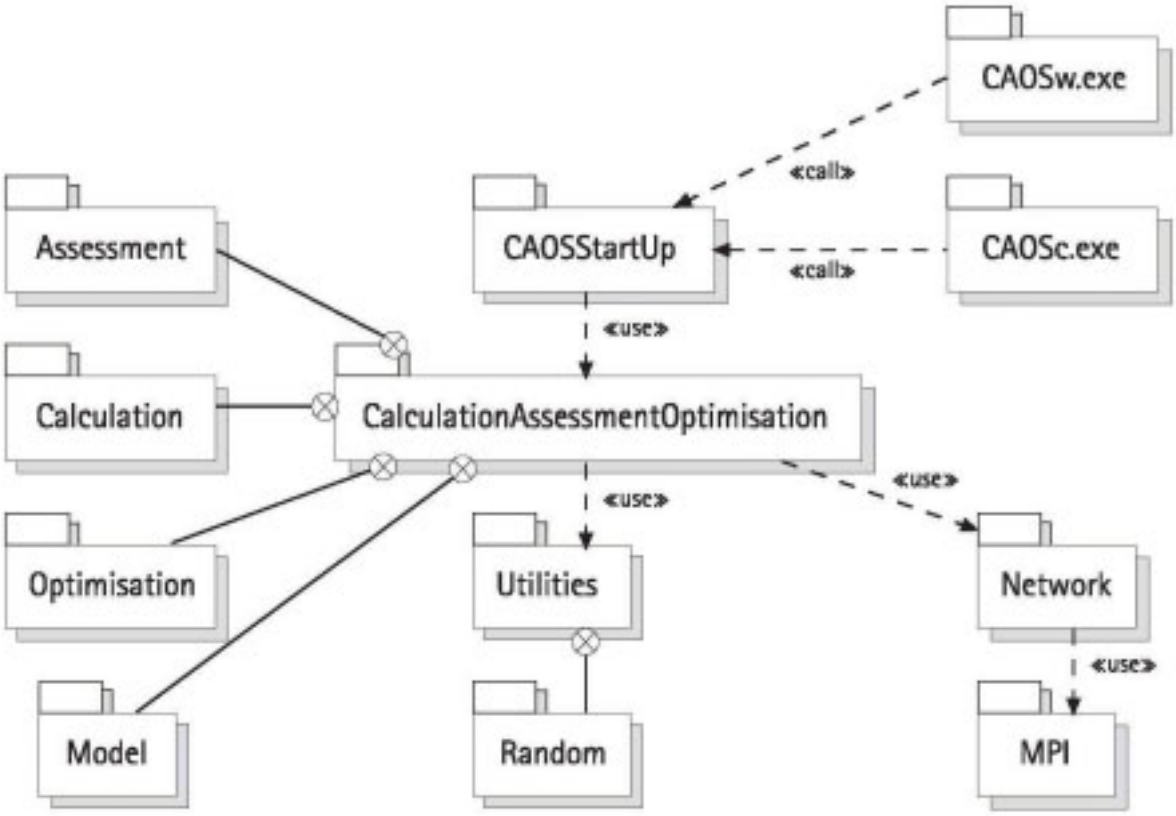
Michael Kämpf

Software-Framework zur Simulationsbasierten Optimierung mit Anwendung auf Produktions- und Lagerhaltungssysteme



Cuvillier Verlag Göttingen

TECHNISCHE UNIVERSITÄT CHEMNITZ

Fakultät für Informatik

Professur für Modellierung und Simulation

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur

(Dr.-Ing.)

Software-Framework zur Simulationsbasierten Optimierung mit Anwendung auf Produktions- und Lagerhaltungssysteme

Dipl.-Inf. Michael Kämpf,

geboren am 10. Januar 1976 in Jena

Chemnitz, den 26. Mai 2009

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.
1. Aufl. - Göttingen : Cuvillier, 2009
Zugl.: (TU) Chemnitz, Univ., Diss., 2009

978-3-86727-964-2

Kämpf, Michael
Software-Framework zur Simulationsbasierten Optimierung mit Anwendung auf Produktions- und Lagerhaltungssysteme
Dissertation zur Erlangung des akademischen Grades Doktoringenieur (Dr.-Ing.), Fakultät für Informatik
Technische Universität Chemnitz, Mai 2009

© CUVILLIER VERLAG, Göttingen 2009
Nonnenstieg 8, 37075 Göttingen
Telefon: 0551-54724-0
Telefax: 0551-54724-21
www.cuvillier.de

Alle Rechte vorbehalten. Ohne ausdrückliche Genehmigung des Verlages ist es nicht gestattet, das Buch oder Teile daraus auf fotomechanischem Weg (Fotokopie, Mikrokopie) zu vervielfältigen.
1. Auflage, 2009
Gedruckt auf säurefreiem Papier

978-3-86727-964-2

Danksagung

Ich danke herzlichst all denjenigen Personen, die mich durch ihre fachlich kompetenten Anregungen bei der Anfertigung dieser Dissertation inspiriert haben und damit ihren Anteil am erfolgreichen Voranschreiten meiner Dissertation hatten.

In erster Linie möchte ich mich bei Herrn Prof. Dr. rer. nat. habil. Dr. oec. Köchel für seine sehr gute Betreuung während der Anfertigung meiner Dissertation und die Übernahme meines Themas herzlichst danken. Er half mir u. a. auch bei der erfolgreichen Beantragung eines Landesstipendiums, durch welches es mir möglich war, an verschiedenen Fachtagungen und Konferenzen teilzunehmen.

Mein weiterer Dank gilt Prof. Dr. rer. nat. Rose, welcher sich mit großem Interesse zur Erstellung eines Gutachtens bereit erklärte.

Nicht minder gilt mein Dank Herrn Prof. Dr. rer. nat. habil. Fellenberg für die Ermunterung zur Anfertigung einer Dissertation und die Suche nach einer geeigneten Professur. Er begleitete mich bereits aktiv während meines ersten Studiums und half mir im Vorfeld der Anfertigung meiner Dissertation ein Umfeld zu finden, in dem sich meine persönlichen Stärken und Fähigkeiten besonderes entwickeln konnten.

Weiterhin bedanke ich mich auch bei verschiedenen Mitarbeitern der Fakultät für Informatik der Technischen Universität Chemnitz, insbesondere den Mitarbeitern der Professur „Modellierung und Simulation“, welche mich auf meinem wissenschaftlichen Weg begleiteten und als geistige Stütze hilfreich zur Seite standen. Vorrangig waren dies Herr Nieländer, Herr Flohrer, Herr Schüler, Frau Wachsmuth (geboren Görlich), Frau Petersen (geboren Riedel) und Frau Schönke.

Außerdem möchte ich mich bei meiner Frau Nicole, meinen Kindern Lara Michelle und Niklas Aaron sowie meinen Eltern und den anderen Familienmitgliedern bedanken, welche durch ihr Verständnis und ihre Rücksichtnahme einen wichtigen Teil am Gelingen meiner Dissertation hatten.

Letzten Endes bin ich Herrn Lawrence M. Wein von der Universität Stanford für die Bereitstellung seiner Vergleichsergebnisse zu Dank verpflichtet.

Chemnitz, Mai 2009 Michael Kämpf

Inhaltsverzeichnis

Abbildungsverzeichnis . . . . . *v*

Tabellenverzeichnis . . . . . *xi*

Verzeichnis der verwendeten Notationen, Operatoren und Symbole . *xix*

1 Einführung *1*

1.1 Problemstellungen heutiger ökonomischer Systeme *1*
1.2 Ziele der Arbeit *5*
1.3 Gliederung der Arbeit *7*

2 Modellgestützte, simulationsbasierte Optimierung *9*

2.1 Einführung *10*
2.2 Charakteristiken und Definitionen *12*
2.2.1 System und Modell *12*
2.2.2 Simulationsmodell und Simulation *14*
2.2.3 Optimierungsproblem *17*
2.2.4 Nebenbedingungen *19*
2.2.5 Zielfunktionsraum *23*
2.2.6 Diskretisierung des Raumes der Entscheidungsvariablen und Permutationsprobleme *27*
2.2.7 Definition zur simulationsbasierten Optimierung *34*
2.3 Lösungsverfahren zur simulationsbasierten Optimierung *34*
2.4 Rechnerinterne Modellierungsvariante *39*
2.4.1 Schritte bei der rechnerinternen Modellierung *42*
2.4.2 Verwendung einer Modelldefinition *44*
2.4.3 Sichtweisen auf ein rechnerinternes Modell *44*
2.4.4 Möglichkeit der Weiterverwendung eines Simulators *45*
2.4.5 Rechnerinterne Abbildung der Entscheidungsvariablen *46*
2.5 Bewertung *46*
2.6 Entscheidungsfindung *50*
2.7 Zusammenfassung *52*

3 Simulation *55*

3.1 Einführung *55*
3.2 Gründe für die Simulation *56*
3.3 Simulationstypen und Zeitfortschreibung bei der Simulation . . . *58*
3.4 Stochastische Prozesse und Zeitreihen *62*
3.5 Dauer der transienten und der stationären Phase *64*

3.6 Verkürzung der Simulationsdauer bei diskreter Simulation 76
3.7 Zufallszahlenerzeugung 84
3.8 Zusammenfassung 88

4 Optimierung 91
4.1 Einführung 91
4.2 Exakte Lösungsverfahren 93
4.2.1 Methoden der lokalen nichtlinearen Optimierung 93
4.2.2 Methoden der globalen nichtlinearen Optimierung 94
4.3 Heuristische Suchverfahren 95
4.3.1 Abbruchkriterien 97
4.3.2 Deterministische Eröffnungsverfahren 98
4.3.3 Stochastische Eröffnungsverfahren 99
4.3.4 Deterministische Verbesserungsverfahren 99
4.3.5 Stochastische Verbesserungsverfahren 101
4.3.6 Stochastische Verbesserungsverfahren mit einer Lösung 103
4.3.7 Stochastische Verbesserungsverfahren mit mehreren Lösungen . . 109
4.4 Verfahren der mehrkriteriellen Optimierung 115
4.5 Methodiken zur Verkürzung der Optimierungsdauer 118
4.5.1 Hybride Optimierung 119
4.5.2 Verteilte und parallele Optimierung 122
4.6 Zusammenfassung 129

5 Implementierung einer Software zur simulationsbasierten Optimierung 131
5.1 Einführung 131
5.2 Kurzer Überblick über ausgewählte Softwaresysteme zur Simulation und Optimierung 133
5.3 Das Softwaresystem `CAOS` 136
5.3.1 Die Programmiersprache `C#` 136
5.3.2 Modularer Aufbau des Softwaresystems (Bibliotheken) 137
5.3.3 Die Bibliothek `CAOSStartUp` 139
5.3.4 Die Bibliothek `Utilities` 139
5.3.5 Die Bibliothek `Network` 142
5.3.6 Die Bibliothek `CalculationAssessmentOptimisation` 144
5.3.7 Anwendungsfälle und Nutzungsmöglichkeiten 168
5.4 Zusammenfassung 169

6 Produktions- und Lagerhaltungssysteme mit stochastischen Einflüssen 171
6.1 Einführung 172
6.2 Grundlegende Problemstellungen und Basismodelle 173
6.3 Simulationsmodell und Optimierungsproblem 178
6.3.1 Modellkomponenten und -parameter 178
6.3.2 Entscheidungsvariablen 181
6.3.3 Restriktionen (Nebenbedingungen) 191
6.3.4 Kostenbetrachtung und Optimierungskriterien 191

6.3.5 Zusammenfassung der genutzten Größen . . . 193
6.4 Modelltypen . . . 195
6.4.1 Modelltyp 1: Eine Fertigungseinheit und N Produkte . . . 202
6.4.2 Modelltyp 2: M Fertigungseinheiten und N Produkte . . . 204
6.4.3 Modelltyp 3: M Fertigungseinheiten, N Produkte und Berücksichtigung von Lieferketten . . . 206
6.5 Modellbeispiele mit Ergebnissen . . . 212
6.5.1 Voruntersuchungen . . . 212
6.5.2 Modellbeispiele und Ergebnisse für Modelltyp 1 . . . 215
6.5.3 Modellbeispiele und Ergebnisse für Modelltyp 2 . . . 223
6.5.4 Vergleichsmodelle von Markowitz, Reimann und Wein . . . 229
6.6 Zusammenfassung . . . 232

7 Resümee und Ausblick . . . 235
7.1 Resümee . . . 235
7.2 Ausblick auf zukünftige Untersuchungsrichtungen . . . 238

A Auszug aus der Unified Modeling Language (UML) . . . 241
A.1 Einführung . . . 241
A.2 Allgemeine Elemente von UML-Diagrammen . . . 243
A.3 Verwendete Verhaltensdiagramme der UML . . . 243
A.3.1 Aktivitätsdiagramme . . . 243
A.3.2 Anwendungsfalldiagramme . . . 245
A.3.3 Kommunikationsdiagramme . . . 246
A.4 Verwendete Strukturdiagramme der UML . . . 247
A.4.1 Klassendiagramme . . . 247
A.4.2 Paketdiagramme . . . 248
A.4.3 Verteilungsdiagramme . . . 249

B Konstrukte für Produktionsablauf-Diagramme und die Sprache EcoSyL 0.1 . . . 251
B.1 Mögliche Elemente eines Produktionssystems . . . 251
B.2 Klassifikation von Scheduling-Problemen . . . 253

C Die Programmiersprache C#, deren Zwischensprache und das .NET-Framework . . . 257
C.1 Begriffserklärungen . . . 257
C.2 Übersetzungs- und Laufzeitverhalten bei C#-Programmen . . . 259
C.3 Vorteile . . . 260
C.4 Nachteile . . . 261
C.5 Sprachbesonderheiten . . . 261
C.6 Vergleich mit der Programmiersprache Java . . . 264
C.7 Frei verfügbare Entwicklungs- und Laufzeitumgebungen . . . 264
C.8 Fazit . . . 264

D Hinweise zum Softwaresystem CAOS . . . 267
D.1 Visuelle Beispiele . . . 268
D.2 Mögliche Nebenbedingungen . . . 273

Literaturverzeichnis ***275***

Stichwortverzeichnis ***299***
Stichwortverzeichnis *299*

Abbildungsverzeichnis

1.1 Zerlegung eines (ökonomischen) Systems aus verschiedenen Sichtweisen 2
1.2 Rechnerinterne Modellabbildung eines (ökonomischen) Systems 3

2.1 Prinzip der angewandten Methode zur modellgestützten, simulationsbasierten Optimierung (Grobgerüst) 10
2.2 Überführung der Ausgabewerte in die Zielfunktionswerte anhand einer Gewichtung 17
2.3 Raum der Entscheidungsvariablen und der Zielfunktionen vor Gewichtung der Ausgabewerte 19
2.4 Beschränkung des Raumes der Entscheidungsvariablen und der Ausgabewerte durch Nebenbedingungen (Beispiel) 21
2.5 Typen von Nebenbedingungen nach [Fu01a] 22
2.6 Pareto-optimale Menge (Beispiel) 24
2.7 Globale und lokale Pareto-optimale Menge bei mehrkriterieller Optimierung (Beispiel) 25
2.8 Visualisierung der Zielfunktion und der Entscheidungsvariablen bei einem SOP ($m = 1$, $n = 1$) 28
2.9 Visualisierungsmöglichkeiten bei MOPs ($m = 2$ und $n = 1$) 29
2.10 Visualisierung der Zielfunktionswerte durch ein Sterndiagramm bei MOP (Beispiel für $m = 5$ und drei sich nicht dominierende Pareto-optimale Lösungen) 30
2.11 Epsilon-Nachbarschaft bei reellwertigen Entscheidungsvariablen 32
2.12 Epsilon-Nachbarschaft bei Permutationsentscheidungsvariablen 33
2.13 Visualisierung einer Zielfunktion und eines Optimierungsverlaufes bei einem SOP ($m = 1, n = 1$) 35
2.14 Algorithmus zur iterativen simulationsbasierten Optimierung (UML-Aktivitätsdiagramm) 36
2.15 Algorithmus zur sequentiellen simulationsbasierten Optimierung (UML-Aktivitätsdiagramm) 36
2.16 Prinzipielle Ansätze der simulationsbasierten Optimierung (nach [CM97], [Fu01a] und [FGA05], Auszug) 38
2.17 Algorithmus zur modellgestützten, simulationsbasierten Optimierung als Grobdarstellung (UML-Aktivitätsdiagramm) 39
2.18 Algorithmus zur Nutzung einer rechnerinternen Modelldarstellung im CAOS (UML-Aktivitätsdiagramm) 41
2.19 Rechnerinterne Darstellung und Raum der Entscheidungsvariablen . . 42
2.20 Notwendige Schritte bei der Modellierung eines Realsystems 43

2.21 Algorithmus zur modellgestützten, simulationsbasierten Optimierung als Feindarstellung (UML-Aktivitätsdiagramm) . . . 44
2.22 Unterschiedliche Sichtweisen auf ein Modell mittels einer 3-Schicht-Architektur (model-view-controler [engl.]) . . . 45
2.23 Zuordnung verschiedener Optimierungsverfahren zu unterschiedlichen Entscheidungsvariablen (Beispiel) . . . 47
2.24 Problem der Gewichtung bei der Transformation eines MOP in ein SOP (Beispiel) . . . 51
2.25 Reduzierung mehrkriterieller Lösungen anhand von Mindestanforderungen bei mehrkriterieller Entscheidungsfindung (Beispiel) . . . 52
2.26 Raum der Zielfunktionen und Entscheidungsraum bei einkriterieller Entscheidungsfindung . . . 53

3.1 Zeitfortschreibung bei diskreter Simulation . . . 60
3.2 Algorithmus zur Durchführung einer diskret ereignisorientierten Simulation (UML-Aktivitätsdiagramm) . . . 61
3.3 Visualisierung von Zeitreihen mittels Plots (Beispiel) . . . 63
3.4 Phasen eines Simulationsexperimentes an einem Beispiel . . . 65
3.5 Auswirkungen unterschiedlicher Dauern bei der Einschwingphase . . . 67
3.6 Vergleich zwischen Warteschlangenlänge und Wartekosten eines Simulationsmodells (Beispiel) . . . 72
3.7 Vergleich des Verhaltens von stark, mittelmäßig und schwach ausgelasteten Simulationsmodellen (Beispiel) . . . 73
3.8 Mehrfache Wiederholung kurzer Simulationsläufe kontra einzelnem, langen Simulationslauf . . . 74
3.9 Heuristik zur Festlegung der Dauer der stationären Phase (Beispiel) . 75
3.10 Synchronisationsverfahren der verteilten und parallelen Simulation . . 77
3.11 Triviale Verteilung der Simulationsläufe eines Simulationsexperimentes (UML-Verteilungsdiagramm) . . . 80
3.12 Verfahren der datenparallelen Simulation (vgl. [Käm03, S. 23, Abb. 4]) 80
3.13 Beispiel für das Epsilon-Abbruch-Verfahren mit absoluter Epsilon-Umgebung . . . 82
3.14 Ausgewählte Typen von rekursiven arithmetischen Zufallszahlengeneratoren . . . 86

4.1 Einteilung heuristischer Optimierungsverfahren zur Suche im Zielfunktionsraum . . . 96
4.2 Einteilungsmöglichkeit von Abbruchkriterien . . . 97
4.3 Algorithmus zur Bestimmung einer definierten Ausgangslösung (UML-Aktivitätsdiagramm) . . . 100
4.4 Überblick über heuristische, stochastische Verbesserungsverfahren (Auswahl) . . . 102
4.5 Einkriterielle Optimierer des CAOS (UML-Klassendiagramm) 103
4.6 Mehrkriterielle Optimierer des CAOS (UML-Klassendiagramm) 118
4.7 Vergleich zwischen Anzahl paralleler Optimierungsverfahren und Anzahl verfügbarer Rechnerprozessoren . . . 119

4.8 Mögliches Klassifizierungsschema der hybriden Optimierungsverfahren nach [Had01, S. 34] *121*
4.9 Triviale Verteilung der Lösungsvektoren bei verteilter und paralleler Optimierung (UML-Verteilungsdiagramm) *128*

5.1 Übersicht über die Pakete im CAOS (UML-Paketdiagramm) *138*
5.2 im CAOS vorhandene Zufallszahlengeneratoren (UML-Klassendiagramm) *141*
5.3 im CAOS vorhandene Verteilungen (UML-Klassendiagramm) *141*
5.4 Prinzipieller Aufbau eines Modells im CAOS (UML-Klassendiagramm) *144*
5.5 Klasseninhalt der Entscheidungsvariablen (Auszug, UML-Klassendiagramm) *145*
5.6 Zusammenspiel zwischen einem Modell und den CAOS-Komponenten zur Entwurfszeit (UML-Kommunikationsdiagramm) *146*
5.7 Zusammenspiel zwischen einem Modell und den CAOS-Komponenten zur Laufzeit (UML-Kommunikationsdiagramm) *146*
5.8 Vorhandene Listen zur Aufbewahrung von Elementen im CAOS (UML-Klassendiagramm) *147*
5.9 Vorhandene Elemente zur Aufbewahrung von Attributen im CAOS (UML-Klassendiagramm) *147*
5.10 Vorhandene Attribute im CAOS (UML-Klassendiagramm) *148*
5.11 Aufbau der Definition eines Modells im CAOS (UML-Klassendiagramm) *148*
5.12 Auszug der im CAOS vorhandenen Berechner (UML-Klassendiagramm) *151*
5.13 Algorithmus zur Ausführung des Optimierungsmanagers im CAOS (UML-Aktivitätsdiagramm) *152*
5.14 Algorithmus zur Initialisierung des Optimierungsmanagers im CAOS (UML-Aktivitätsdiagramm) *153*
5.15 Optimierungsmanager (UML-Klassendiagramm) *154*
5.16 Algorithmus zur Initialisierung eines Optimierers (UML-Aktivitätsdiagramm) *155*
5.17 Algorithmus zum Ausführen eines Optimierungsschrittes (UML-Aktivitätsdiagramm) *155*
5.18 Algorithmus zum Ausführen eines Optimierungsschrittes bei der Simulierten Abkühlung (UML-Aktivitätsdiagramm) *156*
5.19 Algorithmus zur Initialisierung des Optimierers zur Simulierten Abkühlung (UML-Aktivitätsdiagramm) *157*
5.20 Algorithmus zur Initialisierung des Optimierers zur Tabusuche (UML-Aktivitätsdiagramm) *157*
5.21 Algorithmus zum Ausführen eines Optimierungsschrittes bei der Tabusuche (UML-Aktivitätsdiagramm) *158*
5.22 Algorithmus zum Ausführen eines Optimierungsschrittes beim generationsbasierten Genetischen Algorithmus (UML-Aktivitätsdiagramm) . *159*
5.23 Algorithmus zum Ausführen eines Optimierungsschrittes beim Genetischen Algorithmus mit ständiger Ersetzung (UML-Aktivitätsdiagramm) *160*
5.24 Algorithmus zum Ausführen eines Optimierungsschrittes des Hybrid-parallelen Optimierers im CAOS (UML-Aktivitätsdiagramm) *161*

5.25 Algorithmus zur Initialisierung des Hybrid-sequentiellen Optimierers im CAOS (UML-Aktivitätsdiagramm) . . . 161
5.26 Algorithmus zur Initialisierung des Hybrid-parallelen Optimierers im CAOS (UML-Aktivitätsdiagramm) . . . 162
5.27 Algorithmus zum Ausführen eines Optimierungsschrittes des Hybrid-parallelen Optimierers im CAOS (UML-Aktivitätsdiagramm) . . . 163
5.28 Manager für die Kommunikation basierend auf verschiedenen Netzwerkprotokollen (UML-Klassendiagramm) . . . 163
5.29 Verteilte Optimierung basierend auf dem Message Passing Interface (UML-Verteilungsdiagramm) . . . 164
5.30 Algorithmus zur Ausführung eines Slave-Prozesses zur verteilten und parallelen Optimierung im CAOS (UML-Aktivitätsdiagramm) . . . 165
5.31 Algorithmus zur Ausführung des Master-Prozesses zur verteilten und parallelen Optimierung im CAOS (UML-Aktivitätsdiagramm) . . . 166
5.32 Algorithmus zum Beenden der Ausführung des Master-Prozesses bei der verteilten und parallelen Optimierung im CAOS (UML-Aktivitätsdiagramm) . . . 167
5.33 Algorithmus zur Bewertung der Ergebnisse einer Simulation oder Berechnung (UML-Aktivitätsdiagramm) . . . 168
5.34 Mögliche Anwendungsfälle des CAOS (UML-Anwendungsfalldiagramm) 169
5.35 Algorithmus zur Nutzung eines Simulators oder Berechners aus dem CAOS mittels externer Optimierung (UML-Aktivitätsdiagramm) . . . 170

6.1 Vergleich zwischen den LF-Strategien (s, nQ) und (s, S) (Beispiel) . . 187
6.2 Auszug der im CAOS vorhandenen CSLSP-Simulatoren (UML-Klassendiagramm) . . . 196
6.3 Ereignisse zu den CSLSP-Simulatoren im CAOS (UML-Klassendiagramm) . . . 196
6.4 Ereignisroutine des Ereignisses „NewClient“ beim CSLSP-Simulator MultipleItemSystem (UML-Aktivitätsdiagramm) . . . 198
6.5 Ereignisroutine des Ereignisses „ProductionEnd“ beim CSLSP-Simulator MultipleItemSystem (UML-Aktivitätsdiagramm) . . . 199
6.6 Ereignisroutine des Ereignisses „TurninEnd“ beim CSLSP-Simulator MultipleItemSystem (UML-Aktivitätsdiagramm) . . . 199
6.7 Ereignisroutine des Ereignisses „SimulationEnd“ beim CSLSP-Simulator MultipleItemSystem (UML-Aktivitätsdiagramm) . . . 200
6.8 Algorithmus zum Zurücksetzen des CSLSP-Simulators MultipleItemSystem (UML-Aktivitätsdiagramm) . . . 201
6.9 Algorithmus zum Generieren der initialen Simulationsereignisse beim CSLSP-Simulator MultipleItemSystem (UML-Aktivitätsdiagramm) . . . 201
6.10 Darstellung des untersuchten Produktions- und Lagerhaltungssystems beim Modelltyp 1 (EcoSyL-Diagramm) . . . 202
6.11 Darstellung des untersuchten Produktions- und Lagerhaltungssystems beim Modelltyp 2 (EcoSyL-Diagramm) . . . 205
6.12 Beispiel einer Lieferkette und deren Umformung in ein Modell des Modelltyps 3 (EcoSyL-Diagramm) . . . 208

6.13 Darstellung des untersuchten Produktions- und Lagerhaltungssystems beim Modelltyp 3 (EcoSyL-Diagramm) *209*
6.14 Ereignisroutine des Ereignisses „SupplyChainProductionEnd“ beim CSLSP-Simulator MultipleItemSupplyChainSystem *(UML-Aktivitätsdiagramm)* *210*
6.15 Vergleich der Ergebnisse der Voruntersuchungen für verschiedene Optimierungsverfahren (1/3) *213*
6.16 Vergleich der Ergebnisse der Voruntersuchungen für verschiedene Optimierungsverfahren (2/3) *214*
6.17 Vergleich der Ergebnisse der Voruntersuchungen für verschiedene Optimierungsverfahren (3/3) *215*
6.18 Vergleich der unteren Produktionsgrenzen s und Produktionsmengen Q beim Ein-Produkt-Modell und LF-Strategie (s, nQ) *219*
6.19 Vergleich der unteren Produktionsgrenzen s und der oberen Produktionsgrenzen S beim Ein-Produkt-Modell und LF-Strategie (s, S) . . . *220*
6.20 Abgewiesene Kunden bei den Fünf-Produkt-Modellen für die optimalen Lösungen der Untersuchungsmodelle bei Anwendung der R-Strategie LWVQ *und der LF-Strategie* (s, S) *225*
6.21 Vergleich der Ergebnisse der Modelle von Markowitz, Reimann und Wein (1/2) *230*
6.22 Vergleich der Ergebnisse der Modelle von Markowitz, Reimann und Wein (2/2) *231*

A.1 Beispiel eines UML-Aktivitätsdiagrammes *243*
A.2 Beispiel eines UML-Anwendungsfalldiagrammes *246*
A.3 Beispiel eines UML-Kommunikationsdiagrammes *246*
A.4 Beispiel 1 eines UML-Klassendiagrammes *248*
A.5 Beispiel 2 eines UML-Klassendiagrammes *248*
A.6 Beispiel eines UML-Verteilungsdiagrammes *250*

C.1 Historische Entwicklung der für C# *relevanten (objektorientierten) Programmiersprachen (nach [BS03, Abb. 1-1])* *258*
C.2 Programmausführung eines C#*-Programms (UML-Aktivitätsdiagramm)* *260*

D.1 Logo des Softwaresystems CAOS *267*
D.2 Oberfläche des Softwaresystems CAOS *am Beispiel (Modellansicht)* . . *268*
D.3 Oberfläche des CAOS *am Beispiel (Eingabe einer Wahrscheinlichkeitsverteilung)* *269*
D.4 Oberfläche des CAOS *am Beispiel (Simulationsansicht)* *270*
D.5 Oberfläche des CAOS *am Beispiel (Optimierungsansicht)* *271*
D.6 Oberfläche des CAOS *am Beispiel (Sterndiagramm)* *272*

Tabellenverzeichnis

2.1 *Beispiele für Nebenbedingungen* *23*
2.2 *Kategorien eines rechnerinternen Modells* *40*
2.3 *Elemente und Attribute der Kategorien eines rechnerinternen Modells* *40*

3.1 *Ausgewählte Simulationstypen für verschiedene Sichtweisen* *59*
3.2 *Regeln für die Bestimmung des Übergangs von transienter Phase in die stationäre Phase* *68*
3.3 *Pro und Kontra der verteilten und parallelen Simulation (Auszug der innerhalb dieser Arbeit entscheidenden Kriterien)* *78*
3.4 *Typen von rekursiven arithmetischen Zufallszahlengeneratoren (Auswahl)* *87*

4.1 *Unterscheidungsmerkmale bei exakten und heuristischen Optimierungsverfahren (Auswahl)* *92*
4.2 *Allgemeingültige Abbruchkriterien* *98*
4.3 *Deterministische Eröffnungsverfahren (Auswahl)* *99*
4.4 *Deterministische Verbesserungsverfahren (Auswahl)* *100*
4.5 *Unterscheidungskriterien moderner heuristischer Verfahren* *101*
4.6 *Überblick über ausgewählte moderne heuristische Verfahren (eine (parallele) Lösung)* *103*
4.7 *Verfahrensparameter des Optimierungsverfahrens der Simulierten Abkühlung* *106*
4.8 *Verfahrensparameter des Optimierungsverfahrens der Tabusuche* . . . *108*
4.9 *Überblick über ausgewählte moderne heuristische Verfahren (n parallele Lösungen)* *109*
4.10 *Instanzen der Evolutionären Algorithmen* *111*
4.11 *Überblick über die Unterschiede von GA und ES* *112*
4.12 *Verfahrensparameter der Optimierungsverfahren für die Genetischen Algorithmen GA_{gen} und GA_{ss}* *113*
4.13 *Mehrkriterielle Varianten moderner heuristischer Verfahren (Auswahl)* *116*

5.1 *Kommerzielle Software zur simulationsbasierten Optimierung (Auszug, nach [Fu01a] und [Man01])* *134*
5.2 *Gegenüberstellung von spezialisierten Simulationssprachen und konventionellen Programmiersprachen* *134*
5.3 *Gegenüberstellung der Vor- und Nachteile der Programmiersprache* `C#` *137*
5.4 *Implementierte Wahrscheinlichkeitsverteilungen im* `CAOS` *143*
5.5 *Beispiel der rechnerinternen Abbildung einer Modelldeklaration (Auszug)* *149*

6.1 Überblick über die Charakteristiken von Produktions- und Lagerhaltungssystemen . . . 174
6.2 Überblick über Reihenfolge- und Losgrößenprobleme (Auswahl) 176
6.3 Überblick über die Unterschiede der untersuchten Modelle 179
6.4 Überblick über die verschiedenen Fertigungspolitiken zur Bestimmung einer optimalen Losgröße und des Fertigungs-Zeitpunktes (Auswahl) . 183
6.5 Überblick über die Modellparameter, Modellvariablen und Entscheidungsvariablen der untersuchten Modelle 194
6.6 Überblick über die veränderten Modellparameter der untersuchten Modelle beim Modelltyp 2 195
6.7 Überblick über die zusätzlichen Modellparameter, Modellvariablen und Entscheidungsvariablen der untersuchten Modelle beim Modelltyp 3 . . 195
6.8 Überblick über die untersuchten Parametereinstellungen der verwendeten Optimierungsverfahren 213
6.9 Überblick über die gemeinsamen Modellparameter und Entscheidungsvariablen der Modellbeispiele des Modelltypes 1 217
6.10 Überblick über die unterschiedlichen Modellparameter und Entscheidungsvariablen der Modellbeispiele des Modelltypes 1 217
6.11 Ergebnisse der Tests für 1 Produkt (beliebige R-Strategie und alle LF-Strategien) 218
6.12 Ergebnisse der Tests für 2 Produkte (jeweils optimale Lösungen der R-Strategien und LF-Strategien) 222
6.13 Ergebnisse der Tests für 5 Produkte (jeweils optimale Lösungen der R-Strategien und LF-Strategien) 223
6.14 Überblick über die veränderten, unterschiedlichen Modellparameter der Modellbeispiele des Modelltypes 2 im Gegensatz zum Modelltyp 1 . . . 224
6.15 Überblick über die veränderten, gemeinsamen Modellparameter der Modellbeispiele des Modelltypes 2 im Gegensatz zum Modelltyp 1 . . . 224
6.16 Ergebnisse der Tests für 5 Produkte (2 Fertigungseinheiten, $scst^{voll}$, Tabusuche) 226
6.17 Ergebnisse der Tests für 5 Produkte (2 Fertigungseinheiten, $scst^{halb}$, Tabusuche) 227
6.18 Ergebnisse der Tests für 5 Produkte (4 Fertigungseinheiten, $scst^{viertel}$, Tabusuche) 228
6.19 Ergebnisse für die Umrüst- und Produktionskosten pro Fertigungseinheit für die Tests mit 5 Produkten 228

A.1 Allgemeine Elemente von UML-Diagrammen 243
A.2 Elemente eines UML-Aktivitätsdiagramms 244
A.3 Elemente eines UML-Anwendungsfalldiagramms 245
A.4 Elemente eines UML-Kommunikationsdiagramms 246
A.5 Elemente eines UML-Klassendiagramms 247
A.6 Elemente eines UML-Paketdiagramms 249
A.7 Elemente eines UML-Verteilungsdiagramms 249

B.1 Mögliche Elemente der visuellen Beschreibungssprache EcoSyL 0.1 (Teil 1/2) 252

B.2 Mögliche Elemente der visuellen Beschreibungssprache EcoSyL 0.1 (Teil 2/2) *253*
B.3 Bedeutung des Parameters α bei der Klassifizierung von Scheduling-Problemen *254*
B.4 Bedeutung des Parameters β bei der Klassifizierung von Scheduling-Problemen *255*

C.1 Frei verfügbare Entwicklungs- und Laufzeitumgebungen für die Programmiersprache `C#` *265*

Definitionen

2.1 Mehrkriterielles Optimierungsproblem . . . 18
2.2 Einkriterielles Optimierungsproblem . . . 18
2.3 Konvexe Funktion . . . 19
2.4 Zulässiger Bereich eines Optimierungsproblems . . . 20
2.5 Pareto-Optimalität . . . 23
2.6 Pareto-optimale Menge . . . 23
2.7 Globale Pareto-optimale Menge . . . 24
2.8 Lokale Pareto-optimale Menge . . . 24
2.9 Globales Optimum . . . 26
2.10 Lokales Optimum . . . 26
2.11 Permutation . . . 30
2.12 Nachbarschaft . . . 31
2.13 Zwei-Punkt-Tausch-Operator . . . 32
2.14 Zwei-Punkt-Tausch-Nachbarschaft . . . 33
2.15 Simulationsbasierte Optimierung . . . 34
2.16 Güte einer Lösung . . . 51

3.1 Stochastischer Prozess . . . 62
3.2 Starke Stationärität . . . 63
3.3 Schwache Stationärität . . . 63
3.4 Folge von Zufallszahlen . . . 85

Anmerkungen

Innerhalb dieser Arbeit wird für deutsche Fachbegriffe, für welche ein äquivalenter englischer Fachbegriff[1] angegeben ist, folgende Notation verwendet:

deutscher Fachbegriff [dt.; äquivalenter englischer Fachbegriff [engl.]]

oder

deutscher Fachbegriff [dt.; äquivalenter englischer Fachbegriff [engl., (englische) Abkürzung]].

Der englische Fachbegriff wird dabei klein geschrieben, falls nicht die unten angeführte Ausnahme gilt. Bspw. könnte folgender Fachbegriff innerhalb dieser Arbeit auftreten:

einkriterielles Optimierungsproblem [dt.; single optimisation problem [engl.]].

Für englische Fachbegriffe mit einer äquivalenten deutschen Übersetzung gelte Entsprechendes. Ist keine deutsche Entsprechung eines englischen Fachbegriffes bekannt, so dass nur der englische Fachbegriff verwendet wird, ist dies wie folgt kenntlich gemacht:

englischer Fachbegriff [engl.].

Werden Abkürzungen für englische Fachbegriffe verwendet, so werden die betreffenden Buchstaben des englischen Fachbegriffes groß geschrieben und unterstrichen. Bspw. könnte obiger Fachbegriff innerhalb dieser Arbeit auch wie folgt auftreten:

einkriterielles Optimierungsproblem [dt.; *Single Optimisation Problem* [engl., kurz: SOP]].

Eine weitere Ausnahme der Kleinschreibung von englischen Fachbegriffen bilden Worte, welche bereits weit gehend in den deutschen Sprachgebrauch übergegangen sind, wie z. B. Client, Server, Master oder Slave. Diese werden groß geschrieben

[1] Die gleichen Aussagen gelten für lateinische Fachbegriffe.

und müssen nicht zwingend durch „[engl.]“ gekennzeichnet sein. Genauso werden Eigenworte, welche innerhalb dieser Arbeit einen bestimmten Verwendungszweck widerspiegeln sollen, groß geschrieben, wie z. B. AsDefined oder Cyclic.

Anschließend sei noch erwähnt, dass die (eingetragenen) Warenzeichen entsprechend durch „®“ oder „^{TM}“ gekennzeichnet sind. Bspw. sind Microsoft und Windows eingetragene Warenzeichen der Microsoft Corporation und XML ist ein eingetragenes Warenzeichen des World Wide Web Consortium [engl., W3C]], Massachusetts Institute of Technology.

Die vorliegende Arbeit wurde unter Verwendung von LaTeX 2_ε-Quelltexten geschrieben bzw. gesetzt (siehe [Voß06] und [GMR+08]). Zudem wurde die Prüfung der Rechtschreibung mittels des deutschen Dudens in der 24. Auflage durchgeführt (siehe [Wer07]).

Verzeichnis der verwendeten Notationen, Operatoren und Symbole

Zahlenbereiche

$\mathbb{N} = \{0, 1, 2, 3, \ldots\}$	-	Bereich der natürlichen Zahlen bzw. Bereich der positiven ganzen Zahlen[2]
$\mathbb{N}^* = \{1, 2, 3, \ldots\}$	-	Bereich der natürlichen Zahlen ohne die Zahl 0
$\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$	-	Bereich der ganzen Zahlen
$\mathbb{R} = \{x \mid x$ ist rationale oder irrationale Zahl$\}$	-	Bereich der reellen Zahlen
$\mathbb{R}_+ = \{x \mid x \geq 0$ und $x \in \mathbb{R}\}$	-	Bereich der nicht negativen reellen Zahlen
$\mathbb{R}_+^* = \{x \mid x > 0$ und $x \in \mathbb{R}\}$	-	Bereich der positiven reellen Zahlen

Vektoren

$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{\dim(\vec{x})} \end{pmatrix} = (x_1, x_2, \ldots, x_{\dim(\vec{x})})^T$ - „normale“ und transformierte Schreibweise für einen Vektor

Logische Operatoren

$b_1 = b_2$	-	Vergleichsoperator (Ausdruck b_1 ist gleich dem Ausdruck b_2)
$b_1 \wedge b_2$	-	Verknüpfungsoperator (logische Verknüpfung der Ausdrücke b_1 und b_2)

Unäre Operatoren

$\lvert x \rvert$	-	Absolutbetrag des Wertes x
$\lfloor x \rfloor$	-	Abrunden des Wertes xon x auf die nächste ganze Zahl

[2] Nach der DIN-Norm 5473 (siehe [Int07]) gehört die Null zu den natürlichen Zahlen.

Binäre Operatoren

$x \leq y$ - kleiner-als-oder-gleich-Operator (Wert x kleiner als oder gleich dem Wert y)
$x \geq y$ - größer-als-oder-gleich-Operator (Wert x größer als oder gleich dem Wert y)
$x < y$ - kleiner-als-Operator (Wert x kleiner als der Wert y)
$x > y$ - größer-als-Operator (Wert x größer als der Wert y)
$x = y$ - Zuweisungsoperator (Wert x ergibt sich aus dem Ausdruck y)

Binäre Vektoroperatoren

$\vec{x} \circ \vec{y}$ - Verknüpfungsoperator zur Verknüpfung zweier Vektoren, wobei $\vec{x} \circ \vec{y} = (x_1 \cdot y_1, x_2 \cdot y_2, \ldots, x_n \cdot y_n)^T, \quad n = \dim(\vec{x}) = \dim(\vec{y})$
$\vec{x} \succ \vec{y}$ - Dominanzoperator (Dominanz des Vektors $\vec{x}$ über den Vektor $\vec{y}$), wobei $\forall\, i \in \{1, 2, \ldots, \dim(\vec{f})\} : f_i(\vec{x}) \leq f_i(\vec{y}) \wedge \exists\, j \in \{1, 2, \ldots, \dim(\vec{f})\} : f_j(\vec{x}) < f_j(\vec{y})$

Sonstige Operatoren

$\sum_{i=a}^{b} x_i$ - summiert die Werte von $\vec{x}$ beginnend bei dem Wert mit dem Index a bis zum Wert mit dem Index b auf, wobei $a \leq b$ sowie $a \leq \dim(\vec{x})$ und $b \leq \dim(\vec{x})$ gilt

Quantoren

$\forall x$ - All-Quantor (für alle x)
$\exists x$ - Existenz-Quantor (für mindestens ein x)
$\nexists x$ - Nicht-Existenz-Quantor (für kein x bzw. für nicht ein x)

Mengenoperatoren

$M_1 \cap M_2$ - Durchschnitt der Mengen M_1 und M_2 (Ergebnis ist die Menge der Elemente, welche sowohl in M_1 wie auch in M_2 enthalten sind)
M_1 / M_2 - Differenz der Mengen M_1 und M_2 (Ergebnis ist die Menge der Elemente, welche in M_1 und nicht in M_2 enthalten sind)

Intervalle

$[a, b]$ - abgeschlossenes Intervall mit linker Grenze a und rechter Grenze b, so dass $a \leq x \leq b$

Optimierungsziele

$f(\vec{x}) \rightarrow \max$ - Optimierungziel der Maximierung eines Funktionswertes (gesucht ist der maximale Wert einer Funktion $f(\vec{x})$)

$f(\vec{x}) \rightarrow \min$ - Optimierungziel der Minimierung eines Funktionswertes (gesucht ist der minimale Wert einer Funktion $f(\vec{x})$)

Abstandsnormen

$\|\vec{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$ - p-Norm

$\|\vec{x}\|_\infty = \max_{i=1}^{n} |x_i|$ - Unendlichkeitsnorm

Symbole

$(\mu/\rho\#\lambda)$ - Notation für Evolutionsstrategien, wobei μ und ρ die Anzahl der Eltern und λ die Anzahl der Nachkommen ist

$(\mu\#\lambda)$ - Notation für Evolutionsstrategien, wobei μ die Anzahl der Eltern und λ die Anzahl der Nachkommen ist

$(\mu + \lambda)$ - Notation für Evolutionsstrategien, wobei μ die Anzahl der Eltern und λ die Anzahl der Nachkommen ist

(μ, λ) - Notation für Evolutionsstrategien, wobei μ die Anzahl der Eltern und λ die Anzahl der Nachkommen ist

(R, Q) - Notation einer Fertigungspolitik (R Fertigungszeitpunkt, Q Anzahl zu fertigender Teile)

(R, s, S) - Notation einer Fertigungspolitik (R Fertigungszeitpunkt, s untere Grenze, S obere Grenze)

(s, nQ) - Notation einer Fertigungspolitik (s untere Grenze, Q Anzahl zu fertigender Teile, n Anzahl der Q)

(s, S) - Notation einer Fertigungspolitik (s untere Grenze, S obere Grenze)

(s, Q) - Notation einer Fertigungspolitik (s untere Grenze, Q Anzahl zu fertigender Teile)

(s, T) - Notation einer Fertigungspolitik (s untere Grenze, T Fertigungszeit)

$(s, \min Q, T)$ - Notation einer Fertigungspolitik (s untere Grenze, Q Anzahl zu fertigender Teile, T Fertigungszeit)

$\{\vec{f}(\vec{x})\}$ - Menge von Vektoren, welche jeweils die Zielfunktionswerte für die Werte der Entscheidungsvariablen $\vec{x}$ enthalten

$\{\vec{f}(\vec{x}_{opt})\}$ - Menge von Vektoren, welche jeweils die Zielfunktionswerte für die optimalen Werte der Entscheidungsvariablen $\vec{x}$ enthalten

$\{g(\vec{x})\}$ - Nebenbedingungsvektor mit den Ungleichheitsbedingungen (allgemein)

$\{g(\vec{x}), \vec{f_A}(\vec{x}), \vec{f_N}(\vec{x}))\}$ - Nebenbedingungsvektor mit den Ungleichheitsbedingungen (speziell, unter Berücksichtigung der zu bewertenden und nicht zu bewertenden Ausgabewerte eines Berechners oder Simulators)

$\{h(\vec{x})\}$ - Nebenbedingungsvektor mit den Gleichheitsbedingungen (allgemein)

$\{h(\vec{x}, \vec{f_A}(\vec{x}), \vec{f_N}(\vec{x}))\}$ - Nebenbedingungsvektor mit den Gleichheitsbedingungen (speziell, unter Berücksichtigung der zu bewertenden und nicht zu bewertenden Ausgabewerte eines Berechners oder Simulators)

$\{\vec{x}^{real}\}$ - Menge der reellwertigen Entscheidungsvariablen

$\{\vec{x}^{permut}\}$ - Menge der Permutationsentscheidungsvariable

2^{Ω} - Potenzmenge der Menge Ω

δ_{lopt} - positiver, reeller Wert für die Epsilon-Umgebung der Zielfunktionswerte eines lokalen Extrema

$\delta_{abs}^{E_{\Delta t_P, k}}$ - absoluter Wert für die Änderung des k-ten Zielfunktionswertes von seinem Erwartungswert

$\delta_{abs}^{Var_{\Delta t_P, k}}$ - absoluter Wert für die Änderung des k-ten Zielfunktionswertes von seiner Varianz

$\delta_{rel}^{E_{\Delta t_P, k}}$ - relativer Wert für die Änderung des k-ten Zielfunktionswertes von seinem Erwartungswert

$\delta_{rel}^{Var_{\Delta t_P, k}}$ - relativer Wert für die Änderung des k-ten Zielfunktionswertes von seiner Varianz

Δf - Betrag der Differenz der Zielfunktionswerte aus aktuellen und neuen Werten für die Entscheidungsvariablen

Δt - Zeitdauer, um welche die transiente Phase verlängert wird, um die Dauer der stationären Phase zu bestimmen

Δt_P - Prüfintervall zur Prüfung auf vorzeitiges Ende des aktuellen Simulationslaufes beim Epsilon-Abbruch-Verfahren

ϵ - positiver, reeller Wert für eine Epsilon-Umgebung
ϵ_{lopt} - positiver, reeller Wert für die Epsilon-Umgebung der Werte der Entscheidungsvariablen eines lokalen Extrema
ϵ_{SA} - Toleranzbereich, welcher den Grad der Absenkung der Temperatur θ zwischen zwei Temperaturniveaus angibt
$\epsilon_{abs}^{E_{\Delta t_P,k}}$ - Epsilon-Wert für die Prüfung der absoluten Änderung des k-ten Zielfunktionswertes von seinem Erwartungswert
$\epsilon_{rel}^{E_{\Delta t_P,k}}$ - Epsilon-Wert für die Prüfung der relativen Änderung des k-ten Zielfunktionswertes von seinem Erwartungswert
$\epsilon_{abs}^{Var_{\Delta t_P,k}}$ - Epsilon-Wert für die Prüfung der absoluten Änderung des k-ten Zielfunktionswertes von seiner Varianz
$\epsilon_{rel}^{Var_{\Delta t_P,k}}$ - Epsilon-Wert für die Prüfung der relativen Änderung des k-ten Zielfunktionswertes von seiner Varianz

θ - Temperatur, welche die Relevanz der relativen Evaluierungswerte angibt
θ_{min} - minimale Temperatur, welche die untere Grenze für θ angibt
θ_{start} - Anfangstemperatur für das Verfahren der Simulierten Abkühlung

λ - Ankunftsrate
λ_i - Kundenankunftsrate für Produkt i

μ - Produktions- bzw. Fertigungsrate
μ_i - Produktionsrate für Produkt i
μ_{ik} - Fertigungszeit für Produkt i auf der Fertigungseinheit k
$\mu(t)$ - Mittelwertfunktion für t

π - bijektive Abbildung einer Permutation auf eine Permutation
π - aktuelle Strategie für die Bestimmung von Reihenfolge, Zuordnung und Losgröße
$\vec{\pi}$ - Permutationsvektor
π_i - i-ter Wert des Permutationsvektors, welcher die Funktion $p(j)$ zurückliefert
Π - Menge aller Strategien π
Π^n - Permutationsmenge mit n Elementen

ρ - Auslastung
ρ_{mittel} - mittlere Auslastung
$\rho_{schwach}$ - schwache Auslastung
ρ_{stark} - starke Auslastung

$\sigma^2(t)$ - Varianzfunktion für t

$\varphi(\vec{f}(\vec{x}))$ - Funktion zur Bestimmung der Güte einer Lösung

ω - Elementarereignis
Ω - Menge der Elementarereignisse

$auft_{next}(t)$ - nächster zur Fertigung auszuwählender Fertigungsauftrag zum Zeitpunkt t

b_i - maximale Länge der Bedarfswarteschlange für das Produkt i
`B[`p, n`]` - Darstellung der Binomialverteilung im `CAOS` (p Erfolgswahrscheinlichkeit, n Anzahl der Wiederholungen)

c_{cpu} - Normalisierungskonstante abhängig von der Prozessorarchitektur, u. Ä.
$c_i(\Delta t)$ - höchste zu erwartende Kosten bei dynamischer Reihenfolgestrategie Dynamic
$\mathrm{conv}(f,\, I_f)$ - Konvexität der Funktion f innerhalb des Intervalls I_f
$\cos(x)$ - Cosinus für den Wert x
$\mathrm{cov}(t_x,\, t_y)$ - Kovarianzfunktion für t_1 und t_2
ct_{real} - wahre Realzeit
$\overline{ct}_{speedup}$ - Abschätzung über den Zeitbedarf bei verteilter und paralleler Optimierung
ct_{virt} - virtuelle CPU-Zeit
$ct^{max}_{speedup}$ - maximaler Zeitbedarf bei verteilter und paralleler Optimierung
$ct^{min}_{speedup}$ - minimaler Zeitbedarf bei verteilter und paralleler Optimierung
$ct^{real}_{speedup}$ - realer Zeitbedarf bei verteilter und paralleler Optimierung
C - Verlust
C_i - Fertigstellungstermin des Auftrages i [dt.; completion time [engl.]]
$C(\pi)$ - Verlust unter Anwendung der Strategie π

d_i - Kundenbedarfsmenge für Produkt i
$\dim(\vec{x})$ - Funktion, welche die Anzahl der Elemente des Vektors $\vec{x}$ zurückliefert
D_i - absolute Terminabweichung des Auftrages i [dt.; absolute deviation [engl.]]
`D[`$v_1 : p_1; v_2 : p_2; \ldots$`]` - Darstellung der Mehrpunktverteilung im `CAOS` (v_i konkreter i-ter Wert, p_i Wahrscheinlichkeit für v_i)

E_i - Frühzeitigkeit des Auftrages i [dt.; earliness [engl.]]
`E[`λ`]` - Darstellung der Exponentialverteilung im `CAOS` (λ Ausfallrate)
$E[f_t(\omega)]$ - Erwartungswert des (stochastischen) Prozesses $f_t(\omega)$
$E[\overline{f}_{A_i}(\vec{x})]$ - Erwartungswert des Ausgabewertes $\overline{f}_{A_i}(\vec{x})$
$E[f_{t,\,k}(\vec{x})]$ - Erwartungswert des k-ten Zielfunktionswertes zum Zeitpunkt t

$E[X_t]$ - Erwartungswert der Zeitreihe X_t

`Ek[`k, λ`]` - Darstellung der Erlangverteilung im `CAOS` (k Ordnung der Verteilung, λ Ausfallrate)

$E(t_i^P)$ - erwartete Produktionszeit für das Produkt i und evtl. anfängliche Umrüstzeit st_i

f - (reelle) Abbildung auf einen reellen Wert

f_{rand} - Funktion zur Erzeugung einer (Pseudo)Zufallszahl

$f(\vec{x})$ - Zielfunktionswert einer Lösung

$\vec{f}(\vec{x})$ - Vektor mit den Zielfunktionswerten für die Werte der Entscheidungsvariablen $\vec{x}$

$\vec{f}(\vec{x}_{opt})$ - Vektor mit den Zielfunktionswerten einer/der optimalen Lösung

$f(t, \omega)$ - Abbildung mittels der ein stochastischer Prozess erzeugt wird

$f_i(\vec{x})$ - i-ter Zielfunktionswert für die Werte der Entscheidungsvariablen $\vec{x}$

$f_j(w \cdot C)$ - gewichtete Fertigstellungstermine mit w_i für die Gewichtung des Auftrages i

$f_j(C_{max})$ - letzter Fertigstellungstermin aller Aufträge

$f_j(L_{max})$ - maximale Verspätung aller Aufträge

$f_j(T)$ - Summe der Verspätungen aller Aufträge

$f_j(T^{anz})$ - Anzahl aller verspäteten Aufträge

$f_k(\vec{x}_{best})$ - k-ter Zielfunktionswert der derzeit besten Werte der Entscheidungsvariablen $\vec{x}_{best}$

$\vec{f}_A(\vec{x})$ - Vektor mit den zu bewertenden Ausgabewerten eines Berechners oder Simulators

$\vec{f}_{A_i}(\vec{x})$ - i-tes Element des Vektors mit den zu bewertenden Ausgabewerten eines Berechners oder Simulators

$\overline{f}^j_{A_i}(\vec{x})$ - gemittelte Ausgabewerte von A_i des j-ten Simulationslaufes

$\vec{f}_N(\vec{x})$ - Vektor mit den nicht zu bewertenden Ausgabewerten eines Berechners oder Simulators

$\vec{f}_{N_i}(\vec{x})$ - i-tes Element des Vektors mit den nicht zu bewertenden Ausgabewerten eines Berechners oder Simulators

F_{Max} - Menge der zu maximierenden Zielfunktionen

F_{Min} - Menge der zu minimierenden Zielfunktionen

$F(x)$ - Verteilungsfunktion für x

$F_t(x \mid X_0)$ - Verteilungsfunktion der Zeitreihe X_t mit dem Startzustand X_0

$\overrightarrow{goals}$ - Vektor mit den verwendeten Zielfunktionen für die Aufträge $\overrightarrow{jobs}$

Gen_{max} - maximale Anzahl der Generationen

GA_{gen} - Genetischer Algorithmus (generationsbasiert)

GA_{ss} - Genetischer Algorithmus (stetige Ersetzung)

hc_i - Lagerkosten für Produkt i

$\widetilde{hc}_i(t)$ - erwartete Holdingkosten zum Zeitpunkt t für das Produkt i
HC - Lagerkosten
$HP_{M^{parV}_{HybOpt}}$ - Notation für das parallel-hybride Optimierungsverfahren
$HS_{ov^{seqEV}_{HybOpt}, ov^{seqVV}_{HybOpt}}$ - Notation für das seriell-hybride Optimierungsverfahren
$HC(\pi)$ - Lagerkosten pro Zeiteinheit für alle Produkte unter Anwendung der Strategie π
$HC_i(\pi)$ - Gesamtlagerkosten für das Produkt i unter Anwendung der Strategie π

i - individuelle Laufvariable für einen Index; gibt z. B. den i-ten Wert eines Vektors an
i_P - Nummer des aktuellen Prüfzeitpunktes
I - Indexmenge
In - Probleminstanz
I_f - Intervall für die Funktion f
$IN_{CalcSim}$ - Bestandteil eines Modells mit den Eingabewerten für einen analytischen Berechner oder Simulator (Teil eines Modells zur modellgestützten, simulationsbasierten Optimierung)
IN_{Opt} - Bestandteil eines Modells mit den Eingabewerten für einen Optimierer (Teil eines Modells zur modellgestützten, simulationsbasierten Optimierung)
$I_i(t)$ - Inventarmenge I von Produkt i zum Zeitpunkt t
$I_i^+(t)$ - Anzahl der physisch verfügbaren Teile des Produktes i zum Zeitpunkt t
$I_i^-(t)$ - Anzahl der vorgemerkten Teile für das Produkt i zum Zeitpunkt t

$\overrightarrow{jobs}$ - Vektor mit den (Fertigungs-)Aufträgen

$kanb_i$ - Anzahl der Kanbans für Produkt i
K - Anzahl der Dispositions- und Fertigungsstufen

l_{TL} - Länge der Tabuliste
L_i - Verspätung des Auftrages i [dt.; lateness [engl.]]
$\overrightarrow{LFS}$ - Menge der aktuellen Losgrößen- und Freigabestrategien
LFS_i - aktuelle Losgrößen- und Freigabestrategie für das Produkt i
$\texttt{LN[}\mu, \sigma^2\texttt{]}$ - Darstellung der logarithmischen Normalverteilung im CAOS (μ Erwartungswert, σ^2 Varianz/Streuung)

m - individuelle Dimension eines Vektors; gibt z. B. die Dimension des Vektors mit den Zielfunktionen an
$m(\{\vec{x}^{real}\}, \{\vec{x}^{permut}\})$ - Abbildungsfunktion für reellwertige Entscheidungsvariablen und Permutationsentscheidungsvariablen
max - Optimierungziel der Maximierung eines Funktionswertes (gesucht ist der maximale Wert einer Funktion)

min	-	Optimierungziel der Minimierung eines Funktionswertes (gesucht ist der minimale Wert einer Funktion)
M	-	Anzahl der (parallelen) Fertigungseinheiten
M_{cur}	-	Menge mit den potentiellen Werten für die Entscheidungsvariablen
$\overline{M}_{cur}$	-	Menge der noch zu untersuchenden Lösungen der Nachbarschaft $N(\vec{x}_{cur})$
$\underline{M}_{cur}$	-	Menge der bereits untersuchten Lösungen der Nachbarschaft $N(\vec{x}_{cur})$
M_{dis}	-	Menge mit (ausgewählten) diskreten Werten
M_{tabu}	-	Menge der „tabu“-gesetzten Werte für die Entscheidungsvariablen
$M_{cross}^{permut/real}$	-	Menge der Rekombinationsoperatoren
$M_{mutate}^{permut/real}$	-	Menge der Mutationsoperatoren
$M_{select}^{permut/real}$	-	Menge der Selektionsoperatoren
M_{HybOpt}	-	Menge der derzeit verwendbaren Optimierungsverfahren
M_{HybOpt}^{parV}	-	Menge der gleichberechtigt nutzbaren verschiedenen Optimierungsverfahren für das parallel-hybride Optimierungsverfahren
$M_i(t)$	-	Anzahl der Teile, welche in den Fertigungsaufträgen für das Produkt i zum Zeitpunkt t im Pool mit den Losen für die Fertigungsaufträge existieren

n	-	individuelle Dimension eines Vektors; gibt z. B. die Dimension des Vektors mit den Werten einer Entscheidungsvariablen an
n_1	-	Anzahl der elitären Individuen
n_2	-	Anzahl der selektierten Individuen
n_3	-	Anzahl der rekombinierten Individuen
n_4	-	Anzahl der mutierten Individuen
n_{add}	-	Anzahl der Prüfzeitpunkte, bei denen das Endes der transienten Phase noch nicht erreicht ist
n_c	-	Kunden insgesamt
n_{rc}	-	abgewiesene Kunden insgesamt
n_{sim}	-	Anzahl der (durchgeführten/durchzuführenden) Simulationsläufe
n_{st}	-	Multiplikator für die Bestimmung der Dauer der stationären Phase
n_t	-	Anzahl der Teilungen des Zeitraumes
n_{tr}	-	Multiplikator für die Bestimmung der Dauer der transienten Phase
n_L	-	Anzahl der Lösungen bei verteilter und paralleler Optimierung
n_{OV}	-	Anzahl der Optimierungsverfahren
n_{Pr}	-	Anzahl der Prozessoren
n_S	-	Anzahl der Slaves
n_{X_t}	-	Anzahl der Teilungen des Zustandraumes

$n_{elite}^{permut/real}$ - Anzahl elitärer Individuen (Anzahl der Individuen, welche ohne Veränderung in die nächste Generation zu übernehmen sind)

$n_{indGen}^{permut/real}$ - Individuenanzahl (Anzahl der Individuen in einer Generation)

$n_{offset}^{permut/real}$ - Offset-Anzahl (Anzahl der Individuen, deren Fitness gleichzeitig berechnet werden kann)

$n_S^{max}(i)$ - maximale Anzahl parallel verarbeitbarer Lösungen des Optimierungsverfahrens i bei verteilter und paralleler Optimierung

N - Anzahl der zu fertigenden Produkte

$\mathcal{N}_{In}$ - Nachbarschaft bzgl. der Probleminstanz In

$\mathtt{N}[\mu, \sigma^2]$ - Darstellung der Normalverteilung im CAOS (μ Erwartungswert, σ^2 Varianz/Streuung)

$\mathcal{N}(\vec{x})$ - Nachbarschaft des Vektors $\vec{x}$ bei festgelegter Probleminstanz In

$N(\vec{x}_{cur})$ - Nachbarschaft von $\vec{x}_{cur}$

$\mathcal{N}_{permut(\epsilon)}(\vec{x})$ - Nachbarschaft für die Werte der Permutationsentscheidungsvariablen x

$\mathcal{N}_{real(\epsilon)}(\vec{x})$ - Nachbarschaft für die Werte der reellwertigen Entscheidungsvariablen x

$\mathcal{N}_{In}(\vec{x})$ - Nachbarschaft des Vektors $\vec{x}$ bzgl. der Probleminstanz In

ov_{HybOpt}^{seqEV} - Eröffnungsverfahren aus der Menge M_{HybOpt}

ov_{HybOpt}^{seqVV} - Verbesserungsverfahren aus der Menge M_{HybOpt}

$O_i(t)$ - Länge der Bedarfswarteschlange für das Produkt i zum Zeitpunkt t

OUT_{Ass} - Bestandteil eines Modells mit den zu bewertenden Ausgabewerten eines Berechners oder Simulators

OUT_{NoAss} - Bestandteil eines Modells mit den nicht zu bewertenden Ausgabewerten eines Berechners oder Simulators

OV_{xy} - Optimierungsverfahren xy für beliebige Entscheidungsvariablen

OV_{xy}^{permut} - Optimierungsverfahren xy für Permutationsentscheidungsvariablen

OV_{xy}^{real} - Optimierungsverfahren xy für reellwertige Entscheidungsvariablen

p - Periode

p_i - individueller Lagerbedarf für Produkt i

$p_{cross}^{permut/real}$ - Rekombinationswahrscheinlichkeit (Wahrscheinlichkeit für die Rekombination zweier selektierter Individuen)

$p_{mutate}^{permut/real}$ - Mutationswahrscheinlichkeit (Wahrscheinlichkeit für die Mutation zweier selektierter Individuen)

$p_{select}^{permut/real}$ - Selektionswahrscheinlichkeit (Wahrscheinlichkeit für die Selektion eines Individuums)

$p(j)$ - Funktion, welche den Wert, den Bezeichner, die Zeichenkette, o. Ä. für ein Element eines Permuationsvektors zurückliefert

pc_i - Produktionskosten für Produkt i
$\widetilde{pc}_i(t)$ - erwartete Produktionskosten zum Zeitpunkt t für das Produkt i
$\text{pool}(t)$ - Menge der im Pool mit den Losen für die Fertigungsaufträge befindlichen Lose zum Zeitpunkt t
$\text{pool}_i(t)$ - i-tes Los der Menge der im Pool mit den Losen für die Fertigungsaufträge befindlichen Lose zum Zeitpunkt t
P - Gesamtlagergröße
P_0 - Problemklasse
P_i - Gesamtlagergröße für Produkt i
P_{SA} - Wahrscheinlichkeit mit der neue, schlechtere Werte für die Entscheidungsvariablen akzeptiert werden
`P[λ]` - Darstellung der Poissonverteilung im `CAOS` (λ Ereignisrate)
$P(x)$ - Wahrscheinlichkeit für den Ausdruck x
PC - Produktionskosten
$PC(\pi)$ - Produktionskosten pro Zeiteinheit für alle Produkte unter Anwendung der Strategie π
$PC_i(\pi)$ - Gesamtproduktionskosten für das Produkt i unter Anwendung der Strategie π

r_n - n-te erzeugte (Pseudo)Zufallszahl
r_{SA} - lokaler Wiederholungsfaktor, welcher angibt, wie lange auf einem bestimmten Temperaturniveau nach besseren Werten für die Entscheidungsvariablen gesucht werden soll
r_{mutate}^{real} - Mutationsrate (Rate für die Mutation zweier zur Mutation ausgewählter Individuen)
rc_i - Abweiskosten für Produkt i
$\widetilde{rc}_i(t)$ - erwartete Abweiskosten zum Zeitpunkt t für das Produkt i
$\text{rand}(a, b)$ - Funktion, welche eine zufällig gleichverteilte Zahl i aus dem Intervall $[a, b]$ zurückliefert
RC - Abweiskosten
$RC(\pi)$ - Abweiskosten pro Zeiteinheit für alle Produkte unter Anwendung der Strategie π
$RC_i(\pi)$ - Gesamtabweiskosten für das Produkt i unter Anwendung der Strategie π
RS - aktuelle Reihenfolgestrategie

s_i - Bestell-/Freigabezeitpunkt (untere Schranke) für das Produkt i, z. B. bei (s, nQ)- oder (s, S)-Strategie
s_i - i-ter Startwert eines (Pseudo)Zufallszahlengenerators
$\vec{sc}_i$ - Umrüstkosten für Produktwechsel zu Produkt i
$\vec{st}_i$ - Umrüstzeiten für Produktwechsel zu Produkt i
$\widetilde{sc}_i(t)$ - erwartete Umrüstkosten zum Zeitpunkt t für das Produkt i
$\sin(x)$ - Sinus für den Wert x

S_i - Produktionsgrenze (obere Schranke) für das Produkt i, z. B. bei (s, nQ)- oder (s, S)-Strategie
S_{In} - Suchraum der Probleminstanz In
SC - Umrüstkosten
SC_{Gen} - Abbruchkriterium des generationsbasierten Genetischen Algorithmus
$SC(\pi)$ - Umrüstkosten pro Zeiteinheit bei Verwendung der Strategie π
$SC_{ij}(\pi)$ - Gesamtrüstkosten für das Produkt i bei Produktwechsel zum Produkt j unter Anwendung der Strategie π
$SC_{\Delta S}$ - zustandsabhängiges Abbruchkriterium
$SC_{\Delta S,\epsilon}$ - zustandsabhängiges Abbruchkriterium
$SC_{\Delta T}$ - zeitabhängiges Abbruchkriterium
$SC_{\Delta T,\epsilon}$ - zeitabhängiges Abbruchkriterium
SC_S - zustandsabhängiges Abbruchkriterium
SC_T - zeitabhängiges Abbruchkriterium
`S[`v`]` - Darstellung der Einpunktverteilung im `CAOS` (v konkreter Wert)
$SC(\pi)$ - Umrüstkosten pro Zeiteinheit für alle Produkte unter Anwendung der Strategie π

^{TM} - Warenzeichen [dt.; trademark [engl.]]
t_i - Zeitpunkt i
$t_{\Delta tr}$ - Prüfintervall zur Bestimmung des Zeitpunktes des Endes der transienten Phase
t_{akt} - aktuelle Simulationszeit
t_{cl} - Zeitpunkt einer Kundenankunft
t_{prod} - Zeitpunkt des Produktionsbeginns
t_{prf} - Prüfzeitpunkt für die Inventarmenge
$t_{release}$ - Zeitpunkt der Freigabe eines Fertigungsauftrages
t_{st} - Zeitpunkt des Endes der stationären Phase
t_{tr} - Zeitpunkt des Endes der transienten Phase
t_{E_i} - Ereigniszeitpunkt des Ereignisses E_i
T - Anzahl der Fertigungsperioden
T - Menge der Beobachtungszeitpunkte
T_i - Verspätung bzw. Unpünktlichkeit des Auftrages i [dt.; tardiness [engl.]]

u_i - i-te erzeugte gleichverteilte Zufallszahl
U_i - Strafe pro Verspätung [dt.; unit penalty [engl.]]
`U[`a, b`]` - Darstellung der (stetigen) Gleichverteilung im `CAOS` (a linke Grenze, b rechte Grenze)

$Var[\overline{f}_{A_i}(\vec{x})]$ - Varianz des Ausgabewertes $\overline{f}_{A_i}(\vec{x})$
$Var^2[f_t(\omega)]$ - Varianz des (stochastischen) Prozesses $f_t(\omega)$

$\vec{w}$	-	Vektor mit den Gewichten zur Gewichtung der zu bewertenden Ausgabewerte eines Berechners oder Simulators
w_i	-	einzelnes Gewicht zur Gewichtung von einem zu bewertenden Ausgabewert eines Berechners oder Simulators
wc_i	-	Wartekosten für Produkt i
$\widetilde{wc}_i(t)$	-	erwartete Wartekosten zum Zeitpunkt t für das Produkt i
wsl	-	Warteschlangenlänge
wsl_{max}	-	maximale Warteschlangenlänge
WC	-	Wartekosten
`W[`α, β`]`	-	Darstellung der Weibullverteilung im `CAOS` (α charakteristische Lebensdauer, β Ausfallsteilheit)
$WC(\pi)$	-	Wartekosten pro Zeiteinheit für alle Produkte unter Anwendung der Strategie π
$WC_i(\pi)$	-	Gesamtwartekosten für das Produkt i unter Anwendung der Strategie π

$\vec{x}$	-	Werte für die Entscheidungsvariable x
x_i	-	stetiger i-ter Wert der Entscheidungsvariable x
x_{ij}	-	j-ter Wert der i-ten Entscheidungsvariable x
$\vec{x}_{best}$	-	(bisherige) (sub)optimale Werte für die Entscheidungsvariable x
$\vec{x}_{gopt}$	-	global-optimale Werte für die Entscheidungsvariable x
$\vec{x}_i$	-	Werte der i-ten Entscheidungsvariablen
$\vec{x}_{lopt}$	-	lokal-optimale Werte für die Entscheidungsvariable x
$\vec{x}_{opt}$	-	(bestimmte) (sub)optimale Werte für die Entscheidungsvariable x
$\widetilde{x}_i$	-	diskretisierter i-ter Wert der Entscheidungsvariable x
$x_{SC(\pi)}$	-	obere Schranke für die Umrüstkosten pro Zeiteinheit bei Verwendung der Strategie π
x_i^l	-	untere Grenze für den i-ten Wert der Entscheidungsvariable x
x_i^s	-	Schrittweite für den i-ten Wert der Entscheidungsvariable x
x_i^u	-	obere Grenze für den i-ten Wert der Entscheidungsvariable x
$\vec{x}_{cur}$	-	Vektor mit den aktuellen Werten für die Entscheidungsvariablen
$\vec{x}_{new}$	-	Vektor mit den Werten für die neuen Entscheidungsvariablen
$\vec{x}_{start}$	-	Vektor mit den Startwerten für die Entscheidungsvariablen (Startvektor)
$\vec{x}_{sQ}$	-	Entscheidungsvariable der Fertigungsstrategie (s, nQ)
$\vec{x}_{sS}$	-	Entscheidungsvariable der Fertigungsstrategie (s, S)
$\vec{x}_i^{permut}$	-	i-te Permutationsentscheidungsvariable x
$\vec{x}_i^{real}$	-	i-te reellwertige Entscheidungsvariable x
X	-	Zufallszahl
$\mathcal{X}$	-	möglicher Bereich für die Werte der Entscheidungsvariablen
$\mathcal{X}_{gopt}$	-	Menge der globalen Pareto-optimalen Menge

X_0 - Startzustand
X_i - Zufallsgröße
X_t - Zeitreihe
$\mathcal{X}_{lopt}$ - Menge der lokalen Pareto-optimalen Menge
$\mathcal{X}_{Post}$ - zulässiger Bereich für die Werte der Entscheidungsvariablen, welcher erst nach der Durchführung einer Berechnung oder Simulation bestimmt werden kann
$\mathcal{X}_{Pre}$ - zulässiger Bereich für die Werte der Entscheidungsvariablen, welcher im Vorfeld einer Berechnung oder Simulation bestimmt werden kann
$\mathcal{X}_{Zul}$ - zulässiger Bereich für die Werte der Entscheidungsvariablen ($\mathcal{X}_{Pre} \cap \mathcal{X}_{Post}$)
$X_i(\Omega)$ - Abbildung der Menge mit den Elementarereignissen Ω auf eine reelle Zahl für die Zufallsgröße X_i

$z(i)$ - i-tes Element des Zyklus bei zyklischer Reihenfolgestrategie
Z - Zyklus bei zyklischer Reihenfolgestrategie Cyclic
Z_i - Zustand i
ZS - aktuelle Zuordnungsstrategie

Sonstige Symbole

® - eingetragenes Warenzeichen [dt.; registered trademark [engl.]]

Abkürzungsverzeichnis

Abb.	-	Abbildung
Abschn.	-	Abschnitt
AHP	-	Analytical Hierarchy Process [engl.]
BOP	-	Boolean Optimisation Problem [engl.; boolsches Optimierungsproblem [dt.]]
bspw., Bspw.	-	beispielsweise, Beispielsweise
bzgl.	-	bezüglich
bzw.	-	beziehungsweise
ca.	-	circa
`CAO`	-	`CalculationOptimisationAssessment` (Bibliothek des Softwaresystems `CAOS`)
`CAOS`	-	`CalculationAssessmentOptimisationSystem` [engl.]
`CHiC`	-	Chemnitzer Hochleistungs-Linux-Cluster
CIL	-	Common Intermediate Language [engl.]
CLI	-	Common Language Infrastructur [engl.]
`CLiC`	-	Chemnitzer Linux Cluster
CLR	-	Common Language Runtime [engl.]
CLSP	-	Capacitated Lot Scheduling Problem [engl.]
CLSP-SC	-	Capacitated Lot Scheduling Problem with Setup Carryover [engl.]
CLSPL	-	Capacitated Lot Scheduling Problem Linked lot-sizes [engl.]
COP	-	Combinatorial Optimisation Problem [engl.; kombinatorisches Optimierungsproblem [dt.]]
Corp.	-	Corporation [engl.]
CSLP	-	Continuous Setup Lot-sizing Problem [engl.]
CSLSP	-	Capacitated Stochastic Lot Scheduling Problem [engl.; kapazitiertes stochastisches Losgrößenproblem [dt.]]
Def.	-	Definition
d. h.	-	das heißt
DLL	-	Dynamic Link Library [engl.; dynamisch eingebundene Bibliothek [dt.]]
DLSP	-	Discrete Lot Scheduling Problem [engl.]
Dr.	-	Doktor

dt.	-	deutsch
ECMA	-	European Computer Manufacturers Association [engl.]
EcoSyL	-	Eco System Language [engl.]
EDV	-	Elektronische Datenverarbeitung
eIKG	-	explizite Inverse KongruenzGeneratoren
ELSP	-	Economic Lot Scheduling Problem [engl.]
engl.	-	englisch
EOQ	-	Economic Order Quantity [engl.]
et. al.	-	et alii [lat.; und andere [dt.]]
etc.	-	et cetera [lat.; und so weiter [dt.]]
EVA	-	Eingabe-Verarbeitung-Ausgabe
evtl.	-	eventuell
f.	-	folgend
FCL	-	Framework Class Library [engl.]
FE	-	Fertigungseinheit(en)
ff.	-	folgende
GA	-	Genetic Algorithm [engl.; Genetischer Algorithmus [dt.]]
GD	-	Great Deluge [engl.; Sintflutalgorithmus [dt.]]
g. d. w.	-	genau dann, wenn
GE	-	Geldeinheit(en)
ggf.	-	gegebenenfalls
GIGO	-	Garbage-In-Garbage-Out [engl.]
GLS	-	Guided Local Search [engl.; geführte lokale Suche [dt.]]
GRASP	-	Greedy Randomized Adaptive Search Procedures [engl.; gierige, zufällige, adaptive Suchprozeduren [dt.]]
habil.	-	habilitatus [lat.]
HC	-	Hill Climbing [engl.; Bergsteigen [dt.]]
i. Allg.	-	im Allgemeinen
i. d. R.	-	in der Regel
IEC	-	International Electrotechnical Commission [engl.]
IKG	-	Inverse KongruenzGeneratoren
Inc.	-	Incorporated [engl.]
IOP	-	Integer Optimisation Problem [engl.; ganzzahliges Optimierungsproblem [dt.]]
ISO	-	International Organization for Standardization [engl.]
JIT	-	Just-In-Time [engl.]

JSSP	-	Job Shop Scheduling Problem [engl.; Einplanungsproblem von Fertigungsaufträgen [dt.]]
Kap.	-	Kapitel
kLKG	-	kombinierte Lineare KongruenzGeneratoren
lat.	-	lateinisch
LFS	-	Losgrößen- und Freigabestrategie
LKG	-	Lineare Kongruenzgeneratoren
Ltd.	-	Limited [engl.]
MA	-	Metropolis Algorithm [engl.; Metropolisalgorithmus [dt.]]
mLKG	-	multiplikative Lineare KongruenzGeneratoren
MLLP	-	Multi-Level Lot-sizing Problem [engl.]
MOP	-	Multiobjective Optimisation Problem [engl.; mehrkriterielles Optimierungsproblem [dt.]]
`MPI`	-	Message Passing Interface [engl.]
MPM	-	Multi Purpose Machines [engl.; Mehr-Zweck-Maschinen [dt.]]
$\mathcal{NP}$	-	Non-deterministic Polynominal-time [engl.]
o. Ä.	-	oder Ähnliches
o. B. d. A.	-	ohne Beschränkung der Allgemeinheit
oec.	-	oeconomiae [lat.] (steht im Zusammenhang mit „Dr. oec. “ für einen Doktor der Wirtschaftswissenschaften)
OMG	-	Object Management Group [engl.]
OOP	-	Objekt-Orientierte Programmierung
`OpenMP`	-	Open Multi-Processing [engl.]
`ParMacs`	-	Parallel Macros [engl.]
pLKG	-	parallele Lineare KongruenzGeneratoren
Prof.	-	Professor
`PVM`	-	Parallel Virtual Machine [engl.]
QA	-	Quantum Annealing [engl.; Quantum Abkühlung [dt.]]
RE	-	Raumeinheit(en)
rer. nat.	-	rerum naturalium [lat.] (steht im Zusammenhang mit „Dr. rer. nat. “ für einen Doktor der Naturwissenschaften)
RS	-	Reihenfolgestrategie
S.	-	Seite
SA	-	Simulated Annealing [engl.; Simulierte Abkühlung [dt.]]

SELSP	-	Stochastic Economic Lot Scheduling Problem [engl.]
SHC	-	Stochastic Hill Climbing [engl.; zufälliges Bergsteigen [dt.]]
s. o.	-	siehe oben
sog.	-	so genannte
SOP	-	Singleobjective Optimisation Problem [engl.; einkriterielles Optimierungsproblem [dt.]]
s. u.	-	siehe unten
TA	-	Threshold Accepting [engl.; Schwellenakzeptanz [dt.]]
Tab.	-	Tabelle
`TCGMSG`	-	Theoretical Chemistry Group Message Passing System [engl.]
TE	-	Teil(e)
TR	-	Technical Report [engl.; Technischer Bericht [dt.]]
TS	-	Tabu Search [engl.; Tabusuche [dt.]]
TSP	-	Traveling Salesman Problem [engl.; Problem des Handlungsreisenden [dt.]]
u. a.	-	und anderes
u. Ä.	-	und Ähnliches
UML	-	Unified Modeling Language [engl.; vereinheitlichten Modellierungssprache [dt.]]
usw.	-	und so weiter
u. U.	-	unter Umständen
vgl.	-	vergleiche
VM	-	Virtuelle Maschine
VRPTW	-	Vehicle Routing Problem with Time Windows [engl.; Transportplanungsproblem mit Zeitfenstern [dt.]]
WW	-	Wagner-Whitin
XML	-	eXtensible Markup Language [engl.; erweiterbare Auszeichnungssprache [dt.]]
XSD	-	XML-Schema-Definition
z. B.	-	zum Beispiel
ZE	-	Zeiteinheit(en)
z. Z.	-	zur Zeit

Kapitel 1. Einführung

Inhalt

1.1	*Problemstellungen heutiger ökonomischer Systeme*	*1*
1.2	*Ziele der Arbeit*	*5*
1.3	*Gliederung der Arbeit*	*7*

1.1. Problemstellungen heutiger ökonomischer Systeme

In der Praxis vorkommende Systeme, nachfolgend auch als real existierende Systeme oder kurz als *Realsysteme* bezeichnet, sind durch eine hohe Komplexität infolge verschiedenster einwirkender Faktoren gekennzeichnet. Diese Komplexität macht eine rechnergestützte optimale Steuerung des Gesamtsystems unter Echtzeitbedingungen derzeit kaum möglich. Im heutigen Zeitalter global vernetzter Rechnernetze ist es zwar problemlos möglich, bestimmte Informationen über die aktuellen und abgelaufenen Prozesse innerhalb weniger Sekunden oder Bruchteilen von Sekunden abzufragen, eine allen Optimierungsanforderungen genügende Modellbildung für das Gesamtsystem ist jedoch i. Allg. nicht möglich. Ein Lösungsweg für dieses Problem ist die Zerlegung des Gesamtsystems in einzelne Untermodelle, welche getrennt betrachtet und optimiert werden.

Unter Bezug auf den in der Arbeit gewählten Fokus auf Modelle von Systemen der Produktion und Lagerhaltung kann beispielsweise eine getrennte Betrachtung von Produktions- und Investitionsströmen sinnvoll sein. Hieraus ergibt sich bereits eine erste Problemstellung für den Ersteller eines Modells, den Modellierer. Es wird die Frage nach einer sinnvollen Zerlegung eines zu modellierenden Systems anhand der innerbetrieblichen Abläufe aus einer bestimmten Sichtweise aufgeworfen (siehe Abb. 1.1).

Mit hoher Wahrscheinlichkeit wird sich aus logistischer Sichtweise eine andere Modellzerlegung ergeben als aus einer zeitlich orientierten Sichtweise. Die logistische Sichtweise führt u. a. Betrachtungen eines günstigen innerbetrieblichen Materialflusses und sinnvoller Lagerstrategien durch (siehe [AF05] oder [Tem06]), wäh-

Abb. 1.1

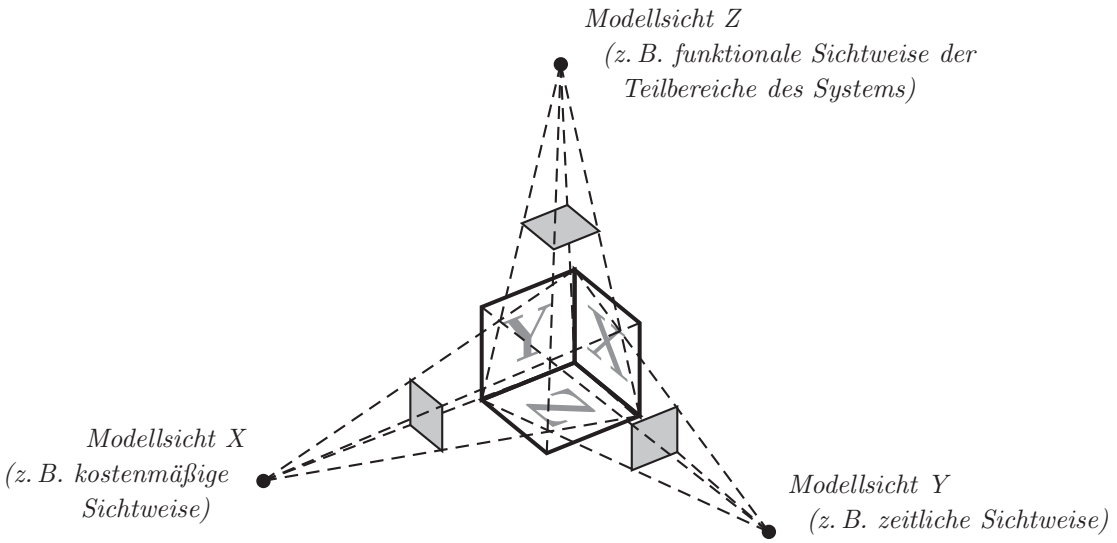


Abb. 1.1.: *Zerlegung eines (ökonomischen) Systems aus verschiedenen Sichtweisen*

renddessen sich die zeitlich orientierte Sichtweise z. B. eher mit Auswirkungen der operativen (kurzfristigen), taktischen (mittelfristigen) oder strategischen (langfristigen) Planung beschäftigt (siehe [Sch02]). Innerhalb dieser Arbeit wird von einer zeitlich orientierten Sichtweise im Hinblick auf eine kurz- und mittelfristige Planung ausgegangen, um die kostenoptimale Lösung des Problems zu einem Produktionssystem mit anschließender Lagerhaltung, nachfolgend als *Produktions- und Lagerhaltungssystem* bezeichnet, anzustreben. Die Betrachtung anderer Sichtweisen oder ein möglicher Vergleich untereinander wären ebenso denkbar, sollen aber nicht Gegenstand der durchgeführten Untersuchungen sein.

Im Gegensatz zur Steigerung der Komplexität von Realsystemen ist die derzeitige Rechentechnik gerade ausreichend weit entwickelt, um ein solches System in seiner gesamten Komplexität auf einem Rechner intern abzubilden und ggf. verschiedene Modellabläufe mittels Animation in Echtzeit graphisch darzustellen. In der Regel ist es aber mit den derzeit industriell eingesetzten Rechnern jedoch nicht möglich, eine optimale Steuerung des gesamten abgebildeten komplexen Realsystems unter Echtzeitanforderungen durchzuführen. Lediglich durch eine Abstraktion der verschiedenen Teilsysteme und den Verzicht auf die Bestimmung der global optimalen Lösung kann es gelingen, Echtzeitanforderungen an den Optimierungsprozess gerecht zu werden. Bei der Untersuchung bisher nicht betrachteter neuartiger Modelle ist eine solche Vorgehensweise aber eher weniger sinnvoll, weil noch keine genauen Aussagen über das Verhalten des modellierten Systems getroffen werden können und somit eine Nachahmung durch abstrahierte Teilsysteme nicht möglich ist. Um dennoch Untersuchungen bzgl. Realsystemen durchführen zu können, wird, unter Verzicht auf Echtzeitanforderungen, u. a. auf Methoden wie die *rechnergestützte Simulation* zurückgegriffen (siehe [Ban98]). Diese ermöglicht es z. B. für verschiedene Szenarien, Vorhersagen über das zukünftige Systemverhalten des abgebildeten Realsystems aufzuzeigen, um eine aus einer bestimmten festgelegten Sichtweise optimale Entscheidung zu treffen. Aus diesem Grund wird bei der Untersuchung von Modellen für Produktions- und Lagerhaltungssysteme auch auf die Methode

der rechnergestützten Simulation sowie der *simulationsbasierten Optimierung* zum Treffen einer kostenoptimalen Produktionsentscheidung zurückgegriffen.

Für die Herausfilterung der wesentlichen Bestandteile eines Realsystems sollte eine Kooperation zwischen einem *Experten des untersuchten Systems* (erste Expertengruppe) und einem *Experten auf dem Gebiet der Modellierung, Simulation und Optimierung* (zweite Expertengruppe) eines solchen Systems erfolgen. Die zweite Expertengruppe kann die Möglichkeiten und Grenzen der aktuellen Forschung und Entwicklung auf dem speziellen Gebiet aufzeigen, wobei sie diese zumeist an bereits existierenden mathematischen Modellen orientieren wird, um die Entscheidung treffen zu können, ob ein neues Modell erstellt werden muss oder auf ein existierendes mit ggf. entsprechenden Änderungen und Erweiterungen zurückgegriffen werden kann. Unter Zuhilfenahme des Wissens der ersten Expertengruppe können dann die notwendigen technologischen und ökonomischen Rand- und Nebenbedingungen des Problems festlegt werden. Für die Untersuchung neuartiger Modelle ist diese Vorgehensweise mit bestimmten Ausnahmen ähnlich sinnvoll. Zwar kann man sich bei diesen an existierenden mathematischen Problemstellungen anlehnen, um eine Klassifizierung des Problems vorzunehmen und aktuelle Entwicklungen zu verdeutlichen, aber die Erstellung neuer Modelle mit einer individuellen Charakteristik ist unabdingbar. Eine solche Vorgehensweise wird auch in der vorliegenden Arbeit gewählt, um eine Klassifizierung des neuartigen Untersuchungsmodells des Produktions- und Lagerhaltungssystems auf der Grundlage existierender Modellansätze durchzuführen.

In diesem Zusammenhang ergibt sich auch die Problemstellung der möglichst exakten *rechnerinternen Modellierung* eines vorliegenden Problems bzw. Realsystems (siehe Abb. 1.2), im Bezug auf die Anwendung der simulationsbasierten Optimierung zur Lösung eines vorliegenden Problems. Es gilt nach wie vor der Grundsatz: So viel wie möglich, aber nur so exakt wie nötig.

Abb. 1.2

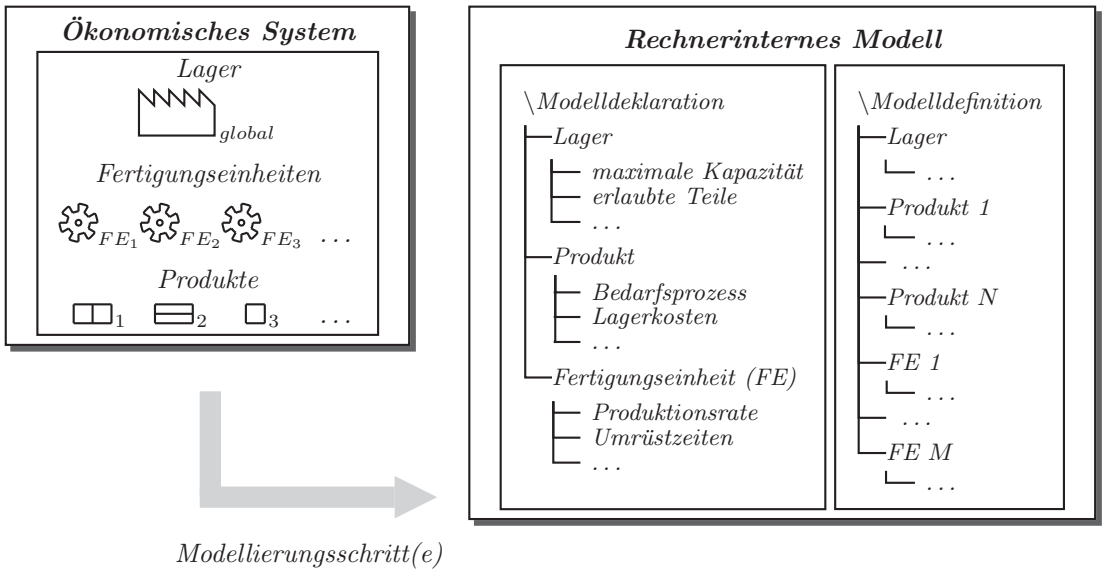


Abb. 1.2.: *Rechnerinterne Modellabbildung eines (ökonomischen) Systems*

Gerade im kommerziellen Bereich gibt es bereits seit einigen Jahren Modellierungs- und Simulationswerkzeuge mit einem weitreichenden Funktionsumfang zur rechnerinternen Repräsentation und einer möglichst exakten graphischen Darstellung eines Modells (evtl. mit Animation eines Simulationslaufes (siehe [RA07])). Bei diesen kommerziellen Softwaresystemen stehen jedoch neben speziellen Kundenwünschen zumeist die Visualisierung des Modells und einzelner Simulationsläufe sowie umfangreiche Schnittstellen zu anderen Softwaresystemen im Vordergrund und weniger die Umsetzung der vielfältigen Möglichkeiten, welche die Modellierung, die Simulation oder auch die Optimierung bieten. Seit der Einführung graphischer Benutzeroberflächen vor über 20 Jahren haben sich die visuellen Gestaltungsmerkmale kommerzieller Simulationswerkzeuge immer mehr verbessert und reichen heute von verschiedenen interaktiven Eingabemöglichkeiten (siehe [Fis01]) bis hin zu dreidimensionalen Darstellungen komplexer ökonomischer Systeme (siehe [Dyn07]). Die vielfältige rechnerinterne Repräsentation eines Modells ist demnach durch die verschiedensten Softwaresysteme bereits sehr gut abgedeckt. Themen wie die Optimierung eines Realsystems, Laufzeitminimierung eines Simulationslaufes oder die Bewertung der Simulationsergebnisse werden hingegen vielfach außer Acht gelassen und als Probleme des Anwenders betrachtet. Allerdings sind diese Gesichtspunkte gerade für die Untersuchung von aktuellen Modellen zu Realsystemen von besonderem Interesse, um das vorhandene Optimierungspotential des Systems besser auszuschöpfen. Zudem besteht bei moderner kommerzieller Simulationssoftware teilweise das Problem, dass die graphische Oberfläche meist nicht ganz abschaltbar ist, was wiederum zu einer *Verlangsamung der Simulation* im Bezug auf die Laufzeit führt. Unter dem Gesichtspunkt der Optimierungsmöglichkeiten verzichten heutige kommerzielle Softwaresysteme oftmals gänzlich auf Optimierungsverfahren, welche sich mit Problematiken wie der *verteilten und parallelen Optimierung* auseinandersetzen. Bei genauerer Recherche kann festgestellt werden, dass die in [Fu01a] angegebene Tabelle mit den genutzten Optimierungsverfahren bis heute noch weitestgehend aktuell ist. Dies macht die Kopplung eines externen Optimierers notwendig, wodurch wiederum kostbare Rechenzeit verloren geht. Zudem ergeben sich häufig lizenzrechtliche Probleme, weil ein Softwareprodukt nicht gleichzeitig auf mehreren Rechnern eingesetzt werden darf. Eine Ausnahme bilden dabei die im Vergleich zu Einzelplatzrechnern deutlich teureren Server-Lizenzen [engl.]. Wird hingegen nicht kommerzielle Simulationssoftware, welche für akademische Zwecke[3], entwickelt ist, damit verglichen, so treten die Probleme in anderer Richtung auf. Bei diesen existiert nach Meinung des Autors teilweise keine oder nur eine spärliche graphische Oberfläche, so dass die Modelleingabe schwer und umständlich i. Allg. über Texteditoren erfolgen muss. Allerdings sind bei diesen nicht kommerziellen Softwaresystemen die internen Algorithmen für die Simulation und Optimierung auf einem höheren Niveau als bei kommerziellen Softwaresystemen, weil sie sich an den aktuellen Forschungsergebnissen orientieren und dadurch die Entwicklung neuer Verfahren und Algorithmen vorantreiben. Nach Auffassung des Autor wird in kommerziellen Softwaresystemen bspw. kaum darauf Rücksicht genommen, verschiedene *sequentielle, parallele und hybride sowie*

[3] Mit Software für akademische Zwecke ist Software gemeint, welche eher im Rahmen von theoretischen Untersuchungen eingesetzt wird.

ein- und mehrkriterielle Optimierungsverfahren zu implementieren. Dies würde es wiederum ermöglichen, eine Menge von Verfahren für die Optimierung von real-existierenden oder neuen, theoretischen ökonomischen Problemstellungen bereitzustellen. Zudem werden in kommerziellen Softwaresystemen oftmals nur spezielle Problemnischen abgedeckt, was zumeist am fehlenden Umfang verschiedener *externer Schnittstellen* liegt. Die Möglichkeiten zur *Kopplung anderer externer Berechner, Simulatoren und Optimierer* wird somit erschwert oder unmöglich. Bei den derzeitigen kommerziellen Softwaresystemen werden außerdem keine *internen Programm- oder Programmierschnittstellen* bereitgestellt, welche die Möglichkeit bieten, die *Weiterentwicklung des Softwaresystems* gezielt durch Dritte voranzutreiben oder ein bestehendes Softwaresystem zu verbessern.

Es erscheint daher sinnvoll, ein nicht kommerzielles Softwaresystem auf akademischem Niveau[4] zu schaffen, welches die Vorteile kommerzieller sowie nicht kommerzieller Softwaresysteme im Bereich der Modellierung, Simulation und Optimierung nutzt. Darüber hinaus erweist es sich auch als sinnvoll, mittels unkomplizierter visueller Hilfsmittel einen Teil der vorhandenen, akademisch geschaffenen Möglichkeiten anhand verschiedener Modellbeispiele aufzuzeigen. Für diese Zwecke wurde vom Autor das *Softwaresystem* `CAOS` (Abkürzung für *CalculationAssessmentOptimisationSystem* [engl.]) entwickelt. Es ist ein *Hilfsmittel zur rechnergestützten Modellierung, Simulation, Optimierung und Bewertung ökonomischer Problemstellungen*, welches moderne Methoden der modularen und objekt-orientierten Softwaretechnologie umsetzt sowie moderne Softwareentwicklungstechniken verfolgt (siehe [Bal05]). `CAOS` stellt zudem verschiedene externe und interne Schnittstellen bereit, welche einerseits die Einbindung eigener Softwareteile und andererseits auch eine gezielte Weiterentwicklung des bestehenden Softwaresystems im Rahmen der vorhandenen Entwicklungslizenz ermöglichen.

1.2. Ziele der Arbeit

Hauptziel der Arbeit ist die Schaffung eines akademischen Softwaresystems mit dessen Hilfe sich verschiedene theoretische Ansätze durch praktische Untersuchungen beweisen lassen. Das Softwaresystem soll dabei den heutigen Anforderungen an moderne Softwaretechnologien, wie bspw. *objektorientierte, modulare Programmierung* und *Mehr-Schichten-Architektur*, entsprechen, um die Mit- und Weiterentwicklung zu ermöglichen und zu vereinfachen. Mit Hilfe dieses Softwaresystems werden innerhalb dieser Arbeit bspw. *Möglichkeiten zur Verkürzung der Simulationsdauer eines Simulationsexperimentes* sowie der *Verkürzung der Optimierungsdauer durch verteilte und hybride Optimierung* dargelegt. Hierbei kommt ein neuartiger Optimierungsansatz zum Einsatz, der es erlaubt, für jede Optimierungsvariable einen individuellen Optimierer einzusetzen, um so ein geeignetes Optimierungsverfahren, angepasst auf die problemspezifischen Charakteristiken, zu verwenden.

[4] Gemeint ist ein Softwaresystem, in welchem gezielt auch neuere theoretische Erkenntnisse einfließen.

Im Hinblick auf die Modellierung erlaubt die Trennung von Modelldeklaration und -definition es dem Softwareentwickler in Zusammenarbeit mit dem Modellierer, eine Deklaration für ein Modell vorzugeben. Dieses dient einem Anwender des Softwaresystems als Unterstützung bei der Eingabe der exakten Modelldaten und der damit verbundenen Erstellung der Modelldefinition. Dieser gewählte, neuartige Ansatz ermöglicht es, die Verifizierung der Eingabedaten für ein Modell anhand seiner Deklaration bereits im Vorfeld einer Berechnung oder Simulation vorzunehmen. Dadurch werden, unter der Voraussetzung der korrekten Modellierung eines Systems, die Fehlermöglichkeiten auf wenige logische Fehler des Anwenders bei der Modelldateneingabe reduziert.

Bei der Modellierung der Elemente eines Systems wird eine spezielle Einteilung in verschiedene Gruppen von Modellelementen vorgenommen, um die Bedeutung einzelner Modellbestandteile für den Anwender besser hervorzuheben. Es soll verdeutlicht werden, wie sich eine solche Vorgehensweise auf das Verständnis eines Modells auswirken kann. Durch die Entwicklung einer neuen graphischen Oberfläche wird die Modelleingabe, -validierung und -auswertung vereinfacht, so dass der Anwender sich auf die eigentlich wesentlichen Ziele seiner Arbeit bei der Nutzung des Softwaresystems, der simulationsbasierten Optimierung, konzentrieren kann.

Praxisbezug erhält das Softwaresystem `CAOS` innerhalb dieser Darlegungen durch die Aufstellung theoretischer Modelle für Produktions- und Lagerhaltungssysteme, deren Implementierung in das Softwaresystem sowie deren Analyse in verschiedene Richtungen mittels dieses Systems vorgenommen wurden. Um eine Einordnung der neuartigen Modelle im Rahmen bisheriger, vorhandener Modelle vornehmen zu können und deren praktische Relevanz hervorzuheben, wird in dieser Arbeit ein Überblick über die vorhandenen theoretischen Modelle für Produktions- und Lagerhaltungssysteme sowie bisherige Untersuchungen gegeben.

Die Komplexität des untersuchten Modells zu Produktions- und Lagerhaltungssystemen entsteht hierbei u. a. durch die Vielfalt der Produkte und deren Variationen die von ein und demselben Produkt erzeugt werden können. Stochastische Einflussgrößen z. B. in Folge beliebiger Verteilungen für Kundenankünfte, Bedarfsmengen, Produktionszeiten, Umrüstzeiten etc., lassen eine analytische Lösung des zugrunde liegenden Problems nicht zu. Darüber hinaus erhöht sich die Komplexität des Modells oftmals noch durch die Betrachtung der verschiedensten vor- oder nachgelagerten innerbetrieblichen und möglicherweise auch außerbetrieblichen Teilproblemstellungen, welche direkt oder indirekt mit dem zu optimierenden Prozess verbunden sind. Aus der Kombination aller Teilproblemstellungen ergibt sich wiederum das Gesamtproblem, so dass dessen Charakteristik durch eine Modellzerlegung sowie der exakten Modellierung der Teilmodelle und deren Wechselwirkungen untereinander nicht verloren gehen darf.

1.3. Gliederung der Arbeit

Nachdem in den vorangehenden Abschnitten bereits vereinzelt verschiedene Inhalte der vorliegenden Arbeit angesprochen wurden, soll nachfolgend aufgezeigt werden, wie sich diese in der Gliederung der vorliegenden Arbeit widerspiegeln.

Im Kap. 2 werden Betrachtungen bezüglich einer möglichen Herangehensweise an die *Modellerstellung aus Sicht der simulationsbasierten Optimierung* durchgeführt. Dazu werden im Vorfeld die notwendigen Grundlagen für die Begriffe im Zusammenhang mit der simulationsbasierten Optimierung definiert. Des Weiteren werden die möglichen Vorgehensweisen und Lösungsverfahren im Rahmen der simulationsbasierten Optimierung vorgestellt. Dabei erfolgt stets eine gleichberechtigte Betrachtung von *ein- und mehrkriteriellen Optimierungsproblemen und -verfahren.* Anschließend wird zur *modellgestützten, simulationsbasierten Optimierung* übergegangen, bei der neben der eigentlichen Vorgehensweise der simulationsbasierten Optimierung auch eine geeignete *rechnerinterne Modellierung der Daten eines Modells zur simulationsbasierten Optimierung* im Vordergrund steht. Das Kapitel wird durch Aussagen zur *Bewertung und Entscheidungsfindung* im Rahmen einer möglichen Modellrealisierung abgerundet.

Als eine fundamentale Basis für die im Rahmen dieser Arbeit gewählte Art und Weise zur Umsetzung der simulationsbasierten Optimierung dient das Verständnis über die ereignisorientierte Simulation. Aus diesem Grund wird diese im Kap. 3 etwas näher betrachtet. Es wird dabei u. a. auf die relevanten Problematiken der Stationarität bei der Simulation und der Zeitfortschreibung bei der diskreten Simulation eingegangen. In diesem Zusammenhang ergibt sich ein neuartiger Ansatz, der es im Rahmen eines Laufes zur simulationsbasierten Optimierung ermöglicht, die Simulationsdauer zu verringern. Des Weiteren wird im Kap. 3 kurz auf das Thema der Generierung von Pseudozufallszahlen eingegangen, welche sowohl bei der Simulation als auch bei der Optimierung benötigt werden.

Den zum Verständnis der gewählten Herangehensweise innerhalb dieser Arbeit erforderlichen Aussagen zur Simulation schließt sich das Kap. 4 an. In diesem werden verschiedene, vorwiegend heuristische Optimierungsverfahren dargestellt und bewertet. Auch an dieser Stelle erfolgt wieder eine Unterscheidung von ein- und mehrkriteriellen Optimierungsproblemen und -verfahren, um einerseits eine weitestgehende Allgemeingültigkeit der getroffenen Aussagen zu erreichen und andererseits auch die Problematiken im Zusammenhang mit mehrkriteriellen Optimierungsproblemen abzudecken. Zudem existieren auch im Kontext der Optimierung einige Möglichkeiten, welche zu einer Verkürzung der Optimierungsdauer im Zusammenhang mit der simulationsbasierten Optimierung führen. Dies sind im Wesentlichen die hybride Optimierung sowie die verteilte und parallele Optimierung. Es wird ein Ansatz vorgestellt, der einen Einsatz unabhängig von den gewählten Optimierungsverfahren ermöglicht.

Die Darstellung eines möglichen Weges der Implementierung der in den Kapiteln 2 bis 4 getroffenen Aussagen sowie der dargestellten Strukturen und Algorithmen

mit Hilfe der objektorientierten Programmierung ist Bestandteil des Kapitels 5. Die Grundlage des entwickelten Softwaresystems CAOS zur rechnerinternen Modellierung, Berechnung, Simulation, Optimierung und Bewertung bildet eine einheitliche rechnerinterne Modelldarstellung des Simulationsmodells eines Systems (Realsystem oder theoretisches Modell eines System). Diese ist durch eine beliebige Modelldefinition gekennzeichnet, welche ihrerseits durch eine notwendige Modelldeklaration für einen Berechner oder Simulator speziell charakterisiert ist. Ein Vorteil des Softwaresystems CAOS ist zudem der gewählte modulare Ansatz, welcher es stets problemlos möglich macht, weitere Berechner, Simulatoren und Optimierer in das Softwaresystem einzubinden. Dazu werden im Kap. 5 die einzelnen Bibliotheken und deren Inhalte vorgestellt.

Im Kap. 6 werden grundlegende Problemstellungen von vorhandenen Basismodellen zu Produktions- und Lagerhaltungssystemen sowie deren wesentliche Charakteristiken betrachtet. Auf diesen aufbauend wird gezeigt, welche Erweiterungen möglich sind und bereits untersucht wurden. Diese Aussagen dienen als Ausgangspunkt für die neuen Untersuchungsmodelle, welche im Rahmen dieser Arbeit vorgestellt werden und auf die Gewinnung neuer Erkenntnisse im Bereich der Produktions- und Lagerhaltungssysteme abzielen. Eines der Untersuchungsmodelle ist ein *Ein-Maschinen-Mehr-Produkt-Modell*, welches zu einem *Mehr-Maschinen-Mehr-Produkt-Modell* und dieses zu einem einfachen *Supply-Chain-Mehr-Produkt-Modell* erweitert wird. Die Modelle besitzen verschiedene Modellcharakteristiken und Restriktionen, welche sie analytisch nur schwer oder nicht lösbar machen. Aus diesem Grunde wird zur Problemlösung auf die modellgestützte, simulationsbasierte Optimierung zurückgegriffen und somit ein sinnvolles Einsatzgebiet des Softwaresystems CAOS aufgezeigt. Für die Untersuchung der verschiedenen Modelle zu Lagerproduktionssystemen soll anhand verschiedener Modellbeispiele im Abschn. 6.4 und deren Ergebnisse im Abschn. 6.5 gezeigt werden, welche Aussagen über die untersuchten Modelle getroffen und welche Rückschlüsse auf die Güte der untersuchten Modelle gezogen werden können. Es wird verdeutlicht, dass verschiedene Produktionsstrategien zu unterschiedlichen Ergebnissen führen. Dabei hat die Wahl einer Produktionsstrategie mitunter nur bedingt Einfluss auf das Treffen allgemein gültiger Aussagen über das Gesamtmodell.

Im abschließenden Kap. 7 ist aufgezeigt, welche Untersuchungsrichtungen sich an diese Arbeit anschließen könnten. Ziel weiterer Untersuchungen wäre es bspw., weitere Verbesserungen innerhalb des Fertigungsprozesses bei den Modellen zu Produktions- und Lagerhaltungssystemen im Bezug auf die Optimierung von Losgrößen und Produktionsreihenfolgen sowie anderer Entscheidungsvariablen zu erzielen. Ein wesentlicher Bestandteil des zukünftigen Entwicklungsprozesses ist es zudem, dass der Zyklus der Entwicklung einer Software, innerhalb dieser Arbeit im Speziellen akademischer Software, oftmals nicht als abgeschlossen betrachtet werden kann. Es ergeben sich stets weitere Anforderungen und Untersuchungsrichtungen durch neue wissenschaftliche Erkenntnisse. Aus diesem Grund ist das entwickelte Softwaresystem CAOS soweit offen und erweiterbar gehalten, dass neue Verfahren und Methoden aus dem Bereich der Simulation und Optimierung problemlos eingebunden werden können.

Kapitel 2. Modellgestützte, simulationsbasierte Optimierung

Inhalt

2.1	*Einführung*	*10*
2.2	*Charakteristiken und Definitionen*	*12*
2.2.1	*System und Modell*	*12*
2.2.2	*Simulationsmodell und Simulation*	*14*
2.2.3	*Optimierungsproblem*	*17*
2.2.4	*Nebenbedingungen*	*19*
2.2.5	*Zielfunktionsraum*	*23*
2.2.6	*Diskretisierung des Raumes der Entscheidungsvariablen und Permutationsprobleme*	*27*
2.2.7	*Definition zur simulationsbasierten Optimierung*	*34*
2.3	*Lösungsverfahren zur simulationsbasierten Optimierung*	*34*
2.4	*Rechnerinterne Modellierungsvariante*	*39*
2.4.1	*Schritte bei der rechnerinternen Modellierung*	*42*
2.4.2	*Verwendung einer Modelldefinition*	*44*
2.4.3	*Sichtweisen auf ein rechnerinternes Modell*	*44*
2.4.4	*Möglichkeit der Weiterverwendung eines Simulators*	*45*
2.4.5	*Rechnerinterne Abbildung der Entscheidungsvariablen*	*46*
2.5	*Bewertung*	*46*
2.6	*Entscheidungsfindung*	*50*
2.7	*Zusammenfassung*	*52*

2.1. Einführung

Die simulationsbasierte Optimierung stellt ein Lösungsverfahren dar, welches die (sub)optimale Wertebelegung für die Entscheidungsvariablen eines vorliegenden Optimierungsproblems ausgibt. Dabei sind verschiedene Vorgehensweisen denkbar, von denen sich im Wesentlichen die iterative und die sequentielle Vorgehensweise als sinnvoll für die durchgeführten Untersuchungen ergaben. Die sequentielle Vorgehensweise wird nur erwähnt und die iterative Vorgehensweise für die weiteren Ausführungen verwendet.

Bei der iterativen Vorgehensweise der simulationsbasierten Optimierung schlagen ein oder mehrere, als geeignet ausgewählte Optimierungsverfahren einem Simulator oder ggf. einem analytischen Berechner konkrete Werte für die Entscheidungsvariablen vor. Der Simulator oder der Berechner führen anhand der konkreten Werte eine Simulation oder Berechnung durch und liefern die zugehörigen Ergebnisse zurück, welche dem Optimierungsverfahren als Bewertungsgrundlage dienen (siehe Abb. 2.1). Dieser Zyklus wiederholt sich solange, bis das Optimierungsverfahren terminiert oder ein bestimmtes, definiertes Abbruchkriterium erfüllt ist.

Abb. 2.1

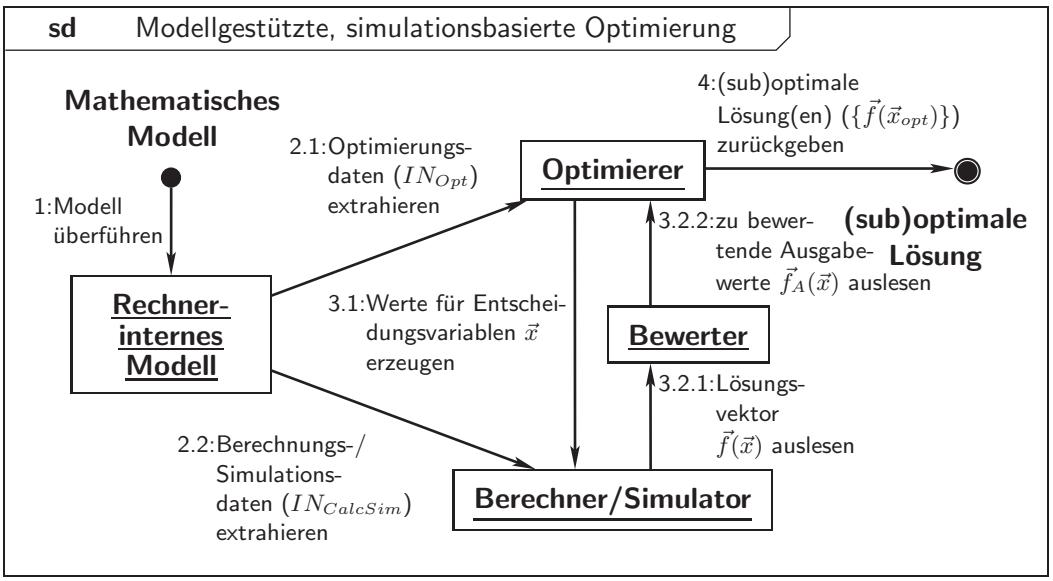


Abb. 2.1.: *Prinzip der angewandten Methode zur modellgestützten, simulationsbasierten Optimierung (Grobgerüst)*

Wie aus Abb. 2.1 erkennbar ist, lässt sich die modellgestützte, simulationsbasierte Optimierung im Wesentlichen in folgende Schritte des EVA-Prinzips[5] gliedern, welche zunächst kurz erwähnt und dann in den betreffenden Abschnitten dieses Kapitels und der folgenden beiden Kapitel näher erläutert werden.

[5]Das EVA-Prinzip (*Eingabe - Verarbeitung - Ausgabe*) gilt als Grundschema der Elektronischen DatenVerarbeitung (kurz: EDV). Es bezieht sich sowohl auf die Organisation der Hardware als auch auf das EDV-System (Hard- und Software) als Ganzes (siehe [PMK97]).

Mathematische Modellierung (Eingabe).
Um Untersuchungen an komplexen Realsystemen mittels der rechnergestützten Simulation, kurz *Rechnersimulation* genannt, durchführen zu können, ist es notwendig, die zur Untersuchung des Realsystems wesentlichen Teile in eine geeignete *logische und mathematische Darstellung* mit entsprechenden *Gleichungen, Ungleichungen und Zielfunktionen* zu überführen (Abbildung eines Realsystems mittels Navier-Stokes-Gleichungen). Dieser Schritt wird als mathematische Modellierung oder oftmals nur als Modellierung bezeichnet.

Rechnerinterne Modellierung (Eingabe).
Durch die rechnerinterne Modellierung entsteht aus dem mathematischen Modell unter Nutzung geeigneter Datenstrukturen das *rechnerinterne Modell eines Realsystems*, aus welchem sich wiederum ein Simulationsmodell durch Hinzunahme geeigneter Algorithmen zur Nachbildung des Systemverhaltens erzeugen lässt. In diesem rechnerinternen Modell werden die einzelnen *Elemente des Modells* [dt.; entities of the model [engl.]] mit ihren individuellen *Attributen* dargestellt. Durch den zusätzlichen Schritt der rechnerinternen Modellierung wird die im Rahmen dieser Arbeit angewandte Form der simulationsbasierten Optimierung auch als modellgestützte, simulationsbasierte Optimierung bezeichnet[6].

Simulation (Verarbeitung).
Unter der rechnergestützten Simulation soll in diesem Fall nicht die Echtzeit-Simulation[7] zu verstehen sein, sondern die *Simulation diskreter Systeme*. Dabei werden bei der ereignisgesteuerten Simulation mit Hilfe von *Ereignissen* verschiedene Aktivitäten, Abläufe oder Prozesse aus einem bestimmten Teilkomplex, z. B. des betreffenden Realsystems, in einem Rechner zur Erlangung der Vorstellung über das momentane oder zukünftige Systemverhalten nachgebildet. Die wesentlichen Ereignisse können unter Ausnutzung des Zeitrafferoder Zeitlupeneffekts anhand der zugehörigen *Ereignisroutinen* ausgeführt werden.

Berechnung/Analyse (Verarbeitung).
Liegt eine analytische Berechnungsvorschrift für ein Modell vor, kann auf eine, zumeist sehr rechenzeitintensive, Simulation verzichtet werden, um die betreffenden Ergebnisse zu erhalten. Eine analytische Betrachtungsweise kann auch bei Diskretisierung verschiedener stochastischer Simulationsmodellmerkmale zur Validierung eines Simulationssystems anhand der exakten Lösung herangezogen werden.

Bewertung (Verarbeitung).
Erst die Bewertung ermöglicht es, eine Aussage über die bei der Simulation oder Berechnung ermittelten Ergebnisse treffen zu können. Dabei erfolgt eine Transformation der ermittelten Ergebnisse in eine für den Entscheidungsfinder (Optimierer) verwertbare, aussagekräftige Form.

[6] Ist nachfolgend von der simulationsbasierten Optimierung die Rede, ist daher gleichzeitig die modellgestützte, simulationsbasierte Optimierung gemeint.
[7] Für die verschiedenen Arten der Simulation siehe bspw. [Ban98].

Optimierung (Verarbeitung).
Um verschiedene Szenarien für eine Simulation bereitzustellen, liefern Optimierungsverfahren basierend auf den Entscheidungsvariablen des Optimierungsproblems die notwendigen *Werte für diese Entscheidungsvariablen*. Der *Vergleich der bewerteten Ergebnisse* der Simulation für die unterschiedlichen Szenarien gibt Aufschluss über den Grad des Erfolges des Vorgehens (vgl. [Lie92]). Zudem ist der Optimierer dafür verantwortlich, dass die vorgegebenen Nebenbedingungen stets eingehalten werden.

Lösung (Ausgabe).
Das beste oder die besten gefundenen Szenarien (konkrete Werte der Entscheidungsvariablen) werden im Anschluss an den Prozess der simulationsbasierten Optimierung zu einer möglichen Weiterverarbeitung ausgegeben.

Die Ergebnisse der simulationsbasierten Optimierung können zumeist unmittelbar oder durch entsprechende, geeignete Umformung der Ergebnisse im Rahmen der *Entscheidungsfindung* auf das untersuchte Realsystem übertragen werden. Der Prozess der Entscheidungsfindung ist somit der simulationsbasierten Optimierung nachgeschaltet. Die ermittelten Ergebnisse dienen hauptsächlich der

- *Validierung* der Kenngrößen eines bestehenden Realsystems,
- *Vorhersage* des Systemverhaltens eines zukünftigen Realsystems oder
- *Erklärung* der genauen Vorgänge innerhalb eines bestehenden oder zukünftigen Realsystems (siehe [Sch85b]).

Zudem kann mit den gewonnenen Erkenntnissen bspw. die geplante technische Realisierung eines modellierten Systems oder die Verbesserung eines existierenden Realsystems erfolgen.

Zum Verständnis der Vorgehensweisen und des Zusammenspiels der Simulation und Optimierung beim Verfahren der simulationsbasierten Optimierung sind in den nachfolgenden Abschnitten zunächst grundlegende Definitionen angegeben. Diese sollen gezielt zur Definition der simulationsbasierten Optimierung im Abschn. 2.2.7 hinführen.

2.2. Charakteristiken und Definitionen

2.2.1. System und Modell

Ein System bezeichne innerhalb dieser Arbeit ein real-existierendes ökonomisches oder technisches System. Ein Modell ist ein Abbild eines solchen Systems durch eine formal geeignete mathematische Darstellung, z. B. mittels Gleichungen und Ungleichungen, bzw. durch eine rechnerinterne Darstellungsform.

Die meisten Begriffe für Systeme treten auch bei den Modellen in ähnlicher Form auf, wodurch eine exakte Trennung der damit zusammenhängenden Begriffe und

deren Bedeutungen mitunter nur schwer möglich ist. Der Begriff Element tritt bspw. sowohl im Zusammenhang mit dem Begriff System als auch mit dem Begriff Modell auf, hat aber ein unterschiedliches Abstraktionsniveau bzw. eine leicht unterschiedliche Bedeutung. Durch Verwendung der Begriffe Systemelement und Modellelement wird jeweils der exakte Bezug hergestellt. Dies ist in den nachfolgenden Ausführungen stets berücksichtigt.

Ein System oder ein Modell setzt sich aus einer *Menge von zueinander in Beziehung stehenden Elementen* zusammen. Elemente müssen dabei nicht unmittelbar auftreten. Sie können auch in Untersystemen des eigentlichen Systems zusammengefasst sein. Die Verbindungen zwischen den Elementen verdeutlichen deren Beziehungen zueinander. Elemente treten dabei als temporäre Elemente, bspw. Kunden und Produktteile, oder als permanente Elemente, bspw. Fertigungseinheiten und Lager, auf.

Durch den Schritt der *Modellbildung* bzw. die *mathematische Modellierung* eines Systems entsteht zunächst ein *mathematisches Modell*. Dieses beinhaltet die Abbildung eines Systems durch eine formal geeignete mathematische Darstellung des Systems mit Hilfe eines geschlossenen Satzes von Formeln, Gleichungen und Ungleichungen. Ziel der mathematischen Modellierung ist zudem meist auch die Reduktion der Komplexität des modellierten Systems innerhalb des entstandenen Modells, indem nur die für die Untersuchung des Realsystems wesentlichen Bestandteile modelliert werden. Es kann festgehalten werden, dass nur so viel modelliert werden sollte, wie nötig ist, um das gewünschte Systemverhalten sinngemäß zu beschreiben. Vorzugsweise soll das Modell eines Realsystems dieses auch für Nichtsystemexperten verständlicher machen. Hinzu kommt, dass die Untersuchung eines gesamten Realsystems mit Hilfe eines entsprechenden Modells zwar theoretisch möglich ist, aber i. Allg. wenig sinnvoll, weil die direkt aus der Modellierung des Gesamtsystems resultierende Komplexität des mathematischen Modells und des enthaltenen Optimierungsproblems meist zu hoch ist. Dadurch wird das Auffinden einer optimalen, einer suboptimalen oder in manchen Fällen sogar nur einer gültigen Lösung des Optimierungsproblems im Rahmen der simulationsbasierten Optimierung in einer vorgegebenen Zeitspanne erschwert. Ist die Komplexität des entstandenen mathematischen Modells dennoch so hoch, dass es nur mit numerischen Methoden ausgewertet werden kann, benötigt man zu dessen Lösung i. Allg. ein so genanntes *Computer-* oder *Rechnermodell*. Ein *Simulationsmodell* ist ein solches, spezielles Rechnermodell (siehe Abschn. 2.2.2 und Kap. 3), dessen Komplexität zudem durch die Existenz stochastischer Einflüsse entsteht. Die Auswertung eines Rechnermodells erfolgt mittels eines Rechners. Sie wird daher auch als *Computer-* bzw. *Rechnersimulation* bezeichnet.

Bei der Modellierung eines Systems muss darauf geachtet werden, dass auch die Systemparameter in geeigneter Form durch entsprechende Modellparameter wiedergegeben werden. Ein *Systemparameter* bezeichne dabei eine *charakterisierende veränderliche Kenngröße des untersuchten Systems*. Ein *Modellparameter* bei der simulationsbasierten Optimierung stelle hingegen eine *charakterisierende veränderliche Kenngröße des* durch Modellierung eines Systems *entstandenen Modells* dar. Er kann innerhalb eines Simulationsmodells (siehe Abschn. 2.2.2) bspw. durch ein

konstantes Merkmal eines Simulationsmodellelementes repräsentiert werden. Im Sinne des, in der simulationsbasierten Optimierung enthaltenen, Optimierungsproblems kann diese charakterisierende Kenngröße eines Modells aber auch eine Steuergröße bzw. einen Einflussfaktor des, infolge der Modellierung entstandenen, Optimierungsproblems darstellen. Diese Art von Modellparametern werden nachfolgend als *Entscheidungsvariable* bezeichnet (siehe auch Abschn. 2.2.3). Die anderen modellierten Systemparameter und -konstanten werden bei der simulationsbasierten Optimierung innerhalb des zugehörigen Simulationsmodells als sog. *Modellvariablen* abgebildet (vgl. Abschn. 2.2.2). Sie stellen dann beliebige, aber feste Werte innerhalb eines Laufes der simulationsbasierten Optimierung[8] dar. Sie beinhalten u. a. auch die Modellausgabewerte, welche während der Laufzeit des Modells bestimmt und entsprechend nach außen gegeben werden.

2.2.2. Simulationsmodell und Simulation

Mit Hilfe eines *Simulationsmodells* wird es möglich, die Rechnersimulation, nachfolgend auch rechnergestützte Simulation genannt, durchzuführen. Dazu wird das modellierte Systemverhalten mittels entsprechender *Algorithmen* und das mathematische Modell mit Hilfe geeigneter *Datenstrukturen* rechnerintern nachgebildet. Ein Simulationsmodell ist dabei ein spezielles Rechnermodell, dessen Gegenstand, Inhalt und Darstellung direkt für Zwecke der Simulation konstruiert wird. Daher wird es auch *konstruktionsprozessorientierte Modelldefinition* genannt. Wie bereits erwähnt, werden dabei nur diejenigen Merkmale eines Systems im Simulationsmodell als Elemente und Merkmale[9] berücksichtigt, welche auch für eine konkrete Problemstellung von Bedeutung sind. Andere Merkmale hingegen, welche von nicht oder nur von minderer Bedeutung für die Problemlösung sind, können i. Allg. vernachlässigt werden. Es können demnach je nach Problemstellung bzw. Sichtweise verschiedene Modelle eines Systems gleichberechtigt nebeneinander existieren (siehe Abb. 1.1).

Simulationsmodellarten. Je nach Verwendungszweck existieren verschiedene Arten von Simulationsmodellen.

Beschreibendes Simulationsmodell.

Beschreibende Simulationsmodelle, auch deskriptive Simulationsmodelle genannt, dienen der *Untersuchung des Zusammenwirkens einzelner Teilsyste-*

[8] Ein Lauf der simulationsbasierten Optimierung bezeichne den Zeitraum der ersten Lösung eines Optimierungsproblems bis zum Zeitpunkt der Erfüllung eines festgelegten Abbruchkriteriums, während dem nur die Werte der Entscheidungsvariablen verändert werden und alle anderen Simulationsmodellparameter konstant bleiben.

[9] Im Sinne der Simulation werden die über einen bestimmten Zeitraum, die Dauer eines Simulationsexperimentes, konstanten Merkmale als *Simulationsmodellparameter* und die veränderlichen Merkmale als *Simulationsmodellattribute* bezeichnet. Beispiele hierfür sind u. a. in [Fis01] angegeben. Bei der simulationsbasierten Optimierung hingegen werden die Simulationsmodellattribute im Vorfeld festgelegt und dann über den Zeitraum eines Optimierungslaufes als konstant angesehen. Die veränderlichen Größen werden durch die Entscheidungsvariablen des zugrunde liegenden Optimierungsproblems gebildet.

me, deren Verhalten bekannt ist. Diese Simulationsmodelle werden bspw. zu Prognosezwecken eingesetzt. Sie können jedoch auch zur Beschreibung oder Erklärung eines komplexen Systems dienen.

Berechnendes Simulationsmodell.
In berechnenden bzw. pragmatisch-normativen Simulationsmodellen wird die *Simulation als Werkzeug der Planung zur Entscheidungsunterstützung* verwendet. Solche Modelle finden bei der simulationsbasierten Optimierung Verwendung, weil es bei diesen insbesondere um Bestimmung einer zufrieden stellenden Fixierung für die Werte der Entscheidungsvariablen geht.

Innerhalb dieser Arbeit werden berechnende Simulationsmodelle angewandt.

Allgemein dient die Simulation der Durchführung von Beobachtungen des Modellverhaltens über eine gegebene Zeit. Sie trägt damit im Sinne der simulationsbasierten Optimierung zur Bestimmung der (sub)optimalen Lösung des zugrunde liegenden Optimierungsproblems anhand konkreter Werte für die Entscheidungsvariablen bei. Die Entscheidungsvariablen werden wie bereits erwähnt als Simulationsmodellparameter abgebildet. Diese können zwar im Laufe des Optimierungslaufes der simulationsbasierten Optimierung unterschiedliche Werte annehmen, bleiben aber im Gegensatz zur dynamischen Optimierung während der Simulationsläufe eines Simulationsexperimentes unverändert. Ein Simulationslauf bezeichne dabei die Durchführung einer einzelnen Simulation anhand konkreter Werte für sämtliche Simulationsmodellparameter. Ein Simulationsexperiment beinhaltet i. d. R. die *wiederholte Durchführung eines Simulationslaufes*, welcher i. Allg. unter Anwendung verschiedener Startwerte für die verwendeten Zufallszahlengeneratoren abläuft. Zufallszahlengeneratoren werden daher im Abschn. 3.7 zur Simulation und im Abschn. 5.3.4 zur Realisierung bestimmter Zufallszahlengeneratoren innerhalb des Softwaresystems `CAOS` etwas näher betrachtet.

Ausgabewerte von Simulationsmodellen. Wie zuvor bereits erwähnt, bilden die Ausgabewerte von Simulationsmodellen bzw. -experimenten einen Teil der Modellvariablen. Sie werden innerhalb dieser Arbeit in

- $\vec{f_A}(\vec{x})$ - für die Gewichtung vorgesehene Ausgabewerte (rechnerintern modelliert in der Modellsektion OUT_{Ass}[10], siehe Abschn. 2.4) sowie
- $\vec{f_N}(\vec{x})$ - für verschiedene Statistiken vorgesehene Ausgabewerte (rechnerintern modelliert in der Modellsektion OUT_{NoAss}[11], siehe Abschn. 2.4)

unterteilt. Somit wird nur der Teil der Ausgabewerte mit den für die Gewichtung vorgesehenen Ausgabewerten zur Lösung des Optimierungsproblems (siehe Abschn. 2.2.3) verwendet. Ohne Gewichtung der Zielfunktionswerte gelte:

$$\vec{f}(\vec{x}) = \vec{f_A}(\vec{x}) \quad .$$

[11] Die Bezeichnung *Ass* bezieht sich auf das englische Wort *assessment* (dt. Bewertung).
[11] Die Bezeichnung *NoAss* bezieht sich auf das englische Wort *no assessment* (dt. keine Bewertung).

Zudem gelte formal Folgendes

$$\vec{f_A}(\vec{x}) = (f_{A_1}(\vec{x}), f_{A_2}(\vec{x}), \ldots, f_{A_d}(\vec{x}))^T \quad \text{und} \tag{2.1}$$
$$\vec{f_N}(\vec{x}) = (f_{N_1}(\vec{x}), f_{N_2}(\vec{x}), \ldots, f_{N_e}(\vec{x}))^T \quad , \tag{2.2}$$

wobei

$d = \dim(\vec{f_A}(\vec{x}))$ - Dimension bzw. Anzahl der zu bewertenden Ausgabevariablen und
$e = \dim(\vec{f_N}(\vec{x}))$ - Dimension bzw. Anzahl der nicht zu bewertenden Ausgabevariablen ist.

Bewertung und Gewichtung. Wie bereits zuvor angedeutet, erfolgt eine Gewichtung der zu bewertenden Ausgabewerte eines Simulationsmodells, bevor diese, für die Optimierung relevanten Größen, an das verwendete Optimierungsverfahren als Funktionswerte der Form $\vec{f}(\vec{x})$ übergeben werden. Es gelte

$$\vec{f}(\vec{x}) = \vec{w} \circ \vec{f_A}(\vec{x})$$
$$\dim(\vec{w}) = \dim(\vec{f_A}(\vec{x})) \quad ,$$

wobei

$$\vec{x} \circ \vec{y} = (x_1 \cdot y_1, x_2 \cdot y_2, \ldots, x_n \cdot y_n)^T, \quad n = \dim(\vec{x}) = \dim(\vec{y})$$

und

$\vec{w}$ - Vektor mit den Gewichten zur Gewichtung der zu bewertenden Ausgabewerte ist.

Im Gegensatz zu dem genannten Ansatz, sind auch andere Formen der Bewertung für die Ausgabewerte eines Simulationsmodells denkbar. Bspw. könnten verschiedene Ausgabewerte vor oder nach der Bewertung zu einem Zielfunktionswert $f(\vec{x})$ zusammengefasst werden. Weitere Aussagen zur Bewertung hinsichtlich verschiedener Typen von Optimierungsverfahren und -problemen u. a. auch im Hinblick auf die Verknüpfung von ein- und mehrkriteriellen Optimierungsproblemen sind im Abschn. 2.5 nachzulesen.

Der gewählte Ansatz zur „multiplikativen“ Gewichtung und die damit verbundene Überführung der zu bewertenden Ausgabewerte (ungewichteter Zielfunktionsraum) in den gewichteten Zielfunktionsraum ist in Abb. 2.2 noch einmal graphisch dargestellt.

Modellvalidierung eines Simulationsmodells. Ein wesentlicher Schritt bei der Modellierung ist u. a. die Validierung des Simulationsmodells. Sie beinhaltet den Nachweis, ob das zu erzielende Systemverhalten und das genutzte Simulationsmodell

Abb. 2.2

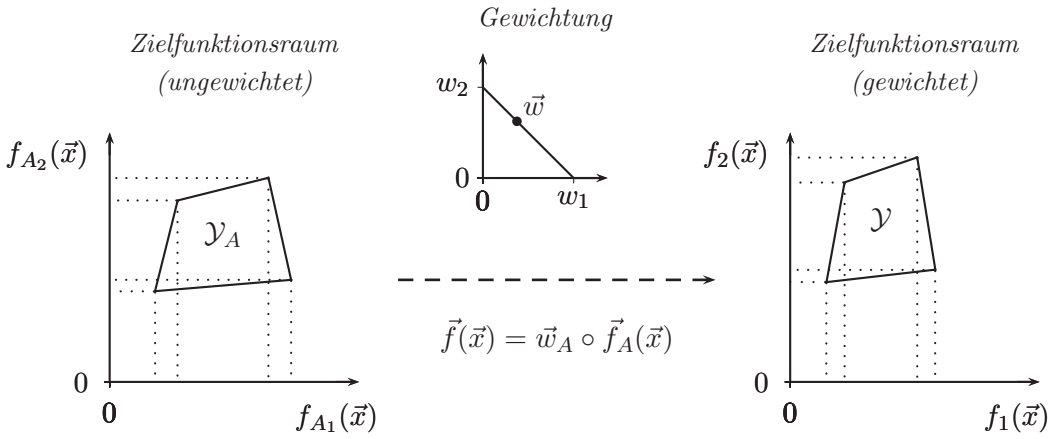


Abb. 2.2.: *Überführung der Ausgabewerte in die Zielfunktionswerte anhand einer Gewichtung*

innerhalb vorgegebener Toleranzen übereinstimmen. Diese Überprüfung kann nur an Hand einzelner Simulationen mit ausgewählten Simulationsdaten erfolgen, die i. Allg. durch Messungen am Realsystem gewonnen und anschließend in das mathematische Modell überführt werden. Daraufhin kann mittels der Durchführung eines Simulationsexperimentes das mathematische Modell mit dem Simulationsmodell verglichen werden. Stimmen diese Ergebnisse überein wird geschlussfolgert, dass das Simulationsmodell auch für nicht überprüfte Systemparameter richtige Ergebnisse liefert. Die Validierung eines Simulationsmodells kann also nur für ein bereits existierendes System erfolgen.

Existiert das System noch nicht, ist es mitunter möglich, das Simulationsmodell mit Hilfe eines analytischen Modells zu überprüfen, wenn ein solches analytisches Modell konstruiert werden kann. Die analytische Lösung wird hierbei mit den Ergebnissen der Simulation verglichen. Es muss jedoch beachtet werden, dass mit dieser Methode, der sog. Rektifikation, lediglich der korrekte Modellaufbau überprüft wird, aber nicht, ob die anfängliche Systemanalyse richtig vorgenommen wurde.

2.2.3. Optimierungsproblem

Einkriteriell kontra mehrkriteriell. Zunächst sei vorausgesetzt, dass die konkreten Werte der Entscheidungsvariablen eine *Lösung* repräsentieren. Damit sind indirekt auch der Wert der Zielfunktion $f(\vec{x})$ bzw. die Werte der Zielfunktionen $\vec{f}(\vec{x})$ für eine Lösung gegeben. Demzufolge wird in der Mathematik ein Problem, das nach der *bestmöglichen Lösung* aus einer Menge von potentiellen Lösungskandidaten fragt, als *Optimierungsproblem* bezeichnet. Bei dieser Bestimmung der bestmöglichen Lösung spielt u. a. auch die Anzahl der Zielfunktionen eine entscheidende Rolle. Es wird i. Allg. zwischen

- *einkriteriellen Optimierungsproblemen* [dt.; *Singleobjective Optimisation Problem* [engl., kurz: SOP]], welche *eine Zielfunktion* besitzen, und

- *mehr-* bzw. *multikriteriellen Optimierungsproblemen* [dt.; Multiobjective Optimisation Problem [engl., kurz: MOP]], welche *mehrere Zielfunktionen* aufweisen,

unterschieden.

Ein- und mehrkriterielle Optimierungsprobleme seien wie folgt definiert.

Def. 2.1 ***Definition 2.1 (Mehrkriterielles Optimierungsproblem):***
Ein gültiges MOP beinhaltet eine Menge von n Entscheidungsvariablen deren Werte in $\vec{x}$ (Lösungen) repräsentiert sind, eine Menge von m Zielfunktionen mit den Werten in $\vec{f}(\vec{x})$ und Mengen von Nebenbedingungen $\{g(\vec{x})\}$ und $\{h(\vec{x})\}$ mit dem Optimierungsziel

$$f_j(\vec{x}) \rightarrow \max / \min \quad j = 1, 2, \ldots, m \quad , \tag{2.3}$$

wobei

$$\begin{aligned} g_k(\vec{x}) &\leq 0 && k = 1, 2, \ldots, p \\ h_l(\vec{x}) &= 0 && l = 1, 2, \ldots, q \\ x_i^l \leq x_i &\leq x_i^u && i = 1, 2, \ldots, n \\ x_i^l,\, x_i,\, x_i^u &\in \mathbb{R} \\ -\infty \leq x_i^l,\, x_i^u &\leq \infty && i = 1, 2, \ldots, n \quad . \end{aligned}$$

□

Dabei stellen x_i^l die untere Schranke für den Wert der Entscheidungsvariablen x_i und x_i^u die obere Schranke für den Wert der Entscheidungsvariablen x_i dar. Im Mathematischen werden die untere Schranke x_i^l und die obere Schranke x_i^u als Randbedingungen betrachtet, welche einen n-dimensionalen beschränkten Hyperraum beschreiben. Dieser Hyperraum sei mit $\mathcal{X} \subset \mathbb{R}^n$ bezeichnet, wobei $n = \dim(\vec{x})$ gelte.

Def. 2.2 ***Definition 2.2 (Einkriterielles Optimierungsproblem):***
Für das allgemeingültige SOP gelten die gleichen Bedingungen wie für ein MOP nach Def. 2.1, wobei der Raum der Zielfunktionen eindimensional sei, d. h. es gelte $m = 1$. □

In Abb. 2.3 ist an einem ausgewähltem Beispiel angegeben, wie sich der Raum der Entscheidungsvariablen und der Raum der Zielfunktionen gestalten könnten.

Klassifizierung. In [Had01] wird neben der Anzahl der Zielfunktionen eine weiterreichende *Klassifizierung für Optimierungsprobleme* nach Eigenschaften wie:

- Struktur der *Ziel- und Nebenbedingungsfunktionen*

Abb. 2.3

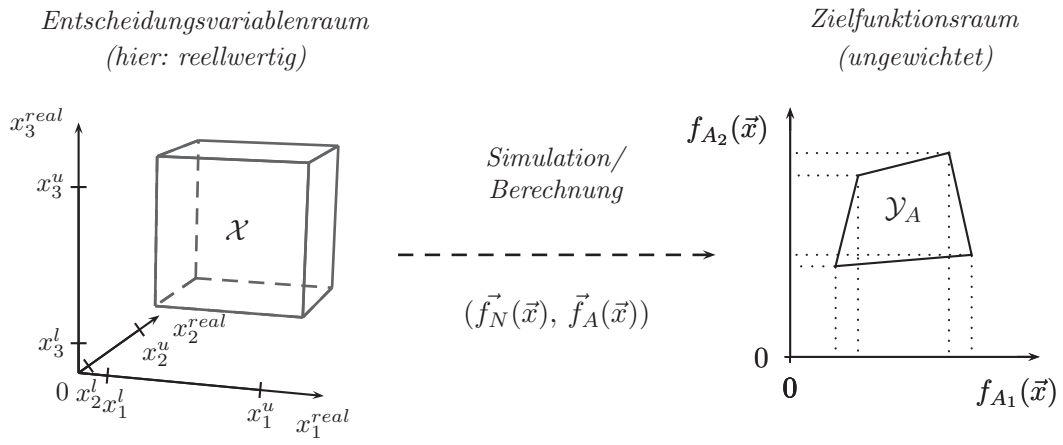


Abb. 2.3.: *Raum der Entscheidungsvariablen und der Zielfunktionen vor Gewichtung der Ausgabewerte*

- Struktur des *Problemparameterraumes*
- Struktur des *Zielfunktionsraumes*

durchgeführt. Dies zielt direkt auf die Darstellung des Optimierungsproblems mittels der Notation nach Großmann und Terno (siehe [GT93]) ab. Nach Hader stellen die meisten der durch Simulation (siehe Abschn. 2.2.2) zu lösenden Optimierungsprobleme ein *analytisch formuliertes, nichtlineares Optimierungsproblem* dar. Die Konvexität von deren Zielfunktion(en) kann nur unter bestimmten Voraussetzungen nachgewiesen werden, so dass die Zielfunktion(en) i. Allg. *nichtkonvex* sind (siehe [GBE+86]).

Definition 2.3 (Konvexe Funktion):
Eine Funktion f von einem Intervall I_f nach $\mathbb{R}$, d. h. $f : I_f \to \mathbb{R}$ heißt konvex, wenn für alle x_1, x_2 aus I_f und t zwischen 0 und 1 gilt:

Def. 2.3

$$f(t \cdot x_1 + (1-t) \cdot x_2) \quad \leq \quad t \cdot f(x_1) + (1-t) \cdot f(x_2) \quad .$$

Diese Beziehung sei durch $\mathrm{conv}(f,\ I_f) \subset \mathbb{R}$ *dargestellt.* □

2.2.4. Nebenbedingungen

Gleichheits- und Ungleichheitsbedingungen. Nebenbedingungen waren in Def. 2.1 zunächst noch allgemeingültig dargestellt. Allerdings soll innerhalb dieser Arbeit auch eine Abhängigkeit der Nebenbedingungen von den bewerteten und nicht bewerteten Zielfunktionswerten ermöglicht werden. Daher werden die Nebenbedingungen aus $\{g(\vec{x}, \vec{f_A}(\vec{x}), \vec{f_N}(\vec{x}))\}$ als *Ungleichheitsbedingungen* und $\{h(\vec{x}, \vec{f_A}(\vec{x}), \vec{f_N}(\vec{x}))\}$ als *Gleichheitsbedingungen* bezeichnet. Somit können die Nebenbedingungen sowohl direkt als auch indirekt von den Werten der Entschei-

dungsvariablen und den Zielfunktionswerten abhängig seien. Sie legen den *zulässigen Bereich* fest, welcher wie folgt definiert ist.

Def. 2.4 ***Definition 2.4 (Zulässiger Bereich eines Optimierungsproblems):***
Der zulässige Bereich $\mathcal{X}_{Zul}$ eines Optimierungsproblems sei abhängig von den Werten der Entscheidungsvariablen $\vec{x}$ sowie den Ausgabewerten des Simulationsmodells $f_A(\vec{x})$ sowie $\vec{f_N}(\vec{x})$ und durch folgende Nebenbedingungen allgemeingültig festgelegt:

$$\begin{aligned} \{g(\vec{x}, \vec{f_A}(\vec{x}), \vec{f_N}(\vec{x}))\} &= (g_1(\vec{x}, \vec{f_A}(\vec{x}), \vec{f_N}(\vec{x})), g_2(\vec{x}, \vec{f_A}(\vec{x}), \vec{f_N}(\vec{x})), \\ &\quad \ldots, g_p(\vec{x}, \vec{f_A}(\vec{x}), \vec{f_N}(\vec{x})))^T \leq \vec{0} \end{aligned} \tag{2.4}$$

$$\begin{aligned} \{h(\vec{x}, \vec{f_A}(\vec{x}), \vec{f_N}(\vec{x}))\} &= (h_1(\vec{x}, \vec{f_A}(\vec{x}), \vec{f_N}(\vec{x})), h_2(\vec{x}, \vec{f_A}(\vec{x}), \vec{f_N}(\vec{x})), \\ &\quad \ldots, h_q(\vec{x}, \vec{f_A}(\vec{x}), \vec{f_N}(\vec{x})))^T = \vec{0} \end{aligned} \tag{2.5}$$

wobei

$$\vec{0} = (0, 0, \ldots, 0)^T \quad .$$

□

Pre- und Post-Nebenbedingungen. In der vorliegenden Arbeit wird vorausgesetzt, dass die Nebenbedingungen sowohl

- von *Zahlenkonstanten*,
- von den *unveränderlichen Werten für die Simulationsmodellattribute*,
- von den *Werten für die Entscheidungsvariablen $\vec{x}$*,
- von den *zu bewertenden Ausgabewerten eines Simulationsexperimentes $\vec{f_A}(\vec{x})$* sowie
- von den *nicht zu bewertenden Ausgabewerten eines Simulationsexperimentes $\vec{f_N}(\vec{x})$*

abhängig sein können. Die unveränderlichen Werte für die Simulationsmodellattribute werden in den Nebenbedingungen zwar in Form von Variablen, durch Angabe ihres Variablennamens, dargestellt, treten während eines Laufes zur simulationsbasierten Optimierung aber als Konstante auf. Dagegen sind die Werte der sonstigen auftretenden Variablen nicht im Vorfeld eines Laufes zur simulationsbasierten Optimierung berechenbar. Entscheidend ist diese Unterteilung in konstante und variable Werte für die Unterteilung der Nebenbedingungen in

Pre-Nebenbedingungen.
Nebenbedingungen, deren Prüfung auf Erfüllbarkeit bereits im Vorfeld des Simulationsexperimentes möglich ist (setzt die Unabhängigkeit von den Ausgabewerten des Simulationsexperimentes voraus) sowie

Post-Nebenbedingungen.
Nebenbedingungen, welche erst im Anschluss an das Simulationsexperiment auf Erfüllbarkeit geprüft werden können.

Aus den Pre-Nebenbedingungen ergibt sich der zulässige Bereich $\mathcal{X}_{Pre}$ für die Werte der Entscheidungsvariablen, wobei $\mathcal{X}_{Pre} \subseteq \mathcal{X}$. Die Rückwirkung der Post-Nebenbedingungen kann diesen zulässigen Bereich weiter einschränken, so dass sich als insgesamt zulässiger Bereich für die Werte der Entscheidungsvariablen $\mathcal{X}_{Zul} = \mathcal{X}_{Pre} \cap \mathcal{X}_{Post} \subseteq \mathcal{X}$ ergibt. Zur Verdeutlichung dieses Zusammenhanges ist in Abb. 2.4 dargestellt, wie sich der Raum der Entscheidungsvariablen und der Raum der Ausgabewerte in Folge der Pre- und Post-Nebenbedingungen gestaltet. Dazu werden im Schritt 1 zu einem gültigen Vektor mit Werten für die Entscheidungsvariablen $\vec{x} \in \mathcal{X}_{Pre}$ die zugehörigen Ausgabewerte des Simulationsexperimentes $\vec{f_A}(\vec{x})$ und $\vec{f_N}(\vec{x})$ bestimmt und auf ihre Gültigkeit anhand der Post-Nebenbedingungen geprüft. Somit schränken die nicht erfüllten Post-Nebenbedingungen im Schritt 2 den Raum für die Werte der Entscheidungsvariablen rückwirkend weiter ein.

Abb. 2.4

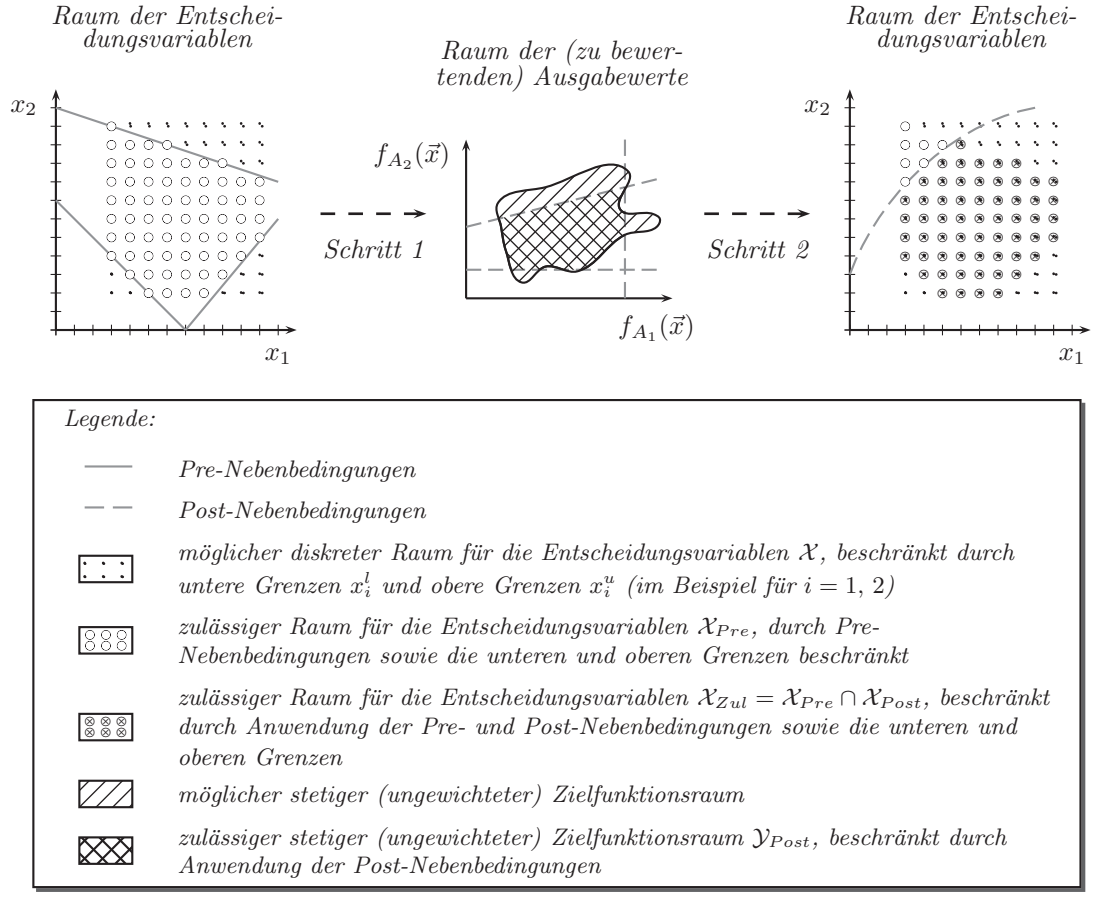


Abb. 2.4.: *Beschränkung des Raumes der Entscheidungsvariablen und der Ausgabewerte durch Nebenbedingungen (Beispiel)*

Als Beispiel sei sich an die im Kap. 6 vorgestellten Produktions- und Lagerhaltungssysteme angelehnt. Innerhalb der dort untersuchten Modelle treten als Losgrößen- und Freigabe-Entscheidung sog. (s, S)-Strategien auf, welche zum Zeit-

punkt der Unterschreitung von s_i eine Menge von $S_i - I_i(t)$ (Teilen) anfordern bzw. produzieren. Diese Strategien besitzen zunächst eine (deterministische) Nebenbedingung, welche es verbietet, dass die obere Schranke S_i kleiner als oder gleich s_i wird, so dass gilt $s_i < S_i$. Diese Nebenbedingung ist eine Pre-Nebenbedingung, weil sie bereits im Vorfeld eines Simulationsexperimentes überprüft werden kann. Als zusätzliche (deterministische) Nebenbedingung, könnte noch eine Forderung auftreten, welche die Umrüstkosten pro Zeiteinheit $SC(\pi)$ auf einen bestimmten oberen Grenzwert $x_{SC(\pi)}$ beschränkt, so dass bspw. gelte $SC(\pi) \geq x_{SC(\pi)}$. Eine solche Nebenbedingung ist folglich eine Post-Nebenbedingung und erst nach einem Simulationslauf überprüfbar, weil im Vorfeld eines Simulationslaufes keine Informationen über den Wert von $SC(\pi)$ bekannt sind.

Daraus lässt sich der gesamte zulässige Bereich $\mathcal{X}_{Zul}$, welcher sich sowohl über den Raum der Entscheidungsvariablen als auch den Raum der zu bewertenden und nicht zu bewertenden Ausgabewerte erstreckt, wie folgt festlegen:

$$\begin{aligned} \mathcal{X}_{Zul} \quad = \quad & \{\vec{x} \in \mathcal{X}_{Pre} \cap \mathcal{X}_{Post} \,|\, \vec{g}(\vec{x}, \vec{f_A}(\vec{x}), \vec{f_N}(\vec{x})) \leq \vec{0} \\ & \wedge \vec{h}(\vec{x}, \vec{f_A}(\vec{x}), \vec{f_N}(\vec{x})) = \vec{0}\} \quad . \end{aligned} \tag{2.6}$$

Als Schlussfolgerung der Existenz von Pre- und Post-Nebenbedingungen ergibt sich, dass stets nur die Ausführung der Simulationsexperimente notwendig ist, bei denen die Erfüllung aller Pre-Nebenbedingungen erfolgreich war.

Deterministische und stochastische Nebenbedingungen. Von der strukturellen Gestalt lassen sich Nebenbedingungen zudem in *deterministische* und *stochastische Nebenbedingungen* unterteilen, wobei sich die deterministischen Nebenbedingungen wiederum in explizite und implizite gliedern lassen (siehe Abb. 2.5). Deterministische Nebenbedingungen werden teilweise auch als *scharfe* bzw. *harte* Nebenbedingungen bezeichnet. Sie können entweder im Vorfeld eines Simulationsexperimentes oder im Anschluss an einen Simulationslauf überprüft werden. Im Gegensatz dazu existieren noch stochastische Nebenbedingungen, auch *unscharfe* bzw. *weiche* Nebenbedingungen genannt. Deren Überprüfung ist während eines Simulationslaufes durchzuführen. In Tabelle 2.1 sind für deterministische und stochastische Nebenbedingungen ausgewählte Beispiele und deren Bedeutung angegeben.

Abb. 2.5

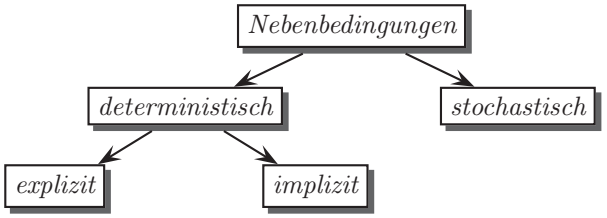


Abb. 2.5.: *Typen von Nebenbedingungen nach [Fu01a]*

Nebenbedingungstyp	Beispiel	Bedeutung
deterministisch, explizit	$x_1 < 100$	Der Wert der Variablen x_1 muss kleiner als 100 sein.
deterministisch, implizit	$\sum_{i=1}^{n} x_i < 100$	Die Summe der Werte der Komponenten des Vektors $\vec{x}$ mit $\|\vec{x}\| = n$ muss kleiner als 100 sein.
stochastisch	$P(t_w > 5) < 0,05$	Die Wahrscheinlichkeit, dass die Wartezeit t_w größer als 5 (Zeiteinheiten) ist, muss kleiner als 0,05 sein.

Tab. 2.1

Tab. 2.1.: *Beispiele für Nebenbedingungen*

2.2.5. Zielfunktionsraum

Je nachdem, ob es sich um ein *Minimierungsproblem* ($f(\vec{x}) \rightarrow \min$) oder *Maximierungsproblem* ($f(\vec{x}) \rightarrow \max$) handelt, ist die Lösung mit der minimalen bzw. maximalen Bewertung gesucht. Eine solche Lösung wird auch als *Optimum* bzw. *optimale Lösung* bezeichnet (siehe Def. 2.9). Spricht man nur von „der Lösung“ wird von einem einkriteriellen Optimierungsproblem ausgegangen. Bei mehrkriteriellen Optimierungsproblemen wird stets von einer *Menge von Pareto-optimalen Lösungen* oder kurz der *Pareto-optimalen Menge* [dt.; *pareto-optimal set* [engl.]] gesprochen (siehe Def. 2.6). Die Pareto-Optimalität sei hierfür wie folgt definiert.

Definition 2.5 (Pareto-Optimalität):
Eine Lösung $\vec{x}_A \in \mathcal{X}_{Zul}$ ist dann Pareto-optimal, wenn es keine andere Lösung $\vec{x}_B \in \mathcal{X}_{Zul}$ gibt, welche die Lösung $\vec{x}_A$ dominiert. Eine Lösung $\vec{x}_A$ dominiert eine Lösung $\vec{x}_B$ g. d. w. er mindestens hinsichtlich eines (bewerteten) Zielfunktionswertes besser und hinsichtlich keines (bewerteten) Zielfunktionswertes schlechter ist. Hierfür gelte folgende Notation, wenn alle Zielfunktionswerte zu minimieren sind:

Def. 2.5

$$\begin{aligned} \vec{x}_A \succ \vec{x}_B \quad \Leftrightarrow \quad & \forall\, i \in \{1, 2, \ldots, m\} : f_i(\vec{x}_A) \leq f_i(\vec{x}_B) \\ & \wedge \exists\, j \in \{1, 2, \ldots, m\} : f_j(\vec{x}_A) < f_j(\vec{x}_B) \quad . \end{aligned} \tag{2.7}$$

Bei Maximierung der Zielfunktionswerte gelte Entsprechendes ($\geq$ und $>$). □

In Abhängigkeit von der Gestalt der Zielfunktionen kommt es bei MOPs sehr häufig vor, dass keine einzelne optimal bewertete Lösung existiert oder gefunden wird, welche in allen Zielfunktionswerten die maximale bzw. minimale Bewertung besitzt. Ziel bei MOP ist vielmehr die Bestimmung mehrerer, in den Zielfunktionswerten unabhängiger, konkurrierender Lösungen, welche in der Pareto-optimalen Menge, auch Pareto-Front oder Pareto-optimale Front genannt, zusammengefasst werden. Aus dieser Pareto-optimalen Menge kann bspw. ein Anwender für seinen (speziellen) Anwendungsfall geeignete Lösungen auswählen (siehe Abschn. 2.6). Die Pareto-optimale Menge werde dazu wie folgt definiert.

Definition 2.6 (Pareto-optimale Menge):
Die Pareto-optimale Menge bezeichnet die Menge der Lösungen mit den (gewich-

Def. 2.6

teten) Zielfunktionswerten, welche bei der mehrkriteriellen Optimierung den Kriterien der Pareto-Optimalität genügen. □

Beispielsweise besteht die Pareto-optimale Menge der in Abb. 2.6 dargestellten Lösungen $\vec{x}_i$, $i \in \{A, B, C, D\}$ aus den Lösungen $\vec{x}_A$ und $\vec{x}_B$, welche von keiner der anderen Lösungen dominiert werden.

Abb. 2.6

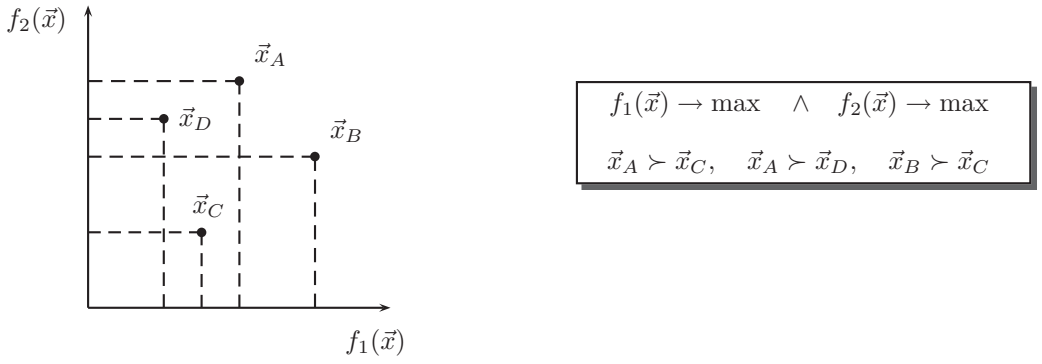


Abb. 2.6.: *Pareto-optimale Menge (Beispiel)*

Die globale Pareto-optimale Menge wird wie folgt definiert.

Def. 2.7

Definition 2.7 (Globale Pareto-optimale Menge):
Eine Menge $\mathcal{X}_{gopt} \subseteq \mathcal{X}_{Zul}$ wird als global Pareto-optimale Menge bezeichnet, g. d. w.

$$\forall\, \vec{x}_A \in \mathcal{X}_{gopt} : \nexists\, \vec{x}_B \in \mathcal{X}_{Zul} : \vec{x}_B \succ \vec{x}_A \quad . \tag{2.8}$$

□

Bei MOPs wird zudem noch von der lokalen Pareto-optimalen Menge gesprochen, welche sich im Verlaufe der Anwendung eines Optimierungsalgorithmus der (globalen) Pareto-optimalen Menge allmählich annähert.

Def. 2.8

Definition 2.8 (Lokale Pareto-optimale Menge):
Eine Menge $\mathcal{X}_{lopt} \subseteq \mathcal{X}_{Zul}$ wird als lokal Pareto-optimale Menge bezeichnet, g. d. w.

$$\begin{aligned} \forall\, \vec{x}_A \in \mathcal{X}_{lopt} : \nexists\, \vec{x}_B \in \mathcal{X}_{Zul} : \quad & \vec{x}_B \succ \vec{x}_A \quad \wedge \quad \| \vec{x}_B - \vec{x}_A \| \leq \epsilon_{lopt} \\ & \wedge \quad \| \vec{f}(\vec{x}_B) - \vec{f}(\vec{x}_A) \| \leq \delta_{lopt} \quad , \end{aligned} \tag{2.9}$$

wobei $\| \cdot \|$ eine beliebige Abstandsnorm ist und $\epsilon_{lopt} > 0$, $\delta_{lopt} > 0$ sind. □

Für endlich-dimensionale Räume $\mathbb{R}^n$ sind bspw. die so genannten *p-Normen* definiert als:

$$\|\vec{v}\|_p = \Big(\sum_{i=1}^{n} |v_i|^p\Big)^{1/p} \tag{2.10}$$

$$\|\vec{v}\|_\infty = \max_{i=1}^{n} |v_i| \quad . \tag{2.11}$$

Dabei ist $p \geq 1$ eine reelle Zahl und $|x_i|$ der Absolutbetrag des i-ten Wertes des Vektors $\vec{x}$.

Darüber hinaus ist in Abb. 2.7 der Unterschied zwischen der globalen und einer möglichen lokalen Pareto-optimalen Menge an einem willkürlichen Beispiel dargestellt. In Abb. 2.7 soll u. a. auch die Annäherung der lokalen Pareto-optimalen Menge an die globale Pareto-optimale Menge im Verlaufe der Optimierung erkennbar sein. Letztendlich kann die lokale Pareto-optimale Menge nach endlich vielen Optimierungsschritten sogar identisch mit der globalen Pareto-optimalen Menge sein.

Abb. 2.7

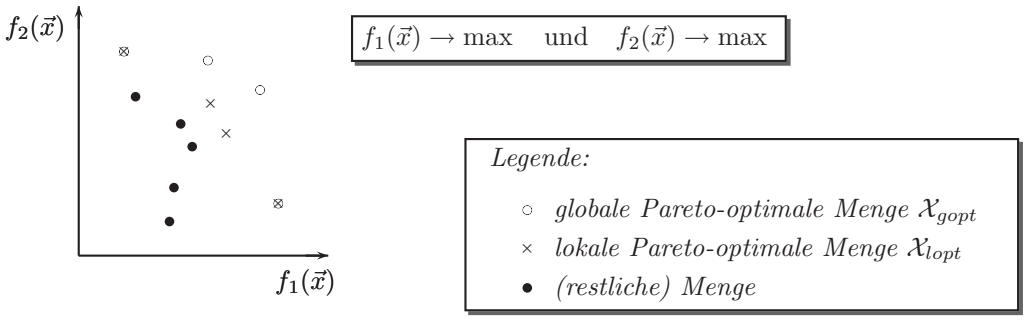


Abb. 2.7.: *Globale und lokale Pareto-optimale Menge bei mehrkriterieller Optimierung (Beispiel)*

Im Falle der mehrkriteriellen Optimierung können Maximierungs- und Minimierungsprobleme auch gemeinsam auftreten, wobei prinzipiell jedes Maximierungsproblem durch folgende Gleichungen in ein Minimierungsproblem überführbar ist und umgekehrt

$$f(\vec{x}) \rightarrow \max \equiv -f(\vec{x}) \rightarrow \min \tag{2.12}$$

$$f(\vec{x}) \rightarrow \min \equiv -f(\vec{x}) \rightarrow \max \quad . \tag{2.13}$$

Die Schwierigkeit eines Optimierungsproblems besteht oftmals nicht darin, eine optimale Lösung zu finden, sondern lediglich eine (sub)optimale Lösung mit geforderter Bewertung. Man spricht dann von einem *Suchproblem* zur Bestimmung der optimalen Lösung des Optimierungsproblems. Zu einem Optimierungsproblem lässt sich somit leicht ein *Entscheidungsproblem* kreieren, indem zur Eingabe noch eine *Zahl als Begrenzung* hinzugenommen und gefragt wird, ob eine Lösung existiert, deren Bewertung kleiner (Minimierungsproblem) bzw. größer (Maximierungspro-

blem) als die angegebene Zahl ist. Der zugehörige Algorithmus, der ein solches Optimierungsproblem löst, wird *Optimierungsalgorithmus* genannt. Analog wird beim Minimierungs- und Maximierungsproblem genauer vom *Minimierungs-* oder *Maximierungsalgorithmus* gesprochen.

Für ein SOP gilt bei Anwendung der Def. 2.5 der triviale Fall, so dass nur die zweite Bedingung aus Gleichung (2.7) von Bedeutung ist. Bei den SOP wird daher nur von der (global) optimalen Lösung bzw. dem (globalen) Optimum gesprochen.

Def. 2.9 ***Definition 2.9 (Globales Optimum):***
Den größtmöglichen Wert (Maximierungsproblem) oder den kleinstmöglichen Wert (Minimierungsproblem), den die (bewertete) Zielfunktion $f(\vec{x})$ für beliebige, aber feste Werte der Entscheidungsvariablen $\vec{x} \in \mathcal{X}_{Zul}$ unter möglicherweise vorgegebenen Nebenbedingungen annehmen kann, wird als globales Optimum bezeichnet. Bei Minimierungsproblemen gelte:

$$\forall\, \vec{x} \in \mathcal{X}_{Zul} : \;\; \exists\, \vec{x}_{gopt} \in \mathcal{X}_{Zul} : \;\; f(\vec{x}_{gopt}) \leq f(\vec{x}) \quad . \tag{2.14}$$

Für Maximierungsprobleme gelte entsprechend die Beziehung $f(\vec{x}_{gopt}) \geq f(\vec{x})$. Darüber hinaus können auch mehrere globale Optima für unterschiedliche Wertebelegungen der Entscheidungsvariablen existieren. □

Die im Abschn. 2.3 erwähnten exakten Optimierungsalgorithmen bzw. Lösungsverfahren liefern bei Anwendung auf ein SOP stets die optimale Lösung des Optimierungsproblems zurück. Heuristische Optimierungsalgorithmen bzw. heuristische Suchverfahren hingegen ermitteln in den meisten Fällen nur eine *lokal optimale* bzw. *suboptimale Lösung* oder kurz *lokales Optimum*, welches bestimmten Gütekriterien genügt (siehe dazu auch Abschn. 2.6). Ein lokales Optimum ist dazu wie folgt definiert.

Def. 2.10 ***Definition 2.10 (Lokales Optimum):***
Gegeben sei der Zielfunktionsraum $\mathcal{Y}$ mit der Zielfunktion $f(\vec{x})$ und der Nachbarschaft $\mathcal{N}_{In}$ (siehe Def. 2.12), dann heißt $f(\vec{x}_{lopt})$ lokal optimal bezüglich $\mathcal{N}_{In}$, falls $f(\vec{x}_{lopt}) \leq f(\vec{x})$, $\forall\, \vec{x} \in \mathcal{N}_{In}(\vec{x}_{lopt})$ (Minimierungsproblem). Für ein Maximierungsproblem gelte in ähnlicher Weise $f(\vec{x}_{lopt}) \geq f(\vec{x})$, $\forall\, \vec{x} \in \mathcal{N}_{In}(\vec{x}_{lopt})$. □

Anzumerken ist, dass das *bestimmte Optimum* $\vec{x}_{opt}$ die globale optimale Lösung $\vec{x}_{gopt}$ zurück liefere, wenn diese durch Anwendung eines exakten Optimierungsverfahren ermittelbar ist bzw. das Optimierungsverfahren nachweislich im globalen Optimum terminiert. Andernfalls wird als bestimmtes Optimum $\vec{x}_{opt}$ das beste lokale Optimum $\vec{x}_{lopt}$ angenommen, welches bei der Terminierung des Optimierungsverfahrens erreicht wurde. Selbiges gelte für die *bestimmte Pareto-optimale Menge* $\mathcal{X}_{opt}$, welche je nach Bestimmbarkeit der globalen Pareto-optimalen Menge $\mathcal{X}_{gopt}$ diese oder lediglich die lokale Pareto-optimale Menge $\mathcal{X}_{lopt}$ annimmt.

Darstellungsmöglichkeiten. In Abhängigkeit der Anzahl der Zielfunktionen gibt es verschiedene Möglichkeiten, die visuelle Darstellung für die lokalen oder globalen Optima bei SOP bzw. für die lokale oder globale Pareto-optimale Menge bei MOP zu gestalten. Es seien für unterschiedliche Anzahlen von Zielfunktionen $m = \dim(\vec{f}(\vec{x}))$ und Entscheidungsvariablen $n = \dim(\vec{x})$ folgende Darstellungsmöglichkeiten vorgeschlagen, wobei von kartesischen Koordinatensystemen ausgegangen wird.

- Für $m = 1$ und $n = 1$ werden die Werte für die Entscheidungsvariablen und die Zielfunktionen in einem gemeinsamen zweidimensionalen Koordinatensystem dargestellt, wobei die Werte für die Entscheidungsvariablen an der Abszisse und die Zielfunktionswerte an der Ordinate abgetragen und die Optima entsprechend gekennzeichnet werden (siehe Abb. 2.8).
- Für $m = 1$ und $n = 2$ oder $m = 2$ und $n = 1$ werden, wenn möglich, ein oder mehrere zweidimensionale Koordinatensysteme oder ein dreidimensionales Koordinatensystem zur Darstellung genutzt (siehe Abb. 2.9 (a) und (b)).
- Für $m = 2$ oder $m = 3$ und $n > 1$ seien nur die Zielfunktionswerte in einem zwei- bzw. dreidimensionalen Koordinatensystem in geeigneter Form abgebildet (siehe Abb. 2.9 (b)).
- Für alle sonstigen Werte von m und n ($m \geq 1$ und $n \geq 2$) seien nur die Zielfunktionswerte entweder in mehreren zwei- bzw. dreidimensionalen Koordinatensystemen oder in so genannten Netz- oder Sterndiagrammen[12] (siehe [BAL+04] oder [Ehr00]) dargestellt (siehe Abb. 2.10), wobei an den jeweiligen Koordinatenachsen die einzelnen aktuellen und maximalen Zielfunktionswerte abgetragen werden.

2.2.6. Diskretisierung des Raumes der Entscheidungsvariablen und Permutationsprobleme

Reellwertige Entscheidungsvariablen zur Parameteroptimierung. Bei Betrachtung der Entscheidungsvariablen soll sich in dieser Arbeit auf *Entscheidungsvariablen für stetige Optimierungsprobleme im Rahmen einer Parameteroptimierung* (siehe [GBE+86]), kurz *reellwertige Entscheidungsvariablen* genannt, und *Entscheidungsvariablen für Permutationsprobleme im Rahmen einer Reihenfolgeoptimierung* (siehe ebenso [GBE+86]), kurz *Permutationsentscheidungsvariablen* genannt, beschränkt werden. Grund dafür ist, dass sich mit diesen beiden Typen von Entscheidungsvariablen ein Großteil der real-existierenden Problemstellungen abdecken lassen. Wie bereits im Abschn. 2.2.3 angesprochen, ist eine reellwertige Entscheidungsvariable x_i^{real} (i-tes Element von $\vec{x}^{real}$) durch eine untere Schranke x_i^l und eine obere Schranke x_i^u charakterisiert[13].

[12] Alternativ könnten vorzugsweise auch andere Formen von graphischen Informationsdarstellungen wie Linien-, Flächen-, Balken-, Stapel-, Tortendiagramme, etc. genutzt werden.

[13] Die oberen Schranken x_i^u und die unteren Schranken x_i^l werden auch als Randbedingungen bezeichnet.

Abb. 2.8

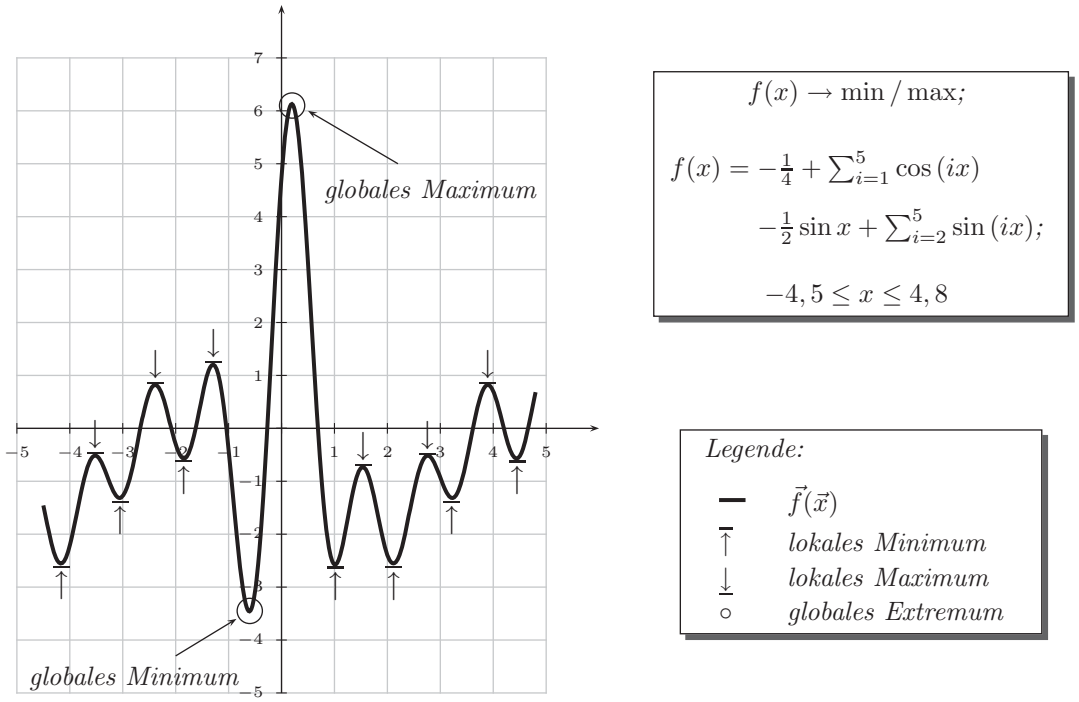


Abb. 2.8.: *Visualisierung der Zielfunktion und der Entscheidungsvariablen bei einem SOP ($m = 1$, $n = 1$)*

Mit Hilfe einer Schrittweite x_i^s lassen sich Entscheidungsvariablen für stetige Optimierungsprobleme bspw. auch für ganzzahlige Optimierungsprobleme verwenden. Durch eine Schrittweite x_i^s wird eine reellwertige Entscheidungsvariable x_i *diskretisiert*, wobei für x_i^s gelte:

$$0 < x_i^s \leq x_i^u - x_i^l, \quad \text{mit} \quad x_i^s \in \mathbb{R} \quad ,$$

so dass x_i Werte der Menge

$$M_{dis} = \{x_i^l + n \cdot x_i^s \,|\, \forall n \in \mathbb{Z} \ \cap \ 0 \leq n \leq \left\lfloor \tfrac{x_i^u - x_i^l}{x_i^s} \right\rfloor\}$$

annehmen kann, wobei $\mathbb{Z}$ den Bereich der ganzen Zahlen darstellt.

Auf diese Art und Weise definierte reellwertige Entscheidungsvariablen können somit u. a. auch

- *ganzzahlige Optimierungsprobleme* (x_i^l und $x_i^u \in \mathbb{N}$ mit $x_i^l < x_i^u$ sowie $x_i^s \in \mathbb{N}^*$, mit $\mathbb{N}$ als Bereich der natürlichen Zahlen inkl. der Zahl 0),

- *diskrete* bzw. *kombinatorische Optimierungsprobleme* ($x_i^l = 0$, $x_i^u = |M_1 \times M_2|$ und $x_i^s = 1$, wobei M_1 und M_2 die Mengen mit den entsprechenden Werten sind, x_i dem Index für den Zugriff auf die Menge $|M_1 \times M_2|$ entspricht und bspw. die Konvertierungsfunktionen $\text{index}(M_1) = x_i \bmod |M_1|$ sowie $\text{index}(M_2) = x_i \operatorname{div} |M_1|$ zur Bestimmung der Indexe für die konkreten Werte aus den Mengen M_1 und M_2 dienen.) sowie

Abb. 2.9

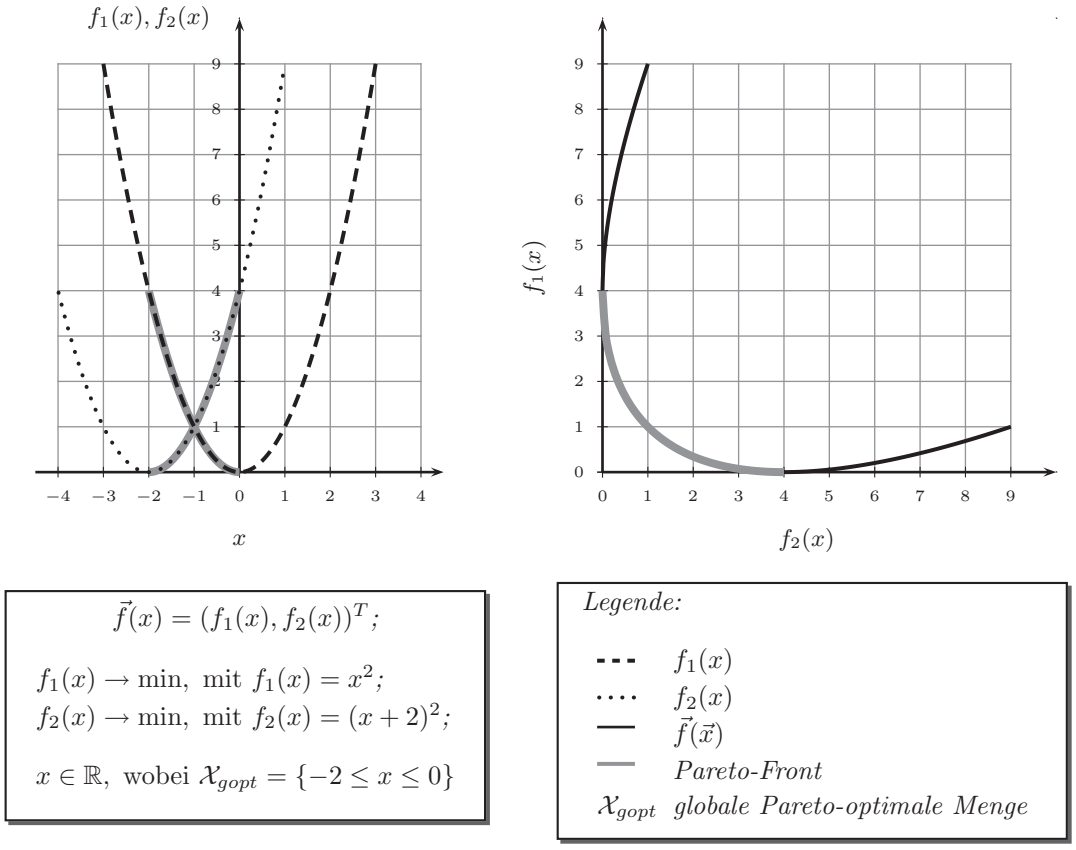


Abb. 2.9.: *Visualisierungsmöglichkeiten bei MOPs ($m = 2$ und $n = 1$)*

- *0/1-* bzw. *boolsche Optimierungsprobleme* ($x_i^l = 0$ und $x_i^u = x_i^s = 1$)

darstellen. In der englischsprachigen Literatur werden ganzzahlige Optimierungsprobleme als *Integer Optimisation Problems* [engl., kurz: *IOP*], die kombinatorischen Optimierungsprobleme als *Combinatorial Optimisation Problems* [engl., kurz: *COP*], und die boolschen Optimierungsprobleme als *Boolean Optimisation Problems* [engl., kurz: *BOP*] bezeichnet.

Permutationsentscheidungsvariablen. Wie bereits erwähnt, wird zur Lösung kombinatorischer Optimierungsprobleme mit spezieller Problemstruktur, wie sie bspw. das *Problem des Handlungsreisenden* [dt.; *Traveling Salesman Problem* [engl., kurz: *TSP*]] und das *Einplanungsproblem von Fertigungsaufträgen* [dt.; *Job Shop Scheduling Problem* [engl., kurz: *JSSP*]] (siehe [GI05] für das TSP und [Bru04] für das JSSP) aufweisen, ein weiterer, eigener Typ für Entscheidungsvariablen, die so genannten *Permutationsentscheidungsvariablen* verwendet. Unter Permutationsentscheidungsvariablen sind bestimmte Tupel von Elementen zu verstehen, welche die Menge X^n bilden. Die Reihenfolge der Elemente ist hierbei von zentraler Be-

Abb. 2.10

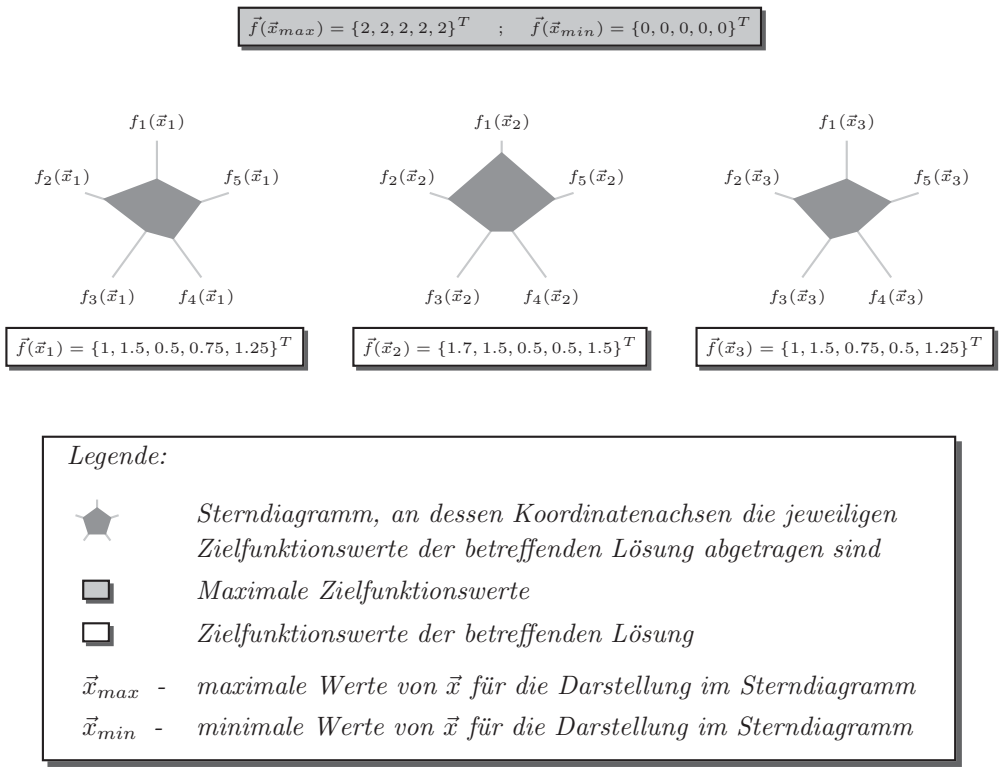


Abb. 2.10.: *Visualisierung der Zielfunktionswerte durch ein Sterndiagramm bei MOP (Beispiel für $m = 5$ und drei sich nicht dominierende Pareto-optimale Lösungen)*

deutung und kann durch eine Permutation der Elemente verändert werden. Eine Permutation ist wie folgt definiert.

Def. 2.11

Definition 2.11 (Permutation):
Unter einer n-stelligen Permutation ist die bijektive Abbildung $\pi : \Pi^n \rightarrow \Pi^n$ der Menge Π^n mit n Elementen auf sich selbst zu verstehen. □

Aus Def. 2.11 ergibt sich, dass sich als Anzahl aller Permutationen von n Elementen $n!$ ergibt. In der gewählten Darstellung für eine Permutation π wird diese als Vektor geschrieben (Permutationsvektor $\vec{\pi} \in \Pi^n$), welcher mehrere Funktionen enthält, die einen bestimmten Wert repräsentierten[14]:

$$\vec{\pi} = (p(1), p(2), \cdots, p(n))^T,$$

wobei $\vec{\pi}_i$ die Funktion $p(j)$ zurück liefere, welche sich an der i-ten Position im Permutationsvektor $\vec{\pi}$ befindet.

Streng genommen sind Permutationsentscheidungsvariablen auch mittels reellwertigen Entscheidungsvariablen als ein 0/1-Optimierungsproblem darstellbar. Aller-

[14] Die Funktionen $p(1)$, $p(2)$, …, $p(n)$ können dabei auch beliebige Bezeichner oder Zeichenketten für die einzelnen Mengenelemente zurückliefern.

dings weisen sie dann eine hohe Anzahl von Nebenbedingungen auf, um abzusichern, dass ein Element nur genau einmal in einer Permutation auftritt und dennoch beliebig verschiedene Permutationen zulässig sind. Die Darstellung einer n-stelligen Permutation mittels reellwertigen Entscheidungsvariablen würde wie folgt lauten:

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, 2, \ldots, n$$
$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, 2, \ldots, n$$

mit

$$x_{ij} \in \{0, 1\} \qquad i, j = 1, 2, \ldots, n \quad ,$$

wobei

x_{ij} - das j-ter Wert der i-ten Entscheidungsvariablen x darstellt.

Durch die hohe Anzahl von einschränkenden Nebenbedingungen ergeben sich sehr viele ungültige Lösungen. Der zeitliche Aufwand zur Überprüfung der Nebenbedingungen innerhalb eines Optimierungsverfahrens würde dadurch mit quadratischem Aufwand steigen. Auf Grund dieser Tatsache werden innerhalb dieser Arbeit die Permutationsentscheidungsvariablen genutzt.

Nachbarschaften. Für die Definition der lokalen Pareto-optimalen Menge in Def. 2.8 und bestimmte, der im Abschn. 4.3.5 vorgestellten Optimierungsverfahren wird eine Nachbarschaftsfunktion benötigt. Nachbarschaften können somit sowohl für Entscheidungsvariablen, als auch Zielfunktionswerte wie folgt definiert sein:

Definition 2.12 (Nachbarschaft):
Eine Nachbarschaft für eine Problemklasse P_0 ist gegeben durch eine Abbildung Def. 2.12

$$\mathcal{N}_{In} : S_{In} \rightarrow 2^{S_{In}}$$

für jede Instanz $In \in P_0$. $\mathcal{N}_{In}(\vec{z})$ heißt die Nachbarschaft von $\vec{z} \in S_{In}$, wobei $\mathcal{N}_{In}(\vec{z}) \subseteq S_{In}$. Ist In eindeutig festgelegt, so kann kurz $\mathcal{N}(\vec{z})$ geschrieben werden. □

Die Nachbarschaft $\mathcal{N}$ für eine lokal optimale Lösung stellt bei heuristischen Suchverfahren (siehe Abschn. 4.3), die bis zu einem bestimmten Zeitpunkt ermittelten Werte für die Entscheidungsvariablen unabhängig von dessen Typ dar. Aus diesen lässt sich dann nach Definition 2.10 und anhand der zugehörigen Zielfunktionswerte das lokale Optimum aus einer Nachbarschaft bestimmen.

Für stetige Optimierungsprobleme sind Nachbarschaften in natürlicher Weise als Epsilon-Umgebung bzgl. einer beliebigen Abstandsnorm (siehe Gleichungen 2.10 und 2.11) definiert. Bspw. gilt für die Nachbarschaft $\mathcal{N}_{real(\epsilon)}(\vec{z})$, wobei $\vec{z}$ für reelle Zielfunktionswerte oder einen Vektor mit Werten für eine reellwertige Entscheidungsvariablen stehen kann:

$$\mathcal{N}_{real(\epsilon)}(\vec{z}_1) \equiv \{\vec{z}_2 \mid \| \vec{z}_1 - \vec{z}_2\| \leq \epsilon\}, \text{ mit } \epsilon \in \mathbb{R}_+ \quad .$$

Die Abb. 2.11 zeigt ein Beispiel dazu, in welchem die zweidimensionale Nachbarschaft $\mathcal{N}_{real(\epsilon)}(\vec{x})$ zu einem zweidimensionalen Punkt $\vec{x} = (x_1, x_2)^T$ abgebildet ist. Der zweidimensionale Punkt $\vec{z} = (z_1, z_2)^T$ stellt dabei einen beliebigen Punkt der zweidimensionalen Nachbarschaft bzw. der Epsilon-Umgebung dar.

Abb. 2.11

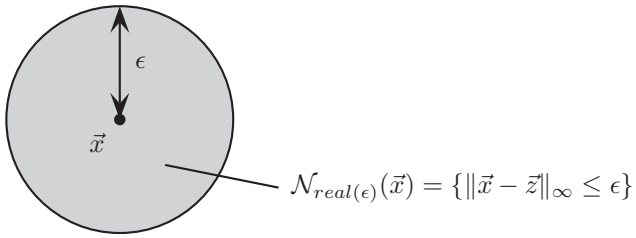


Abb. 2.11.: *Epsilon-Nachbarschaft bei reellwertigen Entscheidungsvariablen*

Für ein kombinatorisches Optimierungsproblem und die gewählte Darstellung mittels Permutationsentscheidungsvariablen ist die Angabe einer Nachbarschaft etwas komplexer (siehe bspw. [Pid96, S. 294f.]). Zunächst muss man sich über die Größe der Nachbarschaft Gedanken machen. Wie bereits zuvor beschrieben (s. o.), hängt bei stetigen Optimierungsproblemen die Größe der Nachbarschaft direkt vom Parameter ϵ ab. Bei Permutations- bzw. Reihenfolgeproblemen, wie dem TSP, muss, infolge der gewählten Darstellung, ein Operator zur Umstrukturierung des Permutationsvektors definiert werden. Im einfachsten Falle kann dies ein *Zwei-Punkt-Tausch-Operator* sein, durch welchen eine *Zwei-Punkt-Tausch-Nachbarschaft* entsteht.

Def. 2.13

Definition 2.13 (Zwei-Punkt-Tausch-Operator):
Ein Zwei-Punkt-Tausch-Operator[15] *vertausche die Funktionswerte* $p(a)$ *und* $p(b)$ *mit den Indexen* i *und* j *der Permutation* π*, wobei* $\dim(\vec{\pi}) \geq 2$ *und* $i \neq j$ *gelte. Die anderen Funktionswerte bleiben unverändert.* □

Dieser Zwei-Punkt-Tausch-Operator bedeutet bspw. für den Permutationsvektor

$$\vec{\pi} = (p(1), p(2), p(3), p(4), p(5), p(6))^T$$

[15] Der Zwei-Punkt-Tausch-Operator bzw. dessen Ergebnis wird auch als *Transposition einer Permutation* bezeichnet. Zum Erhalt der Allgemeingültigkeit der Aussagen innerhalb dieser Arbeit wird jedoch der Begriff Zwei-Punkt-Tausch-Operator verwendet.

und die Tauschpunkte $i = 2$ und $j = 6$ (bzw. $i = 6$ und $j = 2$) das folgender Permutationsvektor entsteht

$$\vec{\pi} = (p(1),\, p(6),\, p(3),\, p(4),\, p(5),\, p(2))^T \quad .$$

Mittels dieses Zwei-Punkt-Tausch-Operators kann eine Zwei-Punkt-Tausch-Nachbarschaft wie folgt gebildet werden.

Definition 2.14 (Zwei-Punkt-Tausch-Nachbarschaft): Def. 2.14
Eine Zwei-Punkt-Tausch-Nachbarschaft ist durch alle $\frac{n\cdot(n-1)}{2}$ voneinander verschiedenen Möglichkeiten des Zwei-Punkt-Tausch-Operators festgelegt. □

Für eine Zwei-Punkt-Tausch-Nachbarschaft gelte folgende Notation:

$$\mathcal{N}_{permut(1,1)}(\pi_i)$$

wobei

- $1, 1$ - angibt, dass ein Punkt mit Index i des Permutationsvektors π gegen einen beliebigen anderen Punkt mit Index $j \neq i$ des selben Permutationsvektors π getauscht werden soll und
- π_i - den Index des Elementes des Permutationsvektors bezeichne, von dem die Vertauschung ausgehen soll.

In Abb. 2.12 ist an einem Beispiel die Zwei-Punkt-Tausch-Nachbarschaft zu einem einzelnen Punkt des Permutationsvektors angegeben.

Abb. 2.12

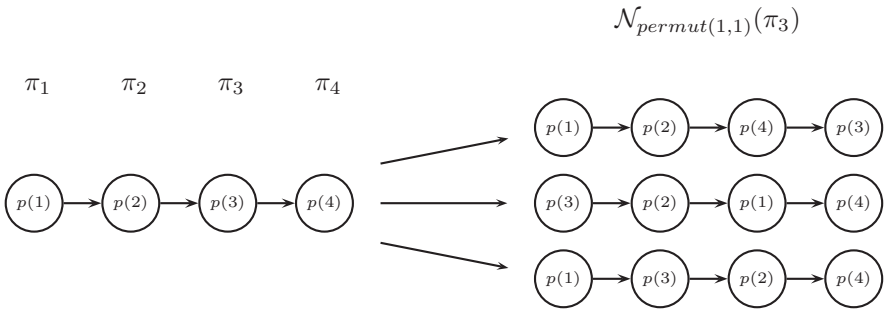


Abb. 2.12.: *Epsilon-Nachbarschaft bei Permutationsentscheidungsvariablen*

Es können durchaus auch andere Tausch-Operatoren definiert werden, welche mitunter wesentlich komplexere Nachbarschaften ergeben, bspw. durch die Festlegung mehrerer Tauschpunkte oder des Austausches ganzer Punkteketten. Dabei ist allerdings zu beachten, dass kleine Nachbarschaften zu einer schnelleren und effizienteren Suche führen als große Nachbarschaften oder längere Ketten. Große Nachbarschaften hingegen bieten mitunter die Möglichkeit ein globales Optimum eher zu finden, indem lokale Optima besser verlassen werden können. Für die in Kap. 4 genutzten heuristischen Suchverfahren reicht allerdings die zuvor definier-

te Zwei-Punkt-Tausch-Nachbarschaft aus, weil die Arbeit der Bestimmung eines lokalen oder globalen Optimums Aufgabe des Suchverfahrens sein soll.

2.2.7. Definition zur simulationsbasierten Optimierung

Ein Lösungsverfahren, das ein Optimierungsproblem näherungsweise löst, wird *Approximationsalgorithmus* genannt. Die simulationsbasierte Optimierung stellt einen solchen Approximationsalgorithmus dar. Sie sei für die weiteren Ausführungen wie folgt definiert.

Def. 2.15

Definition 2.15 (Simulationsbasierte Optimierung):
Die simulationsbasierte Optimierung ist die Optimierung von zulässigen Werten für die Entscheidungsvariablen $\vec{x} \in \mathcal{X}_{Zul}$ anhand von (gewichteten) Zielfunktionswerten $\vec{f}(\vec{x})$ unter vorgegebenen Nebenbedingungen $\vec{g}(\vec{x}, \vec{f}_A(\vec{x}), \vec{f}_N(\vec{x}))$ und $\vec{h}(\vec{x}, \vec{f}_A(\vec{x}), \vec{f}_N(\vec{x}))$ basierend auf den Ausgabewerten $\vec{f}(\vec{x})$ von stochastischen (innerhalb dieser Arbeit vorzugsweise diskret ereignisorientierten) Simulationen. □

Darstellungsmöglichkeit des Optimierungsverlaufes. Neben den Darstellungsmöglichkeiten aus Abschn. 2.2.5 für den Raum der Entscheidungsvariablen und der Zielfunktionen ist es im Rahmen der simulationsbasierten Optimierung auch hilfreich den Verlauf der Optimierung graphisch darzustellen. Dadurch kann während eines Optimierungslaufes, dieser mittels entsprechender Interaktionsmöglichkeiten unterbrochen oder abgebrochen werden, wenn die beste Lösung einen bestimmten Wert erreicht hat. In Abb. 2.13 sei beispielhaft eine Zielfunktion (Abb. 2.13 (a)) und ein denkbarer Optimierungsverlauf (Abb. 2.13 (b)) unter Nutzung eines zweidimensionalen Koordinatensystems eines SOP dargestellt.

2.3. Lösungsverfahren zur simulationsbasierten Optimierung

Aus mathematischer Sichtweise ist die simulationsbasierte Optimierung ein wichtiges *Hilfsmittel zur Lösung analytisch schwer lösbarer Optimierungsprobleme*, wie sie bspw. die $\mathcal{NP}$-schweren Problemstellungen darstellen[16]. Deren exakte Lösung kann numerisch nicht oder nur sehr aufwändig bestimmt werden. Wie bereits beschrieben, nutzt die simulationsbasierte Optimierung die Verfahren der *Simulation und Optimierung zur Bestimmung der optimalen Werte für die Entscheidungsvariablen eines Optimierungsproblems* (vgl. Definition 2.15). Hierbei sind die Simulation und Optimierung entsprechend zu koppeln, wobei allerdings nicht genau geklärt ist, wie diese Kopplung stattfindet. Prinzipiell sind zwei unterschiedliche Vorgehensweisen denkbar:

[16] Näheres zur Berechenbarkeit und Komplexität von Optimierungsproblemen und -algorithmen ist u. a. in [AB02] zu finden. Im Wesentlichen dient der Begriff „$\mathcal{NP}$“ zur Klassifizierung von Problemen der Komplexitätstheorie, welche besonders aufwändig zu berechnen sind.

(a) Verlauf der Zielfunktion *(b) möglicher Optimierungsverlauf* Abb. 2.13

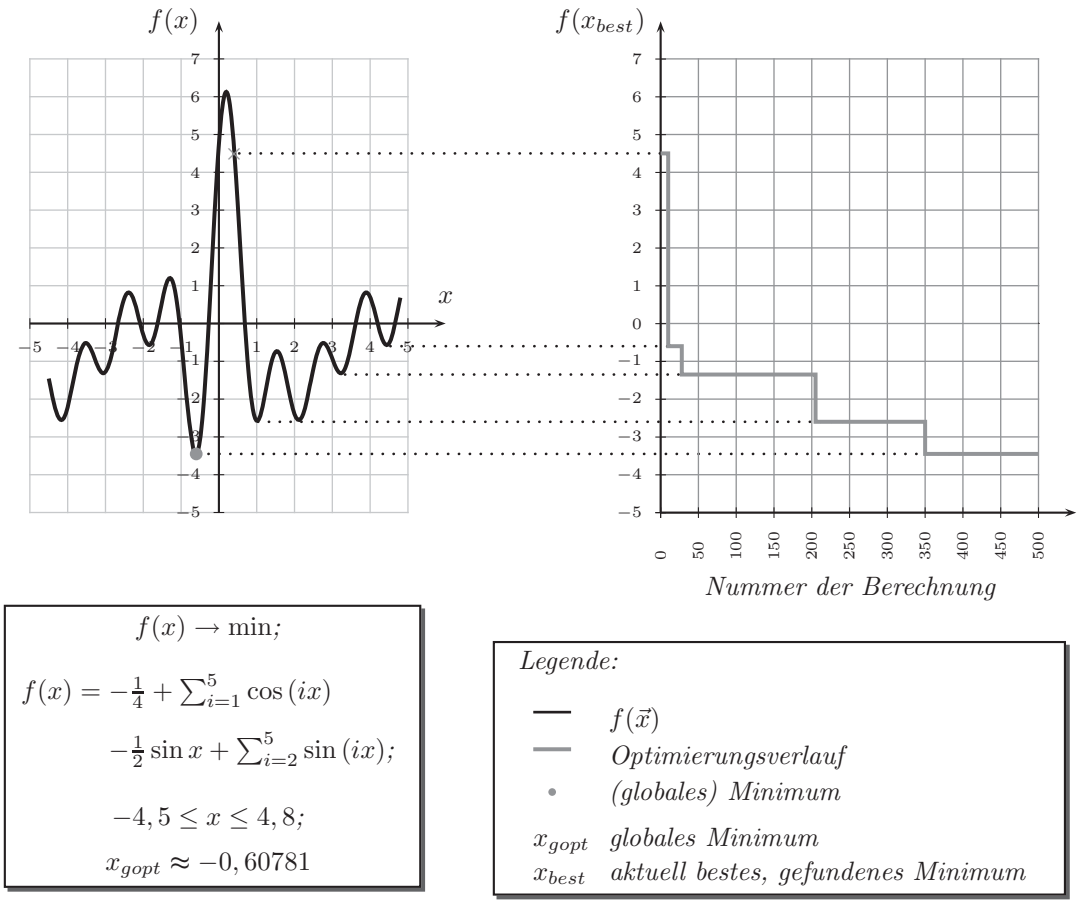


Abb. 2.13.: *Visualisierung einer Zielfunktion und eines Optimierungsverlaufes bei einem SOP ($m = 1, n = 1$)*

Iterative Simulation und Optimierung.
Ein oder mehrere ausgewählte Optimierungsverfahren schlagen gültige Werte für die Entscheidungsvariablen vor, welche dem Simulator als Eingabedaten dienen. Dabei sind mehrere Simulationsläufe mit ein und denselben Werten für die Entscheidungsvariablen möglich (siehe Abb. 2.14).

Sequentielle Simulation und Optimierung.
Es werden mehrere Simulationen mit ausgewählten, unterschiedlichen Werten für die Entscheidungsvariablen durchgeführt und anschließend die beste Belegung mit Hilfe verschiedener Auswahlkriterien bestimmt (siehe Abb. 2.15).

Diese grobe Einteilung kann noch weiter verfeinert werden und führt zu den in Abb. 2.16 dargestellten Kategorien der simulationsbasierten Optimierung. In Abb. 2.16 sind neben den Verfahren, welche zur iterativen und sequentiellen Verfahrensvariante gehören, auch noch weitere Verfahren vorgestellt. Diese werden vom Autor in Abb. 2.16 als eine weitere Verfahrensweise, die Verfahrensweise der simultanen Simulation und Optimierung, zusammengefasst. Bei dieser wird während eines Si-

Abb. 2.14

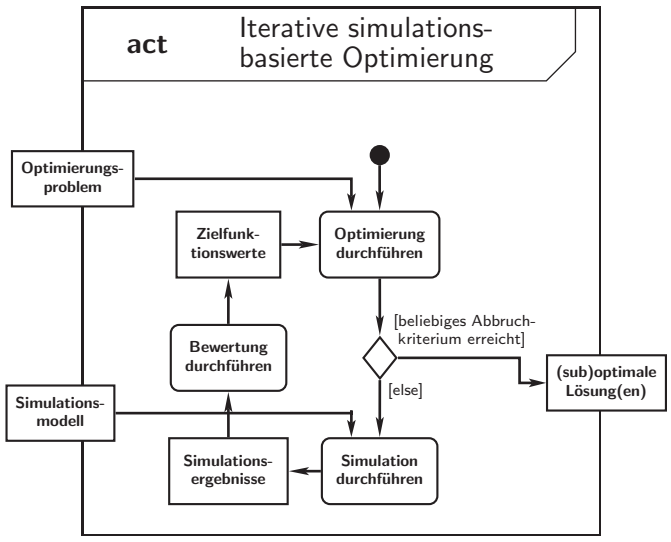


Abb. 2.14.: *Algorithmus zur iterativen simulationsbasierten Optimierung (UML-Aktivitätsdiagramm)*

Abb. 2.15

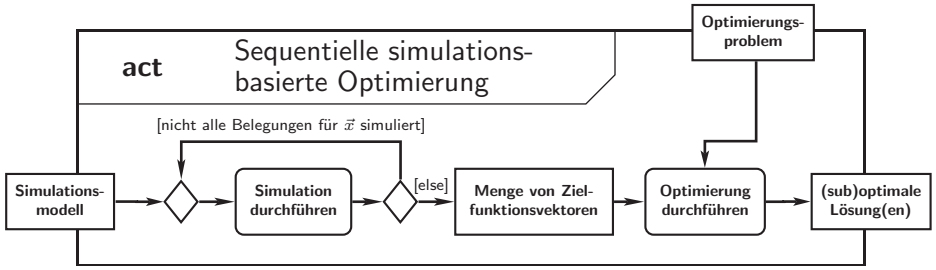


Abb. 2.15.: *Algorithmus zur sequentiellen simulationsbasierten Optimierung (UML-Aktivitätsdiagramm)*

mulationslaufes versucht, gültige, optimale Werte für die Entscheidungsvariablen z. B. durch Ermittlung oder Schätzung des Gradienten der Zielfunktion zu bestimmen. Im Englischen wird dieses Verfahren auch als *all at once* bezeichnet. Bei den in dieser Arbeit untersuchten Optimierungsproblemen handelt es sich jedoch um Optimierungsprobleme mit nichtlinearen Zielfunktionen[17] (siehe Abschn. 2.2.3), bei denen die Bestimmung der Gradienten der Zielfunktionen nur unter bestimmten Annahmen möglich ist. Die Möglichkeit der Anwendung der simultanen Simulation und Optimierung wird somit nicht weiter verfolgt und sei innerhalb dieser Arbeit nur zur Vollständigkeit erwähnt. Hinzu kommt, dass diese Vorgehensweise, nach Meinung des Autors, nicht der eigentlichen Methodik der simulationsbasierten Optimierung entspricht.

Das Verfahren der sequentiellen Simulation und Optimierung zeichnet sich eher bei der Anwendung auf kleine Probleminstanzen (< 100) aus, wie sie bspw. durch die Existenz vieler Nebenbedingungen entstehen können. Dadurch sind i. Allg. nur

[17] Das Gegenteil zu Optimierungsproblemen mit nichtlinearen Zielfunktionen stellen die Optimierungsprobleme mit linearen Zielfunktionen oder kurz lineare Optimierungsprobleme dar, bei denen sowohl die Zielfunktionen als auch alle Nebenbedingungen linear sind. Zu deren Lösung existieren leistungsfähige Optimierungsverfahren, wie bspw. das Simplexverfahren (siehe [GBE+86]).

wenige Szenerien miteinander zu vergleichen, so dass spezielle statistische Methoden zum Lösungsvergleich und zur Entscheidungsfindung angewandt werden. Ein weiterer Grund für die Existenz einer kleinen Anzahl von Probleminstanzen kann die Dauer der Simulation sein. Je länger ein Simulationsexperiment dauert[18], um so weniger Wertekonstellationen für die Entscheidungsvariablen können ggf. unter der Bedingung der Einhaltung eines bestimmten zeitlichen Rahmens untersucht werden. Auf Grund der Tatsache, dass in dieser Arbeit zunächst nur Wissen über die neuen Simulationsmodelle (siehe Kapitel 6) gesammelt werden soll, kommt das Verfahren der sequentiellen Simulation und Optimierung nicht zum Einsatz und wird daher auch nicht näher betrachtet.

Anzumerken ist, dass die Möglichkeiten zur *Einteilung der Lösungsverfahren für Optimierungsprobleme*, auch unabhängig vom Rahmen der simulationsbasierten Optimierung, sehr weitreichend sind. Eine innerhalb dieser Arbeit angewandte Möglichkeit ist die Einteilung in *exakte Lösungsverfahren* und *heuristische Suchverfahren*, wie sie im vorangehenden Abschn. 2.2.5 bereits angesprochen wurde. Während sich die exakten Lösungsverfahren mit der Suche nach der (global) optimalen Lösung bzw. der (global) Pareto-optimalen Menge befassen, ist das Ziel der heuristischen Suchverfahren lediglich das Auffinden einer lokal optimalen Lösung bzw. der lokal Pareto-optimalen Menge, welche bestimmten Gütekriterien genügt[19]. Allerdings kann bereits die Suche nach einer zulässigen Lösung ein $\mathcal{NP}$-schweres Problem darstellen, wenn eine hohe Anzahl von Nebenbedingungen existiert, wie bspw. im Falle eines Transport- bzw. Tourenplanungsproblems mit Zeitfenstern [dt.; Vehicle Routing Problem with Time Windows [engl., kurz: *VRPTW*]] (siehe [Käm06]) oder der Raum der Werte für die Entscheidungsvariablen sehr zerklüftet ist. Diese Problematik ist jedoch nicht Gegenstand der vorliegenden Arbeit und wird daher nicht näher betrachtet. Näheres kann jedoch in [RCN03] oder [KMS03] nachgelesen werden.

Durch die Anwendung der iterativen Simulation und Optimierung, können beide Verfahren für sich betrachtet werden, wodurch auch in den Kapiteln 3 und 4 getrennt auf diese eingegangen wird. Aus den vorangegangenen Bemerkungen ergibt sich, dass nachfolgend nur noch auf die *iterative Vorgehensweise bei der simulationsbasierten Optimierung von nichtlinearen Optimierungsproblemen* eingegangen wird.

[18] Die Durchführung eines Simulationsexperimentes kann evtl. mehrere Stunden dauern, wobei die Anwendung der verteilten und parallelen Simulation oder Optimierung nicht zwingend möglich sein muss.

[19] Ein Gütekriterium könnte bspw. die Entfernung der aktuell besten lokal optimalen Lösung von der global optimalen Lösung, falls diese bekannt ist, anhand einer vorgegebenen Abstandsmetrik sein.

Abb. 2.16

Abb. 2.16.: *Prinzipielle Ansätze der simulationsbasierten Optimierung (nach [CM97], [Fu01a] und [FGA05], Auszug)*

2.4. Rechnerinterne Modellierungsvariante

Bevor die mathematische Abbildung eines Modells und anschließend die rechnerinterne Modellabbildung erfolgt, findet eine *Systemanalyse* statt, in welcher evtl. eine *verbale Modellbeschreibung* aufgestellt wird (siehe Abb. 2.17). Diese verbale Modellbeschreibung wird anschließend in ein *mathematisches Modell* und dieses wiederum in ein *rechnerinternes Modell* überführt. Im verwendeten Kontext bezieht sich die rechnerinterne Modellierung auf die rechnerinterne Abbildung eines Modells für die Anwendung der simulationsbasierten Optimierung. Dies beinhaltet nur die Abbildung der notwendigen, strukturierten Datenwerte eines zu untersuchenden Systems bzw. dessen Modells, d. h. die statische Modellstruktur mit den permanenten Simulationsmodellelementen. Die Modellaktionen bzw. die Logik eines Modells, d. h. die dynamischen Modellbeziehungen und -interaktivitäten, sind in den jeweiligen Simulatoren untergebracht.

Abb. 2.17

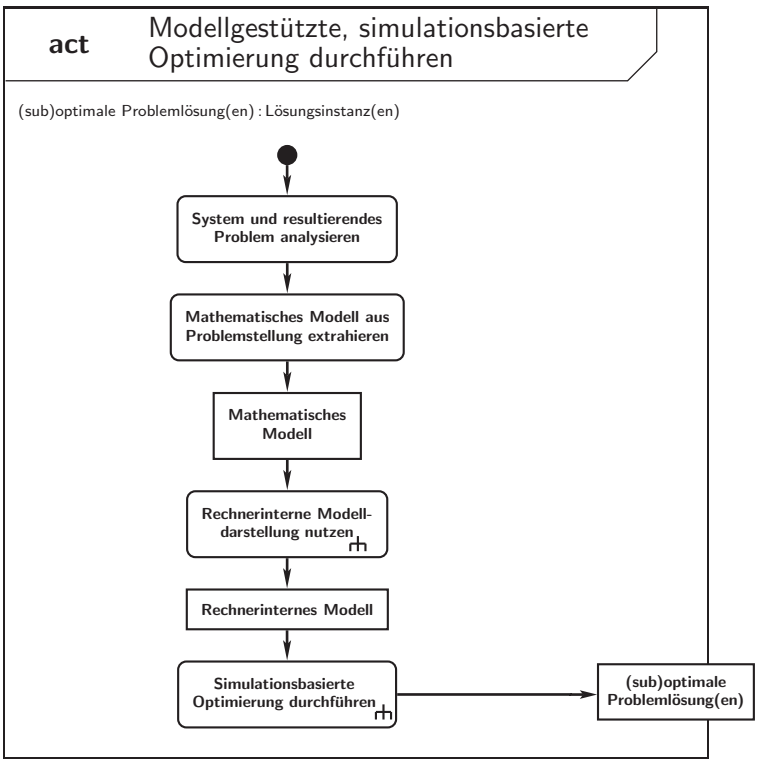


Abb. 2.17.: *Algorithmus zur modellgestützten, simulationsbasierten Optimierung als Grobdarstellung (UML-Aktivitätsdiagramm)*

Das Modell ist demnach so konstruiert, dass es den Ansprüchen der simulationsbasierten Optimierung genügt, d. h. es vereint als Eingabewerte ein Simulationsmodell für die Simulation und das Optimierungsproblem für die Optimierung miteinander. Zudem beinhaltet ein Modell auch die entsprechenden Datenwerte zur Bewertung der Ausgabewerte eines Simulationsexperimentes. Im Sinne der simulationsbasierten Optimierung werden daher rechnerintern jeweils zwei verschiedene Kategorien,

- die Kategorie der *Eingabegrößen* (Unterkategorien $IN_{CalcSim}$ und IN_{Opt}) sowie

- die Kategorie der *Ausgabegrößen* (Unterkategorien OUT_{Ass} und OUT_{NoAss}),

eines modellierten Systems unterschieden. Bereits in Abb. 2.1 war die Nutzung der Unterkategorien teilweise erkennbar. Die konkrete Bedeutung der einzelnen Kategorien und Unterkategorien ist in Tab. 2.2 etwas genauer widergespiegelt.

Tab. 2.2

Kategorie	Unterkategorie	Symbol	Erklärung
Eingabegrößen	veränderliche Eingabewerte des Simulationsmodells	$IN_{CalcSim}$	*statische Werte* während eines Laufes der simulationsbasierten Optimierung
	Entscheidungsvariablen des Optimierungsproblems	IN_{Opt}	*veränderliche Werte* während eines Laufes der simulationsbasierten Optimierung, bilden $\vec{x}$
Ausgabegrößen	zu bewertende Ausgabewerte des Simulationsmodells	OUT_{Ass}	fließen direkt oder indirekt in die *Zielfunktion* $f(\vec{x})$ bzw. den *Zielfunktionsvektor* $\vec{f}(\vec{x})$ ein, bilden $\vec{f_A}(\vec{x})$
	nicht zu bewertende Ausgabewerte des Simulationsmodells	OUT_{NoAss}	besitzen nur *statistischen Charakter*, um ggf. eine Validierung der Eingabegrößen durchzuführen, bilden $\vec{f_N}(\vec{x})$

Tab. 2.2.: *Kategorien eines rechnerinternen Modells*

Die einzelnen *Unterkategorien* beinhalten ihrerseits eine *Menge von Elementen mit mehreren Attributen*, welche jeweils verschiedene Bedeutung besitzen. Diese Bedeutung wird in Tab. 2.3 noch einmal aufgegriffen.

Tab. 2.3

Unterkategorie	Bedeutung eines Elementes	Bedeutung eines Attributes
$IN_{CalcSim}$	Elemente eines Simulationsmodells	Parameter des Elementes eines Simulationsmodells
IN_{Opt}	Feld mit den beiden Typen von Entscheidungsvariablen (reellwertig oder Permutation)	Entscheidungsvariable des jeweiligen Typs
OUT_{Ass}	zu bewertende Ausgabevariable eines Simulationsexperimentes	Gewichte und Optimierungsziel der zu bewertenden Ausgabevariablen
OUT_{NoAss}	Felder mit den statistischen Ausgabevariablen	betreffende statistische Ausgabevariable

Tab. 2.3.: *Elemente und Attribute der Kategorien eines rechnerinternen Modells*

Unterteilung in Modelldefinition und -deklaration (Modellersteller kontra Softwareersteller). Bei dem innerhalb dieser Arbeit gewählten Modellierungsansatz erfolgt eine Unterteilung in rechnerinterne Modelldeklaration und daraus resultierender rechnerinterner Modelldefinition, welche die eigentlichen Datenwerte des Modells enthält. Diese Vorgehensweise ermöglicht zum einen eine Trennung der Aufgaben bei der rechnerinternen Modellierung und zum anderen verringert sie die Fehleranfälligkeit bei der Eingabe der Datenwerte einer Modelldefinition zu einer Modelldeklaration (Prüfung auf logische und syntaktische Fehler einer Modelldefinition, vgl. Abschn. 5.3.6).

Der Ersteller eines Modells (Modellierer), welcher den Simulationsexperten repräsentiert, erstellt die Modelldefinition. Dabei nutzt er die im Vorfeld der Modelleingabe von einem Softwareersteller (Entwickler eines Simulators) bereitgestellte, rechnerinterne Modelldeklaration. Ein Modell ist somit stets an den jeweiligen Simulator und das evtl. daraus resultierende Optimierungsproblem gebunden. Die Erstellung einer Modelldeklaration erfordert i. Allg. eine enge Absprache zwischen Modell- und Softwareersteller zur Anpassung an die jeweiligen Bedürfnisse. Durch diese Interaktion zwischen beiden Akteuren verfeinert sich eine Modelldeklaration, dessen zugehöriger Simulator und zwangsläufig auch die resultierenden Modelldefinitionen iterativ. Wie sich eine solche sequentielle Vorgehensweise zur hinreichend genauen rechnerinternen Modellierung eines mathematischen Modells des zu untersuchenden Systems gestalten kann, ist in Abb. 2.18 dargestellt.

Abb. 2.18

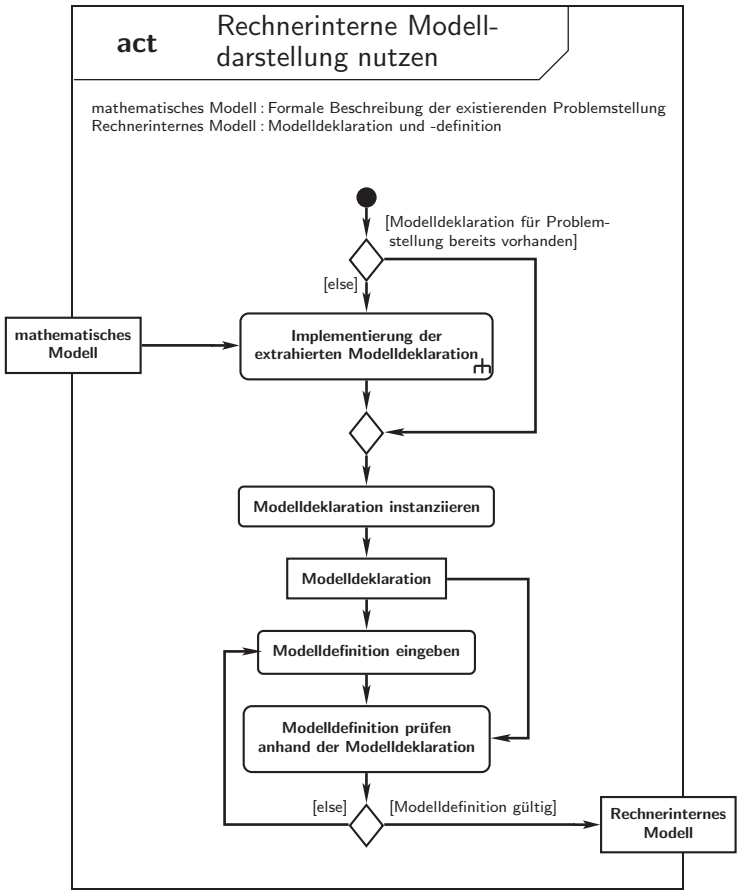


Abb. 2.18.: *Algorithmus zur Nutzung einer rechnerinternen Modelldarstellung im CAOS (UML-Aktivitätsdiagramm)*

Die Bindung der rechnerinternen Modelldeklaration an einen Simulator wird notwendig, um einen definierten Zugriff auf eine Modelldefinition zu besitzen, welcher wiederum unabhängig von den eingegebenen Datenwerten ist. Durch die Angabe von Standardwerten innerhalb einer Modelldeklaration für die Datenwerte wird es zudem möglich, dass die Instanziierung[20] einer Simulator-Klasse bereits eine

[20] Durch die Instanziierung wird bei objektorientierten Programmiersprachen ein neues Objekt von einer bestimmten Klasse erzeugt.

minimale Modelldefinition erzeugt. Mit dieser können dann bereits erste Simulationsexperimente ausgeführt werden. Die Modelldeklaration stellt somit nur ein *Gerüst* [dt.; *template* [engl.]] für eine Modelldefinition dar.

Greift man speziell die Datenwerte der Unterkategorie IN_{Opt} heraus, so lassen sich aus diesen die Werte für die Entscheidungsvariablen entnehmen. Dafür wird eine spezielle *Abbildungsfunktion* [dt.; *mapping function* [engl.]] verwendet, welche die Zuordnung der rechnerinternen Werte zu den Werten der Entscheidungsvariablen vornimmt. Diese Vorgehensweise ist in Abb. 2.19 erkennbar. Notwendig wird die Anwendung einer Abbildungsfunktion u. a. auch weil die implementierten Optimierer nicht direkt auf eine spezielle Modelldefinition zugreifen sollten, um die Unabhängigkeit der Optimierungsverfahren von einem bestimmten Optimierungsproblem und der gewählten rechnerinternen Modellierungsvariante zu ermöglichen.

Abb. 2.19

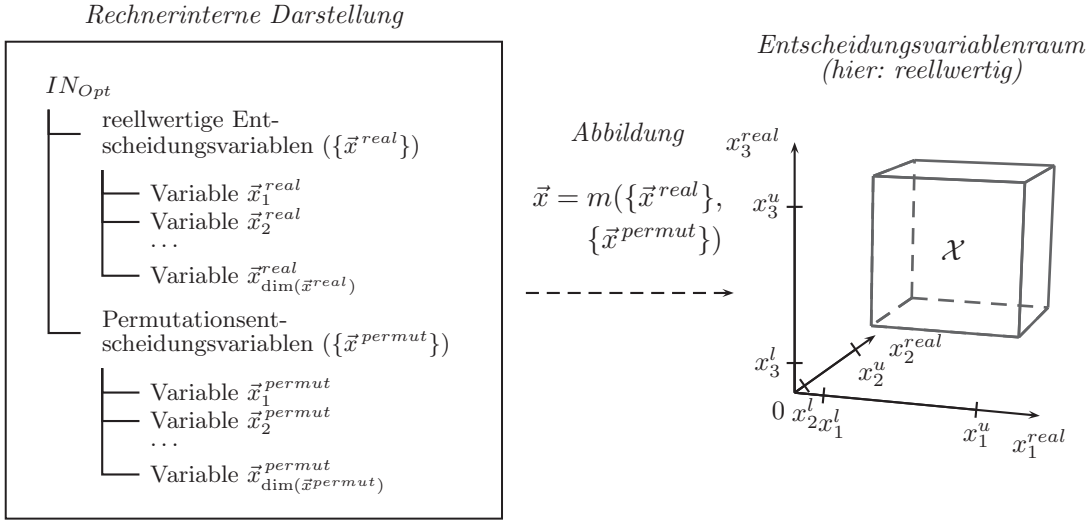


Abb. 2.19.: *Rechnerinterne Darstellung und Raum der Entscheidungsvariablen*

2.4.1. Schritte bei der rechnerinternen Modellierung

Die Fähigkeiten eines Simulators spiegeln sich beim gewählten Ansatz einerseits in der Implementierung des Simulationsablaufes und andererseits auch in dessen angegebener Modelldeklaration wider. Fehler innerhalb der Modelldeklaration eines Simulators ziehen zumeist auch Änderungen innerhalb des implementierten Simulators nach sich und umgekehrt. Daraus entsteht ein iterativer Prozess, der in Abb. 2.20 veranschaulicht ist. Er entsteht durch enge Rücksprache zwischen Modell- und Softwareersteller (siehe Legende zu Abb. 2.20).

Mit einer Modelldeklaration werden einem Modellersteller zwangsläufig auch die festgelegten Grenzen eines rechnerintern abgebildeten Simulationsmodells aufgezeigt. Später können nur die Modelldetails simuliert werden, die auch in der Modelldeklaration und dementsprechend im Simulator implementiert sind.

Abb. 2.20

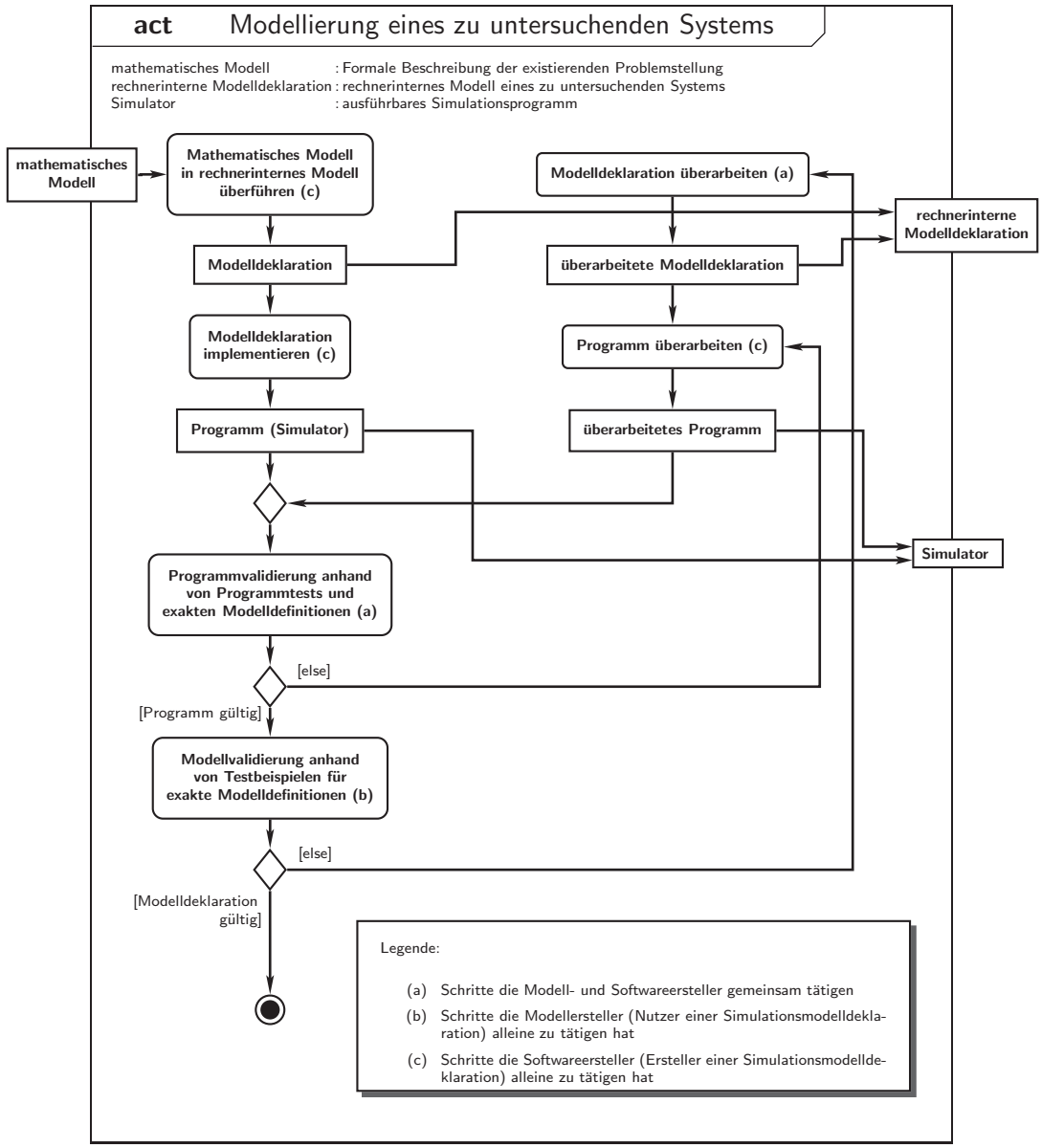


Abb. 2.20.: *Notwendige Schritte bei der Modellierung eines Realsystems*

2.4.2. Verwendung einer Modelldefinition

In Abb. 2.21 ist dargestellt, wie sich die Verwendung einer zulässigen bzw. gültigen Modelldefinition i. Allg. gestaltet. Demnach werden zunächst die verschiedenen Informationen zum Initialisieren des Simulators oder Berechners, des Optimierungsverfahrens sowie des Bewerters ausgelesen und die Initialisierung durchgeführt. Abschließend erfolgt die Durchführung des Laufes zur simulationsbasierten Optimierung und die entsprechende Rückgabe der (sub)optimalen Lösung des modellierten Optimierungsproblems.

Abb. 2.21

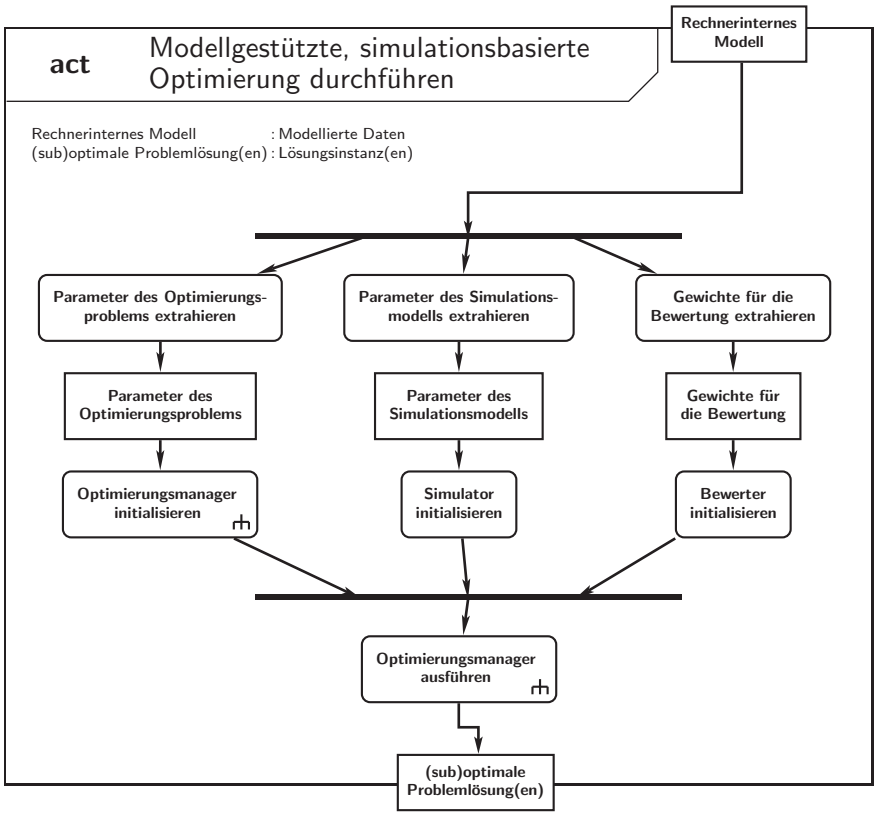


Abb. 2.21.: *Algorithmus zur modellgestützten, simulationsbasierten Optimierung als Feindarstellung (UML-Aktivitätsdiagramm)*

2.4.3. Sichtweisen auf ein rechnerinternes Modell

Durch die Anwendung verschiedener Sichtweisen wird die *logische/virtuelle Gliederung eines Modells* erreicht. Eine *Sichtweise* beschreibt bspw. die *Gruppierung von Elementen zu einer logischen Gruppe* oder einer *Untergruppe einer logischen Gruppe*. Daraus entsteht eine *virtuelle Hierarchie*, welche u. a. als Darstellungshilfe bei der Verständigung über den Aufbau eines Modells dienen kann. Denkbar ist bspw. eine Baumansicht der Kategorien, eine graphische Darstellung der Elemente des Simulationsmodells, eine Ansicht als Baum der zugehörigen XML-Datei, usw., um eine rechnerinterne Modelldeklaration oder -definition mittels verschiedener Ansichten zu repräsentieren. In Abb. 2.22 sind beispielhaft solche verschiedenen Ansichtsmöglichkeiten zur Erzeugung virtueller Hierarchien verdeutlicht. Mit der

im Softwaresystem CAOS gewählten Implementierungsvariante für die Modelldefinitionen und -deklarationen wird sich im Abschn. 5.3.6 etwas detaillierter auseinandergesetzt.

Abb. 2.22

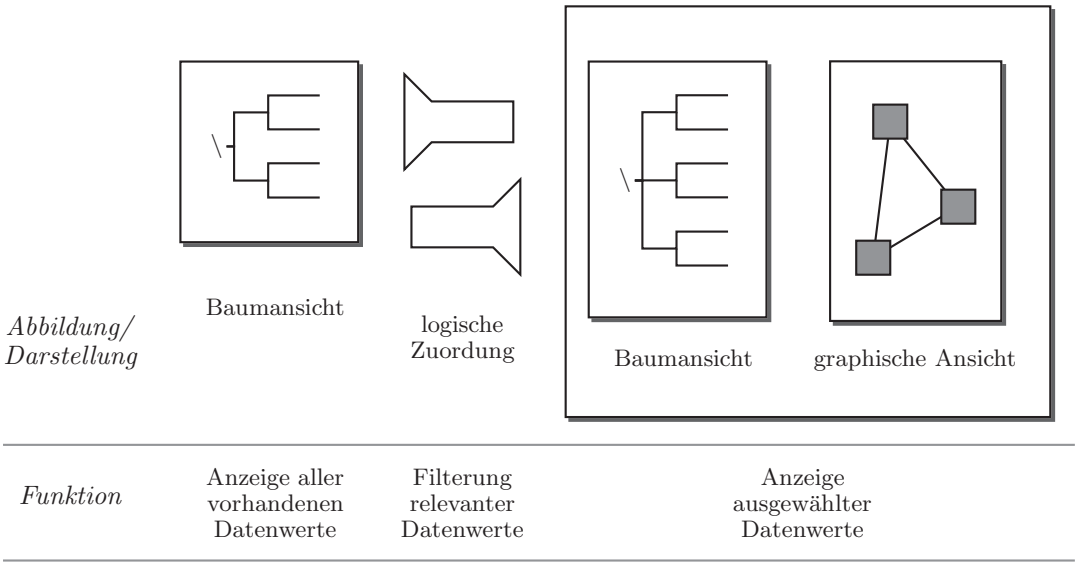


Abb. 2.22.: *Unterschiedliche Sichtweisen auf ein Modell mittels einer 3-Schicht-Architektur (model-view-controler [engl.])*

Denkbar ist auch die Nutzung unterschiedlicher Abstraktionsniveaus, um verschiedene Modellteile nur bis zu einer bestimmten Modelltiefe[21] erkennbar zu machen. Es ergibt sich zwangsläufig, dass die zuvor beschriebene Unterteilung in die vier verschiedenen Unterkategorien eines Simulationsmodells gemäß Tab. 2.3 durch eine Umgruppierung der rechnerinternen Modellelemente verändert werden kann, wobei die Datenwerte der jeweiligen Attribute der einzelnen Elemente der Unterkategorien jedoch grundsätzlich erhalten bleiben. Die Datenwerte der Attribute werden lediglich auf eine andere Sichtweise übertragen. Demnach könnte eine Modelldefinition, neben den eigentlichen Datenwerten, auch noch eine virtuelle Hierarchie oder Sichtweise auf die einzelnen Attribute mit den Datenwerten beinhalten. Diese Modelldefinition wäre dann nur für einen konkreten Anwendungsfall bestimmt.

2.4.4. Möglichkeit der Weiterverwendung eines Simulators

Ein einmal erstellter Simulator mit seiner Modelldeklaration kann teilweise unabhängig von seinem ursprünglichen Verwendungszweck in ähnlichen Bereichen weiterverwendet werden, wenn bspw. die gleiche oder eine ähnliche mathematische Problemstellung zugrunde liegt. Bspw. kann ein Scheduling-Verfahren zur Planung von Maschinenbelegungen (siehe [Bru04]) durch die Ähnlichkeit der grundlegenden

[21] Die Modelltiefe gibt an, auf welche Abstraktionsebene die Modellierung Bezug nimmt.

mathematischen Problemstellung ebenso gut als Planungsverfahren für Personal oder andere Ressourcen unter Verwendung anderer Nebenbedingungen eingesetzt werden. Die Erstellung einer Modelldeklaration ist somit zwar an den betreffenden, nutzenden Simulator, aber nicht zwingend an dessen weiteren Verwendungszweck gebunden[22].

2.4.5. Rechnerinterne Abbildung der Entscheidungsvariablen

In dieser Arbeit wird davon ausgegangen, dass die Möglichkeit besteht, verschiedenen Entscheidungsvariablen auch unterschiedliche Optimierungsverfahren zu zuweisen. Ziel dieser Vorgehensweise ist es, Teilprobleme des gesamten Optimierungsproblems mit dem jeweils am besten geeigneten Verfahren zu lösen. Dadurch kann es möglich sein, dass mehrere Optimierungsverfahren jeweils nur einen Teil der Werte für die Entscheidungsvariablen enthalten. Die Abb. 2.23 zeigt den Unterschied zwischen der allgemein üblichen und der innerhalb dieser Arbeit angewandten Vorgehensweise.

Diese Verknüpfungsmöglichkeit einer Entscheidungsvariablen mit dem jeweiligen Optimierungsverfahren wird im Abschn. 5.3.6 und dort speziell in Abb. 5.5 noch einmal aufgegriffen.

2.5. Bewertung

Zur Lösung von SOP und MOP sind im Rahmen dieser Arbeit im CAOS verschiedene ein- und mehrkriterielle Optimierungsverfahren implementiert worden (siehe Abschnitte 4.3 und 4.4). Der innerhalb dieser Arbeit gewählte Ansatz ermöglicht es, dass zur Lösung eines MOP nicht nur mehrkriterielle Optimierungsverfahren verwendet werden können, sondern auch einkriterielle Optimierungsverfahren bzw. beide Typen von Optimierungsverfahren gemeinsam. Dies geschieht durch *Überführung der verschiedenen Optimierungsprobleme* ineinander, je nachdem welches Optimierungsverfahren ausgewählt ist. Wird ein einkriterielles Optimierungsverfahren verwendet, obwohl das zugrunde liegende Optimierungsproblem ein MOP ist, muss eine Transformation der zu bewertenden Ausgabewerte $\vec{f}_A(\vec{x})$ in einen skalaren Zielfunktionswert $f(\vec{x})$ stattfinden. Dies erfolgt durch Summation der gewichteten Ausgabewerte. Prinzipiell kann aber auch eine beliebige andere Transformationsvorschrift verwendet werden, welche einen skalaren Zielfunktionswert für die Vergleichbarkeit *mehrerer Zielfunktionswerte mit unterschiedlichen Einheiten* bildet[23]. Bei der Summation wird es teilweise auch erforderlich, Werte auf-

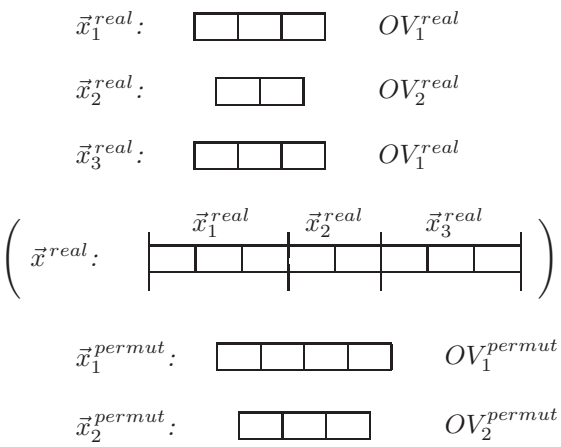
[22] Durch eine Nutzung des objektorientierten Programmierparadigmas, wie es u. a. auch im CAOS angewandt wird, und die damit verbundene Existenz von unterschiedlichen Klassen für Simulatoren ist die Wiederverwendbarkeit einer Modelldeklaration auch in anderen Simulatoren i. Allg. problemlos gewährleistet und durchaus sinnvoll.

[23] Bei der Kombination von Werten mit verschiedenen Einheiten kann die Gewichtung auch als Multiplikation mit einer Normalisierungskonstanten verstanden werden, um einen Wert mit gleichen Einheiten zu erzeugen. Bspw. wird dies im Rahmen des *AHP-Verfahrens* (Abkürzung für *Analytical*

Abb. 2.23

(a) allgemein übliche Vorgehensweise

(b) angewandte Vorgehensweise



Legende:

OV_{xy}	-	*beliebiges Optimierungsverfahren xy (kann sich auch nur in den Parametereinstellungen des Optimierungsverfahrens unterscheiden)*
OV_{xy}^{real}	-	*beliebiges Optimierungsverfahren xy für reellwertige Entscheidungsvariablen*
OV_{xy}^{permut}	-	*beliebiges Optimierungsverfahren xy für Permutationsentscheidungsvariablen*

Abb. 2.23.: *Zuordnung verschiedener Optimierungsverfahren zu unterschiedlichen Entscheidungsvariablen (Beispiel)*

zusummieren, welche *entgegengesetzte Optimierungsziele*, Maximierung oder Minimierung, verfolgen. Hierzu kann die bereits in den Gleichungen 2.12 und 2.13 angegebene Transformation genutzt werden.

Im folgenden Beispiel sind die Zielfunktionswerte eines MOP angegeben, welche in einen skalaren (einkriteriellen) Zielfunktionswert überführt werden:

$$\begin{aligned} f_1(\vec{x}) &\rightarrow \min \\ f_2(\vec{x}) &\rightarrow \max \end{aligned}$$

ergibt umgeformt als einkriterielles Minimierungsproblem bspw.

$$f(\vec{x}) \rightarrow \min$$

mit

$$f(\vec{x}) = w_1 \cdot f_1(\vec{x}) - w_2 \cdot f_2(\vec{x}) \quad ,$$

wobei

w_1, w_2 - Normalisierungkonstanten bzw. Gewichte der Zielfunktionswerte.

Eine andere Möglichkeit zur Umformung in einen skalaren Zielfunktionswert ist die *Hinzunahme von Nebenbedingungen für ausgewählte Zielfunktionen*. Hierbei wird bspw. eine der Zielfunktionen als die primäre Zielfunktion für das einkriterielle Optimierungsverfahren festgelegt. Die anderen Zielfunktionen werden in *einschränkende Nebenbedingungen* umgeformt, welche sich auch im Verlauf der Optimierung verändern können, wodurch sog. *dynamische Nebenbedingungen* entstehen. Die Gefahr, welche durch die Hinzunahme weiterer Nebenbedingungen entsteht, ist die weitere Einschränkung des Parametersuchraums. Das Auffinden einer gültigen Lösung kann dadurch für ein Optimierungsverfahren zu einem nicht-trivialen Problem werden.

Auch die Bildung von Nebenbedingungen aus nur einem *Teil der Zielfunktionen* ist denkbar, um eine Einschränkung der Dimension des Raumes der Entscheidungsvariablen zu erreichen, wodurch es mitunter einem Optimierungsverfahren leichter fällt, das globale Optimum zu finden. Zum anderen existieren weniger Zielfunktionen für die Bildung des skalaren Zielfunktionswertes.

Im Gegenzug der Umwandlung eines MOP in ein SOP kann auch die Umwandlung in umgekehrter Richtung (Umwandlung eines SOP in ein MOP) sinnvoll oder notwendig sein, weil sich zur Lösungsfindung die Anwendung eines mehrkriteriellen Optimierungsverfahrens aus verschiedenen Gesichtspunkten als sinnvoller erweist. Günstig ist dieser Weg, wenn

Hierarchy Process [engl.]) bei der Nutzwertanalyse angewandt (siehe [Köb99, S. 141ff.]).

- die Zusammenfassung zu einem skalaren Zielfunktionswert auf Grund unterschiedlicher Einheiten der Zielfunktionswerte von vornherein schwer fällt,
- keine sinnvolle Umformung der Zielfunktionen in einschränkende Nebenbedingungen möglich ist oder
- eine Menge von Kompromisslösungen angegeben werden soll, aus der die eigentliche Entscheidungsfindung stattfindet[24].

Im Weiteren wird im Rahmen dieser Arbeit davon ausgegangen, dass

1. im Falle der Anwendung eines mehrkriteriellen Optimierungsverfahrens auf ein MOP:
   - einschränkende Nebenbedingungen durch den Anwender per Hand erstellt werden und somit
   - nur die notwendigen Zielfunktionen angegeben sind, welche jeweils mit Gewichten w_k für $f_k(\vec{x})$ versehen werden können und
   - die Gewichtung für $\vec{f}(\vec{x})$ wie folgt vorgenommen wird:

$$\vec{f}(\vec{x}) = \vec{w} \circ \vec{f_A}(\vec{x}), \text{ mit } \dim(\vec{w}) = \dim(\vec{f}(\vec{x})) \text{ und } \vec{w} > \vec{0} \quad ;$$

2. im Falle der Anwendung eines einkriteriellen Optimierungsverfahrens auf ein MOP:
   - sämtliche vorhandene Zielfunktionswerte in einen skalaren, gewichteten Zielfunktionswert gemäß

$$f(\vec{x}) = \sum_{f_{A_i} \in F_{Min}} w_i \cdot f_{A_i} - \sum_{f_{A_j} \in F_{Max}} w_j \cdot f_{A_j}$$

   mit

$$\begin{aligned} F_{Min} &= \{f_{A_i}(\vec{x}), \forall i = 1, 2, \ldots, \dim(f_{A_i}(\vec{x})) \mid f_{A_i}(\vec{x}) \to \min\} \\ F_{Max} &= \{f_{A_i}(\vec{x}), \forall i = 1, 2, \ldots, \dim(f_{A_i}(\vec{x})) \mid f_{A_i}(\vec{x}) \to \max\} \\ w_k &> 0, \quad \forall k \quad , \end{aligned}$$

   wobei

$$w_k \quad \text{-} \quad \text{Gewichte des Zielfunktionswertes von } f_k(\vec{x})$$

   umgewandelt werden können, wobei die w_k vom Nutzer sinnvoll festzulegen sind;

3. im Falle der Anwendung eines einkriteriellen Optimierungsverfahrens auf ein SOP:

[24] Die Bildung der Kompromisslösungen ist zwar auch mit einem einkriteriellen Optimierungsverfahren möglich, indem sich auch die zweitbeste, die drittbeste, etc. Lösung durch das Optimierungsverfahren gemerkt werden, jedoch handelt es sich dabei i. Allg. nicht um eine Kompromissmenge im eigentlichen Sinne.

- keine Transformation erfolgen muss und
- auch eine möglicherweise angegebene Gewichtung der Zielfunktion $f(\vec{x})$ mit w zwar durchgeführt wird, so dass

$$f(\vec{x}) = w \cdot f_A(\vec{x})$$

gelte, was aber vernachlässigt werden kann, weil es für $w \neq 0$ nur eine Skalierung des Zielfunktionswertes bewirkt;

4. im Falle der Anwendung eines mehrkriteriellen Optimierungsverfahrens auf ein SOP:
   - ist diese Form der Anwendung i. Allg. bedeutungslos und nicht sinnvoll.

Speziell die Umwandlung eines MOP in ein SOP, um mittels eines einkriteriellen Optimierungsverfahrens eine optimale Lösung zu bestimmen, birgt Gefahren. Diese entstehen insbesondere durch den Verlust (sub)optimaler Lösungen infolge ungünstiger Gewichtung der Zielfunktionswerte zur Bildung eines gemeinsamen Zielfunktionswertes. In Abb. 2.24 soll dieser Sachverhalt einmal an einem intuitiven Beispiel verdeutlicht sein, um zu zeigen, dass die Probleme bei der Umwandlung durchaus allgemein gültig sind.

2.6. Entscheidungsfindung

Die Entscheidungsfindung ist der simulationsbasierten Optimierung nachgeschaltet und soll der Person des Entscheidungsfinders helfen, eine *Auswahlmöglichkeit bei mehrkriterieller Optimierung*, durch *Reduzierung der Lösungen in der Pareto-optimalen Menge*, zu treffen. Das Problem der Entscheidungsfindung entsteht dadurch, dass es im Falle der Anwendung mehrkriterieller Optimierungsverfahren zur Problemlösung mitunter zu einer hohen Anzahl von Lösungen innerhalb der Pareto-optimalen Menge kommt. Dies macht es einem Entscheidungsfinder im Anschluss an die simulationsbasierte Optimierung schwer, eine für ihn geeignete Lösung mit den entsprechenden Kompromissen aus der Menge der Pareto-optimalen Lösungen auszuwählen. Umgangen werden kann das Problem durch eine abschließende Bewertung dieser Pareto-optimalen Lösungen mit Hilfe der *Angabe von Mindestanforderungen an einzelne gewichtete Zielfunktionswerte*, in Form von absoluten reellen Zahlenwerten oder mittels relativer Prozentangaben zur besten Belegung des Zielfunktionswertes. Diese Form zur Reduzierung der Anzahl der Lösungen innerhalb der Pareto-optimalen Menge sei als *mehrkriterielle Entscheidungsfindung* bezeichnet. In Abb. 2.25 ist ein Beispiel zur mehrkriterielle Entscheidungsfindung abgebildet.

Durch diese mehrkriterielle Entscheidungsfindung werden allerdings Lösungen mit extremen Zielfunktionswerten verworfen, obwohl sie mitunter praktisch die beste Kompromisslösung darstellen können. Günstiger kann daher die Angabe eines Güte-Kriteriums für die einzelnen zusammengefassten, gewichteten Zielfunktions-

Abb. 2.24

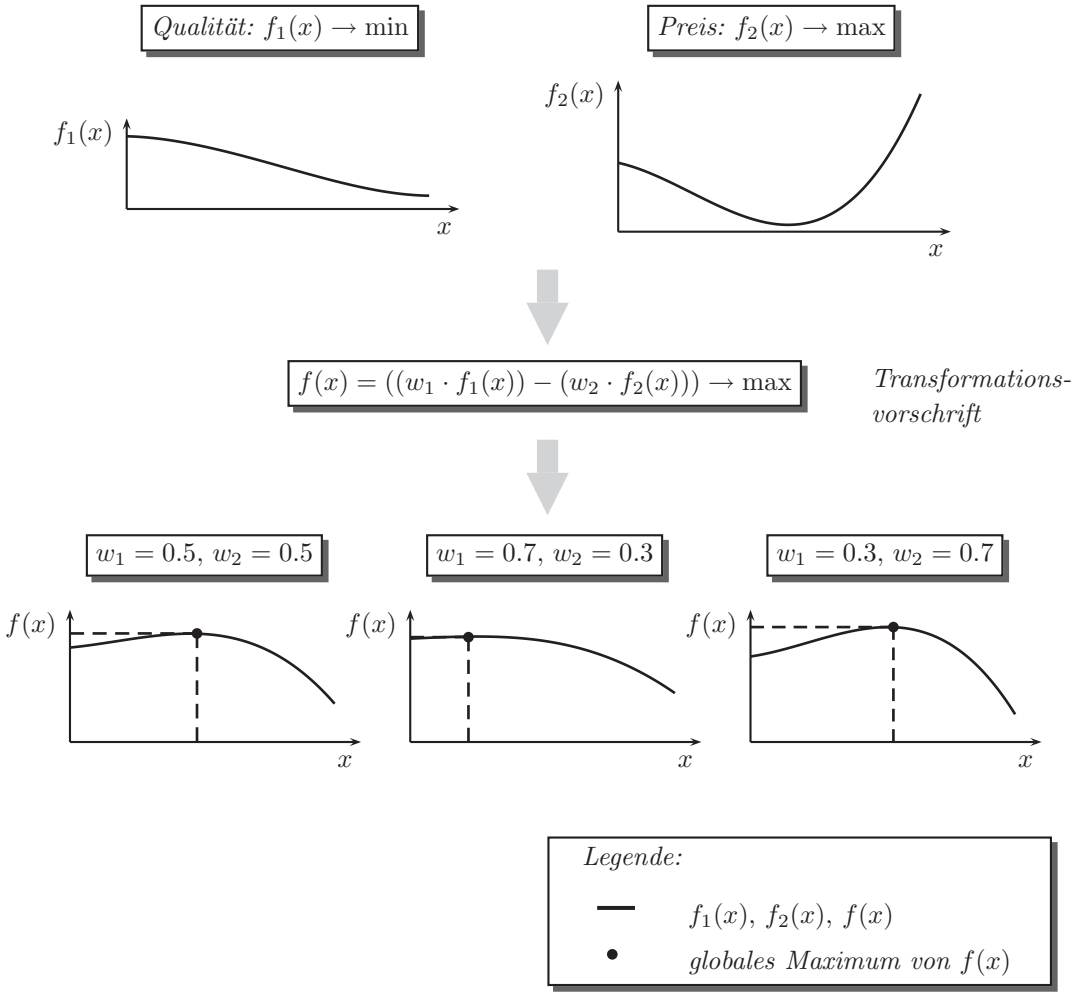


Abb. 2.24.: *Problem der Gewichtung bei der Transformation eines MOP in ein SOP (Beispiel)*

werte sein (*einkriterielle Entscheidungsfindung*). Dabei wird die *Güte jeder potentiellen Lösung* durch ein Maß bewertet, dass der Lösung eine (meist reelle) Zahl zuordnet. Diese Güte wird auch als *Qualität der Lösung* bezeichnet.

Definition 2.16 (Güte einer Lösung):

Def. 2.16

Die Güte einer Lösung kann formal durch Transformation des zugehörigen Zielfunktionswertes bzw. der zugehörigen Zielfunktionswerte in einen geeigneten, vergleichbaren Wert im Sinne des zugrunde liegenden Optimierungsproblems mittels der Funktion $\varphi(\vec{f}(\vec{x}))$ angegeben werden, wobei gilt

$$0 \leq \varphi(\vec{f}(\vec{x})) \leq 1 \quad .$$

□

Um den Unterschied zwischen dem Raum der Zielfunktionen und der Entscheidungsfindung zu verdeutlichen, sei in Abb. 2.26 ein Beispiel dafür angegeben.

Abb. 2.25

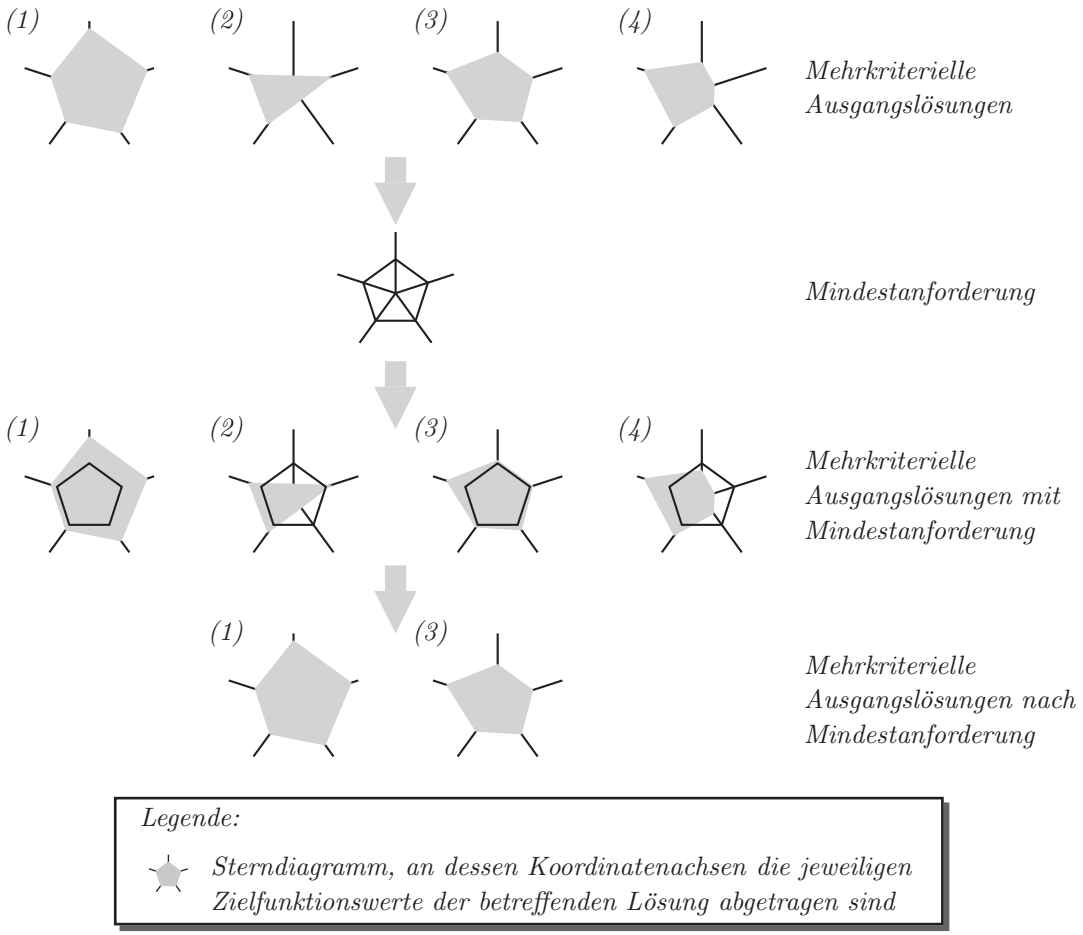


Abb. 2.25.: *Reduzierung mehrkriterieller Lösungen anhand von Mindestanforderungen bei mehrkriterieller Entscheidungsfindung (Beispiel)*

2.7. Zusammenfassung

In diesem Kapitel wurden wichtige Definitionen angegeben, auf welche in den nachfolgenden Kapiteln zurückgegriffen wird. Des Weiteren sind grundsätzliche Aussagen zur Arbeitsweise der modellgestützten, simulationsbasierten Optimierung getroffen worden. In diesem Zusammenhang erfolgte auch die Angabe eines Überblickes über vorhandene Lösungsverfahren zur simulationsbasierten Optimierung. Aus diesen wurde das Verfahren zur iterativen Simulation und Optimierung als Ausgangspunkt für die weiteren Ausführungen dieser Arbeit ausgewählt. Wenn somit nachfolgend von simulationsbasierter Optimierung gesprochen wird, ist stets das Verfahren der iterativen Simulation und Optimierung gemeint. Ansonsten erfolgt eine entsprechende Kennzeichnung.

Im vorliegenden Kapitel wurden zudem die möglichen Strukturen des Zielfunktionsraumes etwas näher betrachtet. Der Zielfunktionsraum wurde dabei aus dem Gesichtspunkt der Anzahl der Zielfunktionen in ein- und mehrkriterielle Zielfunktionsräume untergliedert, was wiederum unmittelbare Auswirkungen auf die resultierenden Bewertungsmöglichkeiten mit sich brachte.

Abb. 2.26

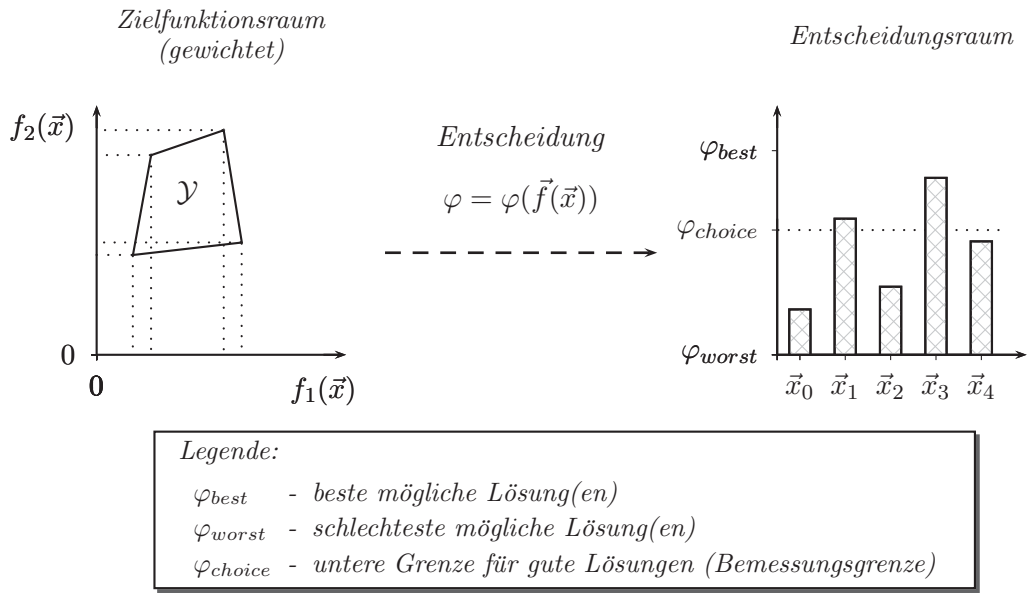


Abb. 2.26.: *Raum der Zielfunktionen und Entscheidungsraum bei einkriterieller Entscheidungsfindung*

Aus der Anwendung des Verfahrens der iterativen Simulation und Optimierung ergab sich direkt die Möglichkeit der Abgrenzung des Simulationsprozesses vom Optimierungsprozess. Dies wiederum ermöglichte die getrennte Betrachtung der angewandten Verfahren bei der Simulation und Optimierung, welche sich unmittelbar in den Kap. 3 und 4 anschließt.

Es wurde dargestellt, dass dem Schritt der Simulation beim Zyklus der iterativen Vorgehensweise zur simulationsbasierten Optimierung der Schritt der Bewertung folgt. Dabei wurde auf die Darstellung von Transformationsmöglichkeiten zur Umwandlung von mehrkriteriellen Zielfunktionswerten in einkriterielle Zielfunktionswerte unter Angabe verschiedener Gewichtungsfaktoren eingegangen. Diese Umwandlungsmöglichkeiten spielen insbesondere bei der Anwendung von einkriteriellen Optimierungsverfahren auf mehrkriterielle Optimierungsprobleme und umgekehrt eine Rolle. Die daraus resultierenden Vor- und Nachteile wurden in diesem Kapitel ebenso angeführt.

Ein weiterer wesentlicher Bestandteil dieses Kapitels war die Aufbereitung eines Problems durch die Möglichkeiten der Modellierung sowie deren rechnerinterne Abbildung. Die dabei angesprochene rechnerinterne Darstellungsform mittels einer Baumstruktur wird an einem speziellen Beispiel im Kap. 5 noch einmal visuell verdeutlicht. Im Zusammenhang mit dem Potential der rechnerinternen Modellierung wurden auch unterschiedliche Sichtweisen auf ein Modell betrachtet und die damit verbundenen Möglichkeiten der Gruppierung verschiedener Modellelemente verdeutlicht.

Abschließend wurden in diesem Kapitel noch die Möglichkeiten der Entscheidungsfindung bei ein- und mehrkriteriellen Optimierungsproblemen betrachtet. Es wurde erklärt, dass die Entscheidungsfindung im Anschluss an einen Lauf der simulations-

basierten Optimierung stattfindet und unmittelbaren Einfluss auf die letztendlich zu wählende Systemkonfiguration hat.

Kapitel 3. Simulation

Inhalt

3.1	*Einführung*	*55*
3.2	*Gründe für die Simulation*	*56*
3.3	*Simulationstypen und Zeitfortschreibung bei der Simulation*	*58*
3.4	*Stochastische Prozesse und Zeitreihen*	*62*
3.5	*Dauer der transienten und der stationären Phase*	*64*
3.6	*Verkürzung der Simulationsdauer bei diskreter Simulation*	*76*
3.7	*Zufallszahlenerzeugung*	*84*
3.8	*Zusammenfassung*	*88*

3.1. Einführung

Die diskrete Simulation wird heutzutage als eine der vorrangigen Methoden zur *Analyse diskreter dynamischer Systeme* eingesetzt (siehe [Man01]). Bei solchen Systemen werden mittels der Simulation *Experimente an einem Modell*, dem *Simulationsmodell* (siehe Abschn. 2.2.2), durchgeführt, um *Erkenntnisse über das zu simulierende (Real-)System* zu gewinnen. Als *Simulator* ist in diesem Zusammenhang die Implementierung des *Simulationsmodells* (Systemstruktur) und des *Simulationsablaufes* (Systemverhalten) auf einem Rechner zu verstehen. Der Ablauf eines Simulators und die damit verbundene Abarbeitung von rechnerinternen Klassenmethoden mit konkreten Werten, auch *Parametrisierung* genannt, wird als *Simulationslauf* bezeichnet. Ein Simulationsexperiment setzt sich wiederum aus mehreren Simulationsläufen zusammen. Die daraus resultierenden Ergebnisse können dann entsprechend validiert, interpretiert und auf das simulierte System übertragen werden. In vereinfachter Weise wird dieses Vorgehen auch in der *Richtlinie VDI 3633* des Vereins Deutscher Ingenieure (kurz: VDI) formuliert (siehe [Ver01]):

> Simulation ist das Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierfähigen Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind. Im weiteren Sinne wird unter Simulation das Vorbereiten, Durchführen und Auswerten gezielter Experimente mit einem Simulationsmodell verstanden.

Im Rahmen dieser Arbeit wird die Simulation zur Untersuchung verschiedener Modelle für allgemein gültige Produktions- und Lagerhaltungssysteme genutzt. Bei diesen Systemen wird davon ausgegangen, dass die Zielfunktion die Minimierung der anfallenden Kosten pro Zeiteinheit beinhaltet. Zusätzlich weist das zugehörige Simulationsmodell verschiedene stochastische Einflussfaktoren auf, deren Betrachtung zusammen mit den Umrüstkosten und -zeiten eine analytische Lösung des Optimierungsproblems verhindert. Die Werte der Entscheidungsvariablen dienen dem Simulationsmodell als Vorgaben für die Parameter der verwendeten Bestellstrategien.

Im Allgemeinen liegt demnach die Aufgabe der Simulation bei der innerhalb dieser Arbeit angewandten wechselweisen Simulation und Optimierung in der Durchführung eines Simulationsexperimentes, um Daten für die Bestimmung des Zielfunktionswertes oder der Zielfunktionswerte zu erhalten. Das Optimierungsverfahren hingegen stellt systematisch neue Werte für die Entscheidungsvariablen, d. h. für die veränderlichen Eingabegrößen der Simulation während eines Laufes der simulationsbasierten Optimierung bereit, um eine Verbesserung des gewünschten Zielfunktionswertes oder der gewünschten Zielfunktionswerte zu erreichen.

Mit der Simulation selbst sind jedoch noch weitere Problemstellungen, wie die Betrachtung der transienten und stationären Phase sowie die Möglichkeit der Verkürzung der Simulationsdauer verbunden. Diese Problemstellungen werden anhand ausgewählter Verfahren innerhalb dieses Kapitels etwas genauer betrachtet. Es wird zudem eine mögliche Heuristik vorgestellt, welche im Rahmen der simulationsbasierten Optimierung ihre Anwendung findet und auf die Verkürzung eines Simulationslaufes abzielt. Des Weiteren wird eine kritische Betrachtung über die Anzahl der durchzuführenden Simulationsläufe innerhalb eines Simulationsexperimentes durchgeführt.

3.2. Gründe für die Simulation

Prinzipiell wird ein rechnerintern dargestelltes Simulationsmodell dazu genutzt, um mit Hilfe der *Simulation* verschiedenste Untersuchungen zur Erkenntnisgewinnung über das Verhalten eines Realsystems durchführen zu können. Hierbei ist die Simulation insbesondere sinnvoll, wenn

- aus *Sicherheits-* oder *Gefahrengründen* das Realsystem nicht oder nur schwer zugänglich (z. B. bei der Untersuchung von Systemen mit gefährlichen Chemikalien oder atomaren Stoffen),
- aus *ökonomischen Gründen* die Untersuchung verschiedener Prozessabläufe im Vorfeld zu kostenintensiv (z. B. bei der Umstellung der Produktionsstrategie eines Produktionssystems oder der Lagerstrategie eines Lagers),
- sich aus *Verfügbarkeitsgründen* das Realsystem nicht direkt untersuchen lässt, d. h. das System ist noch nicht vorhanden (z. B. bei der Errichtung einer neuen zusätzlichen Produktionshalle) oder

- aus *zeitlichen Gründen* eine Untersuchung des Realsystems über einen längeren Zeitraum unmöglich (z. B. bei der mittel- oder langfristige Auswirkung der Einführung eines neuen Produktes)

ist. Die innerhalb dieser Arbeit entscheidenden Vorteile der Simulation ergeben sich dabei u. a. durch

- die Möglichkeit zur *beliebigen Wiederholung eines Simulationsexperimentes*,
- die *beliebige Komplexität eines Simulationsmodells*,
- die *separate Untersuchung einzelner Simulationsmodellvariablen* und
- die Möglichkeit der *Veränderung des zeitlichen Ablaufes* (Zeitraffer- oder Zeitlupeneffekt).

Außerdem ist die Simulation bzw. die simulationsbasierte Optimierung dann eine Alternative, wenn die *analytische Lösung* eines Optimierungsproblems *nicht oder schwer möglich* ist, wie es bei $\mathcal{NP}$*-schweren* oder $\mathcal{NP}$*-harten Optimierungsproblemstellungen* der Fall ist (siehe [Fu01b] oder [Fu01a]), oder ein (abstrahiertes) *Realsystem zur Optimierung* nachgebildet werden soll. Diese beiden Gründe sind auch für die im Abschn. 6.4 untersuchten Modelle ausschlaggebend. Diese verschiedenen Modelle zu Produktions- und Lagerhaltungssystemen sind aus dem *kapazitierten stochastischen Losgrößenproblem* [dt.; *Capacitated Stochastic Lot Sizing Problem* [engl.], kurz: *CSLSP*] (vgl. Abschn. 6.2 und 6.3) hervorgegangen. Es ist denkbar, dass das CSLSP an sich unter bestimmten Voraussetzungen, wie bspw. die Verwendung der Mittelwerte bei Produktionszeiten und identischen Zeiten für Umrüstungen, selbst im Mehrproduktfall, durchaus analytisch lösbar ist. Finden jedoch unterschiedliche produktabhängige, ggf. stochastische Produktions- und Umrüstzeiten Verwendung, entsteht ein $\mathcal{NP}$-schweres Problem, welches u. a. mittels der innerhalb dieser Arbeit genutzten simulationsbasierten Optimierung gelöst werden kann. Hierbei erweist sich der Einsatz der Simulation als Hilfsmittel für die Lösung der untersuchten Probleme als hilfreicher Ausweg, um trotz stochastischer Einflussgrößen möglichst genaue Ergebniswerte zu erhalten. Diese sind wiederum Voraussetzung für eine erfolgreiche Optimierung.

Dass sich bei der exakten Bestimmung der Lösungswerte für verschiedene Modelle Probleme mit analytischen Lösungsmethoden ergeben, ist bedingt durch

- die *große Ungenauigkeit* infolge fehlender Behandlung von Sonderfällen oder speziellen Eigenschaften (z. B. externe Faktoren und Wechselwirkungen),
- die teilweise oder ganz *fehlenden mathematischen Gleichungen und Ungleichungen* aufgrund nicht (detailliert) bekannter Zusammenhänge (z. B. bei Prozessen mit gegenseitigen Blockierungen),
- die *Auswirkungen vielfältiger zufälliger Einflüsse* bei stochastischen Systemen, die nur in speziellen Fällen analytisch berechenbar sind (z. B. exponentialverteilte Zufallsvariablen) und
- das gleichzeitige *Auftreten verschiedenartiger zufälliger Prozesse*.

Dennoch existieren aber auch Methodiken für die Analyse eines $\mathcal{NP}$-schweren Problems, wie bspw.

- den Versuch einer *Verallgemeinerung des untersuchten Modells* durchzuführen sowie
- die Verwendung von *gemittelten oder geschätzten Eingabewerten*, um eine genaue Berechnung der Ergebniswerte anhand dieser Schätzungen zu ermöglichen.

Die zuvor genannten Methoden führen jedoch i. Allg. zu weniger exakten Ergebnissen, als sie durch ein Simulationsexperiment erhalten werden könnten und sollten nach Meinung des Autors daher nur in ausgewählten Problemfällen angewandt werden. Diese zuvor erwähnten beiden Vorgehensweisen werden innerhalb der vorliegenden Arbeit nicht genutzt. Demgegenüber steht allerdings der höhere zeitliche Aufwand bei der Durchführung eines Simulationsexperimentes.

Grenzen der Simulation. Eine Simulation arbeitet nach dem *GIGO-Prinzip* (Abkürzung für *Garbage-In-Garbage-Out* [engl.]). Je genauer die Eingangsdaten, umso genauer sind auch die Ergebnisse. Problematisch sind mitunter die *Definition von Störgrößen* (z. B. Ausfall von Maschinen oder Personal) und die *stark schwankenden Verteilungswerte bei den Parametern eines Simulationsmodellelementes* (z. B. Ziele von Privatfahrzeugen im Straßenverkehr). Eine statische Betrachtung des Einflusses solcher Größen wird unmöglich. Die Simulation kann bei Aufzeichnung der Minimal- und Maximalwerte der Ausgabewerte einzelner Simulationsläufe jedoch Konfidenzintervalle für die ermittelten Ergebnisse zurückliefern. Diese sollten für verschiedene statistische Auswertungen der Simulationsexperimente herangezogen werden.

3.3. Simulationstypen und Zeitfortschreibung bei der Simulation

Simulationstypen. Zur Vollständigkeit der getroffenen Ausführungen seien in Tab. 3.1 einige Typen von Simulationen bei Anwendung unterschiedlicher Sichtweisen erwähnt, um die verschiedenen Probleme bei der Erstellung eines Simulationsmodells zu verdeutlichen. Näheres dazu ist in den angegebenen Literaturverweisen zu finden, so dass an dieser Stelle keine weiteren Aussagen zu bestimmten Typen von Simulationen gemacht werden. Die Tab. 3.1 erhebt keinen Anspruch auf Vollständigkeit.

Eine der am häufigsten verwendete Form der Simulation für ökonomische Bereiche bzw. Probleme des Operations Research [engl.; Operationsforschung [dt.]] ist, nach Meinung des Autors, heutzutage die *diskrete Simulation*. Dies lässt sich u. a. auf die Einfachheit bei der Implementierung eines Simulators zur diskreten Simulation zurückführen. Die diskrete Simulation wird auch bei den zu dieser Arbeit vorliegenden Simulatoren zu Produktions- und Lagerhaltungssystemen genutzt. In

Tab. 3.1

Sichtweise	Beispiele	Literaturverweis(e)
Zeitlich	• *Kontinuierliche bzw. stetige Simulation* • *Diskrete Simulation (zeit- oder ereignisgesteuert)* • *Hybride Simulation*	[BKPR03], [ABGR03], [Khe88]
Komplexität	• *Makrosimulation* (Weltmodelle) • *Mikrosimulation* (Detailmodelle)	[Ban98], [GT05a]
Mathematisch	• *Statistische und probabilistische Simulation* • *Numerische Simulation* • *Deterministische Simulation*	[SFW94], [BS02b], [Koh72]

Tab. 3.1.: *Ausgewählte Simulationstypen für verschiedene Sichtweisen*

diesem Falle war, neben der Einfachheit der Implementierung der zugehörigen Simulatoren, die bereits vorhandene Diskretisierung des untersuchten Systems durch diskrete Produktionsabläufe ausschlaggebend (siehe Abschn. 6.2).

Zeitfortschreibung bei diskreter Simulation. Bei der diskreten Simulation erfolgt die *Zeitfortschreibung zu diskreten Zeitpunkten*, wobei zwischen *zeitgesteuerter* bzw. *(diskret) schrittweiser* und *ereignisgesteuerter* bzw. *(diskret) ereignisorientierter Simulation* unterschieden wird (siehe [ZPK00] und Abb. 3.1).

Durch die Festlegung bestimmter Zeitintervalle, nach deren Ablauf eine erneute Betrachtung des Simulationsmodells und dessen Veränderungen erfolgt, kommt es zur Zeitfortschreibung bei der zeitgesteuerten Simulation. Ist bspw. ein System nur zu bestimmten Zeitpunkten verfügbar oder eine häufige Überwachung des Systems notwendig, um gezielt reagieren zu können, empfiehlt sich der Einsatz einer zeitgesteuerten Simulation. Bei der ereignisgesteuerten Simulation hingegen ist die Betrachtung der Änderungen eines Simulationsmodells an das Eintreten vorgegebener, festgelegter Simulationsereignisse gekoppelt. Die Zeitfortschreibung wird demnach durch konkrete Ereignisse vorangetrieben. In der vorliegenden Arbeit stehen durch die Beschaffenheit der untersuchten Produktions- und Lagerhaltungssysteme das Eintreten konkreter diskreter Simulationsereignisse sowie die damit verbundenen systeminternen Abläufe und Veränderungen im Vordergrund. Es wird daher von einer zeitgesteuerten Simulation abgesehen und die ereignisgesteuerte Simulation kommt zum Einsatz[25].

[25] Ein Vorteil der ereignisgesteuerten Simulation ist zudem die Einfachheit des Simulators. Jedem Simulationsereignis des Simulationsmodells wird eine Ereignisroutine innerhalb des Simulators zugeordnet. In der zumeist einfach zu gestaltenden Ereignisroutine sind nur die mit diesem Simulationsereignis verbundenen Tests und Aktionen auszuführen. Der Simulator wird von einem Zustand in einen anderen Zustand überführt, ohne Zwischenzustände annehmen zu müssen, welche weitere Aktionen nach sich ziehen können. Im Gegensatz dazu existieren auch aktivitäten- und prozessorientierte Simulationszugänge (siehe [Köc07a, Kap. 3]), bei denen es auf Grund deren Charakteristik (unterschiedliche Gruppierung von mehreren Simulationsereignissen) durchaus vorkommt, dass mehrere Zwischenzustände abgearbeitet werden müssen.

Abb. 3.1

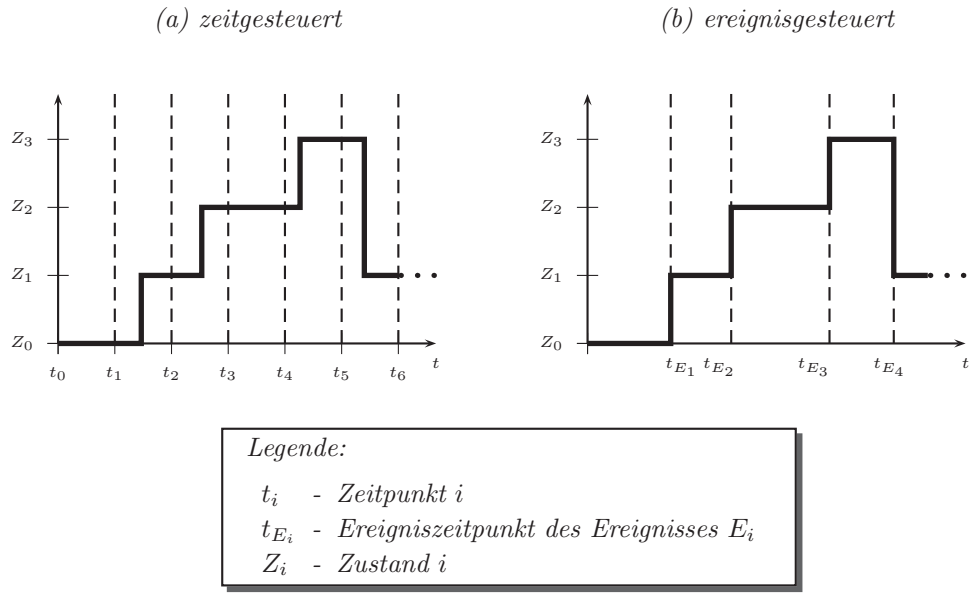


Abb. 3.1.: *Zeitfortschreibung bei diskreter Simulation*

Die ereignisgesteuerte Simulation zeichnet sich u. a. durch folgende Charakteristiken aus:

- *breite Anwendbarkeit*,
- *kaum Einschränkungen bzgl. Beschreibungskonstrukten*,
- *kaum Einschränkungen bzgl. Charakterisierung der Zeit*,
- Möglichkeit der *Untersuchung offener Systeme mit potentiell unendlichen Zustandsräumen*.

Schwierigkeiten bei der ereignisgesteuerten Simulation ergeben sich zumeist bei der *Verwendung seltener Simulationsereignisse* (siehe [GF98]) oder *unterschiedlicher Zeitskalen* (siehe [MO07]). Diese Probleme spielen jedoch bei den vorgenommenen Untersuchungen keine Rolle, weil sie in diesen Modellen nicht existent sind.

Ablauf eines Simulationslaufes bei diskreter ereignisorientierter Simulation. Zur Steuerung eines Simulationslaufes bestehen Simulatoren zur diskreten Simulation im Wesentlichen aus 3 Detaillierungsebenen:

- Die *Ebene 1* besteht aus der *Exekutive*, welche den eigentlichen Ablauf eines Simulationsmodells steuert.
- Die *Ebene 2* beschreibt die *Operationen*, welche innerhalb des Simulationsmodells prinzipiell ablaufen.
- Die *Ebene 3* stellt die *detaillierten Routinen* bereit, welche von den Operationen der Ebene 2 genutzt werden.

Anzumerken sei an dieser Stelle, dass die UML in der Version 2.0, im Speziellen die Aktivitäts- und Sequenzdiagramme, auch als Modellierungsformalismus für

diskrete ereignisorientierte Simulation genutzt werden können. Dies geschieht auch innerhalb der vorliegenden Arbeit (siehe bspw. Abb. 3.2).

Die drei Ebenen bestehen unabhängig vom gewählten Simulationszugang, wobei nachfolgend vom ereignisorientierten Zugang ausgegangen wird (s. o.). In Abb. 3.2 sei zunächst die Ebene 1, d. h. die Exekutive etwas genauer betrachtet. Wie erkennbar wird, hat die Exekutive beim ereignisorientierten Zugang die Aufgaben der Zeitfortschreibung sowie der Ereignisidentifikation und -ausführung zu erledigen. Die Abb. 3.2 repräsentiert jedoch nur eine Möglichkeit zur Umsetzung einer allgemein gültigen Exekutive. Das Detaillierungssymbol „⏃“ verdeutlicht dabei die verschiedenen speziellen Operationen (Ereignisroutinen) der Ebene 2.

Abb. 3.2

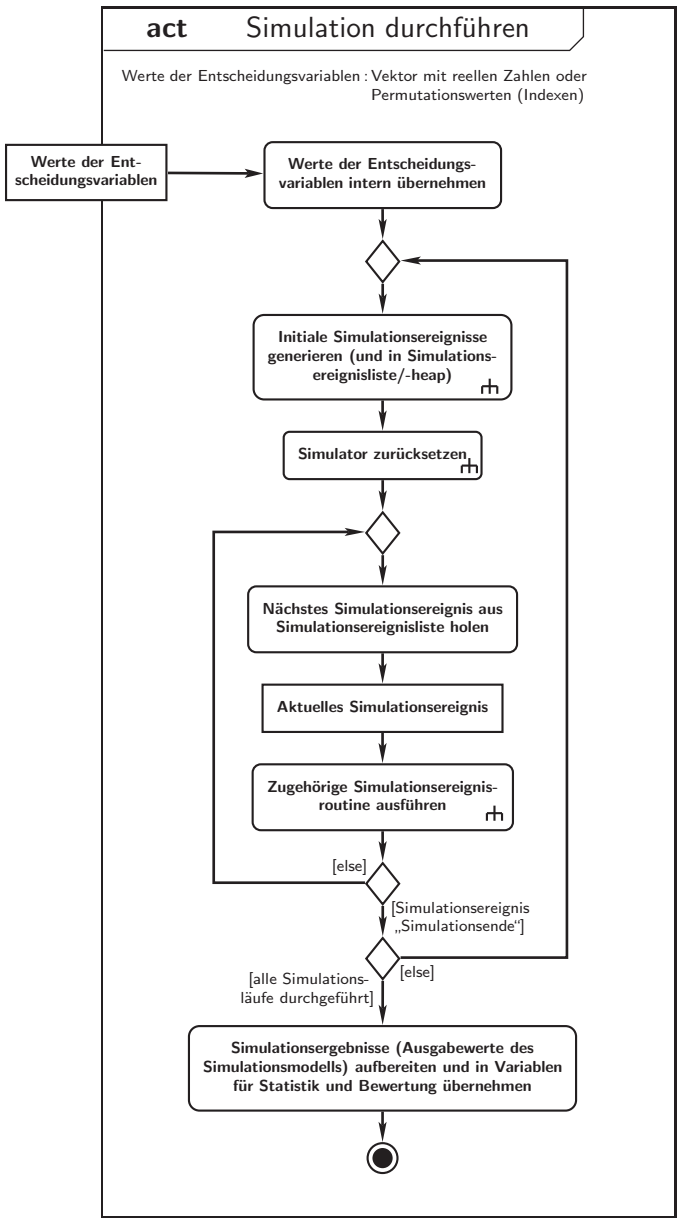


Abb. 3.2.: *Algorithmus zur Durchführung einer diskret ereignisorientierten Simulation (UML-Aktivitätsdiagramm)*

3.4. Stochastische Prozesse und Zeitreihen

Nachdem in den vorangehenden Abschnitten verschiedene formale Grundlagen sowie der Ablauf eines Simulationslaufes erklärt wurden, sollen in diesem Abschnitt einige theoretische Betrachtungen vorgenommen werden. Diese sind notwendig, um verschiedene Problematiken zu verdeutlichen, welche in direktem Zusammenhang mit der Auswertung der Ergebnisse eines Simulationsmodells stehen. Im Wesentlichen sind dies Definitionen, welche sich mit dem Problem der stochastischen Einflüsse und Abhängigkeiten bei der Simulation beschäftigen. Im Abschn. 3.5 kommt als weitere Problematik noch die Bestimmung der Dauer der transienten und der stationären Phase hinzu.

Zufallsgrößen und stochastische Prozesse. Ein Simulationsmodell kann aus verschiedenen Zufallsgrößen X_i bestehen, wobei jede Zufallsgröße eine Abbildung

$$X_i : \Omega \to \mathbb{R}$$

ist. Diese Abbildung ordnet jedem Elementarereignis $\omega \in \Omega$ eine reelle Zahl zu.

Während eines Simulationslaufes nimmt ein Simulationsmodell verschiedene Zustände an, wobei der nächste Zustand zufällig bestimmt wird. Bei wiederholter Beobachtung des Simulationsmodelles werden auf Grund des Zufallsfaktors unterschiedliche Ergebnisse erzielt. Ein Simulationsmodell beschreibt somit einen Vorgang der i. Allg. als stochastischer Prozess $f(t, \omega)$ bezeichnet wird.

Def. 3.1

Definition 3.1 (Stochastischer Prozess):
Eine Abbildung

$$f(t, \omega) : I \times \Omega \to \mathbb{R} \quad ,$$

welche für jeden festen (nicht zufälligen) Parameterwert $t = t_1 \in I$ eine Zufallsgröße $f(t_1, \omega) \equiv f_{t_1}(\omega) \equiv f_{t_1}$ und jedes feste Elementarereignis $\omega \in \Omega$ eine reelle Funktion $f_\omega(t) \equiv f(t)$ darstellt, wird als stochastischer Prozess bezeichnet. □

I heißt Indexmenge. Ist $I = \mathbb{N}$, dann ist der stochastische Prozess zeitdiskret und im Falle von $I = \mathbb{R}$ zeitkontinuierlich. Jede nicht zufällige Funktion $f(t)$ heißt *Realisierung eines stochastischen Prozesses*. Diese entsteht bspw. im Rahmen der Durchführung eines Simulationsexperimentes. Darüber hinaus werden Realisierungen von stochastischen Prozessen mit diskreter Zeit als *Zeitreihen* bezeichnet. Eine Zeitreihe ist somit eine zeitlich geordnete Folge X_t mit $t \in T$ von Beobachtungen der Zufallsvariablen X zu den Zeitpunkten t. Für jeden Zeitpunkt t der Menge T, welche die einzelnen Beobachtungszeitpunkte enthält, liegt dabei genau eine *Beobachtung* vor. Zu diesem Zeitpunkt hat X einen konkreten reellen Wert angenommen.

Visualisierung von Zeitreihen. Zur Visualisierung von Zeitreihen u. a. infolge stochastischer Prozesse können so genannte *Plots* verwendet werden, welche die einzelnen Beobachtungen über die Zeit wiedergeben. In Abb. 3.3 seien bspw. die Ankunftszeitpunkte eines Kunden (Abb. 3.3 (a)) als eindimensionaler Plot und die Anzahl der Kunden in einem modellierten System (Abb. 3.3 (b)) als zweidimensionaler Plot betrachtet.

(a) Ankunftszeitpunkt eines Kunden *(b) Anzahl der Kunden im Modell* Abb. 3.3

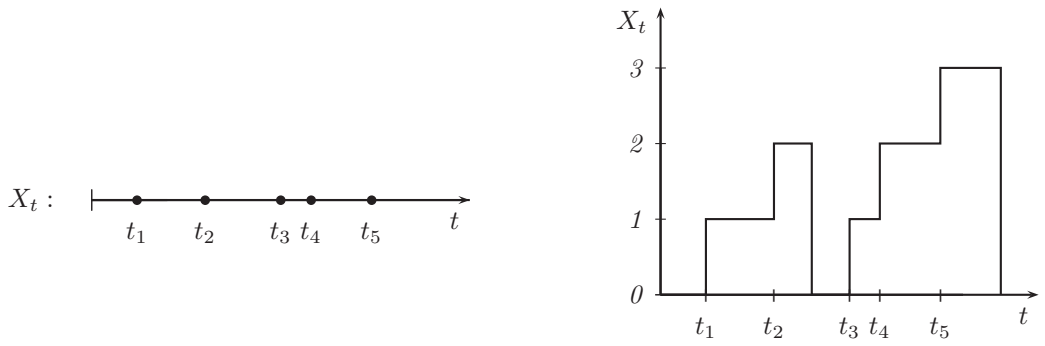


Abb. 3.3.: *Visualisierung von Zeitreihen mittels Plots (Beispiel)*

Stationärität. Die Stationärität ist eine wichtige Eigenschaft von stochastischen Prozessen. Sie gibt die Unabhängigkeit der Verteilung $F_t(x|X_0) = P(X_t \leq x|X_0)$ von dessen Index t an, wobei X_0 den Startzustand der Zeitreihe X_t repräsentiert. Für die Simulation sind insbesondere die starke und die schwache Stationärität von Bedeutung, welche wie folgt definiert sind.

Definition 3.2 (Starke Stationärität): Def. 3.2
Ein stochastischer Prozess $f(t, \omega)$ heißt streng bzw. stark stationär, wenn dessen endlichdimensionalen Verteilungen translationsinvariant sind. Für die eindimensionale Verteilung $F(t, \omega)$ gilt bspw.

$$F(t, \omega) = F(t + \tau, \omega) \quad ,$$

d. h. es besteht eine Unabhängigkeit der Verteilung vom Index $\tau \in I$. □

Definition 3.3 (Schwache Stationärität): Def. 3.3
Ein stochastischer Prozess $f(t, \omega)$ heißt schwach stationär, wenn gilt

1. $\mu(t) = m$, $\forall t \in T$, *mit* $\mu(t) = E[f_t(\omega)]$ *und*
2. $cov(t_1, t_2) = cov(t_1 + \tau, t_2 + \tau) = f(t_2 - t_1)$, $\forall t_1, t_2, \tau \in T$, *mit* $cov(t_1, t_2) = E[(f_{t_1} - \mu(t_1))(f_{t_2} - \mu(t_2))] = E[f_{t_1} \cdot f_{t_2}]$.

woraus sich auch

$$\sigma^2(t) = \sigma^2 < \infty, \ \forall t \in T, \ \text{mit } \sigma^2(t) = Var^2[f_t(\omega)] = E[(f_t(\omega) - \mu(t))^2]$$

ergibt. □

Die Funktionen $\mu(t)$, $\sigma^2(t)$ und $cov(t_1, t_2)$ haben dabei folgende Bedeutung:

$\mu(t)$	-	Erwartungswertfunktion des Prozesses $f(t, \omega)$,
$\sigma^2(t)$	-	Varianzfunktion des Prozesses $f(t, \omega)$, Abweichung der Realisierungen vom mittleren Verlauf $\mu(t)$ sowie
$cov(t_1, t_2)$	-	Kovarianzfunktion des Prozesses $f(t, \omega)$, Maßzahl für die gegenseitige Beeinflussung der Zufallsgrößen f_{t_1} und f_{t_2} .

Näheres zu den Funktionen, deren Bedeutungen und deren Zusammenhänge ist u. a. in [Sch92] zu finden.

3.5. Dauer der transienten und der stationären Phase

Im Nachfolgenden wird von nicht-terminierender Simulation ausgegangen, wobei die untersuchten Modelle ihr stationäres Verhalten unabhängig vom Anfangszustand (s. u.) erreichen werden[26]. Voraussetzung für diese Beobachtung ist eine hinreichend lange Simulationsdauer, um das Verhalten des Modells unter Berücksichtigung eines unendlichen Zeithorizontes möglichst genau nachzubilden. Diese Art von Simulation wird auch als steady-state-Simulation [engl.] bezeichnet, wobei u. a. die bestimmten Konfidenzintervalle verschiedene Aussagen über die statistische Sicherheit der Ausgabewerte eines Simulationsmodells zulassen. Steady-state-Simulationen sind demnach Simulationen, welche einen unendlichen Zeithorizont besitzen[27], wie bspw. die Simulation von Fertigungsprozessen über eine unbestimmte Zeitdauer hinweg. Nicht-terminierend bedeutet, dass sie sich das Modell nach endlicher Zeit bzw. endlich vielen Berechnungsschritten nicht in einem definierten Zustand, dem Endzustand, befindet[28]. Steady-State-Simulationen bzw. nicht-terminierende, stationäre Simulationen durchlaufen stets eine transiente Phase und eine stationäre Phase (s. u.).

Phasen eines Simulationslaufes. Aus Simulationssicht ist nur die starke Stationärität von Bedeutung, weil sie die Unabhängigkeit einer Verteilung von dessen Index eindeutig charakterisiert. Diese ist jedoch auf Grund des hohen Zeitaufwandes bei der Bestimmung der Verteilung $F_t(x|X_0)$ praktisch kaum überprüfbar, wobei t die aktuelle Simulationszeit und X_0 den Startzustand der Zeitreihe X_t wiedergibt (s. o.). Es wird daher im Folgenden von schwacher Stationärität ausgegangen, wenn von Stationärität die Rede ist. Bei der Simulation wird zudem

[26] Mögliche andere Verhaltensweisen bei nicht-terminierender Simulation wären eine zyklische Wiederholungen des Modellzustandes, eine (stetig) wachsende Veränderung des Modellzustandes oder eine chaotische Veränderung des Modellzustandes.

[27] Im Gegensatz dazu existiert bei der terminierenden Simulation eine zeitliche Begrenzung, in dem Bedingungen vorgegeben werden, welche den Startzustand und den Endzustand beschreiben. Ein Beispiel ist die Erstellung eines Hauses, welche mit dem Zeitpunkt der Fertigstellung endet.

[28] Die gleiche Problemstellung der Terminierung einer Berechnung tritt beim Halteproblem von Automaten auf (siehe [AB02]).

die Zufallsverteilung eines bestimmten Modellzeitpunktes stärker betont. Nähert sich das Modellverhalten mit der Zeit[29] einer stationären Zustandsverteilung[30] an, wird das Simulationsmodell als stationär bezeichnet (siehe [Jai91, Kap. 24.3]). In diesem Falle gilt $F_t(x|X_0) \to F(x)$ mit $t \to \infty$ und jedem beliebigen Startzustand X_0. Dies wiederum ist Voraussetzung für den Beginn der stationären Phase, weil der zuvor vorhandene Einfluss des Startzustandes abgeklungen ist und keine Auswirkungen mehr auf die Ausgabewerte des Simulationslaufes besitzt. Ziel muss es daher sein, mittels geeigneter Methoden den Zeitpunkt zu bestimmen, ab dem die stationäre Phase erreicht ist.

Die *transiente Phase* oder *Einschwingphase* (*transient phase* [engl.] oder *warm-up phase* [engl.]) dient demnach dazu, ein sich zuvor im Startzustand befindliches Simulationsmodell mit potentiell unendlichem Zustandsraum in sein definiertes „typisches“ Verhalten, das stationäre Verhalten, zu überführen. Es wird auch davon gesprochen, dass ab dem Zeitpunkt des Vorliegens des definierten Modellverhaltens ein stationäres Regime herrscht. Ausgehend von diesem stationären Verhalten kann in der sich anschließenden *stationäre Phase* oder *Phase der Datensammlung* mit der Sammlung von relevanten Daten über den Simulationslauf begonnen werden. Die beiden Phasen eines Simulationsexperimentes sind in Abb. 3.4 dargestellt und werden nachfolgend etwas näher betrachtet.

Abb. 3.4

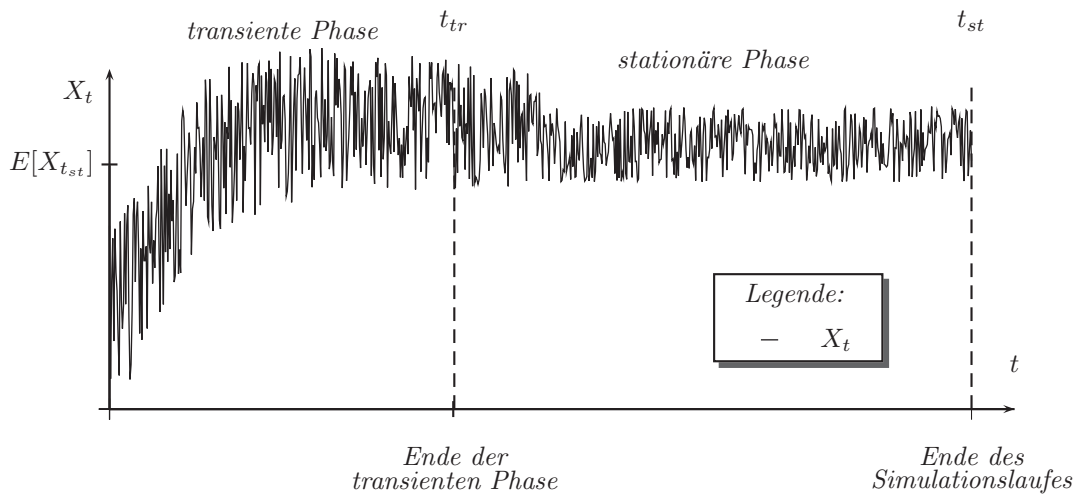


Abb. 3.4.: *Phasen eines Simulationsexperimentes an einem Beispiel*

Transiente Phase. Während der transienten Phase, $t \leq t_{tr}$, weist ein Simulationsmodell ein „untypisches“ Systemverhaltens auf. Es unterliegt mitunter noch (sehr) starken Schwankungen. Das Erkennen des Endes der transienten Phase zum Zeitpunkt t_{tr}, d. h. die *Dauer der transienten Phase* (*transient time* [engl.] oder *warm-up interval* [engl.]), hat somit entscheidenden Einfluss auf die Exaktheit der

[29] Bei der ereignisgesteuerten Simulation wird die Zeitfortscheibung durch die zeitlich zunehmenden Zeitpunkte der Simulationsereignisse vorangetrieben.
[30] Die stationäre Zustandsverteilung beschreibt das Verhalten der Zeitreihe bei $t \to \infty$.

Ergebnisse eines Simulationslaufes und auch direkt auf die Dauer eines Simulationslaufes. Wird sie zu kurz gewählt oder bestimmt, ist das untersuchte Simulationssystem möglicherweise noch nicht eingeschwungen. Es hat das stationäre Verhalten noch nicht erreicht und der Einfluss der transienten Phase ist nach wie vor vorhanden, so dass bspw. die in einem Simulationsmodell entstehenden Ausgabewerte[31] zu hoch oder zu niedrig ausfallen und somit zu ungenaue Werte repräsentieren[32] (siehe Abb. 3.5 (a) und (b)). Eine zu lange Einschwingdauer hat zur Folge, dass die Simulationsdauer insgesamt beeinträchtigt wird. Es wird länger simuliert als notwendig wäre (siehe Abb. 3.5 (d)), so dass mitunter kostbare Rechenzeit verloren geht und ein Lauf der simulationsbasierten Optimierung länger dauert als erforderlich. In Abb. 3.5 (c) ist letztendlich ein Kompromiss von Rechenzeit und Simulationsdauer angegeben.

Zur Entgegenwirkung der Risiken einer zu langen oder zu kurzen transienten Phase wurden verschiedene Methoden zur Schätzung der Länge der Dauer der transienten Phase bzw. heuristische Regeln zur Bestimmung des Zeitpunktes für den Übergang vom transienten zum stationären Verhalten entwickelt und untersucht. Es gibt im Wesentlichen zwei Kategorien:

- *dynamische Methoden* und
- *statische Methoden.*

Dynamische Methoden zum Erkennen des Endes der transienten Phase. Dynamische Methoden oder Regeln bestimmen die Länge der transienten Phase während eines Simulationslaufes. Es ist somit sehr wahrscheinlich, dass weder die Dauer der transienten Phase noch die Dauer des gesamten Simulationslaufes im Vorfeld des Simulationslaufes vorhergesagt werden kann. Es existieren verschiedenste Regeln für die dynamische Bestimmung der Dauer der transienten Phase. In Tab. 3.2 ist ein Auszug dieser Regeln unter Angabe ihrer Nachteile und entsprechender weiterreichender Literaturverweise angegeben.

In [Eic02, Kap. 6] stellt Eickhoff an Untersuchungen eines einfachen $M/M/1$- bzw. $M/M/1/\infty/\infty$-Server-Modells (siehe [BINN01, S. 211f.] und [Köc07b]) fest, dass keine der Regeln für die Bestimmung des Übergangs von der transienten in die stationäre Phase diesen Zeitpunkt t_{tr} eines Simulationslaufes so hinreichend exakt bestimmt, dass die Regel auch auf andere Simulationsmodelle übertragbar ist und ähnlich gute Resultate erzielt. Dies lässt den Schluss zu, dass entweder

- stets *mehrere Regeln* zur Untersuchung des Endes der transienten Phase genutzt,
- für jedes Simulationsmodell die *zu verwendende Regel individuell anzugeben* ist oder
- *andere Methoden* entwickelt und untersucht werden sollten.

Statische Methoden zum Erkennen des Endes der transienten Phase. Bei den statischen Methoden wird zu einem festen Simulationszeitpunkt gesagt, dass

[31] In Abb. 3.5 sind dies Mittelwert und Varianz.
[32] Ein Ausweg würde bspw. darin bestehen, die Simulationsdauer entscheidend zu verlängern, so dass der Einfluss abgeschwächt wird.

Abb. 3.5

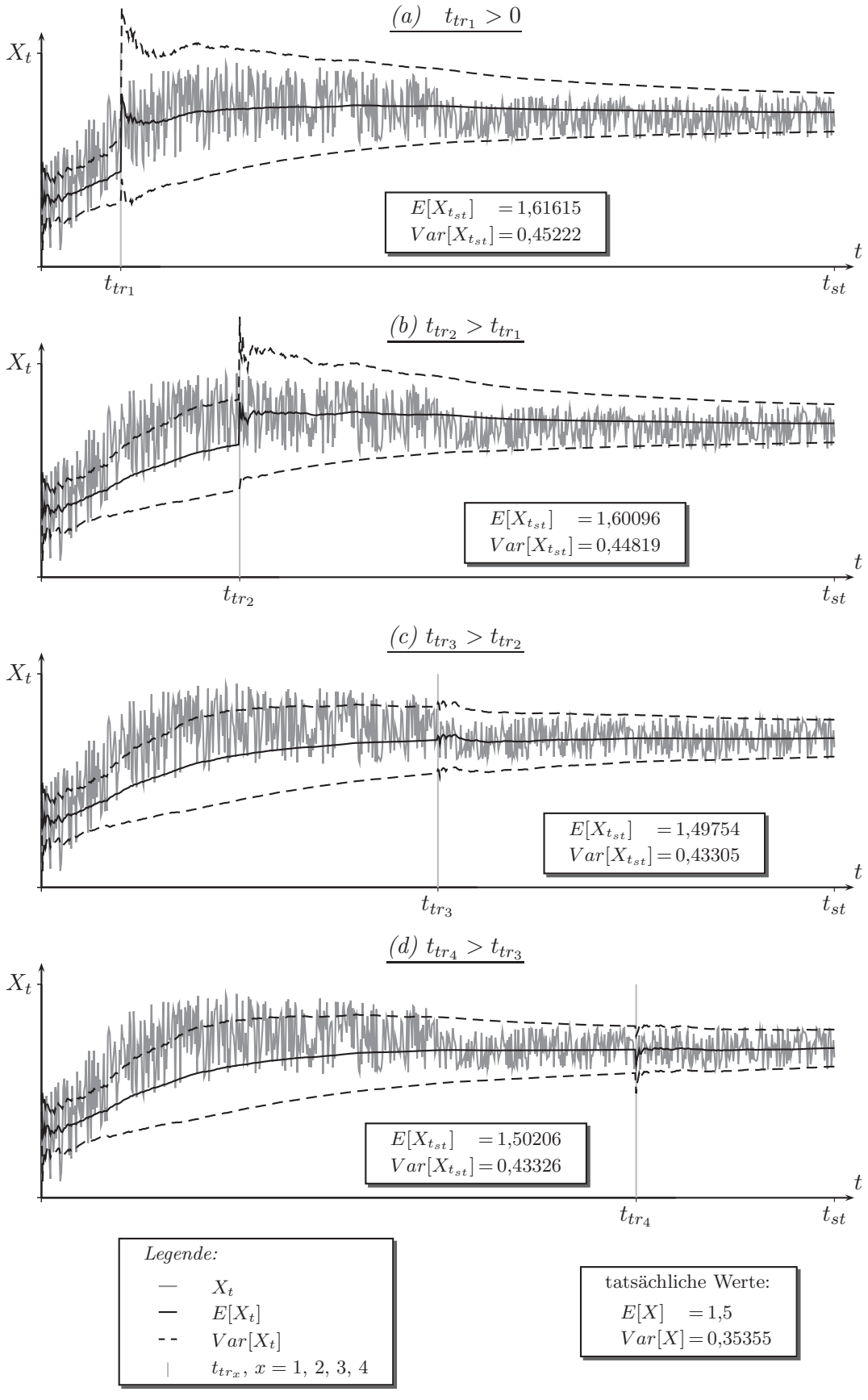


Abb. 3.5.: *Auswirkungen unterschiedlicher Dauern bei der Einschwingphase*

Tab. 3.2

Regel	Nachteil	Literatur-verweis(e)
Conway-Regel	nur bei monotonem Werteverlauf während transienter Phase	[Con63]
Modifizierte Conway-Regel	selber Nachteil wie bei der Conway-Regel	[GAM78]
Tochter-Regel	Zusätzliches Modellwissen erforderlich	[Toc63]
Relaxationsregel	Angabe der Konstante τ durch den Nutzer	[Paw90]
Konstanter-Mittelwert-Regel	Konvergenz des Mittelwertes muss nicht Konvergenz der Verteilung bedeuten	[SE71], [Paw90]
Gruppen-Mittelwert-Regel	selber Nachteil wie die Konstanter-Mittelwert-Regel	[SE71], [Paw90]
Balance-Regel	selber Nachteil wie die Konstanter-Mittelwert-Regel	[SE71]
Balance-χ^2-Regel	selber Nachteil wie die Konstanter-Mittelwert-Regel	[LK00], [Paw90]
Kreuzen des Mittelwertes	selber Nachteil wie die Konstanter-Mittelwert-Regel	[Fis73]
Schrubens-Universeller-Test	aufwändig, wiederholte Tests erforderlich	[Sch83]
Schrubens-Spezieller-Test	setzt Wissen über das Verhalten der transienten Phase voraus	[SST83]
Gordon-Regel	Konvergenz der Varianz muss nicht Konvergenz der Verteilung bedeuten	[GAM78]
Autokorrelations-Regel	direkter Zusammenhang zwischen Autokorrelations-Funktion und Länge der transienten Phase muss nicht bestehen	[Sha75]
Erweiterte-Autokorrelations-Regel	selber Nachteil wie die Autokorrelations-Regel	[Paw90], [Fis71]

Tab. 3.2.: *Regeln für die Bestimmung des Übergangs von transienter Phase in die stationäre Phase*

die transiente Phase beendet und die stationäre Phase erreicht ist, so dass mit der eigentlichen Datensammlung für die Bestimmung der Simulationsergebnisse fortgefahren werden kann. Die Dauer der transienten Phase wird somit bei den statischen Methoden im Vorfeld eines Simulationsexperimentes festgelegt. Gerade bei dieser Vorgehensweise ist die Bestimmung von sinnvollen Werten für die Dauer der transienten Phase notwendig. Hierfür kann z. B. mit *Welchs graphischer Ablesemethode* (siehe [Wel83], [Ale94] oder [Rob05]) im Rahmen von Voruntersuchungen[33] bei der simulationsbasierten Optimierung für ausgewählte oder beliebige Werte der Entscheidungsvariablen die Dauer der transienten Phase bestimmt und anschließend die maximale Dauer der transienten Phase für alle Simulationsexperimente im Rahmen der simulationsbasierten Optimierung genutzt werden. Diese Vorgehensweise hat den Vorteil, dass während eines Laufes der simulationsbasierten Optimierung keine statistischen Daten für die Auswertung des Modellverhaltens gesammelt und numerisch aufbereitet werden müssen. Die verfügbare Rechenzeit kann für die eigentliche Simulation effizient genutzt werden. Nachteilig an der Vorgehensweise ist jedoch, dass bei den Voruntersuchungen durchaus

[33] Voruntersuchungen sind Experimente, welche nicht direkt in die Untersuchungergebnisse eingehen, aber dabei helfen, günstige Parametereinstellungen für verschiedenste Kenngrößen zu ermitteln oder eine Fehlererkennung und -behebung durchzuführen.

ungünstige Werte für die Entscheidungsvariablen ausgewählt werden können, welche dadurch nicht repräsentativ für alle möglichen Werte sind. Es sollten daher verschiedene Voruntersuchungen u. U. auch mit unterschiedlichen Startzuständen vorgenommen werden, um verwertbare Aussagen über die Dauer der transienten Phase treffen zu können. Allerdings kann sich der Vergleich der ermittelten Daten sehr aufwändig gestalten.

Das Manko der ungenauen Vorhersage soll vernachlässigt werden, indem, ohne nähere Untersuchungen, für eine Menge von verschiedenen Werten für die Entscheidungsvariablen und unterschiedlichen Startzuständen die Dauern der transienten Phase ermittelt werden. Die längste ermittelte Dauer wird übernommen und um einen frei gewählten Anteil von 20% bis 50% verlängert, um auch eventuelle Ausreißer mit abzudecken. Anschließend wird sie als konstant während eines Laufes der simulationsbasierten Optimierung angenommen.

Angewandte Methoden. In der vorliegenden Arbeit werden trotz ihrer Nachteile (siehe Tab. 3.2), aber auf Grund ihrer häufigen praktischen Anwendung, die Methoden:

- *Methode des konstanten Mittelwertes* (bzw. Konstanter-Mittelwert-Regel) und
- *Methode des Gruppen-Mittelwertes* (bzw. Gruppen-Mittelwert-Regel)

zur Bestimmung des Endes der transienten Phase genutzt. Der exakte Ablauf der beiden Methoden ist bspw. in [SE71] oder [Paw90] nachzulesen und sei an dieser Stelle nur kurz erwähnt. Die Methode des konstanten Mittelwertes nutzt den laufenden Mittelwert $\overline{X}_i$, welcher für die ersten i Observationen des Modellzustandes X zu den Zeitpunkten $t_{akt} = i \cdot t_{\Delta tr}$ mit $i \in \mathbb{N}^*$ ermittelt wird, wobei $t_{\Delta tr} > 0$ das Prüfintervall angebe. X_i selbst beschreibt den ermittelten Messwert der i-ten Observation einer bestimmten Modellkenngröße[34]. Ändert sich $\overline{X}_i$ ab einem Zeitpunkt $t_{akt} = n_{tr} \cdot t_{\Delta tr}$ nur noch in einem vorgegebenen, konstanten Epsilon-Bereich ϵ_{tr}^{abs} wird angenommen, dass das stationäre Verhalten erreicht ist, d. h. $t_{akt} = t_{tr}$ und

$$\forall i: \quad n_{tr} < i \leq n_{tr} + n_{add} \quad ; \qquad n_{tr},\, n_{add} \in \mathbb{N}^*$$

gelte

$$\frac{|\overline{X}_{n_{tr}+k} - \overline{X}_i|}{|\overline{X}_{n_{tr}+k}|} < \epsilon_{tr}^{abs}$$

mit

n_{add} - Anzahl der Prüfzeitpunkte, bei denen das Endes der transienten Phase noch nicht erreicht ist und

n_{tr} - Multiplikator für die Bestimmung der Dauer der transienten Phase.

Die Methode des Gruppen-Mittelwertes hat einen ähnlichen Ablauf. Allerdings werden bei ihr eine Menge von aufeinander folgenden Messwerten gruppiert und

[34] In der vorliegenden Arbeit kann diese Modellkenngröße bspw. einer der im Vorfeld festzulegenden Ausgabewerte sein.

mit den Gruppen-Mittelwerten als laufenden Mittelwert gearbeitet. Ansonsten ist der Ablauf der Methode identisch mit dem der Methode des konstanten Mittelwertes. Vorteilhaft an dieser Vorgehensweise ist die Reduzierung von starken Schwankungen während der stationären Phase. Durch die Gruppierung mehrerer Messwerte leidet allerdings die Genauigkeit des Verfahrens, weil die Prüfung des Erreichens der stationären Phase immer erst am Ende einer Gruppe vorgenommen wird.

Entscheidend war bei der Auswahl der beiden Verfahren der geringe numerische Aufwand bei der Berechnung der Mittelwerte sowie der geringe Speicherplatzbedarf der beiden Methoden.

Einfluss des Startzustandes. Ein entscheidender Punkt zur Verkürzung der transienten Phase und somit auch der Verkürzung eines Simulationslaufes allgemein, ist die Wahl eines günstigen Startzustandes X_0 für ein Simulationsmodell. Im Wesentlichen können folgende Startzustände betrachtet werden:

„empty and idle“ [engl.].
Es wird ausgehend von einem leeren Ruhezustand, d. h. bspw. leere Kundenwarteschlangen und Fertigungseinheiten im Leerlauf, gestartet. Sinnvoll ist eine solche Vorgehensweise, wenn der Startzustand weitestgehend unbekannt ist.

„steady state mode“ [engl.].
Als Startzustand wird der Zustand des stationären Verhaltens angenommen, welcher mit größter Wahrscheinlichkeit eintritt. Dies hat zwei Nachteile. Zum einen kann im Vorfeld eines Simulationsexperimentes nicht genau gesagt werden, ob überhaupt ein solcher Zustand existiert und zum anderen wie dieser im Falle seiner Existenz aussieht.

„steady state mean“ [engl.].
Dieser Startzustand ist ähnlich dem Startzustand „at steady state mode“, nur dass mit mittlerer Zustandsbelegung gestartet wird.

Im Folgenden und so auch bei den im Kap. 6 durchgeführten Untersuchungen wird stets vom Startzustand „empty and idle“ ausgegangen. Grund dafür ist die Allgemeingültigkeit dieses Ansatzes, dessen Existenz bei allen Simulationsmodellen gewährleistet ist. Daraus folgt, dass der Ansatz auch bei sämtlichen untersuchten Simulationsmodellen dieser Arbeit seine Anwendung findet.

Verhalten bei schwach, mittelmäßig und stark ausgelasteten Simulationsmodellen. Bei den durchgeführten Untersuchungen (siehe Abschn. 6.5) treten stark ausgelastete, aber auch schwach und mittelmäßig ausgelastete Simulationsmodelle auf. Ein stark ausgelastetes System ist im Wesentlichen dadurch gekennzeichnet, dass der Auslastungsgrad $0.9 \leq \rho_{stark} \leq 1$ beträgt[35], wobei i. Allg. $\rho = \frac{\lambda}{\mu}$ und λ die

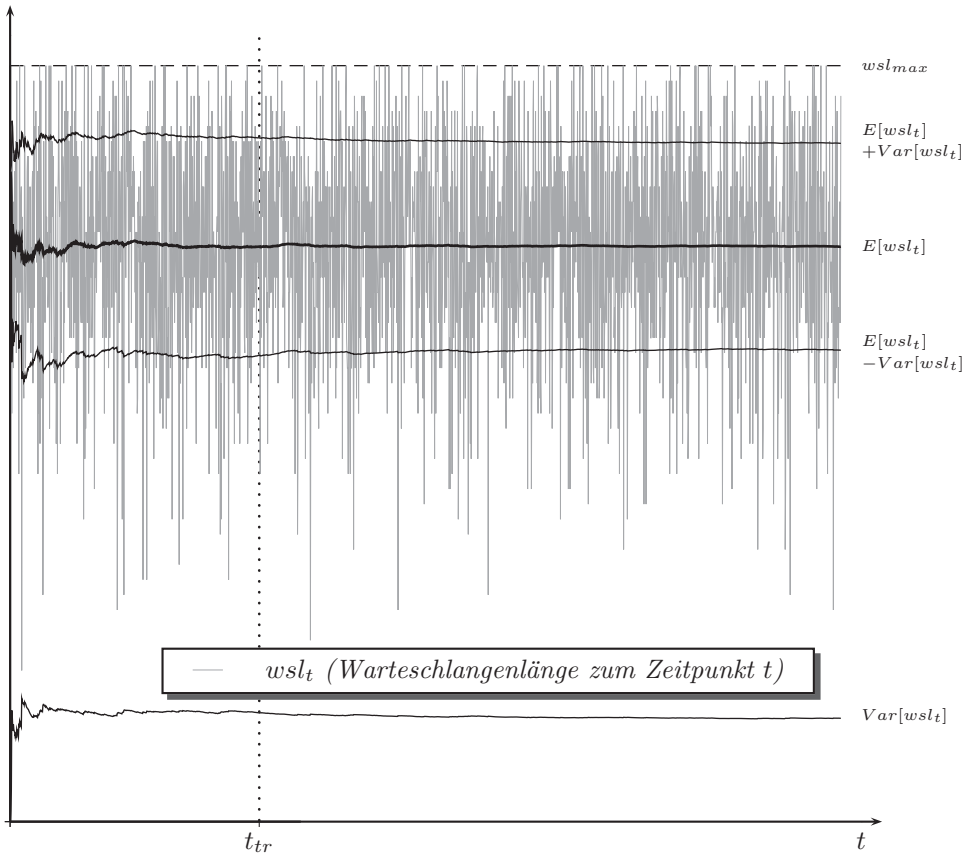
[35] Der Auslastungsgrad eines Simulationsmodells kann maximal 1 betragen. Werte von $\rho > 1$ sind zwar theoretisch möglich ($\lambda > \mu$), aber als abgewiesener oder wartender Bedarf zu modellieren.

Ankunftsrate sowie μ die Bedienrate bezeichne. Es wird dabei davon ausgegangen, dass das Modellverhalten ansonsten nicht weiter beeinflusst wird, d. h. es fallen z. B. keine Umrüstzeiten an. Diese würden den Auslastungsgrad erhöhen, weil die Bedienrate sinkt. Bei schwach ausgelasteten Simulationsmodellen gilt hingegen zumeist $\rho_{schwach} < 0,5$. Der Auslastungsgrad ρ_{mittel} bei mittelmäßig ausgelasteten Simulationsmodellen bewegt sich zwischen stark und schwach ausgelasteten Simulationsmodellen. Die Werte 0,9 als untere Grenze für stark ausgelastete Simulationsmodelle und 0,5 als obere Grenze für schwach ausgelastete Simulationsmodelle zur Spezifizierung eines Simulationsmodells seien nur Beispiele und können in anderer Literatur zum Verhalten von Simulationsmodellen durchaus anders angegeben sein. Es gilt allerdings stets $\rho_{stark} > \rho_{mittel} > \rho_{schwach}$.

Im Folgenden sollen die Auswirkungen der Auslastung am Beispiel des Zusammenhanges zwischen der mittleren Warteschlangenlänge und den mittleren Wartekosten[36] kurz an einem Beispiel betrachtet werden. In Abb. 3.6 (a) ist erkennbar, wie sich die Warteschlangenlänge sowie der zugehörige Erwartungswert und die Varianz eines der untersuchten Simulationsmodelle verhalten. Zum Vergleich zeigt die Abb. 3.6 (b) das Verhalten der mittleren Wartekosten für dasselbe Simulationsmodell während desselben Zeitabschnittes. Es ist erkennbar, dass ein Zusammenhang zwischen der mittleren Warteschlangenlänge und den mittleren Wartekosten besteht. Diese Untersuchung wurde mit verschiedenen Werten für die Entscheidungsvariablen und unterschiedlichen Startwerten für die bei der Simulation verwendeten Zufallszahlengeneratoren durchgeführt und führte zum selben Ergebnis. Das Verhalten war für die untersuchten Simulationsmodelle zu erwarten, weil für alle Kunden unabhängig vom Produkt identische Wartekostenfaktoren bestehen. Durch die Beschränkung der Warteschlangenlänge (obere Grenze b_{all}) ist zudem gesichert, dass auch bei stark ausgelasteten Simulationsmodellen nach einer bestimmten, endlichen Zeit ein stationäres Verhalten eintreten wird. Die gleichen Untersuchungsergebnisse, wie die beim Verhalten zwischen mittlerer Warteschlangenlänge und mittleren Wartekosten, treten auch für die anderen Zielfunktionswerte auf. Allerdings sollen diese Ergebnisse innerhalb dieser Arbeit nicht graphisch dargestellt werden. Durch den Bezug zwischen der mittleren Warteschlangenlänge und den mittleren Wartekosten sollen in der nachfolgenden Abbildung (siehe Abb. 3.6, wobei b_{all} die maximale Warteschlangenlänge angibt) nur noch die mittleren Wartekosten betrachtet werden. Anzumerken sei, dass für vergleichbare Simulationsmodelle ein ähnliches Verhalten zu erwarten ist. Für andere Simulationsmodelle müssten ähnliche Untersuchungen allerdings individuell durchgeführt werden.

[36] Mit Wartekosten werden die (fiktiven) Kosten bezeichnet, welche entstehen, wenn der Bedarfsforderung eines Kunden nicht unmittelbar nachgekommen werden kann. Die Bedarfsforderung des Kunden wird in einer Bedarfswarteschlange einordnet und verweilt dort bis zur Befriedigung des aufgetretenen Bedarfes.

Abb. 3.6 *(a) Warteschlangenlänge (wsl), deren Erwartungswert (E[wsl]) und deren Varianz (Var[wsl])*



(b) Mittlere Wartekosten ($E[wc_t]$)

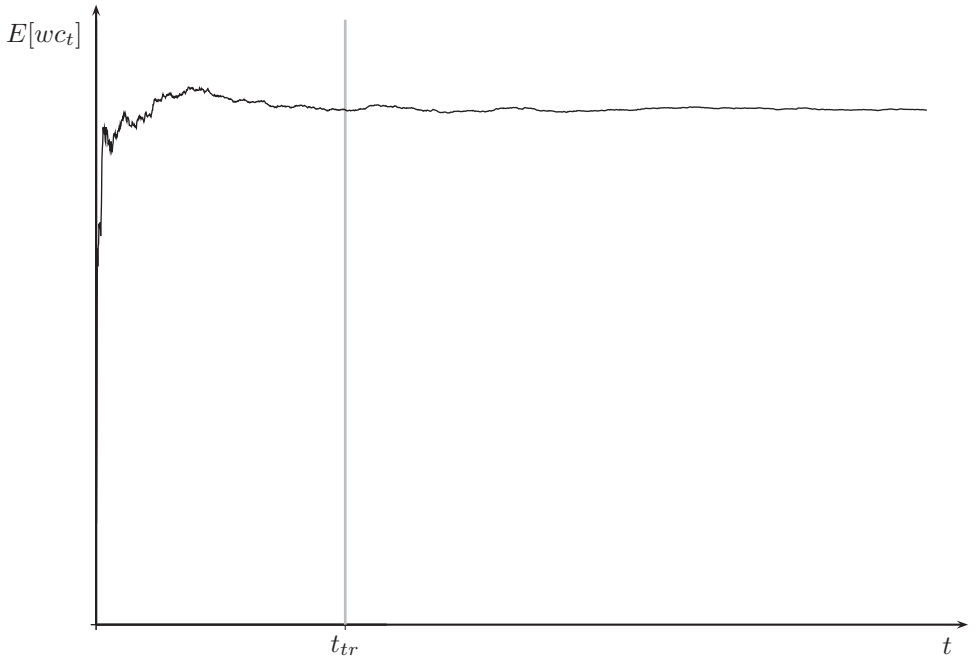


Abb. 3.6.: *Vergleich zwischen Warteschlangenlänge und Wartekosten eines Simulationsmodells (Beispiel)*

Ein weiteres Untersuchungsergebnis ist, dass neben dem Startzustand (s. o.) auch die Auslastung Einfluss auf die Dauer der transienten Phase hat. Anhand der Abb. 3.7 wird erkennbar, dass bei schwach ausgelasteten Simulationsmodellen die transiente Phase am Kürzesten ist (siehe Abb. 3.7 (c)) und bei stark ausgelasteten Simulationsmodellen am Höchsten (siehe Abb. 3.7 (a)). Bei mittelmäßiger Auslastung liegt die Dauer der transienten Phase dazwischen (siehe Abb. 3.7 (b)). Allerdings gilt es zu beachten, dass bei einem anderen Startzustand als dem Startzustand „empty and idle“ das Verhalten durchaus anders sein kann.

Abb. 3.7

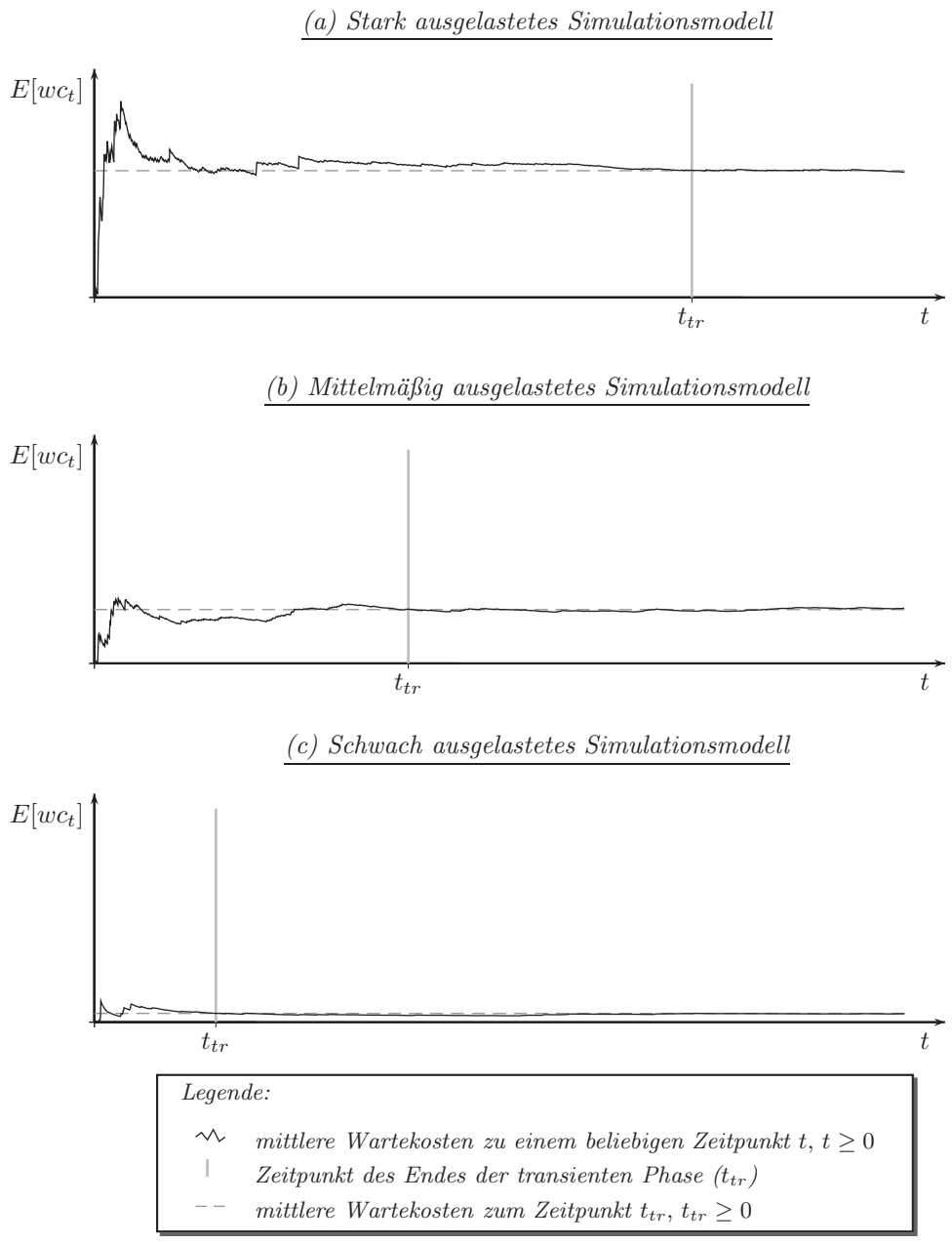


Abb. 3.7.: *Vergleich des Verhaltens von stark, mittelmäßig und schwach ausgelasteten Simulationsmodellen (Beispiel)*

Stationäre Phase. Wie bereits zuvor mehrfach erwähnt, folgt der transienten Phase unmittelbar die stationäre Phase und somit die entscheidende Phase für die Datensammlung der relevanten Ausgabewerte eines Simulationsmodells, auch Simulationsergebnisse genannt. Hierbei haben sich zwei Ansätze zur Reduzierung der Varianz[37] infolge der stochastischen Einflüsse bei der Simulation sowie der Bestimmung der Konfidenzintervalle der Simulationsergebnisse als sinnvoll erwiesen (siehe [LK00, Kap. 9.5]). Einerseits wird ein Ansatz verfolgt, der auf nur einem *einzelnen, langen Simulationslauf* beruht. Zum anderen findet die Methodik der *häufigen Wiederholung möglichst kurzer Simulationsläufe* Anwendung (vgl. Abb. 3.8). Entscheidend ist dabei die Frage, welche *statistische Sicherheit* der zu bestimmende (Mittel-)Wert besitzt. Dazu wurden verschiedene *statistische Methoden zur Bestimmung der Simulationslänge* entwickelt, welche als Ziel die Bestimmung der Simulationslänge unter Einhaltung bestimmter vorgegebener Fehlerniveaus für die zu untersuchenden Werte haben. Diese werden nachfolgend kurz betrachtet.

Abb. 3.8

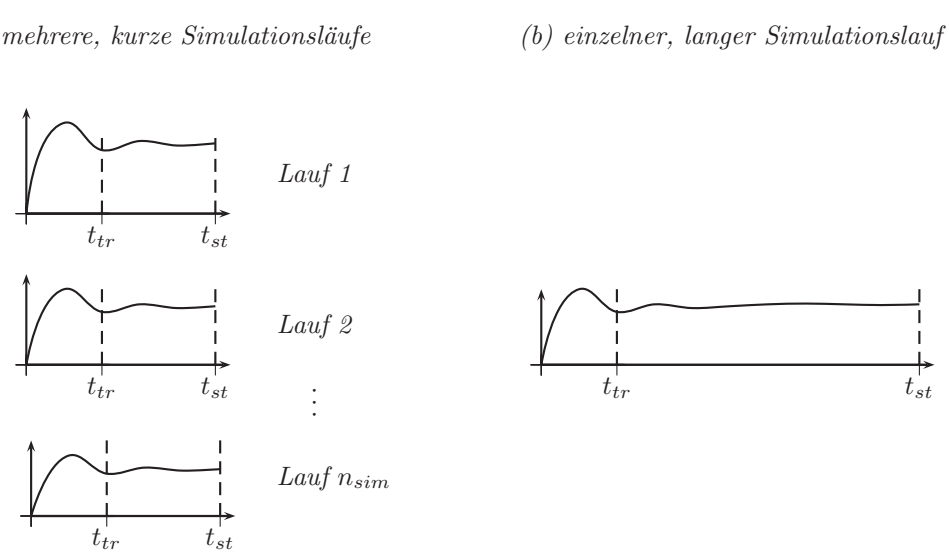


Abb. 3.8.: *Mehrfache Wiederholung kurzer Simulationsläufe kontra einzelnem, langen Simulationslauf*

Vorteilhaft bei einem einzelnen, langen Simulationslauf ist, dass die transiente Phase nur einmal durchlaufen werden muss und somit kaum Rechenzeit für deren wiederholte Bestimmung verloren geht. Jedoch ist bei einem einzelnen Lauf infolge der stochastischen Einflussgrößen auch eine stochastische Abhängigkeit der gesammelten Daten vorhanden.

Dieser Nachteil der stochastischen Abhängigkeit wird durch mehrere kurze Simulationsläufe umgangen, welche ihrerseits aber jeweils die transiente Phase durchlaufen müssen. Ein Mittelweg, welcher beide Ansätze vereint, wäre demnach sinnvoll. Ohne eine genauere Güteanalyse vorzunehmen, könnte bspw. mehrfach ein längerer Simulationslauf mit fest vorgegebener Dauer für die stationäre Phase sowie

[37] Eine kleine Varianz deutet zwangsläufig auf geringe Schwankungen des Mittelwertes hin, was wiederum ein Indiz für das Erreichen des stationären Zustandes sein kann. Bei Systemen, welche das stationäre Verhalten erreicht haben, nimmt die Varianz im Verlaufe eines Simulationslaufes ab.

unterschiedlichen Startwerten des, bei der Simulation verwendeten, Zufallszahlengenerators, zur Realisierung der Unabhängigkeit der einzelnen Simulationsläufe, durchgeführt werden. Diese Methodik sei als *statische Methodik* für die Dauer der stationären Phase eines Simulationslaufes bezeichnet. Die Analyse der stationären Phase erfolgt bei diesen Prozeduren im Anschluss an die Simulationsläufe.

Darüber hinaus können aber auch *dynamische Methodiken* für die Bestimmung einer hinreichenden Dauer der stationären Phase zum Einsatz kommen (siehe [Koz04]). Diese zielen jedoch zunächst nicht direkt auf die Ausgabewerte eines Simulationsmodells ab, sondern formen diese i. Allg. in geeignete, zu untersuchende Kenngrößen um. Diese dynamischen Methoden, zu denen u. a. auch die *Batch-Means-Verfahren* [dt.; Stapel-Mittelwerte-Verfahren [dt.]] gehören, sind numerisch recht aufwändig. Beim *sequentiellen Batch-Means-Verfahren* ist bspw. die Zusammenfassung der Messdaten zu Batch-Mittelwerten, der anschließenden Prüfung der Batch-Mittelwerte auf Autokorrelation, usw. notwendig. Für eine genauere Beschreibung des Batch-Means-Verfahrens, seiner Varianten sowie weiterer dynamischer Methodiken zur Bestimmung des stationären Zustandes siehe bspw. [Jai91], [SH95] oder [Eic02]. Die dynamischen Methoden werden in der Literatur zudem in fixed-sample-size-procedures [engl.; Prozeduren mit fester Abtastlänge [dt.]] und sequentielle Prozeduren unterteilt (siehe [Koz04]). Diese Unterteilung ist ähnlich der bei zeitgesteuerter oder ereignisgesteuerter Simulation. Die Prozeduren mit fester Abtastlänge testen das Erreichen des stationären Zustandes zu bestimmten Zeitpunkten ab und sind somit zeitgesteuert. Näheres zu unterschiedlichen dynamischen Methoden mit fester Abtastlänge ist in [Ban98, Kap. 7] und [Koz04] zu finden. Die sequentiellen Prozeduren hingegen führen die Prüfung auf Erreichen des stationären Zustandes nach dem Eintreten bestimmter Simulationsereignisse durch. Für die Vertiefung zur Thematik der sequentiellen Prozeduren sei auf [Fis01] und [BINN01] verwiesen.

Angewandte Vorgehensweise. Innerhalb dieser Arbeit wird eine semi-dynamische Methode[38] für die Festlegung der Dauer der stationären Phase angewandt. Diese nutzt die ermittelte Dauer der transienten Phase t_{tr}. Die Dauer wird um einen freigewählten Anteil verlängert und als Dauer der stationären Phase t_{st} übernommen, d. h. es gelte $t_{st} = t_{tr} + \Delta t$ oder $t_{st} = n_{st} \cdot t_{tr}$ mit $n_{st} \in \mathbb{R}_+^*$. Der Autor schlägt im Rahmen der hiesigen Untersuchungen die Nutzung der Gleichung $t_{st} = n_{st} \cdot t_{tr}$ für $3 \leq n_{st} \leq 5$ vor (siehe Abb. 3.9).

Abb. 3.9

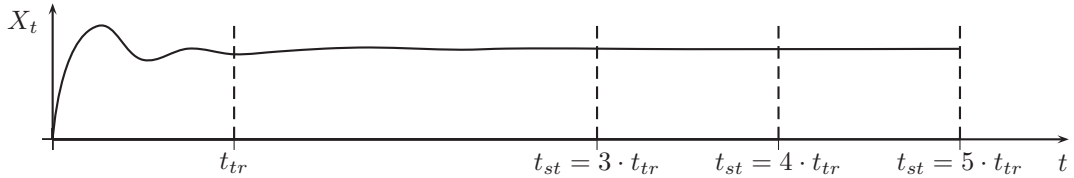


Abb. 3.9.: *Heuristik zur Festlegung der Dauer der stationären Phase (Beispiel)*

[38] Der Begriff „semi-dynamisch“ wird verwendet, weil die Methode abhängig von der Dauer der transienten Phase ist, wobei jedoch die Anzahl der Prüfzeitpunkte bzw. die Größe eines Prüfintervalls im Vorfeld festgelegt wird.

Ein wesentlicher Vorteil dieser Vorgehensweise ist die Reduzierung des numerischen Aufwandes für die Berechnung des Simulationsendes, welches gleichbedeutend mit dem Ende der stationären Phase ist. Die Berechnung beschränkt sich im Wesentlichen auf die Bestimmung des Zeitpunktes für das Simulationsende und das Einfügen des entsprechenden Ereignisses in die Ereignisliste.

Ergebnisaufbereitung bei mehreren Simulationsläufen. Im Anschluss an die einzelnen Simulationsläufe werden die Simulationsergebnisse aufsummiert und gemittelt, so dass gelte:

$$\begin{aligned} f_{A_i}(\vec{x}) = E[\overline{f}_{A_i}(\vec{x})] &= \tfrac{1}{l_{sim}} \sum_{j=1}^{l_{sim}} \overline{f}^{j}_{A_i}(\vec{x})\,; \\ &\quad i = 1, 2, \ldots, \dim(\vec{f_A}(\vec{x})) \\ Var[\overline{f}_{A_i}(\vec{x})] &= \tfrac{1}{n_{sim}-1} \sum_{j=1}^{n_{sim}} (\overline{f}^{j}_{A_i}(\vec{x}) - f_{A_i}(\vec{x}))^2\,; \\ &\quad i = 1, 2, \ldots, \dim(\vec{f_A}(\vec{x}))\,; \end{aligned}$$

wobei

n_{sim}	-	Anzahl der durchgeführten Simulationsläufe,
$\overline{f}^{j}_{A_i}(\vec{x})$	-	gemittelte Ausgabewerte von A_i des j-ten Simulationslaufes,
$E[\overline{f}_{A_i}(\vec{x})]$	-	Erwartungswert des Ausgabewertes $\overline{f}_{A_i}(\vec{x})$ und
$Var[\overline{f}_{A_i}(\vec{x})]$	-	Varianz des Ausgabewertes $\overline{f}_{A_i}(\vec{x})$.

3.6. Verkürzung der Simulationsdauer bei diskreter Simulation

Nachfolgend wird anhand verschiedener Methodiken aufgezeigt, wie die Simulationsdauer eines Simulationslaufes im Rahmen der simulationsbasierten Optimierung reduziert werden kann. Grund für diese Betrachtungen ist die mitunter hohe Laufzeit eines Laufes der simulationsbasierten Optimierung, welche eine Anwendung in der Praxis oftmals erschwert.

Verteilte und parallele Simulation. Die Anwendung der *verteilten und parallelen Simulation* zielt auf eine Reduktion der Verarbeitungszeit beim Ablauf eines einzelnen Simulationslaufes ab (siehe [Fuj00]). Es wird versucht auf einem (vernetzten) Multiprozessorsystem, dessen Kommunikation i. d. R. über ein vorgegebenes Protokoll abläuft, eine verteilte und parallele Simulation durch die Zerlegung eines größeren Simulationmodells in weitestgehend voneinander unabhängige Simulationsmodellteile durchzuführen[39]. Diese gewünschte Unabhängigkeit kann jedoch nicht immer erreicht werden, weil unter Umständen zu bestimmten Zeitpunkten notwendige Informationen über den aktuellen Zustand eines Simulationsteiles mit

[39] In diesem Zusammenhang wird auch von Netzwerksimulation gesprochen.

einem anderen Simulationsteil ausgetauscht werden müssen, um erforderliche Informationen für den weiteren Simulationsablauf zu erhalten. Die Ergebnisse eines Simulationslaufes dürfen durch die Verteilung aber in keinem Fall verändert werden. Entsteht bei einem Simulationsteil ein neuer Systemzustand, kann dies bspw. eine Änderung des Systemzustandes eines anderen Simulationsteils nach sich ziehen usw. Dies macht eine Synchronisation der verschiedenen Simulationsteile zu bestimmten Zeitpunkten erforderlich.

Als Beispiel für einen möglichen Ansatz für die Synchronisation sei auf eine andere vom Autor verfasste Arbeit verwiesen (siehe [Käm03]). Innerhalb dieser wird das *Fleet Sizing and Allocation Problem* [engl.; kurz: *FSAP*] betrachtet, bei dem eine Zerlegung in verschiedene (Teil)Regionen stattfindet, welche jeweils auf einem eigenen Rechner simuliert werden. Die Rechner selbst sind identisch und durch ein Kommunikationsnetzwerk verbunden (siehe [Che07a]). Die verschiedenen Überschuss- und Mangelregionen, welche während eines Simulationslaufes auftreten, müssen zu bestimmten, zufälligen Zeitpunkten bewegliche, zu vermietende Ressourcen untereinander austauschen. Dieser Austausch verhindert es, dass die verschiedenen Simulationsteile unabhängig nebeneinander ablaufen können. Folglich muss eine Kommunikation und ggf. eine Synchronisation zwischen den simulierenden Rechnern vorgenommen werden.

Allgemein kann gesagt werden, dass bei der Synchronisation an sich je nach *Synchronisationszeitpunkt* zwischen *synchronen* und *asynchronen Verfahren* unterschieden wird. Die asynchronen Verfahren werden ihrerseits noch in *konservative* (z. B. *Link-Time-Verfahren*), *optimistische Verfahren* (z. B. *Time-Warp-Verfahren* oder *Global-Virtual-Time-Verfahren*) sowie *hybride Verfahren*, welche konservative und optimistische Verfahren vereinigen, untergliedert (siehe Abb. 3.10). Näheres zu den einzelnen Synchronisationsverfahren und deren Varianten ist u. a. in [Ort94] und [Mei99] zu finden. Die wesentlichen Vor- und Nachteile der beiden Verfahrensarten, synchrone und asynchrone Synchronisation, sowie deren Vereinigung innerhalb einer hybriden Verfahrensart sind zum besseren Verständnis der Problematiken mit den Verfahrensarten noch einmal in Tab. 3.3 zusammengefasst.

Abb. 3.10

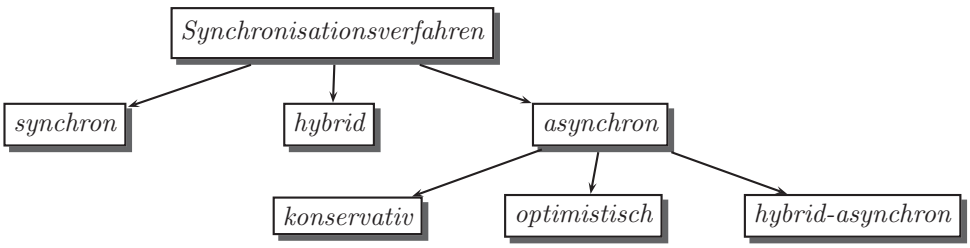


Abb. 3.10.: *Synchronisationsverfahren der verteilten und parallelen Simulation*

Auf Grund der unterschiedlichen *Zeitfortschreibung der einzelnen Simulationsteile* bei der asynchronen Synchronisation sind die aktuellen Simulationszeiten der einzelnen Simulationsteile zumeist unterschiedlich. Es wird eine Unterscheidung zwischen *lokaler Simulationszeit* der einzelnen Simulationsteile und *globaler Simu-*

Tab. 3.3

Synchronisationsart	Pro und Kontra	
asynchron	Pro:	• Verkürzung der Verarbeitungsdauer eines Simulationsexperimentes bei geringer Kommunikation untereinander und etwa gleicher Zeitfortschreibung der Simulationsteile
	Kontra:	• bei hohem Synchronisationsaufwand (häufiges Zurückrollen [dt.; roll backs] beim time-warp-Verfahren [engl.]) mitunter deutlich höhere Verarbeitungsdauer • häufiges Versenden von anti-messages [engl.; Gegennachrichten [dt.]] beim time-warp-Verfahren [engl.] belasten den Kommunikationskanal • Kompromiss zwischen Häufigkeit der Statussicherung und dem damit verbundenen Speicherplatzbedarf sowie der Häufigkeit der Wiederholungen eines Simulationsabschnittes und der damit verbundenen Belastung eines Rechnerprozessors
synchron	Pro:	• bei sinnvoll gewählten Synchronisationszeitpunkten ist der Synchronisationsaufwand und die damit verbundene Belastung des Kommunikationsnetzwerkes deutlich geringer als beim asynchronen Verfahren • Master-Prozess kann deadlocks [engl.; Verklemmungen [dt.]] verhindern
	Kontra:	• nicht für jedes Simulationsmodell geeignet • Master-Prozess zur Synchronisation erforderlich
hybrid	Pro:	• vereint Vorteile der synchronen und asynchronen Verfahren
	Kontra:	• mitunter sehr sensibel und individuell zu implementieren und zu untersuchen

Tab. 3.3.: *Pro und Kontra der verteilten und parallelen Simulation (Auszug der innerhalb dieser Arbeit entscheidenden Kriterien)*

lationszeit des Gesamtsystems notwendig. Der Autor hatte in seiner früheren Arbeit sowohl eine synchrone Verfahrensvariante als auch das Time-Warp-Verfahren als asynchrone Verfahrensvariante untersucht und gezeigt, dass in jedem Falle eine häufige Synchronisation keine Vorteile bei der Abarbeitungsgeschwindigkeit eines Simulationsexperimentes mit sich bringt. Lediglich bei der synchronen Synchronisationsvariante konnte, durch Veränderungen des nichttrivial parallelisierbaren Simulationsmodelles, wie z. B. durch Zerlegung, Umstrukturierung und Gruppierung in weitestgehend unabhängige Simulationsteile (Teilregionen beim FSAP) mit Hilfe von Verfahren der Graphentheorie, ein zeitlicher Vorteil durch die Nutzung der verteilten und parallelen Simulation aufgezeigt werden. Diese Zerlegung des Simulationsmodells für die Durchführung der synchronen Synchronisationsvariante diente dazu, eine weitestgehende Unabhängigkeit der Simulationsteile zu erreichen, womit eine annähernd gleiche Zeitfortschreibung innerhalb der Simulationsteile ermöglicht werden sollte.

Erwähnt sei an dieser Stelle, dass neben den zuvor beschriebenen *statischen Methoden* bei der verteilten und parallelen Simulation, auch *dynamische Methoden* existieren. Währenddessen die statischen Methoden unabhängig von dem während der Simulation genutzten Synchronisationsverfahren im Vorfeld der Simulation eine Aufteilung des Simulationsmodells in verschiedene Simulationsteile vornehmen, kommen die dynamische Methoden besonders in heterogenen Rechnernetzsyste-

men mit schwankender Leistungsfähigkeit der Berechnungsknoten zum Einsatz. Diese dynamischen Methoden versuchen während der Simulation mittels eines *Monitoring-Verfahrens* eine Migration der einzelnen Simulationsteile auf andere Rechner durchzuführen (siehe [MM89] und [Web00, Kap. 4]). Dadurch kann unter erhöhtem Rechenzeitaufwand versucht werden, einen *Lastenausgleich* [dt.; *load balancing* [engl.]] auf die parallel arbeitenden Rechner vorzunehmen. Im Extremfall geht durch das Monitoring und den Lastenausgleich mehr Rechenzeit verloren, als durch den Vorteil der Dynamik entsteht. Der Einsatz dieser Verfahren ist demnach gut abzuwägen.

Zusammenfassend kann gesagt werden, dass eine verteilte und parallele Simulation nur dann sinnvoll ist, wenn

- eine *hohe Lokalität der Simulationsereignisse eines Simulationsmodellteils* vorliegt,
- nur *wenige Simulationsereignisse von anderen Simulationsmodellteilen die lokalen Berechnungen des eigenen Simulationsmodellteils* beeinflussen und
- somit letztendlich der *notwendige Kommunikations- und Synchronisationsaufwand das Maß an möglicher Parallelität nicht übersteigt.*

Es wird für die vorgenommenen Simulationsexperimente, ohne nähere Untersuchungen, davon ausgegangen, dass bei den innerhalb der vorliegenden Arbeit untersuchten Simulationsmodellen (siehe Abschn. 6.3) keine höhere Abarbeitungsgeschwindigkeit durch verteilte und parallele Simulation erreicht werden kann. Grund dafür ist die fehlende Lokalität der Simulationsereignisse innerhalb der Simulationsmodellteile, welche einen erheblich höheren Synchronisationsaufwand nach sich ziehen würden.

Der Autor möchte an dieser Stelle anmerken, dass es für die Durchführung einer verteilten und parallelen Simulation derzeit keine kommerziellen Softwaresysteme gibt. Es existieren nur individuelle, softwaretechnische Einzellösungen, was zum einen an der nichttrivialen Zerlegbarkeit der einzelnen Simulationsmodelle und zum anderen an der Problematik der Verteilung der Simulationssoftware als Folge fehlender Softwarelizenzmodelle für einen solchen Fall liegt.

Triviale parallele Simulation. Durch die wiederholte Durchführung eines Simulationsexperimentes mit den gleichen Werten für die Entscheidungsvariablen innerhalb mehrerer Simulationsläufe, kann, wie bereits beschrieben, der Einfluss der stochastischen Größen reduziert und ggf. ein Konfidenzintervall für die Ergebnisse eines Simlationsexperimentes bestimmt werden (siehe Abschn. 3.2). Hierbei ist die Beschleunigung eines Simulationsexperimentes durch die *triviale Verteilung der einzelnen Simulationsläufe eines Simulationsexperimentes* auf jeweils verschiedene Rechner bzw. bei Mehrprozessorsystemen auf mehrere Prozessoren denkbar (siehe Abb. 3.11).

Abb. 3.11

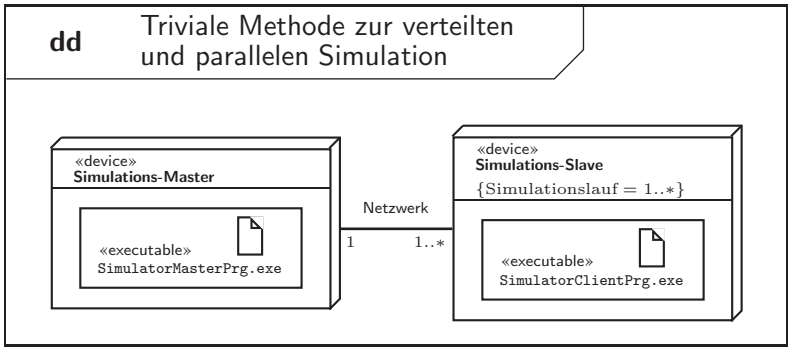


Abb. 3.11.: *Triviale Verteilung der Simulationsläufe eines Simulationsexperimentes (UML-Verteilungsdiagramm)*

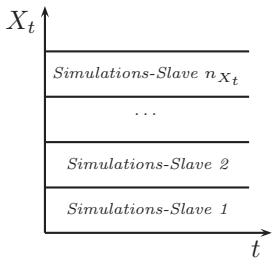
Datenparallele Simulation. Bei der *datenparallelen Simulation* kann sowohl eine Zerlegung des Zeitraumes als auch eine Zerlegung des Zustandsraumes erfolgen (siehe [Käm03]). Die Zerlegung des Zustandsraumes (siehe Abb. 3.12 (a)) erweist sich bei den untersuchten Simulationsmodellen als schwierig, weil die nachfolgenden Simulationsereignisse von den vorherigen chronologisch abhängig sind. Diese Aussage ist für die meisten Simulationsexperimente zutreffend, weil deren Zustandsraum i. Allg. zu groß ist, um eine sinnvolle Zerlegung in eine endliche Anzahl von Zustandsräumen vornehmen zu können. Zudem muss ein Simulationsexperiment nicht zwangsläufig den gesamten möglichen Zustandsraum durchlaufen.

Eine im Rahmen der verteilten und parallelen Simulation sinnvolle Zerlegung des Zeitraumes (siehe Abb. 3.12 (b)) ist bei den meisten Simulationen ebenso schwierig. Es kann i. d. R. nicht genau vorhergesagt werden, welchen Zustand ein Simulationsmodell zu einem bestimmten Zeitpunkt erreicht hat.

Ist es dennoch möglich, für die Zerlegung des Zustands- oder Zeitraumes eine solche vorausschauende Sichtweise zu realisieren, kann i. Allg. ein zeitlicher Vorteil durch die Zerlegung erzielt werden.

Abb. 3.12

(a) Teilung des Zustandsraumes



(b) Teilung des Zeitraumes

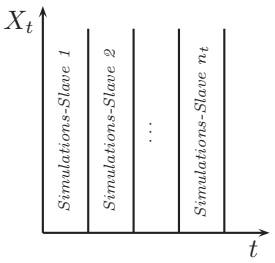


Abb. 3.12.: *Verfahren der datenparallelen Simulation (vgl. [Käm03, S. 23, Abb. 4])*

Neue Methode bei simulationsbasierter Optimierung (Epsilon-Abbruch-Verfahren). Ist ein Simulationsmodell gegeben, bei welchem infolge von Voruntersuchungen

festgestellt wurde, dass keine Zerlegung in verschiedene weitestgehend unabhängige Simulationsmodellteile oder keine Zerlegung des Zustandsraumes möglich ist, sind andere Methoden zur Verkürzung der Abarbeitungszeit eines einzelnen Simulationslaufes erforderlich. Gleiches gilt, wenn eine Zerlegung keinen zeitlichen Vorteil mit sich bringt. Eine solche andere heuristische Methodik könnte bspw. die Verkürzung der Simulationsdauer bei voraussichtlich schlechteren Zielfunktionswerten sein.

Diese heuristische Vorgehensweise, welche sich an der Methode des konstanten Mittelwertes (s. o. (transiente Phase)) anlehnt, wird in dieser Arbeit angewandt. Sie zielt darauf ab, ein Simulationsexperiment dann zu beenden, wenn im weiteren Verlaufe des Simulationsexperimentes *keine Verbesserung gegenüber dem aktuell besten Zielfunktionswert mehr zu erwarten ist* (siehe Abb. 3.13). Mit dem aktuell besten Zielfunktionswert ist der im bisherigen Verlauf des aktuellen Laufes der simulationsbasierten Optimierung beste gefundene Zielfunktionswert $f_k(\vec{x}_{best})$ gemeint. Die Vorgehensweise wird vom Autor als *Epsilon-Abbruch-Verfahren der simulationsbasierten Optimierung* bezeichnet.

Die Heuristik betrachtet folglich den Verlauf der relevanten Ausgabewerte der aktuell durchgeführten Simulation. Die Abweichung des Erwartungswertes eines Ausgabewertes $E[f_{t,\,k}(\vec{x})]$ von $f_k(\vec{x}_{best})$ wird jeweils zu bestimmten Zeitpunkten $t = t_{tr} + i_P \cdot \Delta t_P$ überprüft, wobei $i_P \in \mathbb{N}^*, \Delta t_P \in \mathbb{R}_+^*$ gilt. Die Variable i_P bezeichne die Nummer des aktuellen Prüfzeitpunktes und Δt_P das Prüfintervall, wobei o. B. d. A. $\frac{t_{tr}}{4} \leq \Delta t_P \leq \frac{t_{tr}}{2}$ gewählt werden sollte, wobei $i_P < \lfloor \frac{t_{sim} - t_{tr}}{\Delta t_P} \rfloor$. Die erste Prüfung findet demnach zum Zeitpunkt $t_{tr} + \Delta t_P$ statt.

Es ergibt sich als absoluter Wert für die Veränderung zum Zeitpunkt t

$$\delta_{abs}^{E_{\Delta t_P,\,k}} = |E[f_{t,\,k}(\vec{x})] - f_k(\vec{x}_{best})|$$

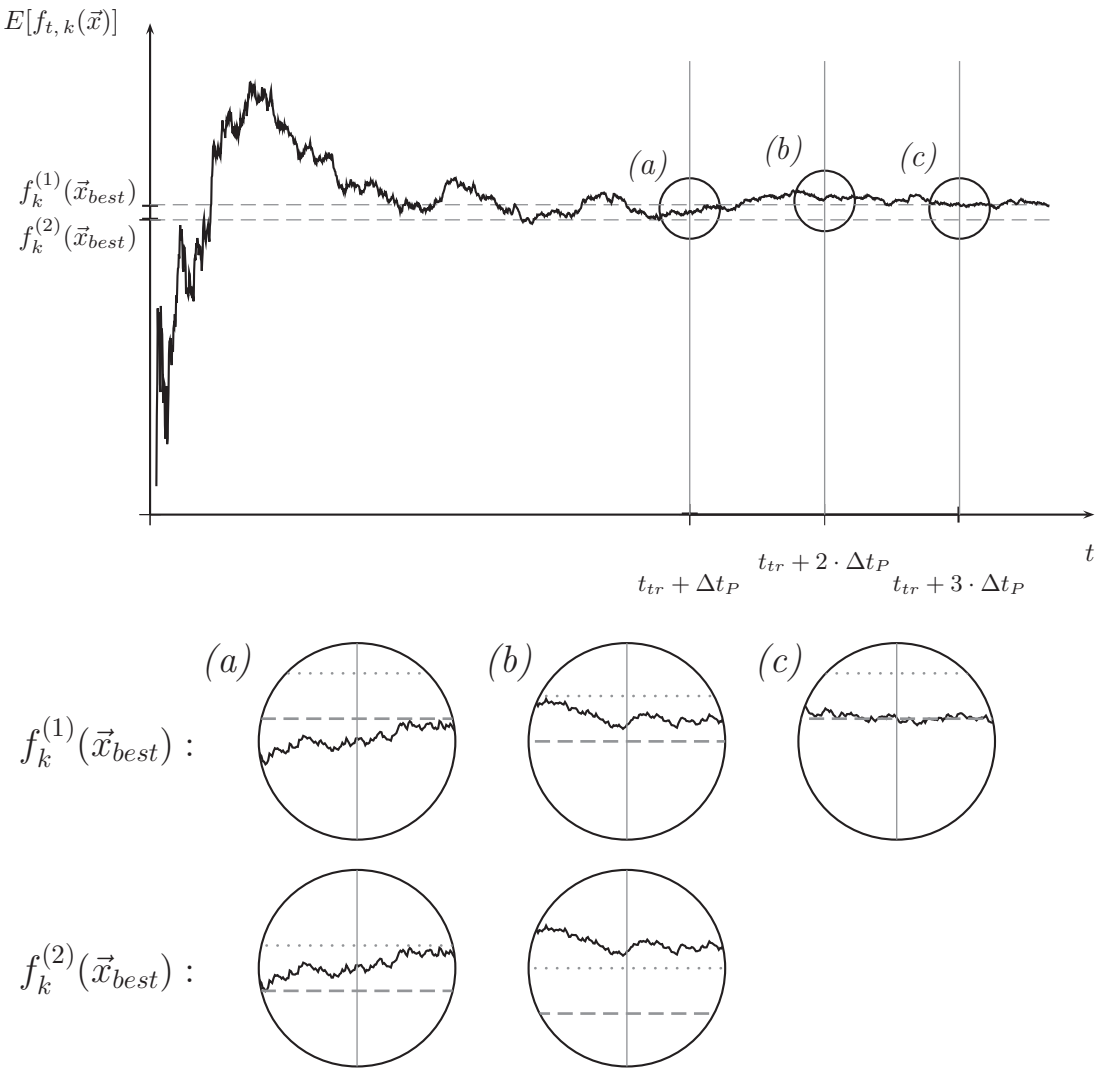
und als relativer Wert

$$\delta_{rel}^{E_{\Delta t_P,\,k}} = \frac{|E[f_{t,\,k}(\vec{x})] - f_k(\vec{x}_{best})|}{|E[f_{t,\,k}(\vec{x})]|} = 1 - \frac{f_k(\vec{x}_{best})|}{E[f_{t,\,k}(\vec{x})]} \quad .$$

Die Heuristik sieht vor, dass bei Unterschreiten eines im Vorfeld der simulationsbasierten Optimierung für alle Vergleiche festgelegten, relativen Epsilon-Wertes $\epsilon_{rel}^{E_{\Delta t_P,\,k}}$ oder absoluten Epsilon-Wertes $\epsilon_{abs}^{E_{\Delta t_P,\,k}}$ (vgl. Epsilon-Umgebung in Abschn. 2.2.6) die Simulation bis zum nächsten Prüfzeitpunkt uneingeschränkt fortgesetzt wird und dann eine erneute Prüfung erfolgt. Andernfalls wird der aktuelle Simulationslauf abgebrochen bzw. vorzeitig beendet. Ist das Ende der stationären Phase vor dem nächsten Prüfzeitpunkt erreicht, wird die Simulation beendet ohne eine weitere Prüfung vorzunehmen.

Zur Eliminierung des zufälligen Eintreten eines günstigen Prüfwertes wird vom Autor vorgeschlagen, diese Bereichsprüfung mehrmals durchzuführen. Als sinnvoll erweist es sich die Prüfung 3 bis 5 mal zu wiederholen, bis der eindeutige Erfolgsfall eintritt. Dies bedeutet, dass in allen Fällen die Prüfung dasselbe „positive“ Ergeb-

Abb. 3.13



Legende:

	$E_t[f_k(\vec{x})]$ *mit* $f_k(\vec{x}) \rightarrow \min$
– –	*Wert von* $f_k^{(i)}(\vec{x}_{best})$
· · · ·	*Wert von* $f_k^{(i)}(\vec{x}_{best}) + \epsilon_{abs}^{E_{\Delta t_P,\,k}}$
|	*Prüfzeitpunkt*

$E[f_{t,\,k}(\vec{x})]$ - *Erwartungswert des k-ten Zielfunktionswertes zum Zeitpunkt t*

$f_k^{(i)}(\vec{x}_{best})$ - *bisher bester k-ter Zielfunktionswert des Laufes der simulationsbasierten Optimierung,* $i = 1, 2$

Δt_P - *Prüfintervall*

Abb. 3.13.: *Beispiel für das Epsilon-Abbruch-Verfahren mit absoluter Epsilon-Umgebung*

nis liefern muss, um eine zuverlässige Aussage über den Abbruch eines Simulationslaufes treffen zu können. Liefert die Prüfung für einen festgelegten Zeitpunkt eine Verletzung der festgelegten Epsilon-Umgebung durch den Prüfwert zurück, obwohl die vorherigen Prüfungen keine Verletzung ergaben, beginnt die Prüfung zum nächsten Prüfzeitpunkt erneut. Dabei wird sich nur das aktuelle Prüfergebnis gemerkt und die vorherigen, andersartigen Prüfergebnisse vergessen. Es wird demnach nur bei einer wiederholten Verletzung der Epsilon-Umgebung durch den Prüfwert der aktuelle Simulationslauf abgebrochen bzw. vorzeitig beendet.

An dieser Stelle sei angemerkt, dass das vorgestellte Epsilon-Abbruch-Verfahren als hinreichendes heuristisches Prüfverfahren zur vorzeitigen Beendigung eines Simulationslaufes für verschiedene Untersuchungen ausreichend ist. Allerdings muss bei dem Verfahren gesichert sein, dass

- die *vorgegebene Epsilon-Umgebung sinnvoll gewählt* ist (weder zu klein noch zu groß)[40],
- die *Prüfung* erst *nach ausreichend fortgeschrittener Simulationsdauer*[41] erfolgt und
- die *Häufigkeit der Prüfung*[42] den eigentlichen Simulationsablauf kaum einschränkt.

Sinnvolle Verfahrensparameter für das Epsilon-Abbruch-Verfahren können bspw. durch verschiedene Voruntersuchungen bestimmt werden.

Zur Sicherstellung des zuvor beschriebenen, gewünschten Verhaltens des Epsilon-Abbruch-Verfahrens kann zusätzlich die Varianz der Zufallsgröße zur Prüfung mit einbezogen werden, was jedoch den numerischen Berechnungsaufwand und den Speicherbedarf erhöht. Dabei könnten als absolute oder relative Werte für die Epsilon-Umgebung der Varianz die Größen $\epsilon_{abs}^{Var_{\Delta t_P,\,k}}$ und $\epsilon_{rel}^{Var_{\Delta t_P,\,k}}$ genutzt werden, welche dann mit den berechneten Werten $\delta_{abs}^{Var_{\Delta t_P,\,k}}$ und $\delta_{rel}^{Var_{\Delta t_P,\,k}}$ zu vergleichen sind.

Eine weitere Möglichkeit des vorgestellten Epsilon-Abbruch-Verfahren besteht darin, dass für verschiedene Ausgabewerte eines Simulationsmodells auch verschiedene Epsilon-Umgebungen angegeben werden können. Die konkreten Ausgabewerte des Simulationsmodells werden dann im Verlauf der Prüfung auf Verlassen der Epsilon-Umgebung getestet. Liegen alle Werte außerhalb ihrer Epsilon-Umgebung, wird der aktuelle Simulationslauf vorzeitig abgebrochen. Diese Vorgehensweise sei auch als *mehrkriterielles Epsilon-Abbruch-Verfahren* bezeichnet und zielt indirekt auf die Verwendung im Zusammenspiel mit einem mehrkriteriellen Optimierungsverfahren ab, welches mit (konkurrierenden) Zielfunktionswerten arbeitet. Allerdings wird es

[40] Diese Problematik soll durch die Angabe des relativen Epsilon-Wertes $\epsilon_{rel}^{E_{\Delta t_P,\,k}}$ teilweise abgeschwächt werden.

[41] Eine zu frühe Prüfung beeinflusst die Prüfung negativ, weil der Einfluss der transienten Phase mitunter noch zu groß ist.

[42] Ein kurzes Prüfintervall erhöht zwangsläufig die Häufigkeit der Prüfung, so dass viele Berechnungen für die Tests notwendig werden. Ein langes Prüfintervall führt dagegen zu einem späteren Abbruch des Simulationslaufes. Der Abbruch des Simulationslaufes ist aber durch das Erreichen des Endes der stationären Phase gesichert.

durch die Angabe mehrerer Epsilon-Umgebungen für einen Anwender schwerer, für diese Epsilon-Umgebungen „sinnvolle“ Epsilon-Werte anzugeben.

Das Ergebnis der Untersuchung des Epsilon-Abbruch-Verfahrens ist, dass es den Optimierungsprozess gezielt unterstützt. Besonders im Anfangsstadium des Optimierungsprozesses, wo dieser bei Anwendung heuristischer Suchverfahren (siehe Abschn. 4.3.5) eher einer zufälligen Suche ähnelt, werden die Simulationsläufe mit „ungünstigen“ Ergebnissen vorzeitig beendet. Das Epsilon-Abbruch-Verfahren ist demnach bei sinnvoller Wahl der Verfahrensparameter eine hilfreiche Erweiterung im Hinblick auf die Verkürzung der Simulationsdauer im Rahmen eines Laufes der simulationsbasierten Optimierung.

3.7. Zufallszahlenerzeugung

Zufallszahlen werden bei der simulationsbasierten Optimierung sowohl bei der Simulation als auch bei der Optimierung verwendet. Bei der Simulation dienen sie im Wesentlichen dazu, eine Wahrscheinlichkeitsverteilung für das Auftreten zufälliger Ereignisse nachzubilden und somit die stochastischen Einflussgrößen im Simulationsmodell zu verankern. Die Verwendung von Zufallszahlen im Zusammenhang mit der Optimierung gestaltet sich etwas anders. Dort dienen sie insbesondere bei den heuristischen Suchverfahren (siehe Abschn. 4.3.5) dazu, den Optimierungsprozess voranzutreiben.

In diesem Abschnitt seien allerdings nur die wesentlichen Charakteristiken des Pseudozufalls genannt, um einen groben Überblick über das Thema Zufall, Zufallszahlengenerierung und den damit zusammenhängenden Wahrscheinlichkeitsverteilungen zu geben.

Pseudozufallszahlen. Wie bereits einleitend erwähnt, werden bei der Simulation zur Nachbildung der Wahrscheinlichkeitsverteilung zufälliger Simulationsereignisse *Zufallszahlen* benötigt. Hierbei können sowohl *echte Zufallszahlen* als auch *Pseudozufallszahlen* genutzt werden. Echte Zufallszahlen sind zumeist an spezielle Hardware geknüpft und ermöglichen keine Reproduzierbarkeit eines Simulationsexperimentes. Auf diese wird daher nicht näher eingegangen. Pseudozufallszahlen hingegen ermöglichen die *Reproduzierbarkeit eines Simulationslaufes* oder *-experimentes*, welche bei der Simulation neben der *Fehlersuche und -behebung* entscheidend für den *erneuten Gültigkeits- bzw. Optimalitätsnachweis* einmal erbrachter Simulationsergebnisse ist. Pseudozufallszahlen bezeichnen dabei *zufällig aussehende Zahlenfolgen*, die durch einen *deterministischen Algorithmus*, einen so genannten *Pseudozufallszahlengenerator* oder nur *Zufallszahlengenerator*, berechnet werden.

Pseudozufallszahlengeneratoren. Zufallszahlengeneratoren werden neben der innerhalb dieser Arbeit angewandten Simulation von stochastischen Systemen u. a.

auch bei der Kryptographie, gesicherten Kommunikationsverbindungen, Computerspielen sowie Online-Glücksspielen benötigt. Softwaretechnische Realisierungen für Zufallszahlengeneratoren können in *arithmetische Zufallszahlengeneratoren* und *rekursive arithmetische Zufallszahlengeneratoren* unterschieden werden. Arithmetische Zufallszahlengeneratoren basieren größtenteils auf Vielfachen des gebrochenen Teils irrationaler Zahlen, wodurch die Rechengenauigkeit begrenzt ist. Sie kommen daher nicht zum Einsatz. Rekursive arithmetische Zufallszahlengeneratoren hingegen basieren auf rationalen Zahlen. Ein solcher Zufallszahlengenerator wird durch die *Startwerte* $s_0, s_1, \ldots, s_{k-1}$ initialisiert. Die Startwerte werden auch als *Saat* [dt.; *seed* [engl.]] bezeichnet. Eine arithmetische Funktion $f_{rand} : \{0, \ldots, m-1\}^k \rightarrow \{0, \ldots, m-1\}$ erzeugt anschließend sukzessive die Werte $r_n := f_{rand}(r_{n-k}, r_{n-k+1}, \ldots, r_{n-1})$, wobei $n \geq k$ gilt. Danach können die $u_i = \frac{r_i}{m}$ als Zufallszahlen verwendet werden, wobei die u_i nach [LS07, S. 3f.] *Folgen von unabhängigen gleichverteilten*[43] *Zufallszahlen* im Intervall $[0, 1)$ darstellen sollten, welche in einem Rechner durch eine unabhängige Folge von Bits der Menge $\{0, 1\}$ repräsentiert werden.

Definition 3.4 (Folge von Zufallszahlen): Def. 3.4
Eine Folge von unabhängigen gleichverteilten Zufallszahlen sei als $u_0, u_1, u_2, \ldots$ *bezeichnet, welche jeweils eine konkrete Realisierung der Zufallsvariablen* $X \sim U(0, 1)$ *darstellen und* $U(0, 1)$ *eine Gleichverteilung im Intervall* $[0, 1)$ *ist.*
□

Zufallszahlen stellen demnach Werte aus der Menge $\left\{0, \frac{1}{m}, \frac{2}{m}, \ldots, \frac{m-1}{m}\right\}$ dar, wobei m eine hinreichend große natürliche Zahl sein sollte. Bei jedem Start des Zufallszahlengenerators mit dem gleichen seed wird demnach die *gleiche pseudozufällige Zahlenfolge* erzeugt. Außerdem sollte dabei ein Zufallszahlengenerator ein kleines und einfaches Rechnerprogramm sein, um schnell und einfach neue Pseudozufallszahlen zu erhalten (vgl. [LS07]).

Die Determiniertheit der Zufallszahlengeneratoren bedingt, dass eine Unabhängigkeit und Gleichverteilung der Folge von Zufallszahlen nicht gegeben ist, wodurch irgendwann eine Schleife erzeugt wird. Es gilt also $n_0 \in \mathbb{N}, p \in \mathbb{N}^*$ mit $r_{n+p} = r_n, \forall n \geq n_0$. Das kleinste positive p wird hierbei als *Periode* bezeichnet. Diese Länge der Periode ist neben der (scheinbaren) Unabhängigkeit der erzeugten Zufallszahlen ein weiterer Identifikator für die Güte eines Zufallszahlengenerators. Je länger eine Periode ist, umso längere Folgen von unabhängigen gleichverteilten Zufallszahlen können generiert werden.

Zur softwaretechnischen Umsetzung der zuvor betrachteten rekursiven arithmetischen Zufallszahlengeneratoren wurden verschiedene Varianten entwickelt, welche in Abb. 3.14 und Tab. 3.4 dargestellt sind. Diese Abbildung und die zugehörige Tabelle erheben dabei keinen Anspruch auf Vollständigkeit. Sie sollen jedoch einen für diese Arbeit ausreichenden Überblick über die Vielfältigkeit der vorhandenen

[43] Gleichverteilt bedeutet in diesem Zusammenhang, dass alle Werte in einem vorgegebenen Intervall mit der gleichen Wahrscheinlichkeit erzeugt werden.

Typen von Zufallszahlengeneratoren geben. Zusätzlich ist ein Ausgangspunkt gegeben, um sich gezielt tiefgründigere Informationen zum Thema der Zufallszahlengeneratoren einzuholen.

Abb. 3.14

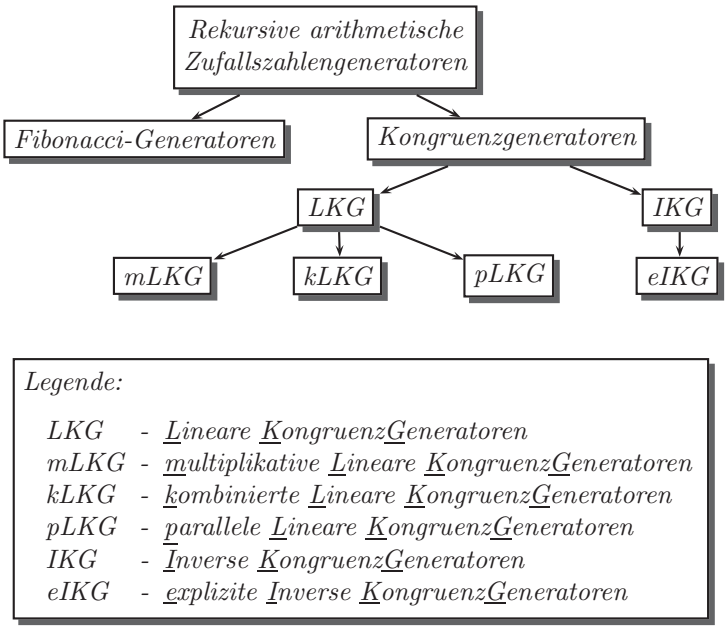


Abb. 3.14.: *Ausgewählte Typen von rekursiven arithmetischen Zufallszahlengeneratoren*

Zum Beweis, dass ein Zufallszahlengenerator tatsächlich eine Folge von (scheinbar) unabhängigen gleichverteilten Zufallszahlen zurückliefert, existieren verschiedene *empirische, statistische Testverfahren*. Auf diese wird u. a. in [Weg95] und [L'E96b] näher eingegangen.

Die verschiedenen Typen von Zufallszahlengeneratoren entstanden einerseits durch die Beseitigung der Probleme von älteren Zufallszahlengeneratoren (zu kurze Periodenlänge, Existenz von Hyperebenen[44], etc.) und andererseits durch die verschiedenen, anfänglich genannten, Anwendungsbereiche von Zufallszahlengeneratoren (Kryptographie, Glücksspiele, Simulation, etc.). Neben der Schaffung weiterer mathematischer Grundlagen im Bereich der Zufallszahlengeneratoren, wird es auch durch neuere Rechentechnik immer wieder möglich die Geschwindigkeit und die Periodenlänge von Zufallszahlengeneratoren zu erhöhen. L'Ecuyer schreibt dazu (siehe [LBC93, S. 2]):

> As computational power gets cheaper, increasingly long sequences of random numbers are used in applications. Generators with longer periods and which are more reliable are then necessary. One alternative is the class of multiple recursive generators (MRGs) ... based on the

[44] Als Ergebnis der Verallgemeinerung einer Ebene des dreidimensionalen Raumes auf den n-dimensionalen Raum erhält man eine Hyperebene. Diese ist demnach eine $(n-1)$-dimensionale Teilmenge eines n-dimensionalen Raumes. Dabei können die n Koordinaten der beschriebenen Hyperebene aus der reellen oder einer anderen Zahlenmenge stammen. Die Koordinaten können bspw. komplexwertig sein.

Tab. 3.4

Typ (siehe Abb. 3.14)	Beispiel mit Literaturverweis
klassische LKG	• `RANDU` (siehe [Leh51], [FG71, S. 15]) • `APPLE` (siehe [KD90]) • LKGs von Knuth, Borosh und Niederreiter (siehe [Knu81])
relevante LKG	• L'EcuyerLKGs (siehe [L'E88]) • `MAPLE` (`rand()`-Kommando des Programmes `Maple` (siehe [KG94])) • `CSHARP` (Zufallszahlengenerator der Programmiersprache C♯)
kLKG	• vier kLKGs (siehe [L'E96a], [L'E99]) • `NAG-PVM`-Bibliothek (mit 273 kombinierbaren MLKGs, wobei vier mLKGs stets kombiniert werden (siehe [Mac89])) • Qube (siehe [Sei06])
pLKG	[And90], [MZ96]
mLKG	• Kombination von 3 MLKGs (siehe [WH82]) • Grube-DieterMLKG (siehe [Gru73]) • L'EcuyerMLKGs (siehe [LBC93])
IKG	[EL86], [Nie95]
eIKG	[EH93]

Tab. 3.4.: *Typen von rekursiven arithmetischen Zufallszahlengeneratoren (Auswahl)*

higher order recursion.

Auf diese Aussage stützt sich auch die Implementierung der Zufallszahlengeneratoren im Softwaresystem `CAOS` (siehe Abschn. 5.3.4). Weitere Implementierungen und Implementierungsvorschläge sind auf den Internetseiten [ran07], [pLa07] oder [Fog07] zu finden.

Wahrscheinlichkeitsverteilungen. Eine *Wahrscheinlichkeitsverteilung* gibt die Verteilung der Wahrscheinlichkeiten auf die möglichen Zufallsergebnisse an. Nachfolgend wird als Kurzform für Wahrscheinlichkeitsverteilung nur noch von Verteilungen gesprochen. Bei real-existierenden Systemen können zufällige Simulationsereignisse oftmals nicht exakt abgebildet werden. Sie werden i. Allg. nur annähernd durch eine theoretische Verteilung dargestellt, welche der tatsächlichen Verteilung weitestgehend nahe kommt. Diese Verteilungen zur Nachbildung eines zufälligen Prozesses werden als Wahrscheinlichkeitsverteilung einer bestimmten Zufallsvariablen bezeichnet. Gebräuchlich ist, zwischen *diskreten* und *stetigen Verteilungen* für eine Zufallsvariable zu unterscheiden. Eine diskrete Verteilung besitzt *abzählbar viele Werte*. Daher wird auch von *Wahrscheinlichkeitsfunktion* oder *Zähldichte* gesprochen. Bei stetigen Verteilungen hingegen werden *Wahrscheinlichkeitsdichten* oder *Dichtefunktionen* verwendet. Die Wahrscheinlichkeiten lassen sich bei diesen Verteilungen durch *Integrale* berechnen.

Eine Wahrscheinlichkeitsverteilung kann durch *geeignete Transformation* der bereits zuvor beschriebenen Pseudozufallszahlen u_i (siehe Def. 3.4), welche mittels eines Pseudozufallszahlengenerators erzeugt wurden, nachgebildet werden. Dies geschieht auch in der entworfenen Klassenbibliothek `CAO` des Softwaresystems `CAOS` (siehe Abschn. 5.3.4).

Anzumerken sei an dieser Stelle noch, dass in der (betriebs-)wirtschaftlichen Praxis häufig auf das Hilfsmittel der Simulation verzichtet wird. Stattdessen wird die analytische Lösung eines eigentlich stochastischen Problems angestrebt. Es wird anhand von charakteristischen Kenngrößen der Verteilungen versucht, das Verhalten eines Realsystems nachzubilden. Diese Kenngrößen sind u. a. die Momente einer Verteilung, wobei das erste Moment den *Mittelwert* einer Verteilung und das zweite Moment die *Streuung* der Verteilung angibt. Dennoch ist die analytische Lösung bestimmter Verteilungsfunktionen nicht oder nur schwer möglich. Daher werden bewusst Verteilungen verwendet, welche durch ihre strukturellen Besonderheiten der Verteilungsfunktion die analytische Lösung eines Problems ermöglichen. Allerdings geben diese Verteilungsfunktionen die Realität mitunter nur ungenau wieder. Eine solche Vorgehensweise sollte daher bei Systemen mit stochastischen Einflüssen i. Allg. vermieden und der Einsatz von Simulation angestrebt werden. Dies ermöglicht es, zu weitestgehend genauen Ergebnissen zu gelangen und die „Verfälschung“ des tatsächlichen Systemverhaltens zu minimieren.

3.8. Zusammenfassung

In diesem Kapitel wurden ausgehend von bekannten Vor- und Nachteilen der Simulation, die innerhalb dieser Arbeit entscheidenden Gründe für die Wahl der Simulation als Lösungsverfahren aufgezeigt. Des Weiteren wurden verschiedene Simulationstypen und -zugänge mit deren jeweiliger Herangehensweise bei der Zeitfortschreibung vorgestellt. Die Entscheidung fiel dabei auf den ereignisgesteuerten Zugang bei der diskreten Simulation, welche im Wesentlichen in der Untersuchung diskreter Systeme begründet lag.

Ein weiterer Gesichtspunkt war die Gestaltung des Ablaufes eines Simulationsexperimentes bei der diskreten Simulation unter Anwendung der im Abschn. 2.4 vorgestellten rechnerinternen Modellierungsform.

Zudem ist in diesem Kapitel auf die Probleme bei der Bestimmung der Dauer der transienten und der stationären Phase hingewiesen worden. In diesem Zusammenhang spielten ausgewählte Methoden zur Bestimmung des stationären Zustandes bzw. des Endes der transienten Phase eine wesentliche Rolle.

Diese gaben wiederum den Ausschlag für die Betrachtung der Möglichkeiten zur Verkürzung der Dauer eines Simulationsexperimentes, welche mit möglichst geringem numerischen und kommunikativen Aufwand erfolgen sollte. Dabei sind auch die Grenzen der verteilten und parallelen Simulation deutlich aufgezeigt worden. Der Autor stellte eine neue heuristische Methode zur Verkürzung der Simulationsdauer eines Simulationsexperimentes im Hinblick auf die Anwendung der simulationsbasierten Optimierung vor. Innerhalb dieser wurde das Ziel der Verkürzung der Simulationsdauer mit einfachen Annahmen angestrebt und gezeigt, dass unter verschiedenen Annahmen für die geführten Untersuchungen hinreichend gute Ergebnisse zu erzielen sind. Diese Methode kann somit, nach evtl. entsprechenden

notwendigen Anpassungen, auch auf andere Simulatoren bzw. Simulationsmodelle übertragen werden.

Ein weiterer wichtiger Punkt bei der Simulation war die Erzeugung von Zufallszahlen bzw. Pseudozufallszahlen, welche für zufällige Simulationsereignisse, die bestimmten Wahrscheinlichkeitsverteilungen unterliegen, benötigt werden. Diese Erzeugung von Zufallszahlen spielt auch bei den im Kap. 4 vorgestellten heuristischen Suchverfahren eine tragende Rolle, um z. B. die evolutionären Vorgänge der Natur innerhalb eines Rechners nachzubilden.

Die Optimierung, der zweite wesentliche Bestandteil bei der simulationsbasierten Optimierung neben der Simulation, wird im nachfolgenden Kapitel näher betrachtet.

Kapitel 4. Optimierung

Inhalt

4.1	*Einführung*	*91*
4.2	*Exakte Lösungsverfahren*	*93*
4.2.1	*Methoden der lokalen nichtlinearen Optimierung*	*93*
4.2.2	*Methoden der globalen nichtlinearen Optimierung*	*94*
4.3	*Heuristische Suchverfahren*	*95*
4.3.1	*Abbruchkriterien*	*97*
4.3.2	*Deterministische Eröffnungsverfahren*	*98*
4.3.3	*Stochastische Eröffnungsverfahren*	*99*
4.3.4	*Deterministische Verbesserungsverfahren*	*99*
4.3.5	*Stochastische Verbesserungsverfahren*	*101*
4.3.6	*Stochastische Verbesserungsverfahren mit einer Lösung*	*103*
4.3.7	*Stochastische Verbesserungsverfahren mit mehreren Lösungen*	*109*
4.4	*Verfahren der mehrkriteriellen Optimierung*	*115*
4.5	*Methodiken zur Verkürzung der Optimierungsdauer*	*118*
4.5.1	*Hybride Optimierung*	*119*
4.5.2	*Verteilte und parallele Optimierung*	*122*
4.6	*Zusammenfassung*	*129*

4.1. Einführung

Die im Abschn. 2.3 vorgestellte Vorgehensweise der iterativen Simulation und Optimierung wird im Sinne der Optimierung auch als *rechnergestützte Optimierung*[45] bezeichnet. Eine detaillierte Betrachtung ausgewählter rechnergestützter Optimierungsverfahren soll in den nachfolgenden Unterabschnitten vorgenommen werden. Insbesondere wird hierbei näher auf die Optimierungsverfahren eingegangen, welche auf Grund ihrer Robustheit, weitestgehenden Problemunabhängigkeit und weit

[45] Die Lösungsmethode der sequentiellen Simulation und Optimierung hingegen wird im Sinne der Optimierung auch als *händische Optimierung* bezeichnet, weil der Nutzer die Werte für die Entscheidungsvariablen selbstständig verändert und nur evtl. nachgelagerte Ranking-and-Selektion-Methoden zur Bestimmung einer Rangfolge und Auswahl der besten Lösung aus der Lösungsmenge anwendet.

verbreiteten Anwendung innerhalb des im Kap. 5 vorgestellten Softwaresystems `CAOS` ihre Anwendung gefunden haben. Des Weiteren wird an den betreffenden Stellen auch auf Verfahren hingewiesen, welche auf Grund verschiedenster Ursachen weniger für eine universelle Anwendbarkeit auf verschiedene Optimierungsprobleme geeignet sind. Daher seien in diesem Kapitel auch die exakten Optimierungsverfahren erwähnt.

Die Einordnung von Optimierungsverfahren kann nach sehr verschiedenen Gesichtspunkten erfolgen. Ein sinnvolles und innerhalb dieser Arbeit verwendetes Kriterium sieht die *Einordnung nach der Verfahrensart* vor. Es wird dabei zwischen *exakten* und *heuristischen Verfahren* unterschieden. Deren, hinsichtlich dieser Arbeit, wesentliche Kriterien sind in Tabelle 4.1 gegenübergestellt.

Tab. 4.1

Kriterium	Exakte Lösungsverfahren	Heuristische Suchverfahren
Lösungsgüte	finden das globale Optimum oder die globale Pareto-optimale Menge	finden i. Allg. nur ein lokales Optimum oder die lokale Pareto-optimale Menge
Größe des Raumes der Entscheidungsvariablen	der (gesamte) gültige Raum der Entscheidungsvariablen wird durchsucht	nur (erfolgsversprechende) Teile des Raumes der Entscheidungsvariablen werden durchsucht
Optimierungsdauer	steigt exponentiell mit der Größe des Raumes der Entscheidungsvariablen	frei wählbar; sollte jedoch in Abhängigkeit der Größe des Raumes der Entscheidungsvariablen (sinnvoll) gewählt werden

Tab. 4.1.: *Unterscheidungsmerkmale bei exakten und heuristischen Optimierungsverfahren (Auswahl)*

Allen Optimierungsverfahren zur globalen Optimierung gemein ist, dass sie wiederholt nach einem bestimmten System lokale Extrema aufsuchen. Dies geschieht auch bei den nachfolgend betrachteten heuristischen Suchverfahren. Grund dafür ist die Struktur des Zielfunktionsraumes, welche bei den untersuchten Optimierungsproblemen i. Allg. nichtlinear ist. Gleiches gilt für die Größe und Struktur des zugrunde liegenden Raumes der Entscheidungsvariablen (siehe Abschn. 2.2.4 und Abschn. 2.2.5).

In den nachfolgenden Ausführungen sei zunächst von Optimierungsverfahren zur Lösung von SOP ausgegangen, so dass nur lokale und globale Optima von Bedeutung sind. Diese Optimierungsverfahren werden als *einkriterielle Optimierungsverfahren* bezeichnet und in den Abschnitten 4.2 und 4.3 etwas genauer betrachtet. Dem gegenüber stehen die *mehrkriteriellen Optimierungsverfahren* zur Lösung von MOP. Ein Überblick über ausgewählte mehrkriterielle Optimierungsverfahren zur Lösung von MOP ist im Abschn. 4.4 zu finden.

In diesem Kapitel werden auch Methodiken zur Verkürzung der Optimierungsdauer vorgestellt (siehe Abschn. 4.5). Dies sind zum einen die *verteilten und parallelen Optimierungsverfahren* und zum anderen die *hybriden Optimierungsverfahren*. Beide Methodiken versuchen auf unterschiedliche Art und Weise eine, im Vergleich zu den anderen Optimierungsverfahren, schnellere Bestimmung einer (sub)optimalen Lösung zu erreichen. Auf Grund dieser Tatsache, dass sie einen Lauf zur simulati-

onsbasierten Optimierung wesentlich verkürzen können, sind sie auch im Softwaresystem CAOS umgesetzt.

4.2. Exakte Lösungsverfahren

Für *Nichtlineare Optimierungsprobleme*, wie sie im Falle der simulationsbasierten Optimierung größtenteils vorliegen, gibt es kaum exakte Lösungsverfahren. Es bieten sich lediglich bei *Diskretisierung des Raumes der Entscheidungsvariablen* (siehe dazu Abschn. 2.2.6) verschiedene Verfahren an. Eines von ihnen ist das *Verfahren der Vollständigen Aufzählung* [dt.; *enumeration* [engl.]], welches alle Werte des Raumes der Entscheidungsvariablen untersucht (siehe bspw. [Ebl96]). Zur zeitlichen Beschränkung dieser zeitaufwändigen Methodik und somit auch zur Beschränkung des Raumes für die Werte der Entscheidungsvariablen, wurden das *branch-and-bound-Verfahren* [engl.; Verzweige und Beschränke [dt.]] (siehe [NM93]) sowie das *Schnittebenenverfahren* [dt.; *cutting planes method* [engl.]] (siehe [NW88]) entwickelt. Diese besitzen jedoch den Nachteil, das ihre Güte eng mit der Wahl der benötigten Heuristiken sowie der Größe des Raumes der Entscheidungsvariablen verknüpft ist. Für das Auffinden guter Schrankenheuristiken ist im Vorfeld eine gute Kenntnis über die Struktur des Raumes der Entscheidungsvariablen und des Raumes der Zielfunktionen erforderlich, welche bei der simulationsbasierten Optimierung jedoch für gewöhnlich nicht vorliegt, aber evtl. im Rahmen von Voruntersuchungen gewonnen werden kann.

Für die in dieser Arbeit untersuchten Modelle (siehe Abschn. 6.3) ist es auf Grund der mangelnden Kenntnis über die Struktur des Raumes der Zielfunktionen lediglich sinnvoll, die Vollständige Aufzählung auf einen stark eingeschränkten Bereich des Raumes der Entscheidungsvariablen anzuwenden, z. B. im Rahmen der Feinoptimierung eines seriell-hybriden Optimierers (siehe Abschn. 4.5.1). Darüber hinaus wird die Methode der Vollständigen Aufzählung teilweise auch nur zur Untersuchung der Struktur des Zielfunktionsraumes in der Umgebung der globalen/lokalen Optima angewandt.

4.2.1. Methoden der lokalen nichtlinearen Optimierung

Zu den klassischen mathematischen Verfahren der *nichtlinearen Optimierung ohne Nebenbedingungen* gehören neben den nicht weiter betrachteten gradientenbasierten Verfahren auch *ableitungsfreie Methoden* wie bspw.

- das *Intervallhalbierungsverfahren* (siehe [Wes00]), der *Goldene Schnitt* (siehe [Wal04]) und andere Verfahren der Liniensuche,
- das *Sekantenverfahren* (siehe [Wes00]) sowie
- das *Downhill-Simplex-Verfahren* (siehe [NM65]).

Diese Methoden bedingen viele Iterationen, sind aber (teilweise) relativ robust gegenüber Problemen bzgl. der Struktur des Zielfunktionsraums, z. B. kleine lokale

Extrema sowie Probleme bei der Bestimmung der Zielfunktionsgradienten.

Bereits durch die Hinzunahme von Nebenbedingungen zu einem bestehenden Optimierungsproblem sind die zuvor genannten Methoden zwar anwendbar, führen aber i. Allg. nicht zu den gesuchten lokalen Optima. Es müssen dann heuristische Optimierungsverfahren zur Suche im Zielfunktionsraum (siehe Abschn. 4.3) oder nachfolgende Methoden der globalen nichtlinearen Optimierung (siehe Abschn. 4.2.2) angewandt werden.

Anzumerken sei, dass der Grund, warum gradientenbasierte Verfahren nicht weiter betrachtet werden, darin liegt, dass für die untersuchten Simulationsmodelle eine Bestimmung der Gradienten der Ausgabewerte nicht möglich ist. Es gibt jedoch auch bei der simulationsbasierten Optimierung Verfahren wie die perturbation analysis [engl.], welche auf die Bestimmung des Gradienten abzielen. Allerdings sind hierfür bestimmte Voraussetzungen innerhalb des Simulationsmodells zu erfüllen (siehe [BS00]).

4.2.2. Methoden der globalen nichtlinearen Optimierung

Im Gegensatz zur lokalen Optimierung ist die globale Optimierung nichtlinearer Optimierungsprobleme mittels exakter Verfahren ein ungelöstes Problem der Mathematik. Es gibt praktisch keine bekannte Methode, bei deren Anwendung als Ergebnis die Werte erhalten werden können, welche mit absoluter Sicherheit oder auch nur großer Wahrscheinlichkeit die globalen Optima darstellen. Die einzige Möglichkeit, welche bei nichtlinearen Optimierungsproblemen anwendbar wäre, ist die Diskretisierung des Wertebereiches für die Entscheidungsvariablen (vgl. Abschn. 2.2.6), welche bspw. wie folgt lauten könnte:

$$\widetilde{x}_i \quad = \quad \left\lfloor \frac{x_i}{x_i^s} \right\rfloor \cdot x_i^s$$

mit

$$\begin{gathered} x_i^l \leq \widetilde{x}_i,\, x_i \leq x_i^u \quad ; \\ x_i^s > 0 \quad ; \\ x_i^l,\, \widetilde{x}_i,\, x_i,\, x_i^u,\, x_i^s \in \mathbb{R} \quad ; \end{gathered}$$

wobei

- x_i - i-ter Wert der Entscheidungsvariablen x (kann auch ein stetiger Wert sein),
- x_i^l - obere Schranke für x_i,
- x_i^u - untere Schranke für x_i,
- x_i^s - Schrittweite für die Werte von x_i und
- $\widetilde{x}_i$ - diskretisierter Wert von x_i.

Durch die Diskretisierung wird es möglich, eine endliche Menge mit abzählbar vielen Elementen zu erzeugen und das zuvor bereits erwähnte Verfahren der Vollständigen Aufzählung anzuwenden. Die Verfahren finden mit absoluter Sicherheit die globalen Optima, weil sämtliche gültige Lösungen des Lösungsraumes ermittelt und deren Lösungswerte bestimmt werden. Die Vorgehensweise ist jedoch sehr intensiv im Bezug auf die Rechenzeit, weil die zu untersuchende Menge exponentiell zur Dimension des Raumes der Entscheidungsvariablen anwächst. Eine solche Diskretisierung des Wertebereiches für die Entscheidungsvariablen, dort Parameterraum genannt, wird bspw. auch in [Kru01, S. 45ff.] angestrebt.

4.3. Heuristische Suchverfahren

Anders als exakte Lösungsverfahren bieten heuristische Suchverfahren, nachfolgend teilweise kurz *Heuristiken* genannt, keine Garantie dafür, dass ein globales Optimum gefunden oder als solches erkannt wird. Heuristiken können i. Allg. ein einmal gefundenes globales Optimum mit gewisser Wahrscheinlichkeit wieder verlassen und ab dann nur noch schlechtere Zielfunktionswerte bestimmen. Daher sollten diese sich stets den bisher besten Zielfunktionswert merken, um diesen ggf. wieder als Ausgangspunkt für weitere Suchrichtungen annehmen zu können. Die heuristischen Suchverfahren haben gegenüber exakten Lösungsverfahren den Vorteil, dass sie Vorgehensregeln beinhalten, welche auch für verschiedene Typen von Optimierungsproblemen (siehe Abschn. 2.2.6) sinnvoll und erfolgsversprechend sind. Aus diesem Grunde sind heuristische Suchverfahren heutzutage weit verbreitet (siehe [PK00]).

Allgemein kann bei den heuristischen Suchverfahren zwischen zwei grundlegenden Paradigmen der Suche nach möglichst guten Zielfunktionswerten zur hinreichend guten Lösung eines Optimierungsproblems unterschieden werden:

Heuristische Optimierungsverfahren zur Suche im Raum der Entscheidungsvariablen.
Diese Verfahren zerlegen komplexe Optimierungsprobleme sukzessive in einfachere Optimierungsprobleme, bis schließlich auf der „Blattebene“ dieser Zerlegungshierarchie eine zufrieden stellende Lösung gefunden wird (vgl. bspw. *Best-First-Search-Verfahren* der Künstlichen Intelligenz (sieh [Cor03])). Sie finden innerhalb dieser Arbeit keine weitere Beachtung, weil keine allgemein gültigen Aussagen über die Zerlegbarkeit eines Optimierungsproblems bei der simulationsbasierten Optimierung getroffen werden können.

Heuristische Optimierungsverfahren zur Suche im Zielfunktionsraum.
Diese Verfahren, zumeist noch in *lokale* und *globale heuristische Suchverfahren* unterteilt, versuchen, einen zufällig oder definiert berechneten, gültigen Ausgangsvektor für die Werte der Entscheidungsvariablen zu finden (*Eröffnungsverfahren*), welcher anschließend durch zufällige oder definierte Suche in neue Werte für die Entscheidungsvariablen zu transformieren ist, welche bessere Zielfunktionswerte zurückliefern (*Verbesserungsverfahren*). Diese können

ihrerseits wiederum Ausgangsvektoren bilden, um abermals eine Verbesserung durchzuführen. Diese iterative Vorgehensweise terminiert infolge des Erreichens eines vorgegebenen Abbruchkriteriums (siehe dazu Abschn. 4.3.1). Weitere Verfahren zur Suche im Zielfunktionsraum stellen die so genannten *Gesamtschrittverfahren* dar, welche von Beginn an eine sehr gute zulässige (optimale) Lösung liefern. Gesamtschrittverfahren sind, genau wie die Eröffnungsverfahren, jedoch problemspezifische Verfahren und können somit nur auf bestimmte Klassen von Optimierungsproblemen[46] angewendet werden. Die Einteilung heuristischer Optimierungsverfahren zur Suche im Zielfunktionsraum ist in Abb. 4.1 noch einmal zusammenfassend graphisch dargestellt.

Abb. 4.1

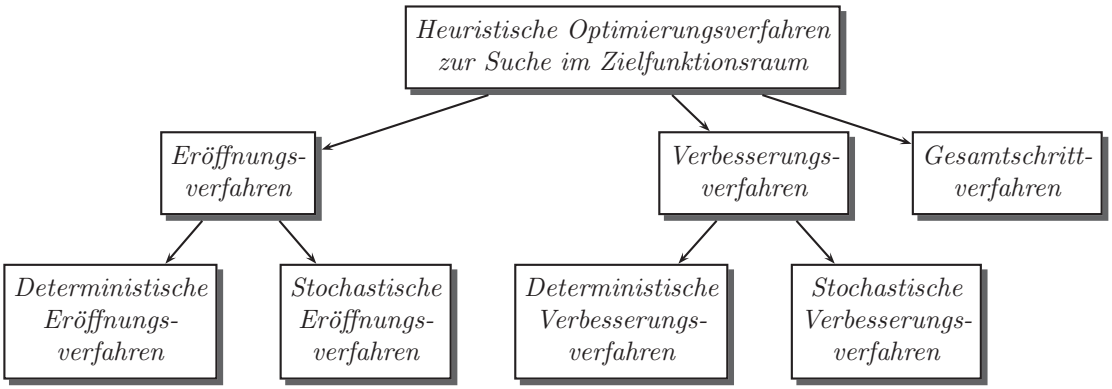


Abb. 4.1.: *Einteilung heuristischer Optimierungsverfahren zur Suche im Zielfunktionsraum*

Die Eröffnungsverfahren werden auch als *Konstruktionsheuristiken* bzw. *-verfahren* bezeichnet. Sie sind meist sehr problemspezifisch angelegt, häufig günstig bzgl. der Rechenzeit sowie dem Speicherplatz und i. Allg. einfach aufgebaut, um schnell gültige Zielfunktionswerte zu erhalten. Eröffnungsverfahren finden üblicherweise nur ein lokales Optimum und sind meist anfällig gegenüber auftretenden Nebenbedingungen, so dass sie nicht zwangsläufig immer gültige Zielfunktionswerte hervorbringen. Verbesserungsverfahren oder *Verbesserungsheuristiken* verwenden i. d. R. *lokale Suchverfahren* oder *Metaheuristiken*, welche sich problemunabhängig gestalten lassen. Sie werden benötigt, um bessere Zielfunktionswerte zu ermitteln.

Wie bereits angedeutet, werden innerhalb dieser Arbeit nur Verfahren zur Suche im Zielfunktionsraum betrachtet, weil für Verfahren zur Suche im Raum der Entscheidungsvariablen nur bedingt allgemein gültige Aussagen getroffen werden können. Von dieser allgemeinen Gültigkeit wird aber in der vorliegenden Arbeit ausgegangen.

[46] Eine bekannte Klasse von Optimierungsproblemen ist bspw. das TSP (siehe [Rei94]).

4.3.1. Abbruchkriterien

Die Abbruchkriterien legen fest, ob die *Optimierung beim Eintreten eines bestimmten Ereignisses*[47] *beendet* werden soll. Bei den Abbruchkriterien kann prinzipiell zunächst zwischen

- *allgemeingültigen* bzw. *globalgültigen* und
- *optimiererspezifischen* bzw. *lokalgültigen*

Abbruchkriterien unterschieden werden. Die allgemeingültigen Abbruchkriterien sind dabei unabhängig von dem jeweils verwendeten Optimierungsverfahren. Abbruchkriterien können weiterhin nach dem Typ des eingetretenen Ereignisses in

- *zeitabhängige* und
- *zustandsabhängige*

Abbruchkriterien unterteilt werden (siehe Abb. 4.2 und Tab. 4.2). Infolge der Nutzung graphischer Oberflächen kommen noch *interaktive Abbruchkriterien* hinzu. Diese können als eine Vereinigung der zuvor genannten Abbruchkriterien angesehen werden, weil dessen Eintreten durch die Reaktion des Nutzer auf den aktuellen Optimierungsverlauf und dessen Zufriedenheit mit diesem Optimierungsverlauf hervorgerufen wird.

Abb. 4.2

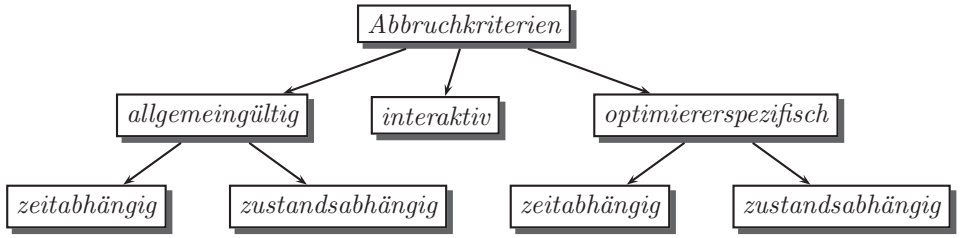


Abb. 4.2.: *Einteilungsmöglichkeit von Abbruchkriterien*

Problematisch bei den zeitabhängigen Abbruchkriterien erweist sich der Fakt, dass die Geschwindigkeit, mit der Berechnungen bzw. Simulationen bei der simulationsbasierten Optimierung durchgeführt werden, direkt von der Prozessorgeschwindigkeit abhängt. Demnach ist bei der Berechnungsdauer eigentlich die virtuelle Rechenzeit ct_{virt} entscheidend und nicht die wahre Realzeit ct_{real} mit

$$ct_{virt} \quad = \quad c_{cpu} \cdot ct_{real} \quad ,$$

wobei

c_{cpu} - Normalisierungskonstante abhängig von der Prozessorarchitektur, u. Ä.

[47] Im Gegensatz zu den in Abschn. 3.3 betrachteten Simulationsereignissen sind an dieser Stelle Optimierungsereignisse gemeint, welche z. B. von einem bestimmten Optimierungszeitpunkt abhängig sind.

Tab. 4.2

Charakteristik	Symbol	Verbale Beschreibung des Abbruchkriteriums
Zeitabhängig	SC_T	die Optimierung wird nach einer bestimmten *Optimierungsdauer* beendet
	$SC_{\Delta T}$	die Optimierung wird beendet, wenn nach einer bestimmten *Optimierungsdauer keine bessere Lösung* gefunden wurde bzw. keine Änderung der Pareto-optimalen Menge auftrat
	$SC_{\Delta T,\epsilon}$	wie $SC_{\Delta T}$, nur dass zusätzlich eine *Epsilon-Umgebung für die bisher beste Lösung* definiert ist, außerhalb der sich die Verbesserung befinden muss
Zustandsabhängig	SC_S	die Optimierung wird nach einer bestimmten *Anzahl von Optimierungsschritten (generierten Lösungen)* beendet
	$SC_{\Delta S}$	die Optimierung wird beendet, wenn nach einer bestimmten *Anzahl von Optimierungsschritten keine bessere Lösung* gefunden wurde bzw. keine Änderung der Pareto-optimalen Menge auftrat
	$SC_{\Delta S,\epsilon}$	wie $SC_{\Delta S}$, nur dass zusätzlich eine *Epsilon-Umgebung für die bisher beste Lösung* definiert ist, außerhalb der sich die Verbesserung befinden muss

Tab. 4.2.: *Allgemeingültige Abbruchkriterien*

Zeitabhängige Abbruchkriterien sollten daher nur dann eingesetzt werden, wenn die Nutzung eines Rechners zeitlich begrenzt ist, oder die bisher besten Zielfunktionswerte des aktuellen Optimierungslaufes nach einer bestimmten Zeit als globales Optimum angenommen werden, ohne deren Optimalität nachgewiesen zu haben. Bei Anwendung von zeitabhängigen Abbruchkriterien sind im Vorfeld stets Voruntersuchungen notwendig, um einen Vergleichsfaktor zwischen verschiedenen Rechnerarchitekturen zu bestimmen. Bei den Voruntersuchungen ist darauf zu achten, dass die Besonderheiten bestimmter Rechnerarchitekturen, wie bspw. pipelining [engl.], multitasking [engl.], hyperthreading [engl.], u. a., keinen oder nur einen geringen Einfluss auf das Ergebnis der Voruntersuchungen nehmen, weil deren Verhalten während eines Optimierungslaufes der simulationsbasierten Optimierung mitunter nicht vorhergesagt werden kann.

Auf die optimiererspezifischen Abbruchkriterien wird in den einzelnen Abschnitten mit den jeweiligen Optimierungsverfahren etwas näher eingegangen (siehe Abschn. 4.3.6 und 4.3.7), um so den direkten Bezug zum jeweiligen Optimierungsverfahren herzustellen.

4.3.2. Deterministische Eröffnungsverfahren

Wie bereits erwähnt, ist auf Grund des problemspezifischen Charakters der deterministischen Eröffnungsverfahren eine Einteilungsmöglichkeit der vorhandenen Eröffnungsverfahren nur für spezielle Klassen von Optimierungsproblemen möglich. Einige ausgewählte Eröffnungsverfahren sind hierfür in Tabelle 4.3 zusammengestellt, um deren Problemspezifik zu verdeutlichen.

Eröffnungsverfahren bieten einen *guten heuristischen Ausgangswert*, welcher z. B. als Möglichkeit des Gütevergleichs mit anderen Eröffnungsverfahren oder als Start-

Tab. 4.3

Problemtyp	Problemklasse	Name des Eröffnungsverfahrens	Literaturverweis(e)
Permutation	TSP	*Heuristik von Christofides*	[Chr76], [Akl80]
	VRP	*Sweep-Algorithmus*	[GM74]
		Spacefilling-Curves-Verfahren	[BP82]
		Clarke-and-Write-Saving	[AG88, §127-147]
	Scheduling	*Shifting-Bottleneck-Algorithmus*	[ABZ88], [PM00]
		Prioritätsregeln	[Kön01]

Tab. 4.3.: *Deterministische Eröffnungsverfahren (Auswahl)*

wert für stochastische Verbesserungsverfahren dienen kann. In Abb. 4.3 ist dazu ein Algorithmus für eine allgemein gültige Vorgehensweise zur Bestimmung eines definierten Ausgangswertes vorgestellt. Erkennbar ist u. a., dass mehrere Startvektoren und Eröffnungsverfahren vom Anwender vorgegeben werden können. Darüber hinaus wird die zufällige Bestimmung der Werte für die Entscheidungsvariablen zumeist bei den heuristischen Verbesserungsverfahren genutzt (siehe Abschn. 4.3.5). Im Falle, dass ein Algorithmus mehrere definierte Ausgangswerten benötigt, ist dieser Algorithmus entsprechend mehrmals zu wiederholen. Grundlage des abgebildeten Algorithmus bilden dabei ein oder mehrere implementierte Eröffnungsverfahren.

4.3.3. Stochastische Eröffnungsverfahren

Prinzipiell sind unter stochastischen Eröffnungsverfahren Methoden zu verstehen, welche die benötigten *Werte für die Entscheidungsvariablen zufällig* bestimmen. Aus diesem Grunde kann auch von *Monte-Carlo-Methoden* gesprochen werden. In Abb. 4.3 ist die Vorgehensweise von stochastischen Eröffnungsverfahren in der Aktivität „Werte für die Entscheidungsvariablen zufällig bestimmen“ zusammengefasst. Der Wert für eine einzelne Entscheidungsvariable x_i wird dabei zufällig gleichverteilt aus Intervall $[x_i^l, x_i^u]$ bestimmt.

4.3.4. Deterministische Verbesserungsverfahren

Die deterministischen Verbesserungsverfahren versuchen einen vorgegebenen Zielfunktionswert durch Veränderung der zugehörigen Werte für die Entscheidungsvariablen sukzessive zu verbessern. Dies kann nach Ansicht des Autors durch *sequentielle* oder *parallele Anwendung vorgegebener Verbesserungsverfahren* geschehen. Bei der sequentiellen Vorgehensweise wird eine Liste der abzuarbeitenden Verbesserungsverfahren vorgegeben. Diese Liste kann fest definiert oder zufällig aus der Menge der vorhandenen Verbesserungsverfahren bestimmt sein. Im Falle der parallelen Anwendung wird die Menge der zu verwendenden Verbesserungsverfahren bestimmt, welche als Eingabe dieselbe Ausgangslösung bekommen und ihre jeweilige Verbesserung zurückgeben. Anschließend wird bspw. die beste Verbesserung ausgewählt und ausgehend von dieser eine erneute Verbesserung durch parallele

Abb. 4.3

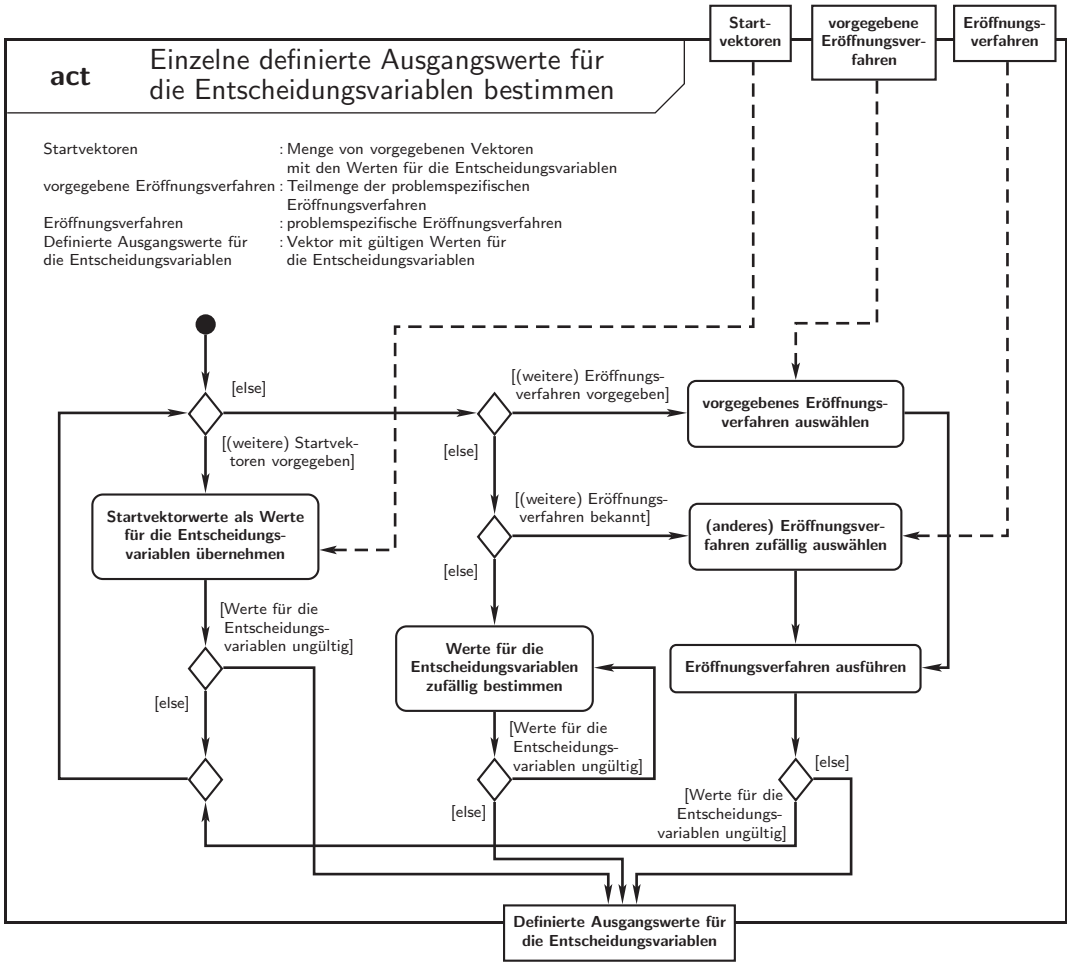


Abb. 4.3.: *Algorithmus zur Bestimmung einer definierten Ausgangslösung (UML-Aktivitätsdiagramm)*

Anwendung der Verbesserungsverfahren versucht. Es ist sinnvoll, sowohl die sequentielle, als auch die parallele Verfahrensvariante terminieren zu lassen, wenn keine oder keine hinreichend gute Verbesserung für den Zielfunktionswert mehr gefunden wird.

Tab. 4.4

Parameter-typ(en)	**Problem-bezeichnung**	**Bekannte Heuristik**	**Literatur-verweis(e)**
Permutation	TSP	*r-opt-Verfahren*	[Lin65]
		variables r-opt-Verfahren	[LK73]
		Or-opt-Verfahren	[Or76]
Permutation/ reellwertig	allgemein-gültig	*Verfahren der Variablenweisen Aufzählung*[48]	[Sch07a]

Tab. 4.4.: *Deterministische Verbesserungsverfahren (Auswahl)*

[48] Das Verfahren der Variablenweisen Aufzählung ist ein heuristisches Verfahren, welches an das Verfahren der Vollständigen Aufzählung angelehnt, jedoch günstiger als dieses Verfahren ist.

4.3.5. Stochastische Verbesserungsverfahren

Ausgehend von einem oder mehreren zufällig oder definiert erzeugten Startvektoren für die Werte der Entscheidungsvariablen sind mehrere Methoden zur Erzeugung eines oder mehrerer neuer Vektoren für diese Werte bekannt. Diese weiteren Methoden der nichtlinearen globalen Optimierung werden auch als *moderne heuristische Optimierungsverfahren* bezeichnet und besitzen verschiedene Merkmale zu ihrer Kategorisierung (siehe Tab. 4.5).

Tab. 4.5

Charakteristik	Mögliche Werte
Anzahl paralleler Vektoren mit Werten für die Entscheidungsvariablen	• 1 • n
Verfahrensart	• lokal • global
Innerhalb dieser Arbeit betrachtete Optimierungsproblemtypen, auf welche die Verfahren anwendbarer sind	• reellwertige Probleme • Permutationsprobleme
Ersetzungsschema bei n Vektoren mit Werten für die Entscheidungsvariablen	• alle Lösungen (Generationsprinzip) • einzelne Lösungen (stetige Ersetzung [dt.; steady state [engl.]])
Elitismus bei Generationsprinzip	• vorhanden • nicht vorhanden
Mehrkriterielle Verfahrensvariante	• existiert • existiert nicht

Tab. 4.5.: *Unterscheidungskriterien moderner heuristischer Verfahren*

Unterscheidung zwischen lokalen und globalen Optimierungsverfahren. Oftmals wird behauptet, dass die Anzahl parallel verwendeter Lösungen direkten Einfluss auf die gewählte Verfahrensart hat. Ein einzelner Vektor wird dabei mit einem lokalen Optimierungsverfahren und mehrere Vektoren mit globalen Optimierungsverfahren in direkte Verbindung gebracht. Allerdings können die meisten Optimierungsverfahren in Abhängigkeit ihrer Einstellungen für die Verfahrensparameter sowohl als lokale wie auch als globale Optimierungsverfahren auftreten. Der Autor hält für die meisten Optimierungsprobleme eine hybride Vorgehensweise für sinnvoll, indem zunächst ein Verfahren mit entsprechenden Verfahrensparametereinstellungen als globales Optimierungsverfahren und anschließend ein anderes oder dasselbe Verfahren mit entsprechenden Verfahrensparametereinstellungen als lokales Optimierungsverfahren arbeitet. Diese Vorgehensweise kann allerdings auch unabhängig von der Anzahl parallel verwendeter Vektoren angewandt werden. Es empfiehlt sich aber oftmals mit parallel verwendeten Vektoren zu arbeiten, um verschiedene Teile des Raumes der Entscheidungsvariablen abzudecken. Diese Vorgehensweise ähnelt der Vorgehensweise bei der hybrid sequentiellen Optimierung, welche i. d. R. zwei verschiedene Verfahren nacheinander ausführt. Ziel ist es dabei die Stärken der Optimierungsverfahren zu nutzen und deren Schwächen zu umgehen. Die Schwierigkeit besteht jedoch zumeist darin, den richtigen Zeitpunkt der Umschaltung vom globalen Optimierungsverfahren zum lokalen Optimierungsver-

fahren zu bestimmen. Diese Problematiken und andere Problematiken der hybrid arbeitenden Optimierungsverfahren seien jedoch im Abschn. 4.5.1 etwas näher betrachtet.

Der Übergang von einem „global suchenden“ zu einem „lokal suchenden“ Optimierungsverfahren tritt u. a. auch bei selbst-adaptiven oder lernfähigen Optimierungsverfahren auf, welche meist in Kombination mit Neuronalen Netzen auftreten (siehe [Mey03]). Diese beginnen ihre Suche noch einer (sub)optimalen Lösung zunächst als globales Optimierungsverfahren und lernen während des Optimierungsverlaufes günstige Einstellungen für ihre Verfahrensparameter, um so i. d. R. zu einem lokalen Optimierungsverfahren überzugehen.

In Abb. 4.4 ist ein Überblick über ausgewählte stochastische Verbesserungsverfahren unter Beachtung der Anzahl parallel verwendeter Lösungen gegeben. Die Abkürzungen für die Verfahren sind in Tab. 4.6 und Tab. 4.9 zu finden. Die hohe Anzahl der Verfahren lässt auf eine weite Verbreitung und Nutzung stochastischer Verbesserungsverfahren schließen.

Abb. 4.4

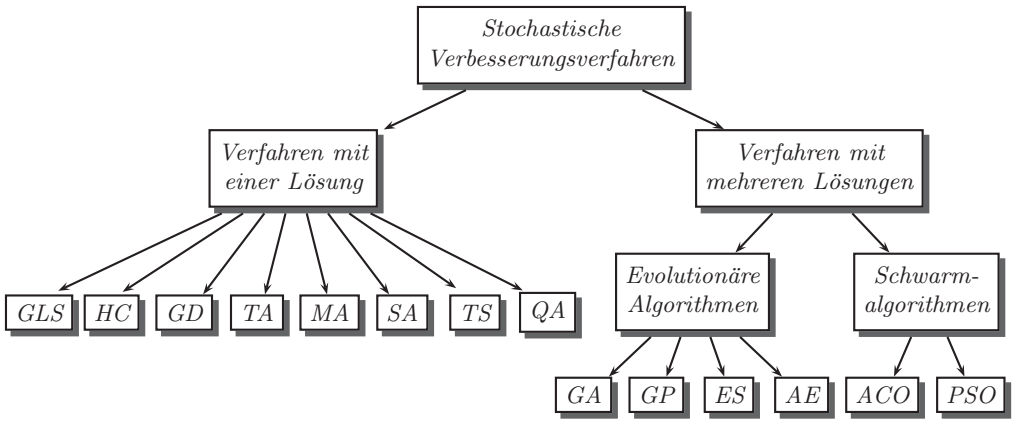


Abb. 4.4.: *Überblick über heuristische, stochastische Verbesserungsverfahren (Auswahl)*

Allgemein kann festgehalten werden, dass, je robuster ein stochastisches Verbesserungsverfahren gegenüber der Struktur des Zielfunktionsraumes ist, desto unbedeutender wird die Bestimmung einer oder mehrerer guter Startvektoren durch ein definiertes Eröffnungsverfahren. Für einen Großteil der praktischen Problemstellungen reicht zumeist die Bestimmung zufällig gewählter Startvektoren aus. Erschwerend kommt zudem hinzu, dass sich für die meisten definierten Eröffnungsverfahren der Umgang mit vorhandenen Nebenbedingungen als problematisch erweist, wodurch oftmals ungültige Startvektoren entstehen. Zur Behebung dieses Problems könnte man versuchen, die ungültigen Startvektoren durch *Korrekturverfahren* in gültige Startvektoren umzuwandeln, oder die benötigten Werte für die Entscheidungsvariablen zufällig zu bestimmen. Eine andere Möglichkeit ist die *Vergabe von Straftermen* (siehe [RR05]).

4.3.6. Stochastische Verbesserungsverfahren mit einer Lösung

In Tab. 4.6 sind ausgewählte moderne stochastische Verbesserungsverfahren mit den entsprechenden Literaturverweisen angeführt, um einen weitreichenden Überblick über die Vielfältigkeit der verfügbaren Heuristiken zu geben. Dabei sind weitere zu den in Abb. 4.4 dargestellten Verfahren aufgeführt.

Tab. 4.6

Verfahrens-bezeichnung	engl. Verfahrensbezeichnung	Literaturver-weise
Geführte lokale Suche	Guided Local Search (kurz: GLS)	[Vou97]
gierige, zufällige, adaptive Suchprozeduren	Greedy Randomized Adaptive Search Procedures (kurz: GRASP)	[RR02]
Bergsteigeralgorithmus	Hill Climbing algorithm (kurz: HC)	[RN95, Kap. 4], [YM93]
Zufälliger Bergsteigeralgorithmus	Stochastic Hill Climbing algorithm (kurz: SHC)	[Spa03]
Sintflutalgorithmus	Great Deluge algorithm (kurz: GD)	[Due06]
Schwellenakzeptanz	Threshold Accepting (kurz: TA)	[Bog98]
Metropolisalgorithmus	Metropolis Algorithm (kurz: MA)	[MRT53]
Simulierte Abkühlung	Simulated Annealing (kurz: SA)	[KGV83], [Cer85], [DC05]
Tabusuche	Tabu Search (kurz: TS)	[Glo89], [Glo90], [GL93], [GL97]
Quantum Abkühlung	Quantum Annealing (kurz: QA)	[DC05]

Tab. 4.6.: *Überblick über ausgewählte moderne heuristische Verfahren (eine (parallele) Lösung)*

In der vorliegenden Arbeit wurden zum Vergleich untereinander sowie der weiteren Nutzung innerhalb hybrider Optimierungsverfahren (siehe Abschn. 4.5.1) nur die Tabusuche und das Verfahren der Simulierten Abkühlung als Verfahren mit einer Lösung zur Implementierung im Softwaresystem `CAOS` genutzt (siehe Abb. 4.5). Daher sollen diese im Folgenden allgemein kurz vorgestellt werden. Ihre spezielle Implementierung innerhalb des Softwaresystems `CAOS` wird im Abschn. 5.3.6 beschrieben, um eine logische Trennung des allgemeinen Verfahrens von seiner konkreten Implementierung zu besitzen.

Abb. 4.5

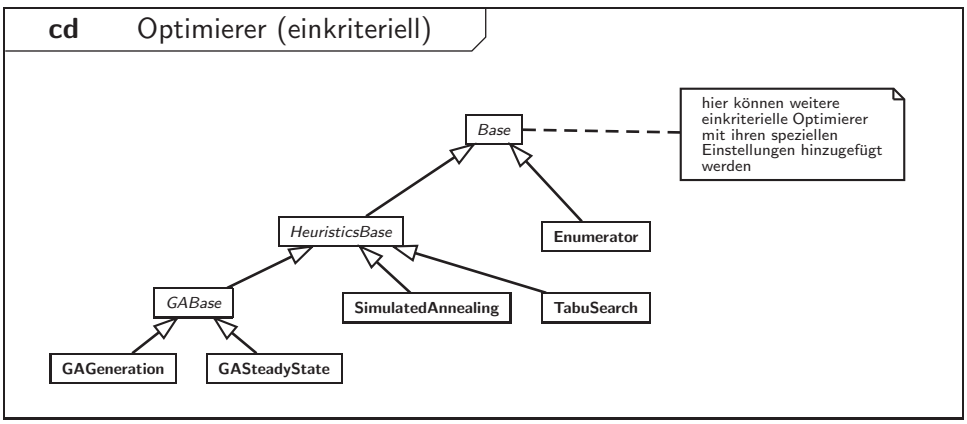


Abb. 4.5.: *Einkriterielle Optimierer des CAOS (UML-Klassendiagramm)*

Für die nachfolgenden Verfahren seien folgende Bezeichnungen für die Vektoren mit den Werten für die Entscheidungsvariablen vereinbart:

$\vec{x}_{start}$	-	Vektor mit den Startwerten für die Entscheidungsvariablen (Startvektor), welcher einen zunächst unbekannten Zielfunktionswert ergibt,
$\vec{x}_{cur}$	-	Vektor mit den Werten für die Entscheidungsvariablen, welcher die aktuellen Zielfunktionswerte ergibt,
$\vec{x}_{new}$	-	Vektor mit den Werten für die Entscheidungsvariablen, welcher die neuen Zielfunktionswerte ergibt,
$\vec{x}_{best}$	-	Vektor mit den Werten für die Entscheidungsvariablen, welcher den aktuell besten Zielfunktionswert des Optimierungsverfahrens ergibt sowie
$N(\vec{x}_{cur})$	-	Nachbarschaft von $\vec{x}_{cur}$.

Simulierte Abkühlung (SA). Der Begriff der Simulierten Abkühlung stammt aus der Thermodynamik. Um Kristalle zu erzeugen, werden dort verschiedene Materialien stark erhitzt und anschließend langsam abgekühlt, bis sie zu den gewünschten Kristallstrukturen erstarren, welche ein bestimmtes Energieniveau besitzen. Metropolis et. al. formulierten dazu den sog. Metropolis-Algorithmus (siehe [MRT53]). Dieser wird bei der Simulierten Abkühlung auf ein Optimierungsproblem übertragen, wobei im Falle eines Minimierungsproblems[49] das globale Minimum dem Zustand mit dem niedrigsten Energieniveau (Grundzustand) entspricht. Das Energieniveau entspricht dem berechneten Zielfunktionswert.

Bei der Simulierten Abkühlung wird zunächst der Startvektor $\vec{x}_{start}$ zufällig oder mittels eines Eröffnungsverfahrens definiert bestimmt. Dieser wird als aktueller Vektor $\vec{x}_{cur}$ angenommen. Im weiteren Verlauf des Verfahrens wird immer ein neuer Vektor $\vec{x}_{new}$ aus dem mutierten, aktuellen Vektor untersucht und die daraus entstehende Lösung mit der Wahrscheinlichkeit p als aktuelle Lösung akzeptiert, falls deren Zielfunktionswert schlechter als der von $\vec{x}_{cur}$ ist. Anderenfalls wird die Lösung verworfen. Ist die Lösung besser, wird sie stets als aktuelle Lösung akzeptiert. Somit werden nicht zwangsläufig nur besser bewertete Lösungen für den nächsten Iterationsschritt des Optimierungsverfahrens ausgewählt, sondern mit gewisser Wahrscheinlichkeit auch „geringfügig“ schlechtere. Dadurch soll eine vorzeitige Konvergenz gegen lokale Optima verhindert werden. Die Wahrscheinlichkeit mit der eine neue, schlechtere Lösung akzeptiert wird, beträgt:

$$P_{SA} = e^{-\frac{\Delta f}{\theta}} \tag{4.1}$$

[49] Bei Maximierungsproblemen ist der Zielfunktionswert entsprechend zu negieren, um den Algorithmus unverändert anwenden zu können.

mit

$$\Delta f \quad = \quad |f(\vec{x}_{cur}) - f(\vec{x}_{new})| \quad ,$$

wobei

Δf	-	Betrag der Differenz der Zielfunktionswerte aus aktueller und neuer Lösung sowie
θ	-	Temperatur, welche die Relevanz der relativen Evaluierungswerte angibt.

Aus Gleichung 4.1 wird erkennbar, dass je größer der Wert für $\frac{\Delta f}{\theta}$ ist, desto geringer wird die Wahrscheinlichkeit, mit der $\vec{x}_{new}$ akzeptiert wird und umgekehrt. Ein Anwender kann allerdings lediglich den Verfahrensparameter θ beeinflussen. Daher beginnt das Verfahren für gewöhnlich zuerst mit großen Werten für θ, was einer zufälligen Auswahl und damit einer eher *explorativen Suche*[50] gleicht. Der Parameter θ wird dann sukzessive verkleinert, was wiederum bedeutet, dass besser evaluierte Lösungen ($\frac{\Delta f}{\theta}$ klein) mit hoher Wahrscheinlichkeit gewählt werden (*exploitative Suche*[51]).

Zusammenfassend kann gesagt werden, dass das Verfahren der Simulierten Abkühlung anfangs beliebig schlechte Lösungen für $\vec{x}_{new}$ aus dem Raum der Entscheidungsvariablen $\mathcal{X}$ herausgreift und gegen Verfahrensende ($p \rightarrow 1$) nur noch lokal, innerhalb einer eingeschränkten Nachbarschaft, nach den optimalen Lösungen gesucht wird. Durch diese Vorgehensweise deckt das Verfahren einen großen Bereich des Suchraums für die Werte der Entscheidungsvariablen ab. Das Verfahren konvergiert gegen ein globales Optimum, wenn die Verkleinerung des Wertes θ (Abkühlung) langsam genug erfolgt, was jedoch den Optimierungsaufwand i. Allg. deutlich erhöht.

Spezifische Abbruchkriterien und Verfahrensparameter. Neben den in Tab. 4.2 aufgezeigten Abbruchkriterien, existiert bei der Simulierten Abkühlung ein weiteres Abbruchkriterium, das äußere Abbruchkriterium für die Schleife mit der wiederholten Temperaturabsenkung. Dieses Abbruchkriterium orientiert sich an der aktuell erreichten Temperatur θ des Abkühlungsvorgangs. Ist die Temperatur unter eine minimale Temperatur θ_{min}, mit $0 \leq \theta_{min} \leq \theta_{start}$ gefallen ($\theta < \theta_{min}$), wird das Verfahren der Simulierten Abkühlung beendet. Das Abbruchkriterium sei mit SC_θ bezeichnet.

Das Verfahren der Simulierten Abkühlung kann auf vielfältige Art und Weise gestaltet werden. Aus diesem Grund existieren auch verschiedenste Verfahrensparameter. Einige der Wesentlichen sind in Tab. 4.7 angegeben.

[50] Die explorative Suche bezeichnet eine globale Suche.
[51] Die exploitative Suche bezeichnet eine lokale Suche.

Tab. 4.7

Symbol	Verfahrensparame-ternname	Bedeutung
θ_{min}	minimale Temperatur	gibt die untere Grenze für θ an; bei Erreichen dieser terminiert das Verfahren
θ_{start}	Anfangstemperatur	anfängliche Temperatur des Verfahrens
r_{SA}	lokaler Wiederholungs-faktor	gibt an wie lange auf einem bestimmten Temperaturni-veau nach besseren Lösungen gesucht werden soll
ϵ_{SA}	Toleranzbereich	gibt den Grad der Absenkung der Temperatur θ zwi-schen zwei Temperaturniveaus an

Tab. 4.7.: *Verfahrensparameter des Optimierungsverfahrens der Simulierten Abkühlung*

Verfahrenserweiterungen. In Anlehnung des beschriebenen klassischen Verfahrens der Simulierten Abkühlung wurden in der Literatur einige *Verfahrenserweiterungen* untersucht. Das klassische Verfahren lässt bspw. die Frage über die Gestaltung des Abkühlungsvorgangs offen, indem lediglich eine konvexe Abkühlungskurve betrachtet wird. Allerdings können auch beliebig komplexe *Abkühlungspläne* festgelegt werden. Im Wesentlichen erfolgt bei diesen eine Unterscheidung zwischen *dynamischen* und *statischen Abkühlungsplänen*. In [Kuh92] wird beispielsweise die Verwendung eines dynamischen, nichtmonotonen Abkühlungsplanes für eine spezielle Klasse von Optimierungsproblemen näher betrachtet. Auch in [AZ04] beschreiben Azizi und Zolfaghari die Anwendung einer adativen Temperaturanpassung im Rahmen eines dynamischen Abkühlungsplanes. Nachteilig an der Nutzung von Abkühlungsplänen ist, dass die verschiedenen Abkühlungspläne für gewöhnlich nicht verallgemeinerbar sind.

Eine andere Untersuchungsrichtung zielt auf die Kombination der Simulierten Abkühlung mit anderen Optimierungsverfahren ab. Beispielsweise wird bei größeren Nachbarschaften eine deterministische Untersuchung dieser Nachbarschaft vorgeschlagen (siehe bspw. [Käm06] für das *r-opt-Verfahren* oder das *variable r-opt-Verfahren* bei Reihenfolgeproblemen). Zudem wird die Simulierte Abkühlung zur Feinoptimierung (lokale Verbesserung) bei der hybriden Optimierung (vgl. Abschn. 4.5.1) eingesetzt. Bei der vorausgehenden Groboptimierung (globale Suche) wird z. B. ein Genetischer Algorithmus angewandt.

Des Weiteren wurden für das Verfahren der Simulierten Abkühlung auch parallelisierbare Verfahrensvarianten (siehe [Aze92] und Abschn. 4.5.2) und mehrkriterielle Verfahrensvarianten (siehe Abschn. 4.4) entworfen und untersucht.

Abschließend sei erwähnt, dass zum Verfahren der Simulierten Abkühlung noch zwei ähnliche Verfahren existieren. Dies ist zum einen das Verfahren der Schwellenakzeptanz [dt.; threashold accepting [engl.]] und zum andren der Sintflutalgorithmus [dt.; great deluge [engl.]] (siehe Tab. 4.6). Beide Verfahren besitzen jedoch ein einfacheren Verfahrensablauf als das Verfahren der Simulierten Abkühlung, so dass sie lokale Extrema meist nur schwer verlassen können.

Tabusuche (TS). Das Verfahren der Tabusuche wurde im Jahre 1986 unabhängig von Glover und Hansen entwickelt (siehe [Glo86] und [Han86]). Es basiert im Gegensatz zum Verfahren der Simulierten Abkühlung nicht auf Wahrscheinlichkeiten für die Akzeptanz von schlechteren Lösungswerten. Stattdessen kommt bei ihm ein *Speicher* [dt.; *memory* [engl.]], teilweise auch als *Gedächtnis* bezeichnet, zum Einsatz, der sicherstellt, dass die Suche möglichst viele Bereiche des Raumes der Entscheidungsvariablen abdeckt. Der Speicher wird durch die so genannte *Tabuliste* TL mit der Länge l_{TL} begrenzt. Dadurch wird erreicht, dass erst nach l_{TL} Iterationen wieder zu einer kurz zuvor bestimmten Lösung zurückgekehrt werden kann. Die Suche wird also möglichst vielseitig gestaltet. Potentielle Werte für die Entscheidungsvariablen werden aus der Menge $M_{cur} = N(\vec{x}_{cur}) \backslash M_{tabu}$ mit $M_{tabu} = \{\vec{x}_{TL} \,|\, \vec{x}_{TL} \in TL\}$ bestimmt. Folglich hängt die Auswahl von benachbarten Werten für die Entscheidungsvariablen von den, im Speicher *protokollierten, vorhergehenden Schritten* [dt. *moves* [engl.]] ab, welche M_{tabu} bilden. Für diesen Zweck müssen verschiedene Transformationen festgelegt werden. Häufig durchgeführte Transformationen werden für eine gewisse Zeitspanne verboten, d. h. „tabu“ gesetzt[52], um Zyklen während des Suchvorgangs zu vermeiden. In der ursprünglichen Version des Verfahrens wird in jedem Schritt die beste benachbarte Lösung angenommen, auch wenn diese schlechter ist als die aktuelle. Diese *Akzeptanzstrategie* wird in der Literatur als *first acceptance* [engl.] bezeichnet. Es besteht in Erweiterung des Verfahrens aber auch die Möglichkeit, eine verbotene Lösung auszuwählen, falls diese wesentlich besser als alle nicht verbotenen Lösungen ist. Durch diese Verfahrenserweiterung soll das Verlassen eines „guten“ Suchbereiches für die Werte der Entscheidungsvariablen vermieden werden. Wann es zulässig ist, dass auch verbotene Lösungen wieder angenommen werden können, wird von einem festzulegenden *Aspirationskriteriums* [dt. *aspiration criterion* [engl.]] bestimmt. Aspirationskriterien überschreiben somit den Tabustatus „interessanter“ Lösungen. Neben der zuvor erwähnten Akzeptanzstrategie „first acceptance“ [engl.] sind noch weitere Akzeptanzstrategien entwickelt worden (siehe [Wac04]).

Ein wesentlicher Punkt bei der Tabusuche ist die Gestaltung eines Eintrages der Tabuliste. Eine klassische und schnell zu implementierende Tabustrategie ist dabei, das Komplement des in einer Iteration ausgeführten Zuges für eine bestimmte Tabudauer, nämlich l_{TL} Iterationen, in der Tabuliste zu speichern und dadurch zu sperren. Ein anderer Ansatz verbietet die Veränderung von bestimmten Teilbereichen des Vektors mit den Werten für die Entscheidungsvariablen für eine bestimmte Zeit bzw. Anzahl von Optimierungsschritten.

Wie bereits erwähnt, ist der Wert des Parameters l_{TL} des Optimierungsverfahrens, welcher die Länge der Tabuliste festlegt, ein wesentlicher Parameter des Verfahrens. Zu kurze Tabulisten können recht schnell zu Zyklen führen. Im Gegensatz dazu verbieten zu lange Tabulisten viele mögliche Lösungen mitunter unnötig lange und schränken die Suche dadurch zu stark ein. Die geeignete Länge der Tabuliste ist daher meist problemspezifisch zu wählen und sollte experimentell im Rahmen von geeigneten Voruntersuchungen bestimmt werden. Soll das Verfahren hingegen möglichst allgemein gültig gehalten werden, ist für die Wahl der Tabulistenlänge

[52] Daher stammt auch der Name des Verfahrens.

l_{TL} eine der folgenden Vorgehensweisen denkbar:

- l_{TL} zufällig aus einem Intervall wählen und in bestimmten Abständen neu bestimmen oder
- für jeden Zug l_{TL} zufällig neu wählen.

Spezifische Abbruchkriterien und Verfahrensparameter. Bei dem zuvor vorgestellten Verfahren der Tabusuche sind keine weiteren Abbruchkriterien als die allgemein gültigen aus Tab. 4.2 bekannt.

Die wesentlichen Verfahrensparameter einer allgemein gültigen Verfahrensweise für die Tabusuche sind in Tab. 4.8 angeführt.

Tab. 4.8

Symbol	Verfahrensparametername	Bedeutung
l_{TL}	Tabulistenlänge	gibt die maximale Anzahl der Einträge an, welche den Status „tabu“ besitzen und sich somit in der Tabuliste befinden

Tab. 4.8.: *Verfahrensparameter des Optimierungsverfahrens der Tabusuche*

Verfahrenserweiterungen. Wesentliche Verfahrenserweiterungen basieren auf dem Gedächtnis der Tabusuche (siehe [PK00]). Es dient nicht nur zur Vermeidung von Zyklen und lokalen Extrema sondern auch der strategischen Leitung der Suche. Im Allgemeinen werden drei Typen von Gedächtnissen bei der Tabusuche unterschieden:

Kurzzeitgedächtnis [dt.; short term memory [engl.]].
Dieses stellt den wesentlichen Teil des Gedächtnisses dar und dient zur Vermeidung von Schleifen bzw. Zyklen. Es basiert auf der zuvor beschriebenen Vorgehensweise der Tabusuche. Die Länge des Kurzzeitgedächtnisses wird durch l_{TL} festgelegt.

Mittelfristiges Gedächtnis [dt.; intermediate term memory [engl.]].
Dient zur *Intensivierung der Suche* in einem bestimmten Bereich des Suchraumes.

Langzeitgedächtnis [dt.; long term memory [engl.]].
Dient zur *Diversifizierung* (Streuung/Vielfältigkeit) der Suche, d. h. es lenkt die Suche zu noch schlecht abgedeckten Bereichen des Suchraums[53].

Untersuchungen ergaben, dass die besten Ergebnisse mit dem Langzeitgedächtnis erzielt werden können (siehe [GL97]). Auf deren Anwendung wird jedoch im Rahmen dieser Arbeit verzichtet, indem bspw. gezielt zufällig erzeugte Lösungen in das Verfahren „eingestreut“ oder zwischenzeitlich nicht nur lokale Veränderungen sondern auch gezielt globale Veränderungen vorgenommen werden.

[53] Es werden für Lösungsattribute, die nicht häufig benutzt werden, Hilfsoperatoren eingeführt. Diese werden als so genannte „Bestrafe“-Operatoren bezeichnet. Mit deren Hilfe erfolgt z. B. die Bestrafung benutzter Lösungsattribute oder die Bevorzugung seltener Lösungsattribute.

4.3.7. Stochastische Verbesserungsverfahren mit mehreren Lösungen

In Tab. 4.9 sind die bekanntesten Ansätze für heuristische Suchverfahren mit mehreren Vektoren aufgelistet.

Tab. 4.9

Verfahrensbezeichnung	engl. Verfahrensbezeichnung	Literaturverweise
Evolutionäre Algorithmen,	Evolutionary Algorithms (kurz: EA),	[Bäc96], [BFM97], [ES03]
auch *Künstliche Evolution*	Artificial Evolution (kurz: AE)	
Ameisenalgorithmus	Ant Colony Optimisation (kurz: ACO)	[Dor92], [DS04]
Partikel-Schwarm-Optimierung	Particle Swam Optimisation (kurz: PSO)	[HHT05]

Tab. 4.9.: *Überblick über ausgewählte moderne heuristische Verfahren (n paralelle Lösungen)*

Von den in Tab. 4.9 dargestellten Verfahren werden die Evolutionären Algorithmen nachfolgend etwas näher betrachtet, weil diese die am besten untersuchten Optimierungsverfahren darstellen (siehe [GKK04] für eine weitreichende Literaturliste). Zwei Varianten von Evolutionären Algorithmen bzw. einer speziellen Instanz, dem Genetischen Algorithmus, sind auch im `CAOS` implementiert (siehe Abschn. 5.3.6).

Evolutionäre Algorithmen (EA). Evolutionäre Algorithmen orientieren sich stark an der biologischen Evolution, welche in der Lage ist, durch Manipulation des Erbgutes selbst komplexe Lebensformen an ihren (neuen) Lebensraum mit dessen Umweltbedingungen anzupassen. Im Sinne der Optimierung ist ein Lebensraum ein mitunter sehr schwieriges Optimierungsproblem. Der genetische Code einer Lebensform repräsentiert dabei die Erbsubstanz der Individuen einer Population. Bei der biologischen Evolution verändert sich diese Erbsubstanz von Generation zu Generation. Die Erbsubstanz beinhaltet alle notwendigen Informationen zum Aufbau, zum Erscheinungsbild und zur Funktionalität eines Individuums. Zur Anpassung an die Umweltbedingungen eines Lebensraumes besitzt die Evolution dazu drei verschiedene, recht einfache Steuerungsmechanismen:

Mutation.
Die Mutation ist die Veränderung des Erbgutes zur Erzeugung von Alternativen und Varianten eines Individuums. Sie dient dazu, die Diversität des Erbgutes zu erhalten. Dies würde in seiner Charakteristik einem lokalen Suchverfahren bei der Optimierung entsprechen und soll dabei helfen, lokale Extrema zu verlassen.

Rekombination.
Die Rekombination wird auch crossover [engl.] genannt. Sie dient dazu, neue Kombinationen von väterlichem und mütterlichem Erbgut zu erstellen und wird als nicht so wichtiger Evolutionsfaktor angesehen, weil durch sie keine neuen Informationen in den Genpool kommen. Wäre nur die Rekombination aktiv, kann die Evolution nur solange bessere Individuen erzeugen, bis alle vorhandenen Informationen genutzt wurden. Im Sinne der Optimierung ent-

spricht dies der Suche in einem Raum, welcher durch eine diskrete Menge begrenzt ist. Dadurch würden nur die besten Werte für die Entscheidungsvariablen in einem eingeschränkten Raum gefunden werden, worin nicht zwingend das globale Optimum enthalten sein muss.

Selektion.
Die Selektion dient der (verstärkten) Auswahl der am besten an ihren Lebensraum angepassten Individuen. Dazu wird den Individuen eine Fitness zugeordnet, welche die Überlebensfähigkeit repräsentiert. Bei der simulationsbasierten Optimierung wird die Fitness bspw. durch den Zielfunktionswert[54] bestimmt, welcher aus den bewerteten Ausgabewerten der Simulationen gebildet wird.

Als eines der wichtigsten Anwendungsgebiete der Evolutionären Algorithmen gelten Optimierungsprobleme, bei denen die exakten Optimierungsverfahren versagen. Dies ist i. Allg. bei nichtlinearen, diskontinuierlichen und multimodalen Zielfunktionen der Fall. Evolutionäre Algorithmen zeichnen sich durch ihre Robustheit aus, welche darin begründet liegt, dass zum einen keine Annahmen über das gestellte Optimierungsproblem getroffen werden, und zum anderen stets mit einer Menge von Individuen (Population von Individuen) gearbeitet wird, welche die (zulässigen) Lösungen repräsentieren. Durch die verschiedenen Individuen einer Population werden gleichzeitig mehrere Wege zum Optimum ausprobiert. Zusätzlich werden, durch die Vererbung bzw. die Rekombination, Informationen über die verschiedenen Wege ausgetauscht. Auf diese Weise wird das Wissen über den Zielfunktionsraum des zugrunde liegenden Optimierungsproblems in der gesamten Population verteilt, wodurch eine frühzeitige Konvergenz während der Optimierung verhindert werden kann.

Bei den evolutionären Algorithmen existieren verschiedene Ansätze, welche sich im Wesentlichen in der rechnerinternen Repräsentation der Vektoren der Individuen, den vorhandenen Manipulationsoperatoren und den Ersetzungsschemas unterscheiden. Die vier prinzipiellen Ansätze sind

- die Genetischen Algorithmen,
- die Evolutionsstrategien,
- die Genetische Programmierung und
- die Evolutionäre Programmierung,

welche in Tab. 4.10 mit entsprechenden Literaturverweisen zusammenfassend dargestellt sind.

Davon werden nachfolgend die Genetischen Algorithmen und die Evolutionsstrategien betrachtet, weil diese und verschiedene Mischformen dieser sich zur Repräsentation der innerhalb dieser Arbeit betrachteten Optimierungsprobleme durch-

[54] Diese direkte Gleichsetzung von Fitness und Zielfunktionswert ist weit verbreitet. In der Literatur sind allerdings auch andere Fitnessfunktionen untersucht worden (siehe z. B. [SHF94]).

Verfahrensbezeichnung	engl. Verfahrensbezeichnung	Literaturverweise
Evolutionsstrategie	Evolution Strategy (kurz: ES)	[Rec73], [Sch81], [Sch95], [BS02a]
Genetische Algorithmen	Genetic Algorithms (kurz: GA)	[Hol69], [Hol75], [Sch04], [Fen05]
Evolutionäre Programmierung	Evolutionary Programming (kurz: EP)	[FOW66], [ES03]
Genetische Programmierung	Genetic Programming (kurz: GP)	[Smi80], [Cra85], [Koz92], [KKS$^+$03]

Tab. 4.10

Tab. 4.10.: *Instanzen der Evolutionären Algorithmen*

gesetzt haben[55]. In Tab. 4.11 sind die GA und die ES in ihren ursprünglichen Varianten und ihren wesentlichen Merkmalen gegenübergestellt.

Nach Meinung des Autors sind heutzutage überwiegend Mischformen aus GA und ES zu finden, um die verschiedenen Vorteile der beiden Instanzen der EA auszunutzen. Jedoch werden die grundlegenden Prinzipien, wie die Nutzung mehrerer Individuen, die Verwendung verschiedener Manipulationsoperatoren, usw., zumeist beibehalten. Selbst Schwefel (siehe [SHF94]) veränderte die ursprünglichen ES bereits so, dass sie bei den „$(\mu \,/\, \rho \,\#\, \lambda)$“-ES[56] schon die Rekombinationsoperation mit als genetischen Operator nutzen. Das Hauptaugenmerk bei den genetischen Operatoren der ES blieb aber auf die Mutation gerichtet.

Spezifische Abbruchkriterien und Verfahrensparameter. Beim generationsbasierten Genetischen Algorithmus kann zu den aus Tab. 4.2 bekannten Abbruchkriterien noch die maximale Anzahl von bisher erzeugten Generationen Gen_{max} herangezogen werden. Das Abbruchkriterium sei mit SC_{Gen} bezeichnet. Für den GA mit stetiger Ersetzung (s. u.) sind keine weiteren Abbruchkriterien vorgesehen. Der GA basierend auf dem Generationsprinzip wird nachfolgend kurz GA_{gen} und der GA zur stetigen Ersetzung GA_{ss} genannt.

Auf Grund der weitestgehenden Problemunabhängigkeit und Vielfältigkeit der beiden Arten von GA existieren verschiedenste Verfahrensparameter für diese Optimierungsverfahren, welche in Tab. 4.12 zusammengefasst sind. Dabei ist im Exponent der verwendeten Symbole angegeben, ob der Verfahrensparameter auf ein Reihenfolgeoptimierungsproblem (*permut*) oder ein reellwertiges Optimierungsproblem (*real*) anwendbar ist.

[55] Die Evolutionäre Programmierung beschäftigt sich mit der möglichst natürlichen Darstellung eines (Optimierungs-)Problems (siehe [FOW66]). Währenddessen bezieht sich die Genetische Programmierung eher auf die dynamische Darstellung, z. B. mittels Baumstrukturen (siehe [Koz92] sowie [KKS$^+$03]).

[56] Die „$(\mu \,/\, \rho \,\#\, \lambda)$“-ES gleichen den „$(\mu \# \lambda)$“-ES, wobei der „#“-Operator für den „+“- oder „,“-Operator der ES steht. Bei der „$(\mu + \lambda)$“-ES werden aus den μ Eltern und den daraus mutierten λ Nachkommen die μ Eltern der nachfolgenden Generation selektiert. Hingegen dienen bei den „(μ, λ)“-ES die μ Eltern nur noch zur Bestimmung der λ mutierten Nachkommen, aus welchen anschließend die μ Eltern der nachfolgenden Generation selektiert werden. Bei den „$(\mu \,/\, \rho \,\#\, \lambda)$“-ES kommt noch hinzu, dass ein Nachkommen nicht mehr nur aus einem Elter sondern aus ρ Eltern erzeugt bzw. rekombiniert wird.

Tab. 4.11

Merkmal	Genetische Algorithmen	Evolutionsstrategien
Repräsentation	binäre Zahlen	reelle Zahlen
Genetische Operatoren		
• *Selektion*	dient nur zur Auswahl der benötigten Eltern aus der Elterngeneration	wird bei der Auswahl der Eltern (und der Kinder) angewandt
• *Selektionsart*	Heirats-Selektion	über Elterngeneration gleichverteilte Selektion (Uniforme Selektion)
• *Mutation*	nur geringe Bedeutung, nicht selbstregulierend	Addition oder Subtraktion einer normalverteilten reellen Zahl, selbstregulierende mutative Schrittweitensteuerung
• *Rekombination*	überwiegend genutzt	nicht genutzt
• *Rekombinationsarten*	verschiedene (bspw. one-point-crossover [engl.], two-point-crossover [engl.], u. a.)	entfällt
Erzeugungsschema	zwei Eltern erzeugen genau zwei Kinder	ein Elter oder mehrere Eltern erzeugen ein oder mehrere Kinder
	Generationsweise (Kindergeneration ersetzt komplette Elterngeneration)	bei „+“-Notation: die Besten aus Kinder- und Elterngeneration bilden die neue Elterngeneration; bei „ ,“-Notation: die Besten aus Kindergeneration ersetzen die gesamte Elterngeneration

Tab. 4.11.: *Überblick über die Unterschiede von GA und ES*

Verfahrenserweiterungen. Verschiedene Rekombinationsoperatoren, wie z. B.

- für reellwertige Optimierungsprobleme
  - die Uniforme Rekombination [dt.; uniform crossover [engl.]],
  - die Ein-Punkt-Rekombination [dt.; one-point-crossover [engl.]],
  - die Zwei-Punkt-Rekombination [dt.; two-point-crossover [engl.]],
  - die Misch-Rekombination [dt.; mixing-crossover [engl.]],
  - die Intermediäre Rekombination [dt.; intermediar crossover [engl.]],
  - die Linksschieben-Rekombination [dt.; shift-left-crossover [engl.]] und
  - die Rechtsschieben-Rekombination [dt.; shift-right-crossover [engl.]] sowie
- für Reihenfolgeoptimierungsprobleme
  - die Uniforme Rekombination [dt.; uniform crossover [engl.]],
  - die Zwei-Punkt-Rekombination [dt.; two-point-crossover [engl.]],
  - die Tausch-Rekombination [dt.; change crossover [engl.]] und

Tab. 4.12

Verfahrensart	Symbol	Verfahrensparametername	Bedeutung
GA_{gen}/ GA_{ss}	$M_{select}^{permut/real}$	Menge der Selektionsoperatoren	legt die Menge der Selektionsoperatoren fest
	$M_{cross}^{permut/real}$	Menge der Rekombinationsoperatoren	legt die Menge der Rekombinationsoperatoren fest
	$M_{mutate}^{permut/real}$	Menge der Mutationsoperatoren	legt die Menge der Mutationsoperatoren fest
	$p_{select}^{permut/real}$	Selektionswahrscheinlichkeit	gibt die Wahrscheinlichkeit für die Selektion eines Individuums an
	$p_{cross}^{permut/real}$	Rekombinationswahrscheinlichkeit	gibt die Wahrscheinlichkeit für die Rekombination zweier selektierter Individuen an
	$p_{mutate}^{permut/real}$	Mutationswahrscheinlichkeit	gibt die Wahrscheinlichkeit für die Mutation zweier selektierter Individuen an
	r_{mutate}^{real}	Mutationsrate	gibt bei reellwertigen Optimierungsproblemen die Rate für die Mutation zweier zur Mutation ausgewählter Individuen an
	$n_{indGen}^{permut/real}$	Individuenanzahl	gibt die (konstante) Anzahl der Individuen in einer Generation an
GA_{gen}	$n_{elite}^{permut/real}$	Anzahl elitärer Individuen	gibt die Anzahl der Individuen an, welche ohne Veränderung in die nächste Generation zu übernehmen sind
GA_{ss}	$n_{offset}^{permut/real}$	Offset-Anzahl	gibt die Anzahl der Individuen an, deren Fitness gleichzeitig berechnet werden kann

Tab. 4.12.: *Verfahrensparameter der Optimierungsverfahren für die Genetischen Algorithmen GA_{gen} und GA_{ss}*

– die teilweise Rekombination [dt.; partially-mapped-crossover [engl.]]

sowie verschiedene Mutationsoperatoren für Reihenfolgeoptimierungsprobleme, wie

- die Ersetzungs-Mutation [dt.; displace mutation [engl.]],
- die Umkehr-Mutation [dt.; inversion mutation [engl.]],
- die Mix-Mutation [dt.; Mixing mutation [engl.]],
- die Schüttel-Mutation [dt.; shake mutation [engl.]] und
- die Einfüge-Mutation [dt.; insertion mutation [engl.]]

sind im Laufe der Zeit entwickelt worden (siehe u. a. [Nie96]), um bei verschiedenen Optimierungsproblemen möglichst das globale Optimum zu erreichen. Des Weiteren kamen verschiedenste neue Selektions-Operatoren hinzu, wie bspw.

- die Tournier-Selektion,
- die Roulette-Selektion,
- die Selektion basierend auf dem linearen Rank und
- die Exponentielle Selektion,

welche Nieländer in [Nie96] vorstellt.

Es wurden jedoch auch andere Untersuchungen unternommen, welche bspw. den *Elitismus* [dt.; survival of the fittest [engl.]] bei generationsbasierten EA betrachten (siehe [SHF94]). Der Elitismus dient hierbei dazu, eine Menge von Individuen der Elterngeneration direkt in die Kindergeneration zu übernehmen, ohne diese durch Anwendung eines genetischen Operators zu verändern.

Eine andere Untersuchungsrichtung zielt darauf ab, Genetische Algorithmen zu nutzen, welche mit nur einer Generation arbeiten und diese versuchen, stetig zu verbessern. Auf Grund der stetigen Ersetzung schlechter bewerteter Individuen werden sie auch *steady-state-GA* [engl.] genannt. Diese neigen jedoch dazu, vorzeitig gegen ein lokales Optimum zu konvergieren, so dass Möglichkeiten geschaffen werden müssen, um diese Konvergenz zu verhindern.

Andere Ansätze sehen das Lernen der verschiedenen veränderbaren Verfahrensparameter eines EA vor, um das Verfahren möglichst noch problemunabhängiger zu gestalten. Dazu wird der Phänotyp[57] der Individuen zusätzlich um diese Verfahrensparameter erweitert.

Weitere moderne heuristische Suchverfahren mit mehreren Lösungen. Neben den Evolutionären Algorithmen existieren noch weitere Verfahren, welche mit mehreren Lösungen arbeiten. Die wichtigsten Vertreter sind die Verfahren der *Schwarmintelligenz*, zu denen auch der *Ameisenalgorithmus* und die *Partikel-Schwarm-Optimierung* gehören (siehe auch Tab. 4.9). Beide Verfahren sind, wie die EA, von der Natur inspiriert und populations-basiert.

Ameisenalgorithmen. Bei den Ameisenalgorithmen wird davon ausgegangen, dass Ameisen nur in der Gruppe fähig sind, Nahrung zu finden, und als Einzelindividuen eher chancenlos sind. Dabei kommunizieren die Ameisen untereinander über *Pheromone*, welche sie auf ihrem Weg zur Nahrung auslegen. Es entsteht eine *Pheromonspur*, die am stärksten auf dem günstigsten Weg zur Nahrung ist. Somit folgen die Ameisen mit einer hohen Wahrscheinlichkeit der stärksten Pheromonspur auf ihrer Nahrungssuche, wobei sie selbst wieder Pheromone auslegen und somit die Pheromonspur verstärken. Demzufolge lässt eine wenig genutzte Pheromonspur im Laufe der Zeit nach, da sie nicht erneuert wird und einen ungünstigeren Weg repräsentiert.

Im Bezug auf die Optimierung kann sich diese Strategie zu Nutze gemacht werden, indem ein Optimierungsproblem mittels eines *geeigneten Graphen nachgebildet* wird. Dieser Graph repräsentiert dann die möglichen Suchwege zum Auffinden einer (sub-)optimalen Lösung. Zur Umsetzung des Prinzips der Pheromonspuren werden die *Kanten des Graphen mit Gewichten versehen*. Während des

[57] Der Phänotyp ist die Ausprägung der Eigenschaften, welche durch den Genotyp des Individuums kodiert sind. Er spiegelt das „äußere Erscheinungsbild“ wieder. Die Entwicklung des Phänotyps kann Nebenbedingungen unterworfen sein (siehe Abschn. 2.2.4). Die genaue Definition des Phänotyps ist zumeist problemabhängig. Der Genotyp hingegen ist die (konkrete) Repräsentation auf der die Rekombinations- und Mutationsoperatoren arbeiten.

Ablaufes des Ameisenalgorithmus wird die Population aus k Ameisen (Lösungen) zunächst mittels einer Konstruktionsheuristik, bspw. einem geeigneten (problemspezifischen) Eröffnungsverfahren, initialisiert. Danach iteriert das Verfahren bis ein betreffendes Abbruchkriterium erreicht ist. Bei jeder Verfahrensiteration wird für ausgewählte oder alle Lösungen eine lokale Veränderung mittels eines Verbesserungsverfahrens oder lokalen Suchverfahrens vorgenommen. Zum Ende einer Iteration erfolgt eine Erneuerung der Pheromonspuren anhand der (neuen) Zielfunktionswerte. Das Verfahren der Ameisenalgorithmen wird u. a. zur Lösung von TSP, Scheduling-Problemen, u. Ä. angewandt. Der wesentliche Nachteil der Verfahren ist, dass sie meist in lokalen Optima enden. Dies hängt allerdings stark von verwendeten Verbesserungsverfahren ab. Die wichtigsten Vertreter der Ameisenalgorithmen sind das *ant colony system* [engl.] (beschrieben in [DS04]) und das *ant system* [engl.] (nachzulesen in [Dor92]).

Partikel-Schwarm-Optimierung. Bei der Partikel-Schwarm-Optimierung wird das soziale Verhalten von Vogel- oder Fischschwärmen nachgebildet. Diese Schwärme sind dadurch gekennzeichnet, dass einer aus dem Schwarm einen sinnvollen, machbaren Weg, z. B. bei der Futtersuche, sieht und die anderen ihm sehr rasch nachfolgen. Das Verhalten der Schwärme kann auf Optimierungsprobleme übertragen werden. Dies geschah erstmals durch Eberhart und Kennedy (siehe [KE01]). Es entstand ein Algorithmus, der teilweise Ähnlichkeiten zu den Evolutionären Algorithmen aufweist. Die Individuen werden bei der Partikel-Schwarm-Optimierung jedoch als Partikel bezeichnet. Der Schwarm bestehe aus der Menge aller Partikel. Die erste Generation (Startpunkte des Schwarms) wird mit zufälligen Werten für die Entscheidungsvariablen initialisiert. Dem Partikel mit der besten Fitness, bspw. das Partikel mit dem besten Zielfunktionswert, wird gefolgt. Dabei überqueren die anderen Partikel weitere Bereiche des Zielfunktionsraumes. Stoßen sie dabei auf bessere Lösungen, wandert der Schwarm in diese Richtung, usw. Zur Veränderung der Werte der Entscheidungsvariablen für die Population werden bei jedem Partikel individuell dessen *aktuelle Geschwindigkeit* sowie dessen *aktueller Standort* und dessen *aktuelle Richtung* genutzt[58]. Problematisch bei diesem Algorithmus erweist sich das Auftreten von Nebenbedingungen, welche u. a. den Zielfunktionsraum sehr zerklüften können.

4.4. Verfahren der mehrkriteriellen Optimierung

Im Fall der mehrkriteriellen Optimierung von nichtlinearen Optimierungsproblemen gibt es derzeit kein exaktes Lösungsverfahren, welches problemunabhängig eingesetzt werden kann. Unter Hinzunahme von Nebenbedingungen kann sogar meist nur das Verfahren der Vollständigen Aufzählung die globale Pareto-optimale Menge genau bestimmen. Dies jedoch auch nur, wenn eine Diskretisierung des Raumes der Entscheidungsvariablen erfolgte (siehe Abschn. 2.2.6). Aus diesem Grund soll sich nachfolgend auf die heuristischen Suchverfahren der mehrkriteriellen Optimierung konzentriert werden.

[58] Im Gegensatz dazu werden bei den EA Rekombinations- und Mutationsoperatoren zur Veränderung der Werte der Entscheidungsvariablen eines Individuums angewandt.

Es sei an dieser Stelle noch einmal erwähnt, dass eine Pareto-optimale Menge i. Allg. aus mehreren optimalen Lösungen besteht (vgl. Pareto-optimale Front im Abschn. 2.2.5), so dass auch die mehrkriteriellen, heuristischen Suchverfahren mit mehreren Lösungen arbeiten sollten. Es bieten sich daher Verfahren an, welche sich an die in Abschn. 4.3.7 bzw. Tab. 4.9 vorgestellten Verfahren anlehnen. Diese sind allerdings noch an die Bedürfnisse der mehrkriteriellen Optimierung anzupassen.

Darüber hinaus existieren jedoch auch mehrkriterielle Verfahrensvarianten für die im Abschn. 4.3.6 vorgestellten heuristischen Suchverfahren, welche mit nur einer Lösung arbeiten. Diese funktionieren zumeist so, dass mehrere Instanzen des jeweiligen einkriteriellen Verfahrens parallel existieren, welche ihre Suche von verschiedenen Startpunkten aus beginnen. In Tab. 4.13 sind verschiedene heuristische, mehrkriterielle Suchverfahren mit ihren entsprechenden Literaturverweisen angeführt.

Tab. 4.13

Einkriterielles Verfahren	**Mehrkriterielle Verfahrensvariante(n)**	**Literaturverweise**
Simulierte Abkühlung	*Mehrkriterielle Simulierte Abkühlung* (kurz: MSA)	[SEF+04], [Smi07]
Tabusuche	*Mehrkriterielle Tabusuche* (kurz: MTS)	[SAH04]
Genetischer Algorithmus	*Mehrkriterieller Genetischer Algorithmus*, Multi-Objective Genetic Algorithm [engl., kurz: MOGA],	[FF93], [FF98]
	Niched Pareto Genetic Algorithm 2 [engl., kurz: NPGA2],	[EMH01]
	Vector Evaluated Genetic Algorithm [engl., kurz: VEGA],	[Sch85a]
	Non-dominated Sorting Genetic Algorithm [engl., kurz: NSGA]	[SD94], [DAPM00]
	Non-dominated Sorting Genetic Algorithm 2 [engl., kurz: NSGA2]	[DAPM00], [DPAM02]
	Jumping Gene Genetic Algorithm [engl., kurz: JGGA]	[CMTK05]
Evolutionärer Algorithmus	*Mehrkriterieller Evolutionärer Algorithmus*, Multi-Objective Evolutionary Algorithms [engl., kurz: MOEA]	[MT05], [CL04]
	Strength Pareto Evolutionary Algorithm [engl., kurz: SPEA],	[ZT99]
	Strength Pareto Evolutionary Algorithm 2 [engl., kurz: SPEA2],	[ZLT01]
Evolutionsstrategie	Pareto Archived Evolution Strategy [engl., kurz: PAES],	[KC99], [KC00]
Partikel-Schwarm-Optimierung	*Mehrkriterielle Partikel-Schwarm-Optimierung*, Multi-Objective Particle Swarm Optimization [engl., kurz: MOPSO]	[MT03]

Tab. 4.13.: *Mehrkriterielle Varianten moderner heuristischer Verfahren (Auswahl)*

Methoden mehrkriterieller Optimierungsverfahren. Die Nutzung mehrkriterieller Optimierungsverfahren stellt den Entscheidungsfinder (Anwender) vor das Pro-

blem, welche der gefundenen Lösungen der Pareto-optimalen Menge er nutzen soll. Zu deren Auswahl sind folgende, verschiedene Methoden denkbar:

A-priori.
Der Entscheidungsfinder legt im Vorfeld die Bedeutung der einzelnen Zielfunktionen, bspw. durch Gewichtung, fest und überführt somit das MOP in ein SOP (siehe Abschn. 2.5 für die Umwandlung eines MOP in ein SOP).

A-posteriori.
Nach der Optimierung bzw. nach einem Lauf der simulationsbasierten Optimierung wählt der Entscheidungsfinder die für ihn sinnvollen Lösungen aus der vorliegenden Pareto-optimalen Menge, bspw. mittels eines Gütekriteriums, aus (siehe Abschn. 2.6 für die Nutzung eines Gütekriteriums zur Überführung einer Lösung in den Entscheidungsraum).

Progressiv.
Es erfolgt eine kontinuierliche Interaktion mit dem Entscheidungsfinder. Diese Vorgehensweise verlangsamt u. U. den Optimierungsprozess, kann aber die Konvergenz zu bestimmten, erwünschten Zielfunktionsräumen besser erzwingen.

Nachfolgend wird von der Nutzung der A-posteriori-Methoden ausgegangen, weil die Unabhängigkeit von weiteren Anwendereingaben während eines Laufes der simulationsbasierten Optimierung durch die wiederholte Interaktion der progressiven Methode nicht gewährleistet ist. Die A-priori-Methoden zielen zudem direkt auf die Nutzung einkriterieller und nicht mehrkriterieller Optimierungsverfahren ab.

Mehrkriterielle Evolutionäre Algorithmen. In den vergangenen Jahren wurden vorrangig mehrkriterielle, heuristische Suchverfahren angewandt, welche auf generationsbasierten EA aufbauen. Die wesentlichen Verfahrensmerkmale der (rechnergestützten) Evolution (vgl. EA in Abschn. 4.3.7) wurden dazu beibehalten. Folglich blieben die Operatoren Selektion, Rekombination und Mutation im Wesentlichen unverändert. Lediglich die Fitness-Funktion wurde an die Forderung der mehrkriteriellen Entscheidungsfindung angepasst, um die Auswahl geeigneter Individuen zur Rekombination zu ermöglichen. Es haben sich dabei im Laufe der Zeit verschiedene Fitness-Funktionen herausgebildet, welche sich bspw. bei rangbasierten Verfahren in verschiedener Art und Weise an der Anzahl der Lösungen orientieren, welche eine andere Lösung dominieren.

Die wichtigsten Vertreter von mehrkriteriellen GA und mehrkriteriellen EA wurden bereits in Tab. 4.13 vorgestellt. Von diesen verschiedenen mehrkriteriellen Verfahren wurden vom Autor besonders die beiden Verfahren SPEA2 und NSGA2 als geeignet erachtet. Dies liegt einerseits an den guten Ergebnissen, die diese Verfahren im Vergleich zu anderen hervorgebracht haben, und andererseits an den wenigen einstellbaren Verfahrensparametern, die diese Verfahren aufweisen. Meyer bemerkt in [Mey02, S. 8] dazu:

Der NSGA2-Algorithmus zählt heute, neben dem SPEA2-Algorithmus zu den effektivsten MOEA.

Durch die geringe Anzahl von einstellbaren Verfahrensparametern ist der Nutzereingriff zur Erzielung eines erfolgversprechenden Verfahrensablaufes minimal. Dennoch sind diese beiden Verfahren robust genug gegenüber Schwierigkeiten im Zielfunktionsraum, wie sie bspw. durch komplexe Nebenbedingungen hervorgerufen werden. Beide Verfahren nutzen den Ansatz des Elitismus und sind auf Grund ihrer Robustheit und guten Untersuchungsergebnisse auch im CAOS implementiert (siehe Abb. 4.6 und [Mey02]), um u. a. einem Anwender des Softwaresystems die Nutzung eines mehrkriteriellen Optimierungsverfahrens zu ermöglichen.

Abb. 4.6

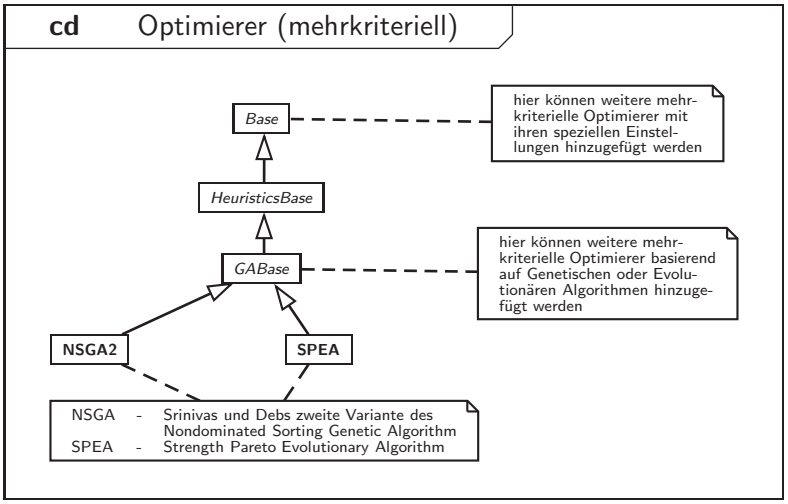


Abb. 4.6.: *Mehrkriterielle Optimierer des CAOS (UML-Klassendiagramm)*

Weitere mehrkriterielle heuristische Optimierungsverfahren. Wie einleitend bereits angemerkt und in Tab. 4.9 erkennbar, existieren noch weitere mehrkriterielle heuristische Optimierungsverfahren. Dazu wurden die bekanntesten und am weit verbreitetsten einkriteriellen Optimierungsverfahren entsprechend den Anforderungen der mehrkriteriellen Optimierung erweitert. Sie seien innerhalb dieser Arbeit nur erwähnt, ohne näher auf diese einzugehen.

4.5. Methodiken zur Verkürzung der Optimierungsdauer

Wie schon bei der Simulation (siehe Abschn. 3.6) ist es auch bei der Optimierung sinnvoll Methodiken zu suchen, welche die Dauer eines Optimierungslaufes bzw. eines Laufes der simulationsbasierten Optimierung verkürzen. Dazu sollen zwei verschiedene Methodiken betrachtet werden. Dies ist zum einen die *hybride Optimierung*, welche vorrangig auf die Verkürzung der Optimierung unter Nutzung eines einzelnen Rechnerprozessors sowie mehrerer unterschiedlicher Optimierungsverfahren abzielt und zum anderen die *verteilte und parallele Optimierung*, welche

die Nutzung mehrerer, zur Kommunikation durch ein Rechnernetzwerk, verbundener Rechner zur gleichzeitigen Simulation mehrerer Lösungen eines Optimierungsverfahrens anstrebt. Beide Methodiken werden in den nachfolgenden Abschnitten etwas genauer betrachtet. Prinzipiell ist aber auch eine Kombination von hybrider Optimierung sowie verteilter und paralleler Optimierung denkbar (vgl. Abb. 4.7). Der Autor ist jedoch bei seiner ausgiebigen und gezielten Literaturrecherche auf kein Verfahren gestoßen, welches die Möglichkeiten von hybrider Optimierung sowie verteilter und paralleler Optimierung vereint.

Abb. 4.7

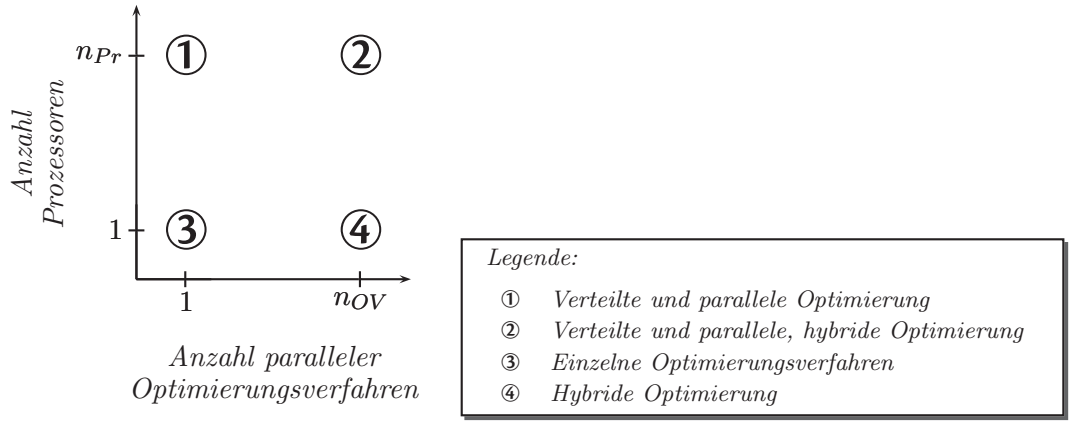


Abb. 4.7.: *Vergleich zwischen Anzahl paralleler Optimierungsverfahren und Anzahl verfügbarer Rechnerprozessoren*

4.5.1. Hybride Optimierung

Jedes der im Abschn. 4.3 vorgestellten heuristischen Suchverfahren besitzt den Nachteil, dass es entweder nur bei bestimmten Optimierungsproblemen, infolge der Strukturen des Entscheidungsvariablen- und Zielfunktionsraumes, oder nur bei geeigneter Einstellung der Verfahrensparameter des Optimierungsverfahrens eine hinreichend gute Lösung und somit zumeist nur ein lokales Optimum findet. Manlig beschreibt in [Man01], dass die Auswahl des anzuwendenden Optimierungsverfahrens von verschiedenen Erfahrungswerten und weiteren problemspezifischen Parametern, wie dem zugrunde liegenden Optimierungsproblem, der Anzahl der zu untersuchenden Alternativen, dem genutzten Simulations- und Optimierungssystem sowie der jeweiligen Hardwarekonfiguration abhängt. Die beiden letzten Einflussparameter seien an dieser Stelle außer Acht gelassen, weil das entwickelte Softwaresystem `CAOS` durch seine Plattformunabhängigkeit auf einer beliebigen Hardwarekonfiguration zum Einsatz kommen kann. Demnach sind nur die beiden erst genannten Einflussfaktoren, das zugrunde liegende Optimierungsproblem und die Anzahl der zu untersuchenden Alternativen, an dieser Stelle von Bedeutung. Es müsste folglich für jedes Optimierungsproblem oder dessen Instanz ein *spezialisiertes Optimierungsverfahren* mit ggf. geeigneten Einstellungen der Verfahrenspa-

rameter[59] entwickelt werden. Dies widerspricht dem innerhalb dieser Arbeit angestrebten Ansatz der möglichst weitgehenden Allgemeingültigkeit und Problemunabhängigkeit der implementierten Verfahren. Einen Ausweg aus dem Dilemma, die optimalen Verfahrensparameter im Rahmen von Voruntersuchungen möglichst geeignet einzustellen, oder wie bei den selbst-adaptiven Verfahren zu erlernen, stellt die hybride Optimierung dar, welche mehrere Optimierungsverfahren in geeigneter Art und Weise miteinander koppelt. Mit den hybriden Optimierungsverfahren soll erreicht werden, dass

1. für einen Anwender die in seinem *Realsystem(-ausschnitt) existierenden* oder *zukünftig auftretenden Optimierungsprobleme* mit gewünschter (hinreichender) Güte *gelöst werden*,
2. ein möglichst *einfach zu handhabendes, breit anwendbares Optimierungsverfahren*, welches *nach außen homogen* erscheint, geschaffen wird und
3. ein Optimierungsverfahren geschaffen wird, das möglichst *effizient mit den zur Verfügung stehenden Ressourcen* arbeitet.

Ob das erste „Ziel“ erfüllt werden kann, wird sich dabei zumeist nur an Hand praktischer Untersuchungen zeigen lassen. Das Erreichen des zweiten „Zieles“ ist meist schwerer, weil nicht genau gesagt werden kann, welche Optimierungsverfahren mit welchen Einstellungen für die Verfahrensparameter geeignet sind. Diese Einstellungen könnten zwar erlernt werden (selbst-adaptive Verfahren), allerdings würde sich dadurch auch der zeitliche Aufwand bei der Suche nach einer geeigneten, hinreichend guten Lösung erhöhen und die gewünschte Einsparung der Dauer eines Optimierungslaufes ginge verloren. Zum Erreichen der Effizienz der Verfahren (drittes „Ziel“) ist wiederum i. Allg. Kenntnis über die Strukturen des Entscheidungsvariablen- und Zielfunktionsraumes eine notwendige Vorraussetzung. Wolpert und MacReady bemerken in [WM95] und [WM97] dazu, dass im Mittel kein Verfahren besser als die gleichverteilte zufällige globale Suche über den gesamten Raum der Entscheidungsvariablen ist. Diese Behauptung wird auch als *no-free-lunch-theorem* [engl.] bezeichnet. Für die hybriden Optimierungsverfahren gilt Selbiges, weil nicht gesichert ist, dass geeignete Optimierungsverfahren mit geeigneten Verfahrenseinstellungen ausgewählt wurden. Dennoch wird innerhalb dieser Arbeit davon ausgegangen, dass hybride Optimierungsverfahren einen entscheidenden Vorteil gegenüber einem einzelnen Optimierungsverfahren besitzen. Darüber hinaus wird verlangt, dass nur Optimierungsverfahren mit unterschiedlichen Charakteristiken bei der Durchmusterung des Raumes der Entscheidungsvariablen oder zumindest identische Optimierungsverfahren mit verschiedenen Einstellungen der Verfahrensparameter zum Einsatz kommen[60]. Dass der Einsatz von hybriden Optimierungsverfahren i. d. R. effizienter ist, wird durch die zahlreichen

[59] Alternativ könnten die Verfahren selbst-adaptiv oder lernfähig sein und sich die geeigneten Einstellungen der Verfahrensparameter selbst aneignen, was jedoch i. Allg. zu Ungunsten der Optimierungsdauer erfolgt. Zu dem sind die Verfahrensparameter meist problemspezifisch, so dass sie bei einem neuen (Typ von) Optimierungsproblem neu erlernt werden müssen.

[60] Teilweise wird sogar verlangt, dass identische Optimierungsverfahren nicht als hybrid bezeichnet werden sollten, weil die Optimierungsverfahren durch Veränderung der Verfahrensparameter ineinander überführbar sind (siehe bspw. [Had01, S. 34]).

in der Literatur zu findenden, mittels hybriden Optimierungsverfahren effizienter[61] gelösten, Optimierungsproblemstellungen bestätigt. Dennoch sollte im Vorfeld der sinnvolle Einsatz von hybriden Optimierungsverfahren evtl. durch den Vergleich mit anderen Optimierungsverfahren getestet werden.

Im Wesentlichen hat sich bis heute in der Literatur kein eindeutiges Klassifizierungsschema zur Einteilung der hybriden Optimierungsverfahren durchgesetzt. Daher soll sich an dem von Hader in [Had01, S. 34] bzw. Wachsmuth in [Wac05] genutzten Klassifizierungsschema orientiert werden. Es spiegelt die im Softwaresystem `CAOS` implementierten Verfahrensvarianten am Besten wider und ist in Abb. 4.8 dargestellt.

Abb. 4.8

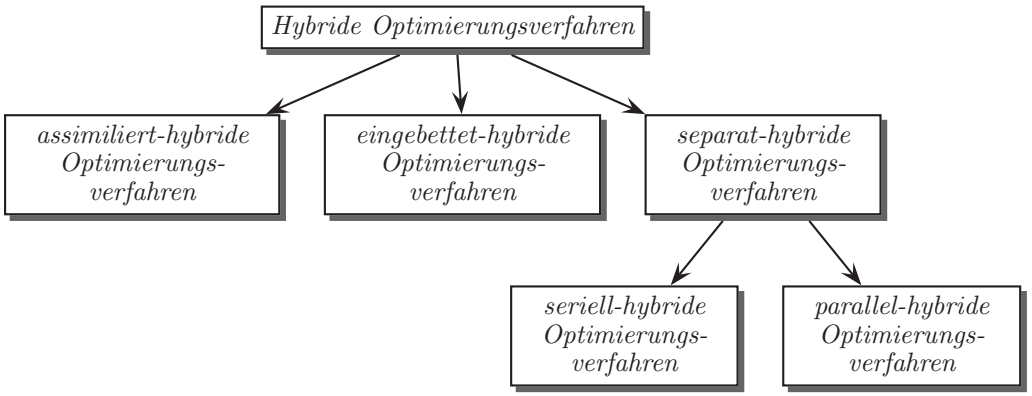


Abb. 4.8.: *Mögliches Klassifizierungsschema der hybriden Optimierungsverfahren nach [Had01, S. 34]*

Die unterschiedlichen Ausprägungen der hybriden Optimierungsverfahren besitzen dabei folgende Charakteristiken (siehe [Had01, S. 35ff.]):

assimiliert-hybride Optimierungsverfahren.
Ein (Basis-)Optimierungsverfahren verwendet bzw. assimiliert verschiedene Charakteristiken anderer Verfahren.

eingebettet-hybride Optimierungsverfahren.
Das Basis-Optimierungsverfahren nutzt ein anderes Optimierungsverfahren zur Lösung bestimmter (wiederkehrender) Teilaufgaben.

separat-hybride Optimierungsverfahren.
Die Optimierungsverfahren besitzen kein einzelnes Basis-Optimierungsverfahren, sondern nutzen eine Reihe gleichberechtigter, „separater“ Optimierungsverfahren, welche miteinander in geeigneter Weise gekoppelt sind. In Abhängigkeit der Kopplung werden diese noch einmal unterteilt.

seriell-hybride Optimierungsverfahren.
Mehrere Optimierungsverfahren werden in fest vorgegebener oder dynamischer Reihenfolge hintereinander ausgeführt, wobei als Startvektoren

[61] Effizienter bedeutet in diesem Fall, effizienter gegenüber dem besten einzelnen der Optimierungsverfahren, welche bei der Hybridisierung eingesetzt werden.

für die Entscheidungsvariablen die besten Lösungen des vorher angewandten Optimierungsverfahrens dienen.

parallel-hybride Optimierungsverfahren.
Es existieren wiederum mehrere Optimierungsverfahren, welche jeweils das Optimierungsproblem in seiner Gesamtheit oder ein Teilproblem dessen betrachten und i. d. R. zu geeigneten Zeitpunkten Informationen über den aktuellen Lösungsverlauf und die besten gefundenen Lösungen austauschen. Diese besten Lösungen können bei gemeinsamer Bearbeitung des Optimierungsproblems als neue Startvektoren für den weiteren Suchverlauf der betreffenden Optimierungsverfahren mit schlechteren Lösungen dienen.

Jede dieser Varianten von hybriden Optimierungsverfahren hat seine individuellen Vor- und Nachteile, welche bei der Auswahl und dem Einsatz des jeweiligen Verfahrens zu berücksichtigen sind. Entscheidendes Kriterium sollte aber die Dominanz des hybriden Optimierungsverfahrens gegenüber anderen Optimierungsverfahren sein. Für weitere und tiefgründige Informationen über hybride Optimierungsverfahren sei auf die entsprechende Literatur verwiesen, wie bspw. [Had01] und [Plo05].

Sind bereits mehrere verschiedene Optimierungsverfahren existent, bietet sich deren Verknüpfung im Rahmen eines separat-hybriden Optimierungsverfahrens an. Welche der beiden Varianten, seriell-hybrid oder parallel-hybrid, sich als geeigneter erweist, sollte jedoch im Vorfeld untersucht werden. Der Vielfältigkeit der Verfahrensumsetzungen sind jedoch kaum Grenzen gesetzt. Allerdings wird sich derzeit nur auf die Nutzung hybrider Optimierungsverfahren im Zusammenhang mit einkriteriellen Optimierungsverfahren konzentriert (siehe [Had01]). Der Autor konnte trotz intensiver Recherche in der Literatur und im Internet aktuell keine hybriden Optimierungsverfahren finden, welche mit verschiedenen mehrkriteriellen Optimierungsverfahren arbeiten. Solche hybriden Optimierungsverfahren, welche auch mehrkriterielle Optimierungsverfahren oder ein- und mehrkriterielle Optimierungsverfahren in Kombination nutzen, könnten somit zum Gegenstand anderer Arbeiten werden.

Die genaue Umsetzung der hybriden Optimierungsverfahren in dieser Arbeit ist in [Wac05] und Abschn. 5.3.6 nachzulesen.

4.5.2. Verteilte und parallele Optimierung

In diesem Abschnitt werden nicht alle Problemstellungen, welche sich im Rahmen der verteilten und parallelen Optimierung ergeben, abgehandelt. Diese Thematik ist dafür viel zu komplex. Selbst die vorhandene Literatur zu diesem Thema ist sehr weitreichend. Allerdings haben sich einige Werke hervorgetan, welche eine gute Übersicht liefern. Dies sind bspw. [Cor02], [HGZ05], [Shv06] oder [GYD+06].

Auf Grund der umfangreichen Problemstellungen, welche mit dem Thema der

verteilten und parallelen Optimierung in direkter Verbindung stehen, seien nachfolgend nur ausgewählte Prinzipien vorgestellt. Diese sollen zunächst einen kurzen Einblick in die verteilte und parallele Optimierung geben. Anschließend soll eine Möglichkeit aufgezeigt werden, welche eine universell einsetzbare Implementierung unabhängig von den gewählten, zu verteilenden bzw. zu parallelisierenden Optimierungsverfahren ermöglicht. Aus dieser Aussage wird bereits erkennbar, dass im Wesentlichen die bereits vorhandenen Optimierungsverfahren genutzt und nur um die Möglichkeit der Verteilung erweitert werden.

Netzwerkprotokolle, Programmierschnittstellen und Programmbibliotheken. Bevor sich mit der Frage einer möglichen Umsetzung der verteilten und parallelen Optimierung beschäftigt werden kann, soll zunächst geklärt werden, mit Hilfe welches *Netzwerkprotokolls* die Rechner eines genutzten Rechnernetzwerkes untereinander kommunizieren. Dazu müssen alle kommunizierenden Rechner die Nutzung des gleichen Netzwerkprotokolls ermöglichen, ansonsten würde der Datenaustausch nur schwer möglich sein. Als Netzwerkprotokoll kann u. a. das *Protokollpaar* `TCP/IP` (Abkürzung für Transmission Control Protocol/Internet Protocol [engl.]) genutzt werden, welches eine Implementierung der Schichten 3 und 4 des OSI-Modells bzw. der Transport- und Vermittlungsschicht des `TCP/IP`-Referenzmodell darstellt (siehe [Wei02]). Auf diesen Netzwerkprotokollen bauen auch *verschiedene Programmpakete* auf, welche die Arbeit mit homogenen und heterogenen Rechnernetzwerken erleichtern. Zwei dieser Programmpakete, `PVM` (Abkürzung für Parallel Virtuell Machine [engl.]) und `MPI` (Abkürzung für Message Passing Interface [engl.]), seien nachfolgend kurz vorgestellt, weil sie auch auf dem `CLiC` (Chemnitzer Linux Cluster) zur Verfügung stehen, um die Kommunikation in einem homogenen Rechnernetzwerk aus (vormals[62]) 528 vernetzten Ein-Prozessor-Rechnern [dt. cluster [engl.]] zu ermöglichen (siehe [Che07a]). Ein Ein-Prozessor-Rechner stellt dabei jeweils einen Knoten dar.

PVM. `PVM` ist ein public-domain-Programmpaket[63] [engl.], welches gemeinsam vom Forschungsverbund der Universität von Tennessee, dem Oak Ridge National Laboratory, der Emory Universität und der Carnegie Mellon Universität Anfang der 1990er Jahre entwickelt wurde (siehe [PVM07]). Es unterstützt heterogene und damit indirekt auch homogene Netzwerke als Grundlage für die Entwicklung von parallelen Programmen. Durch seine hohe Funktionalität unter Nutzung nur weniger Kommandos kann es universell eingesetzt werden. Wie sich dieser Funktionsumfang gestaltet ist bspw. in [Gei97] nachzulesen, welches von Geist, einem Entwickler von `PVM`, verfasst ist.

`PVM` ist für verschiedene Rechnertypen (Desktop-PC, Workstation, Parallelrechner, Vektorrechner, etc.) verfügbar und nutzt im Wesentlichen Unix-Sockets und

[62] Mittlerweile sind alle Rechnerknoten außer Betrieb genommen. Anfang Juli 2007 ist die Abschaltung des gesamten Clusters erfolgt. Alternativ zum `CLiC` ist jedoch der `CHiC` (Chemnitzer Hochleistungs-Linux-Cluster, siehe [Che]) am 7. Februar 2007 in Betrieb genommen worden, welcher ebenfalls ein Hochleistungsrechnen sowie eine verteilte und parallele Optimierung ermöglicht.

[63] Ein Werk ist public domain [engl.; Gemeinfreiheit [dt.]], sofern es keinem Urheberrecht mehr unterliegt (siehe [Sch06a]).

TCP/IP zur Kommunikation zwischen den unterschiedlichen Rechnern. Es besteht aus einem Dämon, der zur Programmlaufzeit auf jedem (physikalischen) Rechner/Knoten benötigt wird, und Programmbibliotheken zur Erstellung sowie Abarbeitung der parallelen Programme. Nachteilig ist das Fehlen von Vorkehrungen für die Fehlertoleranz (z. B. Fehlerbehandlung beim Absturz von Rechnern) und der Mangel an Methoden zur dynamischen Lastenverteilung innerhalb von PVM.

MPI. Im Gegensatz zu PVM besitzt MPI einen größeren Umfang an Kommandos und stellt somit mehr Möglichkeiten zur Gestaltung der Kommunikation bereit (siehe [MPI07] und [For94]). MPI legt dabei nur die Programmierschnittstelle fest und nicht das zu nutzende Netzwerkprotokoll oder die zu nutzende Implementierung. Dadurch existieren auch viele individuelle Realisierungen für unterschiedliche Rechnertypen, Betriebssysteme und Netzwerkprotokolle. Beschrieben ist der offene Standard des MPI bspw. in [GLS94] und [GLS99].

Seit dem Jahre 1997 ist die zweite Version der MPI-Schnittstelle namens *MPI-2* verfügbar. MPI-2 stellt eine Erweiterung von MPI dar, wodurch die Programmierschnittstelle vollständig kompatibel zur ersten Version von MPI ist (siehe [MPI07], [For98] und [GLT99]). In der zweiten Version des MPI ist u. a. eine dynamische Verwaltung der Prozesse (dynamische Knoten), Fehlertoleranz, Debug-Möglichkeiten, Interaktion mit dem zugrunde liegenden Betriebssystem sowie parallele Ein- und Ausgabe durch MPI-IO integriert wurden.

Bedingt durch die beiden Versionen des MPI und die Nichtexistenz einer Bindung an ein spezielles Betriebssystem oder Netzwerkprotokoll, entstanden vielfältige Implementierungen. Auf dem CLiC werden die Implementierungen MPICH (für die erste Version des MPI), MPICH2 (zur Unterstützung der zweiten Version des MPI) und *LAM* (unterstützt MPI- und MPI-2-Standard) bereitgestellt. Der Zugang zu den Knoten des CLiC erfolgt über einen Zugangsrechner unter Nutzung einer Variante des *PBS* (Abkürzung für Portable Batch System [engl.], siehe [Che07b]).

Ein Vorteil an den Bibliotheken MPI und PVM ist, dass sie auch in höheren Programmiersprachen, welche Zwischenkodes erzeugen, genutzt werden können. Dies geschieht durch die Bereitstellung von Schnittstellen zur Abbildung der betreffenden Funktionen bzw. Methoden und wird z. B. von der Programmiersprache C# unterstützt. Der dabei auftretende zusätzliche Zeitaufwand ist verhältnismäßig gering (siehe [WLR06]).

Weitere Programmpakete. Neben dem zur Nutzung des CLiC und CHiC vorhandenen Bibliotheken MPI und PVM existieren weitere alternative Programmpakete bzw. Programmierschnittstellen zur gemeinsamen Arbeit innerhalb eines Rechnernetzes. Beispiele hierfür sind:

TCGMSG.
: Ist eine einfache Message-Passing-Bibliothek für die Programmierung von parallelen Programmen auf Workstations sowie Parallelrechnern, wobei es für shared-memory-Rechner (dt. „Gemeinsamer-Speicher-Rechner“) optimiert ist. Es unterstützt z. Z. nur synchrones Senden und kann in C- und FORTRAN-Programmen genutzt werden (siehe [Pac07]);

`P4`. `P4` ist eine Bibliothek von Makros und Unterroutinen, welche vom Argonne National Laboratory zur Programmierung verschiedener Parallelrechner entwickelt wurde. `P4` unterstützt sowohl shared-memory-Rechner [engl.], basierend auf Monitoren [dt.; monitors [engl.]], sowie distributed-memory-Rechnern [engl.], unter Verwendung von Message-Passing (siehe [BBD+87]);

`ParMacs`.
Ist ein Projekt, dass stark an P4 angelehnt ist. Der essenzielle Unterschied sind die Menge der zusätzlich entwickelten Makros zum `P4`-System vom Argonne National Laboratory und vom GMD (siehe [HHS91]);

`OpenMP`.
Ist eine seit 1997 gemeinschaftlich von verschiedenen Hardware- und Kompiler-Herstellern entwickelte Programmierschnittstelle. Der Standard dient zur Shared-Memory-Programmierung in den Programmiersprachen `C`, `C++` und `FORTRAN` auf Multiprozessor-Rechnern. Während `MPI` auf Prozessebene vollzogen wird, findet die Parallelisierung mittels `OpenMP` auf Thread- bzw. Schleifenebene statt (siehe [Boa07]).

Faktoren und Strukturen. Die Verfahren der verteilten und parallelen Optimierung sind nach Auffassung des Autors im Wesentlichen von den relevanten Faktoren, wie

- gewählte Hierarchieform des verteilten und parallelen Optimierungsverfahrens,
- gewähltes zu verteilendes Optimierungsverfahren,
- gewähltes Netzwerkkommunikationsprotokoll, Programmierschnittstelle und Programmbibliothek sowie
- evtl. der vorhandenen, zu berücksichtigenden Netzwerktopologie

abhängig. Die Hierarchieform eines verteilten und parallelen Optimierungsverfahrens sei an dieser Stelle außer Acht gelassen. Es wird von einer Master-Slave-Struktur ausgegangen, bei der ein Master die Kontrolle über n_S Slaves [engl.] hat. Die wesentlichen Netzwerkkommunikationsprotokolle zur programmiertechnischen Unterstützung bei der Entwicklung von verteilten und parallelen Optimierungsverfahren wurden bereits zuvor aufgezeigt. Wird davon ausgegangen, dass die vorhandene, zu berücksichtigende Netzwerktopologie nicht veränderbar ist, bleibt als einziger Faktor auf den ein Entwickler von verteilten und parallelen Optimierungsverfahren Einfluss hat, das Optimierungsverfahren selbst. Dessen Struktur kann beliebig gestaltet sein. Folgende Ansätze haben sich dabei durchgesetzt:

- Verteilung der Lösungen:
  - mehrere, parallele Lösungen innerhalb des Optimierungsverfahrens:
    * triviale Parallelisierung (direkte Zuordnung der n_L Lösungen zu den n_S Rechenknoten) oder
    * nicht-triviale Parallelisierung (Optimierungsverfahren arbeitet zwar

intern mit mehreren, parallelen Lösungen kann aber auf Grund vielfältiger Abhängigkeiten nur wenige Lösungen für die parallele Optimierung nutzen),

  - eine Lösung innerhalb des Optimierungsverfahrens (Zur Nutzung im Rahmen der verteilten und parallelen Optimierung könnte das Optimierungsverfahren mehrfach mit verschiedenen Startvektoren instanziiert werden.);

- Verteilung des Algorithmus (Das gleiche Optimierungsverfahren wird mit jeweils verschiedenen Parametereinstellungen für das Optimierungsverfahren auf den n_S Rechenknoten[64] genutzt.) oder
- Spezifischer verteilter oder paralleler Optimierungsalgorithmus (Individuelle Anpassung eines Optimierungsverfahrens an die Gegebenheiten des genutzten Rechnernetzes oder die gewählte Hierarchieform).

Die vorangehende Aufzählung erhebt keinen Anspruch auf Vollständigkeit, soll aber aufzeigen, welche verschiedenen Vorgehensweisen prinzipiell möglich sind. Der Autor sieht die Methode der Verteilung der Lösungen als eine der sinnvollsten Methoden bei der verteilten und parallelen Optimierung an. Diese Methode ermöglicht es bereits im Vorfeld der Anwendung der Optimierung eine grobe Abschätzung über den Zeitbedarf $\overline{ct}_{speedup}$ bei der Verwendung von $n_S + 1$ Rechenknoten zu geben:

$$ct^{min}_{speedup} \quad \leq \quad \overline{ct}_{speedup} \quad \leq \quad ct^{max}_{speedup} \quad ,$$

wobei

$$\begin{aligned} ct^{min}_{speedup} &= \frac{ct_{real}}{n_S} + ct_{comm} \quad \text{und} \\ ct^{max}_{speedup} &= ct_{real} + ct_{comm} \end{aligned}$$

mit

$ct^{min}_{speedup}$ - kürzeste Zeitdauer die durch n_S Slaves erzielt werden kann,
$ct^{max}_{speedup}$ - längste Zeitdauer die durch n_S Slaves erreicht werden kann sowie
ct_{real} - Zeitdauer, die benötigt wird, wenn keine Verteilung erfolgt.

Die kürzeste Zeitdauer $ct^{min}_{speedup}$ kann erreicht werden, wenn alle Berechnungen gleichzeitig und gleich schnell ausgeführt werden. Zudem muss bei der zwischen dem Master und den Slaves auftretenden Kommunikation ein gleichzeitiges, paralleles und kollisionsfreies Senden von Informationen möglich sein. Die längste Zeitdauer $ct^{max}_{speedup}$ entsteht, wenn keine Verteilung mehrerer Lösungen möglich und somit nur eine sequentielle Vorgehensweise anwendbar ist. Demnach sind die wesentlichen Faktoren für die Erzielung kürzerer Optimierungsdauern und die Bestimmung des realen Zeitbedarfs $ct^{real}_{speedup}$:

[64] Es wird innerhalb dieser Arbeit davon ausgegangen, dass eine Instanz des Optimierungsverfahrens auf genau einem Rechenknoten läuft. Ein Rechenknoten bestehe dabei aus einem Rechner mit einem Hauptprozessor. Besteht ein Rechner aus mehreren Hauptprozessoren, wird o. B. d. A. angenommen, dass jeder Hauptprozessor einen Rechenknoten bildet.

- der Grad der Parallelisierbarkeit des Optimierungsverfahrens sowie
- die Dauer der Berechnung einer bestimmten Lösung.

Für die zu nutzenden $n_S + 1$ Rechenknoten wird im Rahmen dieser Arbeit davon ausgegangen, dass entweder

- die Anzahl der zur Optimierung bereitgestellten Rechenknoten $n_S + 1$ vom genutzten Netzwerk vorgegeben wird oder
- das Optimierungsverfahren sagt, wie viele Rechenknoten $n_S + 1$ es benötigt und das Rechnernetzwerk entsprechend viele bereitstellt.

Anzumerken sei an dieser Stelle noch, dass die zuvor genannten Ansätze ihre individuellen Vor- und Nachteile besitzen, welche nicht näher betrachtet werden sollen. Dabei spielt u. a. der auftretende Kommunikationsaufwand eine entscheidende Rolle, welcher zum Flaschenhals der verteilten und parallelen Optimierung werden kann.

Kommunikationsablauf bei Master-Slave-Hierarchien. Bevor der gewählte Ansatz betrachtet wird, sei zunächst noch kurz auf den grundsätzlichen Kommunikationsablauf zwischen dem Master und den n_S Slaves eingegangen. Zur Kommunikation selbst werden stets *Steuerbefehle* verwendet. Diesen folgen die evtl. benötigten *Datenwerte zu diesen Steuerbefehlen*. Ein Steuerbefehl im Bezug auf die simulationsbasierte Optimierung könnte bspw. angeben, dass ein Slave einen Vektor mit Werten für die Entscheidungsvariablen empfangen soll. Diesem Steuerbefehl folgen dann die konkreten Werte für die Entscheidungsvariablen. In umgekehrter Richtung sendet ein Slave über einen betreffenden Steuerbefehl die zu bewertenden und nicht zu bewertenden Ausgabewerte eines Simulationsmodells an den Master. Dadurch ist es vollkommen unerheblich, ob das Optimierungsverfahren auf dem Master ein ein- oder mehrkriterielles Optimierungsverfahren ist.

Neuer, gewählter Ansatz. Innerhalb dieser Arbeit wird ein Ansatz gewählt, welcher es einem beliebigen vorhandenen Optimierungsverfahren erlaubt für die verteilte und parallele Optimierung genutzt werden zu können. Das Optimierungsverfahren OV muss lediglich verschiedene Informationen bereitstellen, welche Auskunft darüber geben, ob es verteilbar ist und wie viele Lösungen $n_S^{max}(OV)$ maximal verteilt verarbeitet werden können. Erfolgt innerhalb eines Laufes der simulationsbasierten Optimierung die Nutzung mehrerer, verschiedener Optimierungsverfahren gibt das Optimierungsverfahren mit der kleinsten Anzahl parallel verarbeitbarer Lösungen den Grad der Parallelisierbarkeit[65] an:

$$n_S^{max} = \min_{i \in \{OV\}} n_S^{max}(i) \quad .$$

Es ist vorgesehen, dass den Optimierungsverfahren im Vorfeld der Optimierung

[65] Der Grad der Parallelisierbarkeit gibt die Anzahl der (höchstens) zeitgleich parallel durchführbaren Berechnungen an.

Abb. 4.9

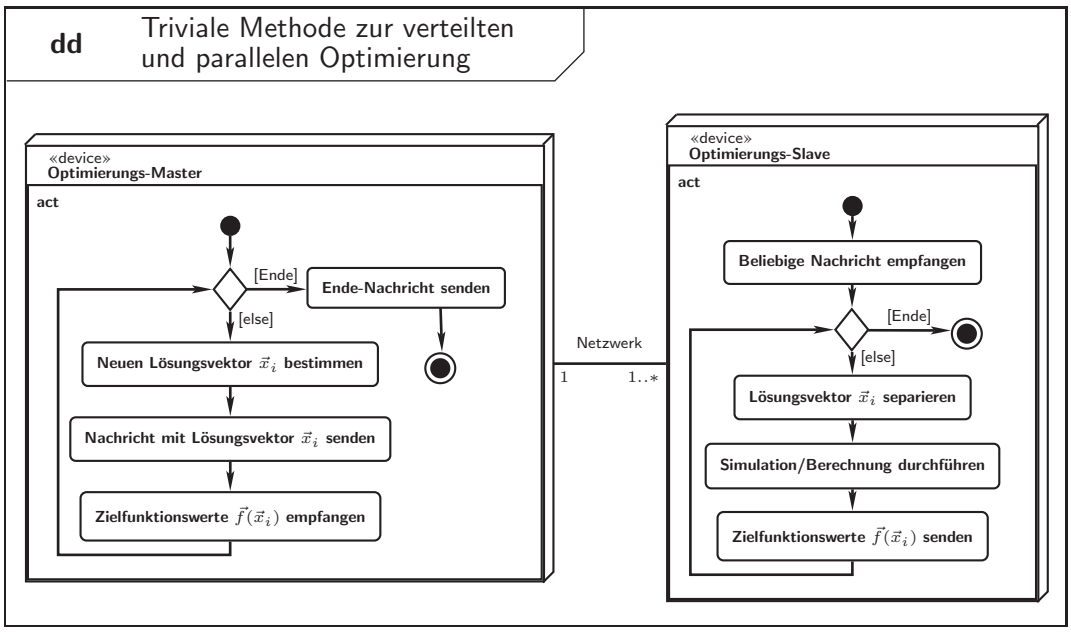


Abb. 4.9.: *Triviale Verteilung der Lösungsvektoren bei verteilter und paralleler Optimierung (UML-Verteilungsdiagramm)*

gesagt wird, dass eine verteilte und parallele Optimierung mit n_S Rechenknoten angestrebt wird. Ein Optimierungsverfahren OV kann dann sagen, wie viele Lösungen $n_S^{max}(OV)$ es maximal parallel verarbeiten kann. Ist $n_S \geq n_S^{max}(OV)$ stehen ausreichend Rechenknoten für dieses Optimierungsverfahren bereit[66]. Für $n_S < n_S^{max}(OV)$ muss durch einen Steuerungsmechanismus für eine entsprechende mehrfache Zuordnung von unterschiedlichen Lösungen zu Rechenknoten gesorgt werden. Prinzipiell kann ein Optimierungsverfahren, welches mit nur einer Lösung arbeitet, jedoch entscheiden,

- ob es die Information über die Parallelisierung ignoriert und weiterhin mit nur einer Lösung arbeitet,
- ob es eine Verteilung durch Parallelisierung der Lösungen, welche aus zufällig gewählten Startvektoren gebildet wurden, anstrebt oder
- ob es eine Verteilung durch parallele Existenz der gleichen Optimierungsverfahrensinstanzen mit verschiedenen Parametereinstellungen erzielen möchte.

Abschließend sei bemerkt, dass durch den gewählten Ansatz der verteilten und parallelen Optimierung die bereits zuvor erwähnte leitende Master-Instanz benötigt wird. Eine schrittweise Ausführung des Optimierungsprozesses, wie sie im implementierte Softwaresystem `CAOS` durchgeführt wird, wirkt sich dabei unterstützend auf den Master-Prozess aus und macht nur geringfügige Änderungen am Optimierungsmanager (siehe 5.3.6) notwendig. Die Nutzung des objektorientierten Programmierparadigmen bei der Implementierung des Optimierungsmanagers erwies sich zusätzlich als hilfreich. Auf die vorgenommene Implementierung inner-

[66] Die überzähligen Rechenknoten werden währenddessen ignoriert oder für andere Anwender freigegeben.

halb des Softwaresystems CAOS wird im Abschn. 5.3.6, welcher sich mit der softwaretechnischen Umsetzung des vorgestellten Verfahrens beschäftigt, noch einmal eingegangen.

4.6. Zusammenfassung

In diesem Kapitel wurde sich mit den unterschiedlichen Vorgehensweisen verschiedener Optimierungsverfahren beschäftigt, womit ein wesentlicher Teil des theoretischen Bereiches zum Thema der simulationsbasierten Optimierung zunächst abgeschlossen ist.

Es erfolgte eine Gegenüberstellung von exakten und heuristischen Optimierungsverfahren, wobei die exakten Verfahren auf Grund ihrer eingeschränkten Einsatzmöglichkeiten im Rahmen der simulationsbasierten Optimierung im weiteren Verlauf des Kapitels nicht näher betrachtet wurden. Für die heuristischen Optimierungsverfahren wurden die wesentlichen Merkmale von Eröffnungs- und Verbesserungsverfahren aufgezeigt, um deren gesonderte Bedeutung kenntlich zu machen. Bei den heuristischen Verbesserungsverfahren kam der Fakt zum Tragen, dass sie sich vorrangig nach der Anzahl von Lösungen, mit denen sie arbeiten, einteilen lassen.

Wie bereits im Kap. 3 angemerkt wurde, ist die simulationsbasierte Optimierung, bedingt durch die im Vergleich zu analytischen Berechnungsverfahren oftmals sehr hohen Laufzeitanforderungen der Simulation, ein sehr aufwändiges Verfahren der Problemlösung. Um dieser Problematik der hohen Laufzeiten entgegen zu wirken, wurden innerhalb dieses Kapitels auch verschiedene Methoden zur hybriden Optimierung sowie zur verteilten und parallelen Optimierung vorgestellt.

Als wesentliches Merkmal der hybriden Optimierung wurde dabei die gezielte Kombination mehrerer, ausgewählter Optimierungsverfahren innerhalb eines Optimierungsverfahrens angesehen. Wie im Abschn. 6.5.1 erkennbar sein wird, kann nicht genau gesagt werden, ob die hybride Optimierung stets besser oder gleich gut wie das beste ausgewählte Einzelverfahren ist. Aus diesem Grunde wurden auch die Methoden zur verteilten und parallelen Optimierung innerhalb dieses Kapitel vorgestellt, welche nach Meinung des Autors zumeist ein größeres Potential bei der Verkürzung der Dauer eines Laufes zur simulationsbasierten Optimierung besitzen und dennoch zufrieden stellende Lösungen liefern. Das wesentliche Ziel der vorgestellten Vorgehensweisen bei den verteilt und parallel arbeitenden Optimierungsverfahren ist es, mit Hilfe mehrerer, in einem Netzwerk kooperierender Rechner, mehrere Berechnungen oder Simulationen parallel durchführen zu lassen. Dabei dienten mehrere Slave-Rechenknoten zur verteilten und parallelen Durchführung mehrerer Simulationsexperimente und ein Master-Rechenknoten zur Steuerung des Kommunikationsablaufes zwischen dem Master-Rechenknoten und den Slave-Rechenknoten sowie der Ausführung des verteilten Optimierungsalgorithmus.

Durch diese parallel arbeitenden Verfahren wird es möglich, auch für laufzeitintensive Simulationen eine Optimierung in einem zeitlich beschränkten Intervall durchzuführen. Diese Betrachtungen waren notwendig, weil sich u. a. auch für die untersuchten Simulationsmodelle gezeigt hat, dass die Laufzeiten der zugehörigen Simulatoren höher als erwartet ausfielen.

Kapitel 5. Implementierung einer Software zur simulationsbasierten Optimierung

Inhalt

5.1	*Einführung*	*131*
5.2	*Kurzer Überblick über ausgewählte Softwaresysteme zur Simulation und Optimierung*	*133*
5.3	*Das Softwaresystem `CAOS`*	*136*
5.3.1	*Die Programmiersprache `C#`*	*136*
5.3.2	*Modularer Aufbau des Softwaresystems (Bibliotheken)*	*137*
5.3.3	*Die Bibliothek `CAOSStartUp`*	*139*
5.3.4	*Die Bibliothek `Utilities`*	*139*
5.3.5	*Die Bibliothek `Network`*	*142*
5.3.6	*Die Bibliothek `CalculationAssessmentOptimisation`*	*144*
5.3.7	*Anwendungsfälle und Nutzungsmöglichkeiten*	*168*
5.4	*Zusammenfassung*	*169*

5.1. Einführung

In diesem Kapitel soll das bereits an verschiedenen Stellen erwähnte Softwaresystem `CAOS` (`CalculationAssessmentOptimisationSystem`) vorgestellt werden, welches im Rahmen dieser Arbeit entwickelt wurde. Mittels `CAOS` ist es im Wesentlichen möglich

- eine spezielle Art der rechnerinternen *Modellierung*,
- eine *numerische Berechnung* oder *Simulation*,
- eine *Optimierung* der Werte der Entscheidungsvariablen sowie
- eine *Bewertung* der Berechnungs- oder Simulationsergebnisse

des Modells eines Realsystems vorzunehmen. `CAOS` zielt damit direkt auf die Durchführung von Läufen zur simulationsbasierten Optimierung ab und somit auf die

Umsetzung der in den vorangehenden Kapiteln 2, 3 und 4 beschriebenen Methodiken. Dazu besitzt CAOS eine rechnerinterne Abbildungsmöglichkeit zur einfachen und gleichstrukturierten Überführung eines Modells von einem Realsystem über eine mathematische Darstellung in eine rechnerinterne Modellbeschreibung. Zusätzlich werden die Informationen, welche bspw. die logischen Beziehungen des Realsystems widerspiegeln, im zugehörigen Simulationsmodell verankert. Die rechnerinterne Modellbeschreibung enthält zudem auch die notwendige Beschreibung des jeweiligen Optimierungsproblems. Das Simulationsmodell kann dann unter Beachtung des Aufbaus des rechnerinternen Modells im Rahmen der Softwareentwicklung innerhalb einer beliebigen Programmiersprache implementiert werden.

Zur Durchführung eines Laufes der simulationsbasierten Optimierung muss im Vorfeld nicht zwingend unterschieden werden, welche Art der simulationsbasierten Optimierung, iterativ oder sequentiell (vgl. Abschn. 2.3), zum Einsatz kommen soll. Diese Wahl kann dem Anwender überlassen werden, so dass Simulation und Optimierung, wie in den vorangegangenen Kapiteln bereits geschehen, getrennt betrachtet werden und eigenständige Softwaresysteme darstellen. Auch mit CAOS ist diese getrennte Nutzung von Simulation und Optimierung möglich. Allerdings steht die Nutzung der iterativen Vorgehensweise bei der simulationsbasierten Optimierung im Vordergrund.

Die Kopplung von Simulation und Optimierung geschieht bei getrennter Nutzung von Simulation und Optimierung i. Allg. über eine (externe) Programmier- bzw. Kommunikationsschnittstelle, welche auf Dateien, gemeinsam genutzten Speicherbereichen, o. Ä. basiert. Hierfür müssen zudem jeweils eine externe Beschreibung des Simulationsmodells und des Optimierungsproblems in einer für CAOS geeigneten Form vorliegen. Gerade die Kommunikationsschnittstelle und die verschiedenen notwendigen Dateien für eine getrennte Simulation und Optimierung sind sehr fehleranfällig und meist nutzerunfreundlich angelegt. Mittels einer automatischen Generierung dieser Dateien kann dieses Manko behoben werden. Dies ist im CAOS möglich. Allerdings wirft bspw. die automatische Generierung eines Optimierungsproblems und des zugehörigen Simulationsmodells, auf Grund der Menge von verfügbaren Softwaresystemen, das Problem auf, dass beide Beschreibungen jeweils immer nur für eine geringe Auswahl der verfügbaren Softwaresysteme automatisch erzeugt werden können. Es fehlt eine einheitliche Schnittstellen zum Datenaustausch der Softwaresysteme untereinander. Dadurch wird der Anwender in seinen Auswahlmöglichkeiten mitunter unfreiwillig auf die vom Softwareersteller vorgegebenen Auswahlmöglichkeiten eingeengt. Mit der im Abschn. 2.4 beschriebenen rechnerinternen Darstellung umgeht das Softwaresystem CAOS diese Schwachstelle anderer Softwaresysteme, indem es eine integrierte, erweiterbare und für Neu- bzw. Weiterentwicklungen offene, komplette und kompakte Software-Lösung zur simulationsbasierten Optimierung anbietet. Darüber hinaus stellt CAOS Möglichkeiten bereit, um auch andere Softwaresysteme entsprechend der Schnittstellen des CAOS zu koppeln und so eine universelle Anwendbarkeit zu ermöglichen. Dem Autor selbst ist derzeit kein vergleichbares Softwaresystem mit ähnlichem Funktionsumfang bekannt, welches die gleiche oder eine vergleichbare Vorgehensweise bei der simulationsbasierten Optimierung verfolgt.

CAOS ist durch die Nutzung objektorientierter Programmierung so entwickelt, dass es stets beliebig erweitert werden kann, z. B. durch die Hinzunahme weiterer Klassen für Simulatoren, Optimierungsverfahren, numerischer Berechnungsvorschriften, Bewertungsmethodiken, u. Ä. Es stellt dazu einerseits verschiedene virtuelle und abstrakte Basisklassen (siehe [Bal05]) mit entsprechenden Methodenrümpfen bereit und ermöglicht es andererseits bereits vorhandene Klassen in anderen vergleichbaren Softwaresystemen zu nutzen oder diese dort entsprechend zu erweitern. Dafür wurde das CAOS in verschiedene Bibliotheken aufgeteilt, welche nachfolgend im Abschn. 5.3 vorgestellt werden. Zunächst sei jedoch im Abschn. 5.2 ein kurzer Überblick über aktuell existierende Softwaresysteme zur Simulation und Optimierung gegeben. Dadurch soll die Abgrenzung des Softwaresystems CAOS von anderen Softwaresystemen erfolgen.

5.2. Kurzer Überblick über ausgewählte Softwaresysteme zur Simulation und Optimierung

Nachfolgend soll sich auf Software für das Verfahren der iterativen Simulation und Optimierung bezogen werden, weil diese Vorgehensweise auch in der vorliegenden Arbeit angewandt wird. Bei der Vorstellung der aktuell existierenden Software wird dabei auf die Problematik der Simulation und Optimierung getrennt eingegangen, weil aktuell kein Softwaresystem bekannt ist, welches Simulation und Optimierung in der Art und Weise gemeinsam anbietet, dass die sich aus der Kombination der beiden Softwarekomponenten ergebenden Möglichkeiten genutzt werden. Es existieren lediglich getrennte Software-Lösungen für Simulation und Optimierung, welche auf geeignete Art und Weise miteinander gekoppelt werden müssen.

Softwaresysteme zur Optimierung. In Tab. 5.1 ist ein Auszug über bekannte Softwaresysteme zur Optimierung gegeben. Bereits diese Übersicht zeigt, dass der Anwender meist kaum oder nur eine kleine Auswahlmöglichkeit darüber hat, welche Optimierungsverfahren er nutzen möchte. Gerade für erfahrene oder geschulte Anwender sollte diese Auswahlmöglichkeit sowie die Möglichkeit zur Veränderung der Parameter der angebotenen Optimierungsverfahren aber gegeben sein. Im Gegensatz dazu bieten auch Softwaresysteme zur Simulation weitaus mehr Einstellungsmöglichkeiten als oftmals angeboten werden, wodurch sie eine größere Mächtigkeit für den Anwender besitzen würden.

Anzumerken ist, dass die Übersicht in Tab. 5.1 aus dem Jahre 2001 stammt. Grund dafür ist, dass der Autor selbst nach längerer Literaturrecherche keine aktualisierte Übersicht gefunden hat und es nur schwer möglich ist, von den Software-Entwicklern kommerzieller Software die gewünschten Informationen über die verwendeten Optimierungsverfahren zu erhalten. Die wenigen vorhandenen und umfassenden Literaturquellen beschränken sich auf [LK00, S. 664, Tab. 12.11], [Fu01a] und [Man01].

Tab. 5.1

Softwaresystem zur Optimierung	Mögliche Optimierungsverfahren	Anbieter	Literaturverweis(e)
AutoStat^{TM}	Enumeration, ES, GA	AutoSimulations, Inc.	[Aut99], [Roh99]
OptQuest®	Scatter search/TS, Neuronale Netzwerke	Optimization Technologies, Inc.	[Opt00], [Opt07]
(Simul8) OPTIMIZ	Neuronale Netzwerke	Visual Thinking International Ltd.	[SIM07]
SimRunner2	EA, GA	PROMODEL Corp.	[Ges01], [Pro07b]
WITNESS Optimizer	Enumaration, HC, SA, TS	Lanner Group, Inc.	[Lan07]
ISSOP	Enumeration, ES, GA, GD	Dualis IT Solutions	[Sol07]

Legende:
Genauere Angaben zu den Abkürzungen für die Bezeichnung der Optimierungsverfahren sind in Abschn. 4.3.5 zu finden.

Tab. 5.1.: *Kommerzielle Software zur simulationsbasierten Optimierung (Auszug, nach [Fu01a] und [Man01])*

Softwaresysteme zur Simulation. Software zur Erstellung von Simulationsmodellen bzw. den zugehörigen Simulatoren selbst existiert mittlerweile in vielfältiger Art und Weise. Dabei kann einerseits auf *universell verwendbare Simulationssprachen* oder aber auf Programmbibliotheken für *konventionelle Programmiersprachen* zurückgegriffen werden. Ist die Ausführungsgeschwindigkeit eines Simulationsexperimentes von besonderem Interesse, sollte auf konventionelle Programmiersprachen zurückgegriffen werden. Dadurch lassen sich besonders sog. *Spezialsimulatoren*, welche i. Allg. nur für einen (stark) eingeschränkten Problemkreis ausgelegt sind, gut erstellen und im Hinblick auf ihre Programmlaufzeit gut optimieren. Simulationssprachen hingegen besitzen eine größere Mächtigkeit, in dem sie die rechnerinterne Abbildung beliebiger Simulationsmodelle zulassen. Es kann prinzipiell gesagt werden, dass sich mit zunehmender Verallgemeinerung eines Simulators auch dessen Ausführungsgeschwindigkeit verringert. Welche wesentlichen, prinzipiellen Unterschiede und Charakteristiken außerdem noch zwischen spezialisierten Simulationssprachen und konventionellen Programmiersprachen bestehen, ist in Tab. 5.2 zu erkennen.

Tab. 5.2

Spezialisierte Simulationssprachen	Konventionelle Programmiersprachen
• abstrakt • schneller änderbar • weniger fehleranfällig bzgl. Erstellung eines Simulationsmodells • mitunter hohe Anschaffungskosten	• effiziente Programme • flexibel • günstig, stets verfügbar • oftmals plattformunabhängig • um beliebig viele Programmzusätze erweiterbar • mitunter hohe Wartungskosten

Tab. 5.2.: *Gegenüberstellung von spezialisierten Simulationssprachen und konventionellen Programmiersprachen*

Bei der Wahl der Erstellung eines Simulationsmodells muss sich zudem entschieden

werden, welches spezielle Modellierparadigma[67] sich zu Nutze gemacht werden soll. Bei der Verwendung von Simulationssprachen ist man diesbezüglich meist an ein einziges Modellierparadigma gebunden, währenddessen man bei der Nutzung konventioneller Programmiersprachen im Vorfeld der Erstellung diesbezüglich offen ist.

Weitere Unterscheidungsmerkmale der Simulatoren[68] bestehen in der Möglichkeit der Erstellung eines Simulators und der Eingabe der Daten eines Simulationsmodells. Mittlerweile sind einige Simulationssprachen, aber auch Spezialsimulatoren, in Softwaresysteme zur Simulation mit entsprechenden graphischen Oberflächen eingebettet, wodurch sich jedoch zumeist die Ausführungsgeschwindigkeit eines Simulationslaufes verringert. Im Softwaresystem `CAOS` wird versucht dieser Verringerung entgegenzuwirken, indem ein Simulationsexperiment oder ein Lauf der simulationsbasierten Optimierung wahlweise auch ohne eine graphische Oberfläche durchgeführt und die graphische Aufbereitung der Ergebnisse optional erst danach getätigt werden kann.

Anzumerken sei, dass in den Internet-Quellen der Hersteller von kommerziellen Softwaresystemen zur Simulation nähere Informationen zu deren Softwaresystemen zu finden sind. Nach Ansicht des Autors sind dies bspw. folgende Internet-Seiten von Herstellern mit (größeren) vergleichbaren, kommerziellen Softwaresystemen zur Simulation

Enterprise Dynamics.
nutzt wie `CAOS` ereignisgesteuerte Simulation (siehe [Dyn07]),

Goldsim.
nutzt wie `CAOS` ereignisgesteuerte Simulation (siehe [Gro07]) sowie

Arena.
eignet sich wie `CAOS` zur Untersuchung von Produktions- und Lagerhaltungssystemen (siehe [RA07]).

Des Weiteren ist in [Kru01, S. 22f.] eine generationsbasierte Übersicht über die Entstehung der verschiedenen Simulationssoftwaresysteme und deren Simulationssprachen zu finden.

Fazit. Die logische Konsequenz aus den verschieden Vor- und Nachteilen existierender Softwaresysteme zur Simulation und Optimierung wäre die Entwicklung einer Software, welche einen Mittelweg verfolgt, d. h.

- zum einen die vorhandenen Vorteile der Softwaresysteme weitestgehend umsetzt,
- deren Nachteile versucht zu umgehen und

[67] Als Modellierparadigmen standen die ereignisorientierte, die prozessorientierte oder die aktivitätenorientierte Zeitfortschreibung der Simulation zur Auswahl (vgl. Zeitfortschreibung bei diskreter Simulation im Abschn. 3.3).
[68] Als Simulator sei das zur Laufzeit eines Simulationslaufes vorliegende Programm zu einem gegebenen Simulationsmodell bezeichnet.

- als weiteres Ziel die Vereinigung der beiden Ansätze, Simulation und Optimierung, zur Durchführung von Läufen zur simulationsbasierten Optimierung verfolgt.

Aus dieser Notwendigkeit zur Schaffung einer neuen Software basiert die Entwicklung des Softwaresystems CAOS. Es stellt einerseits verschiedene Optimierungsverfahren bereit, welche ihrerseits die individuell zu implementierenden Simulatoren verwenden und das daraus resultierende (rechnerinterne) Modell für die Durchführung einer simulationsbasierten Optimierung nutzt. Die zur Lösung eines Problems zu implementierenden speziellen Simulatoren, können dabei allerdings auf bereits vorhandene Klassen von anderen Simulatoren und die vorhandenen Bibliotheken zurückgreifen. Zudem erlauben die Bibliotheken des CAOS (siehe Abb. 5.3.2) eine universelle Nutzbarkeit zur schnellen Schaffung neuer Spezial- oder Universalsimulatoren für verschiedene Anwendungsfälle. Dieses Softwaresystem CAOS mit seinen Bibliotheken und Möglichkeiten wird nachfolgend vorgestellt.

5.3. Das Softwaresystem CAOS

Das Hauptziel, welches mit dem Softwaresystem CAOS verfolgt wurde und wird[69], ist die Schaffung einer plattformunabhängigen Software zur Durchführung von Läufen der simulationsbasierten Optimierung, speziell der iterativen Vorgehensweise, anhand der gewählten rechnerinternen Modellabbildung. CAOS zielt somit direkt auf die Umsetzung der in den Kapiteln 2, 3 und 4 vorgestellten Konzepte ab.

5.3.1. Die Programmiersprache C#

Die gewünschte Plattformunabhängigkeit vom CAOS wird durch die Nutzung der Programmiersprache C# erreicht, welche sich in einer ständigen Weiterentwicklung befindet und aktuell in der Sprachversion 2.0 vorliegt (siehe [Sch06c] und [Sch07c]). Durch die bis dato rasche Entwicklung der Sprachversionen sowie die nicht so rasche und bisher nur teilweise Umsetzung einiger Klassen-Bibliotheken des .NET-Framework in anderen Laufzeitumgebungen wurden im CAOS lediglich Sprachelemente der Sprachversion 1.1 und auf Windows®-basierten Betriebssystemen ausgewählte Klassen des .NET-Framework 2.0 genutzt. Vom .NET-Framework 2.0 erwies dabei nur die Nutzung der neuen Klassen zur weiterreichenden Erstellung graphischer Oberflächen und der verbesserten Nutzung von *threads* [engl.; leichtgewichtigen Prozessen [dt.]] als hilfreich. Auf die neuen Möglichkeiten der Sprachversion 2.0, u. a. zur Erzeugung von generics [engl.; generische Klassen [dt.]], wurde trotz der sich ergebenden Vorteile der leichteren Wartbarkeit des Software-

[69] Die Entwicklung des Softwaresystems hat einen Stand erreicht, der eine weitestgehend stabile Nutzung der Software erlaubt. Allerdings gilt die Entwicklung einer Software nie als abgeschlossen, weil jeder Zeit neue Nutzungsanforderungen und -wünsche auftreten können. Dies gilt auch für das Softwaresystem CAOS, von welchem sich bereits Teile im praktischen Einsatz befinden und dementsprechend weiterentwickelt werden müssen.

systems verzichtet, weil sie tiefgreifende Eingriffe in den Kern des Softwaresystems nach sich gezogen hätten. Stattdessen wurden entsprechende Hilfsklassen entwickelt, welche plattformunabhängig einsetzbar sind.

Die Programmiersprache C# bzw. deren Zwischensprache CIL (siehe Anhang C) bringt verschiedene Besonderheiten wie *Unabhängigkeit von der Rechnerplattform* zur Entwicklungs- und Laufzeit sowie *Unabhängigkeit von der Programmiersprache*[70] zur Entwicklungszeit mit sich. Daraus ergeben sich verschiedene Vor- und Nachteile. Die für die innerhalb dieser Arbeit durchgeführten Untersuchungen wesentlichen sind in Tab. 5.3 überblicksmäßig zusammengefasst. Weitere Ausführungen zur Programmiersprache C# und den damit zusammenhängenden Komponenten sind im Anhang C dieser Arbeit zu finden.

Tab. 5.3

Vorteile	Nachteile
• objektorientierte Programmiersprache • weitestgehend plattformunabhängige[71] Softwareentwicklung mit meist frei[72] verfügbaren Entwicklungs- und Laufzeitumgebungen (siehe Tab. C.1) • automatische Ressourcenverwaltung [dt.; garbage collection [engl.]] • schnelle, einheitliche Entwicklung graphischer Oberflächen • einheitliche und vereinfachte Behandlung von rechnerinternen Ereignissen[73]	• um bis zu 20% höhere Ausführungszeit und längere Antwortzeiten auf rechnerinterne Ereignisse im Vergleich mit einem ähnlichem C++-Programms (durch Übersetzung des Zwischenkodes erst zur Laufzeit, automatische Ressourcenverwaltung, u. Ä. (siehe [DB04]))

Tab. 5.3.: *Gegenüberstellung der Vor- und Nachteile der Programmiersprache C#*

5.3.2. Modularer Aufbau des Softwaresystems (Bibliotheken)

Die im CAOS entwickelten Bibliotheken[74] werden in den nachfolgenden Unterabschnitten etwas detaillierter betrachtet, wobei deren wesentliche Inhalte an dieser Stelle schon einmal kurz vorgestellt seien:

CAOSStartUp.
: Bibliothek, welche den Zugriff auf das Softwaresystem CAOS von verschiedenen Anwendungsfällen aus vereinheitlicht.

[70] Eine Übersicht über die verfügbaren Programmiersprache ist im Internet unter [Sch07d] zu finden.

[73] Dafür wurde von Novell das Mono-Projekt, welches mit der plattformunabhängigen Software Mono eine Open-Source-Portierung des Microsoft®-.NET-Frameworks bereitstellt. Mono basiert, ebenso wie das Microsoft®-.NET-Framework, auf dem common-language-infrastructure-standard [engl.] (siehe [pro07a], [DB04]).

[73] Das Wort „frei“ steht an dieser Stelle im direkten Zusammenhang mit dem Wort freeware [engl.]. Freeware [engl.] ist meistens proprietär und wird vom Urheber zur kostenlosen Nutzung zur Verfügung gestellt.

[73] In diesem Zusammenhang werden rechnerinterne Ereignisse durch Nutzer-Interaktionen, das Betriebssystem oder angeschlossene Peripheriegeräte ausgelöst (vgl. im Gegensatz dazu Simulationsereignisse (Abschn. 3.3) und Optimierungsereignisse (4.3.1)).

[74] In der Programmiersprache C# werden die dynamisch eingebundenen Bibliotheken [dt.; Dynamic Link Library [engl., kurz: DLL]], die Anwendungsprogramme, u. Ä. im Zwischenkode der CIL vorliegende Dateien auch assemblies [engl.] genannt.

`Utilities`.
: Bibliothek mit verschiedenen Hilfsklassen für die schnellere und einfachere Softwareerstellung bei der Arbeit mit graphischen Oberflächen, der Erzeugung von Zufallszahlen, der Ein- und Ausgabe von Dateien, u. Ä. Diese Bibliothek ist nicht ans `CAOS` gebunden.

`Network`.
: Bibliothek für den einheitlichen Umgang mit verschiedenen Netzwerkprotokollen und -nachrichten. Diese Bibliothek ist nicht ans `CAOS` gebunden.

`CalculationAssessmentOptimisation`.
: Hauptbibliothek des Softwaresystems `CAOS`, welche die Optimierungsverfahren, Berechner, Simulatoren und Bewerter beinhaltet. Diese Bibliothek ist nicht ans `CAOS` gebunden, benötigt aber die Bibliotheken `Network` und `Utilities`.

Die Aufteilung in verschiedene, teilweise voneinander abhängige Bibliotheken dient neben der logischen Gliederung der Software hauptsächlich der Erhöhung der Wiederverwendbarkeit verschiedener Teile des Softwaresystems `CAOS`. In Abb. 5.1 sind diese verschiedenen Bibliotheken mit ihren Abhängigkeiten und ihren wichtigsten Teilbibliotheken dargestellt.

Abb. 5.1

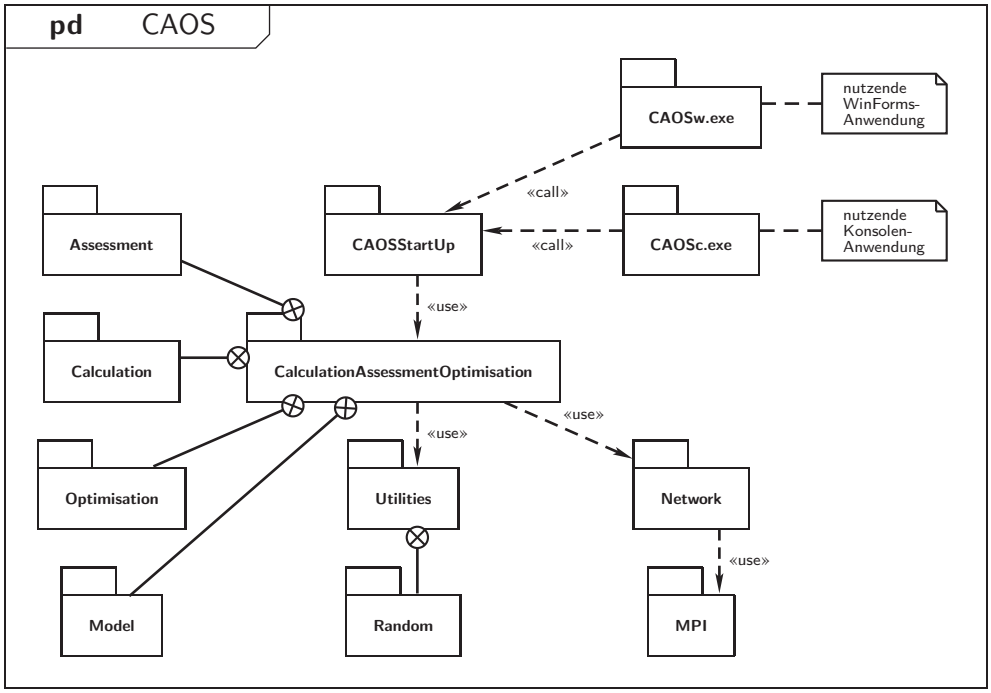


Abb. 5.1.: *Übersicht über die Pakete im CAOS (UML-Paketdiagramm)*

Die Bibliotheken sind intern in verschiedene *Namensräume* [dt.; *namespaces* [engl.]] eingeteilt, um die große Menge an Klassen sowie die enthaltenen Informationen übersichtlicher zu gestalten und eine logische Trennung der Klassen innerhalb der betreffenden Bibliothek zu ermöglichen. Aus diesem Grund wird nachfolgend teilweise nicht nur von den betreffenden Bibliotheken, sondern auch von bestimmten Namensräumen einer Bibliothek die Rede sein.

5.3.3. Die Bibliothek CAOSStartUp

Die Bibliothek CAOSStartUp dient lediglich dem definierten Zugriff auf die CAO-Bibliothek von verschiedenen Anwendungsfällen aus (siehe Abschn. 5.3.7). Bei der Einbindung der CAO-Bibliothek in andere Programme kann die CAOSStartUp-Bibliothek als Vorlage dienen, um aufzuzeigen, wie sich die Nutzung des Softwaresystems CAOS von anderen Softwaresystemen aus gestalten könnte. Im Zusammenhang mit dem Softwaresystem CAOS kapselt die CAOSStartUp-Bibliothek die Nutzung als Konsolen-Anwendung mittels der Anwendungsbibliothek[75] CAOSc.exe sowie die Nutzung als WindowsForms-Anwendung[76] mittels der Anwendungsbibliothek CAOSw.exe. Die zugehörigen Anwendungsbibliotheken der Anwendungsbibliothek CAOSc.exe und der Anwendungsbibliothek CAOSw.exe besitzen zusätzlich sog. Konfigurationsdateien (Dateien mit dem Dateinamen AppName.exe.config), in denen Informationen für die verschiedenen Startobjekte, welche die CAOSStartUp-Bibliothek als Aufrufparameter benötigt, enthalten sind.

5.3.4. Die Bibliothek Utilities

Eine weitere Bibliothek, welche auch unabhängig vom Softwaresystem CAOS verwendet werden kann, ist die Bibliothek Utilities (dt. Hilfswerkzeuge). Sie beinhaltet zum einen Hilfsklassen zur Behebung von Problemen mit der Mono^{TM}-Laufzeitumgebung[77], welche sich u. a. bei verschiedenen Dateizugiffen ergeben, und zum anderen global benötigte Hilfsklassen. Diese sind bspw. Hilfsklassen für

- den kulturunabhängigen Datenzugriff (wichtig für plattformunabhängigen Datenaustausch),
- den externen Zugriff (Serialisierung/Deserialisierung) auf mehrdimensionale Felder,
- die einheitliche Ausgabe von Fehlermeldungen,
- den Zugriff und die Anzeige von sog. Statusdateien[78],

[75] Im engeren Sinne existieren bei C#-Programmen keine Anwendungen, daher ist an dieser Stelle von Anwendungsbibliotheken die Rede. Bei den im CAOS vorhandenen Bibliotheken liegt stets nur übersetzter Zwischenkode vor. In den Anwendungsbibiliotheken ist lediglich ein globaler Einstiegspunkt definiert und evtl. eine mögliche Startroutine enthalten.

[76] Hierfür muss die Laufzeitumgebung die Klassen des Namensraumes System.Windows.Forms (SWF) bereitstellen, was von einigen Laufzeitumgebungen bisher nicht komplett erfolgt ist oder zu einem anderen visuellen Ergebnis als erwartet führt. Daher ist die Anwendungsbibliothek CAOSw.exe derzeit nur mit dem .NET-Framework uneingeschränkt lauffähig.

[77] Die Laufzeitumgebung von Mono^{TM} wurde zum Test der Plattformunabhängigkeit unter den Betriebssystemen Linux und Mac OS X genutzt. Unter dem Betriebssystem Linux kam zudem der Test der verteilten und parallelen Optimierung auf dem CLiC hinzu.

[78] In Statusdateien, auch Log-Dateien genannt, werden bestimmte Informationen, welche während des Ablaufes der Anwendung auftreten protokoliert, welche im Anschluss an die Anwendungsausführung oder währenddessen ausgewertet werden können.

- die Arbeit mit Dateien im IFACE-Format (angestrebtes Datei-Format neben XML, siehe [Had98]),
- den Zugriff auf Schlangen mit vorgegebener Ordnung (FIFO, LIFO, Random, u. Ä.) sowie
- die Arbeit mit sog. Ringpuffern (additiv, nicht additiv)[79].

Weitere wichtige Hilfsklassen befinden sich im Namensraum `Utilities.Windows`, welcher verschiedene Hilfsklassen für fensterbasierte Anwendungen unter dem Betriebssystem Windows®, wie bspw. kombinierte Baum-Listen-Ansichten, Gantt-Charts, Editoren für Knoten und deren Verbindungen, Editoren für Datengitter-Werte sowie Methoden zum Prozess-Management beim multi-threading [engl.] bereitstellt. Darüber hinaus sind im Namensraum `Utilities.Random` Hilfsklassen für die implementierten Verteilungen und Generatoren für Zufallszahlen sowie deren Anzeige implementiert. Diese sollen nachfolgend kurz etwas genauer betrachtet werden.

(Pseudo)Zufallszahlengeneratoren. Die im CAOS vorhandenen (Pseudo)Zufallszahlengeneratoren bzw. kurz nur Zufallszahlengeneratoren (siehe Abb. 5.2) basieren auf den Aussagen des Abschnittes 3.7. In der Klasse RandomNumberGeneratorLEcuyer ist u. a. die in [LA97] bzw. [LSCK02] vorgeschlagene *Kombination aus vier LKGs* umgesetzt. Zudem wurde der *Mersenne Twister* (siehe [MN98]) in der Klasse RandomNumberGeneratorTwister implementiert, weil dieser einer der derzeit schnellsten Zufallszahlengeneratoren ist (siehe [MN98] und [LK00]). Ein weiterer Zufallszahlengenerator ist in der Klasse RandomNumberGeneratorCsharp zu finden. Er kapselt den in der Programmiersprache C# bereits vorhandenen Zufallszahlengenerator der Klasse System.Random (vgl. [Mic07a]), welcher auf einer Implementierung des *Algorithmus eines subtraktiven Zufallszahlengenerators* von Knuth basiert (siehe [Knu81]). Die Kapselung dient der Vereinheitlichung des Zugriffes auf die Zufallszahlengeneratoren. Dies ermöglicht den wahlweisen, beliebigen Austausch eines Zufallszahlengenerators innerhalb einer Anwendung.

Wahrscheinlichkeitsverteilungen. Eine Wahrscheinlichkeitsverteilung kann durch *geeignete Transformation* der, durch einen Zufallszahlengenerators, erzeugten Pseudozufallszahlen u_i nachgebildet werden. Dies geschieht auch in der entworfenen Klassenbibliothek, in der verschiedene Wahrscheinlichkeitsverteilungen mit den entsprechenden Transformationen implementiert sind (siehe Abb. 5.3). Im wesentlichen beruhen diese Klassen, auf den im Verlaufe der Untersuchungen benötigten Wahrscheinlichkeitsverteilungen.

[79] Ringpuffer stellen eine festvorgegebene Menge an Plätzen zur Datenspeicherung zur Verfügung, auf die der Reihe nach zugegriffen wird und nach Erreichen des letzten Platzes wieder beim ersten Platz begonnen wird. Bei der nicht additiven Variante wird der Datenwert des aktuellen Platzes durch den neuen Datenwert überschrieben. Hingegen wird bei der additiven Variante der Datenwert des aktuellen Platzes beibehalten und der neue zusätzlich auf den gleichen Platz abgelegt. Ein Platz zur Datenspeicherung besteht in diesem Falle aus einer Schlange variabler Länge.

Abb. 5.2

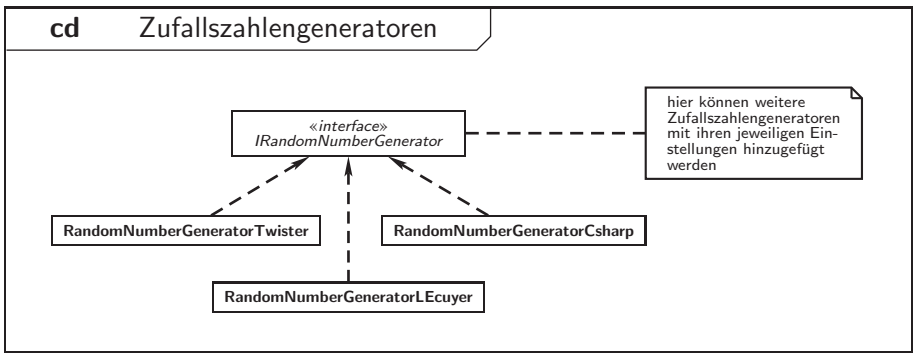


Abb. 5.2.: *im CAOS vorhandene Zufallszahlengeneratoren (UML-Klassendiagramm)*

Abb. 5.3

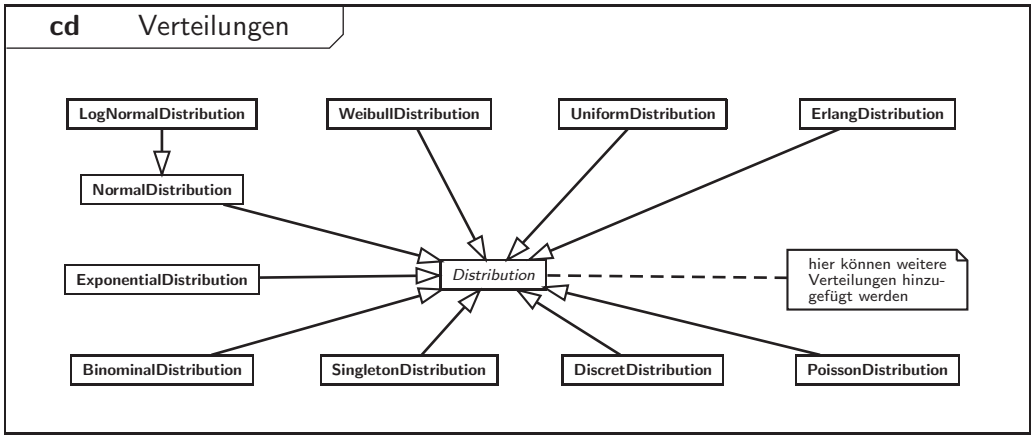


Abb. 5.3.: *im CAOS vorhandene Verteilungen (UML-Klassendiagramm)*

Für eine genaue mathematische Beschreibung der verwendeten Verteilungen für die Zufallsvariablen und die Transformation von Zufallszahlen sei auf [Stö03] und [Göh05] verwiesen.

Tab. 5.4 stellt die im `CAOS` vorhandenen Wahrscheinlichkeitsverteilungen, deren Schreibweise innerhalb einer Modelldefinition, die jeweils dafür implementierte Klasse und die Bedeutung der entsprechenden Verteilungsparameter dar. Dabei können die einzelnen Verteilungen neben der Form `XYZ[`$\text{param}_1, \ldots$`]` auch in der Form `XYZ(`$\exp(X), \text{var}(X)$`)` angegeben werden, wobei

- `XYZ` - das Verteilungskürzel,
- $\exp(X)$ - den Erwartungswert bzw. Mittelwert der Realisierungen der Zufallsvariablen X sowie
- $\text{var}(X)$ - die Streuung der Realisierungen der Zufallsvariablen X um den Mittelwert

angibt.

5.3.5. Die Bibliothek `Network`

Die Bibliothek `Network` enthält im Wesentlichen Hilfsklassen für

- *Netzwerkzugriffe* unabhängig vom verwendeten Netzwerk-Protokoll,
- verschiedene *Nachrichtentypen* sowie
- verschiedene *Kommunikationshierarchien*.

Für die Nachrichtentypen sind momentan nur Nachrichten vorgesehen, welche sich aus einem *Befehl*, der sich automatisch aus der verwendeten Nachrichtentyp-Klasse des Nachrichten-Objektes ergibt, und den zugehörigen *Daten*, welche den Inhalt der Klassenvariablen des Nachrichten-Objektes repräsentieren, zusammensetzt. Die zu sendenden Nachrichten werden dann als Daten im XML-Format aufbereitet, um unabhängig vom Netzwerkprotokoll Datenübertragungsfehler zu erkennen.

Als Kommunikationshierarchien sind prinzipiell Strukturen vom Typ Master-Slave oder Client-Server, bei denen jeweils ein dominanter Kommunikationspartner existiert, sowie Strukturen mit gleichberechtigten Kommunikationspartnern vorgesehen, wie sie bspw. bei bestimmten Synchronisationsvarianten der verteilten und parallelen Simulation (siehe [Fuj00]) oder bei einigen Nachbarschaftsmodellen der verteilten und parallelen Optimierung (siehe [Spr99]) ihre Anwendung finden.

Tab. 5.4

Verteilung	Klasse im CAOS	Darstellung im CAOS	Bedeutung der Verteilungsparameter
Diskrete Verteilungen:			
Binomialverteilung	`BinomialDistribution`	`B[`p, n`]`	p - Erfolgswahrscheinlichkeit n - Anzahl der Wiederholungen
Einpunktverteilung	`SingletonDistribution`	`S[`v`]`	v - konkreter Wert
Mehrpunktverteilung	`DiscreteDistribution`	`D[`$v_1 : p_1; v_2 : p_2; \ldots$`]`	v_i - konkreter i-ter Wert p_i - Wahrscheinlichkeit für v_i
Poissonverteilung	`PoissonDistribution`	`P[`λ`]`	λ - Ereignisrate
Stetige Verteilungen:			
(Stetige) Gleichverteilung	`UniformDistribution`	`U[`a, b`]`	a - linke Grenze b - rechte Grenze
Normalverteilung	`NormalDistribution`	`N[`μ, σ^2`]`	μ - Erwartungswert σ^2 - Varianz/Streuung
Logarithmische Normalverteilung	`LogNormalDistribution`	`LN[`μ, σ^2`]`	μ - Erwartungswert σ^2 - Varianz/Streuung
Exponentialverteilung	`ExponentialDistribution`	`E[`λ`]`	λ - Ausfallrate
Erlangverteilung	`ErlangDistribution`	`Ek[`k, λ`]`	k - Ordnung der Verteilung λ - Ausfallrate
Weibullverteilung	`WeibullDistribution`	`W[`α, β`]`	α - charakteristische Lebensdauer β - Ausfallsteilheit

Tab. 5.4.: *Implementierte Wahrscheinlichkeitsverteilungen im CAOS*

5.3.6. Die Bibliothek `CalculationAssessmentOptimisation`

Die Bibliothek `CalculationAssessmentOptimisation` wird nachfolgend kurz mit `CAO` bezeichnet. Die Trennung der Bibliothek `CAO` in die verschiedenen Namensräume dient neben der logischen Trennung auch der Aufteilung in die verschiedenen Aufgabenbereiche:

- *rechnerinterne Modellierung* (Namensraum `CAO.Model`),
- *Berechnung* und *Simulation* (Namensraum `CAO.Calculation`),
- *Optimierung* (Namensraum `CAO.Optimisation`) sowie
- *Bewertung* (Namensraum `CAO.Assessment`).

Der Inhalt dieser einzelnen Namensräume wird nachfolgend etwas genauer beschrieben.

Der Namensraum `CAO.Model`. Im Abschnitt 2.4 wurden bereits die Möglichkeiten und Vorgehensweise des gewählten Ansatzes zur rechnerinternen Modellierung beschrieben. An dieser Stelle sollen nun die bestehenden Klassenbeziehungen und -hierarchien etwas näher verdeutlicht werden.

In Abb. 5.4 ist zu erkennen, wie sich der charakteristische Aufbau eines rechnerinternen Modells anhand dessen Modelldeklaration und einer Modelldefinition innerhalb der Beziehungen der Klassen des `CAOS` zueinander widerspiegelt. Die abgebildeten Klassenbeziehungen sind insofern abstrahiert, dass in Abb. 5.4 nur die Basisklassen in Beziehung gestellt wurden, um das Augenmerk lediglich auf den vorhandenen Bezug zwischen Modelldefinition und -deklaration zu legen[80].

Abb. 5.4

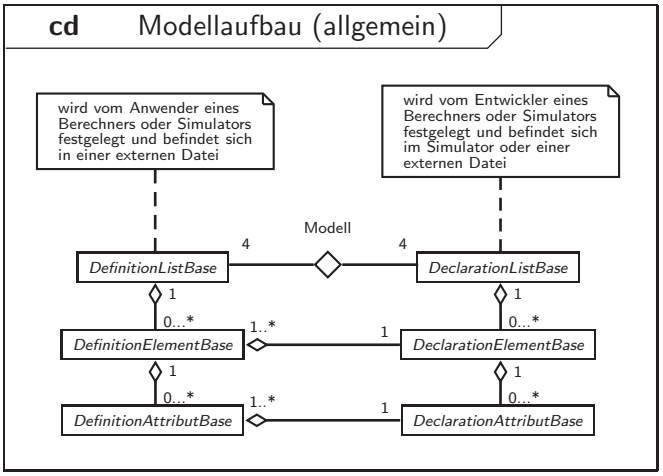


Abb. 5.4.: *Prinzipieller Aufbau eines Modells im CAOS (UML-Klassendiagramm)*

Am Beispiel der Klassen DefinitionElementBase und DeclarationElementBase soll etwas detaillierter verdeutlicht werden, wie sich die Verbindung zwischen den Teilen

[80] Durch die Darstellung aller Unterkategorien von Modelldefinition und -definition sowie deren Beziehungen zueinander bestünde die Gefahr der Unübersichtlichkeit.

der Modelldeklaration und -definition gestaltet. Hierzu sind in Abb. 5.5 zwei konkrete Beziehungen zwischen der Modellelementdefinition und dessen zugehöriger Deklaration unter Angabe weiterer wesentlicher Klassenvariablen für die Entscheidungsvariablen dargestellt.

Abb. 5.5

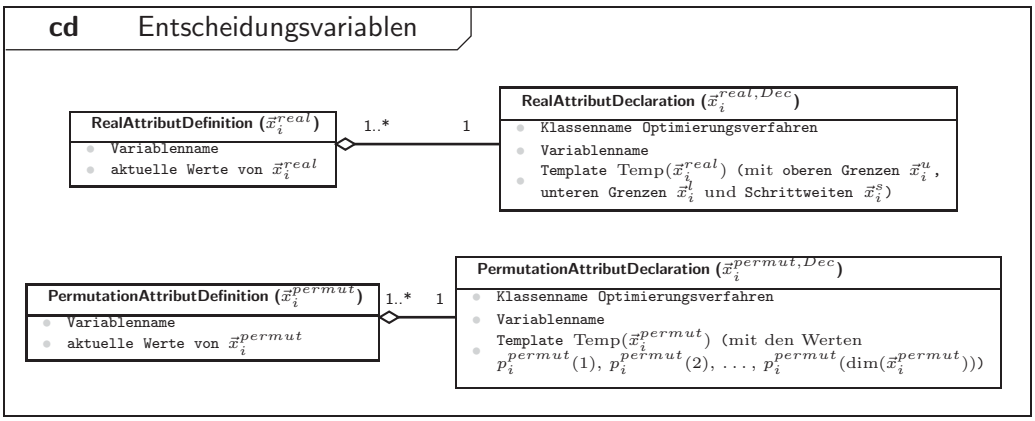


Abb. 5.5.: *Klasseninhalt der Entscheidungsvariablen (Auszug, UML-Klassendiagramm)*

Neben der, im Abschn. 2.4 angesprochenen, Trennung der Aufgaben von Modell- und Softwareersteller zielt die Trennung von Modelldeklaration und -definition auch auf die Vermeidung bestimmter Fehlertypen bei der Eingabe eines Simulationsmodells ab (s. u.), wodurch die Modellerstellung erleichtert wird.

Zu beachten ist, dass die rechnerinterne Modelldeklaration stets an einen Simulator gebunden ist, wodurch der Modellierer den, für sein Simulationsmodell und Optimierungsproblem passenden, zu verwendenden Simulator auswählen muss[81]. Damit erzwingt er die rechnerinterne Erstellung der zugehörigen Modelldeklaration sowie der bereits erwähnten minimalen Modelldefinition unter Nutzung vorhandener Klassen für die Deklaration und Definition der verschiedenen Elemente und deren Attribute. Diese Vorgehensweise ist in Abb. 5.6 zu sehen. Bei der Erstellung der Modelldefinition erweist es sich als sehr hilfreich, die im CAOS vorhandenen graphischen Benutzerschnittstellen zu verwenden, um syntaktische und semantische Fehler zu vermeiden (siehe Anhang D.1).

Nach der Erstellung einer Modelldefinition kann diese genutzt werden. Dazu ist in Abb. 5.7 erkennbar, wie sich diese Nutzung einer Modelldefinition im Rahmen des entwickelten Softwaresystems CAOS gestaltet. Von der dabei genutzten graphischen Oberfläche[82] hängt es ab, in welcher Menge und von welcher Art die ausgegebenen Informationen sind.

Listen, Elemente und Attribute. Das Softwaresystem CAOS basiert, durch die Verwendung der Programmiersprache C#, auf dem objektorientierten Program-

[81] Sollte ein geeigneter Simulator oder Berechner im CAOS noch nicht vorhanden sein, müsste ein solcher im Vorfeld der Nutzung erstellt werden. Dadurch wird die Entwicklung der Berechner und Simulatoren im CAOS maßgeblich vorangetrieben.

[82] Im CAOS sind zwei Varianten von graphischen Ausgaben vorgesehen, Console oder WinForms, je nachdem, ob nur eine alphanumerische, textbasierte Ausgabemöglichkeit (Console) oder auch eine fensterbasierte Ausgabemöglichkeit (WinForms) vorhanden ist (siehe Abschn. 5.3.3).

Abb. 5.6

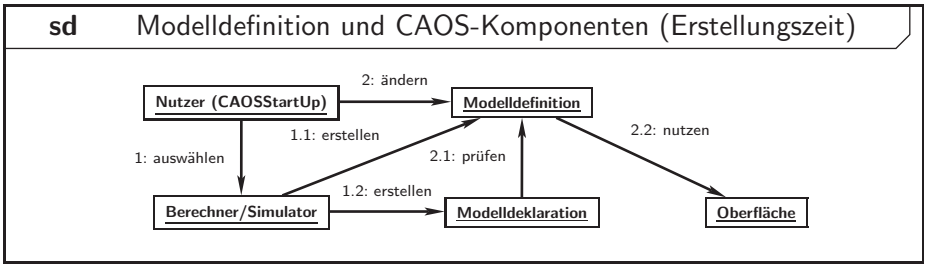


Abb. 5.6.: *Zusammenspiel zwischen einem Modell und den CAOS-Komponenten zur Entwurfszeit (UML-Kommunikationsdiagramm)*

Abb. 5.7

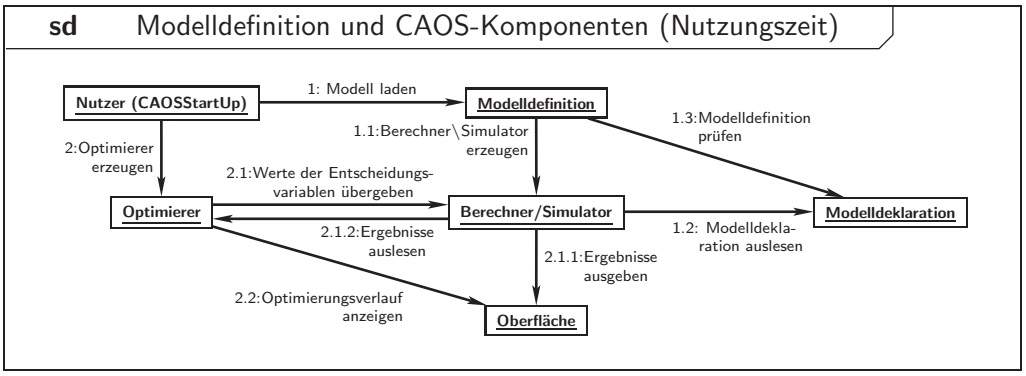


Abb. 5.7.: *Zusammenspiel zwischen einem Modell und den CAOS-Komponenten zur Laufzeit (UML-Kommunikationsdiagramm)*

mierparadigma[83], so dass es sich aus vielen, zueinander in Beziehung stehenden, Klassen zusammensetzt. Die Basisklassen enthalten dabei i. Allg. die grundlegenden Methoden, welche allen abgeleiteten Klassen zur Verfügung stehen sollen (*Prinzip der Vererbung*). Dies gilt auch für die Klassen bei Listen, Elementen und Attributen. In Abb. 5.8 ist bspw. die abstrakte Klasse[84] ListBase abgebildet, welche die grundlegenden Methoden und Eigenschaften für die Listen mit den verschiedenen Elementen einer rechnerinternen Modelldefinition oder -deklaration enthalten (siehe z. B. auch die Klassen DefinitionListBase und DeclarationListBase in Abb. 5.4). Die von der Klasse ListBase abgeleiteten Klassen InputCalculationList, InputOptimisationList, OutputAssessmentList und OutputNoAssessmentList enthalten ihrerseits nur die von der speziellen „Listen"-Klasse zusätzlich benötigten Methoden und die Prüfung, ob der Typ[85] der eingefügten Elemente richtig für den jeweiligen Listentyp ist. Diese Vorgehensweise wird auch in den Vererbungshierarchien der betreffenden Klassen für die Elemente und die Attribute sichtbar. Diese sind in Abb. 5.9 und Abb. 5.10 dargestellt.

[83] Das objektorientierte Programmierparadigma ist auch unter dem Begriff der *Objekt-Orientierten Programmierung (OOP)* bekannt.

[84] Abstrakte Klassen sind Klassen, welche bereits Klassenvariable [dt.; member variables [engl.]] und Klassenmethoden [dt.; class methods [engl.]] enthalten dürfen, aber nicht instanziiert werden können.

[85] Die Programmiersprache C# ist eine streng typ-orientierte Sprache, so dass jedes Objekt einen bestimmten Objekttyp besitzt, welcher wiederum genau einer bestimmten Klasse zugeordnet werden kann.

Abb. 5.8

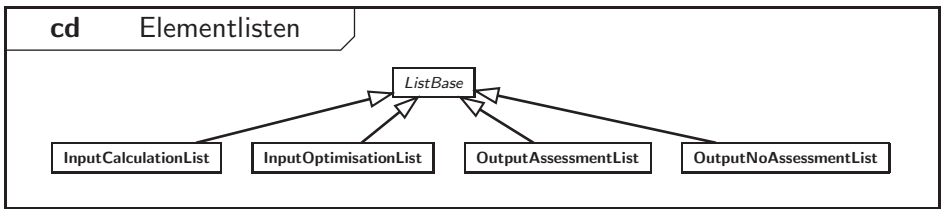


Abb. 5.8.: *Vorhandene Listen zur Aufbewahrung von Elementen im CAOS (UML-Klassendiagramm)*

Abb. 5.9

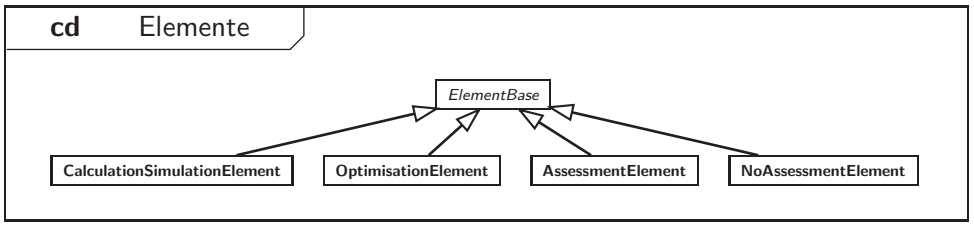


Abb. 5.9.: *Vorhandene Elemente zur Aufbewahrung von Attributen im CAOS (UML-Klassendiagramm)*

Ein Element ist stets so aufgebaut, dass es, durch die Anwendung bestimmter Prüfmechanismen, nur Attribute eines bestimmten Attributtyps aufnehmen kann[86].

Listen, Elemente und Attribute treten auf Grund ihrer unterschiedlichen Bedeutung jeweils in der Modelldeklaration und -definition mit unterschiedlichem internen Aufbau auf, besitzen aber, wegen ihres logischen Zusammenhanges, eine vergleichbare Klassenhierarchie. Diese Trennung von Modelldeklaration und -definition sowie die Beziehung zwischen Listen, Elementen und Attributen war bereits in Abb. 5.4 in allgemeiner Form dargelegt worden. Die Abb. 5.11 greift beispielhaft die Modelldefinition noch einmal heraus und stellt nur die Beziehungen zwischen Elementen und Attributen der unterschiedlichen Typen noch einmal etwas detaillierter dar. Auf die Darstellung der vier verschiedenen Listen für die Unterkategorien (vgl. Abschn. 2.4)

- $IN_{CalcSim}$ (Klasse InputCalculationDefinitionList),
- IN_{Opt} (Klasse InputOptimisationDefinitionList),
- OUT_{Ass} (Klasse OutputAssessmentDefinitionList) und
- OUT_{NoAss} (Klasse OutputNoAssessmentDefinitionList)

ist dabei verzichtet und nur eine Unterscheidung zwischen der Kategorie für die Eingabewerte (Klasse InputSection) und der Kategorie für die Ausgabewerte (Klasse OutputSection) für eine rechnerinterne Modelldefinition vorgenommen worden.

Die Tab. 5.5 zeigt an einem konkreten Beispiel, wie sich die Repräsentation einer

[86] Zur Umsetzung dieser Vorgehensweise hätte sich die Nutzung von generischen Datentypen (siehe Abschn. C.5) angeboten, welche ab der Sprachversion 2.0 der Programmiersprache C# eingeführt wurden. Allerdings stand zum überwiegenden Zeitraum der Implementierung nur die Sprachversion 1.0 bzw. 1.1 zur Verfügung.

Abb. 5.10

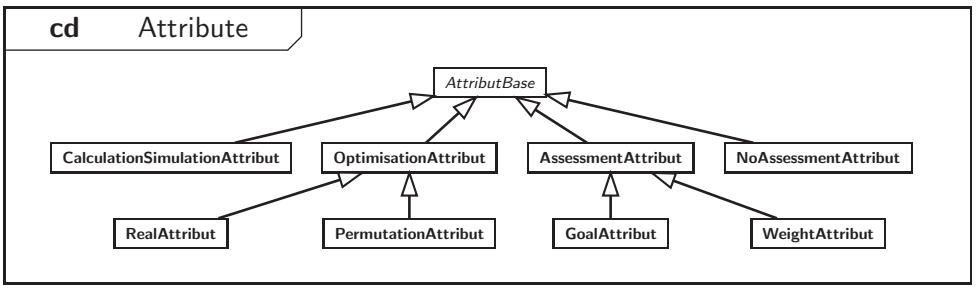


Abb. 5.10.: *Vorhandene Attribute im CAOS (UML-Klassendiagramm)*

Abb. 5.11

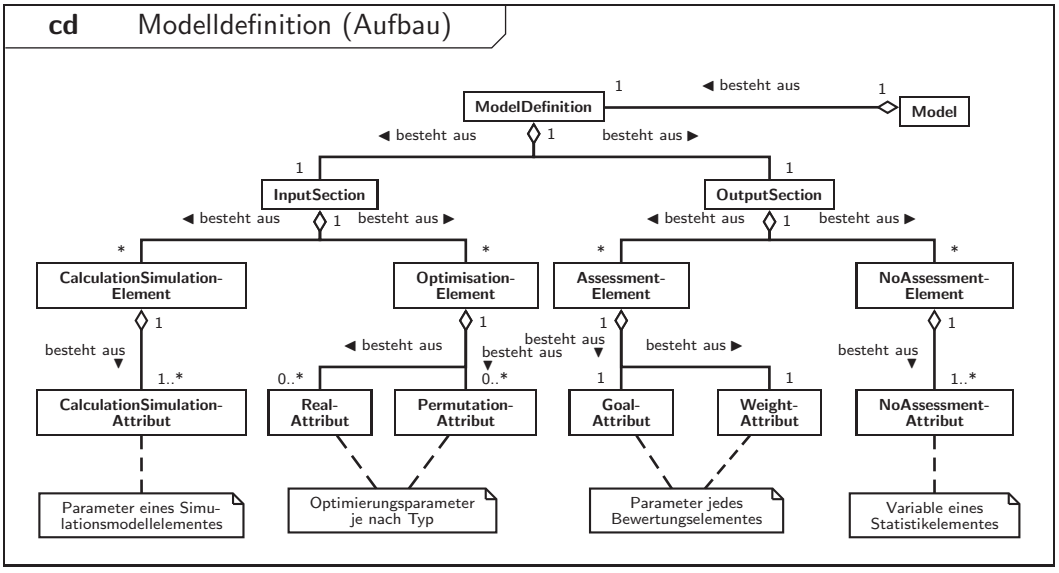


Abb. 5.11.: *Aufbau der Definition eines Modells im CAOS (UML-Klassendiagramm)*

Modelldeklaration in einem Rechner mittels der gewählten Baumstruktur gestaltet. Hierfür ist die rechnerinterne Repräsentation der Modelldeklaration des im Abschn. 6.4.1 vorgestellten Modells für ein Produktions- und Lagerhaltungssystem mit einer Fertigungseinheit dargestellt.

Lexikalische, syntaktische und logische Fehler. In einer Modelldeklaration sind Festlegungen, wie die minimale und maximale Anzahl von Elementen eines Typs, optionale Elemente oder zulässige Datentypen für die Werte von Attributen, getroffen, welche den Modellersteller bei der Erstellung einer Modelldefinition entsprechend den Fähigkeiten des Simulators weitestgehend restringieren. Der Modellersteller kann somit seinerseits unter Anwendung der ausgewählten Modelldeklaration und der vorhandenen graphischen Benutzerschnittstellen zwar die Definition für sein Modell eingeben, aber keine unvorhersehbaren *lexikalischen* oder *syntaktischen Fehler* begehen. Lexikalische Fehler würden, auf Grund der gewählten Implementierung, unmittelbar entsprechende Fehlermeldungen hervorrufen. Bei der Nutzung eines externen Bearbeitungsprogramms zur Erstellung einer neuen oder Veränderung einer bestehenden Modelldefinition erfolgt diese Prüfung auf lexikalische Fehler erst während des Ladens (*strukturelle Gültigkeitsprüfung*) bzw. nach

Tab. 5.5

Unterkategorie	Bezeichnung	Element/Attribut
$IN_{CalcSim}$	Global (Simulation)	Element
	• Dauer der Simulation (t_{st})	Attribut
	• Dauer der transienten Phase (t_{tr})	Attribut
	• Anzahl von Wiederholungen (n_{sim})	Attribut
	Global (Multi-Item-System)	Element
	• Lagergröße (P)	Attribut
	• Reihenfolgestrategie (RS)	Attribut
	• Strategie für die Zyklusbestimmung	Attribut
	Produkt i	Element
	• Kundenankunftsrate (λ_i)	Attribut
	• Kundenbedarfsmenge (d_i)	Attribut
	• Produktionsrate (μ_i)	Attribut
	• Lagerplatz (p_i)	Attribut
	• Umrüstzeiten ($\vec{st}_i$)	Attribut
	• Umrüstkosten ($\vec{sc}_i$)	Attribut
	• Produktionskosten (pc_i)	Attribut
	• Lagerkosten (hc_i)	Attribut
	• Abweiskosten (rc_i)	Attribut
	• Wartekosten (wc_i)	Attribut
IN_{Opt}	reellwertige Parameter	Element
	• Produktionszeitpunkte ($\vec{s}$)	Attribut
	• Produktionsobergrenzen/-mengen ($\vec{Q}/\vec{S}$)	Attribut
	(Nebenbedingungen)	
OUT_{Ass}	Umrüstkosten (SC)	Element
	• Optimierungsziel	Attribut
	• Gewicht	Attribut
	Produktionskosten (PC)	Element
	• Optimierungsziel	Attribut
	• Gewicht	Attribut
	Lagerkosten (HC)	Element
	• Optimierungsziel	Attribut
	• Gewicht	Attribut
	Abweiskosten (RC)	Element
	• Optimierungsziel	Attribut
	• Gewicht	Attribut
	Wartekosten (WC)	Element
	• Optimierungsziel	Attribut
	• Gewicht	Attribut
OUT_{NoAss}	Statistik	Element
	• Verlust (C)	Attribut
	• Kunden insgesamt (n_c)	Attribut
	• abgewiesene Kunden insgesamt (n_{rc})	Attribut

Tab. 5.5.: *Beispiel der rechnerinternen Abbildung einer Modelldeklaration (Auszug)*

dem Laden einer Modelldefinition (*Gültigkeitsprüfung der Modelldefinition und lexikalische Gültigkeit der Datenwerte*).

Erwähnt sei an dieser Stelle, dass für die Gültigkeitsprüfung der Struktur einer Modelldatei, welche als XML-Datei (Abkürzung für eXtensible Markup Language [engl.; erweiterbare Auszeichnungssprache [dt.]]) vorliegt, bspw. eine globale XSD-Datei (XML-Schema-Definition) verantwortlich ist. Währenddessen findet die Gültigkeitsprüfung der Modelldefinition nach dem Laden anhand der Modelldeklaration statt.

Abschließend sei angemerkt, dass Hilfsmittel für die Erkennung *logischer Fehler*[87] innerhalb einer bestehenden Modelldeklaration im Zuge der Durchführung einer weiterreichenden Plausibilitätsprüfung dem Softwareersteller jedoch nicht pauschal bereitgestellt werden können. Sie sind zumeist stark problemspezifisch und von der konkreten Implementierung des Simulators abhängig.

Der Namensraum `CAO.Calculation`. Im Namensraum `CAO.Calculation` sind im Wesentlichen zwei weitere Namensräume enthalten. Dies ist zum Einem der Namensraum `CAO.Calculation.Calculators`, welcher die verschiedenen vorhandenen *Berechner* für die Durchführung analytischer Berechnungsvorschriften enthält, und zum anderen der Namensraum `CAO.Calculation.Simulation.Simulators`, in dem sich die implementierten *Simulatoren* zur ereignisgesteuerten Simulation befinden. Auszüge der in den Namensräumen enthaltenen Klassenhierarchien sind in Abb. 5.12 und Abb. 6.2 dargestellt. Sie verdeutlichen u. a. die Abhängigkeit der Simulator-Klassen von der mit den Berechnern einheitlichen Basisklasse CalculatorBase. Beide Namensräume lassen sich um beliebige Klassen für Berechner oder Simulatoren erweitern, wobei verschiedene abstrakte und virtuelle Methoden noch zu implementieren bzw. zu überschreiben sind. Die Basisklasse CalculatorBase zur Berechnung bildet an dieser Stelle die Oberklasse zur Simulation, weil die Simulation als spezielle Art der Berechnung mit stochastischen Einflüssen aufgefasst werden kann.

Abschließend sei noch angemerkt, dass die verschiedenen Berechner ein Beleg dafür sind, dass das `CAOS` über diese Arbeit hinaus genutzt wird und nutzbar ist. Beispiele, bei denen mit dem `CAOS` bereits sehr gute Erfolge in der Theorie und der Praxis erzielt wurden, sind in [Wac05] und [Sch06b] zu finden.

Der Namensraum `CAO.Optimisation`. Im Namensraum `CAO.Optimisation` sind die verschiedene Optimierungsverfahren enthalten, welche durch den *Optimierungsmanager* gesteuert werden. Daher sollen zunächst die Aufgaben und Funktionsweisen dieses Optimierungsmanagers betrachtet werden.

Der Optimierungsmanager und der Ablauf des Optimierungsprozesses. Zum besseren Verständnis der Vorgehensweise der im Folgenden vorgestellten Optimie-

[87] Logische Fehler sind in diesem Zusammenhang bspw. Fehler bei der rechnerinternen Modellierung eines Realsystems bzw. dessen mathematischen Modells.

Abb. 5.12

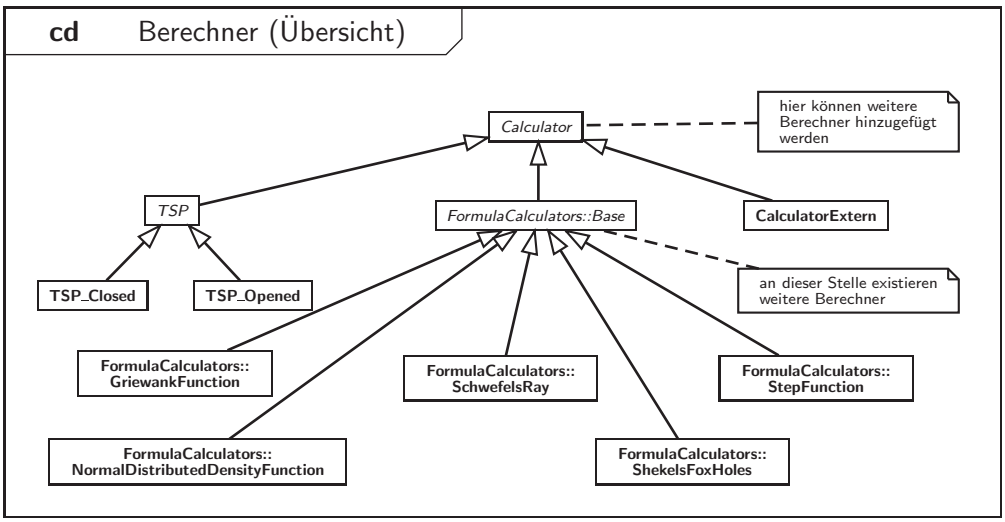


Abb. 5.12.: *Auszug der im CAOS vorhandenen Berechner (UML-Klassendiagramm)*

rungsverfahren sei an dieser Stelle bereits erwähnt, wie sich der allgemeine Ablauf des Optimierungsprozesses gestaltet. Das wesentliche Merkmal des Optimierungsprozesses ist, dass die Optimierungsverfahren ihre *Optimierung schrittweise* ausführen. Das bedeutet ein Optimierungsverfahren berechnet neue Werte für die Entscheidungsvariablen und übergibt diese einer übergeordneten Instanz, welche als Optimierungsmanager bezeichnet wird. Das Optimierungsmanager sorgt dann dafür das die Werte für die Entscheidungsvariablen den eingesetzten Optimierungsverfahren, welche im eigentlichen Sinne für den Optimierungsmanager nur Teilvektoren mit den Werten für die Entscheidungsvariablen darstellen, zu einer kompletten Lösung zusammengesetzt werden. Anschließend erfolgt die Prüfung der Werte für die Entscheidungsvariablen auf Gültigkeit anhand der vorgegebenen Nebenbedingungen[88] und bei deren Erfüllung die Übergabe dieser zusammengesetzten Lösung an einen Simulator oder Berechner. Diese Vorgehensweise weicht im Vergleich zu den meisten anderen Optimierungsverfahren demnach in sofern ab, dass nicht das Optimierungsverfahren die Simulatoren aufruft, sondern der Optimierungsmanager. Dieser steuert zudem die Optimierer. Der Optimierungsmanager ist des Weiteren

- für die Verteilung eines beliebigen Optimierungsverfahren im Rahmen der verteilten und parallelen Optimierung (s. u. sowie Abschn. 4.5.2),
- für die zuvor beschriebene Verteilung der Lösung zur Nutzung verschiedener Optimierungsverfahren sowie
- für die Anwendung verschiedener Typen von Entscheidungsvariablen (reellwertig und Permutation (vgl. Abschn. 2.2.6)) innerhalb eines Optimierungsproblemes

zuständig.

[88] Diese Gültigkeitsprüfung müsste sonst in jedem Optimierer individuell durchführen, wobei er mitunter nicht die Werte aller, für die Prüfung relevanter, Entscheidungsvariablen besitzt.

Nach Beendigung eines Berechners oder Simulators werden die Ausgabewerte ausgelesen sowie dem Bewerter zur Bewertung bereitgestellt. Nach dessen Bewertung stehen die Zielfunktionswerte den einzelnen Optimierern zur Einordnung ihres zuvor erzeugten Vektors mit den Werten für die Entscheidungsvariablen bereit. Dieser Zyklus wird solange wiederholt bis ein allgemeingültiges, optimiererspezifisches oder interaktives Abbruchkriterium erfüllt ist (siehe Abb. 5.13). Anzumerken ist, dass der Optimierungsmanager bevor er mit dem in Abb. 5.13 dargestellten Zyklus beginnt, die verwendeten Optimierungsverfahren entsprechend initialisieren muss. Zudem kann die in Abb. 5.13 angegebene Prüfung auf Abbruch des Optimierungsmanagers durch Erreichen eines allgemeingültigen Abbruchkriteriums auch an anderen Stellen erfolgen.

Abb. 5.13

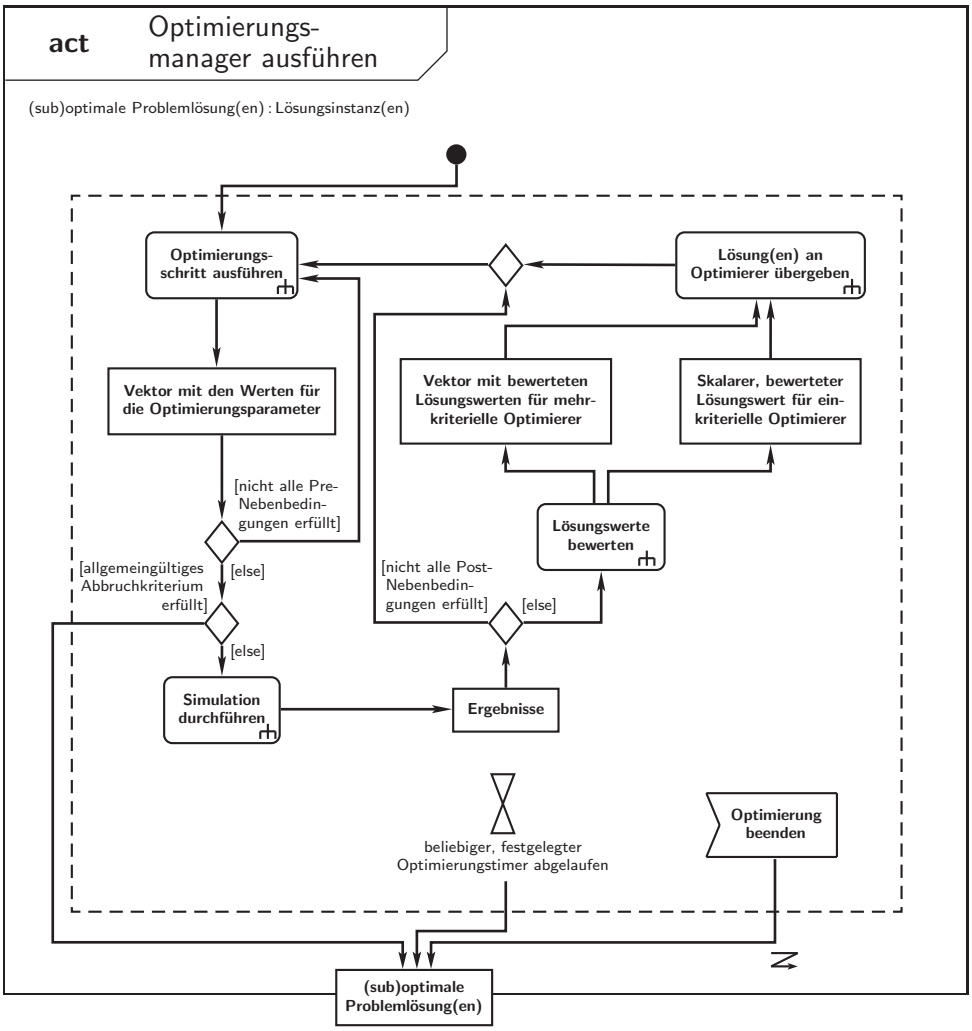


Abb. 5.13.: *Algorithmus zur Ausführung des Optimierungsmanagers im CAOS (UML-Aktivitätsdiagramm)*

In Abb. 5.13 ist auch erkennbar, dass die möglicherweise vorhandenen Nebenbedingungen es notwendig machen, die Werte der Optimierungsverfahren für die Entscheidungsvariablen vor (Pre-Nebenbedingungen) sowie nach (Post-Nebenbedingungen) jedem Optimierungsschritt auf ihre Gültigkeit zu prüfen. Sollten dabei durch ein oder mehrere Optimierungsverfahren ungültige Werte für die Ent-

Abb. 5.14

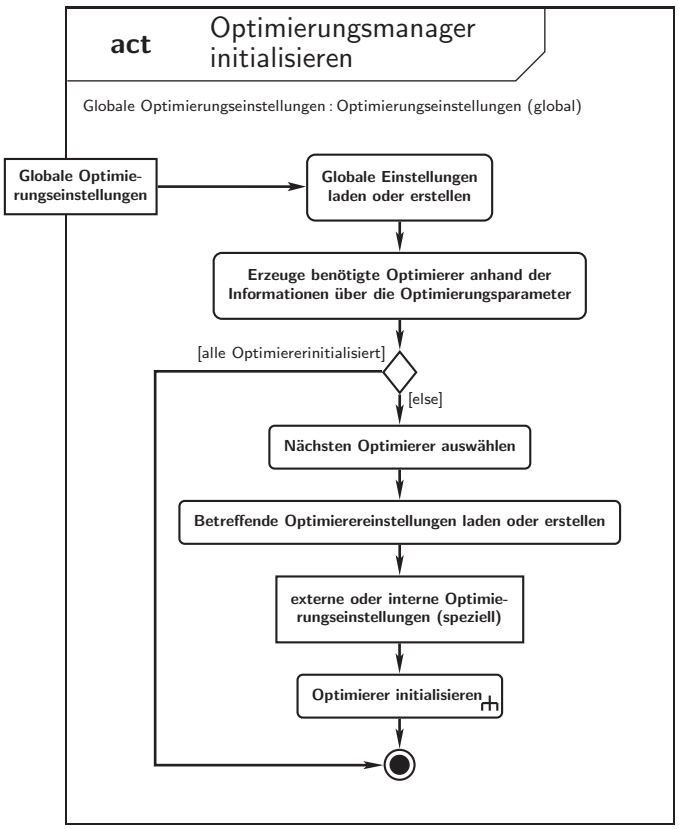


Abb. 5.14.: *Algorithmus zur Initialisierung des Optimierungsmanagers im CAOS (UML-Aktivitätsdiagramm)*

scheidungsvariablen entstanden sein, werden die Zielfunktionswerte auf „ungültig“ gesetzt. Auf diese Weise wird es auch den einzelnen Optimierungsverfahren mitgeteilt, woraufhin diese neue Werte für die Entscheidungsvariablen bestimmen müssen[89].

Günstig an der Nutzung des Optimierungsmanagers sind dabei neben den zuvor genannten Vorteilen wiederum die Vererbungsmöglichkeiten bei der objektorientierten Programmierung (siehe Abb. 5.15). Es wird somit möglich, dem verteilten Optimierungsmanager (Klasse OptimisationManagerDistributed) die gleiche Grundfunktionalität wie dem nichtverteilten Optimierungsmanager (Klasse OptimisationManager) zukommen zu lassen und eine logische Trennung der zusätzlichen Funktionen des verteilten Optimierungsmanagers vorzunehmen. Zusätzlich sind in Abb. 5.15 noch die Klassen OptimisationManagerMasterSlave und OptimisationManagerClientServer abgebildet, welche konkrete Nutzungen der Klasse OptimisationManagerDistributed beinhalten. Dies ist zum einen die verteilte und parallele Optimierung und zum anderen die entfernte Berechnung oder Simulation.

Nachfolgend seien die Algorithmen der verschiedenen implementierten ein- und mehrkriteriellen Optimierungsverfahren angeführt.

[89] Eigentlich müssten nur die Optimierungsverfahren neue Werte bestimmen oder die „ungültigen“ Werte durch entsprechende Korrekturverfahren verändern, die auch „ungültige“ Werte hervorgerufen haben. Der Einfachheit halber, sind aber zunächst nur die beschriebene Vorgehensweise umgesetzt.

Abb. 5.15

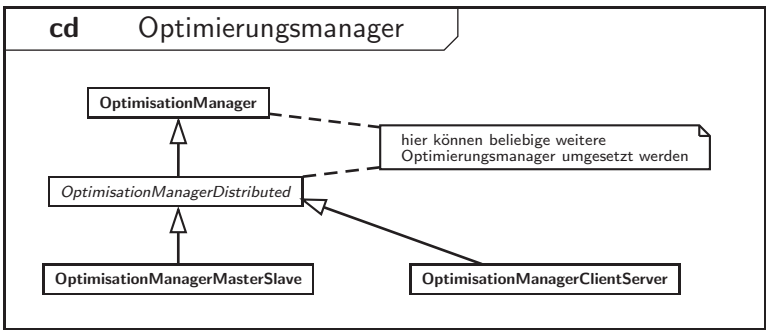


Abb. 5.15.: *Optimierungsmanager (UML-Klassendiagramm)*

Implementierung der Optimierungsverfahren. Bereits in Abb. 4.5 und Abb. 4.6 wurden die verschiedenen implementierten ein- oder mehrkriteriellen Optimierungsverfahren dargestellt, welche in den Abschn. 4.3.6, 4.3.7 und 4.4 näher beschrieben sind. Diese befinden sich in den betreffenden Namensräumen `CAO.Optimisation.Optimisers.Singleobjective` (einkriterielle Optimierungsverfahren) und `CAO.Optimisation.Optimisers.Multiobjective` (mehrkriterielle Optimierungsverfahren). Des Weiteren enthält der Namensraum `CAO.Optimisation.Constraints` die Vorlagen für den programmtechnischen Umgang der möglicher Weise auftretenden Nebenbedingungen bei Optimierungsproblemen, wobei in der derzeitigen Version des Softwaresystems `CAOS` noch keine stochastischen Nebenbedingungen abgebildet werden können (vgl. Abschn. 2.2.4).

Einige der in Abb. 4.4 dargestellten stochastischen Verbesserungsverfahren sind auch im `CAOS` implementiert. Daher sollen deren Implementierung in den nachfolgenden Abschnitten vorgestellt werden. Im Speziellen handelt es sich um die Simulierte Abkühlung, die Tabusuche und zwei Genetische Algorithmen (ein generationsbasierter GA und ein GA mit stetiger Ersetzung). Auf diesen Verfahren basierend sind auch zwei hybride Optimierungsverfahren (seriell und parallel, siehe Abschn. 4.5.1) implementiert. Zudem ist auch das Verfahren der Vollständigen Aufzählung (siehe Abschn. 4.2) softwaretechnisch umgesetzt, um eine Umgebungssuche in Nähe der gefundenen mitunter nur suboptimalen Lösung zu ermöglichen. Jedes der implementierten Optimierungsverfahren zeichnet sich dabei durch einen *Initialisierungsschritt* und einen *Optimierungsschritt* aus, welche in Abb. 5.16 und Abb. 5.17 allgemein gültig, für die verschiedenen Optimierungsverfahren, dargestellt sind. Die optimiererspezifischen Initialisierungs- und Optimierungsschritte, auf welche in den jeweiligen nachfolgenden Abschnitten eingegangen wird, sind durch die Bezeichnung „`(speziell)`“ oder das Detailsymbol „⫛“ gekennzeichnet. Der Initialisierungsschritt wird einmalig zur Überführung des Optimierungsverfahrens in einen definierten Ausgangszustand vom Optimierungsmanager aufgerufen (siehe Abb. 5.14, Aktivität „Optimierer initialisieren“). Der Optimierungsschritt hingegen wird vom Optimierungsmanager zur beschriebenen schrittweisen Abarbeitung wiederholt aufgerufen (s. o.).

Abb. 5.16

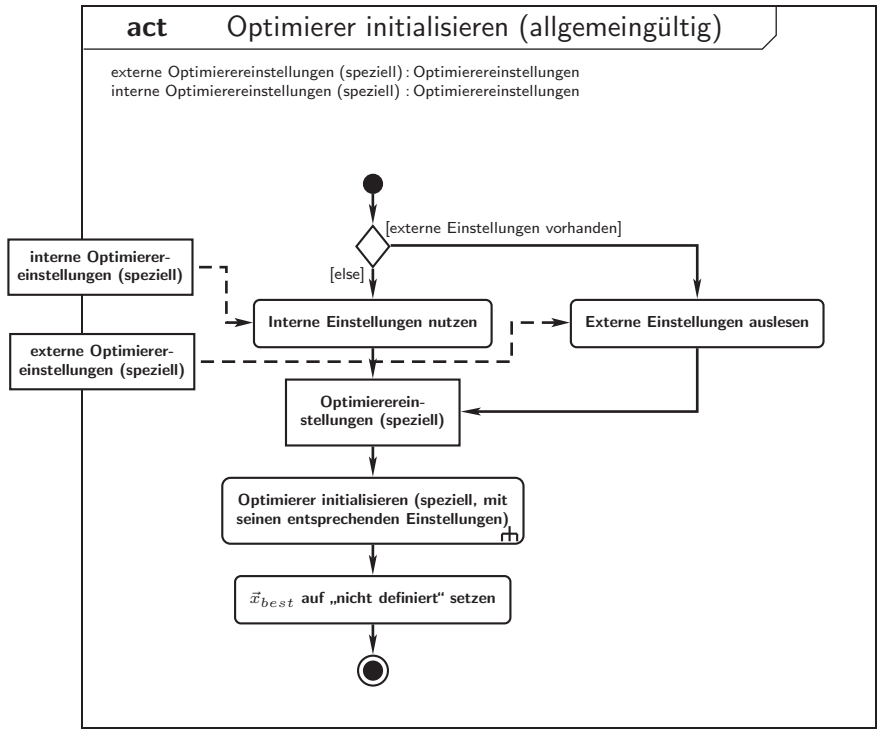


Abb. 5.16.: *Algorithmus zur Initialisierung eines Optimierers (UML-Aktivitätsdiagramm)*

Abb. 5.17

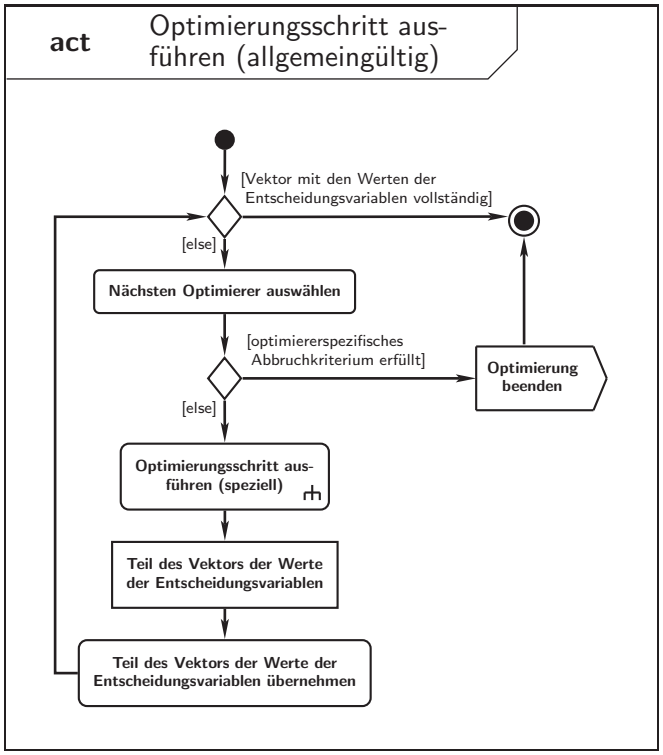


Abb. 5.17.: *Algorithmus zum Ausführen eines Optimierungsschrittes (UML-Aktivitätsdiagramm)*

In Abb. 5.17 ist auch zu sehen, wie das Problem des Erreichens eines optimierer-spezifischen Abbruchkriteriums gelöst ist. Das betreffende Optimierungsverfahren sendet dazu lediglich ein globales Signal („`Optimierung beenden`“), welches vom Optimierungsmanager zu geeigneter Zeit abgefangen wird und dazu führt, dass alle Optimierungsverfahren definiert beendet werden.

Optimierungsverfahren der Simulierten Abkühlung. Die innerhalb dieser Arbeit genutzte Variante der Simulierten Abkühlung wurde im Rahmen einer Studienarbeit von Wachsmuth im Softwaresystem `CAOS` implementiert (siehe [Wac04]). Sie sieht die, bereits zuvor angesprochene, Unterteilung in Initialisierungsschritt (siehe Abb. 5.19) und Optimierungsschritt (siehe Abb. 5.18) vor und setzt diese softwaretechnisch in der konkreten Implementierung um.

Abb. 5.18

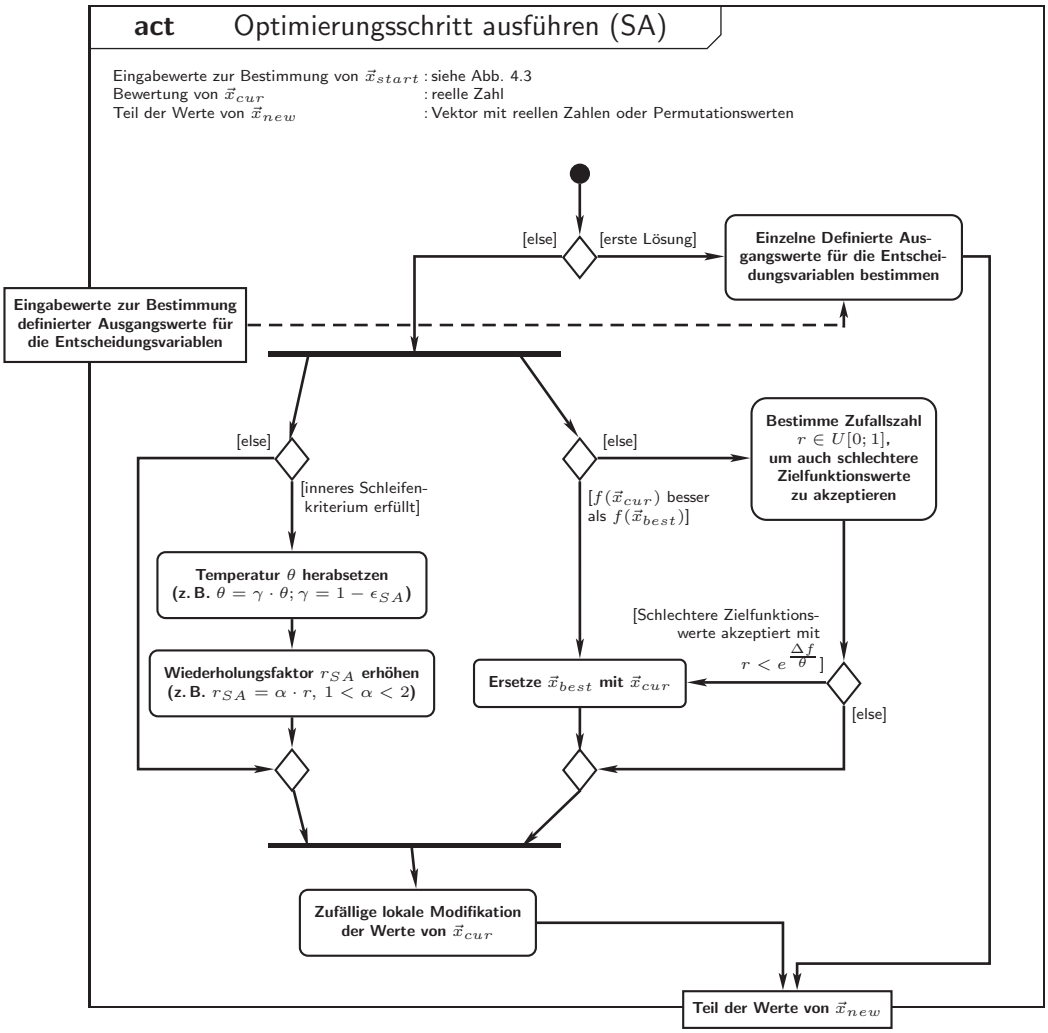


Abb. 5.18.: *Algorithmus zum Ausführen eines Optimierungsschrittes bei der Simulierten Abkühlung (UML-Aktivitätsdiagramm)*

Abb. 5.19

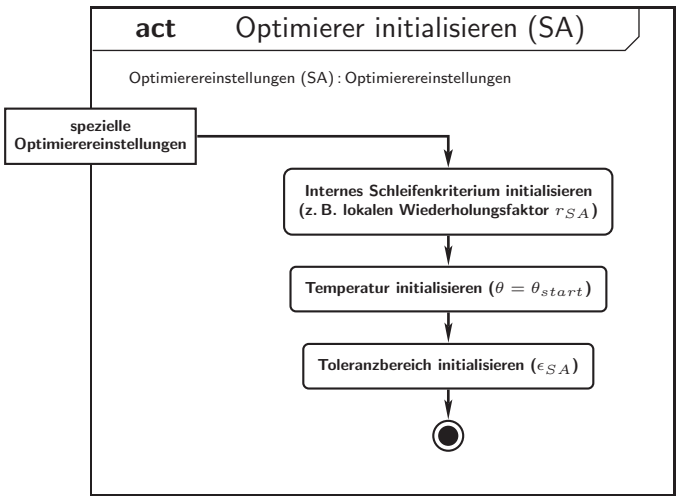


Abb. 5.19.: *Algorithmus zur Initialisierung des Optimierers zur Simulierten Abkühlung (UML-Aktivitätsdiagramm)*

Optimierungsverfahren der Tabusuche. Die implementierte Variante des Optimierungsverfahren zur Tabusuche geht ebenfalls auf die Studienarbeit von Wachsmuth (siehe [Wac04]) zurück. Auch an dieser Stelle ist wieder die geforderte Unterteilung in Initialisierungsschritt (siehe Abb. 5.20) und Optimierungsschritt vorhanden (siehe Abb. 5.21). In Abb. 5.21 sind zudem die Mengen

$\overline{M}_{cur}$ - Menge der noch zu untersuchenden Lösungen der Nachbarschaft $N(\vec{x}_{cur})$ sowie

$\underline{M}_{cur}$ - Menge der bereits untersuchten Lösungen der Nachbarschaft $N(\vec{x}_{cur})$

angegeben.

Abb. 5.20

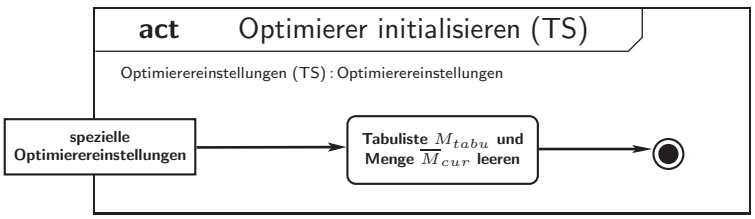


Abb. 5.20.: *Algorithmus zur Initialisierung des Optimierers zur Tabusuche (UML-Aktivitätsdiagramm)*

Abb. 5.21

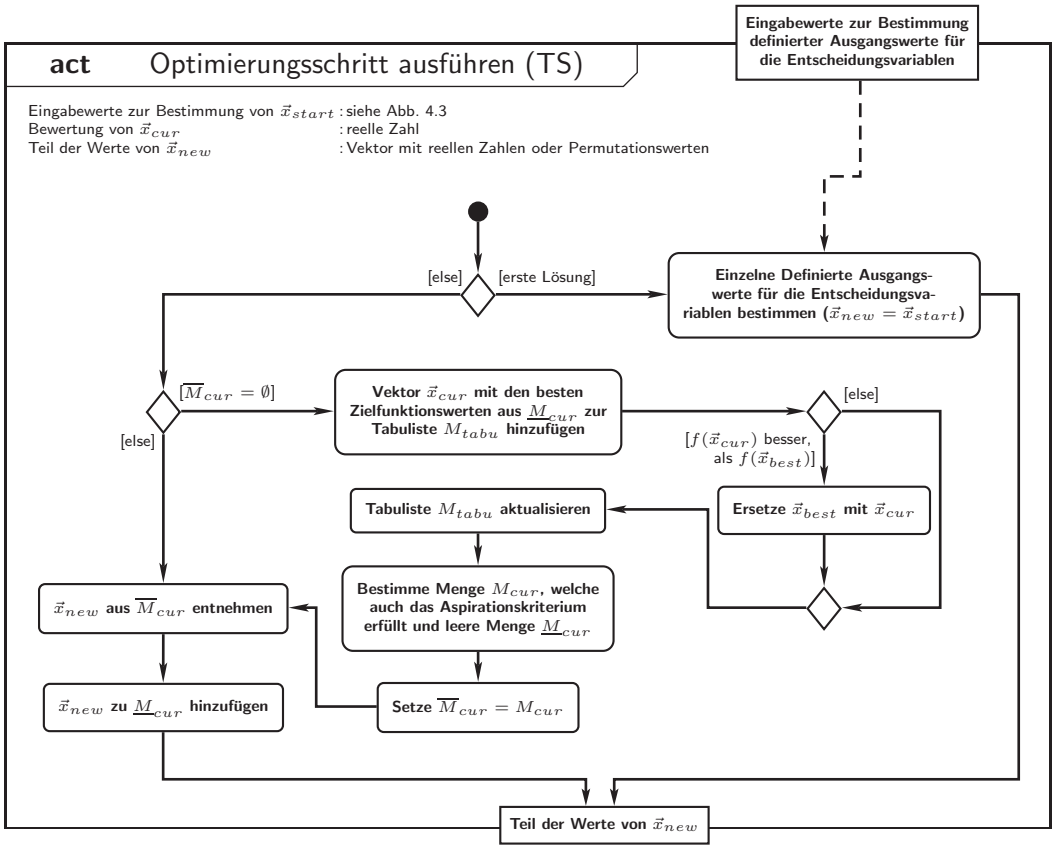


Abb. 5.21.: *Algorithmus zum Ausführen eines Optimierungsschrittes bei der Tabusuche (UML-Aktivitätsdiagramm)*

Optimierungsverfahren der Genetischen Algorithmen. Die Umsetzung von Genetischen Algorithmen im CAOS gestaltet sich wie folgt. Es sind zwei Genetische Algorithmen implementiert:

- ein GA basierend auf dem Generationsprinzip (GA_{gen}) mit der Möglichkeit den Elitismus anzuwenden sowie
- ein GA, welcher den Ansatz der stetigen Ersetzung (GA_{ss}) verfolgt.

Beide Verfahren besitzen keinen speziellen Initialisierungsschritt, weisen allerdings unterschiedliche Optimierungsschritte auf (siehe Abb. 5.22 und 5.23).

In beiden Typen von GA sind in dieser Arbeit die verschiedenen Selektions-, Rekombinations- und Mutations-Operatoren vorhanden, welche im Abschn. 4.3.7 genannt wurden. Dadurch entstehen verschiedene Operatorgruppen:

- die Gruppe der Selektions-Operatoren,
- die Gruppe der Rekombinations-Operatoren sowie
- die Gruppe der Mutations-Operatoren.

Muss nun ein Operator angewandt werden, erfolgt dessen Auswahl vor seiner Anwendung gleichwahrscheinlich aus der Gruppe der betreffenden Operatoren.

Abb. 5.22

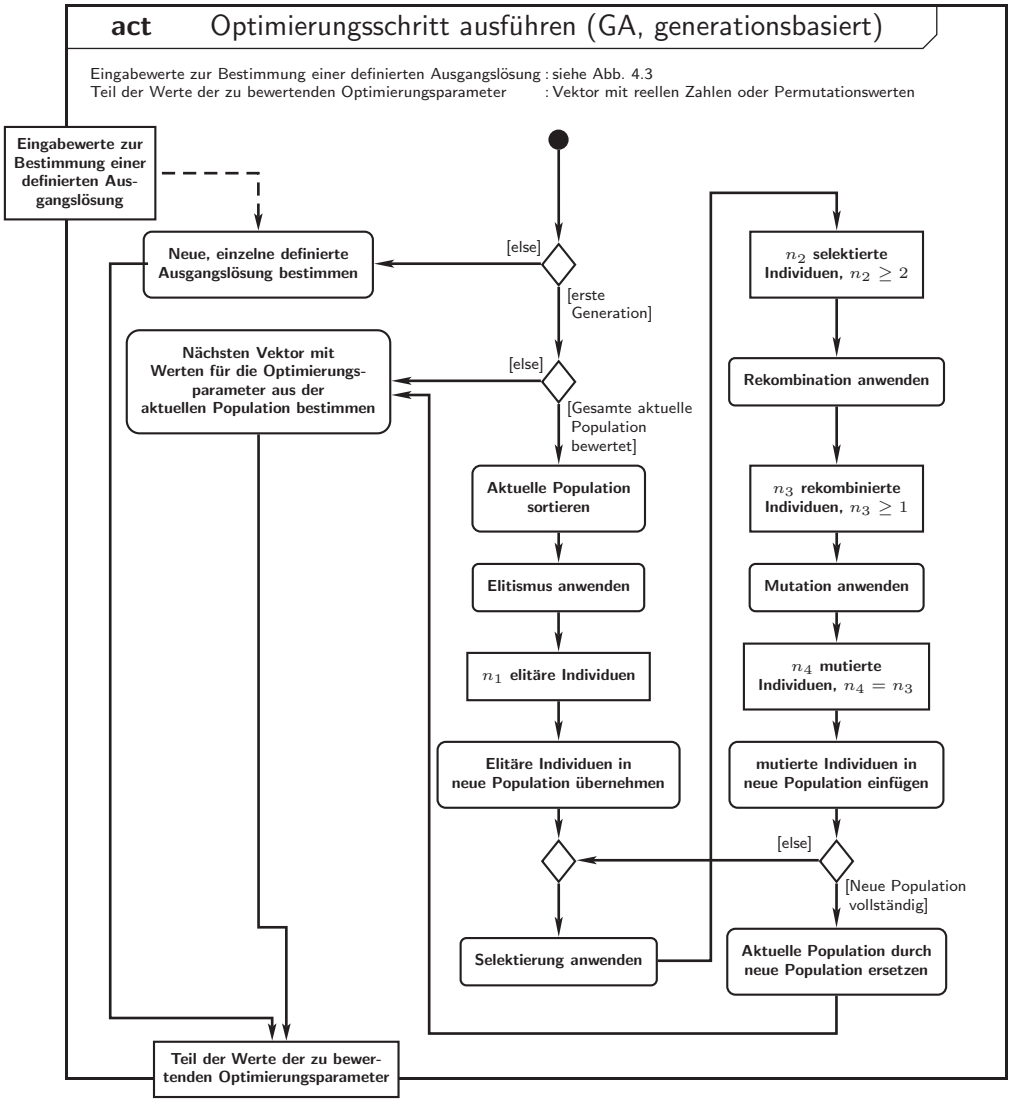


Abb. 5.22.: *Algorithmus zum Ausführen eines Optimierungsschrittes beim generationsbasierten Genetischen Algorithmus (UML-Aktivitätsdiagramm)*

Optimierungsverfahren zur hybriden Optimierung. Die vorangehend genannten Optimierungsverfahren SA, TS, GA_{gen} und GA_{ss} bilden die Menge der derzeit verwendbaren Optimierungsverfahren M_{HybOpt} für die hybriden Optimierungsverfahren. Hinzu kommt allerdings noch das Verfahren der Vollständige Aufzählung (VA), so dass sich für M_{HybOpt} ergibt:

$$M_{HybOpt} = \{SA, TS, GA_{gen}, GA_{ss}, VA\} \quad .$$

Abb. 5.23

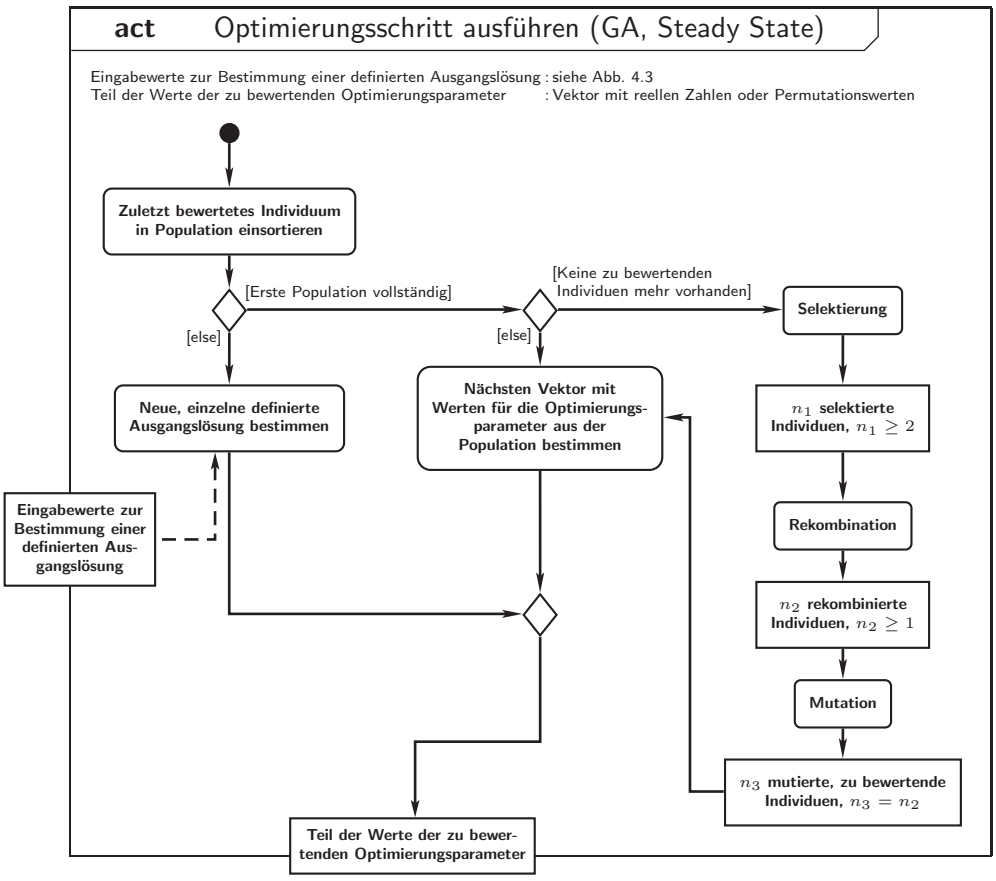


Abb. 5.23.: *Algorithmus zum Ausführen eines Optimierungsschrittes beim Genetischen Algorithmus mit ständiger Ersetzung (UML-Aktivitätsdiagramm)*

Die Nutzung dieser Optimierungsverfahren im Rahmen eines seriell-hybriden und eines parallel-hybriden Optimierungsverfahrens innerhalb des CAOS wurde von Wachsmuth im Rahmen ihrer Diplomarbeit erfolgreich durchgeführt (siehe [Wac05]). Dabei ist anzumerken, dass sich die Nutzung der beiden hybriden Optimierungsverfahren aus Gründen der Allgemeingültigkeit auf die gleiche Weise wie bei den zuvor genannten Optimierungsverfahren gestaltet. Sie besitzen demnach wiederum jeweils einen Initialisierungsschritt und einen Optimierungsschritt.

Bei den *seriell-hybriden Optimierungsverfahrens* wurde eine Variante programmtechnisch umgesetzt, bei welcher im Vorfeld der Optimierung zwei ausgewählte Optimierungsverfahren vom Anwender festzulegen sind. Es handelt sich dabei um ein Eröffnungsverfahren[90] $ov_{HybOpt}^{seqEV} \in M_{HybOpt}$, welches die vorangehende *Groboptimierung* vornimmt, und ein Verbesserungsverfahren $ov_{HybOpt}^{seqVV} \in M_{HybOpt}$, welches für die anschließende *Feinoptimierung* zuständig ist, wobei in dieser Arbeit $ov_{HybOpt}^{seqVV} \neq ov_{HybOpt}^{seqEV}$ sein muss (siehe [Wac05]). Die Groboptimierung dient dazu, den Raum der Entscheidungsvariablen auf Regionen mit „erfolgsversprechenden“ Lösungen zu untersuchen und ähnelt einer globalen, anfänglich zufälligen, Suche

[90] Eröffnungs- und Verbesserungsverfahren besitzen in diesem Zusammenhang eine andere Bedeutung als im Abschn. 4.3.

im Raum der Entscheidungsvariablen. Nach Erreichen eines *Umschaltkriteriums*[91] wird mit dem zur Feinoptimierung ausgewählten Optimierungsverfahren fortgesetzt, welches die beste Lösung des Eröffnungsverfahrens als Startvektor erhält (siehe Abb. 5.24). Anzumerken ist, dass im Rahmen der Nutzung des CAOS sowohl das Eröffnungsverfahren als auch das Verbesserungsverfahren geeignet initialisiert sein müssen (siehe Abb. 5.25). Die Notation für das seriell-hybride Optimierungsverfahren innerhalb dieser Arbeit lautet $HS_{ov^{seqEV}_{HybOpt}, ov^{seqVV}_{HybOpt}}$.

Abb. 5.24

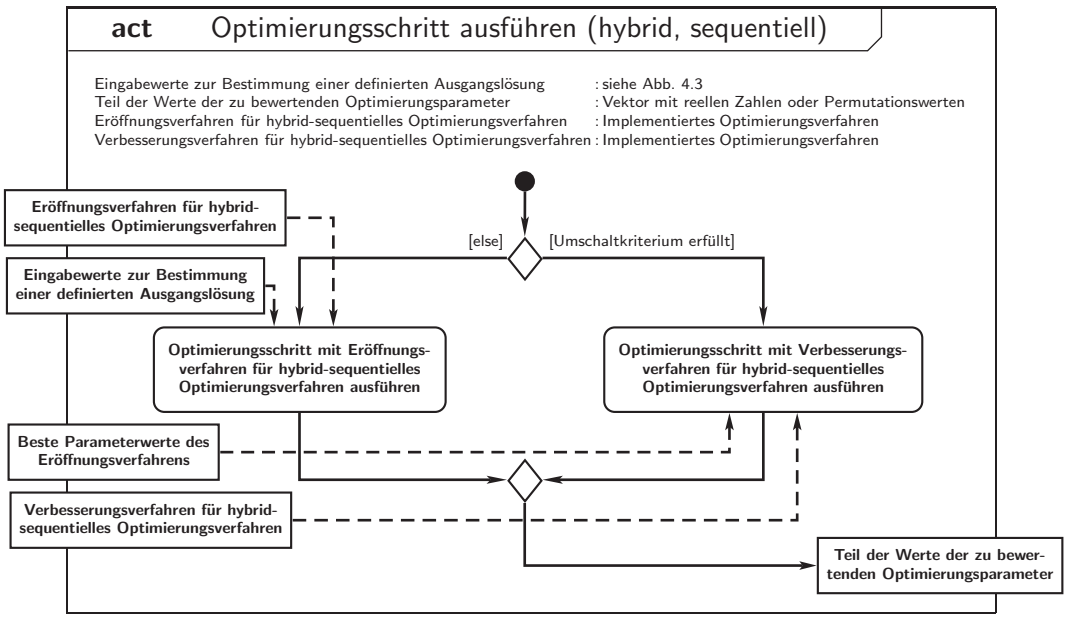


Abb. 5.24.: *Algorithmus zum Ausführen eines Optimierungsschrittes des Hybrid-parallelen Optimierers im CAOS (UML-Aktivitätsdiagramm)*

Abb. 5.25

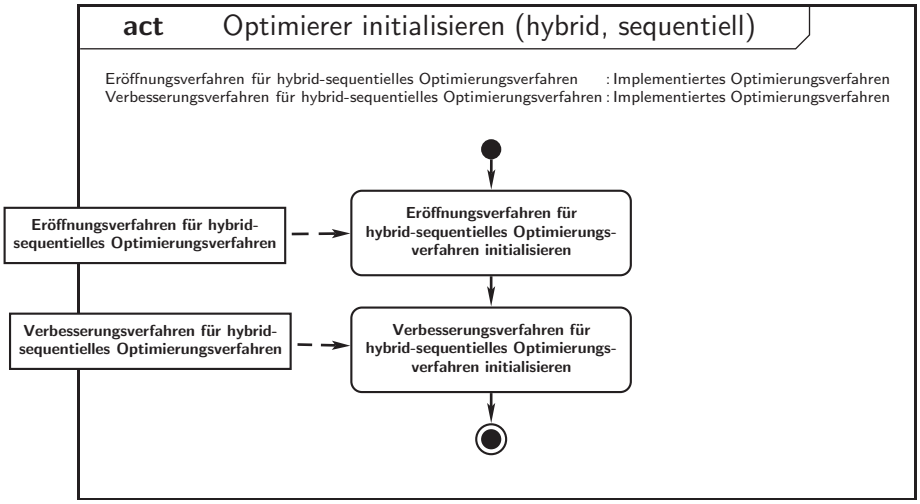


Abb. 5.25.: *Algorithmus zur Initialisierung des Hybrid-sequentiellen Optimierers im CAOS (UML-Aktivitätsdiagramm)*

[91] Es existieren vielfältigste Umschaltkriterien, welche verschiedene Zustände des Optimierungsprozesses berücksichtigen (siehe [Syr97]).

Die im CAOS verwendete Verfahrensvariante für ein *parallel-hybrides Optimierungsverfahren* sieht die gleichberechtigte Nutzung verschiedener Optimierungsverfahren aus M_{HybOpt} vor. Es entsteht eine Menge $M_{HybOpt}^{parV} \subseteq M_{HybOpt}$, deren Elemente sämtlich verschieden sind. Anzumerken ist, dass in M_{HybOpt}^{parV} o. B. d. A. derzeit noch keine zwei Optimierungsverfahren desselben Typs auftreten können, welche sich nur in den Parametern des Optimierungsverfahrens unterscheiden. Für nähere Informationen, wie bspw. den Zeitpunkt, die Art und Weise sowie den Umfang des Austausches von Lösungen zwischen den Optimierungsverfahren der Menge M_{HybOpt}^{parV}, sei wiederum auf die Diplomarbeit von Wachsmuth verwiesen (siehe [Wac05]). Die Notation für das parallel-hybride Optimierungsverfahren innerhalb dieser Arbeit lautet $HP_{M_{HybOpt}^{parV}}$.

Abb. 5.26

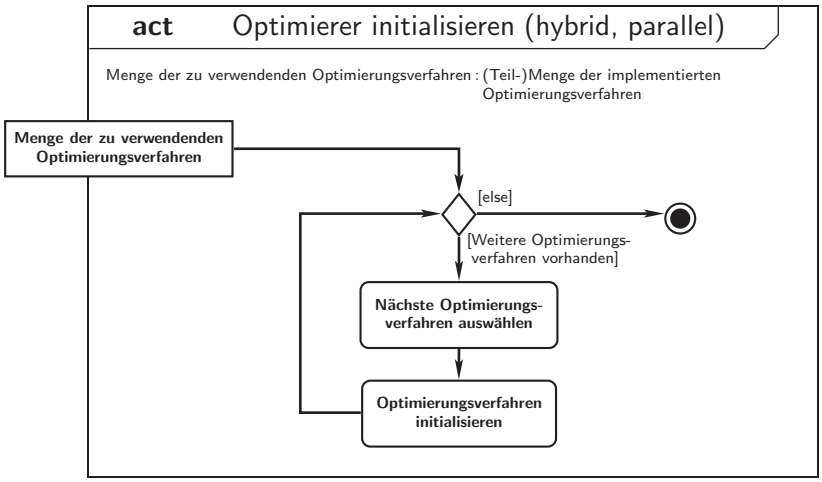


Abb. 5.26.: *Algorithmus zur Initialisierung des Hybrid-parallelen Optimierers im CAOS (UML-Aktivitätsdiagramm)*

Optimierungsverfahren zur verteilten und parallelen Optimierung. Im CAOS wurde ohne genauere Untersuchung MPI unter Nutzung der Bibliothek MPICH zur Implementierung des Optimierungsverfahrens zur verteilten und parallelen Optimierung genutzt (vgl. Abschn. 4.5.2). Dabei wurde sich vorrangig auf das vom Autor in [Käm03] bereits angeeignete Vorwissen gestützt. Prinzipiell ist im CAOS nur der Ablauf des Optimierungsverfahrens zur verteilten und parallelen Optimierung implementiert. Die Nutzung einer speziellen Schnittstelle zur Kommunikation zwischen den vernetzten Rechnern ist nicht festgelegt. Es kann auch eine von MPI verschiedene Implementierung der Kommunikationsschnittstelle gewählt werden. Diese Kommunikationsschnittstelle kann somit durch eine beliebige andere Kommunikationsschnittstelle, welche von der Basisklasse CommunicationManager abgeleitet sein muss, ausgetauscht werden. Soll eine weitere Kommunikationsschnittstelle hinzukommen, muss, durch den Einsatz der Vererbung nur noch der in Form virtueller, abstrakter und überschreibbarer Methoden vorhandene Funktionsumfang durch einen Softwareersteller programmtechnisch umgesetzt werden. In Abb.

Abb. 5.27

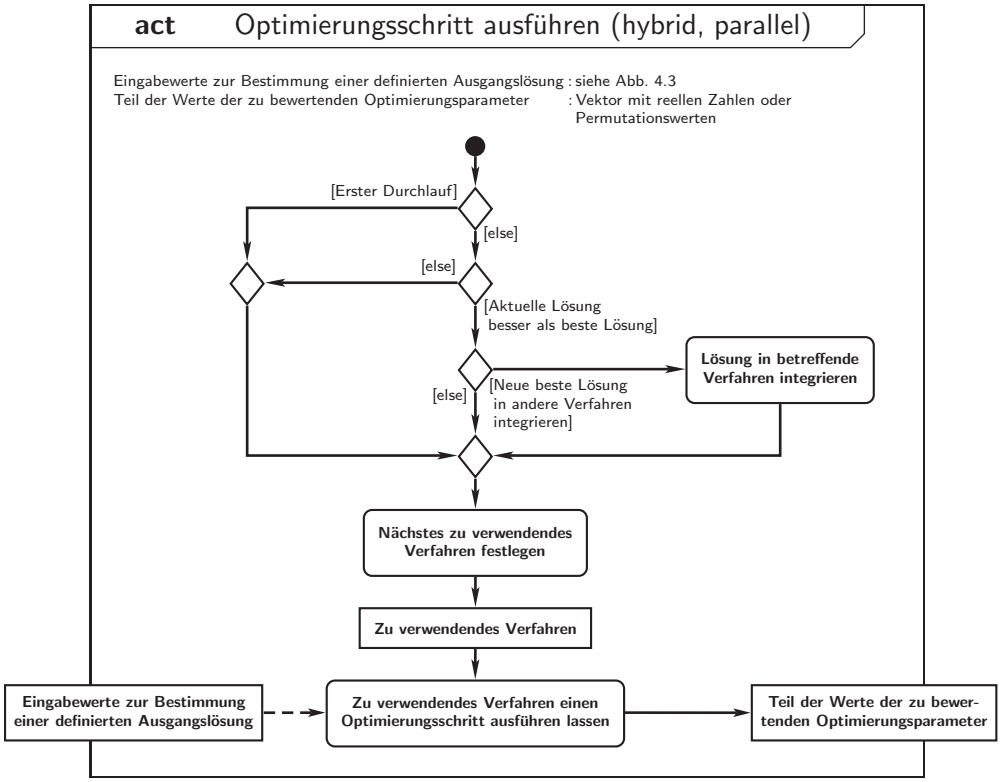


Abb. 5.27.: *Algorithmus zum Ausführen eines Optimierungsschrittes des Hybrid-parallelen Optimierers im CAOS (UML-Aktivitätsdiagramm)*

Abb. 5.28

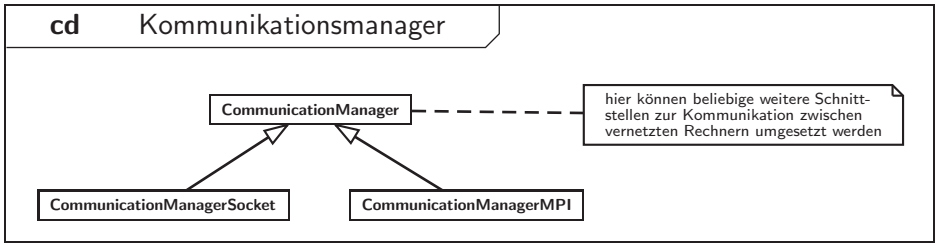


Abb. 5.28.: *Manager für die Kommunikation basierend auf verschiedenen Netzwerkprotokollen (UML-Klassendiagramm)*

5.28 sind die bereits im CAOS vorhandenen Kommunikationsschnittstellen für die Protokolle bei Nutzung von MPI und Sockets[92] abgebildet.

Wie bereits weiter oben beschrieben, wurde der Optimierungsmanager für die Durchführung der verteilten und parallelen Optimierung um eine Realisierung des *Master-Slave-Prinzips* (Klasse OptimisationManagerMasterSlave) erweitert. Dieses Prinzip wurde, bereits im Abschn. 4.5.2 beschrieben und aufgezeigt, dass bei diesem neben dem Master-Rechner noch eine, im Vorfeld der Optimierung festgelegte, statische Anzahl[93] von $n_S = n_R - 1$ Slave-Rechnern existieren. Dabei bezeichne n_R die Gesamtanzahl der zur Verfügung stehenden Rechner für die verteilte und parallele Optimierung. Darüber hinaus steht auch eine *Client-Server-Variante* des Optimierungsmanagers zur Verfügung (Klasse OptimisationManagerClientServer), welcher es erlaubt, Berechnungen und Simulationen auf einem entfernten Rechner auszuführen und den lokalen Rechner nur zur Ausführung des Optimierungsmanagers und der Ergebnisanzeige zu nutzen.

Nachfolgend soll die Arbeitsweise des Master-Slave-Prinzips etwas genauer betrachtet werden. Dazu ist in Abb. 5.29 dargestellt, wie sich die Nutzung des Master-Slave-Prinzips unter Anwendung von MPI als Kommunikationsschnittstelle gestaltet. Dabei führen sowohl der *Master-Rechner* als auch die *Slave-Rechner* jeweils eine Instanz vom CAOS[94] aus und nutzen zum Nachrichtenaustausch die Implementierung MPICH.

Abb. 5.29

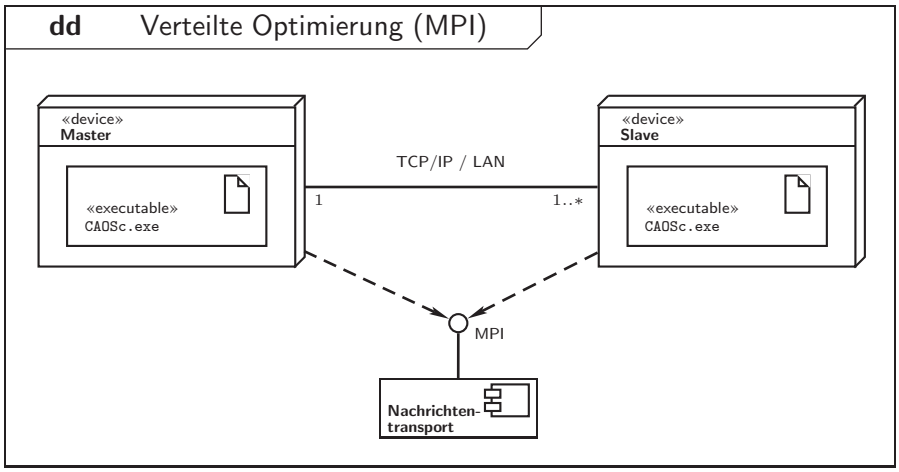


Abb. 5.29.: *Verteilte Optimierung basierend auf dem Message Passing Interface (UML-Verteilungsdiagramm)*

Bei der Ausführung vom CAOS unterscheidet dieses intern selbst, ob der betreffende Rechner Master oder Slave ist und somit, ob es die Master-Methoden (siehe Abb.

[92] Die Socket-Programmierschnittstelle wurde Anfang der 1980er Jahre zunächst in UNIX-Systemen für die Kommunikation zwischen Prozessen eingeführt. Der Begriff Socket bezeichnet dabei den Name für einen Endpunkt einer Kommunikationsverbindung. Die Socket-Schnittstelle ist nicht genormt, stellt aber dennoch einen Industriestandard dar und ist mittlerweile für verschiedene Betriebssysteme verfügbar.

[93] Es wird davon ausgegangen, dass die Anzahl der Slaves während eines Laufes zur simulationsbasierten Optimierung statisch ist, d. h. unverändert bleibt.

[94] In einer Konfigurationsdatei des Optimierungsmanagers wird diesem mitgeteilt, dass er die verteilte und parallele Optimierung unter Nutzung des Master-Slave-Prinzips und MPI durchführen soll.

5.31) oder die Slave-Methoden (siehe Abb. 5.30) ausführen muss. Zum Ende der Optimierung findet noch ein handshake [engl.; Handschlag [dt.]] statt, bei dem der Optimierungsmanager eine Nachricht mit einem Steuerbefehl über das Ende der Optimierung an alle Slaves sendet und auf deren Bestätigung wartet (siehe Abb. 5.30 und Abb. 5.32).

Abb. 5.30

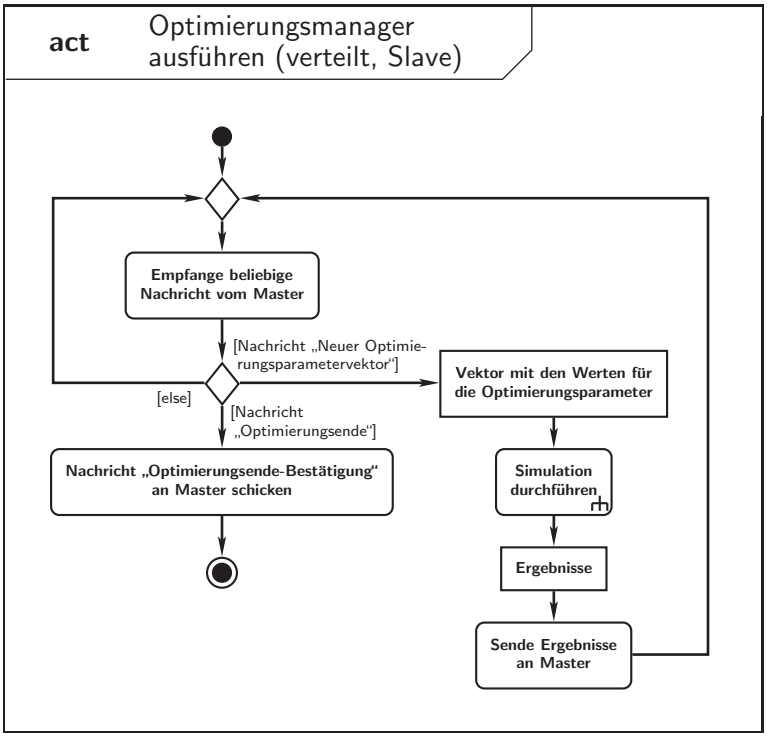


Abb. 5.30.: *Algorithmus zur Ausführung eines Slave-Prozesses zur verteilten und parallelen Optimierung im CAOS (UML-Aktivitätsdiagramm)*

Abb. 5.31

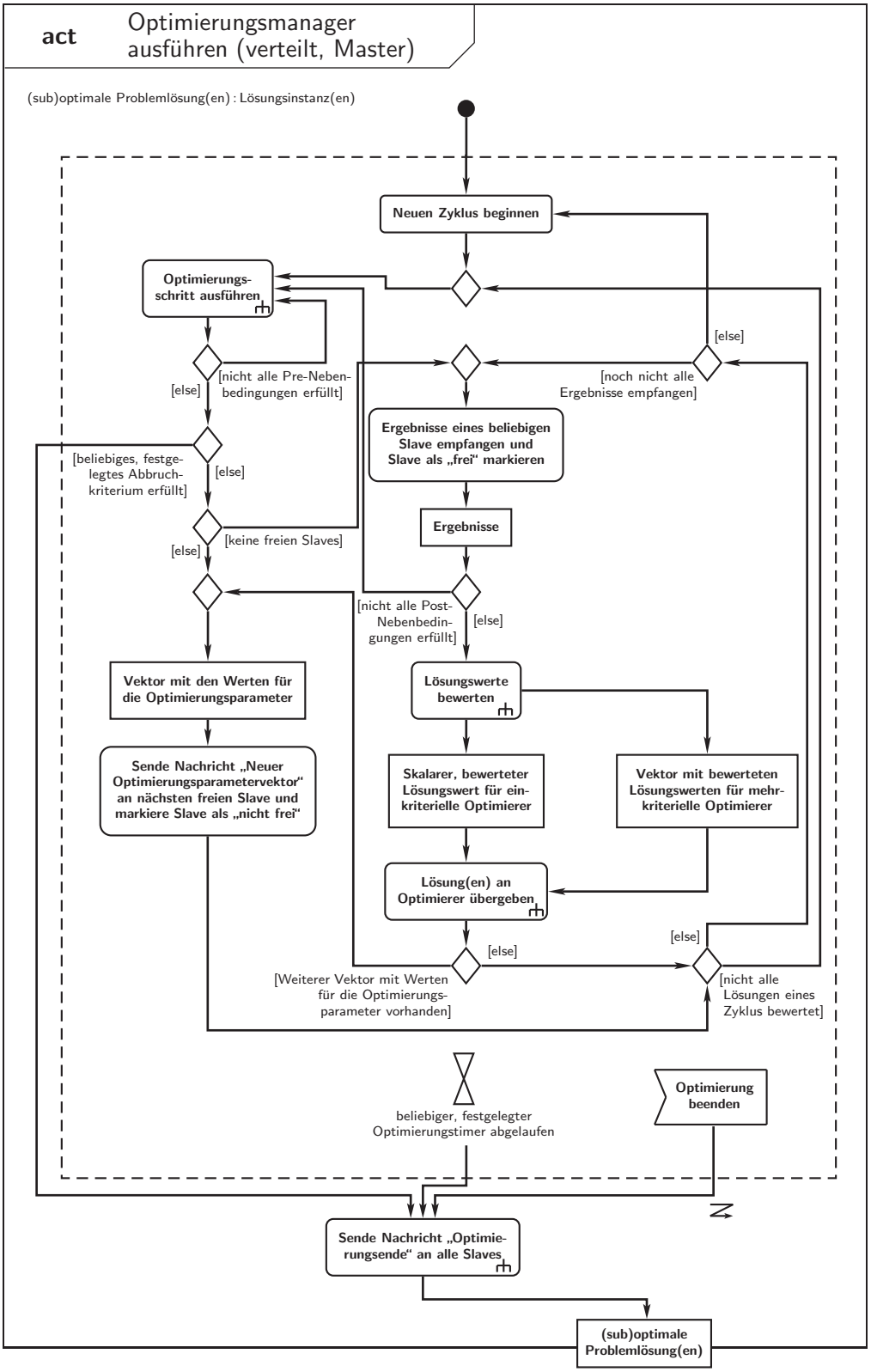


Abb. 5.31.: *Algorithmus zur Ausführung des Master-Prozesses zur verteilten und parallelen Optimierung im CAOS (UML-Aktivitätsdiagramm)*

Abb. 5.32

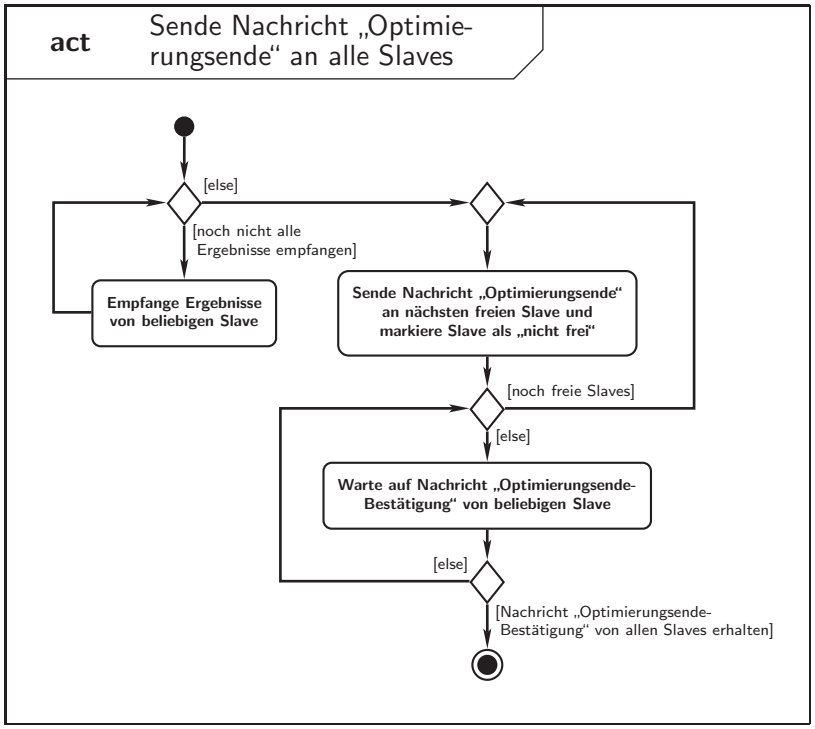


Abb. 5.32.: *Algorithmus zum Beenden der Ausführung des Master-Prozesses bei der verteilten und parallelen Optimierung im CAOS (UML-Aktivitätsdiagramm)*

Der Namensraum `CAO.Assessment`**.** Der Namensraum `CAO.Assessment` enthält die Klassen für mögliche Bewerter, welche z. B. eine gewichtete Güteanalyse von einem Zielfunktionswert oder mehreren Zielfunktionswerten durchführen. Derzeit ist nur die Klasse AssessorLinear zur gewichteten additiven Zusammenfassung mehrerer Zielfunktionswerte enthalten. Der Namensraum `CAO.Assessment` könnte aber um weitere beliebige Klassen zur Zusammenfassung und Beurteilung von Zielfunktionswerten ergänzt werden. Hierfür müsste allerdings auch noch eine geeignete Erweiterung der, allen Modelldefinitionen zugrunde liegenden, einheitlichen Modelldeklaration vorgenommen werden.

Wie bereits im Abschn. 2.4.5 sowie in Abb. 2.23 und Abb. 5.5 gezeigt wurde, können im `CAOS` ein- und mehrkriterielle Optimierungsverfahren gleichzeitig zur Lösung eines Optimierungsproblems, entweder eines SOP oder eines MOP, beitragen. Dazu kann jeder Entscheidungsvariablen ein individuelles Optimierungsverfahren zugeordnet werden. Aus diesem Grund werden bei der Bewertung der Ergebnisse im `CAOS` sowohl die Bewertung für einkriterielle als auch für mehrkriterielle Optimierungsverfahren im Anschluss an eine Simulation gleichberechtigt durchgeführt (siehe Abb. 5.33).

Abb. 5.33

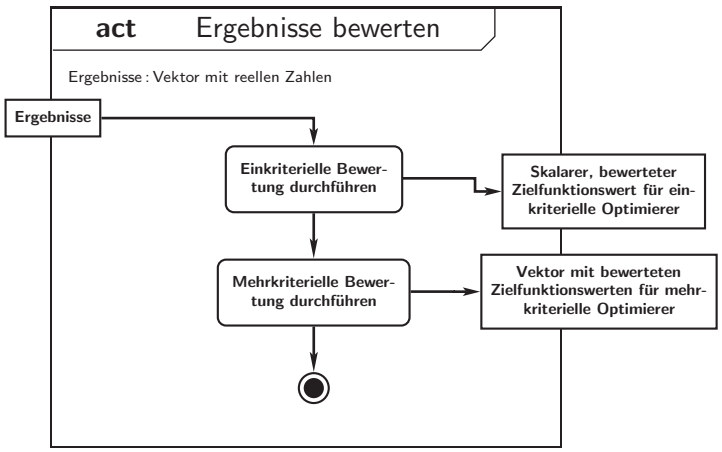


Abb. 5.33.: *Algorithmus zur Bewertung der Ergebnisse einer Simulation oder Berechnung (UML-Aktivitätsdiagramm)*

5.3.7. Anwendungsfälle und Nutzungsmöglichkeiten

Die in Abb. 5.34 aufgezeigten Anwendungsfälle gehen zunächst davon aus, dass sämtliche, zur Durchführung eines Laufes der simulationsbasierten Optimierung, genutzten Optimierungsverfahren, Simulatoren oder analytische Berechnungsvorschriften sowie Bewerter vom Softwaresystem CAOS bereitgestellt werden. Das Softwaresystem CAOS ist durch die Existenz externer Schnittstellen allerdings so flexibel gehalten, dass die Kopplung beliebiger Optimierungsverfahren, Simulatoren oder Berechner möglich ist. Wie sich deren Kopplung gestaltet, wird nachfolgend kurz beschrieben.

Nutzung externer Simulatoren und Berechner. Die Nutzung externer Berechner oder Simulatoren gestaltet sich auf eine recht einfache Art und Weise. Der externe Berechner wird lediglich als neuer Prozess gestartet und die aufgerufene Anwendung, das Softwaresystem CAOS, wartet auf dessen Beendigung. Die für den Programmaufruf benötigten Informationen über das Programm, wie bspw. die zugehörigen Kommandozeilenparameter, die Namen der Ein- und Ausgabedateien, usw. werden in der Modelldefinition eines externen Berechners festgelegt. Ein externer Berechner wird dabei mittels eines Objektes der Klasse CalculatorExtern (siehe Abb. 5.12) gekappselt.

Nutzung externer Optimierungsverfahren. Im Falle, dass ein externes Optimierungsverfahren auf einen Berechner oder Simulator des CAOS bzw. der CAO-Bibliothek (Namensraum CAO) zugreifen möchte, kann dies über die Nutzung der Anwendungsbibliothek CAOSc.exe erfolgen. Diese wird über ihre zugehörige Konfigurationsdatei entsprechend so eingestellt, dass sie keine Optimierung durchführt, sondern die Werte für die Entscheidungsvariablen nur aus der spezifizierten Datei ausliest. Dem geht voraus, dass im Anschluss an die Instanziierung des Berech-

Abb. 5.34

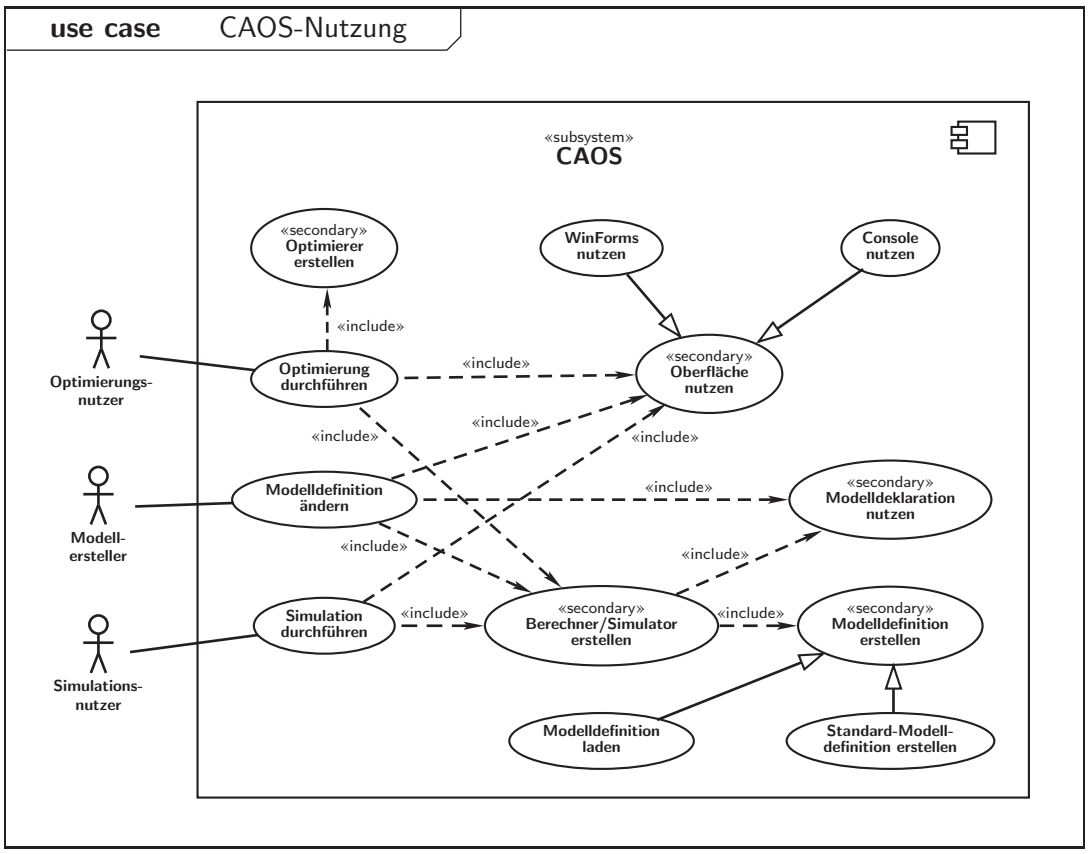


Abb. 5.34.: *Mögliche Anwendungsfälle des CAOS (UML-Anwendungsfalldiagramm)*

ners oder Simulators durch das `CAOS` die geladene Modelldefinition auf Korrektheit überprüft und bei Erfolg die gewünschte Berechnung oder Simulation durchgeführt wird. Die Ergebnisse werden abschließend in der entsprechenden Ergebnisdatei hinterlegt. Dieser Vorgang ist noch einmal in Abb. 5.35 visualisiert. Die weitere Vorgehensweise im Hinblick auf die Verarbeitung der Ergebnisdaten ist von der Implementierung und den bereitgestellten Schnittstellen des nutzenden Softwaresystems mit dem jeweils betreffenden Optimierungsverfahren abhängig.

5.4. Zusammenfassung

In diesem Kapitel wurde der prinzipielle Aufbau des Softwaresystems `CAOS` etwas genauer erläutert, um eine hinreichend detaillierte Wiedergabe der enthaltenen Bibliotheken zu geben, ohne den Umfang dieser Arbeit unnötig zu strapazieren. `CAOS` spiegelt dazu die in den Kapiteln 2, 3 und 4 dargelegten Prinzipien softwaretechnisch wieder.

Aus den Ausführungen dieses Kapitels sollte auch erkennbar werden, dass `CAOS` zwar auf den Verwendungszweck der simulationsbasierten Optimierung und dessen Umfeld beschränkt, aber durch die verschiedensten (externen) Schnittstellen auch sehr flexibel im Hinblick auf Möglichkeiten der Kopplung mit anderen Softwaresys-

Abb. 5.35

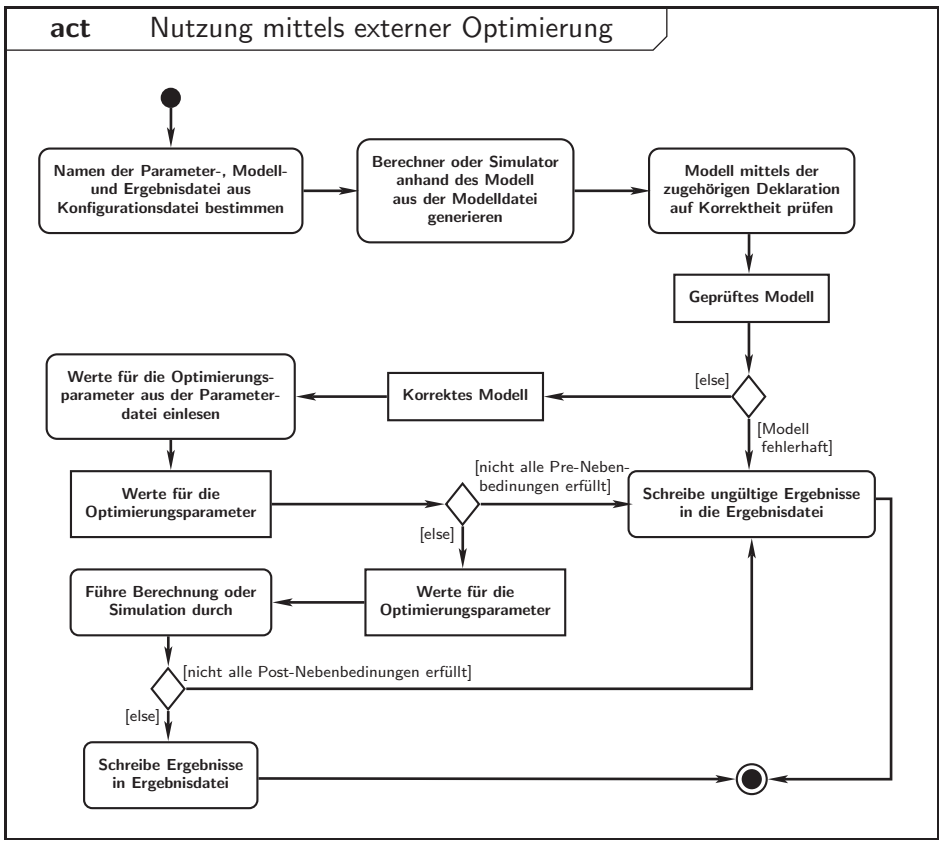


Abb. 5.35.: *Algorithmus zur Nutzung eines Simulators oder Berechners aus dem CAOS mittels externer Optimierung (UML-Aktivitätsdiagramm)*

temen gestaltet ist. Darüber hinaus können und werden Teile des Softwaresystems auch in anderen Anwendungsfeldern eingesetzt, welche nicht im direkten Bezug zu den durchgeführten Untersuchungen stehen. Dies reflektiert die Universalität der verschiedenen entwickelten Bibliotheken vom `CAOS`. Diese Bibliotheken mit unterschiedlichen Zielen bestehen im Wesentlichen aus einer Bibliothek zur Kapselung des Zugriffes auf das Softwaresystem, einer Bibliothek mit verschiedenen Hilfsklassen für Datenzugriffe, Zufallszahlen, usw., einer Bibliothek zum definierten Netzwerkzugriff sowie einer Bibliothek zur Durchführung von Optimierung, Simulation, Berechnung und Bewertung.

Im nachfolgenden Teil der Arbeit wird das Softwaresystem `CAOS` für spezielle Untersuchungen und Optimierungen von, den bereits mehrfach erwähnten, Produktions- und Lagerhaltungssystemen im Rahmen des CSLSP genutzt.

Kapitel 6. Produktions- und Lagerhaltungssysteme mit stochastischen Einflüssen

Inhalt

6.1	*Einführung*	*172*
6.2	*Grundlegende Problemstellungen und Basismodelle*	*173*
6.3	*Simulationsmodell und Optimierungsproblem*	*178*
6.3.1	*Modellkomponenten und -parameter*	*178*
6.3.2	*Entscheidungsvariablen*	*181*
6.3.3	*Restriktionen (Nebenbedingungen)*	*191*
6.3.4	*Kostenbetrachtung und Optimierungskriterien*	*191*
6.3.5	*Zusammenfassung der genutzten Größen*	*193*
6.4	*Modelltypen*	*195*
6.4.1	*Modelltyp 1: Eine Fertigungseinheit und N Produkte*	*202*
6.4.2	*Modelltyp 2: M Fertigungseinheiten und N Produkte*	*204*
6.4.3	*Modelltyp 3: M Fertigungseinheiten, N Produkte und Berücksichtigung von Lieferketten*	*206*
6.5	*Modellbeispiele mit Ergebnissen*	*212*
6.5.1	*Voruntersuchungen*	*212*
6.5.2	*Modellbeispiele und Ergebnisse für Modelltyp 1*	*215*
6.5.3	*Modellbeispiele und Ergebnisse für Modelltyp 2*	*223*
6.5.4	*Vergleichsmodelle von Markowitz, Reimann und Wein*	*229*
6.6	*Zusammenfassung*	*232*

6.1. Einführung

In der Gegenwart sind kaum Produktionssysteme[95] vorhanden, welche ohne ein nachfolgendes *Lager zur Aufbewahrung der gefertigten Produkte* auskommen. Physisch sind Lager im Anschluss an die Produktion stets vorhanden, wenn auch nicht immer unmittelbar ortsgebunden an den Produktionsprozess. Sie treten teilweise in flexibler Form, z. B. als Transportfahrzeuge, auf. Die *Lagerhaltungskosten* werden dadurch zwar verringert, fallen aber zumeist dennoch an, z. B. in Verbindung mit den Transportkosten. Darüber hinaus entstehen durch einen Produktionsprozess i. Allg. *Produktionskosten* und im Falle der Fertigung mehrerer Produkte auf einer Fertigungseinheit zusätzlich *Umrüstkosten* beim *Produktwechsel* von einem Produkt zu einem anderen Produkt oder infolge von Produktionsstillständen. Die Betrachtung weiterer möglicherweise auftretender Kosten führt u. a. zu *Fehlmengenkosten*, welche bei einer Unterdeckung des Lagers entstehen. In theoretischen Modellen können diese in ähnlicher Form durch Warteschlangen für Kundenforderungen abgebildet werden. In einer Warteschlange mit begrenzter oder unbegrenzter Kapazität können unbefriedigte Kundenforderungen aufbewahrt werden, siehe [Lip92]. Dadurch entstehen produkt- und zeitabhängige *Wartekosten*. Ist eine Warteschlange mit begrenzter Kapazität gefüllt, fallen für jede weitere unbefriedigte Kundenforderung *Abweiskosten* an. Fehlmengenkosten oder Warte- und Abweiskosten stehen daher im direkten Bezug zum *Image-Gewinn* oder *-Verlust*.

Die zuvor angeführten, verschiedenen Kostenbestandteile sollen einen ersten Einblick in die Vielfältigkeit bei der Bewertung von Produktions- und Lagerhaltungssystemen geben. Weitere Kostenbestandteile oder quantitative Aussagen, wie bspw. der Servicegrad[96] oder die Maschinenauslastung[97] können für die Betrachtung der Güte eines Modells ebenfalls herangezogen werden. Es ist jedoch darauf zu achten, dass, wie zuvor am Beispiel der Fehlmengen- und Wartekosten gezeigt, nicht verschiedene äquivalente Kostenbestandteile[98] miteinander verglichen werden. Bei den meisten (theoretischen) Modellen bilden die Kosten den Hauptbestandteil bei der Bewertung. Davon gehen auch die nachfolgenden Untersuchungen und die vorgestellten Problemstellungen im Rahmen der Reihenfolge- und Losgrößenoptimierung aus.

Dennoch sind auch weitere Kenngrößen eines Modells von Bedeutung, um einen Vergleich zwischen zwei Modellen durchführen zu können. Hierzu zählt bspw. die Betrachtung des *Zeithorizonts*. Es ist zu entscheiden, ob in einem Modell von *diskreten Zeitintervallen* oder *stetiger Zeit* ausgegangen werden soll. Eine weitere Kenngröße eines Modells stellen die (zu beachtenden) *Kapazitätsbeschränkungen* dar. Beispielsweise sollte eine Fertigungseinheit zur Vermeidung bzw. Minimierung

[95] Die Begriffe Produktion und Fertigung werden in dieser Arbeit synonym verwandt. Produktionssysteme seien innerhalb dieser Arbeit bspw. gleichbedeutend mit Fertigungssystemen.
[96] Der Servicegrad gibt hierbei das prozentuale Verhältnis von befriedigten Kunden zu Gesamtkunden an.
[97] Die Maschinenauslastung bezeichnet das Verhältnis zwischen der Zeitdauer für die auf einer Maschine gefertigten Aufträge zur Fertigungsgesamtzeit.
[98] Die Äquivalenz entsteht an dieser Stelle durch die Umformbarkeit der charakteristischen Merkmale eines Modells ineinander.

von Störfällen nicht zu mehr als ca. 90% und kann, als obere Grenze, nicht zu mehr als 100% ausgelastet sein[99]. Diese Gegebenheiten werden in den nachfolgenden Unterabschnitten etwas genauer betrachtet, um zu den Unterschieden der neuartigen Untersuchungsmodelle hinzuführen. Einen weiter reichenden, allgemein gültigen Überblick über die verschiedenen Modellvarianten oder -variationen, welche sich bei Produktions- und Lagerhaltungssystemen ergeben können, ist in Tab. 6.1 gegeben. Dadurch soll die Vielfalt der Kombinationsmöglichkeiten zur Erstellung von Modellen für Produktions- und Lagerhaltungssysteme verdeutlicht werden.

Nachfolgend wird sich auf den Kreis der (stochastischen) einstufigen Mehr-Produkt-Fertigungsprobleme beschränkt, bei denen die einzelnen Losgrößen der N zu fertigenden Produkte und ggf. deren Produktionsreihenfolge im Hinblick auf die entstehenden Gesamtkosten optimiert werden sollen. Im Abschn. 6.4.3 wird zusätzlich die Anzahl der Fertigungsstufen durch die Festlegung einer Fertigungszeit innerhalb einer Lieferkette [dt.; supply chain [engl.]] erhöht.

6.2. Grundlegende Problemstellungen und Basismodelle

Mit der Produktionsplanung sind konkrete Fragestellungen verbunden, wie z. B.:

1. *Zu welchem Zeitpunkt soll die Fertigung beginnen?*
2. *Welches Produkt ist zu fertigen?*
3. *Wie viel ist von diesem bestimmten Produkt herzustellen?*
4. *Auf welcher Anlage ist das Produkt zu fertigen?*

Diese Fragestellungen zielen direkt auf Probleme ab, welche sich mit der *Optimierung von Losgrößen, Reihenfolge und Zuordnung* sowie indirekt mit der betreffenden *Bestellstrategie* für die einzelnen Produkte beschäftigen. In der Literatur werden diese daher als Losgrößenprobleme [dt.; *batch sizing problems* [engl.] oder *lot sizing problems* [engl.]], Reihenfolgeprobleme [dt.; *scheduling problems* [engl.]] und Zuweisungsprobleme [dt.; *allocation problems* [engl.]] bezeichnet (siehe [Bru04] oder [Qua04]).

Bekannte Problemstellungen. Bereits Anfang des letzten Jahrhunderts hat Harris in [Har13] und [Har15] ein grundlegendes Modell, das *EOQ* (Abkürzung für *Economic Order Quantity* [engl.]), vorgestellt, welches ein Ein-Produkt-Modell ohne Kapazitätsbeschränkungen für einen stetigen Planungszeitraum mit unendlichem Zeithorizont beschreibt. Für dieses Modell zeigt beispielsweise Wilson in [Wil34], wie die optimale Lösung analytisch bestimmt werden kann. Das Modell wurde 1958 von Rogers zu einem Mehr-Produkt-Modell mit Kapazitätsbeschrän-

[99] Der angegebene Wert ist vom Autor nur fiktiv gewählt und kann durchaus variieren. Er soll lediglich angeben, dass eine sehr hohe Auslastung anzustreben ist, aber keine 100%-ige Auslastung erreicht werden sollte.

Tab. 6.1

Charakteristik	Mögliche Werte
Abstraktionsgrad der Daten (z. B. Kundenankünfte, -bedarfe, Produktionszeiten)	• *Deterministisch* • *Stochastisch*
Zeitliche Entwicklung der Modellparameter	• *Statisch (stationär)* • *Dynamisch*
Planungszeitraum (Zeithorizont)	• *Unendlich* • *Endlich (Diskrete Zeitperioden)*
Anzahl zu fertigender Produkte	• *1* • *N*
Anzahl paralleler Fertigungseinheiten	• *1* • *M*
Anzahl der Dispositions- und Fertigungsstufen	• *1* • *K*
Berücksichtigung von Kapazitäten (z. B. Lagerplatz, Produktionsressourcen, Finanzmittel, Fertigungsgeschwindigkeit)	• *ja (endlich)* • *nein (unendlich)*
Berücksichtigung von Fehlmengen	• *nicht erlaubt* • *Verzugsmengen (Back Order)* • *Verlorene Aufträge (Lost Sales)*
Berücksichtigung von Mindestbestellmengen	• *ja (vorgegeben)* • *nein (nicht vorgegeben)*
Berücksichtigung des Zulieferstromes (z. B. für Rohmaterial)	• *ja (endlich, kapazitiert)* • *nein (unendlich)*
Lager	• *nicht vorhanden (Kapazität von 0 Teilen)* • *unkapazitiert* • *vorgegebene Kapazität (Kapazität von P Teilen)*
Durchlaufzeiten	• *(fest) vorgegeben (produktabhängig)* • *(dynamisch) während Modellausführung, z. B. durch Scheduler, bestimmt*
Art der Produktweitergabe	• *offen (beliebige Losgrößen)* • *geschlossen (feste Losgrößen)*
Kostenbestandteile	• *Umrüstkosten [dt.; setup costs [engl.]] (mengenunabhängig/-abhängig, produktunabhängig/-abhängig)* • *Lagerhaltungskosten [dt.; holding costs [engl.]]; auch Kapitalbindung, Opportunitätskosten* • *Produktionskosten [dt.; production costs [engl.]]* • *Fehlmengenkosten [dt.; shortage costs [engl.]]* • *Sonstige, weitere Kosten (z. B. Wartekosten [dt.; waiting costs [engl.]], Abweiskosten [dt.; rejection costs [engl.]])*
Optimierungskriterien	• *Einzelkosten, Gesamtkosten* • *Auslastung* • *Durchsatz* • *Verspätung* • *. . .*

Tab. 6.1.: *Überblick über die Charakteristiken von Produktions- und Lagerhaltungssystemen*

kungen für die einzelne Fertigungseinheit, dem *ELSP* (Abkürzung für *Economic Lot Scheduling Problem*[100] [engl.]), erweitert (vgl. Tab. 6.2). Die Kapazitätsbeschränkungen der Fertigungseinheit treten beim ELSP meist als maximale Fertigungsrate auf. In der *stochastischen Variante des ELSP*, dem *SELSP* (siehe [FK96])[101] sind sämtliche Kundenankunftsraten und -bedarfe stochastisch. Gleiches gilt für die Umrüst- und Produktionszeiten. Hsu bewies in [Hsu83], dass das ELSP von seiner Komplexität her ein $\mathcal{NP}$-vollständiges Problem ist, während das SELSP als $\mathcal{NP}$-schweres Problem klassifiziert ist. Aus diesem Grund kann gesagt werden, das alle auf dem SELSP aufbauenden bzw. erweiternden Problemstellungen als $\mathcal{NP}$-schwer angesehen werden können. Sie erweitern den Raum der Entscheidungsvariablen um zusätzliche Dimensionen.

Die Problemstellungen zum EOQ, ELSP und SELSP sollten nur einen Anhaltspunkt darüber geben, wie vielfältig die Problemstellungen zur Losgrößen- und Reihenfolgeoptimierung gestaltet sein können. In Tab. 6.2 ist ein ergänzender Überblick über einige weitere, bekannte Losgrößen- und Reihenfolgeprobleme dargestellt. Die einzelnen Problemstellungen werden dabei anhand der in Tab. 6.1 angegebenen Charakteristiken eingeordnet. Es wird jeweils von einem anfänglichen Basisproblem ausgegangen und das jeweils daraus, durch Veränderung ausgewählter Charakteristiken, resultierende Problem angegeben. Die Problemstellung zum EOQ als grundlegendes Ausgangs-Basisproblem sei dabei durch folgende, wesentliche Charakteristiken gekennzeichnet:

- ein Produkt,
- keine Kapazitätsbeschränkungen,
- eine einstufige Fertigungsanlage mit deterministischer Fertigungsrate,
- feste Losgröße,
- stetiger Planungszeitraum (unendlicher Zeithorizont),
- fortlaufende, konstante Zwischenankunftszeiten für Kundenankünfte und deterministische Kundenbedarfe,
- deterministische (Umrüst- und) Produktionszeiten sowie
- keine Fehlmengen und Bedarfsvormerkungen erlaubt

mit dem Ziel der *Minimierung von (Umrüst- und) Lagerkosten*.

Eine weiter reichende Übersicht über Losgrößen- und Reihenfolgeprobleme, welche bis etwa Mitte der 90er Jahre untersucht wurden, ist zudem in [KS94] angegeben. In der aktuellen Literatur sind kaum neue Ansätze zu finden. Es wird lediglich der Versuch unternommen, die bisherigen Problemstellungen effizienter zu lösen.

[100] Das economic lot scheduling problem [engl.] wird in seiner vollen Schreibweise auch als economic lot sizing and scheduling problem [engl.] bezeichnet. Diese verkürzte Schreibweise gilt analog auch für alle anderen, erwähnten Losgrößen- und Reihenfolgeprobleme [dt.; lot scheduling problems [engl.]].

[101] Zur Vollständigkeit sei erwähnt, dass die stochastische Version des ELSP in der Literatur auch als *SLSP* (Abkürzung für *Stochastic Lot Scheduling Problem* [engl.], siehe [SM97]), *dynamisches Scheduling-Problem* [dt.; *dynamic scheduling* [engl.]] (siehe [RW98]) oder nur *Losgrößenproblem* [dt.; lot-sizing problem [engl.]] (siehe [WS04]) bezeichnet wird.

Tab. 6.2

Problem (Basisproblem)	Unterschiede zum Basisproblem	Literaturverweis(e)
ELSP (EOQ)	• *N Produkte* • *Kapazitätsbeschränkungen bei der Fertigung*	[Elm78], [Hsu83], [Zip91]
WW (EOQ)	• *endlicher Zeithorizont mit T Perioden* • *dynamische Losgrößen in den einzelnen Perioden* • *variable Kundenbedarfe*	[WW58], [FT91]
SELSP (ELSP)	• *stochastische Zwischenankunftszeiten für Kundenankünfte und variable Kundenbedarfe* • *stochastische Umrüst- und Produktionszeiten* • *Fehlmengen erlaubt* • *zusätzlich noch Minimierung der Kosten für Bedarfsvormerkungen*	[FK96], [MRW00], [SWd04], [WS04]
CLSP (ELSP)	• *endlicher Zeithorizont mit T Perioden [dt.; time buckets [engl.]]* • *dynamische Losgrößen in den einzelnen Perioden (Unterscheidung zwischen Modellen mit kurzen oder langen Perioden [dt.; small time buckets [engl.] oder big time buckets [engl.]])* • *beim CLSP zunächst lange Perioden* • *variable Kundenbedarfe*	[HK96], [Mag01], [SS03], [FMT04]
MLLP (WW)	• *Serienfertigung oder beliebige Fertigungsart*	[PW91], [Atk94]
CSLP (CLSP)	• *nutzt im Gegensatz zum CLSP kurze Perioden*	[KS85], [PW91]
DLSP (CLSP)	• *nur ein Produkt pro Periode (Produktionsstrategie „entweder alles oder nichts“)* • *teilweise mit produktabhängigen Umrüstkosten*	[Fle90], [Fle94]
CLSP-SC (CLSP)	• *nutzt im Gegensatz zum CLSP i. Allg. kurze Perioden* • *Umrüstvorgänge auf mehrere Perioden verteilbar*	[GDBM01]
CLSPL (CLSP)	• *nutzt im Gegensatz zum CLSP i. Allg. kurze Perioden* • *Los über mehrere Perioden fertigbar*	[SS03]
CSLSP (CLSP/-SELSP)	• *Vereinigung von SELSP und CLSP, d. h. stochastische Einflüsse und Einteilung in mehrere Perioden* • *teilweise Kapazitätsbeschränkungen auch beim Lager*	[DM96], [Cla03], [KK06], [HOP07]

Legende:

EOQ	-	Economic Order Quantity (problem) [engl.]
ELSP	-	Economic Lot Scheduling Problem [engl.]
SELSP	-	Stochastic Economic Lot Scheduling Problem [engl.]
CLSP	-	Capacitated Lot Scheduling Problem [engl.]
DLSP	-	Discrete Lot Scheduling Problem [engl.]
MLLP	-	Multi-Level Lot-sizing Problem [engl.]
CSLP	-	Continuous Setup Lot-sizing Problem [engl.]
CLSP-SC	-	Capacitated Lot Scheduling Problem with Setup Carryover [engl.]
CLSPL	-	Capacitated Lot Scheduling Problem Linked lot-sizes [engl.]
CSLSP	-	Capacitated Stochastic Lot Scheduling Problem [engl.]
WW	-	Wagner-Whitin(-Problem)

Tab. 6.2.: *Überblick über Reihenfolge- und Losgrößenprobleme (Auswahl)*

Unterschiedliche Ansätze. Für die verschiedenen Losgrößen- und Reihenfolgeprobleme existieren in der Literatur die *verschiedensten Modellierungs- und Lösungsansätze*. Elmaghraby und Zipkin geben in [Elm78] und [Zip91] sowie Kuik und Salomon in [KS94] einen einführenden Literaturüberblick darüber. Dieser wird in [BDPNN06] um weitere heutige Problemstellungen ergänzt. Die unterschiedlichen Lösungsansätze erstrecken sich dabei von *exakten Lösungsverfahren* (vgl. Abschn. 4.2), welche zumeist auf die Nutzung von mixed integer programming [engl.] zurückgreifen, bis hin zu *heuristischen Suchverfahren* (vgl. Abschn. 4.3.5), wie die (speziell angepasste) Tabusuche (siehe [GDBM01]) oder die (parallelen) Genetischen Algorithmen (siehe [BKS95] und [HK96]). Das gewählte Losgrößen- und Reihenfolgeproblem steht dabei oftmals in direktem Zusammenhang zum genutzten Lösungsansatz, so dass deren Unabhängigkeit nicht gewährleistet ist. Bei Problemstellungen mit Kapazitätsrestriktionen, wie bspw. dem CLSP, wird versucht durch exakte Lösungsverfahren und ggf. Verringerung der stochastischen Einflüsse durch Mittelwert-Bildung oder Nutzung von Verteilungsfunktionen mit einfach zu berechnenden Kenngrößen, wie sie bspw. die Exponentialverteilung aufweist, ein globales Optimum zu bestimmen. Können keine solchen Vereinfachungen oder Verallgemeinerungen getroffen werden bzw. liegen $\mathcal{NP}$-schwere Problemstellungen vor, ist die Nutzung von heuristischen Suchverfahren unabdingbar, um innerhalb einer vorgegebenen Zeit eine (sub)optimale Problemlösung zu bestimmen (vgl. Abschn. 4.3).

Des Weiteren ist in Tab. 6.2 erkennbar, dass im Wesentlichen verschiedene Spezialisierungen der unterschiedlichen Basisprobleme EOQ, ELSP und WW entwickelt wurden. Diese zielen insbesondere auf die Nutzung stochastischer Einflussfaktoren (SELSP und CSLSP), die Existenz mehrstufiger Fertigungseinheiten (MLLP), Kapazitätsbeschränkungen (CLSP, CSLSP), Produktdynamik usw. ab. Neben der Nutzung stochastischer Einflüsse ist der weitere, wesentliche Problem- und Modellunterschied die zeitliche Problembetrachtung (diskrete Zeitintervalle oder stetige Zeitfortschreibung). Auch diese hängt meist direkt mit dem gewählten Lösungsansatz zusammen.

Hiesige Problemstellung und untersuchte Modelle zu Produktions- und Lagerhaltungssystemen. Aus dem vorangehenden Unterabschnitt wird erkennbar, dass sich die Untersuchungen im Bereich von Fertigungssystemen aus Sicht der Produktanzahl in Modelle für Ein-Produkt- und Mehr-Produkt-Systeme unterteilen lassen. Während Ein-Produkt-Systeme im Praktischen kaum noch Anwendung finden, weil sie i. d. R. nur eine erste, einfache, verallgemeinernde Abstraktion eines in der Praxis vorkommenden Systems darstellen und somit nur theoretischen (Vor)Untersuchungen dienen, sind Mehr-Produkt-Systeme durchaus von wirtschaftlichem Interesse. *Mehr-Produkt-Systeme* entstehen meist durch eine Vielfalt von unterschiedlichen Produkten, verschiedenen Variationen eines Produktes oder einer Kombination aus beidem. Sie seien auch Gegenstand der hiesigen Untersuchungen.

Des Weiteren seien *Kapazitätsrestriktionen* sowohl für die *Produktion* als auch für die sich anschließende *Lagerung* in den untersuchten Modellen vorgegeben. Die

maximale Produktionsmenge wird direkt durch die Produktionsgeschwindigkeit der Fertigungseinheit(en) und indirekt durch die notwendigen *Umrüstungen beim Produktwechsel* restringiert. Im Gegenzug ist der entscheidende Parameter eines Lagers dessen *Lagergröße*, d. h. der physische Raum, welcher für die *Zwischen-* oder *Endlagerung* der gefertigten Produkte tatsächlich bereitsteht. Aus diesen Vorgaben resultiert direkt, dass Modelle erstellt werden, welche sich an das *CSLSP* anlehnen.

6.3. Simulationsmodell und Optimierungsproblem

Zurückgehend auf die einleitenden Fragestellungen von Abschn. 6.2 ist noch zu untersuchen, wie sich die optimale Losgröße und die optimale Produktionsreihenfolge bestimmen lassen. Dies soll nach einigen Bemerkungen zu Gemeinsamkeiten der untersuchten Modelle (siehe Abschn. 6.3.1) im Abschnitt 6.3.2 geklärt werden.

6.3.1. Modellkomponenten und -parameter

Modellkomponenten. Im Wesentlichen sind bei der Modellierung eines Simulators für die Losgrößen- und Reihenfolgeoptimierung unter Nutzung der simulationsbasierten Optimierung drei Aspekte zu berücksichtigen:

1. die Auswahl und Abbildung der entscheidenden Modellparameter des Simulationsmodells und der Entscheidungsvariablen des Optimierungsproblems,
2. die Festlegung der notwendigen Nebenbedingungen sowie
3. die Angabe der Zielfunktion(en) unter Nutzung der zu bewertenden Ausgabewerte und ggf. die Angabe der statistischen Ausgabewerte.

Punkt 1 zielt auf die zu treffende Entscheidung ab, welche der betreffenden Modellkomponenten als Modellparameter (Simulationsmodellparameter bei der simulationsbasierten Optimierung) und welche als Modellvariable (Entscheidungsvariable bei der simulationsbasierten Optimierung) abgebildet werden sollen. Diese Entscheidung hängt größtenteils auch von der Entscheidungsstufe ab. Am Beispiel der Lagergröße könnte man diesen Sachverhalt wie folgt verdeutlichen: auf der operativen Ebene (z. B. bei der Reihenfolgeplanung) ist eine feste Lagergröße zu wählen. Hingegen ist auf der taktischen oder strategischen Ebene (z. B. Losgrößen- oder Standortplanung) die Wahl einer variablen Lagergröße in Form einer Entscheidungsvariable sinnvoll. Unter Punkt 2 werden im Wesentlichen die Produktion und das Lager restringiert. Jedoch können auch die gewählten Strategien zur Festlegung der Losgrößen weitere Nebenbedingungen hervorrufen. Die Angabe der Zielfunktionen (Punkt 3) bzw. die in einer einzelnen Zielfunktion enthaltenen Kostenbestandteile kann sich sehr vielfältig gestalten. Bei den meisten Modellen aus der Fachliteratur werden nur spezielle Kosten, wie bspw. die Warte-, Umrüst- und Lagerkosten, berücksichtigt (siehe [MRW00] und [Cla03]). Teilweise spielt noch die Maximierung der Auslastung der Fertigungseinheit(en) im Hinblick auf das Erreichen einer möglichst hohen Fertigungsrate mit wenigen Umrüstvorgängen ei-

ne entscheidende Rolle (siehe [FK96] und [WG06]). Sie kann bspw. als (zusätzliche) Nebenbedingung formuliert werden.

Untersuchungsmodelle. In dieser Arbeit werden drei Modelltypen untersucht, die sich in der Anzahl der Fertigungseinheiten sowie dem Zulieferstrom unterscheiden. Der Zulieferstrom gibt dabei an, wie die Zulieferung von Rohteilen erfolgt. In Tab. 6.3 sind die wesentlichen Unterscheidungsmerkmale gegenübergestellt. Modelltyp 1 ist demzufolge der Modelltyp mit dem höchsten Abstraktionsgrad.

Tab. 6.3

Modell-typ	Anzahl der Produkte	Anzahl der Fertigungs-einheiten	Zulieferstrom	Klasse im CAOS
1	*N*	1	nicht berücksichtigt	MultipleItemSystem
2	*N*	*M* (eine Fertigungsstufe)	nicht berücksichtigt	MultipleItemMultiple-MachineSystem
3	*N*	*M* (eine Fertigungsstufe)	berücksichtigt, Steuerung durch vereinfachte Lieferkette mit stochastischen Lieferein-flüssen	MultipleItemSupply-ChainSystem

Tab. 6.3.: *Überblick über die Unterschiede der untersuchten Modelle*

Modellparameter. Neben den in Tab. 6.1 bereits angegebenen Charakteristiken der Simulationsmodellparameter, wie Zeithorizont, Entwicklung der Modellparameter, Grad der Datenabstraktion, Produktanzahl, Kapazitätsbeschränkungen, u. Ä., sollen nachfolgend diese Modellparameter zum besseren Verständnis vereinzelt noch einmal aufgegriffen und um weitere, wesentliche Simulationsmodellparameter ergänzt werden. In den Tabellen 6.5 bis 6.7 sind die in dieser Arbeit genutzten Simulationsmodellparameter und Entscheidungsvariablen zusammenfassend angegeben.

Kundenbedarfsprozess. Die *Kundenbedarfsmengen* für ein bestimmtes Produkt werden von außen als Modellparameter festgelegt und treten entweder als *konstante* bzw. *zeitinvariante* oder *dynamische* bzw. *zeitvariante* Größen auf. Konstante Kundenbedarfsmengen werden meist bei Modellen mit unendlichem Zeithorizont eingesetzt, währenddessen dynamische Kundenbedarfsmengen unterschiedliche Forderungsmengen in den einzelnen Perioden hervorrufen. Beide Größen können aber zudem jeweils *stochastischen Einflüssen* unterliegen.

Die Kundenbedarfsmengen entstehen infolge einer *einmaligen Forderung* oder durch einen sich *ständig erneuernden (dynamischen) Forderungsprozess*, welcher zu bestimmten Zeitpunkten eintritt (vgl. compound renewel process [engl.]). Diese Zeitpunkte werden durch die *Kundenankunftsrate*, mit der die Kunden ein bestimmtes Produkt mit gewisser Häufigkeit verlangen, festgelegt. Sie können wiederum als konstante oder dynamische Größen auftreten und stochastischen Einflüssen unterliegen.

Produktionsprozess. Die Güte des Produktionsprozesses ist i. Allg. gekennzeichnet durch einen *Servicegrad*, welcher die (unmittelbare) Verfügbarkeit der einzelnen Produkte widerspiegelt. Soll ein Servicegrad von 100% in einem System mit stochastischen Einflüssen erzielt werden, führt dies meist zu erhöhten Kosten für das Umrüsten, die Produktion sowie die Lagerung der Produkte. Die Modelle werden dann oftmals um ein System für Vormerkungen von Kundenbedarfen in Form von Warteschlangen erweitert. Dies führt im Falle einer endlichen, gefüllten Warteschlange mitunter auch zu abzuweisenden Kunden. Folglich sind in solchen Systemen normalerweise auch Kosten für Vormerkungen und Abweisungen vorzusehen und diese in die Zielfunktion mit der Repräsentation der Gesamtkosten mit aufzunehmen. Dies erfolgt auch innerhalb dieser Arbeit.

Des Weiteren werden für die Produktion oftmals *Kapazitätsbeschränkungen* in Form von *maximalen Produktionsraten* oder einer *maximalen Auslastung der Produktion* angegeben. Teilweise werden diese Kapazitätsbeschränkungen jedoch auch mittels der Zielfunktion(en) im Modell abgebildet. Auch für die Produktion fallen i. Allg. Kosten an. Innerhalb der betrachteten Untersuchungsmodelle finden sie allerdings Verwendung.

Eine zusätzliche Kenngröße des Produktionsprozesses sind die *Durchlaufzeiten* für die einzelnen Aufträge bzw. Lose, wobei die Losgrößen entweder von außen vorgegeben (exogen, statisch vorgegeben) oder während der Ausführung eines Simulationsexperimentes bestimmt und dann als ein Simulationsergebnis nach außen gegeben werden (endogen, dynamisch bestimmt). In dieser Arbeit sind die Losgrößen sowohl als exogene Größen während eines Simulationsexperimentes sowie auch als endogene Größen während eines Laufes zur simulationsbasierten Optimierung vorhanden.

Als letzte Kenngröße des Produktionsprozesses sei die *Produktionsstruktur* erwähnt, welche vielfältig gestaltet sein kann. Sie erstreckt sich von einer einzelnen Fertigungseinheit über l_K Fertigungseinheiten bei K Produktionsstufen bis hin zu einer vorgegebenen Produktionsstruktur, welches Reihen- oder Serienfertigung, eine Montagefertigung, eine Variantenfertigung sowie ein beliebiges, allgemein gültiges Fertigungsnetzwerk wiedergeben kann. In der vorliegenden Arbeit wird von einer Fertigungsstufe mit einer oder l_1 Fertigungseinheiten sowie von einem beliebigen Fertigungsnetzwerk ausgegangen. Bei dem betrachteten Fertigungsnetzwerk wird die letzte Produktionsstufe mit l_K Fertigungseinheiten näher betrachtet und die vorangehenden Produktionsstufen werden abstrahiert.

Lager. Als weitere Kenngröße der untersuchten Modelle tritt ein sich an die *Produktion anschließendes Lager* auf. Dieses kann *unbeschränkt* oder auf eine *bestimmte Anzahl von P Raumeinheiten für die Teile einzelner Produkte beschränkt* sein, wobei jedes Produktteil einen produktabhängigen Lagerplatz von p_i, $i = 1, 2, \ldots N$ Raumeinheiten beansprucht. Für die Lagerung der einzelnen Produktteile entstehen zumeist Lagerkosten. Anstatt der oberen Grenze P für alle Produkte könnte auch eine individuelle Grenze für die einzelnen Produkte P_i, $i = 1, 2, \ldots N$ angegeben sein. In dieser Arbeit wird jedoch ohne nähere Untersuchungen von einem gemeinsamen Lager für alle Teile ausgegangen.

Zielfunktionen. Wie bereits zuvor an mehreren Stellen angemerkt, entstehen für die einzelnen ablaufenden Prozesse individuelle Kosten in Form von Umrüst-, Produktions-, Abweis-, Vormerk- und Lagerkosten. Diese werden während eines Simulationslaufes aufsummiert und dann als resultierender Zielfunktionswert in Form von Kosten pro Zeiteinheit zugegeben. Der zu erzielende Gewinn ist dabei zu maximieren bzw. die entstehenden Kosten zu minimieren. Zusätzlich können die einzelnen Kostenbestandteile noch individuell gewichtet werden.

Eine andere Bildung des Zielfunktionswertes, welche oftmals im Zusammenhang mit ausschließlichen Reihenfolgeproblemen auftritt, sieht eine Minimierung der maximalen Fertigungszeit [dt.; completion time [engl.]], der maximalen Verspätung [dt.; lateness [engl.]], der Gesamtfertigungszeit [dt.; total completion time [engl.]] sowie der Gesamtverspätung [dt.; total tardiness [engl.]] vor (siehe [Köb99]). Das Problem an dieser Vorgehensweise ist ebenfalls die sinnvolle Gewichtung der einzelnen Größen.

6.3.2. Entscheidungsvariablen

In [Sch02] wird von Schneeweiss eine oftmals verwendete Trennung von strategischer, taktischer und operativer Ebene zur Einteilung des Produktionsprozesses in verschiedene Entscheidungsebenen beschrieben. Demnach findet auf der taktischen Ebene die Losgrößenoptimierung und auf der operativen Ebene die Reihenfolgeoptimierung statt. Innerhalb dieser Arbeit wird jedoch eine kombinierte Losgrößen- und Reihenfolgeoptimierung betrachtet, so dass die Unterteilung in verschiedene Ebenen weitestgehend aufgehoben wird. Aus diesem Grund können auch die Entscheidungsvariablen beider Optimierungsprobleme innerhalb einer Modelldefinition des `CAOS` auftreten. Die innerhalb dieser Arbeit entscheidenden Probleme der Losgrößen- und Reihenfolgeoptimierung sowie der gewählte hierarchische Ansatz der kombinierten Losgrößen- und Reihenfolgeoptimierung werden nachfolgend beschrieben.

Losgrößenoptimierung. Die Losgrößenoptimierung beschäftigt sich mit Problemen der Produktionsplanung und dort im Speziellen mit der Bestimmung der optimalen Losgröße eines Produktionsloses. Die Losgröße bezeichnet hierbei die Anzahl der zusammenhängend zu fertigenden Teile eines Produktes. Die Komplexität dieses Problems ergibt sich u. a. indirekt durch die Existenz von Umrüstzeiten und -kosten zwischen den Produktionslosen verschiedener oder gleicher Produkte unter Nutzung kapazitierter Produktionsressourcen (siehe [Mag01]). Diese sind bei den untersuchten Modellen die bereits mehrfach angesprochenen Fertigungseinheiten.

Einerseits führt die Produktion von kleinen Losgrößen zu einer besseren Bedarfsbefriedigung. Andererseits erfordert diese i. Allg. höhere Umrüst- und Produktionskosten. Im Gegensatz dazu kommt es bei Mehr-Produkt-Fertigung und großen Losgrößen zu geringeren Umrüstkosten, aber höheren Lagerhaltungskosten. Größere Losgrößen können zudem indirekt zu einer Verschlechterung des Servicegrades

in Folge wartender oder abzuweisender Kunden führen, wodurch zusätzlich höhere Warte- und Abweiskosten entstehen würden. Das Problem der Bestimmung der optimalen Losgröße wird somit im Mehr-Produkt-Fall durch die Existenz von Umrüstkosten zu einem $\mathcal{NP}$-schweren Problem, wie bereits Hsu in [Hsu83] bewies.

Neben der optimalen Losgröße ist auch zu klären, ab welchem Zeitpunkt mit der Produktion eines Produktionsloses begonnen werden soll (*Freigabe-Zeitpunkt*). Zur Beantwortung dieser beiden Fragen, wie viel und wann produziert werden soll, sind in der Literatur verschiedene statistische Inventar-Kontroll-Politiken [dt.; statistical inventory control policies [engl.]] oder kurz Fertigungs- bzw. Bestellpolitiken[102] zu finden, welche jeweils auf die Minimierung bestimmter Kosten abzielen. In Tab. 6.4 sind dazu verschiedene dieser Fertigungspolitiken gegenübergestellt. Sie werden unter dem Begriff Make-To-Stock [engl., kurz: MTS; „Erzeuge auf Lager“ [dt.]] oder produce-to-stock [engl.; „Erzeuge/Produziere auf Lager“ [dt.]] zusammengefasst. Es wird sich bei diesen Fertigungspolitiken nur am *aktuellen Lagerbestand* orientiert, wobei jeweils möglichst so viel zu produzieren ist, wie unmittelbar oder innerhalb der nächsten Zeit durch Kundenforderungen benötigt wird. Bei Mehrproduktion gelangt diese ins Lager. Demgegenüber steht die Fertigungspolitik Make-To-Order [engl., kurz: MTO; „Erzeuge auf Bestellung“ [dt.]] oder produce-to-order [engl.; „Produziere auf Bestellung“ [dt.]], bei der nur genau soviel produziert wird, wie ein Kunde verlangt. Diese Fertigungspolitik orientiert sich daher nur an den Kundenforderungen und eine Mehrproduktion kann dabei nicht entstehen. Näheres zu den verschiedenen Fertigungs- bzw. Bestellpolitiken ist in [Grü94] und [Vry04] zu finden.

Im Vorfeld der Untersuchungen entschied sich der Autor, innerhalb dieser Arbeit die Fertigungspolitiken (s, nQ) und (s, S) näher zu betrachten.

Bei den MTS-Fertigungspolitiken vom Typ (x, S) bleibt die Frage offen, ab welchem Zeitpunkt die obere Fertigungsgrenze S_i zu berücksichtigen ist. Dazu existieren verschiedene Vorgehensweisen, welche aus Polling-Modellen (siehe [Tak86]) bekannt sind und sich an der Menge der Vormerkungen bis zu einem bestimmten Zeitpunkt orientieren. Im Wesentlichen sind folgende Politiken denkbar:

exhaustive service policy [engl.].
Die Fertigungseinheit(en) produzieren solange bis alle Vormerkungen befriedigt sind.

gated service policy [engl.].
Es werden nur die Fertigungsaufträge produziert, welche bis zu einem bestimmten Zeitpunkt vorgemerkt waren.

limited service policy [engl.].
Es wird nur solange produziert bis alle Vormerkungen oder eine bestimmte Menge von Vormerkungen befriedigt ist.

Für die durchgeführten Untersuchungen wurde nur die gated service policy [engl.]

[102] Nachfolgend wird stets von Fertigungspolitiken und nicht von Bestellpolitiken gesprochen, weil in dieser Arbeit Fertigungssysteme und nicht Bestellsysteme im Vordergrund stehen.

Tab. 6.4

Fertigungs-menge	Fertigungs-periode	Fertigungs-politik	Beschreibung
fest	fest	(R, Q)	Es ist zu festen Zeitpunkten R_i eine Losgröße von Q_i Teilen zu fertigen.
	variabel	(s, Q)	Der aktuelle Lagerbestand des Produktes i hat seine untere Grenze s_i unterschritten, wodurch eine Losgröße von Q_i Teilen zu fertigen ist (constant-lot-size policy [engl.]).
		(s, nQ)	Ähnlich der Strategie (s, Q), nur dass Q_i Teile n-mal zu fertigen sind.
		(s, T)	Bei Erreichen von s_i wird eine Zeit von T_i gefertigt (limited-time policy [engl.]).
	fest/variabel	(R, s, Q)	Bei Erreichen von R_i oder s_i wird Q_i produziert.
variabel	fest	(R, S)	Zu festen Zeitpunkten R_i ist bis zur oberen Grenze S_i zu produzieren.
	variabel	(s, S)	Der aktuelle Lagerbestand des Produktes i hat seine untere Grenze s_i erreicht, wodurch so viele Teile zu fertigen sind, bis die obere Grenze S_i erreicht ist (base-stock policy [engl.]), für $s_i < S_i$.
	fest/variabel	(R, s, S)	Bei Erreichen von R_i oder s_i wird bis S_i produziert.
fest/variabel	fest/variabel	$(s, \min(Q, T))$	Beim Erreichen von s_i werden Q_i Teile gefertigt oder ggf. nur T_i Zeiteinheiten lange produziert (limited-production policy [engl.]).

Legende:

- s - untere Fertigungsgrenze (Meldebestand)
- Q - Losgröße/Fertigungsmenge
- S - obere Fertigungsgrenze (Sollbestand)
- R - Fertigungsperiode/-rhythmus

Tab. 6.4.: *Überblick über die verschiedenen Fertigungspolitiken zur Bestimmung einer optimalen Losgröße und des Fertigungs-Zeitpunktes (Auswahl)*

ausgewählt, ohne an dieser Stelle eine nähere Güteanalyse vornehmen zu wollen. In der praktischen Anwendung ist es durchaus denkbar, dass auch die anderen Politiken an verschiedenen Stellen ihre Anwendung finden.

Unter Beachtung der Beschränkung des Lagers auf eine bestimmte Kapazität sind darüber hinaus weitere statische oder dynamische Politiken denkbar. Ein Beispiel ist die vom Autor entwickelte statische *Ideal-Stock-Politik*, welche im Vorfeld eines Simulationsexperimentes die vorhandene Lagerkapazität auf die einzelnen Produkte so aufteilt, dass diese jeweils nur einen eingeschränkten Raum zur Verfügung haben. Wesentliches Kriterium für die Aufteilung ist dabei der jeweilige Kundenbedarf. Je höher der Bedarf an einem Produkt ist, desto mehr Lagerkapazität bekommt es zugewiesen. Auf das Problem der Restriktion des Lagers und den daraus resultierenden Produktionsausfällen wird innerhalb dieser Arbeit jedoch nicht näher eingegangen. Dennoch erlauben die vorgestellten Modelle prinzipiell eine Beschränkung des Lagers.

Abschließend sei noch angemerkt, dass die Aufteilung in Lose und die Optimierung der Losgröße nicht zwingend zur optimalen Produktionsentscheidung beitragen muss. Kuik und Salomon bemerken dazu in [KS94, S. 251]

> Lotsizing is not the real issue. The real issue is to design production processes that are so flexible that production quantities (batches) equal customer demand quantities, and timing of production is such that inventory positions are almost zero.

Jedoch ist bei den meisten reellen Systemen mit stochastischen Einflüssen schwer, diese Behauptung zu berücksichtigen.

Reihenfolgeoptimierung. Stehen ausreichend Fertigungseinheiten zur Verfügung (trivialer Weise $M = N$), um einen Fertigungsauftrag unverzüglich ohne Umrüsten zu produzieren, wäre eine Reihenfolgeoptimierung unnötig. Davon soll nachfolgend allerdings nicht ausgegangen werden. Unter Beachtung beschränkter Produktionskapazitäten, welche einerseits in Form der Anzahl vorhandener Fertigungseinheiten $M < N$ und andererseits der Produktionsgeschwindigkeit dieser Fertigungseinheiten μ_{ik}, $i = 1, 2, \ldots, N$ und $k = 1, 2, \ldots, M$ vorgegeben sind, wird es notwendig, eine Produktionsreihenfolge für die einzelnen Fertigungsaufträge festzulegen. Die Existenz von Umrüstzeiten im Falle eines Produktwechsels verstärkt diese Forderung noch zusätzlich.

Die Reihenfolgeoptimierung einer fest vorgegebenen Menge von Fertigungsaufträgen erzeugt ein *klassisches Schedulingproblem* (siehe [Bru04]). Die Reihenfolgeoptimierung findet daher auf der operativen Ebene statt, d. h. einerseits ist die vorgegebene Menge von Fertigungsaufträgen vorgegeben und andererseits wird damit auch der Zeithorizont beschränkt. Bei der Existenz einer Fertigungseinheit entsteht ein TSP bei dem die Kantengewichte die Umrüstzeiten oder -kosten sind. Es ist die Reihenfolge [dt.; schedule [engl.]] gesucht, welche die geringsten Umrüstzeiten oder -kosten[103] erzeugt. Im Falle mehrerer Fertigungseinheiten würde ein m-TSP entstehen. Für das TSP und das m-TSP siehe auch [GI05] oder [Käm06].

Ist hingegen die Menge von Fertigungsaufträgen offen, d. h. im Vorfeld unbekannt oder deren Berechenbarkeit $\mathcal{NP}$-schwer[104], gestaltet sich die Bestimmung der optimalen Reihenfolge weitaus schwieriger. Es entsteht ein dynamisches Reihenfolgeproblem (siehe [AFL+00]), bei dem das zu lösende TSP oder m-TSP sich dynamisch ändernde Kantengewichte aufweist, welche sich anhand der zukünftig entstehenden und aktuellen Umrüstzeiten oder -kosten ergeben. Solche dynamischen Reihenfolgeprobleme treten auch in den Untersuchungsmodellen auf. Aufgrund dieser Problematik wird von einer Bestimmung der optimalen Reihenfolge abgesehen und der Einsatz heuristischer Methoden angestrebt. Diese werden nachfolgend im Unterpunkt „Gewählter Ansatz“ beschrieben.

[103] An dieser Stelle sei nur von Umrüstkosten und -zeiten ausgegangen. Korrekter Weise müssten aber auch andere Kosten oder ggf. entstehende Produktionszeiten mit einbezogen werden.

[104] Die Komplexität der Berechenbarkeit ergibt sich durch die beliebige Verteilung der Zwischenankunftszeiten der Kundenbedarfe.

Gewählter Ansatz (Kombinierte Losgrößen- und Reihenfolgeoptimierung). In der vorliegenden Arbeit wird ein hierarchischer Ansatz gewählt, welcher eine *kombinierte Losgrößen- und Reihenfolgeoptimierung* vornimmt. Als Losgrößen- und Freigabezeitpunktentscheidung[105] (*LF-Entscheidung*) wird eine Strategie (*LF-Strategie*) gemäß (s, nQ) oder (s, S) angewandt (siehe Tab. 6.4), um die optimale Losgröße und den Freigabezeitpunkt zu bestimmen. Die optimalen Werte für s_i, S_i oder Q_i für $i = 1, 2, \ldots, N$ werden durch ein heuristisches Suchverfahren bestimmt. Es entstehen die Entscheidungsvariablen des Optimierungsproblems $\vec{x}_{sQ} = (\vec{s}, \vec{Q})$ oder $\vec{x}_{sS} = (\vec{s}, \vec{S})$. Unterschreitet die *Inventarmenge* [dt.; *inventory position* [engl.]] eine untere Grenze s_i für ein Produkt i wird ein Fertigungsauftrag abhängig von der gewählten Losgrößenstrategie ausgelöst.

- für LF-Strategie (s, nQ): Es werden zu einem (bestimmten) Zeitpunkt $t_{prüf}$ so viele Lose mit einem Fertigungsauftrag in Höhe von Q_i Teilen ausgelöst (n-mal), bis die Inventarmenge $I_i(t_{prüf})$ die untere Grenze s_i wieder übersteigt $(s_i \leq I_i(t_{prüf}))$.
- für LF-Strategie (s, S): Bei Erreichen der unteren Grenze s_i zum Zeitpunkt $t_{prüf}$, d. h. $s_i \geq I_i(t_{prüf})$, wird ein Los für einen Fertigungsauftrag in Höhe von $S_i - I_i(t_{prüf})$ Teilen ausgelöst. Tritt vor Beginn der Produktion des Loses dieses Fertigungsauftrages $(t \leq t_{prod})$ eine erneute Verletzung der unteren Grenze s_i auf, wird das Los entsprechend angepasst (s. o. „gated service policy [engl.]“).

Die Inventarmenge $I_i(t)$ des Produktes i zum Zeitpunkt t setzt sich dabei wie folgt zusammen:

$$I_i(t) \;=\; I_i^+(t) - I_i^-(t) + M_i(t) \tag{6.1}$$

mit

$$\begin{aligned} I_i^+(t) &= \max(I_i(t) - M_i(t), 0) \quad \text{und} \\ I_i^-(t) &= \max(M_i(t) - I_i(t), 0) \quad , \end{aligned}$$

wobei

$M_i(t)$	-	Anzahl der Teile, welche in den Fertigungsaufträgen für das Produkt i zum Zeitpunkt t im Pool mit den Losen für die Fertigungsaufträge existieren,
$I_i^+(t)$	-	Anzahl der physisch verfügbaren Teile des Produktes i zum Zeitpunkt t sowie
$I_i^-(t)$	-	Anzahl der vorgemerkten Teile für das Produkt i zum Zeitpunkt t.

Die LF-Entscheidung wird demnach mittels verschiedener Kenngrößen des aktuellen Systemzustandes getroffen (wartender/vorgemerkter Kundenbedarf, zu produzierende Teile und Lagerbestand). Bei beiden LF-Strategien (s, nQ) und (s, S)

[105] Nachfolgend wird die Losgrößen- und Freigabezeitpunktentscheidung teilweise nur als Losgrößenentscheidung bzw. Losgrößenstrategie bezeichnet.

ist die zu produzierende Anzahl von Teilen eines Produktes gleich der Losgröße der jeweiligen Fertigungsaufträge dieses Produktes. Zum besseren Verständnis der unterschiedlichen Vorgehensweisen der LF-Strategien (s, nQ) und (s, S) sind diese in Abb. 6.1 an einem Beispiel gegenübergestellt.

Wie bereits erwähnt wurde, gelangen die Fertigungsaufträge, bevor sie die eigentliche Fertigung erreichen, in einen sog. *Pool mit Losen für die Fertigungsaufträge*. Dabei handelt es sich um einen virtuellen Pool, weil dieser physisch nicht existent ist. In dem Pool selbst existiert eine beliebige aber zu jedem Zeitpunkt t feste Ordnung der Lose mit den zur Fertigung anstehenden Aufträgen (kurz: Fertigungslose). Jedes Fertigungslos besteht, wie zuvor schon dargelegt wurde, aus einer festen (LF-Strategie (s, nQ)) oder dynamischen (LF-Strategie (s, S)) Anzahl von Teilen desselben Produktes. Die Ordnung der Fertigungslose wird mittels der Reihenfolgestrategie (*R-Strategie*) festgelegt, welche durch eine vorgegebene Heuristik (*Reihenfolgeheuristik*) bestimmt wird, die sich während eines Simulationslaufes nicht verändert. Zur Untersuchung werden verschiedene *statische* und *dynamische Reihenfolgestrategien* eingesetzt, die nachfolgend etwas näher betrachtet werden.

Statische Reihenfolgestrategien. Bei den statischen Reihenfolgestrategien ist die Reihenfolge für die Abarbeitung der Fertigungsaufträge fest vorgegeben und kann im Pool mit den Losen für die Fertigungsaufträge nicht mehr verändert werden. Es werden allerdings Teile des aktuellen Zustandes des Simulationsmodells berücksichtigt. Im Wesentlichen finden in dieser Arbeit folgende statische Reihenfolgestrategien ihre Anwendung:

FCFS (Abkürzung für First Come First Serve [engl.]).
Das Los, welches als Erstes in den Pool mit den Losen für die Fertigungsaufträge gelegt wurde, gibt Auskunft darüber, welcher Fertigungsauftrag als Nächster zu produzieren ist.

LCFS (Abkürzung für Last Come First Serve [engl.]).
Ähnlich der Strategie FCFS, nur, dass das Los als Nächstes betrachtet wird, welches als Letztes in den Pool gelangt ist.

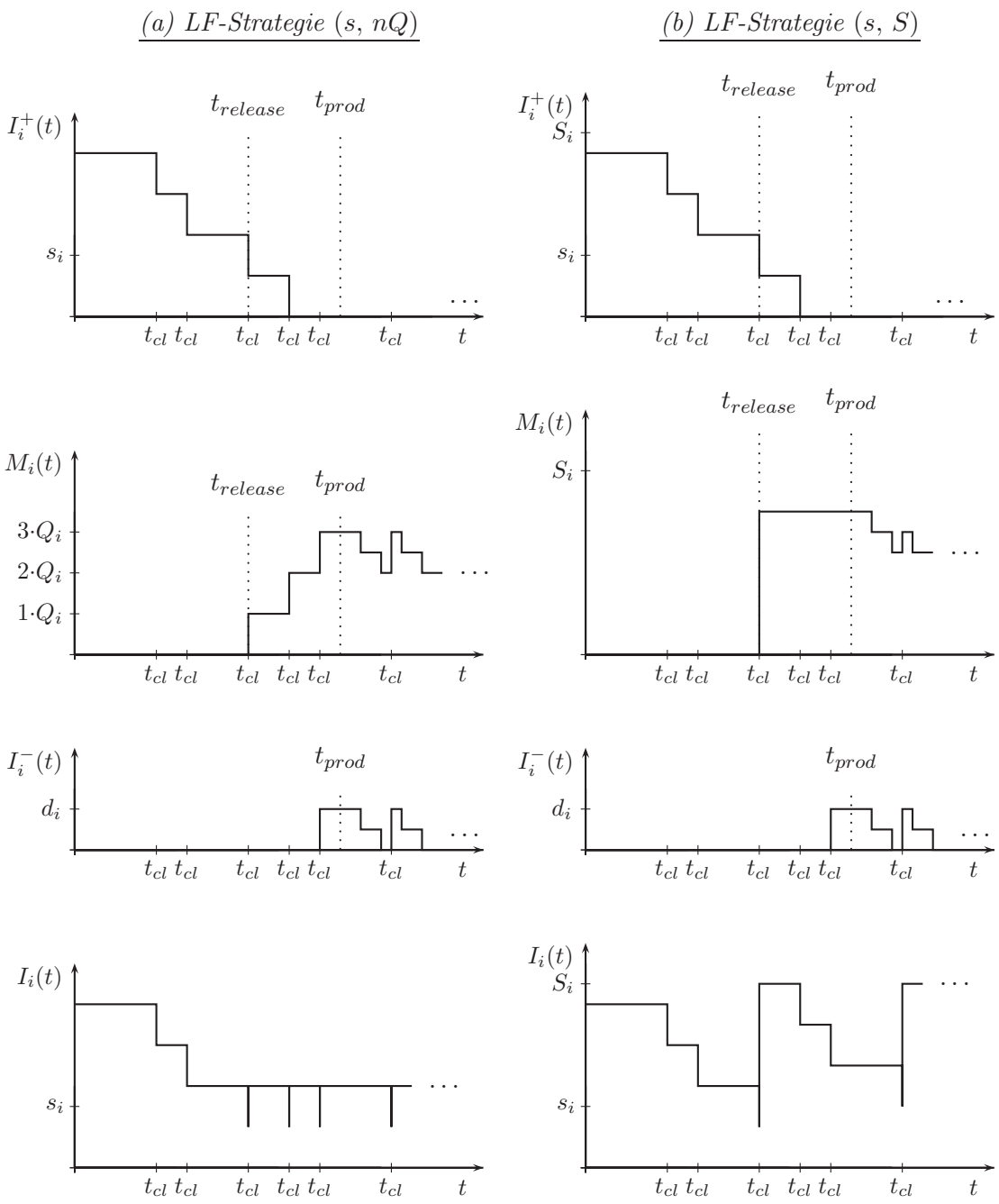
Cyclic [engl.; zyklisch [dt.]].
Das nächste zu wählende Los wird nach einem zyklischen Reihenfolgeprinzip gewählt (s. u.).

Nachfolgend wird betrachtet, wie sich die Bestimmung eines Zyklus bei der zyklischen Reihenfolgestrategie Cyclic gestalten kann. Das Ziel der Zyklusbestimmung ist die Festlegung eines sich wiederholenden, statischen Zyklus

$$Z = (z(1),\, z(2),\, \ldots,\, z(N)) \quad ,$$

welcher über den Zeitraum der Durchführung eines Simulationsexperimentes konstant bleibt. Mit der Bezeichnung „wiederholt“ ist gemeint, dass sich der Zyklus Z in einer fest vorgegebenen Weise stetig wiederholt und nicht nach dem Ende

Abb. 6.1



Legende:

t_{cl}	- *Zeitpunkt einer Kundenankunft*
t_{prod}	- *Zeitpunkt des Produktionsbeginns*
$t_{release}$	- *Zeitpunkt der Freigabe eines Fertigungsauftrages*
$I_i(t)$, $I_i^+(t)$, $I_i^-(t)$, $M_i(t)$	- *s. o.*

Abb. 6.1.: *Vergleich zwischen den LF-Strategien* (s, nQ) *und* (s, S) *(Beispiel)*

eines Zyklus variiert bzw. neu bestimmt wird. Folglich ergibt sich als fortlaufender Zyklus

$$z(1),\, z(2),\, \ldots,\, z(N),\, z(1),\, z(2),\, \ldots,\, z(N),\, z(1),\, \ldots \quad .$$

Genau ein Element $z(i)$ des Zyklus korrespondiert hierbei mit genau einem Produkt j aus der Menge der in der Modelldefinition festgelegten Produkte. Es ergibt sich eine 1:1-Beziehung zwischen den Elementen des Zyklus und den definierten Produkten. Für die Bestimmung des Zyklus existieren

- die *statischen Varianten* AsCreated und AsDefined,
- die *Heuristiken* SmallestSetupCostsFirst, ShortestSetupTimesFirst, HighestSetupCostsFirst, LongestSetupTimesFirst und Random sowie
- eine *optimierende Variante* IsOptimized.

Bei den statischen Varianten wird der Zyklus im Vorfeld eines Laufes der simulationsbasierten Optimierung vom Modellersteller festgelegt. Im Falle der statischen Variante AsCreated [engl.; wie angelegt [dt.]] erfolgt die Festlegung des Zyklus durch die Reihenfolge der Definition der Modellelemente „Produkt“ innerhalb einer Modelldefinition (siehe Tab. 5.5). Das erste Element „Produkt“ in der Modelldefinition wird das erste innerhalb des Zyklus. Bei der Nutzung der statischen Variante AsDefined [engl.; wie vorgegeben [dt.]] muss vom Modellersteller zusätzlich angegeben werden, wie sich der feste Zyklus gestalten soll. Dabei ist eine beliebige Permutation des Zyklus Z erlaubt.

Die Heuristiken bestimmen den Zyklus Z im Vorfeld aus den verschiedenen Informationen einer Modelldefinition, wobei die Anordnung der Elemente „Produkt“ innerhalb einer Modelldefinition ohne Bedeutung bleibt. In dieser Arbeit haben die Heuristiken folgende Bedeutung:

SmallestSetupCostsFirst.
Der Zyklus Z ist aufsteigend nach den Umrüstkosten geordnet, welche bei allen Produktwechseln zum Produkt i von allen anderen Produkten $j \neq i$ entstehen.

ShortestSetupTimesFirst.
Ähnlich der Strategie SmallestSetupCostsFirst, nur, dass die Umrüstzeiten betrachtet werden.

HighestSetupCostsFirst.
Ähnlich der Strategie SmallestSetupCostsFirst, nur, dass der Zyklus Z absteigend nach den Umrüstkosten geordnet wird.

LongestSetupTimesFirst.
Ähnlich der Strategie HighestSetupCostsFirst, nur, dass die Umrüstzeiten betrachtet werden.

Random.
Der Zyklus Z wird zufällig bestimmt. Diese Heuristik kann als Vergleichsmöglichkeit mit den anderen Strategien dienen, d. h. die gemittelten Ergebnisse

dieser Heuristik sollten nicht besser als die gemittelten Ergebnisse einer anderen Heuristik sein, ansonsten ist diese andere Heuristik zu verwerfen[106].

Bei der optimierenden Variante der Zyklusbestimmung wird eine Permutationsentscheidungsvariable in das Simulationsmodell mit aufgenommen, welche dazu dient eine (sub)optimale Reihenfolge der Werte des Zyklus während eines Laufes der simulationsbasierten Optimierung zu bestimmen.

Anzumerken sei noch, dass während der Durchführung eines Simulationsexperimentes unter Verwendung einer zyklischen Reihenfolgestrategie zwangsläufig nicht alle Elemente des Zyklus durchlaufen werden müssen. Liegt für ein Produkt (aktuelles Element z_i im Zyklus) kein Los im Pool mit den Losen für die Fertigungsaufträge vor, so wird dieses Produkt im aktuellen Zyklus nicht gefertigt und das darauf folgende Element z_{i+1} des Zyklus mit der Angabe für das nächste zu fertigende Produkt betrachtet. Ist für dieses auch kein Los im Pool, wird wiederum das nachfolgende Element z_{i+2} des Zyklus betrachtet usw. Die Suche nach einem „gültigen“ Element, welches ein zu fertigendes Los im Pool besitzt, endet mit der Auffindung eines solchen Elementes im Zyklus oder mit dem Erreichen des Ausgangselementes z_i.

Dynamische Reihenfolgestrategien. Die dynamischen Reihenfolgestrategien richten sich nach verschiedenen internen Kenngrößen des Simulationsmodells. Diese entscheiden über die Reihenfolge der Fertigungsaufträge im Pool mit den Losen für die Fertigungsaufträge erst zum Zeitpunkt der Entnahme des nächsten Fertigungsauftrages aus dem Pool. Die im Rahmen dieser Arbeit angewandten dynamischen Reihenfolgestrategien haben folgende Bedeutung, wobei $auft_{next}(t)$ den nächsten zur Fertigung auszuwählenden Fertigungsauftrag zum Zeitpunkt t angibt. Nachfolgend sind nur die innerhalb dieser Arbeit relevanten Formeln für $auft_{next}(t)$ angegeben.

LWQ (Abkürzung für Longest Waiting Queue [engl.]).
Der Pool mit den Losen für die Fertigungsaufträge wird so absteigend geordnet, dass zuerst die Lose mit Fertigungsaufträgen für die Produkte betrachtet werden, welche die längste Kundenwarteschlange besitzen. Es gilt:

$$auft_{next}(t) = \max_{i=1,2,...,N} O_i(t) \quad .$$

SWQ (Abkürzung für Shortest Waiting Queue [engl.]).
Ähnlich der Strategie LWQ, nur, dass die Ordnung in aufsteigender Richtung erfolgt, wobei die Lose mit Fertigungsaufträgen zuerst eingeordnet werden, deren Produkte die kürzesten Kundenwarteschlange besitzen.

LWVQ (Abkürzung für Longest Value Waiting Queue [engl.]).
Ähnlich der Strategie LWQ, nur, dass anstatt der Länge der Kundenwarte-

[106] Für diese heuristische Annahme wird innerhalb dieser Arbeit kein theoretischer oder praktischer Beweis durchgeführt.

schlange die Höhe der wartenden Forderungen betrachtet wird. Es gilt:

$$auft_{next}(t) = \max_{i=1,2,...,N} \left(\sum_{j=1}^{O_i(t)} wsl_{ij}(t) \right) \quad .$$

SWVQ (Abkürzung für Shortest Waiting Value Queue [engl.]).
Ähnlich der Strategie SWQ, nur, dass wie bei der Strategie LWVQ die Höhe der wartenden Forderungen betrachtet wird.

Dynamic [engl.; dynamisch [dt.]].
Es werden die zu erwartenden Kosten für ein Los mit dem Fertigungsauftrag für ein bestimmtes Produkt betrachtet. Es gilt (Berechnung von $c_i(\Delta t)$ s. u.):

$$auft_{next}(t) = \max_{i=1,2,...,N} c_i(\Delta t) \quad .$$

Random [engl.; zufällig [dt.]][108].
Es wird zufällig ein Los aus dem Pool mit den Losen für die Fertigungsaufträge (pool) ausgewählt. Das gewählte Los entscheidet darüber, welcher Fertigungsauftrag als Nächster zu fertigen ist. Es gilt:

$$auft_{next}(t) = \text{pool}_{\text{rand}(1,\,\dim(pool))}(t) \quad .$$

Nachfolgend wird die dynamische Strategie noch etwas genauer betrachtet. Sie zielt auf die Auswahl des nächsten zur Fertigung auszuwählenden Fertigungsauftrages $auft_{next}(t)$ zum Zeitpunkt t mit den höchsten zu erwartenden Kosten $c_i(\Delta t)$ für das Produkt i während der Zeitdauer $\Delta t_i = (E(t_i^P) + st_{ji})$ ab, wobei $E(t_i^P)$ die erwartete Produktionszeit für das Produkt i und st_{ji} die Umrüstzeit vom aktuell gefertigten Produkt j zum Produkt i bezeichne und Folgendes gelte (Bezeichnungen vgl. Tab. 6.5):

$$c_i(\Delta t_i) = \widetilde{wc}_i(\Delta t_i) + \widetilde{hc}_i(\Delta t_i) + \widetilde{rc}_i(\Delta t_i) + \widetilde{pc}_i(\Delta t_i) + \widetilde{sc}_{ji}(\Delta t_i)$$

wobei

- $\widetilde{wc}_i(\Delta t_i)$ - erwartete Wartekosten zum Zeitpunkt t für das Produkt i,
- $\widetilde{hc}_i(\Delta t_i)$ - erwartete Holdingkosten zum Zeitpunkt t für das Produkt i,
- $\widetilde{rc}_i(\Delta t_i)$ - erwartete Abweiskosten zum Zeitpunkt t für das Produkt i,
- $\widetilde{pc}_i(\Delta t_i)$ - erwartete Produktionskosten zum Zeitpunkt t für das Produkt i,
- $\widetilde{sc}_i(\Delta t_i)$ - erwartete Umrüstkosten zum Zeitpunkt t für das Produkt i

[108] Die zufällige Reihenfolgestrategie hätte auch bei den statischen Reihenfolgestrategien angeführt werden können, wenn bereits die Einfügeposition innerhalb der Reihenfolge des Pools zufällig bestimmt worden wäre.

Anzumerken sei an dieser Stelle, dass bei der dynamischen Strategie einzelne zu erwartende Kostenbestandteile durch die Angabe eines Gewichtsfaktors auch entsprechend gewichtet auftreten oder entfallen könnten (vgl. Abschn. 2.5). Im Rahmen dieser Arbeit wird auf die Angabe von Gewichtungen ohne nähere Untersuchungen verzichtet.

6.3.3. Restriktionen (Nebenbedingungen)

Beschränkung des Lagers. Wie im Abschn. 6.3.1 bereits beschrieben, wird vorausgesetzt, dass ein globales Lager für alle Produkte existiert, welches auf P Raumeinheiten beschränkt ist. Ein gefertigtes Teil des Produktes i nimmt dabei einen Platz von p_i Raumeinheiten ein, so dass sich als Lagerrestriktion zu jedem beliebigen Zeitpunkt t ergibt:

$$\sum_{i=1}^{N} p_i \cdot I_i^+(t) \quad \leq \quad P, \quad t \geq 0 \quad . \tag{6.2}$$

Beschränkung der Warteschlangen. Die zweite Restriktion behandelt die Längen der einzelnen Warteschlangen für nicht befriedigte Kundenbedarfe. Dazu kann jede Warteschlange von Kunden des Produktes i (Kundenwarteschlange) eine nach oben begrenzte Anzahl von Kunden b_i aufnehmen, so dass gelte:

$$O_i(t) \quad \leq \quad b_i, \quad \text{für } i = 1, 2, \ldots, N, \quad t \geq 0 \quad , \tag{6.3}$$

wobei

$O_i(t)$ - aktuelle Länge der Kundenwarteschlange i zum Zeitpunkt t.

6.3.4. Kostenbetrachtung und Optimierungskriterien

Als Optimierungsproblem für die Untersuchungen von Modellen des Modelltyps 1 und 2 (siehe Abschn. 6.4) ergibt sich:

$$\min_{\pi \in \Pi} C(\pi) \tag{6.4}$$

mit

$$\begin{aligned} C(\pi) &= \sum_{i=1}^{N}\Bigg(hc_i \cdot HC_i(\pi) + wc_i \cdot WC_i(\pi) + rc_i \cdot RC_i(\pi) + pc_i \cdot PC_i(\pi) \\ &\quad + \sum_{j=1}^{N} sc_{ij} \cdot SC_{ij}(\pi)\Bigg) \\ &= \sum_{i=1}^{N}\Big(hc_i \cdot HC_i(\pi)\Big) + \sum_{i=1}^{N}\Big(wc_i \cdot WC_i(\pi)\Big) + \sum_{i=1}^{N}\Big(rc_i \cdot RC_i(\pi)\Big) \\ &\quad + \sum_{i=1}^{N}\Big(pc_i \cdot PC_i(\pi)\Big) + \sum_{i=1}^{N}\Bigg(\sum_{j=1}^{N} sc_{ij} \cdot SC_{ij}(\pi)\Bigg) \\ &= HC(\pi) + WC(\pi) + RC(\pi) + PC(\pi) + SC(\pi) \quad , \end{aligned}$$

wobei

π	-	aktuelle Gesamtstrategie (aktuell gewählte Reihenfolge-, Zuordnungs- und Losgrößenstrategie),
Π	-	Menge aller Strategien,
$HC_i(\pi)$	-	mittlere Anzahl der gelagerten Teile für Produkt i pro Zeiteinheit,
$WC_i(\pi)$	-	mittlere Anzahl wartender Forderungen für Produkt i pro Zeiteinheit,
$RC_i(\pi)$	-	mittlere Anzahl abgewiesener Kunden für Produkt i pro Zeiteinheit,
$PC_i(\pi)$	-	mittlere Anzahl produzierter Teile für Produkt i pro Zeiteinheit sowie
$SC_{ij}(\pi)$	-	mittlere Anzahl Umrüstvorgänge für Produktwechsel von Produkt i zu Produkt j pro Zeiteinheit
$HC(\pi), WC(\pi), RC(\pi), PC(\pi), SC(\pi)$	-	siehe Tab. 6.5
$hc_i, wc_i, rc_i, pc_i, sc_{ij}$	-	siehe Tab. 6.5

unter Einhaltung der Nebenbedingungen 6.2 und 6.3.

Beim Modelltyp 1 (siehe Abschn. 6.4.1) ergibt sich die Gesamtstrategie π nur aus der globalen Reihenfolgestrategie RS sowie den individuellen Losgrößen- und Freigabestrategien LFS_i für jedes Produkt i, wodurch:

$$\pi = (RS, \overrightarrow{LFS})$$

mit

$$\overrightarrow{LFS} \;=\; (LFS_1,\, LFS_2,\, \ldots,\, LFS_N) \quad .$$

Bei Modelltyp 2 (siehe Abschn. 6.4.2) kommt zusätzlich noch die Zuordnungsstrategie ZS hinzu, so dass sich Folgendes für die Gesamtstrategie π ergibt:

$$\pi \;=\; (RS, \overrightarrow{LFS}, ZS) \quad .$$

Über diese Arbeit hinaus sei noch einmal angemerkt (vgl. Reihenfolgeoptimierung in Abschn. 6.3.2), dass bei ähnlichen Problemstellungen, wie bspw. dem scheduling [engl.][109], auch andere Nebenbedingungen und Kostenbestandteile, wie z. B. Fälligkeiten, Verspätungen, usw. in die zu optimierende Kostenfunktion mit einfließen können. Des Weiteren werden in der Fachliteratur teilweise auch zur hiesigen Kostenfunktion äquivalente Ziele, wie Servicegrad, Kapazitätsauslastung, Bestelldauern, usw., betrachtet (siehe [FK96], [Qua04] und [WG06]). Diese seien jedoch nur der Vollständigkeit halber erwähnt. Sie werden innerhalb der vorliegenden Arbeit nicht näher untersucht und spielen daher keine direkte Rolle.

6.3.5. Zusammenfassung der genutzten Größen

In Tab. 6.5 ist noch einmal alle genutzten Größen der Modelltypen 1 bis 3 als allgemeiner Überblick zusammengefasst. Dabei nimmt die Kategorie Bezug auf die rechnerinterne Modellabbildung (siehe Abschn. 2.4 und Tab. 5.5).

In den Tabellen 6.6 und 6.7 sind zusätzlich zu den in Tab. 6.5 angegebenen Größen noch die für die Modelltypen 2 und 3 zusätzlich relevanten Größen angegeben.

[109] Die Ähnlichkeit zum scheduling [engl.] entsteht dadurch, dass bei diesem eine Reihenfolgeoptimierung erfolgt.

Tab. 6.5

Modellkomponente	Symbol	Kategorie	Einheit	Wertebereich
Produkte	N	$IN_{CalcSim}$	-	$N \in \mathbb{N}^*$
Lagergröße (gesamt)	P	$IN_{CalcSim}$	RE	$P \in \mathbb{N}$
Reihenfolgestrategie	RS	$IN_{CalcSim}$	-	{FCFS, Cyclic, LWQ, LWVQ, Dynamic, Random}
Losgrößen- und Freigabestrategie	LFS_i	$IN_{CalcSim}$	-	MTO
				MTS: (s, nQ), (s, S)
Kundenankunftsrate	λ_i	$IN_{CalcSim}$	1/ZE	$\lambda_i > 0$
Kundenbedarfsmenge	d_i	$IN_{CalcSim}$	TE	$d_i \sim F(x)$
Produktionsrate	μ_i	$IN_{CalcSim}$	TE/ZE	$\mu_i \sim F_i(x)$
Lagerplatz	p_i	$IN_{CalcSim}$	RE/TE	$p_i \in \mathbb{R}_+$
maximale Warteschlangenlänge	b_i	$IN_{CalcSim}$	Kunden	$b_i \in \mathbb{N}$
Umrüstzeiten	st_{ij}	$IN_{CalcSim}$	ZE	$st_{ij} \in \mathbb{R}_+$
Faktor für Umrüstkosten	sc_{ij}	$IN_{CalcSim}$	-	$sc_{ij} \in \mathbb{R}_+$
Faktor für Produktionskosten	pc_i	$IN_{CalcSim}$	-	$pc_i \in \mathbb{R}_+$
Faktor für Lagerkosten	hc_i	$IN_{CalcSim}$	-	$hc_i \in \mathbb{R}_+$
Faktor für Wartekosten	wc_i	$IN_{CalcSim}$	-	$wc_i \in \mathbb{R}_+$
Faktor für Abweiskosten	rc_i	$IN_{CalcSim}$	-	$rc_i \in \mathbb{R}_+$
Produktionszeitpunkt	s_i	IN_{Opt}	TE	$s_i \in \mathbb{N}$
Produktionsmenge	Q_i	IN_{Opt}	TE	$Q_i \in \mathbb{N}^*$
Produktionsobergrenze	S_i	IN_{Opt}	TE	$S_i \in \mathbb{N}^*$
Umrüstkosten (gesamt)	$SC(\pi)$	OUT_{Ass}	GE/ZE	$SC \in \mathbb{R}_+$
Produktionskosten (gesamt)	$PC(\pi)$	OUT_{Ass}	GE/ZE	$PC \in \mathbb{R}_+$
Lagerkosten (gesamt)	$HC(\pi)$	OUT_{Ass}	GE/ZE	$HC \in \mathbb{R}_+$
Wartekosten (gesamt)	$WC(\pi)$	OUT_{Ass}	GE/ZE	$WC \in \mathbb{R}_+$
Abweiskosten (gesamt)	$RC(\pi)$	OUT_{Ass}	GE/ZE	$RC \in \mathbb{R}_+$
Verlust	$C(\pi)$	OUT_{NoAss}	GE/ZE	$C \in \mathbb{R}_+$
Abgewiesene Kunden (gesamt)	nrc	OUT_{NoAss}	Kunden	$nrc \in \mathbb{N}$
Kunden (gesamt)	nc	OUT_{NoAss}	Kunden	$nc \in \mathbb{N}$

Legende:

ZE - Zeiteinheit(en)
TE - Teil(e)
RE - Raumeinheit(en)
GE - Geldeinheit(en)
FE - (Anzahl der) Fertigungseinheit(en)
i, j - Indexe der Produkte
π - gewählte Reihenfolge-, (Zuordnungs-) und Losgrößenstrategie

Tab. 6.5.: *Überblick über die Modellparameter, Modellvariablen und Entscheidungsvariablen der untersuchten Modelle*

Modellkomponente	Symbol	Typ	Einheit	Wertebereich
Fertigungseinheiten	M	$IN_{CalcSim}$	-	$M \in \mathbb{N}^*$
Produktionsrate	μ_{ik}	$IN_{CalcSim}$	TE/ZE	$\mu_{i,k} \sim F_i(x)$
Umrüstzeiten	st_{ijk}	$IN_{CalcSim}$	ZE	$st_{ijk} \in \mathbb{R}_+$
Umrüstkosten	sc_{ijk}	$IN_{CalcSim}$	GE	$sc_{ijk} \in \mathbb{R}_+$
Produktionskosten	pc_{ik}	$IN_{CalcSim}$	GE	$pc_{ik} \in \mathbb{R}_+, \forall k$

Legende (siehe auch Legende von Tab. 6.5):

k - Index der Fertigungseinheiten

Tab. 6.6

Tab. 6.6.: *Überblick über die veränderten Modellparameter der untersuchten Modelle beim Modelltyp 2*

Modellkomponente	Symbol	Typ	Einheit	Wertebereich
Losgrößen- und Freigabestrategie	LFS_i	$IN_{CalcSim}$	-	MTO
				MTS: (s, nQ)
Zulieferwartezeiten	sct_i	$IN_{CalcSim}$	ZE	$sct_i \sim F_i(x)$
Zulieferkosten (gesamt)	$SCC(\pi)$	OUT_{Ass}	GE/ZE	$SCC \in \mathbb{R}_+$
Zulieferkosten	scc_i	$IN_{CalcSim}$	GE	$scc_i \in \mathbb{R}_+$

Legende vgl. Tab. 6.5

Tab. 6.7

Tab. 6.7.: *Überblick über die zusätzlichen Modellparameter, Modellvariablen und Entscheidungsvariablen der untersuchten Modelle beim Modelltyp 3*

6.4. Modelltypen

Bevor auf die einzelnen Modelltypen im Detail eingegangen wird, soll an dieser Stelle zunächst ein grober Überblick über die einzelnen implementierten Simulatoren gegeben werden. In Abb. 6.2 sind die, für die untersuchten Modelle zu Produktions- und Lagerhaltungssystemen, relevanten, im `CAOS` implementierten Simulatoren abgebildet, wobei der Bezug zur Berechnung[110] und zur ereignisorientierten Simulation sichtbar wird.

[110] Bereits im Abschn. 5.3.6 (Unterpunkt „Der Namensraum `CAO.Calculation`“) wurde ersichtlich, dass die Simulation von der Charakteristik der Ein- und Ausgabewerte nur eine spezielle Art der Berechnung ist. Sie kann daher auch als „Black-Box-Berechnung“ bezeichnet werden.

Abb. 6.2

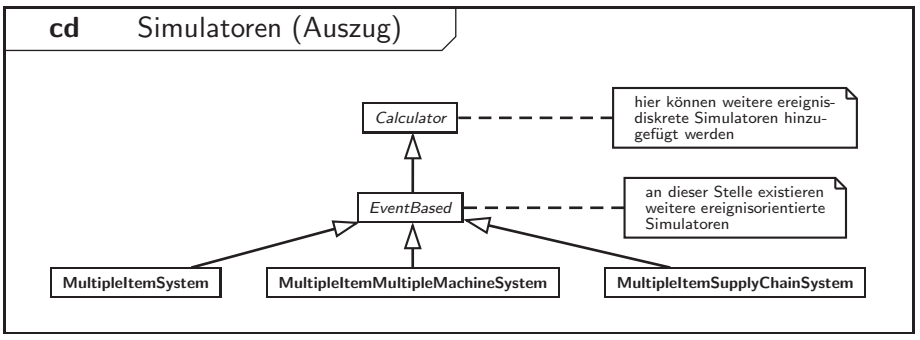


Abb. 6.2.: *Auszug der im* `CAOS` *vorhandenen CSLSP-Simulatoren (UML-Klassendiagramm)*

Simulatoren. Wie bereits im Abschn. 3.3 erwähnt, wird bei den Untersuchungen die ereignis-diskrete Simulation eingesetzt. Die Zeitfortschreibung erfolgt somit anhand konkreter Ereignisse, welche innerhalb der Ereignisliste chronologisch geordnet sind.

Auf Grund der Tatsache, dass der Simulator MultipleItemSystem den einfachsten internen Ablauf besitzt, soll dieser im Folgenden etwas genauer betrachtet werden.

Ereignisse und Ereignisroutinen. Zunächst sei in Abb. 6.3 aufgezeigt, welche Ereignisse in den Simulatoren der drei Modelltypen vorkommen.

Abb. 6.3

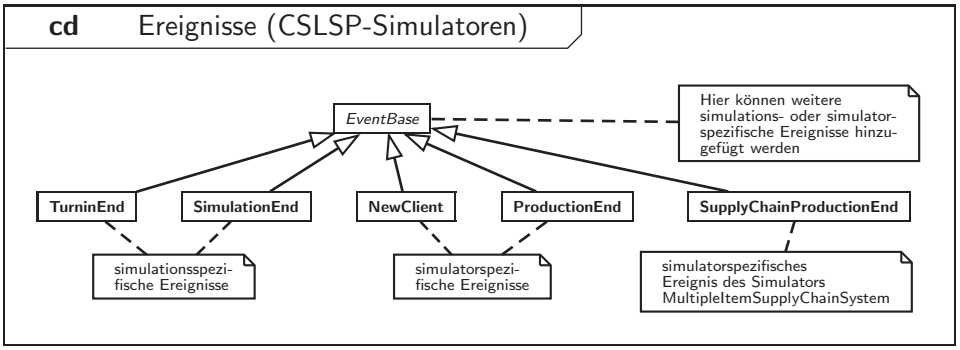


Abb. 6.3.: *Ereignisse zu den CSLSP-Simulatoren im* `CAOS` *(UML-Klassendiagramm)*

Um insgesamt ein besseres Verständnis über den internen Simulationsablauf der Simulatoren zu erhalten, sei in Abb. 6.4 bis Abb. 6.7 der Ablauf der einzelnen Simulationsereignisse anhand ihrer Ereignisroutinen dargestellt. Sie geben die Detailroutinen der Aktivität „`Zugehörige Simulationsereignisroutine ausführen`“ von Abb. 3.2 an.

Die einzelnen Ereignisse aus Abb. 6.3 haben hierbei folgende Bedeutungen:

- Simulatorspezifische Ereignisse:

  Ereignis „NewClient“.
  Das Ereignis „`NewClient`“ beschreibt die Ankunft eines neuen Kunden, bei dem eine Forderung in Höhe von d_i Teilen des Produktes i besteht. Gleichzeitig generiert die zugehörige Ereignisroutine das nächste Ereignis vom Typ „`NewClient`“ für das Produkt i, da es sich um einen erneuerbaren Prozess [dt.; renewal process [engl.]] handelt. Die Abb. 6.4 spiegelt den Ablauf bei der Ankunft eines neuen Kunden wider.

  Ereignis „ProductionEnd“.
  Für die Angabe, dass die Fertigung eines Teils abgeschlossen ist, wird das Ereignis „`ProductionEnd`“ verwendet. Es enthält die Information, für welches Produkt i ein Teil gefertigt und wann mit der Fertigung dieses Teils begonnen wurde. Wie sich der Ablauf der zugehörigen Ereignisroutine verhält, ist in Abb. 6.5 zu sehen.

- Simulationsspezifische Ereignisse:

  Ereignis „TurninEnd“.
  Legt bei der statischen Strategie zur Bestimmung der Einschwingdauer (siehe Abschn. 3.5) den Zeitpunkt des Endes der transienten Phase fest. In Abb. 6.6 ist die dazugehörige Ereignisroutine dargestellt.

  Ereignis „SimulationEnd“.
  Legt bei der statischen Strategie zur Bestimmung des Endes der Simulation (siehe ebenso Abschn. 3.5) den Zeitpunkt des Endes der stationären Phase fest. Die Abb. 6.7 nimmt Bezug auf die zugehörige Ereignisroutine.

Abb. 6.4

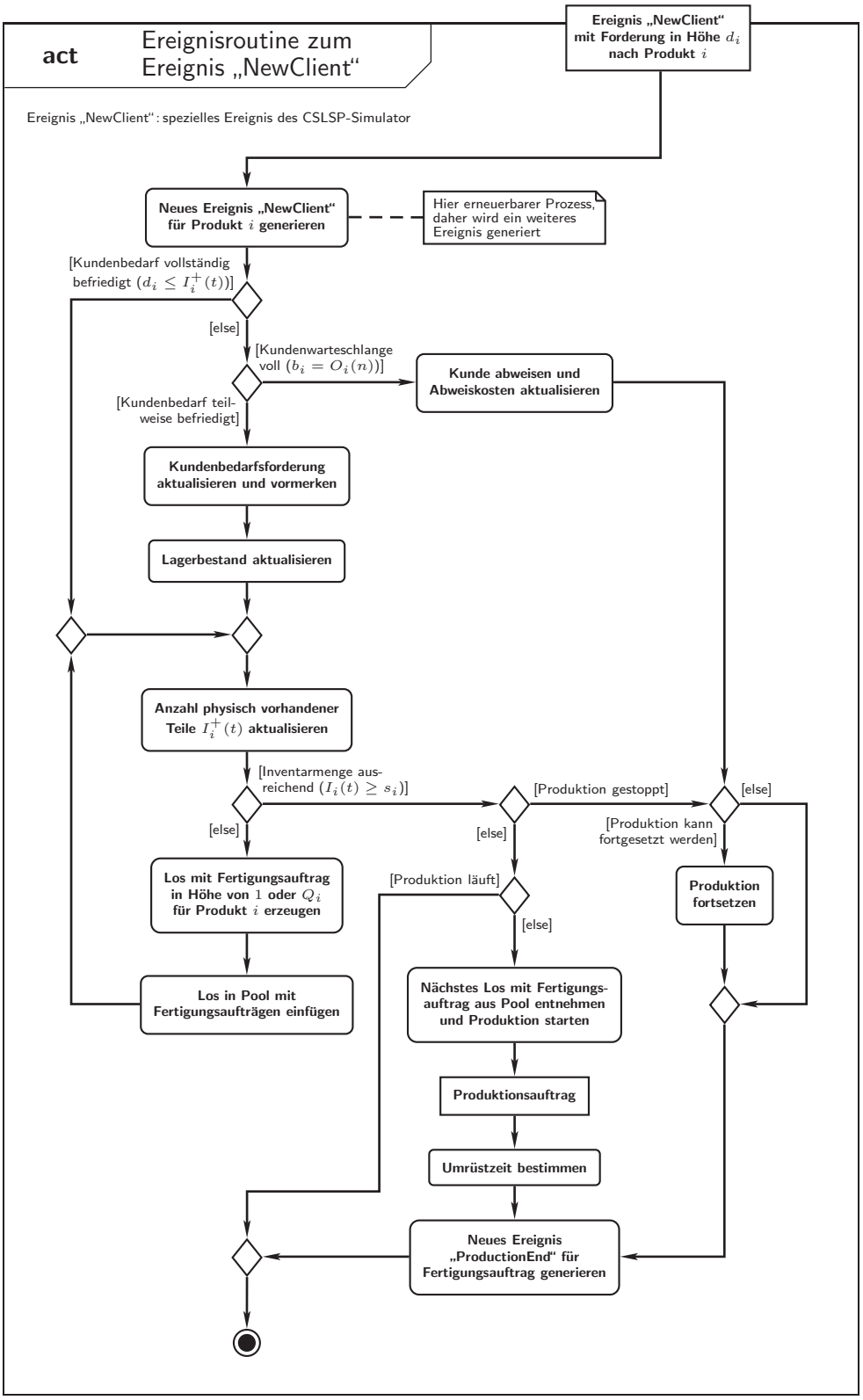


Abb. 6.4.: *Ereignisroutine des Ereignisses „NewClient“ beim CSLSP-Simulator* MultipleItemSystem *(UML-Aktivitätsdiagramm)*

Abb. 6.5

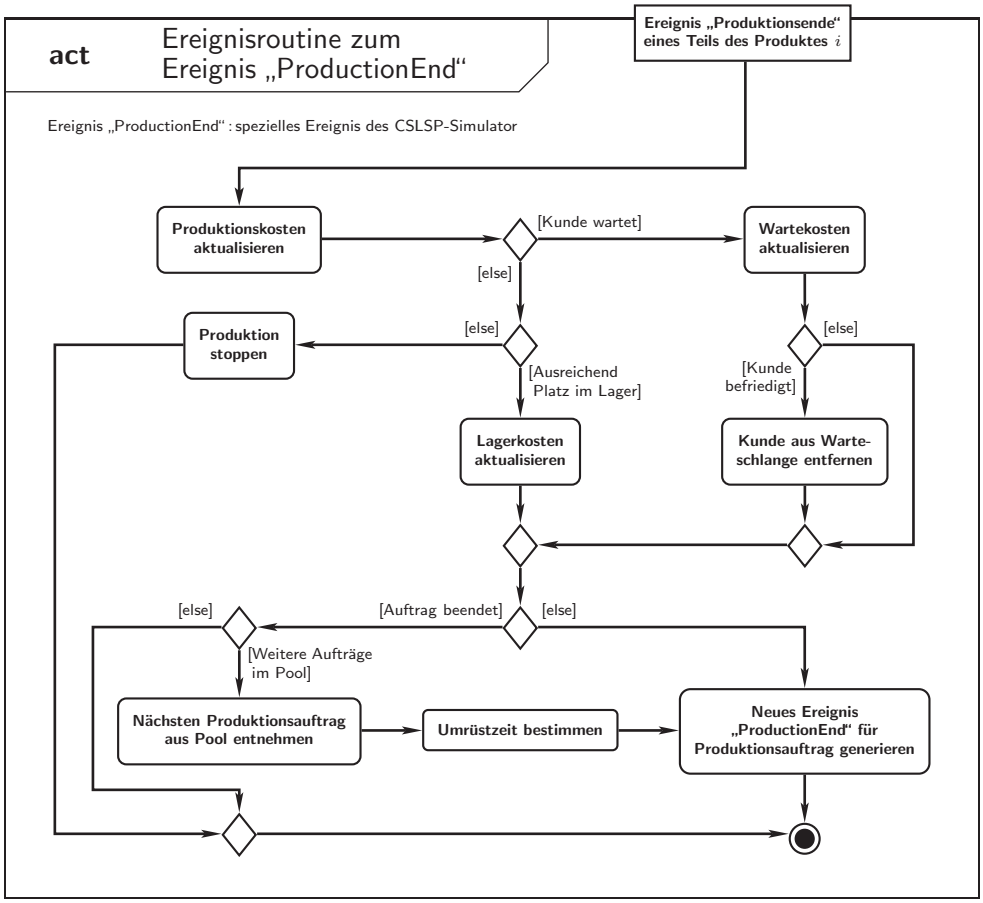


Abb. 6.5.: *Ereignisroutine des Ereignisses „ProductionEnd“ beim CSLSP-Simulator* MultipleItemSystem *(UML-Aktivitätsdiagramm)*

Abb. 6.6

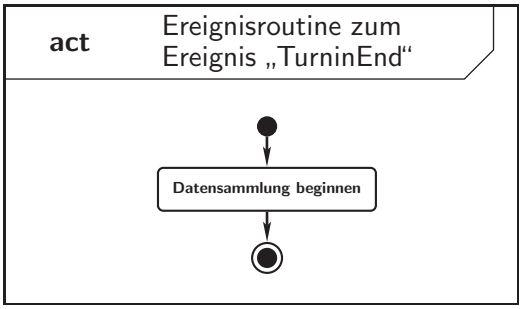


Abb. 6.6.: *Ereignisroutine des Ereignisses „TurninEnd“ beim CSLSP-Simulator* MultipleItemSystem *(UML-Aktivitätsdiagramm)*

Abb. 6.7

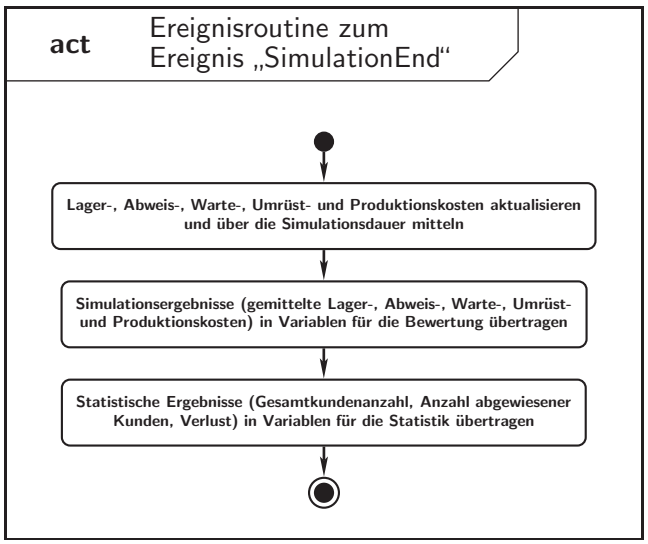


Abb. 6.7.: *Ereignisroutine des Ereignisses „SimulationEnd“ beim CSLSP-Simulator* MultipleItemSystem *(UML-Aktivitätsdiagramm)*

In Abb. 6.8 ist unter Bezugnahme auf Abb. 6.5 dargestellt, wie beim Simulator MultipleItemSystem des Modelltyps 1 die Detailroutine der Aktivität „`Simulator zurücksetzen`“ (vgl. Abb. 3.2) programmtechnisch umgesetzt ist.

Die Abb. 6.9 gibt Auskunft darüber, welche Simulationsereignisse im Vorfeld der Simulation beim Simulator MultipleItemSystem des Modelltyps 1 generiert und zur Ereignisliste hinzugefügt werden müssen (vgl. wiederum Abb. 3.2, Detailansicht der Aktivität „`Initiale Simulationsereignisse generieren`“).

Damit sei die Betrachtung der Ereignisroutinen abgeschlossen und es folgt die Betrachtung der verschiedenen untersuchten Modelltypen. Dazu wurden bereits in den vorangehenden Abschnitten die verschiedenen Modellvariablen und -parameter mit ihren individuellen Charakteristiken tiefgründig erläutert. In den nachfolgenden Abschnitten soll sich daher nur noch auf wesentliche Besonderheiten der einzelnen Modelltypen konzentriert werden.

Abb. 6.8

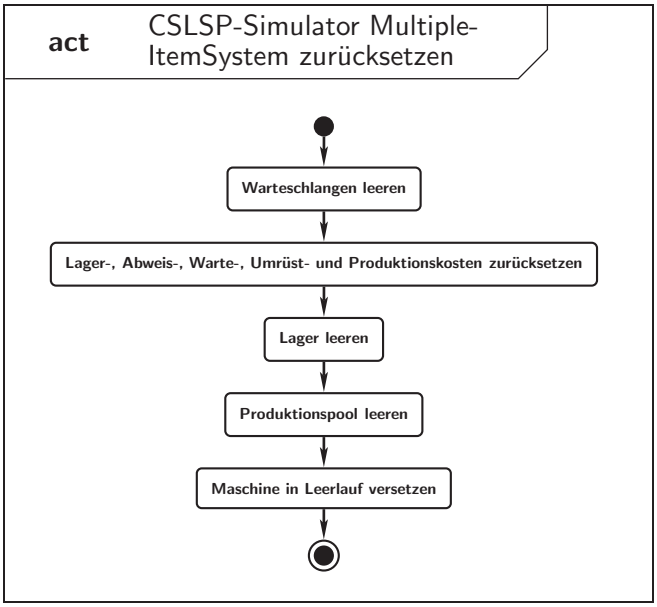


Abb. 6.8.: *Algorithmus zum Zurücksetzen des CSLSP-Simulators* MultipleItemSystem *(UML-Aktivitätsdiagramm)*

Abb. 6.9

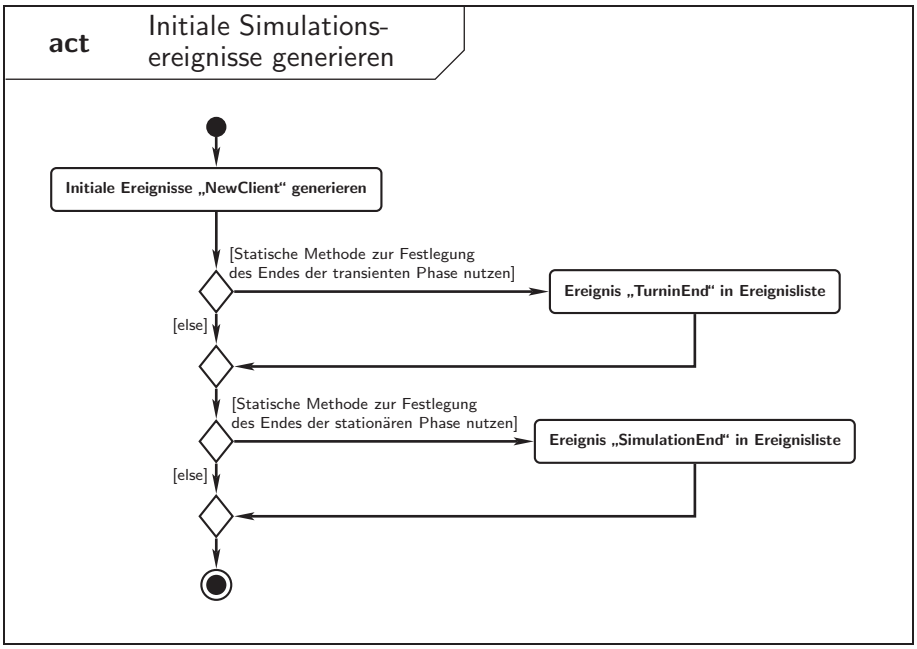


Abb. 6.9.: *Algorithmus zum Generieren der initialen Simulationsereignisse beim CSLSP-Simulator* MultipleItemSystem *(UML-Aktivitätsdiagramm)*

6.4.1. Modelltyp 1: Eine Fertigungseinheit und N Produkte

Visuelle Modellbeschreibung. In Abb. 6.10 ist unter Nutzung der neu entwickelten visuellen Beschreibungssprache des Material- und Informationsflusses bei ökonomischen Simulationssystemen EcoSyL (aktuelle Version 0.1) der Ablauf beim Modelltyp 1 widergespiegelt. Die Sprachelemente von EcoSyL 0.1 werden im Anhang B ausführlich betrachtet. Grundlegend kann jedoch gesagt werden, dass sich die visuelle Modellbeschreibungssprache EcoSyL 0.1 an die Aktivitätsdiagramme der UML 2.0 anlehnt und diese um weitere benötigte Elemente erweitert.

Abb. 6.10

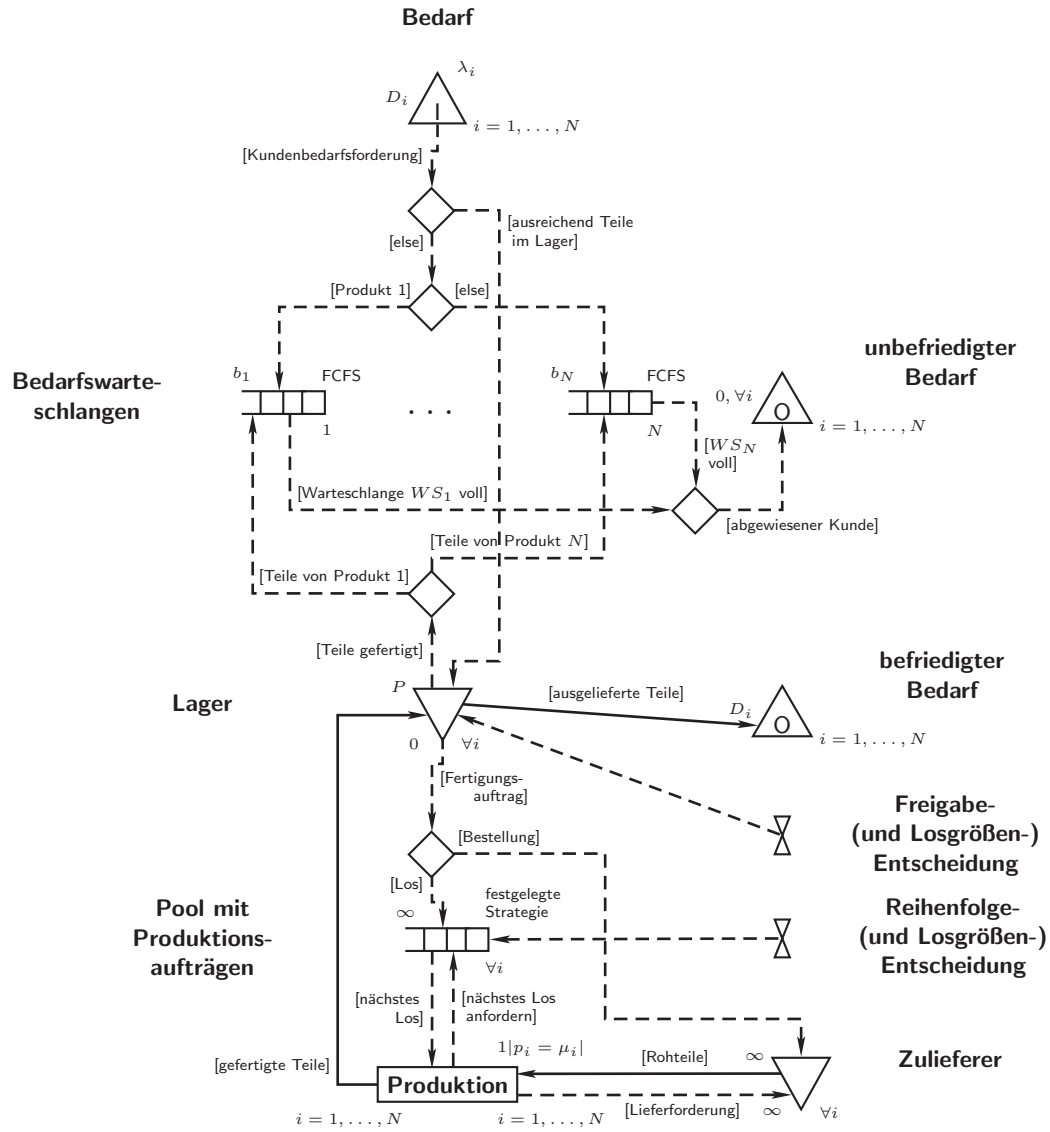


Abb. 6.10.: *Darstellung des untersuchten Produktions- und Lagerhaltungssystems beim Modelltyp 1 (EcoSyL-Diagramm)*

Die Abb. 6.10 zeigt, wie sich die Information über eine entstehende Forderung eines ankommenden Kunden zunächst ans gemeinsame Lager richtet (siehe auch Abb. 6.4). Hat dieses nicht ausreichend Teile des geforderten Produktes zur Verfügung,

wird, wenn noch Teile des Produktes im Lager vorhanden sind, zumindest eine Teilforderung erfüllt und die restliche Forderung in der jeweiligen, produktspezifischen Bedarfswarteschlange vorgemerkt. Voraussetzung für die Abfrage nach der Erfüllbarkeit einer Teilforderung durch das Lager ist allerdings die Existenz freier Plätze in der Bedarfswarteschlange. Trivialer Weise gelangt eine vom Lager nicht erfüllbare Forderung direkt in die Bedarfswarteschlange, wenn noch freie Plätze vorhanden sind. Das Lager prüft seinerseits, sobald eine Forderung vorgemerkt wurde, ob die Bilanz zwischen der Inventarmenge und der unteren Bestellgrenze noch positiv ist (vgl. Gleichung 6.1). Sinkt die Bilanz auf null oder darunter ab, werden entsprechend der gewählten Losgrößenstrategie ein oder mehrere Lose mit Fertigungsaufträgen in den Pool eingefügt.

Die Fertigungseinheit beginnt mit der Produktion der Teile, sobald sich ein oder mehrere Lose mit Fertigungsaufträgen im Pool befinden. Anhand der festgelegten Reihenfolgestrategie wird entschieden, welches Produkt bei Bedarf zu erzeugen ist (siehe Reihenfolgeoptimierung in Abschn. 6.3.2). Die benötigten Rohteile gelangen dabei unmittelbar, d. h. ohne zeitlichen Verzug, von den Zulieferern zur Fertigungseinheit. Im Abschluss werden die gefertigten Teile zur Befriedigung des wartenden Bedarfs verwendet. Können eine oder mehrere Bedarfsforderungen durch die gefertigten Teile befriedigt werden, erfolgt die Entfernung der Bedarfsforderungen aus der entsprechenden Warteschlange. Existieren in der betreffenden Bedarfswarteschlange keine Forderungen, gelangen die gefertigten Teile direkt ins Lager. Unter Berücksichtigung der verschiedenen Kostenfaktoren werden die einzelnen Kostenbestandteile an den betreffenden Stellen berechnet.

Untersuchungsziele. In der Literatur (siehe [KS85], [FK96], [MRW00] und [SWd04]) wird oftmals davon ausgegangen, dass in der Praxis eine zyklische Reihenfolgestrategie optimal ist (siehe statische Reihenfolgestrategie Cyclic im Abschn. 6.3.2). Eine solche Annahme sollte jedoch nicht ohne nähere Untersuchungen anderer Reihenfolgestrategien getroffen werden. Daher ist ein Ziel festzustellen, ob eine zyklische Reihenfolgestrategie tatsächlich alle anderen Reihenfolgestrategien dominiert.

Sollte dies nicht der Fall sein, ist ein weiteres Ziel der Untersuchungen festzustellen, ob eine der möglichen anderen Reihenfolgestrategien dominant gegenüber allen anderen ist.

Als letztes Untersuchungsziel wird versucht herauszufinden, in wie weit es sich auswirkt, detaillierte Informationen über die Produkte zu haben. Dazu wird zunächst nur ein Produkt betrachtet, welches eine unscharfe Informationswiedergabe gemittelt über alle Produkte widerspiegeln soll (generalisiertes Modell). Diese Informationen werden über ein Modell mit zwei Produkten bis hin zu einem Modell mit fünf Produkten sukzessive verfeinert (spezialisierte Modelle). Die entsprechenden Modelle und die zugehörigen Untersuchungsergebnisse sind im Abschn. 6.5.2 angeführt.

6.4.2. Modelltyp 2: M Fertigungseinheiten und N Produkte

Visuelle Modellbeschreibung. Die visuelle Modellbeschreibung mittels EcoSyL gestaltet sich ähnlich der im Falle des Modelltyps 1 (vgl. Abb. 6.10) mit dem Unterschied, dass anstatt einer Fertigungseinheit jetzt M Fertigungseinheiten parallel produzieren können. Ein Fertigungsauftrag ist dabei jedoch nicht auf mehrere Fertigungseinheiten verteilbar nachdem mittels der Zuordnungsentscheidung bestimmt wurde, welche Fertigungseinheit die Produktion des Fertigungsauftrages übernimmt.

Zuordnungsentscheidung. In der vorliegenden Arbeit wird eine einfache Heuristik angewandt, welche die Zuordnung eines Fertigungsauftrages zu genau einer Fertigungseinheit regelt. Diese Heuristik sieht lediglich vor, dass die nächste freie Fertigungseinheit unverzüglich mit der Produktion des nächsten anstehenden Fertigungsauftrages beginnt. Dieser Fertigungsauftrag ergibt sich anhand des entsprechenden Loses, welches mittels der festgelegten Reihenfolgestrategie aus dem Pool mit den Losen für die Fertigungsaufträge entnommen wurde.

Anstelle dieser einfachen Heuristik könnten auch weiter reichende Heuristiken eingesetzt werden. Diese könnten dann das zukünftige Modellverhalten oder das vergangene Modellverhalten berücksichtigen, um evtl. zu entscheiden, dass eine Fertigungseinheit nicht unmittelbar nach der Fertigung des letzten Teiles für ein Los mit der Fertigung der Teile für das nächste Los beginnt.

Hierzu kommt die im Rahmen dieser Arbeit nicht angewandte Möglichkeit der Zuordnung eines Loses mit einem Fertigungsauftrag zu mehreren Fertigungseinheiten. Diese Möglichkeit ist prinzipiell stets dann gegeben, wenn die Zerteilung eines Loses erlaubt ist. Eine solche Vorgehensweise wirft allerdings die Frage auf, an welcher Stelle das Los teilbar ist. In der Literatur (siehe z. B. [Tei98] oder [Köb99]) sind dazu verschiedene Parameter für die Teilbarkeit eines Loses angegeben (vgl. Parameter β_1 bei der Klassifizierung von Scheduling-Problemen im Abschn. B.2). Es kann im Wesentlichen davon ausgegangen werden, dass ein Los *teilbar* ($\beta_2 = 0$) oder *unteilbar* ($\beta_2 \in \{pmtn, div\}$) ist. Im Falle der Teilbarkeit ist noch zu klären, an welchen *Stellen* (mengenabhängig) bzw. zu welchen *Zeitpunkten* (zeitabhängig)[111] ein Los geteilt werden kann. Außerdem kann für die Anzahl der Teilungen im Falle, dass keine beliebige Teilbarkeit erlaubt ist, noch eine obere und untere Schranke angegeben sein. Bereits aus diesen Ausführungen wird erkennbar, dass die Teilbarkeit eines Loses mitunter sehr komplex ist. Die Beachtung von Teilbarkeiten und deren mögliche Auswirkungen auf die Optimierungsergebnisse soll u. a. auf Grund der steigenden Komplexität der sich ergebenden Zuordnungsmöglichkeiten in dieser Arbeit nicht näher betrachtet werden.

Untersuchungsziele. Das Ziel der Untersuchungen bei Modellen dieses Modelltyps ist es im Wesentlichen, Aussagen darüber zu treffen, in wie weit sich detailliertere

[111] Ist eine zeitabhängige Teilbarkeit angegeben, sind u. U. noch die Auswirkungen unterschiedlicher Fertigungsgeschwindigkeiten der Fertigungseinheiten für verschiedene Produkte zu berücksichtigen.

Abb. 6.11

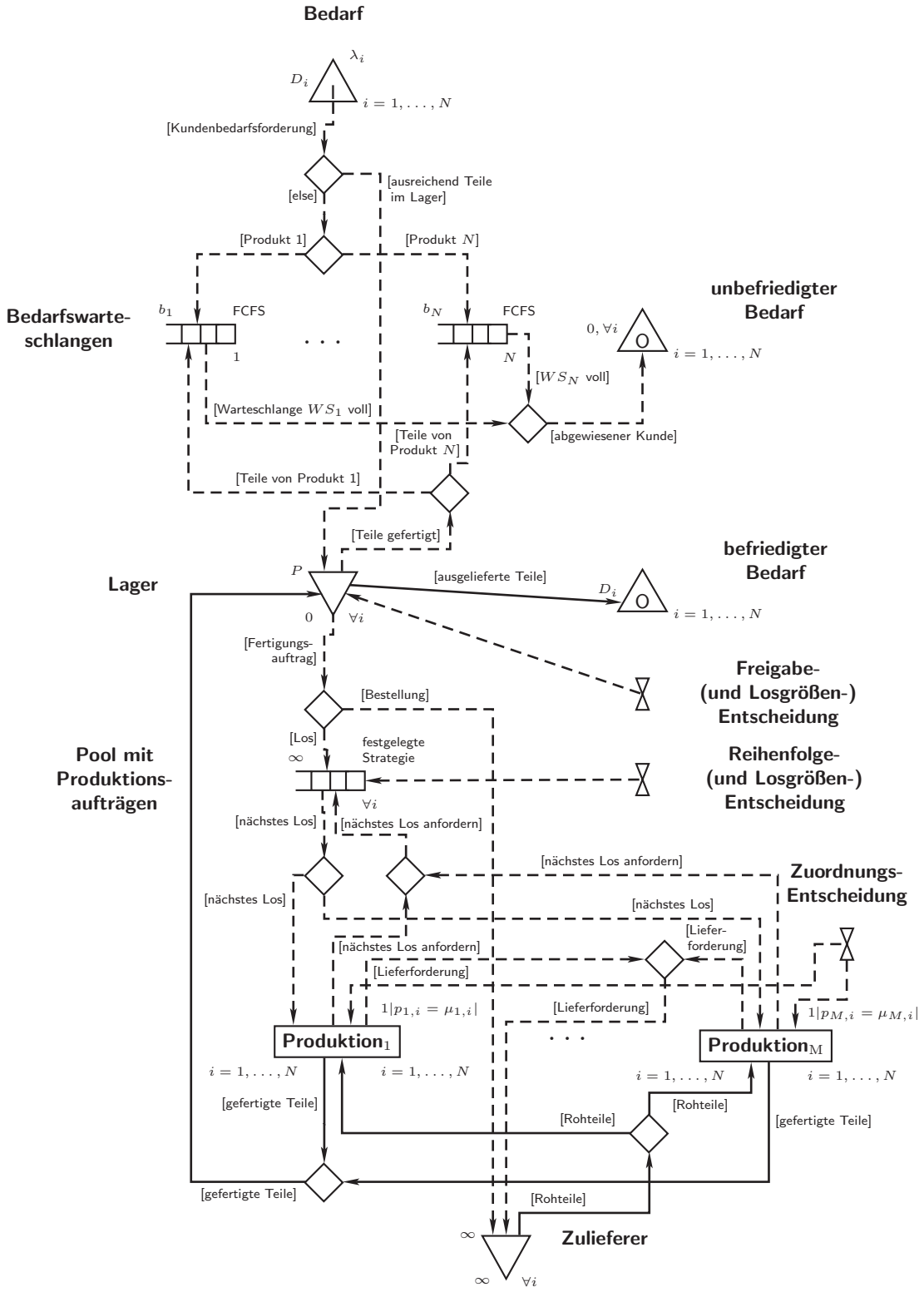


Abb. 6.11.: *Darstellung des untersuchten Produktions- und Lagerhaltungssystems beim Modelltyp 2 (EcoSyL-Diagramm)*

Informationen des Fertigungsprozesses auf die Untersuchungsergebnisse auswirken. Dazu werden ähnliche Werte für die Modellparameter wie beim ersten Modelltyp verwendet.

Beim *ersten Untersuchungsmodell* (Fünf-Produkt-Modell mit zwei Fertigungseinheiten ($scst^{voll}$)) werden die Werte für die Modellparameter vom Untersuchungsmodell zum Modelltyp 1 direkt übernommen. Es kommt lediglich eine weitere Fertigungseinheit hinzu, um festzustellen, welche Entlastung bei einem zuvor stark ausgelasteten Modell zu erzielen ist.

Beim *zweiten Untersuchungsmodell* kommen jeweils zwei bzw. vier Fertigungseinheiten zum Einsatz, d. h. $M = 2$ bzw. $M = 4$ (Fünf-Produkt-Modell mit zwei Fertigungseinheiten ($scst^{voll}$ und $scst^{halb}$) sowie Fünf-Produkt-Modell mit vier Fertigungseinheiten ($scst^{viertel}$)). Diese Fertigungseinheiten besitzen im Gegensatz zu einer einzelnen Fertigungseinheit, welche bei den Untersuchungen des ersten Modelltyps eingesetzt wird, eine reduzierte Produktionsrate $\mu^2_{i,k} = \frac{\mu^1_i}{M}$, $\forall k$ sowie entsprechend reduzierte Umrüstkosten und -zeiten (siehe Tab. 6.14 und Tab. 6.15). Die Abweis-, Warte- und Produktionskosten werden im Vergleich zu den Untersuchungen beim ersten Modelltyp unverändert gelassen. Ziel des Untersuchungsmodells ist es, die Auswirkungen weiterer Fertigungseinheiten zu untersuchen, welche bspw. zur Verringerung der Ausfallwahrscheinlichkeit oder der Verminderung des personellen Aufwandes mit einer geringeren Produktionsrate dienen.

Abzusehen ist bereits an dieser Stelle, dass die Ergebnisse der verschiedenen Untersuchungsmodelle des Modelltyps 2 durch die unterschiedlichen Modellparameter nicht direkt miteinander vergleichbar sein werden. Zudem wird es auch nur bedingt möglich sein, Vergleiche zwischen den Untersuchungsmodellen des Modelltyps 2 und dem Untersuchungsmodell des Modelltyps 1 durchzuführen. Dies liegt zum einen in der getroffenen Zuordnungsentscheidung und zum anderen in der Existenz mehrerer Fertigungseinheiten begründet. Die Zuordnungsentscheidung weist der nächsten freien Fertigungseinheit direkt den nächsten Fertigungsauftrag zu (s. o.). Dabei entstehen bei einem Produktwechsel Umrüstzeiten und -kosten. Verringert man die Umrüstzeiten bei gleichbleibender Produktionsrate führt dies i. Allg. bei stark ausgelasteten Systemen zu einem höheren Durchsatz und geringeren Umrüstkosten. Die Produktionskosten steigen dagegen durch den höheren Durchsatz an. Werden hingegen die Produktionsraten und nicht die Umrüstzeiten verringert, steigen mitunter die Warte-, Abweis- und Produktionskosten an.

Die untersuchten Modelle und deren Untersuchungsergebnisse werden im Abschn. 6.5.3 behandelt.

6.4.3. Modelltyp 3: M Fertigungseinheiten, N Produkte und Berücksichtigung von Lieferketten

Bei den bisherigen Modelltypen 1 und 2 wurde davon ausgegangen, dass jeglicher Bedarf an Rohteilen unmittelbar abgedeckt werden kann. Wird diese Annahme fallen gelassen, ist auch der Zulieferstrom innerhalb eines Modells geeignet ab-

zubilden. Diese Berücksichtigung erfolgt innerhalb des Modelltyps 3, welcher im Wesentlichen auf dem Modelltyp 2 aufbaut. Der Modelltyp 2 stellt wiederum eine Spezialisierung des Modelltyps 1 dar. Ziel des Modelltyps 3 ist es, den „hinteren" Teil eines Fertigungsprozesses nachzubilden, so dass nur die letzte Stufe dieses Fertigungsprozesses von Interesse ist. Folglich sind nur die Zeiten der letzten Stufe eines Fertigungsprozesses von Interesse. Alle anderen Zeiten des Fertigungsprozesses unterliegen stochastischen Einflüssen und werden mittels entsprechenden Wahrscheinlichkeitsverteilungen als „unscharfe Datenwerte" nachgebildet.

Dafür wird als wesentliche Veränderung des Modelltyps 3 gegenüber dem Modelltyp 2 eine weitere Entscheidungsvariable eingeführt, welche die *Anzahl der verfügbaren Kanbans*[112] eines Produktes i beinhaltet. Die Funktionsweise der Kanban-Steuerung in diesem Falle wird nachfolgend erklärt.

Visuelle Modellbeschreibung. Wie bereits zuvor angesprochen, besteht die wesentliche Erweiterung dieses Modelltyps in der Hinzunahme eines Steuerungsmechanismus zur Regulierung des Zulieferstroms von Rohteilen. Hierfür wird eine Methodik eingesetzt, welche der von Kanban-Systemen gleicht. Eine Lieferkette [dt.; supply chain [engl.]] wird bei diesem Modelltyp insofern vereinfacht, dass nur die Angabe der Lieferzeit sct_i und der Lieferkosten scc_i für ein bestimmtes Produkt i erfolgen muss (vgl. Tab. 6.7). Dadurch können beliebig komplexe Lieferketten abgebildet werden, ohne eine komplette Betrachtung der gesamten Lieferkette vornehmen zu müssen. Wie bereits zuvor erwähnt, ist beim Modelltyp 3 nur die letzte Stufe einer Lieferkette von Bedeutung. Dazu wird in Abb. 6.12 an einem Beispiel gezeigt, wie die Umgestaltung einer Lieferkette in ein Modell des Modelltyps 3 erfolgen kann.

Die Höhe einer Bestellforderung bei den (unendlich vielen) Zulieferern entspricht dabei der Größe eines Loses für einen Fertigungsauftrag. Anzumerken sei an dieser Stelle, dass als Freigabe- und Losgrößenstrategie nur die Fertigungspolitik (s, nQ) angewandt wird. Beim Einsatz der Fertigungspolitik (s, S) wäre es nicht möglich, im Vorfeld der Bestellung bei den Zulieferern zu sagen, wie hoch der tatsächliche Fertigungsauftrag zum Zeitpunkt des Produktionsbeginns ist.

Die Gestaltung des Produktions- und Lagerhaltungssystems im Ganzen ist im EcoSyL-Diagramm in Abb. 6.13 dargestellt. Die Bestellung bei einem Zulieferer erfolgt nur in dem Falle, wenn ein Kanban in der Kanban-Warteschlange des betreffenden Produktes verfügbar ist. Ist kein Kanban in der Kanban-Warteschlange vorhanden, wird das Los zunächst in der Bestellwarteschlange, welche eine unendliche Kapazität besitzt, aufbewahrt. Sobald ein leerer Kanban für ein wartendes Los eines Fertigungsauftrages in der Kanban-Warteschlange eintrifft, wird dieser mit der entsprechenden Information zur Bestellung von Rohteilen an einen der Zulieferer geschickt. Ein voller Kanban trifft stets im Pool mit den Produktionsaufträgen ein und bleibt solange an das betreffende Los gebunden, bis dieses als nächstes „zu

[112] Unter dem Begriff Kanban ist eine Methode der Produktionsablaufsteuerung zu verstehen, welche das Kernelement einer flexiblen Produktionssteuerung darstellt. Bei dieser Art der Produktionssteuerung wird nach dem pull-Prinzip [engl.; Hol-Prinzip [dt.]] gearbeitet, wobei sich ausschließlich am Bedarf einer verbrauchenden Stelle im Fertigungsablauf orientiert wird (siehe [GT05b]).

Abb. 6.12

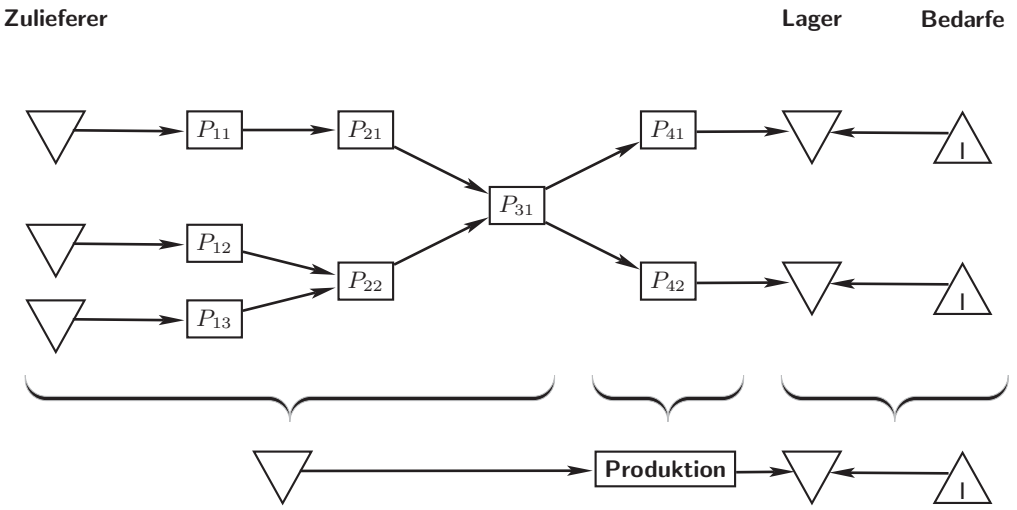


Abb. 6.12.: *Beispiel einer Lieferkette und deren Umformung in ein Modell des Modelltyps 3 (EcoSyL-Diagramm)*

fertigendes“ Los von einer Fertigungseinheit ausgewählt und dessen Produktion abgeschlossen wurde. Ab diesem Zeitpunkt ist der Kanban leer, gelangt zurück in die betreffende Kanban-Warteschlange und kann erneut für die Zulieferung von Rohteilen zur Fertigungseinheit genutzt werden. In allen anderen Vorgehensweisen gleicht der Modelltyp 3 dem Modelltyp 2. Das bedeutet im Speziellen, dass die verfügbaren Zuordnungs-, Losgrößen- und Reihenfolgestrategien identisch sind.

Weiteres Ereignis und zugehörige Ereignisroutine. Im Falle der Betrachtung des Produktions- und Lagerhaltungssystems bei vorangehender Berücksichtigung des Zulieferstromes weist der zugehörige Simulator neben den zuvor genannten simulatorspezifischen Ereignissen (s. o.) ein zusätzliches Ereignis auf.

- Simulatorspezifisches Ereignis:

  Ereignis „SupplyChainProductionEnd“.

  Dieses Ereignis zeigt an, dass die Fertigung einer Menge von zu verarbeitenden Rohteilen durch einen Zulieferer abgeschlossen ist. Die Abb. 6.14 zeigt die zugehörige Ereignisroutine.

Abb. 6.13

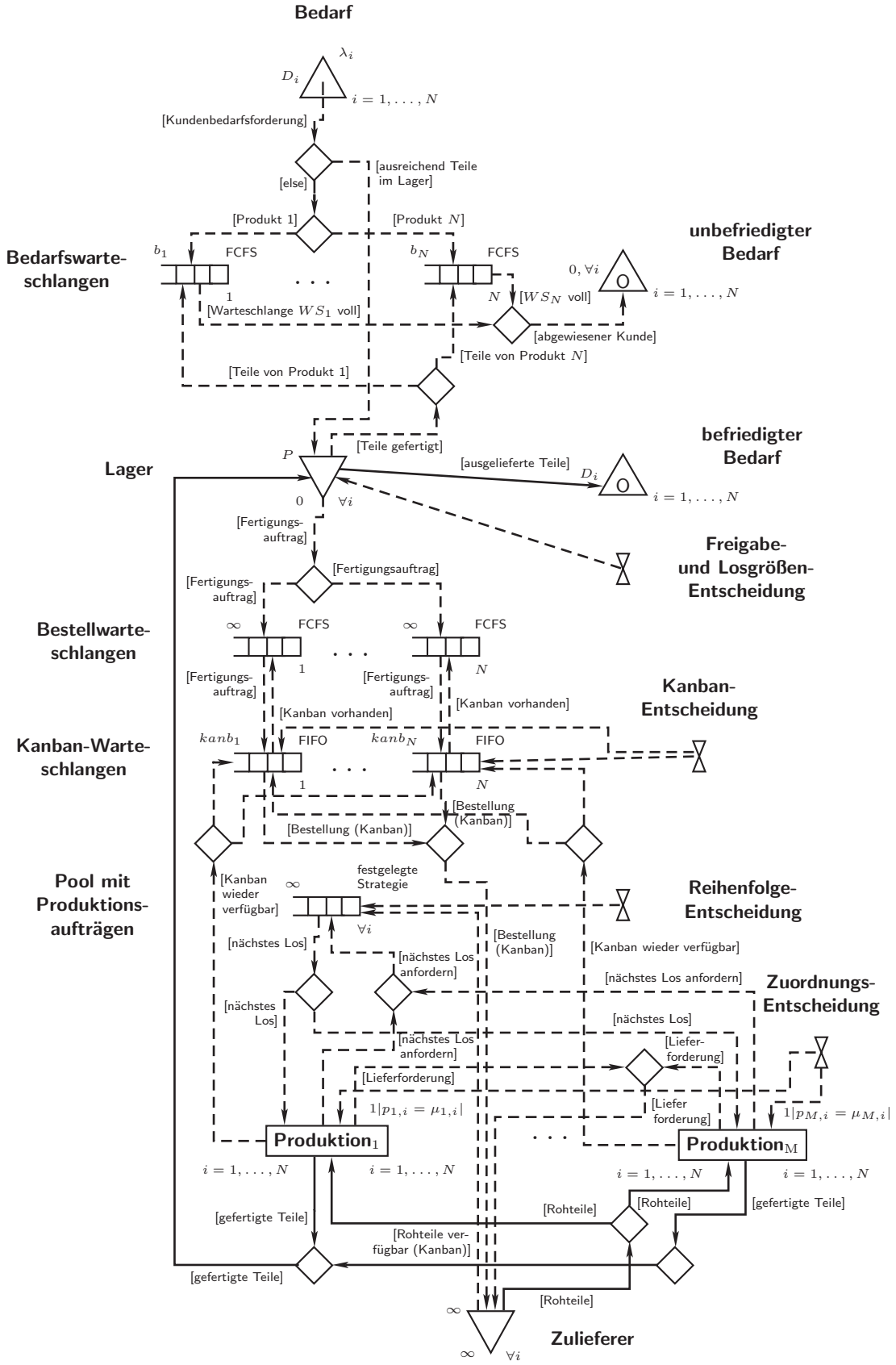


Abb. 6.13.: *Darstellung des untersuchten Produktions- und Lagerhaltungssystems beim Modelltyp 3 (EcoSyL-Diagramm)*

Abb. 6.14

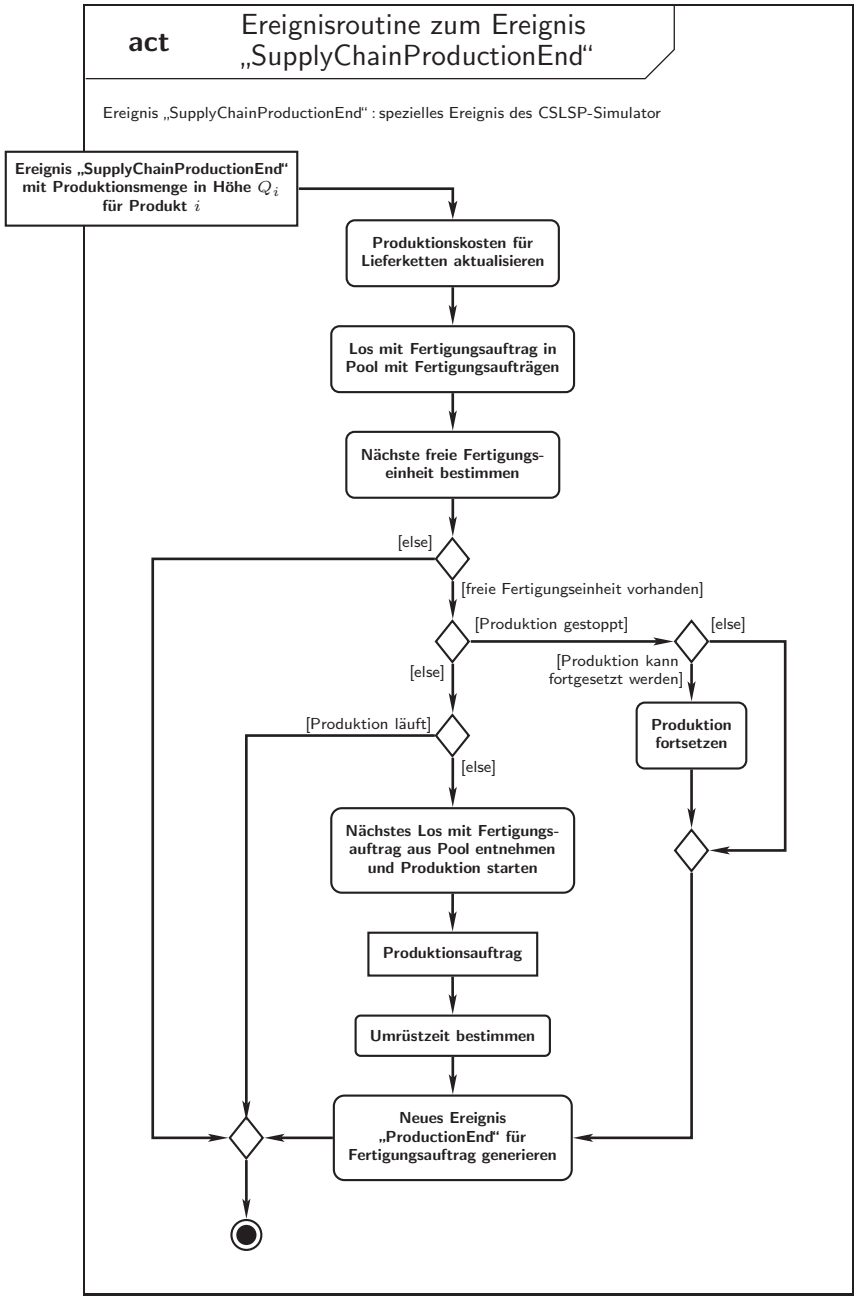


Abb. 6.14.: *Ereignisroutine des Ereignisses „SupplyChainProductionEnd“ beim CSLSP-Simulator* MultipleItemSupplyChainSystem *(UML-Aktivitätsdiagramm)*

Erweiterte Kostenfunktion. Bei Berücksichtigung des Zulieferstromes fallen zusätzliche Kosten an, welche bspw. durch die Produktion beim Zulieferer oder den Transport vom Zulieferer entstehen können. Im ökonomischen Sinne können darunter auch Beschaffungskosten verstanden werden, welche innerhalb dieser Arbeit als $SCC(\pi)$ gekennzeichnet seien (vgl. Tab. 6.7). Als zu minimierende Gesamtkostenfunktion $C_{SC}(\pi)$ ergibt sich:

$$\begin{aligned} C_{SC}(\pi) &= \sum_{i=1}^{N} \Big(hc_i \cdot HC_i(\pi) + wc_i \cdot WC_i(\pi) + rc_i \cdot RC_i(\pi) + pc_i \cdot PC_i(\pi) \\ &\quad + \sum_{j=1}^{N} sc_{ij} \cdot SC_{ij}(\pi) + scc_i \cdot SCC_i(\pi) \Big) \end{aligned}$$

bzw. (ergibt sich durch Umformung (siehe auch Tab. 6.5 und Tab. 6.7))

$$C_{SC}(\pi) = HC(\pi) + WC(\pi) + RC(\pi) + PC(\pi) + SC(\pi) + SCC(\pi) \quad ,$$

wobei

$$\min_{\pi \in \Pi} C_{SC}(\pi) \tag{6.5}$$

mit

π	-	aktuelle Gesamtstrategie (aktuell gewählte Reihenfolge-, Zuordnungs- und Losgrößenstrategie),
Π	-	Menge aller Strategien,
$HC_i(\pi)$	-	mittlere Anzahl der gelagerten Teile für Produkt i pro Zeiteinheit,
$WC_i(\pi)$	-	mittlere Anzahl wartender Forderungen für Produkt i pro Zeiteinheit,
$RC_i(\pi)$	-	mittlere Anzahl abgewiesener Kunden für Produkt i pro Zeiteinheit,
$PC_i(\pi)$	-	mittlere Anzahl produzierter Teile für Produkt i pro Zeiteinheit,
$SC_{ij}(\pi)$	-	mittlere Anzahl Umrüstvorgänge für Produktwechsel von Produkt i zu Produkt j pro Zeiteinheit sowie
$SCC_i(\pi)$	-	Zulieferkosten für Produkt i pro Zeiteinheit

unter Einhaltung der Nebenbedingungen 6.2 und 6.3 gesucht wird.

Weitere Entscheidungsvariable. Beim Modelltyp 3 werden zur Steuerung der Zulieferung von Rohteilen produktspezifische Kanbans verwendet (s. o.). Es existieren pro Produkt eine bestimmte Anzahl von Kanbans. Für diese Anzahl produktspezifischer Kanbans wird eine weitere Entscheidungsvariable namens x_{kanb} eingeführt. Diese enthält wiederum die obere und untere Grenze für die Anzahl der Kanbans,

um die Dimensionierung und Beschränkung des Suchraumes vorzunehmen, sowie die aktuelle Anzahl der Kanbans $kanb_i$ für ein bestimmtes Produkt i zur konkreten Realisierung eines Simulationsexperimentes. Die aktuelle Anzahl der Kanbans innerhalb der Entscheidungsvariablen x_{kanb} gibt an, für maximal wieviel Lose mit jeweils einem Fertigungsauftrag für ein bestimmtes Produkt gleichzeitig die Lieferung von Rohteilen mittels eines Kanbans erfolgen kann. An einen Kanban ist demnach genau ein Los mit einem Fertigungsauftrag gebunden. Zudem entspricht die Höhe des Inhaltes eines Kanbans genau der Anzahl der zu fertigenden Teile innerhalb eines Loses. Über die Anzahl der Kanbans werden die entstehenden Zulieferkosten $SSC_i(\pi)$ gesteuert.

Untersuchungsziele. Die Modelle des Modelltyps 3 können zur Untersuchung der Auswirkungen bei der Existenz detaillierterer Informationen über den Zulieferprozess dienen. Dies ermöglicht die Nachbildung von Wartezeiten bei der Zulieferung von Rohteilen. Die Bewertung der Ergebnisse der dabei durchzuführenden Untersuchungen kann Aufschluss über die Güte des Modells im Vergleich mit realexistierenden Systemen geben.

In der vorliegenden Arbeit werden jedoch keine Untersuchungsergebnisse für Modelle des Modelltyps 3 angegeben. Grund dafür sind u. a. der Umfang der durchzuführenden Untersuchungen, welcher notwendig ist, um aussagekräftige Ergebnisse zu erzielen. Untersuchungen mit Modellen des Modelltyps 3 sollten aber in weiterführenden Arbeiten erfolgen, um eine Bewertung der Auswirkungen bei zusätzlicher Beachtung des Zulieferprozesses vornehmen zu können.

6.5. Modellbeispiele mit Ergebnissen

6.5.1. Voruntersuchungen

Untersuchungsziel. Bevor die Sammlung der Ergebnisse der verschiedenen Modelle zu den Modelltypen 1 und 2 durchgeführt wurde, sollte im Rahmen von Voruntersuchungen herausgefunden werden, welches Optimierungsverfahren sich dafür am Geeignetsten erweist. Dazu wurden die im Abschn. 5.3.6 vorgestellten Optimierungsverfahren unter Anwendung verschiedener Beispielmodelle und verschiedener Verfahrensparameter für die Optimierungsverfahren untersucht.

Fazit. Als Ergebnis der Voruntersuchungen zeigt sich, dass die implementierte Variante der Tabusuche für diesen Anwendungsfall mit eines der besten Optimierungsverfahren ist. Dazu sind in Abb. 6.15 die jeweils minimalen, maximalen und mittleren Lösungswerte bzw. der Lösungsnummern der jeweiligen Optimierungsverfahren angegeben. Die besten Lösungsnummern bezeichnen hierbei die Lösungsnummer, bei welcher die letzte Verbesserung des Zielfunktionswertes aufgetreten ist.

Tab. 6.8

Optimierungsverfahren (Anzahl Voruntersuchungen)	**Untersuchte Parametereinstellungen**
GA_{gen}	Mutationswahrscheinlichkeit und -rate
(21 Voruntersuchungen)	Selektionswahrscheinlichkeit und -art
	Rekombinationswahrscheinlichkeit
	Anzahl von Individuen einer Population
	Anzahl elitärer Individuen
GA_{ss}	Anzahl zu ersetzender Individuen innerhalb der Population
(25 Voruntersuchungen)	Anzahl von Individuen einer Population
Tabusuche	Tabulistenlänge
(7 Voruntersuchungen)	
Simulierte Abkühlung	Mutationsrate
(21 Voruntersuchungen)	Anzahl akzeptierter Lösungen (Abkühlungsfaktor)
	Abkühlungsfaktor
Hybrid-seriell	Eröffnungsverfahren
(6 Voruntersuchungen)	Verbesserungsverfahren
Hybrid-parallel	Menge der parallel verwendeten Optimierungsverfahren
(5 Voruntersuchungen)	

Tab. 6.8.: *Überblick über die untersuchten Parametereinstellungen der verwendeten Optimierungsverfahren*

Abb. 6.15

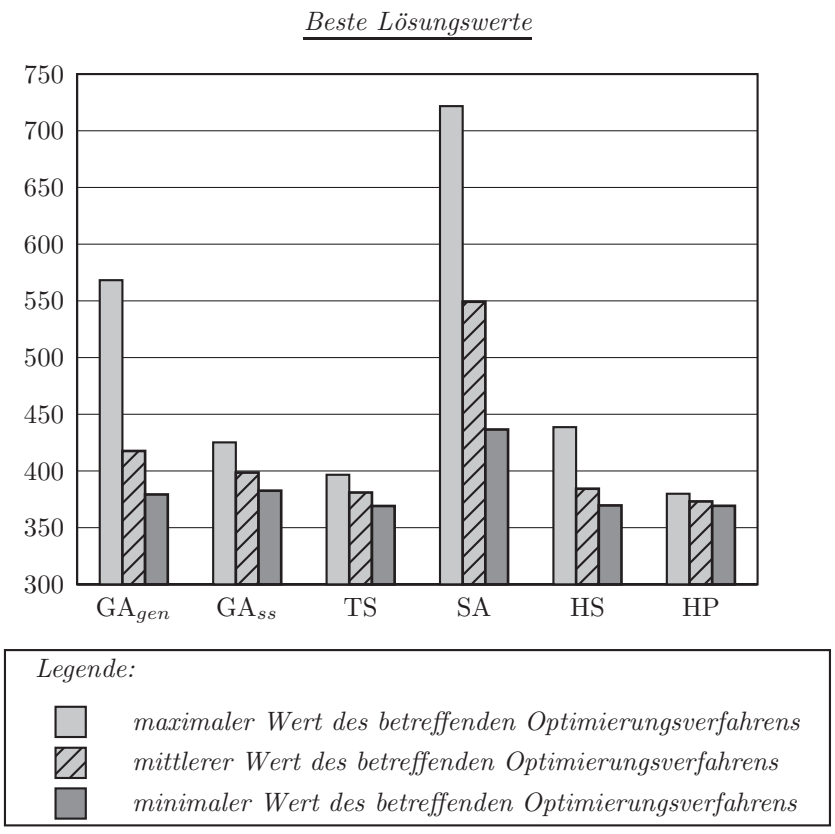


Abb. 6.15.: *Vergleich der Ergebnisse der Voruntersuchungen für verschiedene Optimierungsverfahren (1/3)*

Abb. 6.16

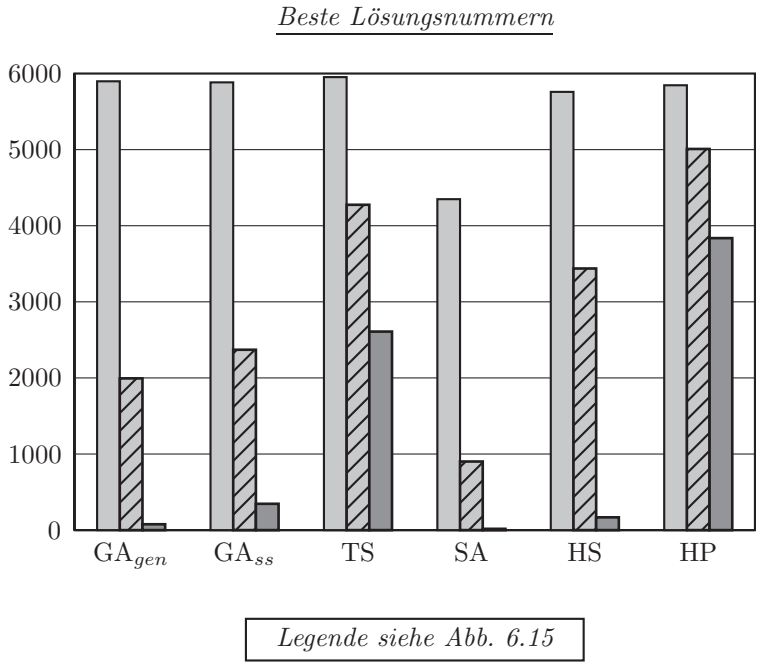


Abb. 6.16.: *Vergleich der Ergebnisse der Voruntersuchungen für verschiedene Optimierungsverfahren (2/3)*

In Abb. 6.17 sind darüber hinaus die jeweils beste Lösungsnummer und der beste Lösungswert der einzelnen untersuchten Optimierungsverfahren zum Vergleich noch einmal ins Verhältnis gesetzt.

Aus den Abbildungen 6.16 und 6.17 wird auch erkennbar, dass sich neben dem Optimierungsverfahren der Tabusuche auch die Variante des Genetischen Algorithmus mit stetiger Ersetzung sowie das hybrid-parallele Optimierungsverfahren als geeignete Optimierungsverfahren im Rahmen der durchzuführenden simulationsbasierten Optimierung erweisen. Ebenso wird deutlich, dass sich das Optimierungsverfahren der Simulierten Abkühlung bei den durchgeführten Untersuchungen als eher ungeeignet erweist. Dies kann u. a. auf die Struktur des Lösungsraumes sowie die implementierte Verfahrensvariante zurückzuführen sein. Des Weiteren wird aus den internen Ergebnisdateien für den Autor ersichtlich, dass sowohl das hybrid-parallele Optimierungsverfahren als auch das hybrid-serielle Optimierungsverfahren dann günstigere Zielfunktionswerte erreichen, wenn das Optimierungsverfahren der Tabusuche in der Menge der anzuwendenden Optimierungsverfahren enthalten ist.

Im Wesentlichen wurden für die nachfolgenden Untersuchungen die geeignetsten Optimierungsverfahren der Voruntersuchungen ausgewählt. Der Autor entschied sich dabei vorrangig für die Verfahren der Tabusuche sowie den Genetischen Algorithmus mit stetiger Ersetzung, weil diese am Robustesten gegenüber Veränderungen einzelner Parameter der Optimierungsverfahren reagierten. Die hybriden Optimierungsverfahren wurden nicht angewandt, weil sie, wie zuvor beschrieben, sich nur dann als günstiges Lösungsverfahren erweisen, wenn die Tabusuche enthalten

Abb. 6.17

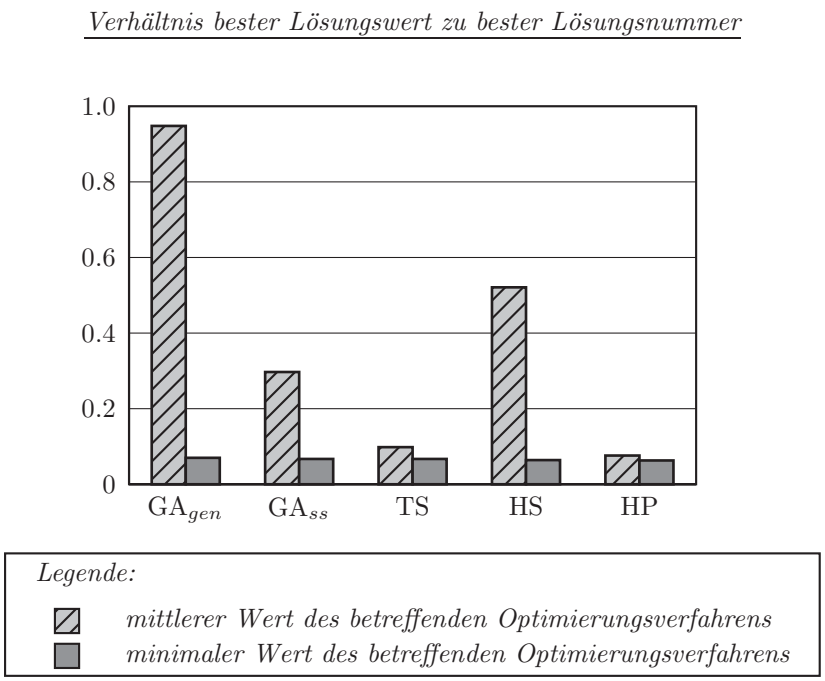


Abb. 6.17.: *Vergleich der Ergebnisse der Voruntersuchungen für verschiedene Optimierungsverfahren (3/3)*

ist. Dennoch weisen die hybriden Verfahren unter der Anwendung der Tabusuche eine langsame Konvergenz des Zielfunktionswertes gegen ein globales Optimum auf, als dies bei ausschließlicher Anwendung der Tabusuche der Fall ist.

6.5.2. Modellbeispiele und Ergebnisse für Modelltyp 1

Wie bereits im Abschn. 6.4.1 beschrieben, wird beim Modelltyp 1 eine Fertigungseinheit zur Produktion von N Produkten verwendet. Er dient zur Bestätigung der Nichtdominanz der zyklischen Reihenfolgestrategie und der gleichzeitigen Suche nach einer dominanten Reihenfolgestrategie. Des Weiteren ist die Untersuchung der Auswirkungen detaillierter Informationen über die Produkte von Interesse.

Modellparameter. In den Tabellen 6.9 und 6.10 sind dazu die gemeinsamen und unterschiedlichen Modellparameter und Entscheidungsvariablen für die 3 Modellbeispiele angegeben.

Anzumerken sei an dieser Stelle die Bedeutung der Angabe $x|0$ bei den Umrüstkosten st_{ij} in Tab. 6.10. Sie bedeutet, dass Umrüstzeiten st_{ij} und -kosten sc_{ij} bei einem Produktwechsel von einem Produkt i auf dasselbe Produkt i in Höhe von x nur in dem Falle entstehen, wenn die Fertigung desselben Produkts nicht unmittelbar folgt. Anderenfalls treten keine Umrüstzeiten und -kosten auf. Eine Gewichtung der zu bewertenden Ausgabewerte eines Simulationslaufes für die Abweiskosten ($RC(\pi)$), die Wartekosten ($WC(\pi)$), die Lagerkosten ($HC(\pi)$), die

Produktionskosten ($PC(\pi)$) und die Umrüstkosten ($SC(\pi)$) zur Bestimmung des Zielfunktionswertes für den Verlust ($C(\pi)$) wurde nicht durchgeführt.

Tab. 6.9

Modellkomponente (Symbol)	Ein-, Zwei- und Fünf-Produkt-Modell
Kundenbedarfsmenge (d_i)	$D \begin{pmatrix} 1 & 10 & 50 \\ 0,6 & 0,3 & 0,1 \end{pmatrix}$; $i = 1, 2, \ldots, N$
Reihenfolgestrategie (*RS*)	{FCFS, Cyclic, LWQ, LWVQ, Dynamic, Random}
Losgrößen- und Freigabestrategie (LFS_i)	*MTS*: (s, nQ), (s, S); $i = 1, 2, \ldots, N$
Produktionsrate (μ_i)	50; $i = 1, 2, \ldots, N$
Lagerplatz (p_i)	1; $i = 1, 2, \ldots, N$
Umrüstkosten (sc_{ij})	st_{ij} (siehe Tab. 6.10)
Produktionskosten (pc_i)	1; $i = 1, 2, \ldots, N$
Lagerkosten (hc_i)	1; $i = 1, 2, \ldots, N$
Wartekosten (wc_i)	1; $i = 1, 2, \ldots, N$
Abweiskosten (rc_i)	20; $i = 1, 2, \ldots, N$
Produktionszeitpunkt (s_i)	$s_i^l = 1$, $s_i^s = 1$, $s_i^u = 500$; $i = 1, 2, \ldots, N$
Produktionsmenge (Q_i)	$Q_i^l = 1$, $Q_i^s = 1$, $Q_i^u = 500$; $i = 1, 2, \ldots, N$
Produktionsobergrenze (S_i)	$S_i^l = 1$, $S_i^s = 1$, $S_i^u = 500$; $i = 1, 2, \ldots, N$

Tab. 6.9.: *Überblick über die gemeinsamen Modellparameter und Entscheidungsvariablen der Modellbeispiele des Modelltypes 1*

Tab. 6.10

Modellkomponente (Symbol)	Ein-Produkt-Modell	Zwei-Produkt-Modell	Fünf-Produkt-Modell
Produkte (N)	1	2	5
Lagergröße (P)	∞	∞	∞
Kundenankunftsrate (λ_i)	$E(5)$	$E(2)$; $i = 1$ $E(3)$; $i = 2$	$E(1)$; $i = 1, 2, \ldots, 5$
maximale Warteschlangenlänge (b_i)	50	20; $i = 1$ 30; $i = 2$	10; $i = 1, 2, \ldots, 5$
Umrüstzeiten (st_{ij})	3|0	$\begin{pmatrix} 1{,}5\|0 & 4 \\ 1{,}5 & 4\|0 \end{pmatrix}$	j; $i, j = 1, 2, \ldots, 5$ $j\|0$; $j = 1, 2, \ldots, 5$

Tab. 6.10.: *Überblick über die unterschiedlichen Modellparameter und Entscheidungsvariablen der Modellbeispiele des Modelltypes 1*

Ein-Produkt-Modell. Mit Hilfe des Ein-Produkt-Modells sollen zunächst Referenzdaten für die Vergleiche mit den Zwei- und Fünf-Produkt-Modellen bestimmt werden. Dazu weist das Ein-Produkt-Modell die gemittelten Werte des Fünf-Produkt-Modells auf (vgl. Tab. 6.10). Beispielsweise werden die gemittelten Umrüstkosten und -zeiten verwendet ($\frac{1+2+3+4+5}{5} = 3$). Zusätzlich werden die fünf Kundenwarteschlangen mit einer jeweiligen maximalen Länge von $b_i = 10$ zu einer Kundenwarteschlange der Länge $b_1 = 50$ „kombiniert“.

Zudem soll anhand der optimalen Ergebnisse des Ein-Produkt-Modells eine Untersuchungung anhand der Visualisierung des Zielfunktionsraumes in der Nähe der gefundenen Optima vorgenommen werden.

Ergebnisse des Ein-Produkt-Modells. Die fünf Untersuchungsergebnisse[113] (Experimente) in Tab. 6.11 zeigen, das die LF-Strategie (s, S) in diesem Falle nicht dominant gegenüber der LF-Strategie (s, nQ) ist. Für die Bestimmung eines Untersuchungsergebnisses wurde aus Ressourcengründen die Zeitdauer für einen Lauf der simulationsbasierten Optimierung auf 48 Stunden festgesetzt[114]. Ein Simulationsexperiment besteht dabei aus 3 Simulationsläufen, wobei während eines Simulationslaufes ca. eine Million Kundenankünfte simuliert werden. Die reale Zeitdauer eines Simulationsexperimentes beträgt bei der eingesetzten Rechentechnik ca. eine Minute, so dass für etwa 3000 gültige Wertebelegungen für die Entscheidungsvariablen eine Simulation durchgeführt werden konnte. Zudem wird wie bei den Voruntersuchungen die Schwäche der Simulierten Abkühlung bei diesen Untersuchungsmodellen und die Dominanz der Tabusuche erkennbar.

Tab. 6.11

LFS	OV	s_{opt}	Q_{opt} / S_{opt}	RC(π)	WC(π)	HC(π)	PC(π)	SC(π)	C(π)
(s, S)	GA_{ss}	133	134	0,151	61,793	30,505	0,859	0,132	93,439
(s, S)	TS	141	142	0,133	57,601	34,468	0,859	0,132	93,194
(s, nQ)	GA_{ss}	137	58	0,136	57,246	37,891	0,860	0,110	96,243
(s, nQ)	TS	141	1	0,133	57,601	34,468	0,859	0,132	93,194
(s, nQ)	SA	489	528	0,008	2,354	377,872	0,861	0,077	381,172

Tab. 6.11.: *Ergebnisse der Tests für 1 Produkt (beliebige R-Strategie und alle LF-Strategien)*

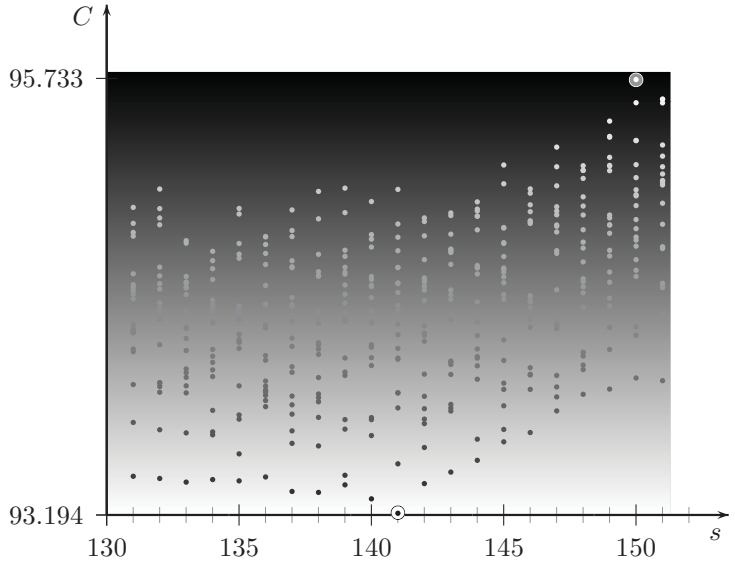
In den Abbildungen 6.18 und 6.19 ist der Zielfunktionsraum in der Nähe der optimalen Lösungen für die LF-Strategien (s, nQ) und (s, S) angegeben. Dabei sind zur besseren Erkennung der Werte jeweils verschiedene Ansichten für die unteren und oberen Produktionsgrenzen sowie die Produktionsmengen angegeben. Es ist ersichtlich, dass von der Tabusuche die optimalen Lösungen gefunden wurden. Zudem ist erkennbar, dass bei der LF-Strategie (s, nQ) der Wert der Entscheidungsvariable für die Produktionsmenge Q sich stärker auf den Lösungswert auswirkt als der Wert der Entscheidungsvariable für die untere Produktionsgrenze s (siehe Abb. 6.18 (a)). Des Weiteren hat bei der LF-Strategie (s, nQ) die Zunahme des Wertes für s stärkere Auswirkungen auf den Zielfunktionswert als die Abnahme von s (siehe Abb. 6.18 (b)). Bei der LF-Strategie (s, S) (siehe Abb. 6.19 (a) und (b)) verhält es sich analog. Zunehmende Werte für s und S führen zu schlechteren Zielfunktionswerten als die vergleichbare Wertabnahme von s und S zur Folge hat.

[113] Ein Untersuchungsergebnis entsteht als Resultat eines (Optimierungs-)Laufes mittels simulationsbasierter Optimierung.

[114] Teilweise wurde bei den Untersuchungen auch auf die im Abschn. 4.5.2 beschriebene Methode der verteilten und parallelen Optimierung zurückgegriffen. Zum besseren Vergleich der Zeitdauern der einzelnen Untersuchungsergebnisse werden allerdings nur die Realzeiten ohne Einsatz dieser Methode betrachtet. Gleiches gilt für den vorzeitigen Abbruch eines Simulationslaufes durch das Epsilon-Abbruch-Verfahren (siehe Abschn. 3.6). Das Verfahren wird bei den Betrachtungen der realen Zeiten außer Acht gelassen, aber dennoch weitestgehend eingesetzt.

Abb. 6.18

(a) Vergleich der unteren Produktionsgrenzen ($131 \leq s \leq 151$*)*



(b) Vergleich der Produktionsmengen ($1 \leq Q \leq 21$*)*

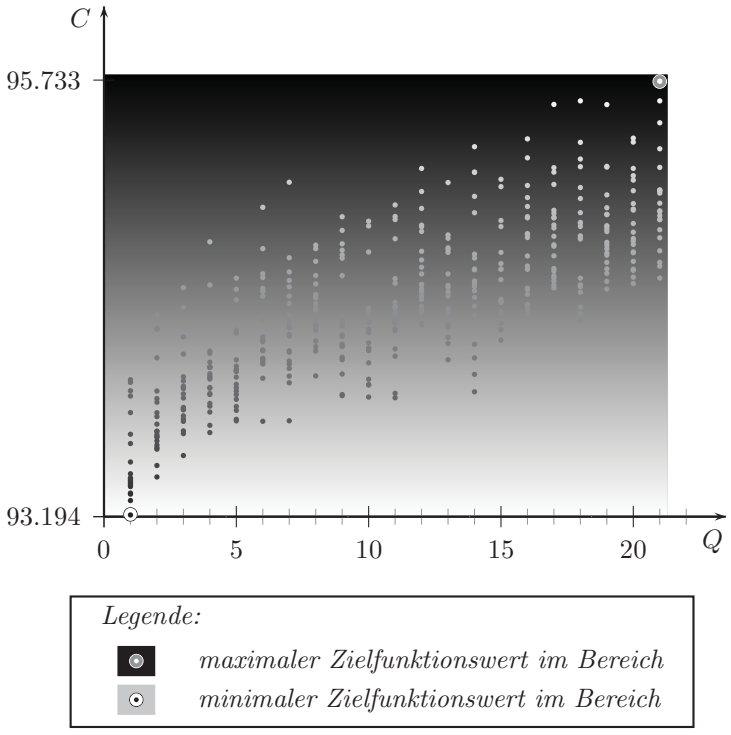
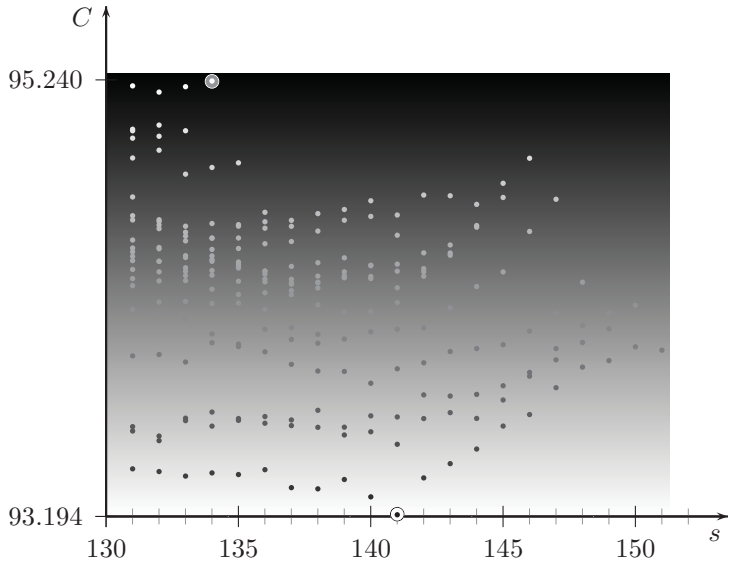


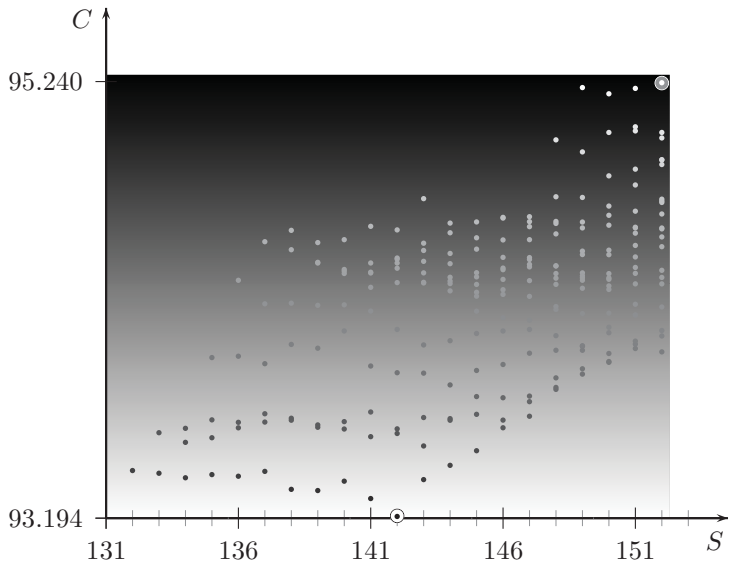
Abb. 6.18.: *Vergleich der unteren Produktionsgrenzen s und Produktionsmengen Q beim Ein-Produkt-Modell und LF-Strategie* (s, nQ)

Abb. 6.19

(a) Vergleich der unteren Produktionsgrenzen ($131 \leq s \leq 151$*)*



(b) Vergleich der oberen Produktionsgrenzen ($132 \leq S \leq 152$*)*



Legende vgl. Abb. 6.18

Abb. 6.19.: *Vergleich der unteren Produktionsgrenzen s und der oberen Produktionsgrenzen S beim Ein-Produkt-Modell und LF-Strategie (s, S)*

Zwei-Produkt-Modell. Das Zwei-Produkt-Modell zielt auf eine Erhöhung des Detaillierungsgrades gegenüber dem Ein-Produkt-Modell ab (vgl. Tab. 6.10). Hierfür wurden u. a. die Warteschlangenlängen b_i entsprechend der jeweiligen Ankunftsraten λ_i als Zusammenfassung der Modellparameter von den Produkten 1 und 2 sowie 3, 4 und 5 des Fünf-Produkt-Modells angepasst. Die Umrüstkosten und -zeiten setzen sich aus den gemittelten Werten des Fünf-Produkt-Modelles zusammen ($\frac{1+2}{2} = 1,5$ und $\frac{3+4+5}{3} = 4$).

Ergebnisse des Zwei-Produkt-Modells. Zur Untersuchung des Zwei-Produkt-Modells wurden insgesamt 72 Experimente (Läufe mittels simulationsbasierter Optimierung) durchgeführt, wobei jeweils die beiden Losgrößen- und Freigabestrategien (LFS), die 6 Reihenfolgestrategien (RS) sowie die 6 implementierten Optimierungsverfahren (OV) zum Gesamtergebnis beitrugen. Die Durchführung eines Experimentes betrug 140 Stunden, wobei wiederum ein Simulationsexperiment aus drei Simulationsläufen bestand und ohne vorzeitigen Abbruch eines Simulationslaufes durch das Epsilon-Abbruch-Verfahren (siehe Abschn. 3.6) eine Million Kundenankünfte pro Simulationslauf simuliert wurden. Ein Simulationslauf dauerte beim Zwei-Punkt-Modell im Mittel ca. 2,5 Minuten.

In Tab. 6.12 sind die besten gefundenen Zielfunktionswerte für die beiden Losgrößen- und Freigabestrategien sowie die 6 Reihenfolgestrategien unter Angabe des betreffenden günstigsten Optimierungsverfahren dargestellt. Aus ihnen wird zum einen die Dominanz der LF-Strategie (s, S) gegenüber der LF-Strategie (s, nQ) und zum anderen die Bestätigung der Annahme, dass das Optimierungsverfahren der Tabusuche am Geeignetsten für die durchgeführten Untersuchungen war, erkennbar. Ersichtlich ist auch, dass in Ausnahmefällen das Optimierungsverfahren der Simulierten Abkühlung durchaus optimale Lösungen finden kann. Weitere Ergebnisse (s. u.) zeigen jedoch, dass bei komplexeren Suchräumen das Optimierungsverfahren der Simulierten Abkühlung kaum eine optimale Lösung mehr findet. Aus Tab. 6.12 wird auch ersichtlich, dass bei diesem Beispiel die dynamischen R-Strategien (Dynamic, LWQ, LWVQ und Random) die statischen R-Strategien (FCFS und Cyclic) dominieren.

Ein weiteres, zu kommentierendes Ergebnis wird aus Tab. 6.12 und weiter unten auch aus Tab. 6.13 erkennbar. Die Reihenfolgestrategie Random ist zumeist kostengünstiger als die Reihenfolgestrategie FCFS. Zudem entstehen bei der Reihenfolgestrategie FCFS höhere Umrüstkosten als bei der Reihenfolgestrategie Random. Die sind darauf zurückzuführen, dass bei Existenz einer hohen Anzahl von Losen mit Fertigungsaufträgen für ein bestimmtes Produkt im Pool mit hoher Wahrscheinlichkeit eine große Fehlmenge für dieses Produkt bei Anwendung der Reihenfolgestrategie FCFS auftritt. Die Warte- und Abweiskosten für Kunden, welche dieses Produkt nachfragen, steigen. Gleichzeitig sinken jedoch die Lagerkosten, was in diesem Falle jedoch eine weniger starke Auswirkung auf die Gesamtkosten hat. Begründet liegt dies darin, dass bei der Reihenfolgestrategie Random mit entsprechender Wahrscheinlichkeit die Produktion für ein Los eines bestimmten Produktes gestartet wird, während bei der Reihenfolgestrategie FCFS erst alle vorangegangenen Lose zu fertigen sind. Diese Auswahlwahrscheinlichkeit steigt mit

der Fehlmenge eines Produktes an und sinkt mit jedem realisierten Los. Zu beachten ist allerdings, dass die untersuchten Modelle eine hohe Auslastung besitzen. Diese Auswirkungen einer hohen Auswahlwahrscheinlichkeit sind besonders stark, wenn sich viele Lose mit kleinen Fertigungsaufträgen, z. B. für einen kleinen Wert für Q_i, in ungeordneter Reihefolge im Pool befinden. Das Optimierungsverfahren muss daher eine Lösung suchen, welche einen Kompromiss zwischen kleinen und großen Fertigungsmengen beinhaltet. Insbesondere die Ergebnisse in Tab. 6.13 spiegeln dies wider.

Tab. 6.12

LFS	RS	OV	$\vec{s}_{opt}$	$\vec{Q}_{opt}$ / $\vec{S}_{opt}$	RC(π)	WC(π)	HC(π)	PC(π)	SC(π)	C(π)
(s, S)	Cyclic	GA_{gen}	238|432	246|435	10,176	155,714	66,768	0,773	0,227	233,659
(s, nQ)	Cyclic	TS	1|1	333|609	11,419	159,900	71,298	0,762	0,217	243,596
(s, S)	Dynamic	HP	107|137	200|143	10,024	151,077	52,795	0,774	0,226	214,895
(s, nQ)	Dynamic	HP/TS	62|169	200|1	16,591	183,434	50,009	0,718	0,282	251,034
(s, S)	FCFS	TS	1|198	200|199	15,694	217,054	22,958	0,726	0,274	256,706
(s, nQ)	FCFS	TS	166|1	1|615	10,459	158,048	68,116	0,770	0,227	237,620
(s, S)	LWQ	GA_{gen}	197|335	216|495	9,665	150,073	72,541	0,777	0,223	233,279
(s, nQ)	LWQ	TS	18|1	345|570	10,720	161,008	67,784	0,768	0,215	240,495
(s, S)	LWVQ	TS	181|327	194|384	10,833	165,324	51,737	0,767	0,233	228,893
(s, nQ)	LWVQ	SA	11|43	303|548	11,450	162,937	64,286	0,762	0,225	239,661
(s, S)	Random	GA_{gen}	238|432	246|435	10,176	155,722	66,766	0,773	0,227	233,665
(s, nQ)	Random	TS	3|138	378|292	10,448	160,715	67,550	0,771	0,222	239,706

Tab. 6.12.: *Ergebnisse der Tests für 2 Produkte (jeweils optimale Lösungen der R-Strategien und LF-Strategien)*

Fünf-Produkt-Modell. Das Fünf-Produkt-Modell ist das Detaillierteste der drei Modelle und soll zeigen, wie genau die Informationen zu einem Modell nötig sind. Die gemeinsamen und unterschiedlichen Modellparameter und Entscheidungsvariablen wurden bereits zuvor in den Tabellen 6.9 und 6.10 angegeben.

Ergebnisse des Fünf-Produkt-Modells. Wie im Falle des Zwei-Produkt-Modells wurden auch für das Fünf-Produkt-Modell insgesamt 72 Experimente durchgeführt (zwei LF-Strategien, 6 R-Strategien und 6 Optimierungsverfahren). Ein Experiment wurde für eine Zeitdauer von ca. 215 Stunden durchgeführt, wobei ein Simulationsexperiment mit drei Simulationsläufen wie beim Zwei-Produkt-Modell etwa 2,5 Minuten dauerte. In Tab. 6.13 sind die besten gefundenen Zielfunktionswerte für die beiden Losgrößen- und Freigabestrategien sowie die 6 Reihenfolgestrategien unter Angabe des Optimierungsverfahrens, welches diese besten Zielfunktionswerte zurücklieferte, dargestellt.

Die Ergebnisse des Zwei-Produkt-Modells können durch die Ergebnisse des Fünf-Produkt-Modells nur teilweise bestätigt werden. Das Optimierungsverfahren der Tabusuche (TS) liefert nach wie vor die meisten optimalen Lösungen. Ein Unterschied ist allerdings, dass die LF-Strategie (s, S) nicht für jede R-Strategie dominant gegenüber der LF-Strategie (s, nQ) ist. Dieses Ergebnis wurde im Vorfeld der Untersuchungen nicht erwartet, weil oftmals von einer Dominanz der LF-Strategie (s, S) über die LF-Strategie (s, nQ) ausgegangen wird.

Ein weiteres Ergebnis, welches nach den Voruntersuchungen und den Ergebnissen des Ein- und Zwei-Produkt-Modells nicht zu erwarten war, ist, dass eine optimale Lösung durch das Optimierungsverfahren der Simulierten Abkühlung bestimmt

Tab. 6.13

LFS	RS	OV	$\vec{s}_{opt}$	$\vec{Q}_{opt}$ / $\vec{S}_{opt}$	RC(π)	WC(π)	HC(π)	PC(π)	SC(π)	C(π)
(s, S)	Cyclic	HP	1|1|1|34|1	9|2|22|52|7	56,970	188,742	5,686	0,371	0,629	252,399
(s, nQ)	Cyclic	TS	779|825|1|18|11	158|160|176|174|165	38,651	261,040	74,440	0,527	0,473	375,131
(s, S)	Dynamic	HP	1|1|1|1|8	4|2|6|10|13	51,445	163,323	0,600	0,418	0,582	216,368
(s, nQ)	Dynamic	HP	5|28|62|26|42	11|1|6|28|11	84,473	129,966	0,000	0,134	0,866	215,439
(s, S)	FCFS	TS	68|71|65|45|56	69|102|83|96|106	37,986	260,496	56,665	0,532	0,468	356,148
(s, nQ)	FCFS	TS	36|38|1|1|1	142|157|177|180|191	36,345	252,033	79,779	0,548	0,452	369,156
(s, S)	LWQ	TS	44|64|52|92|70	72|93|109|111|114	36,506	254,195	65,188	0,546	0,454	356,889
(s, nQ)	LWQ	TS	238|251|247|251|298	161|167|163|173|184	38,382	247,514	70,893	0,530	0,470	357,789
(s, S)	LWVQ	TS	1|2|3|8|8	10|8|9|9|9	58,123	180,090	1,431	0,360	0,640	240,644
(s, nQ)	LWVQ	SA	1|1|1|1|1	1|1|1|1|1	93,261	118,674	0,000	0,058	0,942	212,935
(s, S)	Random	TS	26|95|56|64|8	173|98|62|83|179	45,499	239,237	59,483	0,469	0,531	345,219
(s, nQ)	Random	TS	53|72|53|64|49	77|106|103|101|140	48,220	231,593	53,477	0,446	0,554	334,289

Tab. 6.13.: *Ergebnisse der Tests für 5 Produkte (jeweils optimale Lösungen der R-Strategien und LF-Strategien)*

wurde. Allerdings scheint die ermittelte, optimale Lösung aus ökonomischer Sicht nicht sinnvoll. Die Umrüstkosten deuten auf viele Umrüstvorgänge und vergleichsweise wenige Produktionsvorgänge hin, wodurch nahezu alle Kunden abgewiesen werden. Ein Ausweg wäre eine andere, aus ökonomischer Sicht sinnvollere Gewichtung der ermittelten Zielfunktionswerte vorzunehmen und die Läufe der simulationsbasierten Optimierung für das Fünf-Produkt-Modell erneut durchzuführen. Eine andere Möglichkeit ist die Anwendung eines mehrkriteriellen Optimierungsverfahrens. Aus diesem Grund wurde innerhalb dieser Arbeit umfangreicher auf das Thema der mehrkriteriellen Optimierung eingegangen (siehe Abschnitte 2.2.3, 2.2.5, 2.6 und 4.4).

Fazit. Als Resultat aus allen Ergebnissen der Modelle zum ersten Modelltyp ergibt es sich als sinnvoll, stets das anzuwendende Optimierungsverfahren, die zu nutzende Reihenfolgestrategie sowie die Losgrößen und Freigabestrategie in Kombination zu betrachten. Meist liefert das gleiche Optimierungsverfahren für ein bestimmtes Tupel aus R-Strategie und LF-Strategie die optimale Lösung. Darüber hinaus scheint das Optimierungsverfahren der Tabusuche jedoch dominant zu sein. Um diese Aussagen zu bestätigen wurden weitere Untersuchungen bzw. Experimente durchgeführt.

6.5.3. Modellbeispiele und Ergebnisse für Modelltyp 2

Bevor die Modellparameter der Modelle zur Untersuchung des zweiten Modelltypes näher betrachtet werden, sei angemerkt, dass es sich bei den modellierten Fertigungseinheiten um identisch parallele Fertigungseinheiten handelt, d. h. für die Produktionsrate μ_{ik} gilt $\mu_{ik} = \mu_i$, $k = 1, 2, \ldots, M$. Somit kann, auf Grund der Gleichartigkeit der Fertigungseinheiten, auf die Angabe des Indexes k verzichtet werden. Darüber hinaus können alle Fertigungseinheiten gleichzeitig zur Fertigung eingesetzt werden, wobei eine Fertigungseinheit nach wie vor zu einem bestimmten Zeitpunkt nur einen Fertigungsauftrag bearbeiten kann.

Modellparameter. In den Tabellen 6.14 und 6.15 sind die Modellparameter angegeben, welche sich im Vergleich zu den Modellen des Modelltyps 1 (siehe Tabellen 6.9 und 6.10) verändert haben. Dazu sind in Tab. 6.14 die unterschiedlichen Modellparameter und in Tab. 6.15 die gemeinsamen Modellparameter der untersuchten Modelle des Modelltyps 2 angeführt.

Tab. 6.14

Modellkomponente (Symbol)	Zwei-Fertigungseinheiten-Modell ($scst^{voll}$)	Zwei-Fertigungseinheiten-Modell ($scst^{halb}$)	Vier-Fertigungseinheiten-Modell ($scst^{viertel}$)
Fertigungseinheiten (M)	2	2	4
Produktionsrate (μ_{ik})	25; $i = 1, 2, \ldots, 5$; $k = 1, 2$	25; $i = 1, 2, \ldots, 5$; $k = 1, 2$	12,5; $i = 1, 2, \ldots, 5$; $k = 1, 2, 3, 4$
Umrüstzeiten (st_{ijk})	j; $i, j = 1, 2, \ldots, 5$; $k = 1, 2, \ldots, M$	$\frac{j}{2}$; $i, j = 1, 2, \ldots, 5$; $k = 1, 2, \ldots, M$	j; $i, j = 1, 2, \ldots, 5$; $k = 1, 2, \ldots, M$

Tab. 6.14.: *Überblick über die veränderten, unterschiedlichen Modellparameter der Modellbeispiele des Modelltypes 2 im Gegensatz zum Modelltyp 1*

Tab. 6.15

Modellkomponente (Symbol)	Zwei- und Vier-Fertigungseinheiten-Modell
Umrüstkosten (sc_{ijk})	st_{ijk} (siehe Tab. 6.14)
Produktionskosten (pc_{ik})	1; $i = 1, 2, \ldots, 5$; $k = 1, 2, \ldots, M$

Tab. 6.15.: *Überblick über die veränderten, gemeinsamen Modellparameter der Modellbeispiele des Modelltypes 2 im Gegensatz zum Modelltyp 1*

Fünf-Produkt-Modell mit zwei Fertigungseinheiten ($scst^{voll}$). Zunächst sei die Auslastung des Fünf-Produkt-Modells mit einer Fertigungseinheit für die optimale Lösung $\vec{s}_{opt} = (1, 1, 1, 1, 8)^T$ und $\vec{S}_{opt} = (4, 2, 6, 10, 13)^T$ bei Anwendung der R-Strategie Dynamic sowie der LF-Strategie (s, S) betrachtet (vgl. Tab. 6.13). Dies soll anhand des Verlaufes der Anzahl der abgewiesenen Kunden pro Zeiteinheit geschehen (siehe Abb. 6.20 (a)).

Aus diesem Verlauf wird erkennbar, dass im Mittel mehr als die Hälfte der ankommenden Kunden abgewiesen werden. Zur Befriedigung des Kundenbedarfes ist mitunter die Einführung einer zweiten Fertigungseinheit notwendig[115]. Für die nachfolgenden Untersuchungen soll diese weitere Fertigungseinheit identische Parameter zur ersten Fertigungseinheit besitzen, um deren Auswirkungen auf die optimale Lösung zu ermitteln.

Ergebnisse des Fünf-Produkt-Modells mit zwei Fertigungseinheiten ($scst^{voll}$). Zur Untersuchung wurden 20 unterschiedliche Läufe mittels simulationsbasierter Optimierung durchgeführt. Ein Untersuchungsergebnis beruht dabei auf der Simulation

[115] Aus „betriebswirtschaftlicher Kostensicht“ kann die Betrachtung allerdings anders ausfallen, weil die Investitionskosten für die Anschaffung der Fertigungseinheit höher ausfallen könnten als die Abweiskosten für nicht befriedigte Kundenbedarfe.

Abb. 6.20

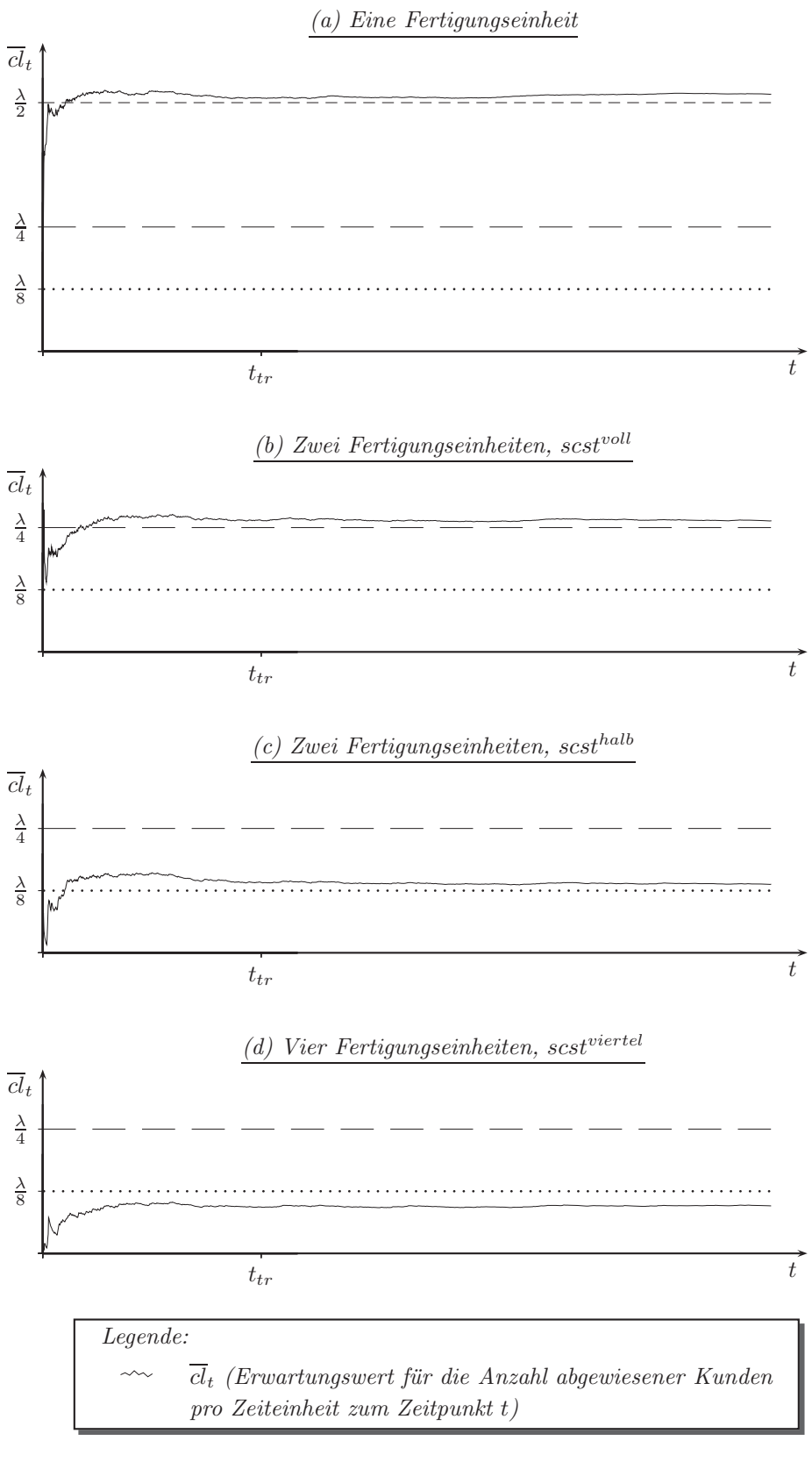


Abb. 6.20.: *Abgewiesene Kunden bei den Fünf-Produkt-Modellen für die optimalen Lösungen der Untersuchungsmodelle bei Anwendung der R-Strategie LWVQ und der LF-Strategie* (s, S)

von 6000 gültigen Werten für die Entscheidungsvariablen. Im Mittel dauerte ein Simulationsexperiment aus drei Simulationsläufen bei der eingesetzten Rechentechnik[116] 2,5 Minuten.

10 Untersuchungsergebnisse entstanden durch Anwendung des Optimierungsverfahrens TS und die anderen 10 durch Nutzung des Optimierungsverfahrens GA_{ss}. Hierbei stellte sich wiederum die Dominanz des Optimierungsverfahrens TS heraus. Aus diesem Grunde sind die in Tab. 6.16 angegeben optimalen Lösungen auf das Optimierungsverfahren TS zurückzuführen. Die für das Optimierungsverfahren GA_{ss} erzielten Lösungen werden auf Grund der stets schlechteren Lösungswerte nicht betrachtet.

Tab. 6.16

LFS	RS	$\vec{s}_{opt}$	$\vec{Q}_{opt}$ / $\vec{S}_{opt}$	RC(π)	WC(π)	HC(π)	PC(π)	SC(π)	C(π)
(s, S)	Cyclic	95|65|84|48|1	105|116|92|108|166	20,506	202,366	74,163	1,367	0,633	299,035
(s, nQ)	Cyclic	1|1|1|1|1	155|167|164|163|181	19,814	197,946	86,389	1,379	0,607	306,134
(s, S)	FCFS	89|79|65|47|37	90|106|106|126|151	20,002	203,068	70,200	1,375	0,625	295,270
(s, nQ)	FCFS	7|19|1|1|1	139|140|166|167|185	19,281	201,783	80,095	1,388	0,603	303,149
(s, S)	LWQ	57|61|73|76|79	107|109|122|126|129	18,559	199,137	73,626	1,401	0,599	293,321
(s, nQ)	LWQ	1|20|1|1|1	137|150|181|180|193	18,486	195,845	87,500	1,412	0,588	303,819
(s, S)	LWVQ	58|47|65|55|60	59|50|69|71|95	26,259	191,788	38,032	1,269	0,731	258,079
(s, S)	Random	40|9|1|1|1	82|117|125|134|166	19,946	199,386	72,478	1,377	0,617	293,804
(s, nQ)	Random	14|1|1|1|1	116|138|139|154|168	21,284	203,041	72,061	1,353	0,637	298,377

Tab. 6.16.: *Ergebnisse der Tests für 5 Produkte (2 Fertigungseinheiten, $scst^{voll}$, Tabusuche)*

Für die optimale Lösung (Kombination der LF-Strategie (s, S) und der R-Strategie LWVQ) ist in Abb. 6.20 (b) der Verlauf der abgewiesenen Kunden pro Zeiteinheit dargestellt. Aus dieser wird erkennbar, dass die Anzahl der abgewiesenen Kunden pro Zeiteinheit in etwa um die Hälfte abnimmt, obwohl die Fertigungsrate jeder Fertigungseinheit sinkt und die Umrüstzeiten der einzelnen Fertigungseinheiten unverändert bleiben. Es werden im Vergleich zum Fünf-Produkt-Modell mit einer Fertigungseinheit (Abb. 6.20 (a)) nur noch etwa ein Viertel der ankommenden Kunden abgewiesen. Das ist eine Reduzierung der abgewiesenen Kunden um die Hälfte. Dies bedeutet, dass sowohl detailliertere Informationen im Bezug auf den Fertigungsprozess wie auch die parallele Fertigung unterschiedlicher Produkte eine positive Auswirkung auf die Anzahl der abgewiesenen Kunden und letztendlich auch auf die Gesamtkostenfunktion haben.

Fünf-Produkt-Modell mit zwei Fertigungseinheiten ($scst^{halb}$). Das Fünf-Produkt-Modell mit zwei Fertigungseinheiten und auf die Hälfte reduzierten Umrüstzeiten und -kosten $sc_{ijk}^{halb} = \frac{sc_{ij}^{voll}}{2}, \forall k$ und $st_{ijk}^{halb} = \frac{st_{ij}^{voll}}{2}, \forall k$ stellt den Ausgangspunkt für die Untersuchung dar, bei welcher wiederum die Auswirkungen detaillierterer Informationen bzgl. des Fertigungsprozesses gezeigt werden sollen. Dabei steht die „Aufteilung der Fertigungseinheit“ im Mittelpunkt, weil eine Fertigungseinheit in zwei Fertigungseinheiten mit halber Fertigungsrate „aufgeteilt“ wird. Aus einer groben Darstellung mittels einer einzelnen Fertigungseinheit werden zwei Fertigungseinheiten, welche jedoch die bereits angesprochenen reduzierten Fertigungsraten besitzen. Die Ergebnisse dieses Modells werden nicht direkt mit denen

[116] Als Rechentechnik wurden mehrere Intel® Pentium® 4 CPUs mit 3,00 GHz und jeweils 1,00 GB RAM eingesetzt. Zudem kam als Betriebssystem Microsoft® Windows® XP Professional mit Service Pack 2 und .NET-Framework 2.0 zum Einsatz.

des Fünf-Produkt-Modells mit einer Fertigungseinheit vergleichbar sein, weil die beiden eingesetzten Fertigungseinheiten eine parallele Fertigung unterschiedlicher Produkte ermöglichen und somit auch doppelte Produktionskosten verursachen.

Ergebnisse des Fünf-Produkt-Modells mit zwei Fertigungseinheiten ($scst^{halb}$). Für die Untersuchung des Fünf-Produkt-Modells mit zwei Fertigungseinheiten ($scst^{halb}$) wurden wiederum 20 Untersuchungen durchgeführt (10 Untersuchungen mit dem Optimierungsverfahren TS und 10 Untersuchungen mit dem Optimierungsverfahren GA_{ss}). Für die Dauern eines Simulationsexperimentes sowie eines Laufes der simulationsbasierten Optimierung gelten dieselben Informationen, wie zuvor für das Fünf-Produkt-Modell mit einer Fertigungseinheit.

Die Untersuchungsergebnisse für das dominante Optimierungsverfahren TS sind in Tab. 6.17 dargestellt. Eine genauere Betrachtung dieser Ergebnisse zeigt, dass die dynamischen R-Strategien die statischen R-Strategien sowie die LF-Strategie (s, S) die LF-Strategie (s, nQ) dominieren.

Tab. 6.17

LFS	RS	$\vec{s}_{opt}$	$\vec{Q}_{opt}$ / $\vec{S}_{opt}$	RC(π)	WC(π)	HC(π)	PC(π)	SC(π)	C(π)
(s, S)	Cyclic	41|26|1|1|1	60|74|107|103|123	10,493	159,646	72,635	1,539	0,443	244,757
(s, nQ)	Cyclic	1|1|1|1|1	122|117|138|136|136	10,675	155,319	88,467	1,536	0,422	256,419
(s, S)	FCFS	39|26|1|1|1	59|72|93|113|122	9,965	164,659	65,439	1,548	0,436	242,048
(s, nQ)	FCFS	4|1|1|1|1	104|110|128|131|152	9,857	161,500	79,186	1,550	0,415	252,507
(s, S)	LWQ	53|43|42|44|1	61|73|91|91|125	9,323	160,030	68,830	1,559	0,438	240,180
(s, nQ)	LWQ	1|2|1|1|1	107|116|142|140|148	9,761	160,124	82,341	1,552	0,411	254,190
(s, S)	LWVQ	46|43|42|34|45	49|46|60|65|71	13,543	150,940	43,453	1,487	0,513	209,936
(s, nQ)	LWVQ	247|263|17|1|206	1|1|1|1|1	84,855	134,760	0,000	0,260	1,740	221,614
(s, S)	Random	32|15|1|1|1	62|65|74|102|106	11,399	158,784	64,697	1,524	0,455	236,858
(s, nQ)	Random	6|1|1|1|1	73|107|112|114|123	12,494	158,012	70,154	1,505	0,463	242,628

Tab. 6.17.: *Ergebnisse der Tests für 5 Produkte (2 Fertigungseinheiten, $scst^{halb}$, Tabusuche)*

Der Verlauf der abgewiesenen Kunden pro Zeiteinheit für die optimale Lösung (Kombination der LF-Strategie (s, S) und der R-Strategie LWVQ) ist in Abb. 6.20 (c) dargestellt. Darin ist erkennbar, dass für die optimale Lösung die Anzahl der abgewiesenen Kunden auf etwa ein Achtel gesunken ist.

Fünf-Produkt-Modell mit vier Fertigungseinheiten ($scst^{viertel}$). Für das Untersuchungsziel des Fünf-Produkt-Modells mit vier Fertigungseinheiten ($scst^{viertel}$) gilt dasselbe wie für das Fünf-Produkt-Modell mit zwei Fertigungseinheiten ($scst^{halb}$). Es dient zur Untersuchung der Existenz weiterer Fertigungseinheiten für genauere Informationen bzgl. des Fertigungsprozesses, wobei in diesem Modell jede Fertigungseinheit mit einer Fertigungsrate von $\mu_{ik}^{viertel} = \frac{\mu_{ik}^{voll}}{4}$ und Umrüstzeiten bzw.-kosten von $sc_{ijk}^{viertel} = \frac{sc_{ij}^{voll}}{4}$, $\forall k$ bzw. $st_{ijk}^{viertel} = \frac{st_{ij}^{voll}}{4}$, $\forall k$ produziert.

Ergebnisse des Fünf-Produkt-Modells mit vier Fertigungseinheiten ($scst^{viertel}$). Die Ergebnisse in Tab. 6.18 stellen die 10 Untersuchungsergebnisse dar, welche mittels des Optimierungsverfahrens TS ermittelt wurden. Wie zuvor beim Fünf-Produkt-Modell mit zwei Fertigungseinheiten wurden zusätzlich noch 10 Untersuchungen mit dem Optimierungsverfahren GA_{ss} durchgeführt, welche jedoch nicht dominant

gegenüber denen des Optimierungsverfahren TS waren und daher nicht dargestellt sind.

Die Betrachtung der Ergebnisse aus Tab. 6.18 zeigt keine Dominanz der dynamischen R-Strategien gegenüber den statischen R-Strategien, jedoch eine Dominanz der LF-Strategie (s, S) gegenüber der LF-Strategie (s, nQ).

Tab. 6.18

LFS	RS	$\vec{s}_{opt}$	$\vec{Q}_{opt}$ / $\vec{S}_{opt}$	$RC(\pi)$	$WC(\pi)$	$HC(\pi)$	$PC(\pi)$	$SC(\pi)$	$C(\pi)$
(s, S)	Cyclic	51|1|1|1|1	52|120|124|149|157	7,793	153,602	73,236	3,172	0,752	238,555
(s, nQ)	Cyclic	20|1|1|1|1	74|135|141|175|189	8,149	155,891	76,515	3,159	0,667	244,380
(s, S)	FCFS	43|10|1|1|1	48|107|127|160|178	7,767	157,668	71,093	3,172	0,720	240,419
(s, nQ)	FCFS	14|7|3|1|1	91|125|137|177|190	8,185	161,535	72,163	3,158	0,670	245,712
(s, S)	LWQ	48|14|1|1|1	53|108|143|160|171	8,073	156,495	74,104	3,162	0,735	242,569
(s, nQ)	LWQ	23|11|1|1|1	89|113|141|182|194	8,605	160,238	74,658	3,144	0,697	247,342
(s, S)	LWVQ	56|25|1|1|1	58|74|115|143|169	9,162	155,539	64,189	3,124	0,817	232,831
(s, nQ)	LWVQ	28|4|1|1|1	56|126|133|125|191	8,831	158,112	72,451	3,135	0,717	243,246
(s, S)	Random	43|28|1|1|1	50|79|134|156|177	8,409	154,561	73,050	3,152	0,752	239,924
(s, nQ)	Random	28|4|1|1|1	56|126|133|166|191	8,831	158,112	72,451	3,135	0,717	243,246

Tab. 6.18.: *Ergebnisse der Tests für 5 Produkte (4 Fertigungseinheiten, $scst^{viertel}$, Tabusuche)*

In Abb. 6.20 (d) ist dargestellt, wie sich der Verlauf der abgewiesenen Kunden pro Zeiteinheit für die optimale Lösung bei der Kombination der LF-Strategie (s, S) und der R-Strategie LWVQ im Verlaufe eines Simulationslaufes gestaltet. Erkennbar ist, dass die Anzahl der abgewiesenen Kunden im Vergleich zu den Abbildungen 6.20 (a) - (c) wiederum abgenommen hat. Dies ist auf die gestiegene parallele Verarbeitungsmöglichkeit durch zwei weitere Fertigungseinheiten zurückzuführen. Die Reduktion der Produktionsrate im Vergleich zum Zwei-Fertigungseinheiten-Modell scheint dabei eine eher geringe Auswirkung zu haben. Diese Behauptung müsste allerdings noch genauer untersucht werden.

Tab. 6.19

		(1 FE)		($scst^{voll}$, 2 FE)		($scst^{halb}$, 2 FE)		($scst^{viertel}$, 4 FE)	
LFS	RS	$\frac{PC(\pi)}{1}$	$\frac{SC(\pi)}{1}$	$\frac{PC(\pi)}{2}$	$\frac{SC(\pi)}{2}$	$\frac{PC(\pi)}{2}$	$\frac{SC(\pi)}{2}$	$\frac{PC(\pi)}{4}$	$\frac{SC(\pi)}{4}$
(s, S)	Cyclic	0,371	0,629	1,367	0,633	1,539	0,443	3,172	0,752
(s, nQ)	Cyclic	0,527	0,473	1,379	0,607	1,536	0,422	3,159	0,667
(s, S)	FCFS	0,532	0,468	1,375	0,625	1,548	0,436	3,172	0,720
(s, nQ)	FCFS	0,548	0,452	1,388	0,603	1,550	0,415	3,158	0,670
(s, S)	LWQ	0,546	0,454	1,401	0,599	1,559	0,438	3,162	0,735
(s, nQ)	LWQ	0,530	0,470	1,412	0,588	1,552	0,411	3,144	0,697
(s, S)	LWVQ	0,360	0,640	1,269	0,731	1,487	0,513	3,124	0,817
(s, nQ)	LWVQ	0,058	0,942	-	-	0,260	1,740	3,135	0,717
(s, S)	Random	0,469	0,531	1,377	0,617	1,524	0,455	3,152	0,752
(s, nQ)	Random	0,446	0,554	1,353	0,637	1,505	0,463	3,135	0,717

Tab. 6.19.: *Ergebnisse für die Umrüst- und Produktionskosten pro Fertigungseinheit für die Tests mit 5 Produkten*

In Tab. 6.19 sind noch einmal die Umrüst- und Produktionskosten ($SC(\pi)$ und $PC(\pi)$) je Fertigungseinheit gegenübergestellt, um die Auswirkungen der einzelnen Modelle untereinander auf die Kosten des Fertigungsprozesses besser deutlich zu machen. Aus den Werten der Tabelle wird erkennbar, dass die genauere Beschreibung des Fertigungsprozesses durch die Möglichkeit der gleichzeitigen Fertigung mehrerer verschiedener Produkte auch genauere Ergebnisse nach sich zieht. Zudem wirkt sich die Abnahme der Umrüstkosten pro Umrüstvorgang auch reduzierend auf die mittleren Umrüstkosten $SC(\pi)$ aus.

6.5.4. Vergleichsmodelle von Markowitz, Reimann und Wein

Modellparameter. In [MRW95] und [MRW00] geben Markowitz, Reimann und Wein ein Modell, kurz MRW-Modell, an, welches vergleichbar mit dem Modelltyp 1 ist. Sie beschreiben jedoch ein SELSP und kein CSLSP, worauf der Modelltyp 1 im eigentlichen Sinne abzielt. Allerdings kann bei Verzicht auf die Kapazitätsbeschränkung des Lagers ein SELSP problemlos mittels Modellen des Modelltyps 1 nachgebildet werden. Auf Grund dieser Tatsache und der weitreichenden Untersuchungsergebnisse des MRW-Modells wurden innerhalb dieser Arbeit Vergleichsergebnisse angestrebt.

Innerhalb des MRW-Modelles werden jeweils mehrere Umrüstkosten- und Umrüstzeitenproblem für je zwei und fünf Produkte betrachtet, welche sowohl symmetrische und asymmetrische Umrüstkosten bzw. Umrüstzeiten beinhalten. Die Modelle werden nachfolgend kurz als MRW-Zwei-Produkt-Modell und MRW-Fünf-Produkt-Modell bezeichnet. Symmetrisch bedeutet in diesem Falle, dass für alle Produkte dieselben Umrüstkosten oder dieselben Umrüstzeiten verwendet werden. Bei den asymmetrischen MRW-Modellen sind die Umrüstkosten oder -zeiten sowie die Lagerkosten produktabhängig. Höhere Produktnummern werfen höhere Kosten oder Zeiten auf. Des Weiteren werden in den MRW-Modellen jeweils niedrige und hohe Umrüstkosten oder -zeiten zur Untersuchung ihrer eingesetzten optimalen Lösungsmethodik herangezogen. Auf Grund der Bestimmung der optimalen Lösung und nicht nur einer suboptimalen Lösung hinreichender Güte müssen sie trotz des Einsatzes von beschränkten Kundenwarteschlangen für nichtbefriedigten Bedarf auf die Betrachtung von Abweis- und Wartekosten verzichten. Eine Berücksichtigung der Produktionskosten erfolgt ebenfalls nicht. Auf weitere Details der MRW-Modelle und deren Modellparameter wird auf Grund der vielen MRW-Modellbeispiele verzichtet. Es soll jedoch ein aussagekräftiger Überblick über die Untersuchungsergebnisse der Vergleichsmodelle des Modelltyps 1 betrachtet werden.

Ergebnisse. Die angegebenen Ergebnisse beruhen auf insgesamt 816 Experimenten für das MRW-Zwei-Produkt-Modell und 240 Untersuchungen für das MRW-Fünf-Produkt-Modell, welche sich multiplikativ wie folgt zusammensetzen:

- 6 R-Strategien (Cyclic, Dynamic, FCFS, LWQ, LWVQ und Random),
- 2 LF-Strategien ((s, nQ) und (s, S)),
- 2 Optimierungsverfahren (TS und GA_{ss}) sowie
- 34 Untersuchungsmodelle beim MRW-Zwei-Produkt-Modell und 10 Untersuchungsmodelle beim MRW-Fünf-Produkt-Modell.

Insgesamt wurden während eines Laufes der simulationsbasierten Optimierung 10000 Lösungen (Simulationsexperimente) betrachtet. Ein Simulationsexperiment dauerte dabei zwischen fünf und zwanzig Sekunden, wodurch ein Lauf der simulationsbasierten Optimierung im Mittel 24 Stunden dauerte. Die Zeitspanne für ein Simulationsexperiment war bspw. abhängig von der Höhe der Werte für die Q_i bei

der LF-Strategie (s, nQ). In der Regel entstanden für kleinere Werte von Q_i und die Untersuchung der R-Strategien LWQ und LWVQ längere Simulationsdauern. Durch den Einsatz des Epsilon-Abbruch-Verfahrens konnten diese Dauern für ein Simulationsexperiment jedoch reduziert werden, ohne die Güte der Simulationsergebnisse zu stark einzuschränken. Ein Verzicht auf das Epsilon-Abbruch-Verfahren hätte die Dauer für ein Simulationsexperiment von teilweise über 3 Minuten und i. d. R. Ausgabewerte, welche 10- bis 100-mal schlechter als die optimale Lösung sind, nach sich gezogen.

Der numerische Vergleich der Ergebnisse dieser Arbeit mit denen von Markowitz, Reimann und Wein weist geringfügige Differenzen auf. Teilweise fallen die Ergebnisse dieser Arbeit höher/schlechter und teilweise niedriger/besser aus. Dies ist auf die Betrachtung der Umrüstkosten und -zeiten zurückzuführen. Bei den Modellen von Markowitz, Reimann und Wein werden bspw. die Umrüstkosten pro Fertigungszyklus betrachtet und nicht wie bei den hier beschriebenen Modellen die Umrüstkosten produktabhängig angegeben. Das bedeutet, dass bei den vom Autor angewendeten Vergleichsmodellen die Umrüstkosten des Fertigungszyklus gleichmäßig auf die einzelnen Produkte aufgeteilt werden. Im Mittel sind die im Rahmen dieser Arbeit erzielten Ergebnisse besser als die von Markowitz, Reimann und Wein.

Die Ergebnisse der Untersuchungen sind in den Abbildungen 6.21 und 6.22 mittels Tortendiagrammen (siehe [FKPT04] und [Sch03]) veranschaulicht. Die Ergebnisse in den jeweils linken Tortendiagrammen („Alle Ergebnisse“) kommen wie folgt zu Stande: ausgehend von der ermittelten besten und schlechtesten optimalen Lösung eines Untersuchungsmodells (kleinster Zielfunktionswert eines Untersuchungsmodells) wird die relative Abweichung der restlichen optimalen Lösungen bestimmt. Diese Werte werden aufsummiert und über alle Untersuchungsmodelle gemittelt.

Abb. 6.21

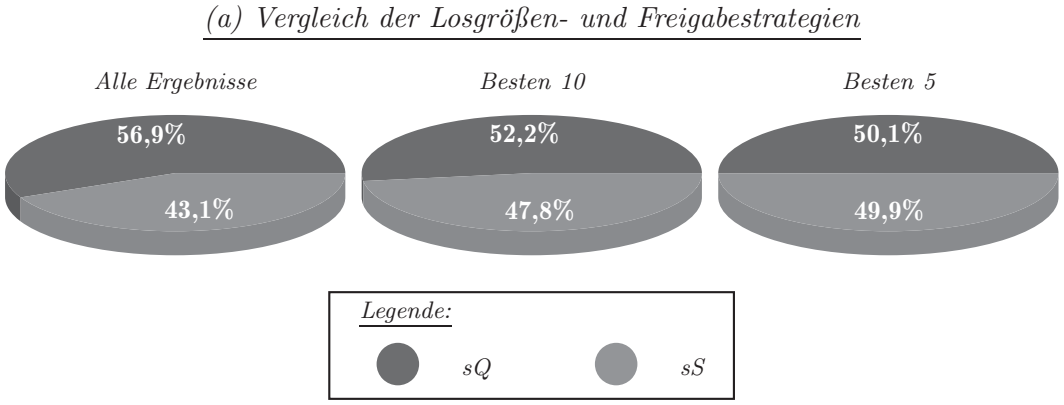


Abb. 6.21.: *Vergleich der Ergebnisse der Modelle von Markowitz, Reimann und Wein (1/2)*

Abb. 6.22

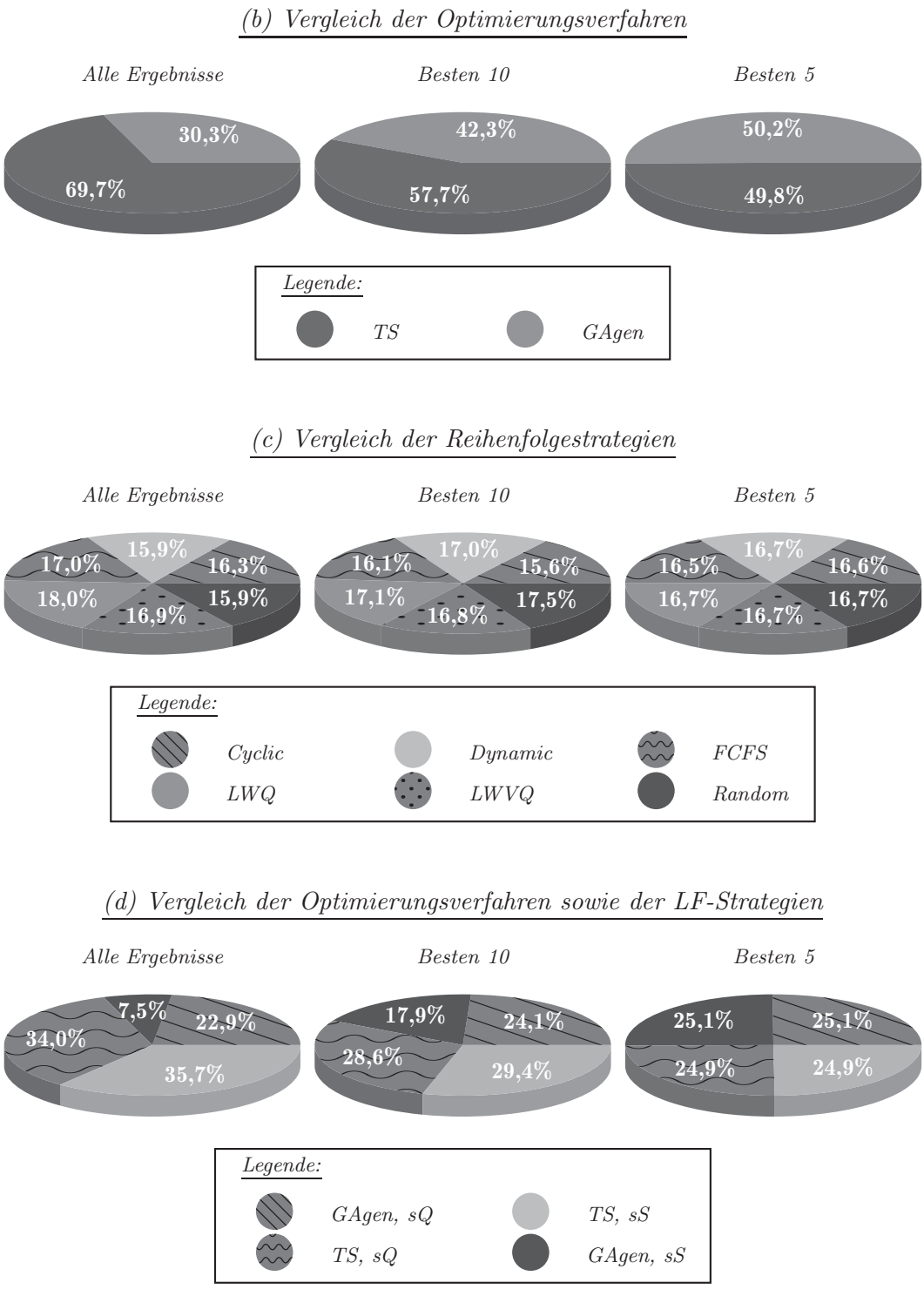


Abb. 6.22.: *Vergleich der Ergebnisse der Modelle von Markowitz, Reimann und Wein (2/2)*

Die Abbildungen 6.21 und 6.22 verdeutlichen die besten 5 Ergebnisse eines Untersuchungsmodells („Besten 5“, jeweils rechtes Tortendiagramm). Unabhängig vom eingesetzen Optimierungsverfahren, der angewandten R-Strategie sowie der eingesetzten LF-Strategie kann nicht gesagt werden, welches Optimierungsverfahren bzw. welche LF- oder R-Strategie die Beste ist. Sie existieren nahezu gleichberechtigt nebeneinander und können jeweils individuell die optimale Lösung eines Untersuchungsmodells bestimmen, ohne dass eine direkte Abhängigkeit von den Modellparameter erkennbar ist. Betrachtet man die besten 10 Ergebnisse, werden bereits kleine Unterschiede sichtbar, welche bei der Betrachtung aller Ergebnisse noch deutlicher werden. Dies lässt den Schluss zu, dass nur bei Betrachtung aller Kombinationsmöglichkeiten aus Optimierungsverfahren, R-Strategie und LF-Strategie für ein Untersuchungsmodell die optimale Lösung gefunden werden kann.

Für die Darstellungen der Ergebnisse in den Abbildungen 6.22 (c) und 6.22 (d) wären auch andere Kombinationsmöglichkeiten denkbar, welche jedoch nicht zwingend aussagekräftiger sind.

6.6. Zusammenfassung

In diesem Kapitel wurde eine Nutzungsmöglichkeit des Softwaresystems CAOS aufgezeigt. Dies geschah am Beispiel der softwaretechnischen Entwicklung von Simulationsmodellen zu verschiedenen Modelltypen für Produktions- und Lagerhaltungssysteme. Bevor sich jedoch mit diesen näher auseinander gesetzt wurde, erfolgte eine allgemeine Betrachtung von Problemen zu Produktions- und Lagerhaltungssystemen u. a. durch einen entsprechenden Literaturüberblick. Dabei wurde erkennbar, dass die real-existierenden Systeme sich mitunter sehr komplex gestalten können und somit einer hinreichend genauen Modellierung bedürfen, um die Realität möglichst exakt wiederzugeben.

Einige sinnvolle Modellparameter und Entscheidungsvariablen wurden näher betrachtet und zur Umsetzung innerhalb der untersuchten Modelltypen ausgewählt. Dies waren u. a. der Kundenbedarfs- und Produktionsprozess sowie Warteschlangen für nicht befriedigten Bedarf und für die Produktion freigegebener Lose mit Fertigungsaufträgen. Als Entscheidungsmöglichkeiten wurden die Losgrößen- und Freigabeentscheidung, die Reihenfolgeentscheidung sowie die Zuordnungsentscheidung betrachtet, welche letztendlich anhand verschiedener Strategien realisiert wurden. Ein weiterer Betrachtungspunkt der untersuchten Modelle waren die verschiedenen Kostenbestandteile, welche für verschiedene Bestandteile des Modells anfielen und zur Bildung der Zielfunktion ausgewählt wurden. Ziel war es dabei, die Gesamtkosten des Modells zu minimieren.

An dieser Stelle sieht der Autor die Möglichkeit des Einsatzes der mehrkriteriellen Optimierung. Weitere Untersuchungen mittels der im CAOS implementierten mehrkriteriellen, heuristischen Optimierungsverfahren könnten weitere Erkenntnisse über die untersuchten Modelle liefern. Bspw. könnte durch die Ermittlung der Menge der Pareto-optimalen Lösungen ersichtlich werden, wie sich die verschieden-

artigen Lösungen im Falle einer mehrkriteriellen Zielfunktion insgesamt gestalten. Nicht nur bei der simulationsbasierten Optimierung kann sich dieses Auffinden der Pareto-optimalen Menge als sinnvoll erweisen. Die mehrkriterielle Optimierung kann immer dann hilfreich sein, wenn ein eindimensionaler Zielfunktionswert die Ergebnisse nicht ausreichend genau charakterisiert oder nur schwer bzw. nicht bestimmbar ist. Aus diesem Grunde wurde auch in den Abschn. 2.2.3, 2.2.5, 2.5, 2.6 und 4.4 näher auf die Pareto-Optimalität, die mehrkriterielle Optimierung und mehrkriterielle, heuristische Lösungsverfahren eingegangen.

Nach der Vorstellung der einzelnen untersuchten Modelltypen des CSLSP (ohne Kapazitätsbeschränkung des Lagers) in der neuen vereinheitlichten visuellen Beschreibungssprache EcoSyL erfolgte die Konzentration auf die verschiedenen Untersuchungsrichtungen. Diese insgesamt über 1100 Untersuchungen lieferten verschiedene Ergebnisse. Zunächst ergaben die Voruntersuchungen sowie weitere anfängliche Untersuchungen, dass das Optimierungsverfahren TS oftmals dominant ist. Diese Aussage ist bei Betrachtung der Kapazitätsbeschränkung des Lagers mitunter nicht mehr zutreffend. Als weiteres Ergebnis wurde festgestellt, dass keine der Reihenfolgestrategien dominant ist. Zudem war auch bei den Losgrößen- und Freigabestrategien keine Dominanz einer LF-Strategie erkennbar. Allerdings kann festgehalten werden, dass bei Betrachtung aller Kostenbestandteile beim Modelltyp 1 eine Dominanz der LF-Strategie (s, S) gegenüber der LF-Strategie (s, nQ) erkennbar ist.

Kapitel 7. Resümee und Ausblick

Inhalt

7.1 Resümee 235
7.2 Ausblick auf zukünftige Untersuchungsrichtungen 238

7.1. Resümee

Innerhalb dieser Arbeit wurde sich mit zwei Themenkomplexen beschäftigt. Im Themenkomplex 1 wurde die simulationsbasierte Optimierung als Lösungsverfahren für (beliebig) komplexe Problemstellungen betrachtet. Diese fand im Themenkomplex 2, welcher sich mit der *Untersuchung verschiedener Modelle zu Produktions- und Lagerhaltungssystemen* beschäftigt, ihre Anwendung. Auf Grund der Nachteile von kommerziellen Softwaresystemen für Simulation und Optimierung entstand zudem die Forderung nach der *Schaffung eines Softwaresystems zur Durchführung von Läufen der simulationsbasierten Optimierung*. Dieses neue Softwaresystem wurde ebenfalls im ersten Themenkomplex näher betrachtet.

Die Verwendung der *simulationsbasierten Optimierung* machte es notwendig, sich mit den einzelnen Teilproblemen dieses Lösungsverfahrens zu beschäftigen. Das waren im Wesentlichen die beiden Komplexe der *Simulation* und der *Optimierung*. Voraussetzung für deren Anwendung ist eine rechnerinterne Modelldarstellung eines zu untersuchenden Systems. Dafür wurde eine *neue Modelldarstellung* entwickelt, welche speziell auf die Durchführung von Läufen der simulationsbasierten Optimierung ausgerichtet ist und dadurch den Begriff der *modellgestützten, simulationsbasierten Optimierung* prägt (siehe Kap. 2). Sie sieht die Verwendung einer beliebig in andere Sichtweisen überführbaren Baumstruktur vor. Deren wesentliches Merkmal ist neben der Strukturierung innerhalb eines Baumes die Unterteilung eines Modells in *Modelldeklaration* und *-definition*. Diese Trennung sollte zum einen die Teilung der Aufgaben während der Modellierung verdeutlichen und bot zum anderen eine Möglichkeit, die Eingabe einer Modelldefinition robust zu gestalten. Die Einführung dieser Modelldarstellung und deren Trennung in Modelldeklaration und -definition erwies sich als sehr hilfreich bei der softwaretechnischen Abbildung verschiedenster Modelle. Zum besseren Verständnis der Vorgehenswei-

sen bei der modellgestützten, simulationsbasierten Optimierung wurden im Kap. 2 auch grundlegende Lösungsverfahren und notwendige Definitionen zur betrachteten Problematik angegeben.

Die *Simulation* als Teilkomplex der simulationsbasierten Optimierung bietet die Möglichkeit der Untersuchung von Modellen, welche *stochastischen Einflüssen* unterliegen (siehe Kap. 3). Jedoch haben gerade diese Einflüsse direkte Auswirkung auf die Ausgabewerte eines Simulationslaufes. Dies führt bspw. zur Aufteilung eines Simulationslaufes in eine *transiente* und *stationäre Phase*. Es wurde an einem Beispiel verdeutlicht, dass deren Wahl einen wesentlichen Einfluss auf die *Güte der Ausgabewerte* und auf die *Dauer eines Simulationslaufes* hat. Eine getroffene Feststellung war, dass das Ende der transienten Phase i. Allg. nur hinreichend genau bestimmt werden kann. Die Bestimmung oder Festlegung der Dauer der stationären Phase bot im Zusammenhang mit der simulationsbasierten Optimierung die Möglichkeit der Verkürzung eines Simulationslaufes, bei welchem „schlechtere" Zielfunktionswerte als die „bisherigen" zu erwarten waren. Diese neue heuristische Methodik wurde als *Epsilon-Abbruch-Verfahren der simulationsbasierten Optimierung* bezeichnet. Darüber hinaus wurde als andere Möglichkeit zur Verkürzung der Dauer eines Simulationslaufes auch die *verteilte und parallele Simulation* betrachtet. Die verteilte und parallele Simulation ist jedoch stark abhängig vom verwendeten Simulationsmodell, wodurch keine allgemein gültigen Aussagen über deren Güte getroffen werden können. Allerdings sollten zukünftig entwickelte Softwaresysteme die verschiedenen Methodiken der verteilten und parallelen Simulation bereit stellen, um einem Anwender die Entscheidung über die Verteilung seines Simulationsmodells prinzipiell zu überlassen.

Auf den zweiten Teilkomplex der simulationsbasierten Optimierung, die Optimierung, wurde innerhalb des Kap. 4 eingegangen. Es wurde gezeigt, dass es nur ein exaktes Optimierungsverfahren, das Verfahren der Vollständigen Aufzählung, gibt, welches dem Anspruch der problemunabhängigen Verwendbarkeit sowie der Ermittlung einer optimalen Problemlösung genügt. Aus diesem Grund wurde sich auf den Einsatz heuristischer Suchverfahren konzentriert, welche es ermöglichen, eine zumindest suboptimale Problemlösung innerhalb einer vorgegebenen Zeitspanne zu bestimmen. In diesem Zusammenhang wurden im Kap. 4 auch verschiedene Möglichkeiten zur Verkürzung eines Optimierungslaufes vorgestellt. Als eine der hierfür am sinnvollsten Methodiken wird die verteilte und parallele Optimierung, speziell die Verteilung der Lösungsvektoren, angesehen. Grund dafür ist, dass sich die Verteilung der Lösungsvektoren für die meisten Optimierungsverfahren problemunabhängig und ohne größeren programmiertechnischen Aufwand gestalten lässt. Hybride Optimierungsverfahren stellen eine andere Möglichkeit dar, um eine Verkürzung der Optimierungsdauer zu erreichen. Diese Verkürzung ist jedoch im Gegensatz zur Verkürzung, welche durch die Methodiken der verteilten und parallelen Optimierung erreicht werden kann, nicht zwingend garantiert. Innerhalb des Kap. 4 wurden zudem ausgewählte Verfahren zur mehrkriteriellen Optimierung kurz betrachtet. Diese mehrkriteriellen Optimierungsverfahren ermöglichen es einem Anwender, nicht nur eine (optimale) Lösung sondern mehrere Lösungsalternativen zu bestimmen. Aus diesen kann der Anwender dann die für ihn optimale

Lösung auswählen oder mittels eines geeigneten Entscheidungsfindungsverfahrens bestimmen lassen. Die mehrkriteriellen Optimierungsverfahren stellen dabei den aktuellen Stand der Forschung dar. Sie sind durch die Menge von Auswahlmöglichkeiten, welche sie bereit stellen, ein wesentliches Gütekriterium moderner heuristischer Suchverfahren.

Als Konsequenz auf die hohen Anschaffungskosten, die fehlenden Verteilungsmöglichkeiten der implementierten Simulations- und Optimierungsverfahren sowie die fehlende Umsetzung moderner ein- und mehrkriterieller Optimierungsverfahren innerhalb aktueller Softwaresysteme für Simulation und Optimierung entstand das Softwaresystem `CAOS`. Es orientiert sich weitestgehend am aktuellen Stand der Forschung auf dem Gebiet der simulationsbasierten Optimierung und setzt dazu wesentliche der innerhalb dieser Arbeit vorgestellten Konzepte und Verfahren zur simulationsbasierten Optimierung um. Durch die Verwendung von Bibliotheken nach außen und die interne Nutzung objektorientierter Programmierung ist `CAOS` so modular gestaltet, dass es sich zu jeder Zeit an neue Bedürfnisse aus Wissenschaft und Praxis anpassen lässt. Die vorhandene Struktur von `CAOS` kann dafür i. Allg. beibehalten werden. Letzten Endes besitzt die zur Realisierung von `CAOS` verwendete objektorientierte Programmiersprache `C#` eine Schlüsselrolle bei moderner Softwareentwicklung. Um dies zu verdeutlichen, war das Kap. 5 im Wesentlichen von der Darstellung der einzelnen erstellten Bibliotheken sowie der Darstellung von ausgewählten wesentlichen Klassenmethoden geprägt. Die graphische Darstellung erfolgte mittels der (modernen) vereinheitlichten Modellierungssprache UML 2.0. Im `CAOS` wurde dabei als wesentliches Ziel die Bereitstellung eines Softwaresystems zur simulationsbasierten Optimierung verfolgt, welches stets die Verwendung weitestgehend allgemeingültiger und robuster Methoden für Simulation und Optimierung in den Vordergrund stellt.

Im Kap. 6 wurde näher auf eine Anwendung des entwickelten Softwaresystems `CAOS` und den damit verbundenen Problemstellungen eingegangen. Diesen liegt das CSLSP zugrunde, welches durch verschiedene Charakteristiken, wie Kapazitätsrestriktionen für die Fertigung und das Lager, der Fertigung mehrerer Produkte sowie der Existenz verschiedener stochastischer Einflüsse gekennzeichnet wird. Das CSLSP vereint dazu verschiedene ähnliche Problemstellungen wie das ELSP und das CSLP in sich. Zur Erstellung von Untersuchungsmodellen für das CSLSP wurden drei verschiedene Modelltypen entwickelt. Diese Modelltypen sind innerhalb dieser Arbeit in der neuen visuellen Modellierungssprache EcoSyL beschrieben. Während im ersten Modelltyp nur eine Fertigungseinheit zum Einsatz kommt, werden im zweiten und dritten Modelltyp beliebig viele Fertigungseinheiten verwendet. Solche Modelltypen, welche stochastische Einflüsse, mehrere Produkte, mehrere Fertigungseinheiten, Kapazitätsbeschränkungen und produktabhängige Umrüstzeiten und -kosten vereinen und dabei verschiedene Kostenbestandteile betrachten, wurden bisher kaum untersucht. Die Bestimmung einer optimalen Lösung wird dabei als $\mathcal{NP}$-schwere Problemstellung aufgezeigt. Aus diesem Grund kamen heuristische Suchverfahren zur Problemlösung zur Bestimmung einer kostenoptimalen Lösung zum Einsatz. Ziel der Untersuchungen war es, verschiedene qualitative Aussagen über die eingesetzten Reihenfolge-, Losgrößen- und Freigabestrategien bei den un-

terschiedlichen Untersuchungsmodelle der einzelnen Modelltypen geben zu können. Zudem konnte anhand dieser Aussagen ein Vergleich der einzelnen Modelltypen untereinander stattfinden.

7.2. Ausblick auf zukünftige Untersuchungsrichtungen

In den einzelnen Kapiteln wurden bereits an bestimmten Stellen verschiedene zukünftige Untersuchungsrichtungen angedeutet. Dabei kann unterschieden werden, in welchem Wissenschaftsbereich sich das jeweilige Teilproblem ansiedeln lässt. Im Wesentlichen werden die Bereiche Informatik, Mathematik, Operationsforschung [dt.; operations research [engl.]] und Ökonomie von tieferer Bedeutung sein. Einige der möglichen zukünftigen Untersuchungsrichtungen, welche sich an dieser Arbeit anschließen könnten, werden nachfolgend angeführt.

Auf dem Gebiet der Informatik kann u. a. nach weiteren verbesserten Möglichkeiten zur rechnerinternen Modellabbildung eines mathematischen Modells gesucht werden. Es bietet sich bspw. an, verschiedene Transformationen der hierarchischen Baumstruktur eines Modells bereitzustellen, um einem Nutzer weitere visuelle Unterstützung bei der Eingabe einer Modelldefinition zu geben. Im Softwaresystem `CAOS` sind dafür bereits erste Ansätze enthalten, welche jedoch innerhalb dieser Arbeit nicht näher betrachtet wurden, weil sie deutlich über den Rahmen dieser Arbeit hinaus gehen würden. Eine sinnvolle Erweiterung des gewählten Modellierungsansatzes ist die Einführung von virtuellen Gruppen und Untergruppen (siehe Abschn. 2.4.3), welche die Gruppierung einzelner Elemente oder Attribute ermöglicht. Zudem erweist es sich mitunter als sinnvoll in Anlehnung an das objektorientierte Programmierparadigma, bei der Programmierung verschiedene Basisdeklarationen für Modellelemente bereitzustellen, welche beliebig erweitert werden können. Hierfür sollte dann auf eine globale Datenbank zurückgegriffen werden, in welcher die Basisdeklarationen für die Modellelemente und deren individuelle Erweiterungen abgelegt sind.

Als zusätzlichen Erweiterungsansatz sieht der Autor die Auslagerung der Modelldeklaration aus dem Implementierungsteil des Simulators vor, um so eine weitere Trennungsebene anzustreben, auf welcher die detailliertere Trennung von Softwareersteller und Ersteller der Modelldeklaration vollzogen wird. Eine Modelldeklaration könnte dadurch in eine externe Datei, bspw. eine Datenbank- oder XML-Datei, ausgelagert werden. Diese Dateien mit den Modelldeklarationen und -definitionen könnten dann um beliebige weitere, für zukünftige Anforderungen notwendige Elemente und Attribute erweitert werden. Es könnte u. a. eine Verankerung des zu verwendenden Zufallszahlengenerators an den Stellen erfolgen, wo im Modell Attribute mit Verteilungen von Zufallszahlen auftreten. Zudem könnte die Angabe einer individuellen Berechnungsvorschrift für den Schritt der Bewertung der Ausgabewerte eines Berechners oder Simulators in einer Modelldefinition durch eine entsprechende Erweiterung der Modelldeklaration angestrebt werden. Einem

Modellersteller wäre es somit möglich, den für seine Modelldefinition geeigneten Bewerter zu nutzen.

Aus softwaretechnischer Sicht gestaltet sich auch die Angabe einer rechnerinternen Modellabbildung des Simulationsmodells als hilfreich. Durch einen solchen Ansatz könnte eine weitere Stufe hin zur Allgemeingültigkeit der Simulationsmodelle innerhalb des Softwaresystems `CAOS` und darüber hinaus erreicht werden. Dazu müsste der vorhandene Modellierungsansatz u. a. auch logische Verhaltensweisen eines Simulationsmodells mit abbilden können. Eine andere weitreichende Möglichkeit entsteht durch den Einsatz von Quelltext-Generatoren. Diese können es ermöglichen, die bspw. in Form von externen Dateien vorliegenden Simulationsmodelle auch unabhängig von einem bestimmten Softwaresystem einzusetzen. Diese Modellierungssprache könnte aber über die Grenzen der rechnerinternen Modellabbildung eines Simulationsmodells hinaus auch direkt zur Entwicklung einer Modellierungssprache zur simulationsbasierten Optimierung dienen.

Unter dem Blickpunkt des operations research [engl.] ist mitunter die Betrachtung des gewählten Ansatzes von Bedeutung. Dieser erlaubt es, unterschiedliche Optimierungsverfahren für die verschiedenen Optimierungsprobleme, welche innerhalb eines Modells auftreten können, anzuwenden. Erreicht wird dies durch die Bindung des Optimierungsverfahrens an eine Entscheidungsvariable. In diesem Zusammenhang kann auch die Untersuchung der implementierten mehrkriteriellen Optimierungsverfahren, SPEA 2 und NSGA 2, oder entsprechender Weiterentwicklungen stehen. Auch bei den mehrkriteriellen Optimierungsverfahren sollte die Bindung unterschiedlicher Optimierungsverfahren an verschiedene Entscheidungsvariablen betrachtet werden, um die entstehenden Vor- oder Nachteile dieser Vorgehensweise aufzuzeigen.

Aus Sicht der Ökonomie steht deutlich die Untersuchung weiterer Modellbeispiele für die unterschiedlichen, vorgestellten Modelltypen sowie die Übernahme der erzielten Ergebnisse ins praktische Umfeld im Vordergrund. Dabei ist u. a. auch die weiterführende Untersuchung der Auswirkungen der Kapazitätsbeschränkungen von Produktion und Lager von wesentlichem Interesse. Dabei sollten verschiedene Strategien zur Aufteilung des Lagers für die einzelnen einzulagernden Produkte betrachtet werden. Hierfür kommen bspw. dynamische Lagerplatzzuordnungsstrategien, welche die Aufteilung des Lagers zustandsabhängig betrachten, und statische Strategien, die im Vorfeld eines Simulationsexperimentes eine Lageraufteilung für die verschiedenen Produkte vornehmen, betrachtet werden. Die zu treffenden Untersuchungen würden u. a. Rückschlüsse daraufhin zu lassen, in welchem Umfang sich der Detaillierungsgrad der jeweiligen Modelle bspw. im Hinblick auf Lieferverzögerungen und Kapazitätsbeschränkungen der Produktion oder des Lagers gestalten muss.

Kapitel A. Auszug aus der Unified Modeling Language (UML)

Inhalt

A.1	*Einführung*	*241*
A.2	*Allgemeine Elemente von UML-Diagrammen*	*243*
A.3	*Verwendete Verhaltensdiagramme der UML*	*243*
A.3.1	*Aktivitätsdiagramme*	*243*
A.3.2	*Anwendungsfalldiagramme*	*245*
A.3.3	*Kommunikationsdiagramme*	*246*
A.4	*Verwendete Strukturdiagramme der UML*	*247*
A.4.1	*Klassendiagramme*	*247*
A.4.2	*Paketdiagramme*	*248*
A.4.3	*Verteilungsdiagramme*	*249*

A.1. Einführung

Die UML (Abkürzung für *Unified Modeling Language* [engl.]) ist eine Sprache zur Visualisierung von Modellen für Softwaresysteme und anderen Systemen, welche nicht direkt auf die Entwicklung von Software abzielen müssen. In der vorliegenden Arbeit dient sie vorrangig zur Spezifikation und Dokumentation der Arbeitsweisen und des internen Aufbaus des Softwaresystems `CAOS`. Darüber hinaus kann die UML aber auch zur Konstruktion von Geschäftsmodellen verwendet werden. Mittels der UML wird es möglich, dass verschiedene Software-Entwickler eine gemeinsame, einheitliche Diskussionsgrundlage beim Entwurf und der Entwicklung des Modells einer Software besitzen. Die UML ist seit November 1997 bei der *Object Management Group (OMG)* standardisiert (siehe Internetseite [Obj07]). Die damalige Version war die Version 1.0. Aktuell liegt die UML in der Version 2.1.1 vom Februar 2007 vor. Im Mai 2007 wurde von der OMG bekannt gegeben, dass bereits an der nächsten Version 2.2[117] gearbeitet wird.

[117] Ein Veröffentlichungsdatum ist zum Zeitpunkt der Erstellung dieser Arbeit noch nicht bekannt.

Die vorliegende Arbeit stützt sich auf die UML 2.0 aus dem Jahre 2005. Die Gründe dafür sind, dass einerseits ein Großteil der Entwicklung des Softwaresystems `CAOS` nach 2005 stattfand und andererseits die Sprachunterversionen seit UML 2.0 für die in der vorliegenden Arbeit angewandten Diagramme ohne Bedeutung sind. Hinzu kommt, dass die UML besonders gut für die Modellierung im Zusammenhang mit der Softwareerstellung unter Nutzung objektorientierter Programmierung geeignet ist. Die Quelltexte vom `CAOS` sind komplett objektorientiert.

Die Literatur zur UML 2.0 ist sehr weitreichend. Im Wesentlichen stützt sich der Autor auf die Werke von Kecher (siehe [Kec06]) und Balzert (siehe [Bal05]).

In den nachfolgenden Abschnitten werden nur die Diagrammtypen der UML betrachtet, welche auch in der vorliegenden Arbeit ihre Verwendung fanden. Prinzipiell besitzt die UML verschiedene *Struktur-* und *Verhaltensdiagramme*. Zu den Strukturdiagrammen gehören

- die *Klassendiagramme* (*),
- die *Komponentendiagramme*,
- die *Kompositionsstrukturdiagramme*,
- die *Objektdiagramme*,
- die *Paketdiagramme* (*) und
- die *Verteilungsdiagramme* (*).

Die Verhaltensdiagramme lassen sich in

- *Aktivitätsdiagramme* (*),
- *Anwendungsfalldiagramme* (*),
- *Interaktionsübersichtsdiagramme*,
- *Kommunikationsdiagramme* (*),
- *Sequenzdiagramme*,
- *Zeitverlaufsdiagramme* und
- *Zustandsdiagramme*

unterteilen. Die Elemente der mit (*) gekennzeichneten Diagrammtypen werden nachfolgend etwas näher betrachtet. Zuvor wird jedoch noch auf die allgemeinen Elemente eingegangen, welche alle UML-Diagramme gemeinsam haben.

A.2. Allgemeine Elemente von UML-Diagrammen

Als einziges Element, welches in allen UML-Diagrammen mit gleicher Bedeutung existiert, sei das Element „Notiz“ genannt, welches in Tab. A.1 angegeben ist.

Tab. A.1

Element	Erklärung
Notiz	*Notiz* zu einem beliebigen Element von UML-Diagrammen mit zugehöriger Verbindungslinie

Tab. A.1.: *Allgemeine Elemente von UML-Diagrammen*

A.3. Verwendete Verhaltensdiagramme der UML

A.3.1. Aktivitätsdiagramme

Ein Aktivitätsdiagramm visualisiert die Vernetzung der elementaren Aktivitäten. Die Verbindungen zwischen den Aktivitäten stellen dabei Kontroll- und Datenflüsse dar. Ziel eines Aktivitätsdiagrammes ist die Darstellung des Ablaufes eines Anwendungsfalls oder einer zu beschreibenden Teilaktivität. Demzufolge können Aktivitätsdiagramme auch zur Modellierung aller Aktivitäten innerhalb eines Systems genutzt werden. Eine für diese Arbeit relevante Teilmenge der Elemente eines Aktivitätsdiagrammes ist in Tab. A.2 zu sehen.

Aktivitätsdiagramme werden mittels des Schlüsselwortes act (Abkürzung für activity [engl.]) im linken oberen Teil des Diagrammes bezeichnet. Dem Schlüsselwort folgt der Name der beschriebenen Aktivität. In Abb. A.1 ist zum besseren Verständnis des Zusammenwirkens der Elemente eines Aktivitätsdiagrammes ein Beispiel angegeben. Dieses enthält einen Startknoten, eine Aktivität, eine Teilaktivität, eine Entscheidung sowie mehrere Objekte (Eingabeobjekt, Ausgabeobjekt oder internes Objekt). Ein Endknoten muss in diesem Beispiel nicht angegeben werden, weil die Abarbeitung mit der Rückgabe eines Objektes endet.

Abb. A.1

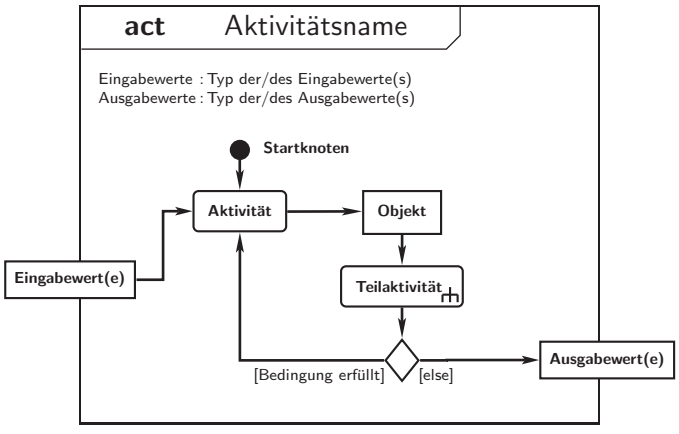
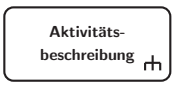
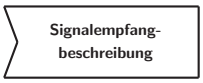
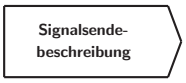
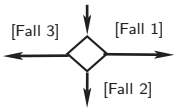
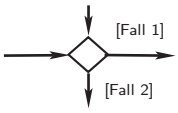
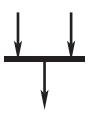
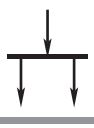


Abb. A.1.: *Beispiel eines UML-Aktivitätsdiagrammes*

Tab. A.2

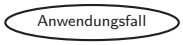
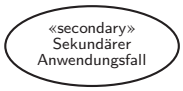
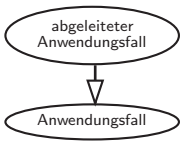
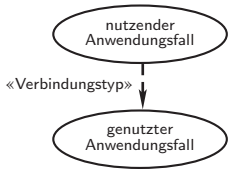
Element	Erklärung
Startknotenname	***Startknoten*** [dt.; initial node [engl.]] einer Aktivität
Endknotenname	***Endknoten*** [dt.; final node [engl.]] einer Aktivität
Aktivitätsbeschreibung	Aufruf einer ***Aktivität*** oder ***Aktion*** beliebiger Dauer
 Aktivitäts-beschreibung	***Teilaktivität***, welche in einem anderen Aktivitätsdiagramm genauer beschrieben ist
Objektbezeichnung	***Objekt***, welches von einer Aktion erzeugt und konsumiert wird
	Fluss [dt.; flow [engl.]] oder ***Kante*** [dt.; edge [engl.]], welche den Übergang zwischen zwei Knoten kennzeichnet
 Signalempfang-beschreibung	***Signalempfang*** beschreibt eine Aktion, welche auf ein Signal wartet
 Signalsende-beschreibung	***Signalausendung*** stellt eine Aktion dar, die ein Signal sendet
Zeitereignis-beschreibung	***Zeitereignisempfang*** kennzeichnet eine Aktion, die auf einen Zeitpunkt wartet
	Unterbrechung eines unterbrechbaren Aktivitätsbereiches durch ein Ereignis
 [Fall 3] [Fall 1] [Fall 2]	***Entscheidung mit Wächterbedingungen*** [dt.; decision with guard conditions [engl.]], so dass in Abhängigkeit der Wächterbedingungen einer der Zweige gewählt wird, wobei sich die Wächterbedingungen gegenseitig ausschließen und die spezielle Wächterbedingung [else] zutrifft, wenn alle anderen Bedingungen nicht erfüllt sind
	Zusammenführung [dt.; merge [engl.]] zweier oder mehrerer Flüsse zu einem
 [Fall 1] [Fall 2]	Kombination von ***Entscheidung und Zusammenführung***
	***Synchronisation*** [dt.; join [engl.]] paralleler Abläufe, wobei solange gewartet wird, bis alle Flüsse eingetroffen sind
	***Aufspaltung*** [dt.; fork [engl.]] eines Flusses in mehrere nebeneinander laufende Flüsse

Tab. A.2.: *Elemente eines UML-Aktivitätsdiagramms*

A.3.2. Anwendungsfalldiagramme

Ein Anwendungsfalldiagramm beschreibt das aus Sicht eines Anwenders erkennbare Verhalten eines Systems, bspw. des Softwaresystems `CAOS`. Dazu besteht ein Anwendungsfalldiagramm aus einer Menge von Akteuren und Anwendungsfällen sowie der Beziehungen dieser untereinander. Die einzelnen Akteure und Anwendungsfälle tragen zu ihrer Identifikation einen eindeutigen Namen. Zudem können Anwendungsfälle hierarchisch geschachtelt werden, wovon in dieser Arbeit kein Gebrauch gemacht wird. In Tab. A.3 sind die verschiedenen Elemente, welche bei UML-Anwendungsfalldiagrammen auftreten können, sowie mögliche Beziehungen zwischen Anwendungsfällen angegeben.

Tab. A.3

Element	Erklärung
	ein ***Anwendungsfall*** gibt an, was ein System tun soll
	Akteure geben an, wer (andere Person) oder was (anderes System) etwas mit dem vorliegenden System macht
	***Sekundäre Anwendungsfälle*** können Teil der Zerlegung eines größeren Anwendungsfalles oder häufig wiederkehrende Anwendungsfälle sein
	ein ***Untersystem*** stellt eine Gruppierung von Elementen dar, welche eine logische oder physische Einheit bilden
	der ***abgeleitete Anwendungsfall*** spezialisiert einen anderen Anwendungsfall (ermöglicht es eine hierarchische Strukturierung von Anwendungsfällen vorzunehmen)
	mittels einer gerichteten gestrichelten Linie wird eine Abhängigkeitsbeziehung von Anwendungsfällen durch eine ***Beinhaltungsverbindung*** (Verbindungstyp «include») oder eine ***Nutzungsverbindung*** (Verbindungstyp «realize») verdeutlicht

Tab. A.3.: *Elemente eines UML-Anwendungsfalldiagramms*

In Abb. A.2 ist an einem Beispiel gezeigt, wie die einzelnen Elemente eines Anwendungsfalldiagrammes zusammenwirken. Darin ist auch erkennbar, dass ein Anwendungsfalldiagramm durch das Schlüsselwort use case [engl.] gekennzeichnet wird.

Abb. A.2

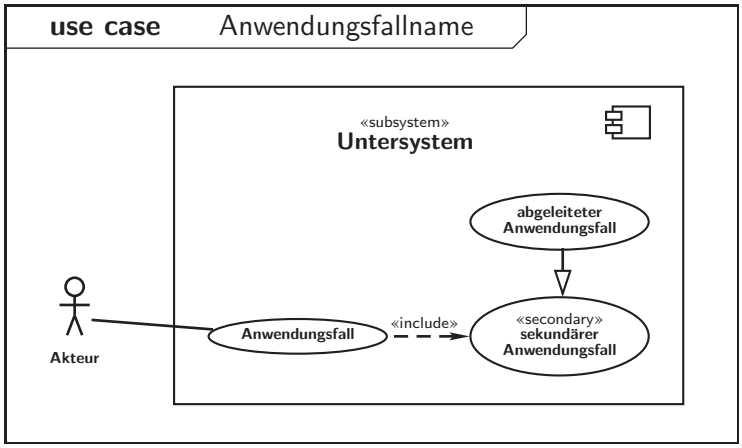


Abb. A.2.: *Beispiel eines UML-Anwendungsfalldiagrammes*

A.3.3. Kommunikationsdiagramme

Ein Kommunikationsdiagramm stellt Interaktionen innerhalb eines Systems graphisch dar. Dazu werden die einzelnen Objekte durch Lebenslinien verbunden, entlang derer Nachrichten ausgetauscht werden (siehe Tab. A.4).

Tab. A.4

Element	Erklärung
Objekt	***Objekt***, welches eine oder mehrere Operationen hervorruft (Sender einer Operation) oder auf welches eine oder mehrere Operationen angewendet werden (Empfänger einer Operation)
x.y: Operation →	***Lebenslinie*** entlang derer eine ***Nachricht*** gesandt wird mit Angabe von Richtung und zugeordnetem Sequenzausdruck, aus dem hervorgeht, was die Vorgängernachrichten sind (Nachricht mit dem Sequenzausdruck $x.y$ folgt auf die Nachricht mit dem Sequenzausdruck x usw., wodurch eine zeitliche Ordnung entsteht), so dass die Sequenzausdrücke aus einer Folge von mit einem Punkt getrennten Sequenztermen bestehen, gefolgt von einem Doppelpunkt und einer Operation

Tab. A.4.: *Elemente eines UML-Kommunikationsdiagramms*

Das Schlüsselwort, mit dem ein Kommunikationsdiagramm gekennzeichnet ist, lautet sd[118] (Abkürzung für sequence diagram [engl.]). Dies wird an einem Beispiel verdeutlicht, welches in Abb. A.3 dargestellt ist.

Abb. A.3

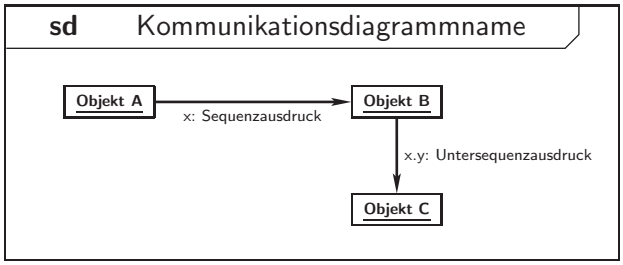


Abb. A.3.: *Beispiel eines UML-Kommunikationsdiagrammes*

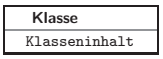
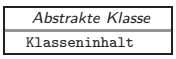
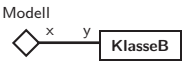
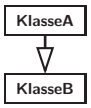
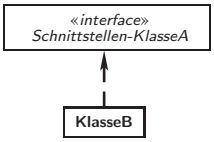
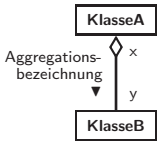
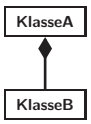
[118] Das Schlüsselwort sd wird auch für Sequenzdiagramme, welche im Rahmen dieser Arbeit nicht näher besprochen werden, verwendet.

A.4. Verwendete Strukturdiagramme der UML

A.4.1. Klassendiagramme

Ein Klassendiagramm stellt im Wesentlichen Klassen[119] sowie deren Beziehungen untereinander graphisch dar. Es dient zur Modellierung eines abgegrenzten Systems während der Analyse- und Entwurfszeit. Dazu besteht ein Klassendiagramm aus verschiedenen Elementen, welche in Tab. A.5 zusammenfassend dargestellt sind.

Tab. A.5

Element	Erklärung
	***Klasse*** [dt.; class [engl.]] mit einem optionalen Klasseninhalt, welcher verschiedene Rubriken [dt.; compartment [engl.]], wie Klassenname, Attribute, Operationen und Eigenschaften, enthalten kann, die jeweils durch eine horizontale Linie getrennt werden
	***Abstrakte Klassen*** [dt.; abstract classes [engl.]] sind Klassen, von denen keine Objekte angelegt werden können; der Klassenname wird kursiv dargestellt
	Schnittstellenklassen [dt.; interfaces [engl.]] sind abstrakte Klassen zur Definition funktionaler Schnittstellen, um u. a. die arbeitsteilige Softwareentwicklung zu erleichtern
	***Assoziation*** beschreibt eine Beziehung zwischen zwei oder mehr Klassen (auch mehrgliedrige Assoziation genannt), wobei x und y die Multiplizitäten, d. h. die unteren und oberen Grenzen, angeben (bspw. steht 0..1 für eine Multiplizität von null oder eins und 0..* für eine Multiplizität von null bis unendlich)
	***Generalisierungsbeziehung*** zwischen zwei Klassen, d. h. Ableitung einer normalen oder abstrakten Klasse von einer anderen normalen oder abstrakten Klasse (KlasseB ist abgeleitet von KlasseA)
	***Nutzung einer Schnittstellen-Klasse*** (KlasseB nutzt *Schnittstellen-KlasseA*)
	***Aggregation*** mit einer Bezeichnung und einer Richtung (Aggregation gibt die Beziehung zwischen einem Ganzen und seinen Teilen in Form von Klassen an), wobei x und y die gleiche Bedeutung wie bei einer Assoziation haben und der schwarze Pfeil die Leserichtung angibt
	***Komposition*** ist ein Spezialfall der Aggregation und bildet den Fall ab, bei dem die Teile (KlasseB) nicht ohne das Ganze (KlasseA) existieren können

Tab. A.5.: *Elemente eines UML-Klassendiagramms*

Für ein Klassendiagramm wird das Schlüsselwort cd (Abkürzung für class diagram [engl.]) verwendet. In den Abb. A.4 und Abb. A.5 sind verschiedene Darstellungen für Klassendiagramme angegeben. In Abb. A.4 ist die Nutzung von abstrakten Klassen und Schnittstellen-Klassen zur Erstellung einer „normalen“ Klasse zu sehen.

[119] Der Begriff der Klasse stammt im eigentlichen Sinne aus der objektorientierten Programmierung (OOP). Eine Klasse dient dort der Klassifizierung von Objekten, d. h. der Beschreibung der Gemeinsamkeiten von Struktur und Verhalten bei Objekten. Ziel einer Klasse ist somit die Verallgemeinerung von Objekten.

Abb. A.4

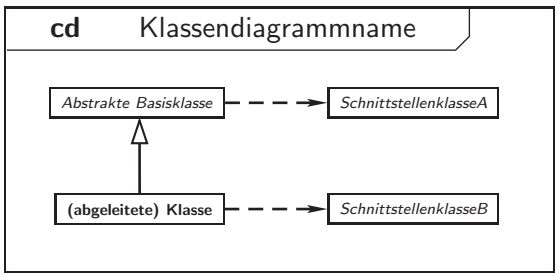


Abb. A.4.: *Beispiel 1 eines UML-Klassendiagrammes*

Die Abb. A.5 zeigt ein weiteres Beispiel für ein Klassendiagramm. Es ist zu sehen, dass die Klasse Wurzel *aus* beliebig vielen, aber mindestens einer Klasse Element *besteht*. Jede Klasse Element *besteht* wiederum *aus* genau einer Klasse Attribut und benötigt die Klasse AttributZusatz.

Abb. A.5

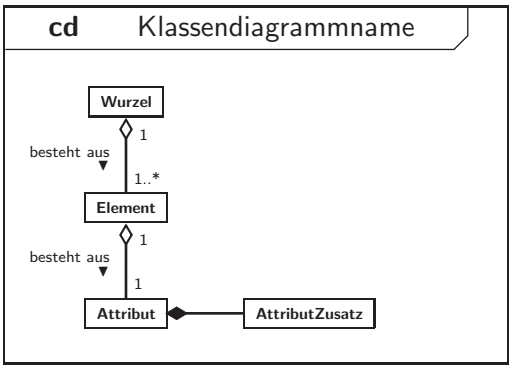


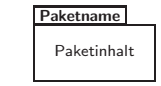
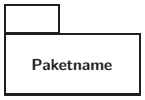
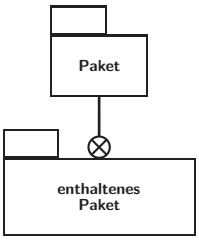
Abb. A.5.: *Beispiel 2 eines UML-Klassendiagrammes*

A.4.2. Paketdiagramme

Das Paketdiagramm unterteilt die Struktur eines (Software)Systems in verschiedene Pakete. Bei der Modellierung von größeren Softwaresystemen finden Paketdiagramme sehr oft Verwendung, um bspw. die Schichten-Architektur einer Software darzustellen. Oftmals wird Software jedoch auch in einzelne Module unterteilt. Auch in diesem Falle können Paketdiagramme zur Verdeutlichung des Zusammenspiels und der Abhängigkeit der einzelnen Module genutzt werden. Ein Paketdiagramm kann aber ebenso eine Übersicht über verschiedene Geschäftsfälle im Rahmen der Geschäftsprozessmodellierung liefern.

Ein Paketdiagramm nutzt die Bezeichnung pd (Abkürzung für package diagram [engl.]) oder teilweise auch package als Schlüsselwort. Auf die Angabe eines Beispieles zu Paketdiagrammen sei an dieser Stelle verzichtet, weil die wichtigsten Informationen zum Verständnis eines Paketdiagrammes bereits in Tab. A.6 enthalten sind.

Tab. A.6

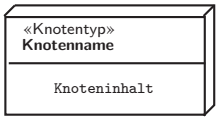
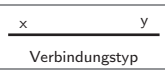
Element	Erklärung
 oder 	ein ***Paket*** wird zur Kapselung von verschiedenen Teilen eines Systems genutzt
	ein ***enthaltenes Paket*** oder ***Unterpaket*** stellt einen Teil eines Paketes dar, welcher zum besseren Verständnis etwas näher spezifiziert wird

Tab. A.6.: *Elemente eines UML-Paketdiagramms*

A.4.3. Verteilungsdiagramme

Ein Verteilungsdiagramm beschreibt die Struktur eines modellierten Systems aus einer bestimmten Sicht, welche bspw. die Verteilung einer Anwendung auf mehreren Rechnern oder innerhalb eines Rechners widerspiegelt. Dazu werden hauptsächlich verschiedene Verteilungsbeziehungen zwischen verschieden Artefakten und verschiedenen Knoten dargestellt. Die Nutzung von Verteilungsbeziehungen basiert dabei u. a. auf verschiedenen Schnittstellen (siehe Tab. A.7).

Tab. A.7

Element	Erklärung
	Artefakte spiegeln das Ergebnis eines Arbeitsprozesses wider und werden durch das Schlüsselwort «artifact» hervorgehoben, wobei für bestimmte Arten von Artefakten vorgegebene Schlüsselworter existieren (ansonsten wird ein Artefakt wie eine Klasse dargestellt)
	mittels eines ***Knotens*** wird ein Element, z. B. ein Rechner, eines Rechnernetzwerkes repräsentiert (Auf den Elementen können z. B. Artefakte einer Software installiert werden.)
	eine ***Komponente*** ist die Spezialisierung einer (Basis-)Klasse, wobei Komponenten so modular sind, dass sie durch eine andere, äquivalente Komponente ersetzt werden könnten (können durch das Schlüsselwort «component» noch weiter hervorgehoben werden)
	eine ***Schnittstelle*** [dt.; interface [engl.]] hat die gleiche Bedeutung wie bei Klassendiagrammen (siehe Tab. A.5), wobei an dieser Stelle eine andere Darstellungsform genutzt wird
	stellt eine ***Assoziation*** (x:y-Objektbeziehung) zwischen zwei Knoten dar, wobei x und y die gleiche Bedeutung wie in Tab. A.5 besitzen
	stellt eine ***Abhängigkeitsbeziehung***, bspw. zwischen einem Knoten und Schnittstelle, dar (bei der gewählten Darstellungsform sind unterschiedliche Typen von Abhängigkeitsbeziehungen möglich, welche durch verschiedene Schlüsselwörter unterschieden werden, z. B. Verwendungsbeziehung (Schlüsselwort «use») und Verteilungsbeziehung (Schlüsselwort «deploy»))

Tab. A.7.: *Elemente eines UML-Verteilungsdiagramms*

Verteilungsdiagramme werden durch das Schlüsselwort dd (Abkürzung für deployment diagram [engl.]) im linken oberen Teil des Diagramms gekennzeichnet. In Abb. A.6 ist ein Beispiel für ein Verteilungsdiagramm angegeben, in welchem zwei Rechner über ein LAN mittels des Protokollpaares TCP/IP Informationen in Form von Nachrichten untereinander austauschen können. Dazu wird eine einheitliche (Kommunikations-)Schnittstelle genutzt, welche eine spezielle Komponente zum Nachrichtentransport implementiert. Jeder der Rechner verwendet ein Artefakt in Form einer ausführbaren Client- oder Server-Anwendung.

Abb. A.6

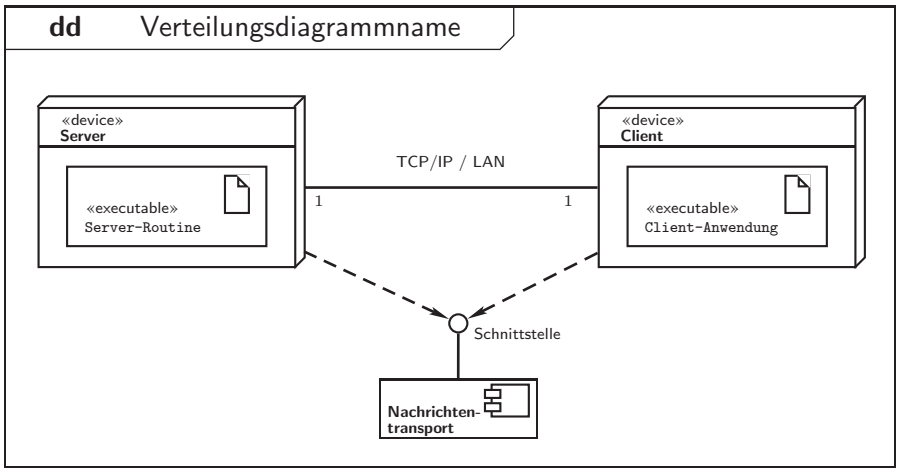


Abb. A.6.: *Beispiel eines UML-Verteilungsdiagrammes*

Kapitel B. Konstrukte für Produktionsablauf-Diagramme und die Sprache EcoSyL 0.1

Inhalt

B.1 Mögliche Elemente eines Produktionssystems 251
B.2 Klassifikation von Scheduling-Problemen 253

B.1. Mögliche Elemente eines Produktionssystems

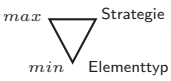
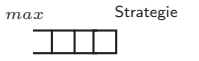
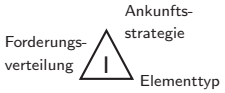
Zur besseren Verständlichkeit mittels visueller Anzeigemöglichkeiten wurde eine neue Beschreibungssprache für die schematische Darstellung von ökonomischen Produktionssystemen entwickelt, weil in der Literatur bisher keine einheitliche Darstellungsmöglichkeit für solche ökonomischen Systeme besteht. Die neue Sprache wird als *EconomicSystemLanguage* mit derzeitiger *Version 0.1* (kurz: *EcoSyL 0.1*) bezeichnet und beinhaltet derzeit graphische Konstrukte für

- Eingänge,
- Ausgänge,
- Produktion,
- Lager,
- Transport,
- Warteschlangen sowie
- externe Eingriffsmöglichkeiten.

EcoSyL 0.1 stellt sowohl *Produktflüsse* (materiell) als auch *Informationsflüsse* (immateriell) graphisch dar, um den Ablauf (Produktfluss) und die Steuerung (Informationsfluss) darzustellen.

Die Gesamtheit aller in EcoSyL 0.1 vorhandenen Sprachelemente zur visuellen Nachbildung eines ökonomischen Produktionssystems sowie der darin enthaltenen Produkt- und Informationsflüsse sind in Tab. B.1 bis Tab. B.2 dargestellt.

Tab. B.1

Element	Elementbeschreibung
Bezeichnung 	***Lager*** mit einer *Bezeichnung* sowie einer *minimalen Lagerkapazität* min und einer *maximalen Lagerkapazität* max für *Elemente eines bestimmten Elementtyps*, welche nach einer vorgegebenen *Lagerstrategie* eingelagert bzw. entnommen werden Hinweise: • ist keine Lagerstrategie angegeben, so ist diese beliebig • ähnliches gilt für den Elementtyp: ist keiner angegeben, so können Elemente eines beliebigen Typs eingelagert werden • die Bezeichnung kann ebenfalls entfallen
Bezeichnung 	***Warteschlange*** mit einer *Bezeichnung* sowie einer *maximalen Kapazität* max für *Elemente eines bestimmten Elementtyps*, welche nach einer vorgegebenen *Warteschlangenstrategie* geordnet sind Hinweise: • ist keine Warteschlangenstrategie angegeben, so wird nach dem First-Come-First-Serve-Prinzip (FCFS) verfahren • ähnliches gilt für den Elementtyp: ist keiner angegeben, so können sich Elemente eines beliebigen Typs in der Warteschlange befinden • die Bezeichnung kann ebenfalls entfallen
Bezeichnung 	***Eingang für Kundenbedarf*** mit einer *Bezeichnung* sowie einer Forderung nach *Elementen eines bestimmten Elementtyps*, wobei die Kunden nach einer bestimmten *Ankunftsstrategie* eintreffen Hinweis: • die Bezeichnung kann entfallen • jedem Eingang muss auch ein Ausgang zugeordnet sein
Bezeichnung 	***Ausgang für Produkte*** mit einer *Bezeichnung* sowie dem *Elementtyp der gefertigten Produkte* und der *Anzahl der gefertigten Produkte* Hinweis: • die Bezeichnung kann entfallen • jedem Ausgang muss auch ein Eingang zugeordnet sein
$\alpha\|\beta\|\gamma$ **Bezeichnung** Erzeugter Elementtyp Ausgangs-elementtyp	***Produktion***, welche die *Transformation eines Ausgangselementtyps* in einen *zu erzeugenden Elementtyp* nach einem bestimmten *Schedulingverfahren mit den Parametern* $\alpha\|\beta\|\gamma$ durchführt Hinweise: • ist kein transformierter Elementtyp angegeben, werden alle Elemente in einen unbestimmten Typ transformiert • die Bezeichnung kann entfallen • die Bedeutung der Parameter $\alpha\|\beta\|\gamma$ ist u. a. in [Bru04] und im Abschn. B.2 beschrieben
Bezeichnung Bezeichnung des Lösungs-algorithmus Parameter-typ und -wertebereich	***Externer Eingriffspunkt*** mit einer *Bezeichnung* und einem vorgegebenen *Parameterraum* (*Typ* und *Wertebereich*), in welchem mit einem *Lösungsalgorithmus* eine (sub)optimale Lösung oder Pareto-optimale Lösungsmenge gesucht wird Hinweis: • die Bezeichnung kann entfallen • der Lösungsalgorithmus kann – ein (iteratives) *exaktes Verfahren*, – ein (iteratives) *heuristisches Verfahren* (*Heuristik*) oder – eine *fest vorgegebene Strategie* beinhalten

Tab. B.1.: *Mögliche Elemente der visuellen Beschreibungssprache EcoSyL 0.1 (Teil 1/2)*

Tab. B.2

Element	Elementbeschreibung
Typ oder Name des Lösungsalgorithmus Problemtyp und Parameter **Bezeichnung** zutransportierender Elementtyp oder: Typ oder Name des Lösungsalgorithmus Problemtyp und Parameter **Bezeichnung** zutransportierender Elementtyp	***Transport***, welcher einen festgelegten *Problemtyp mit bestimmten Parametern* charakterisiert, der mittels eines vorgegebenen *Algorithmus* gelöst wird und für *Elemente eines bestimmten Typs* stattfindet Hinweise: • die Bezeichnung kann entfallen • ist kein zu transportierender Elementtyp angegeben, erfolgt der Transport beliebiger Elemente • fehlt die Angabe des Problemtyps, so liegt ein TSP zugrunde • der Lösungsalgorithmus kann – ein (iteratives) *exaktes Verfahren*, – ein (iteratives) *heuristisches Verfahren* (*Heuristik*) oder – eine *fest vorgegebene Route/Tour* beinhalten (siehe [Käm06])
[Beschreibung]	***Verbindung für einen Produktfluss***, welcher indirekt auch einen Informationsfluss (s. u.) mit beinhaltet und die Beschreibung einen Hinweis über das gesendete Objekt oder die gesendete Nachricht gibt Hinweis: • die Beschreibung kann entfallen
[Beschreibung]	***Verbindung für einen Informationsfluss*** bzw. ***Handlungsablauf***, wobei die Beschreibung einen Hinweis über die gesendete Nachricht (Aktivität) gibt Hinweis: • die Beschreibung kann entfallen
(a) [Fall 1] [else] (b) $[P_1]$ $[P_N]$	***Entscheidung***, welche einen Informationsfluss (a) oder Produktfluss (b) in mehrere aufteilt, abhängig von evtl. angegebenen Bedingungen (a) oder Produkttypen (b)
(a) (b)	***Zusammenführung***, welche mehrere Informationsflüsse (a) oder Produktflüsse (b) zu einem vereint

Tab. B.2.: *Mögliche Elemente der visuellen Beschreibungssprache EcoSyL 0.1 (Teil 2/2)*

B.2. Klassifikation von Scheduling-Problemen

Das nachfolgend in den Tabellen B.3 und B.4 dargestellte Klassifikationsschema basiert auf dem Vorschlag aus [GLLK79]. Es wird innerhalb dieser Arbeit zur Übersichtlichkeit, Nachvollziehbarkeit und Vollständigkeit kurz aufgezeigt. Das Klassifikationsschema selbst wird von verschiedenen Autoren angewandt und weiterentwickelt (siehe bspw. [Bru04] und [Qua04]).

Die Problembeschreibung setzt sich dabei aus drei Feldern zusammen, die mit α, β und γ bezeichnet werden. Die einzelnen Felder werden durch ein |-Zeichen getrennt und bestehen wiederum aus mehreren, durch Kommas getrennten Teilen. Die erste Zeile eines neuen Parameters bzw. Parameterteils gibt dessen Standardwert an, für den Fall, dass für einen Teil der Parameter kein Wert angegeben ist.

Das erste Feld mit dem Parameter α beschreibt die Prozessor- bzw. Maschinen-Umgebung. Es gibt Auskunft über die Anzahl der Prozessoren, ihre Beziehung untereinander (z.B. identisch, unabhängig) und ihre Beziehung zu den Jobs (z. B. Shopsysteme).

Tab. B.3

Symbol	Kategorie	Werte	Bedeutung
α_1	Parallele Maschinen	*1*	Einzelmaschine
		P_m	m identische Maschinen
		Q_m	m gleichartige Maschinen
		R_m	m unabhängige Maschinen
	Multi Purpose Machines [engl., kurz: MPM]	*PMPM*	Identische MPM
		QMPM	Gleichartige MPM
	Festgelegte Maschinen [dt.; dedicated machines [engl.]]	*G*	Allgemeines shop-system [engl.]
		J	job-shop-system [engl.]
		F	flow-shop-system [engl.]
		O	open-shop-system [engl.]
		X	mixed-shop-system [engl.]
α_2	Anzahl der Maschinen	*0*	Variable Anzahl von Maschinen
		k	k-Prozessor-Maschine (k - feste Anzahl)
α_3	Netzwerktopologie	*0*	Kommunikation vernachlässigbar
		conn	Beliebige Kommunikationsverbindungen
		linear array	Linie (eindimensionales Gitter)
		ring	geschlossene Linie (Ring)
		mesh	d-dimensionales Gitter
		hypercube	d-dimensionaler Hyperwürfel
		tree	Netzwerk mit Baumstruktur
α_4	Gleichzeitigkeit von Berechnung und Kommunikation	*0*	Simultane Berechnung und Kommunikation möglich
		no-overlap	keine gleichzeitige Berechnung und Kommunikation

Tab. B.3.: *Bedeutung des Parameters α bei der Klassifizierung von Scheduling-Problemen*

Das zweite Feld mit dem Parameter β spezifiziert die charakteristischen Merkmale der Jobs und der Ressourcen.

Der Parameter γ, mit dem das dritte Feld beschrieben wird, gibt die Charakteristik und die Bestandteile der Zielfunktion an. Hierzu werden die von einer Lösung ermittelten Fertigstellungstermine C_i [dt.; completion time [engl.]] herangezogen. Als Optimierungsprobleme können sich somit bei Verwendung der Zielfunktionen $f_j(C_i)$ zwei Varianten ergeben:

$$\min \sum_{j=1}^{\dim(\overrightarrow{goals})} w_j \cdot \max\{f_j(C_i) \mid i = 1, 2, \ldots, \dim(\overrightarrow{jobs})\}$$

oder

$$\min \sum_{j=1}^{\dim(\overrightarrow{goals})} w_j \cdot \sum_{i=1}^{\dim(\overrightarrow{jobs})} f_j(C_i) \quad ,$$

wobei

$\dim(\overrightarrow{goals})$ - die Anzahl der verwendeten Zielfunktionen,

$\dim(\overrightarrow{jobs})$ - die Anzahl der (Fertigungs-)Aufträge und

$0 \leq w_j \leq 1$ - Gewichtung des Zielfunktionswertes j

ist.

Anstatt der Verwendung des Fertigstellungstermins eines Auftrages i sind auch andere Funktionen zulässig, wie bspw.

Tab. B.4

Symbol	Kategorie	Werte	Bedeutung
β_1	Unterbrechbarkeit	*0*	Unterbrechungen nicht erlaubt
		pmtn	Unterbrechungen erlaubt [dt.; preemption [engl.]]
		div	divisible tasks [engl.] (Unterbrechungen erlaubt)
β_2	Zusätzliche Ressourcen	*0*	keine zusätzlichen Ressourcen
		res	spezifizierte zusätzliche Ressourcen
β_3	Präzedenzrelation	*0*	unabhängige Jobs
		prec	beliebige Präzedenzrelation (DAG)
		permutation	Reihenfolge der Aufträge für alle Maschinen gleich
		uan	uniconnected activity networks [engl.]
		intree, outtree	intree [engl.] bzw. outtree [engl.]
		tree	Baum (entweder in- oder outtree)
		chains	Menge von Ketten
		sp-graph	serien-paralleler Graph
β_4	Freigabezeiten [dt.; release times [engl.]]	*0*	alle Jobs werden zu Beginn ($t = 0$) freigegeben
		r_j	beliebige Freigabezeiten, aber Freigabezeit existiert
β_5	Bearbeitungszeiten	*0*	beliebige Bearbeitungszeiten
		$p_j = p$	konstante Bearbeitungszeiten für alle Aufträge j
		$p_{lb} \leq p_j \leq p_{ub}$	begrenzte Bearbeitungszeiten
β_6	Fälligkeitszeiten [dt.; deadlines [engl.]]	0	keine Fälligkeitszeiten
		d_j	Fälligkeitszeiten können angegeben werden
β_7	Maximale Anzahl von Operationen in Shop-Systemen	*0*	keine Begrenzung oder kein Shop-System
		$n_j \leq k$	Anzahl der Operationen ist auf k begrenzt
β_8	Warten bei festgelegten Maschinen	*0*	unendlich große Puffer
		no-wait, nwt	keine Pufferung, Jobs müssen auf nachfolgenden Maschinen die Bearbeitung unverzüglich fortsetzen
		block	Pufferung einer begrenzten Anzahl von Jobs

Tab. B.4.: *Bedeutung des Parameters β bei der Klassifizierung von Scheduling-Problemen*

$L_i = C_i - d_i$ - *Verspätung* des Auftrages i [dt.; lateness [engl.]] (mit d_i als vorgegebenen Auslieferungs-/ Fertigstellungstermin [dt.; due date [engl.]]),

$T_i = \max\{0, C_i - d_i\}$ - *Verspätung* bzw. Unpünktlichkeit des Auftrages i [dt.; tardiness [engl.]],

$E_i = \max\{0, C_i - d_i\}$ - *Frühzeitigkeit* des Auftrages i [dt.; earliness [engl.]],

$D_i = |C_i - d_i|$ - *absolute Terminabweichung* des Auftrages i [dt.; absolute deviation [engl.]] sowie

$U_i = \begin{cases} 0, & \text{falls } C_i \leq d_i \\ 1, & \text{sonst} \end{cases}$ - *Strafe pro Verspätung* [dt.; unit penalty [engl.]].

Daraus ergeben sich bspw. folgende mögliche Zielfunktionen f_j bei Verwendung einer Maximierungsfunktion:

$f_j(C_{max}) = \max_{i=1,2,...,\dim(\overrightarrow{jobs})} C_i$ - *Letzter Fertigstellungstermin* aller Aufträge und

$f_j(L_{max}) = \max_{i=1,2,...,\dim(\overrightarrow{jobs})} L_i$ - *Maximale Verspätung* aller Aufträge

sowie bei Verwendung einer Summenfunktion:

$f_j(w \cdot C) = \sum_{i=1}^{\dim(\overrightarrow{jobs})} w_i \cdot C_i$ - *Gewichtete Fertigstellungstermine* mit w_i für die Gewichtung des Auftrages i,

$f_j(T) = \sum_{i=1}^{\dim(\overrightarrow{jobs})} T_i$ - *Summe der Verspätungen* aller Aufträge oder

$f_j(T^{anz}) = \sum_{i=1}^{\dim(\overrightarrow{jobs})} T_i^{anz}$ - *Anzahl aller verspäteten Aufträge* mit

$$T_i^{anz} = \begin{cases} 0, & \text{für } T_i = 0 \\ 1, & \text{für } T_i > 0 \end{cases} .$$

Kapitel C. Die Programmiersprache C#, deren Zwischensprache und das .NET-Framework

Inhalt

C.1	*Begriffserklärungen*	*257*
C.2	*Übersetzungs- und Laufzeitverhalten bei C#-Programmen*	*259*
C.3	*Vorteile*	*260*
C.4	*Nachteile*	*261*
C.5	*Sprachbesonderheiten*	*261*
C.6	*Vergleich mit der Programmiersprache Java*	*264*
C.7	*Frei verfügbare Entwicklungs- und Laufzeitumgebungen*	*264*
C.8	*Fazit*	*264*

C.1. Begriffserklärungen

Zum Verständnis der Entwicklungsziele, welche mit der Programmiersprache C# (siehe auch [Sch06c] und [Sch07c]) verfolgt werden, sowie der Programmabarbeitung eines C#-Programmes ist es unumgänglich im Vorfeld zu klären, welche Rolle die Common Language Infrastructur [engl., kurz: CLI] und das .NET-Framework [engl.] besitzen. Es wird dabei davon ausgegangen, dass dem Leser das objektorientierte Programmierparadigma und damit auch Begriffe wie Klasse, Klassenobjekt, Vererbung, Kapselung, Datensichtbarkeit, Polymorphismus, virtuelle Funktionen, u. Ä. bekannt sind.

Die Programmiersprache C#. Zur Festlegung des Sprachumfangs der Programmiersprache C#, wurde dieser von der European Computer Manufacturers Association [engl., kurz: ECMA] in der ECMA-334 und gemeinsam von der internationalen Organisation für Normung (International Organization for Standardization

[engl., kurz: ISO]) sowie der Internationalen elektrotechnischen Kommission (International Electrotechnical Commission [engl., kurz: IEC[]) in der ISO/IEC 23270 spezifiziert. Aktuell liegt die Spezifikation für C# 2.0 vor (ECMA-334, vierte Version vom Juni 2006 (siehe [Ecm06a]) und ISO/IEC 23270:2006 (siehe [ISO06c])). Die Zertifizierung für die aktuellste Sprachspezifikation namens C# 3.0 soll im Laufe des Jahres 2008 erfolgen.

Einige wesentliche Besonderheiten der Programmiersprache C# werden nachfolgend etwas genauer beschrieben (s. u.). Die Programmiersprache C# selbst wurde speziell zur *Erstellung von Zwischenkode nach der CLI* entworfen. Sie hat sich aus verschiedenen, zumeist objektorientierten Programmiersprachen entwickelt (siehe Abb. C.1).

Abb. C.1

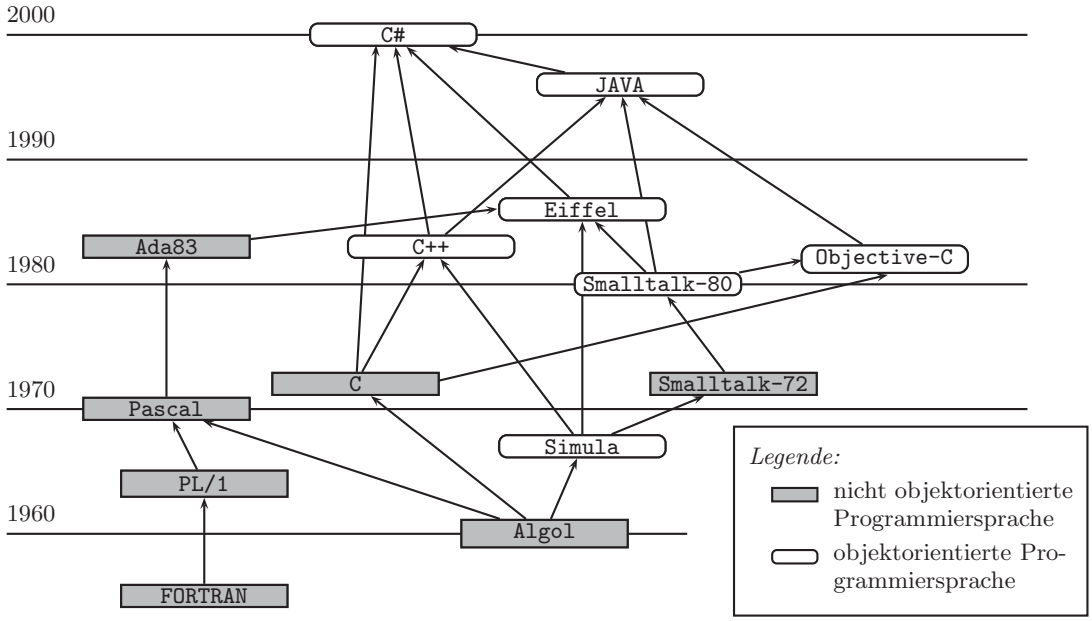


Abb. C.1.: *Historische Entwicklung der für C# relevanten (objektorientierten) Programmiersprachen (nach [BS03, Abb. 1-1])*

Die Spezifikation der CLI und die Zwischensprache CIL. Die CLI ist eine *Spezifikation für einen einheitlichen Zwischenkode*, welche als *Basis zur Anwendungsentwicklung und -ausführung unabhängig von einer Programmiersprache und Prozessorarchitektur* oder allgemein *Rechnerplattform* entworfen wurde. Der entstandene (objektorientierte) Zwischenkode liegt nach der Übersetzung eines entsprechenden Quelltextes in der gemeinsamen Zwischensprache Common Intermediate Language [engl., kurz: CIL] vor. In der CLI ist demnach *keine konkrete Implementierung* für eine bestimmte Rechnerplattform vorgesehen. Die genauen Details der CLI sind in der ECMA-335, welche mittlerweile in der vierten Version [dt.; 4th edition [engl.]] vorliegt, spezifiziert (siehe [Ecm06b]). Zudem wurden die CLI in der ISO/IEC 23271:2006 (siehe [ISO06b]) und die CLI TR (Technical Report [engl.]) in der ISO/IEC TR 23272:2006 (siehe [ISO06a]) festgelegt. Auf Grund der Tatsache, dass die CLI nur die Spezifikation für den einheitlichen Zwischenkode in CIL

angibt, existieren mittlerweile verschiedene weitere Programmiersprachen, welche gleichberechtigt nebeneinander existieren und diesen CIL-Zwischenkode nach der CIL erzeugen können (siehe [Sch07d]). Dadurch kann ein Softwareersteller in seiner gewohnten Programmiersprache arbeiten und auf Wunsch als Erweiterung die Klassen-Bibliotheken des .NET-Framework nutzen.

Das .NET-Framework mit der CLR und der FCL. Das .NET-Framework ist eine konkrete Implementierung der CLI von Microsoft®, welches im Wesentlichen

- die *Laufzeitumgebung* (*Common Language Runtime* [engl., kurz: *CLR*]), welche eine *(objektorientierte) stack-basierte virtuelle Maschine (VM)* darstellt, sowie
- eine *Sammlung von Klassenbibliotheken* (*Framework Class Library* [engl., kurz: *FCL*]) mit grundlegenden und weiter reichenden *Diensten für die Entwicklung eigener Software*

beinhaltet. Zunächst wurde das .NET-Framework von Microsoft® nur für das Betriebssystem Microsoft®Windows® umgesetzt. Mittlerweile existieren aber auch weitere Implementierungen der CLI, welche auch Teile des .NET-Framework für andere Betriebssysteme (siehe auch Tab. C.1) umsetzen. Aktuell liegt das .NET-Framework von Microsoft® in der Version 3.0 vor (siehe [Sch06c] und [Sch07c]).

C.2. Übersetzungs- und Laufzeitverhalten bei C#-Programmen

In Abb. C.2 ist ein kurzer, prinzipieller Überblick über die Vorgehensweise zur plattformunabhängigen Erstellung und Ausführung eines C#-Programms gegeben, so dass dieses auf einer beliebigen Rechnerplattform sowie einem beliebigen Betriebssystem erstellt und einer (anderen beliebigen) Rechnerplattform mit einem (anderen beliebigen) Betriebssystem ausgeführt werden kann.

Wie bereits zuvor angemerkt wurde, entsteht nach der Übersetzung eines C#-Programms ein Zwischenkode in CIL. Zur Ausführung dieses Zwischenkodes wurde u. a. im .NET-Framework analog zur Ausführung des Bytekodes, welcher als Zwischenkode bei der Programmiersprache Java entsteht, so genannte *Just-In-Time-Übersetzer* [engl., kurz: *JIT-Übersetzer* bzw. *JIT-Compiler*] eingeführt. Beim .NET-Framework ist dieser JIT-Übersetzer bspw. in der CLR enthalten. Die JIT-Übersetzer gehen einen Mittelweg zwischen Interpretation und Übersetzung/Kompilierung, indem sie den jeweiligen Befehl oder eine Befehlsfolge des Zwischenkodes zunächst interpretieren und anschließend direkt in den Maschinenkode des jeweiligen Prozessors des Zielrechners übersetzen, woraufhin die Programmausführung unmittelbar erfolgen kann. Dadurch ist ein JIT-Übersetzer in der Lage, den Befehlssatz vom Prozessor des Zielrechners, auf dem die Programmausführung erfolgt, optimal auszunutzen. Bei Programmen, die schon während ihrer Erstellung in Maschinenkode übersetzt werden, ist eine Optimierung des Maschinenkodes zwar ebenso möglich, aber deren Ausführung ist i. Allg. nur auf Rechnern der gleichen Prozessorfamilie möglich. Ein Ausweg wäre die Erstellung einer (potentiellen) Viel-

Abb. C.2

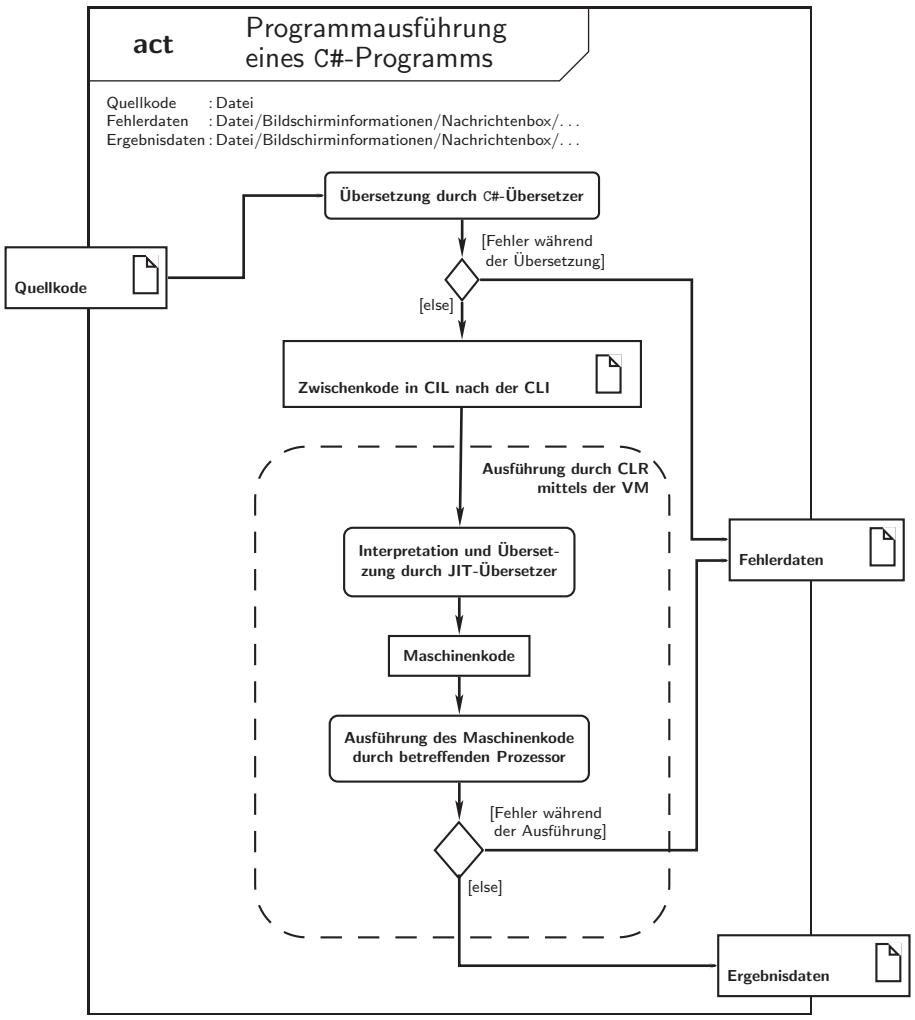


Abb. C.2.: *Programmausführung eines C#-Programms (UML-Aktivitätsdiagramm)*

zahl von verschiedenen Versionen eines Programms. Einen weiteren Ausweg stellt die Nutzung eines Interpreters dar, welcher für verschiedene Rechnerplattformen existiert. Interpreter sind zwar i. d. R. schneller zu erstellen als Übersetzer/Kompiler, besitzen aber i. Allg. eine wesentlich niedrigere Ausführungsgeschwindigkeit als diese.

C.3. Vorteile

Die meisten Vorteile, welche die Programmiersprache C# als unmittelbare Umsetzung der CLI mit sich bringt, sind auch unmittelbar an die Möglichkeiten und die Leistungsfähigkeit der Laufzeitumgebung, d. h. der CLR, gebunden. Eine schlecht implementierte Laufzeitumgebung macht die eigentlichen Vorteile der Programmiersprache C#, zu denen hauptsächlich die beliebige Anforderung und die automatische Freigabe von Ressourcen, z. B. Speicher, (automatische Ressourcenverwaltung [dt.; garbage collection [engl.]]) gehört, zunichte. Diese übernimmt die

Laufzeitumgebung, wodurch auch sichergestellt wird, dass geschützte Speicherbereiche nicht direkt angesprochen oder überschrieben werden können[120]. Der Softwareersteller hat darauf nur dann bedingten Einfluss, wenn er es, z. B. aus Gründen der Leistungssteigerung, für notwendig hält, eine Optimierung der Laufzeit oder Ressourcenverwaltung händisch vorzunehmen.

Des Weiteren sind die grundlegenden Datentypen von C# auf das einheitliche Typensystem der CIL ausgerichtet, so dass jedes Element als Objekt behandelt werden kann, ohne dass, wie bei rein objektorientierten Systemen, wie bspw. Smalltalk, größere Leistungseinbußen hingenommen werden müssen.

C.4. Nachteile

Der Schritt weg von der direkten Übersetzung in den Maschinenkode des Zielsystems führt auch einen Schritt weg von der Geschwindigkeit bei der Programmausführung (Erhöhung der Programmlaufzeit), aber einen Schritt hin zur Erhöhung der Effizienz bei der Programmentwicklung. Eben dieser Schritt kann bspw. bei Programmen mit Echtzeitanforderungen einen Flaschenhals darstellen, weil sich durch die automatische Ressourcenverwaltung nicht nur der Bedarf an Ressourcen erhöht und ein geringfügig zusätzlicher Zeitaufwand entsteht, sondern sich auch die Antwortzeiten von Programmereignissen wesentlich schwieriger berechnen lassen und teilweise deutlich länger ausfallen als erwartet. Dies kommt daher, dass nicht der Softwareersteller über den Zeitpunkt der Freigabe entscheidet, sondern die Laufzeitumgebung. Diese fasst zur Einsparung von Programmlaufzeit bestimmte Freigabeoperationen zusammen, wodurch seinerseits die Antwortzeiten auf Ereignisse in Mitleidenschaft gezogen werden.

Bei der Software-Entwicklung wird jedoch im i. Allg. von einer asymmetrischen Verteilung ausgegangen, d. h. dass ein Großteil einer (durchschnittlichen) Anwendung problemlos durch die automatische Ressourcenverwaltung überwacht werden kann, und lediglich ein deutlich geringerer Prozentsatz (z. B. einzelne Methoden oder Klassen) diesbezüglich optimiert werden muss.

C.5. Sprachbesonderheiten

Die objektorientierte Programmiersprache C# ist eine sich noch aktiv verändernde Programmiersprache, deren Sprachumfang nach wie vor erweitert wird, d. h. es werden zukünftig weitere Versionen der Programmiersprache entstehen. C# basiert aus verschiedenen Gründen im Wesentlichen auf den Programmiersprachen

[120] Diese Aussage trifft allerdings nur auf sog. verwalteten Programmkode [dt.; managed code [engl.]] zu, welcher auch tatsächlich vom JIT-Übersetzer erzeugt wurde. Alternativ dazu kann auch nicht verwalteter Programmkode (sog. unmanaged code) bei der Ausführung genutzt werden, welcher bereits ausführbaren Maschinenkode beinhaltet, dessen Ressourcenanforderungen und -freigaben nicht von der Laufzeitumgebung verwaltet werden.

C++, Visual Basic und Java. Ein Grund dafür ist, dass diese Programmiersprachen eine hohe Durchdringung des Marktes erreicht haben, wodurch sie von einem Großteil der Softwareersteller beherrscht werden. Somit bleiben sie auch in leicht abgewandelter Form, wie sie C# besitzt, für viele Softwareersteller verständlich.

Die zuvor angesprochene Spezifikation der Programmiersprache C# in der ECMA-334 beinhaltet aktuell u. a. folgende, im Vergleich zu den anderen Programmiersprachen jedoch neue Sprachkonstrukte und -besonderheiten[121] wie (siehe auch [Küh06])

Eigenschaften [dt.; properties [engl.]].
Besitzen vom inneren Aufbau her Ähnlichkeit zu den Klassen-Methoden, legen aber i. d. R. den öffentlichen Zugriff (Lesen [dt.; get [engl.]] und/oder Schreiben [dt.; set[engl.]]) auf die privaten Klassen-Variablen fest und dienen somit der Kapselung von privaten Daten und können evtl. mit zusätzlichen Prüfungen beim Zugriff versehen werden.

Benutzerdefinierte Attribute [dt.; custom attributes [engl.]].
Attribute bilden deklarative Informationen ab und dienen somit der Speicherung von Informationen über Klassen, Methoden und Eigenschaften, welche zur Entwicklungszeit festgelegt und zur Laufzeit ausgewertet werden können.

Delegaten [dt.; delegates [engl.]].
Legen den typsicheren[122] Verweis auf eine Methode fest und können zur Laufzeit Informationen über das aufrufende Objekt zurückliefern.

Reflexionen [dt.; reflection [engl.]].
In C# ist es möglich, während der Laufzeit eines Programmes neuen Programmkode über ein Objektmodell zu generieren und diesen somit unmittelbar in eine lauffähige Bibliothek zu übersetzen (dynamische Übersetzung).

Generische Typen [dt.; generics [engl.]].
Generische Typen können auf Strukturen, Schnittstellen, Delegaten sowie Methoden angewandt werden. Sie erlauben es dem Nutzer die Datentypen, die verarbeitet werden sollen, zunächst nur implizit vorzugeben, d. h. nicht genau zu spezifizieren, und diese erst bei der Übersetzung fest zu binden. Generische Typen sind in anderen Programmierspachen, z. B. C++, auch als Templates bekannt, dort aber mitunter nicht so mächtig und von nicht so weitreichender Bedeutung.

Anonyme Methoden [dt.; anonymous methods [engl.]].
Durch anonyme Methoden kann Programmkode direkt einem Delegaten zugewiesen werden und muss nicht zwingend auf eine entsprechende Ereignisbehandlungsroutine verweisen.

[121] Einige der Sprachkonstrukte und -besonderheiten wurden auch in andere Programmiersprachen übernommen. Sie waren aber zu ihrer Spezifikationszeit nur in der Programmiersprache C# vorhanden.

[122] C# ist eine typisierte Programmiersprache, so dass ungültige Typvereinbarungen bereits zur Übersetzungszeit erkannt werden. Alle Klassen und Datentypen sind von der Basisklasse System.Object abgeleitet.

Iteratoren [dt.; iterators [engl.]].
Mit Hilfe eines Iterators kann das sequentielle Durchlaufen einer Menge von Objekten beeinflusst werden.

Namensraum-Qualifizierer `Global::`.
Durch die Angabe des Schlüsselwortes `Global::` oder nur des zweifachen Doppelpunktes (`::`) wird angegeben, dass es sich um die Verwendung einer Definition (voll qualifizierter Klassenname) des globalen Namensraumes handelt.

Wertlose Wertetypen [dt.; nullable value types [engl.]].
Ermöglicht die Zuweisung des Wertes `null`, nicht nur für Referenztypen, sondern auch für Wertetypen, um diese bspw. als „leer“ vorzubelegen.

Partielle Typen [dt.; particial types [engl.]].
Mit Hilfe des Schlüsselwortes `partial` innerhalb der Deklaration eines neuen Types (vor der Angabe des Typnamens) kann dem Übersetzer angezeigt werden, dass die Definition eines Typs auf mehrere Dateien aufgeteilt ist. Dies soll u. a. zur Verbesserung der Übersichtlichkeit von Klassen, z. B. Trennung von Programmlogik, Datenzugriff und Oberflächenzugriff innerhalb einer Klasse bei einem mehrschichtigen Klassenmodell, dienen. Verschiedene Softwareersteller können so gemeinsam an einer Klasse arbeiten, ohne sich gegenseitig zu blockieren. Ein weiterer Anwendungsfall könnte vorsehen, dass ein Teil einer Klasse automatisch, z. B. durch ein CASE-Tool, generiert wird, währenddessen andere Teile händisch kodiert werden.

Statische Klassen [dt.; static classes [engl.]].
Eine statische Klasse besitzt Ähnlichkeit zu statischen/globalen Variablen, hat aber den Vorteil, dass nicht explizit eine globale Variable für eine solche Klasse angelegt werden muss. Dies verringert die Fehlerwahrscheinlichkeit, weil möglicherweise nicht mehrere globale Variablen für ein und dieselbe Klasse existieren. Diese könnten mitunter schwer in der internen Klassenstruktur zu finden sein. Des Weiteren entfällt, durch die Angabe eines (statischen) Konstruktors, die vor ihrem ersten Zugriff zwingend notwendige (explizite) Initialisierung der globalen Variablen.

Zugriffsschutzrecht `internal`.
Das Schlüsselwort `internal` beschränkt den Zugriff von Klassen, Variablen, Methoden und Eigenschaften auf die jeweilige Bibliothek, in der sie enthalten sind.

Die Sprachkonstrukte für generische Typen, anonyme Methoden, Iteratoren, nullbare Wertetypen sowie partielle und statische Klassen sind ab Sprachversion 2.0 hinzugekommen. Die anderen, zuvor genannten, Sprachkonstrukte waren bereits in der Sprachversion 1.0 enthalten und sind daher auch größtenteils im Softwaresystem `CAOS` angewandt worden.

C.6. Vergleich mit der Programmiersprache Java

Die Programmiersprachen C# und Java besitzen aus konzeptioneller Sicht große Ähnlichkeit. Beide Programmiersprachen sind streng typabhängig und objektorientiert. Zudem besitzen sie eine Ressourcenverwaltung und lassen keine Zugriffe über die vereinbarten Grenzen von Feldern hinaus zu, so dass ein Programmierer vor Fehlern bei Speicherzugriffen und falsch definierten Zeigern weitestgehend geschützt ist. C# geht dennoch weiter als Java, indem es mehrdimensionale Felder, Operatoren-Überladung und eine verbesserte Speicherverwaltung für nebenläufige Prozesse unterstützt[123]. Einer der wichtigsten Unterschiede ist allerdings, dass C#, ähnlich der Programmiersprache C++, auch leichtgewichtige Wertetypen zulässt, welche aber nach wie vor objektorientiert sind[124].

C.7. Frei verfügbare Entwicklungs- und Laufzeitumgebungen

Derzeit existieren neben dem .NET-Framework von Microsoft® weitere frei verfügbare[125] Laufzeitumgebungen für die CLI. Diese sind neben den aktuell auf dem Markt verfügbaren Entwicklungsumgebungen für die Programmiersprache C# in Tab. C.1 abgebildet, um die bereits vorherrschenden vielfältigen und plattformunabhängigen Anwendungs- und Entwicklungsmöglichkeiten aufzuzeigen.

C.8. Fazit

Die Programmiersprache C# stellt zusammen mit dem .NET-Framework grundlegende Konzepte zur schnellen Erstellung von Anwendungssoftware bereit. Die Nutzung von C# sollte dennoch für die einzelnen Komponenten einer Software gut abgewogen werden, um den aktuell oder später geforderten Geschwindigkeits- und Ressourcenanforderungen gerecht zu werden. Mitunter erweist sich die Entwicklung von Teilen der Software in einer anderen Programmiersprache und die Einbindung in den restlichen Teil der Software als weitaus unproblematischer im Hinblick auf gewünschte Anforderungen.

Der Schritt weg von der zeitlichen Leistungsfähigkeit einer Anwendung hin zu dessen schnellerer Programmierung, besseren Wartbarkeit, niedrigeren Fehleranfälligkeit und besseren Dokumentierbarkeit sollte also gut bedacht sein. Beim Softwaresystem CAOS stand die weitestgehende Plattformunabhängigkeit, Robustheit gegenüber verschiedenen Fehlerarten, die Wartbarkeit sowie die leichte Dokumen-

[123] Dies spiegelt nur die anfänglichen Vorteile von C# wider. Die Programmiersprache Java hat mittlerweile die meisten Merkmale auch umgesetzt.

[124] Bei der Entwicklung der Programmiersprache Java wurden im Gegenzug andere Merkmale eingeführt, um eine breitere Nutzung zu erzielen und verlorene Marktanteile zurück zu gewinnen.

[125] Frei verfügbar bedeutet in diesem Zusammenhang zumeist, dass die Laufzeitumgebungen kostenlos aus dem Internet herunter geladen und genutzt werden dürfen, der Anbieter der jeweiligen Laufzeitumgebung aber nicht für bestimmte auftretende Fehler verantwortlich gemacht werden kann.

Tab. C.1

Betriebssystem	Microsoft® Windows®	Unix, Linux, FreeBSD	Mac OS X
Frei verfügbare Entwicklungsumgebungen:			
Visual Studio 2005 (Express Edition)	✓	×	×
SharpDevelop (siehe [ICs07])	✓	×	×
KDevelop (siehe [RS07])	×	✓	×
Xcode (siehe [App07])	×	×	✓
Eclipse (siehe [Fou07])	✓	✓	✓
MonoDevelop (siehe [Mon07])	✓	✓	✓
Frei verfügbare Laufzeitumgebungen:			
.NET-Framework und .NET-Compact-Framework von Microsoft® (siehe [Sch07b])	✓	×	×
Rotor (siehe [Mic07b])	✓	✓	✓
Mono^{TM} (siehe [pro07a])	✓	✓	✓
DotGNU und Portable.NET (siehe [Bol07])	✓	✓	✓
Legende:			
✓ - möglich			
× - nicht möglich			

Tab. C.1.: *Frei verfügbare Entwicklungs- und Laufzeitumgebungen für die Programmiersprache C#*

tierbarkeit der Anwendung im Vordergrund, wodurch sich bereits während der Entwurfsphase für eine Entwicklung des gesamten Softwaresystems in der Programmiersprache C# entschieden wurde.

Kapitel D. Hinweise zum Softwaresystem CAOS

Inhalt

D.1	*Visuelle Beispiele*	*268*
D.2	*Mögliche Nebenbedingungen*	*273*

Abb. D.1

Abb. D.1.: *Logo des Softwaresystems CAOS*

D.1. Visuelle Beispiele

Im Nachfolgenden werden zur Gewinnung eines Überblicks über die vorgestellten Möglichkeiten des Softwaresystems CAOS einige visuelle Beispiele vorgestellt.

Modelldarstellung und -eingabe. Die Eingabe eines Modells, für welches im CAOS eine Deklaration hinterlegt ist, gestaltet sich so, wie es bspw. in Abb. D.2 verdeutlicht ist. Im linken Teil der Abbildung ist das Modell als Baumansicht mit den jeweiligen Kategorien verdeutlicht (vgl. Abschn. 2.4). Die Eingabe der konkreten Datenwerte für den ausgewählten Parameter oder die ausgewählte Variable ist im rechten Teil von Abb. D.2 zu sehen. Hierfür existieren in Abhängigkeit des Typs des Parameters oder der Variablen verschiedene Steuerelemente und Dialoge (s. u., Wahrscheinlichkeitsverteilungen).

Abb. D.2

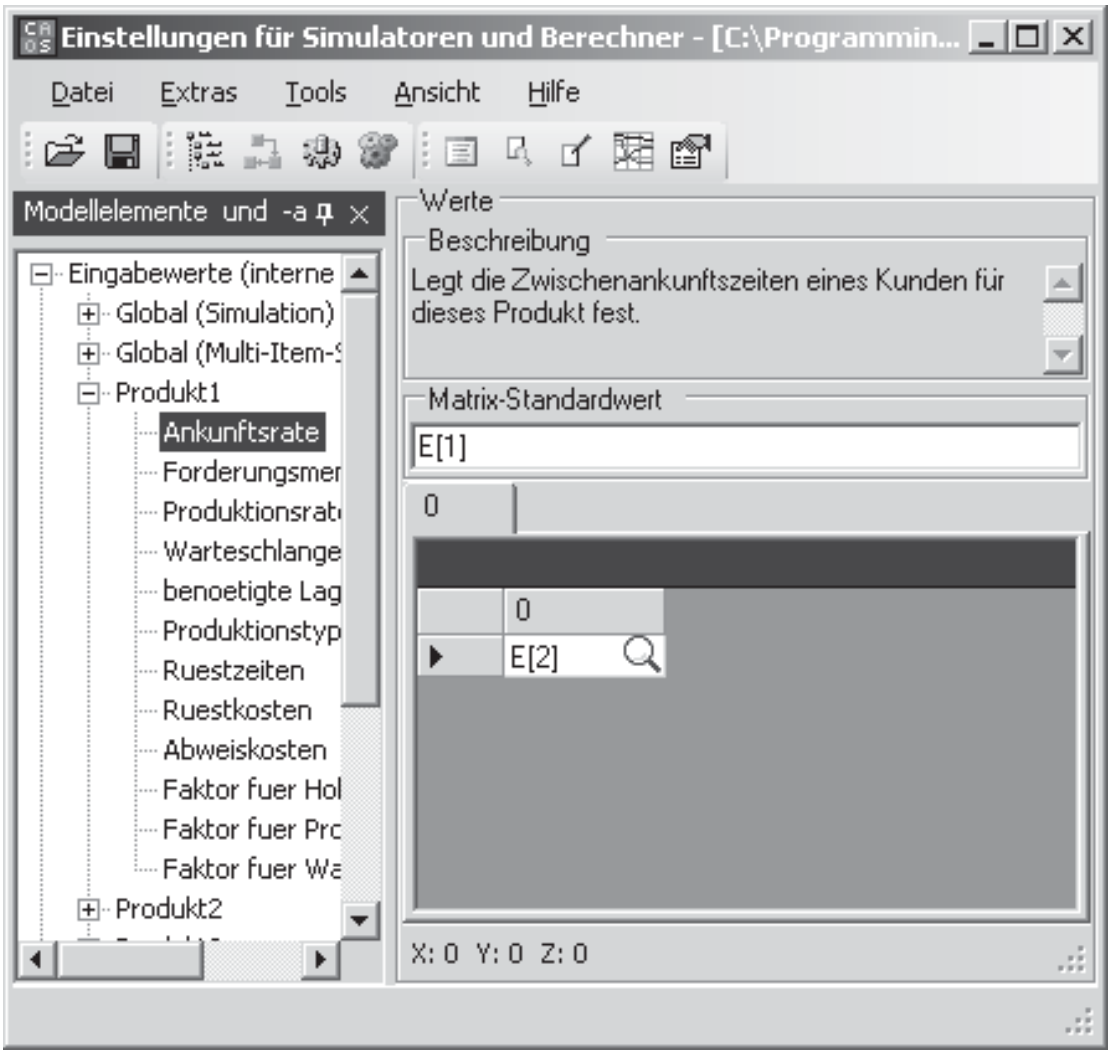


Abb. D.2.: *Oberfläche des Softwaresystems CAOS am Beispiel (Modellansicht)*

Wahrscheinlichkeitsverteilungen. Für die Eingabe von Wahrscheinlichkeitsverteilungen, kurz Verteilungen genannt, existiert im CAOS ein spezieller Dialog, welcher

- die Verteilungsfunktion und
- eine Menge von zufällig bestimmten, entsprechend der inversen Verteilungsfunktion transformierten Zufallszahlen

unmittelbar darstellt. Ein Beispiel dieses Dialoges ist in Abb. D.3 visualisiert. Die Abbildung zeigt die Eingabe einer logarithmischen Normalverteilung mit dem Erwartungswert von 0,5 und einer Varianz von 0,25. Diese spezielle Verteilung ist im Modell-Editor manuell als „`LN(0.5;0.25)`“ einzugeben. Dabei sind im oberen Teil der Abbildung die Parameter der Verteilung und im unteren Teil die graphische Darstellung der Verteilungsfunktion bzw. der transformierten Zufallszahlen zu sehen.

Abb. D.3

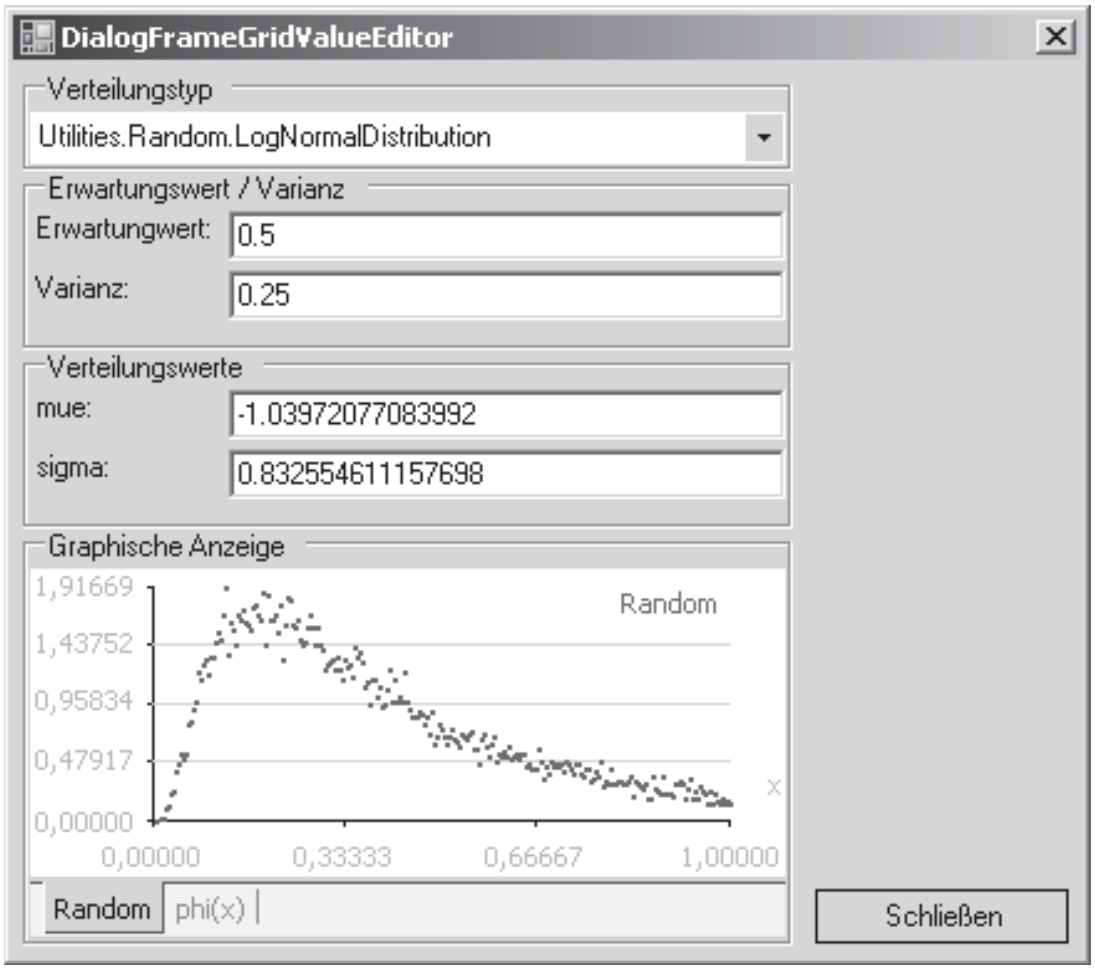


Abb. D.3.: *Oberfläche des CAOS am Beispiel (Eingabe einer Wahrscheinlichkeitsverteilung)*

Berechnungen und Simulationen. Für die Durchführung eines einzelnen Simulationsexperimentes oder einer einzelnen Berechnung ist im CAOS ein spezielles Oberflächen-Steuerelement vorhanden. Das Steuerelement wurde entwickelt, um ermittelte Simulationsergebnisse zu bestätigen und gezielt Simulationen oder Berechnungen als Voruntersuchungen im Vorfeld einer simulationsbasierten Optimierung durchzuführen.

In Abb. D.4 ist ein Beispiel für ein durchgeführtes Simulationsexperiment angegeben. Entscheidend sind die im linken Teil der Abbildung abgebildeten Ergebnisse des Simulationsexperimentes, welche sowohl die zu bewertenden als auch die nicht zu bewertenden Ausgabewerte beinhalten. Des Weiteren sind im rechten Teil der Abbildung auch die Werte für die in der Modelldefinition festgelegten Entscheidungsvariablen aufgelistet.

Abb. D.4

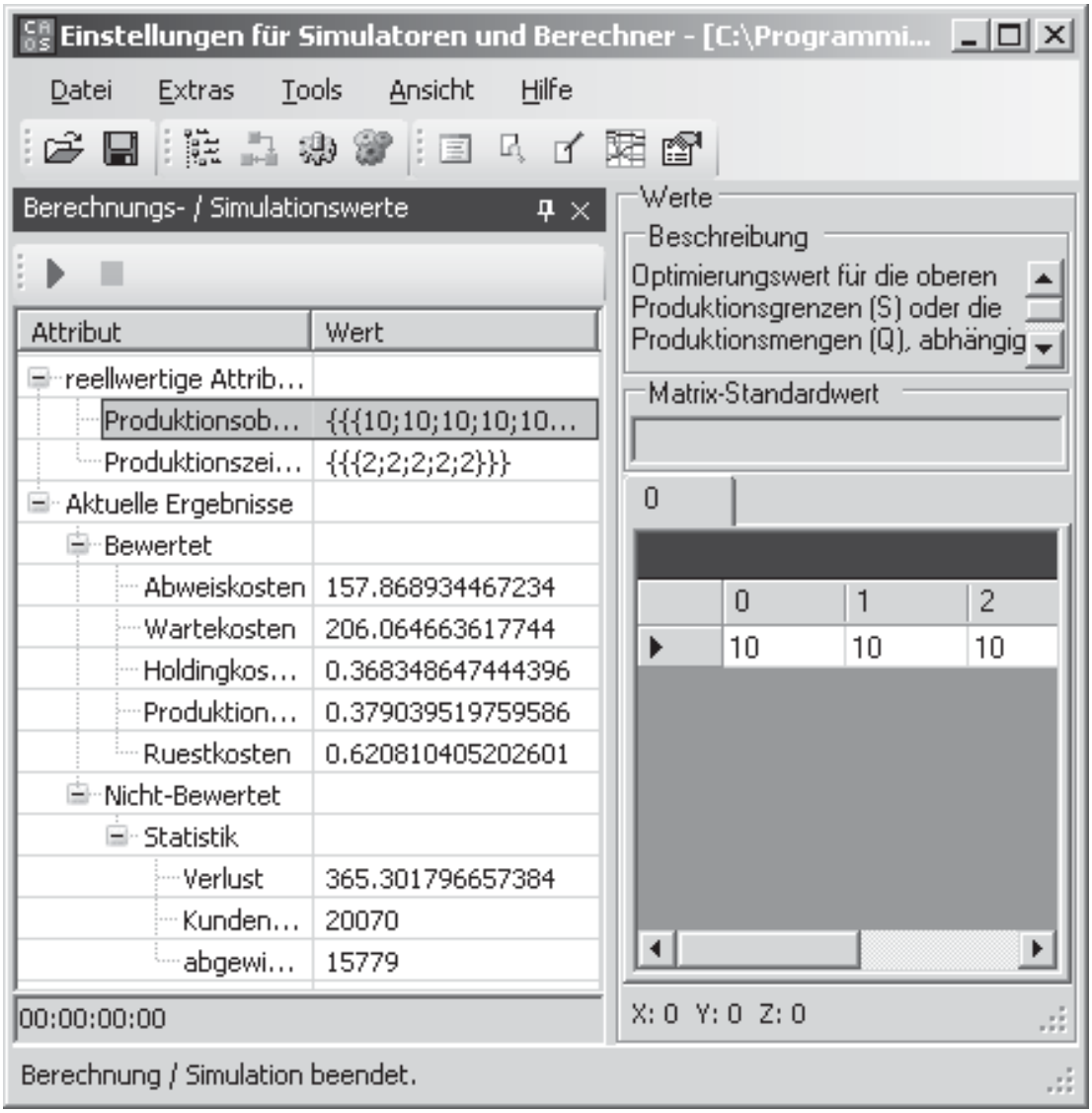


Abb. D.4.: *Oberfläche des CAOS am Beispiel (Simulationsansicht)*

Optimierungen und Optimierungsverlauf. Einzelne Läufe zur simulationsbasierten Optimierung einer zuvor festgelegten Modelldefinition können entweder mittels der Anwendung vom Typ Console (`CAOSc.exe`) oder mittels der Anwendung vom Typ WinForms (`CAOSw.exe`) durchgeführt werden (vgl. Abschn. 5.3.6 und Abschn. 5.3.7). Im Falle einer WinForms-Anwendung kann die graphische Oberfläche des CAOS genutzt werden, um den Optimierungsverlauf des Laufes der simulationsbasierten Optimierung unmittelbar darzustellen. Andernfalls erfolgt der Lauf der simulationsbasierten Optimierung bei Nutzung der Anwendung als Typ Console ohne Oberfläche und der entstandene, abgespeicherte Optimierungslauf kann im Nachhinein graphisch dargestellt werden.

Abb. D.5

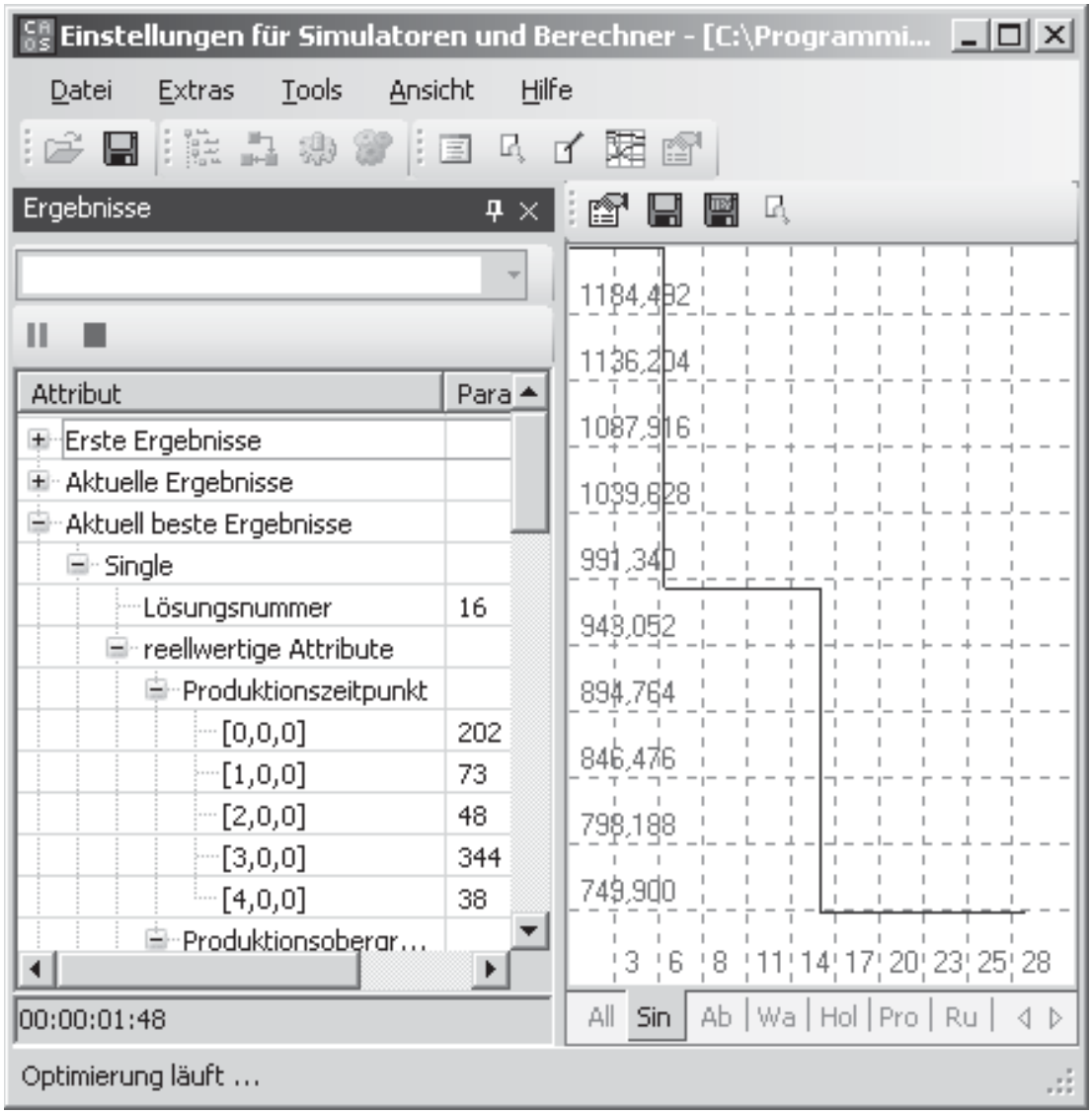


Abb. D.5.: *Oberfläche des CAOS am Beispiel (Optimierungsansicht)*

Sterndiagramme. Als spezielle Darstellungsmöglichkeit für mehrkriterielle Lösungen existiert neben der baumartigen Darstellung der Ergebnisse auch die Möglichkeit der Ausgabe der Pareto-optimalen Lösungen mittels eines Sterndiagrammes (vgl. Abschn. 2.2.5). Das Steuerelement für Sterndiagramme ist Bestandteil der graphischen Oberfläche zur Anzeige von Optimierungsverläufen. Ein solches Sterndiagramm ist in Abb. D.6 zu sehen. In diesem Beispiel zeigt es eine Pareto-optimale Lösung an, welche aus fünf Zielfunktionen besteht. An den (Stern-)Achsen sind die Zielfunktionswerte der aktuellen Pareto-optimalen Lösung und die maximalen Zielfunktionswerte aller Lösungen der aktuellen Pareto-optimalen Menge angegeben.

Abb. D.6

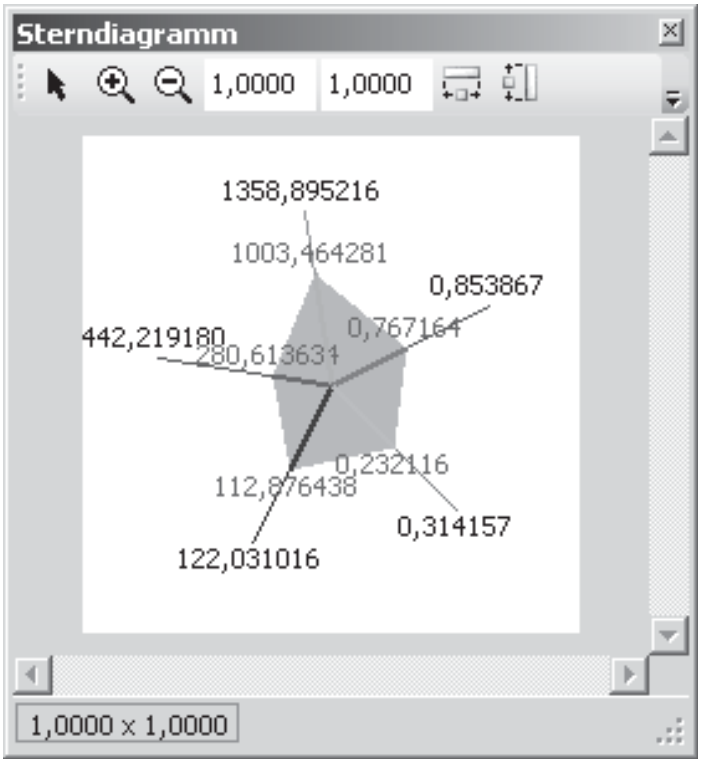


Abb. D.6.: *Oberfläche des* CAOS *am Beispiel (Sterndiagramm)*

D.2. Mögliche Nebenbedingungen

Grammatik der Nebenbedingungen. In der aktuellen Version des Softwaresystems CAOS können verschiedenste Nebenbedingungen angegeben werden. Diese sind alle auf folgende *LR-Grammatik* (siehe [LMW92]) zurückzuführen, welche in *Backus-Naur-Form* (*BNF*) angegeben ist.

```
Start              ::= Boolsche_Bedingung ";" ;
Boolsche_Bedingung ::= Boolsche_Bedingung Boolscher_Operator Vergleichsausdruck ;
Boolscher_Operator ::= "AND" | "OR" ;
Vergleichsausdruck ::= Ausdruck Vergleichsoperator Ausdruck ;
Vergleichsoperator ::= "<" | ">" | ">=" | "<=" | "=" | "\#" ;
Ausdruck           ::= Ausdruck Strich_Operator Term | "(" Ausdruck ")" ;
Strich_Operator    ::= "+" | "-" ;
Term               ::= Term Punkt_Operator Faktor ;
Punkt_Operator     ::= "*" | "/" ;
Faktor             ::= Nummerische_Konstante | Funktion | Entscheidungsvariable_reell
                     | Entscheidungsvariable_permut | Modellparameter ;
```

Die Nichtterminalsymbole `Nummerische_Konstante`, `Funktion`, `Entscheidungsvariable_reell`, `Entscheidungsvariable_permut` und `Modellparameter` werden aus Gründen der Komplexität der zugehörigen Ableitungsregeln (Produktionen) dieser Nichtterminalsymbole innerhalb dieser Arbeit nicht näher erklärt.

Beispiele für Nebenbedingungen. Nachfolgend sollen noch zwei Beispiele für mögliche Nebenbedingungen mit der zuvor definierten Grammatik aufgezeigt werden:

- Beispiel 1:

  ```
  Produktionszeitpunkt[0] < Produktionsobergrenze[0]
  ```

  Bei diesem Beispiel muss der erste Wert des Vektors der reellwertigen Entscheidungsvariablen „Produktionszeitpunkt“ kleiner als der erste Wert des Vektors der reellwertigen Entscheidungsvariablen „Produktionsobergrenze“ sein, damit die Nebenbedingung gültig ist.

- Beispiel 2:

  ```
  'Ergebnis\Ruestkosten pro ZE' < 0.15 + 0.2
  ```

  Für die Gültigkeit der Nebenbedingung muss der Wert des Attributs „Ruestkosten pro ZE“ vom Element „Ergebnis“ kleiner als 0,35 sein.

Literaturverzeichnis

[AB02] Asteroth, A. ; Baier, Ch.: *Theoretische Informatik: Eine Einführung in Berechenbarkeit, Komplexität und formale Sprachen mit 101 Beispielen.* Pearson Studium, 2002

[ABGR03] Antreich, K. (Hrsg.) ; Bulirsch, R. (Hrsg.) ; Gilg, A. (Hrsg.) ; Rentrop, P. (Hrsg.): *Modeling, Simulation, and Optimization of Integrated Circuits.* Birkhäuser Verlag, 2003 (Proceedings of a Conference held at the Mathematisches Forschungsinstitut Oberwolfach)

[ABZ88] In: Adams, J. ; Balas, E. ; Zawack, D.: *The Shifting Bottleneck Procedure for Job Shop Scheduling.* The Institute of Management Science, 1988, S. 391–401

[AF05] Arnold, D. ; Furmans, K.: *Materialfluss in Logistiksystemen.* Springer Verlag, 2005

[AFL+00] *Kapitel* Algorithms for the Online travelling Salesman. In: Ausiello, G. ; Feuerstein, E. ; Leonardi, S. ; Stougie, L. ; Talamo, M.: *Algorithmica.* Springer Verlag, 2000

[AG88] Assad, A. A. ; Golden, B. L.: *Studies in Management Science and Systems.* Bd. 16: *Vehicle Routing: Methods and Studies.* North-Holland, 1988

[Akl80] Akl, S.: The minimal directed spanning graph for combinatorial. In: *Austral. Comput. J.* 4 (1980), Nr. 12, S. 132–136

[Ale94] Alexopoulos, Ch.: A review of advanced methods for simulation output analysis. In: Tew, J. D. (Hrsg.) ; Mnivannan, S. (Hrsg.) ; Sadowski, D. A. (Hrsg.) ; Seila, A. F. (Hrsg.): *Proceedings of the 26th conference on Winter simulation.* Orlando, Florida, USA : Society for Computer Simulation International, San Diego, CA, USA, 1994, S. 133–140

[And90] Anderson, S. L.: Random number generators on vector supercomputers and other advanced architectures. In: *SIAM Rev.* 32 (1990), S. 221–251

[App07] Apple Inc.: *Apple - Mac OS X - Xcode.* Version: April 2007. `http://www.apple.com/de/macosx/features/xcode/`, Abruf: 19.04.2007. Internet-Quelle

[Atk94] Atkins, D. R.: A simple lower bound to the dynamic assembly problem. In: *European journal of operational research* 75 (1994), Nr. 2, S. 462–466

[Aut99] Autosimulation Inc.: *Autosimulation: Help for Autostat for Automod v 1.3 - Studentversion.* 1999

[AZ04] Azizi, N. ; Zolfaghari, S.: Adaptive temperature control for simulated annealing: a comparative study. In: *Computers and Operations Research* 31 (2004), Dezember, Nr. 14, S. 2439–2451

[Aze92] Azencott, R. (Hrsg.): *Simulated annealing: parallelization techniques.* Wiley, 1992

[Bäc96] Bäck, T.: *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, 1996

[BAL+04] Brinkmann, K. ; Auch, E. ; Lehmann, D. ; Riemenschneider, K. M. ; Heidelbach, O.: Alternativenanalyse: Darstellung und Bewertung der Szenarien. In: Rusdea, E. (Hrsg.) ; Reif, A. (Hrsg.) ; Povara, I. (Hrsg.) ; Konold, W. (Hrsg.): *Perspektiven für eine traditionelle Kulturlandschaft in Osteuropa. Ergebnisse eines inter- und transdisziplinären Forschungsprojektes in Osteuropa.* Culterra 34, Freiburg, 2004, S. 307–317

[Bal05] Balzert, H.: *Lehrbuch Grundlagen der Informatik: Konzepte und Notationen in UML 2, Java 5, C++ und C#.* 2. Aufl. Elsevier, 2005

[Ban98] Banks, J. (Hrsg.): *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice.* John Wiley & Sons, Inc., 1998

[BBD+87] Boyle, J. ; Butler, R. ; Disz, T. ; Glickfeld, B. ; Lusk, E. ; Overbeek, R. ; Patterson, J. ; Stevens, R.: *Portable Programs for Parallel Processors.* Holt, Rinehart and Winston, 1987

[BDPNN06] Brahimi, N. ; Dauzere-Peres, S. ; Najid, N. M. ; Nordli, A.: Single item lot sizing problems. In: *European Journal of Operational Research* 168 (2006), S. 1–16

[BFM97] Bäck, T. ; Fogel, D. ; Michalewicz, Z.: *Handbook of Evolutionary Computation.* Oxford University Press, 1997

[BINN01] Banks, J. ; II, J. S. C. ; Nelson, B. L. ; Nicol, D. M. ; Fabrycky, W. J. (Hrsg.) ; Mize, J. H. (Hrsg.): *Discrete-Event System Simulation.* 3. Aufl. Prentice Hall, 2001 (International Series in Industrial and Systems Engineering)

[BKPR03] Bock, H. G. (Hrsg.) ; Kostina, E. (Hrsg.) ; Phu, H. X. (Hrsg.) ; Rannacher, R. (Hrsg.): *Modeling, Simulation and Optimization of Complex Processes.* Springer Verlag, 2003 (Proceedings of the International Conference on High Performance Scientific Computing)

[BKS95] Branke, J. ; Kohlmorgen, U. ; Schmeck, H.: Steuerung einer Heuristik zur Losgrößenplanung unter Kapazitätsbeschränkungen mit Hilfe eines parallelen genetischen Algorithmus. In: Kuhl, J. (Hrsg.): *Evolutionäre Algorithmen in Management- Anwendungen*. Institut für Angewandte Informatik und Formale Beschreibungsverfahren : Universität Karlsruhe, 1995, S. 21–31

[Boa07] Board, OpenMP Architecture R.: *An API for multi-platform shared-memory parallel programming in C/C++ and Fortran*. Version: Juli 2007. `http://www.openmp.org/`, Abruf: 11.07.2007. Internet-Quelle

[Bog98] Bogatzki, A.: *Fabrikplanung - Verfahren zur Optimierung der Maschinenaufstellung*. S. Roderer Verlag, 1998

[Bol07] Bollow, N.: *Das DotGNU-Projekt*. Version: April 2007. `http://www.dotgnu.org/`, Abruf: 19.04.2007. Internet-Quelle

[BP82] Bartholdi, J. ; Platzmann, L.: An O(N log N) Planar Travelling Salesman Heuristic Based on Spacefilling Curves. In: *Operations Research Letters* 1 (1982), S. 121–125

[Bru04] Brucker, P.: *Scheduling Algorithms*. fourth Edition. Berlin, Heidelberg, New York : Springer Verlag, 2004

[BS00] Bonnans, J. F. ; Shapiro, A.: *Perturbation Analysis of Optimization Problems*. 1. Springer Verlag, 2000

[BS02a] Beyer, H.-G. ; Schwefel, H.-P.: Evolution Strategies: A Comprehensive Introduction. In: *Journal Natural Computing* 1 (2002), Nr. 1, S. 3–52

[BS02b] Brenner, S. C. ; Scott, L. R.: *The Mathematical Theory of Finite Element Methods*. Springer Verlag, 2002

[BS03] Büchelin, Th. ; Stäuble, B.: Einführungskurs UML / I&T-Management/E-Business. Version: Juni 2003. `http://www.fhbb.ch/wirtschaft/arbeiten_entw/do08/UML_end.pdf`. 2003. – Fachrichtungsarbeit

[Cer85] Černý, V.: A Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. In: *Journal of Optimization Theory and Applications* 45 (1985), Nr. 1, S. 41–51

[Che] Chemnitz, Technische U.: *CHiC*. `http://www.tu-chemnitz.de/chic/`, Abruf: 19.04.2007. Internet-Quelle

[Che07a] Chemnitz, Technische U.: *CLiC: Überblick*. Version: Januar 2007. `http://www.tu-chemnitz.de/urz/clic/`, Abruf: 23.03.2007. Internet-Quelle

[Che07b] Chemnitz, Technische U.: *OpenPBS - Portable Batch System 2.3*. Version: Juli 2007. `http://www.tu-chemnitz.de/urz/anwendungen/hpc/pbs/OpenPBS-2.3-p15/index.html`, Abruf: 10.07.2007. Internet-Quelle

[Chr76] Christofides, N.: Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem / Carnegie-Mellon University. Graduate School of Industrial Administration, Pittsburgh, PA, 1976 (388). – Management Sciences Research Report

[CL04] Coello, C. A. C. (Hrsg.) ; Lamont, G. B. (Hrsg.): *Advances in Natural Computation*. Bd. 1: *Applications of Multi-Objective Evolutionary Algorithms*. World Scientific, 2004

[Cla03] Clark, A. R.: Hybrid heuristics for planning lot setups and sizes. In: *Computers and Industrial Engineering* 45 (2003), Dezember, Nr. 4, S. 545–562

[CM97] Carson, Y. ; Maria, A.: Simulation Optimization: Methods and Applications. In: Andatóttir, S. (Hrsg.) ; Healy, K. J. (Hrsg.) ; Withers, D. H. (Hrsg.) ; Nelson, B. L. (Hrsg.): *Procceding of the 1997 Winter Simulation Conference*, 1997

[CMTK05] Chan, T. M. ; Man, K. F. ; Tang, K. S. ; Kwong, S.: A Jumping Gene Algorithm for Multiobjective Resource Management in Wideband CDMA Systems. In: *The Computer Journal* 48 (2005), Nr. 6, S. 749–768

[Con63] Conway, R. W.: Some tactical problems in digital simulation. In: *Management Science* 10 (1963), Nr. 1, S. 47–61

[Cor02] Corrêa, R. (Hrsg.): *Models for parallel and distributed computation: theory, algorithmic techniques and applications*. Kluwer Academic Publishers, 2002

[Cor03] Cormen, Th. H. : *Introduction to algorithms*. MIT Press, 2003

[Cra85] Cramer, N. L.: A representation for the Adaptive Generation of Simple Sequential Programs. In: Grefenstette, J. J. (Hrsg.) ; Carnegie Mellon University (Veranst.): *International Conference on Genetic Algorithms and the Applications* Carnegie Mellon University, 1985

[DAPM00] Deb, K. ; Agrawal, S. ; Pratap, A. ; Meyarivan, T.: A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In: Schoenauer, M. (Hrsg.) ; Deb, K. (Hrsg.) ; Rudolph, G. (Hrsg.) ; Yao, X. (Hrsg.) ; Lutton, E. (Hrsg.) ; Merelo, J. J. (Hrsg.) ; Schwefel, H.-P. (Hrsg.): *Proceedings Sixth International Conference Parallel Problem Solving from Nature (PPSN VI)* Bd. 1917. Paris : Springer Verlag, 16.-20. September 2000 (LNCS), S. 849–858

[DB04] Dumbill, E. ; Bornstein, N. M.: *Mono. A Developer's Notebook*. 1. Beijing, Köln : O'Reilly, 2004

[DC05] Das, A. (Hrsg.) ; Chakrabarti, B. K. (Hrsg.): *Lecture Notes in Physics*. Bd. 679: *Quantum Annealing and Related Optimization Methods*. Heidelberg : Springer Verlag, 2005

[DM96] Dellaert, N. P. ; Melo, M. T.: Stochastic lot-sizing: solution and heuristic methods. In: *International Journal of Production Economics* 46-47 (1996), S. 261–276

[Dor92] Dorigo, M.: *Ottimizzazione, Apprendimento Automatico, ed Algoritmi Basati su Metafora Naturale (Optimization, Learning and Natural Algorithms)*. Politecnico di Milano, Italy, Politecnico di Milano, Ph. D. thesis, 1992. – in Italian

[DPAM02] Deb, K. ; Pratap, A. ; Agrawal, S. ; Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. In: *IEEE Trans. Evolutionary Computing* 6 (2002), S. 182–197

[DS04] Dorigo, M. ; Stützle, Th.: *Ant Colony Optimization*. MIT Press, 2004

[Due06] Dueck, G.: *Das Sintflutprinzip: Ein Mathematik-Roman*. Springer Verlag, 2006

[Dyn07] Dynamics, Incontrol E.: *SIMULATE - Enterprise Dynamics.com*. Version: Juli 2007. `http://incontrol.nl/`, Abruf: 13.07.2007. Internet-Quelle

[Ebl96] Eblenkamp, M.: *Planung von Produktionssystemen mit Evolutionsverfahren*, Universität Hannover, Fortschritt-Berichte, 1996

[Ecm06a] Ecma International: *Standard ECMA-334 – C# Language Specification*. Version: Juni 2006. `http://www.ecma-international.org/publications/standards/Ecma-334.htm`, Abruf: 26.04.2007. Internet-Quelle

[Ecm06b] Ecma International: *Standard ECMA-335 – Common Language Infrastructure (CLI)*. Version: 4, Juni 2006. `http://www.ecma-international.org/publications/standards/Ecma-335.htm`, Abruf: 26.04.2007. Internet-Quelle

[EH93] Eichenauer-Herrmann, J.: Statistical independence of a new class of inversive congruential pseudorandom numbers. In: *Math. Comp.* 60 (1993), S. 375–384

[Ehr00] Ehrgott, M.: *Lecture Notes in Economic and Mathematical Systems*. Bd. 491: *Multicriteria Optimization*. Springer Verlag, 2000

[Eic02] Eickhoff, M.: *Statistische Auswertung und Erkennung der stationären Phase in der Simulation zustands-diskreter Systeme.* Fachbereich Informatik, Lehrstuhl IV, Universität Dortmund, Deutschland, Diplomarbeit, Januar 2002

[EL86] Eichenauer, J. ; Lehn, J.: A non-linear congruential pseudo random number generator. In: *Statist. Papers* 27 (1986), S. 315–326

[Elm78] Elmaghraby, S. E.: The economic lot scheduling problem (ELSP): review and extensions. In: *Management Science* 24 (1978), S. 587–598

[EMH01] Erickson, M. ; Mayer, A. ; Horn, J.: The Niched Pareto Genetic Algorithm 2 applied to the design of groundwater remediation systems. In: Zitzler, E. (Hrsg.) ; Deb, K. (Hrsg.) ; Thiele, L. (Hrsg.) ; Coello, C. A. C. (Hrsg.) ; Corne, D. (Hrsg.): *Proceedings First Inernational Conference Evolutionary Multi-Criterion Optimization.* Zürich, Schweiz : Springer Verlag, 7.-9. März 2001, S. 681–695

[ES03] Eiben, A. E. ; Smith, J. E.: *Introduction to Evolutionary Computing.* Springer Verlag, 2003

[Fen05] Fentress, S. W.: *Exaptation as a means of evolving complex solutions*, University of Edinburgh, MA Thesis, 2005

[FF93] Fonseca, C. M. ; Fleming, P. J.: Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In: Forrest, S. (Hrsg.): *Proceedings Fifth International Conference Genetic Algorithms.* Urbana-Champaign, IL : Morgan Kaufmann Publishers, San Mateo, 17.-21. Juni 1993, S. 416–423

[FF98] Fonseca, C. M. ; Fleming, P. J.: Multiobjective optimization and multiple constraint handling with evolutionary algorithms – Part I: a unified formulation. In: *IEEE Transactions Systems, Management, and Cybernetics* 28 (1998), S. 26–37

[FG71] Freiberger, W. ; Grenander, U.: *A Short Course in Computational Probability and Statistics.* Springer Verlag, 1971

[FGA05] Fu, M. C. ; Glover, F. ; April, J.: Simulation optimization: a review, new developments, and applications. In: *Winter Simulation Conference.* Orlando, FL, USA : ACM, Dezember 2005, S. 83–95

[Fis71] Fishman, G. S.: Estimating sample size in computer simulation experiments. In: *Management Science* 18 (1971), S. 21–38

[Fis73] Fishman, G. S.: Statistical analysis for queueing simulation. In: *Management Science* 20 (1973), S. 363–369

[Fis01] Fishman, G. S.: *Discrete-event simulation.* Springer Verlag, 2001 (Series in operations research)

[FK96] Federgruen, A. ; Katalan, Z.: The Stochastic Economic Lot Scheduling Problem: Cyclical Base-stock Policies with Idle Times. In: *Management Science* 42 (1996), Nr. 6, S. 783–796

[FKPT04] Fahrmeir, L. ; Künstler, R. ; Pigeot, I. ; Tutz, G.: *Statistik*. Springer Verlag, 2004

[Fle90] Fleischmann, B.: The discrete lot-sizing and scheduling problem. In: *European Journal of Operational Research* 44 (1990), S. 337–348

[Fle94] Fleischmann, B.: The discrete lot-sizing and scheduling problem with sequence-dependent setup costs. In: *European Journal of Operational Research* 75 (1994), S. 395–404

[FMT04] Federgruen, A. ; Meissner, J. ; Tzur, M.: Progressive Interval Heuristics for Multi-Item Capacitated Lot Sizing Problems. In: *Operations Research* (2004)

[Fog07] Fog, Agner: *Pseudo random number generators*. Version: Februar 2007. `http://www.agner.org/random/`, Abruf: 03.02.2007. Internet-Quelle

[For94] Forum, Message Passing I.: MPI: A message-assing interface standard. In: *International Journal of Supercomputer Applications* 8 (1994), Nr. 3/4, S. 165–414

[For98] Forum, Message Passing I.: MPI2: A Message Passing Interface standard. In: *International Journal of High Performance Computing Applications* 12 (1998), Nr. 1/2, S. 1–299

[Fou07] Foundation, The E.: *Eclipse - an open development platform*. Version: April 2007. `http://www.eclipse.org`, Abruf: 19.04.2007. Internet-Quelle

[FOW66] Fogel, L. J. ; Owens, A. J. ; Walsh, M. J.: *Artificial Intelligence through Simulated Evolution*. John Wiley, 1966

[FT91] Federgruen, A. ; Tzur, M.: A Simple Forward Algorithm to Solve General Dynamic Lot Sizing Models with n Periods in 0(n log n) or 0(n) Time. In: *Management Science* 37 (1991), August, Nr. 8, S. 909–925

[Fu01a] Fu, M. C.: Simulation Optimization. In: Peters, B. A. (Hrsg.) ; Smith, J. S. (Hrsg.) ; Medeiros, D. J. (Hrsg.) ; Rohrer, M. W. (Hrsg.): *Proceedings of the 2001 Winter Simulation Conference*, 2001, S. 53–61

[Fu01b] Fu, M. C.: Simulation Optimization. In: Gass, S. (Hrsg.) ; Harris, C. (Hrsg.): *Encyclopedia of Operations Research and Management Science*. 2nd. Boston : Kluwer Academic Publishers, 2001, S. 756–759

[Fuj00] Fujimoto, R. M.: *Parallel and Distributed Simulation Systems*. Wiley-Interscience, 2000 (Wiley Series on Parallel and Distributed Computing)

[GAM78] Gafarian, A. V. ; Ancker, C. J. ; Morisaku, T.: Evaluation of commonly used rules for detecting steady state in computer simulation. In: *Naval Research Logistics Quarterly* 1 (1978), S. 511–529

[GBE+86] Göpfert, A. (Hrsg.) ; Bittner, L. (Hrsg.) ; Elster, K.-H. (Hrsg.) ; Nozicka, F. (Hrsg.) ; Piehler, J. (Hrsg.) ; Tichatschke, R. (Hrsg.): *Optimierung und optimale Steuerung*. Berlin : Akademie-Verlag, 1986 (Lexikon der Optimierung)

[GDBM01] Gopalakrishnan, M. ; Ding, K. ; Bourjolly, J.-M. ; Mohan, S.: A Tabu-Search Heuristic for the Capacitated Lot-Sizing Problem with Set-up Carryover. In: *Management Science* 47 (2001), Juni, Nr. 6, S. 851–863

[Gei97] Geist, A.: *PVM: a users' guide and tutorial for networked parallel computing*. 4. Aufl. Cambridge, MA : MIT Press, 1997 (Scientific and engineering computation)

[Ges01] Gesellschaft für Betriebsorganisation und Unternehmensplanung mbH: *SimRunner2! Warum SinRunner*. Version: 2001. `http://www.gbumbh.de/`. Internet-Quelle

[GF98] Görg, C. ; Fuss, O.: Simulating Rare Event Details of Delay Time Distributions with RESTART/LRE for ATM Reference Connections. Ilkley, U.K., Jul 1998

[GI05] Grünert, T. ; Irnich, S.: *Optimierung im Transport*. Bd. 2: *Wege und Touren*. Aachen : Shaker Verlag, 2005

[GKK04] Gerdes, I. ; Klawonn, F. ; Kruse, R.: *Evolutionäre Algorithmen*. Vieweg, 2004

[GL93] Glover, F. ; Laguna, M.: Tabu Search. In: Reeves, C. (Hrsg.): *Modern Heuristic Techniques for Combinatorial Problems*. Oxford, England : Blackwell Scientific Publishing, 1993

[GL97] Glover, F. ; Laguna, M.: *Tabu Search*. Norwell, MA : Kluwer Academic Publishers, 1997

[GLLK79] Graham, R. L. ; Lawler, E. L. ; Lenstra, J. K. ; Kan, A. H. G. R.: Optimization and approximation in deterministic sequencing and scheduling: A survey. In: *Ann. Discrete Math.* 5 (1979), S. 287–326

[Glo86] Glover, F.: *Future paths for integer programming and links to artificial intelligence*. Oxford, UK : Elsevier, 1986. – 533–549 S. – 1986

[Glo89] Glover, F.: Tabu search: Part I. In: *Operations Research Society of America (ORSA) Journal on Computing* Bd. 1, 1989, S. 190–206

[Glo90] Glover, F.: Tabu search: Part II. In: *Operations Research Society of America (ORSA) Journal on Computing* Bd. 2, 1990, S. 4–32

[GLS94] Gropp, W. ; Lusk, E. ; Skjellum, A.: *Using MPI: portable parallel programming with the message-passing interface*. Cambridge, MA : MIT Press, 1994 (Scientific and engineering computation)

[GLS99] Gropp, W. ; Lusk, E. ; Skjellum, A.: *Using MPI: portable parallel programming with the message-passing interface*. 2. Aufl. Cambridge, MA : MIT Press, 1999 (Scientific and engineering computation)

[GLT99] Gropp, W. ; Lusk, E. ; Thakur, R.: *Using MPI-2: advanced features of the message-passing interface*. Cambridge, MA : MIT Press, 1999 (Scientific and engineering computation)

[GM74] Gillett, M. ; Miller, L.: An new heuristic algorithm for the vehicle dispatching problem. In: *Operations Research* 22 (1974), S. 240–349

[GMR+08] Goossens, M. ; Mittelbach, F. ; Rahtz, S. ; Roegel, D. ; Voss, H.: *The LATEX graphics companion: tools and techniques for computer typesetting*. 2. ed. Upper Saddle River, NJ : Addison-Wesley, 2008 (Addison-Wesley series on tools and techniques for computer typesetting)

[Göh05] Göhler, W. ; Ralle, B. (Hrsg.): *Formelsammlung Höhere Mathematik*. 16. Aufl. Deutsch Harri GmbH, 2005

[Gro07] Group, GoldSim T.: *Monte Carlo Simulation Software, Decision & Risk Analysis*. Version: Juli 2007. `http://www.goldsim.com`, Abruf: 13.07.2007. Internet-Quelle

[Gru73] Grube, A.: Mehrfach rekursiv-erzeugte Pseudo-Zufallszahlen. In: *Zeitschrift für angewandte Mathematik und Mechanik* 53 (1973), S. T223–T225

[Grü94] Grün, O. ; Schweitzer, M. (Hrsg.): *Industrielle Materialwirtschaft: Das Wirtschaften in Industrieunternehmungen*. München : Vahlen, 1994 (Vahlens Handbücher der Wirtschafts- und Sozialwissenschaften)

[GT93] Grossmann, C. ; Terno, J.: *Numerik der Optimierung*. Stuttgart : Teubner, 1993

[GT05a] Gilbert, N. ; Troitzsch, K.: *Simulation for the Social Scientist*. Open University Press, 2005

[GT05b] Günther, H.-O. ; Tempelmeier, H.: *Produktion und Logistik*. 6. Aufl. Springer Verlag, 2005

[GYD+06] Guo, M. G. v. (Hrsg.) ; Yang, L. T. (Hrsg.) ; DiMartino, B. (Hrsg.) ; Zima, H. (Hrsg.) ; Dongarra, J. (Hrsg.) ; Tang, F. (Hrsg.) ; ISPA (Veranst.): *Parallel and Distributed Processing and Applications*. Sorrento, Italy : Springer Verlag, Dezember 2006 (Lecture Notes in Computer Science)

[Had98] Hader, S.: Das C++ Schnittstellen-Modul I_FACE / Technische Universität Chemnitz. Version: 1998. `http://www.sven-hader.de/papers/h9804.pdf`. Fakultät für Informatik, Professur Modellierung und Simulation, 1998. – Dokumentation

[Had01] Hader, S.: *Ein hybrider Ansatz zur Optimierung technischer Systeme*. Aachen : Shaker Verlag, 2001 (Informatik)

[Han86] Hansen, P.: The steepest ascent mildest descent heuristic for combinatorial programming. In: *Proceedings of Congress on Numerical Method in Combinatorial Programming*, 1986

[Har13] Harris, F. W.: How many parts to make at once. In: *The Magazine of Management* 10 (1913), S. 135–136

[Har15] Harris, F. W.: *Operations Cost*. Chicago : Shaw, 1915 (Factory Management Series)

[HGZ05] Hobbs, M. v. (Hrsg.) ; Goscinski, A. (Hrsg.) ; Zhou, W. (Hrsg.) ; ICA3PP (Veranst.): *Distributed and Parallel Computing*. Melbourne, Australia : Springer Verlag, Oktober 2005 (Lecture Notes in Computer Science)

[HHS91] Hempel, R. ; Hoppe, H.-C. ; Supalov, A.: *PARMACS 6.0 Library Interface Specification*. 1991

[HHT05] Helwig, S. ; Haubelt, Ch. ; Teich, J.: Modeling and Analysis of Indirect Communication in Particle Swarm Optimization. In: *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC2005)* Bd. 2. Edinburgh, Scotland : IEEE Press, September 2005, S. 1246–1253

[HK96] Haase, K. ; Kohlmorgen, U.: Parallel Genetic Algorithm for the Capacitated Lot-Sizing Problem. In: Kleinschmidt et al. (Hrsg.): *Operations Research Proceedings*, 1996

[Hol69] Holland, J. H.: A new kind of turnpike theorem. In: *Bulletin of the American Mathematical Society* 75 (1969), S. 1311–1317

[Hol75] Holland, J. H.: *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI : University of Michigan Press, 1975

[HOP07] Haugen, K. ; Olstad, A. ; Pettersen, B.: Solving Large-scale Profit Maximization Capacitated Lot-size Problems by Heuristic Methods. In: *Journal of Mathematical Modelling and Algorithms* 6 (2007), März, Nr. 1, S. 135–149

[Hsu83] Hsu, W.: On the General Feasibility Test of Scheduling Lot Sizes for Serveral Products on One Machine. In: *Management Science* 29 (1983), Nr. 1, S. 93–105

[ICs07] ICsharp Code: *The Open Source Development Environment for .NET*. Version: April 2007. `http://www.icsharpcode.com /OpenSource/SD/Default.aspx`, Abruf: 19.04.2007. Internet-Quelle

[Int07] *Deutsches Institut für Normung : Homepage DE*. Version: August 2007. `http://www.din.de/`, Abruf: 23.08.2007. Internet-Quelle

[ISO06a] ISO Standards Maintenance Portal: *Information technology – Common Language Infrastructure (CLI) – Technical Report on Information Derived from Partition IV XML File*. Version: 2, 2006. `http://standards.iso.org/ittf/PubliclyAvailableStandards /c042928_ISO_IEC_TR_23272_2006(E).zip`, Abruf: 26.04.2007. Internet-Quelle

[ISO06b] ISO Standards Maintenance Portal: *Information technology – Common Language Infrastructure (CLI) Partitions I to VI*. Version: 2, 2006. `http://standards.iso.org/ittf/Publicly AvailableStandards/c042927_ISO_IEC_23271_2006(E).zip`, Abruf: 26.04.2007. Internet-Quelle

[ISO06c] ISO Standards Maintenance Portal: *Information technology – Programming languages – C#*. Version: 2, 2006. `http://standards.iso.org/ittf/PubliclyAvailableStandards /c042926_ISO_IEC_23270_2006(E).zip`, Abruf: 26.04.2007. Internet-Quelle

[Jai91] Jain, R.: *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, Inc., 1991

[Käm03] Kämpf, M.: *Über parallele Lösungen des Fleet-Sizing-and-Allocation-Problems*. Fakultät für Informatik, Professur Modellierung und Simulation, Technische Universität Chemnitz, Diplomarbeit, Juni 2003

[Käm06] Kämpf, M.: Probleme der Tourenbildung / Technische Universität Chemnitz, Fakultät für Informatik. 2006 (CSR-06-04). – Chemnitzer Informatik-Berichte

[KC99] Knowles, J. D. ; Corne, D. W.: The Pareto archived evolution strategy: a new baseline algorithm for Pareto multiobjective optimization. In: *Proc. 1999 Congress on Evolutionary Computation*. Washington, DC : IEEE, USA, 6.-9. Juli 1999, S. 98–105

[KC00] Knowles, J. D. ; Corne, D. W.: Approximating the nondominated front using the Pareto archived evolution strategy. In: *Evolutionary Computation* 8 (2000), S. 149–172

[KD90] Karian, A. ; Dudewicz, E. J.: *Modern Statistical Systems, and GPSS Simulation*. Computer Science Press, 1990

[KE01] Kennedy, J. ; Eberhart, R. C.: *Swarm Intelligence*. Morgan Kaufmann Academic Press, 2001

[Kec06] Kecher, Ch.: *UML 2.0 - Das umfassende Handbuch*. Galileo Computing, 2006

[KG94] Karian, Z. A. ; Goyal, R.: Random Number Generation and Testing / Birkhauser Verlag AG, Basel, Schweiz. 1994 (1). – Maple Technical Newsletter

[KGV83] Kirkpatrick, S. ; Gelatt, C. D. ; Vecchi, M. P.: Optimization by Simulated Annealing. In: *Science* 220 (1983), 13 May, Nr. 4598, S. 671–680

[Khe88] Kheir, N. A. (Hrsg.): *Systems modeling and computer simulation*. Marcel Dekker, Inc., 1988 (2)

[KK06] Kämpf, M. ; Köchel, P.: Simulation-based sequencing and lot size optimisation for a production-and-inventory system with multiple items. In: *International Journal of Production Economics* 104 (2006), November, Nr. 1, S. 191–200

[KKS+03] Koza, J. R. ; Keane, M. A. ; Streeter, M. J. ; Mydlowec, W. ; Yu, J. ; Lanza: *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003

[KMS03] Ku, K. W. ; Mak, M. W. ; Siu, W. C.: Approaches to Combining Local and Evolutionary Search for Training Neural Networks: A Review and Some New Results. In: Tsutsui, S. (Hrsg.) ; Ghosh, A. (Hrsg.): *Advances in Evolutionary Computing*, Springer Verlag, 2003, S. 615–642

[Knu81] Knuth, D. E.: *The Art of Computer Programming*. Bd. 2: *Seminumerical Algorithms*. 2. Reading, MA : Addison-Wesley, 1981

[Köb99] Köbernik, G.: *Moderne Methoden für die Fertigungssteuerung bei Werkstattfertigung*. 3. Aufl. Lohmar, 1999 (Produktionswirtschaft und Industriebetriebslehre)

[Köc07a] Köchel, P.: Simulation diskreter Prozesse / Technische Universität Chemnitz, Fakultät für Informatik, Professur Modellierung und Simulation. 2007 (Sommersemester 2007). – Vorlesungsskript

[Köc07b] Köchel, P.: Stochastische Modelle und Leistungsbewertung komplexer Systeme / Technische Universität Chemnitz, Fakultät für Informatik, Professur Modellierung und Simulation. 2007 (Sommersemester 2007). – Vorlesungsskript

[Koh72] Kohlas, J.: *Monte-Carlo-Simulation im Operations Research*. Berlin, Heidelberg : Springer Verlag, 1972 (Lecture notes in economics and mathematical systems 63)

[Kön01] König, W.: Management betrieblicher Prozesse / Institut für Wirtschaftsinformatik (IWI) der Universität Frankfurt. Version: 2001. `http://www.is-frankfurt.de/veranstaltung/MBP_WS0102/vorlesung/02_01_21.pdf`. 2001 (Wintersemester 2001/2002). – Vorlesungsskript

[Koz92] Koza, J. R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992

[Koz04] Koziol, Ph.: *Statistische Methoden zur Bestimmung der Simulationslänge*, Universität Karlsruhe (TH), Institut für Fördertechnik und Logistik, Seminararbeit Simulation und Logistik, 2004

[Kru01] Krug, W.: *Modellierung, Simulation und Optimierung für Prozesse der Fertigung, Organisation und Logistik*. Delft : Society for Modeling and Simulation Internaternational, 2001

[KS85] Karmarkar, U. S. ; Schrage, L.: The deterministic dynamic product cycling problem. In: *Operationa Research* 33 (1985), Nr. 2, S. 326–345

[KS94] Kuik, R. ; Salomon, M.: Batching decisions: structure and models. In: *European Journal of Operational Research* Bd. 75. North-Holland : Elsevier, 1994, S. 243–263

[Kuh92] Kuhn, H.: Heuristische Suchverfahren mit simulierter Abkühlung. In: *Wirtschaftswissenschaftliches Studium* Bd. 21. Springer, 1992, S. 387–391

[Küh06] Kühnel, A.: *Visual C# 2005*. Galileo Computing, 2006 `http://www.galileocomputing.de/openbook/visual_csharp/index.htm`

[LA97] L'Ecuyer, P. ; Andres, T. H.: A random number generator based on the combination of four LCGs. In: *Mathematics and Computers in Simulation* 44 (1997), S. 99–107

[Lan07] Lanner Group, Inc.: *Homepage von Lanner Group, Inc.* Version: Mai 2007. `http://www.lanner.com/`, Abruf: 22.05.2007. Internet-Quelle

[LBC93] L'Ecuyer, P. ; Blouin, F. ; Couture, R.: A Search for Good Multiple Recursive Generators. In: *ACM Transactions on Modeling and Computer Simulation* 3 (1993), S. 87–98

[L'E88] L'Ecuyer, P.: Efficient and portable combined random number generators. In: *Comm. ACM* 31 (1988), S. 742–749 und 774

[L'E96a] L'Ecuyer, P.: Combined Multiple Recursive Random Number Generators. In: *Operations Research* 44 (1996), S. 816–822

[L'E96b] L'Ecuyer, P.: Random Number Generators and Empirical Tests. In: Niederreiter, H. (Hrsg.) ; Hellekalek, P. (Hrsg.) ; Larcher, G. (Hrsg.) ; Zinterhof, P. (Hrsg.): *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing* Bd. 127. Springer Verlag, 1996, S. 124–138

[L'E99] L'Ecuyer, P.: Tables of Linear Congruential Generators of Different Sizes and Good Lattice Structure. In: *Mathematics of Computation* 68 (1999), Nr. 225, S. 249–260

[Leh51] Lehmer, D. H.: Mathematical methods in large-scale computing units. In: *Symposium on Large-Scale Digital Calculating Machinery in 1949*. Cambridge, MA : Harvard University Press, 1951, S. 141–146

[Lie92] Liebl, F.: *Simulation: Problemorientierte Einführung*. München : Oldenbourg, 1992

[Lin65] Lin, S.: Computer Solutions to the Traveling Salesman Problem. In: *Bell System Technical Journal* 44 (1965), S. 2245–2269

[Lip92] Lipsky, L. R.: *Queueing theory: a linear algebraic approach*. Macmillan, 1992

[LK73] Lin, S. ; Kernighan, B.: An effective heuristic algorithm for the traveling salesman problem. In: *Operations Research* 21 (1973), S. 498–516

[LK00] Law, A. M. ; Kelton, W. D.: *Simulation Modeling and Analysis*. Third. New York : McGraw-Hill, 2000

[LMW92] Löffler, H. ; Meinhardt, J. ; Werner, D.: *Taschenbuch der Informatik: Hardware - Software - Anwendungen*. Fachbuchverlag Leipzig, 1992

[LS07] L'Ecuyer, P. ; Simard, R.: *TestU01: A C Library for Empirical Testing of Random Number Generators*. unveröffentlicht. `http://www.iro.umontreal.ca/~lecuyer/myftp/papers/testu01.pdf`. Version: Februar 2007. – Veröffentlichung für ACM Transactions on Mathematical Software

[LSCK02] L'Ecuyer, P. ; Simard, R. ; Chen, E. J. ; Kelton, W. D.: An Objected-Oriented Random-Number Package with Many Long Streams and Substreams. In: *Operations Research* 50 (2002), Nr. 6, S. 1073–1075

[Mac89] Maclaren, N. M.: The Generation of Multiple Independent Sequences of Pseudorandom Numbers. In: *Appl. Statist.* 38 (1989), S. 351–359

[Mag01] Magnusson, Th.: *The Capacitated Lot-sizing and Scheduling Problem with Sequence-dependent Setup Costs and Setup Times*, Graduate School of the University of Minnesota, MA Thesis, June 2001

[Man01] Manlig, F.: *Optimierung von Fertigungsprozessen mit Rechnersimulation - Ein Analysebericht*, 2001. `http://mciron.mw.tu-dresden.de/pas/pazat_lit/for_ber/2001/2001-06.pdf`

[Mei99] Meister, G.: *Untersuchung des Parallelisierungspotentials diskreter ereignisgesteuerter Simulationen am Beispiel der Schaltkreissimulation*, Technische Universität Darmstadt, Dissertation, 1999

[Mey02] Meyer, D.: MOMBES: Multiobjective Modelbased Evolution Strategy / Kratzer AG München. 2002 (FKI-246-02). – Forschungsbericht Künstliche Intelligenz

[Mey03] Meyer, D.: *Modellbasierte Mehrzieloptimierung mit Neuronalen Netzen und Evolutionsstrategien*, Technische Universität Ilmenau, Dissertation, 2003

[Mic07a] Microsoft Corporation: *Random-Klasse*. Version: Mai 2007. `http://msdn2.microsoft.com/de-de/library/system.random(VS.80).aspx`, Abruf: 09.05.2007. Internet-Quelle

[Mic07b] Microsoft Corporation: *Rotor projects*. Version: April 2007. `http://research.microsoft.com/programs/europe/rotor/`, Abruf: 19.04.2007. Internet-Quelle

[MM89] Mattern, F. ; Mehl, H.: Diskrete Simulation - Prinzipien und Probleme der Effizienzsteigerung durch Parallelisierung. In: *Informatik-Spektrum* 12 (1989), Nr. 4, S. 198–210

[MN98] Matsumoto, M. ; Nishimura, T.: Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. In: *ACM Trans. on Modeling and Computer Simulations* (1998). `http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html`

[MO07] Müller, M. ; Ostermeyer, G. P.: Beschreibung der Zeitskalenproblematik bei Bremsvorgängen. In: *GAMM Annual Meeting 2006 - Berlin* Bd. 6. Technische Universität Braunschweig, Institut für Dynamik und Schwingungen : WILEY-VCH Verlag, 2007, Kapitel 7, S. 453–454

[Mon07] MonoDevelop: *MonoDevelop*. Version: April 2007. `http://www.monodevelop.com`, Abruf: 19.04.2007. Internet-Quelle

[MPI07] MPI: *The Message Passing Interface (MPI) standard*. Version: 2007. `http://www-unix.mcs.anl.gov/mpi/`, Abruf: 23.03.2007. Internet-Quelle

[MRT53] Metropolis, N. ; Rosenbluth, A. W. und M. N. ; Teller, A. und E.: Equation of State Calculations by Fast Computing Machines. In: *Journal of Chemical Physics* 21 (1953), S. 1087–1092

[MRW95] Markowitz, D. M. ; Reimann, M. I. ; Wein, L. M.: *The Stochastic Economic Lot Scheduling Problem: Heavy traffic analysis of dynamic cyclic policies*. 1995. – unveröffentliches Manuskript

[MRW00] Markowitz, D. M. ; Reimann, M. I. ; Wein, L. M.: The Stochastic Economic Lot Scheduling Problem: Heavy traffic analysis of dynamic cyclic policies. In: *Operations Research* 48 (2000), Januar-Februar, Nr. 1, S. 136–154

[MT03] Mostaghim, S. ; Teich, J.: The role of e-dominance in Multi-objective Particle Swarm Optimization. In: *The Proceedings of the 2003 Congress on Evolutionary Computation*. Canberra, Australia : IEEE, Dezember 2003, S. 1764–1771

[MT05] Mostaghim, S. ; Teich, J.: Quad-trees: A Data structure for storing Pareto-sets in Multi-objective Evolutionary Algorithms with Elitism. In: Abraham, Ajith (Hrsg.) ; Jain, Lakhmi (Hrsg.) ; Goldberg, Robert (Hrsg.): *Evolutionary Multiobjective Optimization – Theoretical Advances and Applications*. London : Springer Verlag, 2005 (Advanced Information and Knowledge Processing), S. 81–104

[MZ96] Masuda, N. ; Zimmermann, F.: PRNGlib: A Parallel Random Number Generator Library / Swiss Center for Scientific Computing. Graduate School of Industrial Administration, Pittsburgh, PA : CiteSeer, 1996. – Technical report

[Nie95] Niederreiter, H.: New developments in uniform pseudorandom number and vector generation. In: Niederreiter, H. (Hrsg.) ; Shiue, P. Jau-Shyong (Hrsg.): *Monte Carlo and Quasi-Monte Carlo Methods in Scientific Computing* Bd. 106. Springer Verlag, 1995, S. 124–138

[Nie96] Nieländer, U.: *Zur optimalen Konfigurierung und Steuerung diskreter Systeme, insbesondere Fertigungssysteme, mittels Evolutionärer Algorithmen und Simulation*, Technische Universität Chemnitz, Diplomarbeit, 1996

[NM65] Nelder, J. A. ; Mead, R. A.: A Simplex Method for Function Minimization. In: *Computer Journal* 7 (1965), S. 308–313

[NM93] Neumann, K. ; Morlock, M.: *Operations Research*. Hanser, 1993

[NW88] Nemhauser, G. ; Wolsey, L.: *Integer and Combinatorial Optimization*. Wiley Interscience, 1988

[Obj07] Object Management Group, Inc.: *UML Resource Page*. Version: Juli 2007. http://www.uml.org/, Abruf: 23.07.2007. Internet-Quelle

[Opt00] OptTek Systems, Inc.: *Combining Simulation & Optimization for Improved Business Decisions*. Version: Oktober 2000. http://www.opttek.com/publications/oqpromo.html. Internet-Quelle

[Opt07] OptTek Systems, Inc.: *Homepage von OptQuest*. Version: Mai 2007. http://www.opttek.com/, Abruf: 22.05.2007. Internet-Quelle

[Or76] Or, I.: *Travelling Salesman Type Combinatorical Problems and Their Relation to the Logistics of Blood Banking*. Northwestern University, Department of Industrial Engineering and Management Science, Ph. D. thesis, 1976

[Ort94] Ortner, H.: *Dynamische Repartitionierung bei verteilter ereignisgesteuerter Logiksimulation mit Time-Warp-Synchronisation*, Technische Universität Berlin, Dissertation, 1994

[Pac07] Pacific Northwest National Laboratory: *TCGMSG Message Passing Library*. Version: Juli 2007. http://www.emsl.pnl.gov/docs /parsoft/tcgmsg/tcgmsg.html, Abruf: 11.07.2007

[Paw90] Pawlikowski, K.: Steady-state simulation of queueing processes: asurey of problems and solutions. In: *ACM Computing Surveys* 22 (1990), S. 123–170

[Pid96] Pidd, M.: *Tools for Thinking*. John Wiley & Sons, 1996

[PK00] Pham, D. T. ; Karaboga, D.: *Intelligent Optimisation Techniques*. Springer Verlag, 2000

[pLa07] pLab: *Random Number Generators*. Version: Februar 2007. http://random.mat.sbg.ac.at/, Abruf: 03.02.2007. Internet-Quelle

[Plo05] Plontke, R.: *Entwurf und Implementierung einer C++ Klassenbibliothek für das Optimierungssystem ISSOP und Untersuchung der Anwendbarkeit hybrider Optimierungsverfahren zur Effizienzverbesserung*, Technische Universität Chemnitz, Diplomarbeit, Dezember 2005

[PM00] Pezzella, F. ; Merelli, E.: A tabu search method guided by shifting bottleneck for the job shop scheduling problem. In: *European Journal of Operational Research* 120 (2000), S. 297–310

[PMK97] Precht, M. ; Meier, N. ; Kleinlein, J.: *EDV-Grundwissen: Eine Einführung in Theorie und Praxis der modernen EDV*. 4. Addison-Wesley-Longman, 1997

[pro07a] PROJECT.COM www.mono: *Main Page - Mono*. Version: April 2007. http://www.mono-project.com/, Abruf: 19.04.2007. Internet-Quelle

[Pro07b] PROMODEL CORPORATION: *Homepage von ProModel*. Version: Mai 2007. http://www.promodel.com/, Abruf: 22.05.2007. Internet-Quelle

[PVM07] PVM: *PVM (Parallel Virtual Machine)*. Version: März 2007. http://www.csm.ornl.gov/pvm/, Abruf: 23.03.2007. Internet-Quelle

[PW91] POCHET, Y. ; WOLSEY, L. A.: Solving Multi-Item Lot-Sizing Problems Using Strong Cutting Planes. In: *Management Science* 37 (1991), Januar, Nr. 1, S. 53–67

[Qua04] QUADT, D.: *Lot-Sizing and Scheduling for Flexible Flow Lines*. Springer Verlag, 2004

[RA07] ROCKWELL AUTOMATION, Inc.: *Rockwell Automation - Arena Simulation Software*. Version: Juli 2007. http://www.arenasimulation.com, Abruf: 13.07.2007. Internet-Quelle

[ran07] RANDOM.ORG: *True Random Number Service*. Version: Februar 2007. http://www.random.org/, Abruf: 03.02.2007. Internet-Quelle

[RCN03] ROCHA, M. ; CORTEZ, P. ; NEVES, J.: Adaptive Learning in Changing Environments. In: *ESANN*, 2003, S. 487–492

[Rec73] RECHENBERG, I.: *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart : frommann-holzbog, 1973

[Rei94] REINELT, G. ; GOOS, G. (Hrsg.) ; HARTMANIS, J. (Hrsg.): *The Traveling Salesman: Computational Solutions for TSP Applications*. Springer Verlag, 1994 (Lecture Note in Computer Science 840)

[RN95] RUSSELL, S. ; NORVIG, P.: *Artificial Intelligence: A Modern Approach*. Prentice Hall, 1995

[Rob05] ROBINSON, S.: Automated analysis of simulation output data. In: KUHL, M. E. (Hrsg.) ; STEIGER, N. M. (Hrsg.) ; ARMSTRONG, F. B. (Hrsg.) ; JOINES, J. A. (Hrsg.): *Proceedings of the 37th conference on Winter simulation*. Orlando, Florida, USA : Winter Simulation Conference, 2005, S. 763–770

[Roh99] ROHRER, M.: AutoMod Product Suite Tutorial (AutoMod, Simulator, AutoStat) by AutoSimulation. In: FARRINGTON, P. A. (Hrsg.) ;

Nembhard, H. B. (Hrsg.) ; Sturrock, D. T. (Hrsg.) ; Evans, G. W. (Hrsg.): *Proceedings of the 1999 Winter Simulation Conference*, Informs, 1999, S. 220–226

[RR02] Resende, M. ; Ribeiro, C. ; Glover (Hrsg.) ; Kochenberger (Hrsg.): *Greedy Randomized Adaptive Search Procedures*. Kluwer Academic Publishers, 2002

[RR05] Reichelt, D. ; Rothlauf, F.: CURE: Eine Reparaturheuristik für die Planung ökonomischer und zuverlässiger Kommunikationsnetzwerke mit Hilfe von heuristischen Optimierungsverfahren. In: *Kommunikation in Verteilten Systemen (KiVS)*, Springer Verlag, 2005 (Informatik aktuell), S. 283–294

[RS07] Rinner, Ch. ; Schwarzer, F.: *KDevelop - an Integrated Development Environment - Homepage*. Version: April 2007. `http://www.kdevelop.org`, Abruf: 19.04.2007. Internet-Quelle

[RW98] Reimann, M. I. ; Wein, L. M.: Dynamic scheduling of a two-class queue with setups. In: *Operations Research* 46 (1998), S. 532–547

[SAH04] Slomka, F. ; Albers, K. ; Hofmann, R.: A Multiobjective Tabu Search Algorithm for the Design Space Exploration of embedded Systems. In: *Tagungsband: Dipes*, Kluwer Academic Publishers, aug 2004, S. 227–236

[Sch81] Schwefel, H.-P.: *Numerical optimization of computer models*. John Wiley & Sons, Ltd., 1981

[Sch83] Schruben, L. W.: Detected initialization bias in simulation output. In: *Operations Research* 31 (1983), S. 1090–1108

[Sch85a] Schaffer, J.: Multiple objective optimization with vector evaluated genetic algorithms. In: *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum, 1985, S. 93–100

[Sch85b] Schmidt, B.: *Fachbericht Simulation*. Bd. 1: *Fachbericht Simulation*. Berlin, Heidelberg, New York, Tokio : Springer Verlag, 1985

[Sch92] Schlitt, H.: *Systemtheorie für stochachstische Prozesse*. Springer Verlag, 1992

[Sch95] Schwefel, H.-P.: *Evolution and Optimum Seeking*. New York : Wiley & Sons, 1995

[Sch02] Schneeweiss, Ch.: *Einführung in die Produktionswirtschaft*. 8. Aufl. Springer Verlag, 2002

[Sch03] Schlittgen, R.: *Einführung in die Statistik*. Oldenbourg Verlag, 2003

[Sch04] Schmitt, L. M.: Theory of Genetic Algorithms II: models for genetic operators over the string-tensor representation of populations and convergence to global optima for arbitrary fitness function under scaling. In: *Theoretical Computer Science* 310 (2004), S. 181–231

[Sch06a] Schricker, G.: *Urheberrecht*. Verlag C. H. Beck, 2006

[Sch06b] Schüller, S.: *Software zur Simulation und Optimierung von Fertigungslinien mit Anwendung auf ein Metallunternehmen*. Fakultät für Informatik, Professur Modellierung und Simulation, Technische Universität Chemnitz, Diplomarbeit, 2006

[Sch06c] Schwichtenberg, H.: *Microsoft .NET 2.0 - Crashkurs*. Bonn : Microsoft Press, 2006

[Sch07a] Schmidt, B.: *Workshop Optimierung und Simulation*. Version: Mai 2007. `http://www.sim-serv.com/training_doc/training_3_11.pdf`, Abruf: 15.05.2007. Internet-Quelle

[Sch07b] Schwichtenberg, H.: *dotnetframework.de - Die unabhängige deutsche Info-Site zum Microsoft .NET Framework*. Version: April 2007. `http://www.dotnetframework.de`, Abruf: 19.04.2007. Internet-Quelle

[Sch07c] Schwichtenberg, H.: *Microsoft .NET 3.0 - Crashkurs*. 1. Microsoft Press Deutschland, 2007

[Sch07d] Schwichtenberg, H.: *Programmiersprachen für das .NET Framework*. Version: April 2007. `http://www.dotnetframework.de/dotnet/produkte/sprachen.aspx`, Abruf: 19.04.2007. Internet-Quelle

[SD94] Srinivas, N. ; Deb, K.: Multiobjective optimization using nondominated sorting in genetic algorithms. In: *Evolutionary Computation* 2 (1994), S. 221–248

[SE71] Sisson, J. R. ; Emshoff, R. L.: *Design and Use of Computer Simulation Models*. The Macmillan Company, 1971

[SEF+04] Smith, K. I. ; Everson, R. M. ; Fieldsend, J. E. ; Misra, R. ; Murphy, C.: Multi-objective simulated annealing for optimisation of CDMA network air-interface. In: *Motorola Software, Systems, and Simulation Symposium*, 2004

[Sei06] Seifert, J.: *Qube*. Version: Februar 2006. `http://jan.seifseit.de/programme/qube/qube.php`, Abruf: 03.02.2007. Internet-Quelle

[SFW94] Scheidt, J. vom ; Fellenberg, B. ; Wöhrl, U.: *Analyse und Simulation stochastischer Schwingungssysteme*. Teubner, 1994 (Leitfäden der angewandten Mathematik und Mechanik 71)

[SH95] Spaniol, O. ; Hoff, S. ; Mahr, B. (Hrsg.) ; Schill, A. (Hrsg.) ; Vossen, G. (Hrsg.): *Ereignisorientierte Simulation*. International Thomson Publishing Company, 1995 (Thomson's Aktuelle Tutorien)

[Sha75] Shannon, R. E.: *Systems simulation: the art and science*. Prentice Hall, 1975

[SHF94] Schöneburg, E. ; Heinzmann, F. ; Feddersen, S.: *Genetische Algorithmen und Evolutionsstrategien: Eine Einführung in Theorie und Praxis der simulierten Evolution*. Addison-Wesley, 1994

[Shv06] Shvartsman, A. A. v. (Hrsg.) ; OPODIS (Veranst.): *Principles of Distributed Systems*. Bordeaux, France : Springer Verlag, Dezember 2006 (Lecture Notes in Computer Science)

[SIM07] SIMUL8 Corporation: *Homepage von Simul8 Corporation*. Version: Mai 2007. `http://www.simul8.com`, Abruf: 22.05.2007. Internet-Quelle

[SM97] Sox, C. R. ; Muckstadt, J. A.: Optimization-based planning for the stochastic lot scheduling problem. In: *IIE Trans.* 29 (1997), S. 349–358

[Smi80] Smith, S. F.: *A Learning System Based on Genetic Adaptive Algorithms*, University of Pittsburgh, PhD dissertation, 1980

[Smi07] Smith, K.: *A Study of Simulated Annealing Techniques for Multi-Objective Optimisation*, University of Exeter, Dissertation, 2007

[Sol07] Solution, DUALIS GmbH I.: *ISSOP - Der intelligente Optimierer*. Version: Mai 2007. `http://www.dualis-it.de/index.php?pageid=358`, Abruf: 22.05.2007. Internet-Quelle

[Spa03] Spall, J. C.: *Introduction to Stochastic Search and Optimization*. Wiley & Sons, 2003

[Spr99] Sprave, J.: *Ein einheitliches Modell für Populationsstrukturen in Evolutionären Algorithmen*, Universität Dortmund, Dissertation, Juni 1999

[SS03] Suerie, Ch. ; Stadtler, H.: The Capacitated Lot-Sizing Problem with Linked Lot Sizes. In: *Management Science* 49 (2003), August, Nr. 8, S. 1039–1054

[SST83] Schruben, L. W. ; Singh, H. ; Tierney, L.: Optimal tests for initialization bias in simulation output. In: *Operations Research* 31 (1983), Nr. 6, S. 1167–1178

[Stö03] Stöcker, H.: *Taschenbuch mathematischer Formeln und moderner Verfahren*. 4. Aufl. Deutsch Harri GmbH, 2003

[SWd04] Smits, S. A. ; Wagner, M. ; de Kok, T. G.: Determination of an order-up-to policy in the stochastic economic lot scheduling model. In: *International Journal of Production Economics* 90 (2004), S. 377–389

[Syr97] Syrjakow, M.: *Verfahren zur effizienten Parameteroptimierung von Simulationsmodellen*. Institut für Rechnerentwurf und Fehlertoleranz, Universität Karlsruhe, Dissertation, 1997

[Tak86] Takagi, H.: *Analysis of Polling Systems*. MIT Press, 1986 (Series in Computer Systems)

[Tei98] Teich, T.: *Optimierung von Maschinenbelegungsplänen unter Benutzung heuristischer Verfahren*, Technische Universität Chemnitz, Dissertation, 1998

[Tem06] Tempelmeier, H.: *Bestandsmanagement in Supply Chains*. Books on Demand, 2006

[Toc63] Tocher, K. D.: *The Art of Simulation*. D. Van Nostrand Co., Inc., 1963

[Ver01] Verein Deutscher Ingenieure: VDI-Richtlinie 3633 Blatt 1: Simulation von Logistik-, Materialfluss- und Produktionssystemen – Grundlagen / VDI-Gesellschaft Fördertechnik Materialfluss Logistik. 2001 (3633). – Technische Regel - Entwurf

[Voß06] Voss, H.: *LATEX in Naturwissenschaften & Mathematik*. Poing : Franzis, 2006 (Franzis professional series)

[Vou97] Voudouris, C.: *Guided Local Search for Combinatorial Optimisation Problems*, University of Essex, Department of Computer Science, Diss., 1997

[Vry04] Vry, W.: *Beschaffung und Lagerhaltung: Materialwirtschaft für Handel und Industrie*. Ludwigshafen (Rhein) : Kiehl, 2004 (Lehrbücher für Fachwirte und Fachkaufleute)

[Wac04] Wachsmuth, M.: *Simulated Annealing und Tabu Search: Ein empirischer Vergleich*. Fakultät für Informatik, Professur Modellierung und Simulation, Technische Universität Chemnitz, Studienarbeit, Oktober 2004

[Wac05] Wachsmuth, M.: *Implementierung eines hybriden Systems der simulationsbasierten Optimierung und empirische Untersuchungen zur Wahl effizienter Verfahrensparameter*. Fakultät für Informatik, Professur Modellierung und Simulation, Technische Universität Chemnitz, Diplomarbeit, September 2005

[Wal04] Walser, H.: *Der Goldene Schnitt*. Edition am Gutenbergplatz, 2004

[Web00] Weber, M.: Verteilte Systeme I / Universität Ulm. Version: 2000. http://medien.informatik.uni-ulm.de/lehre/courses/ss00/vs1/Kap4.pdf. 2000 (Sommersemester 2000). – Vorlesungsskript

[Weg95] Wegenkittl, S.: *Empirical testing of pseudorandom number generators*. Universität von Salzburg, Österreich, Universität von Salzburg, Ph. D. thesis, 1995. – in Italian

[Wei02] Weiss, M. ; Bankhamer, A. (Hrsg.): *TCP-IP-Handbuch: Internet- und Transportprotokolle, Routing-Protokolle, Virtual Private Networks, Voice over IP, IP-Sicherheit*. Poing: Franzis, 2002 (Professional series Network computing)

[Wel83] Welch, P. D.: The Statistical Analysis of Simulation Results. In: Lavenberg, S. (Hrsg.): *The Computer Performance Modeling Handbook*. New York : Academic Press, 1983, S. 268–328

[Wer07] Wermke, M.: *Duden die deutsche Rechtschreibung : das umfassende Standardwerk auf der Grundlage der neuen amtlichen Regeln*. 24. Auflage. Mannheim : Dudenverlag, 2007

[Wes00] Westermann, T.: *Mathematik für Ingenieure mit Maple (Band 1)*. Springer Verlag, 2000

[WG06] Wang, S. ; Guignard, M.: Hybridizing discrete- and continuous-time models for batch sizing and scheduling problems. In: *Computers & Operations Research* 33 (2006), S. 971–993

[WH82] Wichmann, B. A. ; Hill, I. D.: An efficient and portable pseudo-random number generator. In: *Appl. Statist.* 31 (1982), S. 188–190

[Wil34] Wilson, R. H.: A Scientific Routine for Stock Control. In: *Harvard Business Review* 13 (1934), S. 116–128

[WLR06] Willcock, J. ; Lumsdaine, A. ; Robison, A.: Using MPI with C# and the Common Language Infrastructure / Indiana University, Computer Science Department. 2006 (570). – Technical Report

[WM95] Wolpert, D. H. ; Macready, W. G.: No Free Lunch Theorems for Search / Santa Fe Institute. 1995 (SFI-TR-05-010). – Technical Report

[WM97] Wolpert, D. H. ; Macready, W. G.: No Free Lunch Theorems for Optimization. In: *IEEE Transactions on Evolutionary Computation* 1 (1997), April, Nr. 1, S. 67–82

[WS04] Wagner, M. ; Smits, S. R.: A local search algorithm for the optimisation of the stochastic economic lot scheduling problem. In: *International Journal of Production Economics* 90 (2004), Nr. 3, S. 391–402

[WW58] Wagner, H. M. ; Whitin, T. M.: Dynamic Version of the Economic Lot Size Model. In: *Management Science* 5 (1958), Nr. 1, S. 89–96

[YM93] Yuret, D. ; Maza, M.: Dynamic Hillclimbing: Overcoming the Limitations of Optimization Techniques. In: *The Second Turkish Symposium on Artificial Intelligence and Neural Networks*, 1993, S. 208–212

[Zip91] Zipkin, P.: Computing optimal lot sizes in the economic lot scheduling problem. In: *Operations Research* 39 (1991), S. 56–63

[ZLT01] Zitzler, E. ; Laumanns, M. ; Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm / Swiss Federal Institute of Technology Lausanne, Schweiz. 2001 (103). – Technical Report TIK-Report

[ZPK00] Zeigler, B. P. ; Praehofer, H. ; Kim, T. G.: *Theory of Modeling and Simulation: Integrating Discret Event and Continuous Complex Dynamic Systems*. 2. Aufl. Academic Press, 2000

[ZT99] Zitzler, E. ; Thiele, L.: Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. In: *IEEE Trans. Evol. Comput.* 3 (1999), S. 257–271

Stichwortverzeichnis

A

Abbildungsfunktion 42
Abbruchkriterium 10, 14, 97, 152
Absolutbetrag 25
Abstandsnorm 32
AHP-Verfahren . . . . . *siehe* Nutzwertanalyse
Ameisenalgorithmen 114
Analyse 11

B

Baumansicht 44
Beobachtung 62
Berechnung 11
Bestimmtes Optimum 26
Bewerter . . . . . *siehe* Bewertung
Bewertung 11, 44, 46
Bibliotheken
  `CAOSStartUp` 139
  `CalculationAssessment-Optimisation (CAO)` 144
  `Network` 142
  `Utilities` 139
Boolsche Optimierungsprobleme 29

C

CalculationAssessmentOptimisation-System (`CAOS`) . . . . *siehe* Softwaresystem `CAOS`
Capacitated Lot Scheduling Problem (CLSP) 176
Capacitated Lot Scheduling Problem Linked lot-sizes (CLSPL) 176
Capacitated Lot Scheduling Problem with Setup Carryover (CLSP-SC) 176
Capacitated Stochastic Lot Scheduling Problem (CSLSP) 176
Capacitated Stochastic Lot Sizing Problem . 57
Continuous Setup Lot-sizing Problem (CSLP) 176

D

Deterministische Eröffnungsverfahren 98
Deterministische Nebenbedingung 22
Deterministische Verbesserungsverfahren . . . 99
Discrete Lot Scheduling Problem (DLSP) . 176
Diskrete Simulation 58
  Ereignisorientiert 59
  Ereignisroutine 59
  Exekutive 61
  Simulationszugänge 59
  Zeitfortschreibung 59
  Zeitgesteuert 59
Diskretisierung 59
Downhill-Simplex-Verfahren 93
Dynamische Reihenfolgestrategien 189
  Dynamic 190
  LWQ 189
  LWVQ 189
  Random 190
  SWQ 189
  SWVQ 190

E

Echtzeitanforderung 2
Economic Lot Scheduling Problem (ELSP) . . . . . 175 f.
Economic Order Quantity (EOQ) 173, 175
Eingabe-Verarbeitung-Ausgabe-Prinzip (EVA-Prinzip) 10
Einkriterielles Optimierungsproblem 17
  Definition 18
Einplanungsproblem von Fertigungsaufträgen 29
Einschwingphase . . . . . *siehe* Transiente Phase
Entscheidungsfindung 12, 50

Einkriteriell 51
Mehrkriteriell 50
Entscheidungsproblem 25
Entscheidungsvariable 10, 14, 21
Permutation 29
Rechnerinterne Abbildung 46
Reellwertig 27
Epsilon-Abbruch-Verfahren 80
Mehrkriteriell 83
Prüfintervall 81
Prüfwert 81
Prüfzeitpunkt 81
Varianz 83
Voruntersuchungen 83
Epsilon-Umgebung 32
Ergebnisse
Ein-Produkt-Modell 218
Fünf-Produkt-Modell 222, 227
Zwei-Produkt-Modell 221
Erwartungswertfunktion 64
Evolutionäre Algorithmen 109
Exakte Lösungsverfahren 37, 93
Diskretisierung des Wertebereiches . . *siehe* Wertebereich, Diskretisierung
Globale nichtlineare Optimierung . . . *siehe* Globale nichtlineare Optimierung
Gradientenbasierte Verfahren . . . . . *siehe* Gradientenbasierte Verfahren
Lokale nichtlineare Optimierung . . . . *siehe* Lokale nichtlineare Optimierung

F

Fertigungslos 186
Folge von Zufallszahlen
Definition 85

G

Güte einer Lösung
Definition 51
Ganzzahliges Optimierungsproblem 28
Genetische Algorithmen 158
Gewichtung 16
Multiplikative - 16
Gleichheitsbedingung 19
Globale nichtlineare Optimierung 94
Globale Pareto-optimale Menge
Definition 24
Globales Optimum
Definition 26
Gradientenbasierte Verfahren 93
Graphische Ablesemethode von Welch 68
Gütekriterium 37

H

Händische Optimierung 91
Heuristische Suchverfahren 31, 37, 95
Abbruchkriterium *siehe* Abbruchkriterium
Ameisenalgorithmen . . . . . *siehe* Ameisenalgorithmen
Deterministische Eröffnungsverfahren *siehe* Deterministische Eröffnungsverfahren
Deterministische Verbesserungsverfahren . . . . . *siehe* Deterministische Verbesserungsverfahren
Evolutionäre Algorithmen . . . . . *siehe* Evolutionäre Algorithmen
Genetische Algorithmen . *siehe* Genetische Algorithmen
Partikel-Schwarm-Optimierung . . . . . *siehe* Partikel-Schwarm-Optimierung
Simulierte Abkühlung . . . . . *siehe* Simulierte Abkühlung, *siehe* Simulierte Abkühlung
Stochastische Eröffnungsverfahren . . . *siehe* Stochastische Eröffnungsverfahren
Stochastische Verbesserungsverfahren . . . . . *siehe* Stochastische Verbesserungsverfahren, *siehe* Stochastische Verbesserungsverfahren, *siehe* Stochastische Verbesserungsverfahren
Tabusuche . . . . . *siehe* Tabusuche, *siehe* Tabusuche
Hybride Optimierung 119, 159

I

Indexmenge 62
Intervallhalbierungsverfahren 93
Inventarmenge 185
inventory position . . . . . *siehe* Inventarmenge

K

Kanban-Steuerung 207
Kombinatorische Optimierungsprobleme 28
Kombinatorisches Optimierungsproblem 32
Kombinierte Losgrößen- und Reihenfolgeoptimierung 185
Kommunikationsnetzwerk 77
Komplexitätstheorie 34, 58
Konfidenzintervall 58
Konvertierungsfunktion 28
Konvexe Funktion
  Definition 19
Konvexität 19
Kostenbestandteile 172, 174
Kovarianzfunktion 64

L

Lösung 12, 15
  Qualität . . . . . *siehe* Lösungsqualität
Lösungsqualität 51
Lieferkette 173, 207
Lokale nichtlineare Optimierung 93
Lokale Pareto-optimale Menge
  Definition 24
Lokales Optimum
  Definition 26
Losgröße 173
Losgrößen- und Reihenfolgeoptimierung . *siehe* Losgrößen- und Reihenfolgeprobleme
Losgrößen- und Reihenfolgeprobleme 175
  Lösungsansätze 177
Losgrößenoptimierung 181
Losgrößenprobleme 173

M

Maschinenbelegung 45
Mathematische Darstellung 12
Mathematische Modellierung 11, 13
Mathematisches Modell 13, 39
Maximierungsalgorithmus 26
Maximierungsproblem 23, 25
Mehrkriterielle Evolutionäre Algorithmen . 117
Mehrkriterielle Optimierung
  Mehrkriterielle Evolutionäre Algorithmen . . . . . *siehe* Mehrkriterielle Evolutionäre Algorithmen
  Verfahren 115
Mehrkriterielles Optimierungsproblem 18
  Definition 18
Minimierungsalgorithmus 26
Minimierungsproblem 23, 25
Modell 12
  Logische Gliederung 44
Modellbeschreibungssprache EcoSyL . 202, 251
Modellbildung 13
Modelldefinition
  Konstruktionsprozessorientierte - 14
  Rechnerinterne - . . . . *siehe* Rechnerinterne Modelldefinition
Modelldeklaration
  Rechnerinterne - . . . . *siehe* Rechnerinterne Modelldeklaration
Modellelement 13
Modellersteller . . . . . *siehe* Modellierer
Modellierer 41
Modellparameter 13
Modellvariable 14
Modellzerlegung
  Sichtweisen 1
Multi-Level Lot-sizing Problem (MLLP) . . 176

N

Nachbarschaft 31
  Definition 31
Nachbarschaftsfunktion 31
Namensräume
  `CAO.Assessment` 167
  `CAO.Calculation` 150
  `CAO.Model` 144
  `CAO.Optimisation` 150
Nebenbedingung 22
  Deterministisch . . . . *siehe* Deterministische Nebenbedingung
  Post- . . . . . *siehe* Post-Nebenbedingung
  Pre- . . . . . *siehe* Pre-Nebenbedingung
  Stochastisch . . . . . *siehe* Stochastische Nebenbedingung
Nebenbedingungsfunktion 18
Normalisierungskonstante 46

$\mathcal{NP}$-schweres Problem . . 34, 37, 57 f., 175, 177, 182, 237
$\mathcal{NP}$-vollständiges Problem 175
Nutzwertanalyse 48

O

Objekt-orientierte Programmierung (OOP) 133
Instanziierung 41
Objekt-orientiertes Programmierparadigma . 46
Optimale Lösung 23
Optimierung 12
Optimierungsalgorithmus 26
Optimierungsdauer
Verkürzung . . . . . *siehe* Verkürzung der Optimierungsdauer
Optimierungskriterien 174
Optimierungsmanager 150
Optimierungsproblem 13, 39
Einkriteriell . . . . . *siehe* Einkriterielles Optimierungsproblem
Ganzzahlig . . . . . *siehe* Ganzzahliges Optimierungsproblem
Klassifizierung 18
Kombinatorisch . . *siehe* Kombinatorisches Optimierungsproblem
Mehrkriteriell . . . . . *siehe* Mehrkriterielles Optimierungsproblem
Stetig *siehe* Stetiges Optimierungsproblem
Optimierungsprobleme
Boolsch . . . . . *siehe* Boolsche Optimierungsprobleme
Kombinatorisch . . . *siehe* Kombinatorische Optimierungsprobleme
Optimierungsprozess
Echtzeitanforderung . . . . . *siehe* Echtzeitanforderung
Optimierungssoftware 133
Optimierungsverfahren
Einkriteriell 46
Mehrkriteriell 46
Optimierungsverlauf
Darstellungsmöglichkeit 34
Darstellungsmöglichkeiten 27
Optimum . . . . . *siehe* Optimale Lösung
Bestimmtes - . *siehe* Bestimmtes Optimum

P

Parametrisierung 55
Pareto-Front 23
Pareto-optimale Front . . . . . *siehe* Pareto-Front
Pareto-optimale Menge 23
Definition 23
Reduzierung der Lösungen . . . . . *siehe* Entscheidungsfindung
Pareto-Optimalität
Definition 23
Partikel-Schwarm-Optimierung 115
Permutation
Definition 30
Permutationsvektor 32
Planungszeitraum . . . . . *siehe* Zeithorizont
Plot 63
Post-Nebenbedingung 20
Pre-Nebenbedingung 20
Problem des Handlungsreisenden 29
Probleminstanz 36
Problemparameterraum 19
Produktions- und Lagerhaltungsmodell 178
Ein-Produkt-Modell 217
Entscheidungsvariablen 181
Fünf-Produkt-Modell . . . . . 222, 226 f.
Kostenbestandteile 178
Kostenfunktion 191
Kundenbedarfsprozess 179
Lager 180
Losgrößen- und Freigabezeitpunktentscheidung (LF-Entscheidung) 185
Markowitz-Reimann-Wein-Modell 229
Modellparameter . . . . . 178 f.
Modelltypen 195
Modelltyp 1 202
Modelltyp 2 204
Modelltyp 3 206
Modellvariable 178
Nebenbedingungen 178
Optimierungskriterien 191
Optimierungsproblem 178
Produktionsprozess 180
Reihenfolgeentscheidung (R-Entscheidung) 186
Restriktionen 191

Simulationsmodell 178, 196
Statische Reihenfolgestrategien 186
Cyclic 186
FCFS 186
LCFS 186
Voruntersuchungen 212
Zielfunktionen 178, 181
Zuordnungsentscheidung 204
Zwei-Produkt-Modell 221
Produktions- und Lagerhaltungssystem ... 2, 6, 172
Ein-Produkt-Systeme 177
Kapazitätsrestriktionen 177
Mehr-Produkt-Systeme 177
Modelle 173
Produktionsreihenfolge ..... *siehe* Reihenfolge
Produktionssystem 172
Programmbibliothek `PVM` 123
Programmierschnittstelle `MPI` 124
Programmiersprache `C#` 136
Vor- und Nachteile 137
Pseudozufallszahlen 84
Pseudozufallszahlengenerator ..... *siehe* Zufallszahlengenerator, *siehe* Zufallszahlengenerator
Punktekette 33

R

Raum der Entscheidungsvariablen
Diskretisierung 27
Realsystem 12
Erklärung 12
Komplexität 1
Modell 131
Rechnerinterne Modellbeschreibung ... 132
Validierung 12
Vorhersage 12
Rechnergestützte Simulation 14
Rechnerinterne Abbildung 39
Rechnerinterne Darstellung 12
Rechnerinterne Modelldefinition 40
Rechnerinterne Modelldeklaration 40
Rechnerinterne Modellierung 11
Schritte 42
Rechnerinternes Modell 39
Attribute 40
Elemente 40
Kategorien 39
Sichtweisen 44
Unterkategorien 40
Rechnersimulation 13
Reihenfolge 29, 173
Reihenfolgeoptimierung 184
Reihenfolgeprobleme 173
Rektifikation 17

S

Scheduling 45
Scheduling-Probleme
Klassifizierung 204
Schrittweite 28
Schwache Stationärität
Definition 63
Sekantenverfahren 93
Simulation 11
Datenparallele - 79
Ergebnisaufbereitung 76
Gründe 56
Grenzen 58
Nicht-terminierende - 64
Reproduzierbarkeit 84
Simulationstypen 58
Triviale parallele - 79
Verteilte und parallele - 76
Vorteile 57
Simulationsbasierte Optimierung 10
Definition 34
Epsilon-Abbruch-Verfahren ..... *siehe* Epsilon-Abbruch-Verfahren
Iterative Simulation und Optimierung . 35
Lösungsverfahren 34
Modellgestützte - 10
Sequentielle Simulation und Optimierung 35
Verkürzung der Simulationsdauer 81
Simulationsdauer
Verkürzung 56, 76
Simulationsexperiment 15, 37, 55
Simulationslauf 55
Simulationsmodell ..... 13 f., 39
Ankunftsrate 71
Ausgabewerte 15

Auslastungsgrad 70
Bedienrate 71
Berechnendes - 15
Beschreibendes - 14
Gewichtete Ausgabewerte 15
Lastenausgleich 79
Monitoring 79
Nicht zu bewertende Ausgabevariablen 16
Rektifikation . . . . . *siehe* Rektifikation
Simulationsteil 77
Startzustand 65
Statistische Ausgabewerte 15
Synchronisation 77
Asynchrone Verfahren 77
Simulationszeit 78
Synchrone Verfahren 77
Verfahrensarten 77
Synchronisationszeitpunkt 77
Toleranzen 17
Validierung 16
Zerlegung 76
Zu bewertende Ausgabevariablen 16
Simulationsmodellattribut 14
Simulationsmodellparameter 14
Simulationssoftware 4, 134
Simulationssprachen 134
Modellierparadigma 135
Simulator 39
Weiterverwendung 45
Simulierte Abkühlung 104, 156
Skalarer Zielfunktionswert 46
Softwareersteller 41
Softwaresystem `CAOS` 131
Anwendungsfälle 168
Berechnung . . . . . *siehe* Namensräume, `CAO.Calculation`
Bewertung . . . . . *siehe* Namensräume, `CAO.Assessment`
Bibliotheken 137
Hauptziel 136
Modellierung . . . . . *siehe* Namensräume, `CAO.Model`
Optimierung . . . . . *siehe* Namensräume, `CAO.Optimisation`
Optimierungsmanager . . . . . *siehe* Optimierungsmanager
Optimierungsverfahren 154
Plattformunabhängigkeit 136
Starke Stationärität
Definition 63
Stationäre Phase 74
Abtastlänge 75
Batch-Means-Verfahren 75
Bestimmung der Simulationslänge 74
Dynamische Methodik 75
Reduzierung der Varianz 74
Semi-dynamische Methode 75
Statische Methodik 75
Stationäre Zustandsverteilung 65
Stationärität 63
steady-state-Simulation 64
Sterndiagramm 27
Stetiges Optimierungsproblem 28
Stochastic Economic Lot Scheduling Problem (SELSP) . . . . . 175 f.
Stochastische Eröffnungsverfahren 99
Stochastische Nebenbedingung 22
Stochastische Verbesserungsverfahren 101, 103, 109
Stochastischer Prozess
Definition 62
System 12
Systemanalyse 39
Systemelement 13
Systemparameter 13
Systemstruktur 55
Systemverhalten 13, 16, 55

T

Tabusuche 107, 157
Tauschpunkt 33
Transformationsvorschrift 46
Transiente Phase 65
Dauer 66
Dynamische Methoden 66
Einfluss des Startzustandes 70
Methode des Gruppen-Mittelwertes 69
Methode des konstanten Mittelwertes . . 69
Prüfintervall 69
Statische Methoden 66
Welchs graphische Ablesemethode . . . *siehe* Graphische Ablesemethode von Welch

Traveling Salesman Problem (TSP) ..... *siehe* Problem des Handlungsreisenden

U

Ungleichheitsbedingung 19
Untersystem 13

V

Varianzfunktion 64
Verbale Modellbeschreibung 39
Verkürzung der Optimierungsdauer 118
Verteilte und parallele Optimierung .. 122, 162
    Gewählter Ansatz 127
    Kommunikationsablauf 127
Virtuelle Hierarchie 44

W

Wagner-Whitin-Problem (WW) 176
Wahrscheinlichkeitsdichte 87
Wahrscheinlichkeitsverteilung 84, 87, 140
    Dichtefunktion ..... *siehe* Wahrscheinlichkeitsdichte
    Mittelwert 88
    Streuung 88
Wertebereich
    Diskretisierung 94

Z

Zeithorizont ..... 64, 173 f.
Zeitreihe 62
    Startzustand 63
    Visualisierung 63
Zerklüftung 37
Zielfunktion 18
    Nichtlinear 36
Zielfunktionsgradient
    Schätzung 36
Zielfunktionsraum 19, 23
Zielfunktionswert
    Skalar ... *siehe* Skalarer Zielfunktionswert
Zufallsfaktor 62
Zufallsgröße 62
Zufallszahlenerzeugung ..... *siehe* Zufallszahlengenerator
Zufallszahlengenerator 15, 84, 140
    Arithmetisch 85
    Determiniertheit 85
    Gleichverteilung 85
    Hyperebene 86
    Periodenlänge 86
    Rekursiv arithmetisch 85
    Startwerte 85
Zulässiger Bereich eines Optimierungsproblems 22
    Definition 20
Zwei-Punkt-Tausch-Nachbarschaft 32
    Definition 33
Zwei-Punkt-Tausch-Operator 32
    Definition 32