Technische Universität Darmstadt

Department of Electrical Engineering and Information Technology
Department of Computer Science (Adjunct Professor)
Multimedia Communications Lab
Prof. Dr.-Ing. Ralf Steinmetz

Ein heuristisches Optimierungsverfahren zur dienstgütebasierten Komposition von Web-Service-Workflows

Technical Report

von

Michael Spahn, Rainer Berbner,
Oliver Heckmann, Ralf Steinmetz

09. 01. 2006

KOM-TR-2006-02

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AHP	Analytic Hierarchy Process
API	Application Programming Interface
B&B	Branch and Bound
BPE	Business Process Engine
BPEL4WS	Business Process Execution Language for Web Services
BPO	Business Process Outsourcing
CCITT	Commité Consultatif Internationale Télégrafique et Téléfonique
CSV	Comma Separated Value
EAI	Enterprise Application Integration
FSF	Free Software Foundation
GA	Genetische Algorithmen
GLOP	(gemischt) ganzzahliges lineares Optimierungsproblem
GPL	GNU General Public Licence
GUI	Graphical User Interface
HTT	Hyper-Threading-Technologie
IntServ	Integrated Services
ISDN	Integrated Services Digital Network
IT	Informationstechnologie
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
J2SE	Java 2 Platform Standard Edition
JNI	Java Native Interface
LGPL	GNU lesser general public license
LMCKP	Lineares Multiplechoice-Knapsack-Problem
LOP	Lineares Optimierungsproblem
LP	Lineare Programmierung
MCKP	Multiplechoice-Knapsack-Problem
METEOR	Managing End-to-End Operations
METEOR-S	METEOR for Semantic Web Services
MIP	Mixed Integer Programming
OM	Opening Method
OR	Operations Research
PMF	Probability Mass Function
QCWS	QoS-capable Web Service Architecture
QoS	Quality of Service
RSVP	Resource Reservation Protocol
SA	Simulated Annealing

SLA	Service Level Agreement
SMT	Simultaneous Multithreading
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
STP	Straight Through Processing
SWR	Stochastic Workflow Reduction
TSP	Traveling-Salesman-Problem
UDDI	Universal Description, Discovery and Integration
UIM	Unsophisticated Improvement Method
UML	Unified Modeling Language
WSAF	Web Service Agent Framework
WS-BPEL	Web Services Business Process Execution Language
WSDL	Web Service Description Language
WSQoSX	Web Service Quality of Service Architecture Extension

1 Einführung

1.1 Motivation

Die Märkte, auf denen Unternehmen agieren, befinden sich in einem stetigen Prozess der zunehmenden Globalisierung und Deregulierung. Auf diesen sind sie einer großen Anzahl von Wettbewerbern und einem zunehmenden Innovations- und Kostendruck ausgesetzt. Im harten Wettbewerb um Marktanteile können sie nur bestehen, indem sie hochgradig innovativ sind und sich schnell an neue Wettbewerbssituationen anpassen. Geschäftsprozesse der Unternehmen müssen hierzu einerseits flexibel sein, um schnell an neue Anforderungen adaptiert werden zu können, und andererseits effizient und kosteneffektiv umgesetzt werden können [32] [9].

Hierbei können Unternehmen, die es verstehen sich selbst mit anderen Unternehmen zu vernetzen und sie in ihre eigenen Geschäftsprozesse zu integrieren, besondere Vorteile erringen. Hierzu zählen z.B. die Reduktion von Transaktionskosten durch eine hohe Rate der Automatisierung und „Straight Through Processing“ (STP) [68] [69], sowie die Nutzung von Synergie- und Skaleneffekten [11]. Ansatzpunkte sind hierbei die bessere Integration vor- und nachgelagerter Wertschöpfungsstufen der Supply-Chain, sowie die Auslagerung von Geschäftsprozessen, die nicht den Kernkompetenzen der Unternehmung zuzuordnen sind, im Rahmen des „Business Process Outsourcing“ (BPO) [54] [29].

Grundvoraussetzung für die Integration externer Unternehmen in die eigenen Geschäftsprozesse und die Schaffung verteilter, unternehmensübergreifender Geschäftsprozesse [45] ist die Fähigkeit diese mithilfe einer flexiblen und kostengünstigen IT-Infrastruktur abbilden und ausführen zu können. Mit dem Internet steht eine leistungsfähige und kostengünstige Kommunikationsinfrastruktur zur Verfügung um die Netzwerke der Unternehmen zu verbinden. Die eigentliche Herausforderung liegt daher nicht in der Herstellung einer Verbindung zwischen den, an den Geschäftsprozessen beteiligten, Netzwerken, sondern vielmehr in der Überwindung der Heterogenität der beteiligten Systeme, Plattformen und Anwendungen.

Mit Methoden der „Enterprise Application Integration“ (EAI) [39] [5] [6] ist es möglich zunächst die Heterogenität innerhalb der unternehmenseigenen IT-Landschaft zu überwinden. Hierzu müssen heterogene, autonome Anwendungssysteme prozessorientiert in eine geeignete, vernetzte IT-Infrastruktur integriert werden. Über eine reine Schnittstellenadaption durch klassische Middleware hinaus, werden hierbei Funktionalitäten gekapselt und mittels Adapter über eine einheitliche Infrastruktur zugänglich gemacht. Als Systemarchitekturkonzept hat sich hierbei eine „Service Oriented Architecture“ (SOA) als vorteilhaft erwiesen, in welcher fachliche Dienste und Funktionalitäten in Form von Services bereitgestellt werden. Auf dieser Infrastruktur wird die Prozesslogik der Geschäftsprozesse durch eine Orchestrierung der Services modelliert und mittels einer „Business Process Engine“ (BPE) ausgeführt. Die Systeme externer Unternehmen können über Services, die sie nach außen anbieten, in die Geschäftsprozesse einer solchen Infrastruktur integriert werden.

Zur Umsetzung einer derartigen SOA steht mit dem Internet eine leistungsstarke und kostengünstige Kommunikationsinfrastruktur zur Verfügung und mit der darauf basierenden Technologie der Web Services [7] ein plattformunabhängiges, auf offenen Standards basierendes, serviceorientiertes Komponentenmodell. Zur Orchestrierung

von Geschäftsprozessen aus Web Services stehen Standards wie z.B. die „Business Process Execution Language for Web Services“ (BPEL4WS[1]) [8] mit entsprechenden Implementierungen von BPEs zur Verfügung.

Ein wichtiger Aspekt bei der Orchestrierung von Geschäftsprozessen ist jedoch nicht nur die Integration einer Funktionalität durch einen Web Service an sich, sondern auch die Berücksichtigung der Dienstgüte (engl. „Quality of Service“, Abk. QoS) mit welcher die durch den Web Service integrierte Funktionalität bereitgestellt wird [44] [47] [66]. Unternehmen werden nicht bereit sein Web Services in kritischen Geschäftsprozessen zu verwenden, für die der Anbieter nicht bestimmte Eigenschaften garantiert. Nur so kann sichergestellt werden, dass sie bestimmten Anforderungen, z.B. hinsichtlich ihrer Verfügbarkeit, ihrer Antwortzeit, dem verarbeitbaren Volumen, ihrer Sicherheit oder des abgerechneten Nutzungsentgelts genügen. Dienstgütekriterien werden jedoch weder direkt durch die Basisstandards der Web-Service-Technologie, wie SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language) oder UDDI (Universal Description, Discovery and Integration) berücksichtigt, noch durch Standards zur Orchestrierung von Web Services wie BPEL4WS.

Wird ein und dieselbe Funktionalität durch eine Vielzahl von verschiedenen Web Services bereitgestellt, z.B. über entsprechende Verzeichnisse oder Komponentenmarktplätze, so stellen Dienstgütekriterien ein wichtiges Differenzierungsmerkmal dar [53] [37] [79], anhand derer entschieden werden kann, welcher Web Service, gemäß individueller Präferenzen, am besten für einen speziellen Einsatzzweck geeignet ist. Im Rahmen flexibler und dynamisch anpassbarer Geschäftsprozesse [62] [15] sollte die Auswahl geeigneter Web Services zur Durchführung einer im Geschäftsprozess benötigten Funktionalität zur Laufzeit automatisch, ohne manuelle Eingriffe, möglich sein. Hierzu ist es nötig Dienstgütekriterien für Web Services beschreiben und dynamisch nach diesen auswählen zu können [13] [12] [14].

Bei der Ausführung eines Geschäftsprozesses ist es nicht nur wichtig, dass der Geschäftsprozess seine klar definierte Funktionalität erbringt, sondern dass er darüber hinaus auch zusätzlich gewissen nicht-funktionalen Anforderungen genügt. Diese nicht-funktionalen Anforderungen sind entweder durch den Geschäftsprozess an sich, oder dessen Verwendung bedingt. Wird ein Geschäftsprozess beispielsweise Dritten zur Nutzung angeboten, so verlangen die Nutzer meist nicht nur die Garantie, dass der Geschäftsprozess eine bestimmte Funktionalität erbringt, sondern auch, dass er sie mit einer gewissen Dienstgüte erbringt, die z.B. in Form eines „Service Level Agreement“ (SLA) [63] [71] vereinbart wurde. Die Funktionalität eines Geschäftsprozesses wird bei seiner Definition festgelegt, seine nicht-funktionalen Eigenschaften sind jedoch abhängig von den zu seiner Durchführung dynamisch ausgewählten Web Services. Die dynamische Auswahl von Web Services muss daher derart erfolgen, dass nicht nur die gewünschte Funktionalität realisiert wird, sondern dass der Geschäftsprozess zudem bestimmte nicht-funktionale Anforderungen erfüllt.

Existiert eine Vielzahl von Web Services, die zur Ausführung der im Geschäftsprozess geforderten Funktionalitäten verwendet werden können, so existiert meist eine große Menge an Zuordnungsmöglichkeiten von Web Services zu Funktionalitäten, die dazu geeignet sind die festgelegten nicht-funktionalen Anforderungen zu erfüllen. Um

[1] Seit der Version 2.0 auch als WS-BPEL (Web Services Business Process Execution Language) bezeichnet.

aus diesen eine konkrete Zuordnung zur Ausführung des Geschäftsprozesses auswählen zu können, ist es erforderlich Kriterien zu definieren, die es gestatten, eine möglichst optimale Wahl aus der Menge der möglichen Zuordnungen zu treffen. Welche Kriterien hierbei die Optimalität bestimmen, ist im Einzelfall festzulegen. Beispielsweise könnte eine zu erfüllende nicht-funktionale Anforderung (Restriktion) in der Einhaltung einer maximalen Ausführungsdauer des Geschäftsprozesses bestehen, wobei zur Ausführung zugleich eine möglichst preiswerte Zusammenstellung (Optimierungskriterium) von Web Services verwendet werden soll.

Das Problem eine Zuordnung von Web Services zu den Funktionalitäten eines Geschäftsprozesses zu finden, die sowohl alle definierten Restriktionen erfüllt, als auch hinsichtlich aller Optimierungskriterien optimal ist, ist ein NP-schweres Optimierungsproblem [20]. Dies bedeutet, dass es bisher nicht gelungen ist einen Algorithmus anzugeben, der es ermöglicht in einer polynomial beschränkten Rechenzeit eine optimale Lösung des Problems zu ermitteln. Aufgrund der hohen Rechenzeit, die durch exakte Lösungsverfahren, z.B. auf Basis linearer Optimierungsmodelle, benötigt wird um einen konkreten Ausführungsplan eines Geschäftsprozesses zu erstellen, ist ihr Einsatz zur Laufzeit in einer BPE nur bedingt möglich, da hierdurch die Ausführung eines Geschäftsprozesses in inakzeptabler Weise verzögert werden kann [80]. Wird es während der Ausführung eines Geschäftsprozesses nötig einen Ausführungsplan zu revidieren, weil beispielsweise ein Web Service nicht erreichbar ist oder seine Dienstgüteparameter von den vermuteten stark abweichen, so wird eine Neuberechnung des Ausführungsplans (Replanning) nötig, was zu einer weiteren Verzögerung in der Ausführung des Geschäftsprozesses führt.

Wird zur Erstellung eines Ausführungsplans auf die Forderung verzichtet, dass es sich um einen optimalen Ausführungsplan handeln muss, und statt dessen akzeptiert, dass es sich lediglich um einen möglichst guten Ausführungsplan handeln darf, so lässt sich die Berechnungsdauer einer Lösung auf Kosten der Lösungsgüte durch heuristische Verfahren beschleunigen. Heuristische Verfahren stellen stets einen Kompromiss zwischen erreichter Lösungsgüte und investierter Rechenzeit dar.

Gegenstand dieses Technical Reports ist die Untersuchung der Fragestellung, ob die Lösung des beschriebenen Zuordnungsproblems zur optimierten Ausführung eines Geschäftsprozesses durch ein heuristisches Lösungsverfahren hinreichend stark beschleunigt werden kann, sodass die Berechnungszeit auf ein praxistaugliches Maß reduziert werden kann. Hierbei ist eine maximale Reduktion der nötigen Rechenzeit bei gleichzeitig minimaler Reduktion der erzielten Lösungsgüte anzustreben. Im Rahmen unserer Forschungsarbeiten wurde ein entsprechendes heuristisches Lösungsverfahren entworfen, prototypisch implementiert und analysiert.

1.2 Zielsetzung des Technical Reports

1.2.1 Hintergerund

E-Finance Lab

Dieser Technical Report entstand im Rahnem von Forschungsaktivitäten für das „E-Finance Lab Frankfurt am Main“[2]. Das E-Finance Lab ist eine Kooperation zwi-

[2] Siehe auch: http://www.efinancelab.de/

schen der Johann Wolfgang Goethe Universität Frankfurt am Main, der Technischen Universität Darmstadt und einem Netzwerk aus Industriepartnern. Das Ziel des E-Finance Lab ist die Entwicklung industrieller Methoden für den Umgestaltungsprozess der Finanzdienstleistungsindustrie, der allgemeinhin als „Industrialisierung der Finanzindustrie“ bezeichnet wird [59]. Der generelle Ansatz, der hierbei verfolgt wird, ist die Übertragung von Methoden die sich in anderen Industriezweigen, wie z.B. bei der Optimierung der Supply-Chain in der Automobilindustrie, bereits bewährt haben, auf Geschäftsprozesse der Finanzdienstleistungsindustrie. Eine wichtige Herausforderung besteht hierbei in der Entwicklung optimierter Geschäftsprozesse auf Basis einer geeigneten Produktionsinfrastruktur, welche die Anwendung besonders vorteilhafter Sourcing-Strategien ermöglicht.

Die Forschungstätigkeiten des E-Finance Lab verteilen sich auf vier Cluster, wobei jeder Cluster Forschungsarbeiten in einem speziellen Gebiet wahrnimmt. Dieser Technical Report entstand am Cluster II, welcher sich mit der Entwicklung innovativer IT-Architekturen zur Unterstützung von Geschäftsprozessen in der Finanzindustrie beschäftigt. Der Cluster II ist am Fachgebiet „KOM Multimedia Kommunikation“[3] der Technischen Universität Darmstadt ansässig und wird von Prof. Dr.-Ing. R. Steinmetz geleitet. Ein Fokus der Forschungsaktivitäten dieses Clusters besteht in der Untersuchung der Dekomposition von Geschäftsprozessen in einzelne Prozessschritte, welche mittels komponentenbasierter Technologien in verteilten Umgebungen realisiert werden können.

WSQoSX

Im Rahmen vorausgehender Forschungsaktivitäten wurde die Dienstgüte unterstützende Web-Service-Architektur für flexible Geschäftsprozesse *WSQoSX (Web Service Quality of Service Architecture Extension)* [60] entwickelt, welche in der Lage ist, Web Services dynamisch, unter Berücksichtigung von Dienstgütekriterien auszuwählen, in Geschäftsprozesse einzubinden und auszuführen. Die Auswahl eines Web Services erfolgt hierbei zur Ausführungszeit auf Prozessschrittebene, d.h. unmittelbar vor der Ausführung eines Prozessschrittes wird entschieden, welcher konkrete Web Service zur Realisierung der im Prozessschritt benötigten Funktionalität verwendet wird. Funktional gleiche Web Services werden durch das System in Kategorien gruppiert. Zu jeder Kategorie kann eine Präferenzstruktur für die nicht-funktionalen Eigenschaften der Web Services definiert werden. Der Grad der Übereinstimmung der Web Services einer Kategorie mit der Präferenzstruktur wird durch eine sog. Score ausgedrückt. Um einen Prozessschritt auszuführen wird jeweils der Web Service mit der höchsten Score in der entsprechenden Kategorie aufgerufen. Da bei der Auswahl der zu verwendenden Web Services ausschließlich der aktuelle Prozessschritt, unabhängig von bereits ausgeführten und zukünftig auszuführenden Prozessschritten betrachtet und optimiert wird, findet, bezogen auf den gesamten Geschäftsprozess, lediglich eine lokale Optimierung statt.

Diese lokale Optimierung auf der Ebene einzelner Prozessschritte weist jedoch zwei entscheide Nachteile auf. Einerseits führt die Optimierung einzelner Prozessschritte nicht zwingendermaßen zu einer optimalen Ausführung des Prozesses als Ganzes und andererseits lassen sich durch eine lokale Optimierung keine Eigenschaften für den Gesamtprozess erzwingen, da lediglich einzelne Prozessschritte, nicht jedoch der Prozess als Ganzes betrachtet und optimiert wird.

[3] Siehe auch: http://www.kom.tu-darmstadt.de/

Die Untersuchung einer Möglichkeit Geschäftsprozesse als Ganzes optimieren und gewissen Restriktionen unterwerfen zu können, ist daher eine konsequente Weiterentwicklung des Ansatzes der Dienstgüte unterstützenden Web-Service-Architektur für flexible Geschäftsprozesse, die mit WSQoSX begonnen wurde.

1.2.2 Problemstellung

Gegenstand dieses Technical Reports ist die Vorstellung und Analyse eines heuristischen Verfahrens zur dienstgütebasierten Optimierung flexibler Geschäftsprozesse auf Basis von Web Services. Unter Geschäftsprozess wird in diesem Zusammenhang ganz allgemein eine mittels Kontrollkonstrukten strukturierte Ausführungsfolge von Web Services verstanden. Hierbei werden keine konkreten Web Services zur Ausführung spezifiziert, sondern abstrakte Dienste verwendet, welche die Funktionalität beschreiben, die an einer Stelle des Geschäftsprozesses (Prozessschritt) durch einen Web Service zu erbringen ist. Für jeden abstrakten Dienst existiert eine Menge konkreter Dienste (Web Services), welche in der Lage sind die geforderte Funktionalität zu erbringen. Zur Durchführung eines Prozessschrittes stehen daher gemäß des abstrakten Dienstes verschiedene alternative, konkrete Dienste (Prozessschrittkandidaten) zur Verfügung. Prozessschrittkandidaten sind bezüglich ihrer Funktionalität identisch, können sich jedoch hinsichtlich ihrer nicht-funktionalen Eigenschaften, z.B. in ihrer Dienstgüte (vgl. Kapitel 2.3), unterscheiden.

Um einen mittels abstrakter Dienste definierten Geschäftsprozess ausführen zu können, ist ein Ausführungsplan zu erstellen, der jedem Prozessschritt einen konkreten Prozessschrittkandidaten zuordnet, durch dessen Ausführung die Funktionalität des Prozessschrittes erbracht wird. Je nachdem, welche Prozessschrittkandidaten gewählt werden, besitzt der Geschäftsprozess bei seiner Ausführung verschiedene nichtfunktionale Eigenschaften, wie z.B. benötigte Ausführungszeit oder verursachte Kosten. Durch ein Optimierungsverfahren ist hierbei sicherzustellen, dass der zu einem Geschäftsprozess erzeugte Ausführungsplan gewisse Restriktionen bezüglich nichtfunktionaler Eigenschaften einhält und zugleich hinsichtlich gewisser nichtfunktionaler Eigenschaften optimiert wird. Ein Beispiel hierfür ist die Erstellung eines Ausführungsplans derart, dass eine definierte maximale Ausführungszeit nicht überschritten wird und die durch die Ausführung entstehenden Kosten minimal sind. Um einen derartigen Ausführungsplan erstellen zu können ist die Lösung eines NP-schweren Optimierungsproblems (vgl. Kapitel 2.2) nötig.

Eine naive Methode zur Lösung des Optimierungsproblems besteht in der vollständigen Enumeration aller möglichen Ausführungspläne und der Auswahl des optimalen Ausführungsplans aus der Menge jener Ausführungspläne, die alle Restriktionen erfüllen. Dieses Vorgehen stößt jedoch schnell bei steigender Anzahl von Prozessschritten und alternativen Web Services zur Realisierung eines Prozessschrittes an seine Grenzen. Die Optimierung eines sequentiellen Geschäftsprozesses, in welchem 20 Prozessschritte hintereinander ausgeführt werden und für jeden Prozessschritt 15 verschiedene Web Services zur Verfügung stehen, würde die Betrachtung von 20^{15} verschiedenen Ausführungsplänen bedeuten. Selbst wenn die Erstellung und Bewertung von 1000 Ausführungsplänen in einer Millisekunde möglich wäre, würde die Ermittlung des optimalen Ausführungsplans über eine Million Jahre in Anspruch nehmen.

Wird das geschilderte Problem als ein lineares Optimierungsproblem formuliert und das entstehende Modell durch Verfahren wie z.B. dem Simplex-Algorithmus (vgl.

Kapitel 2.2.3) und „Branch & Bound“ (vgl. Kapitel 2.2.4) gelöst, so ist es möglich die Berechnungsdauer auf einige Sekunden oder wenige Minuten zu reduzieren. Doch selbst eine Berechnungsdauer im Minutenbereich stellt für eine Ausführungsoptimierung zur Laufzeit in einer BPE eine in der Praxis nicht tolerierbare Verzögerung der Ausführung dar. Ergeben sich während der Ausführung des Geschäftsprozesses Veränderungen der Dienstgüteeigenschaften der zur Verfügung stehenden Web Services, so erfordert dies zudem eine Neuberechnung des optimalen Ausführungsplans, was jeweils eine zusätzliche deutliche Verzögerung der Ausführung bedeuten würde. Soll durch die Ausführungsoptimierung die Ausführungszeit des Geschäftsprozesses optimiert werden, so ist es durchaus möglich, dass in einem solchen Fall die Ermittlung des schnellstmöglichen Ausführungsplans für den Geschäftsprozess mehr Zeit in Anspruch nimmt, als die Ausführung des langsamsten möglichen Ausführungsplans. Das Konzept der Ausführungsoptimierung würde in diesem Fall ad absurdum geführt werden.

Es bedarf daher eines heuristischen Verfahrens (vgl. Kapitel 2.2.6) um die Ausführungsoptimierung von Geschäftsprozessen in einer für den Einsatz in einer BPE akzeptablen Zeit und Qualität vornehmen zu können. Das Ziel dieses Technical Reports ist nicht die Vorstellung der bestmöglichen aller denkbaren Heuristiken, sondern vielmehr die Untersuchung einer entwickelten Heuristik vor dem Hinterund der Fragetsellung ob diese eine Heuristik, oder Heuristiken im Allgemeinen, prinzipiell für den Einsatz zur Verwendung innerhalb einer BPE geeignet ist und welche Charakteristiken bezüglich Rechenzeit und Lösungsgüte in etwa erreicht werden können.

Um die Komplexität dieser Untersuchung auf ein handhabbares Maß zu reduzieren, werden lediglich sequentielle Geschäftsprozesse betrachtet, d.h. Geschäftsprozesse die nur durch das Kontrollkonstrukt der sequentiellen Ausführung aufgebaut werden und keine weiteren Kontrollkonstrukte wie z.B. Verzweigungen, Schleifen oder Parallelausführungen enthalten. Die Optimierung von Geschäftsprozessen mit weiteren Kontrollkonstrukten, lässt sich jedoch durch Anwendung geeigneter Verfahren[4] auf die Optimierung einzelner sequentieller Teilprozesse zurückführen.

Im Rahmen dieses Technical Reports wurden u.a. folgende Fragestellungen betrachtet:

Durch welches mathematische Modell lässt sich das zu lösende Optimierungsproblem beschreiben? Welche grundlegenden Arten von Dienstgütekriterien können unterschieden werden und wie lässt sich eine variable Anzahl dieser Kriterien möglichst generisch in das Modell integrieren? Wie kann festgelegt werden, nach welchen Kriterien der Ausführungsplan optimiert wird und welche Restriktionen er erfüllen muss? Mithilfe welcher Methoden lässt sich das Optimierungsproblem exakt lösen? Welche Heuristik ist dazu geeignet das Optimierungsproblem mit einem geringeren Aufwand an Rechenzeit zu lösen? Wie ist das Verhältnis der benötigten Rechenzeit zwischen exakter Lösung und der heuristischen Methode? Welche Lösungsgüte kann durch die Heuristik erreicht werden? Wie hängen Lösungsgüte und benötigte Rechenzeit von dem formulierten Optimierungsproblem ab? Wie skaliert das heuristische Lösungsverfahren? Welche Problemgrößen können mit dem heuristischen Lösungsverfahren

[4] Zeng et al. beschreiben in [79] und [80] beispielsweise ein entsprechendes Verfahren. Ein Geschäftsprozess wird in alle möglichen (sequentiellen) Ausführungspfade zerlegt und diese einzeln optimiert. Anschließend werden die Einzelergebnisse wieder zu einem Gesamtergebnis aggregiert.

noch sinnvoll bearbeitet werden? Kann eine Berechnungsdauer erreicht werden, die den Einsatz einer Ausführungsoptimierung in einer BPE ermöglichen würde?

1.3 Aufbau des Technical Reports

Nachdem im ersten Kapitel in den behandelten Themenkomplex eingeführt und die Zielsetzung des Technical Reports erläutert wurde, werden im zweiten Kapitel die nötigen Grundlagen für eine eingehendere Betrachtung des behandelten Themas erläutert. Im dritten Kapitel wird anschließend das, dem Optimierungsansatz zugrundeliegende, mathematische Modell, sowie die zu dessen Lösung verwendete Heuristik beschrieben. Das vierte Kapitel erörtert eine prototypische Implementierung der entwickelten Heuristik. Im fünften Kapitel erfolgt eine Analyse der Heuristik anhand des entwickelten Prototypen, bevor im sechsten Kapitel aktuelle Arbeiten zum behandelten Themenkomplex vorgestellt werden. Das siebte Kapitel bietet eine abschließende Zusammenfassung des Technical Reports und einen Ausblick auf zukünftige Forschungsaktivitäten.

2 Grundlagen

Bevor in den nachfolgenden Kapiteln auf die konkret entwickelte Heuristik zur Ausführungsoptimierung von Geschäftsprozessen eingegangen wird, werden in diesem Kapitel wichtige Grundlagen für das weitere Verständnis geschaffen. Nachdem zunächst kurz auf den Begriff des Geschäftsprozesses eingegangen wird, erfolgt eine an [26] orientierte Einführung in den Themenbereich der Optimierungsprobleme. Der Fokus liegt hierbei auf linearen Optimierungsproblemen und ihren mathematischen Modellen. Zur exakten Lösung von linearen Optimierungsproblemen wird der Simplex-Algorithmus und zur Lösung ganzzahliger linearer Optimierungsprobleme das Branch&Bound-Verfahren vorgestellt. Darüber hinaus wird das Prinzip der näherungsweisen Lösung von Optimierungsproblemen mittels Heuristiken erläutert. Anschließend wird der Begriff der Dienstgüte erörtert. Es erfolgt eine kurze Darstellung einiger Aspekte von Dienstgüte in Netzwerken, verschiedener Dienstgütetypen, Mechanismen zur Durchsetzung von Dienstgüte und Dienstgüte in Bezug auf Web Services.

2.1 Geschäftsprozesse

Der Begriff Geschäftsprozess ist in der Literatur nicht einheitlich definiert. Wyssusek führt beispielsweise in [75] aus: „In der Literatur hat sich bislang noch keine einheitliche Definition für den Begriff des Geschäftsprozesses herausgebildet. [...] Im Kontext der Wirtschaftsinformatik wird [...] unter einem Geschäftsprozeß die inhaltlich abgeschlossene, zeitlich-sachlogische Abfolge von Funktionen verstanden, die zur Bearbeitung eines für die Leistungserbringung des Unternehmens relevanten Objekts erforderlich sind."

Im Folgenden soll unter einem *Geschäftsprozess* (engl. business process) ganz allgemein eine Menge logisch in Beziehung stehender Aktivitäten verstanden werden, die ausgeführt werden um ein genau definiertes Ergebnis zu erzielen, welches zur Leistungserstellung und/oder dem Erfolg eines Unternehmens unmittelbar oder mittelbar beiträgt. Geschäftsprozesse können sich über das Unternehmen hinaus erstrecken und Aktivitäten von Kunden, Lieferanten und Partnern einschließen, wodurch an einem Geschäftsprozess eine unterschiedliche Anzahl unabhängiger, autonomer Teilnehmer beteiligt sein kann.

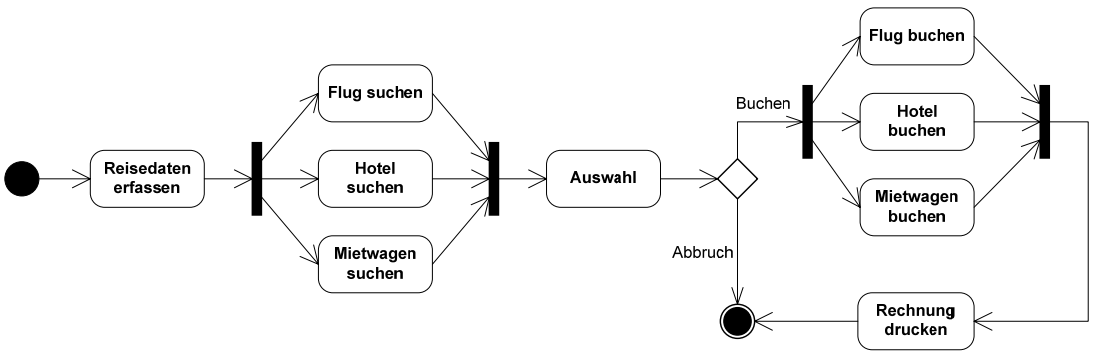

Abbildung 1 - Darstellung eines vereinfachten Geschäftsprozesses als Zustandsdiagramm

In Abbildung 1 wird ein stark vereinfachter Geschäftsprozess, wie er in einem Reisebüro auftreten könnte, in Form eines UML-Zustandsdiagramms dargestellt. Nachdem die Reisedaten des Kunden erfasst wurden, wird parallel nach passenden Flügen, Ho-

tels und Mietwagen gesucht. Aus den verfügbaren Flügen, Hotels und Mietwagen kann ein konkretes Reisepaket zusammengestellt werden oder der Vorgang abgebrochen werden. Wird der Vorgang fortgesetzt, so werden parallel der gewählte Flug, das gewählt Hotel und der gewählte Mietwagen gebucht und anschließend dem Kunden eine entsprechende Rechnung ausgedruckt.

Die Suche und das Buchen von Flügen, Hotels und Mietwagen schließt hierbei Aktivitäten externer Partner ein, wie z.B. Fluggesellschaften, Hotels und Autovermietungen. Zur gemeinschaftlichen Durchführung des Geschäftsprozesses ist es nötig, dass diese Partner miteinander kommunizieren, Informationen austauschen und sich koordinieren. Soll die Ausführung eines Geschäftsprozesses mittels IT-Systemen automatisiert werden, so ist es nötig eine einheitliche Infrastruktur für die Kommunikation der verschiedenen, evtl. heterogenen IT-Systeme der Partner zu schaffen. Die Technologie der Web Services bietet hier die Möglichkeit proprietäre Schnittstellen und Formate durch eine einheitliche Infrastruktur für den Nachrichtenaustausch zu ersetzen. Aktivitäten oder Teilaktivitäten von Geschäftsprozessen können mittels Web Services abgebildet und damit in einheitlicher und interoperabler Weise als Dienste einer „Service Oriented Architecture“ (SOA) zur Verfügung gestellt werden.

Das reine Bereitstellen solcher Dienste mittels Web Services durch die Partner reicht jedoch für die Durchführung von Geschäftsprozessen nicht aus. In einer weiteren Stufe müssen diese Dienste durch eine definierte Geschäftslogik (engl. business logic) zu einem Geschäftsprozess orchestriert und dessen Ausführung über alle beteiligten Partner hinweg koordiniert werden. Eine geeignete Beschreibungssprache für Geschäftsprozesse auf Basis von Web Services ist beispielsweise BPEL4WS [8] [36]. Die Ausführung eines derart definierten Geschäftsprozesses obliegt einer *„Business Process Engine“ (BPE)*. Im Beispiel könnte so die Suche und das Buchen von Produkten der jeweiligen Partner durch Web Services durchgeführt werden, die vom Reisebüro vollständig automatisiert in Anspruch genommen werden könnten. Hierdurch wäre es dem Reisebüro beispielsweise zusätzlich möglich den Geschäftsprozess seinen Kunden über eine Homepage anzubieten und den Kunden dadurch die Prozessschritte der Erfassung seiner Wünsche und die Auswahl der Teilleistungen der Reise vollständig selbständig durchführen zu lassen.

Gehört die Durchführung eines Geschäftsprozesses nicht zu den Kernkompetenzen eines Unternehmens, so kann es vorteilhaft sein, diesen Geschäftsprozess an externe Dienstleister zu übertragen, die eine besondere Exzellenz in der Durchführung derartiger Geschäftsprozesse besitzen. Die Auslagerung von Geschäftsprozessen wird als *„Business Process Outsourcing“ (BPO)* [54] [29] bezeichnet. Durch die Auslagerung von Geschäftsprozessen ist es möglich an der Exzellenz externer Dienstleister zu partizipieren und hierdurch Kosten- und Qualitätsvorteile zu realisieren, die dem Unternehmen Wettbewerbsvorteile verschaffen können. Wird die Buchung von Reisen beispielsweise in einem Unternehmen nur als Unterstützungsleistung der Außendienstmitarbeiter angeboten, so empfiehlt es sich evtl. die Reisebuchungen an einen spezialisierten Dienstleister auszulagern, der sowohl durch Skaleneffekte als auch über spezielle Konditionen, die ihm als Großkunde bei Fluggesellschaften, Hotels und Autovermietungen eingeräumt werden, erhebliche Kostenvorteile erzielen kann. Wird die Kommunikation mit dem externen Dienstleister transparent auf Basis einer SOA realisiert, so sind die entstehenden Schnittstellenkosten minimal.

Wird ein und dieselbe Funktionalität, die in einem Geschäftsprozess benötigt wird, durch Web Services mehrerer Dienstleister angeboten, so ist eine Auswahl des konkret zu verwendenden Web Services anhand der nicht-funktionalen Eigenschaften der alternativen Web Services zu treffen. Bietet das Reisebüro beispielsweise über seine Homepage ein Online-Bezahlverfahren an, so möchte es vielleicht zuvor eine Überprüfung der Bonität des Kunden, beispielsweise durch eine Schufa-Abfrage realisieren. Existieren nun verschiedene Web Services zur Bonitätsprüfung von verschiedenen Anbietern, so könnte durch entsprechende Mechanismen stets automatisch der preiswerteste Anbieter eingebunden werden.

Besteht ein Geschäftsprozess aus vielen Prozessschritten, die jeweils durch eine Vielzahl verschiedener Anbieter, bzw. Web Services realisiert werden können, so besteht eine besondere Herausforderung darin, in einem dynamischen Umfeld wie dem Internet, vor der Ausführung eines solchen Geschäftsprozesses stets die Web Services zur Durchführung der Prozessschritte zu bestimmen, die zu einer optimalen Ausführung des Geschäftsprozesses bei gleichzeitiger Einhaltung gewisser Restriktionen führen. Die Lösung eines solchen Problems bedarf der Lösung eines Optimierungsproblems, auf welche im nächsten Abschnitt näher eingegangen wird.

2.2 Optimierungsprobleme

Im Rahmen von Planungs- oder Entscheidungsprozessen besteht oftmals ein Problem darin, eine Wahl aus einer Menge von alternativen Handlungs- oder Lösungsmöglichkeiten zu treffen. Die Auswahl erfolgt hierbei zumeist zielorientiert, d.h. anhand von Überlegungen, welche der Alternativen für das Erreichen wünschenswerter Zustände am besten geeignet, bzw. optimal ist. Derartige Probleme werden als Optimierungsprobleme bezeichnet.

Generell werden durch den Begriff *Optimierungsproblem* alle jene Entscheidungsprobleme bezeichnet, bei denen die im Hinblick auf die zu verfolgenden Ziele günstigste realisierbare Lösung auszuwählen ist. Ein *Optimierungsmodell* ist die formale Darstellung eines Optimierungsproblems. Es enthält neben allen bei der Entscheidungsfindung zu berücksichtigenden Ursache-Wirkungs-Zusammenhängen Zielrelationen zur Bewertung und Auswahl von Handlungsmöglichkeiten.

Um Optimierungsprobleme strukturiert und effizient lösen zu können, erfolgt die Darstellung des entsprechenden Optimierungsmodells in Form eines *quantitativen (mathematischen) Modells*, in welchem sämtliche im Modell abgebildeten Aspekte des realen Entscheidungsproblems durch kardinal messbare (metrische) Informationen, d.h. durch reelle Zahlen, beschrieben werden. Elemente des realen Systems werden durch Daten(parameter) und Variablen dargestellt und in Form von Gleichungen oder Ungleichungen auf strukturerhaltende Weise miteinander verknüpft. Quantitative Modelle können mit mathematischen Methoden ausgewertet werden um Lösungen eines Entscheidungsproblems zu ermitteln. Eine auf die Entwicklung und den Einsatz quantitativer Modelle und Methoden zur Entscheidungsunterstützung spezialisierte Disziplin ist das *Operations Research (OR)* [26].

2.2.1 Allgemeines mathematisches Modell

Optimierungsmodelle bestehen aus einer *Menge von Alternativen (zulässige Lösungen)* und mindestens einer zu maximierenden oder minimierenden *Zielfunktion*, mit

deren Hilfe eine oder mehrere *optimale Lösungen* identifiziert werden können. Die Menge der zulässigen Lösungen kann explizit vorgegeben werden oder implizit in Form eines Systems von *Restriktionen (Nebenbedingungen)* definiert sein.

Das mathematische Modell eines allgemeinen Optimierungsproblems mit einfacher Zielsetzung lässt sich wie folgt formulieren:

$$\text{Maximiere (oder minimiere) } z = F(\vec{x}) \tag{2.1}$$

unter den Nebenbedingungen

$$g_i(\vec{x}) \begin{Bmatrix} \geq \\ = \\ \leq \end{Bmatrix} 0 \qquad \forall i = 1,...,m \tag{2.2}$$

$$\vec{x} \in R_+^n \text{ oder } \vec{x} \in Z_+^n \text{ oder } \vec{x} \in B^n \tag{2.3}$$

$\vec{x}$ ist ein Variablenvektor mit n Komponenten (Variablen x_1, x_2, ..., x_n). (2.1) entspricht der Zielfunktion, die maximiert oder minimiert werden soll und deren Zielfunktionswert abhängig vom Wert der Variablen des Variablenvektors $\vec{x}$ ist. (2.2) definiert ein System von m Gleichungen und/oder Ungleichungen (Restriktionensystem), welches die Belegungsmöglichkeit der Variablen beschränkt. (2.3) gibt Beispiele für die Definition des Wertebereichs der Variablen. $\vec{x} \in R_+^n$ definiert positive reelle Zahlen, $\vec{x} \in Z_+^n$ positive ganze Zahlen und $\vec{x} \in B^n$ Binärzahlen (0 oder 1) als Wertebereich für alle Variablen. Entgegen diesen Beispielen müssen nicht alle Variablen denselben Wertebereich besitzen und die möglichen Wertebereiche sind nicht auf die im Beispiel genannten beschränkt.

Durch das Restriktionensystem (2.2) und die Festlegung des Wertebereichs (2.3) wird implizit die Menge der zulässigen Lösungen definiert. Eine Belegung der Komponenten des Variablenvektors $\vec{x}$ aus dem Wertebereich (2.3) die alle Restriktionen (2.2) erfüllt, wird zulässige Lösung genannt. Die Menge aller zulässigen Lösungen wird mit X bezeichnet. Eine zulässige Lösung wird optimal genannt und mit $\vec{x}^*$ bezeichnet, wenn es keine andere zulässige Lösung gibt, die hinsichtlich der Zielfunktion besser bewertet wird. Im Falle einer zu minimierenden Zielfunktion bedeutet dies, dass keine andere Lösung einen niedrigeren Zielfunktionswert als $F(\vec{x}^*)$ erzeugt, bzw. im Falle einer zu maximierenden Zielfunktion, dass keine andere Lösung einen höheren Zielfunktionswert als $F(\vec{x}^*)$ erzeugt. Die Menge aller optimalen Lösungen wird mit X^* bezeichnet. Es ist möglich, dass ein Optimierungsproblem keine, eine oder mehrere optimale Lösungen besitzt.

2.2.2 Klassifikation von Optimierungsmodellen

Optimierungsmodelle können nach [26] vor allem nach folgenden Gesichtspunkten klassifiziert werden:

- Hinsichtlich des Informationsgrades in deterministische und stochastische Modelle. Bei *deterministischen Modellen* werden die Parameter der Zielfunktion(en) sowie der Nebenbedingungen als bekannt vorausgesetzt. Ist jedoch mindestens ein Parameter als Zufallsvariable zu interpretieren, so liegt ein *stochastisches Modell* vor. Deterministische Modelle dienen der Entscheidungs-

findung bei Sicherheit, stochastische Modelle der Entscheidungsfindung bei Risiko.

- Hinsichtlich der Anzahl der Zielfunktionen in Modelle mit einer und in Modelle mit mehreren Zielfunktionen. Drückt jede Zielfunktion ein eigenes Ziel aus, welches durch die Optimierung erreicht werden soll, so ist es meist erst nach der Kombination der einzelnen Ziele in einer gemeinsamen Zielfunktion möglich, diese Modelle zu optimieren. Diese Notwendigkeit wird besonders im Falle des Vorliegens konkurrierender Ziele deutlich, bei dem sich der Zielerreichungsgrad des einen Ziels in dem Masse verschlechtert, wie sich der Zielerreichungsgrad eines anderen Ziels verbessert.
- Hinsichtlich des Typs der Zielfunktion(en) und Nebenbedingungen in lineare Modelle mit reellen Variablen, lineare Modelle mit ganzzahligen oder Binärvariablen, nichtlineare Modelle, usw. usf..
- Hinsichtlich der Lösbarkeit in Modelle, die in Abhängigkeit ihrer Größe mit polynomialen Rechenaufwand lösbar sind, und solche, für die bislang kein Verfahren angebbar ist, das jede Problemgröße mit polynomialen Aufwand zu lösen gestattet. Letztgenannte Gruppe wird als Gruppe der *NP-schweren* Probleme bezeichnet.

Im Rahmen dieses Technical Reports ist vor allem eine Klassifizierung hinsichtlich des Typs der Zielfunktion(en) und Nebenbedingungen relevant, da dies maßgeblich darüber bestimmt, ob effiziente Algorithmen zur Lösung des Optimierungsproblems vorliegen. Für die Betrachtungen im Rahmen dieses Technical Reports ist eine Unterscheidung von linearen Optimierungsmodellen, ganzzahligen linearen Optimierungsmodellen und nichtlinearen Optimierungsmodellen ausreichend. Bei allen folgenden Betrachtungen wird von Optimierungsmodellen mit nur einer Zielfunktion ausgegangen.

2.2.3 Lineare Optimierungsmodelle

Ein *lineares Optimierungsmodell* besitzt eine lineare Zielfunktion und lineare Nebenbedingungen. Die Variablen dürfen reelle Werte annehmen, wobei zumeist jedoch noch gefordert wird, dass diese nicht negativ sind. Ein Optimierungsproblem, das durch ein lineares Optimierungsmodell abgebildet wird, wird als *lineares Optimierungsproblem (LOP)* bezeichnet. Bei der Erzeugung und Lösung von LOPen wird auch von der *linearen Programmierung (LP)* gesprochen.

Aufgrund der Linearität der Zielfunktion und der Nebenbedingungen lässt sich jedes LOP durch Umformungen in die folgende Form bringen, die als Normalform eines LOP bezeichnet wird.

$$\text{Maximiere } F(x_1,...,x_n) = \sum_{j=1}^{n} c_j x_j \tag{2.4}$$

unter den Nebenbedingungen

$$\sum_{j=1}^{n} a_{i,j} x_j = b_i \qquad \forall i = 1,...,m \tag{2.5}$$

$$x_j \geq 0 \qquad \forall j = 1,...,n \tag{2.6}$$

In Matrixschreibweise lässt sich das LOP wie folgt aufschreiben:

Maximiere $F(\vec{x}) = \vec{c}^T \vec{x}$ (2.7)

unter den Nebenbedingungen

$A\vec{x} = \vec{b}$ (2.8)

$\vec{x} \geq \vec{0}$ (2.9)

Hierbei sind $\vec{c}$ und $\vec{x}$ n-dimensionale Vektoren, $\vec{b}$ ist ein m-dimensionaler Vektor und A eine (m x n)-Matrix.

Für LOPe existiert mit dem *Simplex-Algorithmus* ein universelles Lösungsverfahren. Der Simplex-Algorithmus löst ein lineares Optimierungsproblem nach endlich vielen Schritten exakt oder stellt die Unlösbarkeit des Problems fest. Der Name Simplex-Algorithmus ist von der Bezeichnung Simplex für ein durch n+1 Punkte des R^n aufgespanntes konvexes Polyeder abgeleitet. Geometrisch betrachtet liegen alle optimalen Lösungen eines LOPs am Rand des durch das Restriktionensystem (2.8) und (2.9) definierten Lösungsraums X. Um eine optimale Lösung aufzufinden, besucht der Simplex-Algorithmus nach einer gewissen Strategie Ecken des Lösungsraums. Ein anderes prinzipielles Vorgehen bei der Lösung von LOPen verfolgen *Interior-Point-Methoden*, die ausgehend von einer im Inneren des Lösungsraums liegenden Lösung eine optimale Lösung suchen. Bekannte Lösungsmethoden dieser Art sind die Ellipsoid-Methode von Khachijan [40] und die projektive Methode von Karmarkar [38], deren Laufzeit im Gegensatz zu der des Simplex-Algorithmus polynomiell beschränkt ist. Diese sind dem Simplex-Algorithmus zwar hinsichtlich ihres Laufzeitverhaltens im ungünstigsten Fall überlegen, nicht jedoch im allgemein zu erwartenden durchschnittlichen Fall. Von jeder Lösungsmethode existieren verschiedene Varianten mit zum Teil auf spezielle Strukturen von LOPen optimierten Strategien. Auf die einzelnen Lösungsalgorithmen soll hier jedoch nicht näher eingegangen werden.

Der Simplex-Algorithmus ist eines der leistungsfähigsten Verfahren zur Lösung von LOPen in der Praxis. Eine Untersuchung des Laufzeitverhaltens des Simplex-Algorithmus kann [50] entnommen werden. Die Laufzeit des Simplex-Algorithmus wächst in der Regel nur linear mit der Anzahl n der Variablen und der Anzahl m der Restriktionen. In ungünstigen Fällen kann die Laufzeit jedoch auch exponentiell anwachsen. Klee und Minty konstruieren in [41] mit den nach ihnen benannten Klee-Minty-Würfeln Fälle, in denen der Simplex-Algorithmus ein exponentielles Laufzeitverhalten besitzt. Klee-Minty-Würfel sind n-dimensionale Polytope, in denen ein aufsteigender Pfad durch alle 2^n Ecken existiert. Wird durch das Modell eines LOP, geometrisch interpretiert, ein Klee-Minty-Würfel konstruiert, so kann nicht garantiert werden, dass der Simplex-Algorithmus die optimale Lösung des LOP ermitteln kann, ohne alle 2^n Ecken zu besuchen. Eine genauere Untersuchung dieses Sachverhaltes kann [51] entnommen werden.

2.2.4 Ganzzahlige lineare Optimierungsmodelle

Ein *ganzzahliges lineares Optimierungsmodell* ist ein lineares Optimierungsmodell in dem mindestens eine Variable nur ganzzahlige Werte annehmen darf. Werden nur einige Variablen auf den Wertebereich der ganzen Zahlen beschränkt, so spricht man auch von *gemischt ganzzahligen linearen Optimierungsmodellen*. Das zugehörige Optimierungsproblem wird als *(gemischt) ganzzahliges lineares Optimierungsprob-*

lem (GLOP) bezeichnet. Bekannte GLOPe sind z.B. das Knapsack-Problem[5] oder das Traveling-Salesman-Problem[6] (TSP). Wird ein GLOP durch Fallenlassen der Ganzzahligkeitsrestriktion in ein LOP überführt, so spricht man von einer *LP-Relaxation*.

Da aufgrund der Ganzzahligkeitsforderung optimale Lösungen nicht mehr am Rand, bzw. in den Ecken des Lösungsraumes X liegen müssen, lässt sich der Simplex-Algorithmus nicht mehr zur Lösung von GLOPen verwenden. Die wichtigsten Verfahren, die eine exakte Lösung von GLOPen in endlich vielen Schritten gewährleisten können, sind unterteilbar in Schnittebenenverfahren und Entscheidungsbaumverfahren. *Schnittebenenverfahren* [56] lösen das durch LP-Relaxation gewonnene LOP und fügen diesem sukzessive zusätzliche Nebenbedingungen hinzu um die Ganzzahligkeitsforderung des GLOP zu erfüllen. *Entscheidungsbaumverfahren* führen eine möglichst begrenzte Enumeration des Lösungsraumes durch, wobei versucht wird so früh wie möglich größtmögliche Teilmengen des Lösungsraums zu identifizieren, die nicht mehr zu einem optimalen Ergebnis führen können, um diese von der weiteren Enumeration des Lösungsraums ausschließen zu können.

Ein sehr bekanntes und universell einsetzbares Entscheidungsbaumverfahren ist *„Branch and Bound" (B&B)*. B&B beinhaltet die beiden Lösungsprinzipien Branching und Bounding.

Das *Branching* zerlegt ein zu lösendes Problem P_0 vollständig in k Teilprobleme $P_1,...,P_k$, d.h. die Vereinigung der Lösungsmengen der Teilprobleme ergibt wieder die Lösungsmenge des Ausgangsproblems P_0 (2.10). Die Zerlegung in Teilprobleme ist möglichst so vorzunehmen, dass eine vollständig disjunkte Zerlegung erreicht wird, d.h. dass der paarweise Schnitt der Lösungsmengen aller Teilprobleme leer ist (2.11). Die Teilprobleme $P_1,...,P_k$ lassen sich analog dieses Verfahrens wiederum zerlegen, wodurch ein (Lösungs-)Baum des Problems mit der Wurzel P_0 entsteht.

$$X(P_0) = \bigcup_{i=1}^{k} X(P_i) \tag{2.10}$$

$$X(P_i) \cap X(P_j) = \varnothing \quad \forall i \neq j \tag{2.11}$$

Das *Bounding* beschreibt ein Prinzip zur Beschränkung des Verzweigungsprozesses des Branchings im Rahmen des Lösungsprozesses. Hierzu werden Schranken für Zielfunktionswerte berechnet, mit deren Hilfe entschieden werden kann, ob Teilprobleme verzweigt werden müssen, oder nicht. Zu Beginn des Verfahrens ist die untere Schranke $\underline{F}$ des Zielfunktionswertes des Problems P_0 unbekannt und wird daher auf $-\infty$ gesetzt. Nun wird das Problem P_0 durch das Braching zerlegt. Nach jeder Zerlegung eines Problems in Teilprobleme, kann für jedes Teilproblem P_i, z.B. mittels einer LP-Relaxation, ein vereinfachtes Problem P_i' formuliert werden, welches durch ein exaktes Verfahren, z.B. dem Simplex-Algorithmus, gelöst werden kann. Der hierbei ermittelte Zielfunktionswert der optimalen Lösung von P_i' stellt eine obere Grenze $\overline{F_i}$ für den Zielfunktionswert von P_i dar. Ein Problem P_i heißt *ausgelotet* und braucht

[5] Gegeben sind Gegenstände mit definiertem Nutzen und Volumen, sowie ein Behälter mit begrenztem Volumen (Knapsack). Finde die Menge von Gegenständen, die den Behälter nutzenmaximal füllen.

[6] Gegeben ist eine (n x n)-Matrix mit den Entfernungen zwischen n Städten. Ein Handlungsreisender möchte, beginnend in einem Ort und am Ende dorthin wieder zurückkehrend, alle n Städte besuchen. Finde eine Rundreise durch die n Städte, für welche die insgesamt zurückgelegte Strecke minimal ist.

nicht weiter betrachtet, d.h. nicht weiter verzweigt zu werden, falls einer der folgenden, sich gegenseitig ausschließenden Fälle eintritt:

- $X(P_i') = \varnothing$. Das Teilproblem P_i' besitzt keine zulässige Lösung.
- $\overline{F_i} \leq \underline{F}$. Die optimale Lösung des Teilproblems P_i' ist schlechter als die bisher beste bekannte Lösung des Gesamtproblems.
- $\overline{F_i} > \underline{F}$ und die optimale Lösung des Teilproblems P_i' ist zulässig für P_i. Da die optimale Lösung von P_i' auch zulässig für P_0 ist und da durch $\overline{F_i} > \underline{F}$ diese Lösung zugleich auch besser als die bisher beste bekannte Lösung für P_0 ist, wird $\underline{F}$ auf $\overline{F_i}$ gesetzt.

Ein Teilproblem wird weiter zerlegt, falls es nicht auslotbar ist. Das Verfahren endet, sobald alle Teilprobleme ausgelotet sind. Die exakte optimale Lösung des Ursprungsproblems P_0 ist die Lösung des Teilproblems P_i', die im Verlauf des Verfahrens zur letzten Anpassung der unteren Schranke $\underline{F}$ führt. Wird die unteren Schranke $\underline{F}$ im Verlauf des Verfahrens nie angepasst, so ist das Ursprungsproblem nicht lösbar.

B&B ist ein *Metasuchverfahren*, da es lediglich ein Prinzip zur Reduzierung des effektiv durchsuchten Problemlösungsraumes beschreibt. B&B legt nicht explizit fest in welcher Art und Weise ein Problem in Teilprobleme zerlegt wird, wie eine Relaxation der Teilprobleme erfolgt und wie die relaxierten Teilprobleme gelöst werden. Die Effizienz eines konkreten B&B-Verfahrens hängt davon ab, ob es durch ein geschicktes Branching gelingt, möglichst viele und große Teilprobleme vorzeitig auszuloten. Da das Ausloten von Teilproblemen auf einen Vergleich von Bounds (Schranken) zurückgeführt wird, hängt die Effizienz auch von der Qualität des Bounding-Verfahrens ab. Für allgemeine GLOPe ist es nicht möglich ein B&B-Verfahren anzugeben, welches eine polynomiell beschränkte Laufzeit aufweist.

Handelt es sich nicht um spezielle GLOPe, bei deren Lösung eine spezielle Struktur oder Strategie zusätzlich ausgenutzt wird, ist die Lösung von GLOPen ein NP-schweres Problem, d.h. es lässt sich kein Algorithmus mit polynomiell beschränkter Laufzeit zu dessen Lösung angeben. In der Praxis auftretende GLOPe werden aufgrund ihrer Größe und Problemkomplexität daher oft nicht exakt, sondern nur näherungsweise mittels Heuristiken gelöst (siehe 2.2.6).

2.2.5 Nichtlineare Optimierungsmodelle

Ein *nichtlineares Optimierungsmodell* besitzt eine nichtlineare Zielfunktion und/oder mindestens eine nichtlineare Nebenbedingung. Zugehörige Optimierungsprobleme werden als *nichtlineare Optimierungsprobleme* bezeichnet.

Viele Zusammenhänge, die in der Realität in Optimierungsproblemen auftreten, sind nichtlinear, wie z.B. Transportkosten in Abhängigkeit von der zu transportierenden Menge und der zurückzulegenden Entfernung. Werden derartige Zusammenhänge exakt in Form nichtlinearer Optimierungsmodelle abgebildet, so erkauft man die präzisere Abbildung in der Regel durch einen, im Gegensatz zu linearen Modellen, wesentlich höheren Rechenaufwand zu deren Lösung. Hierbei kommt erschwerend hinzu, dass für nichtlineare Optimierungsprobleme kein, mit dem Simplex-Algorithmus der LOPe vergleichbares, universelles Lösungsverfahren existiert.

Auf nichtlineare Optimierungsmodelle soll an dieser Stelle nicht weiter eingegangen werden. Wichtig ist die Feststellung, dass es nichtlineare Optimierungsmodelle zwar gestatten durch die Verwendung nichtlinearer Zielfunktionen und Nebenbedingungen Zusammenhänge der Realität exakter abbilden zu können, jedoch steigt die Problemkomplexität der Lösung solcher Optimierungsprobleme sehr stark an.

Soweit möglich, ist es zur Lösung praktischer Probleme oftmals ratsam nichtlineare Zusammenhänge durch lineare Zusammenhänge zu approximieren und mittels eines leichter zu lösenden (G)LOP zu modellieren und zu lösen. Der durch die Approximation bedingte Verlust an Genauigkeit kann in der Praxis oftmals vernachlässigt oder akzeptiert werden, da oft erst durch sie die Lösung eines Optimierungsproblems praxistauglicher Problemgröße in einer praxistauglichen Zeitspanne möglich wird.

2.2.6 Heuristiken

Im Hinblick auf die meist sehr hohen Rechenzeiten, die von exakten Lösungsverfahren benötigt werden, muss man sich in der Praxis oftmals damit zufrieden geben statt optimaler Lösungen lediglich gute zulässige Lösungen zu bestimmen. Verfahren, die zu diesem Zweck entwickelt werden, werden heuristische Verfahren, oder kurz *Heuristiken*, genannt. Sie bieten keine Garantie, dass ein Optimum gefunden, bzw. als solches erkannt wird. Dem Verlust an Genauigkeit steht meist ein erheblicher Laufzeitvorteil im Vergleich zu exakten Lösungsmethoden gegenüber. Die Güte der erzielten Lösung, im Vergleich zum Optimum, ist durch die Ausgestaltung des Verfahrens und damit oft verbunden vom investierten Rechenaufwand abhängig. Heuristiken stellen daher stets einen Kompromiss zwischen benötigter Laufzeit und erzielter Lösungsgüte dar.

Heuristiken lassen sich primär unterteilen in Eröffnungsverfahren, Lokale Such- bzw. Verbesserungsverfahren, sowie unvollständig exakte Verfahren.

Eröffnungsverfahren dienen der Bestimmung einer (ersten) zulässigen Lösung des betrachteten Problems. Wird eine Lösung schrittweise konstruiert und in jedem Verfahrensschritt nach der größtmöglichen Verbesserung des Zielfunktionswertes der bisherigen Teillösung gestrebt, so bezeichnet man diese Verfahren als greedy (engl. gierig) oder myopisch (kurzsichtig). Den Gegensatz dazu bilden vorausschauende Verfahren, die in jedem Schritt abschätzen, welche Auswirkungen z.B. eine Variablenfixierung auf die in nachfolgenden Schritten noch erzielbare Lösungsgüte besitzt. In Kapitel 3.2.2 wird ein konkretes Eröffnungsverfahren beschrieben.

Lokale Such- bzw. Verbesserungsverfahren starten zumeist mit einer zulässigen Lösung des Problems, die entweder zufällig oder durch Anwendung eines Eröffnungsverfahrens bestimmt wird. In jeder Iteration wird von der gerade betrachteten Lösung $\vec{x}$ zu einer Lösung aus der Nachbarschaft NB($\vec{x}$) fortgeschritten. NB($\vec{x}$) enthält alle Lösungen, die sich aus $\vec{x}$ durch eine einmalige Anwendung einer Transformationsvorschrift ergeben. Typische Transformationsvorschriften verändern die Lösung meist an genau einer Stelle oder verschieben, bzw. vertauschen Elemente der Lösung innerhalb ihrer Reihenfolge. Die Durchführung einer solchen Transformation wird Zug genannt. Die einzelnen Verfahren unterscheiden sich bzgl. der Strategien zur Untersuchung der Nachbarschaft und der Auswahl von Zügen. Festzulegen ist z.B. in welcher Reihenfolge die Nachbarlösungen der Nachbarschaft untersucht werden und wann zu einer dieser fortgeschritten wird. Die Reihenfolge der Untersuchung kann zufällig

oder systematisch bestimmt werden. Ein Fortschreiten kann erfolgen, sobald die erste bessere Nachbarlösung gefunden wird (First-Fit-Strategie) oder nachdem alle Nachbarlösungen untersucht wurden, zur besten aller Nachbarlösungen (Best-Fit-Strategie).

Weiterhin lassen sich die Verfahren in deterministische und stochastische Verfahren einteilen. *Deterministische Verfahren* ermitteln bei mehrfacher Anwendung auf ein und dasselbe Problem unter gleichen Startbedingungen stets dieselbe Lösung. *Stochastische Verfahren* enthalten demgegenüber eine zufällige Komponente, die bei wiederholter Anwendung auf ein und dasselbe Problem in der Regel zu unterschiedlichen Lösungen führt. Die zufällige Untersuchung der Nachbarschaft und ein Fortschreiten zur ersten besseren gefundenen Lösungen ist ein Beispiel für ein stochastisches Verfahren.

Reine Verbesserungsverfahren enden, sobald in einer Iteration keine verbessernde Nachbarlösung existiert, bzw. gefunden werden konnte. Die beste erhaltene Lösung stellt für die betrachtete Nachbarschaft ein lokales Optimum dar, deren Zielfunktionswert jedoch deutlich schlechter sein kann als die des globalen Optimums. Ein reines Verbesserungsverfahren wird in Kapitel 3.2.3 beschrieben. Um lokale Optima wieder verlassen zu können, müssen Züge erlaubt werden, die zwischenzeitlich zu einer Verschlechterung des Zielfunktionswertes führen. Heuristiken, die diese Möglichkeit vorsehen, werden *lokale Suchverfahren* genannt. Lässt sich das Grundprinzip zur Steuerung des Suchprozesses auf eine Vielzahl von Problemen und Nachbarschaftsdefinitionen anwenden, so spricht man auch von *Metastrategien*. Beispiele für derartige Verfahren sind z.B. Simulated Annealing [3] [43], Tabu Search [70] [27], genetische Algorithmen [49] oder Ameisenalgorithmen [64] [17]. Simulated Annealing wird in Kapitel 3.2.4 näher beschrieben. Auf eine genauere Darstellung weiterer Verfahren soll jedoch verzichtet werden. Einzelheiten können z.B. Kapitel 1 von [25] und eine vergleichende Übersicht [16] entnommen werden.

Unvollständig exakte Verfahren sind exakte Lösungsverfahren, die jedoch nicht vollständig ausgeführt werden, sondern vorzeitig über ein festgelegtes Abbruchkriterium abgebrochen werden. Mögliche Abbruchkriterien sind z.B. das Überschreiten einer gewissen Maximallaufzeit oder das Erreichen einer bestimmten, geschätzten Lösungsgüte.

2.3 Dienstgüte

2.3.1 Begriff

Der Begriff *Dienstgüte* (englisch „Quality of Service“, Abkürzung „QoS“) stammt aus dem Bereich der Kommunikationsnetzwerke und bezeichnet im Allgemeinen alle nicht-funktionalen Faktoren, welche die Qualität beeinflussen, mit der ein Dienst in ihnen oder auf ihnen erbracht wird. Werden Systeme oder Protokolle als dienstgütefähig (engl. „QoS enabled“) bezeichnet, so soll darunter die Fähigkeit verstanden werden aufgrund des Vorhandenseins entsprechender Mechanismen für die Inanspruchnahme von Diensten eine gewisse Dienstgüte garantieren zu können, oder zumindest Bemühungen zur Erfüllung dieser Dienstgüte zu unternehmen.

Da keine einheitliche Definition des Begriffs Dienstgüte existiert, soll sich das Grundverständnis von Dienstgüte im Folgenden an der Definition des CCITT[7] (Commité Consultatif Internationale Télégrafique et Téléfonique) orientieren, welche Dienstgüte im Jahre 1988 im Zusammenhang mit der ISDN[8]-Technologie sehr allgemein als „The collective effect of service performances which determine the degree of satisfaction of a user of the service." [1] definierte. Hierbei ist unter „user" nicht notwendigerweise ein menschlicher Benutzer zu verstehen, sondern auch nichtmenschliche Benutzer wie elektronische Geräte oder Software.

Je nachdem welche Art von Dienst unter welchen Aspekten betrachtet wird, setzt sich die Beurteilung der Güte eines Dienstes aus anderen Teilaspekten zusammen. Was konkret unter Dienstgüte verstanden werden soll, bzw. welche Aspekte konkret in Bezug auf die Dienstgüte zu berücksichtigen sind, ist daher im jeweiligen Kontext zu definieren. Dienstgüte wird im Folgenden als Oberbegriff für eine Menge zu spezifizierender Dienstgütekriterien verstanden. Eine konkrete Dienstgüte äußert sich entsprechend durch die konkrete Ausprägung der spezifizierten Dienstgütekriterien im Einzelfall.

Um Dienstgütekriterien nicht nur spezifizieren, sondern auch messen, prüfen und steuern zu können, ist es für die Praxis oft von besonderer Relevanz Dienstgütekriterien in Form quantitativer Parameter unter Verwendung einer zu definierenden Metrik ausdrücken zu können. Schmitt definiert Dienstgüte in diesem Zusammenhang als „the well-defined and controllable behaviour of a system with respect to quantitative parameters" [57].

Die Zusicherung einer gewissen Dienstgüte durch einen Dienstanbieter geschieht in Form eines „Service Level Agreement" (SLA) [63] [71]. SLAs sind formale Dokumente mit vertragsähnlichem Charakter, die eine Vereinbarung zwischen dem Anbieter und dem Nutzer einer Leistung darstellen und die Art und Weise der Leistungserbringung sowie relevanter Rahmenbedingungen festlegen, d.h. einen Service-Level definieren.

2.3.2 Dienstgüte in Netzwerken

In paketvermittelnden Netzwerken wie dem Internet werden Daten zur Übertragung in einzelne Transporteinheiten zerlegt und in Form einzelner, adressierter Pakete versendet. Hierbei passieren die Pakete auf dem Weg vom Sender zum Empfänger eine Vielzahl von Zwischensystemen, die jeweils durch Netzwerke mit den unterschiedlichsten Eigenschaften verbunden sind. So unterscheiden sich Teilstrecken z.B. hinsichtlich ihrer möglichen Transportgeschwindigkeit und Systeme bezüglich ihrer verfügbaren Puffergrößen und der Verarbeitungsgeschwindigkeit. Auf dem Weg durch diese Zwischenstationen können Pakete verloren gehen, sie können verzögert weitergeleitet werden, Paketinhalte können korrumpiert werden, die Empfangsreihenfolge

[7] Das CCITT erarbeitet technische Normen, Standards und Empfehlungen für alle Gebiete der Telekommunikation und wurde 1993 unter dem Namen „ITU Telecommunication Standardization Sector" (ITU-T) in die „International Telecommunication Union" (ITU) integriert. Siehe auch: http://www.itu.int/ITU-T/

[8] ISDN (Integrated Services Digital Network) ist ein voll digitales, leitungsvermittelndes Telefonsystem, das mit dem vorrangingen Ziel der Integration von Sprach- und Datendiensten entwickelt wurde. Siehe hierzu z.B. [65], S. 160ff.

der Pakete kann sich von der Sendereihenfolge unterscheiden und viele weitere Ereignisse, welche die Datenübertragung beeinflussen, können auftreten.

Die Güte eines Dienstes, der Daten in einem solchen Netz überträgt, kann anhand einer Vielzahl von Kriterien beurteilt werden, wie z.B. Latenz, Datendurchsatz oder Fehlerrate. Je nachdem zu welchem Zweck ein Dienst verwendet wird, sind die Dienstgütekriterien zur Beurteilung der Güte verschieden zu gewichten. Beispielsweise erfordert die Übertragung von Multimediadaten im Rahmen eines Video-Streamings eine möglichst konstante Datenrate, was jedoch wiederum bei der Übertragung von E-Mails nicht wichtig ist.

2.3.3 Dienstgütetypen

Eine sinnvolle Nutzung mancher Dienste ist nur möglich, falls sichergestellt werden kann, dass sich gewisse Dienstgüteparameter in bestimmten Grenzen bewegen. Beispielsweise stellt die Telefonie gewisse Anforderungen an eine niedrige Latenz, einen möglichst konstanten Datendurchsatz und eine niedrige Fehlerrate. Zu große Verzögerungen oder die häufige Unterbrechung der Übertragung würden kein befriedigendes Telefonieren ermöglichen. Manche Dienste bieten daher die Möglichkeit gewisse Dienstgüteparameter mit dem Nutzer zu vereinbaren. Die Spezifikation der Dienstgüteparameter bestimmt dabei den *Dienstgütetyp*. Prinzipiell lassen sich drei Dienstgütetypen unterscheiden (vgl. [61], S. 242ff): Garantierte, vorhersagbare und Best-Effort-Dienste.

Garantierte Dienste stellen Garantien für Dienstgüte in der Art zur Verfügung, dass sie garantieren, dass Dienstgüteparameter während der Dienstnutzung einen festen Wert besitzen (z.B. möglicher Datendurchsatz 100 MBit/s) oder sich in einem durch Grenzwerte definierten Intervall bewegen (z.B. Latenzzeit von nicht weniger als 10ms und nicht mehr als 100ms).

Ein *vorhersagbarer Dienst* (historischer Dienst) stellt abgeschwächte Garantien auf Basis seines Verhaltens in der Vergangenheit aus. Die vorhergesagten Dienstgüteparameter sind Schätzungen aufgrund des vergangenen Verhaltens, die der Dienst mit einer gewissen Wahrscheinlichkeit auch in Zukunft erfüllen kann. Ein Beispiel hierfür ist ein Dienst, der in seiner Vergangenheit eine durchschnittliche Bandbreite von 100 MBit/s zur Verfügung stellte und daher die Bandbreite von 100 MBit/s als Dienstgüteparameter anbietet. Er kann jedoch nicht garantieren, dass die Bandbreite nicht zeitweise unter 100 MBit/s abfallen kann. Er garantiert lediglich, dass die Bandbreite im Durchschnitt in etwa 100 MBit/s betragen wird, falls seine Umgebungsparameter sich so wie in der Vergangenheit gestalten.

Best-Effort-Dienste sind solche, die entweder keine Garantien ermöglichen oder auf partiellen Garantien basieren. Sie versuchen die Dienstgüte lediglich „so gut es geht“ zu erbringen. Die meisten verfügbaren Netzwerkprotokolle stellen lediglich Best-Effort-Dienste zur Verfügung.

2.3.4 Durchsetzung von Dienstgüte

Damit garantierte Dienste die, den Nutzern zugesicherte, Dienstgüte sicherstellen können, müssen sie ihre verwendeten Ressourcen verwalten. Basiert beispielsweise ein Dienst auf der Nutzung einer Netzwerkverbindung über die 100 MBit/s übertragen

werden können, so kann der Dienst nicht mehr als fünf Nutzern gleichzeitig eine Übertragung auf dieser Netzwerkverbindung mit jeweils 20 MBit/s garantieren. Der Dienst benötigt Informationen über die ihm zur Verfügung stehenden Ressourcen und die Garantien, welche ihm diese Ressourcen zusichern können. Anhand dieser Informationen ist der Dienst in der Lage die Ressourcen aktiv zu verwalten und entsprechende Garantien an seine Nutzer weiterzugeben. Im genannten Beispiel könnte der Dienst einen potentiellen sechsten Nutzer ablehnen oder aber seine Ressourcen durch den Aufbau einer neuen Netzwerkverbindung mit mindestens 20 MBit/s erweitern.

Nutzt ein garantierter Dienst wiederum Dienste, z.B. einer niedrigeren Schicht eines Schichtmodells[9], so müssen auch diese Dienste garantierte Dienste sein, für die rekursiv wiederum das gleiche gelten muss. Die ursprünglichen Dienstgüteanforderungen müssen durch den Dienst in Anforderungen an alle genutzten Dienste übersetzt und durch diese garantiert werden. Nur so ist der Dienst in der Lage die ursprünglich garantierte Dienstgüte erbringen zu können. In einem Schichtenmodell spricht man auch von der Übersetzung der Dienstgüte von Schicht zu Schicht ([61], S. 259ff). Existiert in der Kette der genutzten Dienste ein Dienst, der nicht in der Lage ist Dienstgüte zu garantieren, so kann ein Dienst, der diesem Dienst in der Nutzungskette nachfolgt, keinerlei Dienstgütegarantien mehr aussprechen. Das Durchsetzen von Dienstgüte ist daher in der Praxis oftmals problematisch, da Dienste verschiedenster Schichten hieran beteiligt sind und sich diese Dienste über eine große Anzahl beteiligter Komponenten und Systeme verteilen können.

Im Bereich von Netzwerken existieren verschiedenste Ansätze Dienstgüte durchzusetzen. Die einfachste, aber auch kostspieligste, besteht darin eine hinreichende Anzahl von Ressourcen angemessener Kapazität bereit zu halten, um die Dienstgüte selbst bei Spitzenbelastungen garantieren zu können (Overprovisioning). Problematisch ist hierbei jedoch die Vorhersage zukünftiger Spitzenbelastungen. Effizientere Verfahren basieren auf Reservierungsmechanismen und schalten der Dienstnutzung eine Reservierungsphase vor, in welcher der Dienst zunächst nötige Ressourcen reserviert und dem Nutzer nur dann die Nutzung gestattet, falls alle für die Erbringung der Dienstgüte nötigen Ressourcen verfügbar sind. Ein Beispiel für eine auf Reservierungen basierenden Architektur ist z.B. die IntServ-Architektur[10], welche RSVP[11] als Reservierungsprotokoll verwendet. Abgeschwächte Verfahren garantieren eine reservierte Dienstgüte nur mit einer gewissen Wahrscheinlichkeit, z.B. nur bei durchschnittlicher Belastung des Netzes, wodurch die Kapazitäten der verfügbaren Ressourcen besser ausgelastet werden können. Andere Verfahren beschränken sich darauf die verfügbaren Ressourcen ungleich an verschiedene Nutzer zu verteilen. Hierzu bilden sie Dienstgüteklassen und stellen je mehr Ressourcen zur Verarbeitung von Nutzerdaten zur Verfügung, desto höher die reservierte Dienstgüteklasse des Nutzers bewertet wird (z.B. gemäß Gold-Silber-Bronze-Priorisierung). Darüber hinaus existieren viele weitere Verfahren die hier aufgrund ihrer Fülle nicht erwähnt werden können. Ebenso soll auf eine genaue Darstellung der bereits genannten Verfahren im

[9] Ein Beispiel für ein Schichtmodell ist das ISO-OSI-Referenzmodell (siehe [65], S. 45ff).
[10] IntServ (Integrated Services) bezeichnet eine Architektur [18] zur Durchsetzung von Dienstgüte [58] [73] in einem, auf Internetstandards basierenden Netz. Siehe hierzu z.B. [61], S. 488ff oder [33], S. 17ff.
[11] RSVP (Resource Reservation Protocol) [19] [74] ist das in der IntServ-Architektur verwendete Protokoll zur Reservierung von Verbindungen, bzw. Datenpfaden, mit bestimmten Dienstgüteeigenschaften.

Rahmen dieses Grundlagenkapitels verzichtet werden. Für eine detailliertere Betrachtung des Themengebietes sei z.B. auf [33] verwiesen.

2.3.5 Dienstgüte im Bereich von Web Services

In etablierten Bereichen, in denen garantierte Dienste schon seit langem eine wichtige Rolle spielen, wie z.B. Kommunikationsnetzwerke oder Multimedia- und Echtzeitanwendungen, ist Dienstgüte ein gut untersuchtes Thema. Wichtige Dienstgütekriterien zur Spezifikation von Dienstgüte wurden definiert, Beschreibungsmöglichkeiten für Dienstgüte geschaffen, Abbildungsmechanismen von Dienstgüte auf andere Schichten entwickelt und Verfahren zur Sicherstellung von Dienstgüte durch die Verwendung von spezialisierten Ressourcen-Managern entwickelt.

Dienstgüte in dynamischen, verteilten Systemen, wie sie z.B. in einer SOA auf Web-Service-Basis realisiert werden können, ist im Gegensatz dazu noch ein relativ schlecht untersuchtes Gebiet in dem sich noch keine Standards etablieren konnten. Web Services können eine Fülle von Funktionalitäten bereitstellen und für die verschiedensten Zwecke in den verschiedensten Umgebungen eingesetzt werden. Zusätzlich ist es möglich Web Services unter einer Vielzahl von Aspekten zu betrachten, wie z.B. unter dem Gesichtpunkt ihrer Funktionalität, ihrer Eigenschaften als Softwarekomponente oder ihrer Eigenschaften als Komponente eines Netzwerkes. Es ist daher nicht möglich eine bestimmte Menge von Dienstgütekriterien für die Spezifikation der Dienstgüte eines Web Services für jeden Einsatzzweck zu definieren. Dementsprechend viele Varianten von Dienstgütekriterien für Web Services lassen sich in der Literatur finden (vgl. z.B. [44], [30], [53], [47]).

In [44] werden folgende relevanten Bereiche von Dienstgütekriterien für Web Services identifiziert: Performanz, Zuverlässigkeit, Skalierbarkeit, Kapazität, Robustheit, Ausnahmebehandlung, Akkuratesse, Integrität, Erreichbarkeit, Verfügbarkeit, Interoperabilität, Sicherheit und netzwerkbezogene Kriterien.

Web Services kommunizieren ausschließlich mittels Nachrichten, die über ein Netzwerk zwischen Web-Service-Konsument und Web-Service-Anbieter transportiert werden. Ihre Dienstgüte wird daher in vielerlei Hinsicht von den Eigenschaften des Transportnetzwerkes determiniert. So hängt z.B. die Geschwindigkeit und Fehleranfälligkeit der Nachrichtenübermittlung von Web Services hauptsächlich von den Eigenschaften des verwendeten Transportnetzwerks ab, die wiederum einen hohen Einfluss auf viele weitere Dienstgütekriterien von Web Services haben, wie z.B. die Performanz. In der Praxis ergibt sich daher eine recht netzwerkzentrische Sicht auf die Dienstgüte von Web Services und ihre Dienstgüte wird mit Dienstgütekriterien spezifiziert, die sich an den bekannten Dienstgütekriterien von herkömmlichen Netzwerkdiensten orientieren. Tabelle 1 können einige typische Beispiele für verwendete Dienstgütekriterien in Bezug auf Web Services entnommen werden.

Kriterium	Beschreibung
Antwortzeit	Zeitspanne die vom Absenden einer Anfrage an den Web Service bis zum Erhalt einer Antwort vergeht. Diese Zeitspanne beinhaltet die, vor allem durch das Netzwerk determinierte, Latenz und die Ausführungszeit des Web Services.
Latenz	Die Zeitspanne, die zwischen dem Absender einer Anfrage und dem Empfang einer Antwort zwangsläufig vergehen muss und unabhängig von der Größe der Anfrage und der Anzahl der auszuführenden Operationen durch den Web Service ist. Üblicherweise wird diese Zeitspanne von der Dauer determiniert, die eine Nachricht minimaler Länge benötigt um über das Netzwerk vom Nutzer zum Web Service und wieder zurück transportiert zu werden (minimale Round-Trip-Time).
Ausführungszeit	Zeit die ein Web Service zur Abarbeitung einer Anfrage benötigt. Sie wird gemessen vom Zeitpunkt des Erhalts der Anfrage durch den Web Service bis zum Absenden der Antwort an den Empfänger durch den Web Service.
Verfügbarkeit	Wahrscheinlichkeit mit der ein Web Service im Falle eines Aufrufs erreichbar ist, seine Funktionalität erbringen und eine korrekte Antwort liefern kann.
Max. Durchsatz	Maximale Anzahl von Anfragen die in einem bestimmten Zeitintervall durch einen Web Service ohne Verletzung seiner Dienstgütezusicherung verarbeitet werden können.

Tabelle 1 - Beispiele für Dienstgütekriterien von Web Services

Dienstgüte zusammengesetzter Web Services

Ist ein Web Service aus anderen Web Services orchestriert, so wird seine Dienstgüte maßgeblich durch die Dienstgüte der verwendeten Web Services determiniert. Im einfachsten Fall kapselt ein Web Service lediglich die sequentielle Ausführung einer bestimmten Anzahl weiterer Web Services. In diesem Fall ergibt sich dessen Dienstgüte in guter Näherung aus den Dienstgüteparametern der aufgerufenen Web Services. So addieren sich beispielsweise die Antwortzeiten der aufgerufenen Web Services zur Gesamtantwortzeit, die Verfügbarkeitswahrscheinlichkeiten multiplizieren sich zur Gesamtverfügbarkeitswahrscheinlichkeit und der maximale Gesamtdurchsatz ergibt sich aus dem minimalen Maximaldurchsatz aller verwendeter Web Services. Handelt es sich bei der Orchestrierung nicht um eine rein sequentielle Ausführung, sondern werden komplexere Kontrollkonstrukte wie beispielsweise Verzweigungen, Schleifen, Parallelausführung u.ä. verwendet, so gestaltet sich die Bestimmung der Dienstgüte eines zusammengesetzten Web Service wesentlich komplexer.

Die Bestimmung der Dienstgüte zusammengesetzter Web Services ist jedoch von besonderer Relevanz, soll die Dienstgüte eines solchen Web Services Nutzern, z.B. in Form eines SLAs, garantiert werden. Sind einige oder alle im zusammengesetzten Web Service verwendeten Web Services durch andere funktional gleiche Web Services mit unterschiedlicher Dienstgüte austauschbar, so entsteht hieraus das Problem den Web Service aus genau solchen Web Services zusammenzusetzen, dass die, den Nutzern durch eine SLA garantierte Dienstgüte bestmöglichst erfüllt wird. Kann ein Anbieter seinen Web Service durch eine diesbezügliche Optimierung mit einer besseren Dienstgüte als seine Wettbewerber anbieten, so ist er in der Lage durch diese Differenzierung hohe Wettbewerbsvorteile zu erzielen. Zusätzlich kann er ein und diesel-

be Web-Service-Funktionalität durch verschieden zusammengesetzte Web Services zu unterschiedlichen Dienstgüten anbieten.

Mechanismen, welche die Bestimmung und Optimierung der Dienstgüte orchestrierter Web Services erlauben, sind daher von hohem Interesse. Diese Mechanismen lassen sich auch für andere Bereiche der Web-Service-Nutzung verwenden, wie z.B. Geschäftsprozesse auf der Basis von Web Services. Das Problem anhand einer Geschäftsprozessdefinition, der Vorgabe einzuhaltender (Dienstgüte-)Restriktionen und festgelegten Optimierungskriterien entsprechende Ausführungspläne zur Ausführung eines Geschäftsprozesses auf Basis konkreter Web Services zu erzeugen, die zugleich alle Restriktionen erfüllen, als auch hinsichtlich der Optimierungskriterien bestmöglich optimiert sind, ist Gegenstand der weiteren Betrachtungen dieses Technical Reports.

Nicht weiter betrachtet werden sollen an dieser Stelle Aspekte, welche die Umsetzung von Dienstgütemechanismen im Bereich der Web Services betreffen, wie z.B.

- die Definition und Spezifikation von Dienstgütekriterien,
- die Definition und Spezifikation der Dienstgüte unter Verwendung definierter Dienstgütekriterien,
- die Einführung von Mechanismen zum Abgleich von Dienstgüteanforderungen von Web-Service-Konsumenten und Dienstgüteangeboten von Web-Service-Anbietern,
- die Entwicklung von Methoden zur Durchsetzung und Kontrolle vereinbarter Dienstgüte.

Eine Erläuterung dieser Aspekte kann [60] entnommen werden. Ein Vergleich verschiedener Ansätze, welche diese Aspekte adressieren, wird in [66] gegeben.

3 Mathematisches Modell und Heuristik

Im Rahmen dieses Technical Reports wird ein heuristisches Verfahren zur optimierten Ausführung von Geschäftsprozessen vorgestellt, welches aufgrund seines Rechenzeitbedarfs dazu geeignet ist, zur Laufzeit von einer BPE zur automatischen Optimierung verwendet werden zu können. Gleichzeitig ist diese Heuristik möglichst generisch gehalten, um eine Vielzahl von Kriterien zur Definition von Restriktions- und Optimierungskriterien berücksichtigen zu können. In diesem Kapitel wird die Aufgabenstellung, unter der die Heuristik entwickelt wurde, näher präzisiert und als Optimierungsproblem formuliert. Anschließend wird ein entsprechendes mathematisches Optimierungsmodell vorgestellt und eine vierstufige Heuristik zu dessen Lösung präsentiert.

3.1 Mathematisches Modell

Im Folgenden wird die konkrete Problemstellung zunächst eingehender betrachtet, anhand einiger Beispiele näher beschrieben und als Optimierungsproblem formuliert. Anschließend wird das mathematische Modell des Optimierungsproblems schrittweise entwickelt.

3.1.1 Vorbetrachtung der Problemstellung

Geschäftsprozess

Um die Komplexität der Entwicklung einer ersten Ausführungsoptimierung für Geschäftsprozesse auf ein handhabbares Maß zu reduzieren, werden nur solche Geschäftsprozesse betrachtet, die sich mit dem Kontrollkonstrukt der sequentiellen Ausführung definieren lassen. Ein solcher *sequentieller Geschäftsprozess* besteht aus einer endlichen Anzahl von Prozessschritten, die in einer fest definierten Reihenfolge hintereinander ausgeführt werden. Hierbei ist jedem Prozessschritt eine Funktionalität zugeordnet.

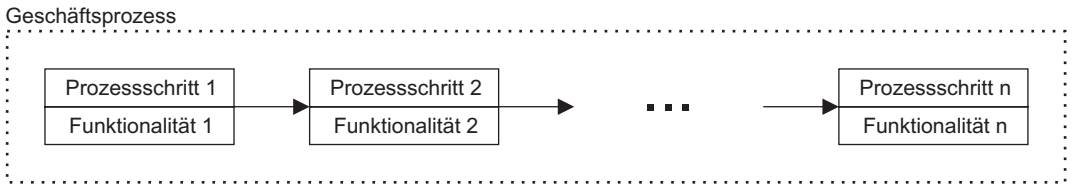

Abbildung 2 - Schematische Darstellung eines sequentiellen Geschäftsprozesses

Bei der Ausführung des Geschäftsprozesses ist es die Aufgabe der BPE für jede im aktuellen Prozessschritt geforderte Funktionalität einen konkreten Web Service auszuwählen und auszuführen. Zur Ausführung eines jeden Prozessschrittes stehen verschiedene, funktional identische Web Services (*Prozessschrittkandidaten*) zur Verfügung, die sich jedoch in ihren nicht-funktionalen Eigenschaften, z.B. hinsichtlich ihrer Dienstgüte, unterscheiden. Funktional identische Web Services werden in *Kategorien* gruppiert.

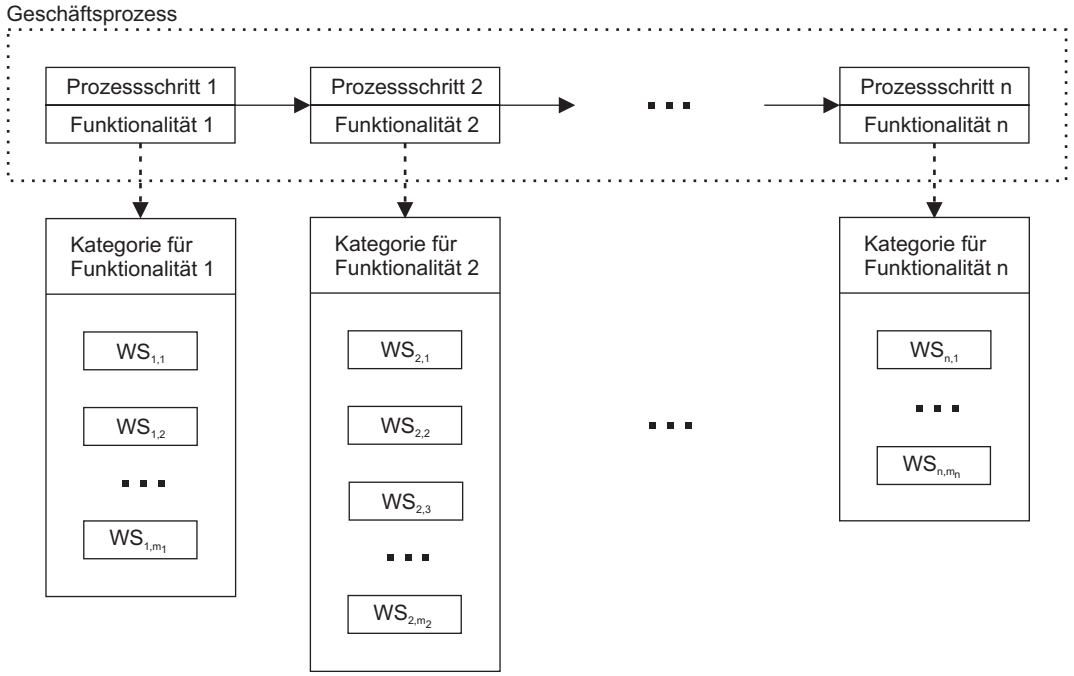

Abbildung 3 - Schematische Darstellung der Zuordnung von Prozessschritten zu Kategorien

Das Ziel der Ausführungsoptimierung ist es, bei der Ausführung des Geschäftsprozesses für jeden Prozessschritt aus der entsprechenden Kategorie einen Web Service so zu wählen, dass der gesamte Geschäftsprozess bestimmte Eigenschaften erfüllt. Zugleich soll der Nutzen des Geschäftsprozesses maximiert werden, wobei der Nutzen über eine Gewichtung der sich aufgrund der konkreten Ausführung ergebenden Eigenschaften des Geschäftsprozesses definiert wird.

Differenzierung der Web Services

Zur Differenzierung der Web Services können diese neben der Gruppierung gemäß ihrer Funktionalität in Kategorien weiterhin durch eine beliebige Anzahl von nichtfunktionalen Eigenschaften beschrieben werden, wie z.B. Dienstgütekriterien (Antwortzeit, Verfügbarkeit, maximaler Durchsatz), Preis, u.ä.. Jede dieser Eigenschaften wird durch einen Parameter, bzw. einen numerischen Parameterwert (z.B. Antwortzeit = 1000 ms, Preis = 0,01 EUR) beschrieben. Aus einem *Ausführungsplan*, der die konkret zur Ausführung eines Geschäftsprozesses aufzurufenden Web Services festlegt, lassen sich die Eigenschaften, bzw. Parameterwerte berechnen, die für die Ausführung des Geschäftsprozesses als Ganzes gelten.

Gesamtparameterwerte des Ausführungsplans

Um die Eigenschaften eines konkreten Ausführungsplans für einen Geschäftsprozess zu ermitteln, werden die Parameterwerte der einzelnen verwendeten Web Services zu entsprechenden *Gesamtparameterwerten* für den Ausführungsplan aggregiert. Hierbei können drei Typen von Parametern unterschieden werden, die sich jeweils in der Art der Aggregation der Einzelparameterwerte zum Gesamtparameterwert unterscheiden:

- additive Parameter
- multiplikative Parameter
- Minimaloperator-Parameter

Additive Parameter werden durch Summierung aggregiert (z.B. Summierung der Antwortzeiten der einzelnen Web Services zur Gesamtantwortzeit), *multiplikative Parameter* durch Multiplikation (z.B. Multiplikation der einzelnen Verfügbarkeitswahrscheinlichkeiten zur Gesamtverfügbarkeitswahrscheinlichkeit) und *Minimaloperator-Parameter* durch Anwenden des Minimaloperators auf die Parameterwerte (z.B. maximaler Gesamtdurchsatz als Minimum des maximalen Durchsatzes aller verwendeter Web Services).

Ein Beispiel:

Ein Geschäftsprozess bestehe aus der Hintereinanderausführung von drei Prozessschritten. Für die Ausführung eines jeden Prozessschrittes stehen in der jeweiligen Kategorie jeweils drei verschiedene, funktional identische Web Services zur Verfügung, die sich in ihren nicht-funktionalen Eigenschaften wie folgt unterscheiden:

Kategorie 1	**Web Service 1**	**Web Service 2**	**Web Service 3**
Antwortzeit [ms]	1000	1500	500
Verfügbarkeit [%]	99,5	99	99,9
Max. Durchsatz [Aufrufe/Min]	2500	5000	1000

Kategorie 2	**Web Service 1**	**Web Service 2**	**Web Service 3**
Antwortzeit [ms]	250	500	750
Verfügbarkeit [%]	99,9	99,6	99,1
Max. Durchsatz [Aufrufe/Min]	1250	1500	2000

Kategorie 3	**Web Service 1**	**Web Service 2**	**Web Service 3**
Antwortzeit [ms]	3000	2000	4000
Verfügbarkeit [%]	99,95	99,99	99,9
Max. Durchsatz [Aufrufe/Min]	4500	3000	6000

Tabelle 2 - Beispiel von Kategorien funktional gleicher Web Services mit unterschiedlichen nicht-funktionalen Eigenschaften

Ein möglicher Ausführungsplan des Geschäftsprozesses könnte zur Ausführung des Prozessschritts 1 den Web Service 1 aus Kategorie 1 festlegen, für Prozessschritt 2 den Web Service 3 aus Kategorie 2 und für Prozessschritt 3 den Web Service 2 aus Kategorie 3 (Kurznotation „1/1 -> 3/2 -> 2/3“). Die Antwortzeit ist ein additiver Parameter, weshalb sich die gesamte Antwortzeit dieses Ausführungsplans durch Addition der Antwortzeiten der gewählten Web Services ergibt (1000 ms + 750 ms + 2000 ms = 3750 ms). Da die Verfügbarkeit ein multiplikativer Parameter ist, ergibt sich die Gesamtverfügbarkeit des Ausführungsplans, d.h. die Wahrscheinlichkeit mit der bei einer Ausführung des Geschäftsprozesses jeder auszuführende Web Service verfügbar ist, durch Multiplikation der Verfügbarkeiten der gewählten Web Services (0,995 * 0,991 * 0,9999 = 0,9859463955 ~ 98,59%). Der Maximaldurchsatz ist ein Minimaloperator-Parameter und sein Gesamtparameterwert ergibt sich aus dem Minimum der Maximaldurchsätze aller gewählten Web Services (Min{2500, 2000, 3000} = 2000). Der betrachtete Ausführungsplan zur Ausführung des Geschäftsprozesses benötigt zusammenfassend eine Antwortzeit von 3750 ms, kann maximal 2000 mal pro Minute aufgerufen werden und kann mit einer Wahrscheinlichkeit von 98,59% aufgrund der Verfügbarkeit der beteiligten Web Services erfolgreich durchlaufen werden.

Nutzen eines Ausführungsplans

Für einen Geschäftsprozess existiert eine Vielzahl von Möglichkeiten einen Ausführungsplan zu erstellen. Im genannten Beispiel existieren 3 * 3 * 3 = 27 verschiedene Möglichkeiten konkrete Web Services für die Prozessschritte des Geschäftsprozesses zu wählen und daher auch 27 verschiedene Ausführungspläne. Jeder dieser Ausführungspläne besitzt andere Gesamtparameterwerte. Im Zuge der Ausführungsoptimierung muss die BPE aus allen diesen möglichen Ausführungsplänen den optimalen Ausführungsplan auswählen. Welcher Ausführungsplan soll hierbei jedoch als optimal angesehen werden? Der Ausführungsplan mit der kürzesten Antwortzeit? Der Ausführungsplan mit der höchsten Verfügbarkeit oder dem höchsten Maximaldurchsatz? Oder gar ein Ausführungsplan mit möglichst kurzer Antwortzeit, möglichst hoher Verfügbarkeit und einem beliebigen Maximaldurchsatz?

Um zu definieren, welcher Ausführungsplan als optimal angesehen werden kann, wird jedem Ausführungsplan ein *Nutzen* zugewiesen, der als numerischer Wert ausgedrückt wird. Der Nutzen, bzw. der Nutzenwert, ist je höher, je besser ein Ausführungsplan zuvor definierte Präferenzen erfüllt. Welche Eigenschaften, bzw. Gesamtparameterwerte des Ausführungsplans welchen Nutzen stiften, wird über Gewichte definiert.

Jedem Gesamtparameterwert des Ausführungsplans (z.B. Gesamtantwortzeit, Gesamtverfügbarkeit, Maximaldurchsatz) wird ein *Gewicht* zugeordnet, welches definiert wie viele Nutzeneinheiten eine Einheit des Gesamtparameterwertes stiftet. Wird dem Maximaldurchsatz beispielsweise ein Gewicht von 0,1 zugewiesen so bedeutet dies, dass eine Zunahme des Maximaldurchsatzes um eine Einheit, einen Nutzenzuwachs von 0,1 Nutzeneinheiten bewirkt, bzw. dass anders ausgedrückt ein Nutzenzuwachs von genau einer Nutzeneinheit durch einen Zuwachs von 10 Einheiten des Maximaldurchsatzes erreicht wird. Wird der Gesamtverfügbarkeit ein Gewicht von 50 zugewiesen, so bewirkt eine Erhöhung der Gesamtverfügbarkeit um eine Einheit einen Nutzenzuwachs um 50 Nutzeneinheiten, bzw. ein Nutzenzuwachs von genau einer Nutzeneinheit wird durch einen Zuwachs von 0,02 Einheiten der Gesamtverfügbarkeit erreicht. Im geschilderten Fall stiftet eine Zunahme des Maximaldurchsatzes um 10 Einheiten und eine Zunahme der Gesamtverfügbarkeit um 0,02 Einheiten jeweils den gleichen Zusatznutzen und ist daher bzgl. der Optimierung gleichwertig. Wird ein Gesamtparameterwert mit 0 gewichtet, so stiftet er gar keinen Nutzen und ist daher für die Berechnung des Nutzens und die Optimierung des Ausführungsplans unerheblich.

Zur Berechnung des Gesamtnutzens eines Ausführungsplans werden zunächst alle Gesamtparameterwerte durch eine Multiplikation mit ihrem Gewicht in Nutzeneinheiten transformiert und anschließend summiert. Hierbei ist zu beachten, dass Gesamtparametern, für welche höhere Gesamtparameterwerte als schlechter angesehen werden (z.B. die Gesamtantwortzeit), ein negatives Gewicht zugeordnet wird. Hierdurch wird sichergestellt, dass eine Erhöhung des Gesamtparameterwertes mit einem Rückgang und nicht etwa mit einer Steigerung des Nutzens einhergeht.

Für das Optimierungsproblem zur Findung des optimalen Ausführungsplans stellt die Funktion zur Berechnung des Nutzens eines Ausführungsplans die zu maximierende Zielfunktion dar.

Wird im eingangs betrachteten Beispiel festgelegt, dass für die Optimierung des Ausführungsplans eine Verbesserung der Antwortzeit um 32 ms, eine Verbesserung der Verfügbarkeit um 0,02% und eine Verbesserung des Maximaldurchsatzes um 10 Aufrufe pro Minute äquivalent sind, so ergibt sich hieraus ein Gewicht von -0,03125 (=-1*1/32) für die Antwortzeit, ein Gewicht von 5000 (=1/0,0002) für die Verfügbarkeit und ein Gewicht von 0,1 (=1/10) für den Maximaldurchsatz. Der im Beispiel betrachtete Ausführungsplan „1/1 -> 3/2 -> 2/3“ besitzt die Gesamtparameter Gesamtantwortzeit 3750 ms, Gesamtverfügbarkeit 98,59% und Maximaldurchsatz 2000 Aufrufe pro Minute. Mit der angegebenen Gewichtung der Gesamtparameterwerte ergibt sich für diesen Ausführungsplan ein Nutzen in Höhe von 3750 * -0,03125 + 0,9859 * 5000 + 2000 * 0,1 = 5012,3125 Nutzeneinheiten. Für diese Gewichtung der Gesamtparameter existiert im Beispiel kein anderer Ausführungsplan, der einen höheren Nutzen aufweist. Bei dem betrachteten Ausführungsplan handelt es sich daher um den optimalen Ausführungsplan für die durch die Gewichte ausgedrückten Präferenzen.

Restriktionen

Die Nutzenoptimierung des Ausführungsplans eines Geschäftsprozesses ermöglicht es zwar über die Gewichtung der Gesamtparameterwerte Präferenzen festzulegen, nach welchen Eigenschaften der Ausführungsplan optimiert werden soll, jedoch ist es hierdurch nicht möglich gewisse Eigenschaften für den Ausführungsplan, wie beispielsweise eine maximale Antwortzeit, sicherzustellen. Zu diesem Zweck lassen sich *Restriktionen* für die Gesamtparameterwerte einführen, die zum Ausschluss aller Ausführungspläne führen, welche diese Restriktionen nicht erfüllen. Ausführungspläne, welche alle Restriktionen erfüllen werden als *zulässig* bezeichnet.

Wird im Beispiel eine Restriktion eingeführt, welche die maximale Gesamtantwortzeit des Geschäftsprozesses auf 3000 ms beschränkt, so verstößt der zuvor als optimal ermittelte Ausführungsplan durch seine Gesamtantwortzeit von 3750 ms gegen diese Restriktion und ist daher nicht mehr zulässig. Unter allen zulässigen Ausführungsplänen besitzt der Ausführungsplan „3/1 -> 1/2 -> 2/3“ mit 5003,5625 Nutzeneinheiten den höchsten Nutzen und ist damit unter der eingeführten Restriktion der optimale Ausführungsplan (Gesamtantwortzeit 2750 ms, Gesamtverfügbarkeit 99,79%, Maximaldurchsatz 1000 Aufrufe pro Minute).

Zusammenfassend besteht die Aufgabe der Ausführungsoptimierung in der Maximierung des Gesamtnutzens des Ausführungsplans unter gleichzeitiger Einhaltung aller bezüglich der Gesamtparameterwerte definierten Restriktionen. Für das Optimierungsproblem zur Findung des optimalen Ausführungsplans stellen die Restriktionen Nebenbedingungen dar, unter denen die Zielfunktion (Nutzen) zu maximieren ist.

3.1.2 Formulierung des mathematischen Modells

Um den optimalen Ausführungsplan eines Geschäftsprozesses zu ermitteln, besteht die prinzipielle Möglichkeit alle möglichen Ausführungspläne zu erzeugen, deren Nutzen zu berechnen und jenen Ausführungsplan unter allen zulässigen Ausführungsplänen zu wählen, welcher den höchsten Nutzen besitzt. Mit zunehmender Anzahl von Prozessschritten und Prozessschrittkandidaten steigt jedoch die Anzahl möglicher Ausführungspläne schnell auf eine Menge an, deren vollständige Enumeration und Auswertung nicht mehr mit einem sinnvollen Zeitaufwand möglich ist. Wird das Optimierungsproblem jedoch als lineares Optimierungsproblem formuliert, so können fortgeschrittene Methoden zur Lösung solcher Optimierungsprobleme verwendet

werden, wodurch der Zeitaufwand zur Lösungsfindung drastisch reduziert werden kann. Aber selbst eine solche Lösungsfindung erfordert unter Umständen einen Zeitaufwand, der für den Einsatz der Ausführungsoptimierung zur Laufzeit in einer BPE nicht mehr akzeptabel ist und den Einsatz von heuristischen Lösungsmethoden erforderlich macht (vgl. Kapitel 2.2.6).

Im Folgenden wird ein gemischt-ganzzahliges lineares Optimierungsmodell zum beschriebenen Optimierungsproblem des Auffindens des optimalen Ausführungsplans für einen Geschäftsprozess entwickelt. Wird das Problem durch das Fallenlassen der Ganzzahligkeitsrestriktion relaxiert (LP-Relaxation), so lässt sich sehr schnell eine exakte Lösung für das relaxierte Problem berechnen. Die exakte Lösung des relaxierten Problems bildet die Grundlage für das, im Rahmen dieses Technical Reports vorgestellten, heuristischen Lösungsverfahren.

Geschäftsprozess, Ausführungsplan und Kategorien

Ein sequentieller Geschäftsprozess bestehe aus n Prozessschritten. Prozessschritt i (i=1,...,n) des Prozesses wird vor Prozessschritt i' (i'=1,...,n) des Geschäftsprozesses ausgeführt, genau dann wenn i<i'. Prozessschritt i entspricht Position i im Ausführungsplan. Für jeden Prozessschritt i stehen m_i alternative Web Services (Prozessschrittkandidaten) zur Verfügung, welche die benötigte Funktionalität des Prozessschrittes erbringen können, und die in der Kategorie i zusammengefasst werden. Für alle m_i Web Services j (j=1,..., m_i) der Kategorie i, wird eine binäre Entscheidungsvariable $x_{i,j}$ eingeführt. Ist $x_{i,j}=1$, so bedeutet dies, dass für den Prozessschritt i der j-te Web Service aus der Kategorie i verwendet wird, ist $x_{i,j}=0$ so wird der entsprechende Web Service nicht verwendet. Damit nur jeweils ein Web Service pro Prozessschritt ausgeführt wird, darf jeweils nur ein Web Service aus einer Kategorie gewählt werden, d.h. es muss stets gelten, dass

$$\sum_{j=1}^{m_i} x_{i,j} = 1 \quad \forall i = 1,\ldots,n\,.$$

Parameterwerte

Die Parameterwerte eines Web Services werden mit $p_{i,j,k}$ bezeichnet, wobei i der Index für den Prozessschritt ist, j (j=1,..., m_i) der Index für den Web Service in der Kategorie i, sowie k (k=1,...,K) der Index für den Parameterwert des Web Services.

Um die Gesamtparameterwerte des Ausführungsplans zu erhalten, ist es nötig die Parameterwerte aller zur Ausführung ausgewählten Web Services zu dem jeweiligen Gesamtparameterwert zu aggregieren. Die Art und Weise der Aggregation unterscheidet sich je nach Parametertyp und ist für additive Parameter, multiplikative Parameter und Minimaloperator-Parameter unterschiedlich im Modell abzubilden. Da die Aggregation jedoch für alle Parameter des gleichen Typs identisch ist, wird im Folgenden auf den Index k der Parameterwerte $p_{i,j,k}$ verzichtet und statt dessen der Typ des Parameters wie folgt angegeben: Parameterwerte additiver Parameter werden mit $p_{i,j}^{+}$, Parameterwerte multiplikativer Parameter mit $p_{i,j}^{\bullet}$ und Parameterwerte von Minimaloperator-Paramentern mit $p_{i,j}^{\min}$ bezeichnet.

Additive Parameter

Zur Aufnahme der Gesamtparameterwerte additiver Parameter wird für jeden additiven Parameter eine zusätzliche Variable x^+ eingeführt. Ein Gesamtparameterwert x^+ ergibt sich durch Addition der Parameterwerte $p_{i,j}^+$ des jeweiligen additiven Parameters über alle zur Ausführung ausgewählten Web Services.

$$x^+ = \sum_{i=1}^{n} \sum_{j=1}^{m_i} p_{i,j}^+ x_{i,j}$$

Um einen Gesamtparameterwert x^+ mittels einer Restriktion zu beschränken kann eine Nebenbedingung der Form

$$x^+ \leq c^+ \text{ oder } x^+ \geq c^+$$

eingeführt werden, wobei c^+ eine Konstante ist, die den Gesamtparameterwert x^+ beschränkt.

Multiplikative Parameter

Zur Aufnahme der Gesamtparameterwerte multiplikativer Parameter wird für jeden multiplikativen Parameter eine zusätzliche Variable $x^\bullet$ eingeführt. Ein Gesamtparameterwert $x^\bullet$ ergibt sich durch Multiplikation der Parameterwerte $p_{i,j}^\bullet$ des jeweiligen multiplikativen Parameters über alle zur Ausführung ausgewählten Web Services.

$$x^\bullet = \prod_{i=1}^{n} \sum_{j=1}^{m_i} p_{i,j}^\bullet x_{i,j}$$

Ein Produkt über Entscheidungsvariablen lässt sich jedoch nicht in ein lineares Optimierungsmodell übernehmen, da dort Entscheidungsvariablen lediglich über Additions- und Subtraktionsoperationen miteinander verbunden sein dürfen. Durch eine Logarithmierung der Gleichung lässt sich jedoch das Produkt in eine Summe überführen.

$$x^\bullet = \prod_{i=1}^{n} \sum_{j=1}^{m_i} p_{i,j}^\bullet x_{i,j} = \prod_{i=1}^{n} \prod_{j=1}^{m_i} p_{i,j}^{\bullet\, x_{i,j}}$$

$$\ln(x^\bullet) = \ln\left(\prod_{i=1}^{n} \prod_{j=1}^{m_i} p_{i,j}^{\bullet\, x_{i,j}} \right) = \sum_{i=1}^{n} \left(\ln\left(\prod_{j=1}^{m_i} p_{i,j}^{\bullet\, x_{i,j}} \right) \right) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} \ln\left(p_{i,j}^{\bullet\, x_{i,j}} \right) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} x_{i,j} \ln\left(p_{i,j}^\bullet \right)$$

Da hierbei jedoch auch $x^\bullet$ logarithmiert wird, wird durch die Summe statt des Gesamtparameterwertes der logarithmierte Gesamtparameterwert berechnet. Zur Berechnung des Nutzenbeitrags, den der Gesamtparameterwert stiftet, ist es in der Zielfunktion jedoch nötig den unlogarithmierten Gesamtparameterwert mit dem Gewicht des Gesamtparameterwertes zu multiplizieren. Den exakten unlogarithmierten Gesamtparameterwert durch eine Summe auszudrücken, wie sie im linearen Optimierungsmodell benötigt wird, ist jedoch nicht möglich.

Unter bestimmten Voraussetzungen ist es jedoch möglich den Gesamtparameterwert durch eine Summe anzunähern. Sind alle Parameterwerte $p^{\bullet}_{i,j}$ sehr nahe an 1, so gilt folgende Approximation:

$$x^{\bullet} = \prod_{i=1}^{n} \sum_{j=1}^{m_i} p^{\bullet}_{i,j} x_{i,j} \approx 1 - \sum_{i=1}^{n} \left(1 - \sum_{j=1}^{m_i} p^{\bullet}_{i,j} x_{i,j} \right)$$

Die Motivation zur Verwendung dieser Approximation liegt in der Tatsache begründet, dass multiplikative Parameter im konkreten Einsatzgebiet des entwickelten Optimierungsverfahrens vor allem zur Abbildung von Wahrscheinlichkeiten verwendet werden. Web Services werden in diesem Zusammenhang meist mit einer Verfügbarkeitswahrscheinlichkeit als nicht-funktionale Eigenschaft näher beschrieben. In praxisrelevanten Fällen ist mit einer Verfügbarkeitswahrscheinlichkeit von weit mehr als 99% zu rechnen, da selbst eine Verfügbarkeitswahrscheinlichkeit von 99% bedeuten würde, dass der entsprechende Web Service bei jedem hundertsten Aufruf nicht zu erreichen wäre. Bei ausschließlicher Verwendung solcher Web Services würde die Ausführung eines Geschäftsprozesses bestehend aus 10 Prozessschritten nur mit einer Wahrscheinlichkeit von 90,44% erfolgreich sein, d.h. bereits jeder zehnte Aufruf scheitern. In der Praxis ist daher eher mit Verfügbarkeitswahrscheinlichkeiten von 99,999% (sog. „five nines availability“) und mehr zu rechnen. Für die angegebene Approximation sind Parameterwerte wie 0,99999 (99,999%) nahe genug an 1 um auch bei längeren Geschäftsprozessen eine hinreichend hohe Genauigkeit zu erzielen. Im Falle der „five nines availability“ ergibt sich bei der Berechnung der Gesamtverfügbarkeit eines Ausführungsplans eines Geschäftsprozesses mit 40 Prozessschritten eine Abweichung von weniger als 10^{-4} zwischen exaktem Wert und der Approximation.

Um einen Gesamtparameterwert $x^{\bullet}$ mittels einer Restriktion zu beschränken könnte eine Nebenbedingung der Form

$$x^{\bullet} \leq c^{\bullet} \text{ oder } x^{\bullet} \geq c^{\bullet}$$

eingeführt werden, wobei $c^{\bullet}$ eine Konstante ist, die den Gesamtparameterwert $x^{\bullet}$ beschränkt. Diese Form der Abbildung einer Restriktion für einen multiplikativen Gesamtparameter auf eine Nebenbedingung besitzt jedoch den Nachteil, dass der Gesamtparameterwert $x^{\bullet}$ nur approximiert ist. Über die zuvor betrachtete Linearisierung eines Produkts durch eine Logarithmierung kann die Restriktion jedoch exakt, d.h. auf den exakten Gesamtparameterwert angewendet, in das Modell integriert werden, denn es gilt:

$$c^{\bullet} \leq \prod_{i=1}^{n} \prod_{j=1}^{m_i} p^{\bullet}_{i,j}{}^{x_{i,j}} \Leftrightarrow \ln\left(c^{\bullet}\right) \leq \sum_{i=1}^{n} \sum_{j=1}^{m_i} \ln\left(p^{\bullet}_{i,j}\right) x_{i,j}$$

$$c^{\bullet} \geq \prod_{i=1}^{n} \prod_{j=1}^{m_i} p^{\bullet}_{i,j}{}^{x_{i,j}} \Leftrightarrow \ln\left(c^{\bullet}\right) \geq \sum_{i=1}^{n} \sum_{j=1}^{m_i} \ln\left(p^{\bullet}_{i,j}\right) x_{i,j}$$

Statt Restriktionen der Form $x^{\bullet} \leq c^{\bullet}$ oder $x^{\bullet} \geq c^{\bullet}$ werden daher für multiplikative Gesamtparameter Restriktionen in der folgenden Form verwendet:

$$\ln(c^{\bullet}) \leq \sum_{i=1}^{n}\sum_{j=1}^{m_i} \ln(p_{i,j}^{\bullet}) x_{i,j} \quad \text{oder} \quad \ln(c^{\bullet}) \geq \sum_{i=1}^{n}\sum_{j=1}^{m_i} \ln(p_{i,j}^{\bullet}) x_{i,j}$$

Minimaloperator-Parameter

Zur Aufnahme der Gesamtparameterwerte von Minimaloperator-Parametern wird für jeden Minimaloperator-Parameter eine zusätzliche Variable $x^{\min}$ eingeführt. Ein Gesamtparameterwert $x^{\min}$ wird durch Anwenden des Minimaloperators auf die Parameterwerte $p_{i,j}^{\min}$ des jeweiligen Minimaloperator-Parameters aller zur Ausführung ausgewählten Web Services erzeugt.

$$x^{\min} = \underset{i=1}{\overset{n}{Min}}\left(\sum_{j=1}^{m_i} p_{i,j}^{\min} x_{i,j} \right)$$

Um die Minimaloperation im linearen Optimierungsmodell abbilden zu können, wird für jeden Prozessschritt i eine Nebenbedingung eingeführt, welche besagt, dass der Gesamtparameter $x^{\min}$ höchstens so groß sein darf, wie der Parameterwert $p_{i,j}^{\min}$ des zur Ausführung des Prozessschritts i gewählten Web Services.

$$x^{\min} \leq \sum_{j=1}^{m_i} p_{i,j}^{\min} x_{i,j} \forall i = 1,...,n$$

Durch diese Nebenbedingungen ist der Gesamtparameter $x^{\min}$ nach oben durch den kleinsten Parameterwert $p_{i,j}^{\min}$ aller zur Ausführung von Prozessschritten ausgewählten Web Services beschränkt. Ein Summand der Zielfunktion des linearen Optimierungsproblems (s.u.) besteht aus dem Produkt des Gesamtparameters $x^{\min}$ und seinem Gewicht und gibt den Nutzenbeitrag an, der durch den Gesamtparameter $x^{\min}$ gestiftet wird. Da das Gewicht positiv ist und im Zuge der Optimierung der Nutzen maximiert wird, wird auch der Gesamtparameter $x^{\min}$ maximiert. Insgesamt nimmt daher $x^{\min}$ den größtmöglichen Wert an, der den kleinsten Parameterwert $p_{i,j}^{\min}$ aller zur Ausführung von Prozessschritten ausgewählten Web Services nicht übersteigt und ist daher exakt das Minimum der Parameterwerte $p_{i,j}^{\min}$ der zur Ausführung von Prozessschritten ausgewählten Web Services.

Um einen Gesamtparameterwert $x^{\min}$ mittels einer Restriktion nach unten zu beschränken kann eine Nebenbedingung der Form

$$x^{\min} \geq c^{\min}$$

eingeführt werden, wobei $c^{\min}$ eine Konstante ist, die den Gesamtparameterwert $x^{\min}$ nach unten beschränkt. Die Beschränkung eines Minimaloperator-Parameters nach oben durch eine Nebenbedingung der Form $x^{\min} \leq c^{\min}$ ist nicht vorgesehen, da in praxisrelevanten Einsatzsituationen des Optimierungsmodells Minimaloperator-Parameter, wie z.B. der maximal mögliche Durchsatz, stets maximiert werden sollen, da höhere Werte einen höheren Nutzen stiften und daher eine Beschränkung nach oben nicht sinnvoll wäre.

Erwähnenswert an dieser Stelle ist, dass die Variable $x^{\min}$ nicht in allen Fällen im Optimierungsmodell exakt den Gesamtparameterwert des Minimaloperator-Parameters annimmt. Wird der Gesamtparameterwert $x^{\min}$ in der Zielfunktion nicht gewichtet, so wird $x^{\min}$ während der Optimierung nicht maximiert und nimmt daher nicht unbedingt das Minimum der Parameterwerte $p_{i,j}^{\min}$ der zur Ausführung von Prozessschritten ausgewählten Web Services an. Im Optimierungsmodell wird der exakte Gesamtparameterwert $x^{\min}$ jedoch lediglich in der Zielfunktion zur Berechnung des Nutzenbeitrags benötigt. Da jedoch der Nutzenbeitrag im Falle der Nichtgewichtung des Gesamtparameterwerts $x^{\min}$ unabhängig des Wertes von $x^{\min}$ konstant bei 0 liegt, ist der Wert des Gesamtparameterwerts $x^{\min}$ für die Zielfunktion unerheblich. Im Falle einer Restriktion, die den Gesamtparameterwert $x^{\min}$ nach unten beschränkt, wird der exakte Wert des Gesamtparameterwertes $x^{\min}$ ebenfalls nicht benötigt. Durch die Nebenbedingungen zur Realisierung des Minimaloperators wird $x^{\min}$ nach oben beschränkt und durch die Nebenbedingung zur Realisierung der Restriktion nach unten. $x^{\min}$ kann sich daher nur in den durch die Nebenbedingungen definierten oberen und unteren Schranken bewegen und sorgt hierdurch dafür, dass nur solche Lösungen im Optimierungsmodell generiert werden können, welche die definierten Restriktionen erfüllen.

Zielfunktion

Der zu maximierende Zielfunktionswert des linearen Optimierungsproblems ist der Gesamtnutzenwert, der durch die Gesamtparameterwerte des Ausführungsplans gestiftet wird und berechnet sich als gewichtete Summe der Gesamtparameterwerte des gewählten Ausführungsplans. Die Gewichte der Gesamtparameterwerte von additiven Parametern werden mit w^{+} bezeichnet, die Gewichte der Gesamtparameter von multiplikativen Parametern mit $w^{\bullet}$ und die Gewichte der Gesamtparameterwerte von Minimaloperator-Parametern mit $w^{\min}$. Existieren im linearen Optimierungsmodell k^{+} additive Parameter, $k^{\bullet}$ multiplikative Parameter und $k^{\min}$ Minimaloperator-Parameter ($k^{+} + k^{\bullet} + k^{\min} = K$), so ergibt sich die zu maximierende Zielfunktion $F(\vec{x})$ in Abhängigkeit vom Vektor $\vec{x}$ der Entscheidungsvariablen als

$$F(\vec{x}) = \sum_{l=1}^{k^{+}} w_l^{+} x_l^{+} + \sum_{l=1}^{k^{\bullet}} w_l^{\bullet} x_l^{\bullet} + \sum_{l=1}^{k^{\min}} w_l^{\min} x_l^{\min}$$

Die Multiplikation eines Gewichts w mit einem Gesamtparameterwert x transformiert den Gesamtparameterwert in Nutzeneinheiten. Sind für einen Parameter höhere Parameterwerte besser, so ist das Gewicht w positiv und die Nutzeneinheiten gehen positiv in die Zielfunktion ein. Sind hingegen für einen Parameter niedrigere Parameterwerte besser, so ist das Gewicht w negativ und die Nutzeneinheiten gehen negativ in die Zielfunktion ein. Das Gewicht definiert den Zusatznutzen (bzw. Nutzenverlust), der durch eine zusätzliche Einheit des Gesamtparameterwertes erzeugt wird. Durch die Gewichte wird festgelegt wie viele Einheiten der verschiedenen Gesamtparameter den gleichen Nutzen stiften. Hierdurch legen die Gewichte die Wichtigkeit der einzelnen Parameter, bzw. Gesamtparameterwerte während des Optimierungsvorgangs fest, d.h. sie definieren nach welchen Kriterien der konkrete Ausführungsplan eines Geschäftsprozesses optimiert wird. Der Zielfunktionswert (Gesamtnutzen) entsteht als

gewichtete Summe der Gesamtparameterwerte und wird während des Optimierungsvorgangs maximiert.

3.2 Heuristik zur Lösung des mathematischen Modells

Bei dem in Kapitel 3.1.2 formulierten linearen Optimierungsproblem handelt es sich um ein gemischt-ganzzahliges lineares Optimierungsproblem, da alle Entscheidungsvariablen $x_{i,j}$ binär sind und alle Variablen $x^+, x^\bullet, x^{min}$ reellwertig. Das Vorhandensein von Variablen mit Ganzzahligkeitsbedingung erfordert zur exakten Lösung des Optimierungsproblems die Anwendung entsprechender Lösungsalgorithmen wie z.B. Branch&Bound (vgl. Kapitel 2.2.4). Mit zunehmender Problemgröße und Stärke der Einschränkung des möglichen Lösungsraums durch Restriktionen steigen die benötigten Rechenzeiten zur Lösung des Problems schnell stark an und verunmöglicht hierdurch die Verwendung eines exakten Lösungsverfahrens zur Laufzeit in einer BPE.

Um die benötigte Rechenzeit zu reduzieren wird ein heuristisches Lösungsverfahren, bestehend aus vier Schritten, eingesetzt:

1. Exakte Lösung des LP-relaxierten Problems durch einen Simplex-Algorithmus.
2. Konstruktion einer ersten zulässigen Lösung mittels eines Backtracking-Algorithmus.
3. Anwendung eines reinen Verbesserungsverfahrens zur (lokalen) Optimierung.
4. Anwendung der Metaheuristik Simulated Annealing zur (möglichst globalen) Optimierung.

Im ersten Schritt wird das Optimierungsproblem durch das Fallenlassen der Ganzzahligkeitsbedingung für die Entscheidungsvariablen $x_{i,j}$ relaxiert. Das relaxierte Optimierungsproblem lässt sich mit einem Algorithmus wie dem Simplex-Algorithmus (vgl. Kapitel 2.2.3) mit einem sehr geringen Aufwand an Rechenzeit exakt lösen. Die hierbei ermittelte Lösung stellt jedoch keine gültige Lösung für das ursprüngliche, unrelaxierte Optimierungsproblem dar. In einem zweiten Schritt wird daher mittels einem Backtracking-Algorithmus eine Lösung konstruiert, welche sowohl die Ganzzahligkeitsbedingung des unrelaxierten Optimierungsproblems erfüllt, als auch alle im Optimimierungsproblem formulierten Restriktionen. Erfüllt die gefundene Lösung nicht gewisse Anforderungen an ihre Lösungsgüte, so wird versucht die Lösung in zwei weiteren Schritten weiter zu verbessern. In einem dritten Schritt wird durch das Ersetzen von Web Services im Ausführungsplan versucht schnell ein lokales Optimum der zu maximierenden Zielfunktion zu erreichen. Anschließend wird in einem vierten Schritt die Metaheuristik Simulated Annealing (vgl. 2.2.6) angewendet um ein evtl. erreichtes lokales Optimum wieder verlassen und möglicherweise das globale Optimum erreichen zu können.

Die einzelnen Schritte des heuristischen Lösungsverfahrens werden im Folgenden ausführlich beschrieben.

3.2.1 Schritt 1: Lösung des relaxierten Problems

Um einen zulässigen Ausführungsplan für einen Geschäftsprozess zu konstruieren, muss an jeder Position des Ausführungsplans ein Web Service zur Ausführung der benötigten Funktionalität des entsprechenden Prozessschrittes gewählt werden. Der hierbei entstehende Ausführungsplan muss alle definierten Restriktionen hinsichtlich

seiner Gesamtparameter erfüllen und zugleich einen möglichst hohen Zielfunktionswert besitzen. Bei der Konstruktion eines solchen Ausführungsplans stellt sich die Frage, in welche Reihenfolge Web Services an den jeweiligen Positionen des Ausführungsplans eingesetzt werden sollen, um möglichst schnell einen Ausführungsplan zu erhalten der alle Restriktionen erfüllt und zudem einen hohen Zielfunktionswert besitzt. Wurde ein zulässiger Ausführungsplan konstruiert und der zugehörige Zielfunktionswert bestimmt, so stellt sich weiterhin die Frage, welche Lösungsgüte der konstruierte Ausführungsplan besitzt, d.h. wie nahe der durch den Ausführungsplan erreichte Zielfunktionswert bereits am Optimum, dem maximalen Zielfunktionswert, liegt.

Um einen möglichst optimalen zulässigen Ausführungsplan effektiv konstruieren zu können, werden Abschätzungen benötigt, welche Web Services an den einzelnen Positionen des Ausführungsplans tendenziell besser geeignet sind, um eine optimale zulässige Lösung zu konstruieren und in welcher Größenordnung sich der maximale Zielfunktionswert, d.h. der Zielfunktionswert der optimalen Lösung, in etwa bewegen wird. Mithilfe dieser Informationen können bei der Konstruktion eines zulässigen Ausführungsplans gezielt zuerst solche Web Services eingesetzt werden, bei denen die Wahrscheinlichkeit, einen zulässigen Ausführungsplan zu erhalten, der zudem noch möglichst optimal ist, möglichst hoch ist. Hierdurch kann bereits nach kurzer Rechenzeit ein Ausführungsplan konstruiert werden, der sowohl zulässig als auch von hoher Lösungsgute ist. Die Lösungsgüte lässt sich mittels der Information über die Größenordnung des Zielfunktionswerts der optimalen Lösung abschätzen. Hierdurch kann vermieden werden, dass eine unnötige weitere Optimierung der Lösung unternommen wird, obwohl die Lösung bereits hinreicht gut oder gar bereits optimal ist.

Informationen darüber, welche Web Services an den einzelnen Positionen des Ausführungsplans mit einer höheren Wahrscheinlichkeit zu einer zulässigen und möglichst optimalen Lösung führen, sowie eine Abschätzung des Zielfunktionswertes der optimalen Lösung können durch die exakte Lösung des relaxierten Problems ermittelt werden.

Das relaxierte Problem entsteht aus dem ursprünglichen Optimierungsproblem durch das Fallenlassen der Ganzzahligkeitsbedingung der binären Entscheidungsvariablen $x_{i,j}$. Entscheidungsvariablen $x_{i,j}$, welche zuvor nur den Wert 0 oder 1 (an Position i des Ausführungsplan wird Web Service j der Kategorie i verwendet) annehmen konnten, können nach der Relaxation beliebige reelle Werte aus dem Intervall [0;1] annehmen. Da hierdurch alle Entscheidungsvariablen reellwertig geworden sind, lässt sich das relaxierte Optimierungsproblem sehr schnell mit einem Verfahren wie dem Simplex-Algorithmus exakt lösen.

Durch die Relaxation ändert sich die Bedeutung der Entscheidungsvariablen $x_{i,j}$. Vor der Durchführung der Relaxation bedeutete $x_{i,j}=1$, dass an Position i des Ausführungsplans der Web Service j der Kategorie i verwendet wird und ein Wert von $x_{i,j}=0$, dass der entsprechende Web Service an dieser Position nicht verwendet wird. Nach der Durchführung der Relaxation können in einem Ausführungsplan an einer Position mehrere Web Services anteilig ausgeführt werden, d.h. es ist z.B. möglich an einer Position des Ausführungsplans einen Web Service zu 75% auszuführen und einen anderen zu 25%. In die Berechnung der Gesamtparameterwerte des Ausführungsplans gehen die Parameterwerte der anteilig ausgeführten Web Services gemäß ihrem Anteil an der Ausführung ein. Dies entspricht der Ausführung von virtuellen, nicht

real existenten Web Services an den Positionen des Ausführungsplans, deren Parameterwerte durch die Linearkombination der Parameterwerte der realen Web Services gemäß ihrer anteiligen Ausführung entstehen. Die Information darüber, aus welchen Web Services und zu welchen Anteilen der virtuelle Web Service an einer Position des Ausführungsplans besteht, kann als Abschätzung dafür verwendet werden, welche Web Services an einer Position des Ausführungsplans mit einer höheren Wahrscheinlichkeit an einer zulässigen optimalen Lösung des unrelaxierten Optimierungsproblems beteiligt sein könnten, als andere. Im genannten Beispiel der anteiligen Ausführung eines Web Services zu 75% und eines anderen Web Services zu 25% an derselben Position, kann davon ausgegangen werden, dass der Web Service mit der anteiligen Ausführung von 75% mit einer höheren Wahrscheinlichkeit Bestandteil der optimalen Lösung des unrelaxierten Optimierungsproblems ist, als jener mit der anteiligen Ausführung von lediglich 25% oder anderer Web Services, die an dieser Position im relaxierten Problem gar nicht zur Ausführung gebracht werden. Als Abschätzung der Wahrscheinlichkeit, mit welcher ein Web Service Bestandteil der optimalen Lösung des unrelaxierten Problems ist, wird der Anteil verwendet, mit der ein Web Service in der optimalen Lösung des relaxierten Problems ausgeführt wird.

Die optimale Lösung des relaxierten Problems erlaubt darüber hinaus eine Abschätzung des Zielfunktionswertes der optimalen Lösung des unrelaxierten Problems. Da durch die Relaxation der Ausführungsplan aus virtuellen Web Services besteht, die aus realen Web Services so konstruiert wurden, dass sie alle Restriktionen möglichst exakt erfüllen und den Zielfunktionswert maximieren, kann der Zielfunktionswert der optimalen Lösung des unrelaxierten Problems keinen höheren Zielfunktionswert aufweisen, da im unrelaxierten Problem nur reale, nicht im Sinne der Optimierung künstlich konstruierte Web Services verwendet werden können und diese daher die Restriktionen nicht so exakt erfüllen können und keinen so hohen Zielfunktionswert erreichen können, wie es den virtuellen Web Services des relaxierten Problems möglich ist. Der Zielfunktionswert der optimalen Lösung des relaxierten Problems stellt daher eine obere Schranke für den Zielfunktionswert der optimalen Lösung des unrelaxierten Problems dar.

Da die Höhe des Unterschiedes zwischen dem Zielfunktionswert der optimalen Lösung des relaxierten und des unrelaxierten Problems unbekannt ist, kann die Güte eines zulässigen Ausführungsplans für das unrelaxierte Problem nicht exakt beurteilt werden. Mithilfe der oberen Schranke des Zielfunktionswerts ist es jedoch möglich eine Bedingung zur Termination der Optimierungsprozesse nachgelagerter Schritte der Heuristik zu formulieren. Unterschreitet die Abweichung zwischen dem Zielfunktionswert eines konstruierten Ausführungsplans und der oberen Schranke für den Zielfunktionswert einen gewissen Wert, z.B. 1%, so wird die Güte des konstruierten Ausführungsplans als hinreichend hoch eingestuft und es werden keine weiteren Schritte zur Optimierung des Ausführungsplans mehr unternommen.

Ein weiterer Fall, in welchem unnötige Optimierungsschritte vermieden werden können, tritt beim Erkennen der Unlösbarkeit des relaxierten Optimierungsproblems auf. Existiert für das relaxierte Optimierungsproblem keine Lösung, so impliziert dies, dass für das unrelaxierte Optimierungsproblem ebenfalls keine Lösung existieren kann. Die Umkehrung gilt jedoch nicht, d.h. es ist durchaus möglich dass eine Lösung für das relaxierte Problem existiert, auch wenn für das unrelaxierte Problem keine Lösung existiert. Wird jedoch die Unlösbarkeit des relaxierten Optimierungsproblems erkannt, so kann die Heuristik abgebrochen werden, da an dieser Stelle schon mit Si-

cherheit bekannt ist, dass keine zulässige Lösung für das unrelaxierte Optimierungsproblem konstruiert werden kann.

3.2.2 Schritt 2: Konstruktion einer gültigen Lösung

Nachdem im ersten Schritt durch die Lösung des relaxierten Optimierungsproblems Informationen darüber gewonnen wurden, welche Web Services an den einzelnen Positionen des Ausführungsplans mit einer höheren Wahrscheinlichkeit zu einer zulässigen und möglichst optimalen Lösung führen, sowie eine obere Schranke für den optimalen Zielfunktionswert gewonnen wurde, wird nun im zweiten Schritt eine konkrete zulässige Lösung, d.h. ein zulässiger Ausführungsplan für den Geschäftsprozess erzeugt.

Backtracking-Verfahren

Die grundlegende Systematik, die hierbei verfolgt wird, ist die des *Backtracking-Algorithmus* (engl. Rückverfolgungsalgorithmus) [72]. Der Backtracking-Algorithmus ist ein Verfahren zur schrittweisen systematischen Lösungssuche, der in der Lage ist nicht erfolgreiche Ansätze der Lösungskonstruktion zu erkennen, zu verwerfen und an einer zurückliegenden Stelle der Lösungssuche erneut mit einem alternativen Ansatz fortzufahren. Der Backtracking-Algorithmus endet mit dem Auffinden der ersten zulässigen Lösung oder dem Erkennen, dass keine zulässige Lösung existiert, nachdem alle Möglichkeiten zur Konstruktion einer Lösung erfolglos angewendet wurden.

Die Konstruktion eines vollständigen Ausführungsplans erfolgt durch das schrittweise Festlegen der Web Services, die zur Erfüllung der benötigten Funktionalitäten der Prozessschritte ausgeführt werden, in den jeweiligen Positionen des Ausführungsplans. Erfolgt (ohne Beschränkung der Allgemeinheit) die Festlegung der Web Services in den Positionen des Ausführungsplans in der Reihenfolge ihrer Ausführung, so wird zunächst der Web Service für Position 1 des Ausführungsplans festgelegt, danach der Web Service für Position 2, usw. usf., bis schließlich der letzten Position n ein Web Service zugewiesen wurde und damit ein vollständiger Ausführungsplan erzeugt wurde. An jeder Position i (i=1,...,n) kann hierbei aus den m_i alternativen Web Services der Kategorie i gewählt werden, wobei der gewählte Web Service durch seinen Index j_i ($j_i \in [1,...,m_i]$) in der Kategorie i repräsentiert wird.

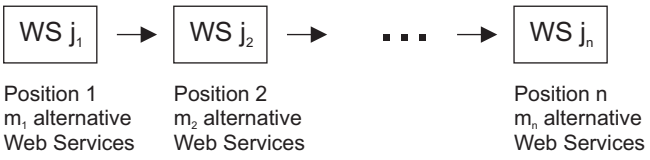

Abbildung 4 - Schematische Darstellung der Belegung eines Ausführungsplans

Das Backtracking-Verfahren beginnt an Position i=1 mit einem leeren Ausführungsplan, in dem noch keiner der n Positionen ein Web Service zugewiesen wurde. Da der aktuellen Position i noch kein Web Service zugewiesen wurde, wird an Position i der Web Service j_i=1, d.h. der erste der m_i alternativen Web Services der Kategorie i, gesetzt. Nach der Festsetzung des Web Services werden die Gesamtparameterwerte des aktuellen (und bisher unvollständigen) Ausführungsplans ermittelt und auf die Einhaltung der definierten Restriktionen überprüft. Werden alle Restriktionen erfüllt, so handelt es sich um einen bisher zulässigen Ausführungsplan und es wird zur nächsten Position (i=i+1) des Ausführungsplans fortgeschritten. An dieser Position wird wieder

der Web Service j_i=1 gesetzt, die Einhaltung der Restriktionen überprüft und im Falle der Erfüllung wieder zur Position i+1 fortgeschritten. Gelingt es, ohne die Restriktionen zu verletzten, bis zur Position n fortzuschreiten, so wurde ein vollständiger, zulässiger Ausführungsplan erzeugt und das Backtracking-Verfahren wird beendet.

Wird jedoch bei der Prüfung der Restriktionen nach der Besetzung einer Position i festgestellt, dass mindestens eine Restriktion verletzt wird, so ist es nicht nötig weitere Positionen i' mit i'>i des Ausführungsplans zu besetzen, da alle Ausführungspläne, die den aktuellen Ausführungsplan als Teillösung enthalten, d.h. an allen bisher besetzten i Positionen mit dem aktuellen Ausführungsplan identische Web Services aufweisen, die Restriktion(en) ebenfalls verletzen werden und damit keinen zulässigen Ausführungsplan darstellen können. Da der aktuelle Ausführungsplan nicht zu einem zulässigen, vollständigen Ausführungsplan ergänzt werden kann, muss die Entscheidung, für Position i den Web Service j_i zu verwenden, revidiert werden. An der betreffenden Position i wird der gewählte Web Service j_i solange durch den noch nicht gewählten Web Service j_i+1 der Kategorie i ersetzt, bis der aktuelle Ausführungsplan entweder wieder alle Restriktionen erfüllt oder alle m_i alternativen Web Services der Kategorie i erfolglos ausprobiert wurden. In letzterem Fall lässt sich durch keinen der m_i alternativen Web Services an Position i ein zulässiger Ausführungsplan erzeugen und es erfolgt ein Rückschritt zur vorherigen Position (i=i-1) des Ausführungsplans. An dieser Position ist das Vorgehen nun analog zum Fall der Verletzung von Restriktionen, d.h. an der betreffenden Position i wird der gewählte Web Service j_i solange durch den noch nicht gewählten Web Service j_i+1 der Kategorie i ersetzt, bis der aktuelle Ausführungsplan entweder alle Restriktionen erfüllt oder alle m_i alternativen Web Services der Kategorie i erfolglos ausprobiert wurden. Ist die Besetzung der Position durch einen Web Service in der Art möglich, dass alle Restriktionen erfüllt werden, so wird wieder zur nächsten Position (i=i+1) fortgeschritten. Kann eine solche Besetzung nicht gefunden werden, so erfolgt wieder ein Rückschritt zur vorhergehenden Position (i=i-1).

Das Verfahren endet entweder mit einem vollständigen, zulässigen Ausführungsplan an Position n oder aber an Position 1 mit einem leeren Ausführungsplan und der Feststellung, dass kein Web Service der Kategorie 1 als Teillösung eines zulässigen Ausführungsplans verwendet werden kann. Der letztere Fall ist identisch mit der Erkenntnis, dass es keinen zulässigen Ausführungsplan gibt und das Optimierungsproblem daher unlösbar ist.

Baumorientierte Betrachtung des Verfahrens

Das Vorgehen bei der Lösungssuche durch den Backtracking-Algorithmus soll im Folgenden anhand eines Lösungsbaums verdeutlicht werden. Abbildung 5 zeigt den Lösungsbaum des Backtracking-Verfahrens für einen Geschäftsprozess mit 3 Prozessschritten, wobei für Prozessschritt 1 m_1=2 alternative Web Services zu dessen Durchführung existieren, für Prozessschritt 2 m_2=3 alternative Web Services und für Prozessschritt 3 m_3=2 alternative Web Services.

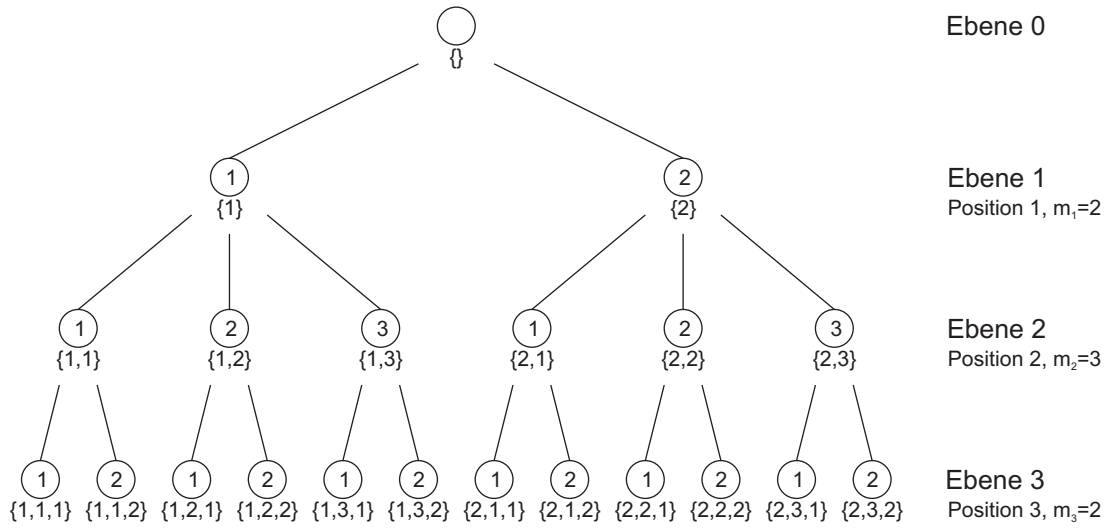

Abbildung 5 - Lösungsbaum eines Backtracking-Verfahrens

Einer Ebene i im Lösungsbaum entspricht der Festsetzung eines Web Services an Position i des Ausführungsplans. Der konkret gewählte Web Service wird durch einen Knoten repräsentiert und ist mit seinem Index j_i beschriftet. Der Grad eines Knotens in Ebene i entspricht der Anzahl m_{i+1} der für die nächste Position i+1 verfügbaren alternativen Web Services der Kategorie i+1. Unter jedem Knoten ist der beim Erreichen des Knotens aktuelle Ausführungsplan als geordnete Menge aller bereits im Ausführungsplan gesetzten Web Services angegeben. Die Blätter entsprechen vollständigen Ausführungsplänen. Die Wurzel repräsentiert die Ausgangskonfiguration des Backtracking-Verfahrens mit einem leeren Ausführungsplan.

Dem Vorgehen des zuvor beschriebenen Backtracking-Verfahrens entspricht eine Tiefensuche im Lösungsbaum, beginnend in der Wurzel. In jedem Knoten wird der aktuelle Ausführungsplan auf die Einhaltung aller Restriktionen geprüft. Werden alle Restriktionen erfüllt, so wird zu dem Kindknoten mit dem kleinsten Index verzweigt, welcher noch nicht besucht wurde. Verletzt der aktuelle Ausführungsplan eines Knotens mindestens eine Restriktion, so wird zum Vaterknoten verzweigt. Das Verfahren endet mit dem Erreichen eines Blattes, dessen Ausführungsplan alle Restriktionen erfüllt, oder dem neuerlichen Erreichen der Wurzel nach dem Besuch aller Kindknoten. Im ersten Fall wurde eine zulässige, vollständige Lösung gefunden, die dem aktuellen Ausführungsplan des Blattes entspricht. Im zweiten Fall existiert kein zulässiger Ausführungsplan.

Die folgende Abbildung 6 zeigt eine Formulierung des Verfahrens in Form von Pseudo-Code.

```
i=1                       // Aktuelle Position im Ausführungsplan
AP={0, 0, ..., 0}         // Anfänglicher Ausführungsplan AP = n
                          // Positionen mit Wert 0
ende=false                // Flag für Abbruch des Verfahrens
while (not ende) {
      repeat {
            // Falls nicht alle Web Services in Kategorie i er-
            // schöpft, nächsten Web Service an Position i setzen.
            if (AP[i]<mi) AP[i]++
      } until (Ausführungsplan AP zulässig oder AP[i]=mi)
      if (Ausführungsplan AP unzulässig) {
            AP[i]=0
            if (i>1)
                  i--         // Schritt zurück, falls nicht Anfang
            else
                  ende=true   // Abbruch, da keine zulässige Lösung
      } else {
            if (i<n)
                  i++         // Schritt vor, falls nicht Ende
            else
                  ende=true   // Abbruch, da zulässige Lösung
      }
}
```

Abbildung 6 - Pseudo-Code des Backtracking-Verfahrens

Nachdem das im Pseudo-Code formulierte Verfahren terminiert hat, befindet sich im Array AP entweder ein zulässiger Ausführungsplan oder ein leerer Ausführungsplan, für den Fall, dass kein zulässiger Ausführungsplan existiert. Im ersten Fall findet sich an jeder Position i des Arrays AP der Index j_i des für diese Position gesetzten Web Services. Im zweiten Fall befindet sich an allen Positionen des Arrays der Wert 0.

Laufzeitverhalten

Bezüglich der Laufzeit des Verfahrens stellt ein Optimierungsproblem ohne Restriktionen den günstigsten Fall dar. Da durch keine Wahl eines Web Services eine Restriktion verletzt werden kann, endet das Verfahren stets nach n Zuweisungen von Web Services an die n Positionen des Ausführungsplans im ersten Blatt des Lösungsbaums. In diesem Fall ist die Laufzeit linear zur Länge n des Geschäftsprozesses, d.h. O(n).

Das schlechteste Laufzeitverhalten wird im Falle eines unlösbaren Optimierungsproblems (impliziert das Vorhandensein von Restriktionen) erreicht, bei welchem die Unzulässigkeit eines Ausführungsplans erst in den Blättern des Lösungsbaums erkannt werden kann. In diesem Fall werden bei der Suche nach einer zulässigen Lösung alle Blätter des Lösungsbaums traversiert, d.h. alle möglichen vollständigen Ausführungspläne für das Optimierungsproblem erzeugt und auf Zulässigkeit überprüft. Da in jeder Position i des Ausführungsplans m_i alternative Web Services verwendet werden können, ergibt sich die Anzahl der besuchten Blätter, bzw. die Anzahl der konstruierten und untersuchten, vollständigen Ausführungspläne als $m_1 \cdot m_2 \cdot \ldots \cdot m_n = \prod_{i=1}^{n} m_i$. Um alle diese Ausführungspläne zu konstruieren müssen alle $1 + m_1 + m_1 \cdot m_2 + m_1 \cdot m_2 \cdot m_3 + \ldots + m_1 \cdot m_2 \cdot \ldots \cdot m_n = 1 + \sum_{i=1}^{n} \prod_{j=1}^{i} m_i$ Knoten des Lösungsbaums traversiert werden. Ist der höchste Grad eines Knotens im Lösungsbaum

m ($m = \max_{i=1}^{n}(m_i)$), so lässt sich die Anzahl der zu besuchenden Knoten im Lösungsbaum nach oben durch $1 + m^1 + m^2 + ... + m^n = \sum_{i=0}^{n} m^i$ abschätzen. Das Backtracking-Verfahren besitzt daher im schlechtesten Fall mit $O(m^n)$ eine exponentielle Laufzeit.

Optimierung des Verfahrens

Die Auswahl der Web Services für die einzelnen Positionen des Ausführungsplans geschieht im geschilderten Verfahren in aufsteigender Reihenfolge ihrer Indizes in ihrer jeweiligen Kategorie. Dieses Auswahlverfahren berücksichtigt weder ob ein Web Service in besonderer Weise dazu geeignet ist evtl. bestehende Restriktionen zu erfüllen, noch welchen Nutzen, bzw. Zielfunktionsbeitrag, durch diesen Web Service gestiftet wird. Dies führt dazu, dass der konstruierte zulässige Ausführungsplan weder einen besonders hohen Zielfunktionswert aufweist, noch dass er im Falle von bestehenden Restriktionen besonders schnell gefunden wird. Da jedoch gerade im Falle von bestehenden Restriktionen das Laufzeitverhalten exponentiell sein kann, ist es nötig die benötigte Rechenzeit zur Konstruktion eines zulässigen Ausführungsplans zu reduzieren. Aus diesen Gründen ist eine Optimierung des Verfahrens in zweierlei Hinsicht erstrebenswert. Einerseits sollte nicht ein beliebiger zulässiger Ausführungsplan konstruiert werden, sondern ein möglichst optimaler Ausführungsplan, d.h. ein Ausführungsplan mit besonders hohem Zielfunktionswert, und andererseits sollten bei der Konstruktion des Ausführungsplans eventuell bestehende Restriktionen berücksichtigt werden, sodass durch geschickte Auswahl von Web Services möglichst schnell ein zulässiger Ausführungsplan konstruiert werden kann.

Durch die Lösung des relaxierten Optimierungsproblems im ersten Schritt der Heuristik wurden Informationen darüber gewonnen, welche Web Services an den einzelnen Positionen des Ausführungsplans mit einer höheren Wahrscheinlichkeit zu einer zulässigen und möglichst optimalen Lösung führen. Diese Informationen lassen sich nun verwenden um eine Optimierung des Backtracking-Verfahrens zu ermöglichen. Die Optimierung wird dadurch erreicht, dass die Web Services in den einzelnen Kategorien und die Kategorien selbst vor Anwendung des Backtracking-Verfahrens gemäß dieser Informationen sortiert werden und dadurch während des Konstruktionsprozesses durch das Backtracking-Verfahren in einer möglichst geschickten Reihenfolge verarbeitet werden. Hierdurch kann sowohl die Güte eines konstruierten Ausführungsplans als auch die benötigte Rechenzeit zu dessen Konstruktion im Falle von bestehenden Restriktionen drastisch verbessert werden.

Der erste Schritt der Optimierung besteht in der Sortierung der Web Services in absteigender Reihenfolge ihrer anteiligen Ausführung $x_{i,j}$ in der optimalen Lösung des relaxierten Problems. Werden zwei Web Services zu gleichen Teilen ausgeführt, so werden diese nach ihrer *Qualität* absteigend sortiert (siehe Beispiel in Tabelle 3). Die Qualität eines Web Services ist eine imaginäre Größe und berechnet sich als Zielfunktionswert eines Ausführungsplans, der an jeder Position den Web Service ausführt, dessen Qualität berechnet wird.

Kategorie 1 (i=1)	$x_{1,j}$	Qualität
Web Service 1 (j_1=1)	0,75	5
Web Service 2 (j_1=2)	0,25	3
Web Service 3 (j_1=3)	0	7
Web Service 4 (j_1=4)	0	4
Web Service 5 (j_1=5)	0	2

Tabelle 3 - Beispiel der Sortierung von Web Services anhand anteiliger Ausführung und Qualität

Die anteilige Ausführung $x_{i,j}$ eines Web Services j_i wird hierbei als Abschätzung der Wahrscheinlichkeit, mit welcher ein Web Service Bestandteil der optimalen Lösung des unrelaxierten Problems ist, verwendet. Werden im Verlauf des Backtracking-Verfahrens die Positionen des Ausführungsplans in der Reihenfolge aufsteigender Indizes j_i mit Web Services besetzt, so werden aufgrund der Sortierung zuerst solche Ausführungspläne konstruiert, die Web Services verwenden, die zu einer hohen Wahrscheinlichkeit Bestandteil des optimalen Ausführungsplans sind. Da bei der Lösung des relaxierten Problems auch evtl. bestehende Restriktionen berücksichtigt wurden, führt die Sortierung darüber hinaus auch dazu, dass zuerst solche Ausführungspläne konstruiert werden, die nicht nur zu einem hohen Zielfunktionswert führen, sondern auch in besonderer Weise dazu geeignet sind die Restriktionen zu erfüllen und daher sehr schnell zu dem Auffinden eines zulässigen Ausführungsplans führen. Bei gleicher anteiliger Ausführung $x_{i,j}$ werden aufgrund der Sortierung nach der Qualität solche Web Services zuerst verwendet, deren Verwendung einen möglichst hohen Zielfunktionsbeitrag erwarten lässt.

In einem zweiten Optimierungsschritt erfolgt eine aufsteigende Sortierung der Kategorien nach der Anzahl der Web Services, die in der jeweiligen Kategorie in dem Ausführungsplan der optimalen Lösung des relaxierten Problems anteilig ausgeführt werden, d.h. nach der Anzahl enthaltener Web Services mit $x_{i,j} \neq 0$. Enthalten mehrere Kategorien die gleiche Anzahl von Web Services mit $x_{i,j} \neq 0$, so werden diese absteigend nach dem maximalen Wert $x_{i,j}$ der in ihnen enthaltenen Web Services sortiert. Ein Beispiel für eine solche Sortierung kann Tabelle 4 entnommen werden. Die Sortierreihenfolge ist dort durch die Abfolge der Kategorien von links nach rechts dargestellt.

Web Service	Kategorie 2	Kategorie 4	Kategorie 5	Kategorie 1	Kategorie 3
Web Service 1 (j_i=1)	1	1	0,9	0,75	0,8
Web Service 2 (j_i=2)	0	0	0,1	0,25	0,15
Web Service 3 (j_i=3)	0	0	0	0	0,05
Web Service 4 (j_i=4)	0	0	0	0	0
Web Service 5 (j_i=5)	0	0	0	0	0

Tabelle 4 - Beispiel der Sortierung von Kategorien anhand der anteiligen Ausführung enthaltener Web Services

Entgegen dem eingangs beschriebenen, grundlegenden Backtracking-Verfahren, weist das optimierte Backtracking-Verfahren den Positionen des Ausführungsplans die Web Services nicht mehr in der Reihenfolge ihrer Ausführung zu (Position 1, Position 2, ..., Position n), sondern in der Reihenfolge, die durch die Sortierung der Kategorien erzeugt wurde (im Beispiel: Position 2, Position 4, Position 5, Position 1, Position 3). Da die anteilige Ausführung $x_{i,j}$ eines Web Services als Abschätzung der Wahrscheinlichkeit verwendet wird, mit welcher ein Web Service Bestandteil der optimalen Lö-

sung des unrelaxierten Problems ist, führt dies dazu, dass zuerst den Positionen des Ausführungsplans Web Services zugewiesen werden, für die mit hoher Wahrscheinlichkeit der Web Service bekannt ist, der im optimalen Ausführungsplan enthalten sein wird. Später besetzte Positionen verfügen über eine höhere Anzahl von Web Services, die im relaxierten Modell anteilig ausgeführt werden. Dies bedeutet, dass an diesen Positionen die Wahl des optimalen Web Services unsicherer ist und im Verlauf des Backtracking-Verfahrens mit einer höheren Wahrscheinlichkeit wieder revidiert werden muss.

Für das Laufzeitverhalten des Backtracking-Verfahrens ist es günstiger unsichere Entscheidungen erst möglichst spät und damit nah an den Blättern des Lösungsbaums zu treffen, da hierdurch die Anzahl zu traversiernder Knoten minimiert wird, bis ein zulässiger Ausführungsplan gefunden werden kann. Im schlimmsten Fall könnte sonst eine falsche Entscheidung bereits durch den Besuch des ersten Kindknotens der Wurzel im Lösungsbaum getroffen werden, jedoch erst in einem Blatt des entsprechenden Teilbaums erkannt werden können. Bis die falsche Entscheidung durch den Besuch eines anderen Kindknotens der Wurzel revidiert würde, müssten alle Blätter des Teilbaums traversiert werden. Weitaus günstiger ist es hingegen eine falsche Entscheidung erst mit dem Besuch eines Blattes zu treffen und zu erkennen. In diesem Fall genügt ein Rückschritt zum Vaterknoten des Blattes um die falsche Entscheidung zu revidieren.

Die Zuweisung von Web Services an den Positionen des Ausführungsplans in der Reihenfolge der geschätzten Sicherheit der Entscheidung für einen Web Service führt genau zu einer solchen Verlagerung potentiell falscher Entscheidungen in die unteren Ebenen des Lösungsbaums und damit nahe an die Blätter. Hierdurch wird die Anzahl der Konstruktionsschritte zum Auffinden einer zulässigen Lösung minimiert. Dies bedeutet vor allem in Anbetracht der evtl. exponentiellen Laufzeit im Falle des Vorhandenseins von Restriktionen ein hohes Einsparungspotential für die benötigte Rechenzeit und rechtfertigt den zusätzlichen Aufwand der Vorsortierung der Web Services und Kategorien vor dem Start des Backtracking-Verfahrens. Eine Untersuchung der Heuristik hat gezeigt, dass es hierdurch oft möglich ist auch beim Vorhandensein schwierig zu erfüllender Restriktionen ein nahezu lineares Laufzeitverhalten zu erzeugen und zudem eine zulässige Lösung hoher Güte zu erhalten.

3.2.3 Schritt 3: Verbesserung durch Suche eines lokalen Optimums

Durch Anwendung des Backtracking-Verfahrens in Schritt 2 der Heuristik wurde ein zulässiger Ausführungsplan als Lösung des unrelaxierten Optimierungsproblems konstruiert. Obwohl durch die vorhergehende exakte Lösung des relaxierten Optimierungsproblems u.a. Abschätzungen darüber in den Konstruktionsprozess einfließen konnten, welche Web Services an den einzelnen Positionen des Ausführungsplans mit einer höheren Wahrscheinlichkeit zu einer möglichst optimalen Lösung des unrelaxierten Optimierungsproblems führen, kann nicht davon ausgegangen werden, dass es sich bei dem konstruierten Ausführungsplan um einen, im Sinne des Optimierungsproblems hinreichend guten oder gar optimalen Ausführungsplan handelt.

Da jedoch in Schritt 1 der Heuristik durch die exakte Lösung des relaxierten Optimierungsproblems eine obere Schranke für den Zielfunktionswert des unrelaxierten Optimierungsproblems bestimmt wurde, kann zumindest versucht werden die Güte des

konstruierten Ausführungsplans abzuschätzen. Ein Ausführungsplan wird als hinreichend gut eingestuft, wenn sein Zielfunktionswert nicht mehr als 1% von der ermittelten oberen Schranke abweicht. Wird erkannt, dass es sich um einen hinreichend guten Ausführungsplan handelt, so wird die Heuristik abgebrochen um unnötige weitere Schritte zu dessen Optimierung einzusparen und die Laufzeit der Heuristik nicht unnötig zu verlängern. Es sei jedoch angemerkt, dass diese Abschätzung der Güte beim Vorliegen von starken Restriktionen, d.h. Restriktionen welche den Anteil von zulässigen, vollständigen Ausführungsplänen an der Menge aller vollständiger Ausführungspläne stark reduziert, zunehmend ungenauer wird, da sich hier der Zielfunktionswert der optimalen Lösung des unrelaxierten und des relaxierten Optimierungsproblems erheblich unterscheiden können.

Weicht der Zielfunktionswert des ermittelten Ausführungsplans mehr als 1% von der ermittelten oberen Schranke ab, so wird davon ausgegangen, dass es sich nicht bereits um einen hinreichend guten oder gar den optimalen Ausführungsplan handelt. Um den suboptimalen Ausführungsplan möglichst schnell möglichst stark zu optimieren, wird im dritten Schritt der Heuristik versucht den Ausführungsplan derart zu verändern, dass möglichst schnell ein lokales Optimum des Zielfunktionswertes erreicht wird. Hierzu wird im aktuellen Ausführungsplan an einer zufälligen Position ein Web Service durch einen zufälligen anderen Web Service der entsprechenden Kategorie ersetzt. Ergibt sich hierbei eine Verbesserung des Zielfunktionswertes, so wird die Änderung in den Ausführungsplan übernommen, andernfalls wird sie verworfen. Dieser Schritt wird solange wiederholt, bis sich in einer bestimmten Anzahl aufeinander folgender Schritte keine weitere Verbesserung mehr erzielen lässt.

Das Verfahren lässt sich folgendermaßen als Pseudo-Code formulieren:

```
maxNoImp = u*n    // Maximale Anzahl Iterationen ohne Verbesserung
curNoImp = 0      // Aktuelle Anzahl Iterationen ohne Verbesserung
lastPos  = 0      // Letzte gewählte Position
while (curNoImp < MaxNoImp) {
      i = Zufällige Position aus [1;n] ohne lastPos
      lastPos = i
      curWS = AP[i]     // Aktuell an Position gesetzter Web Service
      curZF = Zielfunktionswert des Ausführungsplans AP
      WSRandomOrder[] = Web Services aus Kategorie i in zufälliger
                        Reihenfolge ohne curWS
      for (j=1; j<=WSRandomOrder.length; j++) {
            AP[i] = WSRandomOrder[j]
            if (Ausführungsplan AP zulässig) break
      }
      if (Ausführungsplan AP zulässig) {
            if (Zielfunktionswert des Ausführungsplans AP <= curZF) {
                  AP[i] = curWS
                  curNoImp++
            } else
                  curNoImp = 0
      } else {
            curNoImp++
      }
}
```

Abbildung 7 - Pseudo-Code des Verfahrens zur Suche eines lokalen Optimums

Zunächst wird die maximale Anzahl aufeinander folgender Optimierungsschritte festgelegt, die ausgeführt werden dürfen, ohne dass sich eine Verbesserung des Zielfunk-

tionswertes ergibt (maximale Anzahl erfolgloser Optimierungsschritte). Sie wird in Abhängigkeit von der Länge des Ausführungsplans festgesetzt, da bei längeren Ausführungsplänen mehr Positionen und damit auch Positionen für potentiell erfolglose Optimierungsschritte vorhanden sind. Konkret ergibt sich die maximale Anzahl erfolgloser Optimierungsschritte aus der Multiplikation der Länge des Ausführungsplans mit einem Faktor u. Der Standardwert, der für den Faktor u verwendet wird, ist 2, d.h. es werden maximal doppelt so viele aufeinander folgende, erfolglose Optimierungsschritte erlaubt wie Positionen im Ausführungsplan vorhanden sind. Die While-Schleife im Pseudo-Code umschließt den eigentlichen Optimierungsschritt und wird solange wiederholt, bis die zuvor definierte, maximale Anzahl aufeinander folgender, erfolgloser Optimierungsschritte erreicht wurde.

Zu Beginn des Optimierungsschrittes wird eine zufällige Position im Ausführungsplan bestimmt, die im Folgenden modifiziert werden soll. Hierbei wird ausgeschlossen, dass es sich um dieselbe Position im Ausführungsplan handelt, die bereits im letzten Optimierungsschritt gewählt wurde, um die Variabilität in den ausgewählten Positionen zu erhöhen. Um eine evtl. erfolglose Veränderung des Ausführungsplans rückgängig machen zu können, wird der momentan an der gewählten Position gesetzte Web Service festgehalten. Zudem wird der Zielfunktionswert des aktuellen Ausführungsplans festgehalten, um nach einer Veränderung des Ausführungsplans beurteilen zu können, ob sich der Zielfunktionswert verbessert hat.

Bevor die eigentliche Ersetzung des Web Services an der aktuellen Position vorgenommen werden kann, wird eine Liste aller alternativer Web Services für diese Position in einer zufälligen Reihenfolge erstellt. Diese Liste schließt den aktuell an dieser Position gesetzten Web Service nicht ein, da er keine Alternative zu sich selbst darstellt. Anschließend erfolgt die eigentliche Ersetzung des Web Services an der aktuellen Position. Hierzu werden der Reihe nach die in der Liste vorbereiteten alternativen Web Services, in der durch die Liste vorgegebenen, zufälligen Reihenfolge, an der aktuellen Position des Ausführungsplans eingesetzt. Nach jeder Ersetzung wird geprüft, ob der so entstandene neue Ausführungsplan zulässig ist. Die Ersetzung mit Alternativen endet, sobald der erste zulässige Ausführungsplan erzeugt wurde, oder alle Alternativen an der Position eingesetzt wurden, ohne dass hierbei ein zulässiger Ausführungsplan erzeugt werden konnte. Im zweiten Fall wird an der aktuellen Position wieder der ursprünglich gesetzte Web Service eingesetzt, die Anzahl der erfolglosen Optimierungsschritte erhöht und der aktuelle Optimierungsschritt beendet. Im ersten Fall wird der Zielfunktionswert des neuen zulässigen Ausführungsplans bestimmt und mit dem des ursprünglichen Ausführungsplans verglichen. Konnte der Zielfunktionswert gesteigert werden, so wird die Änderung im Ausführungsplan beibehalten und der aktuelle Optimierungsschritt beendet. Konnte der Zielfunktionswert hingegen nicht gesteigert werden, so wird an der aktuellen Position wieder der ursprünglich gesetzte Web Service eingesetzt, die Anzahl der erfolglosen Optimierungsschritte erhöht und der aktuelle Optimierungsschritt beendet.

Laufzeitverhalten

Im, bzgl. der Laufzeit, günstigsten Fall handelt es sich bei jedem unternommenen Optimierungsschritt um einen erfolglosen Optimierungsschritt und das Verfahren bricht nach der maximal erlaubten Anzahl aufeinander folgender, erfolgloser Optimierungsschritte ab. Da sich diese maximale Anzahl aus der doppelten Länge des Ausführungsplans (2*n) ergibt, verhält sich die Laufzeit in diesem Fall linear zur Länge des Ausführungsplans und entspricht daher O(n).

Im, bzgl. der Laufzeit ungünstigsten Fall, ist in der zulässigen Lösung, mit der das Verfahren startet, an jeder Position der schlechtmöglichste Web Service gesetzt. Um die Problemgröße nach oben abzuschätzen und die Betrachtung des Falls zu vereinfachen, sei angenommen, dass jede Kategorie m Web Services besitze, wobei m die größtmögliche Anzahl enthaltener Web Services aller tatsächlichen Kategorien sei, d.h. $m = \max_{i=1}^{n}(m_i)$ gelte. Da alle anderen Web Services einer Kategorie besser sind, muss das Verfahren im ersten Optimierungsschritt einen besseren Web Service finden und, da im zweiten Schritt eine andere Position gewählt werden muss, auch im zweiten Optimierungsschritt. Nach diesen ersten beiden Schritten wäre es möglich, dass zufällig immer wieder diese beiden Positionen in abwechselnder Reihenfolge gewählt werden und hierbei wiederum an jeder Position zufällig stets der jeweils nächstbeste Web Service gewählt wird. Weiterhin sei angenommen, dass alle Ausführungspläne zulässig sind. Nach 2*(m-1) zunächst erfolgreichen Optimierungsschritten ist an jeder dieser beiden Position der beste Web Service gesetzt. Werden nun wiederum stets diese beiden Positionen im Wechsel ausgewählt, so ergeben sich nur noch erfolglose Optimierungsschritte. Nach 2*n-1 erfolglosen Optimierungsschritten werde nun jedoch stets eine von diesen beiden Positionen abweichende Position gewählt, an der noch eine Verbesserung möglich ist. An dieser Position wird nun wiederum der jeweils nächstbeste Web Service gewählt. Wiederholt sich dieses Vorgehen stetig, so wird alle 2*n Optimierungsschritte eine Verbesserung erreicht und das Verfahren beginnt erneut mit der Durchführung von 2*n-1 erfolglosen Optimierungsschritten und einem erfolgreichen Optimierungsschritt. Dies wiederholt sich solange bis nach (n-2)*(m-1)*2*n Optimierungsschritten an allen Positionen der beste Web Service gesetzt ist. Danach terminiert das Verfahren nach weiteren 2*n erfolglosen Optimierungsschritten.

Es ergeben sich so nach dem Beginn der Verfahrens 2*(m-1) zunächst erfolgreiche Optimierungsschritte, gefolgt von (n-2)*(m-1)*2*n Optimierungsschritten bis an jeder Position der beste Web Service gesetzt ist und abschließend 2*n erfolglose Optimierungsschritte bis das Verfahren terminiert. Insgesamt ergeben sich so $2n^2(m-1)-4nm+6n+2m-2$ durchlaufene Optimierungsschritte. In diesem Fall besitzt das Verfahren mit $O(mn^2)$ eine Laufzeit die quadratisch von der Länge des Geschäftsprozesses und linear von der Anzahl der Web Services pro Kategorie abhängig ist und daher als quadratische Laufzeit bezeichnet werden kann.

Kritisch anzumerken ist an dieser Stelle, dass der beschriebene Fall der ungünstigsten Laufzeit mit nahezu an Sicherheit grenzender Wahrscheinlichkeit aus zweierlei Gründen nie eintreten wird. Erstens werden sowohl die Positionen, an denen Web Services ersetzt werden sollen, als auch die Web Services, die an diesen Positionen eingesetzt werden, zufällig ausgewählt. Die Wahrscheinlichkeit, dass sich hierbei exakt die geschilderten Reihenfolgen für die Auswahl der Positionen und der Web Services ergeben ist als verschwindend gering zu beurteilen. Zweitens wurde der Fall mit der Annahme konstruiert, dass an jeder Position des Ausführungsplans mit dem das Verfahren gestartet wird, jeweils der schlechtestmögliche Web Service gesetzt war. Dies wird jedoch durch die Optimierung bei der Konstruktion dieses Ausführungsplans durch das Backtracking-Verfahren in Schritt 2 effektiv verhindert.

3.2.4 Schritt 4: Simulated Annealing zur weiteren Verbesserung

Analog zum Beginn von Schritt drei der Heuristik erfolgt zu Beginn des vierten Schrittes der Heuristik zunächst eine Prüfung, ob der durch den vorhergehenden Schritt entstandene Ausführungsplan einen Zielfunktionswert besitzt, der nicht mehr als 1% von der oberen Schranke des theoretisch erreichbaren Zielfunktionswertes abweicht. Ist die Abweichung geringer, so wird der aktuelle Ausführungsplan als hinreichend gut eingestuft und die Heuristik abgebrochen. Übersteigt die Abweichung 1%, so wird der Ausführungsplan als suboptimal eingestuft und ein Versuch unternommen den Ausführungsplan in einem 4. Schritt der Heuristik zu verbessern.

Mangel des im vorhergehenden Schritt verwendeten Verfahrens

Im vorhergehenden dritten Schritt der Heuristik wurde durch die Modifikation von einzelnen Web Services an zufälligen Positionen des Ausführungsplans versucht, zulässige Nachbarlösungen[12] zu erzeugen, die einen höheren Zielfunktionswert besitzen. Nachbarlösungen wurden nur dann als neue Ausgangsbasis der Optimierungsversuche verwendet, wenn sie einen höheren Zielfunktionswert besaßen. Ein solches Verfahren läuft jedoch in Gefahr bei der Optimierung einen einmal erreichten Bereich um ein lokales Maximum der Zielfunktion nicht mehr verlassen zu können. Man überlege sich leicht, dass z.B. im Falle des Erreichens eines lokalen Maximums keine direkte Nachbarlösung existieren kann, die einen höheren Zielfunktionswert besitzt (sonst wäre sie kein lokales Maximum). Anschaulich kann dieses Verfahren mit einem Bergsteiger verglichen werden, der in einem Gebirge an einer zufälligen Startposition (Ausgangslösung) ausgesetzt wird und das Ziel hat, den höchsten Berg des Gebirges (globales Maximum) zu ersteigen. Bewegt sich der Bergsteiger bei jedem Schritt nur an die Stelle, bei der er den höchsten Höhenzugewinn erzielen kann (Nachbarlösung mit höchstem Zielfunktionswert), so erreicht er zwar den Gipfel eines Berges (ein Maximum), jedoch ist hierbei nicht sicher, ob es sich auch um den höchsten Berg des gesamten Gebirges (globales Maximum) handelt. Verfahren dieser Art werden als greedy (gierig) oder myopisch (kurzsichtig) bezeichnet, da sie lediglich Optimierungsschritte anwenden die sich direkt in einer gesteigerten Qualität der Lösung ausdrücken, aber hierbei nicht berücksichtigen, welche Auswirkung diese Entscheidung auf die noch erreichbare Lösungsgüte in den nachfolgenden Schritten hat.

Um zu verhindern nur ein lokales Maximum des Zielfunktionswertes erreichen zu können, ist es nötig ein Optimierungsverfahren anzuwenden, welches Schritte erlaubt, die zu einer vorübergehenden Verschlechterung des Zielfunktionswertes führen. Im Beispiel des Bergsteigers, müsste es dem Bergsteiger erlaubt werden auch Schritte bergab zu unternehmen, um ihn in die Lage zu versetzen, den aktuellen Berg verlassen und auf einen anderen, potentiell höheren Berg wieder aufsteigen zu können. Eine Metaheuristik, welche diesen Effekt berücksichtigt und eine zwischenzeitliche Verschlechterung des Zielfunktionswertes explizit vorsieht, ist *Simulated Annealing (SA)*. Simulated Annealing wird daher im vierten Schritt der Heuristik verwendet um den aktuellen, als suboptimal eingestuften Ausführungsplan weiter zu verbessern.

[12] Eine Nachbarlösung eines Ausführungsplans entsteht dadurch, dass an einer einzigen Position des Ausführungsplans ein anderer Web Service gesetzt wird. Veränderungen eines Ausführungsplans, die mehr als eine Position betreffen, erzeugen keine Nachbarlösung.

Simulated Annealing

Die Bezeichnung Simulated Annealing (simulierte Abkühlung) nimmt Bezug auf einen physikalischen Effekt, der bei der Abkühlung von Metallen auftritt. Bei hoher Temperatur bewegen sich die Atome in einem Metallstück stark hin und her. Kühlt das Metallstück ab, so nimmt der Bewegungsradius der Atome ab. Tritt die Abkühlung langsam ein, so ordnen sich die Atome so an, dass sie eine Anordnung mit möglichst niedrigem Energieniveau einnehmen. Tritt allerdings eine zu schnelle Abkühlung ein, so haben die Atome nicht die nötige Zeit eine Anordnung mit tatsächlich minimalem Energieniveau einnehmen zu können und das Metallstück verbleibt in einem, energetisch gesehen, lokalen Minimum, statt dem möglichen globalen Minimum. Die Idee, zufällige Veränderungen (Atombewegungen) zuzulassen, aber deren akzeptierte Größe langsam zu reduzieren (abzukühlen), damit sich ein gewisser Zustand (globales Minimum) einstellt, liegt dem Verfahren Simulated Annealing zugrunde.

Simulated Annealing wird auch als Metastrategie bezeichnet [26], da das Grundprinzip zur Steuerung des Optimierungsprozesses auf eine Vielzahl von Problemen angewendet werden kann.

Prinzipiell geht Simulated Annealing so vor, dass, ausgehend von einer zulässigen Startlösung $\vec{x}$, eine zulässige Nachbarlösung $\vec{x}'$, z.B. zufällig, bestimmt wird. Führt die Nachbarlösung $\vec{x}'$ zu einer Verbesserung des Zielfunktionswertes, so wird sie als neue Lösung $\vec{x}$ akzeptiert. Führt die Nachbarlösung hingegen zu einer Verschlechterung des Zielfunktionswertes, so wird sie nur mit einer bestimmten Wahrscheinlichkeit p als neue Lösung akzeptiert. Die Wahrscheinlichkeit p der Akzeptanz ist in diesem Falle abhängig von der Höhe Δ der Verschlechterung und einem sogenannten Temperaturparameter α. Je niedriger der Temperaturparameter, desto geringer wird die Wahrscheinlichkeit, mit der eine schlechtere Nachbarlösung auch bei gleicher Höhe der eintretenden Verschlechterung akzeptiert wird. Im Laufe des Optimierungsprozesses geht der Temperaturparameter α gegen null. Dies führt zu dem Effekt, dass zu Beginn des Verfahrens schlechtere Nachbarlösungen auch bei einer hohen Verschlechterung Δ noch mit einer hohen Wahrscheinlichkeit p akzeptiert werden. Mit Fortschreiten des Optimierungsprozesses reduziert sich jedoch diese Wahrscheinlichkeit und es werden zunehmend nur noch solche Nachbarlösungen akzeptiert, die entweder nur zu einer geringen Verschlechterung Δ führen oder zu einer Verbesserung. Am Ende des Optimierungsprozesses werden nahezu nur noch Nachbarlösungen akzeptiert, die zu einer Verbesserung führen.

Im Beispiel des Bergsteigers würde dem Bergsteiger zu Beginn seiner Wanderung noch erlaubt werden nahezu beliebige Auf- als auch Abstiege vorzunehmen, was es ihm ermöglicht auch andere Berge zu erreichen, als den Berg, in dessen unmittelbarer Nähe er sich befindet. Mit zunehmender Dauer seiner Wanderung wird er jedoch immer mehr dazu angehalten, möglichst nur noch bergauf zu gehen, so dass er am Ende seiner Wanderung einen Berggipfel erreicht.

Von besonderer Bedeutung für die Effektivität von Simulated Annealing ist die Berechnung der Wahrscheinlichkeit p, mit der im Verlauf des Optimierungsverfahrens Verschlechterungen akzeptiert werden. Die Wahrscheinlichkeit p ist Abhängig von der Höhe Δ der Verschlechterung und dem Temperaturparameter α. Die Funktion $p(\Delta, \alpha)$ ist so zu gestalten, dass sehr geringe Verschlechterungen mit einer sehr ho-

hen Wahrscheinlichkeit akzeptiert werden und sehr große Verschlechterungen mit einer sehr geringen Wahrscheinlichkeit. Zudem muss die Akzeptanzwahrscheinlichkeit für Verschlechterung mit abnehmenden Temperaturparameter α geringer werden und am Ende des Verfahrens schließlich nahezu 0 betragen. Eine Funktion, welche die genannten Anforderungen erfüllt ist $p(\Delta,\alpha) = e^{-\frac{\Delta}{\alpha}}$. Es gilt $\lim_{\Delta\to 0} p(\Delta,\alpha) = 1$ und $\lim_{\Delta\to\infty} p(\Delta,\alpha) = 0$.

Zur Illustration des Verlaufs, der durch diese Funktion berechneten Annahmewahrscheinlichkeiten p, ist im folgenden Diagramm die Annahmewahrscheinlichkeit p in Abhängigkeit von der Höhe der Verschlechterung Δ ($\Delta \in [0;1]$) für verschiedene Temperaturparameter α graphisch dargestellt.

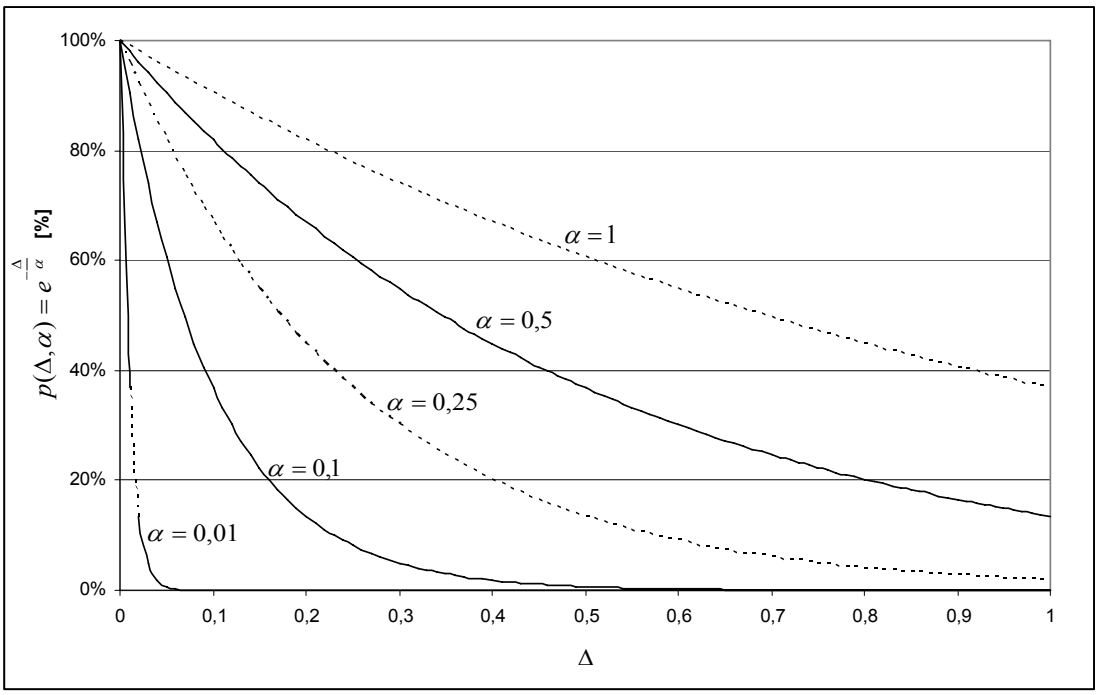

Abbildung 8 - Annahmewahrscheinlichkeit p in Abhängigkeit von Δ und α

Maßgeblich für die Konvergenz und die Laufzeit von Simulated Annealing ist jedoch nicht nur die Berechnung der Annahmewahrscheinlichkeit p, sondern auch die Art und Weise der Reduktion des Temperaturparameters α im Laufe des Optimierungsverfahrens, d.h. die genaue Gestaltung des „Abkühlungsprozesses“, sowie die Definition eines geeigneten Abbruchkriteriums für das Verfahren.

Das in Schritt 4 der Heuristik konkret verwendete Simulated-Annealing-Verfahren ist in zwei Phasen unterteilt, die sich in der Art und Weise der Reduktion des Temperaturparameters α unterscheiden. Das Verfahren terminiert, falls in Phase 2 eine maximale Anzahl von Iterationen durchgeführt wurde, ohne dass eine Verbesserung des Zielfunktionswertes erreicht werden konnte. Das konkret verwendete Verfahren wird im Folgenden näher erläutert.

Das konkret in der Heuristik verwendete Simulated-Annealing-Verfahren

Das in Schritt 4 der Heuristik konkret verwendete Simulated-Annealing-Verfahren ist in zwei Phasen unterteilt. In der ersten Phase wird eine Mindestanzahl von Iterationen durchgeführt, wohingegen die zweite Phase solange Iterationen durchführt bis für eine bestimmte Anzahl aufeinander folgender Iterationen keine Verbesserung mehr erzielt werden kann. In jeder Phase wird eine Simulated-Annealing-Iteration auf die gleiche Art und Weise durchgeführt, jedoch unterscheiden sich die beiden Phasen hinsichtlich der Reduktion des Temperaturparameters α und in der Festlegung der jeweils durchzuführenden Anzahl von Iterationen.

Die Anzahl der durchzuführenden Iterationen in Phase 1 ergibt sich aus der Länge n des Ausführungsplans multipliziert mit einem Faktor u. Während dieser u*n Iterationen wird der Temperaturparameter α von einem Startwert α_1 auf einen Endwert α_2 gleichmäßig reduziert, d.h. in jeder Iteration nimmt α um den Betrag $\frac{\alpha_1 - \alpha_2}{u \cdot n}$ ab. Im konkret angewendeten Verfahren wurde u=4, $\alpha_1 = 1$ und $\alpha_2 = 0{,}1$ gewählt. Dies bedeutet, dass in Phase 1 viermal so viele Iterationen ausgeführt werden, wie der Ausführungsplan Positionen aufweist, wobei der Temperaturparameter α gleichmäßig von dem anfänglichen Wert 1 auf den Wert 0,1 sinkt.

In der sich an Phase 1 anschließenden Phase 2 werden soviele Iterationen durchgeführt, bis eine maximale Anzahl aufeinander folgender Iterationen ohne Verbesserung des Zielfunktionswertes erreicht wird. Die maximale Anzahl aufeinander folgender Iterationen ohne Verbesserung des Zielfunktionswertes ist abhängig von der Länge n des Ausführungsplans und ergibt sich durch Multiplikation der Länge n mit einem Faktor v. In jeder Iteration wird der Temperaturparameter α durch Multiplikation mit einem Faktor c_α $(0 < c_\alpha < 1)$ um einen gewissen Prozentsatz gesenkt. Hierdurch strebt der Temperaturparameter α stetig, aber mit wachsender Anzahl von Iterationen z betragsmäßig langsamer, gegen 0 ($\lim_{z \to \infty} \alpha_2 c_\alpha^z = 0$ da $0 < c_\alpha < 1$). Im konkret angewendeten Verfahren wurde v=2 und $c_\alpha = 0{,}95$ verwendet. Dies bedeutet, dass solange Iterationen durchgeführt werden, bis die Anzahl aufeinander folgender Iterationen ohne Verbesserung des Zielfunktionswertes den Wert der doppelten Länge des Ausführungsplans erreicht hat, wobei in jeder Iteration der Temperaturparameter α um 5% gesenkt wird.

In beiden Phasen wird eine Iteration auf die gleiche Art und Weise durchgeführt. Eine Iteration des Simulated-Annlealing-Verfahrens entspricht einem Optimierungsschritt des in Schritt 3 der Heuristik verwendeten Verfahrens (siehe 3.2.3), unterscheidet sich jedoch im Falle der Nichtverbesserung des Zielfunktionswertes durch eine Nachbarlösung durch die bedingte Akzeptanz der Verschlechterung in Abhängigkeit von der Höhe der Verschlechterung Δ und dem Temperaturparameter α.

Zu Beginn jeder Iteration wird eine Position im aktuellen Ausführungsplan zufällig ausgewählt. Um die Variabilität in den Positionen zu erhöhen, wird hierbei sichergestellt, dass bei aufeinander folgenden Iterationen nicht dieselbe Position ausgewählt wird. Für die gewählte Position i wird nun ein, zum aktuell an dieser Position gesetzten Web Service alternativer, Web Service aus der entsprechenden Kategorie i ausgewählt, um mit diesem eine neue, zulässige Nachbarlösung zu erzeugen. Hierbei werden aus der Kategorie i alle alternativen Web Services in einer zufälligen Reihenfolge

nacheinander an der aktuellen Position des Ausführungsplans gesetzt und es wird jeweils geprüft, ob der entstehende Ausführungsplan zulässig ist. Sobald ein zulässiger Ausführungsplan gefunden wurde, wird das Durchprobieren der Web Services beendet und dieser zulässige Ausführungsplan als zu betrachtende Nachbarlösung festgehalten. Wurden jedoch alle alternativen Web Services der Kategorie i erfolglos durchprobiert, ohne dass ein zulässiger Ausführungsplan erzeugt werden konnte, so wird die Anzahl der aufeinander folgenden Iterationen ohne Verbesserung des Zielfunktionswertes erhöht und die aktuelle Iteration abgebrochen.

Konnte eine zulässige Nachbarlösung $\vec{x}'$ ermittelt werden, so wird der Zielfunktionswert F($\vec{x}'$) der Nachbarlösung mit dem Zielfunktionswert F($\vec{x}$) der aktuellen Lösung $\vec{x}$ verglichen. Besitzt die Nachbarlösung einen höheren Zielfunktionswert (F($\vec{x}'$)>F($\vec{x}$)), so wird die Nachbarlösung als neue aktuelle Lösung $\vec{x}$ übernommen und die Anzahl aufeinander folgender Iterationen ohne Verbesserung des Zielfunktionswertes auf null gesetzt.

Besitzt die Nachbarlösung hingegen einen niedrigeren Zielfunktionswert, so wird die die Anzahl aufeinander folgender Iterationen ohne Verbesserung des Zielfunktionswertes um eins erhöht und die Nachbarlösung nur mit einer bestimmten Wahrscheinlichkeit als neue aktuelle Lösung $\vec{x}$ übernommen. Um die Akzeptanzwahrscheinlichkeit $p(\Delta,\alpha)$ zu berechnen wird zunächst die Höhe Δ der Verschlechterung als prozentualer Unterschied der Zielfunktionswerte mit $\Delta = \frac{F(\vec{x}) - F(\vec{x}')}{\left|F(\vec{x})\right|}$ bestimmt. Anschließend kann die Akzeptanzwahrscheinlichkeit p mit dem aktuellen Temperaturparameter α durch $p(\Delta,\alpha) = e^{-\frac{\Delta}{\alpha}}$ berechnet werden. Um zu entscheiden ob die Akzeptanzwahrscheinlichkeit p zu einer Annahme oder einer Ablehnung der Nachbarlösung $\vec{x}'$ führt, wird eine gleichverteilte Zufallszahl p' aus dem Intervall [0;1] gezogen. Ist $p' \leq p$ so wird die Nachbarlösung $\vec{x}'$ als neue aktuelle Lösung $\vec{x}$ akzeptiert. Ist hingegen $p' > p$, so wird die Nachbarlösung $\vec{x}'$ verworfen und die aktuelle Lösung $\vec{x}$ bleibt unverändert.

Das Simulated-Annealing-Verfahren terminiert nach Erreichen der maximal zulässigen Anzahl von Iterationen ohne Verbesserung des Zielfunktionswertes in Phase 2. Da das Verfahren auch Verschlechterungen des Zielfunktionswertes zulässt, ist es möglich, dass der am Ende erreichte Zielfunktionswert unter den anfänglichen Zielfunktionswert gesunken ist, der bereits am Ende des Schrittes 3 der Heuristik vorlag. In diesem Fall wird die aktuelle Lösung auf die Lösung zurückgesetzt, die am Ende des dritten Schrittes der Heuristik vorlag.

Die folgende Abbildung 9 zeigt eine Formulierung des Verfahrens in Form von Pseudo-Code.

```
x⃗ = AP      // Ausgangslösung ist aktueller Ausführungsplan
x⃗'' = x⃗     // Sicherung Ausgangslösung für abschließenden Vergleich
α = α_1     // Starttemperatur
Δ_α = (α_1 − α_2) / (u · n)      // Reduktion von α um Δ_α in Iteration von Phase 1
iterPhase1 = u*n  // Anzahl Iterationen in Phase 1
z = 0             // Aktuelle Anzahl der durchgeführten Iterationen
maxNoImp = v*n    // Max. Anz. von Iter. ohne Verbesserung in Phase 2
curNoImp = 0      // Aktuelle Anz. von Iterationen ohne Verbesserung

while (curNoImp < maxNoImp) {

      x⃗' = Nachbarlösung(x⃗)  // Analog zu Optimierungsschritt des
                             // Verfahrens in Schritt 3 der Heur.
      if (zulässig(x⃗')) {
            if ( F(x⃗') <= F(x⃗) ) { // Nachbarlösung ist nicht besser
                  if (z >= iterPhase1) curNoImp++
                  Δ = (F(x⃗) − F(x⃗')) / |F(x⃗)|
                  p = e^(−Δ/α)
                  p' = gleichverteilte Zufallszahl aus [0;1]
                  if (p' < p) x⃗ = x⃗'
            } else
                  curNoImp = 0      // Nachbarlösung ist besser
      } else {
            if (z>=iterPhase1)      // Es existierte keine zulässige
                  curNoImp++        // Nachbarlösung an dieser Pos.
      }

      if (z<iterPhase1)
            α = α - Δ_α     // Reduktion von α in Phase 1
      else
            α = α * c_α     // Reduktion von α in Phase 2

}

if ( F(x⃗) < F(x⃗'') )    // Falls akt. Lösung x⃗ schlechter als Aus-
      x⃗ = x⃗''           // gangslösung x⃗'', x⃗ auf x⃗'' zurücksetzen.
```

Abbildung 9 - Pseudo-Code des Simulated-Annealing-Verfahrens

Die While-Schleife des Pseudo-Codes umfasst eine Iteration des Simulated-Annealing-Verfahrens. Sie wird solange wiederholt, bis die maximale Anzahl aufeinander folgender Iterationen ohne Verbesserung des Zielfunktionswertes `maxNoImp` erreicht wird. Die aktuelle Anzahl der aufeinander folgenden Iterationen ohne Verbesserung des Zielfunktionswertes `curNoImp` wird nur in der Phase 2 des Verfahrens erhöht, d.h. nur dann nachdem die Anzahl aller Iterationen z die in Phase 1 durchzuführenden Iterationen `iterPhase1` überschritten hat.

In jeder Iteration wird mittels der Anweisung „$\vec{x}'$ =Nachbarlösung($\vec{x}$)“ eine zulässige Nachbarlösung von $\vec{x}$ ermittelt, die durch den Austausch eines Web Services an einer zufälligen Stelle im Ausführungsplan entsteht. Bei zwei aufeinander folgenden Aufrufen der Funktion Nachbarlösung wird nie dieselbe Position des Ausführungs-

plans verändert. Sollte an der gewählten Position kein anderer Web Service zu einem zulässigen Ausführungsplan führen, so wird ein unzulässiger Ausführungsplan, bzw. eine unzulässige Nachbarlösung $\vec{x}'$ zurückgeliefert. Wird eine zulässige Nachbarlösung $\vec{x}'$ zurückgeliefert, so wird durch den Vergleich der Zielfunktionswerte der aktuellen Lösung F($\vec{x}$) und der Nachbarlösung F($\vec{x}'$) festgestellt, ob eine Steigerung des Zielfunktionswertes erreicht wurde. Eine Verbesserung des Zielfunktionswertes wird in jedem Fall akzeptiert und die Nachbarlösung $\vec{x}'$ wird zur neuen aktuellen Lösung $\vec{x}$. Im Fall einer Verschlechterung erfolgt die Annahme der Nachbarlösung $\vec{x}'$ nur mit der, in Abhängigkeit von der Verschlechterung Δ und dem Temperaturparameter α berechneten, Annahmewahrscheinlichkeit p. Am Ende jeder Iteration erfolgt die Reduktion des Temperaturparameters α, je nach Phase des Verfahrens entweder um den Betrag Δ_α (Phase 1) oder mittels Multiplikation mit dem Faktor c_α (Phase 2).

Nachdem in Phase 2 die maximale Anzahl aufeinander folgender Iterationen ohne Verbesserung des Zielfunktionswertes `maxNoImp` erreicht wurde und damit die While-Schleife verlassen wird, erfolgt abschließend ein Vergleich der ursprünglichen Anfangslösung $\vec{x}''$ mit der aktuellen Lösung $\vec{x}$. Hat sich der Zielfunktionswert der aktuellen Lösung gegenüber dem Zielfunktionswert der ursprünglichen Lösung verschlechtert, so wird die ursprüngliche Lösung wiederhergestellt.

Laufzeitverhalten

Die Phase 1 des geschilderten Simulated-Annealing-Verfahrens durchläuft stets u*n Iterationen und besitzt daher eine von der Länge n des Ausführungsplans linear abhängige Laufzeit, d.h. O(n).

Beim Eintritt in Phase 2 des Simulated-Annealing-Verfahrens ist der Temperaturparameter α bereits auf den Wert α_2 gesenkt worden und wird weiterhin pro Iteration durch die Multiplikation mit dem Faktor c_α ($0 < c_\alpha < 1$) gesenkt. Die Annahmewahrscheinlichkeit $p(\Delta, \alpha)$, mit der Nachbarlösungen mit niedrigerem Zielfunktionswert angenommen werden, konvergiert in dieser Phase gegen 0. Da hierdurch nahezu nur noch Nachbarlösungen mit höherem Zielfunktionswert akzeptiert werden, verhält sich das Verfahren in dieser Phase zunehmend wie das in Schritt 3 der Heuristik angewendete Verfahren (siehe 3.2.3). Mit einem Temperaturparameter $\alpha = 0$ würde die Phase 2 des Simulated-Annealing-Verfahrens dem Verfahren aus Schritt 3 der Heuristik sogar exakt entsprechen. Das Laufzeitverhalten der Phase 2 konvergiert daher zum Laufzeitverhalten des Verfahrens aus Schritt 3 der Heuristik, wobei die Konvergenzgeschwindigkeit vom Parameter c_α abhängig ist. Aufgrund der Konvergenz kann das Laufzeitverhalten von Phase 2 im schlechtestmöglichen Fall in grober Näherung mit dem Laufzeitverhalten $O(mn^2)$ des Verfahrens aus Schritt 3 der Heuristik im schlechtestmöglichen Fall abgeschätzt werden. Wie bereits bei der Erläuterung des Verfahrens des dritten Schrittes der Heuristik erwähnt, wird dieses schlechte Laufzeitverhalten in der Praxis jedoch mit nahezu an Sicherheit grenzender Wahrscheinlichkeit nie erreicht.

Im, für die Laufzeit günstigsten Fall, kann durch keine Iteration eine Verbesserung des Zielfunktionswertes erreicht werden, wodurch Phase 2 nach v*n Iterationen terminiert und damit eine von der Länge n des Ausführungsplans linear abhängige Laufzeit O(n) aufweist.

Genauere Aussagen über das Laufzeitverhalten können aufgrund des nichtdeterministischen Verhaltens des Verfahrens durch den Einfluss des Zufalls nicht getroffen werden. Desweiteren hängt die Laufzeit von der formulierten Problemstellung des Optimierungsproblems und dessen Eingabeparametern ab. Eine eingehendere Analyse würde u.a. die Möglichkeit des Treffens von Verteilungsannahmen über die Eingabeparameter erfordern, die jedoch aufgrund der allgemeinen Anwendbarkeit des Verfahrens nicht getroffen werden können. Eine weitergehende Betrachtung der Laufzeiten würde den Rahmen dieses Technical Reports übersteigen, weswegen hierauf an dieser Stelle verzichtet werden soll. Es sei jedoch auf die Analyse der Heuristik verwiesen (siehe Kapitel 5), die einen Eindruck von der Laufzeit in ausgewählten Testfällen vermitteln kann.

3.3 Replanning

Durch das im vorangegangenen Kapitel 3.2 vorgestellte heuristische Verfahren ist es möglich, zu einem definierten, sequentiellen Geschäftsprozess einen Ausführungsplan zu erstellen, der eine Abfolge von konkreten Web Services festlegt, die bei ihrer sequentiellen Ausführung die im Geschäftsprozess geforderten Funktionalitäten derart erbringen, dass definierte Restriktionen eingehalten und der über Gewichte auf Basis von Gesamtparameterwerten definierte Nutzen optimiert wird. Es kann nicht mit Sicherheit davon ausgegangen werden, dass der durch die Heuristik erzeugte Ausführungsplan ein optimaler Ausführungsplan ist, jedoch wird davon ausgegangen, dass es sich um einen, im Rahmen der Heuristik, bestmöglich optimierten Ausführungsplan handelt.

Treten während der Ausführung des Geschäftsprozesses durch die Abarbeitung des Ausführungsplans keine Änderungen an der Datenbasis[13] auf, welche die Grundlage zur Erstellung des Ausführungsplans bildete, so kann der Ausführungsplan während der ganzen Dauer der Ausführung weiterhin als zulässig und hinreichend optimiert angesehen und Position für Position abgearbeitet werden. Ändert sich jedoch während der Ausführung des Ausführungsplans die Datenbasis, so kann nicht mehr davon ausgegangen werden, dass es sich noch um einen zulässigen und/oder hinreichend optimierten Ausführungsplan handelt.

In der sehr dynamischen Umgebung des Internets und der Nutzung dezentral verfügbarer Web Services unterliegt die Ausführung des Geschäftsprozesses einer Vielzahl von Einflüssen, die teils unvorhersehbar sind. In den einzelnen Kategorien, aus denen Web Services zur Platzierung im Ausführungsplan ausgewählt wurden, können Web Services entfernt werden, es können neue alternative Web Services hinzu kommen oder die Parameter von bereits vorhandenen Web Services können sich ändern. Bei der Ausführung der konkreten Web Services ist es möglich, dass diese momentan nicht verfügbar sind oder dass sich die bei der Optimierung angenommenen Parameterwerte bei der Ausführung als falsch herausstellen, z.B. dass die tatsächliche Antwortzeit eines Web Services die bei der Optimierung veranschlagte Antwortzeit maßgeblich übersteigt. Um in einem solchen Fall die Einhaltung definierter Restriktionen oder eine hinreichende Optimalität des Ausführungsplans sicherstellen zu können, ist es nötig den Ausführungsplan zu überarbeiten und diesen erneut auf Basis der neuen

[13] Der Begriff Datenbasis bezeichnet in diesem Zusammenhang die Menge an Informationen über verfügbare Web Services und ihre nicht-funktionalen Eigenschaften.

Erkenntnisse zu optimieren. Die Neuerstellung eines optimierten Ausführungsplans aufgrund einer geänderten Datenbasis wird im Folgenden als *Replanning* bezeichnet.

Wird nach der Ausführung des Web Services an Position i des Ausführungsplans erkannt, dass sich die Datenbasis geändert hat und ein Replanning nötig wird, so wird der Ausführungsplan in zwei Bereiche unterteilt. Der erste Bereich umfasst alle Positionen $i' \leq i$ des Ausführungsplans, welche bereits ausgeführt wurden. Da ihre Ausführung nicht mehr ungeschehen gemacht werden kann, sind sie fixe, unveränderliche Bestandteile des Ausführungsplans und können im Rahmen des Replannings nicht mehr verändert werden. Der zweite Bereich umfasst alle Positionen $i'' > i$ des Ausführungsplans, welche die noch nicht ausgeführten Web Services enthalten. Nur an diesen Positionen des Ausführungsplans können im Rahmen des Replannings noch Veränderungen vorgenommen werden.

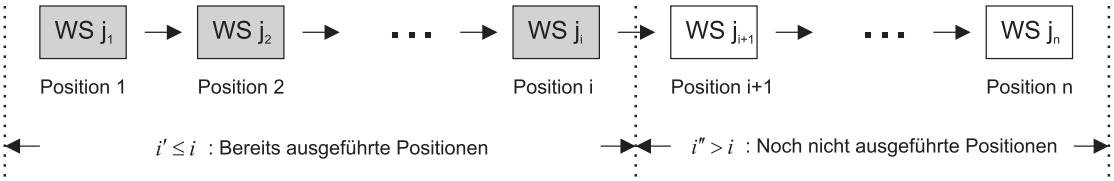

Abbildung 10 - Bereits ausgeführte und noch nicht ausgeführte Positionen des Ausführungsplans

Um einerseits die Problemgröße des Optimierungsproblems bei der Durchführung des Replannings zu reduzieren und andererseits die Unveränderlichkeit der bereits ausgeführten Positionen des Ausführungsplans sicherzustellen, wird im Falle des Replannings ein neues Optimierungsproblem generiert, welches nur den Teil des Geschäftsprozesses umfasst, der noch nicht ausgeführt wurde. Hierzu sind zwei Schritte notwendig. In einem ersten Schritt wird der Geschäftsprozess um jene i Prozessschritte verkürzt, die bereits durch die Ausführung der entsprechenden Web Services im Ausführungsplan ausgeführt wurden. Hierdurch entsteht ein Geschäftsprozess mit n-i Prozessschritten. In einem zweiten Schritt müssen evtl. auf dem Geschäftsprozess definierte Restriktionen angepasst werden, da diese die Gesamtparameterwerte des vollständigen, ungekürzten Geschäftsprozesses beschränken, nicht jedoch die Gesamtparameterwerte eines verkürzten Geschäftsprozesses. Wurde beispielsweise die Gesamtantwortzeit des Geschäftsprozesses durch eine Restriktion auf 1000 ms beschränkt und wurden bereits Web Services mit einer Antwortzeit von 250 ms ausgeführt, so verbleiben noch maximal 750 ms an Antwortzeit für den noch nicht ausgeführten Teil des Geschäftsprozesses. Wird der Geschäftsprozess im Rahmen des Replannings nun um den bereits ausgeführten Teil verkürzt, so ist die maximale Antwortzeit für den noch verbleibenden Teil auf 750 ms zu beschränken, so dass der Geschäftsprozess insgesamt die ursprüngliche Restriktion einer maximalen Antwortzeit von 1000 ms erfüllt.

Die Art und Weise der Anpassung des Schrankenwertes c einer Restriktion erfolgt abhängig von der Art des Gesamtparameters, der durch sie beschränkt werden soll. Es werden Restriktionen für additive Gesamtparameterwerte, multiplikative Gesamtparameterwerte und Minimaloperator-Gesamtparameterwerte unterschieden.

Restriktionen additiver Gesamtparameterwerte

Additive Gesamtparameter x^+ berechnen sich durch Summierung der entsprechenden additiven Parameterwerte, der an der jeweiligen Position im Ausführungsplan ver-

wendeten Web Services (vgl. Kapitel 3.1.2). Zum Zeitpunkt des Replannings lässt sich der Gesamtparameter x^+ in der Form $x^+ = x'^+ + x''^+$ ausdrücken. Hierbei bezeichnet x'^+ den Teil des Gesamtparameterwertes, der durch die bereits erfolgte Ausführung der Web Services an den Positionen $i' \leq i$ tatsächlich bereits entstanden ist. Durch x''^+ wird der Teil des Gesamtparameterwertes bezeichnet der zukünftig noch entsteht, wenn die Web Services der Positionen $i'' > i$ des Ausführungsplans ausgeführt werden. Nach der Verkürzung des Geschäftsprozesses und damit auch des Optimierungsproblems auf den noch nicht ausgeführten Teil, wird nur noch x''^+ als Gesamtparameterwert berechnet.

Um die Einhaltung einer Restriktion der Form $x^+ \leq c^+$ auch im verkürzten Optimierungsproblem gewährleisten zu können, muss diese im verkürzten Optimierungsproblem mittels einer äquivalenten Restriktion der Form $x''^+ \leq c''^+$ ausgedrückt werden, d.h. der Schrankenwert des Gesamtparameters ist im verkürzten Optimierungsproblem von c^+ auf c''^+ anzupassen.

$$
\begin{aligned}
x^+ &\leq c^+ \\
x'^+ + x''^+ &\leq c^+ \\
x''^+ &\leq c^+ - x'^+ \\
\underbrace{\sum_{i''=i+1}^{n} \sum_{j=1}^{m_{i''}} p^+_{i'',j} x_{i'',j}}_{x''^+} &\leq \underbrace{c^+ - \underbrace{\sum_{i'=1}^{i} \sum_{j=1}^{m_{i'}} p^+_{i',j} x_{i',j}}_{x'^+}}_{c''^+}
\end{aligned}
$$

Der neue Schrankenwert c''^+ für eine äquivalente Restriktion im verkürzten Optimierungsproblem ergibt sich daher als

$$c''^+ = c^+ - \sum_{i'=1}^{i} \sum_{j=1}^{m_{i'}} p^+_{i',j} x_{i',j}$$

d.h. durch Subtraktion der entsprechenden additiven Parameterwerte aller bereits ausgeführten Web Services an den Positionen $i' \leq i$ des Ausführungsplans vom ursprünglichen Schrankenwert c^+.

Für den Fall einer Restriktion der Form $x^+ \geq c^+$ erfolgt die Anpassung des Schrankenwertes analog.

Restriktionen multiplikativer Gesamtparameterwerte

Die Korrektur der Schrankenwerte multiplikativer Gesamtparameter $x^\bullet$ erfolgt analog der Überlungen zur Korrektur des Schrankenwertes von Restriktionen additiver Gesamtparameter. Zum Zeitpunkt des Replannings lässt sich der Gesamtparameter $x^\bullet$ in der Form $x^\bullet = x'^\bullet \cdot x''^\bullet$ ausdrücken, wobei $x'^\bullet$ den Anteil des Gesamtparameterwertes bezeichnet, der durch den bereits ausgeführten Teil des Geschäftsprozesses bereits tatsächlich entstanden ist und $x''^\bullet$ den Anteil des Gesamtparameterwertes der zukünftig noch durch die Ausführung der noch ausstehenden Web Services im Ausführungsplan entstehen wird. Die Einhaltung einer Restriktion der Form $x^\bullet \leq c^\bullet$ kann im ver-

kürzten Optimierungsproblem dadurch gewährleistet werden, dass sie im verkürzten Optimierungsproblem durch eine äquivalente Restriktion der Form $x''^{\bullet} \leq c''^{\bullet}$ ausgedrückt wird. Der neue Schrankenwert $c''^{\bullet}$ für die äquivalente Restriktion im verkürzten Optimierungsproblem ergibt sich wie folgt:

$$x^{\bullet} \leq c^{\bullet}$$

$$x'^{\bullet} \cdot x''^{\bullet} \leq c^{\bullet}$$

$$x''^{\bullet} \leq \frac{c^{\bullet}}{x'^{\bullet}}$$

$$\prod_{i''=i+1}^{n} \sum_{j=1}^{m_{i''}} p^{\bullet}_{i'',j} x_{i'',j} \leq \frac{c^{\bullet}}{\underbrace{\prod_{i'=1}^{i} \sum_{j=1}^{m_{i'}} p^{\bullet}_{i',j} x_{i',j}}_{c''^{\bullet}}}$$

$$c''^{\bullet} = \frac{c^{\bullet}}{\prod_{i'=1}^{i} \sum_{j=1}^{m_{i'}} p^{\bullet}_{i',j} x_{i',j}}$$

Der neue Schrankenwert $c''^{\bullet}$ ergibt sich mittels Division von $c^{\bullet}$ durch das Produkt der entsprechenden multiplikativen Parameterwerte aller bereits ausgeführten Web Services an den Positionen $i' \leq i$ des Ausführungsplans.

Um die Restriktion des multiplikativen Gesamtparameters exakt in das lineare Modell des verkürzten Optimierungsproblems abbilden zu können, erfolgt nach der Korrektur des Schrankenwertes eine Linearisierung der Restriktion durch Logarithmierung, wie sie bereits in Kapitel 3.1.2 erläutert wurde. Die konkrete Abbildung der Restriktion erfolgt daher in der folgenden Form.

$$\sum_{i''=i+1}^{n} \sum_{j=1}^{m_{i''}} \ln\left(p^{\bullet}_{i'',j}\right) x_{i'',j} \leq \ln\left(c''^{\bullet}\right)$$

Für den Fall einer Restriktion der Form $x^{\bullet} \geq c^{\bullet}$ erfolgt die Anpassung des Schrankenwertes analog zum Vorgehen im Falle einer Restriktion der Form $x^{\bullet} \leq c^{\bullet}$.

Restriktionen von Minimaloperator-Gesamtparameterwerte

Minimaloperator-Gesamtparameter ergeben sich als Minimum aus den jeweiligen Minimialoperator-Parametern der im Ausführungsplan enthaltenen Web Services.

$$x^{\min} = \underset{i=1}{\overset{n}{Min}} \left(\sum_{j=1}^{m_i} p^{\min}_{i,j} x_{i,j} \right)$$

Sie können durch Restriktionen der Form $x^{\min} \geq c^{\min}$ nach unten beschränkt werden, womit erzwungen wird, dass ein Minimaloperator-Gesamtparameter stets größer oder gleich dem Schrankenwert $c^{\min}$ ist, d.h. kein im Ausführungsplan verwendeter Web Service einen Minimaloperator-Parameterwert besitzt der kleiner als $c^{\min}$ ist. Dieser Schrankenwert behält auch bei der Verkürzung des Geschäftsprozesses im Rahmen des Replannings in exakt dieser Weise auch für alle noch zukünftig auszuführenden

Web Services weiterhin seine Gültigkeit, da diese Restriktion durch jeden Web Service des Ausführungsplans einzeln, unabhängig von allen anderen Web Services im Ausführungsplan erfüllt werden muss. Eine Anpassung des Schrankenwertes ist daher bei einer Verkürzung des Geschäftsprozesses nicht erforderlich.

Erneute Anwendung der Heuristik

Nach der Verkürzung des Geschäftsprozesses und der Anpassung evtl. vorhandener Restriktionen kann das Optimierungsproblem erneut durch Anwendung des heuristischen Verfahrens gelöst werden. Hierbei erfolgt die Optimierung des entstehenden Ausführungsplans auf der Basis der neuen, aktuellen Datenbasis. Das Ergebnis ist ein auf der Grundlage der neuen Datenbasis bezüglich seines Nutzens optimierter Ausführungsplan für alle noch auszuführenden Prozessschritte des Geschäftsprozesses, der alle Restriktionen erfüllt.

Zu beachten bleibt an dieser Stelle, dass es möglich ist, dass beim erneuten Durchlauf des heuristischen Verfahrens erkannt wird, dass das Optimierungsproblem auf der Grundlage der neuen Datenbasis nicht mehr lösbar ist. Ein Beispiel hierfür wäre z.B. dass zur Ausführung des nächsten Prozessschrittes nur ein einziger Web Service zur Verfügung steht, der jedoch nicht mehr erreichbar ist oder aus der entsprechenden Kategorie entfernt wurde. Ein weiteres Beispiel ergibt sich in dem Fall, in welchem die Ausführung des letzten Web Services, entgegen der ursprünglich bei der Optimierung angenommenen Antwortzeit, bereits eine solche Antwortzeit in Anspruch genommen hat, dass die Restriktion der Gesamtantwortzeit bereits überschritten wurde. In diesen beispielhaft genannten Fällen kann es keinen zulässigen Ausführungsplan mehr geben. Wie eine BPE, die das heuristische Verfahren zur optimierten Ausführung von Geschäftsprozessen verwendet, mit solchen Situation zur Laufzeit umgeht, bleibt im konkreten Fall der jeweiligen BPE überlassen und ist nicht Gegenstand des eigentlichen heuristischen Verfahrens. Während es im Falle der Nichterreichbarkeit eines Web Services, zu dem es keine Alternative gibt, tatsächlich nicht mehr möglich ist die Ausführung des Geschäftsprozesses sinnvoll fortzusetzen, existiert im Falle des Überschreitens der Restriktionen jedoch noch die Möglichkeit den zuletzt als optimal bekannten Ausführungsplan in seiner Ausführung fortzusetzen oder das Optimierungsproblem derart umzuformulieren, dass durch eine erneute Anwendung des heuristischen Verfahrens ein Ausführungsplan entsteht, der die Restriktion zwar nicht mehr erfüllt, aber zumindest so wenig wie möglich verletzt.

Ebenso liegt es im Verantwortungsbereich der BPE zu entscheiden in welchen Fällen überhaupt ein Replanning durchgeführt wird. Überwacht die BPE beispielsweise die Dienstgüteeigenschaften, die bei der Ausführung einzelner Web Services gemessen werden können, so kann die BPE aufgrund dieses Monitorings ein Replanning einleiten, sobald die Abweichung zwischen gemessenen Dienstgüteeigenschaften und der aufgrund der Datenbasis erwarteten Dienstgüteeigenschaften einen gewissen Grenzwert überschreitet. Verfügt die BPE über kein solches Monitoring, so kann ein Replanning höchstens noch durch eine Signalisierung der Veränderung der Datenbasis ausgelöst werden, d.h. in dem Fall in dem neue Web Services in die Datenbasis aufgenommen werden, aus ihr entfernt werden, oder ihre Parameterwerte geändert werden. Ob in einem solchen Falle eine Replanning lohnenswert erscheint, hängt davon ab mit welcher Frequenz Änderungen an der Datenbasis zu erwarten sind, wie stark die Optimalität eines Ausführungsplans durch derartige Änderungen beeinflusst werden kann und letztlich auch welche Dauer für ein Replanning im Verhältnis zur angesetzten Gesamtausführungsdauer des Geschäftsprozesses zu erwarten ist.

4 Implementierung

Nachdem in Kapitel 3 das Optimierungsproblem, dessen mathematisches Modell und das heuristische Lösungsverfahren erläutert wurde, wird im vorliegenden Kapitel die prototypische Implementierung des heuristischen Lösungsverfahrens vorgestellt. Zunächst wird ein Überblick über die entwickelte prototypische Software gegeben. Anschließend wird das Datenformat zur Definition eines Optimierungsproblems beschrieben, sowie die Art und Weise wie dieses in ein lineares Optimierungsproblem überführt werden kann, welches durch einen Solver[14] für lineare Optimierungsprobleme gelöst werden kann. Daran schließt sich eine Betrachtung der Implementierung der einzelnen Schritte der Heuristik an, die auf der Lösung des relaxierten linearen Optimierungsproblems durch den Solver aufbauen. Es folgt die Erläuterung der im Prototyp enthaltenen Ausführungsumgebung, welche die Ausführung von Geschäftsprozessen auf Basis der durch die Heuristik erzeugten Ausführungspläne simuliert. Abschließend erfolgt die Betrachtung des ebenfalls im Prototypen enthaltenen Datengenerators, mit welchem nach definierten Vorgaben Testdaten für Optimierungsprobleme erzeugt werden können.

4.1 Überblick

Um die Lösungsgüte und die benötigte Rechenzeit des entwickelten heuristischen Verfahrens zur Erzeugung optimierter Ausführungspläne von sequentiellen Geschäftsprozessen näher untersuchen zu können, wurde die in Kapitel 3.2 vorgestellte Heuristik prototypisch implementiert. Die erstellte prototypische Software trägt den Namen *WorkflowOptimizer* und ist speziell darauf ausgerichtet die Heuristik zu evaluieren und nicht etwa sie produktiv in eine funktionstüchtige BPE zu integrieren. In diesem Kapitel soll ein kurzer Überblick über diese Software vermittelt werden.

4.1.1 Implementierungsbasis

WorkflowOptimizer basiert auf der „Java 2 Platform Standard Edition“ (J2SE)[15] in der Version 1.5.0[16]. Für das Logging, bzw. zur Ausgabe von Debug-Informationen während der Ausführung von WorkflowOptimizer wird Log4J[17] in der Version 1.2.9 eingesetzt. Zur Lösung des, aus dem Optimierungsproblem abgeleiteten, linearen Optimierungsproblems wird der Open-Source-Solver lp_solve[18] in der Version 5.1.1.3 verwendet. Er kann unter der LGPL (GNU lesser general public license)[19] frei genutzt werden und ist für eine Vielzahl von Plattformen verfügbar. Der Solver ist in der Lage lineare Optimierungsprobleme, aber auch (gemischt-)ganzzahlige lineare Optimierungsprobleme zu lösen. Die Ansteuerung der Solver-Bibliothek erfolgt über den,

[14] Ein Solver, engl. für (Problem-)Löser, ist eine spezialisierte Software, die in der Lage ist spezifische Problemstellungen zu verarbeiten und eine entsprechende Lösung zu ermitteln.
[15] Siehe auch: http://java.sun.com/j2se/1.5.0/ und z.B. [67].
[16] Diese Version wird auch als „J2SE 5.0” bezeichnet.
[17] Siehe auch: http://logging.apache.org/log4j/ und z.B. [31].
[18] Siehe auch: http://groups.yahoo.com/group/lp_solve/
[19] Die LGPL ist eine von der „Free Software Foundation“ (FSF) entwickelte Lizenz, welche bezüglich einer Software die Freiheit garantiert, sie für einen beliebigen Zweck nutzen zu dürfen, sie zu vervielfältigen, weiterzugeben, an die eigenen Bedürfnissen anzupassen und geänderte Versionen weiterzugeben. Im Gegensatz zur GPL (GNU General Public Licence) erlaubt die LGPL zudem, dass die Software als Bestandteil einer Software verwendet werden darf, die selbst nicht unter der LGPL bereitgestellt wird. Die LGPL ist unter folgender URL abrufbar: http://www.gnu.org/copyleft/lesser.html

ebenfalls unter der LGPL freigegebenen, Java-Wrapper lpsolve4j in der Version 5.1.1.3 von Jürgen Ebert, der alle API-Aufrufe[20] der Solver-Bibliothek in einer Javaklasse durch JNI[21]-Aufrufe kapselt.

Zur Installation von WorkflowOptimizer wurde ein Installationsarchiv erstellt, welches alle nötigen Bestandteile zum Betrieb der Software, mit Ausnahme von J2SE 1.5.0, enthält. Der Quelltext von WorkflowOptimizer sowie eine ausführliche Installationsanleitung in englischer Sprache sind ebenso in diesem Installationsarchiv enthalten. Auf die nötigen Schritte zur Installation, Konfiguration und den Start der Software soll an dieser Stelle nicht weiter eingegangen werden. Stattdessen sei auf die im Installationsarchiv enthaltene Installationsanleitung in der Datei „`ReadMe.txt`“ verwiesen.

WorkflowOptimizer wurde erfolgreich unter den Betriebssystemen „Microsoft Windows XP“ und „SuSE Linux 9.3“ getestet.

4.1.2 Graphische Benutzeroberfläche und Funktionsumfang

Um die Bedienung von WorkflowOptimizer zu erleichtern, wurde WorkflowOptimizer mit einer graphischen Benutzeroberfläche (GUI[22]) ausgestattet. Abbildung 11 zeigt die graphische Benutzeroberfläche, wie sie sich dem Benutzer direkt nach dem Start des Programms präsentiert.

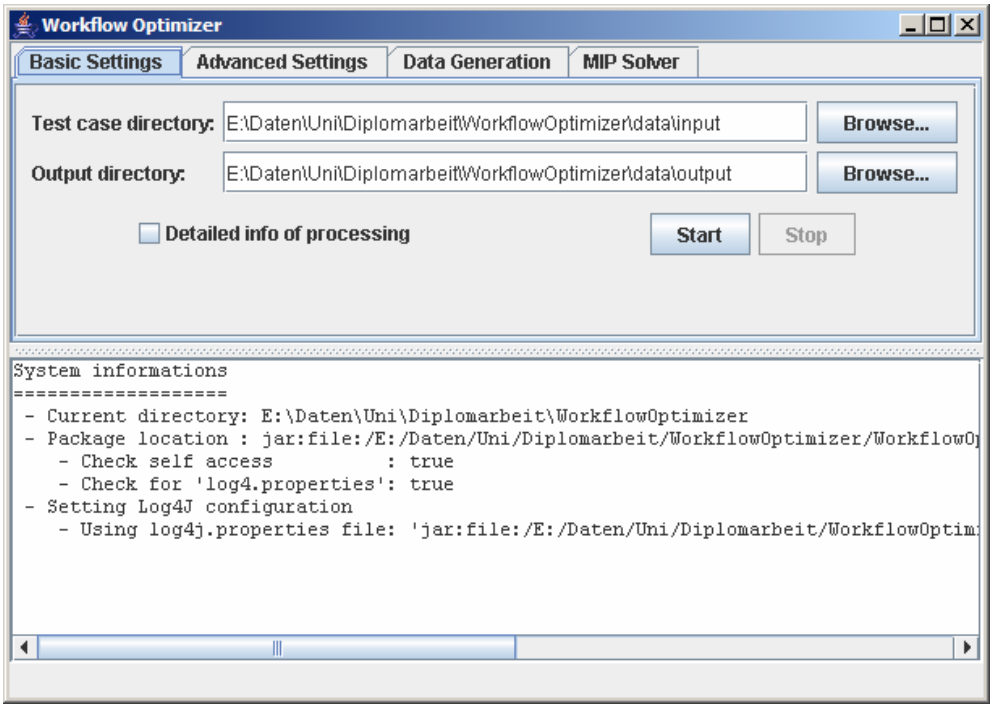

Abbildung 11 - GUI von WorkflowOptimizer, Reiter „Basic Settings“

[20] Durch ein „Application Programming Interface“ (API) definiert eine Software Schnittstellen zum Zugriff auf ihre Funktionalität durch andere Programme.
[21] Das JNI (Java Native Interface) ist eine Java-Schnittstelle, die es ermöglicht aus dem betriebssystemunabhängigen Java heraus betriebssystemspezifische Funktionen aufrufen und hierdurch z.B. betriebssystemspezifische Bibliotheken nutzen zu können. Siehe hierzu auch: http://java.sun.com/j2se/1.5.0/docs/guide/jni/
[22] GUI (Graphical User Interface) ist eine übliche, aus dem englischen Sprachraum stammende Abkürzung zur Bezeichnung einer graphischen Benutzeroberfläche.

Der obere, grau hinterlegte Bereich des Fensters dient zum Setzen von Programmparametern und dem Ausführen von Aktionen. Dieser Bereich sei im Folgenden *Aktionsbereich* genannt. Der untere, weiß hinterlegte Textbereich des Fensters sei im Folgenden als *Statusbereich* bezeichnet und wird von WorkflowOptimizer dazu genutzt um Statusinformationen anzuzeigen, z.B. Informationen über den aktuellen Fortschritt beim Ausführen einer Aktion oder auftretende Zwischen- und Ergebnisse. Die sich am unteren Ende des Fensters befindliche graue Leiste dient als Fortschrittsanzeige während der Ausführung von Aktionen und sei im Folgenden als *Statusleiste* bezeichnet.

Nach dem Start von WorkflowOptimizer sind alle Programmparameter mit Standardwerten belegt. Im Aktionsbereich können durch entsprechende Kontrollelemente Programmparameter verändert und Aktionen aufgerufen werden. Aus Gründen der Übersicht wurden die Kontrollelemente im Aktionsbereich in verschiedene Gruppen unterteilt. Zwischen den einzelnen Gruppen kann über die Reiter des Aktionsbereichs gewechselt werden. Welche Parameter in der jeweiligen Gruppe geändert und welche Aktionen dort aufgerufen werden können, wird im Folgenden erläutert.

Reiter „Basic Settings“

Ein Optimierungsproblem wird zur Verarbeitung durch WorkflowOptimizer auf drei Dateien verteilt definiert. Die drei benötigten Dateien tragen die Namen „`Workflow_<Nummer>.in`“, „`ParameterDefinitions_<Nummer>.in`“ und „`Categories_<Nummer>.in`“, wobei „`<Nummer>`“ ein Platzhalter für eine Zahl ist. Zusammengehörige Dateien, die gemeinsam ein Optimierungsproblem definieren, tragen stets dieselbe Nummer und werden im Folgenden als Testfall, bzw. *Testcase* bezeichnet. In welcher Form ein Optimierungsproblem dort beschrieben werden muss, damit es durch den WorkflowOptimizer verarbeitet werden kann, wird im Kapitel 4.2 näher erläutert. Damit WorkflowOptimizer Testcases einlesen und die durch sie definierten Optimierungsprobleme lösen kann, ist das Eingabeverzeichnis zu definieren, in welchem sich alle zu verarbeiteten Testcases befinden. Das Eingabeverzeichnis kann entweder im Eingabefeld „Test case directory“ eingetragen oder über einen Auswahldialog gewählt werden, der über die Schaltfläche „Browse...“ rechts neben dem Eingabefeld aufgerufen werden kann.

Während der Ausführung von verschiedenen Aktionen von WorkflowOptimizer ist es nötig zu Protokollzwecken Dateien anzulegen, z.B. um Zwischenergebnisse oder gemessene Ausführungszeiten der Heuristik festzuhalten. Das Ausgabeverzeichnis für Dateien kann über das Eingabefeld „Output directory“ angegeben oder über einen Auswahldialog gewählt werden, der über die Schaltfläche „Browse...“ aufgerufen werden kann, die sich rechts neben dem Eingabefeld befindet.

Nach dem Start von WorkflowOptimizer wird das Eingabe- und das Ausgabeverzeichnis auf einen Standardwert gesetzt. Hierzu wird zunächst das Verzeichnis ermittelt, aus welchem heraus WorkflowOptimizer ausgeführt wird. Das Eingabeverzeichnis wird anschließend auf das Unterverzeichnis „`data/input`“ und das Ausgabeverzeichnis auf das Unterverzeichnis „`data/output`“ dieses Verzeichnisses gesetzt.

Um den Benutzer während der Ausführung von Aktionen über den aktuellen Zustand und eventuelle Zwischenergebnisse zu informieren, werden Protokollinformationen im Statusbereich des WorkflowOptimizer-Fensters und in eine Protokolldatei ausge-

geben. Über das Kontrollkästchen „Detailed info of processing“ kann hierbei die Ausgabe besonders ausführlicher Informationen aktiviert werden.

Befinden sich Testcases im Eingabeverzeichnis, so können diese durch WorkflowOptimizer in aufsteigender Reihenfolge ihrer Testcasenummer verarbeitet und zu jedem Testcase eine Lösung durch Anwendung der Heuristik ermittelt werden. Dieser Lösungsvorgang, bzw. Optimierungsprozess kann über die Schaltfläche „Start“ gestartet und mittels der Schaltfläche „Stop“ vorzeitig abgebrochen werden. Sobald der Optimierungsvorgang gestartet wurde, wird die Schaltfläche „Start“ deaktiviert und die Schaltfläche „Stop“ aktiviert. Während des Optimierungsvorganges werden Informationen über den aktuellen Stand und Zwischenergebnisse der Optimierung im Statusbereich und in der Statusleiste angezeigt. Nachdem das Optimierungsverfahren beendet wurde, erscheint ein Dialog, welcher dem Benutzer die Beendigung des Optimierungsvorgangs anzeigt. Der Dialog kann durch Wahl der „Ok“-Schaltfläche geschlossen werden. Hierdurch wird die Schaltfläche „Stop“ wieder deaktiviert und die Schaltfläche „Start“ aktiviert. WorkflowOptimizer ist nun wieder im Ausgangszustand und bereit weitere Testcases zu verarbeiten.

Reiter „Advanced Settings“

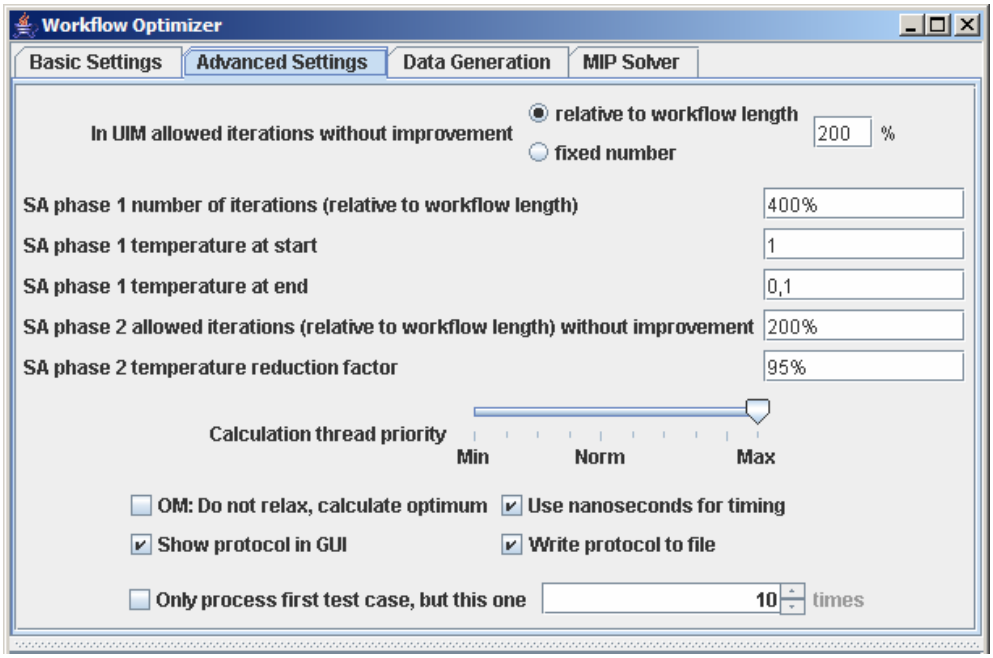

Abbildung 12 - Ausschnitt der WorkflowOptimizer-GUI, Reiter „Advanced Settings“

Da es sich bei WorkflowOptimizer um die prototypische Implementierung eines heuristischen Optimierungsverfahrens zum Zwecke seiner Evaluation handelt, stehen im Bereich „Advanced Settings“ eine Vielzahl von Änderungsmöglichkeiten für Parameter zur Verfügung, die das Verhalten der Heuristik und des Lösungsprozesses beeinflussen.

Unter dem Punkt „In UIM allowed iterations without improvement“ kann die Anzahl der maximal durchzuführenden Iterationen ohne Verbesserung des Zielfunktionswertes in Schritt drei der Heuristik festgelegt werden (vgl. Kapitel 3.2.3). Die Bezeich-

nung UIM (Unsophisticated Improvement Method[23]) wird hier als Synonym für den Schritt drei der Heuristik verwendet. Die Anzahl der maximalen Iterationen ohne Steigerung des Zielfunktionswertes kann entweder relativ zur Länge des Geschäftsprozesses („relative to workflow length“) oder absolut („fixed number“) angegeben werden. Welche Art der Parameterspezifikation gewählt werden soll, kann durch die Wahl des entsprechenden Optionsfeldes festgelegt werden. Je nach gewählter Art der Spezifikation ist der gewünschte Wert anschließend in das Eingabefeld einzutragen, welches sich rechts von den Optionsfeldern befindet.

Die folgenden fünf Eingabefelder, deren Bezeichnung allesamt mit „SA“ (abkürzend für Simulated Annealing) beginnt, dienen der Definition von Parametern bezüglich des Simulated-Annealing-Verfahrens, welches im vierten Schritt der Heuristik (vgl. Kapitel 3.2.4) zur Anwendung gelangt. Unter „SA phase 1 number of iterations (relative to workflow length)“ kann der Faktor u zur Berechnung der Anzahl der durchzuführenden Iterationen in Phase eins des Simulated-Annealing-Verfahrens festgelegt werden. Über „SA phase 1 temperature at start“ und „SA phase 1 temperature at end“ kann der Temperaturparameterwert α_1 beim Eintritt in Phase eins und der Temperaturparameterwert α_2 beim Beenden von Phase eins des Simulated-Annealing-Verfahrens definiert werden. Unter „SA phase 2 allowed iterations (relative to workflow length) without improvement“ kann der Faktor v zur Berechnung der maximal erlaubten Anzahl von Iterationen ohne Verbesserung des Zielfunktionswertes in Phase zwei des Simulated-Annealing-Verfahrens festgelegt werden. Die Abnahme des Temperaturparameters α in Phase zwei geschieht durch Multiplikation mit dem Faktor c_α in jeder Iteration. Der Faktor c_α kann unter „SA phase 2 temperature reduction factor“ definiert werden.

An diesen Bereich von Parametern schließt sich ein Bereich von Programmparametern an, die das Verhalten von WorkflowOptimizer während des Optimierungsprozesses beeinflussen. Mittels des Schiebreglers „Calculation thread priority“ kann die Rechenzeit beeinflusst werden, die dem Optimierungsprozess im Falle seiner Ausführung im Vergleich zu anderen Prozessen und Threads[24] zur Verfügung gestellt wird. Der Schiebregler lässt sich im Bereich von „Min“ bis „Max“ in 10 Stufen verstellen, die den Java-Thread-Prioritäten eins bis zehn entsprechen. Die Standardeinstellung ist hierbei „Max“, wodurch dem Thread, in welchem der Optimierungsprozess ausgeführt wird, die maximal mögliche Rechenzeit zur Verfügung gestellt wird. Hierbei ist zu beachten, dass diese Einstellung, vor allem auf Einzelprozessorsystemen ohne Unterstützung der Hyper-Threading-Technologie[25], zu einem Stillstand des Betriebssystems für die Dauern der Ausführung des Optimierungsprozesses führen kann, da der Thread während dieser Zeit nahezu die vollständige Rechenleistung verbraucht. In diesem Fall ist ggf. vor dem Start des Optimierungsprozesses die Thread-Priorität

[23] Englisch für „naives Verbesserungsverfahren“. Das Verfahren aus Schritt drei der Heuristik wird deshalb als naiv bezeichnet, da es im Vergleich zum Simulated-Annealing-Verfahren des vierten Schrittes der Heuristik myopisch ist und nicht in der Lage ist eine einmal erreichte Umgebung um ein lokales Maximum wieder verlassen und hierdurch ein evtl. existierendes globales Maximum erreichen zu können.

[24] Ein Thread (engl. Faden, Strang) bezeichnet einen Ausführungspfad eines Prozesses, der nebenläufig zu anderen Ausführungspfaden (Threads) desselben Prozesses ausgeführt wird.

[25] Intel bezeichnet seine Implementierung von „Simultaneous Multithreading“ (SMT) in den Prozessoren Pentium 4 und Xeon als Hyper-Threading-Technologie (HTT). Durch die Duplizierung und Aufteilung bestimmter Ressourcen werden hierbei virtuelle Prozessoren (Siblings) generiert, die mehrere Threads weitestgehend parallel ausführen können.

über den Schiebregler soweit zu reduzieren, dass auch während des Optimierungsprozesses weiterhin ein sinnvolles Arbeiten mit dem umgebenden Betriebssystem möglich bleibt. Es bleibt jedoch zu beachten, dass sich die benötigte Ausführungsdauer zur Durchführung des Optimierungsprozesses mit abnehmender Thread-Priorität erhöht. Auf eine genauere Betrachtung von Threads soll an dieser Stelle jedoch verzichtet werden und es sei auf entsprechende Literatur, wie z.B. [35] verwiesen.

Durch Aktivierung des Kontrollkästchens „OM: Do not relax, calculate optimum" kann die Relaxierung des Optimierungsproblems in Schritt eins der Heuristik deaktiviert werden. Diese Option ist nur für sehr spezielle Tests vorgesehen und führt dazu, dass bereits in Schritt eins der Heuristik der optimale Ausführungsplan ermittelt und als Ausgangsbasis für alle folgenden Schritte verwendet wird. Die Abkürzung „OM" in der Beschriftung des Kontrollkästchens steht für „Opening Method" (engl. Eröffnungsmethode) und fasst als Bezeichnung die ersten beiden Schritte der Heuristik zusammen, da diese, in Anlehnung an den Begriff des Eröffnungsverfahrens von Optimierungsproblemen, eine erste zulässige Lösung ermitteln.

Während der Ausführung des Optimierungsprozesses werden die für die einzelnen Schritte der Heuristik benötigten Rechenzeiten gemessen. Über das Kontrollkästchen „Use nanoseconds for timing" kann festgelegten werden ob die Messung der Rechenzeiten auf Basis von Nanosekunden (Kontrollkästchen aktiviert, Standardwert) oder auf Basis von Millisekunden (Kontrollkästchen deaktiviert) vorgenommen wird. Unabhängig von der Wahl der Basis zur Messung der Rechenzeiten werden diese von WorkflowOptimizer stets in der Einheit Millisekunden ausgegeben. Da die Messung auf Basis von Nanosekunden jedoch wesentlich präziser ist, sollte diese Einstellung nicht verändert werden.

Um den Fortschritt und die Zwischenergebnisse während der Ausführung des Optimierungsprozesses zu dokumentieren, werden entsprechende Informationen im Statusbereich des WorkflowOptimizer-Fensters ausgegeben. Durch das Kontrollkästchen „Show protocol in GUI" kann diese Ausgabe aktiviert werden (Kontrollkästchen aktiviert, Standardwert) oder deaktiviert werden (Kontrollkästchen deaktiviert). Die Ausgabe des Protokolls im Statusbereich erfolgt durch einen gesonderten Thread niedriger Priorität, jedoch verbraucht die Bildschirmausgabe des Protokolls durch die Ausführung teurer Ausgabeoperationen unnötige Rechenzeit, die dem Thread des Optimierungsprozesses fehlt und dadurch die Ausführung des Optimierungsprozesses verzögert. Vor der Messung von Rechenzeiten des Optimierungsprozesses sollte diese Option daher deaktiviert werden.

Über das Kontrollkästchen „Write protocol to file" kann analog zur vorherigen Option die Ausgabe des Protokolls des Optimierungsprozesses in eine Datei im Ausgabeverzeichnis aktiviert (Standardwert) oder deaktiviert werden. Die Ausgabe des Protokolls in eine Datei nimmt weniger Rechenzeit in Anspruch als die Ausgabe im Statusbereich des WorkflowOptimizer-Fensters. Im Falle der Messung von Ausführungszeiten sollte daher die Protokollausgabe in eine Datei der Protokollausgabe im Statusbereich des WorkflowOptimizer-Fensters vorgezogen werden, um die benötigte Rechenzeit des Optimierungsprozesses nicht unnötig zu verlängern. Eine noch geringere Verfälschung der benötigten Rechenzeit lässt sich jedoch durch die vollständige Deaktivierung der Protokollausgabe erreichen. Anzumerken ist, dass diese Option die Ausgabe der Performancedatei in das Ausgabeverzeichnis nicht beeinflusst. Diese Datei enthält die, für die einzelnen Schritte der Heuristik benötigten Rechenzeiten im CSV-

Format[26] und wird während der Ausführung des Optimierungsprozesses durch einen eigenen Thread erstellt. Dieser beeinflusst die vom Optimierungsprozess benötigte Rechenzeit aufgrund der wenigen, zur Erstellung der Datei benötigten Ausgabeoperationen nicht merklich.

Durch Aktivierung des Kontrollkästchens „Only process first test case“ ist es möglich die Verarbeitung der Testcases im Eingabeverzeichnis auf den ersten Testcase zu beschränken, dessen Verarbeitung jedoch mehrmals hintereinander ausführen, bzw. wiederholen zu lassen. Die Anzahl der Wiederholungen kann im Eingabefeld rechts von der Beschriftung des Kontrollkästchens eingetragen werden. Diese Option ist nützlich zur Ermittlung der durchschnittlich zur Lösung eines Testcases benötigten Rechenzeit. Durch die Option kann ein Testcase automatisch sehr oft hintereinander ausgeführt werden und es können anschließend, aus den hierbei in der Performancedatei im Ausgabeverzeichnis festgehaltenen Ausführungszeiten der einzelnen Schritte der Heuristik, Durchschnittswerte berechnet werden.

Reiter „Data Generation“

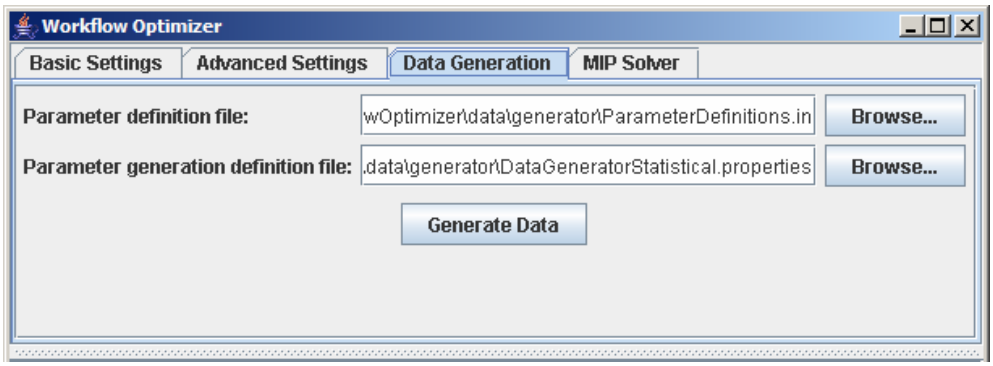

Abbildung 13 - Ausschnitt der WorkflowOptimizer-GUI, Reiter „Data Generation“

WorkflowOptimizer ist nicht nur in der Lage Optimierungsprobleme in Form von Testcases einzulesen und zu lösen, sondern diese auch zu Testzwecken selbständig zu generieren und mit Testdaten zu füllen. Um Testcases erzeugen zu können, ist es nötig die Parameter von Web Services zu definieren, für die im Laufe der Datengenerierung konkrete Parameterwerte, d.h. Testdaten, erzeugt werden sollen. Die Definition der Parameter erfolgt über eine Parameterdefinitionsdatei, deren Aufbau im Kapitel 4.2 näher beschrieben wird. Zusätzlich zur Definition der Parameter ist es nötig die Länge des zu erzeugenden Geschäftsprozesses, die Anzahl alternativer Web Services für jeden Prozessschritt, die Art und Weise der Erzeugung der Parameterwerte für jeden Parameter der Web Services, sowie die Anzahl der zu erzeugenden Testcases zu definieren. Diese Definitionen erfolgen in einer Generatordefinitionsdatei deren Aufbau im Kapitel 4.7 näher erläutert wird.

Bevor die Generierung der Testcases durchgeführt werden kann, ist es nötig die zu verwendenden Definitionsdateien zu spezifizieren. Die Parameterdefinitionsdatei kann im Eingabefeld „Parameter definition file“ angegeben werden. Über die Schaltfläche „Browse...“ rechts des Eingabefeldes kann ein Dialog zur Auswahl einer ent-

[26] „Comma Separated Value“ (CSV) bezeichnet ein Dateiformat für Textdateien, in welchem Informationen separiert durch ein Trennzeichen in Tabellen- oder Listenform strukturiert abgelegt werden können. Textdateien dieses Formates können durch eine Vielzahl von Anwendungen (z.B. Tabellenkalkulationen) verarbeitet werden.

sprechenden Datei aufgerufen werden. Analog hierzu geschieht die Angabe der Generatordefinitionsdatei entweder direkt über das Eingabefeld „Parameter generation definition file“ oder indirekt durch die Dialogauswahl, welche über die Schaltfläche „Browse...“ rechts des Eingabefeldes aufgerufen werden kann.

Nach dem Start von WorkflowOptimizer ist das Eingabefeld zur Angabe der Parameterdefinitionsdatei mit dem Dateinamen „`ParameterDefinitions.in`“ im Unterverzeichnis „`data\generator`“ des WorkflowOptimizer-Verzeichnisses vorbelegt. Das Eingabefeld zur Angabe der Generatordefinitionsdatei ist mit dem Dateinamen „`DataGeneratorStatistical.properties`“ im Unterverzeichnis „`data\generator`“ des WorkflowOptimizer-Verzeichnisses vorbelegt.

Die Generierung der Testcases, gemäß der in den angegebenen Dateien enthaltenen Definitionen, kann durch Wahl der Schaltfläche „Generate Data“ gestartet werden. Die erzeugten Testcases werden in das Ausgabeverzeichnis geschrieben, welches unter dem Reiter „Basic Settings“ angegeben wurde. Bei der Ausgabe der Testcases werden alle nötigen Dateien zur Definition der einzelnen Testcases in das Ausgabeverzeichnis geschrieben, sowie zusätzlich Dateien die Informationen über die erzeugten Testcases enthalten. Wurde die Protokollausgabe im Statusbereich des WorkflowOptimizer-Fensters nicht unter dem Reiter „Advanced Settings“ deaktiviert, so erfolgt dort die Ausgabe aller Dateinamen, der bei der Datengenerierung erzeugten Dateien. Der Abschluss des Generierungsprozesses wird dem Benutzer durch eine entsprechende Dialogbox angezeigt.

Bei den, zusätzlich zu den nötigen Dateien zur Definition der Testcases erzeugten Dateien, handelt es sich um die Dateien „`Categories_all.csv`“, „`Info_overall.txt`“, sowie eine Datei „`Info_<Nummer>.txt`“ für jeden erstellten Testcase, wobei „`<Nummer>`“ ein Platzhalter für die Nummer des jeweiligen Testcases ist. Die Datei „`Categories_all.csv`“ enthält alle erzeugten Parameterwerte aller Web Services in allen Kategorien aller Testcases im CSV-Format. Die Dateien „`Info_<Nummer>.txt`“ enthalten statistische Werte über die erzeugten Parameterwerte des jeweiligen Testcases, sowohl gruppiert nach einzelnen Kategorien als auch zusammenfassend für den ganzen Testcase. Eine zusammenfassende Statistik über die Parameterwerte aller Testcases kann der Datei „`Info_overall.txt`“ entnommen werden. Hierbei liegen die statistischen Werte sowohl gruppiert nach den einzelnen Testcases als auch zusammenfassend für alle Testcases vor.

Reiter „MIP[27] Solver“

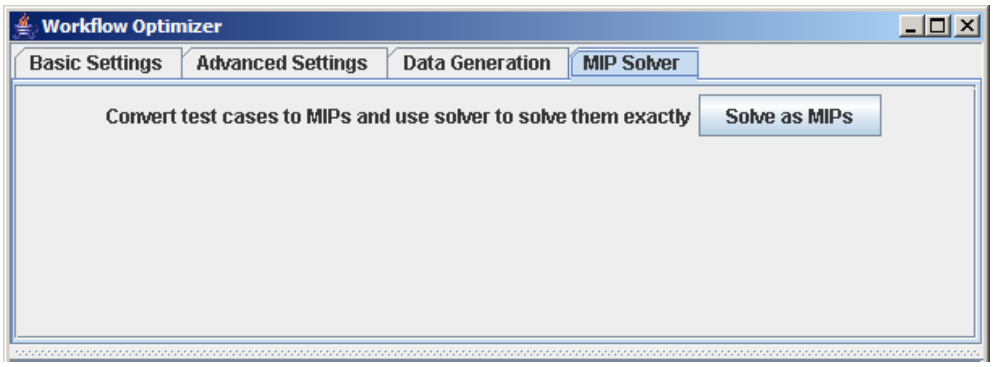

Abbildung 14 - Ausschnitt der WorkflowOptimizer-GUI, Reiter „MIP Solver“

Um die Lösungsgüte der Heuristik beurteilen zu können, ist es nötig die Lösung der Heuristik mit der tatsächlich optimalen Lösung des Optimierungsproblems vergleichen zu können. Zur Ermittlung der tatsächlichen, optimalen Lösung können durch Wahl der Schaltfläche „Solve as MIPs“ alle Testcases im Eingabeverzeichnis exakt gelöst werden. Hierzu werden die Testcases nacheinander eingelesen, als gemischt-ganzzahliges Optimierungsproblem an den Solver übergeben und durch diesen gelöst. Dieses Vorgehen entspricht dem des Schritts eins der Heuristik, jedoch wird im Gegensatz zu diesem das Optimierungsproblem nicht relaxiert, bevor es durch den Solver gelöst wird. Aufgrund der bestehenden Ganzzahligkeitsbedingung für alle Entscheidungsvariablen $x_{i,j}$ kann das Optimierungsproblem durch den Solver nicht mehr durch alleiniges Anwenden des Simplex-Algorithmus gelöst werden, sondern erfordert darüber hinaus die Anwendung des Branch&Bound-Verfahrens. Hierdurch steigt die Rechenzeit zur Lösungsfindung, je nach Schwere des Optimierungsproblems, stark an. Vergleiche der benötigten Rechenzeit und der Lösungsgüte zwischen dem exakten Lösungsverfahren und der Heuristik können Kapitel 5 entnommen werden.

Analog zur Durchführung des Optimierungsprozesses der Heuristik wird bei der Durchführung des exakten Lösungsprozesses eine Performancedatei und eine Protokolldatei im Ausgabeverzeichnis angelegt. Die Ausgabe des Protokolls im Statusbereich des WorkflowOptimizer-Fensters und in die Protokolldatei lässt sich mit den entsprechenden Optionen unter dem Reiter „Advanced Settings“ unterdrücken. Dort kann auch die Option „Only process first test case“ aktiviert werden, um den Lösungsprozess auf den ersten Testcase im Eingabeverzeichnis zu beschränken und die Anzahl der durchzuführenden Wiederholungen des Lösungsprozess festzulegen. Im Gegensatz zur Heuristik kann sich bei den einzelnen Wiederholungen zwar die ermittelte Lösung nicht ändern, jedoch können auf diese Art und Weise mehrere Messwerte für die benötigte Rechenzeit ermittelt werden um hieraus einen Durchschnittswert zu errechnen. Da die Lösung des unrelaxierten Optimierungsproblems vollständig durch die betriebssystemnah programmierte Solver-Bibliothek durchgeführt wird und der Lösungsprozess dort deterministisch ist, d.h. nicht dem Zufall unterliegt, fallen die Schwankungen bezüglich der benötigten Rechenzeit sehr gering aus.

[27] „Mixed Integer Programming“ (MIP) bezeichnet das Aufstellen und/oder Lösen von gemischt-ganzzahligen linearen Optimierungsproblemen (GLOPen; vgl. Kapitel 2.2.4).

4.1.3 Klassenübersicht

Bevor in den folgenden Kapiteln die Implementierung anhand eines typischen Durchlaufs des Lösungsprozesses genauer erläutert wird, soll hier nun zunächst eine Übersicht über die wichtigsten Pakete und Klassen gegeben werden.

Paket de.tud.kom.dynws.optimizer

Dieses Paket enthält benötigte Klassen für die eigentliche Anwendung WorkflowOptimizer mit graphischer Benutzeroberfläche. Neben der ausführbaren Klasse zum Starten der Anwendung zählen hierzu z.B. auch Klassen für die Nachrichtenverarbeitung und die Ausführung von Lösungsprozessen in eigenen Threads, Klassen zum Kapseln der Definitionsdateien aus denen sich ein Testcase zusammensetzt und des linearen Optimierungsproblems das aus einem Testcase erzeugt wird, Klassen zum Messen von Rechenzeiten und dem Festhalten der benötigten Anzahl von Iterationen in den Schritten der Heuristik und des resultierenden Zielfunktionswertes, sowie Klassen zur Berechnung statistischer Werte aus der Datenbasis eines Testcases.

Klasse	Beschreibung
`WorkflowOptimizerGUI`	WorkflowOptimizer-Anwendung mit graphischer Benutzeroberfläche.
`WorkflowOptimizerGUIMessageDispatcher`	Dispatcher zur Verarbeitung von Statusnachrichten in eigenem Thread.
`WorkflowOptimizerGUICSVDispatcher`	Dispatcher zur Ausgabe von Messwerten in CSV-Dateien in eigenem Thread.
`TestCase`	Optimierungsproblem in Form eines Testcases.
`TestSetProcessor`	Einlesen aller Testcases und Weitergabe an `SequentialWorkflowEngine` zur Lösung durch Heuristik.
`TestSetProcessorLinear`	Einlesen aller Testcases und Ermittlung der exakten Lösung durch den Solver.
`MIPModel`	Gemischt-ganzzahliges lineares Optimierungsproblem eines Testcases.
`PerformanceCase`	Messwerte bezüglich Rechenzeit, Iterationen und Zielfunktionswerten eines vollständigen Testcases.
`PerformanceStep`	Messwerte bezüglich Rechenzeit, Iterationen und Zielfunktionswerten eines Laufs der Heuristik in einem Abarbeitungsschritt der `SequentialWorkflowEngine`.
`Statistics`	Berechnung statistischer Informationen zu den Daten eines Testcases.
`Timer`	Durchführung von Messungen der Rechenzeit.

Tabelle 5 - Wichtige Klassen des Pakets de.tud.kom.dynws.optimizer

Paket de.tud.kom.dynws.optimizer.data

Dieses Paket enthält Klassen zur Kapselung der Daten zur Definition eines Optimierungsproblems, wie z.B. Geschäftsprozess, Prozessschritte, Kategorien, Web Services, Eigenschaften von Web Services und Parameterwerte.

Klasse	Beschreibung
`Workflow`	Geschäftsprozess, bestehend aus Prozessschritten (`TaskItems`). Zusätzlich Definition des Optimierungsproblems durch Angabe von Restriktionen und Gewichten für die Zielfunktion (als `ParameterSet`-Objekte).
`TaskItem`	Prozessschritt im Geschäftsprozess. Die geforderte Funktionalität wird durch die zugehörige Kategorie (`Category`) spezifiziert.
`Categories`	Container für alle Kategorien (`Category`-Objekte).
`Category`	Kategorie als Container für alle Web Services (`WebService`-Objekte), welche dieselbe Funktionalität erbringen können.
`WebService`	Web Service mit zugehöriger Menge an Eigenschaften (`ParameterSet`).
`ParameterDefinitions`	Container für alle Parameterdefinitionen (`ParameterDefinition`-Objekte).
`ParameterDefinition`	Parameterdefinition (Name, Bezeichnung der Einheit, Art des Parameters [additiv | multiplikativ | Minimaloperator-Parameter], Verbesserung durch Zu- oder Abnahme des Parameterwerts) zur Definition einer Eigenschaftsart eines Web Services.
`Parameter`	Eine Eigenschaft eines Web Services, ausgedrückt als Parameterwert gemäß einer Parameterdefinition (`ParameterDefinition`).
`ParameterSet`	Menge von Eigenschaften, bestehend aus einem Parameterwert (`Parameter`) zu jeder im Container `ParameterDefinitions` enthaltenen Eigenschaftsart (`ParameterDefinition`).
`Data`	Container für Daten eines Testcases: Geschäftsprozess und Definition des Optimierungsproblems (`Workflow`-Objekt), zugehöriges lineares Optimierungsproblem (`MIPModel`-Objekt), Kategorien (`Categories`-Objekt) und Parameterdefinitionen (`ParameterDefinitions` -Objekt).

Tabelle 6 - Wichtige Klassen des Pakets de.tud.kom.dynws.optimizer.data

Paket de.tud.kom.dynws.optimizer.sequentialheuristics

Dieses Paket enthält Klassen zur Abbildung eines Ausführungsplans eines Geschäftsprozesses, dessen Erstellung und Optimierung durch die Heuristik, sowie der Simulation einer BPE, die zu einem Geschäftsprozess mittels der Heuristik einen optimierten Ausführungsplan erstellt und diesen schrittweise abarbeitet, wobei nach jedem Prozessschritt ein Replanning durchgeführt wird.

Klasse	Beschreibung
`ExecutionPlan`	Ausführungsplan zu einem Geschäftsprozess (`Workflow`), bestehend aus Ausführungsschritten (`ExecutionPlanItems`), der die in den Prozessschritten auszuführenden konkreten Web Services festlegt und damit eine Lösung des Optimierungsproblems kapselt.
`ExecutionPlanItem`	Ein Ausführungsschritt im Ausführungsplan (`ExecutionPlan`). Definiert einen konkreten, zur Erbringung der in einem Prozessschritt benötigten Funktionalität auszuführenden Web Service (`WebService`).
`OpeningMethod`	Eröffnungsverfahren der Heuristik zur Ermittlung einer ersten zulässigen Lösung (`ExecutionPlan`). Fasst Schritt eins und zwei der Heuristik zusammen.
`UnsophisticatedImprovement`	Naives Verbesserungsverfahren zur weitergehenden Optimierung einer Lösung (`ExecutionPlan`) in Richtung eines (evtl. lokalen) Optimums. Entspricht Schritt drei der Heuristik.
`SimulatedAnnealing`	Anwendung des auf Simulated Annealing basierenden Optimierungsverfahrens auf eine zulässige Lösung (`ExecutionPlan`) um ein evtl. erreichtes lokales Optimum weiter verbessern zu können. Entspricht Schritt vier der Heuristik.
`SequentialWorkflowEngine`	Simuliert eine BPE, die zu einem Geschäftsprozess (`Workflow`) mittels der Heuristik einen optimierten Ausführungsplan (`ExecutionPlan`) erstellt und diesen schrittweise abarbeitet, wobei nach jedem Prozessschritt ein Replanning durchgeführt wird.
`SolveSingle`	Findet die optimale Besetzung eines Ausführungsplans (`ExecutionPlan`) der nur einen Ausführungsschritt (`ExecutionPlanItem`) umfasst, durch Ausprobieren aller möglichen Web Services (`WebServices`) der entsprechenden Kategorie (`Category`).

Tabelle 7 - Wichtige Klassen des Pakets de.tud.kom.dynws.optimizer.sequentialheuristics

Paket de.tud.kom.dynws.optimizer.datagenerator

Dieses Paket enthält Klassen zur Realisierung eines Datengenerators, der aus den Angaben einer Parameterdefinitionsdatei und einer Generatordefinitionsdatei entsprechende Testcases erzeugen kann.

Klasse	Beschreibung
`DataGeneratorStatistical`	Datengenerator zum Erzeugen von Testcases gemäß den Angaben in der Parameterdefinitionsdatei und der Generatordefinitionsdatei.
`ParameterGenerationDefinition`	Kapselung der Definition eines einzelnen Parameters und dessen Generation, sowie der Erzeugung entsprechender Parameterwerte.

Tabelle 8 - Wichtige Klassen des Pakets de.tud.kom.dynws.optimizer.datagenerator

In den folgenden Kapiteln wird ein typischer Ablauf bei der Verarbeitung eines Testcases durch den WorkflowOptimizer geschildert. Hierbei werden die wichtigsten Klassen und die durch sie erbrachte Funktionalität näher erläutert.

4.2 Definition und Verarbeitung von Testcases

4.2.1 Definition eines Testcases

Ein Testcase wird durch die drei Dateien „`Workflow_<Nummer>.in`“, „`ParameterDefinitions_<Nummer>.in`“ und „`Categories_<Nummer>.in`“ definiert, wobei „`<Nummer>`“ ein Platzhalter für eine Zahl darstellt. Dateien, die gemeinsam einen Testcase definieren, tragen dieselbe Nummer. Bei den einzelnen Dateien handelt es sich um Textdateien, die jeweils einen spezifischen Teil der Definition eines Optimierungsproblems abbilden. Ihr Aufbau wird im Folgenden erläutert.

ParameterDefinitions_<Nummer>.in

Die Parameterdefinitionsdatei „`ParameterDefinitions_<Nummer>.in`“ definiert die einzelnen Parameter, durch die ein Web Service im Optimierungsproblem beschrieben werden soll, bzw. sie definiert die Eigenschaften eines Web Services, die im Rahmen des zu lösenden Optimierungsproblems zu beachten sind.

```
Response time;ms;<;+
Availability;%;>;*
Throughput;calls/min;>;<
```

Abbildung 15 - Beispiel einer Parameterdefinitionsdatei

Abbildung 15 zeigt das Beispiel einer Parameterdefinitionsdatei, die drei Parameter definiert. Jede Zeile entspricht der Definition eines Parameters. Jeder Parameter wird durch vier Angaben definiert, die durch ein Semikolon getrennt werden:

1. Name des Parameters.
2. Name der Einheit, in welcher der Parameter ausgedrückt wird.
3. Ein Zeichen, welches definiert, ob größere („>“) oder kleinere („<“) Parameterwerte einen höheren Nutzen stiften.
4. Ein Zeichen, welches definiert wie einzelne Parameterwerte des Parameters zu einem Gesamtparameterwert eines Ausführungsplans aggregiert werden müs-

sen. Additive Parameter werden durch „+“, multiplikative Parameter durch „-“ und Minimaloperator-Parameter durch „<“ definiert.

Durch die erste Zeile des Beispiels wird der Parameter „Response time“ definiert, der in der Einheit „ms“ ausgedrückt wird. Durch das Zeichen „<“ wird ausgedrückt, dass kleinere Parameterwerte des Parameters einen höheren Nutzen erzeugen. Das Zeichen „+“ definiert diesen Parameter schließlich als einen additiven Parameter.

In einer Parameterdefinitionsdatei muss mindestens ein Parameter definiert werden. Eine Beschränkung der Anzahl k der definierbaren Parameter nach oben besteht nicht.

Workflow_<Nummer>.in

Die Geschäftsprozessdefinitionsdatei „`Workflow_<Nummer>.in`“ definiert einen sequentiellen Geschäftsprozess, die auf den k Gesamtparameterwerten eines Ausführungsplans anzuwendenden Restriktionen in Form der Schrankenwerte c_k, sowie die in der Zielfunktion zu verwendenden Gewichte w_k zur Umrechnung der Gesamtparameterwerte in Nutzeneinheiten (vgl. Kapitel 3.1.2).

```
w;0.03125;5000;0.1
c;3000;-;-
1/1->2/2->3/3
```

Abbildung 16 - Beispiel einer Geschäftsprozessdefinitionsdatei

Abbildung 16 zeigt das Beispiel einer Geschäftsprozessdefinitionsdatei. Eine Geschäftsprozessdefinitionsdatei umfasst stets drei Zeilen.

Die erste Zeile beginnt mit dem Zeichen „`w`“ und definiert die Gewichte w_k in der Reihenfolge, in der die zugehörigen Parameter in der Parameterdefinitionsdatei definiert wurden. Die Gewichte werden hierbei durch Semikola separiert. Zu beachten ist, dass Gewichte, unabhängig davon, ob höhere oder niedrigere Gesamtparameterwerte einen höheren Nutzen erzeugen, stets positiv angegeben werden.

Die zweite Zeile beginnt mit dem Zeichen „`c`“ und definiert die Schrankenwerte c_k, welche die jeweiligen Gesamtparameterwerte beschränken. Ob ein Gesamtparameterwert durch den Schrankenwert c_k nach oben oder nach unten beschränkt werden soll, muss nicht angegeben werden, da sich dies aus der Parameterdefinition ergibt. So definiert ein Schrankenwert für einen Gesamtparameterwert, der durch Aggregation eines Parameters entsteht, für den höhere Parameterwerte einen geringeren Nutzen erzeugen als niedrigere, implizit eine Restriktion in Form einer oberen Schranke. Soll ein Gesamtparameterwert nicht durch eine Restriktion beschränkt werden, so ist statt eines Schrankenwertes das Zeichen „`-`“ anzugeben.

Durch die Zeile drei wird der eigentliche Geschäftsprozess als Abfolge von Prozessschritten definiert, deren jeweilige Funktionalität mittels einer Kategorie spezifiziert wird. Ein Prozessschritt wird durch eine Prozessschrittnummer und eine Kategorienummer definiert, die durch das Zeichen „`/`“ separiert werden. Die Prozessschrittnummer dient lediglich der eindeutigen Bezeichnung eines Prozessschrittes, wohingegen die Kategorienummer die Nummer der Kategorie spezifiziert, durch die alle Web Services gruppiert werden, welche die benötigte Funktionalität des Prozessschrittes

erbringen können. Wird die Funktionalität einer Kategorie mehrmals in einem Geschäftsprozess benötigt, so können mehrere Prozessschritte mit derselben Kategorienummer definiert werden. Folgt ein Prozessschritt auf einen anderen, d.h. ist ein Prozessschritt der Nachfolger eines Prozessschrittes, so wird er durch das Zeichen „->“ von diesem getrennt und direkt hinter diesem angegeben. Da auf diese Art und Weise zu jedem Prozessschritt nur jeweils ein Nachfolger definiert werden kann, können nur sequentielle Geschäftsprozesse definiert werden.

Das Beispiel in Abbildung 16 zeigt die Definition eines Geschäftsprozesses, der aus drei Prozessschritten besteht. Der Geschäftsprozess beginnt mit Prozessschritt eins, in welchem die Funktionalität eines Web Services aus Kategorie eins benötigt wird. Nach Prozessschritt eins wird Prozessschritt zwei ausgeführt, der die Funktionalität eines Web Services aus Kategorie zwei benötigt. Schließlich endet der Geschäftsprozess mit der Ausführung des Prozessschrittes drei, der die Funktionalität eines Web Services aus Kategorie drei benötigt. Durch die zweite Zeile wird eine Restriktion auf den ersten Gesamtparameterwert definiert. Zulässige Ausführungspläne zu diesem Geschäftsprozess sind hierdurch nur solche, die im ersten Gesamtparameterwert die Schranke von 3000 Einheiten nicht verletzen. In Zeile eins werden die Gewichte zur Umrechnung der Gesamtparameterwerte in Nutzeneinheiten wie folgt festgelegt: Die Nutzeneinheiten die durch den ersten Gesamtparameterwert erzeugt werden, entstehen durch Multiplikation des ersten Gesamtparameterwertes mit dem Gewicht 0,03125, die Nutzeneinheiten die durch den zweiten Gesamtparameterwert erzeugt werden, entstehen durch Multiplikation des zweiten Gesamtparameterwertes mit dem Gewicht 5000 und die Nutzeneinheiten die durch den dritten Gesamtparameterwert erzeugt werden, entstehen durch Multiplikation des dritten Gesamtparameterwertes mit dem Gewicht 0,1.

Categories_<Nummer>.in

Nachdem durch die Parameterdefinitionsdatei die Parameter definiert wurden, die zur Beschreibung der nicht-funktionalen Eigenschaften der Web Services verwendet werden und in der Geschäftsprozessdefinitionsdatei der Geschäftsprozess in Form einer Abfolge der benötigten Funktionalitäten durch Kategorien von Web Services, wird nun durch die Kategoriedefinitionsdatei „`Categories_<Nummer>.in`“ definiert, welche Web Services, mit welchen Eigenschaften in den einzelnen Kategorien zur Bildung eines entsprechenden Ausführungsplans verfügbar sind.

Abbildung 17 zeigt das Beispiel einer Kategoriedefinitionsdatei, in welcher drei Kategorien mit jeweils drei Web Services definiert werden, die durch jeweils drei Parameterwerte beschrieben werden.

```
1
c;-;-;-
1;1000;0.995;2500
2;1500;0.99;5000
3;500;0.999;1000
-
2
c;-;-;-
1;250;0.999;1250
2;500;0.996;1500
3;750;0.991;2000
-
3
c;-;-;-
1;3000;0.9995;4500
2;2000;0.9999;3000
3;4000;0.999;6000
-
```

Abbildung 17 - Beispiel einer Kategoriedefinitionsdatei

Die Definition einer Kategorie beginnt stets mit einer Zeile, welche die Nummer der Kategorie definiert. Anschließend folgt eine Zeile die mit dem Zeichen „c“ beginnt und Schrankenwerte für die Parameterwerte der Web Services in dieser Kategorie definiert. Diese Zeile ist analog der Zeile zur Definition der Schrankenwerte für Gesamtparameterwerte in der Geschäftsprozessdefinitionsdatei (siehe oben) aufgebaut, jedoch werden durch diese Schrankenwerte lediglich die zulässigen Web Services innerhalb einer Kategorie definiert. Nach der Zeile zur Definition der Schrankenwerte folgt nun je eine Zeile zur Definition eines Web Services in dieser Kategorie. Die Zeile beginnt jeweils mit der Nummer des Web Services, die ihn innerhalb dieser Kategorie eindeutig identifizieren muss. Danach folgen, durch Semikola getrennt, die k Parameterwerte p_k des Web Services in der Reihenfolge der Definition der zugehörigen Parameter in der Parameterdefinitionsdatei. Die Definition einer Kategorie wird durch eine Zeile abgeschlossen, die nur das Zeichen „-“ enthält. Die Kategoriedefinitionsdatei entsteht durch eine sequentielle Abfolge von Kategoriedefinitionen dieser Art.

Das Beispiel zeigt die Definition der drei Kategorien „1“, „2“ und „3“, die jeweils die Web Services „1“, „2“ und „3“ enthalten. Die Web Services werden hierbei jeweils durch drei Parameterwerte beschrieben. Die Parameterwerte des Web Services „1“ in Kategorie „1“ sind beispielsweise 1000 Einheiten für Parameter eins, 0,995 Einheiten für Parameter zwei und 2500 Einheiten für Parameter drei. In keiner der Kategorien erfolgt eine Einschränkung der zulässigen Web Services durch die Angabe von Schrankenwerten auf den Parameterwerten.

4.2.2 Einlesen von Testcases

In der soeben geschilderten Art und Weise kann eine beliebige Anzahl von Optimierungsproblemen in Form von Testcases definiert werden. Beim Start des Optimierungsprozesses werden alle Testcases, die sich im Eingabeverzeichnis befinden durch ein Objekt der Klasse `TestSetProcessor` eingelesen.

Wird der Optimierungsprozess durch Wahl der Schaltfläche „Start“ unter dem Reiter „Basic Settings“ im Aktionsbereich gestartet, so wird eine neue Instanz der Klasse

`TestSetProcessor` erzeugt, die über ihre Methode `run` das Interface `Runnable` implementiert. Anschließend wird eine neue Instanz der Klasse `Thread` erzeugt, wobei das `TestSetProcessor`-Objekt als im Thread auszuführende `Runnable`-Instanz übergeben wird. Nachdem die Thread-Priorität des `Thread`-Objekts gemäß den Einstellungen unter dem Reiter „Advanced Settings“ gesetzt wurde, wird der Thread gestartet. Die Verarbeitung der Testcases geschieht nun in einem neuen Thread, unabhängig des Threads zur Ausführung der graphischen Benutzeroberfläche der WorkflowOptimizer-Anwendung.

Die Testcases werden über die Methode `loadTestCases` des `TestSetProcessor`-Objekts eingelesen. Für jeden Testcase, der im Eingabeverzeichnis gefunden wird, wird ein `TestCase`-Objekt erzeugt um über dessen Funktion `load` alle drei Definitionsdateien des Testcases einzulesen. Liefert diese Funktion als Rückgabewert `false`, so ist das Einlesen mindestens einer, zu diesem Testcase gehörenden Definitionsdatei fehlgeschlagen und der Testcase wird nicht in die Liste der abzuarbeitenden Testcases übernommen.

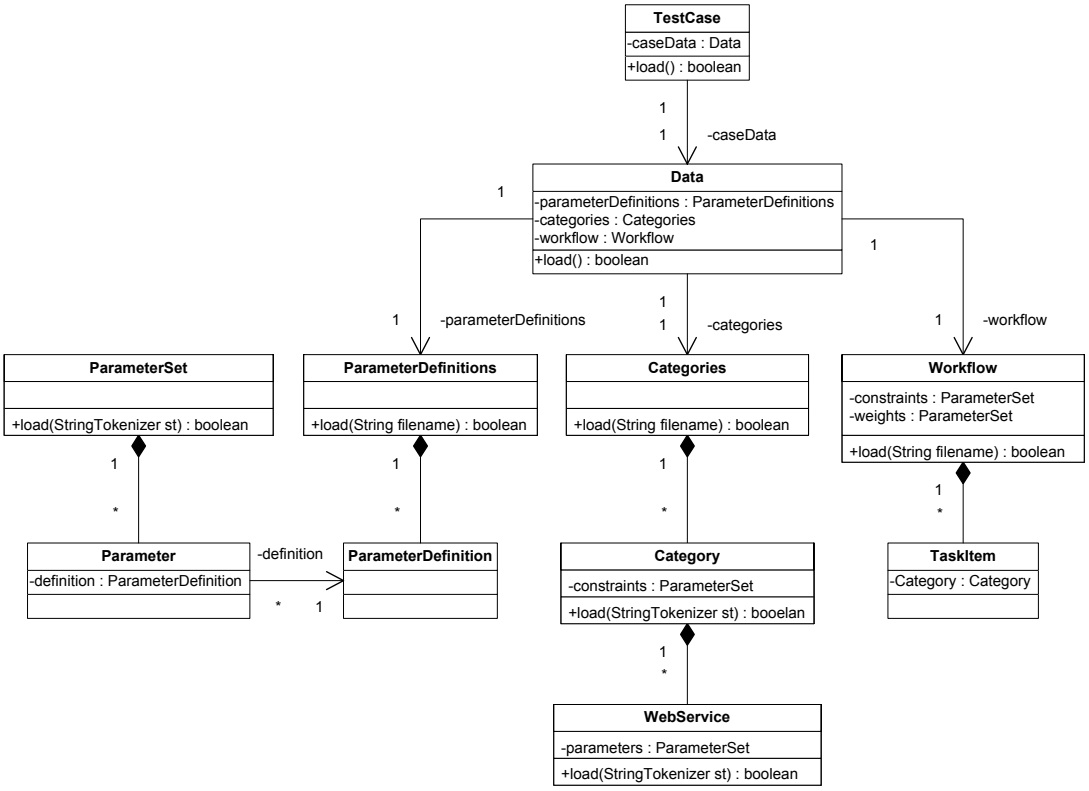

Abbildung 18 - Wichtige Elemente der beim Einlesen eines Testcases aufgebauten Datenstruktur

Die Funktion `load` des `TestCase`-Objekts liest die einzelnen Definitionsdateien eines Testcases nicht selbst ein, sondern erzeugt zum Einlesen und Speichern aller Definitionsdaten ein `Data`-Objekt, und delegiert das Einlesen der Testcases an dessen `load`-Funktion. Die `load`-Funktion des `Data`-Objekts erzeugt zum Einlesen und zur Ablage der Definitionsdaten für jede Definitionsdatei ein eigenes Objekt. Parameterdefinitionen werden in einem `ParameterDefinitions`-Objekt abgelegt, Kategoriedefinitionen in einem `Categories`-Objekt und die Geschäftsprozessdefinition in einem `Workflow`-Objekt. Das Einlesen der Daten aus den jeweiligen Dateien wird durch die `load`-Funktion der jeweiligen Ablageobjekte selbständig ausgeführt. Hier-

bei wird ein `StreamTokenizer`-Objekt verwendet um den Zeicheneingabestrom aus einer Definitionsdatei in einzelne Teilzeichenketten (Tokens) zu zerlegen und einzulesen. Die Interpretation der Tokens übernimmt hierbei entweder das jeweilige Objekt selbst, oder es verwendet weitere, spezialisiertere Klassen zum Einlesen und Ablegen der Informationen aus den Definitionsdateien (vgl. Abbildung 18).

Das `ParameterDefinitions`-Objekt erzeugt beim Einlesen für jede Parameterdefinition ein `ParameterDefinition`-Objekt, die es in einem privaten `Vector`-Attribut speichert und durch öffentliche Funktionen gekapselt zugreifbar macht.

Analog legt das `Categories`-Objekt für jede Kategorie der Kategoriedefinitionsdatei ein `Category`-Objekt an. Hierbei wird das Einlesen der Kategoriedaten weiter an die `load`-Funktion des `Category`-Objekts delegiert. Die Restriktionen einer Kategorie werden durch ein `ParameterSet`-Objekt repräsentiert und werden über dessen `load`-Funktion eingelesen. Ein `ParameterSet`-Objekt erzeugt hierbei für jeden eingelesenen Parameter ein `Parameter`-Objekt, welches den Parameterwert kapselt und die Art des Parameters durch einen Verweis auf das entsprechende `Parameter-Definition`-Objekt festlegt. Weiterhin legt das `Category`-Objekt für jeden Web Service der Kategorie ein `WebService`-Objekt an, welches wiederum über seine eigene `load`-Funktion eingelesen wird. Zum Einlesen und Speichern der Parameterwerte eines Web Services verwendet das `WebService`-Objekt wiederum ein `ParameterSet`-Objekt.

Beim Einlesen der Geschäftsprozessdefinitionsdatei liest das `Workflow`-Objekt zunächst die Gewichte zum Umrechnen der Gesamtparameterwerte eines Ausführungsplans in Nutzeneinheiten, sowie die Schrankenwerte bezüglich der Restriktionen auf den Gesamtparameterwerten ein. Hierbei wird in beiden Fällen ein `ParameterSet`-Objekt zum Einlesen und Ablegen der Gewichte, bzw. Schrankenwerte verwendet. Anschließend wird die Zeile zur Definition des Geschäftsprozesses in einzelne Prozessschritte zerlegt und für jeden Prozessschritt ein `TaskItem`-Objekt erzeugt, wobei die im Prozessschritt benötigte Funktionalität durch einen Verweis auf das entsprechende `Category`-Objekt festgelegt wird.

4.2.3 Erstellen des mathematischen Modells

Nachdem durch das `TestSetProcessor`-Objekt alle Testcases aus dem Eingabeverzeichnis in `TestCase`-Objekte eingelesen wurden, werden die einzelnen Tastcases in einer Schleife nacheinander abgearbeitet. Zur Vorbereitung auf die bevorstehende Optimierung durch das heuristische Optimierungsverfahren werden zunächst alle Kategorien durchlaufen und alle Web Services aus den Kategorien entfernt, welche den jeweiligen Restriktionen der Kategorie nicht entsprechen. Hierdurch befinden sich in den Kategorien nur noch jene Web Services, die zur Bildung eines Ausführungsplans tatsächlich verwendet werden können. Anschließend wird aus dem Testcase ein gemischt-ganzzahliges lineares Optimierungsproblem (GLOP) derart erzeugt, dass es durch den Solver lp_solve verarbeitet werden kann.

Das GLOP wird durch ein `MIPModel`-Objekt gekapselt, welches in einem privaten Attribut des `Data`-Objekts des Testcases abgelegt wird. Die Transformation des Testcases in ein GLOP wird durch die Methode `create` des `MIPModel`-Objekts vorgenommen. Das Optimierungsproblem wird hierbei genau wie in Kapitel 3.1.2 be-

schrieben aufgebaut. Die API des Solvers unterstützt zur Definition des Optimierungsproblems die Spezifikation einer Zielfunktion in der Form $F(\vec{x}) = \vec{w}^T \cdot \vec{x}$, sowie die Spezifikation von Nebenbedingungen in der Form $\vec{a}^T \cdot \vec{x} \leq \vec{c}$, $\vec{a}^T \cdot \vec{x} = \vec{c}$ und $\vec{a}^T \cdot \vec{x} \geq \vec{c}$. Die Abbildung der Nebenbedingungen geschieht in einer matrixorientierten Form, in der die Vektoren $\vec{a}$ der Nebenbedingungen die Zeilen einer Matrix A bilden. Abbildung 19 skizziert die schematische Matrixdarstellung des Optimierungsproblems.

$x_{1,1}$	$x_{1,2}$	...	x_{1,m_1}	...	$x_{n,1}$	$x_{n,2}$	...	x_{n,m_n}	x^+	$x^\bullet$	$x^{\min}$			
1	1	...	1									=	1	
				...								...		(1)
					1	1	...	1				=	1	
$p^+_{1,1}$	$p^+_{1,2}$	...	p^+_{1,m_1}		$p^+_{n,1}$	$p^+_{n,2}$	...	p^+_{n,m_n}	-1			=	0	(2)
									1			$\leq/\geq$	c^+	(3)
$-(1-p^\bullet_{1,1})$	$-(1-p^\bullet_{1,2})$	...	$-(1-p^\bullet_{1,m_1})$	...	$-(1-p^\bullet_{n,1})$	$-(1-p^\bullet_{n,2})$	...	$-(1-p^\bullet_{n,m_n})$		-1		=	-1	(4)
$\ln(p^\bullet_{1,1})$	$\ln(p^\bullet_{1,2})$	...	$\ln(p^\bullet_{1,m_1})$	...	$\ln(p^\bullet_{n,1})$	$\ln(p^\bullet_{n,2})$	...	$\ln(p^\bullet_{n,m_n})$				$\leq/\geq$	$\ln(c^\bullet)$	(5)
$p^{\min}_{1,1}$	$p^{\min}_{1,2}$	...	$p^{\min}_{1,m_1}$								-1	≥	0	
				...								...		(6)
					$p^{\min}_{n,1}$	$p^{\min}_{n,2}$	...	$p^{\min}_{n,m_n}$			-1	≥	0	
											1	≥	$c^{\min}$	(7)
									$\pm w^+$	$\pm w^\bullet$	$\pm w^{\min}$	→	max	(8)

Abbildung 19 - Schematische Matrixdarstellung des Optimierungsproblems

Bevor die Nebenbedingungen und die Zielfunktion des Optimierungsproblems festegelegt werden können, ist es zunächst nötig den Vektor $\vec{x}$ der Entscheidungsvariablen festzulegen (vgl. erste Zeile der Matrixdarstellung in Abbildung 19) und den Wertebereich jeder Variablen darin zu definieren. Hierzu werden alle Prozessschritte (`Task-Item`-Objekte) des Geschäftsprozesses (`Workflow`-Objekts) durchlaufen und zu jedem Prozessschritt über die ihm zugeordnete Kategorie (`Category`-Objekt) alle möglichen Web Services (`WebService`-Objekte) zur Realisierung der im Prozessschritt benötigten Funktionalität ermittelt. Für alle m_i Web Services zur Realisierung des Prozessschrittes i (i=1,...,n) wird eine Entscheidungsvariable $x_{i,j}$ eingeführt. Die Variablen $x_{i,j}$ werden als Binärvariablen deklariert, d.h. sie können nur den Wert 0 (Web Service wird nicht ausgeführt) oder 1 (Web Service wird ausgeführt) annehmen.

Im Anschluss daran wird zur Abbildung der k Gesamtparameterwerte für jeden der definierten k Parameter eine reellwertige Variable eingeführt. Zur Ermittlung der Anzahl und des Typs der Parameter werden die k `ParameterDefintion`-Objekte des `ParameterDefintions`-Objekts durchlaufen. Existieren k^+ additive Parameter, $k^\bullet$ multiplikative Parameter und $k^{\min}$ Minimaloperator-Parameter ($k^+ + k^\bullet + k^{\min} = k$), so werden k^+ Variablen x^+, $k^\bullet$ Variablen $x^\bullet$ und $k^{\min}$ Variablen $x^{\min}$ eingeführt. Aus Gründen der Übersicht und da alle Parameter gleichen Typs gleich behandelt werden, wird, ohne Beeinträchtigung der Allgemeinheit, im Folgenden lediglich von den Variablen x^+, $x^\bullet$ und $x^{\min}$ gesprochen, ohne diese weiter zu indizieren (vgl. erste Zeile der Abbildung 19 zwischen den zwei gestrichelten Linien).

Nachdem die Entscheidungsvariablen des Lösungsvektors $\vec{x}$ im Solver deklariert wurden, kann eine matrixorientierte Abbildung des Optimierungsproblems durchgeführt werden. Hierbei wird das mathematische Modell exakt wie bereits in Kapitel 3.1.2 beschrieben aufgebaut. Da dort bereits eine sehr ausführliche Darstellung des mathematischen Modells gegeben wurde, wird an dieser Stelle auf eine erneute detaillierte Betrachtung verzichtet. Stattdessen soll im Folgenden kurz dargelegt werden,

welche Bestandteile des im Kapitel 3.1.2 erläuterten mathematischen Modells in den einzelnen Abschnitten der matrixorientierten Darstellung aus Abbildung 19 umgesetzt werden. Die Nummerierung der Abschnitte kann der letzten Spalte in Abbildung 19 entnommen werden.

(1) Es muss exakt ein Web Service in jedem Prozessschritt ausgeführt werden.

$$\sum_{j=1}^{m_i} x_{i,j} = 1 \forall i = 1,...,n$$

(2) Definition eines additiven Gesamtparameterwertes x^+ auf Basis der Parameterwerte $p_{i,j}^+$ des zugrundeliegenden additiven Parameters p^+.

$$x^+ = \sum_{i=1}^{n} \sum_{j=1}^{m_i} p_{i,j}^+ x_{i,j}$$

(3) Restriktion eines additiven Gesamtparameterwertes x^+ durch einen Schrankenwert c^+ in der Form $x^+ \leq c^+$ oder $x^+ \geq c^+$.

(4) Definition eines multiplikativen Gesamtparameterwertes $x^\bullet$ auf Basis der Parameterwerte $p_{i,j}^\bullet$ des zugrundeliegenden multiplikativen Parameters $p^\bullet$. Das Produkt der Parameterwerte wird hierbei durch eine Summe angenähert.

$$x^\bullet = \prod_{i=1}^{n} \sum_{j=1}^{m_i} p_{i,j}^\bullet x_{i,j} \approx 1 - \sum_{i=1}^{n} \left(1 - \sum_{j=1}^{m_i} p_{i,j}^\bullet x_{i,j} \right)$$

(5) Restriktion eines multiplikativen Gesamtparameterwertes $x^\bullet$ durch einen Schrankenwert $c^\bullet$. Im Gegensatz zur Definition des Gesamtparameterwertes $x^\bullet$ kann die Restriktion exakt durch eine Linearisierung mittels Logarithmierung ausgedrückt werden und bedarf keiner Approximation eines Produkts durch eine Summe.

$$c^\bullet \leq \prod_{i=1}^{n} \prod_{j=1}^{m_i} p_{i,j}^{\bullet\, x_{i,j}} \Leftrightarrow \ln\left(c^\bullet\right) \leq \sum_{i=1}^{n} \sum_{j=1}^{m_i} \ln\left(p_{i,j}^\bullet\right) x_{i,j} \text{ oder}$$

$$c^\bullet \geq \prod_{i=1}^{n} \prod_{j=1}^{m_i} p_{i,j}^{\bullet\, x_{i,j}} \Leftrightarrow \ln\left(c^\bullet\right) \geq \sum_{i=1}^{n} \sum_{j=1}^{m_i} \ln\left(p_{i,j}^\bullet\right) x_{i,j}$$

(6) Definition eines Minimaloperator-Gesamtparameterwertes $x^{\min}$ auf Basis der Parameterwerte $p_{i,j}^{\min}$ des zugrundeliegenden Minimaloperator-Parameters $p^{\min}$. Die Definition geschieht durch mehrere Nebenbedingungen. Für jeden Prozessschritt i wird eine Nebenbedingung eingeführt, welche festlegt, dass der Gesamtparameterwert kleiner oder gleich dem Parameterwert des im jeweiligen Prozessschritt ausgeführten Web Services ist. Insgesamt ist er daher höchstens so groß wie der kleinste Parameterwert aller ausgeführter Web Services, d.h. er ist deren Minimum.

$$x^{\min} \leq \sum_{j=1}^{m_i} p_{i,j}^{\min} x_{i,j} \forall i = 1,...,n$$

(7) Restriktion eines Minimaloperator-Gesamtparameterwertes $x^{\min}$ durch einen Schrankenwert $c^{\min}$ in der Form $x^{\min} \geq c^{\min}$.

(8) Definition der zu maximierenden Zielfunktion F(x). Als zu maximierender Zielfunktionswert wird die gewichtete Summe der Gesamtparameterwerte, d.h. der Gesamtnutzen, berechnet.

$$F(x) = \pm w^+ x^+ \pm w^\bullet x^\bullet \pm w^{\min} x^{\min} \rightarrow \max!$$

Ob ein Gewicht $w^+, w^\bullet$ oder $w^{\min}$ positiv oder negativ eingeht, ist davon abhängig ob für den zugehörigen Gesamtparameterwert $x^+, x^\bullet$ oder $x^{\min}$ größere Werte einen höheren Nutzen stiften sollen oder niedrigere Werte. Dies ergibt sich aus der Definition des jeweiligen, dem Gesamtparameter zugrundeliegenden Parameters.

Die beim Aufbau des mathematischen Modells benötigten Informationen werden der Datenstruktur des `TestCase`-Objekts entnommen (vgl. Kapitel 4.2.2). Im Folgenden wird die Herkunft der einzelnen Informationen kurz erläutert.

$p^+, p^\bullet, p^{\min}$ Die Definitionen eines jeden Parameters ist im `ParameterDefinitions`-Objekt in Form eines `ParameterDefinition`-Objekts abgelegt. Aus dieser Definition ergibt sich, ob es sich bei einem Parameter um einen additiven Parameter (p^+), einen multiplikativen Parameter ($p^\bullet$) oder einen Minimaloperator-Parameter ($p^{\min}$) handelt und ob größere oder kleiner Parameterwerte des Parameters einen größeren Nutzen stiften.

$x^+, x^\bullet, x^{\min}$ Die Art und Weise, wie die Parameterwerte eines Parameters zu einem Gesamtparameterwert aggregiert werden müssen, d.h. ob es sich bei einem Gesamtparameter um einen additiven Gesamtparameter (x^+), einen multiplikativen Gesamtparameter ($x^\bullet$) oder um einen Minimaloperator-Gesamtparameter ($x^{\min}$) handelt, wird durch die Art des Parameters impliziert und ergibt sich daher ebenfalls aus den Informationen des dem Parameter zugeordneten `ParameterDefinition`-Objekts.

$c^+, c^\bullet, c^{\min}$ Die Schrankenwerte c, die den Gesamtparameterwert eines additiven Parameters (Schrankenwert c^+), eines multiplikativen Parameters (Schrankenwert $c^\bullet$) und eines Minimaloperator-Parameters (Schrankenwert $c^{\min}$) beschränken, können als `Parameter`-Objekte dem `ParameterSet`-Objekt im Attribut `constraints` des `Workflow`-Objekts entnommen werden. Ist dort ein solcher Schrankenwert nicht gesetzt, so existiert keine Restriktion auf dem entsprechenden Gesamtparameterwert. Der durch den Schrankenwert beschränkte Gesamtparameterwert ist über das `ParameterDefinition`-Objekt bestimmt, welches dem `Parameter`-Objekt im Attribut `definition` zugeordnet ist.

$p^+_{i,j}, p^\bullet_{i,j}, p^{\min}_{i,j}$ Die Eigenschaften eines Web Services in Form der ihm zugeordneten additiven Parameterwerte ($p^+_{i,j}$), multiplikativen Parameterwerte ($p^\bullet_{i,j}$) und Minimaloperator-Parameterwerte ($p^{\min}_{i,j}$), können als `Parameter`-Objekte dem `ParameterSet`-Objekt im Attribut `parameters` des `WebService`-Objekts des jeweiligen Web Services entnommen werden. Die Art des Parameters wird hierbei über das `ParameterDefinition`-Objekt bestimmt, welches dem `Parameter`-Objekt im Attribut `definition` zugeordnet ist.

$w^+, w^\bullet, w^{\min}$ Das Gewicht mit dem der Gesamtparameterwert eines additiven Parameters (Gewicht w^+), eines multiplikativen Parameters (Gewicht $w^\bullet$) und eines Minimaloperator-Parameters ($w^{\min}$) in der Zielfunkti-

on multipliziert wird um den Gesamtparameterwert in Nutzeneinheiten zu transformieren kann als `Parameter`-Objekt aus dem `ParameterSet`-Objekt im Attribut `weights` des `Workflow`-Objekts entnommen werden. Über das dem `Parameter`-Objekt zugeordneten `ParameterDefinition`-Objekt wird der Parameter und dadurch impliziert auch der Gesamtparameter bestimmt, der durch das Gewicht in der Zielfunktion zu gewichten ist. Ob das Gewicht positiv oder negativ in die Zielfunktion eingeht, wird über die Angabe im `ParameterDefinition`-Objekt festgelegt, die bestimmt ob größere oder kleinere Parameterwerte einen höheren Nutzen stiften.

Auf eine Darstellung der benutzten API des Solvers und des verwendeten Java-Wrappers der API soll an dieser Stelle verzichtet werden. Weiterführende Informationen hierzu können der Dokumentation des Solvers [2] entnommen werden.

4.3 OpeningMethod – Schritt 1 und 2 der Heuristik

Zur Vorbereitung des heuristischen Lösungsprozesses wurde der Testcase aus den Definitionsdateien in die Testcase-Datenstruktur eingelesen und hieraus das mathematische Modell des Optimierungsproblems erzeugt, das durch den verwendeten Solver lp_solve gelöst werden kann. Damit sind alle Voraussetzungen gegeben, die einzelnen Schritte der Heuristik abarbeiten zu können. Tabelle 9 zeigt, welche Klassen die einzelnen Schritte der Heuristik implementieren.

Schritt der Heuristik		Implementierende Klasse
1.	Exakte Lösung des LP-relaxierten Problems durch einen Simplex-Algorithmus.	`OpeningMethod`
2.	Konstruktion einer ersten zulässigen Lösung mittels eines Backtracking-Algorithmus.	
3.	Anwendung eines reinen Verbesserungsverfahrens zur (lokalen) Optimierung.	`UnsophisticatedImprovement`
4.	Anwendung der Metaheuristik Simulated Annealing zur (möglichst globalen) Optimierung.	`SimulatedAnnealing`

Tabelle 9 - Implementierende Klassen der Schritte der Heuristik

Die Heuristik beginnt im ersten Schritt mit der exakten Lösung des relaxierten Optimierungsproblems durch den Solver. Anschließend kann auf Basis dieser Lösung durch das Backtracking-Verfahren im Schritt zwei der Heuristik eine erste zulässige Lösung für das unrelaxierte Optimierungsproblem erzeugt werden.

Eine Lösung des Optimierungsproblems, d.h. ein konkreter Ausführungsplan zum Geschäftsprozess, wird durch die Klasse `ExecutionPlan` gekapselt, welche die einzelnen Prozessschritte und den dort jeweils auszuführenden Web Service in einer Liste von `ExecutionPlanItem`-Objekten ablegt. Um einen leeren Ausführungsplan passender Länge zu erstellen, der im Laufe der Heuristik mit konkreten Web Services zur Ausführung der Prozessschritte gefüllt wird, kann die statische Funktion `createCurrentExecutionPlan` der `ExecutionPlan`-Klasse verwendet werden, der als

Argument das `Workflow`-Objekt übergeben wird, das den zu optimierenden Geschäftsprozess repräsentiert.

Die Schritte eins und zwei der Heuristik, die einen ersten zulässigen Ausführungsplan erzeugen, sind in der Klasse `OpeningMethod` zusammengefasst. Bei der Erzeugung einer neuen Instanz der Klasse wird im Konstruktor ein leerer Ausführungsplan des Geschäftsprozesses übergeben. Durch Ausführung der Methode `solve` des erzeugten Objekts werden die ersten beiden Schritte der Heuristik ausgeführt und der hierbei erzeugte Ausführungsplan in jenem `ExecutionPlan`-Objekt abgelegt, welches im Konstruktor übergeben wurde. Die Schritte eins und zwei der Heuristik werden hierbei wie in den Kapiteln 3.2.1 und 3.2.2 beschrieben durchgeführt. Die Klasse `OpeningMethod` implementiert das dort erläuterte Vorgehen.

4.3.1 Realisierung des ersten Schritts der Heuristik

Zur Umsetzung des ersten Schritts der Heuristik wird das, für den Solver aufgebaute mathematische Optimierungsmodell, welches durch ein `MIPModel`-Objekt gekapselt wird, zunächst relaxiert und anschließend durch den Solver gelöst. Die Relaxation wird durch einen Aufruf der Methode `relax` des `MIPModel`-Objekts durchgeführt und geschieht durch die Aufhebung der Ganzzahligkeitsbedingung auf den Entscheidungsvariablen $x_{i,j}$. Der Wertebereich der Entscheidungsvariablen $x_{i,j}$ bleibt auf das Intervall [0;1] beschränkt, jedoch kann eine Entscheidungsvariable $x_{i,j}$ nach der Relaxation einen beliebigen nicht-ganzzahligen Wert aus diesem Intervall annehmen. Durch den Aufruf der Funktion `solve` des `MIPModel`-Objekts wird das relaxierte Optimierungsproblem durch den Solver gelöst. Durch die Relaxation ist der Solver in der Lage die optimale Lösung des Optimierungsproblems durch den Simplex-Algorithmus zu bestimmen. Da eine zusätzliche Anwendung des aufwendigen Brach&Bound-Verfahrens aufgrund des Fehlens ganzzahliger Entscheidungsvariablen nicht nötig ist, kann die optimale Lösung des relaxierten Problems sehr performant erzeugt werden. Konnte der Solver eine Lösung ermitteln, so liefert die Funktion `solve` den Wahrheitswert `true` zurück, ansonsten `false`. Kann keine Lösung für das relaxierte Optimierungsproblem ermittelt werden, so bedeutet dies, dass auch das unrelaxierte Optimierungsproblem keine Lösung besitzen kann und die Methode `solve` des `OpeningMethod`-Objekts wird verlassen, ohne den ursprünglich übergebenen, leeren Ausführungsplan zu verändern.

Konnte hingegen eine Lösung für das relaxierte Optimierungsproblem gefunden werden, so handelt es sich dabei um die optimale Lösung des relaxierten Optimierungsproblems. Der Zielfunktionswert dieser Lösung wird vom Solver erfragt und wird im `MIPModel`-Objekt im Attribut `lastOptimalValue` gespeichert. Dieser Zielfunktionswert bildet eine obere Schranke für den Zielfunktionswert des unrelaxierten Optimierungsproblems und wird im Folgenden beim Übergang zwischen den einzelnen Schritten der Heuristik dazu verwendet, die Güte des bisher ermittelten Ausführungsplans abzuschätzen um so die Heuristik beim Erreichen einer hinreichend hohen Lösungsgüte vorzeitig abbrechen zu können.

4.3.2 Realisierung des zweiten Schritts der Heuristik

Der zweite Schritt der Heuristik wird ebenfalls innerhalb der Funktion `solve` des `OpeningMethod`-Objekts ausgeführt. Mithilfe des Backtracking-Verfahrens wird

hier ein erster zulässiger Ausführungsplan ermittelt. Um die Effektivität des Backtracking-Verfahrens zu steigern, werden vor der Anwendung des Backtracking-Verfahrens die Prozessschritte und die jeweils für einen Prozessschritt möglichen Web Services gemäß der Lösung des relaxierten Optimierungsproblems vorsortiert. Die Vorsortierung wird durch die Funktion `init` des `OpeningMethod`-Objekts durchgeführt. Die Sortierung erfolgt, wie im Abschnitt „Optimierung des Verfahrens" des Kapitels 3.2.2 beschrieben, hauptsächlich nach der geschätzten Wahrscheinlichkeit mit der ein Web Service für einen bestimmten Prozessschritt die optimale Besetzung darstellt und wie sicher eine Entscheidung auf Basis der geschätzten Wahrscheinlichkeit für jeden Prozessschritt jeweils ist. Die zur Sortierung benötigten Werte der Entscheidungsvariablen $x_{i,j}$ aus der optimalen Lösung des relaxierten Optimierungsproblems werden über entsprechende API-Aufrufe vom Solver ausgelesen und in Arrays von `ExecutionPlanItemWSSortInfo`-Objekten, gruppiert nach Prozessschritten im jeweils zugehörigen `ExecutionPlanItem`-Objekt abgelegt. Die eigentliche Sortierung wird anschließend auf Basis der statischen Methode `java.util.Arrays.sort` und einer angepassten Implementierung des `java.util.Comparator`-Interfaces realisiert, worauf an dieser Stelle jedoch nicht näher eingegangen werden soll. Nach der Sortierung ist für jeden Prozessschritt bekannt, in welcher Reihenfolge die Web Services an der entsprechenden Position im Ausführungsplan durch das Backtracking-Verfahren gesetzt werden müssen. Zusätzlich wird die Liste `positionVector` erstellt, welche die Indizes der Positionen des Ausführungsplans in der Reihenfolge enthält, in der das Backtracking-Verfahren diese Positionen mit Web Services zu besetzten hat.

Als Rückgabewert der Funktion `init`, welche die Vorsortierung der Web Services und Prozessschritte vorgenommen hat, wird der erstellte `positionVector` geliefert. Dieser wird nun als Argument an die Funktion `solveWithPositionVector` des `OpeningMethod`-Objekts übergeben, die das eigentliche Backtracking-Verfahren realisiert. Die Positionen des Ausführungsplans (`ExecutionPlanItem`-Objekte) werden, in der durch den `positionVector` vorgegebenen Reihenfolge, mit Web Services besetzt, wobei die Web Services in der Reihenfolge ihrer Vorsortierung gesetzt werden. Die Reihenfolge, mit der die Web Services an einer Position zu setzen sind, kann in jedem `ExecutionPlanItem`-Objekt dem sortierten Array von `ExecutionPlanItemWSSortInfo`-Objekten entnommen werden. Das eigentliche Backtracking-Verfahren wurde wie in Kapitel 3.2.2 beschrieben implementiert, weshalb an dieser Stelle auf eine genauere Betrachtung verzichtet werden soll. Anzumerken ist jedoch, dass zum Test der Zulässigkeit eines Ausführungsplans im Rahmen des Backtracking-Verfahrens, die Funktion `satisfiesConstraints` des `ExecutionPlan`-Objekts verwendet wird. Sie berechnet alle Gesamtparameterwerte des Ausführungsplans gemäß der Parameterdefinitionen aus den Parametern der im Ausführungsplan gesetzten Web Services und vergleicht Sie mit den jeweiligen Schrankenwerten definierter Restriktionen. Wird mindestens eine Restriktion durch den Ausführungsplan verletzt, so liefert die Funktion den Wahrheitswert `false` zurück, ansonsten `true`.

Ist es dem Backtracking-Verfahren möglich alle Positionen des Ausführungsplan so mit Web Services zu besetzen, dass dieser zulässig ist, so ist ein erster zulässiger Ausführungsplan gefunden und die Funktion `solveWithPositionVector` liefert den Wahrheitswert `true` zurück. Konnte hingegen kein zulässiger Ausführungsplan gefunden werden, so wird `false` zurückgeliefert. Da zur Feststellung, dass kein zuläs-

siger Ausführungsplan existiert, vom Backtracking-Verfahren bereits alle möglichen Ausführungspläne erzeugt wurden, ist das unrelaxierte Optimierungsproblem unlösbar und die Heuristik kann an dieser Stelle abgebrochen werden.

4.4 UnsophisticatedImprovementMethod – Schritt 3 der Heuristik

Konnte durch das `OpeningMethod`-Objekt ein zulässiger Ausführungsplan erzeugt werden, so wird der Zielfunktionswert der Lösung, die durch diesen Ausführungsplan repräsentiert wird, über die Funktion `quality`, des ihn repräsentierenden `ExecutionPlan`-Objekts, ermittelt. Hierzu berechnet die Funktion zu allen durch die Parameterdefinitionen definierten Parametern den Gesamtparameterwert durch Aggregation der entsprechenden Parameterwerte aller im Ausführungsplan gesetzten Web Services. Anschließend werden die Gesamtparameterwerte, durch Multiplikation mit dem ihnen in der Zielfunktion zugeordneten Gewicht, in Nutzeneinheiten transformiert und die Nutzeneinheiten zum Gesamtnutzen addiert. Je nachdem ob für einen Parameter definiert wurde, dass kleinere oder größere Parameterwerte einen höheren Nutzen stiften sollen, gehen die jeweiligen, durch den Gesamtparameterwert des Parameters erzeugten Nutzeneinheiten, positiv oder negativ in die Summe des Gesamtnutzens ein. Weicht der so berechnete Zielfunktionswert des Ausführungsplans weniger als 1% von der, im Schritt eins der Heuristik berechneten, oberen Schranke für den Zielfunktionswert ab, so wird die Güte des Ausführungsplans als hinreichend hoch eingestuft und die Heuristik beendet. Beträgt die Abweichung zur Schranke mehr als 1%, so wird vermutet, dass es sich um einen suboptimalen Ausführungsplan handelt und die Heuristik wird mit Schritt drei fortgesetzt (siehe hierzu auch Kapitel 3.2.3).

Schritt drei der Heuristik versucht durch ein gieriges, myopisches Verbesserungsverfahren eine schnellstmögliche Steigerung des Zielfunktionswertes herbeizuführen. Hierzu werden zum aktuell besten, bekannten Ausführungsplan Nachbarlösungen durch den zufälligen Austausch je eines Web Services erzeugt. Wird hierbei eine Nachbarlösung gefunden, die einen höheren Zielfunktionswert besitzt, so wird sie zur aktuell besten, bekannten Lösung und es wird mit der Untersuchung ihrer Nachbarlösungen fortgefahren. Das Verfahren endet, sobald in einer bestimmten Anzahl aufeinander folgender Schritte nur solche Nachbarlösungen erzeugt wurden, die zu keiner Steigerung des Zielfunktionswertes geführt haben.

Das in Schritt drei der Heuristik angewendete Optimierungsverfahren wird durch die Klasse `UnsophisticatedImprovement` bereitgestellt. Nach der Ausführung der Funktion `solve` des `OpeningMethod`-Objekts, liegt der mittels des Backtracking-Verfahrens erstellte Ausführungsplan in dem `ExecutionPlan`-Objekt vor, welches dem `OpeningMethod`-Objekt bei seiner Erzeugung im Konstruktor übergeben wurde. Um diesen Ausführungsplan durch das Optimierungsverfahren von Schritt drei der Heuristik weiter optimieren zu lassen, wird eine Instanz der Klasse `UnsophisticatedImprovement` erzeugt und das `ExecutionPlan`-Objekt im Konstruktor übergeben. Durch den Aufruf der Funktion `optimize` des `UnsophisticatedImprovement`-Objekts wird das Optimierungsverfahren auf den Ausführungsplan angewendet und der optimierte Ausführungsplan als Rückgabewert in Form eines `ExecutionPlan`-Objekts von der Funktion zurückgeliefert.

Die Implementierung der Funktion `optimize` setzt das Optimierungsverfahren des Schritts drei der Heuristik wie in Kapitel 3.2.3 bereits ausführlich beschrieben um und sei daher an dieser Stelle nicht weiter erläutert. Die zur Durchführung des Optimierungsverfahrens nötigen Informationen werden den Datenstrukturen entnommen, die im bisherigen Teil des Kapitels 4 bereits vorgestellt wurden. Der Faktor u, der zur Berechnung der Anzahl der maximal aufeinander folgenden Optimierungsschritte ohne Verbesserung des Zielfunktionswertes verwendet wird, wird den Angaben unter dem Reiter „Advanced Settings“ des Aktionsbereichs des WorkflowOptimizer-Fensters entnommen.

4.5 SimulatedAnnealing – Schritt 4 der Heuristik

Nachdem das Optimierungsverfahren aus Schritt drei der Heuristik auf den Ausführungsplan angewendet wurde und der optimierte Ausführungsplan in Form eines `ExecutionPlan`-Objekts von der Funktion `optimize` des `Unsophisticated-Improvement`-Objekts zurückgeliefert wurde, wird wiederum geprüft, ob der Zielfunktionswert des Ausführungsplans nicht weniger als 1% von der oberen Schranke des Zielfunktionswertes abweicht. Ist die Abweichung geringer als 1%, so wird die Güte des Ausführungsplans als hinreichend hoch eingestuft und die Heuristik beendet. Beträgt die Abweichung mehr als 1%, so wird angenommen, dass es sich immernoch um einen suboptimalen Ausführungsplan handelt und die Heuristik wird mit Schritt vier fortgesetzt um den Ausführungsplan weiter zu optimieren.

In Schritt vier der Heuristik wird das Simulated-Annealing-Verfahren zur Optimierung des Ausführungsplans angewendet. Eine ausführliche Beschreibung des Verfahrens kann Kapitel 3.2.4 entnommen werden. Das Verfahren akzeptiert im Gegensatz zum gierigen, myopischen Optimierungsverfahren, welches in Schritt drei der Heuristik zur Anwendung kommt, auch solche Nachbarlösungen, die einen schlechteren Zielfunktionswert aufweisen. Nachbarlösungen mit besserem Zielfunktionswert werden jederzeit durch das Verfahren akzeptiert, wohingegen die Annahmewahrscheinlichkeit einer Nachbarlösung mit schlechterem Zielfunktionswert von der Höhe der Verschlechterung des Zielfunktionswertes und einem Temperaturparameter abhängig ist. Die Akzeptanzwahrscheinlichkeit ist bei stärkeren Verschlechterungen niedriger und sinkt zudem mit abnehmendem Temperaturparameter, der im Laufe des Verfahrens gesenkt wird. Dies führt dazu, dass zu Beginn des Verfahrens noch oft solche Nachbarlösungen akzeptiert werden, die einen niedrigeren Zielfunktionswert aufweisen, jedoch werden im Verlauf des Verfahrens Nachbarlösungen gleicher Verschlechterung mit einer stetig abnehmenden Wahrscheinlichkeit akzeptiert, bis zum Ende des Verfahrens hin nahezu nur noch Nachbarlösungen akzeptiert werden, die zu einer Verbesserung des Zielfunktionswertes führen. Dieses Verhalten ermöglicht es lokale Optima des Zielfunktionswertes durch die temporäre Akzeptanz von Verschlechterungen verlassen zu können um hierdurch zum Ende des Verfahrens hin das globale Optimum des Zielfunktionswertes erreichen zu können.

Das in Schritt vier der Heuristik angewendete Simulated-Annealing-Verfahren wird durch die Klasse `SimulatedAnnealing` bereitgestellt. Nach der Ausführung der Funktion `optimize` des `UnsophisticatedImprovement`-Objekts, wird der mittels des Optimierungsverfahrens des dritten Schritts der Heuristik optimierte Ausführungsplan als Rückgabewert der Funktion in einem `ExecutionPlan`-Objekt zurückgegeben. Um diesen Ausführungsplan durch das Simulated-Annealing-Verfahren weiter optimieren zu lassen, wird eine Instanz der Klasse `SimulatedAnnealing` er-

zeugt und das `ExecutionPlan`-Objekt im Konstruktor übergeben. Durch den Aufruf der Funktion `optimize` des `SimulatedAnnealing`-Objekts wird das Simulated-Annealing-Verfahren auf den Ausführungsplan angewendet und der optimierte Ausführungsplan als Rückgabewert in Form eines `ExecutionPlan`-Objekts von der Funktion zurückgeliefert.

Die Implementierung der Funktion `optimize` setzt das Simulated-Annealing-Verfahren wie in Kapitel 3.2.4 bereits ausführlich beschrieben um und sei daher an dieser Stelle nicht weiter erläutert. Die zur Durchführung des Simulated-Annealing-Verfahrens nötigen Informationen werden den Datenstrukturen entnommen, die im bisherigen Teil des Kapitels 4 bereits vorgestellt wurden. Parameter, die das Verhalten des Simulated-Annealing-Verfahrens beeinflussen werden den Angaben unter dem Reiter „Advanced Settings“ des Aktionsbereichs des WorkflowOptimizer-Fensters entnommen. Hierzu zählen z.B. der Faktor u zur Berechnung der Anzahl der durchzuführenden Iterationen in Phase eins des Simulated-Annealing-Verfahrens, der Temperaturparameterwert α_1 beim Eintritt in Phase eins, der Temperaturparameterwert α_2 beim Beenden von Phase eins, der Faktor v zur Berechnung der maximal erlaubten Anzahl von Iterationen ohne Verbesserung des Zielfunktionswertes in Phase zwei, sowie der Faktor c_α, durch den die Abnahme des Temperaturparameters α in Phase zwei mittels Multiplikation herbeigeführt wird.

4.6 SequentialWorkflowEngine – Simulation einer BPE

In Kapitel 4.2.2 wurde geschildert, dass nach dem Start des Optimierungsprozesses über die graphische Benutzeroberfläche von WorkflowOptimizer ein `TestSet-Processor`-Objekt erzeugt und in einem eigenen Thread ausgeführt wird um die Testcases aus dem Eingabeverzeichnis einzulesen, zu initialisieren und nacheinander abzuarbeiten. In den Kapiteln 4.3 bis 4.5 wurde anschließend beschrieben wie zu dem Geschäftsprozess eines Testcases durch die stufenweise Anwendung von Klassen, welche die Schritte der Heuristik implementieren, ein optimierter Ausführungsplan erstellt wird. An dieser Stelle wurde jedoch aus Gründen der Vereinfachung unterschlagen, dass die Erstellung des optimierten Ausführungsplans in einer simulierten BPE stattfindet.

Die BPE hat die Aufgabe einen Geschäftsprozess durch den Aufruf einzelner Web Services auszuführen. Hierzu werden alle Prozessschritte des Geschäftsprozesses nacheinander durchlaufen, wobei zu jedem Prozessschritt ein Web Service ausgeführt wird, der die im Prozessschritt geforderte Funktionalität realisiert. Ein solcher Schritt in der Abarbeitung des Geschäftsprozesses wird *Step* genannt. In jedem Step wird der zur Ausführung eines Prozessschrittes zu verwendende Web Service bestimmt und anschließend ausgeführt. Die Auswahl eines konkreten Web Services, aus der Menge aller für diesen Prozessschritt geeigneten Web Services, kann hierbei nicht unabhängig von allen anderen Prozessschritten erfolgen, da die Ausführung des Geschäftsprozesses insgesamt bestimmte Restriktionen erfüllen muss und zudem nach gewissen Kriterien optimiert erfolgen soll. Daher ist bereits im ersten Step ein Ausführungsplan zu erstellen, der alle Prozessschritte, d.h. den gesamten Geschäftsprozess umfasst. Nur so kann die BPE sicherstellen, dass auch in den folgenden Steps nur solche Web Services ausgeführt werden, die in ihrer Gesamtheit die Restriktionen nicht verletzten und zu einem optimalen Gesamtverhalten des Geschäftsprozesses führen werden.

Ändert sich die Datenbasis nicht, aufgrund derer der Ausführungsplan im ersten Step erstellt wurde, so bleibt er während der kompletten Ausführung des Geschäftsprozesses gültig. In diesem Fall muss die BPE lediglich nacheinander die Web Services ausführen, die ihr durch den Ausführungsplan vorgegeben werden. Wird jedoch festgestellt, dass die Annahmen über bereits ausgeführte Web Services falsch waren, oder dass sich die Eigenschaften noch nicht ausgeführter Web Services geändert haben, so wird der Ausführungsplan ungültig und die BPE muss während der Ausführung des Geschäftsprozesses einen neuen optimierten Ausführungsplan erstellen, d.h. ein Replanning durchführen (siehe Kapitel 3.3). Nur durch die Abarbeitung des neuen Ausführungsplans kann die BPE sicherstellen, dass auch weiterhin alle Restriktionen eingehalten und die Ausführung des Geschäftsprozesses optimal verläuft. Ist die Datenbasis sehr ungenau oder ändert sie sich häufig, muss die BPE sehr oft einen neuen Ausführungsplan erstellen. Im schlimmsten Fall muss in jedem Step ein Replanning durchgeführt werden.

WorkflowOptimizer verwendet zur simulierten Ausführung eines Geschäftsprozesses eine BPE die vom Worst-Case ausgeht und in jedem Step ein Replanning durchführt. Im ersten Step wird ein Ausführungsplan für den gesamten Geschäftsprozess erstellt. In jedem weiteren Step wird, bedingt durch das Replanning, ein Ausführungsplan für ein jeweils um einen Prozessschritt verkürzten Geschäftsprozess erstellt. Hierdurch wird es möglich das Verhalten der Heuristik sowohl für den Fall zu untersuchen, in welchem es auf den gesamten ursprünglichen Geschäftsprozess angewendet wird, als auch in den Fällen, in denen es auf die, durch das Replanning entstehenden, verkürzten Geschäftsprozesse angewendet wird. Zusätzlich kann das Gesamtverhalten einer BPE im Worst-Case untersucht werden, welches durch das permanente Replanning absichtlich herbeigeführt wird.

4.6.1 Simulierte Ausführung

Die BPE zur simulierten Ausführung der Geschäftsprozesse wird durch die Klasse `SequentialWorkflowEngine` bereitgestellt. Nach der Initialisierung eines Testcases im `TestSetProcessor`-Objekt wird eine Instanz der Klasse `SequentialWorkflowEngine` erzeugt und hierbei das `Data`-Objekt des Testcases im Konstruktor übergeben. Hierdurch verfügt die BPE über alle Daten des Testcases und insbesondere über den Geschäftsprozess, den sie in simulierter Art und Weise auszuführen hat. Durch den Aufruf der Methode `simulateExecution`, des erzeugten `SequentialWorkflowEngine`-Objekts, wird die simulierte Ausführung des Geschäftsprozesses gestartet.

In der Methode `simulateExecution` werden die Prozessschritte (`TaskItem`-Objekte) des Geschäftsprozesses (`Workflow`-Objekts) in einer Schleife nacheinander durchlaufen. Ein Schleifendurchlauf entspricht einem Step. In jedem Step wird ein Replanning durchgeführt und der Web Service für den nächsten Prozessschritt in simulierter Form ausgeführt. Die Simulation der Ausführung besteht hierbei lediglich in der Feststellung, welcher Web Service zur Realisierung des Prozessschrittes durch eine reale BPE ausgeführt worden wäre. Dieser Web Service wird im ausgeführten Prozessschritt, d.h. dem aktuell betrachteten `TaskItem`-Objekt, im Attribut `usedWebService` gespeichert.

Durch das permanente Replanning der BPE ist es nötig vor der Ausführung jedes einzelnen Prozessschrittes erneut einen optimalen Ausführungsplan durch die Heuristik

berechnen zu lassen. Mit jedem ausgeführten Prozessschritt verkürzt sich das im nächsten Prozessschritt zu lösende Optimierungsproblem um den letzten, bereits ausgeführten Prozessschritt. Vor der Lösung des Optimierungsproblems ist dieses daher in jedem Prozessschritt an den aktuellen Stand der Abarbeitung des Geschäftsprozesses anzupassen und anschließend durch die Heuristik zu lösen um einen neuen, optimierten Ausführungsplan zu erhalten. Die Anpassung des mathematischen Optimierungsmodells, welches durch das `MIPModel`-Objekt gekapselt wird, geschieht in jedem Step durch den Aufruf der Methode `adjustToStep` des `MIPModel`-Objekts. Anschließend müssen alle Schritte der Heuristik wie in den Kapiteln 4.3 bis 4.5 beschrieben ausgeführt werden um einen neuen, optimierten Ausführungsplan zu erhalten. Anschließend kann der Web Service an der ersten Position des Ausführungsplans ausgeführt, bzw. im `TaskItem`-Objekt des aktuellen Prozessschrittes gespeichert werden. Nachdem alle Steps durchlaufen sind, d.h. alle Prozessschritte ausgeführt wurden, ist die Ausführung des gesamten Geschäftsprozesses abgeschlossen, der Testcase damit abgearbeitet und die BPE kann verlassen werden.

An dieser Stelle sei darauf hingewiesen, dass im letzten Step nicht mehr die Heuristik dazu verwendet wird um einen optimierten Ausführungsplan zu erstellen. Da im letzten Prozessschritt das Optimierungsproblem nur noch einen Geschäftsprozess mit nur einem Prozessschritt umfasst, lässt sich der optimale Web Service für diesen Prozessschritt sehr leicht durch das Ausprobieren aller, für diesen Prozessschritt geeigneten, Web Services ermitteln. Diese Funktionalität wird durch die Klasse `SolveSingle` und ihrer Funktion `solve` bereitgestellt.

Das Vorgehen des `SequentialWorkflowEngine`-Objekts beim simulierten Ausführen eines Geschäftsprozesses durch die Methode `simulateExecution` soll durch das folgende Aktivitätsdiagramm noch einmal zusammenfassend veranschaulicht werden.

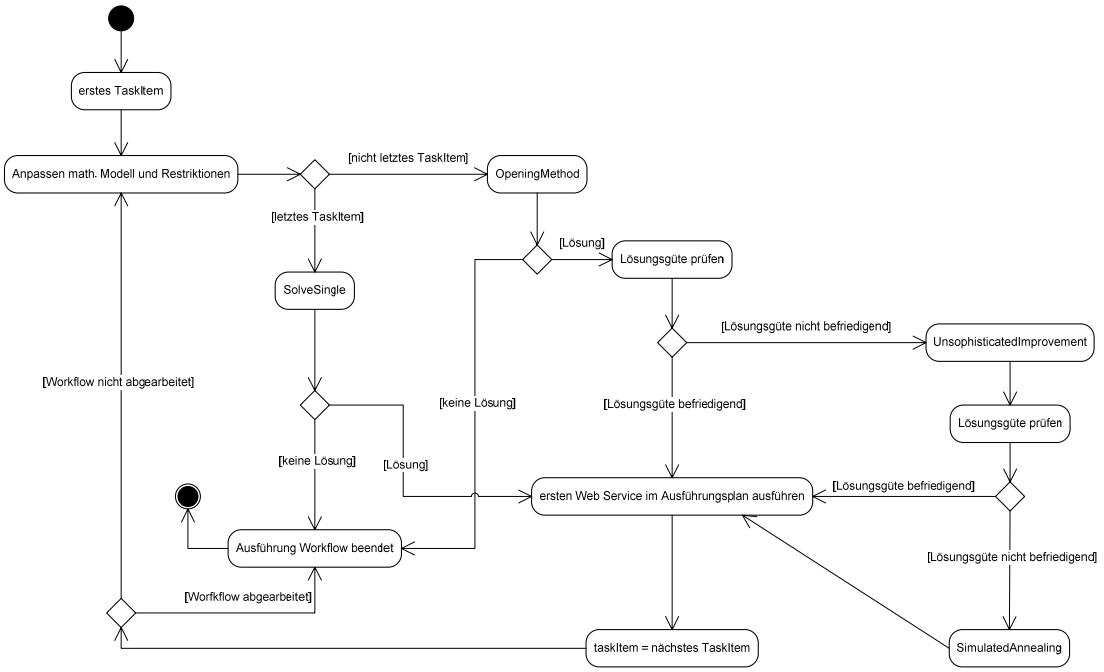

Abbildung 20 - Geschäftsprozessausführung durch die Klasse SequentialWorkflowEngine

Nachdem der Geschäftsprozess durch das `SequentialWorkflowEngine`-Objekt ausgeführt wurde, können die tatsächlich in den Prozessschritten ausgeführten Web Services dem Attribut `usedWebService` der jeweiligen `TaskItem`-Objekte des `Workflow`-Objekts entnommen werden. Die Klasse `ExecutionPlan` stellt darüber hinaus die statische Funktion `createExecutionPlan` bereit, die aus dem übergebenen `Workflow`-Objekt einen Ausführungsplan (`ExecutionPlan`-Objekt) erstellt, der genau die Web Services enthält, die tatsächlich ausgeführt wurden. Über die Funktion `quality` des `ExecutionPlan`-Objekts lässt sich anschließend der Zielfunktionswert des Ausführungsplans berechnen.

Der tatsächliche Ausführungsplan ist das eigentliche Ergebnis, welches durch die BPE erreicht wurde. Die zuvor ermittelten Ausführungspläne waren, bedingt durch das permanente Replanning, lediglich theoretischer Natur, da ihre Bestimmung nur dazu diente den Web Service für den nächsten auszuführenden Prozessschritt zu bestimmen.

4.6.2 Einfluss des Replannings auf die Gesamtlösung

Durch das permanente Replanning wird in jedem Step vor der Ausführung eines Prozessschrittes durch die Heuristik ein neuer Ausführungsplan für den noch nicht ausgeführten Teil des Geschäftsprozesses erstellt. Für einen Geschäftsprozess mit n Prozessschritten werden hierdurch im Laufe seiner Abarbeitung n Ausführungspläne erstellt. Da zur Ausführung eines Prozessschrittes in einem Step jeweils der erste Web Service des aktuellen Ausführungsplans ausgeführt wird, ergibt sich der tatsächlich zur Ausführung kommende Ausführungsplan aus der Aneinanderreihung von n Web Services aus n unterschiedlichen Ausführungsplänen. Bedingt durch den Nichtdeterminismus der Heuristik kann die Güte der n Ausführungspläne jeweils schwanken, wodurch die Güte des tatsächlich ausgeführten Ausführungsplans sowohl positiv, als auch negativ beeinflusst werden kann.

Ein Ausführungsplan lässt sich als geordnete Liste von Indizes x_i darstellen. Der erste Index x_1 definiert den im ersten Prozessschritt auszuführenden Web Service, der zweite Index x_2 definiert den im zweiten Prozessschritt auszuführenden Web Service, usw. usf.. Jeder Index stellt hierbei den Index des Web Services in der jeweiligen, dem Prozessschritt zugeordneten, Kategorie dar. Ein Ausführungsplan a lässt sich so kompakt in der Form $a = <x_1, x_2, ..., x_n>$ ausdrücken. Wird beispielsweise zur Realisierung des ersten Prozessschrittes im Ausführungsplan der vierte Web Service der entsprechenden Kategorie festegesetzt, so ist x_1 mit dem Wert 4 zu belegen. Ein konkreter Ausführungsplan für einen Geschäftsprozess mit drei Prozessschritten könnte $a = <4, 1, 3>$ sein, d.h. in Prozessschritt eins wird aus der entsprechenden Kategorie der Web Service mit dem Index vier ausgeführt, in Prozessschritt zwei wiederum aus der entsprechenden Kategorie der Web Service mit dem Index eins und in Prozessschritt drei letztlich aus der entsprechenden Kategorie der Web Service mit dem Index drei.

Zu einem beliebigen Zeitpunkt der Ausführung des Geschäftsprozesses sei ein Ausführungsplan a ermittelt worden. Der durch den Index x_1 repräsentierte erste Web Service des Ausführungsplans wird nun ausgeführt. Anschließend erfolgt ein Replanning. Bei einem Replanning wird der Geschäftsprozess um den zuletzt ausgeführten Prozessschritt gekürzt, die Restriktionen an den verbleibenden Geschäftsprozessteil angepasst und durch die Heuristik erneut ein Ausführungsplan a' für diesen erstellt.

Bleibt die Datenbasis konstant und wäre die Heuristik deterministisch, so würde der neue Ausführungsplan a' aus dem vorherigen Ausführungsplan $a = <x_1, x_2, ..., x_n>$ durch das Entfernen des ersten Index entstehen, da in einem optimierten Ausführungsplan zur Realisierung des noch ausstehenden Teils des Geschäftsprozesses exakt dieselben Web Services ausgewählt würden wie zuvor, d.h. a' wäre $<x_2, ..., x_n>$. Als Beispiel sei ein Geschäftsprozess betrachtet, der aus drei Prozessschritten besteht. Würde im ersten Step durch die Heuristik der Ausführungsplan <4, 1, 3> ermittelt werden, so wäre der im zweiten Step ermittelte Ausführungsplan <1, 3> und der im dritten Step ermittelte Ausführungsplan <3>. Da zur Realisierung eines Prozessschrittes jeweils der erste Web Service des aktuellen Ausführungsplans ausgeführt wird, wäre der tatsächlich zur Ausführung gebrachte Ausführungsplan <4, 1, 3> und damit identisch zu dem im ersten Step erstellten Ausführungsplan.

Da die Heuristik jedoch nicht deterministisch ist, ist es nicht sicher, dass sie angewendet auf dasselbe Optimierungsproblem stets denselben Ausführungsplan mit derselben Güte erzeugt. Dies bedeutet, dass es durch das Replanning möglich ist, dass auch bei konstanter Datenbasis für ein und denselben zukünftig noch auszuführenden Prozessschritt in jedem Ausführungsplan ein anderer Web Service als optimal angesehen werden kann. Hierdurch ergibt sich ein neu erstellter Ausführungsplan a' nicht mehr zwangsläufig durch das Kürzen des vorherigen Ausführungsplans um den ersten Index. Im genannten Beispiel könnte so im zweiten Step der ermittelte Ausführungsplan <2, 2> und im dritten Step der ermittelte Ausführungsplan <4> sein. Tatsächlich wäre also bei der Ausführung des Geschäftsprozesses der Ausführungsplan <4, 2, 4> ausgeführt worden. Dieser Ausführungsplan war jedoch nie direkt das Ergebnis einer Optimierung durch die Heuristik, sondern wurde aus mehreren Ausführungsplänen unterschiedlicher Güte, die durch verschiedene Läufe der Heuristik erstellt wurden, zusammengesetzt. Der Ausführungsplan erfüllt zwar alle Restriktionen, die an die Ausführung des Geschäftsprozesses gestellt wurden, jedoch wird die Güte des tatsächlich ausgeführten Ausführungsplans durch das Mischen von Teilergebnissen aus qualitativ unterschiedlichen Ausführungsplänen beeinflusst. Im ersten Step wurde bereits ein Ausführungsplan für alle Prozessschritte des Geschäftsprozesses bestimmt. Bei Konstanz der Datenbasis wird der Unterschied der Güte dieses Ausführungsplans zu der Güte des tatsächlich ausgeführten Ausführungsplans nur durch den Nichtdeterminismus der Heuristik und dem Mischen unterschiedlicher Ausführungspläne erzeugt. Die Güte kann sich hierdurch sowohl verbessern, als auch verschlechtern.

4.6.3 Statusinformationen

Sofern die Ausgabe von Statusinformationen unter dem Reiter „Advanced Settings" im Aktionsbereich von WorkflowOptimizer nicht explizit deaktiviert wurde, werden während der simulierten Ausführung des Geschäftsprozesses im Statusbereich Informationen zum aktuellen Stand der Ausführung ausgegeben. Die wichtigsten hierbei ausgegebenen Statusinformationen werden im Folgenden erläutert und teilweise anhand des Beispiels in Abbildung 21 illustriert.

```
================================================================ (1)
== TEST CASE  1                                               ==
================================================================
 - Time for preparing data of test case: 67.35043 ms.

[...]

************************************                             (2)
* Workflow step   1, item (  1/  1) *
************************************
 - Now to solve:
    - ExecutionPlan:
       - (  1/  1)  ->  (  2/  2)  ->  (  3/  3)

[...]

- Time for calculating initial valid solution: 28.272476 ms.     (3)
 => Initial valid solution found by opening method:
    - ExecutionPlan:
       - (  1/  1)  ->  (  2/  2)  ->  (  3/  3)
       -   3 (  2)        1 (  0)        2 (  1)
       [...]
       - Overall parameters:   0. Response time:    2750,00000
                               1. Availability:        0,99790
                               2. Throughput:       1000,00000
       - Quality: 5003.5684

[...]

=> Quality of current solution does not differ more than 1% from (4)
   theoretical maximum - the solution is optimal or nearly
   optimal.

[...]

- Time for processing this step: 33.507706 ms.                   (5)
```

Abbildung 21 - Ausschnitte aus einem Protokoll der simulierten Ausführung eines Geschäftsprozesses

Zu Beginn der Verarbeitung eines Testcases durch das `TestSetProcessor`-Objekt wird die Nummer des verarbeiteten Testcases ausgegeben, sowie die Zeit, die nötig war um den Testcase zu initialisieren (siehe Abschnitt 1 in Abbildung 21).

Nachdem ein `SequentialWorkflowEngine`-Objekt erstellt und die Ausführung des Geschäftsprozesses des Testcases gestartet wurde, werden zu jedem durchlaufenen Step Informationen über die erzeugten Teilergebnisse angezeigt. Zunächst wird angezeigt welcher Step gerade durchlaufen wird und für welchen Prozessschritt des Geschäftsprozesses in diesem Step ein Web Service zu dessen Realisation gefunden werden muss (siehe Abschnitt 2 in Abbildung 21). Im Beispiel wird die Abarbeitung des ersten Steps angezeigt („`Workflow step 1`“). Der erste Prozessschritt muss durch einen Web Service aus der Kategorie eins realisiert werden („`item (1/1)`“). In Angaben der Form „`(1/1)`“ entspricht die erste Zahl der Identifikationsnummer des Prozessschrittes und die zweite Zahl nach dem Schrägstrich der Identifikationsnummer der Kategorie, aus der ein Web Service zur Realisierung des Prozessschrittes zu entnehmen ist. Die Identifikationsnummern entsprechen jenen, die in den entsprechenden Definitionsdateien des Testcases für den Prozessschritt, bzw. die Kategorie vergeben wurden. Für welchen Teil des Geschäftsprozesses in diesem Step ein Aus-

führungsplan erstellt wird, ist unter dem Abschnitt „`Now to solve`“ angegeben. Im Beispiel muss in diesem Step ein Ausführungsplan für den Geschäftsprozess „`(1/1)->(2/2)->(3/3)`“ erzeugt werden.

Der Ausführungsplan wird durch Anwendung der Heuristik erzeugt. Hierbei werden die Schritte eins und zwei der Heuristik durch die Klasse `OpeningMethod` realisiert (vgl. Kapitel 4.3). Nach Ausführung der ersten beiden Schritte der Heuristik wird der erste zulässige Ausführungsplan für den aktuellen Geschäftsprozessteil angezeigt (siehe Abschnitt 3 in Abbildung 21). In der Beispielausgabe wird zunächst angezeigt, dass das `OpeningMethod`-Objekt 28,272476 ms benötigte um die ersten beiden Schritte der Heuristik durchzuführen. Hierbei konnte ein zulässiger Ausführungsplan ermittelt werden. Der konkrete Ausführungsplan wird durch die Angabe der konkreten Web Services dargestellt, welche für die jeweiligen Prozessschritte des Geschäftsprozesses verwendet werden. Die Angabe „`3(2)`“ unter dem Prozessschritt „`(1/1)`“ bedeutet hierbei, dass im erzeugten Ausführungsplan zur Realisierung des ersten Prozessschrittes der Web Service mit der Identifikationsnummer 3 und der Indexnummer 2 vorgesehen ist. Die Identifikationsnummer entspricht der Nummer unter welcher der Web Service in der Kategoriedefinitionsdatei definiert wurde. Die Indexnummer entspricht dem Index des Web Services in der zugehörigen Kategorie. Unter „`Overall parameters`“ werden die Gesamtparameterwerte des erzeugten Ausführungsplans angezeigt. Die Bezeichnung der Gesamtparameterwerte erfolgt gemäß der Definition der Parameter in der Parameterdefinitionsdatei. Im Beispiel verfügt der erzeugte Ausführungsplan über eine Gesamtausführungsdauer von 2750 Einheiten, eine Verfügbarkeit von 0,9979 Einheiten und einen maximalen Durchsatz von 1000 Einheiten. Die sich aus der gewichteten Summe der Gesamtparameterwerte ergebende Zielfunktionswert, bzw. Gesamtnutzenwert des Ausführungsplans wird unter „`Quality`“ aufgeführt. Im Beispiel erzeugt der Ausführungsplan einen Gesamtnutzenwert von 5003,5684 Einheiten.

Wird die Güte des Ausführungsplans nicht als hinreichend gut beurteilt, so wird zur weiteren Verbesserung des Ausführungsplans der Schritt drei der Heuristik durch die Klasse `UnsophisticatedImprovement` ausgeführt (vgl. Kapitel 4.4). Die Ausgabe der, zur Ermittlung des verbesserten Ausführungsplans, benötigten Zeit und des ermittelten Ausführungsplans, erfolgt in der bereits beschriebenen Form analog zum Ergebnis des zweiten Schrittes der Heuristik. Wird die Güte des Ausführungsplans nach dem dritten Schritt der Heuristik immernoch als nicht als hinreichend gut beurteilt, so erfolgt die Ausführung des vierten Schrittes der Heuristik durch die Klasse `SimulatedAnnealing` (vgl. Kapitel 4.5) und ebenfalls eine Ausgabe der benötigten Zeit und des ermittelten Ergebnisses in der bereits beschriebenen Form.

Wird jedoch vor der Ausführung des Schrittes drei oder vier der Heuristik festgestellt, dass die Güte des Ausführungsplans hinreichend gut ist, so wird von einer weiteren Optimierung des Ausführungsplans abgesehen. In diesem Fall erfolgt eine entsprechende Meldung analog des Abschnitts 4 in Abbildung 21.

Zum Abschluss eines Steps wird die insgesamt für diesen Step benötigte Rechenzeit ausgegeben (siehe Abschnitt 5 in Abbildung 21). Nachdem alle Steps abgearbeitet wurden, wird abschließend der tatsächlich ausgeführte Ausführungsplan, seine Gesamtparameterwerte, sein Zielfunktionswert und die insgesamt zur Ausführung des Geschäftsprozesses benötigte Zeit in einem Abschnitt „`Whole workflow`

`processed`“ ausgegeben. Die Ausgabe erfolgt in derselben Form wie sie bei der Ausgabe der Zwischenergebnisse eines Steps erfolgt.

Wird durch das `TestSetProcessor`-Objekt mehr als ein Testcase verarbeitet, so erfolgt anschließend die Ausgabe der Protokollinformationen aller weiteren verarbeiteten Testcases analog zu den soeben beschriebenen Ausgaben eines Testcases.

Es sei darauf hingewiesen, dass im Rahmen dieses Technical Reports nicht alle möglichen Protokollausgaben aller möglichen Aktionen von WorkflowOptimizer erläutert werden können. Vor allem sei auf eine Erläuterung der Protokollausgaben verzichtet, die durch die Wahl der Option „Detailed info of processing“ im Reiter „Basic Settings“ zusätzlich erzeugt werden können. Die obigen Erläuterungen ermöglichen es jedoch die Zwischenergebnisse und das Gesamtergebnis einer optimierten Ausführung eines Geschäftsprozesses durch den WorkflowOptimizer interpretieren zu können. Zusätzlich ausgegebenen Informationen erschließen sich leicht aus dem Kontext, in welchem sie ausgegeben werden.

Zur weitergehenden Analyse der Ausführungszeiten und Zielfunktionswerte müssen diese Informationen jedoch nicht mühsam aus den Protokollinformationen extrahiert werden. Sie befinden sich in übersichtlicher Form für alle verarbeiteten Testcases in der Performancedatei „`performance_<id>.csv`“[28] im Ausgabeverzeichnis. Die Datei besitzt das CSV-Format, wodurch ein problemloser Import der Daten in eine Vielzahl von Programmen wie z.B. Tabellenkalkulationen gewährleistet ist.

4.7 Datengenerierung

Wie bereits in Kapitel 4.1.2 beschrieben, verfügt WorkflowOptimizer über einen Datengenerator zur automatischen Generierung von Testcases. Der Datengenerator ist im Aktionsbereich von WorkflowOptimizer unter dem Reiter „Data Generation“ verfügbar und wird über die Schaltfläche „Generate Data“ gestartet. Der Datengenerator wird durch die Klasse `DataGeneratorStatistical` realisiert. Beim Betätigen der Schaltfläche wird eine neue Instanz der Klasse erstellt und die Funktion `load` zum Einlesen der Definitionsdateien aufgerufen. Anschließend wird die Generierung der Daten durch den Aufruf der Funktion `generateData` durchgeführt.

4.7.1 Aufbau der Definitionsdateien

Zur Datengenerierung werden zwei Definitionsdateien benötigt: Die Parameterdefinitionsdatei und die Generatordefinitionsdatei. Die Parameterdefinitionsdatei definiert die Parameter, durch welche die Web Services in den zu generierenden Testcases beschrieben werden sollen. Die Generatordefinitionsdatei definiert die Art und Weise wie zu diesen Parametern konkrete Parameterwerte zu bestimmen sind. Darüber hinaus legt sie die Anzahl der zu erzeugenden Testcases, die Länge der hierin zu erzeugenden Geschäftsprozesse und die Anzahl der pro Prozessschritt zu erzeugenden alternativen Web Services fest.

[28] Der Namensbestandteil „<id>“ repräsentiert eine Identifikationsnummer, die lediglich dazu verwendet wird einer Performancedatei einen eindeutigen Dateinamen zu geben und so das Überschreiben zuvor erzeugter Performancedateien zu verhindern.

Der Aufbau der Parameterdefinitionsdatei wurde bereits in Kapitel 4.2.1 und ihr Einlesevorgang in Kapitel 4.2.2 erläutert, weswegen an dieser Stelle nicht näher hierauf eingegangen werden soll. Nach dem Einlesen der Parameterdefinitionsdatei sind alle Parameter in Form von `ParameterDefinition`-Objekten in einem `ParameterDefinitions`-Objekt abgelegt.

```
# DataGeneratorStatistical - Generation settings
# ----------------------------------------------

# Number of test cases to generate
generator.cases.number=1

# Number of categories (and number of workflow steps)
# to generate per test case
generator.cases.categories=80

# Number of web services per category
generator.cases.webservices=40

# Definition of parameter 1 generation
generator.param.1.type=fixed
generator.param.1.value=1000
generator.param.1.variation.down=0.6
generator.param.1.variation.up=1

# Definition of parameter 2 generation
generator.param.2.type=correlated
generator.param.2.correlation.param=1
generator.param.2.correlation.fixed=0.015
generator.param.2.correlation.factor=-0.000005
generator.param.2.variation.down=0.5
generator.param.2.variation.up=0.5

# Definition of parameter 3 generation
generator.param.3.type=fixed
generator.param.3.value=0.999
generator.param.3.variation.down=0.009
generator.param.3.variation.up=0.001

# Definition of parameter 4 generation
generator.param.4.type=fixed
generator.param.4.value=100000
generator.param.4.variation.down=0.8
generator.param.4.variation.up=1
```

Abbildung 22 - Beispiel einer Generatordefinitionsdatei

Die Generatordefinitionsdatei entspricht in ihrem Aufbau einer Java-Properties-Datei und wird über die Klasse `java.util.Properties` eingelesen. Alle Zeilen der Properties-Datei die mit dem Zeichen „`#`“ beginnen, stellen Kommentare dar. Leerzeilen werden ignoriert. Alle Zeilen der Form „`<Eigenschaftsname>=<Wert>`“ definieren eine Eigenschaft (engl. Property). `<Eigenschaftsname>` steht hierbei stellvertretend für den Namen der Eigenschaft deren Wert gesetzt wird und `<Wert>` stellvertretend für den Wert auf den die Eigenschaft gesetzt wird. Über die Funktion `getProperty`, des zum Einlesen verwendeten `java.util.Properties`-Objekts, kann über den Namen einer Eigenschaft auf deren Wert zugegriffen werden.

In der Generatordefinitionsdatei werden zuerst jene Eigenschaften definiert, welche die Größe und Anzahl der zu erzeugenden Testcases festlegen. Die hierbei zur Definition verwendeten Eigenschaften können Tabelle 10 entnommen werden.

Eigenschaftsname	Beschreibung
`generator.cases.number`	Anzahl der zu erzeugenden Testcases.
`generator.cases.categories`	Anzahl der Prozessschritte des im Testcase zu erzeugenden Geschäftsprozesses.
`generator.cases.webservices`	Anzahl alternativer Web Services, die jeweils für die Ausführung eines Prozessschritts zur Verfügung stehen sollen.

Tabelle 10 - Eigenschaften zur Definition der Größe und Anzahl zu erzeugender Testcases

Im Beispiel aus Abbildung 22 wird die Erzeugung eines Testcases definiert, der einen Geschäftsprozess bestehend aus 80 Prozessschritten enthält. Für jeden Prozessschritt des Geschäftsprozesses wird eine eigene Kategorie erstellt und jeweils mit 40 alternativen Web Services zur Ausführung dieses Prozessschrittes befüllt.

Um Parameterwerte zur Beschreibung der Web Services in den Kategorien erzeugen zu können, muss im weiteren Teil der Generatordefinitionsdatei für jeden einzelnen, in der Parameterdefinitionsdatei definierten, Parameter festegelegt werden, wie die Parameterwerte erzeugt werden sollen. Zur Erzeugung eines Parameterwertes wird ein Schwankungsintervall und ein Referenzwert festgelegt, um den die erzeugten Parameterwerte gleichverteilt im definierten Schwankungsintervall schwanken. Der Referenzwert um den hierbei geschwankt wird, ist entweder ein fixer Wert oder aber ein Wert, der aus einem bereits zuvor erzeugten Parameterwert berechnet wird und dadurch mit diesem korreliert wird.

Definition des Referenzwertes `generator.param.<ID>.type=[fixed|correlated]`		
`fixed`	Definiert einen festen Referenzwert.	
	`value=r`	Setzt Referenzwert auf r.
`correlated`	Definiert einen variablen Referenzwert in Abhängigkeit des Wertes eines anderen Parameters (Korrelation).	
	`correlation.param=<id>`	Referenzwert r ergibt sich aus Wert p des Parameters mit Index <id> in der Form $r = ap + b$.
	`correlation.factor=a`	
	`correlation.fixed=b`	
Definition der Schwankung um den Referenzwert `generator.param.<ID>.variation=[up|down]`		
`variation`	`up=u`	Definiert Intervall [d;u] für prozentuale Schwankung um Referenzwert.
	`down=d`	

Tabelle 11 - Übersicht der Eigenschaften zur Generationsdefinition von Parametern

Alle Eigenschaften zur Generationsdefinitionen von Parametern haben einen Namen der mit „`generator.param.<ID>`“ beginnt. `<ID>` steht hierbei für die Identifikationsnummer des Parameters, dessen Erzeugung durch die Eigenschaft beeinflusst werden soll. Der erste in der Parameterdefinitionsdatei definierte Parameter besitzt die Identifikationsnummer eins, der zweite dort definierte Parameter die Identifikationsnummer zwei, usw. usf..

Die Schwankung um einen Wert wird für jeden Parameter durch die beiden Eigenschaften `generator.param.<ID>.variation.down` und `generator.param.<ID>.variation.up` festgelegt. Die Eigenschaftswerte bezeichnen hierbei die maximale prozentuale Schwankung um den Referenzwert nach unten, bzw. oben. Ein Parameter, der um einen festen Referenzwert schwanken soll, wird durch die Belegung der Eigenschaft `generator.param.<ID>.type` mit dem Wert `fixed` definiert. Der Referenzwert wird über die Eigenschaft `generator.param.<ID>.value` festgelegt.

Im Beispiel aus Abbildung 22 wird der Parameter mit der Identifikationsnummer eins als ein Parameter definiert, der um den fixen Wert 1000 schwankt. Die maximale Schwankung nach oben beträgt 100% und die maximale Schwankung nach unten 60%. Hieraus ergibt sich für diesen Parameter der Wertebereich [400;2000] aus dem der Parameterwert als Ausprägung einer gleichverteilten, reellwertigen Zufallsvariable bestimmt wird.

Um einen Parameter zu definieren, der mit einem anderen korreliert, wird dessen Referenzwert r in Abhängigkeit von dem Parameterwert p eines anderen Parameters ausgedrückt. Die Abhängigkeit wird in Form einer linearen Funktion der Form $r = ap + b$ festgelegt. Diese Art der Parametergeneration wird durch die Belegung der Eigenschaft `generator.param.<ID>.type` mit dem Wert `correlated` definiert. Welcher andere Parameter als Variable p der linearen Gleichung verwendet werden soll wird durch die Eigenschaft `generator.param.<ID>.correlation.param` bestimmt, die auf die Identifikationsnummer des entsprechenden Parameters gesetzt wird. Die Steigung a wird durch den Wert der Eigenschaft `generator.param.<ID>.correlation.factor` und der Achsenabschnitt b durch den Wert der Eigenschaft `generator.param.<ID>.correlation.fixed` festgelegt.

Im Beispiel aus Abbildung 22 wird der Parameter mit der Identifikationsnummer zwei als ein Parameter definiert, der um einen Wert schwankt, der sich aus dem Parameterwert des ersten Parameters ergibt, d.h. mit diesem korreliert. Nach der Erzeugung eines Parameterwertes p_1 für den ersten Parameter, wird der Referenzwert r_2 des zweiten Parameters durch die Funktion $r_2 = -0{,}000005\,p_1 + 0{,}015$ berechnet. Anschließend ergibt sich der Parameterwert p_2 durch eine Variation des Referenzwertes im Bereich von maximal 50% nach unten und maximal 50% nach oben. Die gewählte Schwankung ergibt sich als Ausprägung einer gleichverteilten, reellwertigen Zufallsvariable aus dem definierten Schwankungsintervall. Da p_1 aus dem Wertebereich [400;2000] stammt und sich der Referenzwert r_2 gemäß $r_2 = -0{,}000005\,p_1 + 0{,}015$ berechnet, besitzt r_2 einen Wertebereich von [0,005;0,013]. Durch das definierte Schwankungsintervall ergibt sich daraus für den Parameterwert des zweiten Parameters ein Wertebereich von [0,0025;0,0195].

Ein Parameter lässt sich nicht nur mit Parametern korrelieren die um einen festen Wert schwanken, sondern mit beliebigen Parametern. Dadurch ist es möglich Parameter zu definieren, die mit einem Parameter korrelieren, der selbst wiederum mit einem anderen Parameter korreliert ist. Es ist beispielsweise möglich einen Parameter in Abhängigkeit eines anderen Parameters zu definieren, der wiederum selbst in Abhängigkeit von einem anderen Parameter definiert wurde. Hierbei ist jedoch darauf zu achten, dass keine zirkulären Beziehungen definiert werden, d.h. dass ein Parameter 1

nicht von einem Parameter 2 abhängen darf, der direkt oder indirekt wieder von Parameter 1 anhängt.

Die Generatordefinitionen der einzelnen Parameter werden nach dem Einlesen in ein `java.util.Properties`-Objekt aus diesem extrahiert und die Generatordefinition jedes einzelnen Parameters in einem `ParameterGenerationDefinition`-Objekt abgelegt. Die Klasse `ParameterGenerationDefinition` kapselt sowohl die Daten der Generatordefinition eines Parameters, als auch das Verhalten, welches nötig ist um auf der Basis der Generatordefinition konkrete Parameterwerte zu erzeugen.

4.7.2 Generierung der Testcases

Nach dem Einlesen der Generatordefinitionsdatei durch die Funktion `load` des `DataGeneratorStatistical`-Objekts wird die definierte Anzahl von Testcases durch den Aufruf der Funktion `generateData` erzeugt. In den erzeugten Testcases wird für jeden Prozessschritt des Geschäftsprozesses eine eigene Kategorie angelegt und mit der festgelegten Anzahl alternativer Web Services befüllt. Zur Erzeugung der Parameterwerte, die einen Web Service beschreiben, wird hierbei die Funktionalität verwendet, die in den `ParameterGenerationDefinition`-Objekten der jeweiligen Parameter gekapselt ist.

In der Geschäftsprozessdefinitionsdatei eines erzeugten Testcases wird kein Gesamtparameterwert durch Restriktionen beschränkt. Die Gewichte, mit denen die Gesamtparameterwerte in die Zielfunktion des Optimierungsproblems eingehen, werden in Abhängigkeit vom Wertebereich der Parameter automatisch so festgelegt, dass jeder Parameter für die Optimierung eine ähnlich hohe Relevanz aufweist. Das Gewicht eines Gesamtparameters wird hierbei so bestimmt, dass aufgrund des Wertebereichs eines einzelnen Parameterwertes der Gesamtparameterwert multipliziert mit seinem Gewicht nur um 100 Einheiten schwanken kann. Da hierdurch der Zielfunktionsbeitrag eines jeden Gesamtparameters nur um 100 Einheiten schwanken kann, kann durch jeden Gesamtparameter eine gleich hohe Verbesserung oder Verschlechterung des Zielfunktionswertes herbeigeführt werden. Bezüglich der Maximierung des Zielfunktionswertes im Zuge des Optimierungsprozesses ist dadurch jeder Gesamtparameter gleich wichtig. Eine solche Gewichtsvergabe in den generierten Testcases führt daher dazu, dass bei der Optimierung eines Geschäftsprozesses alle seine Eigenschaften, ausgedrückt durch die Gesamtparameterwerte, gleichmäßig berücksichtigt und optimiert werden.

Bei der Ausgabe der Testcases in das Ausgabeverzeichnis werden alle nötigen Dateien zur Definition der einzelnen Testcases geschrieben, d.h. jeweils eine Parameterdefinitionsdatei, eine Geschäftsprozessdefinitionsdatei und eine Kategoriedefinitionsdatei. Zusätzlich werden dort noch weitere Dateien angelegt, die weitere Informationen zu den erzeugten Testcases enthalten. Die Datei „`Categories_all.csv`“ enthält alle während der Datengenerierung erzeugten Parameterwerte und kann für eine Auswertung der erzeugten Datenbasis durch externe Programme wie z.B. Tabellenkalkulationen verwendet werden. Die Textdatei „`Info_overall.txt`“ enthält eine Übersicht einiger statistischer Werte zu den erzeugten Daten aller Testcases, z.B. die theoretischen Wertebereiche der Parameter, den tatsächlichen Wertebereich der erzeugten Parameterwerte, den Mittelwert der erzeugten Parameter und weiter Kenngrößen. Für jeden einzelnen Testcase enthält die Datei „`Info_<Nummer>.txt`“ eine Übersicht dieser Kenngrößen, gruppiert nach den einzelnen Kategorien des Testcases. Der Na-

mensbestandteil „<Nummer>“ im Dateinamen entspricht hierbei der Nummer des erzeugten Testcases, zu dem die Datei weitere Informationen enthält.

5 Analyse

Zur optimierten Ausführung sequentieller Geschäftsprozesse wurde in Kapitel 3 ein entsprechendes mathematisches Modell eines Optimierungsproblems entwickelt und eine Heuristik zu dessen Lösung vorgestellt. Um das Verhalten der Heuristik näher untersuchen zu können wurde die in Kapitel 4 beschriebene prototypische Implementierung WorkflowOptimizer der Heuristik erstellt. Mittels des Prototypen wurden eine Vielzahl von Testcases gelöst und die hierbei entstandenen Performancedateien ausgewertet. Hieraus ließen sich eine Reihe von Informationen gewinnen, die z.B. Aufschluss über das Laufzeitverhalten und die erzielte Lösungsgüte unter dem Einfluss verschiedenster Parameter geben. Gegenstand dieses Kapitels ist die Betrachtung einiger ausgewählter Testreihen, die den Einfluss der wichtigsten Parameter auf die Laufzeit und die erzielte Lösungsgüte darstellen. Als wichtigste Einflussgrößen werden im Folgenden die Problemgröße und die Stärke von Restriktionen näher untersucht.

5.1 Überblick

Die *Problemgröße* des Optimierungsproblems setzt sich aus den drei Einzelfaktoren Geschäftsprozesslänge, Prozessschrittkandidatenanzahl und Parameteranzahl zusammen. Besteht ein Geschäftsprozess aus n Prozessschritten und wird jedem Prozessschritt eineindeutig (injektiv) eine Kategorie zugeordnet und besitzen alle Kategorien auf die abgebildet wird dieselbe Anzahl m von Prozessschrittkandidaten, welche durch dieselbe Anzahl von p Parameterwerten beschrieben werden, so handelt es sich um die *Normalform* des Optimierungsproblems und seine Problemgröße lässt sich in der Form <n>x<m>x<p> ausdrücken. Hierbei ist <n> ein Platzhalter für die Geschäftsprozesslänge n, <m> ein Platzhalter für die Anzahl m der Prozessschrittkandidaten in jeder Kategorie und <p> ein Platzhalter für die Anzahl p der Parameter durch deren Werte die Prozessschrittkandidaten beschrieben werden. In diesem Kapitel werden nur Testcases untersucht, die ein Optimierungsproblem in Normalform definieren. Die Analyse des Einflusses der Problemgröße auf die Heuristik erfolgt getrennt nach den Einflussgrößen Geschäftsprozesslänge n, Prozessschrittkandidatenanzahl m und Parameteranzahl p.

Ein weiterer wichtiger Einflussfaktor auf Laufzeit und Lösungsgüte der Heuristik ist die Stärke der Restriktionen durch welche die Gesamtparameterwerte einer Lösung beschränkt werden können. Je stärker eine Restriktion ist, d.h. je mehr sie den Lösungsraum einschränkt, desto schwieriger wird es eine zulässige Lösung oder gar die optimale Lösung aus der Menge der zulässigen Lösungen zu finden. Zur Definition der *Restriktionsstärke* wird im Folgenden ein prozentualer Wert verwendet, der die Höhe der Annäherung eines Schrankenwertes einer Restriktion an den bestmöglichen Gesamtparameterwert beschreibt, der durch ihn beschränkt wird. Kann ein Gesamtparameterwert aufgrund der Parameterwerte in der Datenbasis nur Werte zwischen a und b annehmen und stellt b den bestmöglichen Wert dar, so ist eine Restriktion des Gesamtparameters auf b die stärkste definierbare Restriktion. Ihr wird die Restriktionsstärke 100% zugewiesen. Eine Restriktion des Gesamtparameters auf a entspricht keiner Restriktion, da aufgrund der Definition von a als schlechtestmöglicher Gesamtparameterwert jeder Gesamtparameterwert die Restriktion erfüllt. Einer Restriktion auf a wird daher die Restriktionsstärke 0% zugewiesen. Einer Restriktionsstärke von 50% entspricht eine Restriktion auf den Mittelwert aller Gesamtparameterwerte zwischen a

und b, da im Mittel die Hälfte aller erzeugbarer Gesamtparameterwerte die Restriktion erfüllt.

Um den Einfluss eines, die Lösungsgüte oder Laufzeit der Heuristik betreffenden Parameters näher untersuchen zu können, wurden Testreihen von Testcases gebildet, die ceteris paribus jeweils einen Parameter variieren. Die Testreihen wurde mithilfe des Prototypen gelöst und die hierbei in der Performancedatei protokollierten Messwerte ausgewertet. Um die Laufzeit und die Lösungsgüte der Heuristik einem exakten Lösungsverfahren gegenüberstellen zu können, wurden alle Testreihen auch in Form eines unrelaxierten Optimierungsmodells durch den Solver gelöst. Zur Approximation des durchschnittlichen Verhaltens wurden alle Testreihen in mehreren Testläufen mehrfach gelöst und Mittelwerte aus den Messwerten gebildet.

Bei den folgenden Betrachtungen der einzelnen Testreihen wird eine Analyse gemäß eines gleichbleibenden Schemas durchgeführt. Zunächst wird die Laufzeit der Heuristik und des Solvers gegenübergestellt und das Verhältnis aus benötigter Laufzeit und erreichter Lösungsgüte verglichen. Anschließend folgt eine nähere Untersuchung der Heuristik. Hierbei wird betrachtet welchen Anteil an der Gesamtlaufzeit der Heuristik durch die einzelnen Abschnitte der Heuristik (OM[29], UIM[30], SA[31]) entstanden ist und welcher Abschnitt welchen Anteil zur erreichten Lösungsgüte beigetragen hat. Zur Untersuchung der Laufzeit der Abschnitte UIM und SA wird diese noch in die Anzahl der durchgeführten Iterationen und die durchschnittliche Iterationsdauer zerlegt. Die Betrachtung einer Testreihe schließt mit einer kurzen Zusammenfassung der Beobachtungen.

Ergänzend sei an dieser Stelle noch erwähnt, dass als Referenzrechner zur Ermittlung der Messwerte ein Rechner mit einem Prozessor des Typs „Intel Pentium 4 530J“ (3 GHz), sowie 1 GB Arbeitsspeicher unter dem Betriebssystem „Microsoft Windows XP Professional“ verwendet wurde.

5.2 Einfluss der Problemgröße

Um den Einfluss einer der, die Problemgröße eines Optimierungsproblems in Normalform bestimmenden Parameter Geschäftsprozesslänge n, Prozessschrittkandidatenanzahl m und Parameteranzahl p auf das Laufzeitverhalten und die Lösungsgüte der Heuristik untersuchen zu können, wurden Testreihen von Testcases in Normalform gebildet, die ceteris paribus jeweils einen dieser Problemgrößenparameter variieren.

5.2.1 Einfluss der Geschäftsprozesslänge

Die beiden Testreihen zur Untersuchung des Einflusses der Geschäftsprozesslänge und der Prozessschrittkandidatenanzahl basieren beide auf derselben Ausgangsbasis eines Testcases der Größe 80x80x04, der mithilfe des Datengenerators des Prototypen erstellt wurde. Zur Generierung der einzelnen Testcases der Testreihen wurde der Basistestcase entsprechend modifiziert. Die Gesamtparameterwerte des Geschäftspro-

[29] OM wird abkürzend für die, durch die Klasse `OpeningMethod` gekapselten, Schritte eins und zwei der Heuristik verwendet.
[30] UIM wird abkürzend für den, durch die Klasse `UnsophisticatedImprovementMethod` gekapselten, Schritt drei der Heuristik verwendet.
[31] SA wird abkürzend für den, durch die Klasse `SimulatedAnnealing` gekapselten, Schritt vier der Heuristik verwendet.

zesses wurden keinen Restriktionen unterworfen. Die Gewichte in der Zielfunktion wurden in jedem Testcase so gewählt, dass der Zielfunktionsbeitrag, der durch jeden Gesamtparameter gestiftet werden kann, um 100 Einheiten schwanken kann. Bezüglich der Erstellung eines optimierten Ausführungsplans werden daher alle Parameter gleichmäßig berücksichtigt, d.h. sie sind bezüglich der Optimierung von gleicher Relevanz. Als Parameter zur Beschreibung der einzelnen Prozessschrittkandidaten wurden zwei miteinander korrelierte additive Parameter (z.B. Antwortzeit und Preis), ein unkorrelierter multiplikativer Parameter (z.B. Verfügbarkeitswahrscheinlichkeit), sowie ein unkorrelierter Minimaloperatorparameter (z.B. maximaler Durchsatz) verwendet.

Zur Untersuchung des Einflusses der Geschäftsprozesslänge n wurden Testcases mit Problemgrößen von 05x40x04 bis 80x40x04 gebildet und Lösungen hierzu mittels des Solvers und der Heuristik ermittelt. Die sich hierbei ergebenden Laufzeiten und Zielfunktionswerte können in übersichtlicher Form Tabelle 12 entnommen werden.

Problem-größe	**Solver**		**Heuristik**		$\frac{t_h}{t_s}$	$\frac{F(\vec{x}_h)}{F(\vec{x}_s)}$
	t_s [ms]	$F(\vec{x}_s)$	**t_h [ms]**	$F(\vec{x}_h)$		
05x40x04	7,9329	2.143,4390	7,4680	2.143,4390	94,14%	100,00%
10x40x04	365,7752	1.063,2938	46,3051	1.056,9507	12,66%	99,40%
15x40x04	3.213,5676	720,6985	74,7306	715,5157	2,33%	99,28%
20x40x04	43.058,2734	537,9250	138,7717	532,0076	0,32%	98,90%
40x40x04	?	?	458,1601	260,3245	?	?
60x40x04	?	?	1.275,0992	179,9881	?	?
80x40x04	?	?	2.488,7103	133,9539	?	?

Tabelle 12 - Vergleich von Solver und Heuristik bei variierender Geschäftsprozesslänge

In obiger Tabelle wird die vom Solver benötigte Zeit zur Ermittlung des optimalen Ausführungsplans mit t_s bezeichnet und der hierbei ermittelte Zielfunktionswert (Gesamtnutzen des Ausführungsplans) mit $F(\vec{x}_s)$. Die von der Heuristik benötigte Zeit zur Ermittlung eines optimierten Ausführungsplans wird mit t_h bezeichnet und der zugehörige Zielfunktionswert mit $F(\vec{x}_h)$. In der Spalte $\frac{t_h}{t_s}$ wird die Lösungszeit der Heuristik in Prozent der benötigten Lösungszeit des Solvers ausgedrückt. Die Spalte $\frac{F(\vec{x}_h)}{F(\vec{x}_s)}$ drückt den Zielfunktionswert der Lösung der Heuristik in Prozent des Zielfunktionswertes der optimalen Lösung des Solvers aus. Einträge mit „?“ bedeuten, dass der Testlauf unter Verwendung des Solvers nach mehreren Stunden Laufzeit ohne Lösungsfindung abgebrochen wurde und daher in diesen Fällen keine Messwerte vorliegen.

Wie dem Vergleich der Laufzeiten in obiger Tabelle leicht entnommen werden kann, steigt der Laufzeitvorteil der Heuristik gegenüber dem Solver mit zunehmender Geschäftsprozesslänge schnell stark an. Bei einer Steigerung der Geschäftsprozesslänge von 5 auf 20 Prozessschritte (Steigerung um Faktor 4) steigt die benötigte Rechenzeit des Solvers von ca. 8 ms auf ca. 43.000 ms an (Steigerung um Faktor 5375). Die benötigte Rechenzeit der Heuristik hingegen steigt lediglich von ca. 7,5 ms auf ca. 140 ms an (Steigerung um ca. Faktor 18,7). Bei einer Problemgröße von 20x40x04 benö-

tigt die Heuristik nur 0,32% der Laufzeit des Solvers, erzeugt jedoch eine Lösung deren Zielfunktionswert 98,9% des Zielfunktionswertes der optimalen Lösung erreicht. Mit zunehmender Geschäftsprozesslänge nimmt jedoch die Abweichung vom Optimum leicht zu.

Die Entwicklung der Laufzeit von Solver und Heuristik bei steigender Geschäftsprozesslänge ist in Abbildung 23 graphisch dargestellt. In diesem und in den folgenden Diagrammen wurden die diskreten Messwerte durch interpolierten Linien-, bzw. Kurvenverläufe miteinander verbunden. Diese Art der Darstellung dient lediglich der besseren Visualisierung der Messwerte.

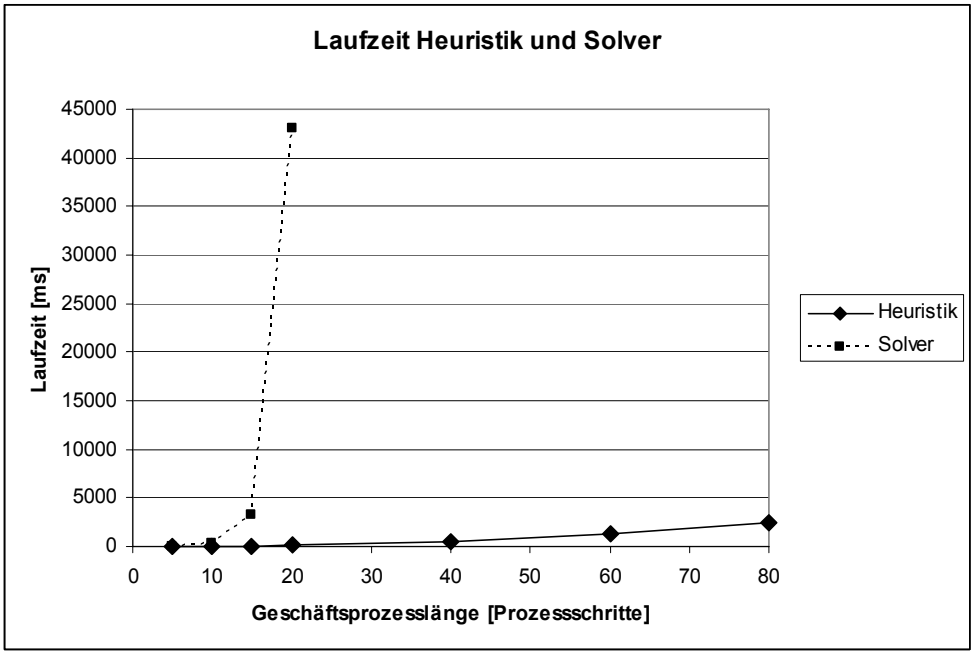

Abbildung 23 - Vergleich der Laufzeit von Solver und Heuristik

Wie schnell die Laufzeit des Solvers gegenüber der Laufzeit der Heuristik selbst bei geringen Geschäftsprozesslängen ansteigt, wird in Abbildung 23 deutlich. Die Laufzeit des Solvers ähnelt hierbei auffällig stark einem exponentiellen Verlauf, wohingegen die Laufzeit der Heuristik einen noch polynomiellen Verlauf zu haben scheint. Um das Laufzeitverhalten besser beurteilen zu können, wird die Laufzeit von Solver und Heuristik in Abbildung 24 mit einer logarithmisch skalierten Ordinate (Laufzeit-Achse) dargestellt.

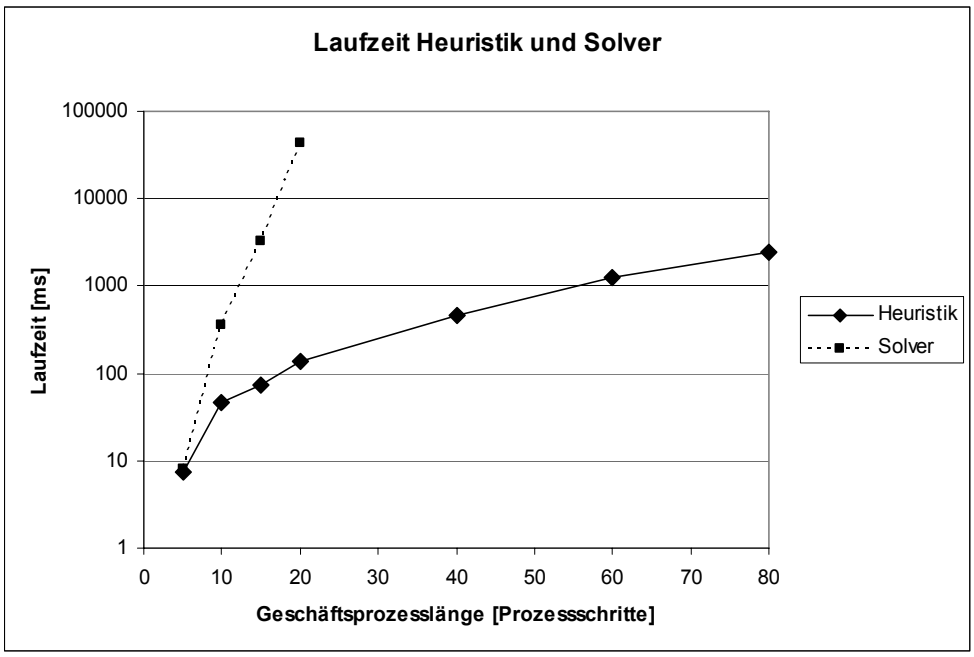

Abbildung 24 - Vergleich der Laufzeit von Solver und Heuristik bei logarithmischer Skalierung

In Diagrammen mit logarithmisch skalierter Ordinate stellen sich Funktionen, die exponentielles Wachstum besitzen als Gerade dar. Funktionen mit polynomiellem, z.B. quadratischem, Wachstum zeigen hingegen einen Verlauf mit monoton fallender Steigung. Der Verlauf der Laufzeit der Heuristik besitzt eindeutig einen Verlauf mit monoton fallender Steigung, weswegen die Heuristik in Abhängigkeit von der Geschäftsprozesslänge kein exponentielles Laufzeitverhalten besitzen kann, sondern polynomiell beschränkt sein muss. Aufgrund der hohen Rechenzeit liegen bezüglich des Solvers zu wenige Messwerte vor um deren weiteren Verlauf beurteilen zu können. Im gemessenen Bereich ist jedoch eine Entwicklung der Laufzeit erkennbar, die in guter Näherung als exponentiell beschrieben werden kann.

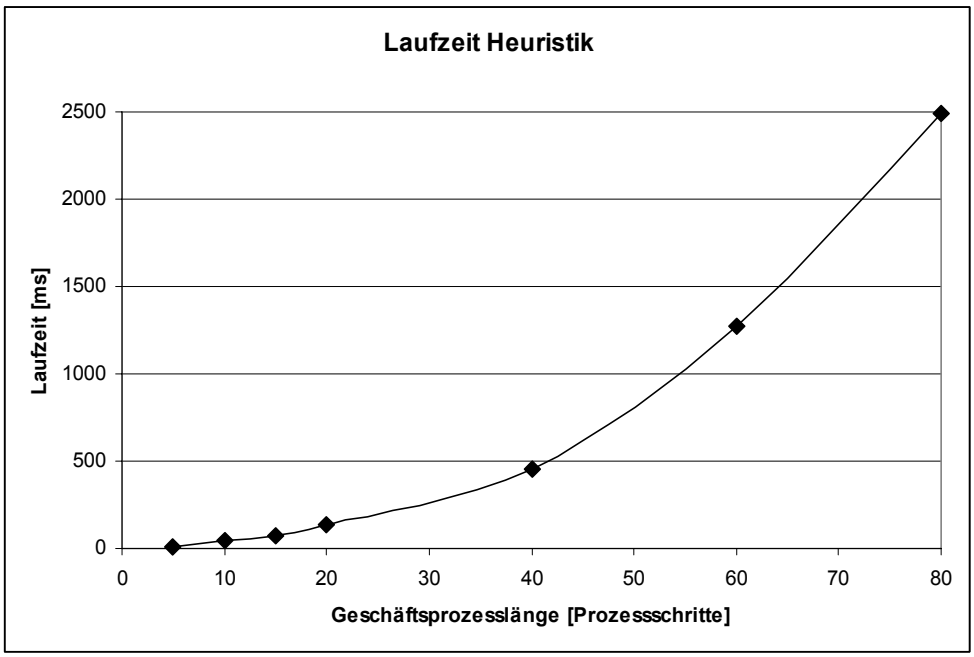

Abbildung 25 - Laufzeit der Heuristik

Die alleinige Darstellung der gemessenen Laufzeit der Heuristik in Abbildung 25 lässt den Verlauf dieser besser erkennen. Von besonderem Interesse ist hierbei, wie sich die Gesamtlaufzeit der Heuristik aus den einzelnen Laufzeiten der OM, der UIM und des SA zusammensetzt. Eine Darstellung der einzelnen Laufzeiten kann Abbildung 26 entnommen werden.

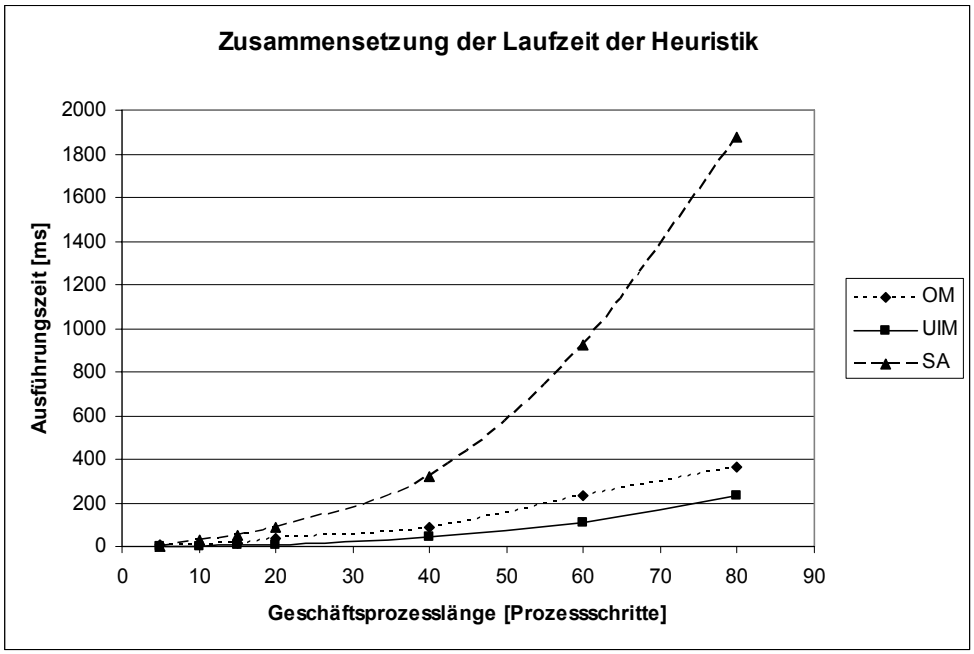

Abbildung 26 - Zusammensetzung der Laufzeit der Heuristik

Auffällig ist hierbei, dass die Laufzeit des SA mit steigender Geschäftsprozesslänge zunehmend schneller ansteigt, als die sich annähernd gleichförmig entwickelnden Laufzeiten der OM und der UIM. Mit steigender Geschäftsprozesslänge dominiert die Laufzeit des SA daher zunehmend die Gesamtlaufzeit der Heuristik. Durch die Darstellung des prozentualen Laufzeitanteils der Abschnitte OM, UIM und SA an der Gesamtlaufzeit der Heuristik in Abbildung 27, lässt sich die zunehmende Dominanz von SA an der Gesamtlaufzeit noch besser visualisieren.

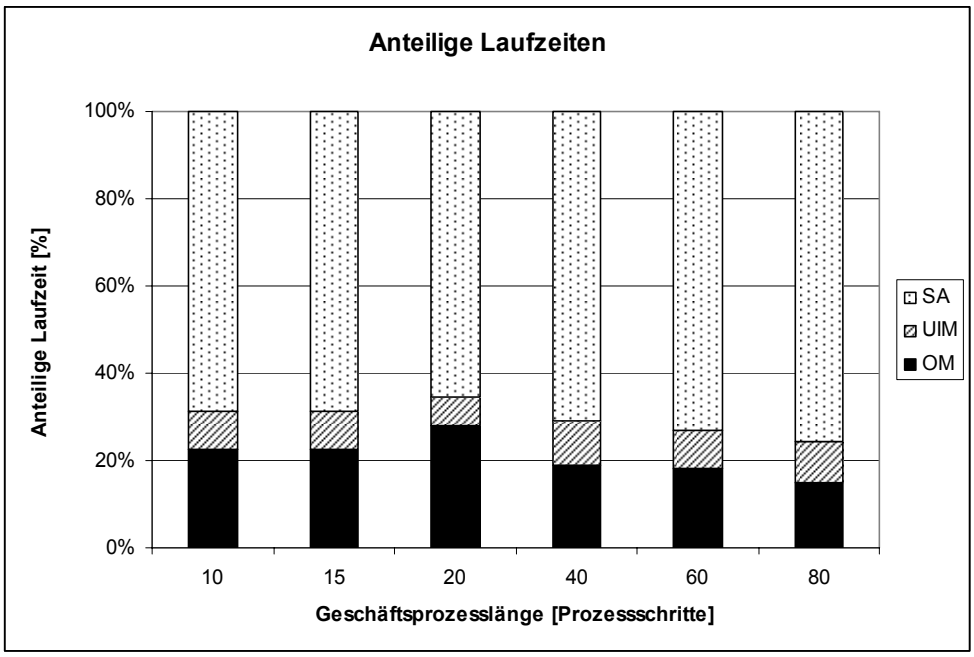

Abbildung 27 - Anteil von OM, UIM und SA an der Gesamtlaufzeit

Bei einer Geschäftsprozesslänge von 80 Prozessschritten beansprucht SA ca. 75% der Gesamtlaufzeit der Heuristik für sich. Eine hohe Laufzeit ist vor allem in jenen Abschnitten der Heuristik gerechtfertigt, die einen hohen Anteil am endgültig durch die Heuristik erreichten Zielfunktionswert liefern.

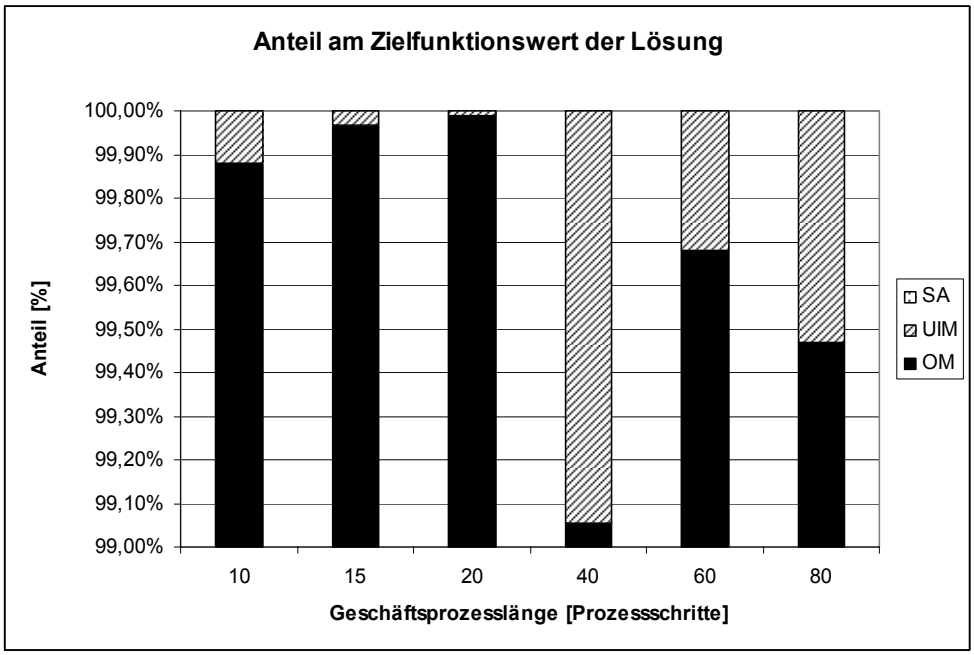

Abbildung 28 - Anteil von OM, UIM und SA am Zielfunktionswert der Lösung

Betrachtet man jedoch in Abbildung 28 den Anteil am Zielfunktionswert der Lösung, den OM, UIM und SA liefern, so ist festzustellen, dass bereits OM stets einen Anteil von über 99% am letztlich durch die Heuristik erreichten Zielfunktionswert besitzt. UIM kann den Zielfunktionswert noch geringfügig verbessern, jedoch gelingt SA in

keinem Fall der Testreihe eine weitere Verbesserung. Dies ist vor allem deshalb bedauerlich, da SA den größten Teil der Gesamtlaufzeit beansprucht.

Es ist jedoch falsch hieraus direkt den Schluss zu ziehen, dass SA ineffektiv sei. Da bereits OM eine Lösung liefert die sehr nahe am erreichbaren Optimum ist, fällt es jeder in der Heuristik nachgelagerten Optimierungsmethode zunehmend schwerer noch eine weitere Verbesserung herbeizuführen, auch wenn hierzu anteilig eine hohe Rechenzeit investiert wird. Die ausbleibende Verbesserung durch SA ist daher eher auf die sehr gute Lösung der vorausgehenden Schritte der Heuristik zurückzuführen und nicht etwa auf ein Versagen des SA.

UIM und SA führen solange Iterationen aus, bis ein Abbruchkriterium erreicht wird. Die Dauer einer Iteration von UIM und SA sind hierbei fast identisch und kann in Abhängigkeit von der Geschäftsprozesslänge Abbildung 29 entnommen werden.

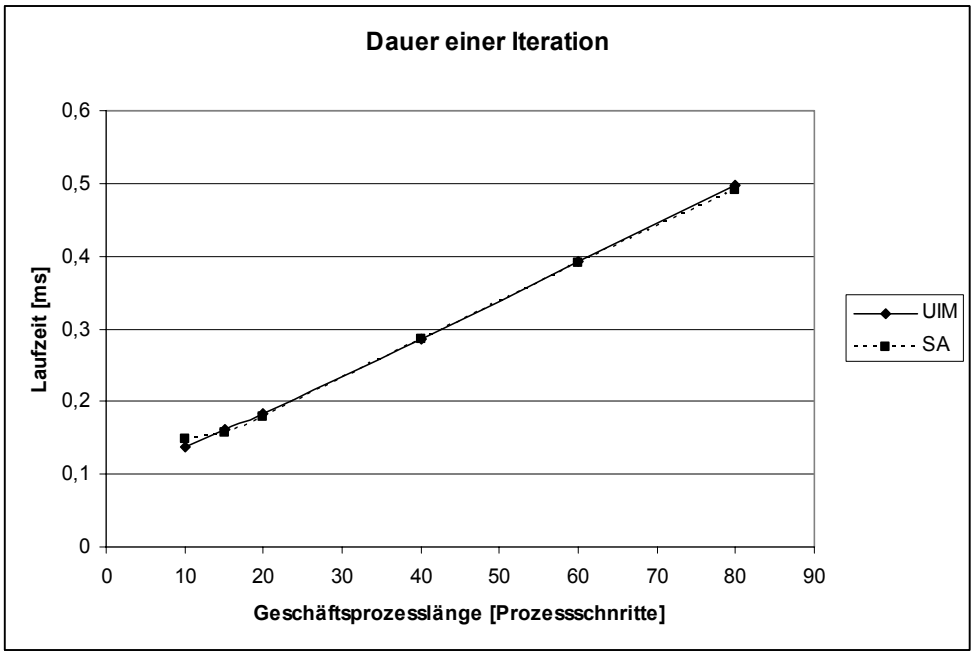

Abbildung 29 - Dauer einer Iteration von UIM und SA

Die Dauer einer Iteration von UIM und SA steigt in sehr guter Näherung linear mit der Länge des Geschäftsprozesses an. Da die Dauer einer Iteration von UIM und SA annähernd gleich ist, ergeben sich die beobachteten unterschiedlichen Anteile an der Gesamtlaufzeit aus der unterschiedlichen Anzahl ausgeführter Iterationen, welche Abbildung 30 entnommen werden kann.

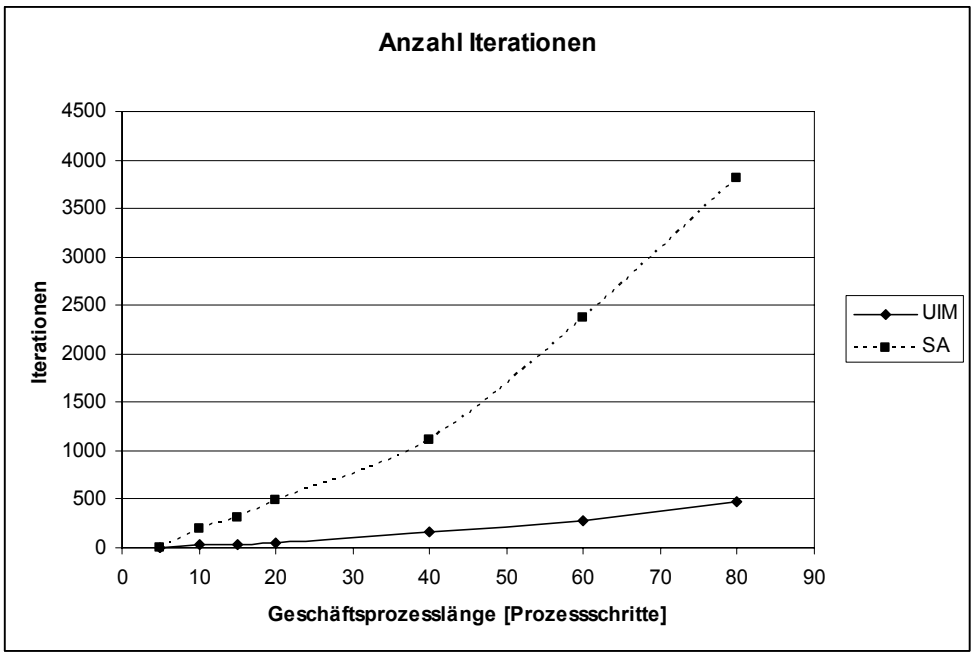

Abbildung 30 - Anzahl der Iterationen von UIM und SA

Zusammenfassend kann festgehalten werden, dass es der Heuristik mit zunehmender Geschäftsprozesslänge gelingt einen enormen Laufzeitvorteil gegenüber der exakten Lösung durch den Solver zu erreichen, wobei sich die Laufzeit als polynomiell beschränkt darstellt. Die Lösungsgüte ist hierbei jedoch nur geringfügig schlechter und sinkt nur langsam mit zunehmender Geschäftsprozesslänge ab. Die Heuristik liefert bereits nach der Durchführung der ersten beiden Schritte (OM) eine sehr gute Lösung nahe am Optimum, die auch mit einem zunehmenden Aufwand an Rechenzeit in den nachfolgenden Schritten der Heuristik nur noch geringfügig verbessert werden kann. Obwohl sich die Dauer einer Iteration von UIM und SA linear mit der Geschäftsprozesslänge entwickelt, wird hierbei durch die zunehmende Anzahl der durchgeführten Iterationen in diesen Phasen ein hoher Anteil der Gesamtrechenzeit investiert, ohne eine nennenswerte Verbesserung des Zielfunktionswertes erzielen zu können.

5.2.2 Einfluss der Anzahl von Prozessschrittkandidaten

Zur Untersuchung des Einflusses der Prozessschrittkandidatenanzahl m wurden Testcases mit Problemgrößen von 20x10x04 bis 20x80x04 erstellt. Die sich bei der Lösung der Testcases durch den Solver und die Heuristik ergebenden Laufzeiten und Zielfunktionswerte könne Tabelle 13 entnommen werden, welche analog der Tabelle 12 aus Kapitel 5.2.1 aufgebaut ist.

Problem-größe	**Solver**		**Heuristik**		$\frac{t_h}{t_s}$	$\frac{F(\vec{x}_h)}{F(\vec{x}_s)}$
	t_s [ms]	$F(\vec{x}_s)$	**t_h [ms]**	$F(\vec{x}_h)$		
20x10x04	10.943,8584	605,1078	74,0416	602,2472	0,68%	99,53%
20x20x04	15.496,6680	553,6658	84,8892	550,9455	0,55%	99,51%
20x40x04	43.002,4023	537,9250	136,1075	531,9503	0,33%	98,90%
20x60x04	91.141,9609	533,3203	193,2284	526,6455	0,21%	98,75%
20x80x04	223.962,0938	530,1653	282,4100	522,5867	0,13%	98,57%

Tabelle 13 - Vergleich von Solver und Heuristik bei variierender Prozessschrittkandidatenanzahl

Wie bei der Erhöhung der Geschäftsprozesslänge, so steigt auch bei der Erhöhung der Prozessschrittkandidatenanzahl die benötigte Rechenzeit des Solvers und der Heuristik an. Der Anstieg der benötigten Rechenzeit fällt jedoch im Gegensatz zur Erhöhung der Geschäftsprozesslänge wesentlich geringer aus. Der Unterschied in den benötigten Laufzeiten zwischen Solver und Heuristik ist deutlich. Bei einer Steigerung der Anzahl der Prozessschrittkandidaten von 10 auf 80 (Steigerung um Faktor 8) steigt die benötigte Rechenzeit des Solvers von ca. 11.000 ms auf ca. 224.000 ms an (Steigerung ca. um Faktor 20). Die benötigte Rechenzeit der Heuristik hingegen steigt lediglich von ca. 74 ms auf ca. 282 ms an (Steigerung ca. um Faktor 4). Zur Lösung der Problemgröße 20x80x04 benötigt die Heuristik nur 0,13% der Rechenzeit des Solvers, erzeugt jedoch eine Lösung die einen Zielfunktionswert in Höhe von 98,57% der optimalen Lösung besitzt. Der Laufzeitvorteil der Heuristik steigt mit zunehmender Anzahl von Prozessschrittkandidaten an, jedoch sinkt die Güte der ermittelten Lösung geringfügig ab.

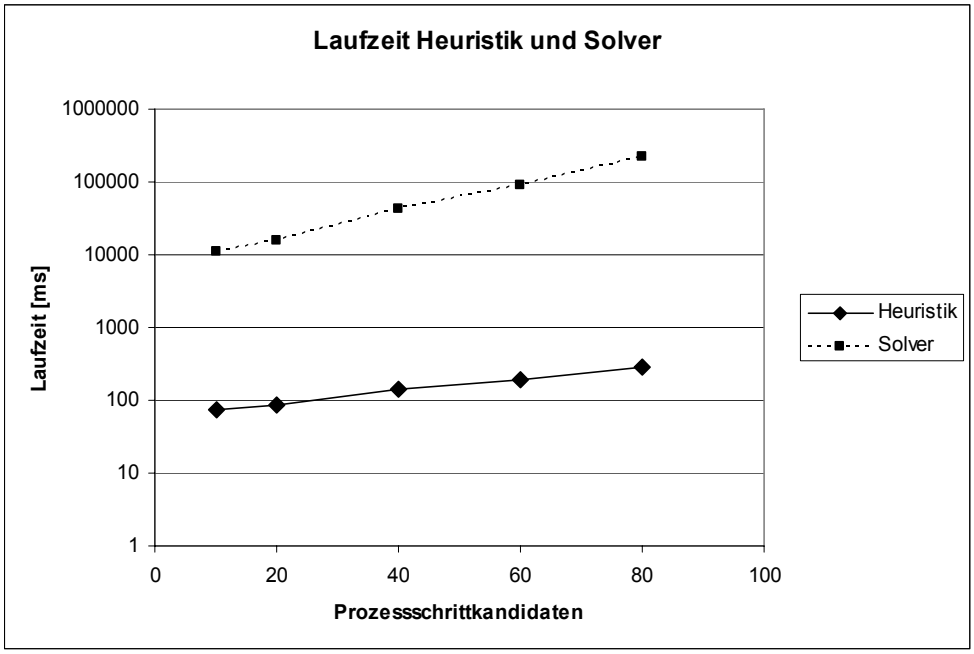

Abbildung 31 - Laufzeiten von Solver und Heuristik in Abhängigkeit von der Anzahl der Prozessschrittkandidaten

Abbildung 31 stellt die Laufzeit des Solvers und die der Heuristik mit einer logarithmisch skalierten Ordinate in Abhängigkeit von der Anzahl der Prozessschrittkandidaten dar. Die deutlichen Laufzeitunterschiede, sowie das stärkere Wachstum der Laufzeit des Solvers sind erkennbar. Untersucht man die Entwicklung der Laufzeit der Heuristik genauer, so lässt sich erkennen, dass die Laufzeit der Heuristik mit steigender Prozessschrittkandidatenanzahl zunehmend von der Laufzeit des Abschnittes OM bestimmt wird (s.u.) und dass OM eine eindeutig polynomiell beschränkte Laufzeit besitzt. Insgesamt nähert sich so die Laufzeit der Heuristik zunehmend einem eindeutig polynomiell beschränkten Laufzeitverhalten an. Bezüglich der Laufzeit des Solvers kann hingegen lediglich festgestellt werden, dass sich diese im gemessenen Bereich annähernd exponentiell entwickelt.

In Abhängigkeit von der Anzahl der Prozessschrittkandidaten verteilt sich die Laufzeit der Heuristik wie in Abbildung 32 dargestellt auf die Abschnitte OM, UIM und SA der Heuristik.

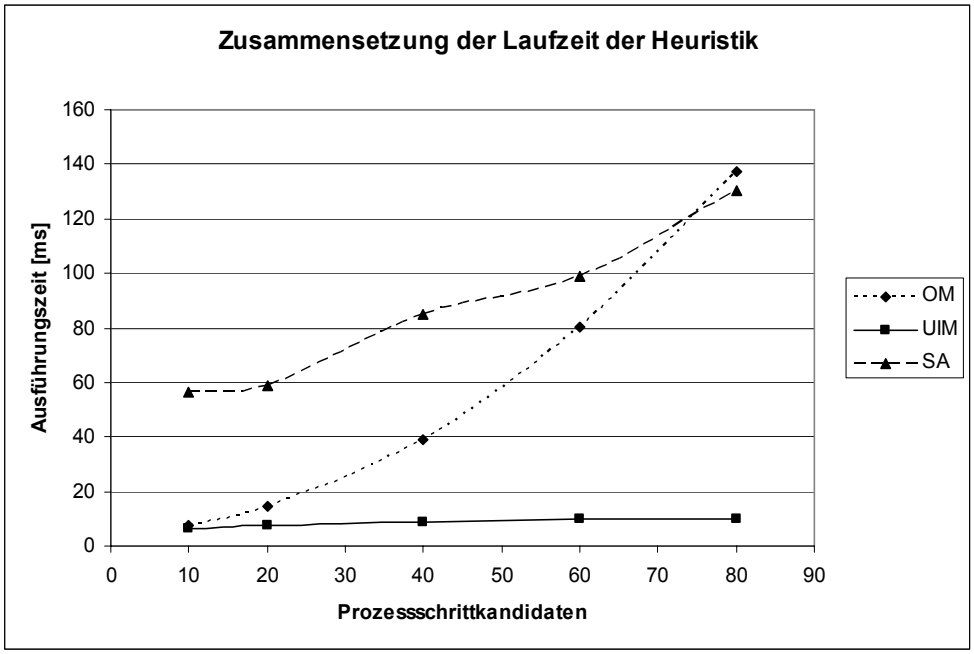

Abbildung 32 - Zusammensetzung der Laufzeit der Heuristik

Mit zunehmender Geschäftsprozesslänge stieg in der vorhergehenden Testreihe die benötigte Laufzeit von SA schneller als die Laufzeiten von OM und UIM und bildete den größten Anteil an der Gesamtlaufzeit der Heuristik. Mit zunehmender Anzahl von Prozessschrittkandidaten ist es jedoch OM, welche den schnellsten Anstieg der Laufzeit besitzt und die Laufzeit von SA schließlich übersteigt. Entgegen den Steigerungen der Laufzeit von OM und SA ist die Laufzeit der UIM fast konstant und nimmt daher mit steigender Anzahl von Prozessschrittkandidaten einen immer geringeren Anteil an der Gesamtlaufzeit der Heuristik ein. Abbildung 33 verdeutlicht nochmals die anteilige Entwicklung der Laufzeiten von OM, UIM und SA an der Gesamtlaufzeit der Heuristik.

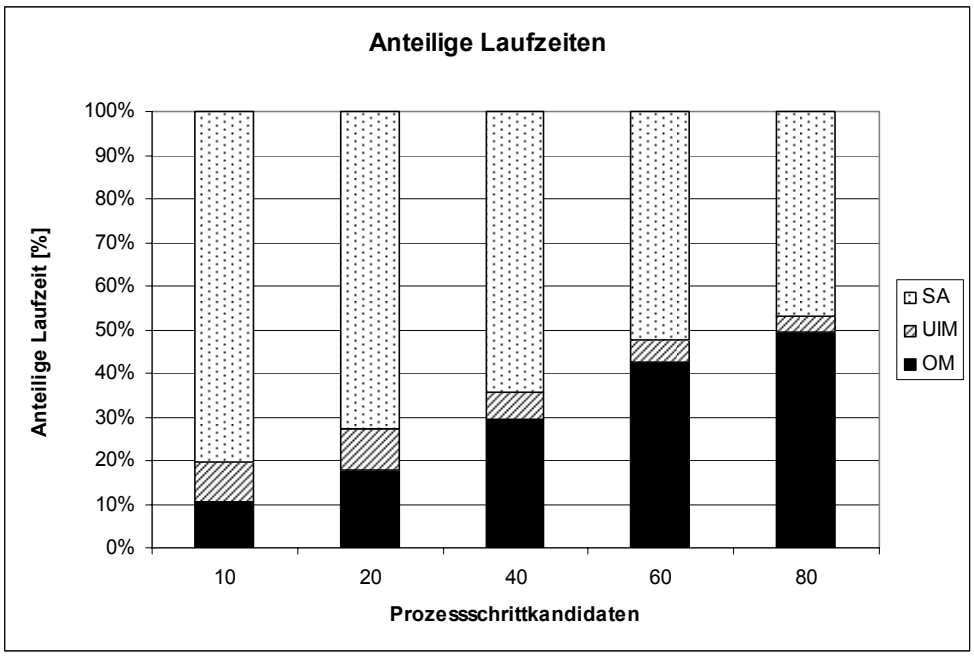

Abbildung 33 - Anteil von OM, UIM und SA an der Gesamtlaufzeit

In den in dieser Testreihe gemessenen Fällen liefert die OM bereits eine erste zulässige Lösung die einen Zielfunktionswert in der Höhe von über 99,8% der Lösung der Gesamtheuristik besitzt. Der UIM gelingt nur noch eine marginale Verbesserung dieser Lösung, wohingegen das SA in keinem Fall zu einer weiteren Verbesserung einer Lösung führen konnte. Der Anteil am Zielfunktionswert der Lösung der Gesamtheuristik, den OM, UIM und SA liefern, kann Abbildung 34 entnommen werden.

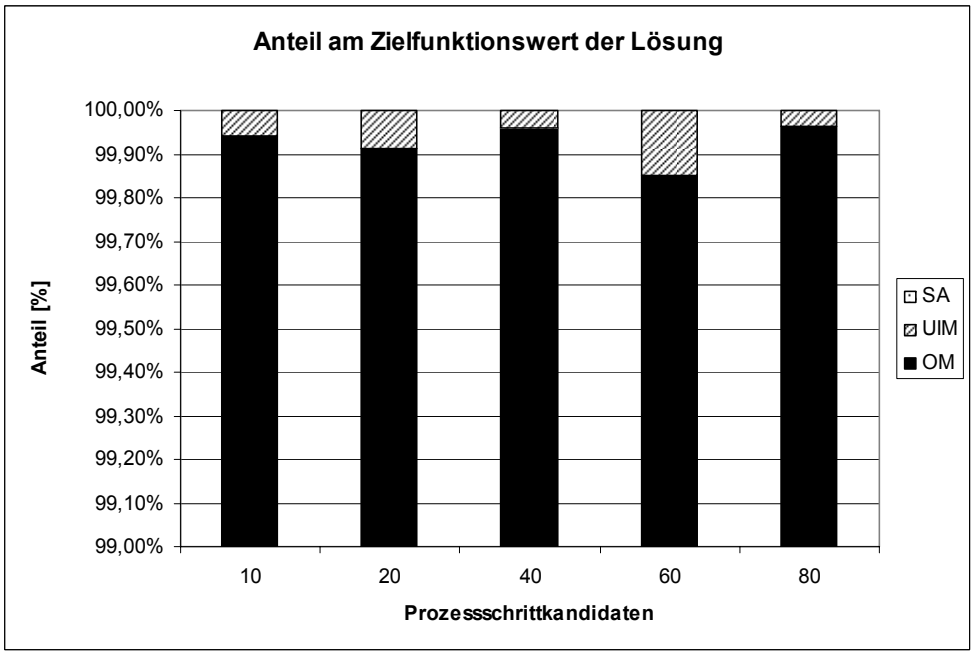

Abbildung 34 - Anteil von OM, UIM und SA am Zielfunktionswert der Lösung

Entgegen der vorherigen Testreihe wird mit zunehmender Anzahl von Prozessschrittkandidaten die Laufzeit der Gesamtheuristik zunehmend für den Teil OM verwendet, der maßgeblich zur Güte der Gesamtlösung der Heuristik beiträgt. In der vorherigen

Testreihe wurde mit zunehmender Geschäftsprozesslänge die Laufzeit hingegen zunehmend für die Teile der Heuristik aufgewendet, die zu keiner oder nur zu einer geringfügigen Verbesserung der Lösungsgüte führten. Die Zunahme der Anzahl der Prozessschrittkandidaten führt jedoch dazu, dass die Laufzeit der Gesamtheuristik zunehmend für den Teil verwendet wird, der maßgeblich zur Güte der Gesamtlösung der Heuristik beiträgt, d.h. dass die Rechenzeit zunehmend effizienter genutzt wird.

Die Laufzeiten einer Iteration von UIM und SA sind auch wie in der vorherigen Testreihe nahezu identisch. Sie steigen linear zur Anzahl der Prozessschrittkandidaten an, jedoch fällt der Anstieg geringer aus als bei einer Zunahme der Geschäftsprozesslänge. Eine Iteration in einem betrachteten Testcase der Größe 20x80x04 nimmt beispielsweise weniger Rechenzeit in Anspruch, als bei einem Testcase der Größe 40x40x04, wobei jedoch beide Problemgrößen 1.600 Web Services umfassen. Abbildung 35 zeigt die Dauer einer Iteration von UIM und SA in Abhängigkeit der Anzahl von Prozessschrittkandidaten.

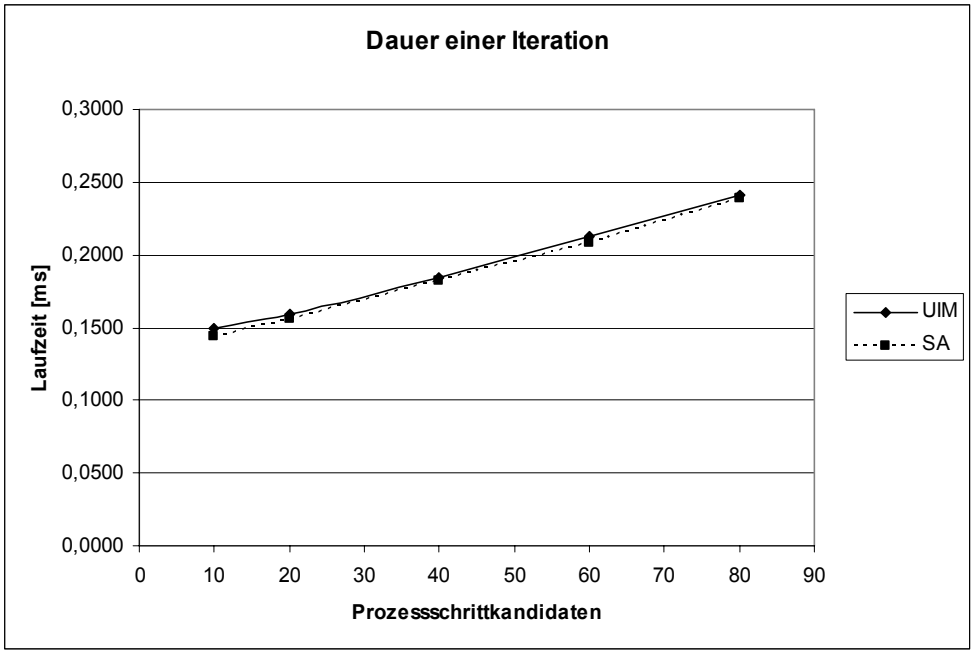

Abbildung 35 - Dauer einer Iteration von UIM und SA

Im Gegensatz zur vorherigen Testreihe mit variabler Geschäftsprozesslänge steigt die Anzahl der Iterationen von SA mit zunehmender Anzahl von Prozessschrittkandidaten langsamer und unregelmäßiger an. Die maximale Anzahl aufeinander folgender Iterationen ohne Verbesserung des Zielfunktionswertes sind sowohl bei SA als auch UIM direkt von der konstanten Geschäftsprozesslänge abhängig. Im Falle von SA ergeben sich jedoch mit zunehmender Anzahl von Prozessschrittkandidaten auch zunehmend mehr alternative Nachbarlösungen die zu einer akzeptierten Verschlechterung und anschließender Verbesserung führen und daher zufallsbedingt für eine höhere Schwankung bei der Anzahl der Iterationen von SA sorgen. Da die Güte, der von OM erzeugten ersten zulässigen Lösung, sehr hoch ist und UIM nur die, in sehr geringer Anzahl vorhandenen, besseren Nachbarlösungen akzeptiert, ist die Anzahl der Iterationen der UIM nahezu konstant. Die Anzahl der ausgeführten Iterationen von UIM und SA können Abbildung 36 entnommen werden.

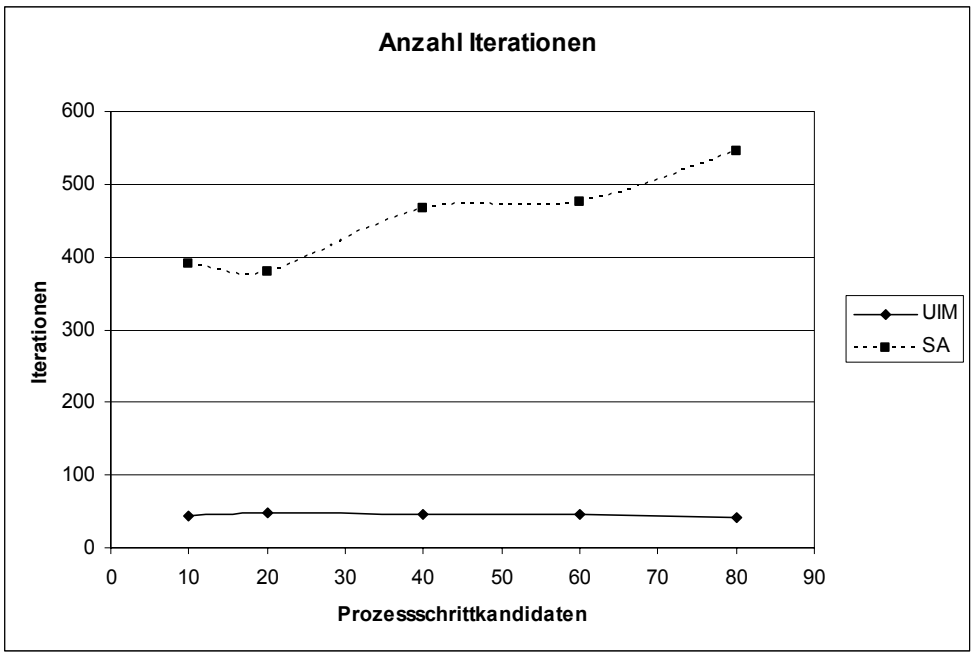

Abbildung 36 - Anzahl der Iterationen von UIM und SA

Zusammenfassend lässt sich feststellen, dass es mit steigender Anzahl von Prozessschrittkandidaten der Heuristik gelingt ihren Laufzeitvorteil gegenüber der exakten Lösung des Problems durch den Solver noch zu verbessern, wobei ihre Laufzeit zunehmend vom eindeutig polynomiell beschränkten Abschnitt OM bestimmt wird. Die hierbei erreichte Lösungsgüte nimmt zwar mit zunehmender Anzahl von Prozessschrittkandidaten ab, liegt aber immernoch sehr nahe an der optimalen Lösung. Die Lösungsgüte der ersten zulässigen Lösung, die durch die OM erzeugt wird, ist so gut, dass sie durch die UIM nur noch marginal und durch SA gar nicht mehr verbessert werden kann. Positiv ist hierbei, dass der Anteil von UIM und SA an der Gesamtlaufzeit der Heuristik mit Zunahme der Anzahl der Prozessschrittkandidaten rückläufig ist, d.h. dass die Gesamtlaufzeit zunehmend für den Teil OM der Heuristik verwendet wird, der den höchsten Anteil zur Lösungsgüte beiträgt. Die Dauer einer Iteration von UIM und SA steigt in Abhängigkeit von der Anzahl der Prozessschrittkandidaten ebenfalls linear, aber weniger steil an, als in Abhängigkeit von der Geschäftsprozesslänge. Die Anzahl der Iterationen von UIM ist nahezu konstant und die Anzahl der Iterationen von SA steigt wesentlich langsamer an, als in Abhängigkeit von der Geschäftsprozesslänge.

5.2.3 Einfluss der Anzahl von Parametern

Ein weiterer, die Problemgröße definierender Faktor ist die Anzahl p von Parametern, mit deren Werten die Eigenschaften eines Web Services beschrieben werden. Ausgehend von einer Problemgröße von 11x25x03, wurden durch das sukzessive Hinzufügen von jeweils drei weiteren Parametern Testcases bis zu einer Problemgröße von 11x25x15 gebildet. Bei jeder Vergrößerung wurde jeweils ein additiver Parameter, ein multiplikativer Parameter und ein Minimaloperator-Prarameter hinzugefügt um den Einfluss aller drei Parametertypen zu erfassen. Keine diese Parameter sind miteinander korreliert. Die Festlegung der Gewichte in der Zielfunktion erfolgte derart, dass der durch jeden Gesamtparameter in der Zielfunktion gestiftete Nutzen um 100 Nutzeneinheiten schwanken kann. Kein Gesamtparameter wurde einer Restriktion unterworfen.

Die Laufzeiten von Solver und Heuristik, sowie die Lösungsgüte der Heuristik im Vergleich zur optimalen Lösung des Solvers können Tabelle 14 entnommen werden.

Problem-größe	Solver		Heuristik		$\frac{t_h}{t_s}$	$\frac{F(\vec{x}_h)}{F(\vec{x}_s)}$
	t_s [ms]	$F(\vec{x}_s)$	t_h [ms]	$F(\vec{x}_h)$		
11x25x03	38,5662	1.070,9763	8,3539	1.066,8497	21,66%	99,61%
11x25x06	18.637,9961	2.056,3684	58,5567	2.048,6273	0,31%	99,62%
11x25x09	32.013,4121	3.057,8445	76,0335	3.049,2357	0,24%	99,72%
11x25x12	30.989,9199	4.065,5017	99,7332	4.051,7551	0,32%	99,66%
11x25x15	166.338,4688	4.977,6914	121,5672	4.959,2883	0,07%	99,63%

Tabelle 14 - Vergleich von Solver und Heuristik bei variierender Parameteranzahl

Die Erhöhung der Anzahl der Parameter führt sowohl beim Solver, als auch bei der Heuristik zur Erhöhung der Laufzeit. Die Laufzeit des Solvers steigt hierbei gegenüber der Laufzeit der Heuristik wesentlich schneller an, wodurch der Laufzeitvorteil der Heuristik schnell wächst. Bei einer Erhöhung der Parameteranzahl von drei auf 15 steigt die benötigte Laufzeit des Solvers ca. um den Faktor 4300, wohingegen sich die Laufzeit der Heuristik nur ca. um den Faktor 14 erhöht. Bei der getesteten Problemgröße von 11x25x15 benötigt die Heuristik lediglich 0,07% der Rechenzeit des Solvers. Eine Zunahme der Parameteranzahl führt jedoch nicht zur Verschlechterung der Lösungsgüte der Heuristik, die in der ganzen Testreihe bei ca. 99,6% lag.

Die Entwicklung der Laufzeit des Solvers und der Heuristik sind in Abbildung 37 graphisch dargestellt.

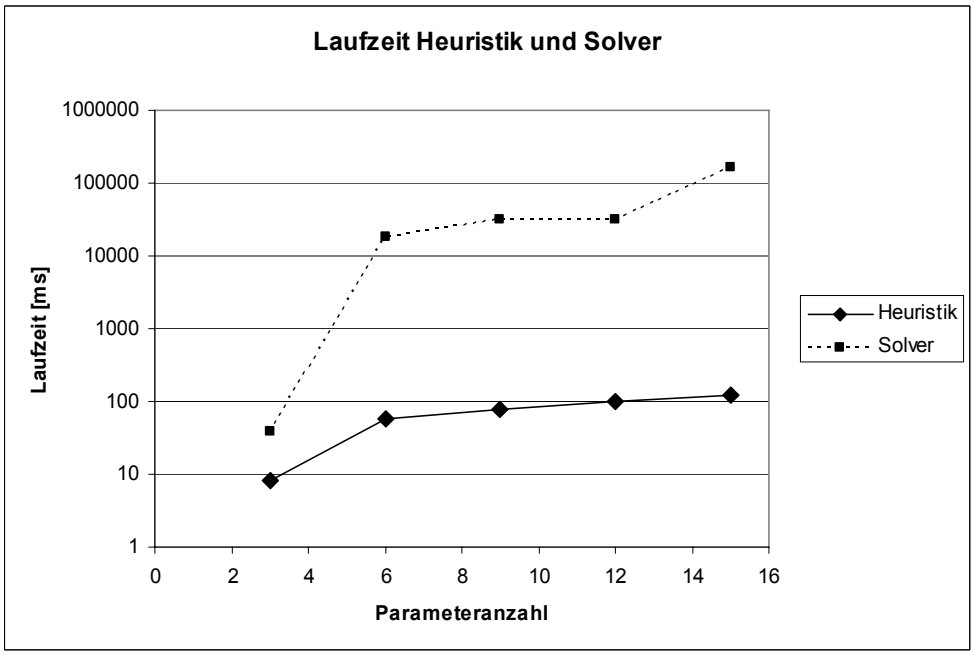

Abbildung 37 - Laufzeiten von Solver und Heuristik in Abhängigkeit von der Parameteranzahl

Am logarithmisch skalierten Verlauf der Laufzeiten kann das polynomielle Laufzeitverhalten der Heuristik deutlich erkannt werden. Im Bereich der vorliegenden Messwerte besitzt der Solver ein Laufzeitverhalten, welches sich am besten als gute Nähe-

rung eines polynomiellen Laufzeitverhaltens beschrieben lässt. Die Laufzeit steigt jedoch im Falle von 15 Parametern nochmals stärker an als in den vorherigen Problemgrößen, weswegen eine Aussage über den weiteren Verlauf hier nicht getroffen werden kann.

Die Laufzeit der Heuristik verteilt sich in Abhängigkeit von der Parameteranzahl wie in Abbildung 38 dargestellt auf die Abschnitte OM, UIM und SA der Heuristik. Bei der Problemgröße 11x25x03 ist zu beachten, dass hierbei die Abschnitte UIM und SA der Heuristik nicht ausgeführt wurden, da OM eine erste zulässige Lösung mit einem Zielfunktionswert von über 99% des theoretischen Optimums lieferte.

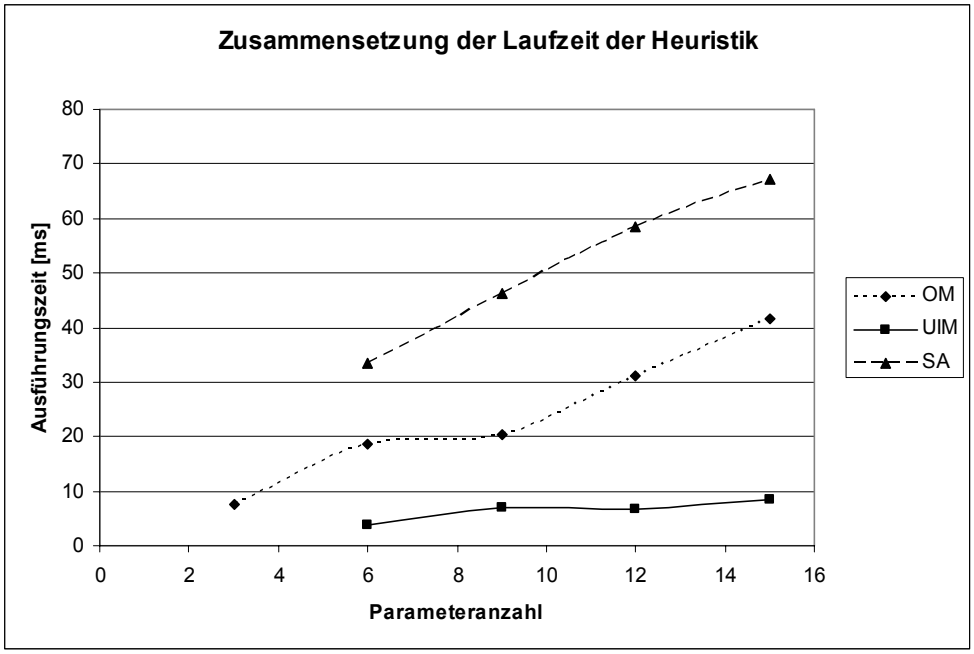

Abbildung 38 - Zusammensetzung der Laufzeit der Heuristik

In den getesteten Fällen sind die Verhältnisse der Laufzeiten der einzelnen Abschnitte OM, UIM und SA an der Gesamtlaufzeit der Heuristik annähernd gleichbleibend. Bei der Zunahme der Parameteranzahl ist keine eindeutige Tendenz einer Verschiebung der anteiligen Laufzeit von einem zum anderen Abschnitt der Heuristik erkennbar. Mit Ausnahme des ersten Testfalls der Testreihe benötigte die OM ca. ein Drittel der Gesamtlaufzeit um eine erste zulässige Lösung zu erstellen. UIM versuchte in ca. 7% der Gesamtlaufzeit diese Lösung weiter in Richtung eines lokalen Maximums zu verbessern. Die restlichen ca. 60% wurden anschließend durch SA für den Versuch verbraucht eine weitere Verbesserung der Lösung herbeizuführen. Abbildung 39 verdeutlicht nochmals die anteilige Entwicklung der Laufzeiten von OM, UIM und SA an der Gesamtlaufzeit der Heuristik.

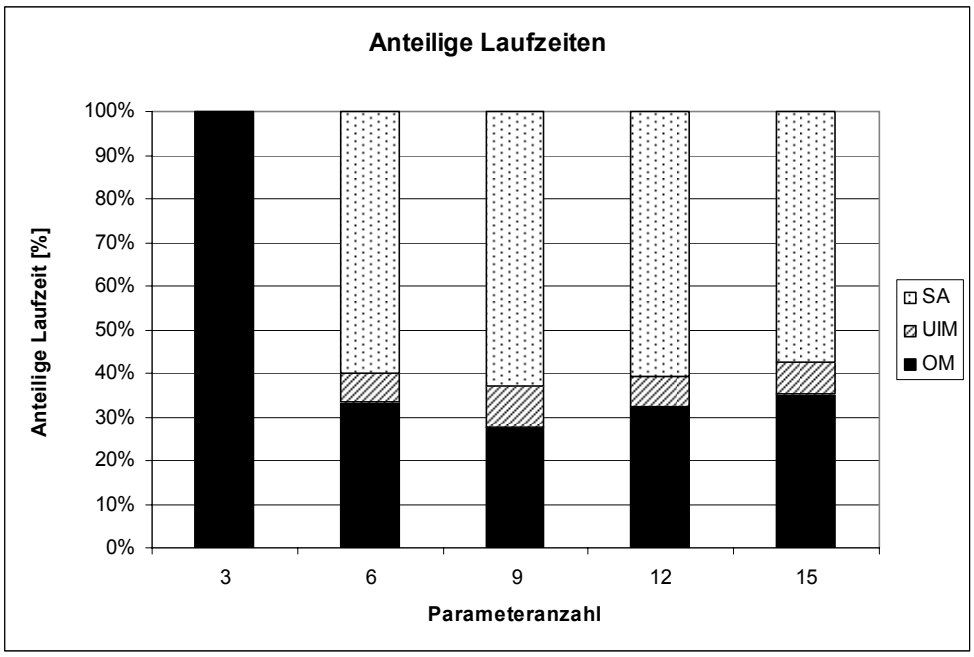

Abbildung 39 - Anteil von OM, UIM und SA an der Gesamtlaufzeit

Die durch OM erzeugte erste zulässige Lösung besitzt in den, in dieser Testreihe gemessenen Fällen einen Anteil von über 99,9% am Zielfunktionswert der Gesamtlösung der Heuristik. In allen Fällen in denen UIM ausgeführt wurde, konnte hierdurch eine weitere Verbesserung des Zielfunktionswertes erzielt werden, jedoch fiel der erreichte Zuwachs mit einem Anteil von unter einem Promille des Zielfunktionswertes der Gesamtlösung gering aus. SA konnte in zwei Testfällen ebenfalls zu einer Verbesserung des Zielfunktionswertes beitragen, die jedoch noch wesentlich marginaler ausfiel. Der Anteil am Zielfunktionswert der Lösung der Gesamtheuristik, den OM, UIM und SA liefern, kann Abbildung 40 entnommen werden.

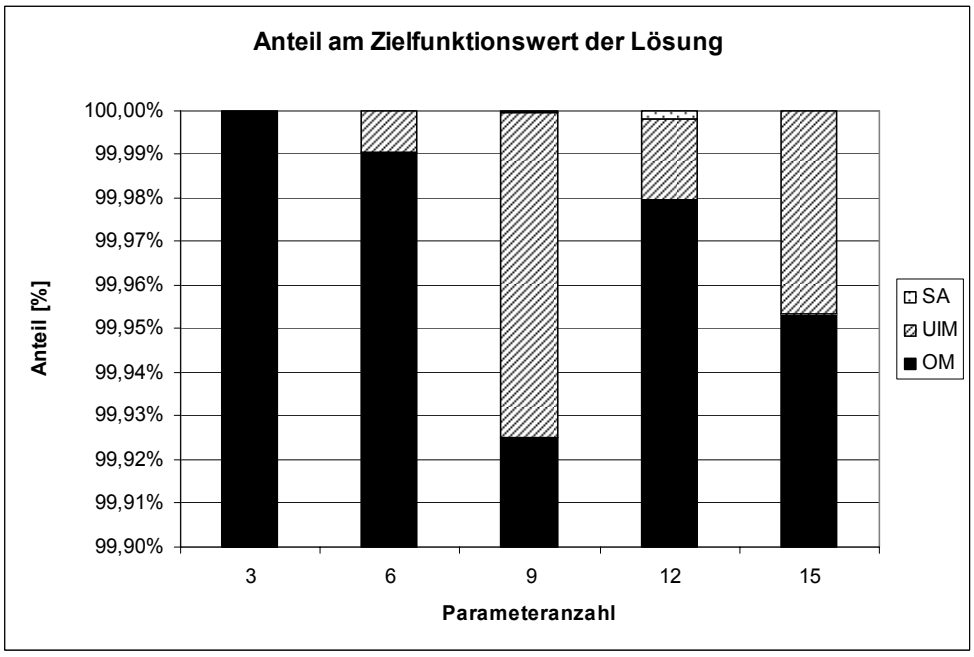

Abbildung 40 - Anteil von OM, UIM und SA am Zielfunktionswert der Lösung

Bei einer Betrachtung der Laufzeiten einer Iteration von UIM und SA kann festgestellt werden, dass diese, wie in den vorherigen Testreihen, nahezu identisch sind und linear mit der Veränderlichen skalieren. Bei der Zunahme der Parameteranzahl von drei auf 15 (Faktor 5) nahm die Dauer einer Iteration von UIM und SA von ca. 15 ms auf ca. 30 ms (Faktor 2) zu. Die Dauer einer Iteration von UIM und SA wird in Abbildung 41 in Abhängigkeit von der Parameteranzahl graphisch dargestellt.

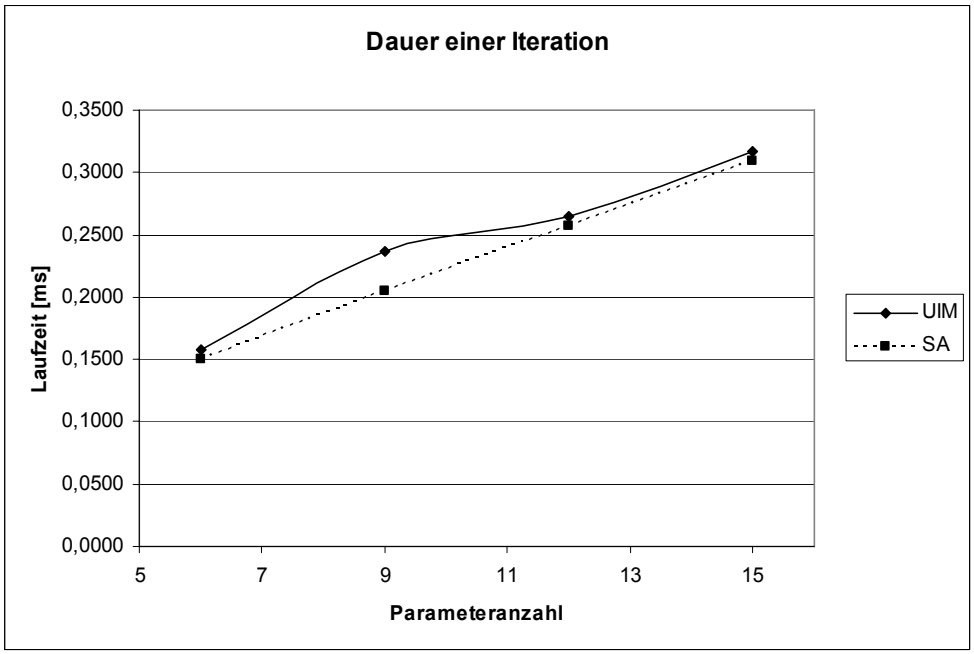

Abbildung 41 - Dauer einer Iteration von UIM und SA

Die Anzahl der Iterationen, die UIM und SA durchführen zeigt sich unabhängig von der Parameteranzahl weitestgehend konstant (siehe Abbildung 42). Die Anzahl maximal aufeinander folgender Iterationen ohne Verbesserung des Zielfunktionswertes ist sowohl bei UIM als auch bei SA direkt von der konstanten Geschäftsprozesslänge abhängig. Im Falle variabler Prozessschrittkandidatenanzahl stieg die Anzahl der Iterationen von SA trotz konstanter Geschäftsprozesslänge noch an, da sich die Anzahl alternativer Nachbarlösungen erhöhte. In dieser Testreihe ist jedoch auch die Anzahl der Prozessschrittkandidaten konstant, wodurch die Anzahl von UIM als auch von SA relativ konstant bleibt.

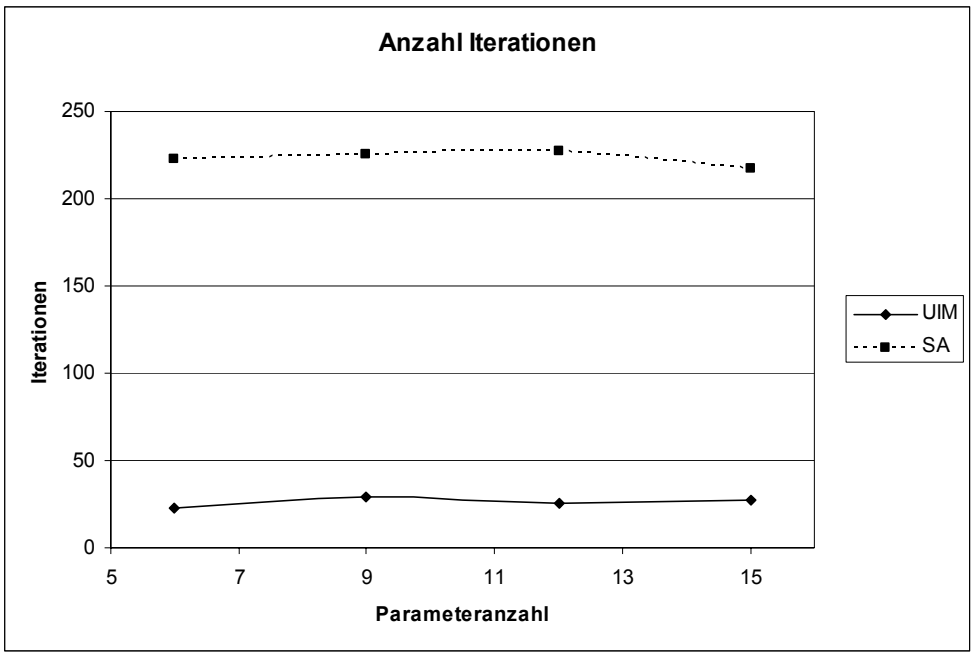

Abbildung 42 - Anzahl der Iterationen von UIM und SA

Für die Testreihe mit zunehmender Parameteranzahl lässt sich zusammenfassend feststellen, dass es der Heuristik mit steigender Parameteranzahl zunehmend gelingt ihren Laufzeitvorteil gegenüber der exakten Lösung des Problems durch den Solver zu verbessern, wobei sie ein polynomielles Laufzeitverhalten zeigt. Ein Absinken der Lösungsgüte der Heuristik ist im Gegensatz zu allen anderen Testreihen nicht zu erkennen. Mit ca. 99,6% des Zielfunktionswertes der optimalen Lösung kann die Lösungsgüte als sehr hoch eingestuft werden. Über 99,9% des erreichten Zielfunktionswertes wird hierbei in einem Drittel der Laufzeit durch OM erzeugt. Die Anteile der einzelnen Abschnitte OM, UIM und SA an der Gesamtlaufzeit der Heuristik, wie auch an der erbrachten Lösungsgüte, bleiben bei Zunahme der Parameteranzahl nahezu konstant.

5.3 Einfluss der Restriktionsstärke

Neben der Problemgröße wird die Laufzeit und die Lösungsgüte der Heuristik maßgeblich durch die Stärke der Restriktionen beeinflusst, welche die Gesamtparameter einer zulässigen Lösung im Optimierungsproblem beschränken. Um den Einfluss der Restriktionsstärke näher zu untersuchen, wurde eine Testreihe gebildet, in denen die Gesamtparameter unterschiedlich starken Restriktionen unterworfen wurden.

5.3.1 Aufbau der Testreihe

Als Ausgangsbasis zur Bildung dieser Testreihe wurde der Testcase mit der Problemgröße 20x40x04 verwendet, der bereits in der Testreihe zur Untersuchung des Einflusses der Geschäftsprozesslänge und der Prozessschrittkandidatenanzahl verwendet wurde. Zur leichteren Benennung der Parameter seien die im Kapitel 5.2.1 genannten beispielhaften Parameternamen, die zusammen mit weiteren Informationen nochmals in Tabelle 15 zusammengefasst werden, im Folgenden übernommen.

Index	Parametername	Parametertyp	größere Werte besser	korreliert mit
1	Antwortzeit	Additiv	Nein	-
2	Preis	Additiv	Nein	Antwortzeit
3	Verfügbarkeits-wahrscheinlichkeit	Multiplikativ	Ja	-
4	Maximaler Durchsatz	Minimaloperator	Ja	-

Tabelle 15 - In der Testreihe verwendete Parameter zur Beschreibung eines Web Services

In der Zielfunktion der Testcases wurde nur die Gesamtantwortzeit gewichtet. Alle anderen, in der Zielfunktion ungewichteten, Gesamtparameter wurden mit Restriktionen jeweils gleicher Restriktionsstärke belegt. Hierdurch wird jeweils ein additiver, ein multiplikativer und ein Minimaloperatorgesamtparameter restringiert, der nicht bereits durch die Gewichtung in der Zielfunktion optimiert wird. Die einzelnen Testcases der Testreihe wurden gebildet, indem die Restriktionsstärke der restringierten Gesamtparameter in Schritten zu je 12,5% von 25% auf 75% gesteigert wurde.

Wird ein Gesamtparameter in der Zielfunktion gewichtet und gleichzeitig einer Restriktion unterworfen, so führt ein höherer Zielerreichungsgrad der Optimierung des Gesamtparameters automatisch auch zu einem höheren Zielerreichungsgrad der Einhaltung der Restriktion bezüglich dieses Gesamtparameters. Wird beispielsweise die Gesamtantwortzeit optimiert und gleichzeitig gefordert, dass die Gesamtantwortzeit eine gewisse Schranke nicht überschreiten darf, so führt die Optimierung der Gesamtantwortzeit automatisch auch zur besseren Einhaltung der Schranke. Durch die Einführung von Restriktionen auf genau den Gesamtparametern, die nicht in der Zielfunktion gewichtet werden, wird die Problemkomplexität gesteigert, da eine Optimierung des Zielfunktionswertes nicht automatisch mit einer besseren Erfüllung der Restriktionen einhergeht. Im speziellen Fall dieser Testreihe ist der Preis eines Web Services mit seiner Antwortzeit negativ korreliert, d.h. ein Web Service mit niedriger Antwortzeit besitzt tendenziell einen hohen Preis, weshalb durch die Optimierung der Gesamtantwortzeit und der Restriktion des Gesamtpreises ein Zielkonflikt entsteht. Durch die Optimierung der Antwortzeit wird versucht genau solche Lösungen zu finden, die möglichst schnell sind, jedoch erzeugt dies automatisch Lösungen die sehr teuer sind und dadurch die Restriktion des Gesamtpreises verletzen. Je besser die Optimierung der Gesamtantwortzeit ausfällt, desto schlechter kann die Restriktion des Gesamtpreises eingehalten werden. Durch die negative Korrelation von Antwortzeit und Preis wurde die Komplexität dieser Testreihe noch zusätzlich gesteigert.

5.3.2 Analyse der Testreihe

Die Lösung der einzelnen Testcases der beschriebenen Testreihe durch Solver und Heuristik führte zu den in Tabelle 16 aufgeführten Laufzeiten und Lösungsgüten.

Problemgröße - Restriktionsstärke	Solver		Heuristik		$\frac{t_h}{t_s}$	$\frac{F(\vec{x}_h)}{F(\vec{x}_s)}$
	t_s [ms]	$F(\vec{x}_s)$	t_h [ms]	$F(\vec{x}_h)$		
20x40x04 - 25%	181,66	-9.161,34	123,19	-9.355,92	67,81%	97,92%
20x40x04 - 37,5%	502,11	-9.866,28	133,80	-10.084,55	26,65%	97,84%
20x40x04 - 50%	7.420,57	-11.590,63	148,83	-12.457,14	2,01%	93,04%
20x40x04 - 62,5%	11.974,19	-13.051,94	400,82	-15.095,48	3,35%	86,46%
20x40x04 - 75%	2.566.891,75	-19.651,26	401,38	-21.522,66	0,02%	91,30%

Tabelle 16 - Vergleich von Solver und Heuristik bei variierender Restriktionsstärke

Durch die Erhöhung der Restriktionsstärke steigt sowohl die Laufzeit des Solvers, als auch die der Heuristik an. Der Anstieg der Laufzeit des Solvers fällt jedoch wesentlich höher aus, als die der Heuristik, wodurch sich der Laufzeitvorteil der Heuristik gegenüber dem Solver mit zunehmender Restriktionsstärke zunehmend erhöht. Besonders deutlich wird dies im Fall der Erhöhung der Restriktionsstärke von 62,5% auf 75%. Die Laufzeit des Solvers steigt hier von ca. 12.000 ms auf ca. 2.567.000 ms an (ca. Faktor 214). Die Laufzeit der Heuristik bleibt hingegen nahezu konstant und ist zusätzlich noch in der Lage die Lösungsgüte zu steigern. In diesem Fall benötigt die Heuristik nur 0,02% der Laufzeit des Solvers, liefert aber eine Lösung deren Zielfunktionswert 91,3% des Zielfunktionswertes der optimalen Lösung beträgt. Im Vergleich zu den anderen betrachteten Testreihen sinkt die Lösungsgüte der Heuristik mit zunehmender Restriktionsstärke in dieser Testreihe deutlicher ab.

Die Entwicklung der Laufzeit des Solvers und der Heuristik ist in Abbildung 43 graphisch dargestellt.

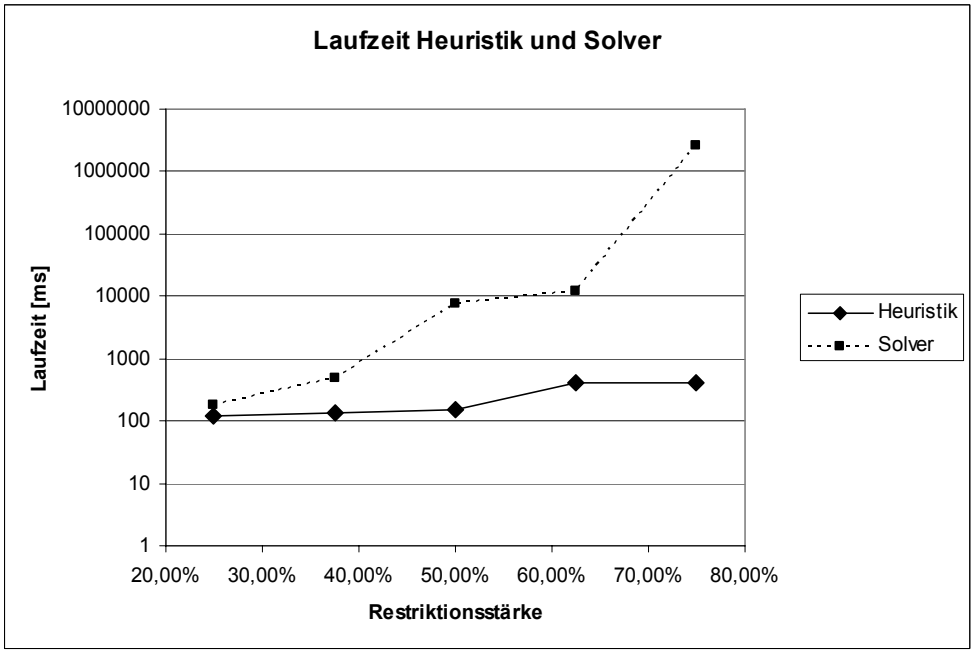

Abbildung 43 - Laufzeiten von Solver und Heuristik in Abhängigkeit von der Restriktionsstärke

Die Laufzeiten von Solver und Heuristik zeigen mit zunehmender Restriktionsstärke tendenziell eine Zunahme der Steigung in der logarithmisch skalierten Darstellung, d.h. ein mehr als exponentielles Wachstum, welches im Falle des Solvers wesentlich stärker ausfällt, als im Falle der Heuristik. Betrachtet man die Laufzeiten der Abschnitte OM, UIM und SA ebenfalls in einer logarithmisch skalierten Darstellung so

erkennt man, dass sich die Laufzeit von UIM und SA eindeutig mit mehr als exponentiellem Wachstum entwickeln und diese mit steigender Restriktionsstärke zunehmend die Gesamtausführungszeit der Heuristik bestimmen.

Die Interpretation der Messwerte ist jedoch in dieser Testreihe schwierig, da mit zunehmender Restriktionsstärke der Zufall eine stetig steigende Bedeutung im Lösungsverfahren einnimmt. Nachdem das relaxierte Optimierungsproblem durch den Solver gelöst wurde, nimmt OM eine Sortierung der Prozessschrittkandidaten vor und versucht hierdurch möglichst schnell eine zulässige Lösung mit hohem Zielfunktionswert zu generieren. Dabei kann die Sortierung der Prozessschrittkandidaten in besonderem Maße geeignet sein die Restriktionen zu erfüllen, oder auch gegenteilig, besonders hinderlich. In den sich anschließenden Abschnitten UIM und SA werden sukzessive Nachbarlösungen zufällig besucht. Die Wahrscheinlichkeit hierbei einen Weg durch die Nachbarlösung zu finden, der nicht in einer Nachbarlösung ohne weitere zulässigen Lösungen endet, sinkt mit der Zunahme der Restriktionsstärke. Auch der Solver verwendet bei der Lösung des unrelaxierten Optimierungsproblems eine Lösungsstrategie, die je nach konkretem Testfall besonders geeignet dazu sein kann möglichst schnell zugleich optimale, als auch zulässige Lösungen zu finden, jedoch auch gegenteilig besonders nachteilig. Unter dem Einfluss von Restriktionen und mit der Zunahme der Restriktionsstärke wird daher sowohl das Verhalten der Heuristik, als auch das Verhalten des Solvers zunehmend vom Zufall bestimmt. Ein Beispiel hierfür ist das Verhalten der Heuristik im konkreten Testfall mit der Restriktionsstärke von 62,5%. OM benötigt hier wesentlich mehr Zeit als in den Testfällen mit der Restriktionsstärke 50% oder 75%.

Durch derartige Effekte sind keine idealtypischen Laufzeitverläufe zu erwarten und die tendenzielle Entwicklung der Laufzeit kann nur grob geschätzt werden. In grober Näherung kann jedoch festgehalten werden, dass der Einfluss der Restriktionsstärke sowohl auf die benötigte Laufzeit, als auch auf die Lösungsgüte erheblich ist und dass eine Erhöhung der Restriktionsstärke zu einem mehr als exponentiellen Wachstum der Laufzeiten führen kann.

Die Aufteilung der Laufzeit der Heuristik auf die einzelnen Abschnitte OM, UIM und SA kann Abbildung 44 entnommen werden.

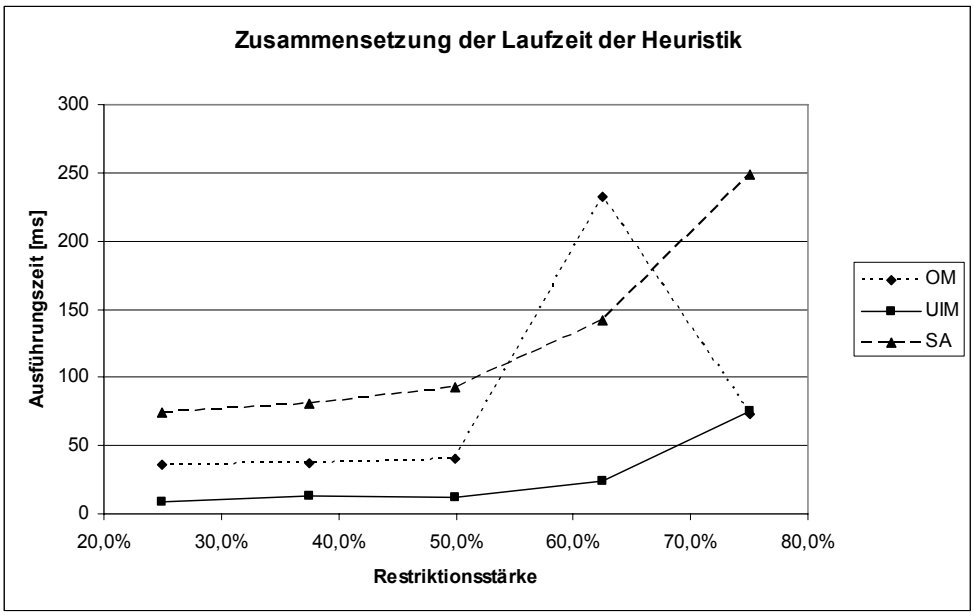

Abbildung 44 - Zusammensetzung der Laufzeit der Heuristik

UIM nimmt den geringsten Teil der Gesamtlaufzeit in Anspruch und entwickelt sich annähernd gleichförmig zur Laufzeit von SA, welches in den meisten Fällen den höchsten Anteil an der Laufzeit beansprucht. Die Laufzeit von OM liegt zunächst zwischen der von UIM und SA, entwickelt sich jedoch mit zunehmender Restriktionsstärke sprunghaft. Eine Aussage über die anteilige Entwicklung der Laufzeiten bei der Steigerung der Restriktionsstärke lässt sich aus diesem Grund schwer treffen. Die gemessenen anteiligen Laufzeiten von OM, UIM und SA an der Gesamtlaufzeit der Heuristik sind in Abbildung 45 in Abhängigkeit von der Restriktionsstärke dargestellt.

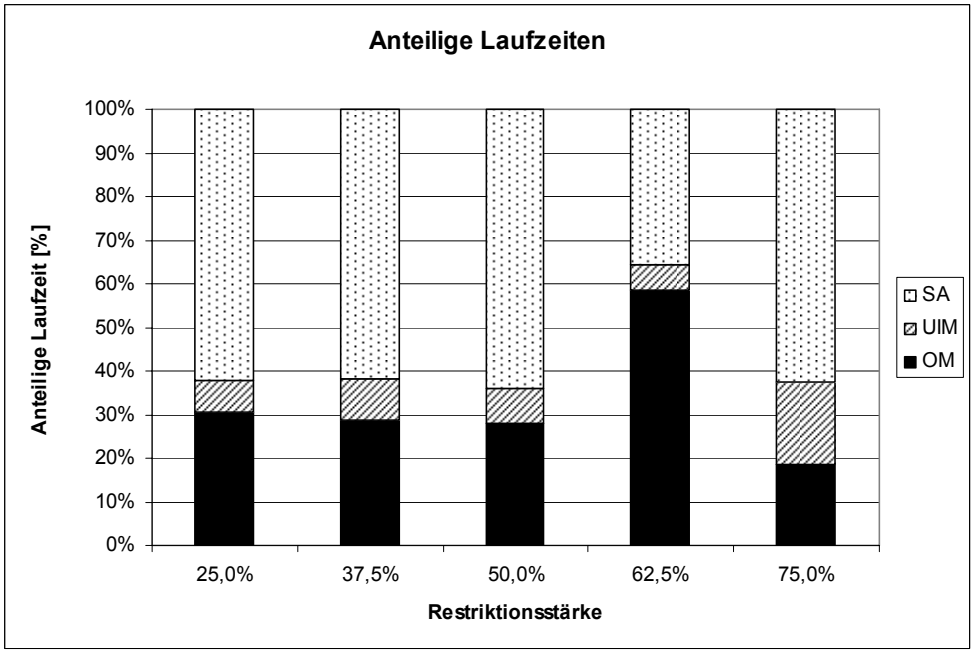

Abbildung 45 - Anteil von OM, UIM und SA an der Gesamtlaufzeit

In den Testreihen, die zur Untersuchung des Einflusses der Problemgröße verwendet wurden, lieferte OM stets eine erste zulässige Lösung, die einen Anteil von über

99,9% am Zielfunktionswert der Gesamtlösung der Heuristik besaß. Die Lösungsgüte der Gesamtlösung unterschritt in keinem beobachteten Fall 98%. In dieser Testreihe fällt sowohl der Anteil von OM am Zielfunktionswert der Gesamtlösung, als auch die Lösungsgüte der Gesamtlösung geringer aus. In einem Fall beträgt der Anteil von OM am Zielfunktionswert der Gesamtlösung nur ca. 95% und die Lösungsgüte sinkt in einem anderen Fall auf ca. 86,5% ab. Zusammengenommen bedeutet dies, dass OM bei steigender Restriktionsstärke zunehmend erste zulässige Lösungen mit zunehmend geringerer Güte liefert, als dies in allen anderen Testreihen bisher beobachtet wurde. Die Restriktionsstärke scheint daher einen hohen Einfluss auf die Güte der ersten zulässigen Lösung zu besitzen, die durch OM erzeugt wird. Der Anteil am Zielfunktionswert der Lösung der Gesamtheuristik, den OM, UIM und SA liefern, kann Abbildung 46 entnommen werden.

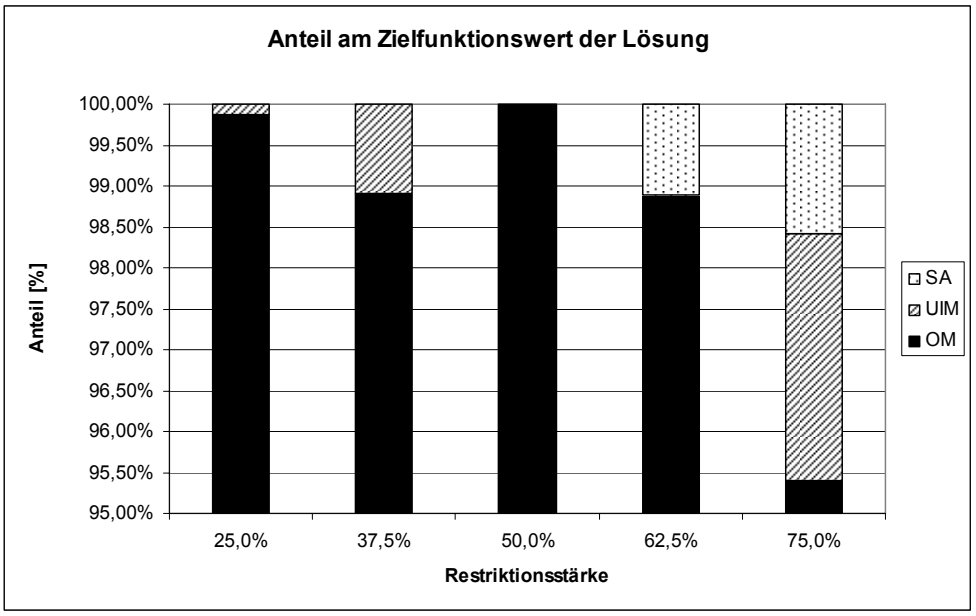

Abbildung 46 - Anteil von OM, UIM und SA am Zielfunktionswert der Lösung

Aufgrund der sinkenden Lösungsgüte der von OM erzeugten ersten zulässigen Lösung steigt die Wahrscheinlichkeit an, dass die nachgelagerten Abschnitte UIM und SA hier weitere Verbesserungen erzielen können. Im Gegensatz zu den zuvor betrachteten Testreihen, kann in dieser Testreihe nun nicht nur UIM einen signifikanten Beitrag zur Steigerung der Lösungsgüte beitragen, sondern auch SA. An dieser Stelle zeigt sich, dass die nachgelagerten Stufen UIM und SA vor allem dann greifen, wenn die erste zulässige Lösung von geringer Güte ist. Die Lösung des Optimierungsproblems ist jedoch bei hohen Restriktionsstärken sehr schwierig, weshalb die Lösungsgüte der Gesamtlösung der Heuristik nicht mehr die aus anderen Testreihen gewohnte hohe Lösungsgüte liefern kann. In Anbetracht dieser Schwierigkeit, kann die Lösungsgüte dennoch als gut eingestuft werden.

Die Dauer einer Iteration von UIM und SA in Abhängigkeit von der Restriktionsstärke kann Abbildung 47 entnommen werden.

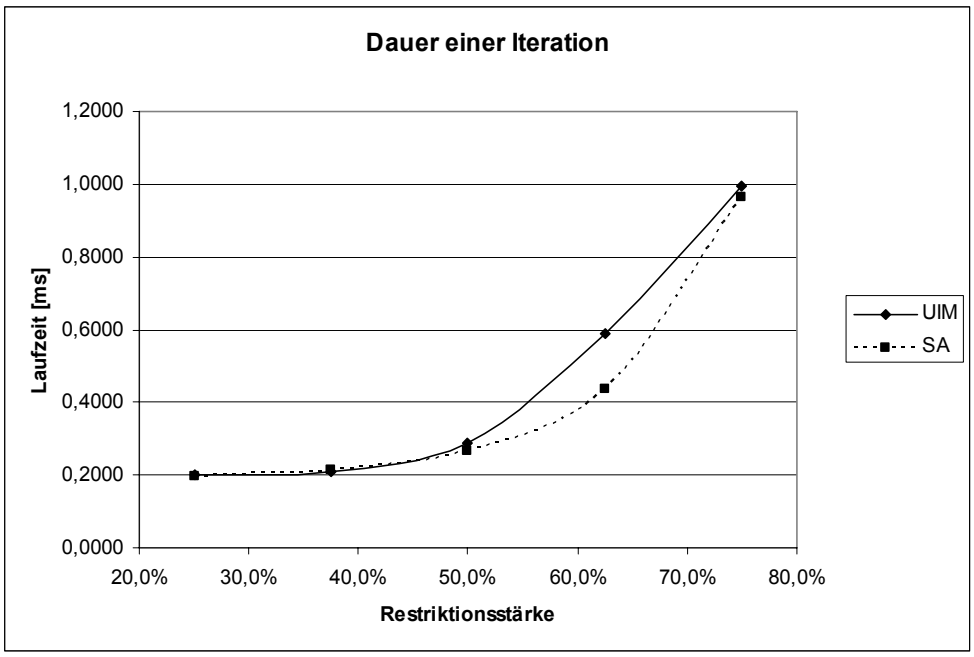

Abbildung 47 - Dauer einer Iteration von UIM und SA

Die Dauer einer Iteration von UIM und SA ist in dieser Testreihe in grober Näherung zwar in etwa gleich, jedoch entwickelt sich die Iterationsdauer entgegen allen zuvor beobachteten Testreihen nicht mehr linear, sondern mehr als exponentiell. Dieser Umstand lässt sich dadurch erklären, dass bei der Suche einer zulässigen Nachbarlösung, die in jeder Iteration durchgeführt wird, solange an einer bestimmten Position des Ausführungsplans alternative Prozessschrittkandidaten ausprobiert werden, bis eine zulässige Nachbarlösung gefunden wird. Mit zunehmender Restriktionsstärke sinkt jedoch der Anteil der Prozessschrittkandidaten, die zu einer zulässigen Nachbarlösung führen, stärker als die Zunahme der Restriktionsstärke ab. Hierdurch steigt die Anzahl der pro Iteration auszuprobierenden Prozessschrittkandidaten stärker als die Restriktionsstärke an.

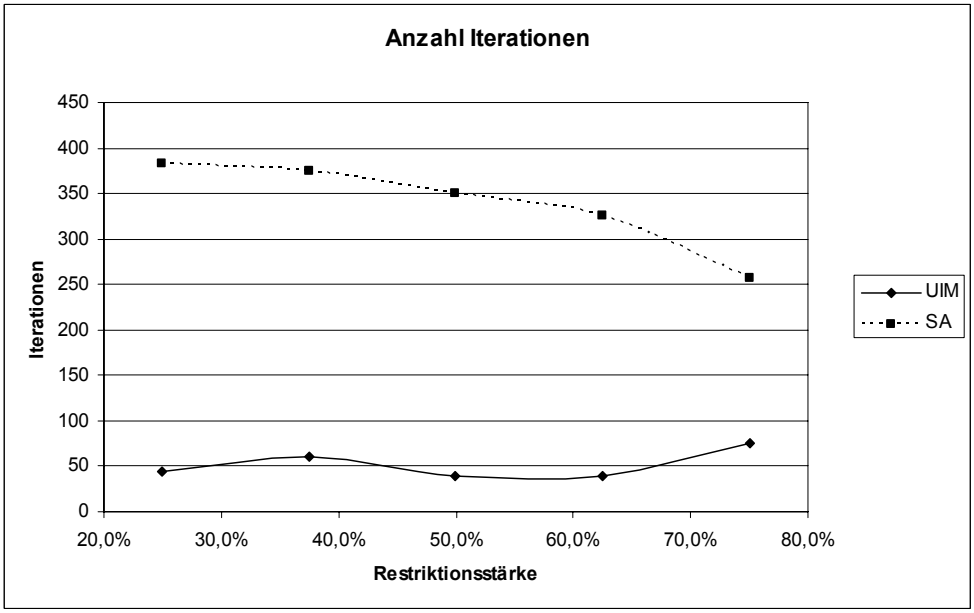

Abbildung 48 - Anzahl der Iterationen von UIM und SA

Betrachtet man die Anzahl der von UIM und SA durchgeführten Iterationen in Abbildung 48, so schwankt die Iterationsanzahl von UIM scheinbar um einen konstanten Wert, während die Iterationsanzahl von SA kontinuierlich sinkt. Die Anzahl der maximal hintereinander auszuführenden Iterationen ohne Verbesserung des Zielfunktionswertes hängt sowohl bei UIM als auch bei SA von der, in dieser Testreihe konstanten, Geschäftsprozesslänge ab. Mit zunehmender Restriktionsstärke fällt es sowohl UIM als auch SA zunehmend schwerer durch zufälliges Ausprobieren eine zulässige Nachbarlösung und unter diesen eine zulässige Nachbarlösung mit besserem Zielfunktionswert zu finden. Da jedoch OM eine erste zulässige Lösung geringerer Güte liefert, kann UIM diesen Effekt durch das höhere verbleibende Optimierungspotential in dieser Lösung kompensieren und schwankt daher um einen annähernd konstanten Wert. SA versucht anschließend die bereits von UIM optimierte Lösung weiter zu optimieren. Hierbei bricht SA nach einer immer geringer werdenden Anzahl von Iterationen ab. Erstaunlich ist hierbei jedoch, dass SA gerade in den Fällen hoher Restriktionsstärke und niedriger Iterationsanzahl einen höheren Beitrag zum Zielfunktionswert der Gesamtlösung der Heuristik leistet.

Für die Testreihe mit zunehmender Restriktionsstärke lässt sich zusammenfassend feststellen, dass es der Heuristik mit steigender Restriktionsstärke in hervorragender Weise gelingt ihren Laufzeitvorteil gegenüber dem Solver auszubauen. Mit ansteigender Restriktionsstärke kann die Laufzeit des Solvers, als auch die der Heuristik, mehr als exponentiell anwachsen. Die Lösungsgüte fällt hierbei deutlicher ab, als dies in Abhängigkeit von der Problemgröße beobachtet wurde. Die schlechteste gemessene Lösungsgüte von ca. 86% kann in Anbetracht der Komplexität des Optimierungsproblems im Falle hoher Restriktionsstärken jedoch noch als gut eingestuft werden. OM liefert mit zunehmender Restriktionsstärke vermehrt erste zulässige Lösungen geringerer Güte. Es zeigt sich jedoch, dass das in der geringen Güte der Lösung liegende verbleibende Optimierungspotential durch die nachfolgenden Abschnitte UIM und SA zunehmend für eine weitere Optimierung der Lösung genutzt werden kann und diese hierdurch zu einem immer größeren Anteil der Lösungsgüte der Gesamtlösung der Heuristik beitragen können. Vor allem SA erzielt in dieser Testreihe die ersten signifikanten Beiträge zur Güte der Gesamtlösung der Heuristik.

5.4 Zusammenfassende Betrachtung

In den vorangegangenen Abschnitten wurde der Einfluss der wichtigsten Größen auf die Laufzeit und die Lösungsgüte der Heuristik untersucht, wobei ein Vergleich mit der exakten Lösung des Optimierungsproblems durch den Solver durchgeführt wurde. Hierbei wurde der Einfluss der Problemgröße, bestimmt durch Geschäftsprozesslänge, Prozessschrittkandidatenanzahl und Parameteranzahl, sowie der Einfluss der Restriktionsstärke untersucht.

Unabhängig von der untersuchten Einflussgröße weist die Heuristik einen erheblichen Laufzeitvorteil gegenüber der exakten Lösung des Optimierungsproblems durch den Solver auf. Der Laufzeitvorteil der Heuristik verstärkt sich zudem mit der Zunahme der Einflussgröße merklich. In einem Extremfall konnte ein Laufzeitvorteil von 2.566.891,75 ms zu 401,38 ms gemessen werden, was einem Unterschied von über 42 Minuten entspricht. Hieran verdeutlicht sich nicht nur die Überlegenheit der Heuristik gegenüber der exakten Lösung durch den Solver in Bezug auf die Laufzeit, sondern auch der Umstand, dass es nur durch eine Heuristik möglich ist, ein solches Optimie-

rungsproblem mit einem Rechenzeitbedarf zu lösen, der es ermöglicht eine solche Optimierung zur Laufzeit in einer BPE einsetzen zu können.

Die Lösungsgüte der Heuristik liegt in allen Testreihen bezüglich des Einflusses der Problemgröße bei über 98%, d.h. die Heuristik erreicht trotz der erheblich kürzeren Laufzeit einen Zielfunktionswert von über 98% des Zielfunktionswertes der exakten, optimalen Lösung. Das Verhältnis aus benötigter Laufzeit und erreichter Lösungsgüte kann daher als überaus gut eingestuft werden. Lediglich in einer Testreihe zur Untersuchung des Einflusses der Restriktionsstärke, für welche ein Optimierungsproblem hoher Lösungskomplexität eingesetzt wurde, sank die Lösungsgüte bei höherer Restriktionsstärke weiter ab. Hier wurde die niedrigste gemessene Lösungsgüte mit 86,5% erreicht, was in Anbetracht der Komplexität des Optimierungsproblems jedoch durchaus noch als gut bewertet werden kann. In allen betrachteten Testreihen wurde der Großteil der Lösungsgüte durch den Abschnitt OM der Heuristik erzeugt, wohingegen die nachgelagerten Abschnitte UIM und SA nur selten einen signifikanten Beitrag zur Lösungsgüte liefern konnten, hierbei jedoch einen Großteil der Rechenzeit beanspruchten.

In den Testreihen bezüglich der, die Problemgröße beeinflussenden Parameter Geschäftsprozesslänge, Prozessschrittkandidatenanzahl und Parameteranzahl zeigte die Heuristik stets ein polynomielles Laufzeitverhalten. Der Solver zeigte im gemessenen Bereich in Bezug auf Geschäftsprozesslänge und Prozessschrittkandidatenanzahl ein Laufzeitverhalten, welches am ehesten als exponentiell bezeichnet werden kann. In der Testreihe bezüglich des Einflusses der Parameteranzahl zeigte jedoch auch der Solver ein annähernd polynomielles Laufzeitverhalten.

Die Testreihe zur Untersuchung des Einflusses der Restriktionsstärke zeigte einen hohen Einfluss der Restriktionsstärke auf die Laufzeit und Lösungsgüte. Eine genaue Systematik dieses Zusammenhangs aufzuzeigen ist jedoch nicht möglich, da in restringierten Optimierungsproblemen der Zufall eine größerwerdende Bedeutung erhält. Eine gewählte Lösungsstrategie kann in einem Fall besonders förderlich sein, Lösungen aufzuspüren die sowohl möglichst optimal sind und zugleich besonders geeignet sind die Restriktionen zu erfüllen, kann jedoch auch in einem anderen Fall ebenso hinderlich sein und hierdurch die Lösungsdauer erhöhen und die Lösungsgüte senken. Mit der abnehmenden Anzahl von zulässigen Lösungen im Lösungsraum durch die Zunahme der Restriktionsstärke, führt eine sukzessive zufällige Untersuchung von Nachbarlösungen immer seltener zu der Auswahl eines Weges durch die Nachbarlösungen, der zugleich hinsichtlich der Optimierung, als auch hinsichtlich der Erfüllung der Restriktionen vielversprechend oder gar optimal ist. Durch die Nichtexistenz einer für alle Fälle optimalen Lösungsstrategie und dem zunehmenden Einfluss des Zufalls auf den Verlauf des Optimierungsprozesses wird der Ausgang der Anwendung der Heuristik zunehmend unvorhersagbarer und hängt vom konkreten Einzelfall ab. Generelle Aussagen können hierdurch nicht mehr getroffen werden. Festgestellt werden kann jedoch, dass eine Zunahme der Restriktionsstärke sowohl bei der Heuristik, als auch beim Solver zu einer mehr als exponentiellen Zunahme der Laufzeit führen kann.

Die Laufzeit der Heuristik scheint in Abhängigkeit von der Problemgröße polynomiell beschränkt zu sein. In Abhängigkeit vom Probleminhalt (z.B. Restriktionsstärke) kann sich die Laufzeit jedoch exponentiell entwickeln. Da die Laufzeit der Heuristik nicht alleine durch die Problemgröße, sondern auch vom Probleminhalt determiniert wird,

handelt es sich bei der Heuristik nicht um einen polynomiell beschränkten Algorithmus, sondern um einen pseudopolynomiell[32] beschränkten Algorithmus.

Alle Einflussparameter auf das Verhalten der Heuristik zu bestimmen und allgemeingültige Zusammenhänge, vor allem für den kombinierten Einfluss dieser, aufzuzeigen ist nicht möglich. Auch die Erkenntnisse über den Einfluss der Restriktionsstärke relativieren die Erkenntnisse über andere Einflussparameter. Es stellt sich auch die Frage inwieweit eine weitergehende Analyse der Heuristik sinnvoll wäre, da es sich bei der Heuristik nur um einen ersten Prototypen handelt, an welchem untersucht werden soll, ob es eine solche Heuristik überhaupt ermöglicht, das Optimierungsproblem in einer Laufzeit und mit einer Lösungsgüte lösen zu können, die ihren Einsatz zur Laufzeit in einer BPE ermöglicht. Die vorgenommene Analyse bestätigt bereits diese prinzipielle Einsatzmöglichkeit.

Im Sinne von weiterführenden Analysen ist es möglich, den Einfluss der Parameter zu untersuchen, welche das Verhalten der Heuristik in den einzelnen Schritten beeinflussen, wie z.B. die Parameter des Simulated-Annealing-Verfahrens. Veränderungen dieser Parameter führen jedoch in bestimmten Einzelfällen zu Verbesserungen und in anderen Einzelfällen zu Verschlechterungen, weswegen auf ihre Betrachtung in einer ersten allgemeinen Analyse der Heuristik verzichtet worden ist. Sinnvoll ist eine genaue Analyse jedoch in den Fällen, in welchen genauere Informationen über den konkret zu lösenden Typ von Optimierungsproblemen vorliegen, und Präferenzen für die Eigenschaften einer Optimierung, z.B. hinsichtlich Laufzeit oder Lösungsgüte vorliegen. Durch die Veränderung bestimmter Parameter lässt sich die Heuristik für bestimmte Optimierungsprobleme hinsichtlich gewisser Kriterien optimieren, was jedoch zu einer Spezialisierung der Heuristik auf bestimmte Optimierungsprobleme führen würde. Solche Betrachtungen gestalten sich sehr aufwendig, da sie auch konkrete Verteilungsannahmen über Daten der Datenbasis einschließen und berücksichtigen müssen. Derartige Betrachtungen können jedoch nicht Gegenstand einer ersten allgemeinen Analyse sein und würden den Rahmen dieses Technical Reports deutlich sprengen.

[32] Bei pseudopolynomiellen Algorithmen handelt es sich um eine Teilmenge der exponentiellen Algorithmen, die in Abhängigkeit von der Problemgröße eine scheinbar polynomiell beschränkte Laufzeit zeigen, deren Laufzeitverhalten jedoch auch durch den Probleminhalt determiniert wird, was zu einem exponentiellen Laufzeitverhalten führen kann.

6 Related Work

In diesem Kapitel wird ein Überblick über Arbeiten gegeben, die sich dem gleichen oder einem zu diesem Technical Report verwandten Themenkomplex widmen.

Zeng et al. stellen in [79] und [80] die Middleware-Plattform *AgFlow* vor. Sie ermöglicht es abstrakt beschriebene zusammengesetzte Web Services auszuführen, wobei zum Aufrufzeitpunkt ein optimaler Ausführungsplan mit konkret zu verwendenden Web Services erstellt wird, welcher die Dienstgütekriterien verfügbarer Web Services, definierte Optimierungskriterien und Restriktionen berücksichtigt. Ein zusammengesetzter Web Service wird durch ein Zustandsdiagramm definiert, welches die Kontrollkonstrukte sequentielle Ausführung, bedingte Verzweigung und Parallelausführung enthalten darf. Ein Zustand im Diagramm definiert entweder eine auszuführende Operation oder ist wiederum ein Zustandsdiagramm. Die auszuführende Operation wird abstrakt durch eine Dienstklasse ausgedrückt. Über ein erweitertes UDDI-Repository können anhand einer frei definierbaren Ontologie für Dienstklassen konkrete Web Services zur Ausführung einer Operation gefunden werden. Im Zustandsdiagramm des zusammengesetzten Web Services wird ein Weg vom Startzustand zum Endzustand als Ausführungspfad bezeichnet. Durch das Auflösen evtl. vorhandener Zyklen lässt sich zu jedem Zustandsdiagramm die endliche Menge aller möglichen Ausführungspfade angeben. Zu jedem Ausführungspfad wird ein optimaler Ausführungsplan bestimmt, der zu jeder Operation einen konkret aufzurufenden Web Service definiert. Die Ausführungspläne werden anschließend zu einem Gesamtausführungsplan aggregiert. Werden einem Zustand (Operation) in verschiedenen Ausführungsplänen verschiedene Web Services zugeordnet, so wird der Web Service in den Gesamtausführungsplan übernommen, der in dem Ausführungspfad festgelegt ist, der am häufigsten ausgeführt wird (sog. heißer Pfad). Wird nicht der heiße Pfad ausgeführt so ist es jedoch möglich, dass nicht alle Restriktionen eingehalten werden können.

Web Services werden durch die Dienstgüteparameter Ausführungszeit, Verfügbarkeit, Erfolgsrate, Preis und Reputation differenziert. Jedem Dienstgüteparameter ist eine Aggregationsfunktion zugeordnet, die zur Berechnung der Gesamtdienstgüteparameter eines Ausführungsplans verwendet wird. Durch die Multiplikation der normierten Gesamtdienstgüteparameter mit definierbaren Gewichten wird einem Ausführungsplan eine nutzenbewertende Score zugewiesen. Zur Erstellung des optimierten Ausführungsplans zu einem Ausführungspfad wird ein gemischt ganzzahliges Optimierungsproblem durch einen Solver exakt gelöst. Während der Ausführung des Gesamtausführungsplans wird ein Monitoring der Dienstgüteparameter durchgeführt. Wird festgestellt, dass die erbrachte Dienstgüte ausgeführter Web Services signifikant von der vermuteten Dienstgüte abweicht oder tritt eine signifikante Änderung von Dienstgüteparametern noch nicht ausgeführter Web Services ein, so wird ein Replanning ausgeführt. Die Aufzeichnung gemessener Ausführungszeiten kann dazu verwendet werden die Minimierung der Varianz der vorhergesagten Ausführungszeit als zusätzliches Optimierungsziel aufzunehmen.

Yu und Lin [77] befinden den Lösungsansatz von Zeng et al. als nicht geeignet um Entscheidungen bzgl. der Zusammensetzungen von Web Services zur Laufzeit treffen zu können, da die Lösung des komplexen ganzzahligen linearen Optimierungsmodells zuviel Rechenzeit beansprucht. Der von ihnen im Rahmen der *„QoS-capable Web Service Architecture“ (QCWS)* [24] entwickelte Lösungsansatz vereinfacht das Prob-

lem, zu einem abstrakt beschriebenen, zusammengesetzten Web Service konkrete Web Services zu dessen Ausführung zu bestimmen, zunächst auf zusammengesetzte Web Services mit rein sequentieller Struktur. Jede der hintereinander zu erbringenden Funktionalitäten ist abstrakt durch eine Dienstklasse beschrieben. Eine Dienstklasse enthält funktional gleiche Web Services, die durch den Anbieter in verschiedenen Serviceleveln angeboten werden können, d.h. in verschiedenen Ausprägungen mit jeweils unterschiedlichen Dienstgüteeigenschaften. Ein Kandidat zur Aufnahme in den Ausführungsplan definiert sich hierdurch durch einen Web Service und einen konkreten Servicelevel. Die Erstellung eines Ausführungsplans für den zusammengesetzten Web Service lässt sich nur einer einzigen Restriktion, der Beschränkung der Gesamtantwortzeit, unterwerfen. Die Auswahl der einzelnen Web Services erfolgt derart, dass sich insgesamt eine Nutzenmaximierung ergibt. Den Nutzen, den ein Web Service bei seiner Verwendung stiftet ist abhängig von den hierdurch entstehenden Kosten und der Auslastung des Servers[33], der ihn bereitstellt. Der Nutzen steigt mit sinkender Serverlast und Kosten.

Das Problem lässt sich aufgrund des Vorliegens nur einer Restriktion als *Multiple-choice-Knapsack-Problem (MCKP)* formulieren. Aus jeder Dienstklasse ist ein Kandidat auszuwählen und dem Knapsack zuzuführen. Einem Kandidaten werden sein Nutzen, sowie seine Antwortzeit zugeordnet. Der Knapsack ist nutzenmaximal zu füllen, wobei die Kapazität des Knapsackes durch eine maximale Antwortzeit beschränkt ist. Zur Lösung des NP-schweren MCKP wird der Algorithmus von Pisinger [52] verwendet. Dieser vereinfacht das MCKP zunächst durch Anwenden einer LP-Relaxation zum linearen Multiplechoice-Knapsack-Problem (LMCKP) und löst dieses exakt in linearer Laufzeit O(n) z.B. mittels der Algorithmen von Zemel [78] oder Dyer [28]. Anschließend werden Lösungsverfahren der dynamischen Programmierung verwendet um aus der optimalen Lösung des LMCKP die optimale Lösung des MCKP zu konstruieren. Der Algorithmus von Pisinger löst das MCKP sehr effizient und die von Yu und Lin in Experimenten gemessenen, sehr kleinen Laufzeiten sprechen für eine Eignung des Lösungsansatzes zur Optimierung zur Laufzeit. Hierbei ist jedoch zu beachten, dass das Modell sehr einfach gehalten ist und nur eine einzige Restriktion, sowie wenige Dienstgüteeigenschaften berücksichtigt. Für den Fall von zusammengesetzten Web Services mit nicht-sequentieller Struktur schlagen Yu und Lin vor, für jeden möglichen Ausführungspfad den Pisinger-Algorithmus zu verwenden und anschließend den Ausführungsplan des Ausführungspfades zu wählen, der den höchsten Nutzenwert erzielt. Wie jedoch verfahren werden soll, wenn dieser Ausführungspfad nicht auch tatsächlich ausgeführt wird, bleibt unbeantwortet. Ebenso berücksichtigen Yu und Lin kein Replanning, d.h. keine Revidierung des Ausführungsplans zur Laufzeit.

Canfora et al. [20] beschäftigen sich ebenfalls mit der dienstgüteberücksichtigenden Komposition zusammengesetzter Web Services, jedoch verwenden sie eine Heuristik auf Basis *Genetischer Algorithmen (GA)* zur Lösung des Optimierungsproblems. Die Struktur des zusammengesetzten Web Services darf die Kontrollkonstrukte Sequenz,

[33] Eine niedrige Serverlast stellt für Yu und Lin einen wichtigen Aspekt dar um die Clients mit einer hohen Dienstgüte versorgen zu können. In [76] stellen sie daher die Algorithmen HQ und RQ zur Ressourcenallokation vor, die es Servern in der QWCS-Architektur ermöglicht Ressourcen so auf die anfragenden Clients zu verteilen, dass zugleich die Serverlast als auch die Anzahl der nötigen Rekonfigurationen zur dienstgüteadäquaten Versorgung aller Clients minimiert wird. Hierdurch sollen Schwankungen bzgl. netzwerkzentrischer Dienstgüteeigenschaften vermieden werden können, mit denen ein Server anfragende Clients versorgt.

bedingte Verzweigung, Schleife und Parallelausführung enthalten. Die verschiedenen Kontrollkonstrukte werden durch eine jeweils unterschiedliche Aggregation der Gesamtparameterwerte berücksichtigt, wodurch eine Betrachtung einzelner Ausführungspfade entfällt. Berücksichtigte Dienstgütekriterien sind Ausführungszeit, Kosten, Verfügbarkeit und Zuverlässigkeit. Eine Erweiterung der Dienstgütekriterien ist möglich. Hierzu ist für zusätzliche Dienstgütekriterien eine Aggregationsfunktion für jedes mögliche Kontrollkonstrukt zu definieren. Da das Optimierungsproblem nicht als (G)LOP formuliert wird, müssen weder die Aggregationsfunktionen, noch die Zielfunktion oder die zu beachtenden Restriktionen linear sein, was eine freizügigere Modellierung der Dienstgütekriterien zulässt. Die zu optimierende Zielfunktion kombiniert die normierten und gewichteten Gesamtparameter eines Ausführungsplans in einer nicht linearen Art und Weise zu einem Nutzenwert. Da genetische Algorithmen bei der Optimierung nicht direkt Restriktionen berücksichtigen, geht die Summe der Abweichung von der Erfüllung definierter Restriktionen als hoch gewichtetes Strafmass in die Zielfunktion ein. Die Optimierung des Zielfunktionswertes führt daher implizit auch zu einer bestmöglichen Erfüllung der Restriktionen. Als Gen, bzw. Individuum wurde im GA eine Array gewählt, welches, analog des in diesem Technical Report vorgestellten Ausführungsplans, zu jedem abstrakt definierten Web Service in der Struktur des zusammengesetzten Web Services den Index des konkret verwendeten Web Services enthält. Zur Optimierung wird eine gewisse Population an Individuen erzeugt und durch Kreuzungs- und Mutationsoperationen verändert, bzw. vermehrt. Die Individuen mit den besten Zielfunktionswerten überleben und bilden die nächste Generation. Dieses Vorgehen wird solange wiederholt bis ein Abbruchkriterium erfüllt wird. Canfora et al. schlagen mehrere alternative Abbruchkriterien vor, wie z.B. das Erreichen einer maximalen Anzahl von Iterationen, das Erfüllen aller Restriktionen oder die Unveränderlichkeit des Zielfunktionswertes des besten Individuums über eine gewisse Anzahl von Iterationen hinweg. In der Analyse des GA haben Canfora et al. die Laufzeit und Lösungsgüte mit der Lösung eines entsprechenden GLOP verglichen. Um die gleiche Lösungsgüte wie die des GLOP zu erreichen benötigt der GA bei wenigen Kandidaten pro Web Service eine höhere Laufzeit. Steigt jedoch die Anzahl der Kandidaten pro Web Service an, so steigt die Laufzeit zur Lösung des GLOP exponentiell, wohingegen sich die Laufzeit des GA als nahezu konstant darstellt, da sie hauptsächlich durch die Anzahl der abstrakten Web Services in der Struktur des zusammengesetzten Web Services determiniert wird, nicht jedoch durch die Anzahl der konkreten Kandidaten für diese abstrakten Web Services. Die Problematik der Überwachung der Dienstgüte zur Ausführungszeit und des Einleitens und Durchführens eines ggf. nötigen Replannings wird von Canfora et al. ebenso berücksichtigt und in einer weiteren Arbeit [21] ausführlich diskutiert. Zur architektonischen Umsetzung wird eine Proxy-Architektur vorgeschlagen, die der Architektur von WSQoSX [60] sehr ähnlich ist.

Aggarwal et al. stellen in [4] ein „*Constraint Driven Web Service Composition Tool*“ für das Framework METEOR-S[34] vor. Mittels eines „Abstract Process Designer“ ist es möglich BPEL4WS-Prozesse zu definieren, die keine konkreten Web Services, sondern semantische Beschreibungen benötigter Web Services in Form von „Service Templates“ enthalten. Ein Service Template beschreibt den benötigten Web Service

[34] Das Framework METEOR-S (METEOR for Semantic Web Services) fokussiert die semantische Beschreibung von Web Services und daraus komponierter Prozesse durch die Erweiterung von Standards durch semantische Annotationen und stellt eine entsprechende Laufzeitumgebung zur Ausführung solcher Web Services und Prozesse bereit. Weitere Informationen können der Homepage des Projekts entnommen werden: http://lsdis.cs.uga.edu/projects/meteor-s/

mittels Ontologien in Bezug auf die zu verarbeitenden Daten, die durchzuführende Funktionalität und gewünschte nicht-funktionale Eigenschaften (z.B. die gewünschte Dienstgüte). Informationen über verfügbare, konkrete Web Services werden mit entsprechenden semantischen Zusatzinformationen in ein erweitertes UDDI-Repository abgelegt. Die „Discovery Engine" ist in der Lage zu den Service Templates der abstrakten Prozessbeschreibung die Menge der geeigneten Web Services aus dem erweiterten UDDI-Repository zu ermitteln. Der „Constraint Analyser" ist in der Lage aus der Menge der geeigneten Web Services konkrete Web Services derart auszuwählen, dass definierte „Business Constraints" und „Process Constraints" bestmöglich erfüllt werden. Business Constraints definieren Präferenzen die den Prozess inhaltlich betreffen, z.B. welcher Anbieter bevorzugt werden sollte, falls im Verlauf des Prozesses ein bestimmter Warentyp gekauft werden sollte. Process Constraints definieren Anforderungen an die nicht-funktionalen Eigenschaften des Prozesses (wie z.B. Ausführungszeit, Kosten, Verlässlichkeit und Verfügbarkeit), die aus den nicht-funktionalen Eigenschaften der zur Durchführung des Prozesses verwendeten Web Services aggregiert werden. Zur Berechnung der nicht-funktionalen Eigenschaften stehen Standard-Aggregationsfunktionen (z.B. Addition, Produkt, Minimal-/Maximaloperation) zur Verfügung, es können jedoch auch spezifische Metriken für nicht-funktionale Eigenschaften definiert werden, denen ebenso spezifische Aggregationsfunktionen zugewiesen werden können. Die Anforderungen an die nicht-funktionalen Eigenschaften des Prozesses werden als ganzzahliges lineares Optimierungsproblem (GLOP) formuliert, dessen Zielfunktion aus einer Linearkombination der numerisch ausgedrückten nicht-funktionalen Eigenschaften des Prozesses besteht. Das GLOP wird mittels des Solvers LINDO[35] gelöst um eine Menge zulässiger und möglichst optimaler Realisierungen des Prozesses zu bestimmen. Wie es hierbei jedoch gelingt individuell definierbare Aggregationsfunktionen für alle in einem BPEL4WS-Geschäftsprozess möglichen Kontrollkonstrukte angeben und automatisiert in ein lineares Optimierungsproblem übertragen zu können, wird nicht näher erläutert. Angaben zur Laufzeit, die für eine derartige Optimierung nötig ist, finden sich ebensowenig. Da beschrieben wird, dass der Prozessdesigner nach der Optimierung eine Auswahl aus der Menge zulässiger Prozessrealisierungen treffen kann, ist davon auszugehen, dass diese Art der Erstellung von optimierten Ausführungsplänen von Prozessen nicht primär dafür vorgesehen ist, voll automatisch zur Laufzeit in einer BPE eingesetzt werden zu können. Durch den „Binder" können die, in Form von Service Templates abstrakt definierten Web Services im BPEL4WS-Prozess jedoch automatisch durch die konkret gewählten Web Services ersetzt werden und an die BPE BPWS4J[36] übergeben werden. Eine vollständige Automatisierung wäre daher denkbar. Hinweise dazu, ob die nicht-funktionalen Eigenschaften der verwendeten Web Services während der Prozessausführung durch eine Monitoring-Komponente überwacht und protokolliert werden und ob ein automatisches Replanning durchgeführt werden kann, finden sich nicht.

Cardoso et al. widmen sich in [22] sehr ausführlich der Thematik die nicht-funktionalen Eigenschaften eines aus Komponenten orchestrierten Prozesses zu bestimmen. Hierzu definieren sie ein Dienstgütemodell aus den drei nicht-

[35] Nähere Informationen zu LINDO können der Homepage des Herstellers entnommen werden: http://www.lindo.com/

[36] Die „Business Process Execution Language for Web Services Java Run Time" (BPWS4J) ist eine BPE für mittels BPEL4WS spezifizierte Geschäftsprozesse. Weitere Informationen können der entsprechenden Projektseite des Herstellers IBM (alphaWorks) unter folgender URL entnommen werden: http://www.alphaworks.ibm.com/tech/bpws4j

funktionalen Eigenschaften Zeit, Kosten und Zuverlässigkeit, durch die sich die Komponenten (Tasks) eines Prozesses näher beschreiben lassen. Zur Definition eines Prozesses können sechs Kontrollkonstrukte verwendet werden: Sequentielle Ausführung, parallele Ausführung (in zwei Varianten), bedingte Verzweigung, Schleife und Unterprozess. Die nicht-funktionalen Eigenschaften der Tasks, sowie die Wahrscheinlichkeiten mit denen im Prozess von Task zu Task verzweigt wird, sind mit Annahmen über das zu erwartende Minimum, Maximum, den Durchschnitt, sowie einer Verteilungsannahme zu initialisieren. Anschließend können die nicht-funktionalen Eigenschaften durch einen als *„Stochastic Workflow Reduction" (SWR)* bezeichneten Algorithmus berechnet werden. Der Algorithmus reduziert den Prozess schrittweise durch Anwendung von sechs Reduktionsoperationen, die Inverse der zur Konstruktion erlaubten Kontrollkonstrukte darstellen, auf einen einzigen atomaren Task. Bei jeder Reduktion erfolgt die Aggregation der nicht-funktionalen Eigenschaften der Tasks eines Kontrollkonstrukts, bis schließlich dem verbleibenden atomaren Task die aggregierten, nicht-funktionalen Eigenschaften des gesamten Prozesses zugewiesen werden. Während der Ausführung eines Prozesses werden die tatsächlichen nicht-funktionalen Eigenschaften der Tasks, sowie die tatsächlichen Durchlaufpfade durch den Prozess durch eine Monitoring-Komponente protokolliert. Mittels dieser Informationen werden die Annahmen über die nicht-funktionalen Eigenschaften der Tasks, sowie die Wahrscheinlichkeiten mit denen im Prozess von Task zu Task verzweigt wird, aktualisiert. Cardoso et al. postulieren, dass hierdurch die Schätzung der nicht-funktionalen Eigenschaften eines Prozesses zunehmend genauer wird, jedoch präsentieren sie nur Ergebnisse, die sie aus lediglich zehn Ausführungen eines Prozesses in einer gemäß ihres Ansatzes erweiterten Variante des METEOR-Systems[37] gewonnen haben. Die Thematik der Auswahl von Komponenten, gemäß den Anforderungen an die nicht-funktionalen Eigenschaften eines Prozesses, wird nicht angesprochen. Cardoso et al. beschränken sich auf einen Ansatz, der es ermöglicht die nicht-funktionalen Eigenschaften eines Prozesses, bestehend aus fest definierten Komponenten, möglichst präzise vorhersagen zu können.

Hwang et al. stellen in [34] ein Dienstgütemodell vor, in welchem Dienstgütewerte als Ausprägung einer Zufallsvariablen aufgefasst werden und Schätzungen der Dienstgüte von Prozessen durch die Aggregation von Verteilungsannahmen der Zufallsvariablen gewonnen werden. Ein Dienstgütekriterium wird als diskrete Zufallsvariable modelliert. Der Wertebereich einer Zufallsvariablen wird in Intervalle unterteilt und die Verteilungsannahme in Form einer Wahrscheinlichkeitsfunktion (*probability mass function, PMF*) definiert, die jedem Intervall die Wahrscheinlichkeit zuordnet mit der eine Ausprägung der Zufallsvariablen in ihm liegt. Durch das Monitoring von Dienstgüte bei der Ausführung von Web Services und einer Auswertung der Aufzeichnungen, ist es möglich automatisch empirische PMFs für einzelne Dienstgütekriterien zu ermitteln. Hwang et al. betrachten konkret die Dienstgütekriterien Antwortzeit, Zuverlässigkeit, Genauigkeit und Nutzungskosten. Zur Orchestration von Prozessen aus Web Services können fünf Kontrollkonstrukte verwendet werden: Sequenz, Parallelausführung (in zwei Varianten), bedingte Verzweigung und Schleife. Für jede Kombination aus Kontrollkonstrukt und Dienstgütekriterium wird eine Aggregationsfunktion definiert, die festlegt, wie das Dienstgütekriterium aller durch das Kontrollkonstrukt eingeschlossenen Aktivitäten zu einer Verteilungsannahme des Dienstgütekriteriums für das gesamte Kontrollkonstrukt aggregiert werden kann. Eine Aktivität

[37] METEOR (Managing End-To-End OpeRations) ist die Bezeichnung der Vorgängerversion, des von Aggarwal et al. (s.o.) erweiterten Systems METEOR-S. Weitere Informationen können der Projekthomepage entnommen werden: http://lsdis.cs.uga.edu/projects/past/METEOR/

kann hierbei entweder ein Web Service oder ein weiteres Kontrollkonstrukt sein. Die Verteilung eines Dienstgütekriteriums für den gesamten Prozess, lässt sich durch rekursive Anwendung der Aggregationsfunktionen erzeugen. Bei der Aggregation der Verteilungen kann sich die Anzahl der zu betrachtenden Intervalle in der diskreten Verteilung exponentiell erhöhen, was die nötige Dauer einer Berechnung schnell sehr stark ansteigen lassen würde. Nach jeder Aggregation wird daher die Anzahl der Intervalle, der sich ergebenden diskreten Verteilung, wieder reduziert. Um eine Reduktion zu erreichen, muss der Wertebereich so in eine festgelegte Anzahl von Intervallen partitioniert werden, dass die Abweichung zwischen der sich ergebenden neuen Verteilung (mit reduzierter Intervallzahl) und der ursprünglichen Verteilung (mit der ursprünglichen Intervallzahl) minimiert wird. Hwang et al. stellen einen Algorithmus auf Basis der dynamischen Optimierung vor, der das Optimierungsproblem mit exponentieller Laufzeit exakt löst, sowie eine Heuristik, die das Optimierungsproblem in logarithmischer Laufzeit näherungsweise löst. In Experimenten zeigen sie, dass es sowohl dem exakten als auch dem heuristischen Algorithmus gelingt, die tatsächliche Verteilung der Dienstgütekriterien eines Prozesses sehr gut anzunähern, d.h. die Dienstgüte des Prozesses sehr genau vorherzusagen. Die Heuristik ist hierbei geringfügig schlechter, besitzt jedoch einen enormen Laufzeitvorteil. Die Orchestration eines Prozesses aus Web Services, gemäß festgelegter Dienstgüteanforderungen, wird nicht behandelt. Hwang et al. beschränken sich auf einen Ansatz, um die Dienstgüte eines Prozesses, bestehend aus fest definierten Komponenten, möglichst präzise auf der Basis von empirisch gewonnenen Verteilungsannahmen vorhersagen zu können.

Maximilien und Singh stellen in [48] das *„Web Service Agent Framework" (WSAF)* vor, welches die Vertrauenswürdigkeit der Aussagen von Providern über die Dienstgüte ihrer angebotenen Web Services durch einen Austausch von Informationen über die tatsächlich von den Web Services erbrachte Dienstgüte durch Agenten berücksichtigt. Web Services werden im WSAF von den Nutzern nicht direkt aufgerufen, sondern indirekt über Agenten, die als Proxy dem Nutzer dieselbe Schnittstelle zur Verfügung stellen, jedoch eine dynamische Auswahl des konkret auszuführenden Web Services vornehmen. Die Auswahl eines konkreten Web Services erfolgt anhand von Dienstgüteeigenschaften, die mittels einer Ontologie definiert werden können. Ein Provider definiert sein Dienstgüteangebot und ein Nutzer seine Dienstgüteanforderungen in Form einer „Policy" auf Basis der definierten Dienstgüteontologie. Damit der Agent einen konkreten Web Service aufrufen kann, wird ein Matching der Policies von Provider und Nutzer durchgeführt um für den Nutzer einen möglichst geeigneten Web Service ermitteln zu können. Bei dieser Auswahl wird neben den Informationen der Policy des Providers zusätzlich berücksichtigt, welche Dienstgüte der Web Service in der Vergangenheit tatsächlich erbracht hat. Je stärker die Abweichung von der durch eine Policy versprochenen Dienstgüte in der Vergangenheit war, desto weniger Vertrauen wird dieser Policy geschenkt. Bei einem Aufruf des Web Services protokolliert der Agent die tatsächlich erbrachte Dienstgüte des Web Services und berücksichtigt sie bei zukünftigen Entscheidungen. Um die Verbreitungsgeschwindigkeit der Vertrauensinformationen bzgl. der Policies zu steigern, können Agenten über sog. „Agencies" Informationen über die erbrachte Dienstgüte von Web Services mit anderen Agenten austauschen. Eine Agency verwaltet diese Informationen und ermöglicht es so, Agenten bei der Auswahl von Web Services an den Erfahrungen anderer Agenten partizipieren zu lassen. In einer Simulation verfälschen Maximilien und Singh die von Providern angebotenen Dienstgüten und lassen lediglich einen Provider die versprochene Dienstgüte erbringen. Sie zeigen, dass der Aufruf eines Web Services über einen Agenten mit steigender Anzahl von Aufrufen zunehmend präziser

den Web Service wählt, dessen Angebot unverfälscht, d.h. vertrauenswürdig ist. Bei einem Austausch von Informationen von mehreren Agenten über eine Agency konvergiert die Auswahl schneller zu dem vertrauenswürdigen Web Service. Durch den Einsatz des WSAF kann eine optimierte Auswahl von Web Services auf der Basis einzelner Prozessschritte in einem Geschäftsprozess erreicht werden, jedoch ist es nicht möglich einen Ausführungsplan für den gesamten Geschäftsprozess unter Einhaltung globaler Restriktionen und Optimierungskriterien zu erstellen. Der Ansatz liefert jedoch Informationen darüber, wie es möglich wird Informationen über die Vertrauenswürdigkeit eines Anbieters in ein Dienstgütemodell zu integrieren und entsprechende Informationen unter den Nutzern zu propagieren.

Liu et al. sprechen in [46] Empfehlungen aus, welche Eigenschaften für ein Dienstgütemodell erstrebenswert sind, aus welchen Quellen Dienstgüteinformationen bezogen werden können und wie die Berechnung eines Wertes erfolgen kann, der den Grad der Übereinstimmung der Dienstgüte eines Web Service mit den Dienstgüteanforderungen eines Nutzers ausdrücken kann. Ein Dienstgütemodell sollte nach Liu et al. nicht auf einige Dienstgütekriterien beschränkt sein, sondern um beliebige Dienstgütekriterien erweitert werden können. Auf Basis des Dienstgütemodells muss es für Nutzer möglich sein ihre Anforderungen an die Dienstgüte eines Web Services präzise zu formulieren, damit der Grad der Eignung eines Web Services bestimmt werden kann, der zur Auswahl aus einer Menge von alternativen Web Services herangezogen werden kann. Als wichtigste Informationsquellen für Dienstgüteinformationen werden gesehen: Übermittlung von Dienstgüteeigenschaften durch den Web-Service-Provider, Ermittlung von Dienstgüteeigenschaften durch ein Monitoring während der Nutzung des Web Services, sowie Feedback der Nutzer über die erbrachte Dienstgüte nach der Nutzung des Web Services. Ein Verzeichnis von Web Services, welches Dienstgüteinformationen berücksichtigt, sollte alle drei Informationsquellen unterstützen und sowohl Web-Service-Providern, Monitoring-Komponenten als auch Web-Service-Nutzern entsprechende Schnittstellen zur Aktualisierung der Dienstgüteinformationen bereitstellen. Unter den Dienstgütekriterien unterscheiden Liu et al. zwischen generischen Kriterien, die in Bezug auf alle Web Services verwendet werden können, und geschäfts-, bzw. domänenspezifische Kriterien mit besonderer Relevanz im spezifischen Kontext der Nutzung. Als Beispiele für generische Kriterien werden die Kosten der Ausführung (sollten durch den Provider übermittelt werden), die benötigte Ausführungszeit (sollte durch Monitoring gemessen werden) und die Reputation des Anbieters (sollte durch Feedback der Nutzer ermittelt werden) genannt. Zur Bestimmung des Grades der Übereinstimmung der Dienstgüte, die durch einen Web Service angeboten wird, mit den Dienstgüteanforderungen eines Nutzers, wird die Berechnung einer Maßzahl vorgeschlagen. Vor der Berechnung dieser Maßzahl werden Dienstgütekriterien zu Gruppen zusammengefasst und für jede Gruppe ein Gewicht definiert, welches die Wichtigkeit der Gruppe von Dienstgütekriterien für den Nutzer ausdrückt. Zur Berechnung der Maßzahl wird zunächst eine Normalisierung der Dienstgütewerte eines Web Services anhand des Durchschnitts aller Dienstgütewerte und einem definiertem Maximum durchgeführt. Die normalisierten Dienstgütewerte werden gemäß den definierten Gruppen aggregiert und erneut anhand der Gruppendurchschnitte und eines weiteren definierbaren Maximums normalisiert. Die Maßzahl ergibt sich als gewichtete Summe der normalisierten Dienstgütegruppenwerte. Die Maßzahl ist je höher, je besser die mittels der definierten Gewichte ausgedrückten Anforderungen des Nutzers erfüllt werden. Hierdurch läßt sich ein Ranking der Web Services nach deren Eignung zur Erfüllung der Nutzeranforderungen durchführen. Ein Web-Service-Verzeichnis, welches Dienstgüteinformationen berücksichtigt, sollte

dem Nutzer eine entsprechende Schnittstelle bieten um eine solches Ranking von Web Services abrufen zu können. Durch das von Liu et al. vorgeschlagene Ranking von Web Services anhand einer Maßzahl, ist es möglich eine optimierte Auswahl von Web Services auf der Basis einzelner Prozessschritte in einem Geschäftsprozess zu erreichen. Es ist jedoch nicht möglich einen Ausführungsplan für einen gesamten Geschäftsprozess unter Berücksichtigung globaler Restriktionen und Optimierungskriterien zu erstellen.

Chang et al. bemängeln in [23], dass bei der Vorhersage der Dienstgüte von Web Servicen und aus diesen orchestrierten Prozessen oft von zu statischen Annahmen, bzw. Szenarien ausgegangen wird. Sie schlagen vor, durch Simulationen der Verhaltensweisen von Web Servicen und Prozessen realistischere Aussagen über die erreichbare Dienstgüte zu treffen. Anbieter von Web Servicen können hierdurch das Verhalten ihrer Web Services unter bestimmten Bedingungen (z.B. Entwicklung der Antwortzeit beim Ansteigen der Anzahl gleichzeitiger Nutzer) besser abschätzen und durch die Analyse relevanter Szenarien präzisere Aussagen, z.B. in Form von SLAs, bzgl. der Dienstgüte ihrer Web Services geben. Nutzer können mithilfe dieser Informationen einerseits das Verhalten von aus Web Servicen orchestrierten Prozessen analysieren und andererseits durch Variation der verwendeten Web Services Entscheidungen über die Auswahl konkreter Web Services treffen, die in einem Prozess verwendet werden sollen. Zur Lösung komplexerer Problemstellungen, wie z.B. der Auswahl konkreter Web Services zur Einhaltung gewisser Restriktionen bei gleichzeitiger Optimierung von Dienstgütekriterien, schlagen sie die Formulierung und Lösung ganzzahliger Optimierungsprobleme vor. Sie erkennen jedoch, dass zum Lösen derartiger Optimierungsprobleme ein hoher Rechenzeitbedarf entsteht, der ihren Einsatz höchstens zur Entwurfszeit eines Prozesses ermöglicht, nicht jedoch um eine Auswahl dynamisch zur Laufzeit einer BPE durchzuführen. Sie schlagen vor statt exakter Lösungsmethoden entsprechende Heuristiken zur näherungsweisen Lösung des Problems zu verwenden, ohne jedoch hierauf näher einzugehen. Zur Unterstützung der Simulation von Prozessen skizzieren sie ein Framework, welches es ermöglichen soll, einen Prozess graphisch zu entwerfen und hieraus zugleich eine Prozessdefinition in einer konkreten Beschreibungssprache (z.B. BPEL4WS) zu erstellen, als auch ein Simulationsmodell, welches unter verschiedenen Szenarien mittels eines Analyse-Tools untersucht werden kann. Wie dies technisch umgesetzt werden soll, wie Szenarien zur Analyse definiert werden können, aus welchen Quellen Dienstgüteinformationen stammen, wie diese zur Dienstgüte von Prozessen aggregiert werden können und wie Ergebnisse verarbeitet und genutzt werden können, wird jedoch nicht näher erläutert.

Die von Saaty [55] entwickelte Methode ***AHP (Analytic Hierarchy Process)*** zur Entscheidungsunterstützung, kann zu einer feingranularen Beurteilung von Alternativen anhand einer Vielzahl von Kriterien, bzw. Zielen verwendet werden. Eine derartige Beurteilung von Alternativen kann auch bei der Auswahl von Softwarekomponenten verwendet werden (siehe z.B. Kontio [42] und Becker [10]). Die Durchführung von AHP bedingt den paarweisen Vergleich aller Alternativen. Im Falle einer lokalen Optimierung eines Geschäftsprozesses, bei welcher für einen einzelnen Prozessschritt ein optimaler Web Service aus einer Menge von Alternativen zu wählen ist, lässt sich ein solcher paarweiser Vergleich noch durchführen. Zur Bestimmung eines optimalen Gesamtausführungsplans lässt sich AHP jedoch nicht mehr einsetzen, da dies die Enumeration aller möglichen Ausführungspläne und deren paarweisen Vergleich bedingen würde. Dies kann Aufgrund der Menge an alternativen Ausführungsplänen nicht mehr in einer praktikablen Zeit erfolgen. Becker erkennt dies in [10] (S. 69) und

führt aus: „Die meisten der in der Literatur auffindbaren Prozesse betrachten die Auswahl einer einzelnen Komponente. Dies ist jedoch eine Betrachtung, die an den Problemstellungen der Realität vorbeigeht. Dort ist es nämlich meist notwendig, ganze Konfigurationen auszuwählen, [...] daher bestehen die Alternativen des Entscheidungsproblems dann in Konfigurationen von Komponenten [...]. Die Menge an Alternativen kann dabei - je nach Generierungsmethode - so groß werden, dass der Einsatz eines Entscheidungsverfahrens mit paarweisen Vergleichen (wie dem AHP) nicht mehr praktikabel ist.“

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

In einer Umgebung, in der Geschäftsprozesse auf Basis von Web Services ausgeführt werden und in der dieselbe Funktionalität durch eine Vielzahl alternativer Web Services zur Verfügung gestellt wird, stellt sich die Frage, welche dieser Web Services verwendet werden sollen um einen Geschäftsprozess auszuführen. Eine zielgerichtete Auswahl kann erfolgen, falls festgelegt wird, hinsichtlich welcher Präferenzen eine Auswahl durchgeführt werden soll und welche Restriktionen es bei der Auswahl zu beachten gilt. Ein wichtiges Differenzierungsmerkmal von Web Services stellen ihre nicht-funktionalen Eigenschaften (z.B. ihre Dienstgüte) dar. Werden die nicht-funktionalen Eigenschaften aller alternativen Web Services beschrieben, sowie die Präferenzen und Restriktionen hinsichtlich der gewünschten nicht-funktionalen Eigenschaften des Geschäftsprozesses formuliert, so ist eine automatische Auswahl von Web Services zur Ausführung eines Geschäftsprozesses möglich. Das Auswahlproblem kann als Optimierungsproblem aufgefasst werden, Web Services anhand ihrer nicht-funktionalen Eigenschaften derart auszuwählen, dass alle bzgl. eines Geschäftsprozesses formulierten Restriktionen eingehalten und zugleich alle formulierten Präferenzen bestmöglich erfüllt werden. Damit ein solches NP-schweres Optimierungsproblem unmittelbar vor der Ausführung eines Geschäftsprozesses zur Laufzeit gelöst werden kann, ist es nötig eine effiziente Heuristik zu entwickeln, damit die benötigte Rechenzeit auf ein praktikables Maß reduziert werden kann.

Im Rahmen unserer Forschungsarbeiten wurde ein mehrstufiges, heuristisches Optimierungsverfahren entwickelt, welches es ermöglicht, zu einer aus abstrakten Diensten orchestrierten Geschäftsprozessdefinition, eine derartige Zuordnung konkreter Web Services zu abstrakten Diensten (Ausführungsplan) zu finden, dass die Ausführung des Geschäftsprozesses mittels der zugeordneten Web Services zu einer, hinsichtlich nicht-funktionaler Eigenschaften optimierten, Erbringung der Funktionalität des Geschäftsprozesses führt (z.B. unter Minimierung der benötigten Ausführungszeit). Wie stark hierbei nach welchen nicht-funktionalen Eigenschaften optimiert wird, kann über Gewichte frei definiert werden. Zusätzlich können die nicht-funktionalen Eigenschaften der Geschäftsprozessausführung Restriktionen unterworfen werden (z.B. Beschränkung der verursachten Kosten), wodurch es möglich wird Garantien über nicht-funktionale Eigenschaften des Geschäftsprozesses, z.B. in Form von SLAs, aussprechen zu können. Die nicht-funktionalen Eigenschaften, die sich durch die Ausführung des Geschäftsprozesses ergeben (Gesamtparameterwerte), lassen sich durch Aggregation der nicht-funktionalen Eigenschaften (Parameterwerte) der zur Ausführung ausgewählten Web Services berechnen.

Funktional gleiche Web Services werden in Kategorien gruppiert und stellen Alternativen zur Erbringung einer, in der Geschäftsprozessdefinition mittels eines abstrakten Dienstes definierten, Funktionalität dar. Zur Differenzierung funktional gleicher Web Services kann eine beliebige Anzahl nicht-funktionaler Eigenschaften (Parameter) definiert werden, wobei zur Definition jedes Parameters aus einem von drei Grundtypen (Parametertyp) gewählt werden kann (additiver Parameter, multiplikativer Parameter und Minimaloperator-Parameter). Der Parametertyp entscheidet über die Art der Aggregation der einzelnen Parameterwerte der Web Services zu einem Gesamtparameterwert des Ausführungsplans.

Aus Gründen der Vereinfachung werden momentan nur solche Geschäftsprozesse berücksichtigt, die abstrakte Dienste mittels des Kontrollkonstrukts der sequentiellen Ausführung orchestrieren (sequentielle Geschäftsprozesse). Die Optimierung von Geschäftsprozessen mit weiteren Kontrollkonstrukten, lässt sich jedoch durch Anwendung geeigneter Verfahren (vgl. z.B. [79] und [80]) auf die Optimierung einzelner sequentieller Teilprozesse zurückführen.

Das Problem, zu einem sequentiellen Geschäftsprozess einen Ausführungsplan zu erstellen, der zugleich alle spezifizierten Restriktionen einhält, als auch hinsichtlich aller spezifizierten Optimierungskriterien optimal ist, ist ein NP-schweres Optimierungsproblem. Um das Optimierungsproblem trotzdem effizient lösen zu können, wird es als gemischt-ganzzahliges lineares Optimierungsproblem formuliert und durch eine vierstufige Heuristik näherungsweise gelöst:

1. Schritt: Exakte Lösung des LP-relaxierten Optimierungsproblems durch einen Simplex-Algorithmus.
2. Schritt: Konstruktion einer ersten zulässigen Lösung aus den Ergebnissen des ersten Schritts durch einen Backtracking-Algorithmus.
3. Schritt: Anwendung eines reinen Verbesserungsverfahrens zur weiteren (lokalen) Optimierung der ersten zulässigen Lösung.
4. Schritt: Anwendung der Metaheuristik Simulated Annealing zur weiteren (möglichst globalen) Optimierung der bisherigen Lösung.

Ein mittels der Heuristik erzeugter Ausführungsplan kann während seiner Ausführung solange als näherungsweise optimal angesehen werden, bis Erkenntnisse vorliegen, die gegen seine Optimalität sprechen. Beispiele hierfür sind z.B. die Nichterreichbarkeit eines verplanten Web Services oder geänderte Annahmen über seine nichtfunktionalen Eigenschaften, die signifikant von jenen abweichen, von welchen bei der Erstellung des Ausführungsplans durch die Heuristik ursprünglich ausgegangen wurde. In diesem Fall ist eine Revidierung des Ausführungsplans durchzuführen und es ist für den noch zur Ausführung ausstehenden Teils des Geschäftsprozesses ein neuer Ausführungsplan unter Einbeziehung der neuen Erkenntnisse zu erstellen. Zur Durchführung eines solchen sog. Replannings wurde eine Methode entwickelt, das Optimierungsproblem auf den noch nicht ausgeführten Teil des Geschäftsprozesses zu reduzieren und einen neuen Ausführungsplan durch Anwendung der Heuristik auf das Optimierungsproblem reduzierter Problemgröße zu gewinnen. Durch dieses Vorgehen sinkt der Aufwand eines Replannings mit zunehmender Vollständigkeit der Ausführung eines Geschäftsprozesses.

Die Heuristik wurde zum Zwecke ihrer Evaluation in Form eines Prototypen implementiert. Anhand des Prototyps wurde eine Analyse der Heuristik durchgeführt, welche den Einfluss der Problemgröße und des Probleminhalts (am Beispiel der Restriktionsstärke) auf die benötigte Laufzeit und die erzeugte Lösungsgüte untersucht. Zu Vergleichszwecken wurde hierbei das gemischt-ganzzahlige Optimierungsproblem jeweils auch exakt durch einen entsprechenden Solver gelöst. Die exakte Lösung des Optimierungsproblems durch den Solver benötigt eine wesentlich höhere Rechenzeit als die näherungsweise Lösung des Optimierungsproblems durch die Heuristik, wobei jedoch die Lösungsgüte in den meisten Fällen nicht wesentlich unter derjenigen des Solvers liegt. Die Analyseergebnisse lassen eine pseudopolynomiell beschränkte Laufzeit der Heuristik vermuten, da in guter Näherung ein polynomieller Zusammenhang von Laufzeit und Problemgröße, aber ein teilweise exponentieller Zusammen-

hang von Laufzeit und Probleminhalt beobachtet werden konnte. Die Laufzeit des Solvers zur exakten Lösung des Optimierungsproblems scheint hingegen bereits exponentiell von der Problemgröße abhängig zu sein. Die Ergebnisse bestätigen nicht nur die Überlegenheit der Heuristik gegenüber der exakten Lösung in Bezug auf die Laufzeit, sie verdeutlichen auch durch die festgestellten enormen Laufzeiten einer exakten Lösung, dass eine Erstellung eines Ausführungsplans zur optimierten Ausführung von Geschäftsprozessen nur dann sinnvoll zur Laufzeit in einer BPE durchgeführt werden kann, wenn das Optimierungsproblem näherungsweise durch eine Heuristik gelöst wird.

7.2 Ausblick

Bei der Entwicklung der Heuristik wurde die Möglichkeit des Replannings explizit berücksichtigt und in Kapitel 3.3 beschrieben wie ein Replanning durch die Anpassung des Optimierungsproblems und die erneute Anwendung der Heuristik auf dieses durchgeführt werden kann. Die Durchführung eines Replannings nimmt eine gewisse Zeit in Anspruch und verzögert hierdurch die weitere Ausführung des Geschäftsprozesses zur Laufzeit. Es stellt sich daher die Frage in welchen Situationen ein Replanning von einer BPE ausgelöst werden sollte, so dass das Verhältnis aus zusätzlich aufgewendeter Berechnungszeit und erreichter Optimalität der Ausführung des Geschäftsprozesses möglichst lohnenswert erscheint. Unsere zukünftigen Forschungsaktivitäten werden daher die Untersuchung von effektiven Replanningstrategien beinhalten. Ein Fokus hierbei wird die Untersuchung des Verhältnisses zwischen zusätzlich aufgewendeter Berechnungszeit für das Replanning und die hierdurch erreichte Genauigkeit der Einhaltung der vorgegebenen Restriktionen sein.

Literaturverzeichnis

[1] *CCITT Blue Book, VIII.2, Data Communication Networks: Services and Facilities, Interfaces*, Band 2, Genf, ITU, 1989.

[2] *lp_solve Reference Guide (5.1.1.3)*, http://lpsolve.sourceforge.net/5.1/, Abruf am 10.10.2005.

[3] Aarts E. H. L., Korst J. H. M.: *Simulated annealing and Boltzmann machines: A stochastic approach to combinatorial optimization and neural computing*, Chichester, New York, Wiley, 1989.

[4] Aggarwal R., Verma K., Miller J., Milnor W.: *Constraint Driven Web Service Composition in METEOR-S*, in: Proceedings of IEEE International Conference on Services Computing (SCC'04), Shanghai, China, 2004, S. 23-30.

[5] Aier S., Schönherr M.: *Enterprise Application Integration – Flexibilisierung komplexer Unternehmensarchitekturen*, Band 1, Berlin, GITO, 2003.

[6] Aier S., Schönherr M.: *Enterprise Application Integration – Serviceorientierung und nachhaltige Architekturen*, Band 2, Berlin, GITO, 2004.

[7] Alonso G., Casati F., Kuno H., Machiraju V.: *Web Services. Concepts, Architectures and Applications.*, Berlin, Heidelberg, Springer, 2004.

[8] Andrews T., Curbera F., Dholakia H., Goland Y., Klein J., Leymann F., Liu K., Roller D., Smith D., Thatte S., Trickovic I., Weerawarana S.: *Business Process Execution Language for Web Services Version 1.1*, ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf, BEA Systems, IBM, Microsoft, SAP AG, Siebel Systems, 05.05.2003, Abruf am 10.10.2005.

[9] Becker J., Kugler M., Rosemann M.: *Process Management: A Guide for the Design of Business Processes*, Berlin, New York, Springer, 2003.

[10] Becker S.: *Konfigurationsmanagement komponentenorientierter betrieblicher Anwendungen*, Diplomarbeit, Technische Universität Darmstadt, Fachgebiet Operations Research, Darmstadt, 2003.

[11] Beimborn D., Franke J., Weitzel T.: *Drivers and Inhibitors for Outsourcing Financial Processes – A Comparative Survey of Economies of Scale, Scope, and Skill*, in: Proceedings of the 11th Americas Conference on Information Systems (AMCIS'05), Omaha, NE, USA, 2005.

[12] Berbner R., Grollius T., Repp N., Heckmann O., Ortner E., Steinmetz R.: *An approach for the Management of Service-oriented Architecture (SoA) based Application Systems*, in: Enterprise Modelling and Information Systems Architectures (EMISA'05), Klagenfurt, Österreich, 2005.

[13] Berbner R., Heckmann O., Mauthe A., Steinmetz R.: *Eine Dienstgüte unterstützende Web Service-Architektur für flexible Geschäftsprozesse*, in: Wirtschaftsinformatik, 47 (2005) 4, Wiesbaden, Vieweg.

[14] Berbner R., Heckmann O., Steinmetz R.: *An Architecture for a QoS driven composition of Web Service based Workflows*, in: Networking and Electronic Commerce Research Conference (NAEC'05), Gardasee, Italien, 2005.

[15] Berbner R., Mauthe A., Steinmetz R.: *Unterstützung dynamischer E-Finance-Geschäftsprozesse*, in: Konferenzband der Konferenz Elektronische Geschäftsprozesse 2004 (EGP'04), Klagenfurt, Österreich, 2004, S. 44-54.

[16] Blum C., Roli A.: *Metaheuristics in combinatorial optimization: Overview and conceptual comparison*, in: ACM Computing Surveys, 35 (2003) 3, New York, ACM Press, S. 268-308.

[17] Boysen N.: *Ameisenalgorithmen*, http://www.ameisenalgorithmus.de/downloads/ameisenalgorithmen.pdf, Abruf am 10.10.2005.
[18] Braden R., Clark D., Shenker S.: *Integrated Services in the Internet Architecture: an Overview*, Request for Comments 1633, http://www.ietf.org/rfc/rfc1633.txt, The Internet Society, Juni 1994, Abruf am 10.10.2005.
[19] Braden R., Zhang L., Berson S., Herzog S., Jamin S.: *Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification*, Request for Comment 2205, http://www.ietf.org/rfc/rfc2205.txt, The Internet Society, Sep. 1997, Abruf am 10.10.2005.
[20] Canfora G., Penta M. D., Esposito R., Villani M. L.: *An approach for QoS-aware service composition based on genetic algorithms*, in: Proceedings of Genetic and Evolutionary Computation Conference (GECCO'05), Washington DC, USA, 2005, S. 1069-1075.
[21] Canfora G., Penta M. D., Esposito R., Villani M. L.: *QoS-Aware Replanning of Composite Web Services*, in: Proceedings of IEEE International Conference on Web Services (ICWS'05), Orlando, FL, USA, 2005, S. 121-129.
[22] Cardoso J., Sheth A., Miller J., Arnold J., Kochut K.: *Quality of Service for Workflows and Web Service Processes*, in: Web Semantics: Science, Services and Agents on the World Wide Web, 1 (2004) 3, Elsevier, S. 281-308.
[23] Chang H., Song H., Kim W., Lee K., Park H., Kwon S., Park J.: *Simulation-Based Web Service Composition: Framework and Performance Analysis*, in: Proceedings of the third Asian Simulation Conference (AsiaSim'04), Jeju Island, Korea, 2004, S. 352-360.
[24] Chen H., Yu T., Lin K.-J.: *QCWS: An Implementation of QoS-Capable Multimedia Web Services*, in: Proceedings of the 5th IEEE International Symposium on Multimedia Software Engineering (ISMSE'03), Taichung, Taiwan, 2003, S. 38.
[25] Domschke W.: *Logistik: Rundreisen und Touren*, Band 2, 4. Aufl., München, Wien, Oldenbourg, 1997.
[26] Domschke W., Drexl A.: *Einführung in Operations Research*, 4 Aufl., Berlin, Heidelberg, New York, Barcelona, Budapest, Hongkong, London, Mailand, Paris, Santa Clara, Singapur, Tokio, Springer, 1998.
[27] Domschke W., Klein R., Scholl A.: *Taktische Tabus. Tabu Search: Durch Verbote schneller optimieren*, in: c't - Magazin für Computer Technik, Heft 12/1996, S. 326-332.
[28] Dyer M. E.: *An O(n) algorithm for the multiple-choice knapsack linear program*, in: Mathematical Programming, 29 (1984), Amsterdam, North-Holland Pub. Co., S. 57-63.
[29] Fröschl F.: *Vom IuK-Outsourcing zum Business Process Outsourcing*, in: Wirtschaftsinformatik, 41 (1999) 5, Wiesbaden, Vieweg, S. 458-460.
[30] Gouscos D., Kalikakis M., Georgiadis P.: *An Approach to Modeling Web Service QoS and Provision Price*, in: Proceedings of the 4th International Conference on Web Information Systems Engineering Workshops (WISEW'03), Rom, Italien, 2003, S. 121-130.
[31] Gupta S.: *Pro Apache Log4j*, 2. Aufl., Berkeley, Apress, 2005.
[32] Hammer M., Champy J.: *Rengineering the Corporation: A Manifesto for Business Revolution*, New York, HarperBusiness, 2003.

[33] Heckmann O.: *A System-oriented Approach to Efficiency and Quality of Service for Internet Service Providers*, Dissertationsschrift, Technische Universität Darmstadt, Fachbereich Informatik, Darmstadt, 2004.
[34] Hwang S.-Y., Wang H., Srivastava J., Paul R. A.: *A Probabilistic QoS Model and Computation Framework for Web Services-Based Workflows*, in: Proceedings of 23rd International Conference on Conceptual Modeling (ER'04), Shanghai, China, 2004, S. 596-609.
[35] Hyde P.: *Java Thread Programming*, Indianapolis, Sams, 1999.
[36] Juric M. B., Mathew B., Sarang P.: *Business Process Execution Language for Web Services: BPEL and BPEL4WS*, Birmingham, Packt Publishing, 2004.
[37] Kalepu S., Krishnaswamy S., Loke S. W.: *Verity: A QoS Metric for Selecting Web Services and Providers*, in: Proceedings of the 4th International Conference on Web Information Systems Engineering Workshops (WISEW'03), Rom, Italien, 2003, S. 131-139.
[38] Karmarkar N.: *A new polynomial-time algorithm for linear programming*, in: Combinatorica, 4 (1984) 4, Secaucus, Springer, S. 373-395.
[39] Keller W.: *Enterprise Application Integration. Erfahrungen aus der Praxis.*, Heidelberg, dpunkt, 2002.
[40] Khachiyan L. G.: *A polynomial algorithm in linear programming.*, in: Soviet Mathematics Doklady (1979) 20, American Mathematical Society, S. 191-194.
[41] Klee V., Minty G. J.: *How good is the simplex algorithm?* in: Inequalities, III, Shisha O. (Hrsg.), New York, Academic Press, 1972, S. 159-175.
[42] Kontio J.: *A case study in applying a systematic method for COTS selection*, in: Proceedings of 18th International Conference on Software Engineering (ICSE'96), Berlin, 1996, S. 201-209.
[43] Laarhoven P. J. M., Aarts E. H. L.: *Simulated annealing: theory and applications*, Norwell, Kluwer Academic, 1987.
[44] Lee K., Jeon J., Lee W., Jeong S.-H., Park S.-W.: *QoS for Web Services: Requirements and Possible Approaches*, W3C Working Group Note, http://www.w3c.or.kr/kr-office/TR/2003/ws-qos/, W3C, 25.11.2003, Abruf am 10.10.2005.
[45] Lindert F., Wiedeler M.: *Organisationsübergreifendes Geschäftsprozessmanagement*, in: IT Information Technology, 46 (2004) 4, München, Oldenbourg, S. 175-183.
[46] Liu Y., Ngu A. H. H., Zeng L.: *QoS Computation and Policing in Dynamic Web Service Selection*, in: Proceedings of 13th international World Wide Web conference on Alternate track papers & posters (WWW'04), New York, NY, USA, 2004, S. 66-73.
[47] Mani A., Nagarajan A.: *Understanding quality of service for Web services*, http://www-106.ibm.com/developerworks/webservices/library/ws-quality.html, IBM developerWorks, 01.01.2002, Abruf am 10.10.2005.
[48] Maximilien E. M., Singh M. P.: *Toward Autonomic Web Services Trust and Selection*, in: Proceedings of International Conference On Service Oriented Computing (ICSOC'04), New York, NY, USA, 2004, S. 212-221.
[49] Michalewicz Z.: *Genetic algorithms + data structures = evolution programs*, 2. Aufl., Berlin, New York, Springer, 1994.
[50] Papadimitriou C. H., Steiglitz K.: *Combinatorial Optimization: Algorithms and Complexity*, Englewood Cliffs, Prentice-Hall, 1982.
[51] Pfeifle J.: *Extremal Constructions for Polytopes and Spheres*, Dissertationsschrift, Technischen Universität Berlin, Fakultät II, Mathematik und Naturwissenschaften, Berlin, 2003.

[52] Pisinger D.: *A minimal algorithm for the Multiple-choice Knapsack Problem*, in: European Journal of Operational Research, 83 (1995) 2, Amsterdam, North-Holland Pub. Co., S. 394-410.
[53] Ran S.: *A model for web services discovery with QoS*, in: ACM SIGecom Exchanges, 4 (2003) 1, New York, ACM Press, S. 1-10.
[54] Riedl R.: *Begriffliche Grundlagen des Business Process Outsourcing*, in: Information Management & Consulting, 18 (2003) 3, Saarbrücken, mc information multimedia communication, S. 6-10.
[55] Saaty T. L.: *The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation*, New York, London, McGraw-Hill International Book Co., 1980.
[56] Salkin H. M.: *Integer programming*, Reading, Addison-Wesley, 1975.
[57] Schmitt J. B.: *Heterogeneous Network Quality of Service Systems*, Boston, Kluwer Academic, 2001.
[58] Shenker S., Partridge C., Guerin R.: *Specification of Guaranteed Quality of Service*, Request for Comments 2212, http://www.ietf.org/rfc/rfc2212.txt, The Internet Society, Sep. 1997, Abruf am 10.10.2005.
[59] Sokolovsky Z., Löschenkohl S.: *Handbuch Industrialisierung der Finanzwirtschaft*, Wiesbaden, Gabler, 2005.
[60] Spahn M., Berbner R., Heckmann O., Mauthe A., Steinmetz R.: *WSQoSX – Eine dienstgütebasierte Web-Service-Architektur*, Technische Universität Darmstadt, Multimedia Communications Lab, Darmstadt, Technical Report KOM-TR-2004-07, 2004.
[61] Steinmetz R.: *Multimedia-Technologie: Grundlagen, Komponenten und Systeme*, 3. Aufl., Berlin, Heidelberg, New York, Barcelona, Hongkong, London, Mailand, Paris, Singapur, Tokio, Springer, 2000.
[62] Steinmetz R., Berbner R., Martinovic I.: *Web Services zur Unterstützung flexibler Geschäftsprozesse in der Finanzwirtschaft*, in: Handbuch Industrialisierung der Finanzwirtschaft, Sokolovsky Z. und Löschenkohl S. (Hrsg.), Wiesbaden, Gabler, 2004.
[63] Sturm R., Morris W.: *Foundations of Service Level Management*, Indianapolis, Sams, 2000.
[64] Stützle T., Hoos H. H.: *Ameisenalgorithmen zur Lösung kombinatorischer Optimierungsprobleme*, in: KI, 01 (2001) 1, Böttcher IT, S. 45-51.
[65] Tanenbaum A. S.: *Computernetzwerke*, 3 Aufl., München, Prentice Hall, 1998.
[66] Tian M., Gramm A., Ritter H., Schiller J., Winter R.: *A Survey of current Approaches towards Specification and Management of Quality of Service for Web Services*, in: PIK - Praxis der Informationsverarbeitung und Kommunikation, 27 (2004) 3, München, K.G. Saur, S. 132-139.
[67] Ullenboom C.: *Java ist auch eine Insel*, 4. Aufl., Bonn, Galileo Press, 2004.
[68] Weitzel T., König W., Busse R.: *Beiträge zur Industrialisierung von Finanzprozessen durch Web Services – Externes Straight Through Processing für Master-KAGen*, in: Information Management & Consulting, 19 (2004) Sonderausgabe zum 50. Geburtstag von Helmut Krcmar, Saarbrücken, mc information multimedia communication, S. 64-72.
[69] Weitzel T., Martin S. V., König W.: *Straight Through Processing auf XML-Basis im Wertpapiergeschäft*, in: Wirtschaftsinformatik, 45 (2003) 4, Wiesbaden, Vieweg, S. 409-420.
[70] Werra D. d., Hertz A.: *Tabu search techniques: A tutorial and an application to neural networks*, in: OR Spektrum, 11 (1989), Berlin, Heidelberg, Springer, S. 131-141.

[71] Westerinen A., Schnizlein J., Strassner J., Scherling M., Quinn B., Herzog S., Huynh A., Carlson M., Perry J., Waldbusser S.: *Terminology for Policy-Based Management*, Request for Comments 3198, http://www.ietf.org/rfc/rfc3198.txt, The Internet Society, November 2001, Abruf am 10.10.2005.

[72] Wirth N.: *Algorithms + Data Structures = Programs*, Englewood Cliffs, Prentice-Hall, 1976.

[73] Wroclawski J.: *Specification of the Controlled-Load Network Element Service*, Request for Comments 2211, http://www.ietf.org/rfc/rfc2211.txt, The Internet Society, Sep. 1997, Abruf am 10.10.2005.

[74] Wroclawski J.: *The Use of RSVP with IETF Integrated Services*, Request for Comments 2210, http://www.ietf.org/rfc/rfc2210.txt, The Internet Society, Sep.1997, Abruf am 10.10.2005.

[75] Wyssusek B.: *Geschäftsprozeßmodell, Geschäftsprozeßmodellierung*, in: Lexikon der Wirtschaftsinformatik, Aufl. 4., Mertens P. et al. (Hrsg.), Berlin, Heidelberg, New York, Barcelona, Hongkong, London, Mailand, Paris, Tokio, Springer, 2001, S. 210f.

[76] Yu T., Lin K.-J.: *The Design of QoS Broker Algorithms for QoS-Capable Web Services*, in: Proceedings of IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'04), Taipei, Taiwan, 2004, S. 17-24.

[77] Yu T., Lin K.-J.: *Service Selection Algorithms for Web Services with End-to-End QoS Constraints*, in: Proceedings of IEEE International Conference on E-Commerce Technology (CEC'04), San Diego, CA, USA, 2004, S. 129-136.

[78] Zemel E.: *An O(n) Algorithm for the Linear Multiple Choice Knapsack Problem and Related Problems*, in: Information Processing Letters, 18 (1984) 3, Amsterdam, North-Holland Pub. Co., S. 123-128.

[79] Zeng L., Benatallah B., Dumas M., Kalagnanam J., Sheng Q. Z.: *Quality Driven Web Services Composition*, in: Proceedings of 12th International Conference on World Wide Web (WWW'03), Budapest, Ungarn, 2003, S. 411-421.

[80] Zeng L., Benatallah B., Ngu A. H. H., Dumas M., Kalagnanam J., Chang H.: *QoS-Aware Middleware for Web Services Composition*, in: IEEE Transactions on Software Engineering, 30 (2004) 5, New York, Institute of Electrical and Electronics Engineers, S. 311-327.