. Gleiches gilt für *TGP23* und *TGP32*. Somit ist stets eine Marke in einem der Plätze *GP1*, *GP2* oder *GP3*, was die aktuell gewählte Preisgruppe bzw. den Stromtarif repräsentiert. Der Preis für den gesamten Strom, der pro Jahr aus dem öffentlichen Netz bezogen wird, berechnet sich nach Formel 7.8.

$$\begin{aligned} GridCost = \#GP1 * PricePerKW_1 \\ +\#GP2 * PricePerKW_2 \\ +\#GP3 * PricePerKW_3 \end{aligned} \tag{7.8}$$

Berechnung der jährlichen Kosten für Netzstrom

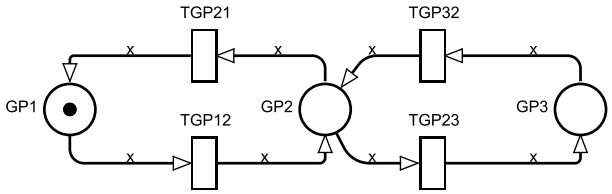


Abbildung 7.5: Hilfsmodell zur Preisberechnung

Ein weiteres interessantes Leistungsmaß ist die Auslastung der Batterie. Aufgrund ihrer hohen Anschaffungskosten ist es interessant zu wissen, inwieweit die Batterie genutzt wird bzw. ob überhaupt zeitweise mehr Strom von der Solaranlage produziert wird, als vom Haushalt verbraucht werden kann. In der Realität kann dieser überflüssige Solarstrom zwar auch ins öffentliche Netz eingespeist werden. Zur Vereinfachung ist dies aber im verwendeten Modell nicht umgesetzt. Die Batterie wird durch eine eingehende und eine ausgehende Transition nachgebildet sowie durch den Platz *Tank*, dessen Kapazität durch einen Konfigurationsparameter festgelegt werden kann. Die Auslastung der Batterie berechnet sich nach Formel 7.9

$$t_{solar} = \frac{100 * \#Tank}{BatteryCapacity * Precision} \tag{7.9}$$

Berechnung der durchschnittlichen Batterieauslastung

Da im aktuellen Modell kein Solarstrom produziert werden kann, wenn kein Strom abgenommen wird, ist die Betrachtung der jährlichen Stromproduktion aufschlussreich. Wenn jedoch eine Einspeisung des produzierten Stromes in das öffentliche Netz beachtet wird, ist diese Information prinzipiell überflüssig, da die Anlage dann stets mit maximal möglicher Leistung arbeiten kann. Der Verbrauch eines Haushaltes pro Jahr ist zwar als Leistungsmaß im Modell angelegt, jedoch weniger interessant. Der durchschnittliche Verbrauch eines Haushaltes dient als Eingangsdatum für die Simulation. Seine Aufteilung auf Tages- und Nachtzeit sowie die Auswirkung auf die Gesamtkosten sind daher von Interesse und fließen implizit in den Netzstromverbrauch und die Gesamtkosten ein. Aus Gründen der Anschaulichkeit ist dieses Leistungsmaß dennoch im Modell.

7.3 Genauigkeitssteuerung über internen Parameter

In den Formeln für Leistungsmaße und Schaltzeiten tauchte bereits mehrfach der Parameter *Precision* auf. Über diesen Parameter kann die Genauigkeit der Simulation beeinflusst werden. Er reguliert die zeitliche Diskretisierung der Simulationszeit und steht bei der Berechnung aller Transitionsschaltzeiten im Nenner. Der Parameter kann Werte größer 0 annehmen, wobei 1 bedeutet, dass ein Simulationszeitschritt von TimeNET einer Stunde in der Realität entspricht. Je größer *Precision* gewählt wird, desto exakter wird die Simulation. Die Schaltzeiten werden dadurch kleiner, die reale Zeit in der Simulation feiner aufgelöst. Zum Verständnis ist im Modell der Bezug zwischen Marken und Elektrischer Leistung für $Precision = 1$ angegeben. Dabei entspricht eine Marke einem Kilowatt. Eine durchschnittliche Markenanzahl von eins in *HHME* bedeutet in diesem Fall einen durchschnittlichen Verbrauch von 1 KWh pro Simulationszeitschritt, da *HHME* mit der deterministischen Schaltzeit 1 von *HHDRAIN* entleert wird und Stunden als Zeitbasis angenommen wird.
Durch Erhöhung von *Precision* wird die Zeitbasis der Simulation schrittweise verkleinert, wobei dies in den Formeln zur Berechnung der Leistungsmaße berücksichtigt ist. Ein Wert von 1000 bedeutet demnach, dass eine Marke einem tausendstel Watt entspricht und ein Simulationszeitschritt dem Tausendstel einer Stunde Realzeit.
Dadurch bewirkt die Erhöhung von *Precision* eine exaktere Simulation bei gleichzeitig erhöhtem Bedarf an Rechenzeit. Diese Auswirkung der Variation von *Precision*

auf benötigte Rechenzeit und Varianz der Leistungsmaße ist in Tabelle 7.2 zu erkennen.

7.4 Validierung des Modells

Um den praktischen Nutzen des Modells nachzuweisen, ist eine entsprechende Validierung notwendig. Dies wurde anhand der beiden Behelfs-Leistungsmaße für Stromverbrauch aus dem Netz und Gesamtverbrauch des Haushalts durchgeführt. Die Kapazität der Solaranlage wurde hierfür auf 0 gesetzt. Trotz des exponentiellen Schaltverhaltens der Verbrauchstransitionen sollte der Eingangswert für den durchschnittlichen Jahresverbrauch ungefähr dem Wert des entsprechenden Leistungsmaßes entsprechen. Außerdem sollte die Summe aus Netzstrom und Solarstrom stets konstant und gleich dem Jahresdurchschnittsverbrauch sein.
Das Ergebnis ist in Tabelle 7.1 zu sehen. Die Versuche wurden jeweils mit maximaler Genauigkeit (Konfidenzniveau und maximalem relativen Fehler) durchgeführt. Die vereinfachte Implementierung des Energieverbrauchs eines Haushaltes (Variante 1) lieferte eine maximale Abweichung von 5,3 %, die exaktere Implementierung mit simulierter Uhrzeit bis zu 20,5 %.
Um den Einfluss des internen Präzisionsparameters zu untersuchen, wurden Simulationen mit Werten von 1 bis 100 hierfür durchgeführt. Die Ergebnisse sind in Tabelle 7.2 zu sehen. Es ist zu erkennen, dass mit größerem Wert für *Precision* die Abweichung vom Idealwert sowie die Varianz der Messwerte sinkt. Außerdem steigt die benötigte CPU-Zeit schrittweise an. Beides entspricht dem erwarteten Verhalten.
Die Erhöhung der CPU-Zeit ist nicht linear abhängig vom verwendeten Wert für *Precision*. Dies liegt am Abbruchkriterium für stationäre SCPN-Simulationen. Eine Simulation wird beendet, sobald alle Leistungsmaße innerhalb der vorgegebenen Präzisionsanforderungen (Konfidenzniveau und maximalen relativen Fehler) liegen. Mit steigendem Wert für *Precision* wird zwar mehr CPU-Zeit pro Simulationszeitschritt benötigt, jedoch werden auch weniger Simulationszeitschritte berechnet, bevor das geforderte Genauigkeitsmaß erreicht wird. Wie viele Simulationszeitschritte berechnet werden, entscheidet das Simulationstool TimeNET. Um den Zusammenhang zu verdeutlichen, ist das Verhältnis zwischen Simulationszeit (Anzahl an Simulationszeitschritten), CPU-Zeit und Wert für *Precision* in Bild 7.6 dargestellt.

Eingabewert	Messung	Abweichung (%)
Variante 1		
500	513	2,6
1000	1036	3,7
2000	2106	5,3
3000	3142	4,7
4000	4173	4,3
5000	5205	4,1
6000	6238	4,0
Variante 2		
500	398	20,5
1000	856	14,4
2000	1710	14,5
3000	2576	14,1
4000	3635	9,1
5000	4335	13,3
6000	5295	11,7

Tabelle 7.1: Validierung anhand des Jahresdurchschnittsverbrauchs

Das Optimierungsproblem für dieses Modell besteht darin, die günstigste Kombination aus Solaranlage und Batteriegröße zu finden. Daher haben diese beiden Parameter im Modell das Präfix „Config“, während alle anderen Parameter während der Optimierungsläufe konstant bleiben. Zur besseren Verständlichkeit zeigt Bild 7.7 die entstehende Wertelandschaft des Leistungsmaßes für die Gesamtkosten bei Variation der Parameter **SolarCapacity** und **BatterCapacity**. Zu sehen ist, dass die Wertelandschaft viele lokale Optima ausweist, was die Suche nach der kostengünstigsten Konfiguration erschwert bzw. für die Untersuchung verschiedener Heuristiken interessant macht.

Precision	CPU-Zeit	Simulations-zeitschritte	Varianz
1	580	12000	180693,1
10	1025	7550	170932,4
20	1041	3350	111150,2
30	1089	2150	76043,3
40	1223	1700	57431,69
50	1376	1350	45827,72
60	1264	1100	37701,86
70	1219	900	32096,94
80	1531	800	28449,8
90	1396	700	25129,88
100	1514	650	23132,5

Tabelle 7.2: Verhältnis von internem Präzisionsparameter zu CPU-Zeit und Varianz eines Leistungsmaßes

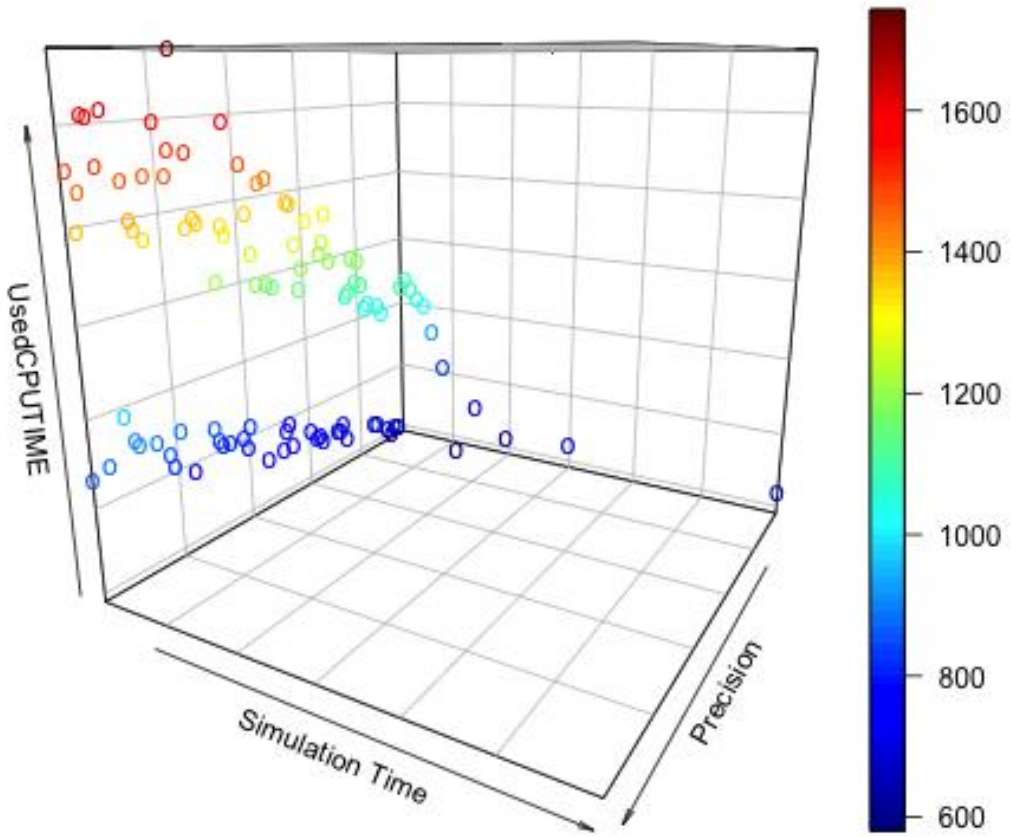


Abbildung 7.6: Rechenaufwand (CPU-Zeit) im Verhältnis zum internen Präzisionsparameter

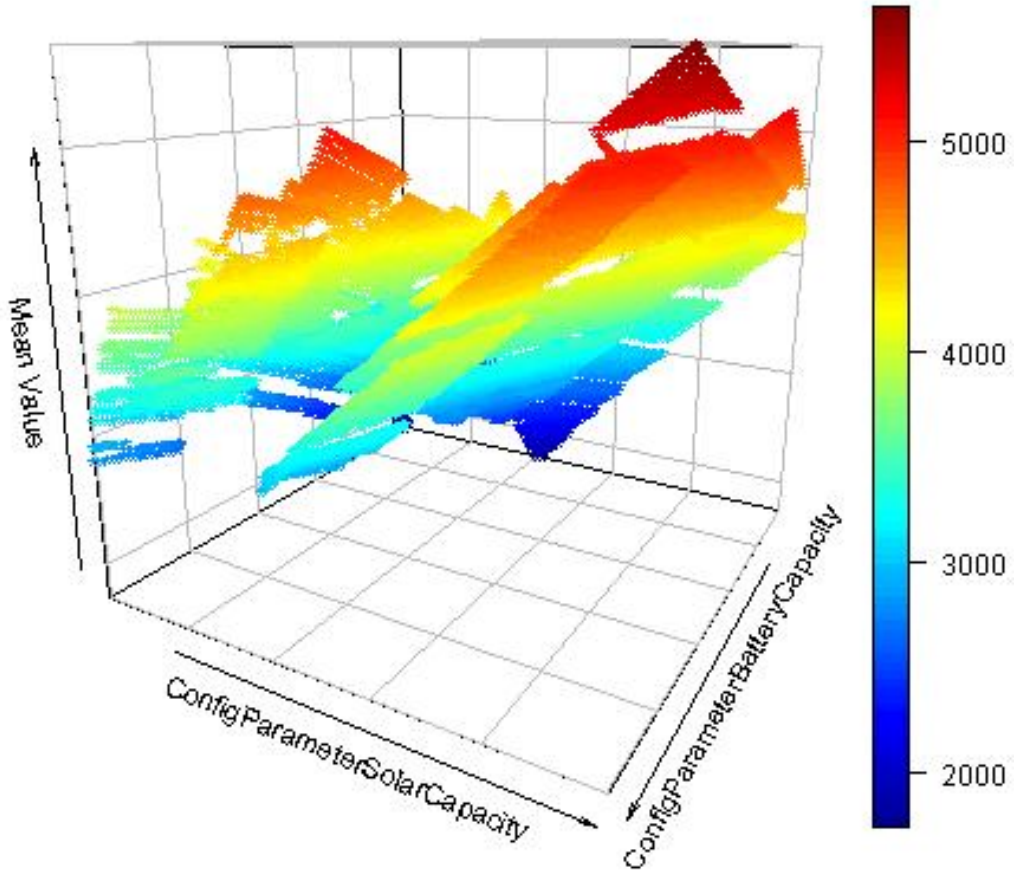


Abbildung 7.7: Wertelandschaft bei Hausvariante 1

7.5 Verwendete Modellkonfiguration

Unabhängig von den möglichen Optimierungsparametern lässt das Modell diverse Variationen und Präzisierungen zu. Um vergleichbare Messungen/Simulationen zu ermöglichen, werden hier alle tatsächlich verwendeten Parameterwerte und Definitionen aufgelistet. Parameter werden global definiert und können über ein separates Eingabefenster in TimeNET verändert werden. Sie können vom Datentyp real, int, string oder boolean sein.
Definitionen sind ähnlich Parametern, können aber direkt in der Zeichenfläche abgelegt werden. Damit sind sie jedem Nutzer des Modells sofort ersichtlich. Sie haben keinen festgelegten Datentyp. Der Nutzer muss jedoch beachten, dass der Wert und Typ einer Definition ihrer späteren Verwendung (in Berechnungen) entspricht.

7.5.1 Globale Modellparameter

Prinzipiell kann jeder Parameter durch das Optimierungssystem verändert werden. Da in den meisten Fällen jedoch viele Eigenschaften eines Hauses nicht verändert werden können während die optimale Investitionsentscheidung gesucht wird, sind die Namen der Parameter entsprechend gewählt. **AssumptionParameter** sind solche, die während der Optimierung nicht geändert werden, da sie in der Realität auch kaum beeinflusst werden können. **ConfigParameter** sind jene, für die ein optimaler Wert gefunden werden soll.
Die Leistung des Netzanschlusses wurde bewusst beschränkt, um auch Situationen mit begrenzter elektrischer Leistung und ggf. Stromausfälle im Modell zu provozieren.

HouseholdImplementation = 1 Es wird also das präzisere Modell des Haushaltes simuliert.

Precision = 1 Eine Marke entspricht damit einem KW, eine Simulationszeiteinheit entspricht einer Stunde Realzeit. Für exaktere Simulationen kann dieser Wert beliebig erhöht werden. Ein Wert von 50 würde bedeuten, dass eine Marke $\frac{1}{50}KW$ und eine Simulationszeiteinheit $\frac{1}{50}h$ entspricht.

ConfigParameterSolarCapacity = 0..5 KWP (KiloWattPeak), ein typisches Maß für die theoretische Leistung bzw. Größe privater Photovoltaikanlagen.

ConfigParameterBatteryCapacity = 0..5 KWh Kapazität der verbauten Speicherbatterie.

AssumptionParameterBatteryCostPerKWh = 230 € Preis der Speicherbatterie pro KWh.

AssumptionParameterSolarEfficiencyFactor = 0,8 Typischer Wirkungsgrad einer PV-Anlage.

AssumptionParameterSolarPricePKWP = 1800 € Preis pro KilowattPeak, der gebräuchlichen Größeneinheit für PV-Anlagen.

AssumptionParameterHouseHoldYearlyUsage = 3500 KWh Durchschnittlicher Jahresverbrauch eines 2-3 Personenhaushalts.

AssumptionParameterGridVoltage = 230V Netzspannung

AssumptionParameterGridCurrent = 16A Maximale Stromstärke bei Versorgung aus dem öffentlichen Netz

7.5.2 Definitionen im Modell

Für die Simulation werden verschiedene feststehende Annahmen über Einflussfaktoren benötigt. Diese lassen sich im Modell durch sogenannte „Definitionen“ festlegen. Nachfolgend sind die verwendeten Werte erläutert.

DayStart & DayEnd Tagesbeginn und -ende. Im Modell beginnt der Tag um 6:00 Uhr und endet um 18:00 Uhr.

daytimeconsumption Anteil am Stromverbrauch pro Tag (vom Gesamtverbrauch/24h), Es wird im Modell 20 % des gesamten Stromes tagsüber verbraucht.

GridPriceTresholdX Durchsatzgrenze (1 & 2) Wenn der simulierte Jahresverbrauch an Netzstrom die jeweilige Grenze überschreitet, wird für die Preisberechnung die nächsthöhere Preisstufe herangezogen.
Verwendete Werte: 2500 KWh, 5500 KWh

GridPricePerKWX Preisstufe X, der jeweilige Preis pro KWh. Im Modell werden 3 Preisstufen verwendet.
Verwendete Werte: 1,1 €; 0,3 €; 0,1 €

SolarPriceTresholdX Grenzwerte für Preisstufen der Solaranlage in KWP. Anhand dieser Grenzen wird entschieden, welcher Basispreis pro KWP für die Preisberechnung der Anlage herangezogen wird.
Verwendete Werte: 1 KWP, 4 KWP

SolarPricePerKWPX Jeweiliger Preis pro KWP für die 3 Preisstufen.
Verwendete Werte: 1400 €, 700 €, 330 €

BatteryPriceTresholdX Grenzwerte für Preisstufen der Batteriekosten. Die Berechnung erfolgt analog zur Preisberechnung der Solaranlage.
Verwendete Werte: 1 KWh, 4 KWh

BatteryPricePerKWhX Der jeweilige Basispreis pro KWh Speicherkapazität der Batterie.
Verwendete Werte: 800 €, 320 €, 200 €

7.6 Weiterentwicklung des Modells

Das Modell verdeutlicht, dass es möglich ist, den Energiefluss und -verbrauch grundsätzlich mit farbigen stochastischen Petri-Netzen zu modellieren und zu simulieren. Es werden dabei jedoch viele Annahmen und Vereinfachungen vorgenommen, die in zukünftigen Modellversionen präzisiert bzw. überhaupt erst modelliert werden sollten.
Einige Anregungen für die weitere Entwicklung und den Einsatz des Modells sollen hier gegeben werden.

Fixkosten Stromtarife bestehen oft aus einem Preis pro KWh und einem monatlichen Grundpreis. Ein solcher Grundpreis ist im aktuellen Modell nicht berücksichtigt.

Schwankungen der Solarstromproduktion Derzeit wird für die Produktion von Solarstrom ein sehr stark vereinfachtes Modell (eine Transition) verwendet.

Analog zum komplexeren Modell des Haushaltes sollte jedoch die, abhängig von der Tageszeit, schwankende Stromproduktion (ggf. auch jahreszeitabhängig) berücksichtigt werden.

Stromeinspeisung Derzeit wird nicht benötigter Solarstrom schlicht nicht produziert. In der Realität wird dieser jedoch ins Stromnetz eingespeist und generiert, abhängig von Stromtarif, Tageszeit und anderen Faktoren, Einnahmen.

Zuverlässigkeit In der Realität besteht stets das Risiko, Komponenten des Systems könnten ausfallen und müssen ersetzt bzw. repariert werden. Betrachtungen der Zuverlässigkeit an Modellen, die auf realen Beobachtungsdaten basieren, können wichtige Hinweise für die Wartung und Entwicklung dieser Systeme liefern.

Verschleiß Unabhängig vom Ausfallrisiko steigen bei jedem System mit dem Alter auch die Kosten für Wartung und i.d.R. sinkt der Wirkungsgrad. Diese Entwicklung muss bei der Investitionsentscheidung mit berücksichtigt werden.

Unabhängig von den genannten Punkten lassen sich noch diverse Details hinzufügen oder verbessern. Unterschiedliches Nutzerverhalten ist einer dieser Punkte. Aber auch weitere Teilsysteme wie elektrische Kraftfahrzeuge oder die Einbindung in ein globales Energienetz sind denk- und realisierbar.

7.7 Anmerkungen

Einige Formeln, Definitionen und Leistungsmaße aus dem Modell sind hier nicht genannt. Andere sind leicht verändert, um die Lesbarkeit zu erhöhen. Im Modell sind alle Elemente englisch und im **CamelCase** benannt. Daraus ergeben sich teilweise sehr lange Formeln, die zwar selbsterklärend aber im Druck unübersichtlich sind.
Die Formulierung „Energie wird erzeugt“ bzw. „Energie wird verbraucht“ taucht in diesem Text wie auch in großen Teilen der Literatur zum Thema Erneuerbare Energien auf. Dem Autor ist bewusst, dass diese Formulierung wissenschaftlich nicht korrekt ist, denn Energie kann nicht erzeugt oder verbraucht, sondern nur umgewandelt werden (Energieerhaltungssatz). Um die Arbeit jedoch allgemeinverständlich zu halten, wurde diese Formulierung dennoch gewählt.

8 Bewertung der Optimierungsalgorithmen

Im vorliegenden Kapitel werden die umgesetzten Optimierungsheuristiken anhand der genannten Beispielnetze und Benchmark-Funktionen bewertet. Jede Heuristik besitzt verschiedene Parameter zur Anpassung an spezifische Problemstellungen bzw. Wertelandschaften, deren Einfluss hier untersucht wird.
Die beschriebene Architektur des Optimierungswerkzeuges erlaubt es, jede Heuristik mit diversen Simulatoren zu verwenden. Die jeweilige Heuristik hat dabei keine Information über die Art des Simulators.
Aus Gründen der Übersichtlichkeit werden in dieser Arbeit stets 2-dimensionale Probleme betrachtet. Die Algorithmen wurden so implementiert, dass Probleme beliebig viele Dimensionen, die betrachteten SCPNs demnach beliebig viele variable Parameter haben können.

8.1 Bewertungskriterien

Um die Qualität der Optimierungsheuristiken vergleichen zu können, müssen zunächst Kriterien hierfür aufgestellt werden, welche für alle Heuristiken überprüft werden können. Folgende Eigenschaften wurden für die Bewertung herangezogen.

Entfernung zum globalen Optimum Dieses Kriterium kann besonders einfach bei Verwendung von Benchmark-Funktionen untersucht werden, da der Funktionswert des Optimums und der entsprechende Parametersatz (Vektor) bekannt ist. Bei real durchgeführten Simulationen wird die Cache-unterstützte Simulation verwendet, wobei im Cache das absolute Optimum vorhanden ist (soweit bekannt). In der Auswertephase bestimmt das Programm TOE dann den Abstand zu diesem Wert bzw. Parametersatz.

Der Abstand zum absoluten Optimum wird auf zwei Arten berechnet und bewertet.

- Der relative Abstand des gefundenen Optimums im Wertebereich. Abhängig von Minimum und Maximum der berechneten Funktionswerte bzw. Simulationsergebnisse wird der Abstand zum Funktionswert des Optimums in Prozent angegeben.
- Der relative euklidische Abstand in Bezug zum maximal möglichen Abstand im Definitionsraum. Dies ist ebenfalls in Prozent angegeben.

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Euklidischer Abstand von Punkten im mehrdimensionalen Raum

Beide Kenndaten werden in den Ergebnistabellen mit **Abstand** und **Abstand (EUKLID)** abgekürzt.

Anzahl benötigter Simulationsläufe (Sim#) Dies ist ein Hauptkriterium für die Qualität neuer Heuristiken der simulationsbasierten Optimierung. Erste Ergebnisse bezüglich der vorgestellten Multiphasenoptimierung wurden bereits in [7] vorgestellt. Da der Großteil der CPU-Zeit bei simulationsbasierten Optimierungsalgorithmen auf die Simulation selbst entfällt, ist es ein primäres Ziel, die Anzahl dieser Simulationsläufe zu reduzieren.

Benötigte CPU-Zeit aller Simulationsläufe (CPU) Im Falle eines konstanten CPU-Zeitbedarfs über alle Simulationsläufe ist dieser Wert trivial berechenbar. Die Erweiterung der Optimierungsheuristiken um Variation der Simulationsgenauigkeit liefert hier jedoch wegweisende Ergebnisse.

Um auch bei Verwendung von Benchmark-Funktionen aussagekräftige Daten für dieses Kriterium zu bekommen, wird dabei die benötigte CPU-Zeit gemäß der empirisch bestimmten Funktion 4.2 berechnet.

Bei echter Simulation der SCPNs wird die Summe aller einzelnen Simulationsläufe gebildet. Dies gilt auch bei verteilter und damit möglicherweise paralleler Simulation mehrerer Parametersätze.

Cache-Trefferrate (CHR) Das Zwischenspeichern von Simulationsergebnissen ist prinzipiell nützlich und erspart die redundante Simulation von Parametersätzen, auch wenn dies in der untersuchten Heuristik selbst nicht vorgesehen ist.

Abhängig von verwendeter Auflösung und Heuristik wird ggf. öfters ein Simulationsergebnis angefordert, das bereits vorliegt. Daher ist das Verhältnis zwischen neu zu berechnenden Simulationsergebnissen und bereits durchgeführten Simulationen interessant; insbesondere im Zusammenhang mit der Gesamtsumme der notwendigen Simulationen und dem Abstand zum globalen Optimum.

Die Cache-Trefferrate spiegelt auch die klassische Exploration/Exploitation-Problematik wider. Ein besonders hoher Wert bedeutet, der Algorithmus hat die gleichen Bereiche mehrfach durchsucht. Ein niedriger Wert deutet auf eine umfassendere Untersuchung des Definitionsraumes hin.

Es ist zu erwarten, dass mit steigender Auflösung des Definitionsbereichs die Cache-Trefferrate sinkt. Abhängig vom verwendeten Basissystem für die Optimierungssteuerung (TOE) wird die Verwendung eines Ergebniscaches früher oder später aufgrund des erhöhten Speicherbedarfs unwirtschaftlich.

8.2 Ablauf der Bewertung

Zunächst werden alle Basisheuristiken anhand von Benchmark-Funktionen untersucht. Dies ist eine verbreitete Art der Bewertung von Optimierungsalgorithmen mit verschiedenen Vorteilen. Einerseits spart es gegenüber echten Simulationen viel Zeit, andererseits ist das absolute Optimum bekannt oder berechenbar. Nach Abschluss eines Optimierungslaufs kann demnach sofort beurteilt werden, ob nur ein lokales Optimum gefunden wurde bzw. wie weit der gefundene Wert vom globalen Optimum entfernt ist. Um verlässliche Informationen zu erhalten, wird jede Heuristik und jede ihrer Konfigurationen 100 mal durchgeführt und aus den Ergebnissen die durchschnittliche Qualität dieser Heuristik in Bezug auf das bearbeitete Problem bestimmt. Grundsätzlich beginnen alle Algorithmen jeweils mit einem zufällig gewählten Parametersatz.

Die verwendeten Benchmark-Funktionen, wie in 6.3 genannt, ermöglichen die Beurteilung der umgesetzten Heuristiken anhand typischer Wertelandschaften von Kostenfunktionen. Sehr langsam konvergierende Funktionen wie auch solche mit vielen lokalen Optima sind dabei vorhanden, wodurch die Ergebnisse der Untersuchung Rückschlüsse auf den Einsatz bei tatsächlichen SCPN-Simulationen bieten.
Begonnen wird die Bewertung mit dem verbreitetsten und und vergleichsweise einfach zu implementierenden Algorithmus, **Hill-Climbing**. Trotz seines einfachen Grundprinzips bietet es diverse Variationen hinsichtlich der Strategie zur Wahl des jeweils nächsten Parametersatzes. Anschließend folgt die Untersuchung von **Simulated Annealing** und dessen unterschiedlichen Konfigurationen. Im Vergleich dazu wird der umgesetzte genetische Algorithmus, **Simulated Binary Crossover** (SBX) in gleicher Weise untersucht.
Nachdem die bestmöglichen Konfigurationen dieser Basisheuristiken gefunden sind, werden diese in **zwei-** und **mehrstufigen** Verfahren unter Variation der Definitionsraumdiskretisierung, der Simulationsgenauigkeit und der Phasenanzahl zur Anwendung gebracht, um eine bestmögliche Kombination aus Basisheuristik und Metaheuristikkonfiguration zu finden. Dabei wird zunächst die zweiphasige Untersuchung mit Simulated Annealing separat betrachtet, deren Idee dem Buch „Stochastic Discrete Event Systems - Modeling, Evaluation, Applications“ [67] entlehnt ist.
Nicht verändert wird die Größe und Diskretisierung des Definitionsraumes zu Beginn aller Untersuchungen mit Ausnahme derer des Energiemodells. Diese sind stets auf 2 Parameter mit jeweils 10000 Werten beschränkt. Abgesehen von den Algorithmen, welche die Diskretisierung prinzipiell nicht auswerten, ergibt sich eine Gesamtgröße des Definitionsbereichs von 10^8 Werten.
Die Darstellung der Ergebnisse erfolgt in Gruppen jeweils für jede gewählte Heuristik für **Benchmark-Funktionen mit wenigen/keinen lokalen Optima** – Matya und Sphere– und **Benchmark-Funktionen mit vielen lokalen Optima** –Schwefel und Ackley. Anschließend erst für das vorgestellte **Basis-SCPN (keine lokalen Optima)** und schließlich für das **Energiemodell-SCPN (viele lokale Optima)**.

8.2.1 Wertelandschaften der verwendeten Benchmark-Funktionen

Zum besseren Verständnis werden nachfolgend die Wertelandschaften der Benchmark-Funktionen, als auch der realen SCPN-Leistungsmaße dargestellt. Sie wurden mit gröberer Diskretisierung erstellt, als sie die eigentliche Optimierung verwendet. Insbesondere bei realer Simulation ergibt sich dadurch eine große Zeitersparnis. Ihre Umsetzung ist in Kapitel 6.3 näher erläutert.

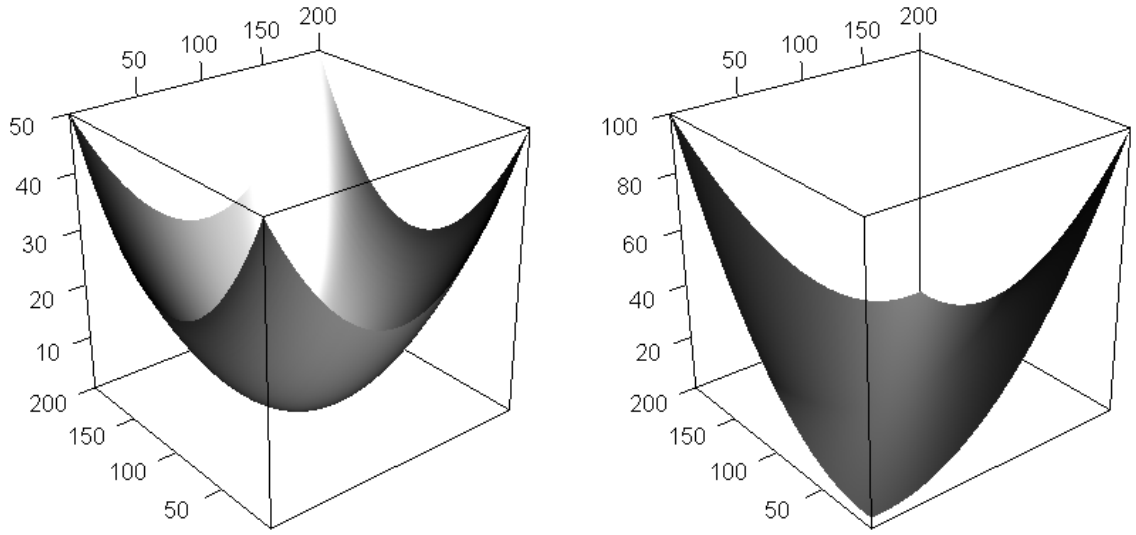


Abbildung 8.1: Sphere- und Matyafunktion (keine lokalen Optima)

Die Funktionen Sphere und Matya sind Vertreter der einfachsten Benchmark-Funktionen, denn ihre Wertelandschaften weisen keine lokalen Optima auf und sie konvergieren stetig hin zu ihrem Optimum (Bild 8.1). Erwartungsgemäß finden auch einfache Heuristiken das Optimum, wenn auch mit unterschiedlich vielen Simulationsversuchen. Beide Funktionen haben ihr Optimum im Koordinatenursprung, wobei Matya nur für 2 Dimensionen definiert ist. Jedoch bietet Matya andere interessante Eigenschaften. Entlang der Diagonale aus x- und y-Achse konvergiert diese Funktion sehr langsam in Richtung des globalen Optimums. Dies stellt für einige Heuristiken eine große Herausforderung dar.

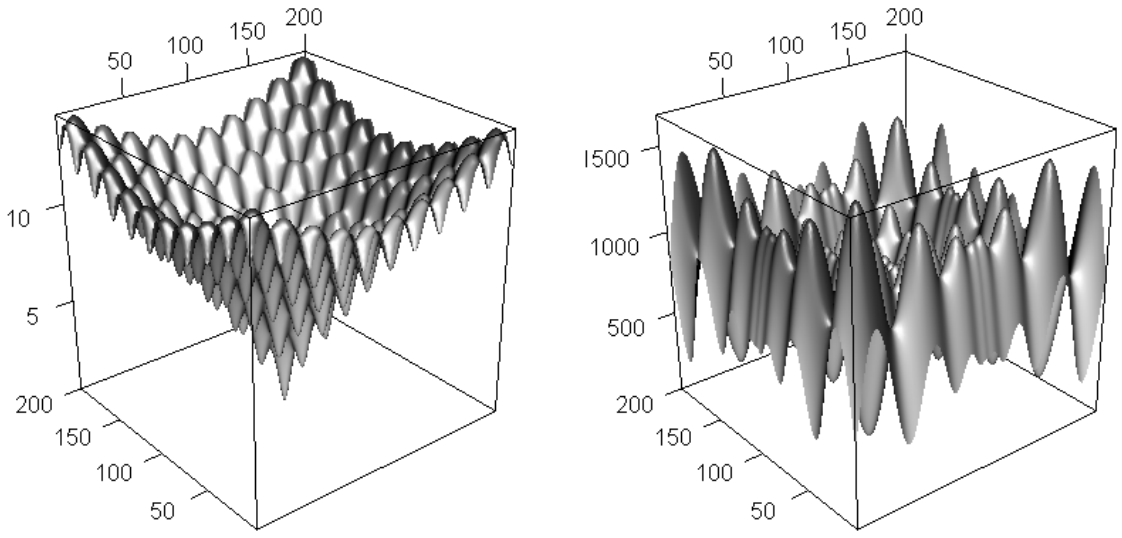


Abbildung 8.2: Ackley- und Schwefelfunktion (viele lokale Optima)

Im Gegensatz dazu haben die Funktionen Ackley und Schwefel viele lokale Optima (Bild 8.2). Ein typisches Szenario, um neue Optimierungsheuristiken zu testen, denn natürliche Prozesse haben oft viele „gute“ Konfigurationen. Die Frage, ob eine weitere Exploration des Definitionsraumes sinnvoll ist, taucht daher oft bei der Optimierung von Prozessen auf. Dass ähnliche Szenarios auch bei der Optimierung von SCPNs auftreten können, wird in Kapitel 7 erläutert.
Das globale Optimum einer jeden Benchmark-Funktion ist in Kapitel 6.3 beschrieben. Es handelt sich bei allen Benchmark-Funktionen um Standardfunktionen für die Analyse von Optimierungsalgorithmen, deren Details auch in der Fachliteratur zu finden sind [48].

8.2.2 Wertelandschaften, Simulationskonfiguration und Optima der verwendeten SCPNs

Die Analyse der Heuristiken anhand von realen SCPN-Simulationen unterscheidet sich in einigen Punkten von der Benchmark-basierten Untersuchung. Wichtigste Unterschiede sind dabei die Dauer jeder einzelnen Simulation, was die Analyse anhand realer Simulation sehr zeitaufwendig macht, sowie die Beurteilung des gefundenen Optimums.
Im Gegensatz zu den Benchmark-Funktionen ist das globale Optimum für die

verwendeten SCPNs nicht bekannt und muss vorher experimentell durch Batch-Simulation ermittelt werden. Dies wurde in Vorbereitung auf die hier erläuterten Versuche für beide verwendete Beispiel-SCPNs getan. Unabhängig von der dabei gewählten Diskretisierung des Definitionsraumes kann dennoch nicht ausgeschlossen werden, dass das tatsächliche globale Optimum nicht gefunden wurde. Insbesondere die Beurteilung von Heuristiken, welche grundsätzlich mit kontinuierlichem Definitionsraum arbeiten, wird dadurch erschwert bzw. in Frage gestellt.
Das konkrete Vorgehen für die Bewertung anhand realer SCPN-Simulation umfasst jeweils folgende Schritte:

1. Festlegen der externen Simulationsparameter (Simulationsgenauigkeit)
2. Batch-Simulation (nach Möglichkeit) des gesamten Definitionsraumes
3. Bestimmung des globalen Optimums aus den Resultaten der Batch-Simulation
4. Erstellen einer Cache-Datei, deren Inhalt ausschließlich das tatsächliche Optimum sowie Minimum und Maximum der Kostenfunktion ist
5. Laden der erstellten Cache-Datei und anpassen des Definitionsraumes
6. Festlegen des bekannten Wertes für das Optimum als Zielwert der Optimierung
7. Analyse der eingestellten Heuristik mittels Cache-gestützter Simulation

Das Tool TOE wertet die gefundenen Optima nach jeder vollendeten Optimierung automatisch aus. Im Fall der Benchmark-Simulation wird dabei der Abstand zum tatsächlich bekannten bzw. berechenbaren globalen Optimum beurteilt. Bei realer SCPN-Simulation wird das globale Optimum aus den Werten der Cache-Datei bestimmt. Ebenso werden die Maximal- und Minimalwerte der Kostenfunktion bzw. der Simulationsergebnisse aus dieser Datei gelesen. Unter der Annahme, dass diese Werte korrekt sind, ist damit eine ähnlich präzise Beurteilung der gefundenen Optima hinsichtlich Abstand im Definitions- und Wertebereich möglich, wie bei Benchmark-Funktionen. Im Unterschied zu Benchmark-Funktionen muss jedoch der Wert des bekannten globalen Optimums als Zielwert der Optimierung in der Software angegeben werden. Die Auswahl der Cache-gestützten Simulation ist - wie bei Benchmark-Funktionen - notwendig, um die Trefferrate bzw. Cache-Hit-Ratio

bestimmen zu können. Bei Standard-Simulationen wird kein Cache verwendet. Sie eignen sich hauptsächlich für Optimierungsalgorithmen, welche selbstständig Mehrfachsimulationen vermeiden.

Basis-SCPN

Das experimentell ermittelte Optimum des Beispiel-SCPN liegt bei *TDrain*=9744, *TSource*=9524 und hat den Wert 0,709401. Weitere mögliche Kandidaten sind in Tabelle 8.1 aufgelistet. Die Tabelle erhebt dabei keinen Anspruch auf Vollständigkeit, da aus Mangel an Ressourcen nur Teile des Definitionsbereichs simuliert wurden. Die Auswahl der zu simulierenden Teile erfolgte auf Basis erster Voruntersuchungen, deren Ergebnisse z.B. in Abbildung 4.2 zu sehen sind. Die genaue Konfiguration des Modells bzw. der jeweiligen Simulationsläufe findet sich in Tabelle 8.2.
Wie im Plot zu sehen, besitzt das Leistungsmaß nahezu optimale Werte am entgegengesetzten Ende des Wertebereichs; eine große Herausforderung für Optimierungsheuristiken.

TDrain	TSource	MeasureP
9744	9524	0,7094010
9921	9697	0,7094011
9788	9567	0,7094012
9832	9610	0,7094013
9965	9740	0,7094013
9876	9653	0,7094014
9743	9523	0,7094015
310	303	0,7094016
9787	9566	0,7094016

Tabelle 8.1: Lokale Optima des Beispiel-SCPN

Energiemodell

Das experimentell ermittelte Optimum des Energiemodells liegt bei 0,1 und 0,11 für die Parameter **ConfigParameterBatteryCapacity** und **ConfigParameterSolarCapacity**, sofern die restlichen Parameter die in 7.5 genannten Werte haben und die Parameter zur Steuerung der Simulationsgenauigkeit sowie die entsprechenden Abbruchkriterien gemäß Tabelle 8.2 konfiguriert sind. Einschränkend wird hier ein

Definitionsbereich von 241081 Parametersätzen verwendet, um die Analyse auf typischer PC Hardware in realistischer Zeit durchführen zu können. Die beiden Parameter **ConfigParameterBatteryCapacity** und **ConfigParameterSolarCapacity** können dabei jeweils Werte von 0,1 bis 5 annehmen mit einer Auflösung von 0,01. Dieser Schritt erwies sich aufgrund der größeren Simulationszeit pro Parametersatz als notwendig. Davon nicht betroffen sind Algorithmen, die mit kontinuierlichen Definitionsbereichen arbeiten.
Als Optimum gilt der niedrigste Wert für CostSum, also die Summe aller Kosten (Netzstrom + Anschaffungskosten für Batterie und Solaranlage). Dieser ist hier 1983,671 €. Jedoch hat auch dieses Modell eine nahezu optimale Parameterkonfiguration, die im Definitionsbereich sehr weit entfernt vom Optimum zu finden ist, wie in Tabelle 8.3 zu sehen.

Parameter	Wert
Konfidenzintervall	90 %
Maximaler relativer Fehler	5 %
Seed	3
Max. Simulationsschritte	0 (unendlich)
Maximale CPU-Zeit	500 Sekunden

Tabelle 8.2: Simulationskonfiguration für beide SCPNs

Es ist zu erwarten, dass die implementierten Heuristiken ähnliche Ergebnisse liefern, wie bei der Verwendung von Benchmark-Funktionen mit vielen lokalen Optima. Eine Darstellung der Wertelandschaft findet sich auf Seite 117.

ConfigParameter-BatteryCapacity	ConfigParameter-SolarCapacity	CostSum
0,10	0,11	1983,671
0,10	0,12	1984,825
	...	
4,20	4,60	1994,330
	...	
0,15	0,12	2007,832
0,14	0,10	2008,027
	...	

Tabelle 8.3: Lokale Optima des Energieverbrauchsmodells

Die gezeigten Simulationsergebnisse der verwendeten SCPNs, insbesondere die Werte und Positionen der Optima, gelten nur für die angegebenen Modellkonfigurationen. Zur Bestätigung der gefundenen optimalen Parametersätze empfehlen sich weitere Simulationswiederholungen mit erhöhter Simulationsgenauigkeit und variierendem Seed.

8.3 Ergebnisse

8.3.1 Hill-Climbing

Der grundsätzliche Ablauf von Hill-Climbing ist in Kapitel 2.4.1 beschrieben. Die konkrete Umsetzung im Optimierungstool arbeitet nach dem in Kapitel 6.6.1 dargelegten Algorithmus.
Daraus sind bereits die wichtigsten Variablen für den Einsatz von Hill-Climbing ersichtlich: die Bestimmung des nächsten Parametersatzes und die Abbruchbedingungen für eine Optimierung.
Als Abbruchbedingung werden hier die Parameter WRONG_SOLUTIONS_IN_A_ROW und WRONG_SOLUTION_PER_DIRECTION variiert. Ersterer gibt die maximale Anzahl der Simulationsläufe ohne Verbesserung des Ergebnisses an und wird bei allen Strategien zur Parameterbestimmung ausgewertet. Letzterer kommt nur bei der Strategie **Diskret, auf/ab** zur Anwendung und beschränkt die Anzahl von Simulationsläufen ohne Verbesserung, während ein Parameter in eine Richtung geändert wird. Bei Richtungswechsel oder Änderung eines anderen Parameters beginnt die Zählung von vorn.
Von den in 2.4.1 beschriebenen Strategien zur Bestimmung des jeweils nächsten Parametersatzes werden an dieser Stelle nur 3 Varianten näher untersucht. Die erste und wichtigste ist **Diskret, auf/ab**. Sie verspricht die größte Trefferwahrscheinlichkeit für das globale Optimum, sofern keine lokalen Optima vorhanden sind. Interessant in diesem Zusammenhang ist, inwiefern die Werte für WRONG_SOLUTIONS_IN_A_ROW und WRONG_SOLUTION_PER_DIRECTION die Trefferwahrscheinlichkeit bei Existenz lokaler Optima verändern.
WRONG_SOLUTIONS_IN_A_ROW wird mit Werten von 2..21 in Schrittweite von 5 geprüft. Werte kleiner 2 führen zu keinem sinnvollen Ergebnis, da hier praktisch sofort abgebrochen wird. WRONG_SOLUTION_PER_DIRECTION sollte

stets kleiner als WRONG_SOLUTIONS_IN_A_ROW gewählt werden, da ansonsten niemals die Richtung oder der zu ändernde Parameter gewechselt wird.
Im Vergleich dazu wird die Variante **Zufällig, diskret in Nachbarschaft** betrachtet. Erwartet wird eine Verbesserung bei Problemen bzw. Kostenfunktionen mit vielen lokalen Optima. **Zufällig, kontinuierlich in Nachbarschaft** implementiert im Prinzip das gleiche Vorgehen, wobei hier die Diskretisierung des Definitionsraumes ignoriert wird[1]. Daraus können keine zeitlichen Verbesserungen durch Verwendung der Ergebnisdatenbank erzielt werden, jedoch wird eine präzisere Bestimmung des Optimums erwartet. Variiert wird während der Untersuchung hauptsächlich die Größe der Nachbarschaft in %, also des Radius, innerhalb dessen der jeweils nächste Parameterwert zufällig ausgewählt wird.
Bei den Benchmark-Funktionen Matya und Schwefel ist zu erwarten, dass klassische Hill-Climbing Versionen sehr gute Ergebnisse erzielen. Ohne lokale Optima sollten diese Varianten die optimale Konfiguration finden, wenngleich hierfür viele Simulationsschritte notwendig sein könnten, denn die Schrittweite der Parameteränderungen bleibt stets konstant.

Benchmark-Funktionen mit wenigen/keinen lokalen Optima

Erwartungsgemäß werden bei stetigen Kostenfunktionen ohne lokale Optima bereits mit wenigen erlaubten Fehlversuchen (WRONG_SOLUTIONS_IN_A_ROW) sehr gute Ergebnisse erzielt. Insbesondere beim Standardverfahren Diskret Auf/Ab liefert eine Erhöhung der Anzahl erlaubter Fehlversuche nur marginale Verbesserungen bei teilweise drastisch erhöhter Simulationszahl. Gleiches gilt für die Anzahl der Fehlversuche pro Richtung (WRONG_SOLUTION_PER_DIRECTION).
Die Verfahren, welche neue Parametersätze zufällig innerhalb eines Gebietes um den aktuellen Parametersatz bestimmen, zeigen ähnliche Ergebnisse bei diesen Kostenfunktionen. Die Größe des Suchgebietes sowie die Anzahl möglicher Fehlversuche bewirkt lediglich eine erhöhte Simulationszahl, bei sehr geringer Verbesserung des Ergebnisses.
Die dargestellte CPU-Zeit ist im Zusammenhang mit Benchmark-Funktionen von untergeordnetem Interesse, da diese nur berechnet wird, aber tatsächlich kaum Aus-

[1] Tatsächlich werden die Parameter mit einer Genauigkeit von 3 Dezimalstellen bestimmt. Dies kann für spätere Analysen im Quelltext des Programms angepasst werden.

wirkung auf die Gesamtzeit der Optimierung hat. Sie gibt dennoch Hinweise auf die Wirtschaftlichkeit bestimmter Konfigurationen der Optimierungsalgorithmen beim späteren Einsatz für die Optimierung von SCPNs mit ähnlichen Wertelandschaften der Simulationsergebnisse.

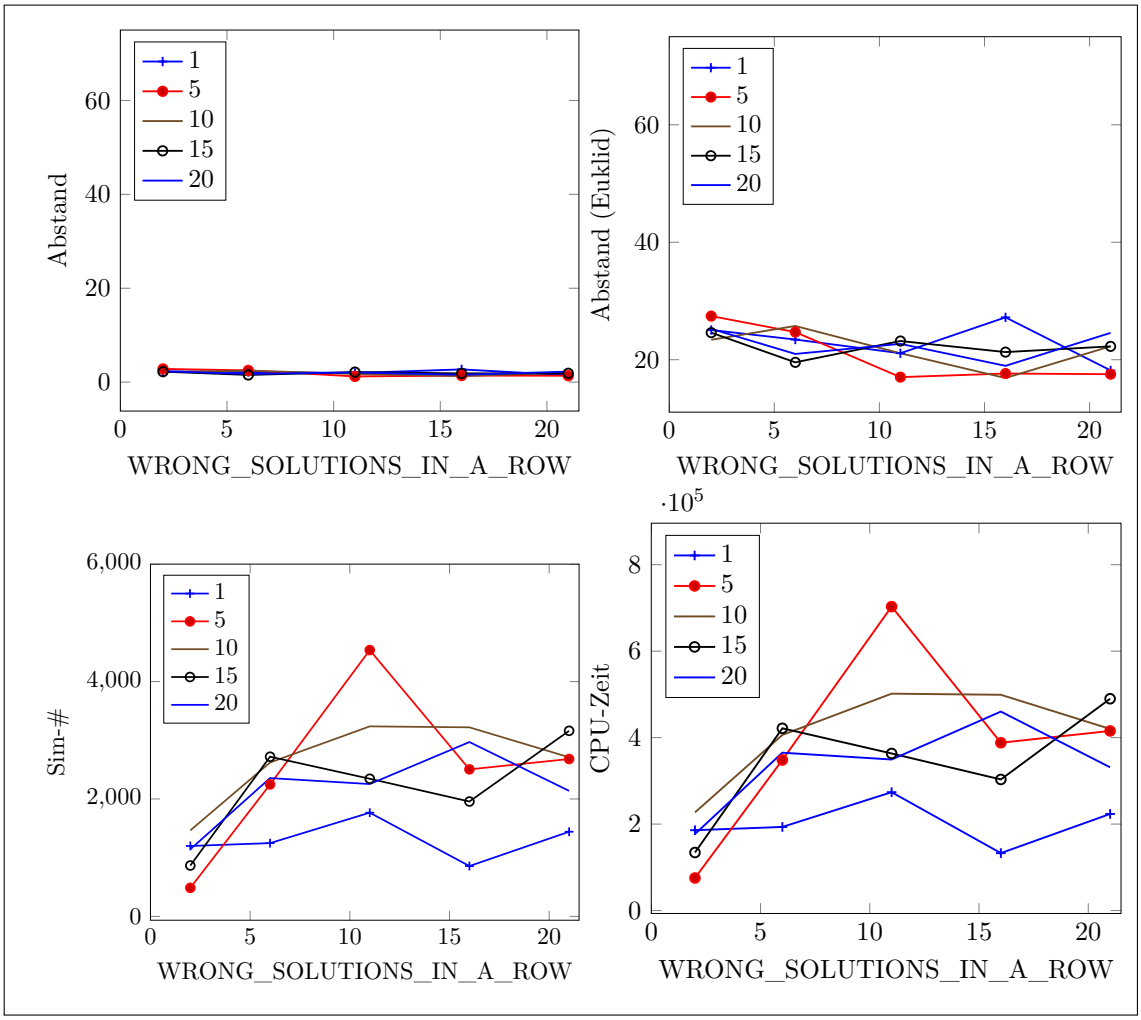


Abbildung 8.3: Hill-Climbing & diskret Auf/Ab auf Benchmark-Funktion Sphere
Alle Messergebnisse finden sich in Tabelle B.1 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweils 1, 5, 10, 15, 20 Fehlversuchen pro Richtung (WRONG_SOLUTIONS_PER_DIR).
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

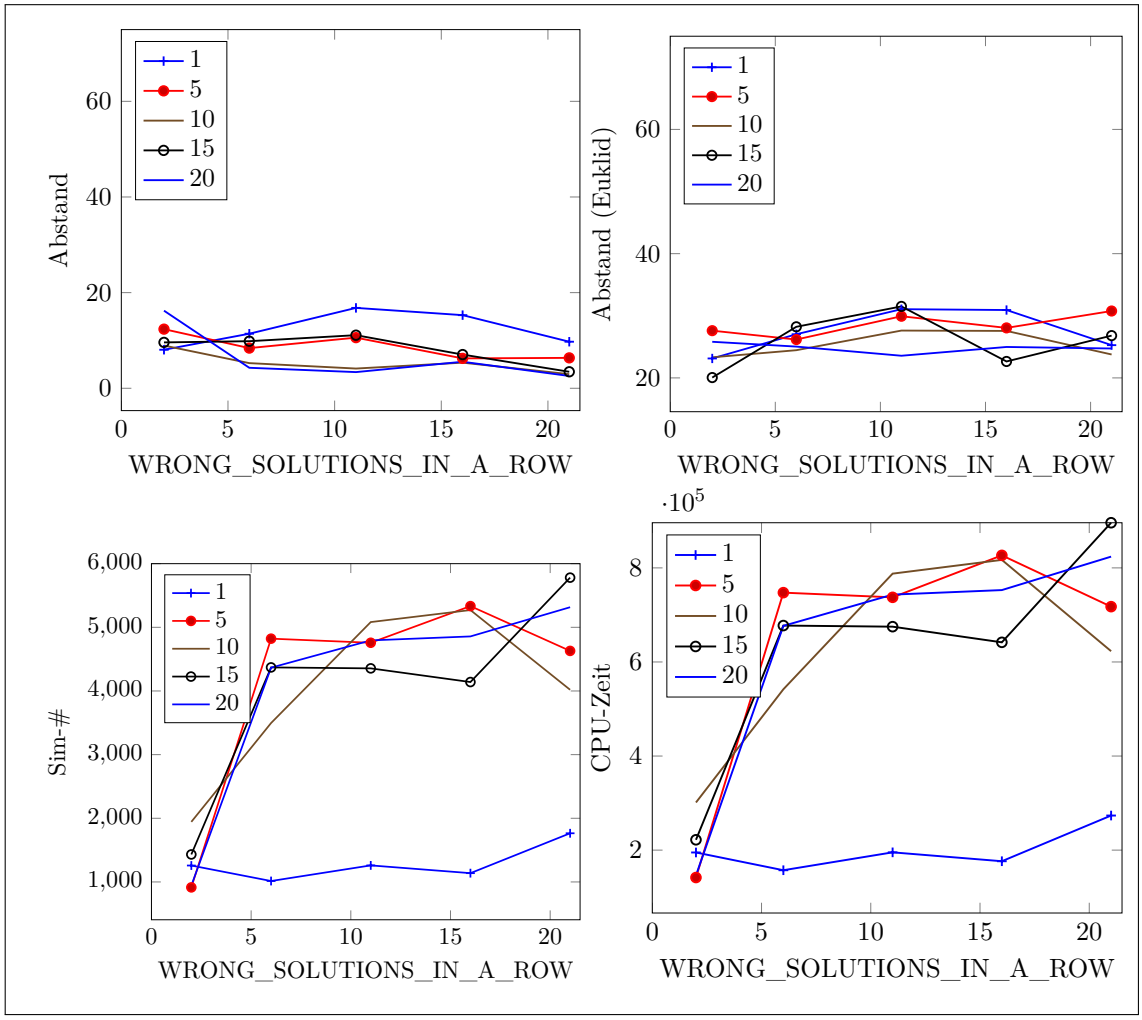


Abbildung 8.4: Hill-Climbing & diskret Auf/Ab auf Benchmark-Funktion Matya.
Alle Messergebnisse finden sich in Tabelle B.2 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweils 1, 5, 10, 15, 20 Fehlversuchen pro Richtung (WRONG_SOLUTIONS_-PER_DIR).
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

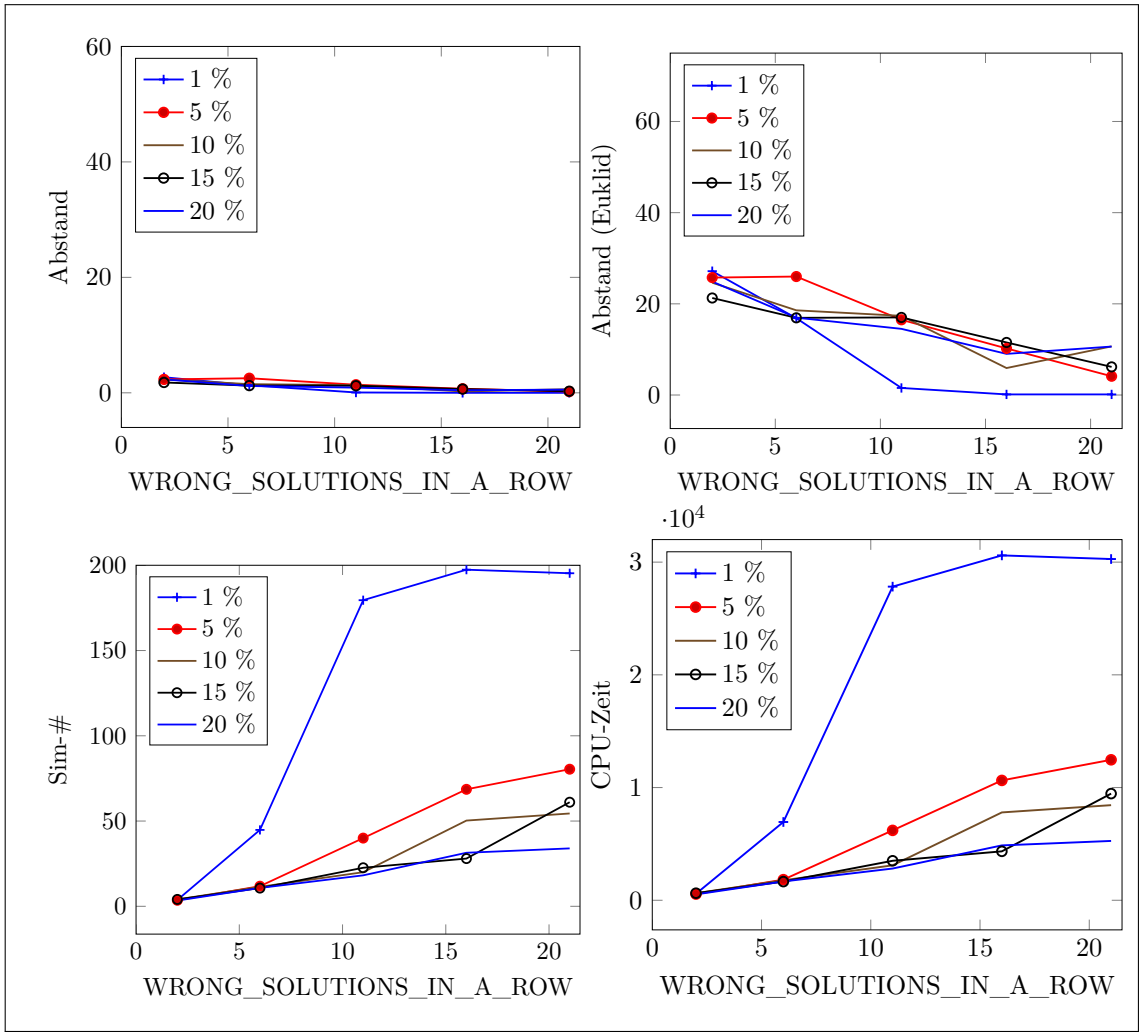


Abbildung 8.5: Hill-Climbing & zufällig, diskret in Nachbarschaft auf Benchmark-Funktion Sphere.
Alle Messergebnisse finden sich in Tabelle B.3 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweiliger Größe (in %) der Nachbarschaft für die Bestimmung des nächsten Parametersatzes.
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

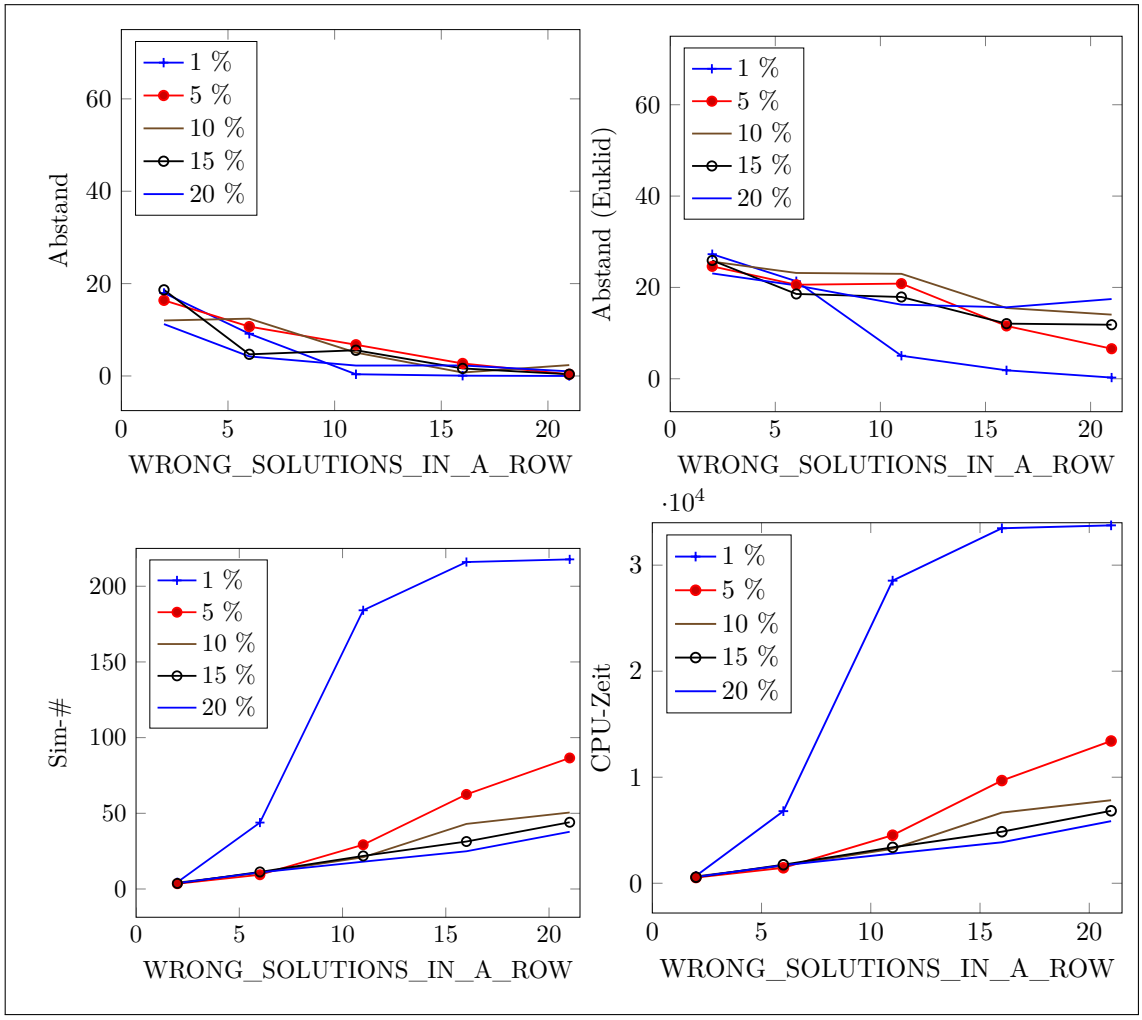


Abbildung 8.6: Hill-Climbing & zufällig, diskret in Nachbarschaft auf Benchmark-Funktion Matya.
Alle Messergebnisse finden sich in Tabelle B.4 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweiliger Größe (in %) der Nachbarschaft für die Bestimmung des nächsten Parametersatzes.
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

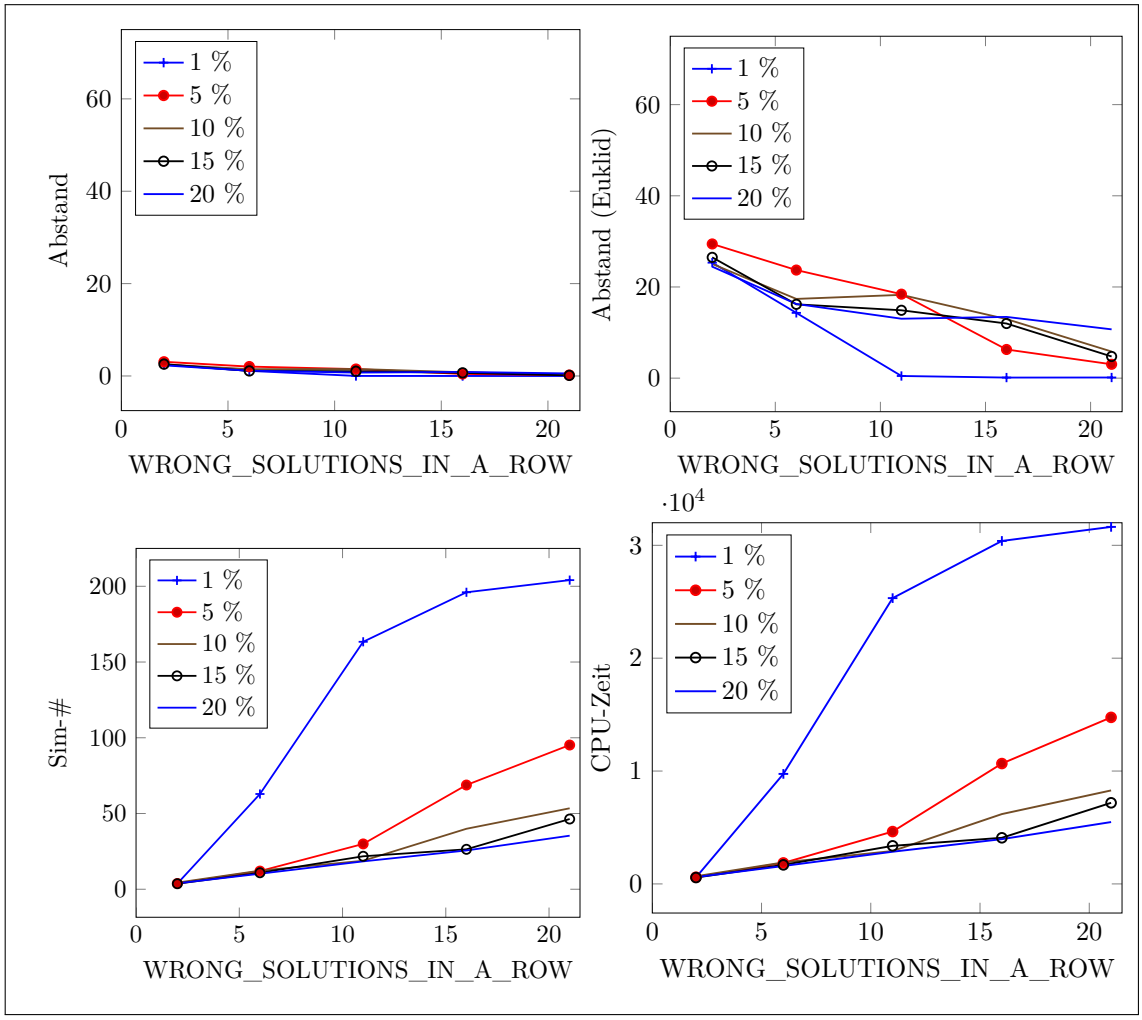


Abbildung 8.7: Hill-Climbing & zufällig, kontinuierlich in Nachbarschaft auf Benchmark-Funktion Sphere.
Alle Messergebnisse finden sich in Tabelle B.5 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweiliger Größe (in %) der Nachbarschaft für die Bestimmung des nächsten Parametersatzes.
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

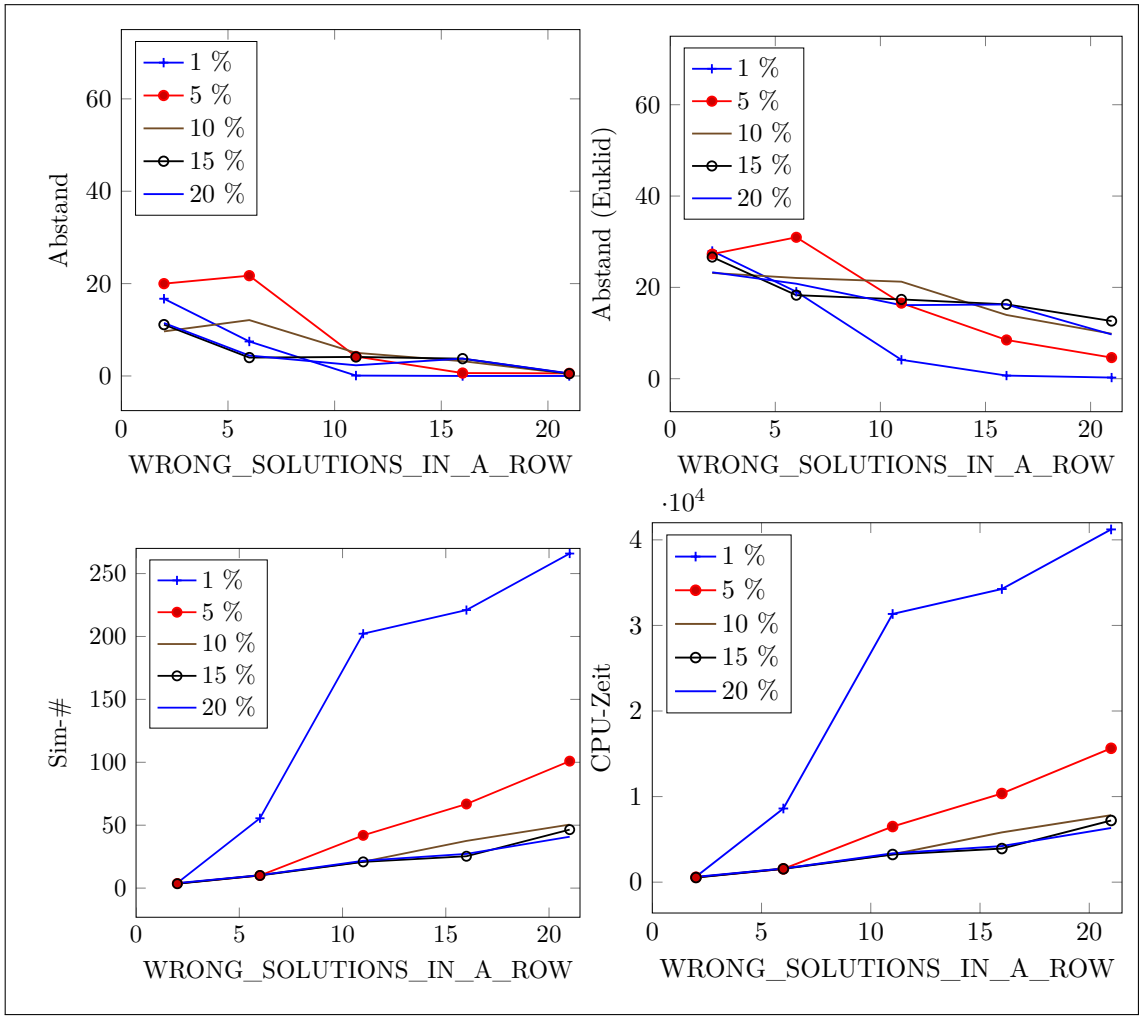


Abbildung 8.8: Hill-Climbing & zufällig, kontinuierlich in Nachbarschaft auf Benchmark-Funktion Matya.
Alle Messergebnisse finden sich in Tabelle B.6 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweiliger Größe (in %) der Nachbarschaft für die Bestimmung des nächsten Parametersatzes.
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

Benchmark-Funktionen mit vielen lokalen Optima

Bei Vorhandensein vieler lokaler Optima führt erwartungsgemäß eine geringe Anzahl erlaubter Fehlversuche, sowohl insgesamt als auch pro Parameter und Richtung, zu

schlechten Ergebnissen, da der Algorithmus beim Durchschreiten des Definitionsbereichs kaum das Gebiet eines gefundenen lokalen Optimums verlassen kann. Dies gilt für alle Varianten von Hill-Climbing. Es bleibt dem Nutzer überlassen, anhand der jeweiligen Wertelandschaft eine Abwägung zwischen voraussichtlicher Simulationszahl (und damit Zeit der gesamten Optimierung) und erwarteter Nähe zum globalen Optimum zu treffen.

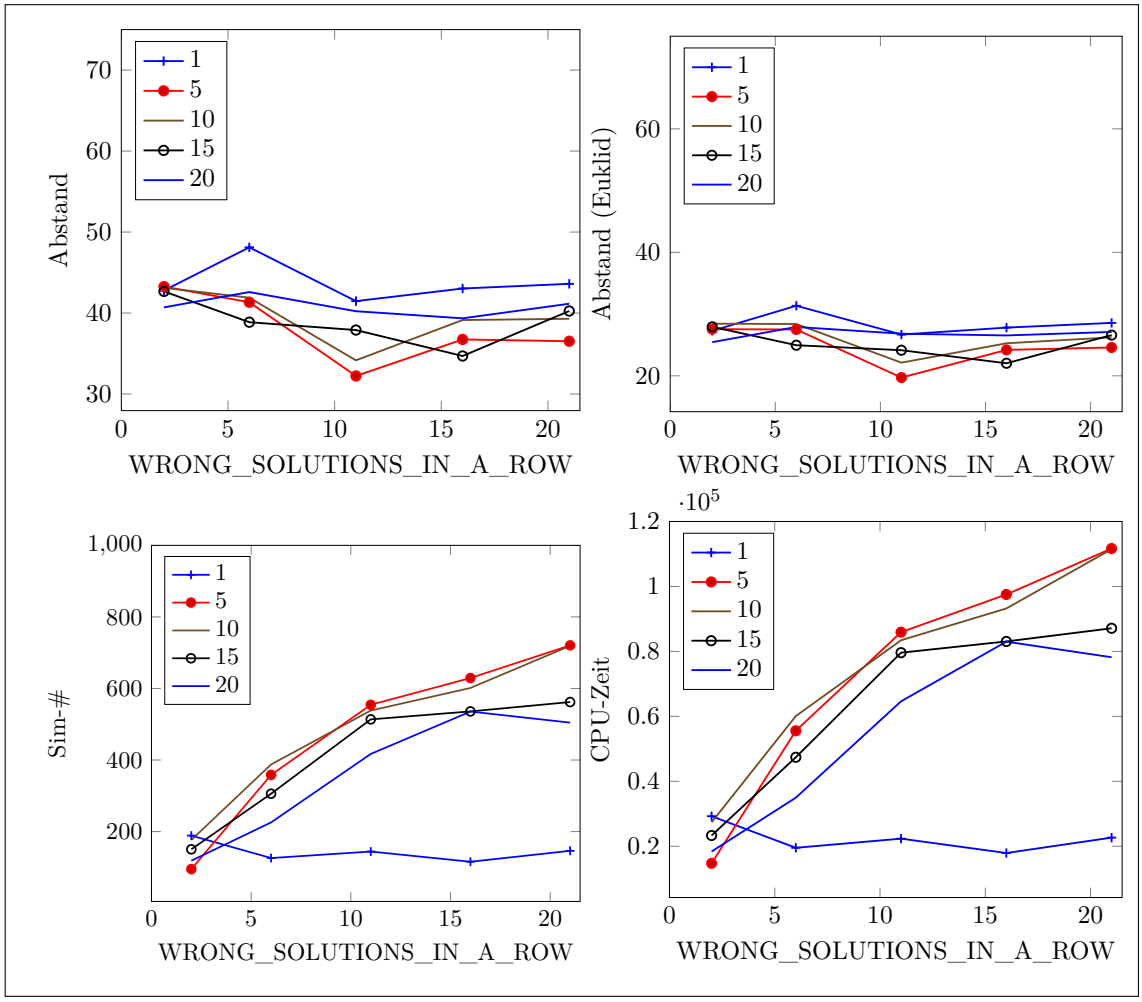


Abbildung 8.9: Hill-Climbing & diskret Auf/Ab auf Benchmark-Funktion Ackley.
Alle Messergebnisse finden sich in Tabelle B.7 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweils 1, 5, 10, 15, 20 Fehlversuchen pro Richtung (WRONG_SOLUTIONS_-PER_DIR).
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

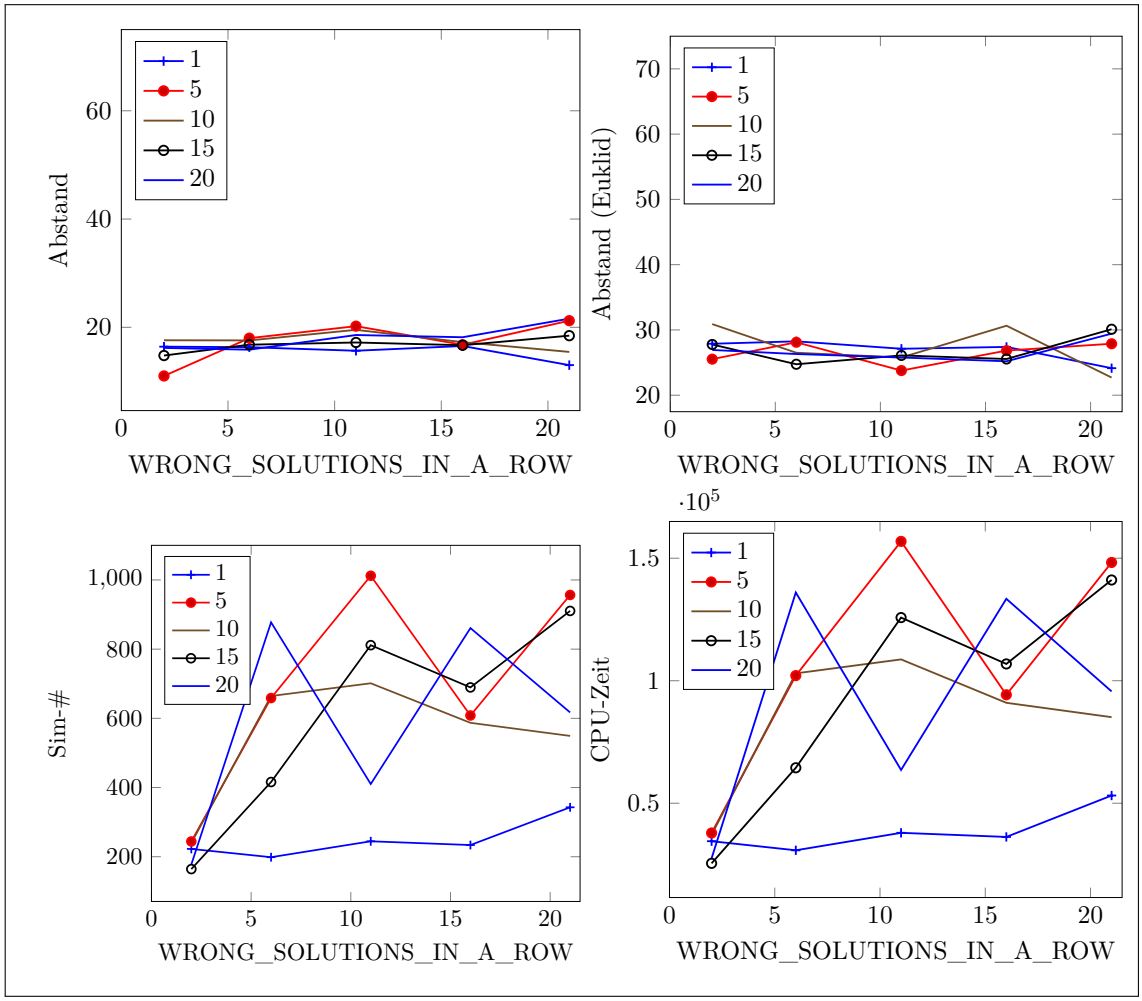


Abbildung 8.10: Hill-Climbing & diskret Auf/Ab auf Benchmark-Funktion Schwefel.
Alle Messergebnisse finden sich in Tabelle B.8 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweils 1, 5, 10, 15, 20 Fehlversuchen pro Richtung (WRONG_SOLUTIONS_PER_DIR).
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

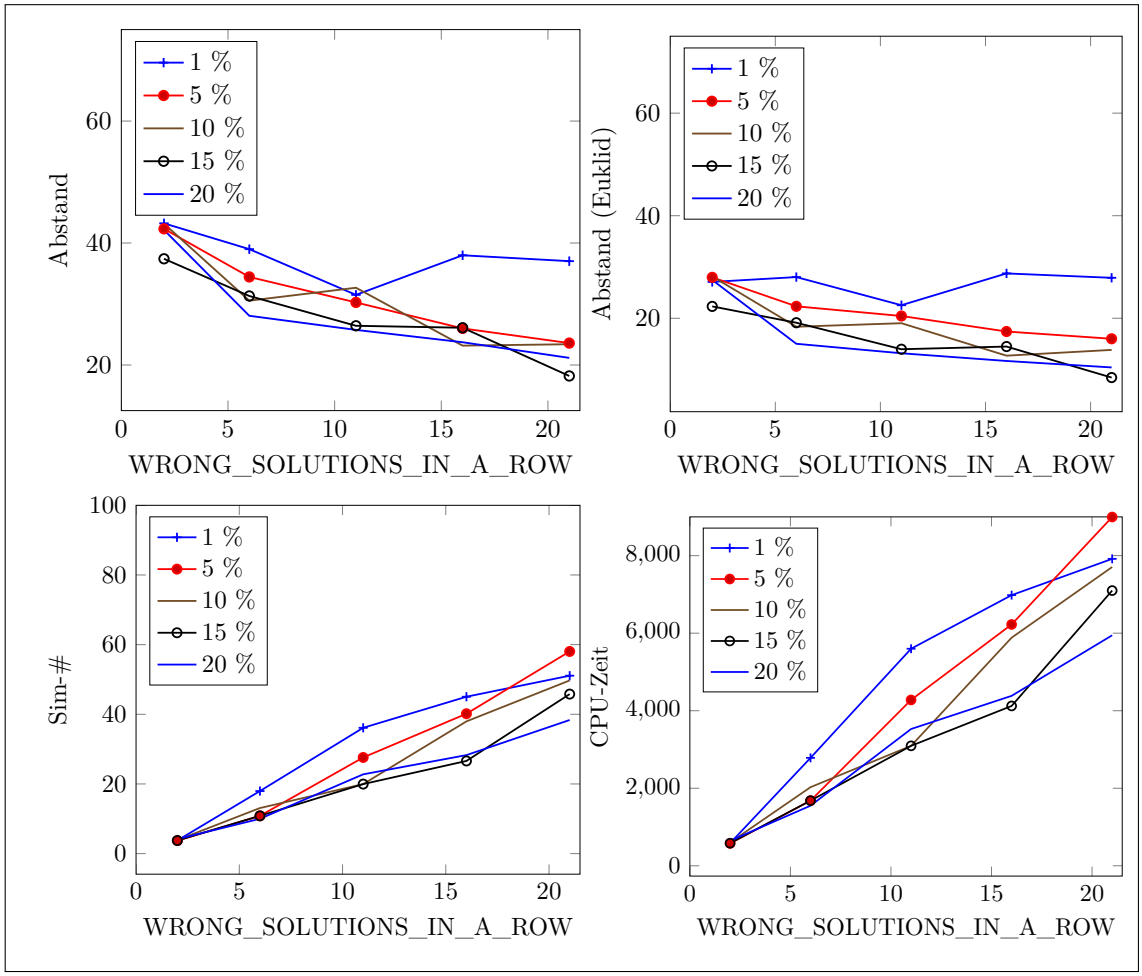


Abbildung 8.11: Hill-Climbing & zufällig, diskret in Nachbarschaft auf Benchmark-Funktion Ackley.
Alle Messergebnisse finden sich in Tabelle B.9 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweiliger Größe (in %) der Nachbarschaft für die Bestimmung des nächsten Parametersatzes.
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

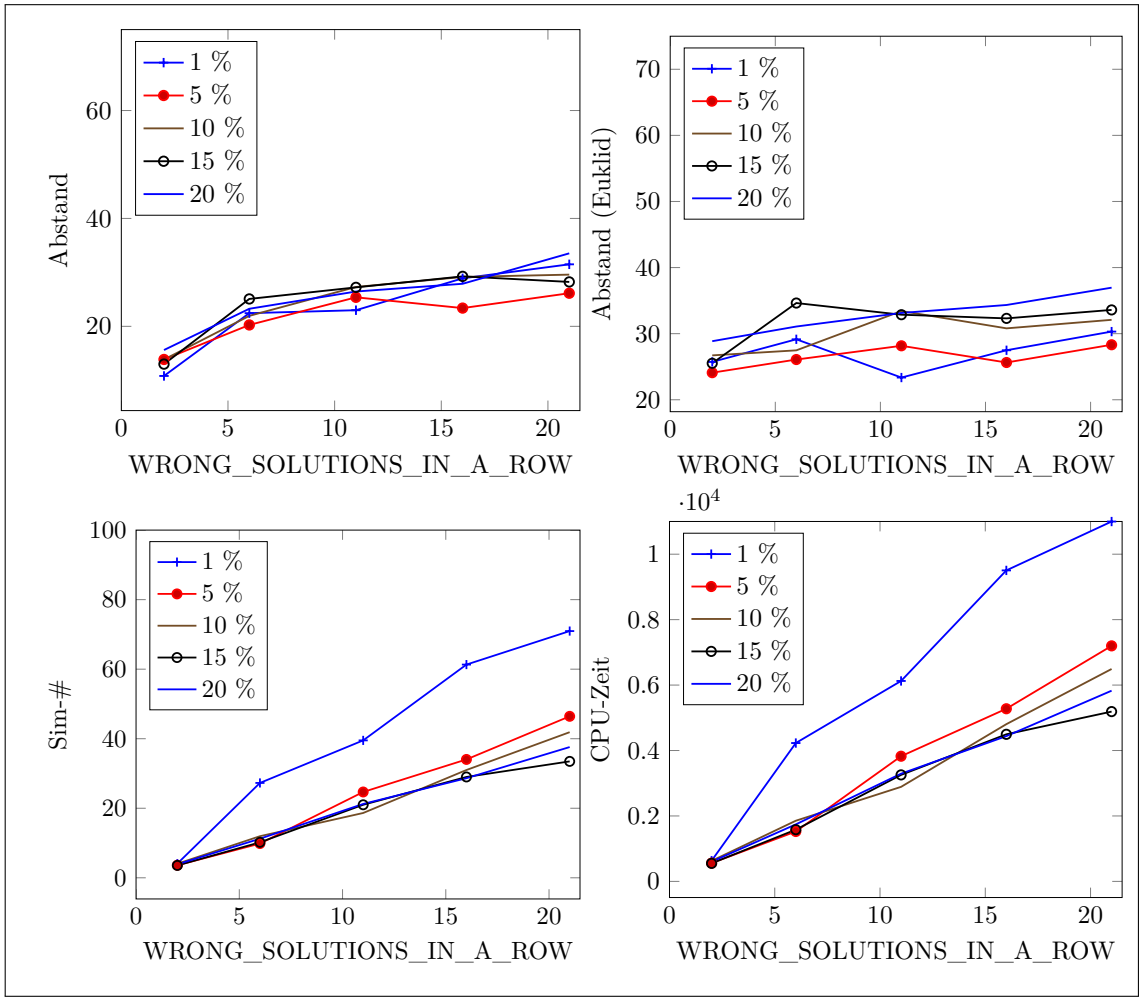


Abbildung 8.12: Hill-Climbing & zufällig, diskret in Nachbarschaft auf Benchmark-Funktion Schwefel.
Alle Messergebnisse finden sich in Tabelle B.10 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweiliger Größe (in %) der Nachbarschaft für die Bestimmung des nächsten Parametersatzes.
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

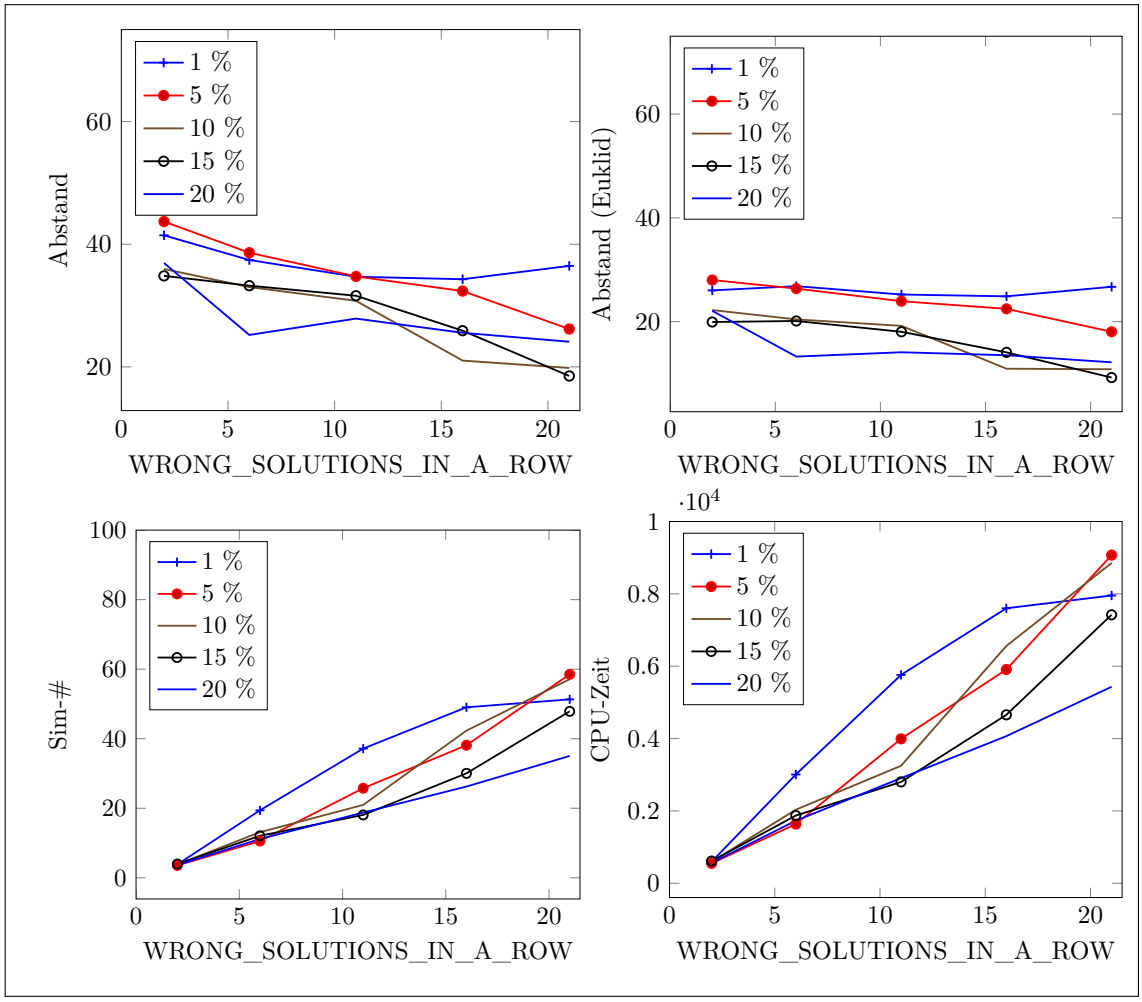


Abbildung 8.13: Hill-Climbing & zufällig, kontinuierlich in Nachbarschaft auf Benchmark-Funktion Ackley
Alle Messergebnisse finden sich in Tabelle B.11 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweiliger Größe (in %) der Nachbarschaft für die Bestimmung des nächsten Parametersatzes.

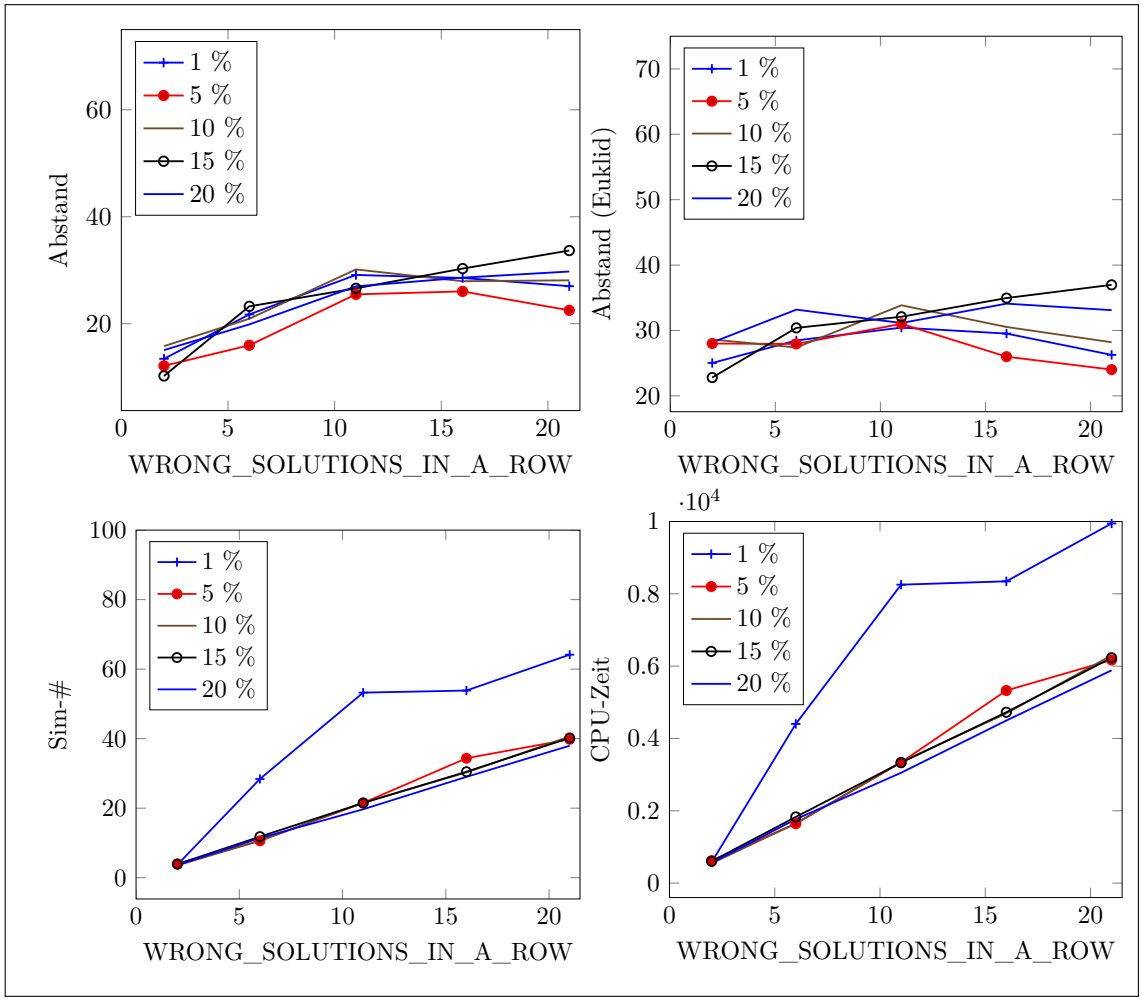


Abbildung 8.14: Hill-Climbing & zufällig, kontinuierlich in Nachbarschaft auf Benchmark-Funktion Schwefel.
Alle Messergebnisse finden sich in Tabelle B.12 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweiliger Größe (in %) der Nachbarschaft für die Bestimmung des nächsten Parametersatzes.
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

Basis-SCPN (keine lokalen Optima)

Bei der Optimierung des Basis-SCPN sind ähnliche Ergebnisse zu erwarten wie bei der Verwendung von Benchmark-Funktionen mit wenigen bis keinen lokalen Optima. Die auf Seite 43 dargestellte Wertelandschaft lässt dies vermuten. Die lokalen

Optima an entgegengesetzten Enden des Wertebereichs stellen dennoch eine Herausforderung für Optimierungsheuristiken dar (siehe Tabelle 8.1).

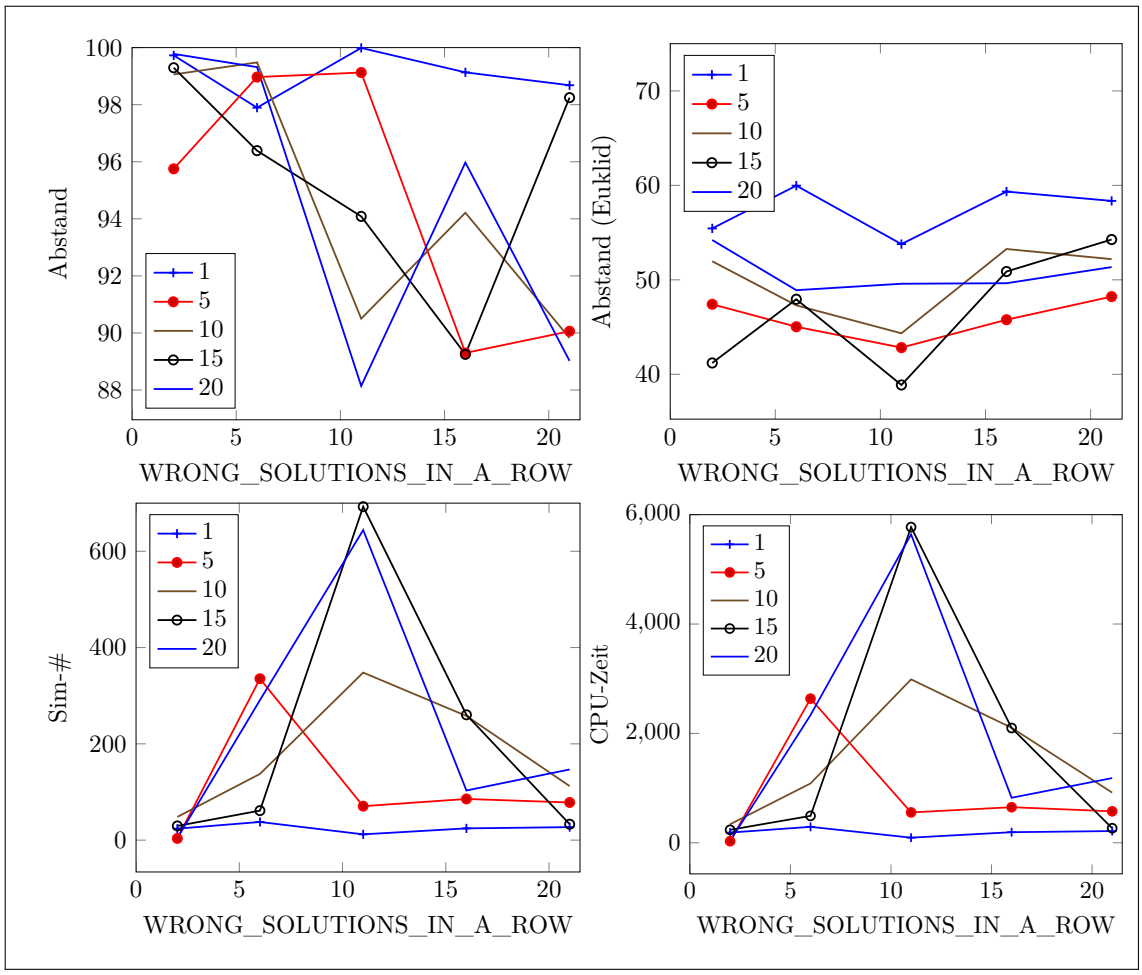


Abbildung 8.15: Basis-SCPN-Optimierung mit Hill-Climbing & diskret Auf/Ab.
Alle Messergebnisse finden sich in Tabelle B.13 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweils 1, 5, 10, 15, 20 Fehlversuchen pro Richtung (WRONG_SOLUTIONS_PER_DIR).
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

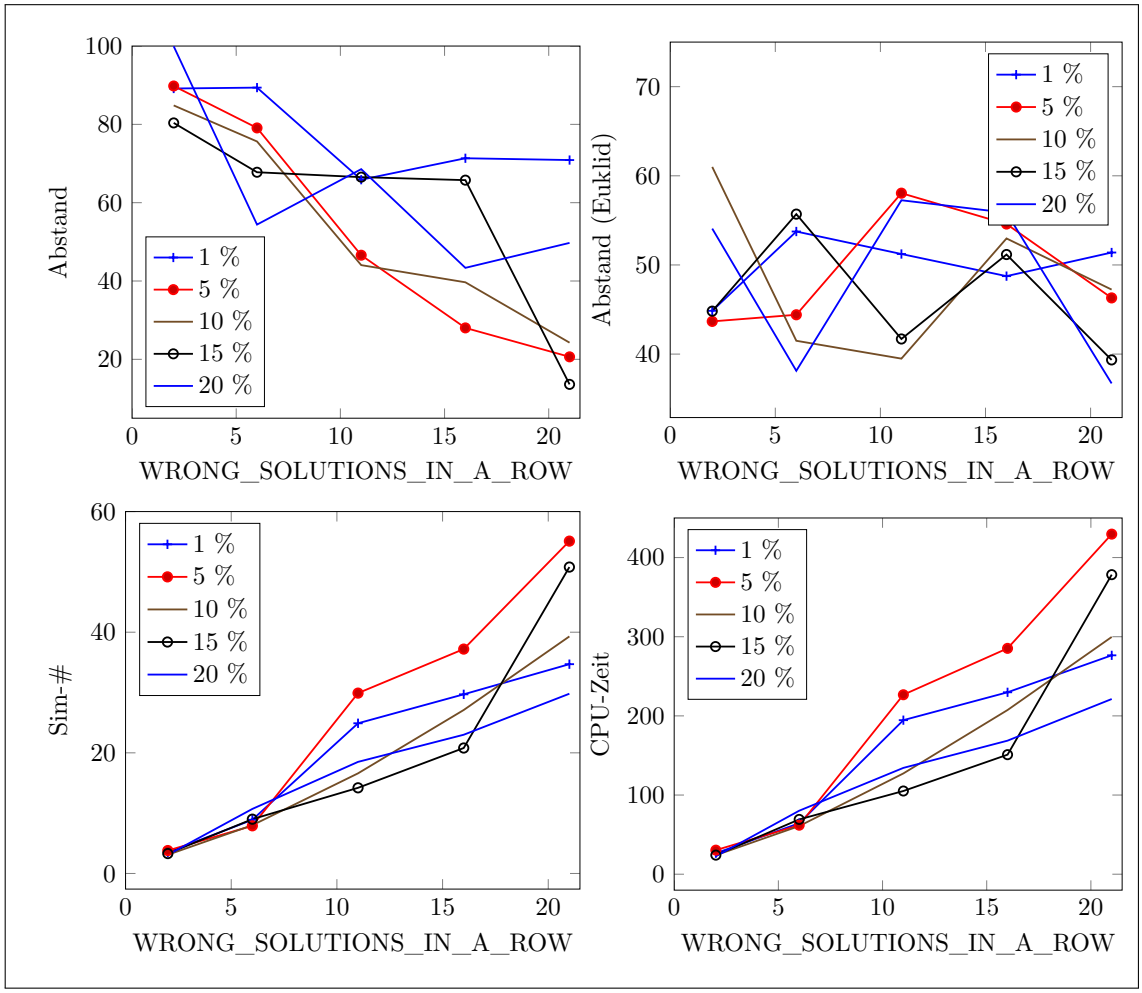


Abbildung 8.16: Basis-SCPN-Optimierung mit Hill-Climbing & zufällig, diskret in Nachbarschaft.
Alle Messergebnisse finden sich in Tabelle B.14 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweiliger Größe (in %) der Nachbarschaft für die Bestimmung des nächsten Parametersatzes.
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

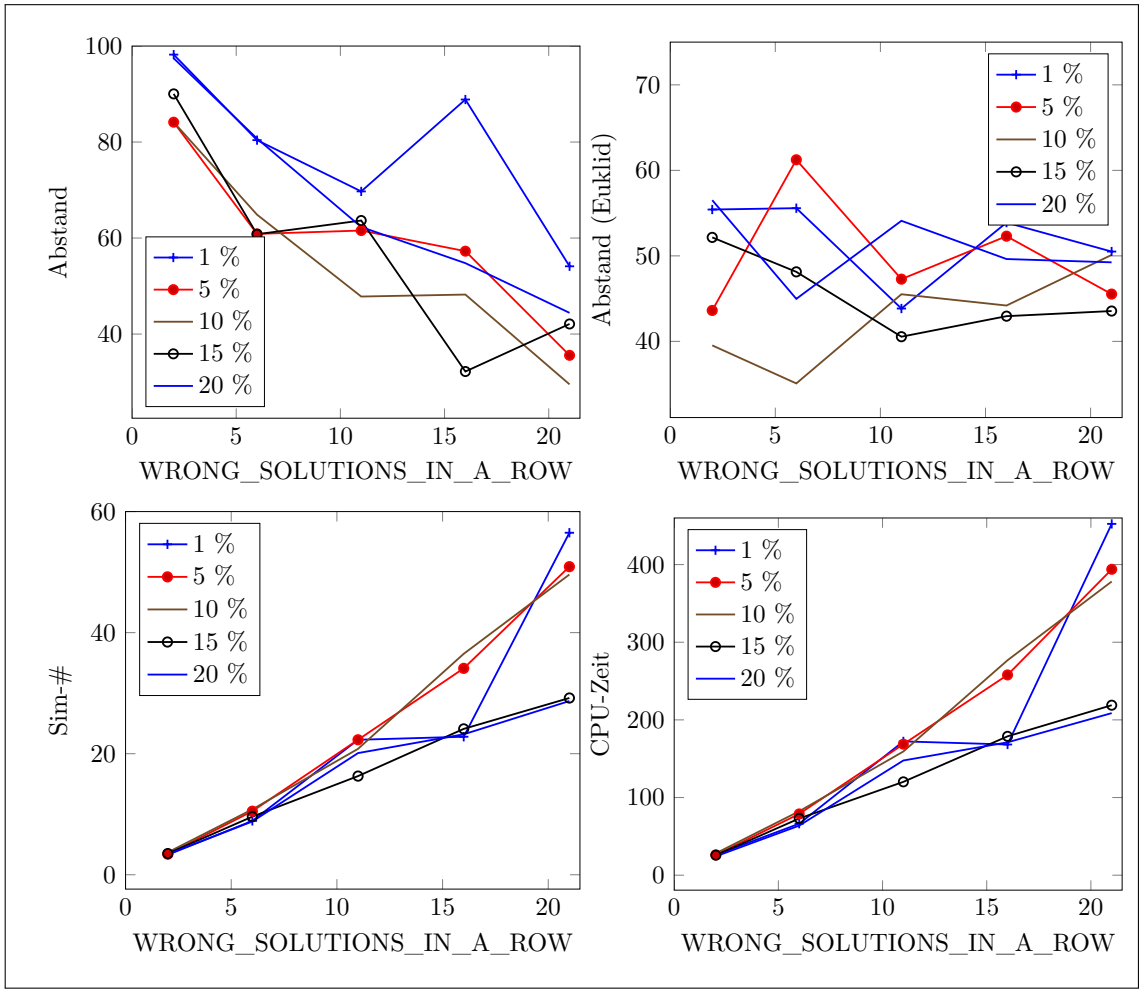


Abbildung 8.17: Basis-SCPN-Optimierung mit Hill-Climbing & zufällig, kontinuierlich in Nachbarschaft.
Alle Messergebnisse finden sich in Tabelle B.15 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweiliger Größe (in %) der Nachbarschaft für die Bestimmung des nächsten Parametersatzes.
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

Energiemodell-SCPN (viele lokale Optima)

Wie in Bild 7.7 und Tabelle 8.3 dargestellt, besitzt das Hausenergienetz viele lokale Optima. Trotz des kleineren Wertebereichs (241081 Werte) stellt dies eine entsprechende Herausforderung für Optimierungsheuristiken dar.

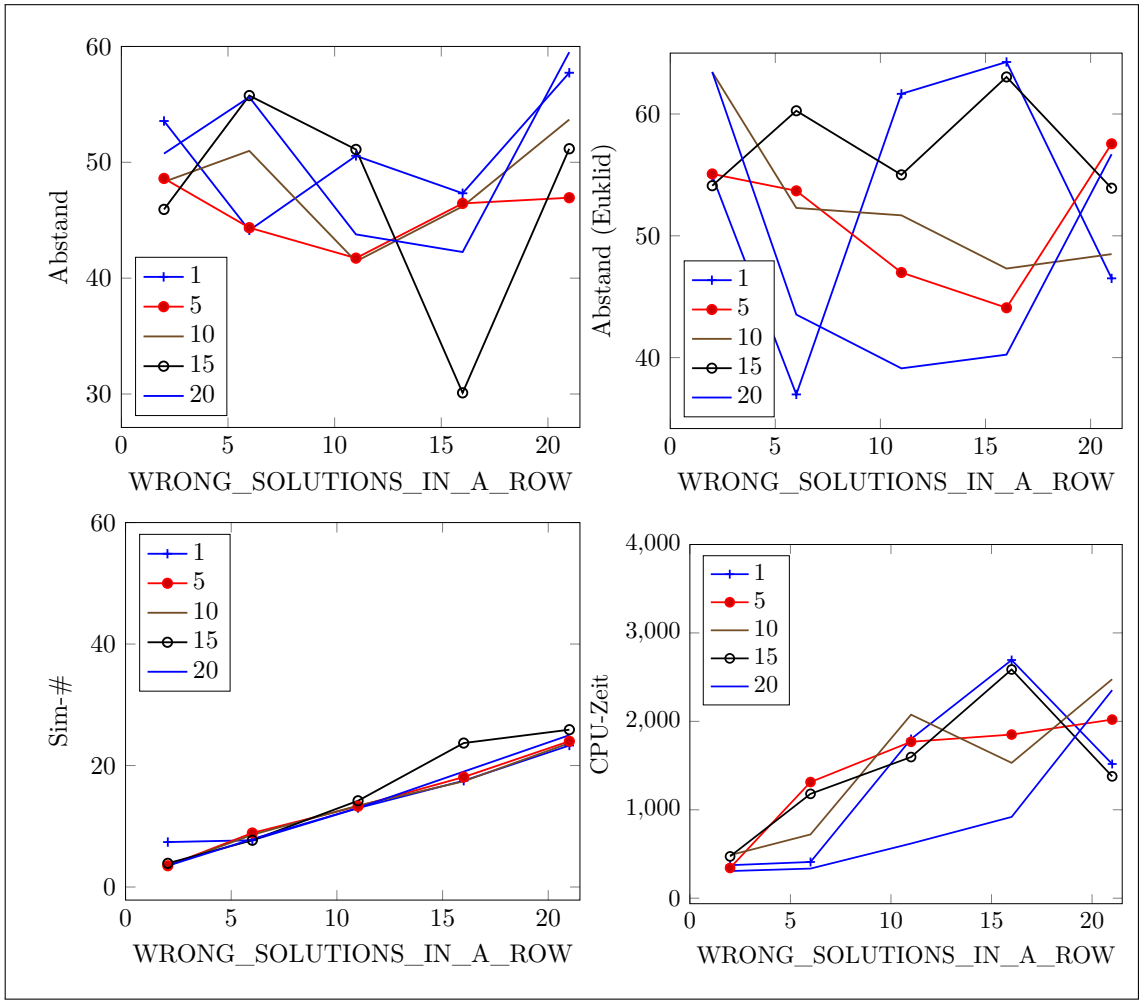


Abbildung 8.18: Optimierung des Hausenergiemodells mit Hill-Climbing & diskret Auf/Ab.
Alle Messergebnisse finden sich in Tabelle B.16 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweils 1, 5, 10, 15, 20 Fehlversuchen pro Richtung (WRONG_SOLUTIONS_-PER_DIR).
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

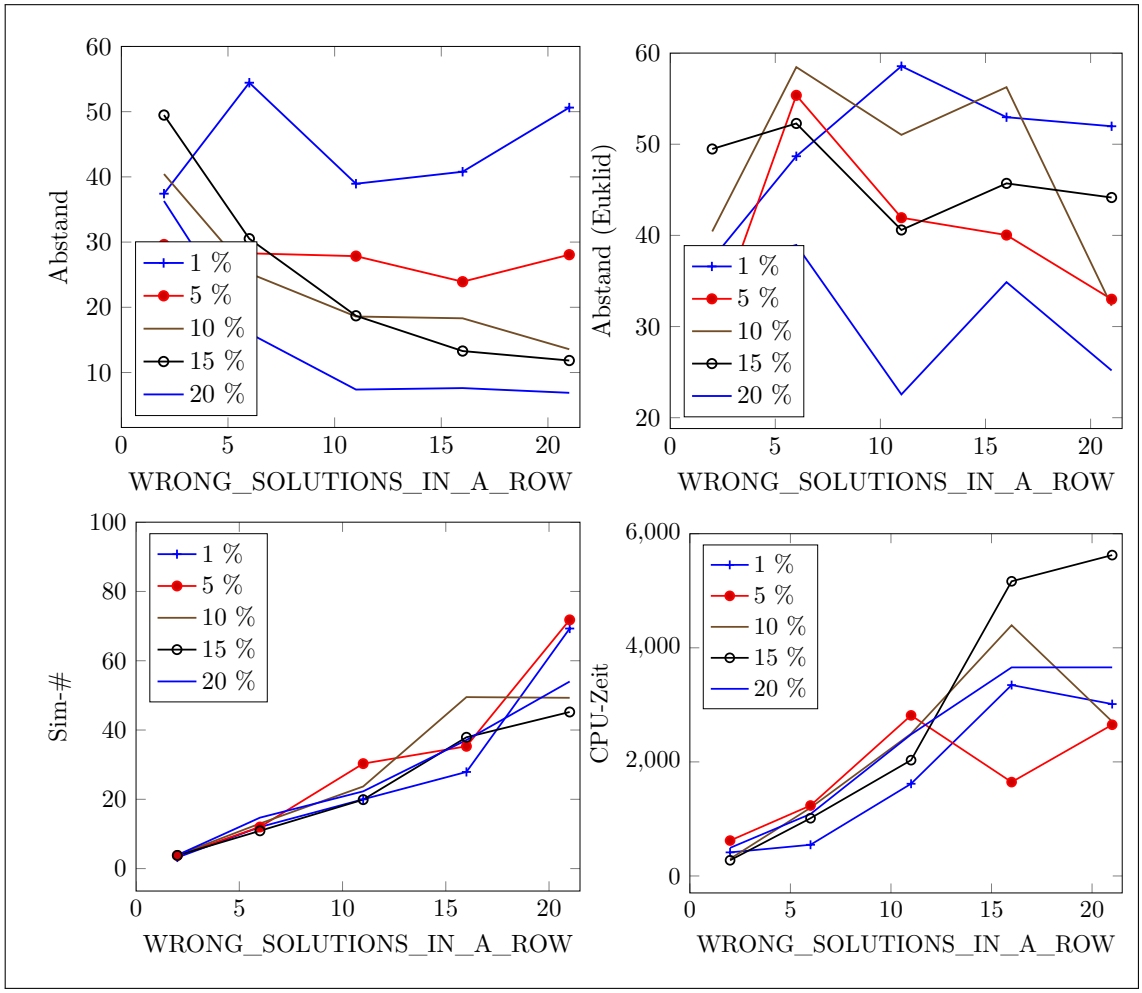


Abbildung 8.19: Optimierung des Hausenergiemodells mit Hill-Climbing & zufällig, diskret in Nachbarschaft.
Alle Messergebnisse finden sich in Tabelle B.17 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweiliger Größe (in %) der Nachbarschaft für die Bestimmung des nächsten Parametersatzes.
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

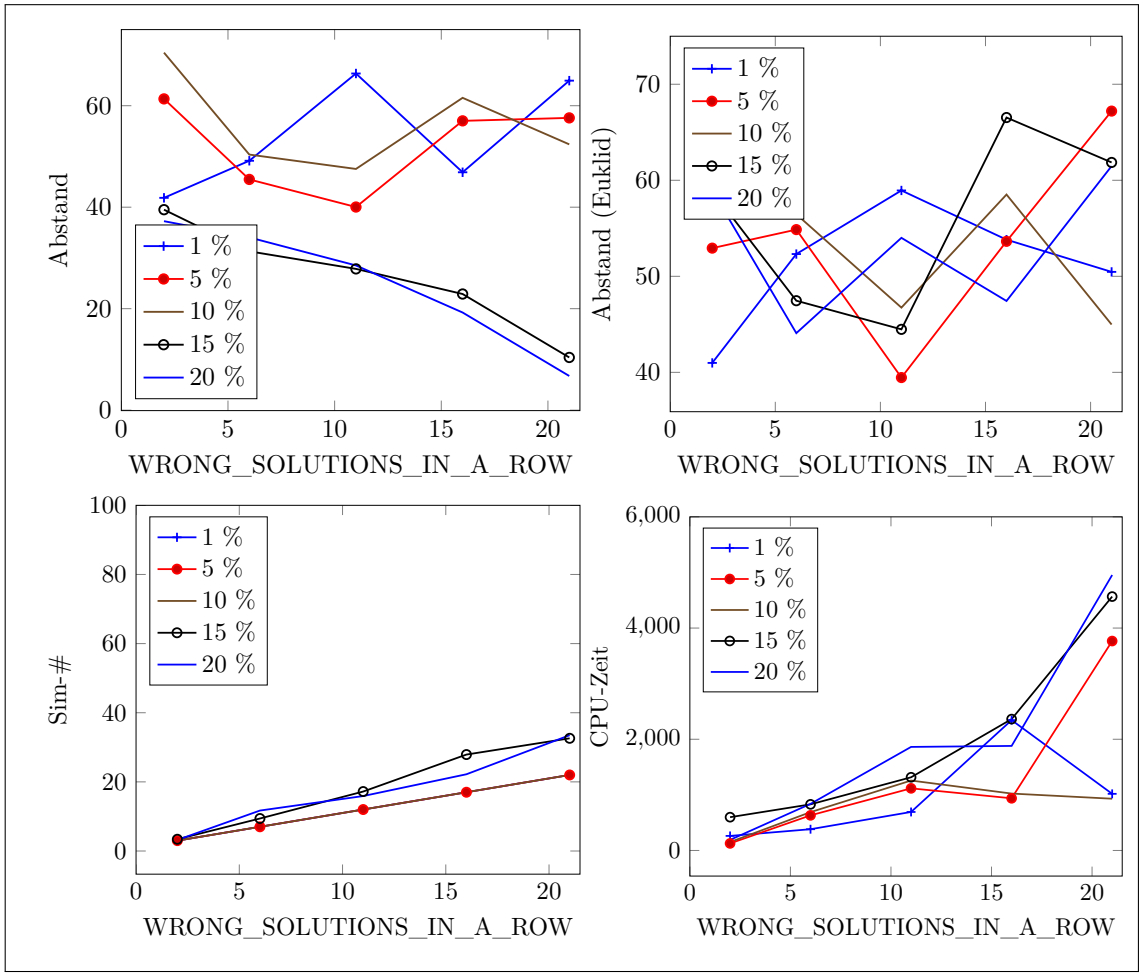


Abbildung 8.20: Optimierung des Hausenergiemodells mit Hill-Climbing & zufällig, kontinuierlich in Nachbarschaft.
Alle Messergebnisse finden sich in Tabelle B.18 im Anhang.
Die einzelnen Graphen zeigen die Versuchsgruppen mit jeweiliger Größe (in %) der Nachbarschaft für die Bestimmung des nächsten Parametersatzes.
WRONG_SOLUTIONS_IN_A_ROW ist die Anzahl erlaubter, aufeinanderfolgender Fehlversuche.

Die Ergebnisse der Optimierungsläufe bestätigen die Vermutung, dass Hill-Climbing sehr gut für die Optimierung geeignet ist, wenn es keine oder wenige lokale Optima gibt und die Kostenfunktion stetig verläuft. Des Weiteren zeigt sich, dass durch Variation der beiden Abbruchbedingungen sehr unterschiedliche Ergebnisse erzielt werden. Im schlimmsten Fall könnte der Optimierungsalgorithmus durch falsche

Konfiguration zur Analyse des gesamten Definitionsraumes gebracht werden.
Die besondere Form der Matya-Benchmarkfunktion offenbart eine Schwäche von Hill-Climbing (Plots 8.4, 8.6, 8.8). Es gibt sehr viele Parametersätze, deren Funktionswerte nahe beieinander liegen. Dadurch liefert der Algorithmus Ergebnisse, die zwar im Wertebereich, nicht jedoch im Definitionsbereich nahe am globalen Optimum liegen. Ähnliches zeigt sich auch bei der Analyse des Basis-SCPN, welches ein lokales Optimum hat, das im Definitionsbereich sehr weit vom globalen Optimum entfernt liegt.
Des Weiteren zeigt sich, dass Hill-Climbing -egal in welcher Ausprägung- nur sehr eingeschränkt für Kostenfunktionen mit sehr vielen lokalen Optima geeignet ist. Am Beispiel des Hausenergiemodells (Plot 8.18) sowie der beiden Benchmark-Funktionen Ackley (Plot 8.9) und Schwefel (Plot 8.10) zeigt sich, wie unwahrscheinlich das Finden des globalen Optimums in diesen Fällen ist. Eine zufallsbehaftete Parameterbestimmung liefert hier etwas bessere Ergebnisse, wie in den Bildern 8.19, 8.11 und 8.12 zu sehen.
Eine Besonderheit der simulationsbasierten Optimierung zeigt sich beim Basis-SCPN. Dessen Kostenfunktion (siehe Seite 43) ist zwar stetig und ohne lokale Optima. Dennoch liefert Hill-Climbing in der Standardvariante sehr schlechte Ergebnisse, unabhängig von der Anzahl erlaubter Fehlversuche. Der Grund hierfür liegt in der Natur der Simulation. Auch wenn die Kostenfunktion scheinbar stetig ist, liefern viele Parametersätze stark abweichende Werte im Vergleich zu ihren direkten Nachbarn, da das Simulationsergebnis immer auch zufallsbehaftet ist. Unter diesen Umständen arbeiten -ebenfalls- zufallsbehaftete Varianten deutlich zuverlässiger. Eine erhöhte Anzahl erlaubter Fehlversuche führt dabei zu einer starken Ergebnisverbesserung, siehe Plots 8.16, 8.16, 8.19 und 8.20.

8.3.2 Simulated Annealing

Der grundsätzliche Ablauf von Simulated Annealing ist in Kapitel 2.4.2 beschrieben. Im Optimierungstool wurden verschiedene Verfahren umgesetzt, um diese vergleichen zu können.
Die entscheidenden Variablen bei Simulated Annealing sind die Art der Abkühlungsfunktion sowie die Temperaturgrenze. Da bei gleicher Temperaturgrenze die Anzahl an Simulationsläufen zwischen den Abkühlungsfunktionen sehr stark variiert, wurde

dies in der Untersuchung vereinheitlicht. Für alle Untersuchungen wurde eine Grenze gewählt, die ca. 100 Simulationsläufe verursacht. Dadurch unterscheiden sich die verwendeten Abkühlungsfunktionen primär im Anstieg. Schwerpunkt der Experimente war es, den Einfluss verschiedener Strategien für die Berechnung der jeweils nächsten Parametersätze zu untersuchen. Tabellen 8.4, 8.5, 8.6 und 8.7 zeigen die Ergebnisse bei Anwendung der bekannten Benchmark-Funktionen.
Die Tabellen zeigen nur den jeweils erreichten relativen Abstand im Wertebereich sowie den euklidischen Abstand im Definitionsbereich des gefundenen Optimums zum tatsächlichen. Ein Vergleich der geschätzten CPU-Zeit und Simulationsschritte entfällt hier, da dieser für alle Optimierungsläufe gleich ist. Theoretisch werden 1020 Simulationsschritte mit insgesamt 16422 Sekunden CPU-Zeit pro Optimierung verwendet. Die Anzahl an Simulationsläufen ist, wie eingangs erwähnt, auf 100 beschränkt. Durch Details der Implementierung von Vergleichsoperationen für Fließkommazahlen schwankt die tatsächliche Anzahl an Simulationsläufen zwischen den Simulationsläufen um ca.$\pm$2%.
Bemerkenswert ist, dass alle Implementierungsvarianten auf allen Benchmark-Funktionen und für beide Beispielnetze eine relativ hohe Cache-Trefferrate aufweisen.
Die Ergebnisse der Versuche mit dem Basis-SCPN sind jedoch deutlich schlechter als erwartet, obwohl die Wertelandschaft ähnlich ist wie bei Benchmark-Funktionen ohne lokale Optima. Simulated Annealing liefert hier ähnlich schlechte Ergebnisse wie bei der Optimierung von Benchmark-Funktionen mit vielen lokalen Optima. Das lässt den Schluss zu, dass diese Heuristik für die simulationsbasierte Optimierung grundsätzlich eher ungeeignet ist, unabhängig von der (vermuteten) Wertelandschaft.

Benchmark-Funktionen mit wenigen/keinen lokalen Optima

Aus den Tabellen 8.4 und 8.5 geht hervor, wie sich die unterschiedlichen Implementierungen von Simulated Annealing bei Anwendung auf Kostenfunktionen ohne lokale Optima verhalten. Offensichtlich eignen sich alle Implementierungen für derartige Probleme. Die Standardvariante bzw. die Standardvariante mit schrittweiser Parameterbestimmung schneidet stets am besten ab. Mit Blick auf die Antwortfunktionsgebirge (siehe Seite 126) der Benchmark-Funktionen ist dies auch nicht überraschend.

Abkühlungs-funktion	Parameter-bestimmung	Abstand (%)	Abstand-Euklid (%)	Cache-ratio (%)
Boltzmann	Standard	0,503	10,215	25,98
	Standard-stepwise	0,693	13,018	24,51
	Simple	0,922	21,791	25,00
	Simple-stepwise	0,731	15,256	24,51
Fast	Standard	1,959	29,733	21,53
	Standard-stepwise	2,120	30,674	27,04
	Simple	2,659	39,329	25,41
	Simple-stepwise	1,814	29,055	26,43
Very Fast	Standard	0,524	11,433	25,59
	Standard-stepwise	0,820	16,099	25,98
	Simple	1,064	19,905	23,73
	Simple-stepwise	0,848	16,117	24,12

Tabelle 8.4: Simulated Annealing: Matya

Abkühlungs-funktion	Parameter-bestimmung	Abstand (%)	Abstand-Euklid (%)	Cache-ratio (%)
Boltzmann	Standard	0,252	8,129	27,65
	Standard-stepwise	0,477	10,951	26,96
	Simple	1,199	18,823	20,10
	Simple-stepwise	0,974	16,396	25,10
Fast	Standard	1,446	20,465	23,57
	Standard-stepwise	1,168	17,717	23,27
	Simple	1,914	22,048	28,27
	Simple-stepwise	2,124	23,583	26,63
Very Fast	Standard	0,454	10,911	25,49
	Standard-stepwise	0,483	10,792	26,57
	Simple	0,833	15,156	21,18
	Simple-stepwise	0,498	11,420	25,49

Tabelle 8.5: Simulated Annealing: Sphere

Benchmark-Funktionen mit vielen lokalen Optima

Kostenfunktionen mit vielen lokalen Optima stellen auch für Simulated Annealing eine Herausforderung dar. Aus den Ergebnistabellen 8.6 und 8.7 ist ersichtlich, dass die gefundenen Werte teils deutlich vom tatsächlichen Optimum abweichen. Dass der euklidische Abstand im Definitionsbereich bei Schwefel besonders groß ist, ist dabei hervorzuheben (Plot auf Seite 127).

Abkühlungs-funktion	Parameter-bestimmung	Abstand (%)	Abstand-Euklid (%)	Cache-ratio (%)
Boltzmann	Standard	25,425	12,095	26,176
	Standard-stepwise	23,253	11,213	26,176
	Simple	34,031	20,609	24,804
	Simple-stepwise	30,717	16,099	20,686
Fast	Standard	34,955	20,904	22,653
	Standard-stepwise	30,485	16,603	24,898
	Simple	32,898	18,986	25,204
	Simple-stepwise	33,802	20,631	27,755
Very Fast	Standard	26,828	12,868	25,882
	Standard-stepwise	24,355	11,847	26,176
	Simple	26,680	13,427	22,647
	Simple-stepwise	24,298	11,927	22,157

Tabelle 8.6: Simulated Annealing: Ackley

Abkühlungs-funktion	Parameter-bestimmung	Abstand (%)	Abstand-Euklid (%)	Cache-ratio (%)
Boltzmann	Standard	35,936	44,510	28,235
	Standard-stepwise	34,577	44,739	27,451
	Simple	31,641	33,768	23,039
	Simple-stepwise	32,824	35,739	21,961
Fast	Standard	29,451	42,722	24,592
	Standard-stepwise	29,777	42,009	24,898
	Simple	26,854	39,391	25,816
	Simple-stepwise	24,174	45,349	25,408
Very Fast	Standard	33,839	41,537	25,686
	Standard-stepwise	34,619	40,078	24,412
	Simple	30,268	37,106	22,451
	Simple-stepwise	29,964	37,081	24,510

Tabelle 8.7: Simulated Annealing: Schwefel

Basis-SCPN (keine lokalen Optima)

Tabelle 8.8 zeigt die Ergebnisse des Vergleichstests mit gleichen Konfigurationen für Simulated Annealing bei echter Simulation des Basisnetzes. Es fallen ähnliche Unterschiede wie bei der Optimierung mit Hill-Climbing auf. Aufgrund der im Voraus berechneten Wertelandschaft ist davon auszugehen, dass die Ergebnisse ähnlich vielversprechend sind, wie bei Benchmark-Funktionen ohne lokale Optima. Jedoch liegen die gefundenen Werte mit 15 - 45 % Abstand weit vom tatsächlichen Optimum entfernt. Eine Erklärung könnte auch hier im Einfluss zufälliger Faktoren bei der Simulation liegen, was auch die Optimierung mit Hill-Climbing erschwert (siehe Plot 8.15).

Abkühlungs-funktion	Parameter-bestimmung	Abstand (%)	Abstand-Euklid (%)	Cache-ratio (%)
Boltzmann	Standard	34,99	36,93	0,29
	Standard-stepwise	25,64	51,42	0,30
	Simple	19,32	73,96	0,22
	Simple-stepwise	22,50	65,40	0,22
Fast	Standard	33,54	63,61	0,24
	Standard-stepwise	45,42	72,60	0,27
	Simple	28,90	82,01	0,28
	Simple-stepwise	19,84	81,62	0,27
Very Fast	Standard	21,74	54,35	0,27
	Standard-stepwise	15,15	50,98	0,28
	Simple	20,50	71,04	0,24
	Simple-stepwise	24,35	74,86	0,25

Tabelle 8.8: Simulated Annealing: Basis-SCPN

Energiemodell-SCPN (viele lokale Optima)

Ein völlig anderes Bild ergibt sich bei der Simulation des Hausenergiemodells. Aufgrund der diffusen Wertelandschaft mit vielen lokalen Optima wäre zu erwarten, dass das globale Optimum nur selten gefunden wird. Wie aus Tabelle 8.9 hervorgeht, wird das globale Optimum aber praktisch immer gefunden oder zumindest mit vernachlässigbarem Abstand angenähert. Unterschiede sind eher in der Cache-Trefferrate (cache-ratio) zu sehen, obgleich sie bei allen Versuchen vergleichsweise hoch ist. Die relativ grobe Diskretisierung des Definitionsraumes ist hierfür möglicherweise die Ursache. Auch die Tatsache, dass schrittweises Vorgehen im Prinzip gleich gute Ergebnisse erzielt wie die kontinuierliche Parameterwertbestimmung, deutet darauf hin.

Abkühlungs-funktion	Parameter-bestimmung	Abstand (%)	Abstand-Euklid (%)	Cache-ratio (%)
Boltzmann	Standard	0,040	0,144	0,30
	Standard-stepwise	0,040	0,144	0,88
	Simple	0,040	0,144	0,99
	Simple-stepwise	0,040	0,144	0,99
Fast	Standard	0,000	0,000	0,28
	Standard-stepwise	0,004	0,014	0,92
	Simple	0,040	0,144	0,99
	Simple-stepwise	0,040	0,144	0,99
Very Fast	Standard	0,040	0,144	0,27
	Standard-stepwise	0,040	0,144	0,89
	Simple	0,040	0,144	0,99
	Simple-stepwise	0,040	0,144	0,99

Tabelle 8.9: Simulated Annealing: Hausenergiemodell

8.3.3 Genetische Algorithmen (SBX)

In ersten Voruntersuchungen wurden die Einflüsse verschiedener Konfigurationsmöglichkeiten der umgesetzten genetischen Algorithmen ergründet. Die beiden Parameter mit dem größten Einfluss auf die Qualität des gefundenen Optimums sind die Größe der Population und die Mutationswahrscheinlichkeit. Ihr Einfluss bei verschiedenen Kostenfunktionen wird in den Ergebnisplots dieses Abschnitts dargestellt.
Erwähnenswert ist, dass die Anzahl an Optimierungsläufen ohne Verbesserung sowie die Kreuzungsanzahl pro Generation keinen nennenswerten Einfluss auf die Qualität des gefundenen Optimums haben. Von einer näheren Untersuchung des Einflusses dieser Parameter wurde daher im Rahmen dieser Arbeit abgesehen.
Schon bei der Untersuchung der Benchmark-Funktionen fällt auf, dass mit steigender Mutationswahrscheinlichkeit und Populationsgröße auch die Chance steigt, das globale Optimum zu finden. Weit auseinander liegende lokale Optima, z.B. bei Matya, stellen dabei dennoch eine Herausforderung dar, wie die Ergebnisse in Tabelle 8.10 verdeutlichen. Bei vielen lokalen Optima liefert eine hohe Mutationswahrscheinlichkeit ebenfalls relativ gute Ergebnisse bei geringer Simulationsanzahl, wie Tabelle 8.11 zeigt. Hier scheint jedoch der Einfluss der Populationsgröße wichtiger zu sein.
Die hier dargestellten Ergebnisse geben einen Überblick über den Einfluss der beiden Konfigurationsparameter. Die kompletten Ergebnisse sind im Anhang in Tabelle

B.20 zu finden. Dabei fallen auch Sonderfälle mit 0 % Mutationswahrscheinlichkeit oder Populationsgrößen von 1 auf. Bei 0 % Mutationswahrscheinlichkeit wird die gleiche Population mehrfach untersucht, was zu einer Cache-Trefferrate von 100 % führt. Je höher die Mutationswahrscheinlichkeit ist, desto geringer ist die Chance, eine Parameterkombination doppelt zu wählen und damit einen Cache-Treffer zu erzeugen. Der gesamte Definitionsraum wird damit umso umfassender analysiert, je höher die gewählte Wahrscheinlichkeit ist. Dies ist insbesondere für die künftige Optimierung von Netzen hilfreich, deren vermutete Kostenfunktion viele lokale Optima hat.

Die Optimierung von realen SCPNs zeigt vollkommen andere Erbnisse als die der Benchmark-Funktionen. So scheint hier eine Erhöhung der Mutationswahrscheinlichkeit nur geringen Einfluss auf die Ergebnisqualität zu haben. Viel wichtiger ist der Einfluss der Populationsgröße. So lassen sich bereits mit 10 % Mutationswahrscheinlichkeit Ergebnisse finden, die sehr nahe am globalen Optimum liegen. Hervorzuheben ist, dass dies nur für die Nähe des gefundenen Optimums im Wertebereich der Kostenfunktion gilt. Im Definitionsbereich liegen die gefundenen Werte teilweise zu weit entfernt, um in der Praxis von Nutzen sein zu können. Überraschend ist, dass dieser Effekt sowohl beim Basis-SCPN (Tabelle 8.12) mit wenigen lokalen Optima als auch beim Hausenergiemodell (Tabelle 8.13) mit vielen lokalen Optima (bei deutlich kleinerem Definitionsbereich) auftritt.

Zusammenfassend lässt sich sagen, je größer die Population und Mutationswahrscheinlichkeit ist, desto größer die Chance, das globale Optimum zu finden. Folglich eignet sich SBX sehr gut für die Optimierung mit verteilter bzw. paralleler Simulation. Durch Vorgabe der Populationsgröße lässt sich das Verfahren leicht anpassen, um unter bestmöglicher Ausnutzung der verfügbaren Ressourcen sehr gute Ergebnisse zu erzielen. Dennoch bleibt zu untersuchen, warum die Erhöhung der Mutationswahrscheinlichkeit bei realer Simulation nicht zu besseren Ergebnissen führt.

Populations-größe	Mutationswahr-scheinlichkeit (%)	Sphere				Matya			
		Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	Cache-ratio	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	Cache-ratio (%)
	10	1,724	21,172	7,2	33,333	10,175	24,567	6,3	33,333
	20	0,869	14,394	6,9	33,333	1,518	19,117	7,5	33,333
	30	1,553	21,285	6,6	33,333	2,608	12,085	7,5	33,333
	40	1,596	21,550	6,3	33,333	2,665	22,899	8,4	33,333
	50	0,906	15,531	7,5	33,333	4,801	20,851	6,6	33,333
5	10	0,361	9,632	17,5	11,905	0,912	12,536	20,3	12,333
	20	0,425	10,256	18,2	11,905	0,806	12,775	16,1	13,810
	30	0,299	8,550	19,6	13,286	0,950	19,589	16,8	12,143
	40	0,377	9,555	19,6	13,524	1,079	15,377	18,9	14,000
	50	0,573	12,475	17,5	12,024	1,049	13,624	18,2	12,857
10	10	0,234	7,743	31,2	7,708	0,450	9,790	38,4	7,542
	20	0,058	3,695	32,4	7,639	0,511	11,257	28,8	8,333
	30	0,231	7,618	31,2	8,125	0,381	8,390	27,6	7,500
	40	0,306	7,954	31,2	8,125	0,355	7,418	32,4	6,750
	50	0,075	4,469	33,6	7,847	0,661	14,950	32,4	8,333
15	10	0,098	4,332	47,6	5,471	0,329	9,010	42,5	5,490
	20	0,190	6,849	40,8	5,196	0,212	6,826	47,6	5,245
	30	0,092	4,915	39,1	5,392	0,444	9,287	42,5	5,686
	40	0,076	4,500	54,4	5,353	0,267	6,054	39,1	5,686
	50	0,078	4,603	47,6	5,686	0,265	8,768	39,1	5,686

Tabelle 8.10: SBX: Sphere / Matya

Populations-größe	Mutationswahr-scheinlichkeit (%)	Ackley				Schefel			
		Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	Cache-ratio (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	Cache-ratio (%)
5	10	32,109	17,117	6,6	33,333	37,612	28,592	6,9	33,333
	20	31,545	15,949	6,3	33,333	38,397	29,413	7,5	33,333
	30	32,533	17,835	6,3	33,333	35,050	29,787	7,5	33,333
	40	30,086	15,966	8,1	33,333	34,557	24,470	7,5	33,333
	50	38,920	22,220	7,5	33,333	31,939	28,806	7,8	33,333
	10	22,959	10,873	16,1	11,905	20,909	31,891	18,2	12,738
	20	22,912	10,751	14,0	12,857	22,363	33,695	17,5	13,095
	30	17,978	8,714	21,7	11,810	20,561	35,143	21,7	11,333
	40	20,062	8,522	20,3	13,000	22,837	29,383	17,5	12,738
	50	24,646	12,750	15,4	12,381	24,699	33,320	15,4	13,571
10	10	17,576	7,461	30,0	8,056	19,184	37,313	28,8	8,125
	20	19,505	8,405	37,2	7,917	21,660	32,693	31,2	7,014
	30	18,031	8,076	31,2	7,208	17,778	34,380	30,0	7,750
	40	14,436	4,998	32,4	7,847	22,959	35,049	33,6	8,167
	50	18,061	6,331	31,2	7,778	15,994	36,707	30,0	8,056
15	10	16,455	7,053	49,3	5,294	14,036	37,987	47,6	5,735
	20	14,854	6,003	47,6	5,882	15,153	38,168	47,6	5,049
	30	16,204	6,597	42,5	5,882	18,116	35,200	42,5	5,588
	40	17,231	7,016	47,6	5,588	16,797	32,899	44,2	5,441
	50	17,058	5,488	44,2	5,539	15,669	36,646	49,3	5,392

Tabelle 8.11: SBX: Ackley / Schwefel

Populations-größe	Mutationswahr-scheinlichkeit (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sek.)	Cache-ratio (%)
1	10	79,729	30,322	7,8	58,2	33,333
	20	71,790	51,832	7,5	54,7	33,333
	30	71,805	41,254	6,9	51,3	33,333
	40	67,691	39,955	6,9	50,2	33,333
	50	66,367	34,309	6,9	50,7	33,333
5	10	22,164	31,063	16,8	130,3	14,286
	20	35,500	22,305	18,2	140,8	14,286
	30	49,492	30,921	15,4	118,2	14,286
	40	30,287	25,809	18,2	139,7	14,286
	50	28,325	29,483	21,7	166,1	14,286
10	10	19,897	27,758	31,2	242,7	8,333
	20	26,683	13,950	31,2	242,6	8,333
	30	26,260	33,134	31,2	241,5	8,333
	40	25,431	20,889	31,2	240,6	8,333
	50	32,930	41,980	30,0	235,1	8,333
15	10	20,846	33,890	42,5	335,0	5,882
	20	12,933	30,670	47,6	375,5	5,882
	30	15,665	27,097	44,2	348,5	5,882
	40	16,657	11,710	39,1	307,8	5,882
	50	16,497	44,541	45,9	364,9	5,882

Tabelle 8.12: SBX: Basis-SCPN

Populations-größe	Mutationswahr-scheinlichkeit (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sek.)	Cache-ratio (%)
1	0	20,183	53,835	11,0	2271,5	81,818
	10	17,077	68,723	8,4	1041,0	34,095
	20	17,243	39,235	8,2	912,7	38,667
	30	12,295	63,137	8,6	1366,8	36,000
	40	20,733	43,521	7,8	782,7	41,333
	50	19,084	49,864	7,8	816,8	41,333
5	0	5,255	76,202	30,4	7893,6	66,968
	10	2,705	63,184	21,7	2796,2	12,163
	20	3,405	52,693	22,2	2238,9	13,586
	30	3,735	56,393	21,5	2348,6	12,962
	40	5,033	66,786	23,0	2507,6	13,083
	50	3,965	65,103	21,4	2986,8	13,925
10	0	3,407	64,527	56,0	7744,6	64,286
	10	2,459	59,353	40,6	4416,1	7,425
	20	2,289	68,445	38,6	5372,8	7,803
	30	2,667	57,382	38,8	5158,1	7,811
	40	2,498	68,146	36,1	4573,6	7,057
	50	4,142	51,659	39,0	4703,3	7,740
15	0	2,732	71,272	80,4	12841,1	62,682
	10	1,882	68,133	57,0	7043,0	5,273
	20	1,991	66,308	59,8	7073,7	5,034
	30	1,753	53,051	56,8	6477,9	5,293
	40	2,196	65,187	58,6	7891,9	5,129
	50	1,953	60,052	59,0	7578,9	5,099

Tabelle 8.13: SBX: Hausenergiemodell

8.3.4 Zweiphasen-Optimierung

Die Optimierung mit zwei Phasen basiert im wesentlichen auf der wiederholten Anwendung von Simulated Annealing auf das gleiche Problem, wobei die zweite Phase als Startwert das gefundene Optimum der ersten Phase verwendet. Insofern entspricht es einer Art bedingungslosem Re-Annealing. In den Versuchen wurden die besten Konfigurationen von Simulated Annealing aus Abschnitt 8.3.2 genutzt und hinsichtlich der Verteilung der Simulationsanzahl auf die beiden Phasen untersucht. Die Gesamtzahl an Simulationen bleibt dabei stets gleich bei ca. 100. Die Begrenzung wird durch Variation von Epsilon, also der Abbruchtemperatur erreicht.

	Sphere			Matya		
Verhältnis Phase1/Phase2	Abstand (%)	Abstand-Euklid (%)	Cache-ratio (%)	Abstand (%)	Abstand-Euklid (%)	Cache-ratio (%)
10/90	2,548	25,079	93,905	3,645	33,340	94,000
20/80	1,623	17,336	89,714	2,966	24,103	90,190
30/70	1,384	19,903	85,143	2,657	30,750	85,048
40/60	1,992	23,536	80,667	1,529	29,307	81,143
50/50	2,745	26,478	77,429	1,471	34,334	76,571
60/40	2,357	22,222	71,810	0,953	25,805	71,905
70/30	4,391	35,055	68,000	1,069	33,233	68,190
80/80	4,747	39,029	63,524	0,959	28,259	76,970
90/10	5,994	44,736	59,238	0,648	35,418	60,000

Tabelle 8.14: Zweiphasige Optimierung: Sphere / Matya

Aus den Versuchen mit Benchmark-Funktionen lässt sich ableiten, dass bei gleicher Konfiguration von Simulated Annealing eine zweite Optimierungsphase relativ geringen Einfluss auf die Qualität des gefundenen Optimums hat. Teilweise sind die Ergebnisse schlechter, als mit Simulated Annealing allein. Ackley ist die einzige Funktion, bei der die zweiphasige Optimierung deutlich bessere Ergebnisse (Abstand 5 %) als die einzelne Ausführung von Simulated Annealing (23,3 %) liefert.
Das Verhältnis zwischen erster und zweiter Phase scheint keinen nennenswerten Einfluss auf die Ergebnisse zu haben. Im Rahmen einer Optimierung empfiehlt es sich anhand der hier ermittelten Ergebnisse, Simulated Annealing allein einzusetzen,

Verhältnis Phase1/Phase2	Ackley			Schwefel		
	Abstand (%)	Abstand-Euklid (%)	Cache-ratio (%)	Abstand (%)	Abstand-Euklid (%)	Cache-ratio (%)
10/90	4,918	14,514	94,190	26,827	26,006	93,524
20/80	14,496	26,203	89,429	30,857	19,990	88,762
30/70	11,292	12,372	85,333	18,194	23,613	84,000
40/60	16,554	28,180	81,143	32,675	27,492	79,143
50/50	19,905	25,133	76,476	30,966	39,075	75,192
60/40	25,826	35,199	72,476	29,034	22,196	70,291
70/30	29,273	34,971	67,048	28,287	34,512	65,481
80/80	28,093	39,767	63,429	36,678	41,601	61,100
90/10	29,965	36,686	58,857	26,311	38,199	55,784

Tabelle 8.15: Zweiphasige Optimierung: Ackley / Schwefel

anstatt in dieser Form des unbedingten Re-Annealings.
Bei allen Versuchen, insbesondere jedoch bei der realen Simulation von SCPNs, fällt auf, dass die Cache-Trefferrate sinkt, je mehr Simulationen in Phase 2 durchgeführt werden. Dies geschieht, obwohl sich beide Phasen einen gemeinsamen Cache teilen. Die Ergebnisse der zweiphasigen Optimierung realer SCPNs zeigen außerdem die Tendenz, dass eine gleichmäßige Verteilung der Gesamtanzahl von Simulationen auf zwei Phasen bessere Ergebnisse liefern. Dabei sind die Ergebnisse beim Basis-SCPN deutlich besser, als mit Simulated Annealing allein und bei der Simulation des Hausenergienetzes zeigt eine gleichmäßige Verteilung der Simulationsanzahl auf beide Phasen bessere Ergebnisse als bei den Benchmark-Funktionen.

Verhältnis Phase1/Phase2	Basis-SCPN			Hausenergiemodell		
	Abstand (%)	Abstand-Euklid (%)	Cache-ratio (%)	Abstand (%)	Abstand-Euklid (%)	Cache-ratio (%)
10/90	5,832	24,847	90,190	0,843	14,927	89,020
20/80	4,847	31,445	81,905	0,832	22,488	79,900
30/70	6,563	12,935	76,667	0,036	0,130	72,115
40/60	4,732	23,145	67,905	0,892	30,748	62,524
50/50	3,043	25,406	59,524	0,383	7,804	53,010
60/40	6,102	58,879	51,238	0,236	7,695	44,951
70/30	4,930	44,192	42,571	0,144	0,222	34,519
80/80	4,584	29,681	32,762	0,881	40,772	26,600
90/10	5,482	33,627	24,667	0,409	14,300	17,255

Tabelle 8.16: Zweiphasige Optimierung: Basis-SCPN vs. Hausenergienetz

8.3.5 Multiphasen-Optimierung

Die Multiphasen-Optimierung stellt eine Art Weiterentwicklung des bekannten Zweiphasen-Ansatzes dar, wobei jedoch andere Basisheuristiken verwendet werden können. Hinzu kommt, dass hier die Größe des Definitionsraumes und auch die Simulationsgenauigkeit zwischen den Phasen verändert wird. Dadurch wird ermöglicht, dass einfache Heuristiken wie Hill-Climbing durch Verkleinerung des Definitionsbereiches und Erhöhung der Simulationsgenaugikeit beschleunigt werden. Erste Ergebnisse, die diese These stützen, finden sich in [5].
Im Kern der durchgeführten Versuche steht die Frage, ob die Erhöhung der Phasenanzahl die Ergebnisqualität erhöht und inwiefern sich die Variation der Simulationsgenauigkeit auf die gesamte Optimierungszeit (CPU-Zeit) auswirkt. Dies ist bei Simulated Annealing aufgrund der bekannten Simulationsanzahl pro Optimierung nur von eingeschränktem Nutzen. Der Bedarf an CPU-Zeit wird für die Benchmark-Funktionen mit der Formel 4.2 (Seite 46) abgeschätzt.
Anders als bei der Zweiphasen-Optimierung wird nicht die jeweils beste Konfiguration für die entsprechende Benchmark-Funktion bzw. das Modell aus den vorangegangenen Kapiteln verwendet. Stattdessen wird für jede Heuristik eine Konfiguration gewählt und auf alle Probleme/Netze gleichermaßen angewendet. Die verwendeten Konfigurationen sind in Tabelle 8.17 aufgelistet.
Die besten Konfigurationen jeder einzelnen Heuristik können den Ergebnistabellen im Anhang entnommen werden, wobei die besten Ergebnisse pro Heuristik in den Tabellen markiert sind. Zur Beurteilung wurde dabei nur die Nähe des gefundenen zum globalen Optimum herangezogen.
Bei jedem Optimierungslauf wird die Genauigkeit sukzessive erhöht, bis zum Maximum in der letzten Phase. Dabei wird das geforderte Konfidenzintervall von 85 % - 99 % erhöht und der maximale relative Fehler von 15 % - 1 % verringert. Das typische Verhältnis von Simulationsgenauigkeit zu benötigter CPU-Zeit wird in Kapitel 4.2.1 dargestellt.
Von einer vollständigen Untersuchung aller Konfigurationsmöglichkeiten wurde abgesehen; dies kann Aufgabe für die Entwicklung sogenannter Hyper-Heuristiken sein, welche die optimale Konfiguration und Kombination verschiedener Heuristiken zur Optimierung eines bestimmten Problemtyps zu finden versuchen.
Die Auswertung der Versuche lässt deutliche Unterschiede zwischen Ergebnissen

Heuristik	Konfiguration
Hill-Climbing	**Strategie** = **Kontinuierlich in Nachbarschaft** $WRONG\_SOLUTIONS\_IN\_A\_ROW$ = 21 $WRONG\_SOLUTION\_PER\_DIRECTION$ = – **Nachbarschaft (%)** = 20 % **Quelle** Tabelle B.18
Simulated annealing	**Abkühlungsfunktion** = **Fast** **Parameterbestimmung** = **Standard** **Simulationsanzahl** = **100** **Quelle** Tabelle 8.9
Genetische Algorithmen (SBX)	**Populationsgröße** = **15** **Mutationswahrscheinlichkeit** = **30** **Quelle** Tabelle 8.13

Tabelle 8.17: Konfiguration der Basisheuristiken in der Multiphasen-Optimierung

der Optimierung von Benchmark-Funktionen und realen Simulationen erkennen. Die Ergebnisplots verdeutlichen den Zusammenhang zwischen Phasenanzahl und erreichter Ergebnisqualität, also Nähe zum tatsächlichen Optimum.
Bei Benchmark-Funktionen zeigt sich zum großen Teil eine Verbesserung der Ergebnisse mit erhöhter Phasenanzahl. Besonders bei Hill-Climbing ist dies beobachtbar. Die Variation der Simulationspräzision führt zum bemerkenswerten Ergebnis, dass die verbesserten Optimierungsergebnisse mit nur marginal vergrößertem Einsatz an CPU-Zeit erkauft werden müssen, obwohl die Gesamtzahl an Simulationen mit jeder Phase steigt. Insbesondere Hill-Climbing kann in diesem Sinne profitieren.
Genetische Algorithmen wie das getestete SBX-Verfahren liefern umso bessere Ergebnisse, je mehr Phasen verwendet werden, wobei jedoch auch Simulationszahl und CPU-Zeit-Bedarf stark steigen. Die Kombination aus SBX und mehrphasiger

Optimierung scheint insbesondere für Kostenfunktionen mit vielen lokalen Optima geeignet zu sein (Bild 8.23 u. 8.24).
Einzig Simulated Annealing scheint nicht vom Einsatz mehrerer Phasen zu profitieren. Hier zeigt sich, dass eine Erhöhung der Phasenanzahl teilweise keine Verbesserung oder sogar eine Verschlechterung mit sich bringt.
Anders stellt sich die Situation bei der Optimierung echter SCPNs dar. Wie schon bei der Bewertung anderer Heuristiken, sind die Optimierungsergebnisse weniger vielversprechend. Die Erhöhung der Phasenanzahl führt bei Simulated Annealing zur Verschlechterung der Ergebnisse. Bei anderen Basisheuristiken verbessert sich zwar das Ergebnis, jedoch scheint es noch unbekannte Einflussfaktoren zu geben. So führt die Erhöhung der Phasenanzahl für genetische Algorithmen beim Basis-SCPN zu einer stetigen Verbesserung der Ergebnisse, beim Hausenergienetz jedoch zur Verschlechterung. Eine eindeutige Empfehlung für künftige SCPN-Optimierungen lässt sich damit nicht geben (Bild 8.25 u. 8.26).

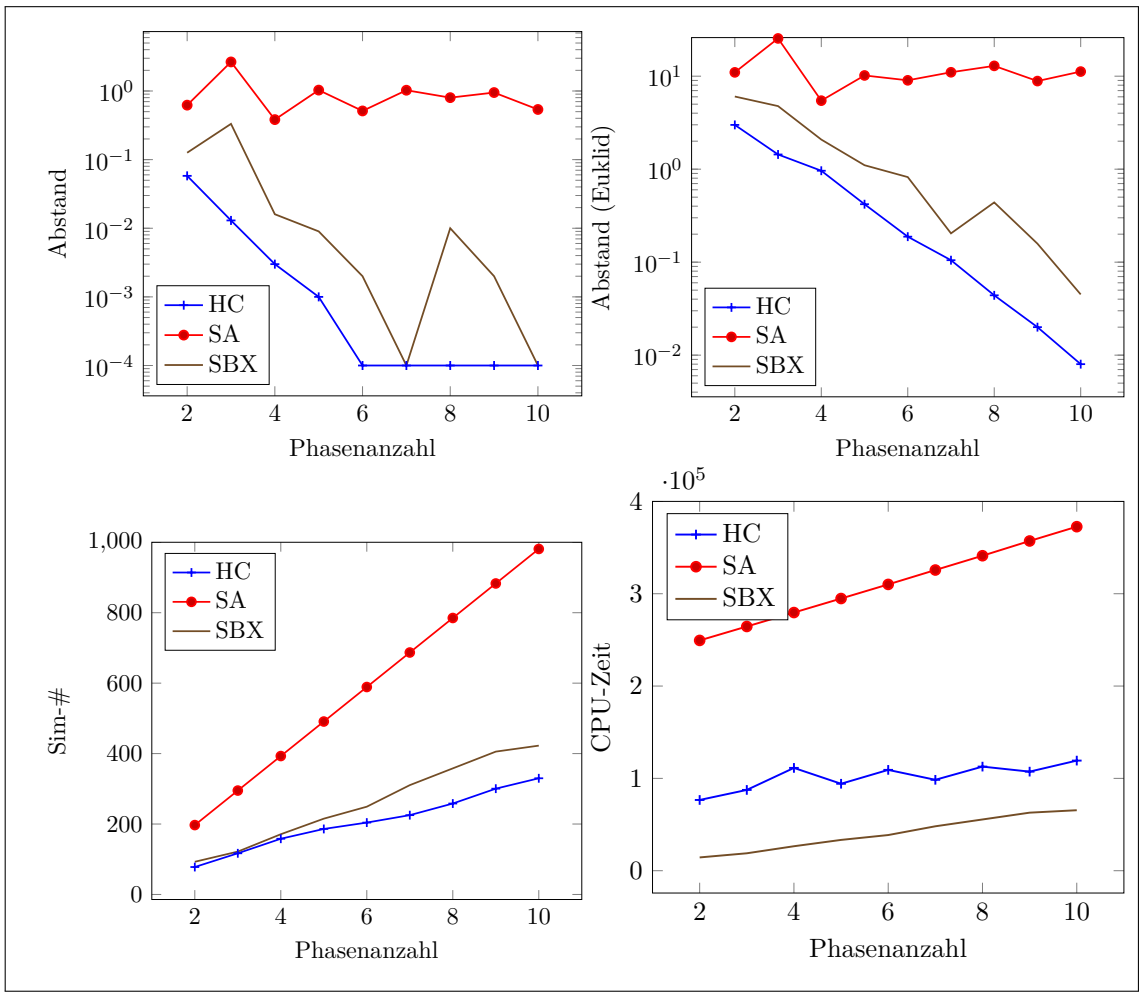


Abbildung 8.21: Multiphasenoptimierung der Benchmark-Funktion Sphere mit verschiedenen Basisheuristiken
Alle Messergebnisse finden sich in Tabelle B.23 im Anhang.

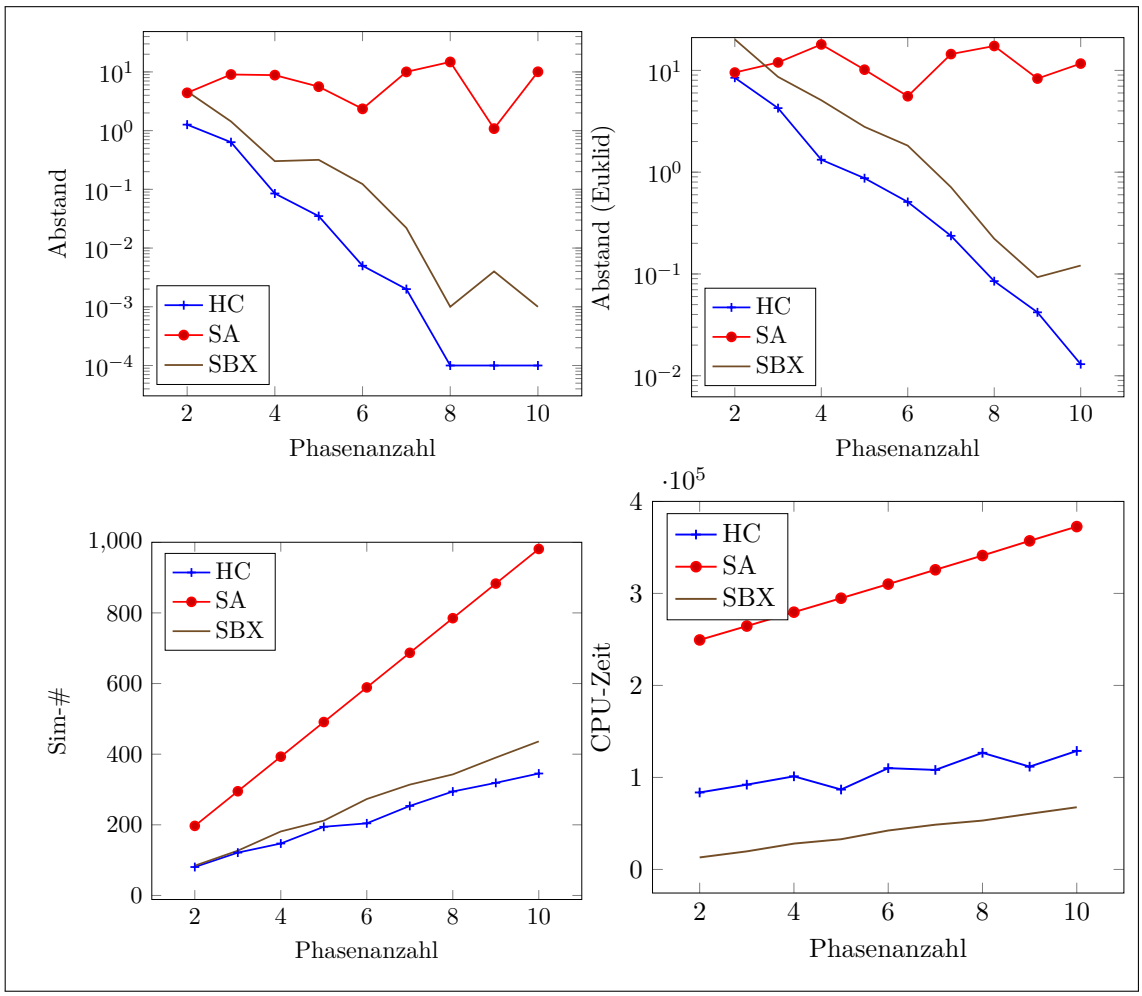


Abbildung 8.22: Multiphasenoptimierung der Benchmark-Funktion Matya mit verschiedenen Basisheuristiken
Alle Messergebnisse finden sich in Tabelle B.24 im Anhang.

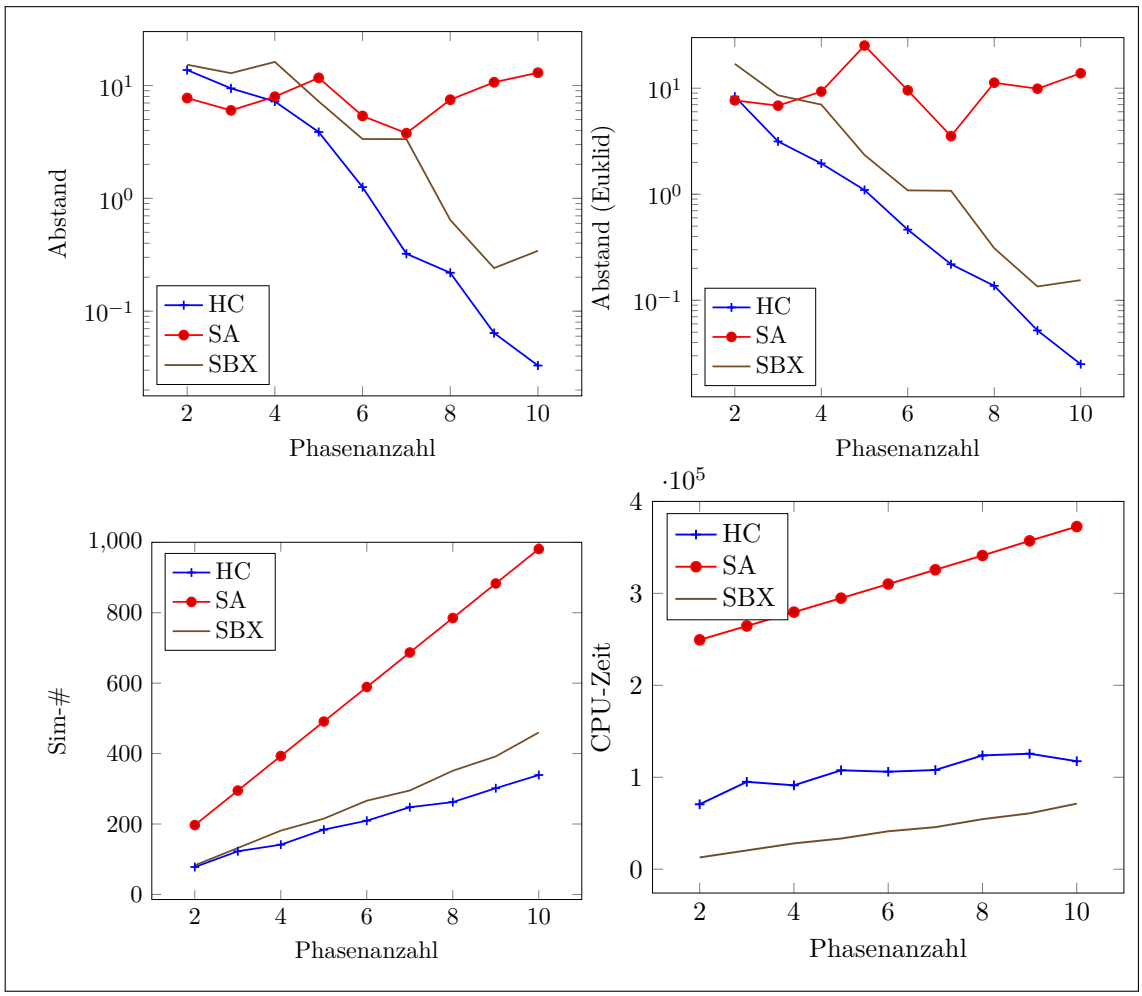


Abbildung 8.23: Multiphasenoptimierung der Benchmark-Funktion Ackley mit verschiedenen Basisheuristiken
Alle Messergebnisse finden sich in Tabelle B.25 im Anhang.

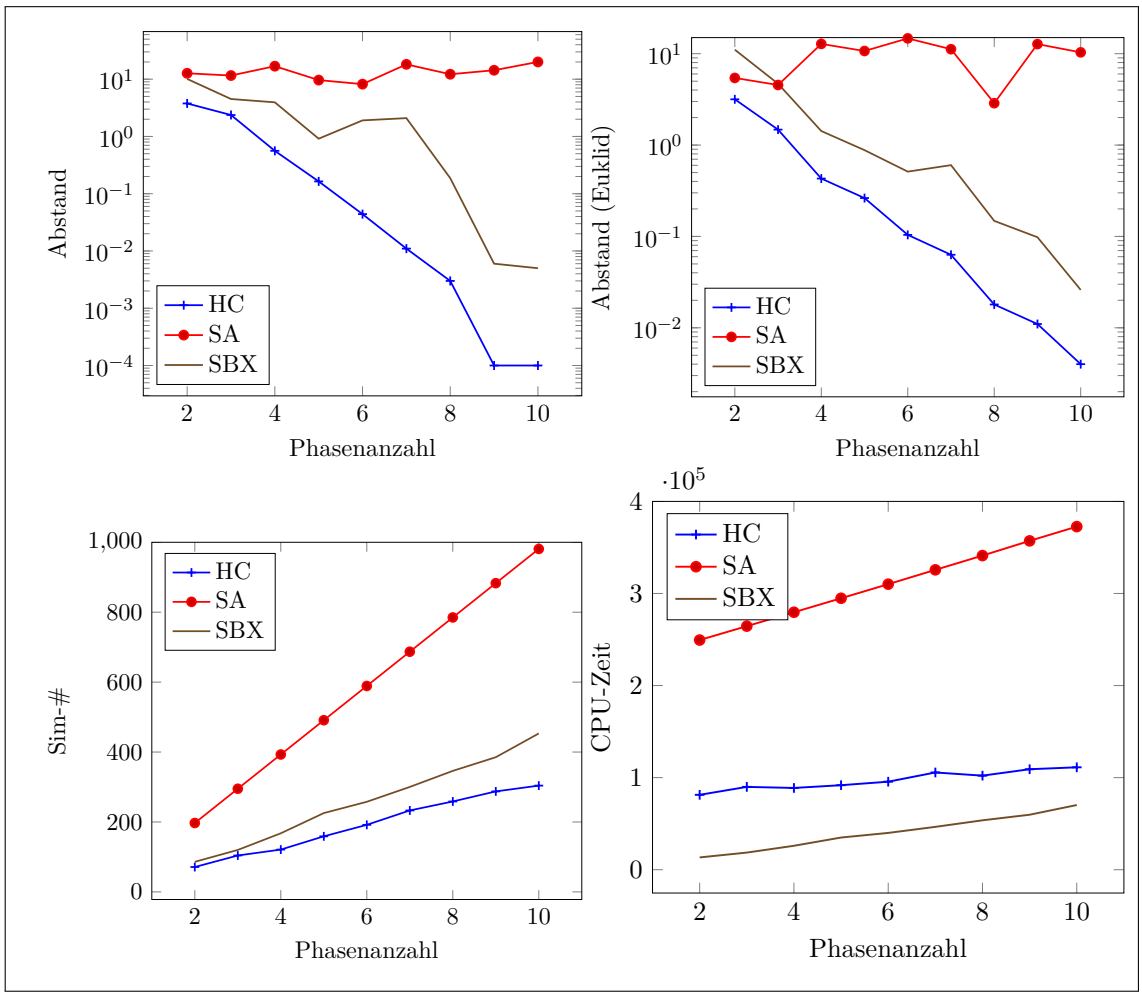


Abbildung 8.24: Multiphasenoptimierung der Benchmark-Funktion Schwefel mit verschiedenen Basisheuristiken
Alle Messergebnisse finden sich in Tabelle B.26 im Anhang.

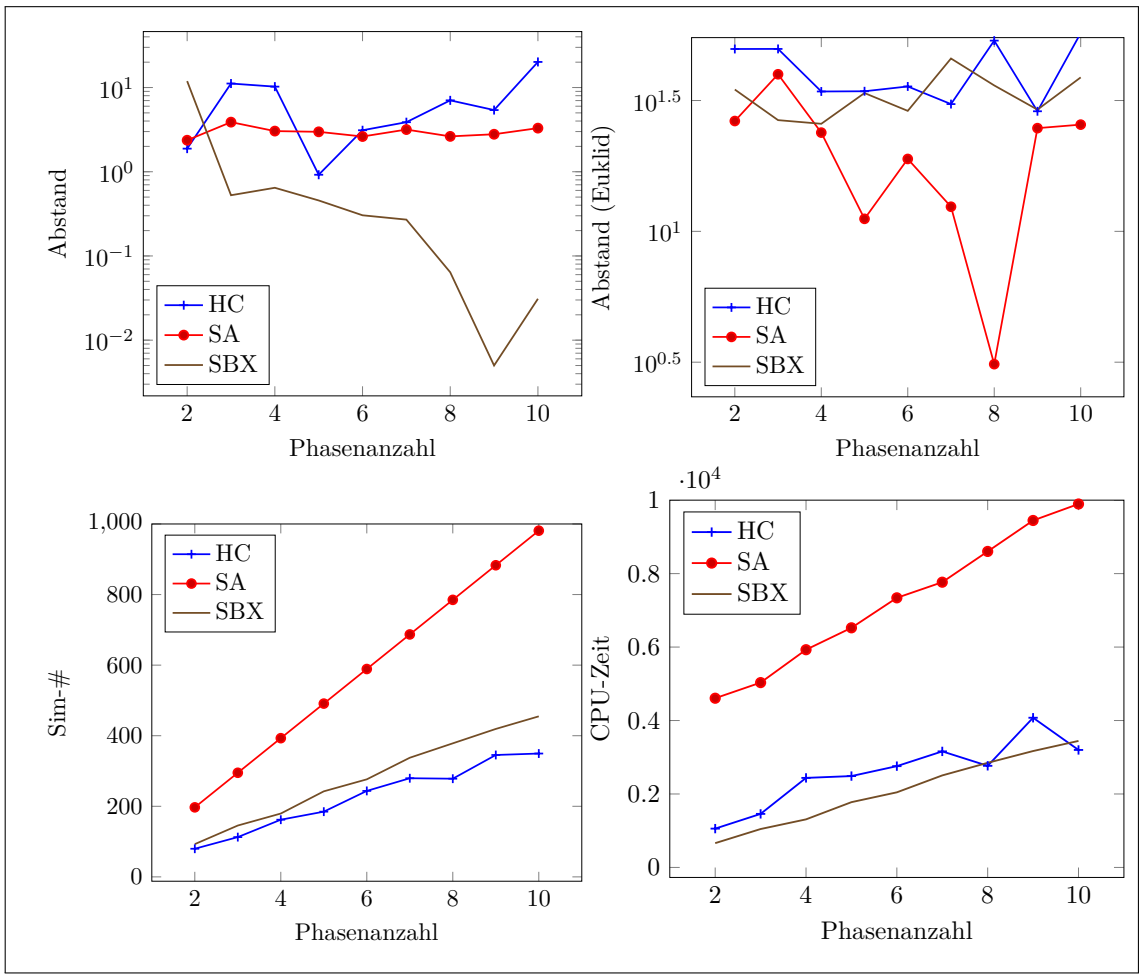


Abbildung 8.25: Multiphasenoptimierung des Basis-SCPN mit verschiedenen Basisheuristiken
Alle Messergebnisse finden sich in Tabelle B.27 im Anhang.

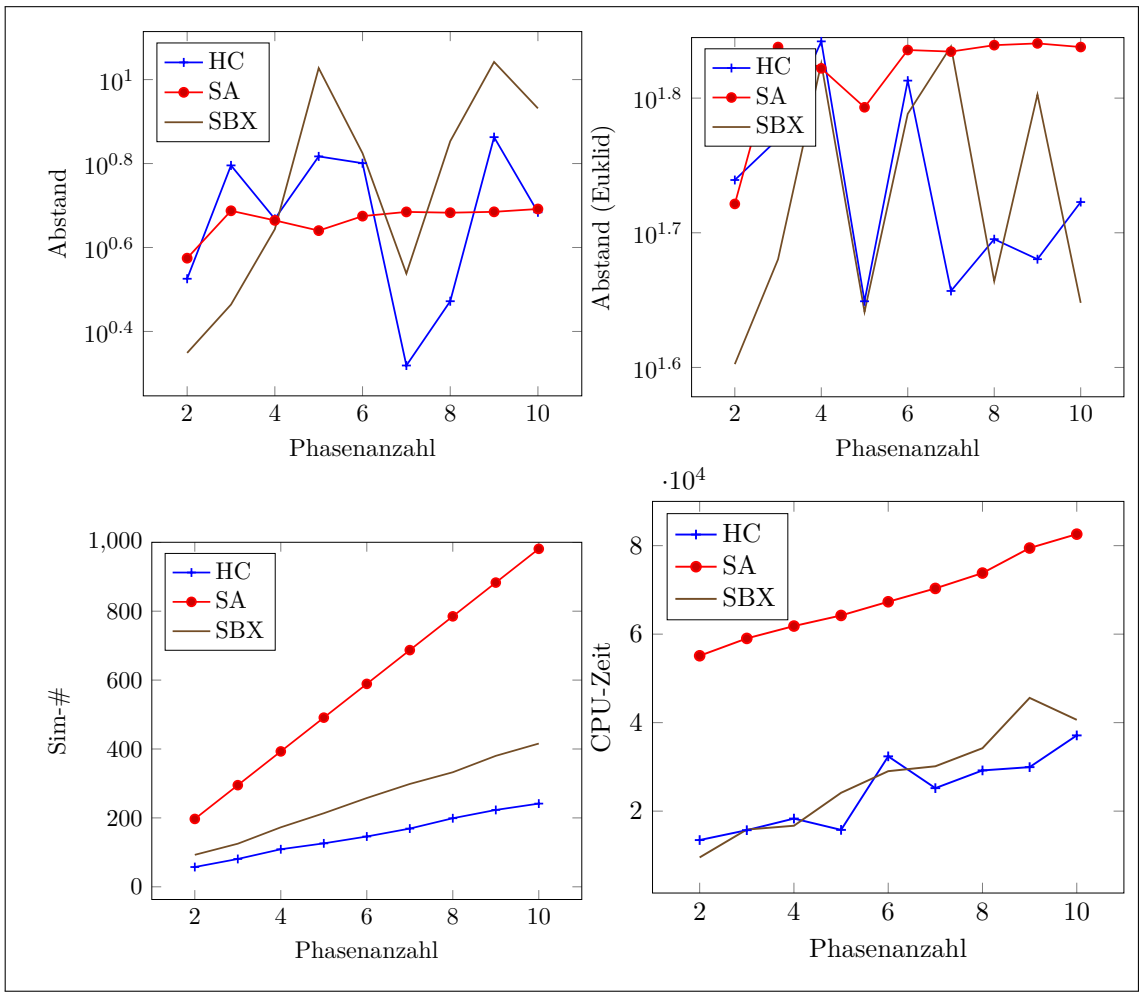


Abbildung 8.26: Multiphasenoptimierung des Hausenergiemodells mit verschiedenen Basisheuristiken
Alle Messergebnisse finden sich in Tabelle B.28 im Anhang.

8.4 Zusammenfassung der Bewertung

Ziel der Bewertung war es, die vorgestellten und umgesetzten Optimierungsheuristiken hinsichtlich ihrer Eignung für bestimmte Kostenfunktionen bzw. Wertelandschaften zu untersuchen. Hierzu wurden zunächst Bewertungskriterien definiert. Das wichtigste Kriterium ist dabei der relative Abstand des gefundenen vom tatsächlichen, globalen Optimum. Bei Benchmark-Funktionen ist das globale Optimum bekannt. Für die Untersuchung anhand echter SCPN-Simulationen wurde das Optimum experimentell bestimmt.
Zunächst wurde untersucht, welche Konfiguration der einzelnen Heuristiken bei welcher Art von Benchmark-Funktion die besten Ergebnisse liefert. Es wurde anschließend geprüft, inwieweit sich diese Erkenntnisse auf reale Simulationen von SCPNs übertragen lassen. Dazu wurden jeweils zwei Benchmark-Funktionen mit vielen und mit wenigen lokalen Optima herangezogen. Anschließend wurden die gleichen Heuristiken anhand von SCPNs getestet, deren Leistungsmaße bei Variation entsprechender Modellparameter augenscheinlich ähnliche Wertelandschaften liefern.
Die Ergebnisse lassen bei jeder Heuristik eine gewisse Tendenz der Auswirkungen von Konfigurationsänderungen erkennen. Der Leser kann die gewonnenen Erkenntnisse auf die Wahl der Heuristik für eigene Modelle und Optimierungsprobleme übertragen. Eine der Erkenntnisse dieser Arbeit ist jedoch, dass die Erkenntnisse aus der Optimierung anhand von Benchmark-Funktionen im Allgemeinen nicht direkt auf die Optimierung von SCPNs übertragbar sind. Es bleibt zu untersuchen, woher diese Unterschiede in der Ergebnisqualität kommen. Eine Vermutung ist, dass die Verteilung der Kostenfunktionswerte bei realer Simulation weitaus unsteter ist, als bei Benchmark-Funktionen. Dies konnte aber bis dato trotz hunderttausendfacher Simulation im Rahmen der Arbeit nicht bestätigt werden.
Neben der Vorstellung des Optimierungswerkzeuges 2014 wurden in vorausgehenden Arbeiten bereits erste Ergebnisse und Hinweise auf den Nutzwert der mehrphasigen Optimierung veröffentlicht. Der Nachweis der Effizienz erfolgte anhand erster Benchmarkfunktionen sowie an konkreten Simulationen von SCPNs bereits 2015 [5, 7, 6]. Die Ergebnisse der Bewertung in dieser Arbeit erweitern Erkenntnisse der genannten Veröffentlichungen und lassen sich in folgenden Punkten zusammenfassen:

- Trotz ähnlicher Wertelandschaft sind die Konfigurationen der Heuristiken zwischen Benchmark-Funktionen und realer Simulation nicht ohne weiteres über-

tragbar bzw. liefern unterschiedliche Ergebnisse.

- Simulated Annealing profitiert nicht von mehrfacher Anwendung auf das gleiche Problem, egal ob Benchmark-Funktion oder realer Simulation.
- Wenn die vermutete Kostenfunktion viele lokale Optima besitzt, eignen sich Heuristiken mit zufälligen Einflussgrößen, wie genetische Algorithmen, am besten für die Optimierung.
- Die Integration in den Multiphasen-Ansatz führt mit Erhöhung der Phasenanzahl zu steigender Ergebnisqualität für Hill-Climbing und genetische Algorithmen.

Zusammenfassend lässt sich sagen, dass die Qualität der gefundenen scheinbaren Optima und die Laufzeit der jeweiligen Optimierung stark von der Konfiguration der verwendeten Heuristik abhängen. Die Zahl der möglichen Variationen von Basisheuristiken ist so groß, dass nur eine kleine Auswahl untersucht werden konnte. Eine Kombination dieser Heuristiken in Verfahren wie dem vorgestellten Multiphasen-Ansatz bietet schier unendlich viele Anpassungsmöglichkeiten. Untersuchungen in diese Richtung sind in der Literatur unter dem Begriff **Hyperheuristik** zu finden. Dies bezeichnet die heuristische Suche nach einer optimalen Konfiguration und Kombination diverser Heuristiken [12].
Tabelle 8.17 zeigt die verwendeten Heuristiken während der Multiphasen-Optimierung. Diese Konfigurationen können als Ausgangspunkt für die Wahl und Konfiguration einer Heuristik bei künftigen Untersuchungen herangezogen werden. Zusätzlich finden sich im Anhang Ergebnistabellen mit allen untersuchten Heuristik-Konfigurationen.

9 Zusammenfassung und Ausblick

In dieser Arbeit wurde zunächst eine allgemeine Einführung in die Modellierung mit Petri-Netzen und das zentrale Problem, den Zeitbedarf simulationsbasierter Optimierungen zu reduzieren, gegeben. Anschließend werden bekannte und neue Verfahren zur effizienten simulationsbasierten Optimierung von SCPNs vorgestellt. Zur Untersuchung der Verfahren anhand verschiedener Kostenfunktionen wurde ein spezielles SW-Werkzeug entworfen und umgesetzt.
Die Ergebnisse der Arbeit geben Hinweise zur Auswahl geeigneter Optimierungsverfahren für spezielle Problemklassen, auch für die simulationsbasierte Optimierung anderer Modelle als SCPNs.

9.1 Ergebnisse der Arbeit

Die Ergebnisse der Arbeit lassen sich in zwei Bereiche einteilen: Einerseits sind die Ergebnisse der Untersuchung bekannter und neuer Verfahren zur simulationsbasierten Optimierung sowie ein neues, genauigkeitsvariables Optimierungsverfahren zu nennen und andererseits die Nebenprodukte in Form von Werkzeugen, die zur Durchführung der Untersuchungen geschaffen wurden.
Die Werkzeuge bestehen aus einer lokalen Optimierungssteuerung und einem Server zur Verteilung von Simulationsaufgaben und erlauben künftigen Nutzern:

- die schnelle Analyse des Definitionsraumes für parametrisierte SCPNs
- den Test verschiedenster Optimierungsheuristiken und deren Konfiguration anhand bestehender Benchmark-Funktionen
- die Durchführung lokal und verteilt simulierter Optimierungen von SCPNs auf Basis der implementierten Heuristiken
- die grafische Darstellung von Optimierungsabläufen

- die Erweiterung der Werkzeuge um weitere Heuristiken, Benchmark-Funktionen und Simulationssysteme

Diese Werkzeug steht nunmehr für weitere Untersuchungen zur Verfügung.
Die untersuchten, bekannten Verfahren zur effizienten simulationsbasierten Optimierung machen sich hauptsächlich folgende Prinzipien zu Nutze: So wird die verwendete Heuristik an das jeweilige Problem angepasst, um die Anzahl der benötigten Simulationsläufe zu reduzieren und zur Verkürzung der Simulationszeit werden Modelle vereinfacht sowie die Simulationsgenauigkeit reduziert.
Das neu vorgestellte Verfahren der Multiphasen-Optimierung mit schrittweiser Variation der Simulationsgenauigkeit kombiniert dabei verschiedene bekannte Strategien.
Die Ergebnisse der durchgeführten Versuche im Rahmen dieser Arbeit sind die nachfolgenden Erkenntnisse:

- Verfahren, die bei Benchmark-Funktionen sehr effizient arbeiten, können dennoch bei realer Simulation deutlich schlechtere Ergebnisse erzielen, auch wenn die Wertelandschaften Ähnlichkeit vermuten lassen.

- Die mehrfache Durchführung einer Optimierung (wobei das jeweils gefundene Optimum den Startwert für die nächste Optimierung darstellt) verbessert bei einigen Heuristiken die Ergebnisqualität, jedoch nicht bei allen. So führte die mehrfache Anwendung von Simulated Annealing zu keiner Verbesserung. Daher ist dieses Verfahren prinzipiell nicht für die Multiphasen-Optimierung geeignet.

- Heuristiken mit zufälligen Einflussgrößen bei der Bestimmung der Parametersätze eignen sich am Besten für die Optimierung von Modellen, deren vermutete Kostenfunktion viele lokale Optima hat. Dies gilt insbesondere, wenn das gefundene Optimum auch auf Grund der Entfernung zum tatsächlich existierenden Optimum im Definitionsraum bewertet wird.

- Die Integration in den Multiphasen-Ansatz führt mit Erhöhung der Phasenanzahl zu steigender Ergebnisqualität für Hill-Climbing und genetische Algorithmen.

9.2 Ausblick

Die Ergebnisse der Versuche während der Bewertung deuten darauf hin, dass auf dem Gebiet der Heuristiken zur simulationsbasierten Optimierung noch viele Faktoren zu untersuchen sind. Insbesondere die Übertragung von Benchmark-Ergebnissen auf reale SCPN-Simulationen ist dabei von großer Wichtigkeit. Mit dem aktuellen Stand des SW-Werkzeuges lassen sich bereits viele weitere Faktoren untersuchen, die über den Umfang der Arbeit hinausgehen.
Aufbauend auf der vorliegenden Arbeit erscheinen sowohl weitere Analysen mit dem existierenden SW-Werkzeug, als auch dessen Weiterentwicklung vielversprechend.
Die Übertragung der Ergebnisse der Optimierungen auf Basis von Benchmark-Funktionen auf reale Simulationen kann und sollte weiter untersucht werden. Nicht nur für die Simulation von SCPNs, sondern auch bei späterem Einsatz mit anderen Modellierungs- und Simulationswerkzeugen wären die Ergebnisse nützlich.
Die Weiterentwicklung der Software selbst lässt sich in 6 Teilgebiete aufteilen:

Optimierungsheuristiken Es ist möglich, die Software um weitere Optimierungsheuristiken zu erweitern. Die entsprechenden Schnittstellen sind dokumentiert. Einige Heuristiken, die nicht im Rahmen dieser Arbeit betrachtet wurde, sind bereits implementiert. So z.B. Artificial Bee Colony, alternative Kreuzungsvarianten für genetische Algorithmen und Mean-Variance-Mapping-Optimization [50]. Eine umfangreiche Prüfung der Implementierung steht jedoch noch aus.

Abgesehen davon ist vorstellbar, die existierenden Heuristiken zu erweitern, um Präzisionssteuerung und Schrittweitenanpassung auch ohne Multiphasen-Optimierung nutzen zu können.

Wertelandschaften / Benchmark-Funktionen Ähnlich der Integration anderer Optimierungsheuristiken lässt sich die Software um weitere Benchmark-Funktionen erweitern. Dabei gilt es, jeweils den Definitionsbereich zu beachten und für eine umfangreiche Analyse die Berechnung des globalen Optimums zu implementieren.

Usability Während der Entwicklung des SW-Tools zur Analyse der Heuristiken war die einfache Benutzbarkeit stets eine wichtige Anforderung. Menschen, die mit

TimeNET vertraut sind, sollten sich relativ schnell in die Benutzung einarbeiten können. Dies wird unter anderem durch Zustandsabhängiges Deaktivieren von GUI-Elementen unterstützt. Nutzereingaben, die zu Fehlern führen würden, werden zum großen Teil unterbunden. Des Weiteren werden alle Programmeinstellungen und Optimierungskonfigurationen im Nutzerordner persistiert: ein erwartetes Standardverhalten der meisten aktuellen SW-Pakete.

Dennoch sind auch hier diverse Verbesserungen denkbar. So ließen sich ebenfalls die netzabhängigen Konfigurationsparameter automatisch persistieren, um eine rasche Wiederaufnahme der Arbeit nach Unterbrechungen zu erleichtern. Einige Eingaben werden bislang noch nicht auf Fehlerfreiheit geprüft, was zu Problemen führen kann. Außerdem ist vorstellbar, die Installation zu vereinfachen. Derzeit muss der Nutzer noch selber die Verzeichnisse für TimeNET und R auswählen. Trotz Hilfestellung bei der Pfadauswahl für R, ist damit manch ein Nutzer überfordert. Eine Automatisierung bietet sich an dieser Stelle an.

Refactoring Zwar wurde während der Entwicklung auf leichte Lesbarkeit und Wartbarkeit des Quelltextes geachtet, aber einige Schwächen in Struktur und Nomenklatur sind dennoch zu enthalten. Von strukturellen Änderungen gegen Ende der Heuristikbewertung wurde zugunsten der SW-Stabilität abgesehen.

Es bietet sich an, derartige Verbesserungen in Verbindung mit einem unabhängigen Code-Review mit Schwerpunkt auf künftige Erweiterbarkeit durchzuführen.

Client-Server-Infrastruktur Die verteilte Simulation von SCPNs ist ein nützliches Hilfsmittel bei der Optimierung, vor allem aber zur groben Analyse des Definitionsbereichs. Zusammen mit der graphischen Darstellung der Simulationsergebnisse ermöglicht es die rasche Einschätzung der Komplexität des Optimierungsproblems. Auch wenn die aktuelle Implementierung zuverlässig arbeitet, so wurden einige wichtige Aspekte bei der Entwicklung ausgeklammert.

Sicherheit Es wird keine verschlüsselte Verbindung genutzt. Es gibt keine Zugangsbeschränkungen, weder für Master-Rechner, noch für die Clients. Nichtöffentliche Modelle sollten demnach aktuell nur im lokalen Netzwerk verteilt werden.

Fairness Auch wenn der Server derzeit durchaus die Simulationsdateien diverser Master-Rechner verwalten kann, so erfolgt keine faire Verteilung der Client-Rechner. Dies hat zur Folge, dass ein einzelner Teilnehmer alle Rechenressourcen vereinnahmen kann.

Informationen Der Server stellt eine einfache Website mit Statusinformationen zur Verfügung, die noch deutlich erweitert und benutzerfreundlicher gestaltet werden kann.

Headless Simulation Die Clients sind derzeit in Java implementiert und benötigen sowohl für sich selbst als auch für die notwendige TimeNET-Installation eine graphische Benutzeroberfläche. Mit einer Kommandozeilenapplikation ließen sich Ressourcen und Einrichtungsaufwand einsparen.

Simulationswerkzeuge Die Integration künftiger Heuristiken und Benchmark-Funktionen wurde von Anfang an bedacht und entsprechend umgesetzt. Jedoch wurde die Anbindung an TimeNET relativ fix in der Programmstruktur verankert. Ein Nebenziel eines evtl. erfolgenden Refactorings wird die Modularisierung zur erleichterten Integration alternativer Simulationswerkzeuge sein. In vielen Forschungsprojekten kommen Tools wie Matlab Simulink oder ML-Designer zum Einsatz. Mit einer Anbindung dieser Werkzeuge könnten viele weitere Forscher und Projekte von den Ergebnissen dieser Arbeit, wie den umgesetzten Optimierungsheuristiken oder der verteilten Simulation, profitieren.

Es sind noch viele weitere Verbesserungen und Erweiterungen des Werkzeugs denkbar, je nach persönlichen Anforderungen. Aus diesem und anderen Gründen steht der Quelltext der Software jedem zum Download und zur Weiterentwicklung bereit.

9.3 Thesen

- In einem mehrstufigen Optimierungsprozess kann zu Beginn die Simulationsgenauigkeit reduziert werden, ohne dass die Qualität der Ergebnisse reduziert wird.
- Der Zeitbedarf für SCPN-Simulationen ist von internen und externen Faktoren abhängig, dies muss vor der simulationsbasierten Optimierung überprüft werden.
- Die Erkenntnisse zur Konfiguration von Heuristiken sind nur eingeschränkt von Benchmark-Funktionen auf SCPN-Simulationen übertragbar. Die Gründe hierfür sind noch nicht abschließend geklärt.
- Die Ergebnisqualität einer Optimierungsheuristik ist stark vom simulierten Modell bzw. der erwarteten Wertelandschaft abhängig.
- Die wiederholte Optimierung mit der gleichen Heuristik, wobei das gefundene Optimum als Startwert für die nachfolgende Optimierung genutzt wird, liefert für viele Heuristiken bessere Ergebnisse, als einmalige Optimierung.
- Die wiederholte Optimierung mit Simulated Annealing (bei gleicher Konfiguration) liefert keine besseren Ergebnisse als die einmalige Optimierung.
- Die Optimierungszeit kann bei gleichbleibender Ergebnisqualität (Wahrscheinlichkeit, das globale Optimum gefunden zu haben) reduziert werden, indem die Simulationsgenauigkeit zu Beginn reduziert und dann schrittweise auf das geforderte Niveau angehoben wird.
- Heuristiken, deren Bestimmung von Parametersätzen zufallsbehaftet ist, eignen sich besonders gut zur Optimierung von Problemen, deren Kostenfunktion viele lokale Optima hat.
- Populationsbasierte Heuristiken profitieren ohne weitere Anpassungen von verteilter/paralleler Simulation.

Anhang A

Literaturverzeichnis

[1] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: An experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, Nov. 2002.

[2] F. Bause and P. Buchholz. Aggregation and disaggregation in product form queueing petri nets. In *Petri Nets and Performance Models, 1997., Proceedings of the Seventh International Workshop on*, pages 16 –25, jun 1997.

[3] P. Bazan. Approximate analysis of stochastic models by self-correcting aggregation. *Quantitative Evaluation of Systems*, 2005.

[4] J. Biel, E. Macias, and M. Perez de la Parte. Simulation-based optimization for the design of discrete event systems modeled by parametric Petri nets. In *Computer Modeling and Simulation (EMS), 2011 Fifth UKSim European Symposium on*, pages 150–155, 2011.

[5] C. Bodenstein, T. Dietrich, and A. Zimmermann. Computationally efficient multiphase heuristics for simulation-based optimization. In *Proceedings of the 5th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*, pages 95–100, 2015.

[6] C. Bodenstein and A. Zimmermann. TimeNET optimization environment. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, 2014.

[7] C. Bodenstein and A. Zimmermann. Extending design space optimization heuristics for use with stochastic colored petri nets. In *2015 IEEE International Systems Conference (IEEE SysCon 2015)*, Vancouver, Canada, Apr. 2015.

[8] D. Brand. Hill climbing with reduced search space (logic optimization). In *Computer-Aided Design, 1988. ICCAD-88. Digest of Technical Papers., IEEE International Conference on*, pages 294–297, Nov 1988.

[9] P. Buchholz. Aggregation and reduction techniques for hierarchical gcspns. In *Petri Nets and Performance Models, 1993. Proceedings., 5th International Workshop on*, pages 216 –225, oct 1993.

[10] P. Buchholz. Hierarchies in colored gspns. In M. Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 106–125. Springer Berlin Heidelberg, 1993.

[11] P. Buchholz. Hierarchical high level petri nets for complex system analysis. In R. Valette, editor, *Application and Theory of Petri Nets 1994*, volume 815 of *Lecture Notes in Computer Science*, pages 119–138. Springer Berlin Heidelberg, 1994.

[12] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. Hyper-heuristics. *J Oper Res Soc*, 64(12):1695–1724, Dec 2013.

[13] Y. Carson and A. Maria. Simulation optimization: Methods and applications. In *Proceedings of the 29th Conference on Winter Simulation*, WSC '97, pages 118–126, 1997.

[14] P. Civicioglu and E. Besdok. A conceptual comparison of the cuckoo-search, particle swarm optimization, differential evolution and artificial bee colony algorithms. *Artificial Intelligence Review*, 39(4):315–346, Apr 2013.

[15] M. D. V. M. A. Colorni. The ant system: Optimization by a colony of cooperating agents. 2011.

[16] K. Deb and R. B. Agrawal. Simulated Binary Crossover for Continuous Search Space. *Complex Systems*, 9:115–148, 1995.

[17] K. Deb, A. Anand, and D. Joshi. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evol. Comput.*, 10(4):371–395, Dec. 2002.

[18] C. Dutheillet and S. Haddad. Aggregation of states in colored stochastic Petri nets: application to a multiprocessor architecture. In *Petri Nets and Performance Models, 1989. PNPM89., Proceedings of the Third International Workshop on*, pages 40–49, 1989.

[19] R. Fehling. A concept of hierarchical petri nets with building blocks. In *Papers from the 12th International Conference on Applications and Theory of Petri Nets: Advances in Petri Nets 1993*, pages 148–168, London, UK, UK, 1993. Springer-Verlag.

[20] M. Feldmann. *Naturanaloge Verfahren: Metaheuristiken zur Reihenfolgeplanung*. Schriften zur quantitativen Betriebswirtschaftslehre. Deutscher Universitätsverlag, 1999.

[21] J. Freiheit and A. Heindl. Novel formulae for GSPN aggregation. In *Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002. MASCOTS 2002. Proceedings. 10th IEEE International Symposium on*, pages 209–216, 2002.

[22] J. Freiheit and A. Zimmermann. Extending a response time approximation technique to colored stochastic petri nets, 1998.

[23] M. C. Fu. Optimization via simulation: a review. *Annals of Operations Research*, pages 199–248, 1994.

[24] M. C. Fu. Optimization for simulation: Theory vs. practice. *INFORMS Journal on Computing*, 2002.

[25] M. C. Fu, F. W. Glover, and J. April. Simulation optimization: a review, new developments, and applications. In *Proceedings of the Winter Simulation Conference, 2005.*, pages 13 pp.–, Dec 2005.

[26] H. Ghasemieh, A. Remke, and B. R. Haverkort. Hybrid petri nets with multiple stochastic transition firings. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, VALUETOOLS '14, pages 217–224, ICST, Brussels, Belgium, Belgium, 2014. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[27] The Hague, Netherlands. *Two Heuristics for the Improvement of a Two-Phase Optimization Method for Manufacturing Systems*, Oct. 2004.

[28] J. Hu, M. C. Fu, and S. I. Marcus. A model reference adaptive search method for global optimization. *2007 Oper. Res*, 55:549–568, 2008.

[29] L. Ingber. Very fast simulated re-annealing. *Mathematical and Computer Modelling*, 12(8):967 – 973, 1989.

[30] K. Jensen. Coloured petri nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties*, volume 254 of *Lecture Notes in Computer Science*, pages 248–299. Springer Berlin Heidelberg, 1987.

[31] H. Jungnitz and A. Desrochers. Flow equivalent nets for the performance analysis of generalized stochastic Petri nets. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 122–127, 1991.

[32] C. Kelling. A framework for rare event simulation of stochastic petri nets using "restart". In *Simulation Conference, 1996. Proceedings. Winter*, pages 317–324, 1996.

[33] F. Künne. Analyse und visualisierung des laufzeitverhaltens stochastischer simulationen. Master's thesis, TU Ilmenau, 2017.

[34] J.-I. Latorre and E. Jiménez. Simulation-based optimization of discrete event systems with alternative structural configurations using distributed computation and the petri net paradigm. *SIMULATION*, 2013.

[35] A. C. Lavista. Rare event algorithms analysis and simulation. Master's thesis, TU-Ilmenau, 12 2014.

[36] L. Liberti. Introduction to global optimization, 2008.

[37] N. Lilith, J. Billington, and J. Freiheit. Approximate closed-form aggregation of a fork-join structure in generalised stochastic petri nets. In *valuetools '06: Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*. ACM, Oct. 2006.

[38] R. A. Mahale and P. S. D. Chavan. A survey: Evolutionary and swarm based bio-inspired optimization algorithms, 2012.

[39] F. Mansourinia, A. Aghaie, and A. Haddad. A novel approach of classification in simulation optimization methods. In *5thSASTech 2011*. Khavaran Higher-education Institute, Mashhad, Iran, 2011.

[40] T. Marechal, A. Smith, V. Ustun, J. Smith, and A. Lefeber. Optimizing a physical security configuration using a highly detailed simulation model. In *Handbook of military industrial engineering*, chapter 2-1/17. CRC Press, 2009.

[41] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, June 1953.

[42] S. D. Müller. *Bio-inspired optimization algorithms for engineering applications*. PhD thesis, ETH Zürich, 2002.

[43] H. Müller-Merbach. Heuristics and their design: a survey. *European Journal of Operational Research*, 8(1):1 – 23, 1981.

[44] B. L. Nelson, J. Swann, D. Goldsman, and W. Song. Simple procedures for selecting the best simulated system when the number of alternatives is large. *Operations Research*, 49:950–963, 1999.

[45] T. Osogami and T. Itoko. Finding probably better system configurations quickly. *SIGMETRICS Perform. Eval. Rev.*, 34(1):264–275, June 2006.

[46] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Universität Hamburg, 1962.

[47] G. C. Pflug. *Optimization of Stochastic Models*. The Interface Between Simulation and Optimization. Springer, Sept. 1996.

[48] A. Rajasekhar, A. Abraham, and M. Pant. Levy mutated artificial bee colony algorithm for global optimization.

[49] A. Rajasekhar, A. Abraham, and M. Pant. Levy mutated artificial bee colony algorithm for global optimization. In *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on*, pages 655–662, Oct 2011.

[50] A. Seidel. Heuristik-analyse für petri-netz-optimierungen in timenet. Master's thesis, TU Ilmenau, 1 2015.

[51] Y. Shi. *Rare Events in Stochastic Systems*. PhD thesis, Columbia University, 2013.

[52] D. E. Smith. An 'optimizer' for use in computer simulation: Background and design concepts,. 1970.

[53] D. E. Smith. Requirements of an "optimizer" for computer simulations. *Naval Research Logistics Quarterly*, 20(1):161–179, 1973.

[54] O. Spaniol and S. Hoff. *Ereignisorientierte Simulation: Konzepte und Systemrealisierung*. Number ISBN 3-8266-0122-X. Thomson Publishing International, 1995.

[55] A. Stevenson. *Oxford Dictionary of English*. Oxford Dictionary of English. OUP Oxford, 2010.

[56] J. R. Swisher, S. H. Jacobson, and E. Yücesan. Discrete-event simulation optimization using ranking, selection, and multiple comparison procedures: A survey. *ACM Trans. Model. Comput. Simul.*, 13(2):134–154, Apr. 2003.

[57] M. Syrjakov and H. Szczerbicka. Combination of direct global and local optimization methods. In *Evolutionary Computation, 1995., IEEE International Conference on*, volume 1, pages 326–, Nov 1995.

[58] M. Syrjakow. http://goethe.ira.uka.de/ syrjakow/remo.html, 1999.

[59] M. Syrjakow, E. Syrjakow, and H. Szczerbicka. Tool support for performance modeling and optimization. *International Journal of Enterprise Information Systems*, 2005.

[60] H. Szu and R. Hartley. Fast simulated annealing. *Physics Letters A*, 122(3):157 – 162, 1987.

[61] S. Taylor, A. Fatin, and T. Delaitre. Estimating the benefit of the parallelisation of discrete event simulation. In *Simulation Conference Proceedings, 1995. Winter*, pages 674–681, Dec 1995.

[62] TU-Ilmenau, http://www2.tu-ilmenau.de/sse_file/timenet/Manual-HTML4/UserManual.html. *TimeNET 4.0 Manual*, 2017.

[63] M. Vakil-Baghmisheh and A. Navarbaf. A modified very fast simulated annealing algorithm. In *Telecommunications, 2008. IST 2008. International Symposium on*, pages 61–66, Aug 2008.

[64] S. K. C. D. G. M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, Mai 1983.

[65] Y. V. G. Venayagamoorthy. Particle swarm optimization: Basic concepts, variants and applications in power systems. 2008.

[66] A. Zimmermann. Applied RESTART Estimation of General Reward Measures. pages 196–204. RESIM, 2006.

[67] A. Zimmermann. *Stochastic Discrete Event Systems - Modeling, Evaluation, Applications*. Springer-Verlag New York Incorporated, Nov. 2007.

[68] A. Zimmermann. Restart simulation of colored stochastic petri nets. In *Proc. 7th Int. Workshop on Rare Event Simulation (RESIM 2008)*, pages 143–152, 2008.

[69] A. Zimmermann and C. Bodenstein. Towards accuracy-adaptive simulation for efficient design-space optimization. In *2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 1230 –1237, Oct. 2011.

[70] A. Zimmermann and C. Bodenstein. Towards Efficient Design Optimization for Embedded Systems. In *MBEES 2011*, 2011.

[71] A. Zimmermann, T. Hotz, and A. C. Lavista. A hybrid multi-trajectory simulation algorithm for the performance evaluation of stochastic petri nets. In *Quantitative Evaluation of Systems*, 2017.

[72] A. Zimmermann, D. Rodriguez, and M. Silva. Modelling and optimisation of manufacturing systems: Petri nets and simulated annealing. In *1999 European Control Conference (ECC)*, pages 3926–3932, Aug 1999.

[73] A. Zimmermann, D. Rodriguez, and M. Silva. A two-phase optimisation method for Petri net models of manufacturing systems. *Journal of Intelligent Manufacturing*, 12(5/6):409–420, Oct. 2001. Special issue "Global Optimization Meta-Heuristics for Industrial Systems Design and Management".

[74] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms - a comparative case study. pages 292–301. Springer, 1998.

[75] R. Zurawski. Petri net models, functional abstractions, and reduction techniques: applications to the design of automated manufacturing systems. *Industrial Electronics, IEEE Transactions on*, 52(2):595 – 609, april 2005.

Anhang B

Simulationsergebnisse

WRONG_SOLUT-IONS_IN_A_ROW	*WRONG_SOLUT-IONS_PER_DIR*	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2		2,280	25,062	1199,12	185863,6	0,0
6		2,147	23,435	1247,64	193384,2	27,7
11	1	1,936	21,104	1766,24	273767,2	27,7
16		2,711	27,212	857,12	132853,6	51,1
21		1,461	18,179	1441,68	223460,4	42,2
2		2,810	27,431	485,92	75317,6	0,0
6		2,499	24,703	2247,04	348291,2	0,0
11	5	1,215	17,034	4535,72	703036,6	1,9
16		1,397	17,649	2504,16	388144,8	13,3
21		1,359	17,535	2680,48	415474,4	21,0
2		2,128	23,392	1464,88	227056,4	0,0
6		2,437	25,746	2620,88	406236,4	0,0
11	10	1,755	21,097	3236,32	501629,6	0,0
16		1,299	16,885	3219,80	499069,0	7,3
21		2,020	22,283	2710,24	420087,2	13,6
2		2,229	24,607	866,24	134267,2	0,0
6		1,533	19,552	2719,52	421525,6	0,0
11	15	2,134	23,185	2344,04	363326,2	0,0
16		1,854	21,299	1956,32	303229,6	0,0
21		1,876	22,272	3160,60	489893,0	7,4
2		2,318	25,257	1142,28	177053,4	0,0
6		1,746	20,983	2355,56	365111,8	0,0
11	20	2,056	22,701	2253,92	349357,6	0,0
16		1,570	18,957	2969,24	460232,2	0,3
21		2,237	24,570	2137,68	331340,4	0,1

Tabelle B.1: Hillclimbing (Diskret, auf/ab) – Sphere

WRONG_SOLUT–IONS_IN_A_ROW	*WRONG_SOLUT–IONS_PER_DIR*	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2		8,056	23,129	1258,00	194990,0	0,0
6		11,407	27,029	1014,60	157263,0	41,2
11	1	16,807	31,061	1259,76	195262,8	48,2
16		15,293	30,927	1138,04	176396,2	46,7
21		9,734	25,273	1764,60	273513,0	42,2
2		12,363	27,604	914,4	141732,0	0,0
6		8,400	26,175	4821,72	747366,6	0,1
11	5	10,587	29,925	4757,44	737403,2	0,6
16		6,248	28,048	5333,16	826639,8	2,8
21		6,357	30,768	4629,56	717581,8	1,0
2		8,932	23,315	1943,76	301282,8	0,0
6		5,258	24,459	3496,32	541929,6	0,1
11	10	4,124	27,619	5082,32	787759,6	1,3
16		5,358	27,562	5271,08	817017,4	0,2
21		2,944	23,77	4018,84	622920,2	1,2
2		9,592	20,054	1431,92	221947,6	0,0
6		9,845	28,225	4370,24	677387,2	0,9
11	15	11,104	31,507	4355,04	675031,2	1,0
16		7,046	22,645	4140,28	641743,4	1,3
21		3,449	26,807	5780,52	895980,6	0,3
2		16,227	25,814	943,52	146245,6	0,0
6		4,285	25,040	4366,04	676736,2	0,0
11	20	3,382	23,567	4794,68	743175,4	0,1
16		5,545	24,979	4856,92	752822,6	0,1
21		2,573	24,734	5315,72	823936,6	0,1

Tabelle B.2: Hillclimbing (Diskret, auf/ab) – Matya

WRONG_SOLUT–IONS_IN_A_ROW	Nachbar-schaft (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2		2,695	27,164	3,72	576,6	0,0
6		1,313	16,849	44,80	6944,0	0,0
11	1	0,059	1,547	179,52	27825,6	0,0
16		0,000	0,131	197,40	30597,0	0,0
21		0,000	0,132	195,32	30274,6	0,0
2		2,322	25,759	3,52	545,6	0,0
6		2,524	25,985	11,80	1829,0	0,0
11	5	1,399	16,501	40,00	6200,0	0,0
16		0,700	10,220	68,60	10633,0	0,0
21		0,221	4,115	80,40	12462,0	0,0
2		2,317	24,634	4,04	626,2	0,0
6		1,496	18,578	11,44	1773,2	0,0
11	10	1,327	17,366	20,00	3100,0	0,0
16		0,235	5,915	50,28	7793,4	0,0
21		0,627	10,670	54,44	8438,2	0,0
2		1,792	21,282	4,04	626,2	0,0
6		1,246	16,929	10,68	1655,4	0,0
11	15	1,225	17,018	22,60	3503,0	0,3
16		0,649	11,529	28,00	4340,0	0,3
21		0,272	6,167	61,04	9461,2	0,2
2		2,334	24,882	3,36	520,8	0,0
6		1,184	16,966	10,76	1667,8	0,0
11	20	0,899	14,514	18,16	2814,8	0,0
16		0,442	9,026	31,40	4867,0	0,7
21		0,553	10,629	33,96	5263,8	1,6

Tabelle B.3: Hillclimbing (Diskret in Nachbarschaft) – Sphere

WRONG_SOLUT_IONS_IN_A_ROW	Nachbarschaft (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2		17,968	27,276	4,48	694,4	0,0
6		9,143	21,360	43,88	6801,4	0,0
11	1	0,370	5,039	184,12	28538,6	0,0
16		0,065	1,850	216,00	33480,0	0,0
21		0,001	0,262	217,76	33752,8	0,0
2		16,379	24,615	3,48	539,4	0,0
6		10,692	20,589	9,40	1457,0	0,0
11	5	6,759	20,832	29,24	4532,2	0,0
16		2,707	11,564	62,48	9684,4	0,0
21		0,358	6,578	86,56	13416,8	0,0
2		12,018	25,654	4,2	651,0	0,0
6		12,411	23,167	10,68	1655,4	0,0
11	10	5,058	22,977	20,96	3248,8	0,3
16		0,749	15,494	43,00	6665,0	0,0
21		2,358	14,042	50,52	7830,6	0,1
2		18,651	25,903	3,68	570,4	0,0
6		4,689	18,571	11,28	1748,4	0,0
11	15	5,566	17,905	21,84	3385,2	1,3
16		1,632	12,071	31,36	4860,8	0,0
21		0,370	11,831	44,08	6832,4	0,2
2		11,229	23,065	3,88	601,4	0,0
6		4,230	20,435	10,96	1698,8	0,2
11	20	2,254	16,220	18,04	2796,2	0,2
16		2,277	15,651	24,92	3862,6	1,6
21		0,996	17,458	37,80	5859,0	1,8

Tabelle B.4: Hillclimbing (Diskret in Nachbarschaft) – Matya

WRONG_SOLUT–IONS_IN_A_ROW	Nachbar-schaft (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2		2,466	25,335	3,72	576,6	0,0
6		1,107	14,317	62,88	9746,4	0,0
11	1	0,009	0,472	163,36	25320,8	0,0
16		0,000	0,124	196,04	30386,2	0,0
21		0,000	0,139	204,08	31632,4	0,0
2		3,067	29,437	3,80	589,0	0,0
6		2,029	23,709	12,00	1860,0	0,0
11	5	1,487	18,421	29,92	4637,6	0,0
16		0,445	6,287	68,80	10664,0	0,0
21		0,188	3,037	95,20	14756,0	0,0
2		2,506	25,049	4,36	675,8	0,0
6		1,456	17,379	12,28	1903,4	0,0
11	10	1,511	18,269	18,80	2914,0	0,0
16		0,801	12,953	39,92	6187,6	0,0
21		0,225	5,834	53,44	8283,2	0,0
2		2,562	26,496	3,64	564,2	0,0
6		1,090	16,194	10,88	1686,4	0,0
11	15	1,023	14,886	21,76	3372,8	0,6
16		0,675	11,984	26,36	4085,8	0,9
21		0,127	4,751	46,36	7185,8	0,5
2		2,303	24,454	3,88	601,4	0,0
6		1,134	16,271	10,24	1587,2	0,0
11	20	0,738	13,035	18,40	2852,0	1,1
16		0,872	13,430	25,60	3968,0	1,2
21		0,544	10,702	35,36	5480,8	0,7

Tabelle B.5: Hillclimbing (Kontinuierlich in Nachbarschaft) – Sphere

WRONG_SOLUT_IONS_IN_A_ROW	Nachbarschaft (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2		16,72	27,970	4,12	638,6	0,0
6		7,471	19,047	55,52	8605,6	0,0
11	1	0,084	4,139	202,20	31341,0	0,0
16		0,003	0,685	221,00	34255,0	0,0
21		0,000	0,249	265,92	41217,6	0,0
2		19,990	27,290	3,68	570,4	0,0
6		21,717	30,968	10,00	1550,0	0,0
11	5	4,213	16,548	41,92	6497,6	0,0
16		0,629	8,488	66,84	10360,2	0,0
21		0,568	4,625	100,92	15642,6	0,0
2		9,653	23,155	3,68	570,4	0,0
6		12,100	22,063	10,32	1599,6	0,0
11	10	5,012	21,247	20,96	3248,8	0,0
16		3,205	13,959	37,52	5815,6	0,4
21		0,453	9,852	50,48	7824,4	0,1
2		11,121	26,641	3,48	539,4	0,0
6		3,983	18,295	10,04	1556,2	0,2
11	15	4,120	17,360	20,84	3230,2	0,2
16		3,746	16,285	25,32	3924,6	0,0
21		0,499	12,628	46,48	7204,4	0,3
2		11,468	23,305	4,00	620,0	0,0
6		4,424	20,804	10,24	1587,2	0,3
11	20	2,309	16,109	21,72	3366,6	0,6
16		3,766	16,256	27,20	4216,0	0,5
21		0,562	9,689	40,84	6330,2	1,3

Tabelle B.6: Hillclimbing (Kontinuierlich in Nachbarschaft) – Matya

WRONG_SOLUT-IONS_IN_A_ROW	*WRONG_SOLUT-IONS_PER_DIR*	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2		42,786	27,287	188,76	29257,8	1,3
6		48,115	31,347	126,00	19530,0	37,2
11	1	41,473	26,687	144,16	22344,8	43,1
16		43,029	27,808	115,64	17924,2	57,8
21		43,609	28,570	146,20	22661,0	44,1
2		43,268	27,558	95,12	14743,6	0,0
6		41,330	27,504	358,44	55558,2	0,1
11	5	32,232	19,718	554,36	85925,8	11,4
16		36,741	24,195	629,20	97526,0	13,0
21		36,519	24,590	720,48	111674,4	15,0
2		43,110	28,443	176,56	27366,8	0,0
6		41,896	28,392	387,60	60078,0	0,1
11	10	34,161	22,134	538,28	83433,4	0,1
16		39,138	25,269	601,36	93210,8	7,2
21		39,291	26,228	719,08	111457,4	12,7
2		42,671	27,937	150,40	23312,0	0,0
6		38,870	24,957	305,80	47399,0	0,0
11	15	37,900	24,147	513,60	79608,0	0,1
16		34,696	22,032	535,88	83061,4	0,0
21		40,250	26,597	562,04	87116,2	5,9
2		40,703	25,450	118,56	18376,8	0,0
6		42,592	27,915	225,56	34961,8	0,0
11	20	40,224	26,821	416,96	64628,8	0,0
16		39,350	26,538	535,44	82993,2	0,2
21		41,149	27,107	504,52	78200,6	0,2

Tabelle B.7: Hillclimbing (Diskret, auf/ab) – Ackley

WRONG_SOLUT-IONS_IN_A_ROW	*WRONG_SOLUT-IONS_PER_DIR*	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2		16,421	27,882	222,56	34496,8	0,0
6		16,323	28,253	198,64	30789,2	33,2
11	1	15,670	27,115	244,64	37919,2	37,1
16		16,546	27,396	233,80	36239,0	50,2
21		13,014	24,141	342,84	53140,2	38,0
2		11,012	25,523	244,2	37851,0	0,0
6		17,991	28,109	658,64	102089,2	0,0
11	5	20,210	23,775	1012,08	156872,4	10,7
16		16,772	26,837	608,2	94271,0	17,3
21		21,225	27,883	956,56	148266,8	11,5
2		17,605	30,894	239,72	37156,6	0,0
6		17,580	26,503	664,52	103000,6	0,0
11	10	19,533	25,779	701,32	108704,6	0,0
16		17,260	30,644	587,04	90991,2	9,8
21		15,452	22,704	549,20	85126,0	17,5
2		14,786	27,771	164,28	25463,4	0,0
6		16,787	24,741	415,92	64467,6	0,0
11	15	17,195	26,094	811,44	125773,2	0,0
16		16,688	25,547	689,36	106850,8	0,0
21		18,456	30,106	910,44	141118,2	4,7
2		16,193	26,931	177,60	27528,0	0,0
6		15,865	26,308	877,52	136015,6	0,0
11	20	18,574	25,774	410,16	63574,8	0,0
16		18,163	25,197	860,88	133436,4	0,1
21		21,601	29,430	617,00	95635,0	0,0

Tabelle B.8: Hillclimbing (Diskret, auf/ab) – Schwefel

WRONG_SOLUT–IONS_IN_A_ROW	Nachbar-schaft (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2		43,230	27,089	3,760	582,8	0,0
6		38,995	28,031	17,96	2783,8	0,3
11	1	31,497	22,546	36,12	5598,6	0,0
16		37,990	28,743	45,04	6981,2	0,4
21		37,027	27,890	51,08	7917,4	0,4
2		42,326	27,982	3,72	576,6	0,0
6		34,439	22,328	10,84	1680,2	0,0
11	5	30,258	20,442	27,60	4278,0	0,0
16		25,998	17,417	40,16	6224,8	0,0
21		23,592	16,000	58,04	8996,2	0,0
2		43,171	28,164	3,88	601,4	0,0
6		30,547	18,348	13,08	2027,4	0,0
11	5	32,666	19,036	19,96	3093,8	0,0
16		23,175	12,705	37,96	5883,8	0,3
21		23,403	13,830	49,72	7706,6	0,4
2		37,414	22,314	3,76	582,8	0,0
6		31,315	19,129	10,84	1680,2	0,0
11	15	26,428	13,959	19,96	3093,8	0,1
16		26,105	14,483	26,60	4123,0	0,6
21		18,200	8,436	45,80	7099,0	0,1
2		42,148	27,472	4,16	644,8	0,0
6		28,067	15,022	10,00	1550,0	0,0
11	20	25,755	13,173	22,76	3527,8	0,4
16		23,728	11,671	28,28	4383,4	1,7
21		21,165	10,409	38,36	5945,8	0,4

Tabelle B.9: Hillclimbing (Diskret in Nachbarschaft) – Ackley

WRONG_SOLUT–IONS_IN_A_ROW	Nachbar-schaft (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2		10,804	25,726	4,08	632,4	0,0
6		22,441	29,148	27,28	4228,4	0,0
11	1	22,989	23,360	39,52	6125,6	0,0
16		28,895	27,490	61,32	9504,6	0,1
21		31,494	30,313	70,96	10998,8	0,1
2		13,861	24,095	3,60	558,0	0,0
6		20,257	26,085	9,84	1525,2	0,0
11	5	25,378	28,164	24,68	3825,4	0,0
16		23,382	25,636	34,04	5276,2	0,1
21		26,136	28,322	46,44	7198,2	0,0
2		13,844	26,715	4,04	626,2	0,0
6		21,943	27,476	11,96	1853,8	0,0
11	10	27,293	33,420	18,64	2889,2	0,0
16		29,156	30,804	31,00	4805,0	0,1
21		29,582	32,093	41,88	6491,4	0,3
2		12,999	25,520	3,56	551,8	0,0
6		25,078	34,633	10,20	1581,0	0,0
11	15	27,222	32,868	21,00	3255,0	0,0
16		29,260	32,313	29,00	4495,0	0,3
21		28,232	33,594	33,48	5189,4	0,8
2		15,610	28,861	3,88	601,4	0,0
6		23,255	31,080	11,24	1742,2	0,0
11	20	26,452	33,134	21,24	3292,2	0,4
16		27,888	34,338	28,64	4439,2	0,7
21		33,529	36,959	37,60	5828,0	1,4

Tabelle B.10: Hillclimbing (Diskret in Nachbarschaft) – Schwefel

WRONG_SOLUT–IONS_IN_A_ROW	Nachbar-schaft (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2		41,452	26,022	3,84	595,2	0,0
6		37,429	26,839	19,40	3007,0	0,0
11	1	34,700	25,226	37,16	5759,8	0,0
16		34,293	24,877	49,04	7601,2	0,4
21		36,467	26,713	51,32	7954,6	0,0
2		43,700	28,017	3,56	551,8	0,0
6		38,620	26,366	10,56	1636,8	0,0
11	5	34,748	23,935	25,76	3992,8	0,0
16		32,365	22,466	38,12	5908,6	0,0
21		26,175	18,056	58,52	9070,6	0,1
2		35,969	22,231	3,84	595,2	0,0
6		33,012	20,433	13,12	2033,6	0,0
11	10	30,772	19,187	20,96	3248,8	0,7
16		21,026	10,930	42,32	6559,6	0,0
21		19,825	10,830	57,12	8853,6	0,1
2		34,842	19,929	3,96	613,8	0,0
6		33,241	20,125	12,08	1872,4	0,7
11	15	31,601	18,043	18,08	2802,4	0,0
16		25,901	14,082	30,04	4656,2	0,4
21		18,518	9,222	47,88	7421,4	0,4
2		36,945	22,085	3,64	564,2	0,0
6		25,208	13,264	11,16	1729,8	0,0
11	20	27,875	14,100	18,76	2907,8	0,0
16		25,561	13,524	26,24	4067,2	1,7
21		24,099	12,164	35,04	5431,2	0,6

Tabelle B.11: Hillclimbing (Kontinuierlich in Nachbarschaft) – Ackley

WRONG_SOLUT–IONS_IN_A_ROW	Nachbar-schaft (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2		13,438	25,026	3,72	576,6	0,0
6		21,736	28,471	28,40	4402,0	0,3
11	1	29,117	30,437	53,24	8252,2	0,3
16		28,556	29,525	53,84	8345,2	0,4
21		27,020	26,266	64,16	9944,8	0,0
2		12,123	28,003	4,00	620,0	0,0
6		15,967	27,942	10,60	1643,0	0,0
11	5	25,487	31,015	21,52	3335,6	0,2
16		26,045	25,991	34,36	5325,8	0,0
21		22,501	24,012	39,80	6169,0	0,0
2		15,799	28,617	3,52	545,6	0,0
6		20,930	27,393	10,64	1649,2	0,0
11	10	30,151	33,851	21,60	3348,0	0,5
16		27,946	30,535	30,28	4693,4	0,4
21		28,115	28,197	40,64	6299,2	0,1
2		10,230	22,798	3,88	601,4	0,0
6		23,246	30,395	11,80	1829,0	0,0
11	15	26,614	32,099	21,48	3329,4	0,8
16		30,308	34,956	30,48	4724,4	1,3
21		33,681	36,979	40,20	6231,0	0,6
2		15,067	28,166	3,60	558,0	0,0
6		19,871	33,187	11,40	1767,0	2,0
11	20	26,984	31,139	19,68	3050,4	0,2
16		28,611	34,103	29,00	4495,0	1,4
21		29,760	33,105	37,96	5883,8	0,2

Tabelle B.12: Hillclimbing (Kontinuierlich in Nachbarschaft) – Schwefel

WRONG_SOLUT–IONS_IN_A_ROW	*WRONG_SOLUT–IONS_PER_DIR*	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2		99,728	55,442	23,8	186,2	0,0
6		97,895	59,969	37,7	291,2	40,7
11	1	99,988	53,784	12,0	93,4	75,0
16		99,133	59,341	24,4	193,3	73,6
21		98,681	58,356	27,0	214,1	80,4
2		95,752	47,413	3,5	28,2	0,0
6		98,968	45,032	335,3	2635,8	0,0
11	5	99,126	42,825	70,5	555,0	23,4
16		89,299	45,775	85,5	650,4	35,1
21		90,060	48,223	78,2	575,2	43,8
2		99,059	51,962	48,2	338,7	0,0
6		99,481	47,300	137,5	1085,0	0,0
11	10	90,501	44,340	348,0	2988,1	4,10
16		94,213	53,262	257,9	2103,0	16,6
21		89,825	52,189	112,2	917,7	26,1
2		99,294	41,200	29,5	236,8	0,0
6		96,389	47,943	61,3	491,7	0,0
11	15	94,085	38,868	692,7	5772,6	2,5
16		89,249	50,879	260,3	2098,8	0,0
21		98,247	54,265	33,1	266,1	18,1
2		99,779	54,212	4,1	32,9	0,0
6		99,316	48,903	290,8	2334,8	0,0
11	20	88,143	49,586	644,1	5639,7	0,0
16		95,972	49,642	103,1	823,0	0,0
21		89,022	51,349	146,8	1183,0	0,0

Tabelle B.13: Hillclimbing (Diskret, auf/ab) – Basis-SCPN

WRONG_SOLUT-IONS_IN_A_ROW	Nachbar-schaft (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2		89,158	44,856	3,5	26,5	0,0
6		89,384	53,751	8,9	64,4	0,0
11	1	65,867	51,227	24,9	194,6	0,0
16		71,345	48,745	29,7	229,8	0,0
21		70,888	51,394	34,7	276,5	0,0
2		89,800	43,668	3,8	30,4	0,0
6		79,088	44,411	7,9	61,9	0,0
11	5	46,585	58,052	29,9	226,7	0,0
16		28,025	54,590	37,2	285,3	0,0
21		20,634	46,303	55,1	429,6	0,0
2		84,837	60,997	3,1	24,0	0,0
6		75,631	41,494	8,0	60,8	0,0
11	10	44,056	39,492	16,6	127,3	0,0
16		39,671	52,972	27,1	207,4	0,0
21		24,256	47,236	39,3	299,7	0,0
2		80,373	44,817	3,3	24,1	0,0
6		67,772	55,701	9,0	69,2	1,8
11	15	66,521	41,695	14,2	105,1	0,8
16		65,756	51,168	20,8	151,0	1,2
21		13,603	39,348	50,8	378,4	1,8
2		99,947	54,077	3,1	22,6	0,0
6		54,408	38,127	10,7	80,2	1,5
11	20	68,564	57,257	18,5	134,3	1,7
16		43,359	55,861	23,0	168,7	0,0
21		49,731	36,710	29,8	221,3	3,5

Tabelle B.14: Hillclimbing (Diskret in Nachbarschaft) – Basis-SCPN

WRONG_SOLUT_IONS_IN_A_ROW	Nachbarschaft (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2		98,217	55,420	3,3	26,2	0,0
6		80,399	55,577	8,9	66,3	0,0
11	1	69,718	43,843	22,3	172,3	0,0
16		88,860	53,892	22,8	168,3	0,0
21		54,115	50,504	56,5	452,4	0,0
2		84,135	43,617	3,4	26,1	0,0
6		60,866	61,239	10,5	78,9	0,0
11	5	61,564	47,278	22,3	168,7	1,4
16		57,267	52,302	34,1	257,9	0,0
21		35,566	45,532	50,9	393,9	0,0
2		84,140	39,530	3,7	28,2	0,0
6		64,873	35,082	10,8	82,3	1,5
11	10	47,814	45,512	20,8	159,3	0,0
16		48,226	44,191	36,5	276,8	0,0
21		29,515	50,112	49,6	378,2	0,3
2		90,033	52,160	3,5	25,8	0,0
6		60,796	48,131	9,6	73,0	0,0
11	15	63,651	40,546	16,3	120,2	0,0
16		32,194	42,948	24,1	178,8	1,3
21		42,103	43,548	29,2	218,9	0,5
2		97,448	56,513	3,4	24,3	0,0
6		80,725	44,973	8,8	63,8	0,8
11	20	62,305	54,102	20,1	147,7	2,7
16		54,770	49,630	23,2	171,1	0,0
21		44,413	49,244	28,7	208,7	3,8

Tabelle B.15: Hillclimbing (Kontinuierlich in Nachbarschaft) – Basis-SCPN

WRONG_SOLUT-IONS_IN_A_ROW	*WRONG_SOLUT-IONS_PER_DIR*	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
6		53,564	54,921	7,4	373,4	54,4
6		44,129	36,978	7,7	409,8	52,5
11	1	50,558	61,654	13,0	1800,4	69,6
16		47,325	64,269	17,5	2693,6	80,1
21		57,729	46,506	23,3	1518,2	81,9
2		48,607	55,079	3,5	341,8	0,0
6		44,355	53,692	8,9	1313,5	0,0
11	5	41,727	46,985	13,3	1769,1	37,9
16		46,451	44,091	18,1	1850,7	55,7
21		46,942	57,553	24,0	2020,0	63,0
2		48,311	63,446	3,5	486,8	0,0
6		50,987	52,287	8,6	721,1	0,0
11	10	41,445	51,681	13,4	2075,5	6,7
16		46,192	47,310	17,4	1530,7	28,8
21		53,687	48,496	23,7	2476,1	42,5
2		45,934	54,111	3,9	472,9	0,0
6		55,759	60,264	7,7	1180,2	0,0
11	15	51,109	55,005	14,2	1596,3	0,0
16		30,103	63,041	23,7	2586,5	4,7
21		51,170	53,906	25,9	1377,8	26,1
2		50,750	63,448	3,5	307,2	0,0
6		55,625	43,532	7,9	335,1	0,0
11	20	43,777	39,117	13,0	618,4	0,0
16		42,255	40,244	19,0	919,1	0,0
21		59,505	56,697	25,0	2353,2	0,0

Tabelle B.16: Hillclimbing (Diskret, auf/ab) – Hausenergiemodell

WRONG_SOLUT–IONS_IN_A_ROW	Nachbar-schaft (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2		37,421	52,284	3,2	413,7	3,3
6		54,436	48,683	12,0	546,9	5,2
11	1	38,947	58,562	20,0	1616,4	4,6
16		40,787	52,965	27,9	3346,9	6,4
21		50,624	51,973	69,3	3013,9	14,4
2		29,625	55,835	3,8	620,6	0,0
6		28,274	55,364	12,0	1235,3	0,0
11	5	27,845	41,940	30,3	2815,4	1,3
16		23,925	40,035	35,3	1645,6	1,8
21		28,067	32,997	71,8	2650,9	0,6
2		40,432	46,276	3,9	300,9	0,0
6		25,213	58,463	13,0	1195,9	1,1
11	10	18,588	51,030	23,7	2494,4	0,3
16		18,313	56,259	49,5	4395,5	0,8
21		13,568	32,279	49,3	2706,9	0,7
2		49,478	55,572	3,8	275,0	0,0
6		30,517	52,280	10,9	1012,2	0,5
11	15	18,699	40,593	19,9	2030,1	2,4
16		13,292	45,706	37,9	5165,9	1,2
21		11,840	44,154	45,2	5625,1	5,8
2		36,300	58,750	3,9	492,9	0,0
6		16,016	38,954	14,7	1086,3	6,7
11	20	7,385	22,554	22,3	2481,4	10,5
16		7,613	34,873	37,1	3656,4	6,0
21		6,887	25,184	54,0	3656,9	10,9

Tabelle B.17: Hillclimbing (Diskret in Nachbarschaft) – Hausenergiemodell

WRONG_SOLUT–IONS_IN_A_ROW	Nachbar-schaft (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit (Sekunden)	Cache-Hit-Ratio (%)
2	1	41,843	40,982	3,0	261,9	33,3
6		49,150	52,319	7,0	382,2	71,4
11		66,346	58,940	12,0	692,4	83,3
16		46,882	53,817	17,0	2344,3	88,2
21		64,938	50,461	22,0	1018,6	90,9
2	5	61,344	52,934	3,0	126,9	33,3
6		45,472	54,864	7,0	631,4	71,4
11		40,032	39,456	12,0	1117,2	83,3
16		57,015	53,642	17,0	938,4	88,2
21		57,599	67,205	22,0	3766,4	90,9
2	10	70,457	63,443	3,0	151,2	33,3
6		50,368	56,464	7,0	689,5	71,4
11		47,536	46,740	12,0	1257,6	83,3
16		61,551	58,515	17,0	1023,4	88,2
21		52,384	44,978	22,0	930,6	90,9
2	15	39,509	58,874	3,4	598,6	21,7
6		31,303	47,460	9,4	829,5	47,0
11		27,841	44,477	17,2	1316,2	53,7
16		22,890	66,534	27,9	2362,8	60,3
21		10,387	61,857	32,6	4566,1	62,3
2	20	37,243	58,938	3,2	190,7	16,7
6		34,045	44,081	11,7	838,1	28,5
11		28,520	54,006	15,9	1862,4	43,6
16		19,242	47,432	22,2	1880,2	50,5
21		6,737	61,481	33,5	4953,9	52,2

Tabelle B.18: Hillclimbing (Kontinuierlich in Nachbarschaft) – Hausenergiemodell

Populations-größe	Mutationswahr-scheinlichkeit (%)	Sphere					Matya				
		Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU- Zeit (Sekunden)	Cache-Hit-Ratio (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU- Zeit (Sekunden)	Cache-Hit-Ratio (%)
1	0	2,973	27,923	6,0	930,0	100,000	15,419	27,301	6,0	930,0	100,000
	10	1,724	21,172	7,2	1116,0	33,333	10,175	24,567	6,3	976,5	33,333
	20	0,869	14,394	6,9	1069,5	33,333	1,518	19,117	7,5	1162,5	33,333
	30	1,553	21,285	6,6	1023,0	33,333	2,608	12,085	7,5	1162,5	33,333
	40	1,596	21,55	6,3	976,5	33,333	2,665	22,899	8,4	1302,0	33,333
	50	0,906	15,531	7,5	1162,5	33,333	4,801	20,851	6,6	1023,0	33,333
5	0	0,837	14,95	14,0	2170,0	100,0	1,563	17,588	14,0	2170,0	100,0
	10	0,361	9,632	17,5	2712,5	11,905	0,912	12,536	20,3	3146,5	12,333
	20	0,425	10,256	18,2	2821,0	11,905	0,806	12,775	16,1	2495,5	13,810
	30	0,299	8,55	19,6	3038,0	13,286	0,950	19,589	16,8	2604,0	12,143
	40	0,377	9,555	19,6	3038,0	13,524	1,079	15,377	18,9	2929,5	14,000
	50	0,573	12,475	17,5	2712,5	12,024	1,049	13,624	18,2	2821,0	12,857
10	0	0,456	11,24	24,0	3720,0	100,000	1,231	14,65	24,0	3720,0	100,000
	10	0,234	7,743	31,2	4836,0	7,708	0,450	9,79	38,4	5952,0	7,542
	20	0,058	3,695	32,4	5022,0	7,639	0,511	11,257	28,8	4464,0	8,333
	30	0,231	7,618	31,2	4836,0	8,125	0,381	8,39	27,6	4278,0	7,500
	40	0,306	7,954	31,2	4836,0	8,125	0,355	7,418	32,4	5022,0	6,750
	50	0,075	4,469	33,6	5208,0	7,847	0,661	14,95	32,4	5022,0	8,333
15	0	0,361	10,185	34,0	5270,0	100,000	1,016	14,595	34,0	5270,0	100,000
	10	0,098	4,332	47,6	7378,0	5,471	0,329	9,010	42,5	6587,5	5,49
	20	0,190	6,849	40,8	6324,0	5,196	0,212	6,826	47,6	7378,0	5,245
	30	0,092	4,915	39,1	6060,5	5,392	0,444	9,287	42,5	6587,5	5,686
	40	0,076	4,500	54,4	8432,0	5,353	0,267	6,054	39,1	6060,5	5,686
	50	0,078	4,603	47,6	7378,0	5,686	0,265	8,768	39,1	6060,5	5,686

Tabelle B.19: SBX: Sphere / Matya, alle Ergebnisse

Populations-größe	Mutationswahr-scheinlichkeit (%)	Ackley					Schefel				
		Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU- Zeit (Sekunden)	Cache-Hit-Ratio (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU- Zeit (Sekunden)	Cache-Hit-Ratio (%)
1	0	42,148	26,946	6,0	930,0	100,000	45,782	28,673	6,0	930,0	100,000
	10	32,109	17,117	6,6	1023,0	33,333	37,612	28,592	6,9	1069,5	33,333
	20	31,545	15,949	6,3	976,5	33,333	38,397	29,413	7,5	1162,5	33,333
	30	32,533	17,835	6,3	976,5	33,333	35,050	29,787	7,5	1162,5	33,333
	40	30,086	15,966	8,1	1255,5	33,333	34,557	24,470	7,5	1162,5	33,333
	50	38,920	22,220	7,5	1162,5	33,333	31,939	28,806	7,8	1209,0	33,333
5	0	33,240	17,636	14,0	2170,0	100,000	32,723	32,659	14,0	2170,0	100,000
	10	22,959	10,873	16,1	2495,5	11,905	20,909	31,891	18,2	2821,0	12,738
	20	22,912	10,751	14,0	2170,0	12,857	22,363	33,695	17,5	2712,5	13,095
	30	17,978	8,714	21,7	3363,5	11,810	20,561	35,143	21,7	3363,5	11,333
	40	20,062	8,522	20,3	3146,5	13,000	22,837	29,383	17,5	2712,5	12,738
	50	24,646	12,750	15,4	2387,0	12,381	24,699	33,320	15,4	2387,0	13,571
10	0	21,867	9,089	24,0	3720,0	100,000	29,144	30,588	24,0	3720,0	100,000
	10	17,576	7,461	30,0	4650,0	8,056	19,184	37,313	28,8	4464,0	8,125
	20	19,505	8,405	37,2	5766,0	7,917	21,660	32,693	31,2	4836,0	7,014
	30	18,031	8,076	31,2	4836,0	7,208	17,778	34,380	30,0	4650,0	7,75
	40	14,436	4,998	32,4	5022,0	7,847	22,959	35,049	33,6	5208,0	8,167
	50	18,061	6,331	31,2	4836,0	7,778	15,994	36,707	30,0	4650,0	8,056
15	0	19,933	8,000	34,0	5270,0	100,000	21,276	36,360	34,0	5270,0	100,000
	10	16,455	7,053	49,3	7641,5	5,294	14,036	37,987	47,6	7378,0	5,735
	20	14,854	6,003	47,6	7378,0	5,882	15,153	38,168	47,6	7378,0	5,049
	30	16,204	6,597	42,5	6587,5	5,882	18,116	35,200	42,5	6587,5	5,588
	40	17,231	7,016	47,6	7378,0	5,588	16,797	32,899	44,2	6851,0	5,441
	50	17,058	5,488	44,2	6851,0	5,539	15,669	36,646	49,3	7641,5	5,392

Tabelle B.20: SBX: Ackley / Schwefel, alle Ergebnisse

Populations-größe	Mutationswahr-scheinlichkeit (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU- Zeit (Sekunden)	Cache-Hit-Ratio (%)
1	0	100,000	46,370	6,0	56,4	100,000
	10	79,729	30,322	7,8	58,2	33,333
	20	71,790	51,832	7,5	54,7	33,333
	30	71,805	41,254	6,9	51,3	33,333
	40	67,691	39,955	6,9	50,2	33,333
	50	66,367	34,309	6,9	50,7	33,333
5	0	73,177	45,608	14,0	107,9	100,000
	10	22,164	31,063	16,8	130,3	14,286
	20	35,500	22,305	18,2	140,8	14,286
	30	49,492	30,921	15,4	118,2	14,286
	40	30,287	25,809	18,2	139,7	14,286
	50	28,325	29,483	21,7	166,1	14,286
10	0	31,892	31,031	24,0	186,7	100,000
	10	19,897	27,758	31,2	242,7	8,333
	20	26,683	13,950	31,2	242,6	8,333
	30	26,260	33,134	31,2	241,5	8,333
	40	25,431	20,889	31,2	240,6	8,333
	50	32,930	41,980	30,0	235,1	8,333
15	0	34,145	23,336	34,0	265,1	100,000
	10	20,846	33,890	42,5	335,0	5,882
	20	12,933	30,670	47,6	375,5	5,882
	30	15,665	27,097	44,2	348,5	5,882
	40	16,657	11,710	39,1	307,8	5,882
	50	16,497	44,541	45,9	364,9	5,882

Tabelle B.21: SBX: Basis-SCPN, alle Ergebnisse

Populations-größe	Mutationswahr-scheinlichkeit (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU- Zeit (Sekunden)	Cache-Hit-Ratio (%)
1	0	20,183	53,835	11,0	2271,5	81,818
	10	17,077	68,723	8,4	1041,0	34,095
	20	17,243	39,235	8,2	912,7	38,667
	30	12,295	63,137	8,6	1366,8	36,000
	40	20,733	43,521	7,8	782,7	41,333
	50	19,084	49,864	7,8	816,8	41,333
5	0	5,255	76,202	30,4	7893,6	66,968
	10	2,705	63,184	21,7	2796,2	12,163
	20	3,405	52,693	22,2	2238,9	13,586
	30	3,735	56,393	21,5	2348,6	12,962
	40	5,033	66,786	23,0	2507,6	13,083
	50	3,965	65,103	21,4	2986,8	13,925
10	0	3,407	64,527	56,0	7744,6	64,286
	10	2,459	59,353	40,6	4416,1	7,425
	20	2,289	68,445	38,6	5372,8	7,803
	30	2,667	57,382	38,8	5158,1	7,811
	40	2,498	68,146	36,1	4573,6	7,057
	50	4,142	51,659	39,0	4703,3	7,740
15	0	2,732	71,272	80,4	12841,1	62,682
	10	1,882	68,133	57,0	7043,0	5,273
	20	1,991	66,308	59,8	7073,7	5,034
	30	1,753	53,051	56,8	6477,9	5,293
	40	2,196	65,187	58,6	7891,9	5,129
	50	1,953	60,052	59,0	7578,9	5,099

Tabelle B.22: SBX: Hausenergiemodell, alle Ergebnisse

Basis-heuristik	Phasen-anzahl (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit	Cache-ratio
Hill-Climbing	2	0,058	2,993	77,9	76616,2	0,000
	3	0,013	1,439	116,9	87490,8	0,235
	4	0,003	0,962	158,2	111333,7	0,125
	5	0,001	0,420	186,0	94180,7	0,176
	6	0,000	0,188	204,1	109192,6	0,734
	7	0,000	0,105	225,2	98442,5	0,089
	8	0,000	0,044	258,2	112749,7	0,500
	9	0,000	0,020	300,4	107305,2	1,181
	10	0,000	0,008	329,9	119259,5	1,536
Simulated-Annealing	2	0,622	10,991	197,0	249425,0	75,381
	3	2,651	25,468	295,0	264517,0	83,085
	4	0,383	5,456	393,0	279609,0	87,379
	5	1,030	10,184	491,0	294799,0	89,552
	6	0,512	9,026	589,0	310087,0	91,070
	7	1,026	11,018	687,0	325669,0	92,271
	8	0,800	12,910	785,0	341153,0	93,159
	9	0,949	8,860	883,0	357127,0	93,681
	10	0,538	11,211	981,0	372611,0	94,281
Genetische-Algorithmen	2	0,126	6,045	92,8	14384,0	6,452
	3	0,332	4,767	121,7	18863,5	12,244
	4	0,016	2,082	171,0	26505,0	14,009
	5	0,009	1,103	215,2	33356,0	14,255
	6	0,002	0,822	249,2	38626,0	15,059
	7	0,000	0,204	310,4	48112,0	15,072
	8	0,010	0,439	358,0	55490,0	15,723
	9	0,002	0,158	405,6	62868,0	16,509
	10	0,000	0,045	422,6	65503,0	18,375

Tabelle B.23: Multiphasenoptimierung: Sphere

Basis-heuristik	Phasen-anzahl (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit	Cache-ratio
Hill-Climbing	2	1,265	8,464	80,5	83655,6	0,362
	3	0,635	4,260	121,3	92151,8	0,000
	4	0,085	1,326	147,1	101214,9	0,592
	5	0,035	0,873	194,4	86856,1	0,728
	6	0,005	0,510	204,2	110093,2	1,055
	7	0,002	0,237	253,5	108166,3	1,396
	8	0,000	0,085	294,3	126770,1	1,274
	9	0,000	0,042	318,8	111718,3	1,247
	10	0,000	0,013	345,3	128818,6	5,963
Simulated-Annealing	2	4,402	9,529	197,0	249425,0	75,431
	3	9,031	11,969	295,0	264517,0	83,153
	4	8,811	17,981	393,0	279609,0	87,125
	5	5,607	10,162	491,0	294799,0	89,470
	6	2,351	5,570	589,0	310087,0	91,002
	7	10,013	14,457	687,0	325669,0	92,154
	8	14,786	17,362	785,0	341153,0	93,083
	9	1,082	8,309	883,0	357127,0	93,703
	10	10,048	11,663	981,0	372611,0	94,2
Genetische-Algorithmen	2	4,656	20,185	84,3	13066,5	7,091
	3	1,433	8,637	126,8	19654,0	13,625
	4	0,302	5,092	181,2	28086,0	15,173
	5	0,318	2,789	211,8	32829,0	15,984
	6	0,123	1,823	273,0	42315,0	17,374
	7	0,022	0,712	313,8	48639,0	17,298
	8	0,001	0,222	342,7	53118,5	16,172
	9	0,004	0,093	390,3	60496,5	19,367
	10	0,001	0,121	436,2	67611,0	18,007

Tabelle B.24: Multiphasenoptimierung: Matya

Basis-heuristik	Phasen-anzahl (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit	Cache-ratio
Hill-Climbing	2	13,729	8,344	77,8	70625,7	0,127
	3	9,439	3,151	122,6	95007,6	0,744
	4	7,223	1,953	141,3	91248,4	0,192
	5	3,875	1,097	184,0	107591,5	0,109
	6	1,258	0,464	209,4	106033,5	0,106
	7	0,323	0,219	247,7	107880,1	0,238
	8	0,219	0,137	262,2	123747,9	0,500
	9	0,064	0,052	301,6	125602,9	0,476
	10	0,033	0,025	339,1	117446,9	1,093
Simulated-Annealing	2	7,748	7,679	197,0	249425,0	75,635
	3	6,025	6,848	295,0	264517,0	83,254
	4	7,971	9,293	393,0	279609,0	87,328
	5	11,698	25,229	491,0	294799,0	89,674
	6	5,376	9,557	589,0	310087,0	91,222
	7	3,778	3,534	687,0	325669,0	92,169
	8	7,490	11,269	785,0	341153,0	93,083
	9	10,689	9,888	883,0	357127,0	93,692
	10	12,992	13,839	981,0	372611,0	94,190
Genetische-Algorithmen	2	15,296	16,974	82,6	12803,0	8,200
	3	12,870	8,570	131,9	20444,5	12,977
	4	16,189	7,009	181,2	28086,0	13,500
	5	7,231	2,349	215,2	33356,0	15,573
	6	3,358	1,088	266,2	41261,0	16,374
	7	3,349	1,079	295,1	45740,5	17,288
	8	0,644	0,310	351,2	54436,0	15,652
	9	0,241	0,135	392,0	60760,0	15,281
	10	0,343	0,155	460,0	71300,0	18,281

Tabelle B.25: Multiphasenoptimierung: Ackley

Basis-heuristik	Phasen-anzahl (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit	Cache-ratio
Hill-Climbing	2	3,772	3,169	71,1	81322,8	0,387
	3	2,370	1,476	104,1	89930,2	0,181
	4	0,561	0,430	121,0	88786,1	1,302
	5	0,164	0,263	158,9	91776,5	0,234
	6	0,044	0,104	191,8	95574,3	1,043
	7	0,011	0,063	232,9	105599,7	1,050
	8	0,003	0,018	258,6	102152,9	2,280
	9	0,000	0,011	287,4	109118,4	2,062
	10	0,000	0,004	304,0	111207,7	4,925
Simulated-Annealing	2	12,680	5,429	197,0	249425,0	75,279
	3	11,586	4,543	295,0	264517,0	83,356
	4	16,880	12,799	393,0	279609,0	87,099
	5	9,648	10,701	491,0	294799,0	89,430
	6	8,199	14,713	589,0	310087,0	91,154
	7	18,170	11,199	687,0	325669,0	92,140
	8	12,205	2,873	785,0	341153,0	93,146
	9	14,326	12,736	883,0	357127,0	93,635
	10	20,019	10,346	981,0	372611,0	94,220
Genetische-Algorithmen	2	10,112	11,064	86,0	13330,0	6,671
	3	4,509	4,692	120,0	18600,0	15,867
	4	3,945	1,424	167,6	25978,0	20,631
	5	0,914	0,880	225,4	34937,0	22,001
	6	1,909	0,512	257,7	39943,5	18,382
	7	2,090	0,603	300,2	46531,0	23,024
	8	0,187	0,148	346,1	53645,5	21,425
	9	0,006	0,098	385,2	59706,0	23,572
	10	0,005	0,026	453,2	70246,0	26,377

Tabelle B.26: Multiphasenoptimierung: Schwefel

Basis-heuristik	Phasen-anzahl (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit	Cache-ratio
Hill-Climbing	2	1,881	49,759	79,7	1056,9	4,632
	3	11,154	49,793	112,7	1458,0	3,268
	4	10,248	34,240	162,0	2437,8	0,422
	5	0,921	34,306	184,8	2487,5	2,826
	6	3,109	35,790	243,6	2758,0	1,333
	7	3,877	30,648	279,5	3159,6	5,094
	8	7,043	53,567	278,2	2766,7	9,875
	9	5,401	28,768	345,4	4075,5	8,820
	10	20,132	57,234	349,5	3198,9	12,181
Simulated-Annealing	2	2,366	26,400	197,0	4607,0	52,386
	3	3,882	39,837	295,0	5032,4	68,203
	4	3,039	23,841	393,0	5928,0	75,751
	5	2,979	11,164	491,0	6524,1	80,489
	6	2,620	18,911	589,0	7340,3	83,158
	7	3,169	12,420	687,0	7766,2	85,662
	8	2,629	3,108	785,0	8603,6	87,274
	9	2,787	24,796	883,0	9446,5	88,766
	10	3,295	25,576	981,0	9895,4	89,664
Genetische-Algorithmen	2	11,914	34,832	92,8	661,3	5,036
	3	0,526	26,608	145,5	1045,7	4,753
	4	0,645	25,765	179,5	1308,9	4,535
	5	0,456	33,769	242,4	1777,3	4,472
	6	0,304	28,868	276,4	2045,0	4,531
	7	0,270	45,770	337,6	2505,4	4,627
	8	0,064	36,114	378,4	2848,2	4,775
	9	0,005	29,184	419,2	3169,1	7,163
	10	0,031	38,761	454,9	3444,6	10,571

Tabelle B.27: Multiphasenoptimierung: Basis-SCPN

Basis-heuristik	Phasen-anzahl (%)	Abstand (%)	Abstand-Euklid (%)	Simulations-Anzahl	CPU-Zeit	Cache-ratio
Hill-Climbing	2	3,354	54,868	57,4	13461,6	67,865
	3	6,245	58,870	80,8	15693,9	78,028
	4	4,657	69,540	109,0	18318,5	80,240
	5	6,561	44,590	126,1	15759,7	82,914
	6	6,321	65,032	146,0	32377,1	85,548
	7	2,084	45,383	169,0	25197,3	86,397
	8	2,965	49,586	199,0	29199,0	86,678
	9	7,293	47,917	223,1	29942,5	87,373
	10	4,829	52,834	241,5	37112,2	88,203
Simulated-Annealing	2	3,755	52,666	197,0	55111,7	53,249
	3	4,869	68,876	295,0	59033,9	68,746
	4	4,619	66,412	393,0	61826,8	75,954
	5	4,367	62,131	491,0	64222,9	80,367
	6	4,728	68,521	589,0	67316,9	83,497
	7	4,839	68,338	687,0	70333,1	85,779
	8	4,818	69,094	785,0	73804,2	87,312
	9	4,841	69,304	883,0	79457,6	88,550
	10	4,917	68,874	981,0	82599,0	89,684
Genetische-Algorithmen	2	2,233	40,046	92,8	9536,2	56,281
	3	2,911	47,910	125,1	15835,0	67,064
	4	4,402	66,985	172,7	16697,2	69,787
	5	10,649	43,793	213,5	24139,7	76,311
	6	6,677	61,446	257,7	29028,0	78,257
	7	3,451	68,945	298,5	30136,2	80,143
	8	7,131	46,148	332,5	34221,7	81,058
	9	11,016	63,501	380,1	45591,9	86,187
	10	8,540	44,471	415,8	40631,3	83,992

Tabelle B.28: Multiphasenoptimierung: Hausenergiemodell

Abbildungsverzeichnis

Tabellenverzeichnis

Glossar

Antwortfunktionsgebirge siehe Wertelandschaft. 154

Brute-Force-Methode Auch Methode der rohen Gewalt genannt. Das Systematische Durchprobieren aller, oder zumindest sehr vieler Möglichkeiten. 24

Cache Ergebnisdatenbank, Sammlung von Simulationsergebnissen bzw. Abbildung von Parametersätzen auf Werte von Leistungsmaßen. 65

Cache-Ratio Verhältnis von Simulationsergebnissen, die im Cache gefunden im Vergleich zu allen angeforderten Simulationsergebnissen des Optimierungssystems. 66

CamelCase Schreibweise von Variablennamen mit Binnenversalien um die Lesbarkeit zu verbessern. 121

csv Comma Separated Values, Komma-separierte Daten. Einfaches, typisches Format für Speicherung von strukturierten Daten, wie Tabellen. 84

Definition Festlegung innerhalb eines SCPN, welche nicht durch Parameter verändert werden kann. Kann auch Berechnungsvorschriften enthalten. 119

Definitionsraum Menge aller möglichen Werte aller veränderbaren Parameter eines Modells. Bei Fließkommawerten theoretisch unbegrenzt. 21, 22, 24, 30, 31, 64

Eingangsvektor Menge der Eingangsparameter. Auch: Parametersatz. 20

Erreichbarkeitsgraph Gerichteter Graph zur Darstellung aller Zustände eines Petri-Netzes, die ausgehend von der Initialmarkierung durch Schalten von Transitionen erreicht werden können. 35

global guard Siehe globale Schaltbedingung. 108

globale Schaltbedingung Eine zusätzliche Bedingung, die erfüllt sein muss, damit eine Transition schaltfähig wird. In SCPNs kann dies diverse Werte, Eigenschaften oder auch Leistungsmaße des gesamten Netzes beinhalten. 108

GSPN Generalized Stochastic Petri Net, Verallgemeinertes stochastisches Petri-Netz, Vorstufe von SCPNs. 35

Guard Zusätzliche Schaltbedingungen. 42

GUI Graphical User Interface, Grafische Benutzerschnittstelle, Auch: Nutzeroberfläche. 96

Hyperheuristik Heuristik, welche auf anderen Heuristiken basiert, diese kombiniert oder eine optimale Konfiguration dieser Heuristiken und ihrer Kombination zu finden versucht. 30, 178

JVM Java Virtual Machine, Virtuelle Maschine zur Ausführung von Java Bytecode. 53

Konfidenzniveau Häufigkeit, mit der der Simulationswert innerhalb der Grenzen des festgelegten Konfidenzintervalls um den Erwartungswert liegt . 16

Leistungsmaß Bewertungskriterium eines Modells bzw. einer Simulation. Englisch: performance measure. 15, 31, 37, 39, 42, 44, 45, 52, 53, 58, 59, 61, 62, 63, 64, 67, 71, 72, 73, 80, 83, 84, 96, 97, 98, 104, 108, 110, 111, 112, 113, 114, 115, 121, 126, 221, 226

LOC Lines of Code, Anzahl der Programmzeilen eines Programmes, grobes Maß für die Komplexität einer Software. 78

Nutzeroberfläche Grafische Oberfläche über die der Nutzer durch Maus- oder Tastatureingabe Daten eingibt und das Programm steuert. 95

Parametersatz Menge aller Parameter bzw. deren Belegung, die für eine Simulation benötigt werden. Auch: Systemkonfiguration. 21, 22

PV-Anlage Photovolatikanlage, Anlage zur direkten Umwandlung von Sonnenlicht in Elektrizität. 119

SCPN Stochastic Colored Petri Net, Farbiges stochastisches Petri-Netz. 11

Seed Startwert zur Initialisierung eines Zufallsgenerators. Deterministische Zufallsgeneratoren liefern bei gleichem Seed auch die gleiche Folge von Pseudo-Zufallszahlen. 45, 53

Substitutionstransition Eine Transition, die als Platzhalter für ein oder mehrere Subnetze dient. 109

Wertelandschaft Manchmal auch Optimierungsgebirge oder Antwortfunktionsgebirge genannt. Der entstehende Plot der Kostenfunktion für alle bzw. viele Wertepaare der Eingangsparameter. Bei genetischen Optimierungsfunktionen auch Fitnesslandschaft genannt. 43, 80, 114, 122, 125, 221

Zustandsraum Menge aller Zustände eines Systems oder Modells. 34, 35

Anhang C

Selbstständigkeitserklärung

Hiermit erkläre ich, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und nur die angegebene Literatur und Hilfsmittel verwendet zu haben.

Ilmenau, den 19. August 2018