Fachgebiet für
IT in Produktion und Logistik
Prof. Dr.-Ing. M. Rabe

| Fakultät Maschinenbau | Technische Universität Dortmund

MASTERARBEIT

Konzeptentwicklung für die Kopplung heuristischer Optimierung und ereignisdiskreter Simulation für Ablaufplanungsprobleme

bearbeitet von: Ilmo Eckhardt

Studiengang: Wirtschaftsingenieurwesen
Matrikel-Nr.: 124156

Ausgegeben am: 07.01.2015
Eingereicht am: 10.07.2015

Prüfer: Prof. Dr.-Ing. Markus Rabe

Betreuer: Dipl.-Geoinf. Maik Deininger

Inhaltsverzeichnis

Abkürzungsverzeichnis

ACO	Ant Colony Optimization
AS	Arbeitsschritt
CI-CPU	Computer-Independent Central Processing Unit
EDD	Earliest Due-Date
E-Zug	Erlaubter Zug
FIFO	First In - First Out
GT	Giffler & Thompson
LPT	Longest Processing Time
SPT	Shortest Processing Time
VNP-Zug	Verbotener nicht-profitabler Zug
VP-Zug	Verbotener profitabler Zug
ZE	Zeiteinheit

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmusverzeichnis

Formelverzeichnis

Symbolverzeichnis

A	Menge der verbotenen profitablen Züge
A_k	Koeffizientenmatrix
$\mathbb{B}$	Menge der Binärzahlen
B_r	Block r auf dem kritischen Pfad
b	Grenzvektor mit q Komponenten $(b_1, \dots, b_q)$
C	Menge der gerichteten konjunktiven Kanten
c	Variablenvektor mit r Komponenten $(c_1, \dots, c_r)$
$\hat{c}$	Aktuell beste obere Schranke
$\check{c}$	Untere Schranke der Lösung $\check{x}$
C_i	Zeitpunkt der Fertigstellung von J_i
C_{max}	Gesamtzykluszeit
D	Menge der ungerichteten disjunktiven Kanten
D_i	Durchlaufzeit von J_i
D_{max}	Maximale Durchlaufzeit
d	Anzahl der durchgeführten Simulationsdurchläufe
d_i	Liefertermin von J_i
E	Ereignistyp
F	Flow Shop
G	Disjunktiver Graph
g	Mindestanzahl durchzuführender Simulationsdurchläufe
I_j	Leerzeit von M_j
i	Iterationsschritt
i_{max}	Maximale Anzahl an Iterationen
J	Job Shop
J_i	Job i
K	Begrenzende Variable
K_a	Menge aller Lösungskomponenten
k_{n_a}	Lösungskomponente von x_{n_a}

L	Menge aller ungelösten Probleme
L_i	Terminabweichung von J_i
L_{max}	Maximale Terminabweichung
l	Länge der Tabu-Liste
M	Mittelwert aller bis zum aktuellen Zeitpunkt berechneter Zielgrößen
M_j	Maschine j
m	Anzahl der Maschinen
m_i	Anzahl der Arbeitsschritte von J_i
m_u	Maximale Anzahl der Arbeitsschritte
N	Anzahl aller Arbeitsschritte
$N(x)$	Nachbarschaft der Lösung x (Menge aller von x aus erreichbaren Lösungen)
n	Anzahl der Jobs
n_a	Anzahl der Ameisen
O	Open Shop
O_{i1}	1. Bearbeitungsschritt von J_i
O_{ij}	Arbeitsschritt von J_i auf M_j
O_{im_i}	Letzter Bearbeitungsschritt von J_i
P	Parallele Maschinen
P_{i+1}	Folgepopulation
P_y	Anzahl an Individuen
P_0	Initialpopulation
p	Bearbeitungszeit
p_i	Bearbeitungszeit von J_i
p_{ij}	Bearbeitungszeit von J_i auf M_j
p_l	Minimale Bearbeitungszeit
p_s	Wahrscheinlichkeit eine schlechtere Lösung zu akzeptieren
p_u	Maximale Bearbeitungszeit
Q	Uniforme parallele Maschinen

Q_k	Unterproblem
Q_{relax}	Relaxiertes Problem Q_k
R	Heterogene parallele Maschinen
R_a	Ausgangsproblem
R_h	Ressource h
$\mathbb{R}^+$	Menge der positiven reellen Zahlen
r_{hi}	Beanspruchter Teil von R_h durch J_i
r_i	Auftragsfreigabetermin von J_i
S	Menge konjunktiver Kanten (Selektion)
S_i	Startzeitpunkt von Operation i
S_j	Startzeitpunkt von Operation j
S_*	Startzeitpunkt der Senke
s	Anzahl der Ressourcen
s_{IM}	Anzahl an Übereinstimmungen der Input-Matrix mit der aktuellen Stellgröße
$\widehat{s_{IM}}$	Anzahl erlaubter Übereinstimmungen der Input-Matrix mit der aktuellen Stellgröße
T	Tabu-Liste
T'	Aktualisierte Tabu-Liste
T_i	Verspätung von J_i
t	Toleranzwert des Input-Matrix-Updates
t_i	Temperatur im Iterationsschritt i
U_i	Terminüberschreitung von J_i
V	Menge aller Knoten des disjunktiven Graphen
$V(x)$	Menge der von x möglichen Züge v
v	Zug zu einer Lösung
v_{ij}	Bearbeitungsgeschwindigkeit von M_j für J_i
v_j	Bearbeitungsgeschwindigkeit von M_j
v'	Zug zur Lösung x'

v'^{-1}	Umkehrung des Zuges v'
W	Wertebereich der Variablen $(c_1, \dots, c_r)$
W_i	Gesamte Wartezeit von J_i
W_{ij}	Wartezeit von J_i vor M_j
w_i	Gewichtung von J_i
X	Lösungsraum
x	Zulässige Lösung
$\hat{x}$	Aktuell beste Lösung
$\check{x}$	Optimale Lösung für Q_{relax}
x'	Nachbarschaftslösung
x'_i	Ausgewählte Nachbarschaftslösung im Iterationsschritt i
x_{n_a}	Lösung der Ameise n_a
y_{ij}	Binärvariable
y_j	Individuum j
y_1	1. selektiertes Individuum
y_2	2. selektiertes Individuum
y'_1	Individuum y_1 nach Kreuzung
y'_2	Individuum y_2 nach Kreuzung
y''_1	Mutiertes Individuum y'_1
y''_2	Mutiertes Individuum y'_2
$\mathbb{Z}^+$	Menge der positiven ganzen Zahlen
$z(c)$	Zielfunktion in Abhängigkeit von $(c_1, \dots, c_r)$
$z(x)$	Zielfunktionswert der Lösung x
$z(y_j)$	Fitnesswert des Individuums y_j
α	Maschinenumgebung (gesamt)
α_1	Maschinenumgebung
α_2	Maschinenanzahl
β	Nebenbedingungen
γ	Zielfunktion

ω	Gewichtung der Elemente der Input-Matrix
$\{i, j\}$	Disjunktive Kante (von i zu j oder j zu i)
(i, j)	Konjunktive Kante (von i zu j)

1 Einleitung

Heutzutage bewegen sich produzierende Unternehmen in Märkten, die von sehr großer Dynamik geprägt sind. Verursacht wird diese Dynamik vor allem durch die kurzen Produktlebenszyklen und die nicht exakt planbare Auftragslage. Erschwerend kommt hinzu, dass die Kunden nicht nur eine hohe Individualisierbarkeit der Produkte erwarten, sondern auch die Einhaltung der Liefertermine fordern. Bereits kleine Abweichungen zu einem vereinbarten Liefertermin können Konventionalstrafen nach sich ziehen, und im schlimmsten Fall muss mit dem Verlust des Kunden gerechnet werden. Das stellt Unternehmen vor eine große Herausforderung, denn das Produktionssystem muss einerseits flexibel genug sein, um die unterschiedlichsten Produktvarianten produzieren zu können, andererseits muss es unter Berücksichtigung der Liefertermine einen effizienten sowie effektiven Umgang mit den Produktionsfaktoren ermöglichen. Unternehmen sind jedoch durch komplexe Produktions- und Logistikstrukturen charakterisiert, sodass vor allem die zeitliche Einplanung der Aufträge, also die Ablaufplanung, eine große Herausforderung darstellt. Deshalb setzen sie immer häufiger die Simulation als Werkzeug in der Produktionsplanung und -steuerung ein, um die Konsequenzen der unterschiedlichen Ablaufpläne zu veranschaulichen sowie zu untersuchen. Auch wenn die Bewertung der einzelnen Ablaufpläne mit der Simulation möglich ist, ist es bei praxisrelevanten Problemgrößen nahezu unmöglich, ohne systematische Suche eine optimale Lösung bzgl. einer oder mehrerer Zielfunktionen zu finden. Hierfür werden Optimierungsmethoden angewandt, die mit einem systematischen Suchprozess für ein vorliegendes Ablaufplanungsproblem eine hinreichend gute Lösung finden können. Die Optimierungsverfahren werden in diesem Zusammenhang mit der Simulation gekoppelt, um die verschiedenen Lösungsalternativen mit der Simulation zu evaluieren.

Das Ziel dieser Arbeit ist die Entwicklung eines neuen Konzeptes zur Kopplung eines heuristischen Optimierungsverfahrens mit der ereignisdiskreten Simulation im Rahmen der Ablaufplanung. Die dafür eingesetzte Heuristik soll in der Lage sein, Ablaufplanungsprobleme besonders schnell und in angemessener Qualität zu lösen. Innerhalb des neuen Konzeptes soll die Simulation nicht nur die Zielgrößenberechnung übernehmen, sondern durch die Entwicklung eines neuen Kopplungselements auch indirekt am Lösungsprozess beteiligt werden. Auf diese Weise soll ein optimales Zusammenwirken von Simulation und Optimierung entstehen, sodass die effiziente und effektive Lösung von großen sowie schwer lösbaren Ablaufplanungsproblemen ermöglicht wird.

Die Ablaufplanung ist Teil vieler Wissenschaftsbereiche, sodass sie zu Beginn dieser Arbeit nur im Kontext der Produktion betrachtet wird, damit deren Aufgaben und Funktionen, die für diese Arbeit relevant sind, deutlich werden. Die folgende Einführung

grundlegender Begrifflichkeiten und Notationen ermöglicht die Komplexitätsbetrachtung von Optimierungsproblemen, die innerhalb der Ablaufplanung auftreten. Die Ergebnisse dieser Betrachtung dienen zur Ableitung von Anforderungen, die an die Optimierungsmethodik gestellt werden. Anschließend werden exakte sowie heuristische Optimierungsverfahren auf die Eignung zur Lösung von Ablaufplanungsproblemen hin untersucht. Die folgende Analyse der Resultate wichtiger Benchmarktests erlaubt schlussendlich die Wahl einer Optimierungsmethodik, die die Anforderungen erfüllt und somit im weiteren Verlauf der Arbeit mit der ereignisdiskreten Simulation gekoppelt wird.

Im Anschluss werden die Einsatzgebiete der Simulation innerhalb der Ablaufplanung betrachtet, damit der Verwendungszweck des neuen Konzeptes verdeutlicht werden kann. Daran anknüpfend werden die derzeitigen Konzepte zur simulationsbasierten Optimierung erläutert, damit die Forschungslücke, die durch das neue Konzept geschlossen werden soll, identifiziert werden kann. Mit Hilfe dieser Forschungslücke werden sowohl Anforderungen an das Konzept zur Kopplung von Simulation und Optimierung, als auch Anforderungen an die eingesetzte Optimierungsmethodik, die durch die Kopplung mit der Simulation entstehen, abgeleitet. Zunächst wird geprüft, ob das ausgewählte Optimierungsverfahren auch diese Anforderungen erfüllt. Nachfolgend wird das in dieser Arbeit entwickelte Konzept zur Kopplung von Simulation und Optimierung erläutert. Die detaillierten Ausführungen des neuen Kopplungselements sowie dessen Kernkomponenten verdeutlichen die Funktionsweise und Besonderheiten des neuen Konzeptes. Die abschließende Validierung gibt Aufschluss, inwieweit die identifizierte Forschungslücke mit dem neuen Konzept zur simulationsbasierten Optimierung von Ablaufplänen geschlossen wird und identifiziert mögliche Schwachstellen, die durch zukünftige Forschung genauer untersucht werden sollten.

2 Ablaufplanung

Die Ablaufplanung ist Gegenstand unterschiedlicher wissenschaftlicher Disziplinen und Forschungsaktivitäten wie der Mathematik, Informatik, Wirtschaftswissenschaft und Ingenieurswissenschaft (vgl. Klemmt, 2012). In der vorliegenden Arbeit wird die Ablaufplanung im ingenieurswissenschaftlichen Kontext betrachtet und ist somit als Teil der Produktion zu verstehen.

2.1 Einordnung der Ablaufplanung in den Kontext der Produktion

Der Begriff Produktion, im Weiteren auch als Produktionsprozess und Throughput bezeichnet, beschreibt die „Kombination und Transformation von Produktionsfaktoren zum Zwecke der Erstellung von Sach- und/oder Dienstleistungen“ (Zäpfel, 1982, S. 1). Die für den Produktionsprozess notwendigen Faktoren repräsentieren den Input (vgl. Domschke et al., 1997) und lassen sich nach Gutenberg (1983) wie folgt klassifizieren:

- Menschliche Arbeitskraft
- Betriebsmittel
- Werkstoffe

Die durch den Throughput hergestellten Produkte werden in materielle und immaterielle Güter differenziert (vgl. Domschke et al., 1997) und als Output bezeichnet (vgl. Hoitsch, 1993). Abbildung 1 stellt den Sachverhalt zwischen Input, Throughput und Output dar.

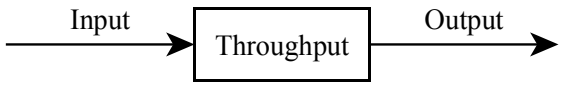

Abbildung 1: Input, Throughput und Output (Domschke et al., 1997)

Werden der Input, Throughput und Output aggregiert, entsteht das Produktionssystem (vgl. Zäpfel, 1982). Das Produktionssystem ist ein Subsystem der Unternehmung und steht innerhalb des Gesamtsystems Unternehmung mit anderen Subsystemen sowie deren Umwelt in Verbindung (vgl. Schwartz, 2004).

2.1.1 Managementebenen in der Produktion

Das Management des Produktionssystems umfasst sämtliche Planungs-, Steuerungs- und Überwachungsaufgaben (vgl. Hansmann, 2001). Diese Aufgaben sind die Kernelemente des Produktionsmanagements und entsprechend ihrer zeitlichen Reichweite (s. Abbildung 2) lassen sie sich den folgenden drei hierarchischen Ebenen zuordnen (vgl. Zäpfel, 2000).

1. Strategisches Produktionsmanagement
2. Taktisches Produktionsmanagement
3. Operatives Produktionsmanagement

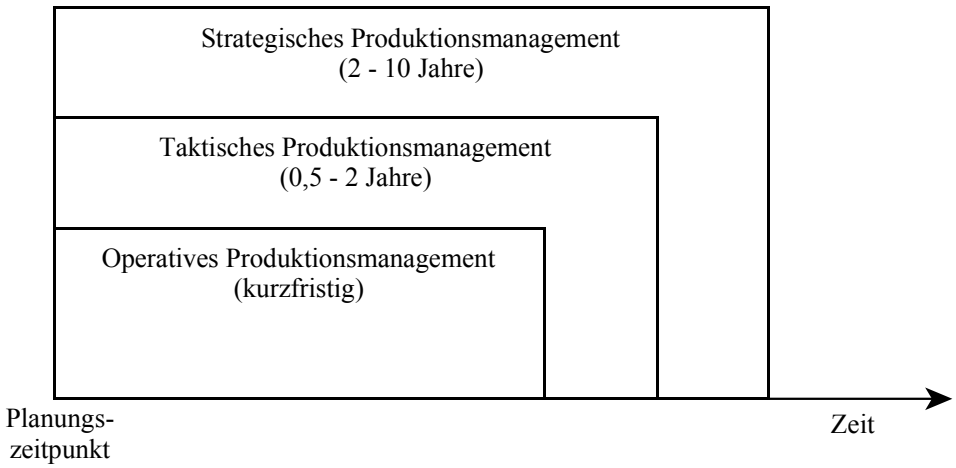

Abbildung 2: Zeitliche Reichweite der Managementebenen (Domschke et al., 1997)

Das strategische Produktionsmanagement leitet aus den übergeordneten Unternehmenszielen (z.B. mit welchen Produkten auf welchen Märkten) das langfristige Produktionsprogramm ab (vgl. Acker, 2011) und bestimmt die hierfür notwendigen Ressourcen (vgl. Kern, 1992). Im Mittelpunkt stehen der Erhalt und Aufbau eines leistungs- und wettbewerbsfähigen Produktionssystems (vgl. Hoitsch, 1993).

Das taktische Produktionsmanagement initiiert die erforderlichen Maßnahmen, um die Vorgaben und Ziele des strategischen Produktionsmanagement umzusetzen (vgl. Schwartz, 2004). Hierzu zählen Programmentscheidungen, die Auswirkungen auf das Produktportfolio haben, sowie Ausstattungsentscheidungen, die das Produktionssystem an die Unternehmensumwelt anpassen (vgl. Zäpfel, 1996).

Das operative Produktionsmanagement muss eine effektive und effiziente Produktion unter den getroffenen Entscheidungen des strategischen und taktischen Produktionsmanagements durch laufende Anpassungsentscheidungen gewährleisten (vgl. Zäpfel, 2000). Zu den Kernaufgaben des operativen Produktionsmanagements zählen somit (vgl. Domschke et al., 1997):

1. Produktionsprogrammplanung
2. Bereitstellungsplanung
3. Produktionsprozessplanung

Die drei Hauptaufgaben des operativen Produktionsmanagements repräsentieren im Allgemeinen die wesentlichen Elemente der Produktionsplanung (vgl. Adam, 1996).

2.1.2 Produktionsplanung und -steuerung

Die einzelnen Aufgaben der Produktionsplanung und -steuerung müssen chronologisch und im Detail betrachtet werden, damit die Rolle und Funktion der Ablaufplanung im Kontext der Produktionsplanung und -steuerung skizziert werden kann (s. Abbildung 3).

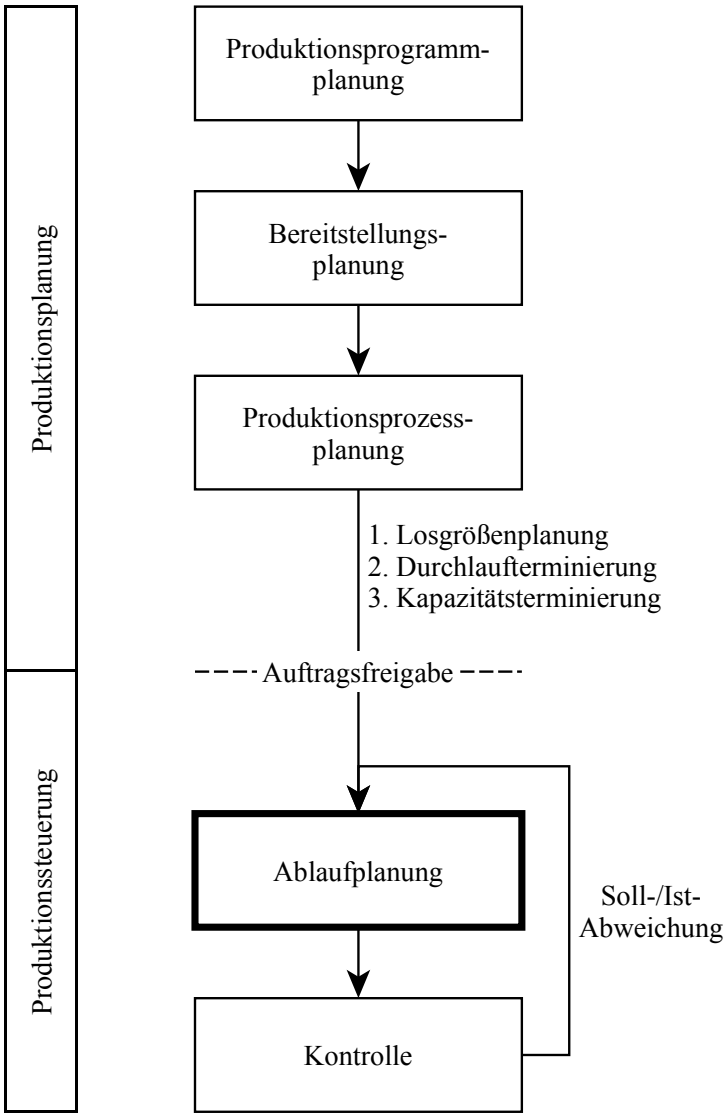

Abbildung 3: Produktionsplanung und -steuerung (nach Schwartz, 2004 u. Domschke et al., 1997)

In der Produktionsprogrammplanung wird die Art und Menge der zu produzierenden Güter für einen definierten Planungszeitraum mit Hilfe von Absatzprognosen und/oder Kundenaufträgen determiniert (vgl. Zäpfel, 1996; Scheer, 1997). Anhand der Menge der zu produzierenden Güter werden im Rahmen der Bereitstellungsplanung die für die Produktion benötigten Produktionsfaktoren hinsichtlich Qualität, Quantität und Bereitstellungszeitpunkt ermittelt (vgl. Domschke et al., 1997). Die Produktionsprozessplanung dient zur zeitlichen und räumlichen Koordination des Fertigungsprozesses, aber auch zur

Steuerung der Produktion und lässt sich in Losgrößenplanung, Durchlauf- und Kapazitätsterminierung untergliedern (vgl. Domschke et al., 1997). Innerhalb der Losgrößenplanung werden kostenoptimale Losgrößen und deren zeitliche Verteilung innerhalb eines Produktionszeitraumes gesucht (vgl. Domschke et al., 1997). Die Durchlaufterminierung ermittelt unter Beachtung der technologischen Bearbeitungsreihenfolge der Aufträge, aber unter Vernachlässigung der bestehenden Kapazitäten, die groben Start- und Endtermine für die Arbeitsvorgänge (vgl. Schwartz, 2004). Im Verlauf der Kapazitätsterminierung wird der zur Auftragsrealisierung benötigte Kapazitätsbedarf dem verfügbaren Kapazitätsbestand gegenübergestellt und im Falle einer Abweichung ein Kapazitätsabgleich durchgeführt (vgl. Domschke et al., 1997). Anschließend werden die Aufträge zur Produktion freigegeben und die Produktionsplanung ist abgeschlossen. Der Moment der Auftragsfreigabe kennzeichnet den Übergang von der Produktionsplanung zur Produktionssteuerung (vgl. Schwartz, 2004). Im Rahmen der nachfolgenden Ablaufplanung werden die Auftragsreihenfolgen für die einzelnen Ressourcen bestimmt und die einzelnen Arbeitsvorgänge der Aufträge zeitlich detailliert eingeplant (vgl. Schwartz, 2004). Während der Fertigung werden die geplanten Soll-Daten mit den Fertigungs-Ist-Daten abgeglichen (vgl. Zäpfel, 1982). Liegt eine Abweichung außerhalb des Toleranzbereiches vor, werden entsprechende Maßnahmen wie z.B. die Änderung des Ablaufplans eingeleitet.

2.2 Definition Ablaufplanung

Die Ablaufplanung ist eine detaillierte zeitliche und räumliche Planung von Produktionsprozessen (vgl. Adam, 1969; Seelbach, 1975; Gutenberg, 1983) und kann in drei Planungsschritte gegliedert werden (vgl. Zelewski et al., 2008):

1. Bearbeitungsplanung

Im Rahmen der Bearbeitungsplanung werden Maschinenfolgen für die einzelnen Produkte erstellt, indem jeder einzelne Bearbeitungsschritt eines Auftrages einer bestimmten Maschine zugeordnet wird (vgl. Acker, 2011). Diese Maschinenfolge ist meist determiniert und kann nicht beeinflusst werden (vgl. Schwartz, 2004).

2. Reihenfolgeplanung

Das zentrale Element der Ablaufplanung ist die Reihenfolgeplanung. Hier wird die Reihenfolge der zu bearbeitenden Aufträge für jede einzelne Maschine bestimmt (vgl. Acker, 2011). Die Auftragsfolge repräsentiert das Ergebnis der Reihenfolgeplanung (vgl. Schwartz, 2004).

3. Terminplanung

Im letzten Schritt werden den einzelnen Arbeitsvorgängen der Aufträge sowie den Aufträgen als Ganzes frühestmögliche und die spätest zulässigen Start- und Endzeitpunkte zugeordnet (vgl. Zäpfel, 1982).

Das Ergebnis der Ablaufplanung ist ein nach bestimmten Zielfunktionen hinreichend optimaler Maschinenbelegungsplan mit einer zeitlichen Reichweite von 7 bis 14 Tagen, der ein vorher definiertes Ablaufplanungsproblem löst (vgl. Kern, 1992).

2.3 Definition Ablaufplanungsproblem

Ein Ablaufplanungsproblem ist „ein durch ein Tripel $\alpha|\beta|\gamma$ dargestelltes Optimierungsproblem, bei dem eine Zuteilung von“ (Jaehn und Pesch, 2014, S. 12) n Jobs J_i $(i = 1, \dots, n)$ zu m Maschinen M_j $(j = 1, \dots, m)$ für ein bestimmtes Zeitintervall ermittelt werden muss (vgl. Graham et al., 1979; Brucker, 2004). Das Ergebnis ist ein nach bestimmten Kriterien zulässiger Ablaufplan (*engl. schedule*), der als Gantt-Diagramm dargestellt werden kann. Dieses kann sowohl maschinen- als auch joborientiert sein (vgl. Brucker, 2004). Jeder Job J_i wird dabei durch folgende Variablen (s. Tabelle 1) genau spezifiziert (vgl. Pinedo, 2012).

Tabelle 1: Übersicht Jobvariablen

Bezeichnung	Bedeutung
m_i	Anzahl der Arbeitsschritte von J_i
p_{ij} (*engl. processing time*)	Bearbeitungszeit von J_i auf M_j
r_i (*engl. release date*)	Auftragsfreigabetermin von J_i, d.h. frühestmöglicher Zeitpunkt zur Einplanung von J_i
d_i (*engl. due date*)	Liefertermin von J_i, d.h. Zeitpunkt, an dem J_i fertiggestellt sein sollte
w_i (*engl. weight*)	Gewichtung von J_i, z.B. Wertigkeit, Kosten, Priorität (vgl. Jaehn und Pesch, 2014)

Die von Graham et al. (1979) eingeführte Dreifeldnotation ermöglicht es, Ablaufplanungsprobleme nach ihrer Maschinenumgebung α, den Ablauf- und Nebenbedingungen β sowie der angestrebten Zielfunktion γ zu klassifizieren (vgl. Jaehn und Pesch, 2014). Die Basis für die nachfolgenden Ausführungen bildet das Schema von Graham et al. (1979).

2.3.1 Klassifikation nach Maschinenumgebung

Das erste Feld der Notation wird durch $\alpha = \alpha_1\alpha_2$ spezifiziert. Die Maschinenumgebung wird durch α_1 und die Maschinenanzahl durch α_2 beschrieben. Ist $\alpha_1 \in \{\emptyset, P, Q, R\}$ bestehen alle Jobs J_i $(i = 1, \dots, n)$ aus nur einem Arbeitsschritt, der auf jeder Maschine M_j $(j = 1, \dots, m)$ mit der Bearbeitungszeit p_{ij} durchgeführt werden kann.

$\alpha_1 = \emptyset$, nur eine Maschine M_1 steht zur Bearbeitung aller Aufträge zur Verfügung. Die Bearbeitungszeit ist $p_{i1} = p_i$.

$\alpha_1 = P$, identische parallele Maschinen stehen zur Verfügung und können gleichzeitig eingesetzt werden. Die Bearbeitungszeit ist $p_{ij} = p_i$ für $j = 1, \dots, m$).

$\alpha_1 = Q$, uniforme parallele Maschinen mit individuellen Bearbeitungszeiten stehen zur Verfügung und können gleichzeitig eingesetzt werden. Die Bearbeitungszeit ist $p_{ij} = p_i/v_j$, wobei v_j die Bearbeitungsgeschwindigkeit der Maschine M_j für $(j = 1, \dots, m)$ ist (vgl. Pinedo, 2012).

$\alpha_1 = R$, heterogene parallele Maschinen stehen zur Verfügung und können gleichzeitig eingesetzt werden. Die Bearbeitungszeit ist $p_{ij} = p_i/v_{ij}$. Die Bearbeitungsgeschwindigkeit v_{ij} ist sowohl maschinen- als auch jobabhängig (vgl. Pinedo, 2012; Domschke et al., 1997).

Bestehen die einzelnen Jobs aus mehreren Arbeitsschritten, die auf verschiedenen Maschinen ausgeführt werden müssen, ist $\alpha_1 \in \{O, F, J\}$ (vgl. Domschke et al., 1997).

$\alpha_1 = O$ (Open Shop): Jeder Auftrag J_i für $(i = 1, \dots, n)$ muss jede Maschine M_j für $(j = 1, \dots, m)$ genau einmal durchlaufen. Es existiert keine vorgegebene Reihenfolge, in der die Jobs auf den Maschinen bearbeitet werden müssen (vgl. Pinedo, 2012).

$\alpha_1 = F$, (Flow Shop): Jeder Auftrag J_i für $(i = 1, \dots, n)$ durchläuft jede Maschine M_j für $(j = 1, \dots, m)$ genau einmal. Alle Aufträge haben die gleiche Reihenfolge (vgl. Pinedo, 2012) und diese entspricht der Maschinenfolge $(M_1, M_2, \dots, M_m)$ (vgl. Domschke et al., 1997).

$\alpha_1 = J$, (Job Shop): Jeder Auftrag J_i für $(i = 1, \dots, n)$ durchläuft die Maschinen in einer individuell festgelegten Reihenfolge (vgl. Pinedo, 2012). Aufträge können einzelne Maschinen auslassen, aber auch mehrfach auf einer Maschine bearbeitet werden (vgl. Domschke et al., 1997; Pinedo, 2012).

2.3.2 Klassifikation nach Ablaufeigenschaften und Nebenbedingungen

Das zweite Feld des Tripels klassifiziert die Eigenschaften und Nebenbedingungen der Jobs, für die ein Ablaufplan erzeugt werden soll. Graham et al. (1979) schlagen eine

Schreibweise mit sechs Komponenten $(\beta_1, \dots, \beta_6)$ vor, um die Auftragseigenschaften bzw. Ablaufcharakteristika zu spezifizieren. In der Literatur existieren noch umfangreichere Ausführungen, diese sollen aber an dieser Stelle unbeachtet bleiben, da nur ein kurzer Überblick über mögliche Jobspezifikationen gegeben werden soll.

$\beta_1 \in \{pmnt, \emptyset\}$

$pmnt$ (*engl. preemption*): Die Bearbeitung der Aufträge darf unterbrochen und zu einem späteren Zeitpunkt (ggf. auf einer anderen Maschine) wieder fortgesetzt werden (vgl. Domschke et al., 1997).

$\emptyset$: Die Bearbeitung der Aufträge darf auf keiner Maschine unterbrochen werden. Wartezeiten zwischen zwei Arbeitsschritten hingegen sind erlaubt.

$\beta_2 \in \{res, res1, \emptyset\}$

res (*engl. ressource)*: Die zur Verfügung stehenden s Ressourcen R_h für $(h = 1, \dots, s)$ sind beschränkt. Es wird angenommen, dass jeder Job J_i für $(i = 1, \dots, n)$ einen Teil r_{hi} von R_h für $(h = 1, \dots, s)$ während der Bearbeitung beansprucht.

$res1$: Es existiert nur eine Ressource zur Bearbeitung.

$\emptyset$: Es existieren keine Ressourcenbeschränkungen.

$\beta_3 \in \{prec, \emptyset\}$

$prec$ (*engl. precedence relations*): Zwischen den Jobs J_i für $(i = 1, \dots, n)$ bestehen Reihenfolgebeziehungen, die in Form eines gerichteten azyklischen Graphen repräsentiert werden. Enthält der Graph z.B. einen Weg von J_1 nach J_2, dann muss J_1 beendet sein, bevor die Bearbeitung von J_2 beginnen darf.

$\emptyset$: Es existieren keine Reihenfolgebeziehungen.

$\beta_4 \in \{r_i, \emptyset\}$

r_i: Der Job J_i für $(i = 1, \dots, n)$ besitzt einen individuellen Auftragsfreigabetermin, sodass vor Zeitpunkt r_i die Bearbeitung des Jobs J_i nicht begonnen werden darf.

$\emptyset$: Es wird angenommen, dass $r_i = 0$ ist.

$\beta_5 \in \{m_i \leq m_u, \emptyset\}$

$m_i \leq m_u$: Für den Fall, dass $\alpha_1 = J$, kann für die Jobs J_i für $(i = 1, \dots, n)$ eine maximale Anzahl an Arbeitsschritten m_u festgelegt werden.

$\emptyset$: Die Anzahl der Arbeitsschritte für alle Jobs J_i für $(i = 1, \dots, n)$ ist beliebig.

$\beta_6 \in \{p_{ij} = p, p_l \leq p_{ij} \leq p_u, \emptyset\}$

$p_{ij} = p$: Alle Arbeitsschritte der Aufträge J_i für $(i = 1, \dots, n)$ haben eine einheitliche Bearbeitungszeit p.

$p_l \leq p_{ij} \leq p_u$: Die Bearbeitungszeiten der Aufträge J_i für $(i = 1, \dots, n)$ sind beschränkt und müssen zwischen der minimalen p_l und maximalen Bearbeitungszeit p_u liegen.

$\emptyset$: Für alle Aufträge J_i für $(i = 1, \dots, n)$ sind beliebige Bearbeitungszeiten zugelassen.

2.3.3 Klassifikation nach Zielfunktion

Im Rahmen der Ablaufplanung werden zulässige Maschinenbelegungspläne gesucht, die sowohl die determinierte Maschinenfolge als auch weitere in Form von Ablaufeigenschaften und Nebenbedingungen beschriebene Restriktionen (z.B. Auftragsfreigabezeitpunkte) einhalten (vgl. Acker, 2011). Es ist allerdings nicht ausreichend, ein Ablaufplanungsproblem mit einem „nur“ zulässigen Maschinenbelegungsplan zu lösen, vielmehr muss der Fokus bei der Ablaufplanung auf die Optimierung gewisser Zielgrößen gelegt werden (vgl. Klemmt, 2012). Nach Domschke et al. (1997) lassen sich diese Zielgrößen in folgende Kategorien einteilen:

- Durchlaufzeitbezogene Ziele
- Kapazitätsorientierte Ziele
- Terminorientierte Ziele

Durchlaufzeitbezogene Ziele haben positiven Einfluss auf die ablaufbedingten Kosten. Kürzere Durchlaufzeiten reduzieren die Kapitalbindungskosten der Aufträge und ermöglichen eine Abnahme der Lagerungszeit der Zwischenprodukte, was eine Reduktion der Lagerhaltungskosten nach sich zieht (vgl. Schwartz, 2004; Seelbach, 1975; Acker, 2011). Des Weiteren werden durch kurze Durchlaufzeiten Erlöse früher erzielt (vgl. Hansmann, 2001). Schnelle Produktionsabläufe haben nicht nur positive Kostenwirkungen, sondern auch Wettbewerbsvorteile. Kürzere Durchlaufzeiten erlauben den Unternehmen auf Änderungen der Nachfrage besser reagieren zu können (vgl. Hoitsch, 1993).

Die Durchlaufzeit D_i basiert auf dem Auftragsfreigabetermin r_i und dem Zeitpunkt der Auftragsfertigstellung C_i.

$$D_i = C_i - r_i, \text{ für } r_i = 0\text{: } D_i = C_i \tag{2-1}$$

Die Wartezeit W_{ij} gibt die Zeit an, die Auftrag J_i vor Maschine M_j wartet, während W_i die gesamte Wartezeit von Auftrag J_i angibt.

$$W_i = \sum_{j=1}^{m} W_{ij} \tag{2-2}$$

Aus der Durchlaufzeit und Wartezeit ergeben sich somit folgende durchlaufzeitbezogene Zielfunktionen:

- Minimierung der Summe der (gewichteten) Fertigstellungszeiten

$$\gamma = \sum_{i=1}^{n}(w_i)C_i \tag{2-3}$$

- Minimierung der Summe der Durchlaufzeiten

$$\gamma = \sum_{i=1}^{n} D_i \tag{2-4}$$

- Minimierung der maximalen Durchlaufzeit

$$\gamma = D_{max} = \max\{D_i \mid i = 1, \dots, n\} \tag{2-5}$$

- Minimierung der Summe der Wartezeiten

$$\gamma = \sum_{i=1}^{n} W_i \tag{2-6}$$

Kapazitätsbezogene Ziele fokussieren auf die Auslastung der für den Bearbeitungsprozess benutzten Ressourcen. Bei einer hohen Auslastung der Maschinen sinken die Leerzeiten und somit reduzieren sich die Leerkosten, sodass auch kapazitätsorientierte Ziele einen positiven Effekt auf die Fertigungskosten haben (vgl. Zäpfel, 1982).

Die Gesamtzykluszeit C_{max} gibt den Zeitraum von dem Beginn der Bearbeitung des ersten Auftrages J_1 bis zur Fertigstellung des letzten Auftrages J_n an. Sie kann somit auch als Belegungszeit interpretiert werden (vgl. Domschke et al., 1997).

$$C_{max} = max\{C_i \mid i = 1, \dots, n\} \tag{2-7}$$

Die Leerzeit I_j ist die Summe der Zeiten, in denen die Maschine M_j innerhalb der Gesamtzykluszeit keinen Auftrag bearbeitet (vgl. Domschke et al., 1997).

$$I_j = C_{max} - \sum_{i=1}^{n} p_{ij} \tag{2-8}$$

Aus der Zykluszeit und Leerzeit ergeben sich somit folgende Optimierungsfunktionen:

- Minimierung der Summe der Leerzeiten

$$\gamma = \sum_{j=1}^{m} I_j \tag{2-9}$$

- Minimierung der Gesamtzykluszeit

$$\gamma = C_{max} \tag{2-10}$$

Terminorientierte Ziele dienen hauptsächlich der Verbesserung der Termintreue und versuchen entstehende Kosten durch verspätete Aufträge zu verhindern. Mangelnde Termintreue kann hohe Konventionalstrafen und Opportunitätskosten, verursacht durch die Abwanderung von Kunden, nach sich ziehen (vgl. Siegel, 1974). Ausgangsbasis für

die terminorientierte Optimierung sind die individuellen Fertigstellungszeiten der Aufträge C_i sowie deren Liefertermine d_i.

Die Terminabweichung L_i eines Jobs repräsentiert die Abweichung der Fertigstellungszeit eines Jobs zu seinem Liefertermin.

$$L_i = C_i - d_i \tag{2-11}$$

Für $L_i > 0$ ist der Job verspätet, für $L_i \leq 0$ kann der Job vor dem Fälligkeitstermin fertiggestellt werden. Daraus resultierend wird die Verspätung T_i von J_i wie folgt dargestellt:

$$T_i = \max(C_i - d_i, 0) = \max(L_i, 0) \tag{2-12}$$

Die Terminüberschreitung U_i bietet eine weitere Analysemöglichkeit, ob geforderte Fälligkeiten eingehalten werden oder nicht.

$$U_i = \begin{cases} 1 & \text{wenn } C_i > d_i \\ 0 & \text{sonst} \end{cases} \tag{2-13}$$

Anhand dieser Zielgrößen lassen sich folgende terminorientierte Zielfunktionen formulieren.

- Minimierung der maximalen Terminabweichung

$$\gamma = L_{max} = \max\{L_i \mid i = 1, \dots, n\} \tag{2-14}$$

- Minimierung der Summe der (gewichteten) Verspätung

$$\gamma = (w)T = \sum_{i=1}^{n} (w_i) T_i \tag{2-15}$$

- Minimierung der Summe der (gewichteten) Terminüberschreitungen

$$\gamma = \sum_{i=1}^{n} (w_i) U_i \tag{2-16}$$

Zwischen den beschriebenen Zielgrößen können Beziehungen bestehen, die als komplementär, konkurrierend oder indifferent bezeichnet werden können (vgl. Acker, 2011). Sind die Zielgrößen zueinander komplementär, wirkt sich die Verfolgung des einen Ziels positiv auf das andere Ziel aus. So beeinflusst die Minimierung der Summe der Fertigstellungszeitpunkte (2-3) auch die Minimierung der Summe der Durchlaufzeiten (2-4). Nicht nur Zielgrößen innerhalb einer Zielkategorie können komplementär zueinander sein, sondern auch Zielgrößen verschiedener Zielkategorien können sich positiv beeinflussen (vgl. Rinnooy Kan, A. H. G., 1976). Wird im Rahmen der Ablaufplanung eine Minimierung der Durchlaufzeit angestrebt (2-4), so wird auch gleichzeitig die Summe der Verspätungen minimiert (2-15).

Beeinflusst eine Zielverfolgung eine andere Zielgröße negativ, so sind diese Ziele konfliktionär zueinander. Wird eine durchlaufzeitorientierte Zielgröße wie die

Minimierung der Wartezeit (2-6) angestrebt, geschieht dies auf Kosten der Leerzeit (2-9), die eine kapazitätsorientierte Zielgröße darstellt. Gutenberg (1983) beschreibt diesen Zusammenhang als das Dilemma der Ablaufplanung. Bei zwei konkurrierenden Zielgrößen kann kein Ablaufplan gefunden werden, der beide Ziele optimal erfüllt; vielmehr muss mit Methoden der Mehrzieloptimierung nach einer Kompromisslösung gesucht werden (vgl. Acker, 2011; Fandel, 1972). In diesem Zusammenhang repräsentiert die Paretomenge alle optimalen Kompromisse (vgl. Weigert und Rose, 2011). Woolsey, R. E. D. (1990) schlägt die lexikographische Ordnung vor, um Zielkonflikte zu überwinden. Dabei wird das Optimierungsproblem mit zwei konkurrierenden Zielgrößen zuerst hinsichtlich einer Zielgröße optimiert und anschließend die Kompromisslösung gewählt, die bezüglich der zweiten Zielgröße am besten ist. Die Zielgewichtung nach van Wassenhove und Gelders (1980) ist eine weitere Alternative zur Kompromissfindung. Die einzelnen Zielgrößen werden anhand ihrer Relevanz bewertet und miteinander verknüpft. So entsteht eine Zielfunktion mit gewichteten Zielgrößen, die die optimalen Kompromisslösungen beschreibt.

Sind zwei Zielgrößen hingegen völlig unabhängig und es besteht keine Wechselwirkung zwischen ihnen, sind sie indifferent.

2.3.4 Typen von Ablaufplanungsproblemen

Damit Ablaufplanungsprobleme gelöst werden können, ist es nicht nur notwendig, das Problem mit all seinen Ablauf- und Nebenbedingungen sowie der Zielfunktion zu beschreiben, es bedarf auch noch einer Differenzierung der Ablaufplanungsprobleme in verschiedene Typen. Hier liegt der Fokus vor allem auf dem Optimierungsgegenstand sowie den Eigenschaften der Daten, die für die Lösung des Ablaufplanungsproblems notwendig sind.

2.3.4.1 Primale und duale Ablaufplanungsprobleme

Ist die technologische Bearbeitungsreihenfolge (Maschinenfolge) für jeden Auftrag determiniert und eine optimale Auftragsfolge soll gefunden werden, liegt ein primales Ablaufplanungsproblem vor (vgl. Acker, 2011). Ist hingegen die Auftragsfolge gegeben und die Maschinenfolge der Aufträge soll bestimmt werden, wird von einem dualen Ablaufplanungsproblem gesprochen (vgl. Acker, 2011).

2.3.4.2 Statische und dynamische Ablaufplanungsprobleme

Liegen zum Planungszeitpunkt alle erforderlichen Aufträge bereit, d.h. $r_i = 0$, dann liegt ein statisches Ablaufplanungsproblem vor (vgl. Schwartz, 2004). In diesem Fall wird für den ganzen Auftragsbestand ein Ablaufplan erstellt und erst nachdem dieser vollständig abgearbeitet ist, wird ein neuer Ablaufplan angefertigt (vgl. Schwartz, 2004). Im

Gegensatz dazu stehen die dynamischen Ablaufplanungsprobleme. Hier liegen zum Planungsbeginn noch nicht alle Aufträge vor, da sie unterschiedliche Auftragsfreigabezeitpunkte besitzen (vgl. Domschke et al., 1997). Demzufolge müssen neu ankommende Aufträge bei der Maschinenbelegung kontinuierlich berücksichtigt werden (vgl. Schwartz, 2004).

2.3.4.3 Deterministische und stochastische Ablaufplanungsprobleme

Bei deterministischen Ablaufplanungsproblemen sind alle für die Planungsperiode benötigten Daten, wie die Auftragsfreigabezeitpunkte, die Maschinenfolge, die Anzahl und Eigenschaften der Maschinen sowie die Bearbeitungszeiten der Ressourcen, a priori bekannt (vgl. Domschke et al., 1997; Jacob und Adam, 1990). Ist nur eine dieser Daten unbekannt oder unsicher, liegt ein stochastisches Ablaufplanungsproblem vor (vgl. Seelbach, 1975).

Die Produktionssteuerung unterliegt in der Realität laufend Veränderungen und Unsicherheiten, bedingt durch Maschinenausfälle oder Nachfrageschwankungen, auf die angemessen reagiert werden muss. Die betriebliche Praxis wird folglich nach den oben genannten Differenzierungsmöglichkeiten mit dynamisch-stochastischen Modellen abgebildet (vgl. Acker, 2011). Dynamische Modelle fordern allerdings eine kontinuierliche Planänderung oder Neuplanung von Auftragsfolgen, verursacht durch eintretende Veränderungen wie neue Auftragsfreigaben, sodass eine unzweckmäßig hohe Planungsfrequenz erforderlich werden kann (vgl. Kurbel und Meynert, 1987). Aus diesem Grund werden in Unternehmen meist statisch-deterministische Modelle betrachtet (vgl. Acker, 2011). Auch in dieser Arbeit wird Abstand von dynamisch-stochastischen Modellen genommen, sodass im Folgenden nur primale statisch-deterministische Ablaufplanungsprobleme betrachtet werden.

2.4 Komplexität von Ablaufplanungsproblemen

Primale Ablaufplanungsprobleme, die Reihenfolgeprobleme darstellen, sind Teil der kombinatorischen Optimierung (vgl. Domschke und Drexl, 2005). Ziel der kombinatorischen Optimierung ist das Finden einer optimalen Lösung für das zugrunde liegende Optimierungsproblem aus einer endlichen Lösungsmenge (vgl. Domschke und Drexl, 2005). Die Zeit, die ein Algorithmus zur Lösungsfindung benötigt, hängt maßgeblich von der Komplexität des vorliegenden Optimierungsproblems ab. In der Komplexitätstheorie werden Ablaufplanungsprobleme deshalb in die zwei Klassen P (engl. *polynomial*) und NP (engl. *nondeterministic polynomial*) unterteilt. Ein Problem gehört der Klasse P an, wenn die Anzahl der Rechenoperationen eines Lösungsverfahrens im Verhältnis zur Problemgröße nicht stärker als mit einem Polynom wächst (vgl. Troßmann, 1996). Probleme der Klasse P sind somit effizient lösbar (vgl. Domschke und Drexl, 2005). In

der Literatur wird oftmals behauptet, dass die Klasse NP alle Probleme enthält, die nicht in Polynomialzeit lösbar sind. Dies ist allerdings falsch, da die Menge aller in polynomieller Zeit lösbaren Probleme (vermutlich) eine Teilmenge von NP ist ($P \subseteq NP$) (vgl. Krumke und Noltemeier, 2012). Vielmehr sind es die NP-schweren Probleme, die nicht in Polynomialzeit lösbar sind (vgl. Domschke und Drexl, 2005). Die Rechenzeit wächst mit der Problemgröße exponentiell, sodass bereits kleine Probleme mit dem heutigen Stand der Technik nicht gelöst werden können (vgl. Acker, 2011). Knust (2009) gibt einen umfassenden Überblick über den aktuellen Stand der Forschung bezüglich der Zuordnung von Ablaufplanungsproblemen zu den Klassen P und NP. Hier ist erkennbar, dass der Großteil der Ablaufplanungsprobleme NP-schwer und somit nicht effizient lösbar ist.

Eine Zuteilung von Ablaufplanungsproblemen zu den Komplexitätsklassen P und NP ist von großer Bedeutung, denn davon ist es abhängig, welche Lösungsverfahren zur Problemlösung herangezogen werden können (s. Kapitel 3.1 – 3.4). Einfache Reduktionsregeln (s. Abbildung 4) erlauben ein Ablaufplanungsproblem schnell der Klasse P oder NP zuzuordnen (vgl. Brucker, 2004).

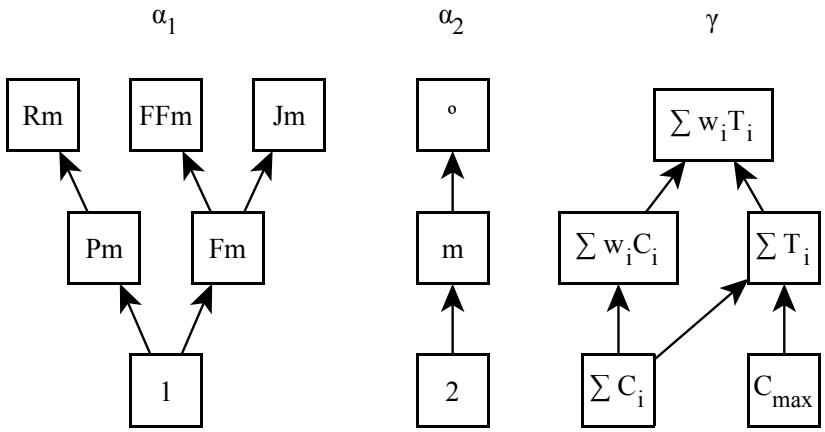

Abbildung 4: Reduktionsregeln (Klemmt, 2011)

Dafür ist es notwendig, die Komplexitätsordnung des Ausgangsproblems zu kennen, denn ist dieses Problem NP-schwer, so ist auch das Problem, welches durch die Reduktionsregeln erzeugt werden kann, NP-schwer. Im folgenden Beispiel soll mit Hilfe der Reduktionsregeln nachgewiesen werden, dass es sich bei $Jm||\sum w_i C_i$ um ein NP-schweres Problem handelt. Da $F2||\sum C_i$ ein NP-schweres Problem ist, lässt es sich wie folgt reduzieren bzw. spezialisieren (vgl. Garey et al., 1976).

$$F2||\sum C_i \propto J2||\sum C_i \propto Jm||\sum C_i \propto Jm||\sum w_i C_i \quad \blacksquare \tag{2-17}$$

2.5 Das Job-Shop-Problem

Das Job-Shop-Problem ist wie folgt definiert (vgl. Schutten, 1998; Jain und Meeran, 1999):

Es müssen n Jobs J_i für $(i = 1, \dots, n)$ zu m Maschinen M_j für $(j = 1, \dots, m)$ zugeordnet werden. Jeder Job J_i muss auf jeder Maschine M_j genau einmal bearbeitet werden und besteht aus m_i Arbeitsschritten $O_{i1}, \dots, O_{im_i}$, die in einer determinierten Reihenfolge eingeplant werden müssen. Es existieren $N = \sum_{i=1}^{n} m_i$ Arbeitsschritte insgesamt. Für den Fall, dass $m_i = m$, existieren $N = n \times m$ Operationen. O_{ij} repräsentiert den Arbeitsschritt von Job J_i, der auf Maschine M_j für die Bearbeitungszeit p_{ij} ohne Unterbrechung ausgeführt werden muss. Jeder Job hat eine individuelle Maschinenfolge und ist unabhängig von den anderen Jobs. Jede Maschine kann nur einen Job gleichzeitig bearbeiten und jeder Job kann zeitgleich auf nur einer Maschine bearbeitet werden. Die Dauer, bis alle Operationen von allen Jobs ausgeführt wurden, wird als Gesamtzykluszeit C_{max} bezeichnet (s. Kapitel 2.3.3). Für alle Jobs gilt $r_i = 0$ und auch alle Maschinen stehen zum Zeitpunkt 0 zur Verfügung. Rüstzeiten der Maschinen und Transportzeiten der Jobs zwischen den Maschinen werden vernachlässigt. Wartezeiten vor den Maschinen sowie Leerzeiten der Maschinen sind zulässig (vgl. Henning, 2002).

Das Ziel des Job-Shop-Problems ist die Gesamtzykluszeit C_{max} zu minimieren. Ein Maschinenbelegungsplan ist demnach zulässig, wenn er die oben beschriebenen Bedingungen erfüllt. Das Job-Shop-Problem kann mit Hilfe der 3-Feld-Notation von Graham et al. (1979) als $J||C_{max}$ ausgedrückt werden.

Lenstra et al. (1977) lieferten den mathematischen Beweis, dass es sich bei $J||C_{max}$ um ein NP-schweres Problem handelt. Tabelle 2 gibt einen kleinen Überblick über die Komplexitätszuordnungen von verwandten Problemen. $J||C_{max}$ stellt dabei die Generalisierung der aufgelisteten Probleme dar. Tabelle 2 zeigt deutlich, dass es sich bei den meisten Job-Shop-Problemen, bei denen die Gesamtzykluszeit C_{max} minimiert werden soll, um NP-schwere Probleme handelt. Lediglich ein Problem mit 2 Maschinen, einer Bearbeitungszeit von einer Zeiteinheit und unterschiedlichen Auftragsfreigabezeitpunkten ist noch in polynomialer Laufzeit lösbar. Bereits ein Job-Shop-Problem mit 2 Maschinen und der Zielfunktion C_{max} ist NP-schwer.

Tabelle 2: Komplexitätsübersicht von verwandten Problemen des Job-Shop-Problems

Problem	Komplexität	Quelle
$J2\|p_{ij}=1,r_{ij}\|C_{max}$	P	Timkovsky (1997)
$J2\|\|C_{max}$	NP-schwer	Lenstra und Rinnooy Kan (1979)
$J2\|pmnt\|C_{max}$	NP-schwer	Lenstra und Rinnooy Kan (1979)
$J3\|n=3\|C_{max}$	NP-schwer	Sotskov und Shakhlevich (1995)
$J3\|p_{ij}=1\|C_{max}$	NP-schwer	Lenstra und Rinnooy Kan (1979)
$J\|\|C_{max}$	NP-schwer	Lenstra und Rinnooy Kan (1979)

Ein von Fisher und Thompson (1963) veröffentlichtes Benchmarkproblem für $J||C_{max}$ mit 10 Jobs und 10 Maschinen blieb trotz großer wissenschaftlicher Anstrengung 25 Jahre ungelöst (vgl. Schuster, 2003). Die Anzahl möglicher Reihenfolgen (inklusive unzulässiger) des Job-Shop-Problems entspricht $(n!)^m$. Ein Problem mit 20 Maschinen und 10 Jobs besitzt folglich $\approx 7{,}2651 \times 10^{183}$ mögliche Reihenfolgen und verdeutlicht damit, dass das $J||C_{max}$ Problem eines der schwersten Probleme der kombinatorischen Optimierung ist (vgl. Brinkkötter und Brucker, 2001). Das Job-Shop-Problem zählt zu den bestuntersuchten Problemen der Ablaufplanung (vgl. Schuster, 2003). Dies liegt zum einen an der Schwierigkeit des Problems und zum anderen an der Tatsache, dass das Optimierungsproblem die Grundlage für viele komplizierte Probleme aus der Praxis bildet (vgl. Schuster, 2003, Brinkkötter und Brucker, 1999). Zusätzlich ermöglichen Erweiterungen des Job-Shop-Problems die Modellierung realistischer Produktionsvorgänge, weshalb im weiteren Verlauf der Arbeit dem Job-Shop-Problem ($J||C_{max}$) besondere Aufmerksamkeit geschenkt wird (vgl. Henning, 2002).

3 Optimierung und Simulation

In diesem Kapitel werden zunächst exakte und heuristische Lösungsverfahren auf die Eignung zur Lösung von Ablaufplanungsproblemen hin untersucht und ein Verfahren zur Lösung des Job-Shop-Problems wird ausgewählt. Im Anschluss wird die Rolle der Simulation in der Produktion verdeutlicht und die Einsatzgebiete der simulationsbasierten Optimierung werden im Rahmen der Ablaufplanung erläutert. Der Abschluss dieses Kapitels ist ein Überblick über die derzeitigen Methoden der simulationsbasierten Optimierung. Die dabei identifizierten Schwachstellen bilden die Grundlage für die angestrebte Konzeptentwicklung zur Kopplung von Simulation und Optimierung.

3.1 Exakte Optimierung

Reihenfolgeprobleme, die im Rahmen der primalen Ablaufplanung auftreten, sind Bestandteil der Kombinatorik (vgl. Littger, 1992). Sollen diese Reihenfolgeprobleme unter Berücksichtigung einer Zielfunktion gelöst werden, liegt eine kombinatorische Optimierungsaufgabe vor (vgl. Benker, 2003). Die Lösungsmenge des kombinatorischen Optimierungsproblems besteht aus einer endlichen Menge an Permutationen aus einer endlichen Menge an Elementen (vgl. Benker, 2003), sodass enumerative Verfahren zur Lösungsfindung eingesetzt werden (vgl. Garfinkel und Nemhauser, 1972).

Exakte Verfahren beruhen auf dem Prinzip der Enumeration und können die optimale Lösung eines Optimierungsproblems in endlich vielen Schritten finden und verifizieren (vgl. Rieck, 2009) oder stellen fest, dass das Problem nicht lösbar ist (vgl. Acker, 2011). Der Lösungsalgorithmus kann allerdings in Abhängigkeit von der Komplexität des vorliegenden Problems erhebliche Laufzeiten benötigen (vgl. Schwartz, 2004). Erst der kontinuierliche Anstieg der Rechnerleistung ermöglicht überhaupt die Anwendung von exakten Optimierungsverfahren für das Job-Shop-Problem (vgl. Schuster, 2003). Die Lösung von Ablaufplanungsproblemen mit exakten Optimierungsmethoden erfolgt hauptsächlich durch die Anwendung der gemischt-ganzzahligen Programmierung und dem „Branch & Bound“-Verfahren (vgl. Troßmann, 1996). Viele Lösungsverfahren für Ablaufplanungsprobleme, so auch die genannten exakten Verfahren, basieren auf dem Prinzip des von Roy und Sussman (1964) eingeführten disjunktiven Graphen (vgl. Schuster, 2003).

3.1.1 Disjunktiver Graph

Mit Hilfe des disjunktiven Graphen können Shop-Probleme und folglich auch das Job-Shop-Problem graphisch modelliert werden (vgl. Klemmt, 2012). Der disjunktive Graph repräsentiert bei einer regulären Zielfunktion alle zulässigen Ablaufpläne des Shop-Problems und enthält somit auch die optimale Lösung des Optimierungsproblems (vgl.

Brucker, 2004). Das Vorhandensein der optimalen Lösung innerhalb des Graphen erklärt, weshalb der disjunktive Graph als Basis für exakte Optimierungsverfahren dient.

Der disjunktive Graph $G = (V, C, D)$ ist dabei wie folgt definiert (vgl. Brucker, 2004):

V repräsentiert die Menge aller Knoten des Graphen. Die Anzahl der Knoten setzt sich aus der Menge aller Arbeitsschritte und zwei fiktiven Knoten, einer Quelle $0 \in V$ sowie einer Senke $* \in V$, zusammen. Die Knoten des Graphen sind gewichtet, wobei das Gewicht der Quelle und Senke 0 ist. Die Knoten der Arbeitsschritte sind mit den spezifischen Prozesszeiten p_{ij} gewichtet.

C ist die Menge der gerichteten konjunktiven Kanten. Diese Kanten repräsentieren die Reihenfolgebeziehungen zwischen den einzelnen Operationen. $(i, j) \in C$ bedeutet, dass innerhalb eines Jobs der Vorgang j erst startet, nachdem der Vorgang i abgeschlossen ist. Es existieren auch konjunktive Kanten zwischen der Quelle und allen Operationen, die keinen Vorgänger haben, sowie zwischen der Senke und allen Operationen, die keinen Nachfolger haben. Ein Graph $R = (V, C)$ wird folglich auch als Reihenfolgegraph bezeichnet (vgl. Schuster, 2003).

D ist die Menge der ungerichteten disjunktiven Kanten. Sie verbinden Arbeitsschritte (Knoten) unterschiedlicher Jobs, die auf der gleichen Maschine gefertigt werden müssen. Für eine disjunktive Kante $\{i, j\} \in D$ besteht die Wahlmöglichkeit zwischen der Kante (i, j) und (j, i) (vgl. Schuster, 2003). Disjunktive Kanten repräsentieren somit die Kapazitätsbeschränkungen der Maschinen (vgl. Henning, 2002).

Das Ziel ist nun, alle disjunktiven Kanten zu fixieren, d.h. in gerichtete konjunktive Kanten zu überführen. Eine Selektion S bezeichnet dabei eine Menge von konjunktiven Kanten und ist vollständig, wenn

- jede disjunktive Kante fixiert wurde und
- der resultierende Graph $G(S) = (V, C \cup S)$ azyklisch ist.

Aus jeder vollständigen Selektion kann ein zulässiger Ablaufplan erzeugt werden (vgl. Klemmt, 2012). Bezugnehmend auf das Job-Shop-Problem kann das Optimierungsproblem optimal gelöst werden, wenn „der längste Weg von der Quelle zur Senke (sog. kritischer Pfad) minimal ist“ (Henning, 2002, S.4).

Folgendes Beispiel soll den Aufbau des disjunktiven Graphen verdeutlichen. Es existieren drei Jobs (J_1, J_2, J_3), die auf drei Maschinen (M_1, M_2, M_3) bearbeitet werden müssen. Tabelle 3 stellt die Jobs mit den Arbeitsschritten (AS), den zugehörigen Maschinen M_j sowie den Bearbeitungszeiten p_j dar.

Tabelle 3: Arbeitsschritte, Maschinen und Bearbeitungszeiten

	J_1			J_2			J_3		
AS	1	2	3	4	5	6	7	8	9
M_j	M_1	M_3	M_2	M_2	M_1	M_3	M_1	M_2	M_3
p_j	5	6	4	5	5	4	5	4	5

Abbildung 5 illustriert den zu Tabelle 3 zugehörigen disjunktiven Graphen. Das Ziel ist nun, alle disjunktiven Kanten zu fixieren und so die Ressourcenkonflikte zu lösen.

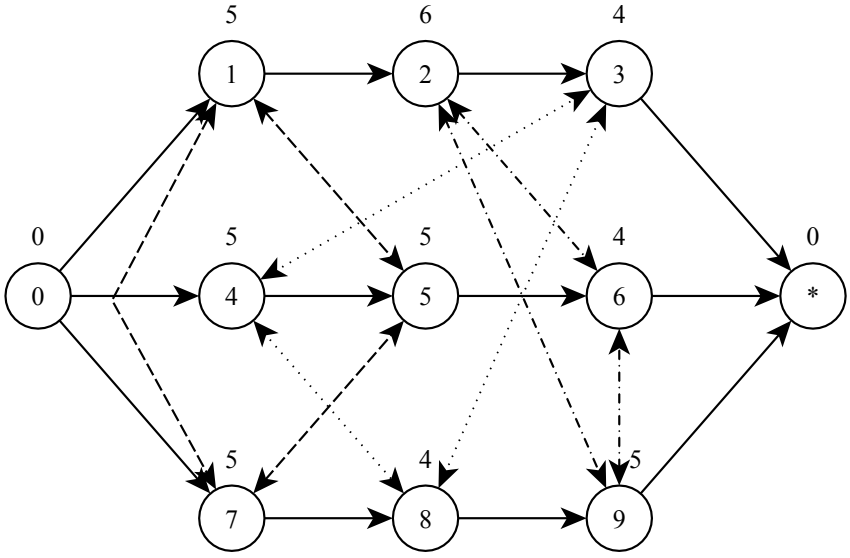

Abbildung 5: Disjunktiver Graph (nach Wennink, 1995)

Das Ergebnis ist eine, wie in Abbildung 6 dargestellte, vollständige Selektion.

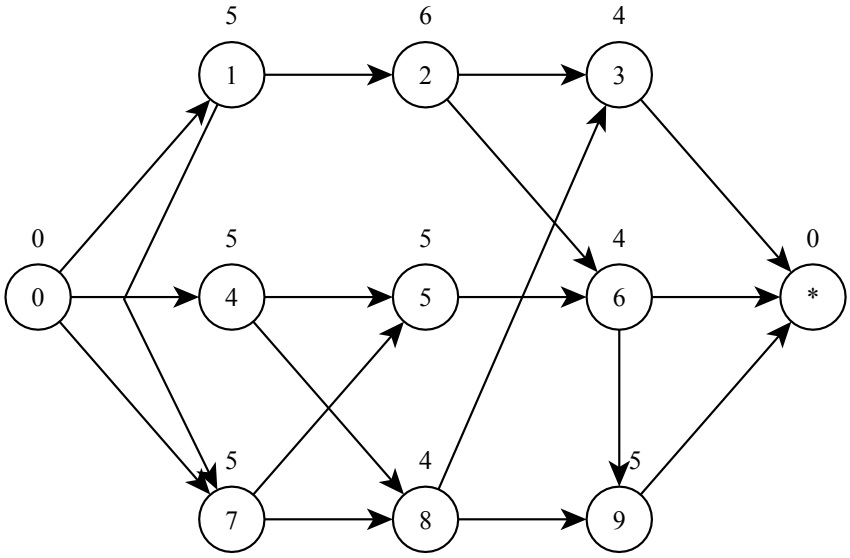

Abbildung 6: Vollständige Selektion

Wie in dem Gantt-Diagramm (s. Abbildung 7) zu erkennen ist, bilden die Operationen (1,7,5,6,9) den kritischen Pfad in dieser Selektion (s. Abbildung 6) und liefern eine Gesamtzykluszeit von 24 ZE. Optimierungsverfahren, die auf dem disjunktiven Graphen beruhen, versuchen nun durch die Erzeugung von verschiedenen Selektionen einen Ablaufplan zu finden, bei dem der kritische Pfad minimal ist.

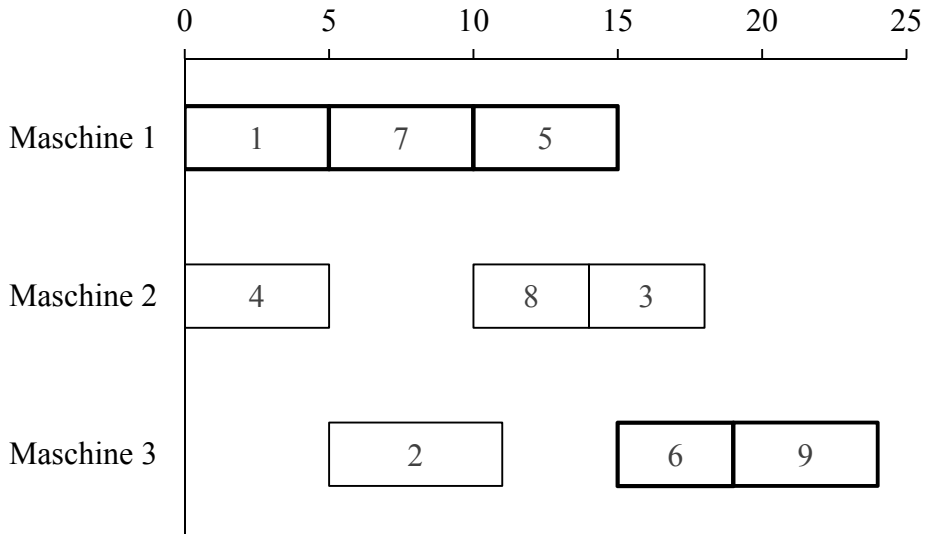

Abbildung 7: Gantt-Diagramm (Wennink, 1995)

3.1.2 Gemischt-ganzzahliges Optimierungsmodell

Ablaufplanungsprobleme lassen sich nicht nur graphisch darstellen, sondern können auch im Rahmen der exakten Optimierung als mathematische Programme formuliert werden (vgl. Domschke und Drexl, 2005). Rinnooy Kan, A. H. G. (1976) sieht in diesen mathematischen Programmen sogar eine effektive Lösungsmöglichkeit von Ablaufplanungsproblemen. Eine in diesem Zusammenhang angewandte Methodik zur Lösung von Ablaufplanungsproblemen ist die gemischt-ganzzahlige Programmierung (engl. *mixed-integer programming*) (vgl. Kallrath, 2013). Die gemischt-ganzzahlige Optimierung beruht auf dem Prinzip der vollständigen Enumeration, d.h. dass alle möglichen Ablaufpläne generiert werden, um die optimale Lösung zu finden (vgl. Kallrath, 2013). Im Allgemeinen liegt ein solches Modell in folgender Form vor (vgl. Domschke et al., 1997; Domschke und Drexl, 2005; Unger, 2007):

Minimiere oder maximiere:

$$z(c) \tag{3-1}$$

unter den Nebenbedingungen:

$$A_k c \leq b \tag{3-2}$$

$$c \in W_1 \times \ldots \times W_r, \text{ mit } W_j \in \{\mathbb{R}^+, \mathbb{Z}^+, \mathbb{B}\} \text{ für } j = 1, \ldots, r \quad (3\text{-}3)$$

Dabei haben die verwendeten Symbole folgende Bedeutung:

$z(c)$	Zielfunktion in Abhängigkeit von $(c_1, \ldots, c_r)$
c	Variablenvektor mit r Komponenten $(c_1, \ldots, c_r)$
b	Grenzvektor mit q Komponenten $(b_1, \ldots, b_q)$
$A_k = (a_{ij})_{i=1,j=1}^{q,r}$	Koeffizientenmatrix mit q Zeilen und r Spalten
W	Wertebereich der Variablen $(c_1, \ldots, c_r)$
$c_j \in \mathbb{R}^+$	Kontinuierliche positive Variable
$c_j \in \mathbb{Z}^+$	Ganzzahlige positive Variable
$c_j \in \mathbb{B}$	Binäre Variable

(3-1) beschreibt die Zielfunktion, die minimiert oder maximiert werden soll. Im Fall des Job-Shop-Problems soll durch die geeignete Wahl der Variablen die Zielfunktion C_{max} minimiert werden. Für weitere relevante Zielfunktionen der Ablaufplanung s. Kapitel 2.3.3.

(3-2) ist ein lineares Nebenbedingungssystem, welches aus Gleichungen und Ungleichungen bestehen kann.

(3-3) legt den Wertebereich der Entscheidungsvariablen fest. Die Variablen können einen reellen, ganzzahligen oder binären Wert annehmen. Die binären Werte eignen sich besonders, um Reihenfolgebeziehungen innerhalb eines Ablaufplanungsproblems abzubilden (vgl. Seelbach, 1993).

Die optimale Lösung eines Ablaufplanungsproblems mit Hilfe der gemischt-ganzzahligen Programmierung entspricht folglich einem Vektor c, der die Bedingungen (3-2) und (3-3) erfüllt und für den die Zielfunktion (3-1) ein globales Maximum oder Minimum annimmt (vgl. Domschke et al., 1997). Ein sehr bekanntes und oftmals angewendetes gemischt-ganzzahliges Programm zur Lösung des Job-Shop-Problems ist das Modell nach Manne (1960), dessen Funktionsweise im Folgenden erläutert werden soll.

3.1.3 Job-Shop-Modellierung nach Manne

Das Modell nach Manne (1960) ist laut Brüggemann (1995) eine effiziente Möglichkeit, um das Job-Shop-Problem zu modellieren. Klemmt (2012) bestätigt das mit seinen Benchmarktests und sieht in dem Modell nach Manne (1960) ebenfalls Potential zur Lösung des Job-Shop-Problems. Das Job-Shop-Problem kann nach Manne (1960) wie folgt modelliert werden (vgl. Brucker und Knust, 2006; Manne, 1960):

Minimierung der Zielfunktion:

$$z(x) = C_{max} = S_* \tag{3-4}$$

Mit den Nebenbedingungen:

$$S_i + p_i \leq S_* \text{ für } i \rightarrow * \in C \tag{3-5}$$

$$S_i + p_i \leq S_j \text{ für } i \rightarrow j \in C \tag{3-6}$$

$$S_i + p_i - K(1 - y_{ij}) \leq S_j \text{ für } (i - j \in D) \tag{3-7}$$

$$S_j + p_j - K y_{ij} \leq S_i \text{ für } (i - j \in D) \tag{3-8}$$

$$S_i \geq 0 \text{ für } i = (0, \dots, *) \tag{3-9}$$

$$y_{ij} \in \{0,1\} \text{ für } (i - j \in D) \tag{3-10}$$

Die Zielfunktion des Job-Shop-Problems, die Minimierung der Gesamtzykluszeit, wird durch (3-4) ausgedrückt. S_* beschreibt dabei den imaginären Startzeitpunkt der Senke des disjunktiven Graphen und repräsentiert somit den Endzeitpunkt der letzten Bearbeitung. Die Bedingung (3-5) beschreibt die letzte Operation des kritischen Pfades von $i \rightarrow *$ und verbindet somit die Zielfunktion mit dem Restriktionensystem (3-6) – (3-8). (3-6) entspricht der individuellen Maschinenfolge der Aufträge und stellt sicher, dass Operation j erst starten kann, wenn Operation i beendet ist. Konkurrieren zwei Aufträge i und j um die gleiche Maschine, dargestellt durch eine disjunktive Kante im Ablaufgraphen, kann entweder i vor j (3-7) oder j vor i (3-8) bearbeitet werden. Die hinreichend große Variable K stellt dabei sicher, dass nur eine der Bedingungen begrenzend wirkt. Manne (1960) führt für diesen Ressourcenkonflikt die Binärvariable y_{ij} ein (3-11).

$$y_{ij} = \begin{cases} 1 & \text{falls } i \text{ vor } j \\ 0 & \text{sonst} \end{cases} \tag{3-11}$$

Der zulässige Definitionsbereich des Manne-Modells ist mit (3-9) und (3-10) angegeben.

Das Modell von Manne (1960) eignet sich allerdings nur zur exakten Lösung von Job-Shop-Problemen mit maximal 20 Jobs und 20 Maschinen (vgl. Klemmt, 2012). Im Allgemeinen eignen sich gemischt-ganzzahlige Programme nur für kleine oder spezielle Probleminstanzen (vgl. Preßmar, 2002). Der exponentielle Anstieg des Rechenaufwands, bedingt durch die Zunahme der Binärvariablen, macht das Lösen von Problemen in praxisrelevanten Größen nicht möglich (vgl. Schwartz, 2004; Kurbel und Rohmann, 1995).

3.1.4 „Branch & Bound“-Verfahren

Eine weitere exakte Lösungsmethode ist das „Branch & Bound“-Verfahren. Im Gegensatz zur gemischt-ganzzahligen Programmierung ist das ein Verfahren der impliziten Enumeration (vgl. Kallrath, 2013). Im Rahmen der impliziten Enumeration werden

zunächst alle Lösungen betrachtet, aber während des Lösungsprozesses schrittweise die Lösungsalternativen ausgeschlossen, die für die optimale Lösung nicht mehr in Frage kommen (vgl. Siegel, 1974). Die Idee des „Branch & Bound"-Verfahrens ist es somit, ein Ablaufplanungsproblem schrittweise in kleine, leichter lösbare Unterprobleme zu zerlegen, wobei die beste Lösung aller Unterprobleme das globale Optimum darstellt (vgl. Klemmt, 2012; Achterberg, 2007).

Die allgemeine Vorgehensweise eines „Branch & Bound"-Verfahren kann mit Algorithmus 1 skizziert werden (vgl. Achterberg, 2007).

Algorithmus 1: Branch & Bound

(1): Initialisiere $L \leftarrow \{R_a\}, \hat{c} \leftarrow \infty$
(2): Teile R_a *in die Unterprobleme* $R_a = Q_1 \cup \ldots \cup Q_k$ *und* $L \leftarrow L \cup \{Q_1, \ldots, Q_k\}$
(3): ***Wenn*** $L = \emptyset$, ***dann***
(4): *Stopp: Gebe* $\hat{x}$ *und* $\hat{c}$ *aus.*
(5): ***Sonst***
(6): *Wähle* $Q_k \in L$ *und* $L \leftarrow L \backslash \{Q\}$
(7): *Bilde und löse die Relaxation* Q_{relax} *von* Q_k.
(8): ***Wenn*** $Q_{relax} \neq \emptyset$, ***dann***
(9): $\check{c} \leftarrow z(\check{x})$
(10): ***Wenn*** $\check{c} \geq \hat{c}$, ***dann*** *gehe zu (3)*
(11): ***Sonst***
(12): ***Wenn*** $\check{x}$ *zulässig für* R_a ***dann***
(13): $\hat{x} \leftarrow \check{x}$ *und* $\hat{c} \leftarrow \check{c}$, *gehe zu (3)*
(14): ***Sonst***
(15): $R_a \leftarrow Q_k$, *gehe zu (2)*
(16): ***Ende Wenn***
(17): ***Ende Wenn***
(18): ***Ende Wenn***
(19): *gehe zu (3)*
(20): ***Ende Wenn***

Der „Branch & Bound"-Algorithmus startet mit der Initialisierung der Menge *L*, die alle ungelösten Probleme enthält, und der aktuell besten oberen Schranke $\hat{c}$ (1). Das Ausgangsproblem R_a wird in einfacher zu lösende Unterprobleme geteilt (*engl. branching*) und der Menge *L* hinzugefügt (2). Dadurch entsteht ein Suchbaum (s. Abbildung 8), dessen Wurzel R_a das Ausgangsproblem darstellt (vgl. Achterberg, 2007).

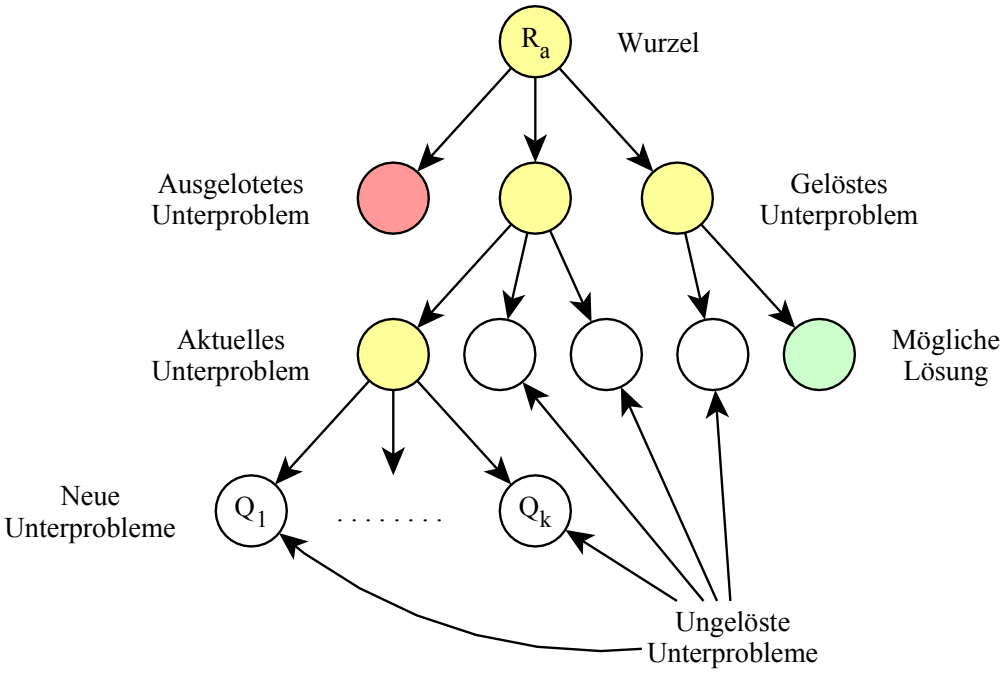

Abbildung 8: Suchbaum (Achterberg, 2007)

Jeder dieser Knoten entspricht dabei einem disjunktiven Graphen und enthält somit alle Lösungen des Teilproblems (vgl. Brucker, 2004). Aus L wird ein Problem Q_k ausgesucht (6) und dessen Relaxation Q_{relax} gebildet und gelöst (7). Existiert keine zulässige Lösung für Q_{relax}, wird das Problem nicht weiter betrachtet und ein neues Problem aus L wird ausgewählt (6), vorausgesetzt (3) ist nicht erfüllt. Ansonsten (8) stellt $\check{x}$ die optimale Lösung für Q_{relax} und $\check{c}$ den dazugehörigen Zielfunktionswert dar (9). Ist die untere Schranke $\check{c}$ größer als die aktuell beste obere Schranke $\hat{c}$, wird Q_k nicht weiter betrachtet (10), da es die optimale Lösung von R_a niemals enthalten kann (*engl. bounding*). Auf diese Weise wird die vollständige Enumeration aller möglichen Lösungen von R_a verhindert. Verbessert die Lösung $\check{x}$ von Q_{relax} die aktuell beste obere Schranke (11) und ist eine zulässige Lösung für R_a (12), wird sowohl die aktuell beste Lösung als auch die aktuell beste obere Schranke ersetzt (13). Ansonsten (14) wird das aktuell betrachtete Problem in weitere kleinere Teilprobleme verzweigt (15). Ein Problem wird solange verzweigt, bis das (Teil-)Problem ausgelotet ist (vgl. Domschke et al., 1997). Ein Problem wird als ausgelotet bezeichnet, wenn es nicht mehr weiter betrachtet werden muss, da einer der sich gegenseitig ausschließenden Fälle eingetreten ist: (vgl. Domschke und Drexl, 2005):

1. Für Q_{relax} existiert keine zulässige Lösung.
2. Die beste Lösung eines Unterproblems ist schlechter als die aktuell beste Lösung.
3. Eine neue verbessernde Lösung ist gefunden und zulässig für Q_{relax} und R_a.

Das „Branch & Bound“-Verfahren ist beendet, sobald L keine ungelösten Probleme mehr enthält (3). Dann wird die aktuell beste Lösung sowie die aktuell beste obere Schranke ausgegeben (4).

Zur Lösung des Job-Shop-Problems existieren vielen unterschiedliche „Branch & Bound“-Verfahren, die auf dem dargestellten Funktionsprinzip basieren. Die heutzutage besten „Branch & Bound“-Verfahren zur Lösung des Job-Shop-Problems stammen von Brucker et al. (1994), Martin und Shmoys (1996), Applegate und Cook (1991) sowie Caseau und Laburthe (1996) und können (10×10) Probleminstanzen problemlos lösen (vgl. Schuster, 2003). Bei praxisrelevanten Problemgrößen stoßen aber auch die „Branch & Bound“-Methoden schnell an ihre Grenzen (vgl. Jain und Meeran, 1999). Brinkkötter und Brucker (1999) demonstrierten das mit dem Versuch, ungelöste Benchmarkprobleme zu lösen, wobei sie die Berechnung nach mehreren Tagen abbrechen mussten, weil das Optimum bis dahin nicht gefunden wurde. Die Grenzen solcher „Branch & Bound“-Verfahren liegen bei Problemgrößen von (15×15) bis maximal (20×15), wobei das kleinste offene Benchmarkproblem (20×10) groß ist (vgl. Henning, 2002).

Für die Lösung des Job-Shop-Problems in praxisrelevanten Größen kommen exakte Verfahren somit nicht in Frage, weshalb heuristische Methoden eine weit verbreitete Anwendung finden (vgl. Andersen et al., 1997). Ist ein Problem allerdings der Klasse P zugeordnet, sollte zur optimalen Lösung ein exaktes Verfahren angewendet werden (vgl. Acker, 2011). Trotz der eingeschränkten Nutzbarkeit von exakten Verfahren zur Lösung des Job-Shop-Problems spielen die gemischt-ganzzahlige Programmierung und das „Branch & Bound“-Verfahren eine wichtige Rolle, denn mit ihnen kann an kleinen Probleminstanzen die Lösungsqualität heuristischer Verfahren festgestellt werden (vgl. Rieck, 2009).

3.2 Heuristische Verfahren

Die optimale Lösung von Ablaufplanungsproblemen mittels exakter Verfahren ist wie in vorherigem Kapitel erörtert nicht bzw. nur eingeschränkt für kleine bzw. Probleme der Klasse P möglich, sodass praxisrelevante NP-schwere Probleme nur mittels heuristischer Verfahren näherungsweise gelöst werden können (vgl. Kistner und Steven, 2001). Heuristische Lösungsverfahren sollten demnach nur zum Einsatz kommen, wenn keine effizienten Algorithmen existieren, die das vorliegende Problem lösen können (vgl. Brucker, 2004; Klemmt, 2012).

3.2.1 Definition Heuristik

In der Literatur existiert eine Vielzahl an unterschiedlichen Definitionen für heuristische Lösungsverfahren bzw. Heuristiken, dennoch erkennt Streim (1975) drei Merkmale, die

in allen Definitionen enthalten sind. Demnach bezeichnet man eine Lösungsstrategie als heuristisch, wenn sie alle der drei folgenden Punkte erfüllt (vgl. Streim, 1975):

1. Nichtwillkürliche Entscheidungsoperatoren kommen zum Einsatz
2. Potentielle Lösungen werden vom Suchprozess ausgeschlossen
3. Es existiert keine Garantie, dass das Problem optimal gelöst wird

Heuristische Lösungsverfahren sind demnach Methoden, die unter Zuhilfenahme von problemabhängigen Suchregeln nach zulässigen, hinreichend guten Lösungen in vertretbarer Rechenzeit suchen, die gefundenen Lösungen allerdings nicht dem globalen Optimum entsprechen müssen (vgl. Zanakis et al., 1989). Im Vergleich zu exakten Lösungsverfahren kann bei einer Lösung, die durch eine Heuristik bestimmt wurde, die Güte der Lösung nicht beurteilt und es können auch keine Aussagen getroffen werden, wie weit diese Lösung vom Optimum entfernt ist (vgl. Klemmt, 2012; Acker, 2011).

Heuristische Lösungsverfahren lassen sich in unterschiedliche Gruppen einteilen (vgl. Domschke et al., 1997):

1. Unvollständig ausgeführte exakte Verfahren
2. Relaxationsbasierte Verfahren
3. Eröffnungsverfahren
4. Verbesserungsverfahren
5. Kombination aus 1.–4.; üblich ist Kombination von 3. und 4.

Neben der Einteilung in obenstehende Gruppen können heuristische Methoden auch in deterministische und stochastische Verfahren eingeteilt werden (vgl. Domschke et al., 1997). Deterministische Lösungsverfahren liefern bei wiederholter Anwendung auf das gleiche Problem und den gleichen Anfangsbedingungen stets dieselbe Lösung. Stochastische Heuristiken hingegen liefern bei mehrfacher Anwendung auf ein Ablaufplanungsproblem aufgrund einer zufälligen Komponente innerhalb des Algorithmus unterschiedliche Ergebnisse trotz identischer Startbedingungen (vgl. Domschke et al., 1997).

3.2.2 Unvollständig ausgeführte exakte Verfahren

Exakte Verfahren, wie der „Branch & Bound“-Algorithmus, werden beendet, bevor die optimale Lösung gefunden wurde (vgl. Domschke und Scholl, 2006). Dies geschieht z.B., wenn eine bestimmte Lösungsgüte erreicht ist (vgl. Domschke et al., 1997).

3.2.3 Relaxationsbasierte Verfahren

Im Rahmen von Relaxationsverfahren werden die Nebenbedingungen und Restriktionen eines Problems gelockert, sodass durch die Lösung des weniger restringierten Problems

eine gute zulässige Lösung für das Ausgangsproblem gefunden werden kann (vgl. Domschke et al., 1997; Domschke und Scholl, 2006).

3.2.4 Eröffnungsverfahren

Eröffnungsverfahren ermitteln für ein zu lösendes Problem eine einzige Lösung, ohne weitere Lösungen während des gesamten Lösungsprozesses zu generieren (vgl. Müller-Merbach, 1970). Die Gesamtlösung wird dabei durch die sukzessive Vervollständigung von Teillösungen konstruiert (vgl. Zäpfel und Braune, 2005). Die ermittelte zulässige Lösung von Eröffnungsverfahren kann als Lösung des Problems, aber auch als Startlösung für Verbesserungsverfahren dienen (vgl. Müller-Merbach, 1970).

Das Prioritätsregelverfahren ist ein weitverbreitetes Eröffnungsverfahren und wird in der Praxis sehr häufig für die Lösung des Job-Shop-Problems angewandt (vgl. Acker, 2011; Van Dyke Parunak, H., 1991; Pinedo, 2012). Prioritätsregelverfahren basieren nach Domschke et al. (1997) auf zwei Schritten:

1. Bestimme den Rangwert für ein Objekt anhand einer Prioritätsregel
2. Bestimme die Auftragsreihenfolge auf Basis der Rangwerte

In Fertigungssystemen können mehrere Jobs zum gleichen Zeitpunkt um eine Maschine konkurrieren. Dieser Sachverhalt wird in einem disjunktiven Graphen durch ungerichtete disjunktive Kanten dargestellt (s. Kapitel 3.1.1). Im Rahmen der Ablaufplanung muss nun eine Auftragsfolge für die Maschine gefunden werden, damit dieser Ressourcenkonflikt gelöst werden kann. Dies geschieht durch den Einsatz sogenannter Prioritätsregeln, die den einzelnen Jobs Rangwerte, bezogen auf die Maschine, um die sie konkurrieren, zuweisen (Schritt 1.). Auf Basis dieser Rangwerte werden im 2. Schritt die Jobs aus der Konfliktmenge in der entsprechenden Reihenfolge schrittweise der Maschine zugeordnet. Mit Prioritätsregelverfahren können für einfache Probleme, wie Ein-Maschinenprobleme oder Flow-Shop Probleme, optimale Lösungen gefunden werden (vgl. Schwartz, 2004). Für komplexe Problemstellungen liefern solche Verfahren allerdings nur Lösungen mit starken Abweichungen vom Optimum (vgl. Schwartz, 2004). Dies liegt an der sukzessiven Vorgehensweise der Prioritätsregelverfahren, sodass im Verlauf des Verfahrens immer weniger konkurrierende Jobs zur Einplanung zur Verfügung stehen (vgl. Domschke et al., 1997). Dadurch nimmt die Wahrscheinlichkeit, eine ungünstige Entscheidung bezüglich der Zielfunktion getroffen zu haben, zu (vgl. Domschke et al., 1997). Teich (1998) untersuchte die Lösungsqualität von Prioritätsregelverfahren an Problemen, bei denen die Gesamtzykluszeit minimiert werden sollte, und stellte fest, dass jede Lösung mindestens 20 % vom Optimum entfernt war. Stellt die Lösung eines Prioritätsregelverfahrens eine Startlösung für Verbesserungsverfahren dar, so sind die starken Abweichungen weniger bedeutend (vgl. Schwartz, 2004).

Zur Lösung von komplexen Problemen, wie dem Job-Shop-Problem, führt die reine Anwendung von Prioritätsregeln zu deutlichen und unnötigen Verzögerungen im Produktionsablauf (vgl. Jaehn und Pesch, 2014). Um das Prioritätsregelverfahren auch für komplexe Problemstrukturen anwenden zu können, werden die Prioritätsregeln in spezielle Eröffnungsverfahren integriert (vgl. Baker, 1974). Der von Giffler und Thompson (1960) entwickelte Algorithmus (GT-Algorithmus) wird in diesem Zusammenhang oftmals zur Erstellung von Ablaufplänen verwendet und bildet die Basis für Heuristiken, die auf der Anwendung von Prioritätsregeln beruhen (vgl. Błażewicz et al., 1996). Auf eine genaue Erläuterung des GT-Algorithmus soll an dieser Stelle verzichtet werden.

Tabelle 4: Exemplarische Auswahl von Prioritätsregeln

Name	Kürzel	Auswahl abhängig von…	Höchste Priorität für Auftrag…
Zufallsregel	RANDOM	Zufall	… mit dem geringsten Wert einer gleichverteilten Zufallszahl
Operationszeit-regel	SPT/LPT	Prozesszeiten	… mit der kürzesten/längsten Prozesszeit an betrachteter Maschine
Liefertermin-regel	EDD	Fertigstellungsterminen	… mit dem frühsten Liefertermin
Wertregel	VALUE	Werten	… mit dem höchsten Produktwert

Tabelle 4 zeigt eine exemplarische Auswahl an Prioritätsregeln, die häufig in der Praxis eingesetzt werden (vgl. Brah und Wheeler, 1998; Hansmann, 2001; Zäpfel, 1982). Diese Regeln wurden von den verschiedenen Zielfunktionen der Ablaufplanungsprobleme abgeleitet (z.B. EDD von terminorientierten Zielfunktionen, SPT/LPT von durchlaufzeitbezogenen Zielfunktionen) und dienen somit zur Unterstützung der Zielerreichung im Rahmen des Lösungsprozesses (vgl. Hoitsch, 1993).

3.2.5 Verbesserungsverfahren

Verbesserungsverfahren arbeiten nach dem Prinzip „find a feasible solution and then search for better ones“ (vgl. Hillier, 1969, S. 603). Sie benötigen zum Start demnach eine bereits zulässige Lösung für das vorliegende Ablaufplanungsproblem, die zufällig erzeugt oder durch ein Eröffnungsverfahren, wie dem Prioritätsregelverfahren, bereitgestellt wurde (vgl. Domschke et al., 1997). Sie arbeiten dann wie folgt:

Von dieser zulässigen Lösung x ausgehend suchen Verbesserungsverfahren iterativ den Lösungsraum nach einer Verbesserungsmöglichkeit ab. Im Gegensatz zu Verfahren der vollständigen Enumeration (s. Kapitel 3.1.3) beschränken sie sich bei der Lösungssuche

nicht auf den gesamten Lösungsraum X, sondern nur auf einen Teil des Lösungsraumes, die sogenannte Nachbarschaft $N(x) \subset X$ (vgl. Dowsland, 1993). Die Nachbarschaft $N(x)$ definiert somit den Teil des gesamten Lösungsraumes, der von x aus mit einfachen Transformationen zu erreichen ist, und wird durch eine Nachbarschaftsfunktion, die die Transformationsregeln festlegt, bestimmt (vgl. Acker, 2011; Zäpfel und Braune, 2005). Eine Transformation der Lösung ergibt sich dabei aus (vgl. Domschke und Scholl, 2006):

1. der Veränderung der Lösung an einer Stelle
2. dem Vertauschen von Elementen
3. dem Verschieben von Elementen

Eine Lösung $x' \in N(x)$ gilt somit als benachbart, wenn sie in einem einzigen Schritt im Rahmen der Transformationsvorschriften zu erreichen ist (vgl. Reeves und Beasley, 1993). Die Überführung einer Lösung x zu einer Lösung x' wird auch als Zug v bezeichnet (vgl. Domschke und Drexl, 2005). Die Gestalt der Nachbarschaft ist ausschlaggebend für die Erreichbarkeit der Lösungen im Lösungsraum und hat somit maßgeblichen Einfluss auf die Effizienz und Effektivität des Suchverfahrens (vgl. Glover und Laguna, 1993). Unterscheiden sich die Lösungen einer Nachbarschaft nur minimal von der betrachteten Lösung, hat dies eine detaillierte Lösungssuche zur Folge, was wiederum in einer langsamen Annäherung an das Optimum resultiert (vgl. Schwartz, 2004). Sind die Unterschiede hingegen größer, wird der Lösungsraum zwar schneller durchsucht, es besteht aber auch die Gefahr, dass das Optimum übersehen wird (vgl. Dueck et al., 1993). Das Prinzip der Nachbarschaftssuche sowie die Zusammenhänge zwischen den akzeptierten Lösungen und den Nachbarschaften ist in Abbildung 9 dargestellt (vgl. Zäpfel und Braune, 2005).

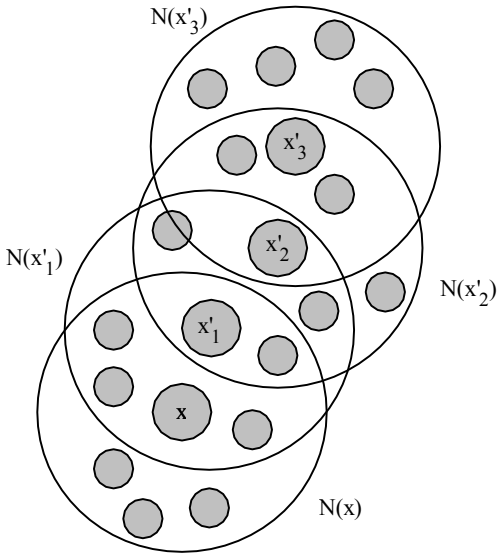

Abbildung 9: Prinzip der Nachbarschaftssuche (Zäpfel und Braune, 2005)

Um das Job-Shop-Problem mit Verbesserungsverfahren zu lösen, werden verschiedene Nachbarschaften vorgeschlagen. Exemplarisch soll an dieser Stelle die Nachbarschaft N1 erwähnt werden, die auf der Umkehrung eines disjunktiven Pfeiles beruht (vgl. Geiger, 2005). Im Rahmen der Anwendung von Verbesserungsverfahren zur Lösung von Ablaufplanungsproblemen ist nicht nur die Struktur der Nachbarschaft zu bestimmen, sondern auch eine Auswahlstrategie, auf deren Basis eine untersuchte Nachbarschaftslösung akzeptiert wird, um von dort eine neue Iteration zu starten. Anhand der Auswahlstrategie können Verbesserungsverfahren in zwei Kategorien eingeteilt werden (vgl. Domschke et al., 1997):

1. **Reine Verbesserungsverfahren**

Bei reinen Verbesserungsverfahren wird die Nachbarschaftslösung nur akzeptiert, wenn ihr Zielfunktionswert im Vergleich zur aktuellen Lösung größer bei Maximierungsproblemen und kleiner bei Minimierungsproblemen ist. Dabei kann die erste bessere Lösung gewählt werden (*engl. first fit*) oder nach Durchsuchen der gesamten Nachbarschaft die Lösung, die zur größtmöglichen Verbesserung des Zielfunktionswertes führt (*engl. best fit*). Wird keine verbessernde Nachbarlösung gefunden, endet das Verfahren. Die gefundene Lösung kann zwar für die Nachbarschaftsdefinition ein lokales Optimum darstellen, kann aber weit von dem globalen Optimum entfernt sein (vgl. Brucker, 2004).

Abbildung 10 verdeutlicht diesen Sachverhalt anhand der in Abhängigkeit von x zu minimierenden Funktion $z(x)$ und zeigt, dass die Nachbarschaftslösung x'_1 das lokale, aber nicht das globale Minimum darstellt. Da reine Verbesserungsverfahren innerhalb einer Iteration nur Lösungen zulassen, die den Zielfunktionswert minimieren, kann dieses lokale Optimum folglich nie verlassen werden.

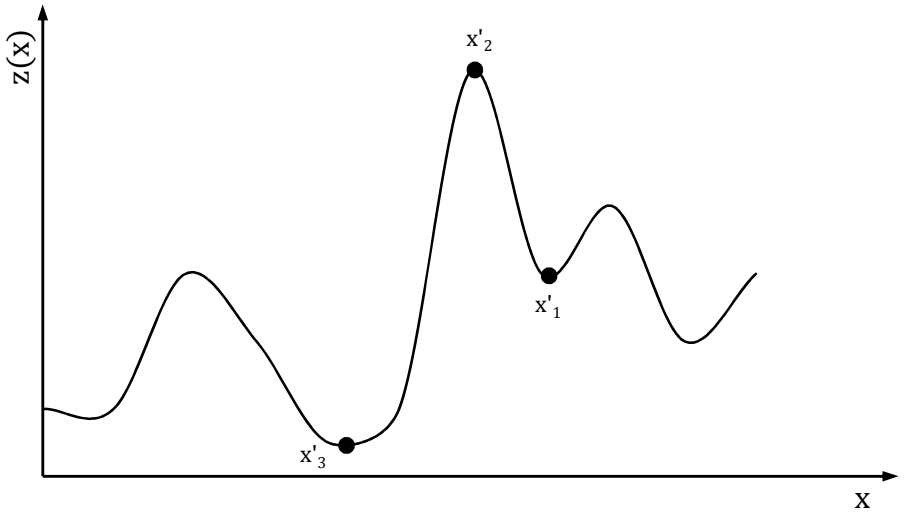

Abbildung 10: Lösungslandschaft (nach Domschke et al., 1997)

2. Lokale Suchverfahren

Lösungsverfahren, die Nachbarschaftslösungen akzeptieren, deren Zielfunktionswert schlechter ist als der der aktuellen Lösung, werden als lokale Suchverfahren bezeichnet (vgl. Domschke und Scholl, 2006). Durch die Akzeptanz von schlechteren Lösungen können lokale Optima verlassen werden. Damit in Abbildung 10 das globale Optimum x'_3 erreicht werden kann, muss in der nächsten Iteration eine Verschlechterung (x'_2) in Kauf genommen werden. Auch bei lokalen Suchverfahren existiert die First- und Best-Fit-Strategie, diese gilt es aber für den Fall von Zielfunktionsverschlechterungen zu adaptieren. Die First-Fit-Strategie erlaubt es, einen schlechteren Zielfunktionswert zufällig, in Abhängigkeit des Ausmaßes der Verschlechterung oder bei Mangel einer verbessernden Lösung zu wählen (vgl. Domschke et al., 1997). Im Rahmen der Best-Fit-Strategie wird die bestmögliche Lösung oder – für den Fall der Nichtexistenz einer verbessernden Lösung – die Lösung mit der kleinsten Verschlechterung der gesamten Nachbarschaft akzeptiert (*engl. steepest descent* bei Minimierungsproblemen) (vgl. Domschke et al., 1997).

Verbesserungsverfahren können erfolgreich für die Lösung des Job-Shop-Problems genutzt werden (vgl. Reeves, C. 1995). Damit die lokalen Optima überwunden und/oder vielversprechende Nachbarschaften untersucht werden können, wurden Metaheuristiken entwickelt, die die Verbesserungsverfahren nach bestimmten Prinzipien steuern (vgl. Domschke und Scholl, 2006). „Metaheuristiken sind dadurch charakterisiert, dass eine übergeordnete Strategie eine andere, meist problemspezifische Heuristik steuert und modifiziert, um bessere Lösungen zu finden, als jene, die die untergeordnete Heuristik allein finden würde." (vgl. Zäpfel und Braune, 2005, S. 25). Metaheuristiken sind problemunabhängig und können auf eine Vielzahl an Problemstellungen angewandt werden (vgl. Pirlot, 1996). Im Folgenden werden die populärsten Metaheuristiken, die zur Lösung des Job-Shop-Problems eingesetzt werden, ausführlich dargestellt und deren Funktionsweise erläutert (vgl. Zäpfel und Braune, 2005).

3.2.5.1 Simulated Annealing

Simulated Annealing ist eine Metaheuristik, die auf der lokalen Suche basiert und wurde in den 1980er-Jahren unabhängig von Kirkpatrick et al. (1983) und Černý (1985) zur Lösung von kombinatorischen Optimierungsproblemen entwickelt (vgl. Zäpfel und Braune, 2005). Das Verfahren basiert dabei auf dem Algorithmus von Metropolis et al. (1953), der den Erstarrungsvorgang einer flüssigen Materie zum Festkörper widerspiegelt. Das Ziel eines solchen Abkühlungsvorgang ist das Erreichen eines Zustandes minimaler Energie (analog zur Gesamtzykluszeit) (vgl. Zäpfel und Braune, 2005). Algorithmus 2 soll die Funktionsweise des Simulated Annealing verdeutlichen (vgl. Geiger, 2005).

Algorithmus 2: Simulated Annealing

(1): *Generiere eine Ausgangslösung* x

(2): ***Wiederhole***

(3): *Erzeuge ein Element* $x' \in N(x)$

(4): *Evaluiere* x'

(5): ***Wenn*** $z(x') \leq z(x)$ ***dann***

(6): $x \leftarrow x'$

(7): ***Sonst***

(8): $x \leftarrow x'$ *mit* $p_s = e^{-\frac{z(x')-z(x)}{t_i}}$

(9): ***Ende Wenn***

(10): *Vermindere* t_i

(11): ***Bis*** *Abbruchkriterium erfüllt*

Die Startlösung x des Algorithmus wird zufällig oder durch ein Eröffnungsverfahren bereitgestellt. Im Rahmen der Transformationsvorschriften wird eine Nachbarschaftslösung von x erzeugt (3). Nachdem der Zielfunktionswert der Nachbarschaftslösung x' bestimmt wurde (4), wird der Zielfunktionswert von x' mit dem Zielfunktionswert der aktuellen Lösung x verglichen (5). Ist der Zielfunktionswert der Nachbarschaftslösung x' kleiner, wird die Lösung x ersetzt (6), andernfalls (7) wird die schlechtere Lösung mit einer gewissen Wahrscheinlichkeit akzeptiert (8). Der zentrale Steuerungsparameter t_i des Algorithmus entspricht im ursprünglichen Algorithmus von Metropolis et al. der Temperatur im Iterationsschritt i. In Analogie zum Abkühlungsprozess wird der Steuerungsparameter t_i nach einer Verminderungsregel im Laufe des Algorithmus reduziert (10), sodass die Akzeptanzwahrscheinlichkeit einer schlechteren Lösung für $i \to \infty$ minimiert wird (8) (vgl. Geiger, 2005). Der Algorithmus sucht solange nach einer den Zielfunktionswert verbessernden Lösung, bis ein bestimmtes Abbruchkriterium, z.B. die fehlende Verbesserung des Zielfunktionswertes über einen bestimmten Zeitraum, erfüllt ist (11) (vgl. Zäpfel und Braune, 2005).

3.2.5.2 Genetische Algorithmen

Genetische Algorithmen wurden von Holland (1975) entwickelt und sind heuristische Optimierungsverfahren, die auf der Evolutionsstrategie basieren (vgl. Pesch und Voß, 1995). Sie ermöglichen die Lösung von Ablaufplanungsproblemen durch die Anwendung von Prozessen der biologischen Evolution (vgl. Zäpfel und Braune, 2005). Im Gegensatz zum Simulated Annealing betrachten genetische Algorithmen während eines Iterationsschritts nicht nur eine Lösung, sondern können eine ganze Menge von Lösungsmöglichkeiten (sog. Population) auf Verbesserungen untersuchen (vgl. Fritzsche, 2009; Goldberg,

1989). Der Suchprozess von genetischen Algorithmen besteht im Wesentlichen, analog zum Evolutionsprozess, aus der Selektion, Rekombination und Mutation (vgl. Domschke und Scholl, 2006).

- **Selektion**

Aus einer Ausgangspopulation werden Individuen, die die Lösung eines Problems in kodierter Form enthalten, ausgewählt und in die nachfolgende temporäre Population übertragen (vgl. Pesch und Voß, 1995). Die Auswahlwahrscheinlichkeit hängt dabei stark vom Fitnesswert (Zielfunktionswert) der Individuen ab (vgl. Domschke und Scholl, 2006). Je besser der Zielfunktionswert bzw. je fitter das Individuum ist, desto höher ist die Wahrscheinlichkeit, als Elternindividuum ausgewählt zu werden (vgl. Schwartz, 2004). Auf diese Weise kann das Prinzip von Darwin „survival of the fittest“ im Rahmen der Lösungssuche umgesetzt werden (vgl. Goldberg, 1989). Um lokale Optima überwinden zu können, werden teilweise auch schlechtere Lösungen für die nachfolgende Population selektiert (vgl. Schwartz, 2004).

- **Rekombination**

Im Rahmen der Rekombination werden zwei fitte Individuen (sog. Eltern) miteinander gekreuzt, um nach Möglichkeit neue und noch fittere Individuen (sog. Kinder) für die nachfolgende Population (sog. Generation) zu erzeugen (vgl. Zäpfel und Braune, 2005). Die Kombination bzw. der Austausch von Genen (Teillösungen) ermöglicht eine Erweiterung des Lösungsraums und somit ein Verlassen von lokalen Optima (vgl. Acker, 2011). Die Folge der Rekombination von fitten Individuen ist die Steigerung der durchschnittlichen Fitness bzw. des Zielfunktionswertes der gesamten Population über mehrere Generationen (vgl. Heistermann, 1994).

- **Mutation**

Mit einer geringen Wahrscheinlichkeit werden die durch die Rekombination erzeugten Individuen an zufälliger Stelle geringfügig verändert (vgl. Domschke und Scholl, 2006). Die Mutation von Individuen ermöglicht die Exploration von neuen Gebieten im Lösungsraum und verhindert somit eine verfrühte Konvergenz des Verfahrens, die durch die immer ähnlicher werdenden Individuen verursacht werden würde (vgl. Acker, 2011).

Der Algorithmus 3 soll die Funktionsweise des genetischen Algorithmus verdeutlichen (vgl. Zäpfel und Braune, 2005).

Algorithmus 3: Genetischer Algorithmus

(1): Erzeuge zufällig $|P_y|$ *Individuen für Initialpopulation* $P_0 = \{y_1, y_2, \dots, y_{P_y}\}$

(2): Bewerte jedes Individuum y_j*, für* $j = 1, \dots, P_y$*, d.h. ermittle* $z(y_j)$ *für alle* y_j

(3): $i \leftarrow 0$

(4): ***Wenn*** $i < i_{max}$ ***dann***

(5): *Initialisiere Folgepopulation:* $P_{i+1} \leftarrow \emptyset$

(6): ***Wenn*** $|P_{i+1}| < |P_y|$ ***dann***

(7): *Selektiere zwei Individuen* y_1 *und* y_2 *nach bestimmter Selektionsregel aus* P_i

(8): *Erzeuge zwei neue Individuen* y_1' *und* y_2' *durch Anwendung eines Kreuzungsoperators auf* y_1' *und* y_2'

(9): *Mutiere* y_1' *und* y_2' *mit einem Mutationsoperator mit gegebener Wahrscheinlichkeit zu* y_1'' *und* y_2''

(10): *Bewertung von* y_1'' *und* y_2''

(11): *Füge* y_1'' *und* y_2'' *zur Folgepopulation hinzu:* $P_{i+1} \leftarrow P_{i+1} \cup \{y_1'', y_2''\}$*, gehe zu (6)*

(12): ***Ende Wenn***

(13): $i \leftarrow i + 1$*, gehe zu (4)*

(14): ***Ende Wenn***

Zu Beginn des genetischen Algorithmus werden $|P_y|$ Individuen für die Initialpopulation P_0 erzeugt und anschließend werden die Fitnesswerte der Individuen z(y_j) bestimmt (2). Für die Folgepopulation werden zwei Individuen basierend auf ihren Fitnesswerten ausgewählt (7) und entsprechend des verwendeten Kreuzungsoperators miteinander gekreuzt (8). Nachdem die Nachkommen mit einer geringen Wahrscheinlichkeit mutiert sind (9), wird ihr Fitnesswert bestimmt (10) und sie werden der Folgepopulation hinzugefügt (11). Die Schritte (7) – (11) werden solange ausgeführt, bis die gewünschte Größe der Generation erreicht ist. Der Algorithmus terminiert (14) wenn i_{max} Generationen erzeugt wurden (4). Die Lösung mit dem besten Fitnesswert der letzten Generation ist die beste Lösung des Problems nach einer Anzahl an i Iterationen.

Ausschlaggebend für den Erfolg eines genetischen Algorithmus ist die Kodierung der Lösungskandidaten (vgl. Mitchell, 1996). Die Lösungskodierung ist problemspezifisch und für das Job-Shop-Problem existieren viele Kodierungsvorschläge. Als Beispiel soll an dieser Stelle die von Nakano und Yamada (1991) vorgeschlagene binäre Kodierung genannt werden. Sie basiert auf dem disjunktiven Graphen und repräsentiert mit den Werten 0 und 1 ein disjunktives Pfeilepaar. Eine umfassende Übersicht über weitere

Lösungskodierungen für das Job-Shop-Problem ist in Teich (1998), Zäpfel und Braune (2005) und Rixen (1997) gegeben.

3.2.5.3 Tabu-Suche

Die Tabu-Suche ist ein von Glover (1986) und Hansen (1986) beschriebenes Verbesserungsverfahren, das auf dem Prinzip der lokalen Suche basiert und in jedem Iterationsschritt die Nachbarschaft $N(x)$ vollständig durchsucht und (nicht verbotene) Nachbarschaftslösungen nach der Best-Fit-Strategie akzeptiert (vgl. Zäpfel und Braune, 2005; Domschke und Scholl, 2006). Ausgehend von einer bekannten Lösung x werden alle Nachbarschaftslösungen erzeugt (vgl. Geiger, 2005). Die beste Nachbarschaftslösung wird akzeptiert, auch wenn sie einen schlechteren Zielfunktionswert als die aktuelle Lösung aufweist und dient als Ausgangslösung für den nächsten Iterationsschritt $x_1' \in N(x), x_2' \in N(x_1'), \dots, x_i' \in N(x_{i-1}')$ (vgl. Geiger, 2005). Schlechtere Lösungen werden im Rahmen der Tabu-Suche und im Gegensatz zum Simulated Annealing allerdings nur akzeptiert, wenn keine Verbesserungsmöglichkeit in der gesamten Nachbarschaft existiert (vgl. Domschke et al., 1997). Damit das Verfahren nicht durch die Auswahl der besten Nachbarschaftslösung zu ehemaligen Lösungen zurückkehrt und somit in Zyklen gerät, werden bereits betrachtete Lösungen bzw. Züge, die zu ihnen führen, tabuisiert (vgl. Zäpfel und Braune, 2005). Die Liste, die Informationen über die verbotenen Lösungen bzw. Züge enthält, wird Tabu-Liste genannt und ist das zentrale Element des Verfahrens (vgl. Glover, 1989). Ist ein Zug verboten, erfüllt aber ein gewisses Aspirationskriterium (z.B. „ist besser als die aktuell beste Lösung“), kann dieser Zug dennoch ausgewählt werden (vgl. Zäpfel und Braune, 2005). Die Länge l der Tabu-Liste hat maßgeblichen Einfluss auf die Eigenschaften der Suche. Kurze Listen führen zu einer intensiven Suche, d.h. die Suche fokussiert sich auf die vielversprechenden Regionen des Lösungsraumes, während lange Listen zu einer diversifizierten, d.h. einer möglichst breiten, Suche führen (vgl. Glover, 1990). Die Tabu-Liste stellt somit ein Gedächtnis dar, das den Suchverlauf aufzeichnet und die Suchrichtung steuert, mit dem Ziel, eine Balance zwischen intensiver und diversifizierter Suche zu erreichen (vgl. Ting et al., 2003).

Einige Wissenschaftler sehen großes Potential in diesem Suchprozess, sodass sie die Tabu-Suche bzw. Elemente von ihr mit anderen Heuristiken verbinden und so hybride Verfahren entwickeln (vgl. Ting et al., 2003). Die Funktionsweise der Tabu-Suche soll mit Hilfe des Algorithmus 4 dargestellt werden (vgl. Zäpfel und Braune, 2005).

Algorithmus 4: Tabu-Suche

(1): *Erzeuge initiale Lösung* x

(2): $T \leftarrow \emptyset$

(3): $\hat{x} \leftarrow x$

(4): ***Wenn*** *Abbruchkriterium nicht erreicht* ***dann***

(5): *Bestimme Menge* $V(x)$ *der von Lösung* x *aus möglichen Züge*

(6): *Konstruiere Lösungen in* $N(x)$ *durch Anwendung aller erlaubten Züge* $v \in V(x)$ *mit* $v \notin T$

(7): *Wähle einen erlaubten Zug* v' *der zu bestem* $x' \in N(x)$ *führt*

(8): $x \leftarrow x'$

(9): $T \leftarrow T \cup \{v'^{-1}\}$

(10): ***Wenn*** $z(x) < z(\hat{x})$ ***dann***

(11): $\hat{x} \leftarrow x$

(12): ***Ende wenn***

(13): *Gehe zu (4)*

(14): ***Ende wenn***

Die Tabu-Suche startet mit einer zulässigen Lösung x (1) und einer leeren Tabu-Liste, wobei die Länge der Tabu-Liste ausschlaggebend für die Performance des Algorithmus ist (2). Eine lange Tabu-Liste schränkt die Suche stark ein, während eine kurze Tabu-Liste die Gefahr birgt in Zyklen zu geraten (vgl. Zäpfel und Braune, 2005). Die Lösung x ist zum Startzeitpunkt die aktuell beste Lösung $\hat{x}$ des Ablaufplanungsproblems (3). Solange das Abbruchkriterium nicht erreicht ist (4), wird die Menge aller möglichen Züge von x bestimmt (5) und durch die Anwendung aller erlaubten Züge werden die Nachbarschaftslösungen erzeugt (6). Von diesen Lösungen wird der Zug v' ausgewählt, der zur bestmöglichen Nachbarschaftslösung x' führt (7). Die Nachbarschaftslösung x' ist die Ausgangslösung x für den nächsten Iterationsschritt (8). Die Umkehrung des Zuges v', der zu der besten Nachbarschaftslösung geführt hat, wird auf die Tabu-Liste T gesetzt und ist somit für die folgenden Iterationsschritte verboten (9). Ist der Zielfunktionswert der neuen Ausgangslösung x kleiner als der Zielfunktionswert der aktuell besten Lösung $\hat{x}$ (10), wird diese ersetzt (11). Der Algorithmus terminiert, sobald ein bestimmtes Abbruchkriterium, wie z.B. die maximale Anzahl an Iterationsschritten oder die maximale Anzahl an Iterationsschritten ohne Verbesserung, erreicht wird.

3.2.5.4 Ant Colony Optimization

Ameisenalgorithmen wurden in den 1990ern von Dorigo et al. (1996) eingeführt und sind eine von der Natur inspirierte Metaheuristik zur Lösung von kombinatorischen Optimierungsproblemen (vgl. Dorigo und Blum, 2005). Die Ant-Colony-Optimization-Algorithmen (ACO-Algorithmen) von Dorigo und Di Caro (1999), Dorigo et al. (1999) und Dorigo und Stützle (2004) sind dabei die populärste Klasse der Ameisenalgorithmen (vgl. Dorigo und Stützle, 2004). Eine umfassende Forschungsübersicht zu ACO-Algorithmen liefert Dorigo (2014). Die ACO-Algorithmen basieren auf dem Verhalten einer Ameisenkolonie bei der Futtersuche. Die Ameisen suchen die Gegend um ihr Nest auf eine willkürliche Art und Weise nach Futter ab. Bei der Bewegung sondern die Ameisen einen chemischen Duftstoff, das sog. Pheromon, ab, den andere Ameisen riechen können. Steht eine Ameise vor einer Wegentscheidung, wählt sie mit großer Wahrscheinlichkeit den Weg mit der stärksten Pheromonkonzentration. Hat eine Ameise Nahrung gefunden, bewertet sie die Menge und Qualität des Futters und bringt ein Teil davon zurück ins Nest. Auf dem Rückweg verändert sich die abgesonderte Pheromonmenge in Abhängigkeit von der Quantität und Qualität des Futters. Auf diese Weise werden andere Ameisen auf diesen Weg aufmerksam und folgen der Pheromonspur zur Futterquelle. Die indirekte Kommunikation der Ameisen durch den Duftstoff Pheromon ermöglicht das Finden der kürzesten Wege zwischen dem Nest und der Futterquelle. Die durch das Zusammenwirken bei der Futtersuche entstehenden Fähigkeiten werden als Schwarmintelligenz bezeichnet (vgl. Bonabeau et al., 1999; Bonabeau et al., 2000). Die ACO-Algorithmen nutzen diese Schwarmintelligenz, um ein Multi-Agenten-System auf Basis des kollektiven Verhaltens von Ameisen zu entwickeln und so kombinatorische Optimierungsprobleme zu lösen. Die Ameisen (Agenten) sind dabei ein Teil der Ameisenkolonie (Agentensystem) (vgl. Blum, 2005; Dorigo und Blum, 2005).

Algorithmus 5 soll die Funktionsweise eines ACO-Algorithmus verdeutlichen (vgl. Blum, 2005; Zäpfel und Braune, 2005). Zu Beginn des ACO-Algorithmus wird die Anzahl der Ameisen n_a bestimmt (1) und die Pheromonspuren werden initialisiert (2). Jede künstliche Ameise konstruiert im Verlauf des Algorithmus eine Lösung x_{n_a} als Reihenfolge von Lösungskomponenten k_{n_a} unter der Annahme, dass eine gute Lösung aus guten Lösungskomponenten besteht (vgl. Dorigo und Blum, 2005). Die Lösungskonstruktion startet mit einer leeren Reihenfolge x_{n_a} (4). Im Anschluss werden die für die Erweiterung der Lösung x_{n_a} möglichen Lösungskomponenten $N(x_{n_a})$ bestimmt, wobei $N(x_{n_a}) \subseteq K_a \setminus x_{n_a}$ mit $K_a = \{k_1, \dots, k_n\}$ (5). Die Auswahl der nächsten Lösungskomponente erfolgt in jedem Konstruktionsschritt probabilistisch, basierend auf den Pheromonwerten (7).

Algorithmus 5: Ant Colony Optimization

(1): Bestimme Anzahl der Ameisen n_a
(2): Initialisiere Pheromonspuren
(3): ***Wenn*** *Abbruchkriterium für alle* n_a *nicht erreicht* ***dann***
(4): *Setze* $x_{n_a} = \{\}$
(5): *Bestimme* $N(x_{n_a})$
(6): ***Wenn*** x_{n_a} *nicht vollständig* ***dann***
(7): $k_{n_a} \leftarrow$ *wähle von* $N(x_{n_a})$
(8): $x_{n_a} \leftarrow$ *erweitere die Lösung um die Lösungskomponente* k_{n_a}
(9): *Bestimme* $N(x_{n_a})$*, gehe zu (6)*
(10): ***Ende Wenn***
(11): *Aktualisiere Pheromonspuren*
(12): *Führe Dämon-Aktivitäten durch*
(13): *gehe zu (3)*
(14): ***Ende Wenn***

Nachdem die Lösung x_{n_a} erweitert wurde (8), werden alle möglichen Lösungskomponenten für den nächsten Konstruktionsschritt erzeugt (9). Die Konstruktionsschritte 7–9 werden solange ausgeführt, bis die Lösung x_{n_a} vollständig konstruiert wurde (6). Damit das Verfahren nicht zu schnell konvergiert und lokale Optima nicht mehr verlassen werden können, werden alle Pheromonwerte abgeschwächt (Pheromonverdunstung). Dies entspricht einem „Vergessen“ und ermöglicht die Exploration des Lösungsraumes (11). Zudem werden aber auch Pheromonpfade, auf denen die Lösungskomponenten der aktuellen Lösung liegen, verstärkt (11). Mit Hilfe der Dämon-Aktivitäten (12) werden zentrale Eingriffe in das Verfahren vorgenommen, die durch die einzelnen Ameisen nicht realisiert werden können. So kann der Dämon z.B. entscheiden, ob einer Lösungskomponente, die zur aktuell besten Lösung gehört, ein erhöhter Pheromonwert zugewiesen wird, um diese Suchrichtung zu verstärken. Der Algorithmus terminiert mit dem Eintreten eines bestimmten Abbruchkriteriums, wie z.B. das Ablaufen einer definierten Laufzeit.

3.2.6 Vergleich der heuristischen Optimierungsverfahren

Tabelle 5 fasst die wesentlichen Merkmale der in den letzten Kapiteln vorgestellten Metaheuristiken zusammen und lässt somit einen kurzen abschließenden Vergleich der Verfahren zu. Die folgenden Ausführungen basieren, wenn nicht anders angegeben, auf denen von Zäpfel und Braune (2005).

Tabelle 5: Vergleich der heuristischen Optimierungsverfahren (Zäpfel und Braune, 2005)

Merkmale der Metaheuristiken	Simulated Annealing	Genetische Algorithmen	Tabu-Suche	ACO-Algorithmen
Hamming-Distanz	1	≥ 1	1	≥ 1
Lokale Suchprinzipien	• Nachbarschaftssuche • Bessere Lösung wird akzeptiert • Schlechtere Lösung wird mit bestimmter Wahrscheinlichkeit akzeptiert		• Nachbarschaftssuche • Beste Lösung der Nachbarschaft wird akzeptiert	
Naturanaloge Prinzipien	Abkühlprozess einer flüssigen Materie zum Festkörper	Selektion, Kreuzung, Mutation		Indirekte Kommunikation der Ameisen mittels Pheromon
Gedächtnisbasierte Prinzipien		Basierend auf dem Selektionsprozess der biologischen Evolution („Survival of the fittest“)	Tabu-Liste mit verbotenen Zügen	Forcierung von guten Lösungen durch Erhöhung des Pheromonwertes
Populationsbasiert	nein	ja	nein	ja

Das Simulated Annealing und die Tabu-Suche erzeugen in jedem Iterationsschritt Lösungen mit der Hamming-Distanz 1. Das bedeutet, dass die im Suchprozess generierten Lösungen sich um jeweils einen disjunktiven Pfeil im disjunktiven Graphen unterscheiden. Die Lösungssuche, die auf diese Weise zustande kommt, wird auch als Tiefensuche (sog. Intensifikation) bezeichnet. Metaheuristiken mit einer Hamming-Distanz ≥ 1, wie Genetische Algorithmen oder ACO-Algorithmen, erzeugen Lösungen, die sich in mehreren disjunktiven Pfeilen im Ablaufgraphen unterscheiden. Auf diese Weise werden Lösungen erzeugt, die sich stark unterscheiden bzw. weit voneinander entfernt sind. Hieraus resultiert die sogenannte Breitensuche (sog. Diversifikation), mit der neue unbekannte Regionen des Lösungsraumes erschlossen werden können. Eine erfolgsversprechende Metaheuristik fokussiert sich allerdings nicht nur auf die Breiten- oder Tiefensuche allein, sondern strebt vielmehr ein ausgeglichenes Verhältnis zwischen Intensifikation und Diversifikation an. So kann eine längere Tabu-Liste im Rahmen der Tabu-Suche und eine hohe Anfangstemperatur beim Simulated-Annealing-Verfahren neben der Tiefensuche auch zu einem gewissen Grad die Breitensuche ermöglichen. Umgekehrt lässt die Selektion von fitten Individuen bei den Genetischen Algorithmen und die Erhöhung der Pheromonwerte für vielversprechende Lösungen bei den ACO-Algorithmen neben der ausgeprägten Breitensuche auch die Tiefensuche zu. Immer häufiger werden allerdings Metaheuristiken, die sich auf die Tiefensuche forcieren mit Verfahren der Breitensuche kombiniert, um den Suchprozess zu optimieren (vgl. Ting et al., 2003). Eine in diesem Zusammenhang häufig auftretende Kombination ist die der Tabu-Suche mit Genetischen Algorithmen (vgl. Ting et al., 2003).

Metaheuristiken, die auf der Nachbarschaftssuche basieren, suchen pro Iterationsschritt eine bessere Lösung, wobei auch schlechtere Lösungen unter bestimmten Bedingungen akzeptiert werden, während populationsbasierte Verfahren ganze Lösungsmengen in einem Iterationsschritt untersuchen. Populationsbasierte Verfahren, aber auch Simulated-Annealing-Verfahren wenden naturanaloge Prinzipien zur Lösung von Ablaufplanungsproblemen an.

Heuristiken, die erfolgreiche Lösungen bzw. Lösungskomponenten oder aber verbotene Lösungen bzw. Züge speichern, besitzen ein „Gedächtnis“ zur effektiven Gestaltung des Suchprozesses und ermöglichen das Verlassen von lokalen Optima oder die Verstärkung der Suche auf einen vielversprechenden Bereich des Lösungsraumes.

Welche heuristischen Suchverfahren nun mehr Erfolg versprechen, wird unter dem Begriff „No Free Lunch“ diskutiert (vgl. Wolpert und Macready, 1997). Ursprünglich besagt diese Metapher, dass in einer Stadt alle Restaurants im Mittel die gleichen Preise über die gesamte Speisekarte fordern (vgl. Fritzsche, 2009). In Bezug auf Metaheuristiken bedeutet dies, dass die Suchverfahren im Mittel für verschiedene Zielfunktionen über einem

Lösungsraum gleich gute Ergebnisse liefern (vgl. Wolpert und Macready, 1997). Metaheuristiken sind allgemeine Verfahren und können für viele unterschiedliche Problemstellungen angewandt werden (vgl. Fritzsche, 2009). Jedoch schränken „eine Reihe von Annahmen über den Lösungsraum und die Zielfunktion“ (Fritzsche, 2009, S.41), die bei der Entwicklung von Metaheuristiken getroffen werden, die Problemstellungen ein, sodass eine Metaheuristik für ein Problem sehr gute Lösungen, für ein anderes hingegen nur schlechte Ergebnisse liefern kann (vgl. Fritzsche, 2009). Dennoch können Metaheuristiken für bestimmte Lösungsmengen und Zielfunktionen sehr leistungsfähig sein (vgl. Wolpert und Macready, 2005). Damit die Leistungsfähigkeit der Metaheuristiken für das Job-Shop-Problem beurteilt werden kann, müssen sie an eben diesem Problem unter gleichen Bedingungen getestet werden.

3.3 Auswahl einer Metaheuristik zur Lösung des Job-Shop-Problems

In den vorherigen Kapiteln wurden verschiedene Metaheuristiken vorgestellt, die sich grundsätzlich zur Lösung des Job-Shop-Problems eignen. Damit die Lösungsverfahren miteinander verglichen werden können, um eine leistungsfähige Metaheuristik für das Job-Shop-Problem zu bestimmen, müssen sie an den gleichen Problemen getestet werden. Die Benchmarkprobleme liefern dafür einen einheitlichen Standard, um die Verfahren zu testen und die Ergebnisse miteinander vergleichen zu können. Benchmarkprobleme haben unterschiedliche Problemdimensionen und Schwierigkeitsgrade, sodass die Grenzen und Fähigkeiten der Heuristiken aufgezeigt werden können. In der Literatur existieren viele Benchmarkprobleme, die von Fisher und Thompson (1963), Lawrence (1984), Adams et al. (1988), Applegate und Cook (1991), Storer et al. (1992), Yamada und Nakano (1992), Taillard (1993) und Demirkol et al. (1998) formuliert wurden.

Jain und Meeran (1999) haben die bekanntesten Heuristiken an den oben aufgelisteten Benchmarkproblemen getestet. Die Ergebnisse ihrer Tests sollen im Folgenden dazu dienen, eine leistungsfähige Heuristik zu identifizieren, die sich zur Lösung des Job-Shop-Problems und zur Kopplung mit der Simulation eignet. Wie in Kapitel 3.1 dargestellt, liegen die Grenzen von exakten Methoden zur Lösung des Job-Shop-Problems je nach Verfahren bei Problemgrößen von (20×15) („Branch & Bound“) und (20×20) (Manne-Modell), sodass ein heuristisches Optimierungsverfahren gesucht wird, das vor allem für größere Probleme gute Lösungen generiert. Die von Taillard (1993) und Demirkol et al. (1998) formulierten Benchmarkprobleme stellen die dafür notwendigen Probleminstanzen mit Größen von bis zu (100×20) bzw. (50×20) bereit und ermöglichen die Beurteilung der Leistungsfähigkeit von Heuristiken für größere Job-Shop-Probleme. Vaessens, R. J. M. et al. (1996) weisen darauf hin, dass die oberen

Grenzen (obere Grenzen nähern sich dem optimalen Wert von „oben“ und die unteren Grenzen von „unten“) einfacher zu bestimmen sind und sie meistens auch die beste bisher erzielte Lösung darstellen. Für die folgende Analyse der Ergebnisse sind deshalb nur die oberen Grenzen von Relevanz.

Tabelle 10 (s. Anhang A1) zeigt die Ergebnisse der Benchmarktests für die Probleme von Taillard (1993). Es ist zu erkennen, dass ab einer Größe von (30×15) bis (30×20) die besten oberen Schranken hauptsächlich von Balas und Vazacopoulos (1998) erzeugt werden. Lässt man die Ergebnisse von Taillards (1994) Tabu-Suche unbeachtet, da er selbst die Benchmarkprobleme formuliert hat und sie für Benchmarkprobleme anderer Autoren keine guten oberen Schranken generiert, liefert ab einer Problemgröße von (50×15) bis (100×20) hauptsächlich die Heuristik von Nowicki und Smutnicki (1996) die besten Ergebnisse. Allerdings sind auch beide Verfahren in der Lage, kleine Probleme optimal zu lösen. So findet die Heuristik von Nowicki und Smutnicki (1996) für das Problem TD14 (20×15) die optimale Lösung und das Verfahren von Balas und Vazacopoulos (1998) ermöglicht die optimale Lösung von TD10 (15×15). Für die Benchmarkprobleme von Demirkol et al. (1998) mit den Größen (20×15), (20×20), (30×15), (30×20), (40×15), (40×20), (50×15) und (50×20) liefert fast ausschließlich die Tabu-Suche von Nowicki und Smutnicki (1996) die besten oberen Schranken und sogar auch zwei optimale Lösungen für (40×15) Probleme (s. Tabelle 11, Anhang A1). Wird Tabelle 10 (s. Anhang A1) betrachtet, so fällt auf, dass viele große Probleme bereits optimal gelöst wurden (TD71-TD80), während hingegen kleine Probleme (TD21-TD30) noch immer ungelöst sind. Matsuo et al. (1988) und Taillard (1989; 1994) stellen in diesem Zusammenhang fest, dass Probleme leichter zu lösen sind, wenn das Verhältnis von Jobs zu Maschinen größer als vier ist. Des Weiteren sind Probleminstanzen, die in ihrer Dimension quadratisch sind, schwerer zu lösen (vgl. Jain und Meeran, 1999). An dieser Stelle wäre es folglich unzureichend, eine Heuristik zur Lösung des Job-Shop-Problems auszuwählen, die nur für die großen Benchmarkprobleme gute obere Schranken bestimmt, aber an den kleineren schweren Probleminstanzen scheitert. Vaessens, R. J. M. et al. (1996) stellten ein Set bestehend aus 13 Problemen zusammen, die von anderen Autoren und von ihnen als schwer lösbar bezeichnet wurden, um einen Performance-Vergleich von Heuristiken zu ermöglichen. Laut Vaessens, R. J. M. et al. (1996) soll eine gute Heuristik in der Lage sein, diese 13 Probleme innerhalb von 100 CI-CPU-Sekunden mit einer mittleren relativen Abweichung von maximal 2 % (insgesamt) zu lösen. Tabelle 12 (s. Anhang A1) repräsentiert die Ergebnisse der Heuristiken für diese 13 Benchmarkprobleme. Im Gegensatz zu dem „Branch & Bound“-Verfahren von Martin (1996) ist zu erkennen, dass keine Heuristik in der Lage ist, alle 13 Probleme optimal zu lösen. Am besten schneidet das Verfahren von Balas und Vazacopoulos (1998) mit 11 optimal gelösten Problemen ab. Sowohl die Heuristik von

Nuijten und Le Pape (1998) als auch die Heuristik von Thomsen (1997) lösen 10 der 13 Probleme optimal. Das Simulated-Annealing-Verfahren von Yamada und Nakano (1996) und die Tabu-Suche von Nowicki und Smutnicki (1996) können hingegen 7 Probleme lösen. Würde sich die Auswahl der Heuristik lediglich auf die Anzahl der optimal gelösten Probleme beschränken, sollte das Verfahren von Balas und Vazacopoulos (1998) gewählt werden. Allerdings spielt nach Vaessens, R. J. M. et al. (1996) neben der benötigten Zeit auch die mittlere relative Abweichung bei der Beurteilung der Leistungsfähigkeit eine übergeordnete Rolle (s. Tabelle 13, Anhang A1). Betrachtet man die Summe der mittleren relativen Abweichung, erzielen Balas und Vazacopoulos (1998) das beste Ergebnis mit 0,60 %. Die Tabu-Suche von Nowicki und Smutnicki (1996) erzielt hingegen eine mittlere relative Abweichung von 2,59 % und liefert den schlechtesten Wert. Obwohl die Tabu-Suche Nowicki und Smutnicki (1996) die Grenze von Vaessens, R. J. M. et al. (1996) überschreitet, ist eine mittlere relative Abweichung von 2,59 % ein immer noch sehr guter Wert (vgl. Jain und Meeran, 1999). Im Vergleich zu den anderen Heuristiken ist die Tabu-Suche von Nowicki und Smutnicki (1996) mit einer durchschnittlichen Dauer von ≈ 107 CI-CPU-Sekunden das schnellste Verfahren und kann die ausgewählten Benchmarkprobleme bis zu 274-mal so schnell lösen. Das „Branch and Bound"-Verfahren von Martin (1996) kann zwar alle der 13 Probleme lösen, benötigt dafür allerdings durchschnittlich ≈ 97160 CI-CPU-Sekunden. Dieser Vergleich zeigt eindrucksvoll, dass mit einer Heuristik eine hinreichend gute Lösung in deutlich kürzerer Zeit erreicht werden kann.

Alle der in Tabelle 13 (s. Anhang A1) aufgeführten Heuristiken sind in der Lage, die von Vaessens, R. J. M. et al. (1996) ausgewählten Benchmarkprobleme mit einer akzeptablen Abweichung vom Optimum zu lösen. Die Verfahren von Balas und Vazacopoulos (1998) und Nowicki und Smutnicki (1996) erzielen zusätzlich gute Ergebnisse für größere Job-Shop-Probleme (s. Tabelle 10/11, Anhang A1). Die Tabu-Suche von Nowicki und Smutnicki (1996) liefert fast alle besten oberen Schranken der großen und schweren Benchmarkprobleme von Demirkol et al. (1998), während die Heuristik von Balas und Vazacopoulos (1998) die meisten oberen Schranken für die von Taillard (1993) formulierten Probleme liefert. Die Tabu-Suche von Nowicki und Smutnicki (1996) kann im Vergleich zur Heuristik von Balas und Vazacopoulos (1998) weniger der 13 Benchmarkprobleme optimal lösen und die mittlere relative Abweichung ist ebenfalls schlechter, dafür ermöglicht sie die Lösung der Probleme innerhalb kürzester Zeit. Die Lösung von Problemen mit einer hoher Geschwindigkeit und einer hinreichend guten Qualität birgt großes Potential für die Kopplung mit der Simulation (s. Kapitel 4.3.3), sodass die Tabu-Suche von Nowicki und Smutnicki (1996) im Folgenden Teil der Arbeit mit der Simulation gekoppelt wird.

3.4 Tabu-Suche von Nowicki und Smutnicki

Die von Nowicki und Smutnicki (1996) entwickelte Tabu-Suche dient zur Reduzierung der Gesamtzykluszeit im Rahmen des Job-Shop-Problems. Algorithmus 6 stellt die Funktionsweise und Besonderheiten der Tabu-Suche von Nowicki und Smutnicki (1996) dar.

Algorithmus 6: Tabu-Suche von Nowicki und Smutnicki

(1): Erzeuge initiale Lösung x
(2): Initialisiere $T = \emptyset$
(3): $\hat{x} \leftarrow x$
(4): $i = 0$
(5): ***Wenn*** $i < i_{max}$ ***dann***
(6): $i \leftarrow i + 1$
(7): Bestimme die Menge $V(x)$ *der von Lösung* x *aus möglichen Züge*
(8): ***Wenn*** $V(x) = \emptyset$ ***dann***
(9): Stopp! Gebe aktuell beste Lösung $\hat{x}$ *aus*
(10): ***Sonst***
(11): Bestimme die Menge der VP-Züge A
(12): ***Wenn*** $(V(x)\backslash T) \cup A \neq \emptyset$ ***dann***
(13): Wähle $v' \in (V(x)\backslash T) \cup A$*, sodass*
$C_{max}(N(x, v')) = \min\{C_{max}\big(N(x, v)\big) \mid v \in (V(x)\backslash T) \cup A\}$
(14): ***Oder Wenn*** $|V(x)| = 1$, ***dann***
(15): Wähle $v' \in V(x)$
(16): ***Sonst***
(17): ***Wiederhole*** $T \leftarrow T \oplus T_l$, ***solange*** *bis* $V(x)\backslash T \neq \emptyset$
(18): Wähle $v' \in V(x)\backslash T$
(19): ***Ende Wenn***
(20): ***Ende Wenn***
(21): $x' \leftarrow N(x, v')$
(22): $T' \leftarrow T \oplus v'^{-1}$
(23): $x \leftarrow x'$
(24): $T \leftarrow T'$
(25): ***Wenn*** $C_{max}(x) < C_{max}(\hat{x})$ ***dann***
(26): $\hat{x} \leftarrow x$
(27): $i \leftarrow 0$

Fortsetzung Algorithmus 6

(28): ***Ende Wenn***

(29): *gehe zu (5)*

(30): ***Ende Wenn***

Der Algorithmus startet mit einer initialen Lösung für das Ablaufplanungsproblem x (1) und einer leeren Tabuliste T (2). Die initiale Lösung stellt zu Beginn die aktuelle beste Lösung $\hat{x}$ dar (3). i repräsentiert innerhalb des Algorithmus einen Zähler, der die Iterationen ohne Verbesserungen erfasst und wird zu Beginn auf 0 gesetzt (4). So lange dieser Zähler kleiner ist als ein vorher bestimmtes Maximum, wird der Algorithmus ausgeführt (5) und der Zähler hochgesetzt (6). Nun wird die Menge aller möglichen Züge in der Nachbarschaft von x bestimmt (7). Die Anzahl der erlaubten Züge hängt dabei maßgeblich von der Nachbarschaftsfunktion ab. Die meisten Tabu-Suchen erzeugen Nachbarschaftslösungen, indem zwei aufeinanderfolgende Operationen, die auf dem kritischen Pfad liegen, miteinander vertauscht werden. Die Größe der Nachbarschaft ist somit von der Anzahl der kritischen Pfade sowie der Anzahl an Operationen jedes kritischen Pfades abhängig und kann dementsprechend groß werden (vgl. van Laarhoven, Peter J. M. et al., 1992; Taillard, 1989). Taillard (1989) versucht solchen großen Nachbarschaften durch die Bestimmung von unteren Schranken gerecht zu werden, sodass eine vollständige Berechnung der Gesamtzykluszeit jeder Nachbarschaftslösung nicht erforderlich ist, wohingegen Nowicki und Smutnicki (1996) die Nachbarschaften direkt reduzieren. Hierzu werden zuerst die Operationen, die auf einem zufällig ausgewählten kritischen Pfad liegen, in Blöcke eingeteilt $B_1, \dots, B_r$. Anschließend können pro Zug entweder die ersten oder letzten beiden Operationen von $B_2, \dots, B_{r-1}$, die letzten beiden Operationen von B_1 oder die ersten beiden Operationen von B_r vertauscht werden (s. Abbildung 11).

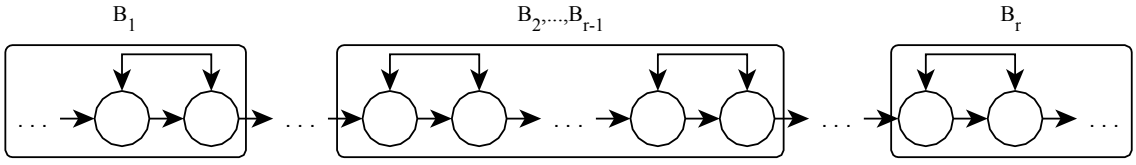

Abbildung 11: Transformationsregeln von Nowicki und Smutnicki (Zäpfel und Braune, 2005)

Existieren keine möglichen Züge (8), terminiert der Algorithmus und die aktuell beste Lösung $\hat{x}$ wird ausgegeben (9). Andernfalls können die Züge drei Klassen zugeordnet werden:

1. $V(x)\backslash T$: Erlaubte Zug (E-Züge)
2. $A = v \in V(x) \cap T : C_{max}\big(N(x,v)\big) < C_{max}(\hat{x})$: Verbotene, aber profitable Züge (VP-Züge)

3. $V(x) \cap T \backslash A$: Verbotene und nicht-profitable Züge (VNP-Züge)

Aus einer Menge von E- und VP-Zügen (12) wird der Zug v' ausgewählt, der zur Nachbarschaftslösung $N(x, v')$ mit dem besten Zielfunktionswert führt (13). Existieren weder E- noch VP-Züge, ist für den Suchprozess entscheidend, welcher VNP-Zug selektiert wird. Enthält die Menge nur einen VNP-Zug (14), wird dieser zur Erzeugung der Nachbarschaftslösung gewählt (15), andernfalls (16) wird der „älteste" Zug gewählt und die Tabu-Liste T durch Replikation des „jüngsten" Zuges solange modifiziert bis $V(x) \neq \emptyset$ (17). Dann enthält die Menge $V(x) \backslash T$ genau einen möglichen Zug, der als nächstes ausgewählt werden kann (18). Das folgende Beispiel illustriert diesen besonderen Selektionsprozess von Nowicki und Smutnicki (1996).

Man betrachte eine Menge von VNP-Zügen $V(x) = \{(3,2), (\mathbf{4},\mathbf{6})\}$ und eine Tabu-Liste

$$T = \{(1,2), (\mathbf{4},\mathbf{6}), (1,4), (3,2), (4,3)\}$$

mit der Länge $l = 5$. Nach der beschriebenen Selektionsregel wird nun der Zug, der am längsten auf der Tabu-Liste steht und auch in der Menge $V(x)$ enthalten ist, gewählt $(4,6)$. Die Tabu-Liste wird durch Replikation des letzten Zuges $(4,3)$ solange verändert, bis der ausgewählte Zug $(4,6)$ nicht mehr tabu ist. Das Ergebnis ist die modifizierte Tabu-Liste

$$T = \{(1,4), (3,2), (4,3), (4,3), (4,3)\},$$

die die Wahl des Zuges $(4,6)$ erlaubt.

Nachdem einer der drei Fälle (12), (14) oder (16) eingetreten ist, wird die Nachbarschaftslösung x' durch Anwendung des gewählten Zuges v' erzeugt (21) und die Tabuliste durch Hinzufügen der Umkehrung des Zuges v'^{-1} modifiziert (22). Anschließend wird die Lösung x durch die Nachbarschaftslösung x' (23) und die Tabuliste T durch die aktualisierte Tabuliste T' (24) ersetzt. Ist der Zielfunktionswert der Lösung x kleiner als der aktuell beste Zielfunktionswert (25), wird die aktuell beste Lösung $\hat{x}$ durch die Lösung x ersetzt (26) und der Zähler i wieder auf 0 gesetzt (27). Der Algorithmus terminiert, wenn die maximale Anzahl an Iterationen ohne Verbesserungen (i_{max}) erreicht wird (5).

3.5 Simulation

Die Analyse und Beurteilung von technischen Systemen, wie dem Produktionssystem, anhand mathematisch-analytischer Modelle ist oftmals nicht möglich, da zu viele zeit- und zufallsabhängige Variablen sowie Wirkzusammenhänge existieren (vgl. Wenzel, 2000, März und Weigert, 2011, VDI 3633 Blatt 1). Damit solche hochkomplexen Systeme dennoch geplant, realisiert und betrieben werden können, findet die Simulationstechnik Anwendung (vgl. VDI 3633 Blatt 1).

3.5.1 Grundlagen Simulation

Zu Beginn dieses Kapitels sollen zunächst die wichtigsten Begrifflichkeiten, die im direkten Zusammenhang mit der Simulation stehen, definiert werden, um jederzeit ein einheitliches Begriffsverständnis zu gewährleisten. Die Definitionen folgen dabei, wenn nicht anders angegeben, der vom Verein Deutscher Ingenieure (2014) veröffentlichten Richtlinie 3633 Blatt 1.

Definition System

Ein *System* ist „eine von ihrer Umwelt abgegrenzte Menge von Elementen, die miteinander in Beziehung stehen“ (VDI 3633 Blatt 1, S. 4). Die *Entitäten* sind die Elemente des Systems, dessen Verhalten detailliert betrachtet werden soll, und werden mittels Attribute charakterisiert (vgl. Pidd, 1998). Eine *Aktivität* innerhalb eines Systems, ausgelöst durch ein Ereignis, besitzt eine spezifische Dauer und transformiert den Zustand der entsprechenden Entität (vgl. Pidd, 1998). Ein eintretendes *Ereignis* kann den Systemzustand folglich mit sofortiger Wirkung ändern. Der *Systemzustand* wird zu jedem beliebigen Zeitpunkt durch alle zur Beschreibung notwendigen Zustandsvariablen der Entitäten definiert (vgl. Banks et al., 2014). Tabelle 6 stellt das Job-Shop-System mit seinen Komponenten exemplarisch dar.

Tabelle 6: Komponenten des Job-Shop-Systems

System	Entitäten	Attribute	Ereignis	Zustands-variable
Produktions-system (Job-Shop)	Maschinen	Prozesszeit, Kapazität	Start/Ende der Bearbeitung	Zustand der Maschine (frei, belegt)
	Aufträge	Maschinenfolge, Umfang	Ankunft an Maschine, Verlassen der Maschine	Zustand des Auftrags (wartend, in Bearbeitung)

3.5.1.1 Definition Modell

Ein *Modell* ist eine „vereinfachte Nachbildung eines geplanten oder existierenden Systems mit seinen Prozessen in einem anderen begrifflichen oder gegenständlichen System. Es unterscheidet sich hinsichtlich der untersuchungsrelevanten Eigenschaften nur innerhalb eines vom Untersuchungsziel abhängigen Toleranzrahmens vom Vorbild“ (VDI 3633 Blatt 1, S. 3). Ein Modell eines Systems, das mit Hilfe der Simulation analysiert und untersucht wird, bezeichnet man als *Simulationsmodell* (vgl. Carson und Maria, 1997). Das Simulationsmodell bildet das zu untersuchende System in einem geringeren Detaillierungsgrad („so abstrakt wie möglich, so detailliert wie nötig“

(Wenzel, 2000, S. 6)) ab und beschreibt das Optimierungsproblem mit seinen Nebenbedingungen und Restriktionen (z.B. Maschinenfolge der Aufträge, Kapazität der Maschinen, etc.). Law (2007) teilt solche Simulationsmodelle in folgende Klassen ein:

- Statisch vs. dynamisch

Statische Simulationsmodelle repräsentieren das System zu einem bestimmten Zeitpunkt oder sie stellen ein System dar, indem Zeit keine Rolle spielt (z.B. Monte Carlo Modelle). Dynamische Modelle hingegen zeigen, wie sich ein System über die Zeit entwickelt respektive verhält (z.B. eine Fördereinrichtung in einem Produktionssystem).

- Deterministisch vs. stochastisch

Ein deterministisches Simulationsmodell enthält keine zufallsabhängigen Komponenten, sodass bei einer unveränderten Inputvariation der Output determiniert ist (z.B. ein Simulationsmodell zur primalen Ablaufplanung). Mit stochastischen Modellen können Systeme nachgebildet werden, die zufallsabhängige Elemente enthalten (z.B. Prozesszeiten mit bestimmten Verteilungsfunktionen). Der Output von stochastischen Modellen ist zufallsbedingt und kann somit nur als Schätzung der eigentlichen Eigenschaften des Modells dienen.

- Kontinuierlich vs. diskret

Kontinuierliche Simulationsmodelle erlauben kontinuierliche Zustandsänderungen des Systems über den Zeitverlauf. Diskrete Modelle hingegen können die Systemzustände nur zu diskreten Zeitpunkten ändern.

3.5.1.2 Definition Validierung

Im eigentlichen Sinne versteht man unter der „Validierung [.] die kontinuierliche Überprüfung, ob die Modelle das Verhalten des abgebildeten Systems hinreichend genau wiedergeben“ (Rabe et al., 2008, S. 15). Da in dieser Arbeit kein Modell entwickelt wird, sondern ein Konzept zur Kopplung von Simulation und Optimierung, soll die Validierung Aufschluss geben, ob die funktionalen Anforderungen an das Konzept erfüllt werden.

3.5.1.3 Definition Stellgrößen

Stellgrößen bezeichnen die Eingriffspunkte in das Simulationsmodell (vgl. Klemmt, 2012) und haben somit maßgeblichen Einfluss auf das Verhalten des Modells (vgl. Weigert et al., 2009). Weigert und Rose (2011) unterscheiden zwischen drei Typen von Stellgrößen:

1. Parametervariation: Mit Hilfe einer Parametervariation kann z.B. der Puffer einer Maschine oder die Kapazität einer Maschine verändert werden.

2. Zuordnungen/Auswahl: Diese Stellgrößen ermöglichen z.B. die Zuordnung von Jobs zu Maschinen oder die Auswahl von Wartedisziplinen für ein Pufferlager (z.B. FIFO).
3. Permutation: Reihenfolgen, wie z.B. die Auftragsfolge, werden durch Permutationen dargestellt. Die Veränderung einer Permutation hat einen wirksamen Einfluss auf das Ablaufverhalten eines Systems (vgl. Weigert und Rose, 2011).

Treten im Rahmen des Optimierungsproblems Zuordnungs- und Permutationsvariablen auf, ist dies ein starker Hinweis, dass die simulationsbasierte Optimierung angewandt werden sollten (vgl. Weigert und Rose, 2011). Das Job-Shop-Problem aus Kapitel 2.5 ist hierfür ein gutes Beispiel. Die Maschinenfolge repräsentiert eine Zuordnungsvariable und die gesuchte Auftragsfolge stellt eine Stellgröße vom Typ Permutation dar. Folgendes Beispiel illustriert eine mögliche Stellgröße im Rahmen des Job-Shop-Problems.

Beispiel: Innerhalb eines disjunktiven Graphen besteht auf einer Maschine M_1 ein Ressourcenkonflikt zwischen J_2, J_5 und J_8. Zu dessen Lösung wird nun eine Stellgröße $S_1 = \{1,2,3\}$ vom Typ Permutation erzeugt. Im Anschluss werden die Jobs in 3! unterschiedliche Reihenfolgen gebracht und mit der Stellgröße S_1 eindeutig verknüpft (z.B. $J_5 \rightarrow 1$, $J_8 \rightarrow 2, J_2 \rightarrow 3$). Im Anschluss wird die Stellgröße S_1 mit der Maschine M_1 verknüpft und die Zielgröße kann, basierend auf den verschiedenen Reihenfolgealternativen, beurteilt werden.

Nicht alle Stellgrößen haben Auswirkungen auf die Zielgrößen, sodass nur die Stellgrößen berücksichtigt werden sollten, die das Systemverhalten beeinflussen, um den Rechenaufwand so gering wie möglich zu halten (vgl. Weigert und Rose, 2011).

3.5.1.4 Definition Zielgrößen

Sind die Stellgrößen identifiziert, müssen die Zielgrößen, die im Fokus der Optimierung liegen, bestimmt werden. Ein Überblick über die gängigen Zielgrößen der Ablaufplanung ist in Kapitel 2.3.3 zu finden. Wichtig ist vor allem der Zusammenhang zwischen den Stell- und Zielgrößen (vgl. Weigert und Rose, 2011). Ist der Zusammenhang nicht oder nur mit sehr hohem Aufwand mathematisch formulierbar (s. Grenzen des Manne-Modells), wird die simulationsbasierte Optimierung angewandt (vgl. Weigert und Rose, 2011). Die Simulation liefert nach jedem Simulationsdurchlauf für bestimmte Werte und Ausprägungen der Stellgrößen konkrete Werte der Zielgrößen (vgl. Weigert und Rose, 2011). Im Rahmen des Job-Shop-Problems ist lediglich der Wert der Gesamtzykluszeit (C_{max}) von Interesse.

3.5.1.5 Definition Simulation

Simulation ist das „Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierbaren Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind. Insbesondere werden die Prozesse über die Zeit entwickelt“ (VDI 3633 Blatt 1, S. 3). Nach Wenzel (2000) kann die Simulation wie folgt klassifiziert werden (s. Abbildung 12).

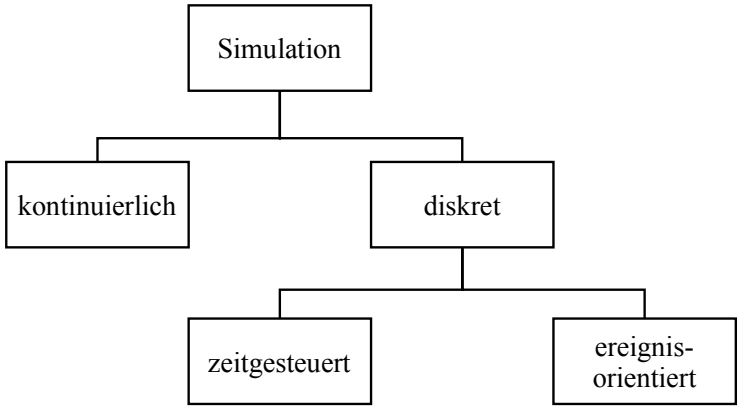

Abbildung 12: Klassifikation der Simulation (Wenzel, 2000)

Analog zu den Simulationsmodellen existieren kontinuierliche und diskrete Simulationssysteme. Im Rahmen der kontinuierlichen Simulation ändert sich der Zustand des Modells stetig über die Zeit (s. Abbildung 13).

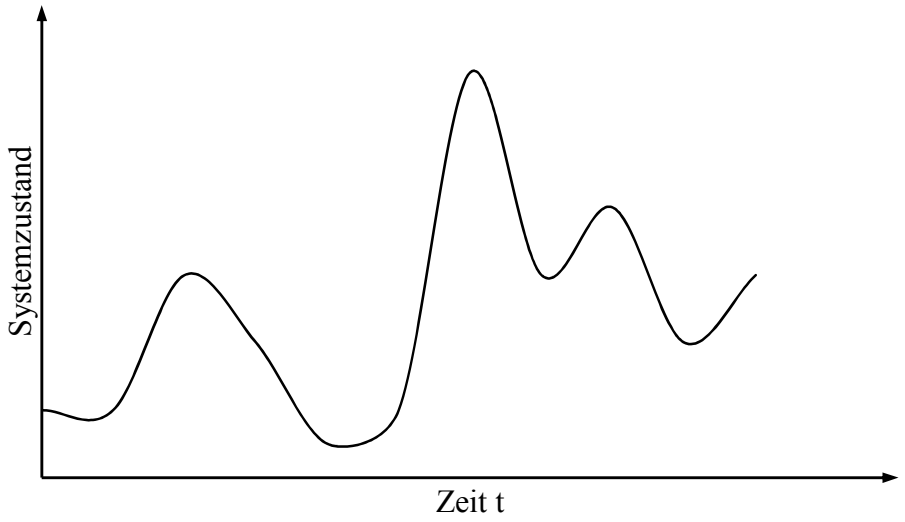

Abbildung 13: Kontinuierliche Systemzustandsänderung (Banks, 2014)

Innerhalb diskreter Simulationssysteme ändert sich der Systemzustand hingegen sprunghaft zu diskreten Zeitpunkten (s. Abbildung 14).

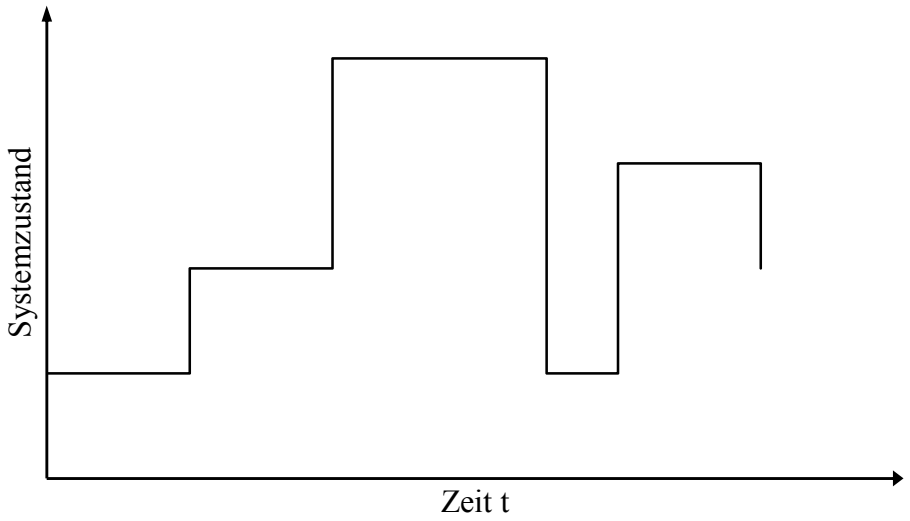

Abbildung 14: Diskrete Systemzustandsänderung (Banks, 2014)

Diskrete Simulationen können entweder zeitgesteuert oder ereignisorientiert sein. Wird die Simulationszeit um ein konstantes Zeitintervall erhöht und alle in diesem Zeitraum aufgetretenen Zustandsänderungen werden durchgeführt, liegt eine zeitgesteuerte-diskrete Simulation vor. Im Gegensatz hierzu wird eine Zustandsänderung in einem ereignisorientierten System durch das Eintreten eines internen Ereignisses herbeigeführt (vgl. Horn, 2008). Die Zeit zwischen Ereignissen verbraucht Simulationszeit, aber keine Rechenzeit, während die Zustandsänderungen beim Eintreten eines Ereignisses Rechenzeit, aber keine Simulationszeit verbrauchen (vgl. Wenzel, 2000).

Die Simulation wurde in der Vergangenheit hauptsächlich zur Planungsabsicherung verwendet. Heutzutage wird sie in der Produktionsplanung, -realisierung sowie -steuerung angewandt (VDI 3633 Blatt 1) und wird somit auch im Rahmen der Ablaufplanung eingesetzt (vgl. Fowler et al., 2006).

3.5.2 Einsatzgebiete der Simulation innerhalb der Ablaufplanung

Viele Autoren befassen sich mit dem Einsatz der Simulation in den verschiedensten Bereichen der Produktion und Logistik und geben einen guten Überblick über die möglichen Einsatzgebiete der Simulation. Fowler et al. (2006) fokussieren sich dabei auf den Bereich der Ablaufplanung und identifizieren mehrere Einsatzmöglichkeiten der Simulation:

- Bildung von Ablaufplänen

Die Simulation wird genutzt, um Ablaufpläne mit einem zeitlichen Horizont von mehreren Stunden bis zu einem Tag zu generieren. Prioritätsregeln können dabei Teil der Simulation sein und die einzelnen Jobs den verschiedenen Maschinen zuweisen.

- Bildung von detailgetreuen Ablaufplänen

Existiert bereits ein Ablaufplan, wird die Simulation genutzt, um das Systemverhalten, wie z.B. die Start- und Endzeitpunkte der einzelnen Jobs, besser schätzen zu können. Hierfür wird das Modell, das zur Bildung des initialen Ablaufplans verwendet wurde, mit detaillierten Informationen ergänzt. So basieren die Prozesszeiten nicht mehr auf deterministischen Werten, sondern werden durch entsprechende Verteilungsfunktionen ersetzt, um die Realität besser abbilden zu können.

- Parametrisierung von Heuristiken sowie Generierung von Testinstanzen für Heuristiken

Viele Heuristiken müssen vor ihrem Einsatz mit Hilfe der Simulation parametrisiert werden, damit sie an die vorliegenden Situationen adaptiert werden können. Des Weiteren können mit der Simulation Testinstanzen und Benchmarks für Heuristiken erzeugt werden.

- Bewertung von Heuristiken

Bevor eine Heuristik in der Praxis zur Ablaufplanung verwendet wird, muss das Verhalten des Produktionssystems in Abhängigkeit der zur Ablaufplanung verwendeten Heuristik mit Hilfe der Simulation analysiert werden.

- Optimierung von Ablaufplänen

Mit Hilfe einer Optimierungsmethode wird ein zulässiger Ablaufplan gesucht, der eine oder mehrere Zielfunktionen optimal erfüllt. Die Simulation dient in diesem Zusammenhang zur Bestimmung der Zielgrößen für die Lösungsalternativen und ermöglicht somit die Bewertung der unterschiedlichen Ablaufpläne (vgl. Fu, 2002).

Das Einsatzgebiet des im Rahmen dieser Arbeit entwickelten Konzeptes zur Kopplung von Simulation und Optimierung liegt innerhalb dieses Bereichs. Die Gründe für die Entwicklung eines neuen Konzeptes sowie die damit verbundenen Anforderungen werden in Kapitel 3.5.6 erörtert.

3.5.3 Ereignisdiskrete Simulation

Die ereignisdiskrete Simulation ist die häufigste im Bereich der Produktion und Logistik angewandte Methode zur Planung, Bewertung, Verbesserung und Steuerung von Systemen und Prozessen (vgl. Rose und März, 2011). Sie wird erfolgreich zur Reihenfolgeplanung und -optimierung sowie Ablaufplanung genutzt (vgl. Krug und Schwope, 2011, Iltzsche et al., 2011, Schwede et al., 2011, Heib und Nickel, 2011). Jedes ereignisdiskrete System besteht aus folgenden Kernkomponenten (vgl. Law, 2007):

- Systemzustand: Menge von Zustandsvariablen, die das System zu einem bestimmten Zeitpunkt beschreiben.
- Simulationsuhr: Variable mit der aktuellen Simulationszeit.
- Ereignisliste: Liste mit Zeitpunkten, zu denen die jeweiligen Ereignistypen eintreten.
- Statistischer Zähler: Variablen zur Speicherung von statistischen Informationen über das Systemverhalten.
- Initialisierungsroutine: Unterprogramm zur Initialisierung des Simulationsmodells zum Zeitpunkt 0.
- Zeitführungsroutine: Unterprogramm zur Bestimmung des nächsten Ereignisses aus der Ereignisliste und zum Vorstellen der Simulationszeit auf den Startzeitpunkt des nächsten Ereignisses.
- Ereignisroutine: Unterprogramm zur Aktualisierung des Systemzustands entsprechend des eintretenden Ereignistyps. Es existiert jeweils eine Ereignisroutine pro Ereignistyp.
- Bibliotheksroutinen: Menge von Unterprogrammen zur Erzeugung von Zufallsgrößen, die durch die Wahrscheinlichkeitsverteilung im Simulationsmodell festgelegt sind.
- Berichtsroutine: Unterprogramm, das nach Ende der Simulation auf Basis der statistischen Zähler Schätzwerte für die gewünschten Systemleistungsgrößen generiert und einen Bericht erzeugt.
- Hauptprogramm: Unterprogramm, das durch Aufrufen der Zeitführungsroutine das nächste Ereignis bestimmt und die entsprechende Ereignisroutine zur Änderung des Systemzustandes aufruft. Es überprüft des Weiteren, ob die Simulation endet und ruft im gegebenen Fall die Berichtsroutine auf.

Die folgende Abbildung 15 stellt den Zusammenhang der oben aufgelisteten Komponenten dar und repräsentiert zugleich den Ablauf einer ereignisdiskreten Simulation.

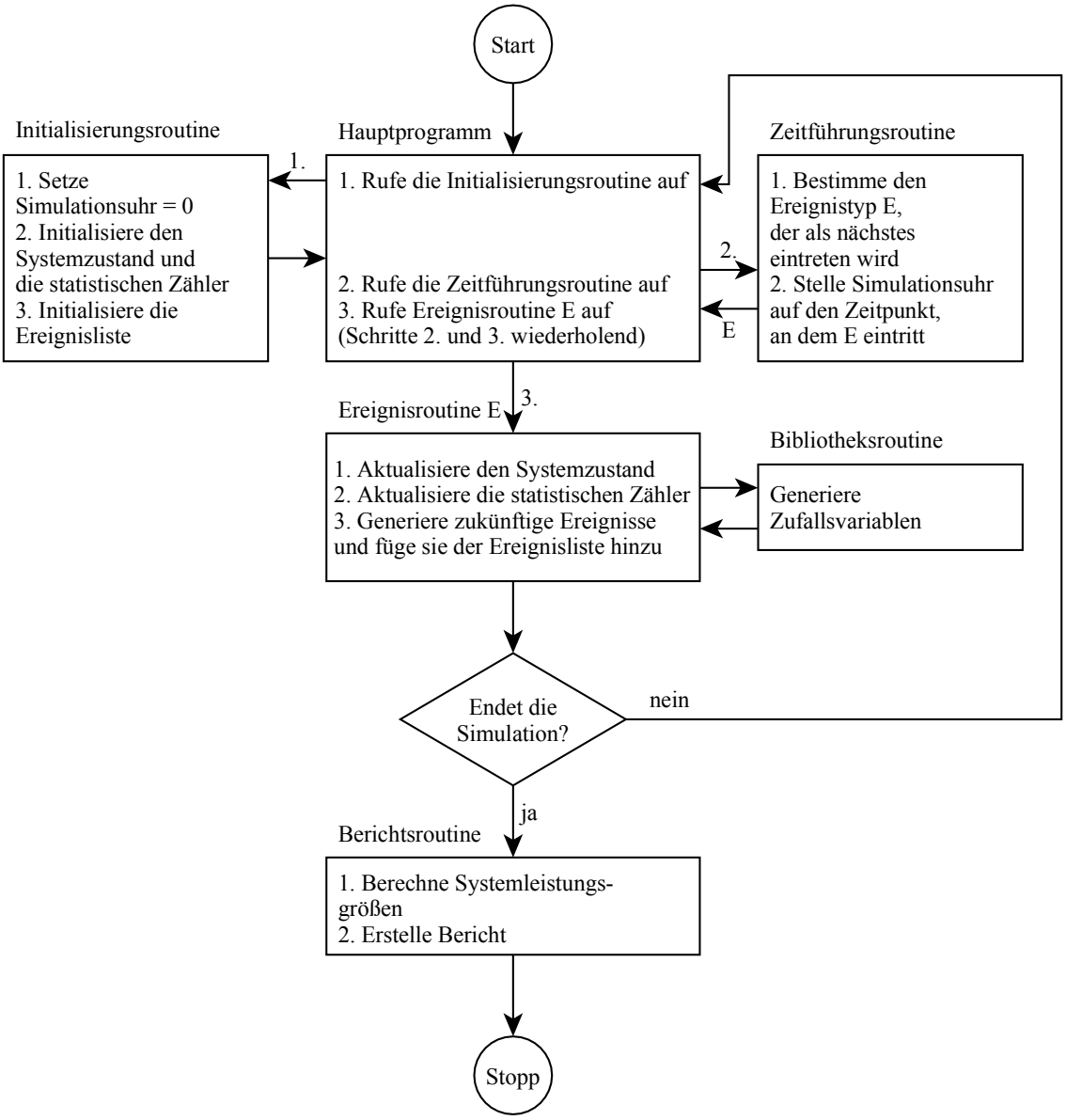

Abbildung 15: Ablauf einer ereignisdiskreten Simulation (Law, 2007)

Die Simulation startet mit dem Hauptprogramm und ruft die Initialisierungsroutine auf. Hier wird die Simulationsuhr auf den Zeitpunkt 0 gesetzt. Dann werden alle Zustandsvariablen, statistischen Zähler sowie die Ereignisliste initialisiert. Im Anschluss wird durch das Hauptprogramm die Zeitführungsroutine aufgerufen. Innerhalb der Zeitführungsroutine wird mit Hilfe der Ereignisliste der Ereignistyp E bestimmt, der als nächstes eintreten wird. Die Simulationsuhr wird auf den Zeitpunkt, an dem Ereignistyp E eintritt, vorgestellt. Danach ruft das Hauptprogramm die Ereignisroutine des Ereignistyps E auf. Diese besteht üblicherweise aus drei Schritten.

1. Der Systemzustand wird entsprechend des eintretenden Ereignistyps E verändert
2. Die Information über das Systemverhalten wird durch die Aktualisierung der statistischen Zähler gesammelt

3. Die Zeiten für zukünftig eintretende Ereignisse werden erzeugt und der Ereignisliste hinzugefügt. Oftmals werden Zufallsgrößen, die durch die Bibliotheksroutinen bereitgestellt werden, verwendet, um zukünftige Ereigniszeitpunkte zu erzeugen.

Ist die Ereignisroutine ausgeführt und es stehen keine weiteren Ereignisse auf der Ereignisliste, wird die Simulation beendet. Das Hauptprogramm ruft die Berichtsroutine auf, um die gewünschten Systemleistungsgrößen (z.B. Gesamtzykluszeit) zu berechnen und einen abschließenden Bericht zu erstellen.

Abbildung 16 und 17 repräsentieren zwei mögliche Ereignistypen, die im Rahmen des Job-Shop-Modells auftreten können und sollen den Ablauf einer Ereignisroutine verdeutlichen. Das Ereignis „Arrival“ stellt die Ankunft eines Jobs an einer Maschine dar, während das Ereignis „Departure“ den Zeitpunkt der Entladung der Maschine repräsentiert.

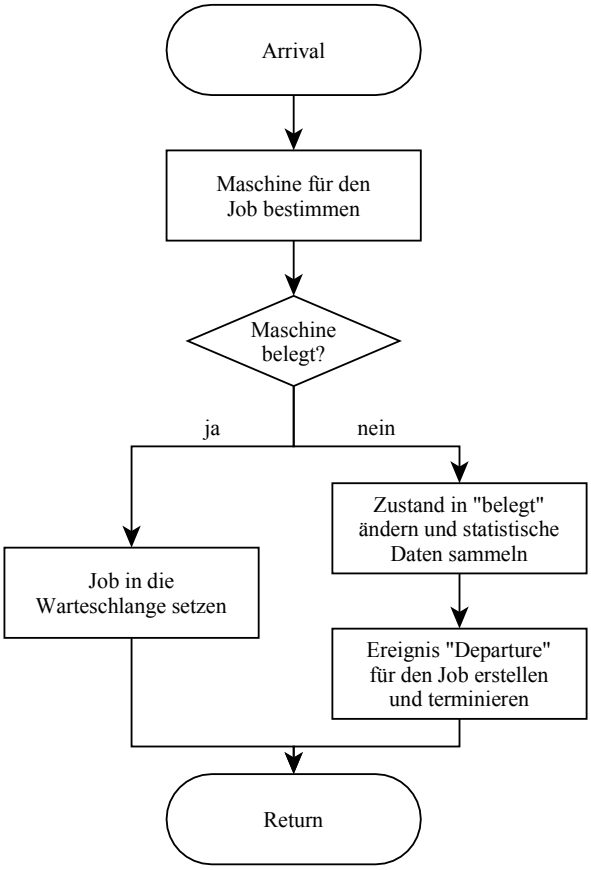

Abbildung 16: Ereignis „Arrival“ (nach Law, 2007)

Ist das nächste eintretende Ereignis vom Typ „Arrival“, wird die dazugehörige Ereignisroutine aufgerufen (s. Abbildung 16). Zuerst wird die Maschine, auf der der Job zu bearbeiten ist, bestimmt. Ist diese Maschine „belegt“, d.h. bearbeitet einen anderen Job, wird der ankommende Job in die Wartschlange gesetzt und die Ereignisroutine ist beendet. Ist

die Maschine hingegen „leer“, wird ihr Zustand in „belegt“ gewechselt und Daten für den statistischen Zähler werden gesammelt. Bevor die Ereignisroutine endet, wird für den Job das Ereignis „Departure“ erstellt und zeitlich terminiert. An dieser Stelle können die Zufallsgrößen der Bibliotheksroutine die zeitliche Planung des Ereignisses „Departure“ beeinflussen, z.B. durch zufällig eintretende Störungen während der Bearbeitung. Ist das Ereignis „Departure“ dieses Jobs das nächste eintretende Ereignis (zwischenzeitlich können andere Ereignisse eintreten, z.B. Ankunft eines anderen Jobs an einer anderen Maschine), wird die Ereignisroutine „Departure“ aufgerufen (s. Abbildung 17).

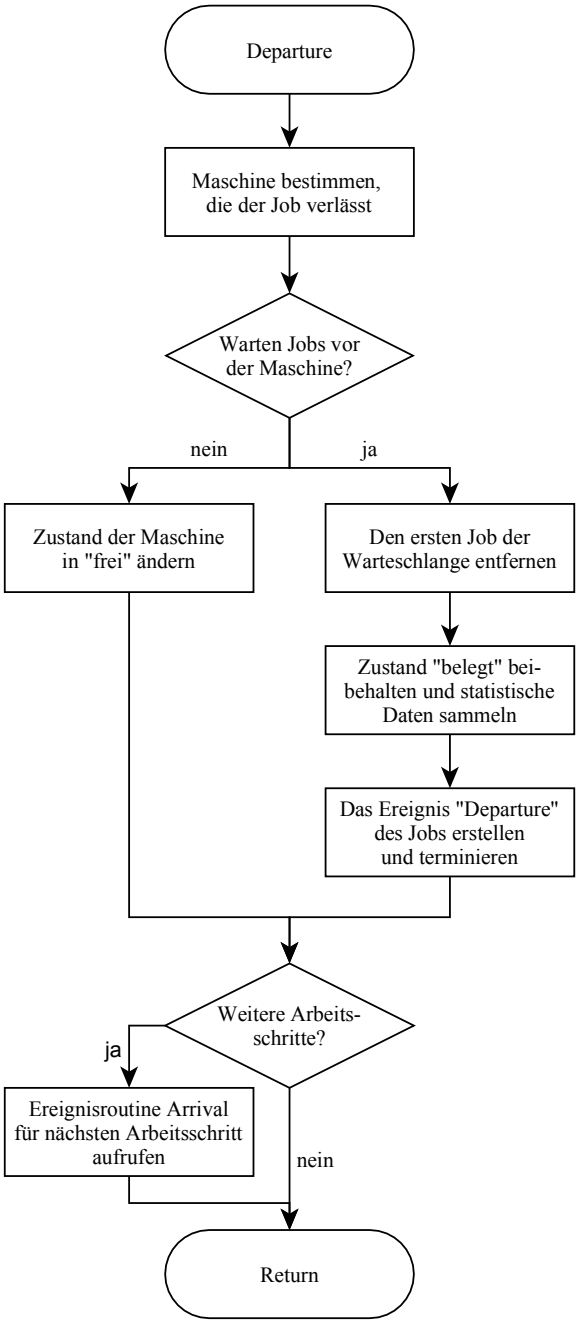

Abbildung 17: Ereignis „Departure“ (nach Law, 2007)

Zu Beginn der Ereignisroutine wird die Maschine bestimmt, die der Job verlässt. Ist die Warteschlange vor dieser Maschine leer, wird der Zustand der Maschine in „frei“ geändert. Warten hingegen Jobs auf die Bearbeitung vor dieser Maschine, wird der erste Job der Wartschlange entfernt. Der Zustand der Maschine bleibt „belegt“ und es werden Daten für den statistischen Zähler gesammelt. Anschließend wird das „Departure“-Ereignis des ersten Jobs aus der Warteschlange, auch hier mit möglichem Einfluss der Zufallsgröße, zeitlich terminiert. Müssen keine weiteren Arbeitsschritte des Jobs, der die Maschine verlässt, ausgeführt werden, endet die Ereignisroutine. Ansonsten wird die Ereignisroutine „Arrival“ für den nächsten Arbeitsschritt des Jobs aufgerufen und die Ereignisroutine „Departure“ endet.

Durch den in Abbildung 15 beschriebenen Simulationsdurchlauf kann das Systemverhalten über die Zeit mit zu Beginn festgelegten Stellgrößen untersucht werden. Im Rahmen eines Simulationsexperiments (s. Abbildung 18) werden nun die Stellgrößen des Simulationsmodells bei jedem Simulationslauf systematisch verändert, um das Verhalten der Zielgrößen, in Abhängigkeit der Stellgrößen, zu ergründen (vgl. Carson und Maria, 1997).

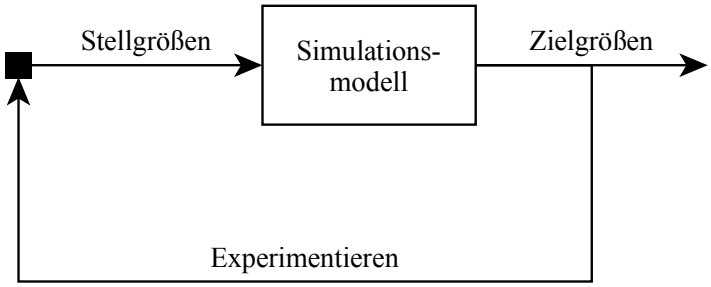

Abbildung 18: Simulationsexperiment (Pidd, 1998)

Ziel ist es, auf Basis der Ergebnisbewertung des Simulationsexperiments eine optimale Stellgrößenkonfiguration zu finden, sodass eine oder mehrere Zielvorgaben optimal erfüllt werden (vgl. Wenzel, 2000).

Da die Simulation selbst keine optimierenden Komponenten enthält (vgl. VDI 3633 Blatt 1), können mit ihrer Hilfe lediglich verschiedene Lösungsalternativen bewertet werden (vgl. Wenzel, 2000). Um die optimale Stellgrößenkonfiguration für eine Zielvorgabe bestimmen zu können, muss die Simulation folglich mit einer Optimierungsmethode gekoppelt werden (s. Abbildung 19) (vgl. Carson und Maria, 1997).

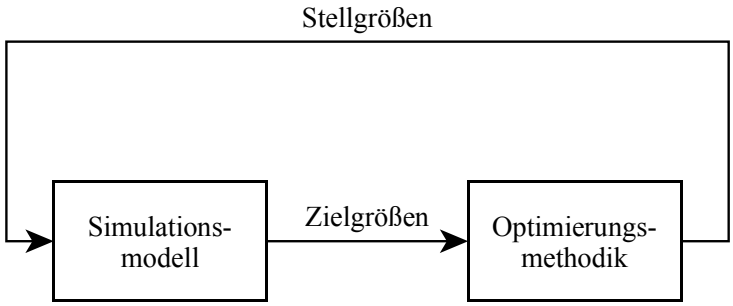

Abbildung 19: Kopplung Simulation und Optimierung (Carson und Maria, 1997)

Die Optimierungsmethode übernimmt in diesem Zusammenhang, basierend auf den Zielgrößen, die Variation der Stellgrößen, um zu einer optimalen Konfiguration zu gelangen.

3.5.4 Optimierungsmethoden zur Kopplung mit der Simulation

Fu (2002) identifiziert vier Optimierungsmethoden, die im Rahmen der simulationsbasierten Optimierung aktueller Gegenstand der Forschung sind. Tabelle 7 stellt die Hauptmerkmale der Verfahren kurz da.

Tabelle 7: Optimierungsmethoden zur simulationsbasierten Optimierung

Methode	Hauptmerkmale
Stochastic Approximation (Gradient-based Approaches)	- Gradient-based Approaches legen die Suchrichtung in Abhängigkeit des Gradienten der Zielfunktion fest (vgl. April et al., 2003) - Stochastic-Approximation-Algorithmen ahmen diese Suche in einer zufallsabhängigen Art und Weise nach (vgl. Fu, 2002)
Response Surface Methodology	- Ziel ist eine approximierte funktionale Beziehung zwischen Stell- und Zielgrößen zu erhalten (vgl. Fu, 2002, Fu et al., 2005) - Dadurch entstehen sog. Metamodelle, die die Zielfunktion entweder für den globalen Lösungsbereich (vgl. Barton, 1998) oder für einen lokalen Bereich charakterisieren (vgl. April et al., 2003)
Random Search	- Nachbarschaftslösung wird zufällig ausgewählt (vgl. April et al., 2003) - Zentrales Element ist die Definition einer geeigneten Nachbarschaft (vgl. Fu 2002)

Fortsetzung Tabelle 7

Methode	Hauptmerkmale
Sample Path Optimization	- Optimiert wird nicht das Ursprungsproblem, sondern ein deterministisches Optimierungsproblem, dass sich diesem annähert (vgl. Andradottir, 1998) - Das deterministische Optimierungsproblem beruht auf Zufallsvariablen (vgl. April et al., 2003) - Um möglichst nah am ursprünglichen Problem zu liegen, muss die Anzahl der Zufallsvariablen sehr groß sein (vgl. Andradottir, 1998)

Den oben beschriebenen Verfahren wird in der Fachliteratur große Aufmerksamkeit geschenkt (vgl. Fu, 2002, Andradottir, 1998, April et al., 2003, Carson und Maria, 1997, Fu, M. 2015, Fu et al., 2005), jedoch findet keine dieser Methoden praktische (kommerzielle) Anwendung (vgl. April et al., 2003). Andradottir (1998) liefert hierfür zwei Gründe.

„Although simulation optimization has received a fair amount of attention from the research community in recent years, the current methods generally require a **considerable amount of technical sophistication** on the part of the user, and they often require a **substantial amount of computer time** as well“ (Andradottir, 1998, S. 156).

In kommerziellen Simulationsanwendungen werden hauptsächlich Metaheuristiken zur Optimierung eingesetzt (vgl. April et al., 2003), denn sie sind leicht verständlich und liefern hinreichend gute Lösungen in kurzer Zeit (s. Kapitel 3.3). Abbildung 20 zeigt die Funktionsweise der simulationsbasierten Optimierung mit einer Metaheuristik als Optimierungsmethode.

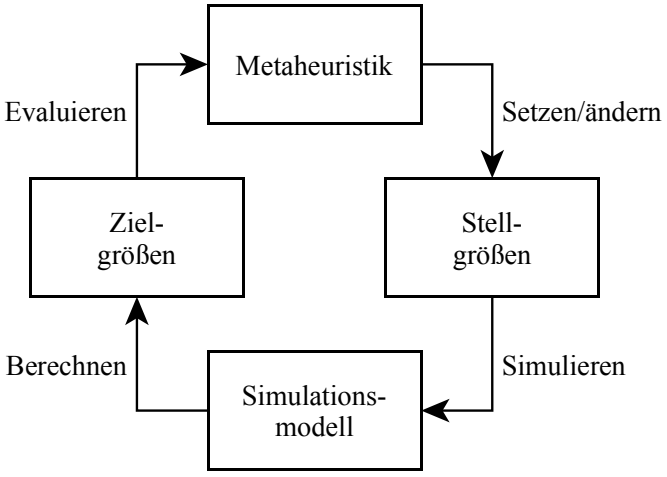

Abbildung 20: Simulationsbasierte Optimierung mit einer Metaheuristik (Weigert et al., 2009)

Zu Beginn des Optimierungsprozesses wird eine Stellgrößenkonfiguration durch die Heuristik selbst oder ein Eröffnungsverfahren (z.B. Prioritätsregelverfahren) bestimmt (s. Kapitel 3.2.4). Das Simulationsmodell wird mit den Stellgrößen ausgeführt und das Systemverhalten kann durch die Berechnung der Zielgröße beurteilt werden. Die Metaheuristik evaluiert den Wert der Zielgröße und verändert die Stellgrößen nach den Regeln der angewandten Heuristik für den nächsten Simulationsdurchlauf. Während des Optimierungsprozesses kommunizieren die Simulation und der Suchalgorithmus ausschließlich über die Stell- und die Zielgrößen (vgl. Weigert et al., 2009).

Im Rahmen des Optimierungszyklus kommen oftmals Metamodelle zum Einsatz, die voraussichtlich schlechtere Lösungen (im Vergleich zur aktuell besten Lösung) herausfiltern sollen, bevor ein Simulationsdurchlauf startet (vgl. April et al., 2003). In Abbildung 21 ist der Zusammenhang zwischen Heuristik, Metamodell und Simulationsmodell illustriert. Wird die Zielgröße einer möglichen Lösung anhand des Metamodells als schlecht eingeschätzt, wird die Lösung verworfen. Andernfalls wird sie zur Simulation freigegeben und ihre genaue Zielgröße wird berechnet.

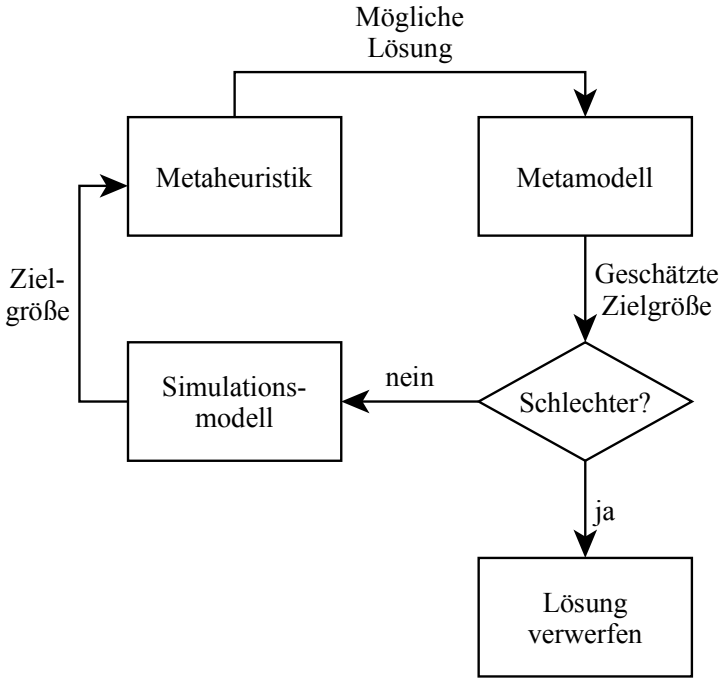

Abbildung 21: Simulationsbasierte Optimierung mit einem Metamodell (April et al., 2003)

Nach Laguna und Marti (2002) sind solche Metamodelle sehr wichtig, da Simulationen sehr rechenintensiv sind und die durch Metamodelle eingesparte Rechenzeit dazu genutzt werden kann, den Lösungsraum umfassender nach einer optimalen Lösung zu durchsuchen. Ein weiteres Merkmal der simulationsbasierten Optimierung ist die Möglichkeit Constraints zu spezifizieren, die die Zulässigkeit von Lösungen definieren (vgl. April et al., 2003). Basieren Constraints nur auf Stell- oder Zielgrößen, kann die Zulässigkeit einer

Lösung vor oder nach dem Simulationslauf überprüft werden (vgl. April et al., 2003). Beziehen sich die Constraints nur auf Stellgrößen, kann auch hier durch den Ausschluss einer unzulässigen Lösung vor dem Simulationslauf wertvolle Zeit gespart und stattdessen zur Lösungssuche benutzt werden.

3.5.5 Derzeitige Konzepte zur Kopplung von Optimierung und Simulation

Die Interaktion zwischen der Optimierungsmethode und der (ereignisdiskreten) Simulation lässt sich in zwei Varianten aufteilen (vgl. März und Krug, 2011):

1. Sequentielle Interaktion
2. Hierarchische Interaktion

Sind die Optimierungsmethoden und die Simulation sequentiell miteinander gekoppelt, ist die vorhergehende Phase vollständig ausgeführt, bevor die nachfolgende Phase startet (s. Abbildung 22 u 23). In Abbildung 22 ist die Optimierung der Simulation vorgelagert. In diesem Zusammenhang wird die Realität mit einem einfachen mathematischen Modell dargestellt, welches die Berechnung des Optimums ermöglicht. Da das Modell nicht alle Bedingungen und Restriktionen berücksichtigt, wird die Lösung mit der nachfolgenden Simulation überprüft (vgl. März und Krug, 2011).

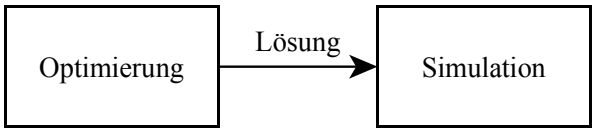

Abbildung 22: Simulation zur Überprüfung der Machbarkeit (März und Krug, 2011)

Ist die Simulation hingegen der Optimierung vorgelagert, stellt die Simulation nach vollständiger Ausführung Eingangsgrößen bereit, die zum Start der Optimierungsmethode benötigt werden (s. Abbildung 23).

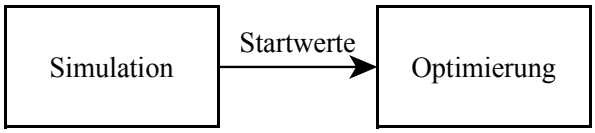

Abbildung 23: Simulation zur Startwertbereitstellung (März und Krug, 2011)

Sind die Simulation und die Optimierung hierarchisch miteinander verknüpft, wird die jeweils untergeordnete Komponente durch die Führungskomponente aufgerufen (s. Abbildung 24 und 25). Im Gegensatz zur sequentiellen Verknüpfung wird eine Phase immer wieder unterbrochen, um notwendige Ergebnisse von der untergeordneten Komponente zu erhalten.

In Abbildung 24 wird im Rahmen eines Simulationsexperiments die Optimierung aufgerufen, um eine gegebene Aufgabestellung zu lösen. Die Optimierungsmethode liefert die geforderten Ergebnisse an die Simulation zurück und diese wird fortgesetzt.

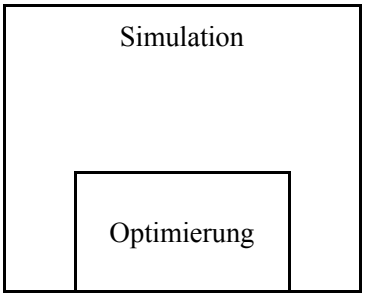

Abbildung 24: Optimierung innerhalb der Simulation (nach März und Krug, 2011)

Wird die Simulation durch die Optimierung aufgerufen, dient der Simulationsdurchlauf der Bewertung der durch die Optimierungsmethode bereitgestellten Lösung (s. Abbildung 25). Unter Berücksichtigung dieser Bewertung wird die Lösungssuche fortgesetzt und eine andere Lösungsalternative der Simulation zur Bewertung übergeben.

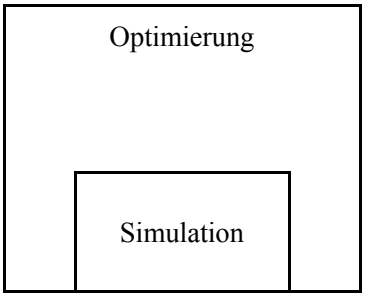

Abbildung 25: Simulation zur Bewertung (nach März und Krug, 2011)

3.5.6 Forschungsvorhaben im Bereich der Kopplung von Optimierung und Simulation

In einem umfassenden Vergleich von Theorie und Praxis der simulationsbasierten Optimierung zeigt Fu (2002) die unterschiedlichen Entwicklungen auf und identifiziert dabei einige Schwachstellen, die durch zukünftige Forschung behoben werden sollten. Viele Defizite beziehen sich auf Systeme mit zufallsabhängigem Charakter, aber dennoch lassen sie sich zum Teil auf deterministische Systeme, wie im Fall der primalen statisch-deterministischen Ablaufplanung, übertragen.

Werden die aktuellen kommerziellen Anwendungen zur simulationsbasierten Optimierung betrachtet, so ist eine Überlegenheit der Simulation gegenüber der Optimierungsmethode zu erkennen, die lediglich ein Unterprogram der Simulation darstellt (s. Abbildung 26).

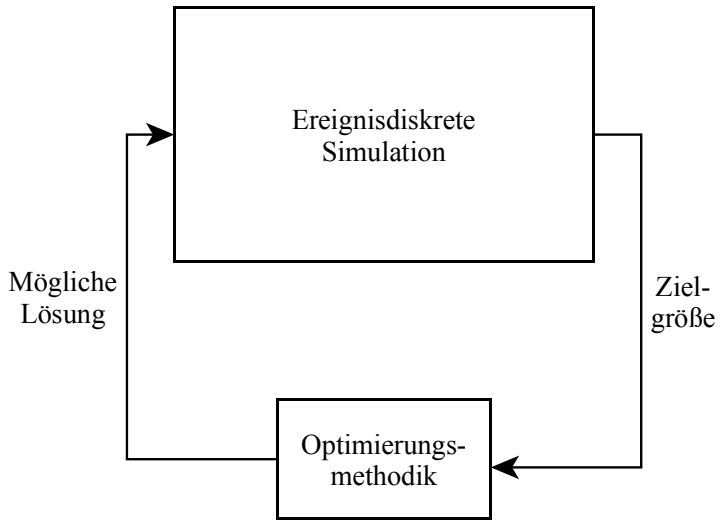

Abbildung 26: Optimierungsmethode als Unterprogramm der Simulation (Fu, 2002)

Aus praktischer Sicht steht dies allerdings in einem Widerspruch. Hier liegt der Fokus auf der Optimierung, also dem Generieren von möglichen Lösungen (vgl. Fu, 2002). Die Simulation dient lediglich zur Bewertung der Lösungsalternativen, d.h. Berechnung der Zielgröße (vgl. Fu, 2002). Abbildung 27 stellt diesen Sachverhalt dar und illustriert ebenfalls die höhere Rechenintensität der Optimierungsmethode im Vergleich zur Simulation.

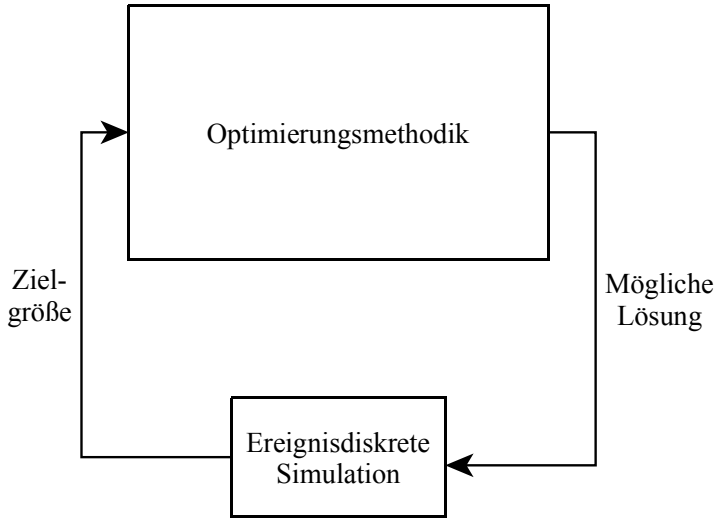

Abbildung 27: Praktische Sicht auf simulationsbasierte Optimierung (Fu, 2002)

Damit dieser Gegensatz beseitigt werden kann, stellt Fu (2002) folgende Anforderungen an die zukünftige simulationsbasierte Optimierung:

- Allgemeingültigkeit: Die Optimierungsmethode soll in der Lage sein, viele verschiedene Probleme zu lösen.
- Transparenz: Mit einer einfachen Konfiguration der Optimierungsmethode soll der mathematische Anspruch an den Nutzer sinken. In diesem Zusammenhang sei noch erwähnt, dass ein zu hoher mathematischer Anspruch oftmals ein Hemmnis

für die Anwendung von Simulationswerkzeugen darstellt (vgl. März und Weigert, 2011).

- Hohe Dimensionalität: Die implementierten Algorithmen sollen in der Lage sein auch Probleme mit hohen Dimensionen zu lösen.
- Effizienz: Die Simulation und Optimierung sollen optimal zusammenwirken, sodass Rechnerressourcen effizienter genutzt werden können und so die Lösung von größeren Problemen ermöglicht wird.

Während die Allgemeingültigkeit, Transparenz und hohe Dimensionalität ausschließlich von der ausgewählten Optimierungsmethode abhängig ist, kann die Effizienz durch konzeptuelle Maßnahmen verbessert werden. Zur Effizienzsteigerung sollen nach Fu (2002) vor allem unnötige Simulationsdurchläufe vermieden und einflussreiche Stellgrößen identifiziert werden.

4 Kopplung der Tabu-Suche mit der ereignisdiskreten Simulation zur Lösung des Job-Shop-Problems

In diesem Kapitel wird ein neues Konzept zur Kopplung der Tabu-Suche von Nowicki und Smutnicki (1996) – im Folgenden nur noch als Tabu-Suche bezeichnet – mit der ereignisdiskreten Simulation entwickelt, um die in Kapitel 3.5.6 aufgeführten Anforderungen zu erfüllen. Abbildung 28 bildet die Ausgangssituation des Kapitels und illustriert die Zusammenhänge zwischen den identifizierten Anforderungen und den einzelnen Komponenten der simulationsbasierten Optimierung.

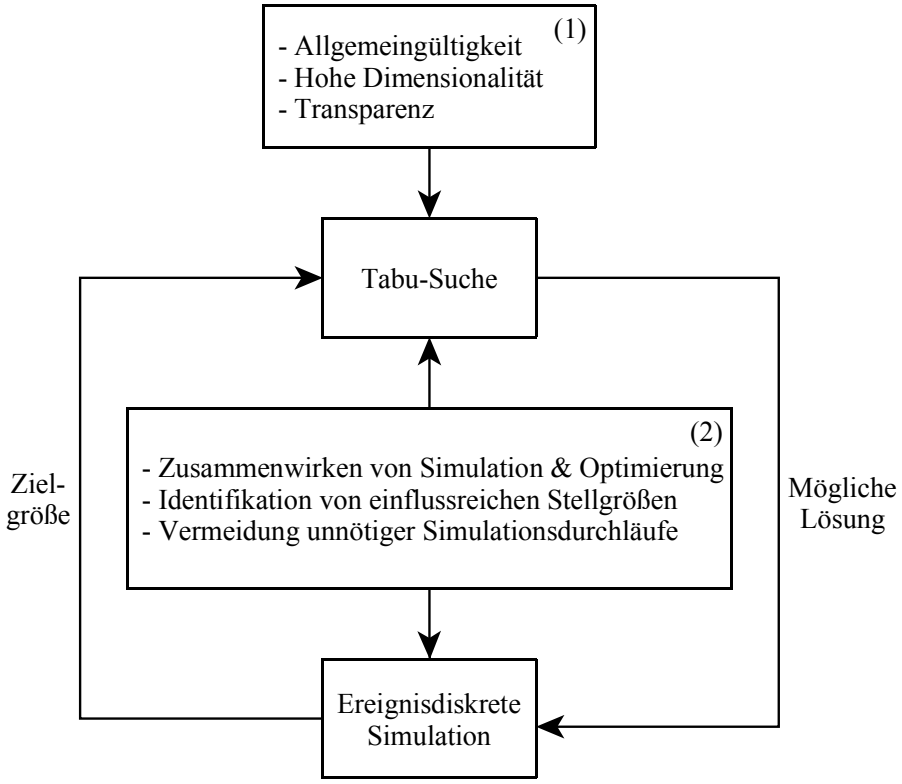

Abbildung 28: Funktionale Anforderungen an das neue Konzept

In Abbildung 28 ist zu erkennen, dass sich nicht alle Anforderungen nur auf die Kopplung von Simulation und Optimierung beziehen (2), sondern auch auf die Tabu-Suche (1). Zuerst wird deshalb geprüft, ob die Tabu-Suche auch den Anforderungen, die nicht das Konzept der Kopplung betreffen, gerecht werden kann. Erst im Anschluss wird das neue Konzept erläutert. Die abschließende Validierung prüft, ob und inwieweit die Anforderungen der Praxis mit dem neuen Konzept erfüllt werden können. Gleichzeitig soll die Validierung aber auch mögliche Schwachstellen aufdecken und somit die Elemente identifizieren, denen im Rahmen zukünftiger Forschung besondere Aufmerksamkeit zuteilwerden sollte.

4.1 Eignung der Tabu-Suche zur Kopplung mit der Simulation

Die in Abbildung 28 unter (1) dargestellten Anforderungen können nicht alle direkt von der Optimierungsmethode erfüllt werden. Das Ablaufplanungsproblem, das ausgewählt wurde, um die Lösungsqualität und -performance der Heuristik zu beurteilen, spielt hierbei eine ebenso wichtige Rolle. Die hohe Dimensionalität und Transparenz sind ausschließlich von der ausgewählten Optimierungsmethode abhängig, während die Allgemeingültigkeit der Heuristik indirekt durch das zur Evaluation benutzte Ablaufplanungsproblem beeinflusst wird.

4.1.1 Hohe Dimensionalität

Zur Lösung des Job-Shop-Problems wurde in Kapitel 3.3 eine geeignete Heuristik gesucht. Als Grundlage für die Evaluation dienten die Ergebnisse verschiedener Benchmarktests. Der Fokus lag dabei einerseits auf Problemen mit besonders hohen Dimensionen und andererseits auf von Vaessens, R. J. M. et al. (1996) als schwer bezeichneten Problemen. Die Tabu-Suche konnte nicht für alle schweren Probleme optimale Lösungen erzielen, erreichte aber eine akzeptable Abweichung vom Optimum und lieferte die Ergebnisse in einer sehr hohen Geschwindigkeit. Im Zusammenhang mit der simulationsbasierten Optimierung ist aber vor allem die Fähigkeit, Probleme mit hohen Dimensionen zu lösen, von großer Bedeutung. Die Tabu-Suche erzielte für viele Probleme mit Dimensionen von bis zu (100×20) teilweise optimale Lösungen oder die besten oberen Schranken. Die im Vergleich sehr hohe Lösungsgeschwindigkeit der Tabu-Suche bei den 13 schweren Problemen lässt erwarten, dass auch für noch größere Probleme hinreichend gute Lösungen in kürzester Zeit bereitgestellt werden können. Das Kriterium einer hohen Dimensionalität wird von der Tabu-Suche somit erfüllt.

4.1.2 Transparenz

Werden die Forschungsanstrengungen im Bereich der simulationsbasierten Optimierung bzgl. der Optimierungsmethode und die in kommerzieller Software implementierten Verfahren betrachtet, so ist zu erkennen, dass keine der Methoden, die in der Forschung große Aufmerksamkeit erfährt, in der Praxis angewandt wird (s. Kapitel 3.5.4). Dies liegt vor allem in dem hohen mathematischen Anspruch, den die Optimierungsmethoden, die in der Forschung entwickelt werden, an den Nutzer stellen. Damit sich eine Optimierungsmethode für den Einsatz in der simulationsbasierten Optimierung eignet, sollte sie keine hohen fachlichen Ansprüche an den Nutzer stellen. Folglich sollte die Konfiguration der Heuristik keine tiefgründigen mathematischen Verständnisse erfordern und die

Vorgehensweise der Heuristik sollte demnach möglichst transparent sein, sodass jede Operation des Verfahrens zu jedem Zeitpunkt nachvollzogen werden kann.

Obwohl dieses Kriterium nicht die Auswahl der Heuristik in Kapitel 3.3 beeinflusste, denn hier stand vor allem die Performance und Lösungsqualität im Vordergrund, kann die Tabu-Suche diese Anforderung erfüllen. Die Tabu-Suche besteht aus nur zwei Hauptkomponenten, der Tabu-Liste und der Transformationsregel, die die Lösungssuche maßgeblich steuern. Die Funktionsweise der Tabu-Liste wie auch die von Nowicki und Smutnicki (1996) verwendete Transformationsregel zur Definition der Nachbarschaft erfordert kein tiefgründiges Verständnis. Hier werden lediglich zwei Operationen, die auf dem kritischen Pfad liegen, miteinander vertauscht (s. Abbildung 11). Ist dem Anwender bewusst, wie die Länge der Tabu-Liste die Breiten- und Tiefensuche beeinflusst, gibt es keine weiteren fachlichen Herausforderungen, die bei dem Einsatz der Heuristik auftreten können.

4.1.3 Allgemeingültigkeit

Im Gegensatz zur hohen Dimensionalität und Transparenz, kann die Forderung nach Allgemeingültigkeit von der Tabu-Suche nicht direkt erfüllt werden. Wie in Kapitel 3.2.6 unter dem Stichwort „No Free Lunch“ bereits diskutiert, ist keine Heuristik in der Lage nur sehr gute Lösungen für die unterschiedlichsten Probleme zu erzielen. Der Anspruch auf eine allgemeingültige Anwendbarkeit der Optimierungsmethode fordert allerdings genau diese Fähigkeit, zu der eigentlich (noch) keine Heuristik imstande ist. Folglich lässt allein die Einschränkung auf Ablaufplanungsprobleme die Allgemeingültigkeit der Heuristik für andere Probleme, die in Produktion und Logistik auftreten können, nicht zu. Um dennoch der Forderung nach Allgemeingültigkeit nachkommen zu können und so zumindest innerhalb der Ablaufplanung die breite Anwendbarkeit der Heuristik zu ermöglichen, wurde ein bestimmtes Ablaufplanungsproblem zur Bewertung und Auswahl der Heuristiken herangezogen.

Das Job-Shop-Problem gilt als ein sehr generelles Problem, das die Grundlage für viele weitere Probleme der Ablaufplanung bildet (s. Kapitel 2.5). Da die Tabu-Suche das Problem in Anbetracht der Lösungsqualität und Performance sehr gut lösen kann (s. Kapitel 3.3), sollte sie auch qualitativ gute Ergebnisse in kurzer Zeit für andere Ablaufplanungsprobleme erzielen können, die auf dem Job-Shop-Problem basieren. Demzufolge kann zumindest innerhalb der Ablaufplanung ein gewisser Grad an Allgemeingültigkeit sichergestellt werden.

Die Tabu-Suche erfüllt nicht nur die Anforderungen, die im Zusammenhang mit der Suche nach einer optimalen Lösung für Ablaufplanungsprobleme bestehen (s. Kapitel 3.3), sondern sie kann auch den Forderungen, die an die Optimierungsmethode im

Rahmen der Kopplung von Simulation und Optimierung gestellt werden (s. Abbildung 28), nachkommen. Aufgrund dieser hervorragenden Eigenschaften wird die Tabu-Suche in dieser Arbeit zur simulationsbasierten Optimierung von Ablaufplanungsproblemen eingesetzt und übernimmt im neuen Konzept die Suche nach Lösungsalternativen.

4.2 Konzeptentwicklung zur Kopplung der Tabu-Suche mit der ereignisdiskreten Simulation

In Kapitel 4.1 wurde gezeigt, dass die Tabu-Suche die Anforderungen, die an die Optimierungsmethode im Kontext der simulationsbasierten Optimierung gestellt werden, erfüllt. Nun gilt es, den Anforderungen an die Kopplung von Simulation und Optimierung, wie sie in Abbildung 28 unter (2) aufgeführt sind, mit einem neuen Konzept gerecht zu werden. Zunächst wird das neue Konzept sehr allgemein und als Ganzes betrachtet, um einen groben Überblick über die Hauptelemente und ihre Beziehung zueinander zu erhalten. Im Anschluss wird das zentrale Element des neues Konzeptes detailliert betrachtet, damit die Funktionsweise und Besonderheiten verdeutlicht werden können.

4.2.1 Hauptkomponenten des neuen Konzeptes

In Abbildung 29 ist das kreierte Konzept zur Kopplung von Simulation und Optimierung grob skizziert. Der Hauptunterschied zu den Kopplungsmethoden, die in Kapitel 3.5.5 vorgestellt wurden, ist dennoch direkt zu erkennen. Während bei diesen Methoden das Optimierungsverfahren und die ereignisdiskrete Simulation direkt miteinander kommunizieren, verläuft die Kommunikation in dem neuen Konzept indirekt und ausschließlich über den im Rahmen dieser Arbeit entworfenen Filter. In der folgenden Erläuterung wird der Filter vorerst als Black-Box betrachtet, bevor er im nächsten Kapitel detailliert vorgestellt wird.

Die simulationsbasierte Optimierung beginnt mit einer zulässigen Lösung, die z.B. durch ein Eröffnungsverfahren bereitgestellt wurde. Die Tabu-Suche startet, basierend auf der Ausgangslösung, mit dem Suchprozess und erzeugt durch die Anwendung der spezifischen Transformationsregel von Nowicki und Smutnicki (1996) eine Menge an Nachbarschaftslösungen. Jede dieser Nachbarschaftslösungen stellt eine potentielle Lösung des Optimierungsproblems dar. Um die Nachbarschaftslösungen miteinander vergleichen zu können, muss für jede Lösung die Zielgröße, in diesem Fall die Gesamtzykluszeit, berechnet werden. Hierfür übergibt die Tabu-Suche die mögliche Lösung an den Filter.

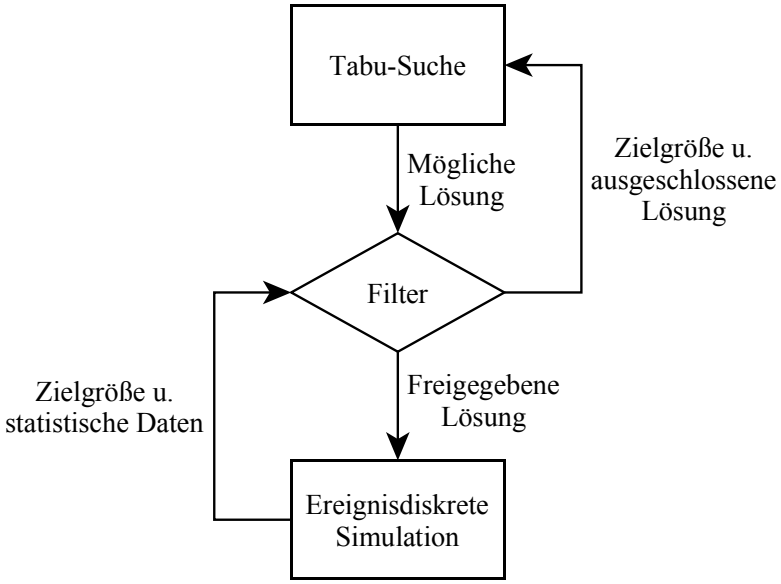

Abbildung 29: Konzept zur Kopplung von Simulation und Optimierung

Im Gegensatz zu den Verfahren in Kapitel 3.5.5 wird in diesem Konzept nicht für jede mögliche Lösung die Zielgröße durch einen Simulationsdurchlauf berechnet. Der Filter bestimmt, ob eine Lösung zur Simulation freigegeben oder ausgeschlossen wird. Gibt der Filter die potentielle Lösung nicht zur Simulation frei, wird ihre Zielgröße nicht berechnet und die Lösungssuche wird durch die Tabu-Suche ohne Berücksichtigung der ausgeschlossenen Lösung fortgesetzt. Andernfalls wird ein Simulationsdurchlauf gestartet und die Zielgröße der Lösung bestimmt. Die statistischen Daten des Simulationsdurchlaufes werden dem Filter übermittelt, sodass er die Daten für zukünftige Entscheidungen berücksichtigen kann. Die Vorgehensweise wiederholt sich so lange, bis entweder für jede Nachbarschaftslösung die Zielgröße berechnet oder die Lösung abgelehnt wurde. Abbildung 30 repräsentiert diesen Sachverhalt mit der Nachbarschaft der Ausgangslösung x. Für jede grüne Nachbarschaftslösung wurde durch einen Simulationsdurchlauf die Zielgröße bestimmt; die roten Lösungen hingegen wurden durch den Filter vom Lösungsprozess ausgeschlossen.

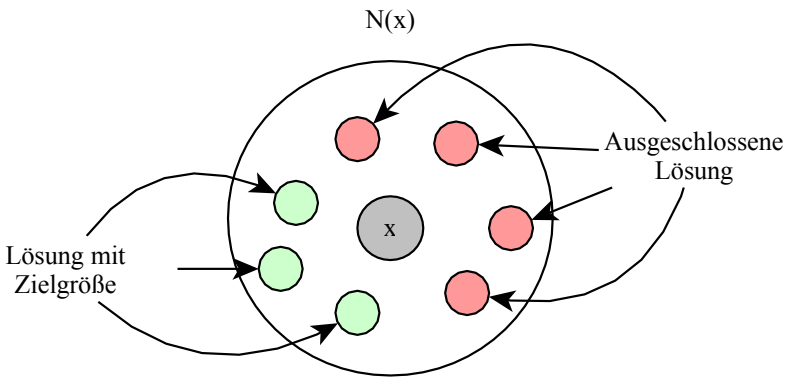

Abbildung 30: Nachbarschaft mit ausgeschlossenen und freigegebenen Lösungen

Wurden alle Nachbarschaftslösungen betrachtet, vergleicht die Tabu-Suche die Zielgrößen der Lösungen miteinander und wählt eine Lösung nach der Best-Fit-Strategie aus. Diese Lösung bildet die Ausgangslösung für die nächste Iteration. Der Suchprozess wird solange ausgeführt, bis das Abbruchkriterium der Tabu-Suche erfüllt ist. Die aktuell beste Lösung ist die „optimale“ Lösung des Ablaufplanungsproblems.

Die Beschreibung der Funktionsweise des Schemas zur Kopplung von Simulation und Optimierung verdeutlicht, dass der Filter das zentrale Element des neuen Konzepts bildet und maßgeblich den Lösungsprozess steuert.

4.2.2 Der Filter als zentrales Element des Konzeptes

Im vorherigen Kapitel wurde bei der Betrachtung des Konzepts aufgezeigt, dass dem Filter eine besonders wichtige Rolle zukommt. Nun soll der Fokus ausschließlich auf den Filter gelegt werden, damit die internen Abläufe erläutert werden können. Bevor die ausführliche Erklärung der Funktionsweise erfolgt, sei noch erwähnt, dass die Tabu-Suche und die ereignisdiskrete Simulation bei der Betrachtung vorerst vernachlässigt werden. Abbildung 31 repräsentiert somit den Filter mit seinen Komponenten und Prozessen.

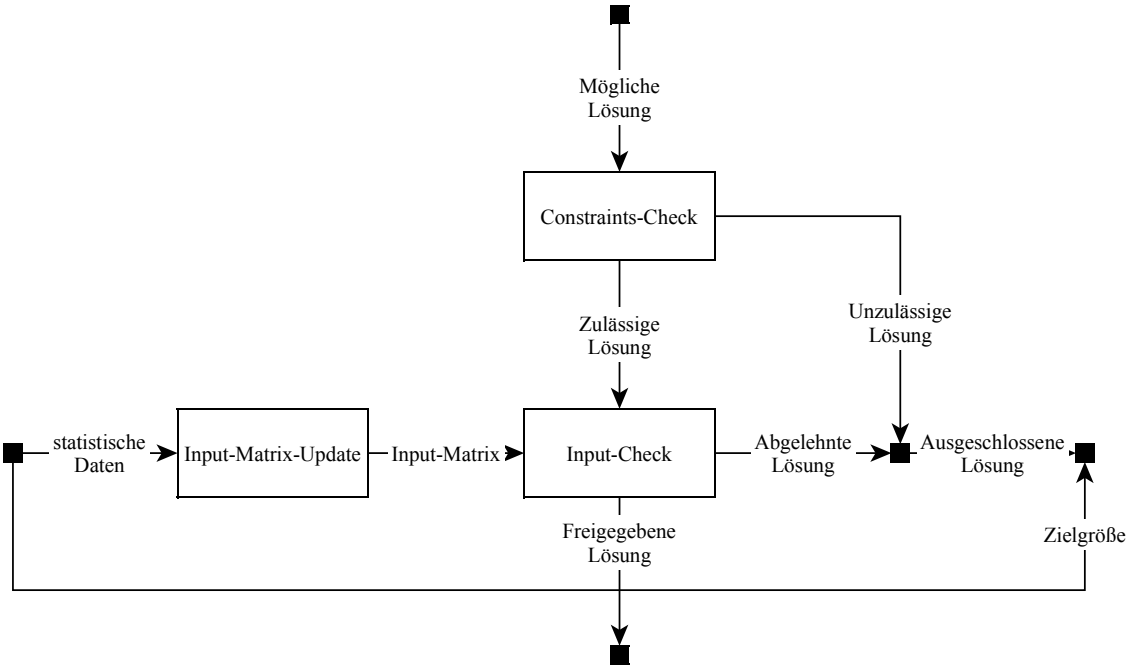

Abbildung 31: Überblick über die Elemente und internen Prozesse des Filters

Dem Filter wird von der Tabu-Suche eine mögliche Lösung übergeben. Innerhalb des *Constraints-Checks* wird überprüft, ob es sich bzgl. des vorliegenden Problems um eine zulässige Lösung handelt. Ist das der Fall, wird die Lösung an den *Input-Check* übergeben. Andernfalls ist die Lösung unzulässig und wird ausgeschlossen. Im Input-Check

wird die Lösung unter Verwendung der *Input-Matrix* auf ihre voraussichtliche Qualität geprüft. Handelt es sich um eine vermutlich schlechte Lösung (z.B. Zielgröße ist größer als ein Grenzwert), wird die Lösung abgelehnt und nicht weiter betrachtet. Ist die Lösung vermutlich gut (z.B. Zielgröße ist kleiner als ein Grenzwert) wird sie freigegeben und der ereignisdiskreten Simulation übergeben. Im Anschluss an die ereignisdiskrete Simulation werden die statistischen Daten dem *Input-Matrix-Update* zur Verfügung gestellt und die Zielgröße der Lösung wird der Tabu-Suche übermittelt.

Nachdem die internen Prozesse des Filters kurz erläutert wurden, ist zu erkennen, dass der Filter aus vier Komponenten besteht, die für die Freigabe oder den Ausschluss von Lösungen verantwortlich sind. In den nachfolgenden Kapiteln werden deshalb die Kernelemente detailliert betrachtet und ausführlich erklärt.

4.2.2.1 Constraints-Check

Der Constraints-Check dient zur schnellen Kontrolle, ob eine vorgeschlagene Lösung auch zulässig ist. Im Rahmen der simulationsbasierten Optimierung repräsentieren die Constraints die ablauforganisatorischen Bedingungen und dürfen nicht verletzt werden (s. Kapitel 3.5.4). Der Constraints-Check prüft folglich, ob die vorgeschlagene Lösung diese Nebenbedingungen nicht verletzt und somit eine zulässige Lösung ist. Abbildung 32 dient zur Veranschaulichung dieses Prinzips.

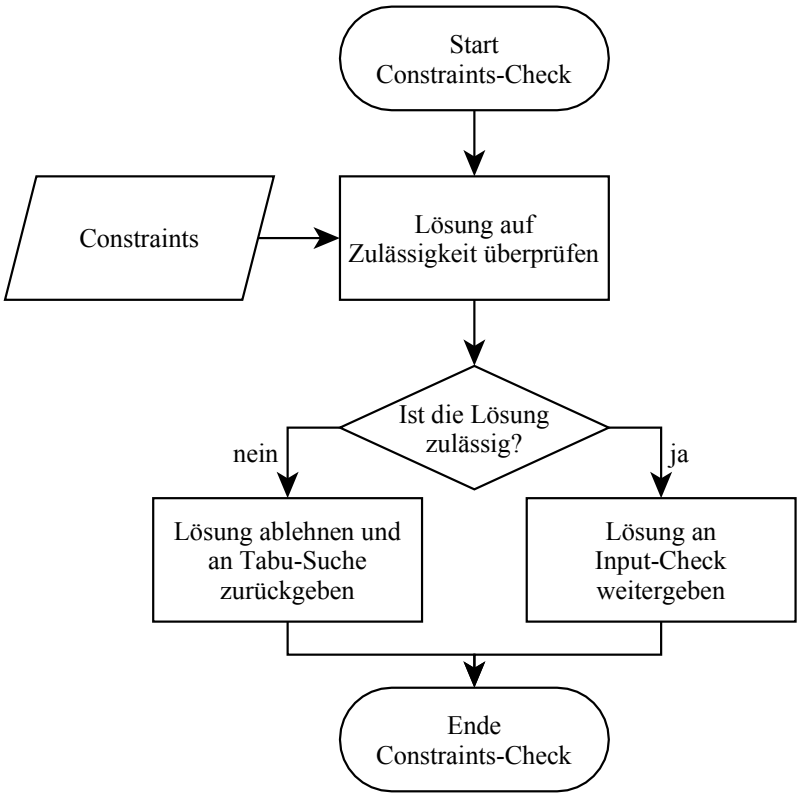

Abbildung 32: Constraints-Check

Mit der Übergabe einer möglichen Lösung von der Tabu-Suche startet der Constraints-Check. Nun wird, basierend auf den entsprechenden Constraints des Job-Shop-Problems, überprüft, ob es sich um eine zulässige Lösung handelt. Die Constraints bestehen hier lediglich aus Reihenfolgebedingungen, die eingehalten werden müssen. Auf diese Weise kann ausgeschlossen werden, dass durch die Transformationsregeln der Tabu-Suche, die auf dem Vertauschen von Elementen basiert, die Maschinenfolgen der Jobs verletzt werden. Die übrigen Constraints, wie z.B. die Ressourcenkapazität, werden durch die entsprechende Modellierung des Systems eingehalten und müssen nicht explizit überprüft werden. Hält der Lösungsvorschlag die Maschinenfolgen nicht ein, wird die Lösung ausgeschlossen und an die Tabu-Suche zurückgegeben. Andernfalls ist die Lösung zulässig und wird an den Input-Check weitergeben.

4.2.2.2 Input-Check

Im Gegensatz zum Constraints-Check, der nur die Integration einer bereits bestehenden Methodik darstellt, ist der Input-Check eine Neuheit. Abbildung 33 stellt den Ablauf des Input-Checks dar und soll die Besonderheiten verdeutlichen.

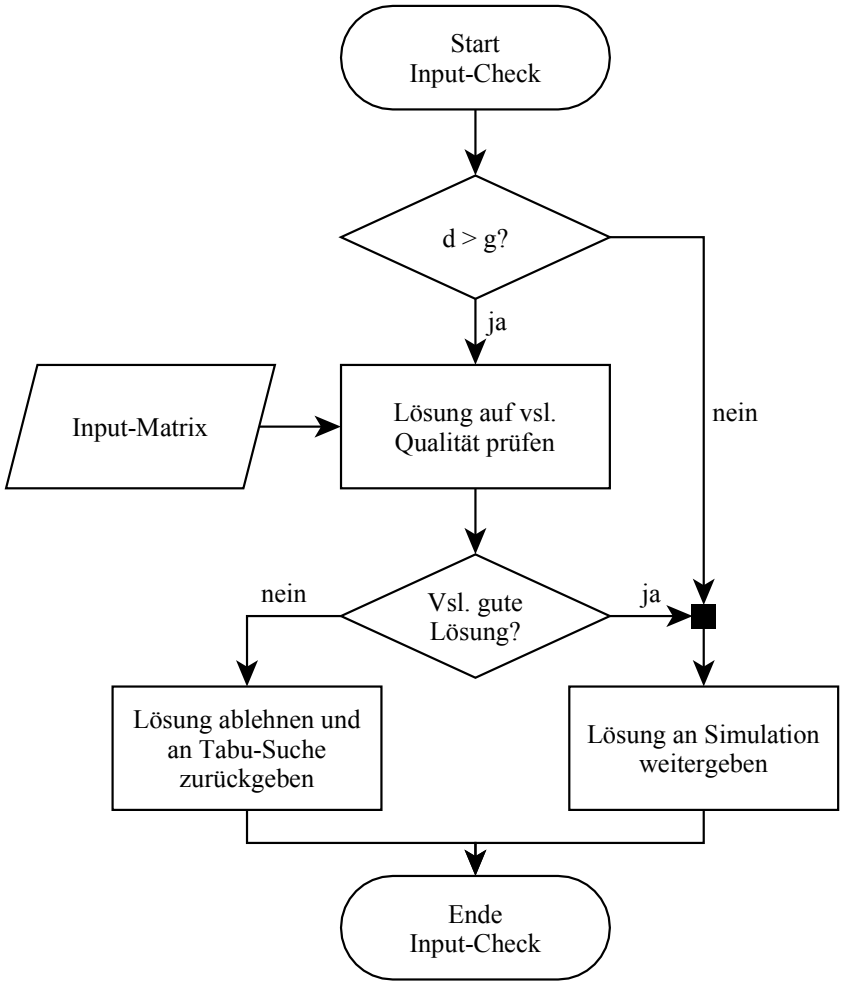

Abbildung 33: Input-Check

Der Input-Check startet mit der Übergabe einer zulässigen Lösung durch den Constraints-Check. Die Lösung wird jedoch nur geprüft, wenn bereits eine gewisse Anzahl an Simulationsdurchläufen d erreicht wurde. Die Variable g definiert die Grenze, die überschritten werden muss, und bestimmt somit die Aktivität des Input-Checks. Wurden noch nicht genügend Simulationsdurchläufe ausgeführt ($d < g$), wird die Lösung ohne Prüfung an die Simulation weitergegeben, um die Gefahr eine falsche Entscheidung aufgrund einer zu geringen Datengrundlage zu treffen, zu reduzieren. Übersteigt die Anzahl der bisherigen Simulationsdurchläufe die festgelegte Grenze ($d > g$), wird die voraussichtliche Qualität der Lösung unter Zuhilfenahme der Input-Matrix überprüft. Die Input-Matrix beinhaltet Informationen über Stellgrößen, die in der Vergangenheit zu schlechten Lösungen geführt haben. Die Input-Matrix wird in Kapitel 4.2.2.4 detailliert betrachtet und spielt zum Verständnis des Input-Checks an dieser Stelle eine untergeordnete Rolle. Lässt die Input-Matrix in Bezug auf die Stellgröße der Lösung Rückschlüsse zu, dass die Lösung voraussichtlich gut ist (z.B. Zielgröße ist kleiner als ein Grenzwert), wird sie zur Simulation freigegeben. Ist die Wahrscheinlichkeit hingegen hoch, dass die Lösung schlecht ist (z.B. Zielgröße ist größer als ein Grenzwert), wird die Lösung von der weiteren Betrachtung ausgeschlossen. Der Input-Check endet folglich entweder mit der finalen Freigabe der Lösung für die Simulation oder mit dem Ausschluss der Lösung.

Abbildung 33 zeigt deutlich, dass die Entscheidung innerhalb des Input-Checks, ob es sich um eine gute oder schlechte Lösung handelt, auf der Input-Matrix basiert. Die Input-Matrix ist somit maßgeblich für die Leistungsfähigkeit des Input-Checks verantwortlich und wird durch das Input-Matrix-Update erstellt.

4.2.2.3 Input-Matrix-Update

Das Input-Matrix-Update hat durch die Erzeugung und Bereitstellung der Input-Matrix großen Einfluss auf den Lösungsprozess und ist somit eine der wesentlichen Komponenten des neuen Konzeptes. Im Folgenden wird zuerst das Input-Matrix-Update ausführlich dargestellt, bevor die Struktur und Inhalte der Input-Matrix anhand eines Beispiels verdeutlicht werden.

In Abbildung 34 ist der Ablauf des Input-Matrix-Updates, das mit dem Ende des Simulationsdurchlaufs startet, illustriert. Wurde für die freigegebene Lösung x' mit Hilfe der Simulation die Zielgröße bestimmt, werden im Anschluss die statistischen Daten an das Input-Matrix-Update übergeben. Innerhalb eines Simulationsdurchlaufs können viele statistische Daten generiert werden, jedoch sind hier nur zwei von besonderem Interesse:

1. Die Zielgröße $z(x')$ der aktuell betrachteten Lösung und
2. der Mittelwert M aller bis jetzt berechneten Zielgrößen.

Die Verwendung des Mittelwerts M zur Beurteilung der Qualität der Lösung hat den Vorteil, dass der Mittelwert keine statische Grenze bildet. Im Verlauf der simulationsbasierten Optimierung passt er sich den Regionen des Lösungsraumes an, sodass in schlechten Regionen ein Großteil aller Lösungen nicht direkt vom Lösungsprozess ausgeschlossen wird. Das reduziert die Gefahr, dass Lösungen unbeachtet bleiben, die im weiteren Verlauf der Lösungssuche zu einer besseren oder vielleicht zur optimalen Lösung geführt hätten.

Anhand der Zielgröße $z(x')$, des Mittelwerts M sowie dem Toleranzwert t wird nun innerhalb des Input-Matrix-Updates die Qualität der Lösung nach einem Simulationsdurchlauf festgestellt. Ist $z(x') > M + t$ wird die Lösung x' als schlecht definiert und nimmt mit ihrer Stellgröße Einfluss auf die Input-Matrix (s. Kapitel 4.2.2.4). Zukünftige Lösungen mit ähnlichen Stellgrößen werden nun innerhalb des Input-Checks als voraussichtlich schlecht betrachtet und vom weiteren Lösungsprozess ausgeschlossen.

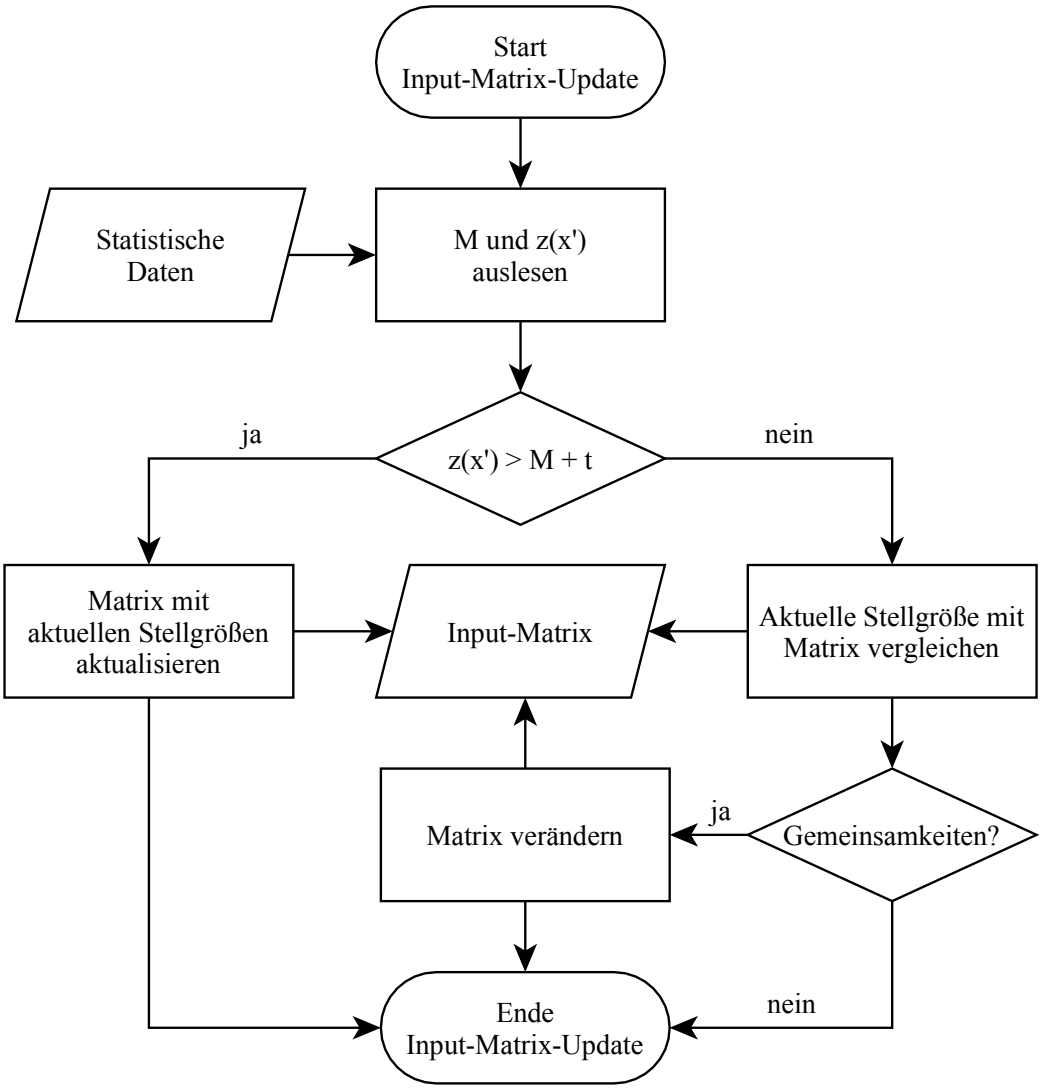

Abbildung 34: Input-Matrix-Update

In diesem Zusammenhang kann auch die Funktion der Variable t verdeutlicht werden. Der Toleranzwert t soll an dieser Stelle verhindern, dass jede Lösung x' mit $z(x') > M$ als schlechte Lösung definiert wird. Vielmehr muss die Zielgröße der Lösung x' auch außerhalb der Toleranz liegen ($z(x') > M + t$), damit die Lösung als schlecht gilt und mit ihrer Stellgröße die Input-Matrix beeinflusst. Der Einfluss einer Lösung, die nur „etwas“ größer als der Mittelwert ist, kann dazu führen, dass zukünftige Lösungen aufgrund von Gemeinsamkeiten ihrer Stellgrößen mit der Input-Matrix vom Lösungsprozess ausgeschlossen werden, obwohl sie aus dem lokalen Optimum herausführen könnten. Denn wie in Kapitel 3.2.5 erläutert und in Abbildung 10 illustriert, müssen oftmals schlechtere Lösungen akzeptiert werden, damit das globale Optimum überhaupt erreicht werden kann.

Wird im Rahmen des Input-Matrix-Updates festgestellt, dass $z(x') < M + t$, wird die Lösung x' als gut definiert. Anschließend wird geprüft, ob die Stellgröße Gemeinsamkeiten mit der Input-Matrix besitzt. Sollte das der Fall sein, wird die Input-Matrix ebenfalls aktualisiert (s. Kapitel 4.2.2.4). Auf diese Weise wird verhindert, dass falsche Strukturen innerhalb der Input-Matrix entstehen und so unbeabsichtigt voraussichtlich gute Lösungen vom Lösungsprozess ausgeschlossen werden.

4.2.2.4 Input-Matrix

Der Ablauf des Input-Checks macht deutlich, dass Entscheidungen, die zum Ausschluss einer Lösung führen, von der Input-Matrix abhängig sind. In diesem Kapitel wird nun erläutert, wie die Stellgrößen die Input-Matrix beeinflussen, sodass bis zum Zeitpunkt $d = g$ (s. Abbildung 33) eine effektive Input-Matrix erzeugt werden kann, die im weiteren Verlauf der simulationsbasierten Optimierung ($d > g$) die Unterscheidung von voraussichtlich guten und schlechten Lösungen ermöglicht.

Im folgenden Beispiel wird ein Job-Shop-Problem der Größe (3×3) betrachtet. Die simulationsbasierte Optimierung startet mit einer Anfangslösung x_1, die durch ein Eröffnungsverfahren bereitgestellt wurde. An dieser Stelle sei noch erwähnt, dass $d < g$ ist und deshalb jede erzeugte Lösung zur Simulation freigegeben wird. Die Stellgröße jeder Lösung ist eine Permutation, die als 3×3-Matrix die Auftragsfolge der jeweiligen Maschine (Zeile 1 – Maschine 1, Zeile 2 – Maschine 2 usw.) repräsentiert. Wie schon in Kapitel 3.2.4 erwähnt, handelt es sich bei Lösungen, die durch Eröffnungsverfahren bereitgestellt wurden, oftmals um Lösungen mit einer erheblichen Abweichung vom Optimum. Deshalb enthält die Input-Matrix, ebenfalls eine 3×3-Matrix, zu Beginn die Stellgröße x_1. Jedes Element der Input-Matrix erhält die Gewichtung $\omega = 0$, da noch keine Stellgröße die Matrix beeinflusst bzw. Elemente der Matrix bestätigt hat.

$$x_1 = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{bmatrix} \tag{4-1}$$

$$\text{IM} = \begin{bmatrix} 1_{\omega=0} & 2_{\omega=0} & 3_{\omega=0} \\ 3_{\omega=0} & 2_{\omega=0} & 1_{\omega=0} \\ 2_{\omega=0} & 1_{\omega=0} & 3_{\omega=0} \end{bmatrix} \tag{4-2}$$

Im Rahmen der Lösungssuche wird nun durch die Tabu-Suche eine neue Lösung x'_1 generiert und zur Simulation freigegeben. Das Input-Matrix-Update definiert die Lösung im Anschluss an den Simulationsdurchlauf als schlecht, sodass x'_1 die Input-Matrix verändert.

$$x'_1 = \begin{bmatrix} 3 & 1 & 2 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{bmatrix} \tag{4-3}$$

Die Veränderung der Input-Matrix basiert, nachdem Übereinstimmungen zwischen der Input-Matrix und der aktuellen Lösung gesucht wurden, auf folgenden Regeln:

1. Elemente der Input-Matrix, die mit Elementen der Stellgröße der schlechten Lösung übereinstimmen, werden verstärkt. Die Gewichtung der übereinstimmenden Elemente wird um 1 erhöht.
2. Die Gewichtung der Elemente der Input-Matrix, die nicht mit Elementen der schlechten Lösung übereinstimmen, wird um 1 reduziert. Wird durch die Reduktion dabei eine Gewichtung negativ, wird das zugehörige Element der Input-Matrix durch das entsprechende Element der aktuellen Lösung ersetzt.

Die Anwendung der Regeln 1. und 2. verändert die Input-Matrix (4-2), sodass sich die aktualisierte Input-Matrix (4-4) ergibt.

$$\text{IM} = \begin{bmatrix} 3_{\omega=0} & 1_{\omega=0} & 2_{\omega=0} \\ 2_{\omega=0} & 3_{\omega=0} & 1_{\omega=1} \\ 3_{\omega=0} & 1_{\omega=1} & 2_{\omega=0} \end{bmatrix} \tag{4-4}$$

Sobald im weiteren Verlauf der simulationsbasierten Optimierung wieder eine Lösung innerhalb des Input-Matrix-Updates als schlecht definiert wird, erfolgt eine erneute Aktualisierung der Input-Matrix (4-4). Die Lösung x'_2 (4-5) stellt in diesem Zusammenhang die nächste schlechte Lösung dar.

$$x'_2 = \begin{bmatrix} 2 & 3 & 1 \\ 3 & 1 & 2 \\ 1 & 3 & 2 \end{bmatrix} \tag{4-5}$$

Nachdem die Gemeinsamkeiten und Unterschiede zwischen der Input-Matrix und der Stellgröße von x'_2 identifiziert wurden, wird die Input-Matrix (4-4) durch die Anwendung der Regeln in die Input-Matrix (4-6) überführt.

$$\mathrm{IM} = \begin{bmatrix} 2_{\omega=0} & 3_{\omega=0} & 1_{\omega=0} \\ 3_{\omega=0} & 1_{\omega=0} & 1_{\omega=0} \\ 1_{\omega=0} & 1_{\omega=0} & 2_{\omega=1} \end{bmatrix} \tag{4-6}$$

Die oben aufgeführten Updates der Input-Matrix beschreiben, wie schlechte Lösungen die Input-Matrix beeinflussen bzw. verändern. Abbildung 34 zeigt aber auch, dass gute Lösungen Einfluss auf die Input-Matrix haben. Um den Einfluss einer guten Lösung auf die Input-Matrix (4-6) zu verdeutlichen, wird das Beispiel mit einer guten Lösung x'_3 (4-7) fortgesetzt.

$$x'_3 = \begin{bmatrix} 2 & 1 & 3 \\ 3 & 2 & 1 \\ 1 & 2 & 3 \end{bmatrix} \tag{4-7}$$

Der Vergleich der Stellgröße von x'_3 (4-7) mit der Input-Matrix (4-6) zeigt, dass Übereinstimmungen vorhanden sind. Im weiteren Verlauf der simulationsbasierten Optimierung sollen allerdings gute Lösungen nicht unbeabsichtigt vom Lösungsprozess ausgeschlossen werden. Deshalb müssen die Stellen der Input-Matrix, an denen Übereinstimmungen existieren, freigegeben werden. Daraus ergibt sich die Input-Matrix (4-8).

$$\mathrm{IM} = \begin{bmatrix} - & 3_{\omega=0} & 1_{\omega=0} \\ - & 1_{\omega=0} & - \\ - & 1_{\omega=0} & 2_{\omega=1} \end{bmatrix} \tag{4-8}$$

Die nächste schlechte Lösung x'_4 (4-9) besetzt die freigewordenen Stellen der Input-Matrix (4-8). Die neuen Elemente werden mit null gewichtet und die anderen Elemente wie gewohnt verändert. (4-10) repräsentiert die durch x'_4 veränderte Input-Matrix.

$$x'_4 = \begin{bmatrix} 2 & 3 & 1 \\ 2 & 3 & 1 \\ 3 & 2 & 1 \end{bmatrix} \tag{4-9}$$

$$\mathrm{IM} = \begin{bmatrix} 2_{\omega=0} & 3_{\omega=1} & 1_{\omega=1} \\ 2_{\omega=0} & 3_{\omega=0} & 1_{\omega=0} \\ 3_{\omega=0} & 2_{\omega=0} & 2_{\omega=0} \end{bmatrix} \tag{4-10}$$

Das Beispiel zeigt, wie die Input-Matrix nach jedem Simulationsdurchlauf entweder mit der Stellgröße einer schlechten Lösung oder ggf. durch Übereinstimmungen mit der Stellgröße einer guten Lösung verändert wird. Nun wird anhand des Beispiels noch erläutert, wie im Rahmen des Input-Checks für $(d > g)$ unter Verwendung der Input-Matrix eine Lösung ausgeschlossen oder zur Simulation freigegeben wird. Hierfür müssen allerdings noch zwei weitere Variablen eingeführt werden. Die Variable s_{IM} repräsentiert die Anzahl an Übereinstimmungen der Input-Matrix mit der Stellgröße der aktuellen Lösung und $\widehat{s_{IM}}$ die Anzahl erlaubter Übereinstimmungen, die dabei nicht überschritten werden darf. Ist $s_{IM} > \widehat{s_{IM}}$, handelt es sich voraussichtlich um eine schlechte

Lösung, was dazu führt, dass sie vom weiteren Lösungsprozess ausgeschlossen wird. Ist $s_{IM} \leq \widehat{s_{IM}}$, ist die Lösung voraussichtlich gut und wird zur Simulation freigegeben.

Dem Input-Check wird die Lösung x'_5 (4-11) übergeben. Diese soll nun mit der Input-Matrix (4-10) überprüft werden. Ob die Lösung x'_5 ausgeschlossen oder freigegeben wird, hängt von der Grenze $\widehat{s_{IM}}$ (4-12) ab.

$$x'_5 = \begin{bmatrix} 1 & \mathbf{3} & 2 \\ 1 & \mathbf{3} & 2 \\ 2 & 1 & 3 \end{bmatrix} \tag{4-11}$$

$$\widehat{s_{IM}} = 6 \tag{4-12}$$

Im Rahmen des Input-Checks wird nun s_{IM} bestimmt. Zwei Elemente von x'_5 stimmen mit zwei Elementen der Input-Matrix (4-10) überein. Da $2 \leq 6$, wird die Lösung an die ereignisdiskrete Simulation übergeben. Die nächste Lösung x'_6 (4-13) hat hingegen 8 übereinstimmende Elemente mit der Input-Matrix (4-10) und wird deshalb vom weiteren Lösungsprozess ausgeschlossen ($8 > 6$).

$$x'_6 = \begin{bmatrix} \mathbf{2} & \mathbf{3} & \mathbf{1} \\ \mathbf{2} & \mathbf{3} & \mathbf{1} \\ \mathbf{3} & \mathbf{2} & 1 \end{bmatrix} \tag{4-13}$$

Das Beispiel zur Funktionsweise der Input-Matrix zeigt, dass die Input-Matrix eine dynamische Komponente innerhalb des neuen Konzeptes darstellt. Nach jedem Simulationsdurchlauf wird die Input-Matrix durch Stellgrößen schlechter Lösungen aktualisiert und Gemeinsamkeiten mit Stellgrößen guter Lösungen werden eliminiert. Wie Abbildung 9 zeigt, entspricht die Suche nach Nachbarschaftslösungen einer Wanderung durch den Lösungsraum. Das bedeutet, dass sich Lösungen, die im Lösungsraum weit voneinander entfernt sind, stark unterscheiden. Eine statische Input-Matrix würde folglich dazu führen, dass die Identifikation von schlechten Lösungen innerhalb einer Region im Lösungsraum zwar möglich ist, aber aufgrund der Verschiedenheit der Lösungen in anderen weitentfernten Regionen scheitert. Das hat zur Folge, dass unbeabsichtigt gute Lösungen von der Lösungssuche ausgeschlossen werden könnten. Ist die Input-Matrix hingegen dynamisch, lässt die kontinuierliche Weiterentwicklung eine Anpassung an die jeweilige Region im Lösungsraum zu. Das ermöglicht eine effektive Unterscheidung von guten und schlechten Lösungen, unabhängig von der Position im Lösungsraum.

Ein weiterer Vorteil der Input-Matrix ist die Flexibilität bezüglich der unterschiedlichen Lösungskodierungen (s. Kapitel 3.2.5.2). Da im Rahmen des Input-Matrix-Updates nur nach Übereinstimmungen in der Struktur zwischen der Input-Matrix und der Stellgröße einer guten oder schlechten Lösung gesucht wird, ist es irrelevant, ob die Stellgröße mit Binärzahlen kodiert ist oder – wie im oben aufgeführten Beispiel – die Stellgröße die

Auftragsfolgen der einzelnen Maschinen repräsentiert. Die Input-Matrix muss lediglich in der gleichen Form wie die Stellgrößen der Lösungen kodiert sein. Das wird erreicht, indem zu Beginn der simulationsbasierten Optimierung der Input-Matrix die Stellgröße der Ausgangslösung, die durch das Eröffnungsverfahren bereitgestellt wurde, zugewiesen wird.

Nachdem alle Komponenten des Filters ausführlich dargestellt und erläutert wurden, wird im Folgenden der Einfluss der verschiedenen Variablen, die in den letzten Kapiteln aufgeführt wurden, identifiziert.

4.2.3 Parametrisierung des Filters

Der Input-Check, das Input-Matrix-Update sowie die Input-Matrix enthalten Variablen, die das Verhalten des neuen Konzeptes maßgeblich steuern und somit Einfluss auf den Erfolg der simulationsbasierten Optimierung haben. In den folgenden Ausführungen werden vor allem die Funktionen und Aufgaben der verschieden Variablen dargelegt.

4.2.3.1 Aktivität des Input-Checks

Der Input-Check dient zur Überprüfung der voraussichtlichen Qualität von Lösungen. Die Überprüfung findet allerdings nur dann statt, wenn schon eine gewisse Anzahl an Simulationsdurchläufen ($d > g$) erreicht wurde (s. Abbildung 33). Die Grenze g bestimmt folglich die Aktivität des Input-Checks und sollte so groß sein, dass die Input-Matrix ausreichend Informationen über Stellgrößen, die zu schlechten Lösungen führen, erhalten kann. Je höher die Gewichtung der Elemente der Input-Matrix ist, d.h. je häufiger Gemeinsamkeiten zwischen schlechten Lösungen gefunden wurden, desto höher ist die Wahrscheinlichkeit, dass schlechte Lösungen im Input-Check identifiziert werden können. Die logische Konsequenz ist nun, ein großes g zu wählen. Das führt aber zu einer langen Inaktivität des Input-Checks, sodass zunächst jede Lösung zur Simulation freigegeben wird und erst sehr spät Lösungen vom Lösungsprozess ausgeschlossen werden. Das steht aber im Widerspruch zu der Anforderung an das Konzept, dass unnötige Simulationsdurchläufe vermieden werden sollen. Ist g hingegen zu klein, werden aufgrund mangelnder Informationen innerhalb des Input-Checks keine zuverlässigen Entscheidungen getroffen. Dadurch entsteht die Gefahr, dass unbeabsichtigt gute Lösungen vom Lösungsprozess ausgeschlossen werden.

4.2.3.2 Toleranzrahmen des Input-Matrix-Updates

Nachdem die Zielgröße einer Lösung durch die Simulation berechnet wurde, wird im Input-Matrix-Update bestimmt, ob es sich um eine gute oder schlechte Lösung handelt (s. Abbildung 34). In diesem Zusammenhang spielt der Toleranzwert t eine wichtige Rolle. In Kombination mit dem Mittelwert M bildet er eine Toleranzzone, damit nicht

jede Lösung, die nur größer als der Mittelwert M ist, Einfluss auf die Input-Matrix nimmt (s. Kapitel 4.2.2.3). Mit dem Start der simulationsbasierten Optimierung ist es nun wichtig, dass t nicht zu klein, aber auch nicht zu groß ist. Ein zu kleiner Toleranzwert kann dazu führen, dass eine Lösung, deren Zielgröße nur minimal größer als der Mittelwert ist, als schlecht definiert wird. Aus dem daraus resultierenden Einfluss auf die Input-Matrix werden ähnliche Lösungen im Input-Check ausgeschlossen, obwohl sie im weiteren Verlauf evtl. zu einer guten Lösung geführt hätten (s. Abbildung 10). Ist der Toleranzwert hingegen zu groß, wird ein Großteil der Lösungen als gut definiert und zu wenig schlechte Lösungen haben Einfluss auf die Input-Matrix, sodass keine Gemeinsamkeiten in den Stellgrößen schlechter Lösungen festgestellt werden können. Das wiederum würde eine effektive Unterscheidung von guten und schlechten Lösungen im Input-Check verhindern.

4.2.3.3 Anzahl der Übereinstimmungen mit der Input-Matrix

Die Stellgröße einer Lösung wird im Rahmen des Input-Checks mit der Input-Matrix verglichen. Übersteigt die Anzahl an Gemeinsamkeiten s_{IM} die Grenze $\widehat{s_{IM}}$, wird die Lösung nicht zur Simulation freigegeben und vom weiteren Lösungsprozess ausgeschlossen. Die Grenze $\widehat{s_{IM}}$ legt folglich fest, ab wieviel übereinstimmenden Elementen eine Lösung voraussichtlich eine schlechte Lösung ist. Auch bei dieser Variablen muss eine geeignete Balance gefunden werden. Ist die Grenze $\widehat{s_{IM}}$ zu groß, werden nur wenige Lösungen von der Simulation ausgeschlossen. Die Lösungen, die ausgeschlossen werden, sind mit hoher Wahrscheinlichkeit schlechte Lösungen. Ist $\widehat{s_{IM}}$ hingegen sehr klein, werden sehr viele Lösungen ausgeschlossen, sodass auch evtl. gute Lösungen von der weiteren Betrachtung ausgeschlossen werden.

4.2.4 Die ereignisdiskrete Simulation und Tabu-Suche innerhalb des neuen Konzeptes

Zum Abschluss der Ausführungen des neuen Konzeptes zur Kopplung der ereignisdiskreten Simulation mit der Tabu-Suche wird aus Gründen der Vollständigkeit noch dargestellt, wie die ereignisdiskrete Simulation die für das Konzept erforderlichen Daten bereitstellt und wie eine ausgeschlossene Lösung die weitere Lösungssuche beeinflusst.

Wird eine Lösung x' durch den Filter zur Simulation freigegeben, wird das Simulationsmodell mit der Stellgröße der Lösung x' ausgeführt. Während der Simulation werden die erforderlichen Ereignisroutinen aufgerufen (s. Abbildung 15). Innerhalb der jeweiligen Ereignisroutine wird der statistische Zähler mit Daten über den aktuellen Systemzustand aktualisiert (s. Abbildung 16 & 17). Ist die Simulation beendet, berechnet die Berichtsroutine, basierend auf den Daten des statistischen Zählers, die Zielgröße $z(x')$ und den

Mittelwert M. Die zwei Größen werden dem Filter übergeben, damit die Qualität der Lösung x' beurteilt werden kann (s. Abbildung 29 & 31). Die berechnete Zielgröße $z(x')$ wird auch der Tabu-Suche übergeben und der Lösung x' zugeordnet (s. Abbildung 35).

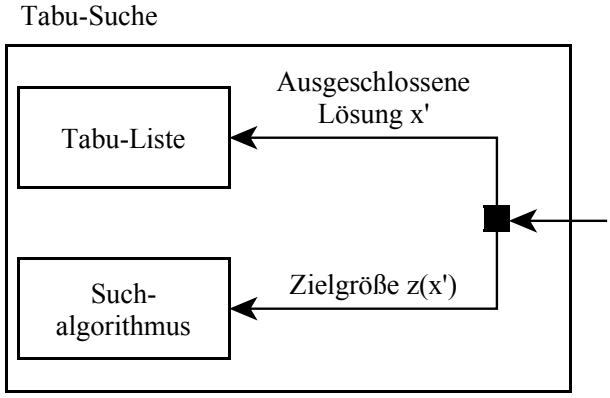

Abbildung 35: Tabu-Suche im Detail

Wird eine Lösung x' hingegen nicht zur Simulation freigegeben und vom Filter von dem weiteren Lösungsprozess ausgeschlossen, wird die Lösung der Tabu-Suche zurückgegeben (s. Abbildung 29 & 31). Damit sie auch im weiteren Verlauf (zunächst) nicht mehr als Lösung in Frage kommt, wird die ausgeschlossene Lösung auf die Tabu-Liste gesetzt (s. Abbildung 35). Die ausgeschlossene Lösung könnte auch als Constraint formuliert und durch den Constraints-Check von der weiteren Lösungssuche ausgeschlossen werden. In diesem Fall ist die Lösung endgültig vom restlichen Lösungsprozess ausgeschlossen. Wird die ausgeschlossene Lösung hingegen auf die Tabu-Liste gesetzt, hat das einen entscheidenden Vorteil. Ist die Tabu-Liste voll, d.h. jede Stelle der Tabu-Liste ist durch eine tabuisierte oder ausgeschlossene Lösung belegt, wird durch eine hinzukommende Lösung die erste („älteste“) Lösung auf der Tabu-Liste wieder freigegeben. Die anderen Lösungen wandern um eine Stelle nach oben, sodass am Ende der Tabu-Liste Platz für die neue Lösung entsteht. Das Prinzip ist in Kapitel 3.4 veranschaulicht. Das führt dazu, dass eine Lösung, die ausgeschlossen wurde, nach einer gewissen Zeit wieder als Lösung in Frage kommen kann. Hier liegt der Vorteil gegenüber der Formulierung als Constraint. Eine Lösung, die in einer Region des Lösungsraumes aufgrund einer voraussichtlich schlechten Zielgröße vom Lösungsprozess ausgeschlossen wurde, könnte in einer anderen Region des Lösungsraumes zu einem späteren Zeitpunkt eine gute Lösung darstellen. Wäre die Lösung als Constraint formuliert worden, gäbe es keine Möglichkeit, die Lösung auszuwählen. Stand die Lösung auf der Tabu-Liste und ist zwischenzeitlich wieder freigegeben, kann sie in den folgenden Iterationen von der Tabu-Suche ausgewählt werden.

Nachdem das Gesamtkonzept und die Kernkomponenten in den letzten Kapiteln ausführlich dargestellt wurden, soll mit der folgenden Validierung geprüft werden, ob das neue Konzept alle in Kapitel 3.5.6 identifizierten funktionalen Anforderungen erfüllt.

4.3 Validierung des Konzeptes

Bevor die Validierung des Konzeptes erfolgt, werden nochmals alle Komponenten des Konzeptes und ihre Beziehungen zueinander betrachtet. Hierfür wurde der Filter aus Abbildung 29 „aufgelöst“, sodass das neue Konzept in der gleichen Form, in der auch die Methoden zur Kopplung in Kapitel 3.5.5 illustriert wurden, dargestellt werden kann (vgl. Abbildung 27). Das erleichtert die anschließende Validierung, da die wesentlichen Unterschiede und die damit verbundenen Auswirkungen direkt erkennbar sind. Vor der Validierung wird das Konzept in Abbildung 36 kurz erläutert, um die Wirkzusammenhänge aller in Kapitel 4.2 dargestellten Komponenten zu verdeutlichen.

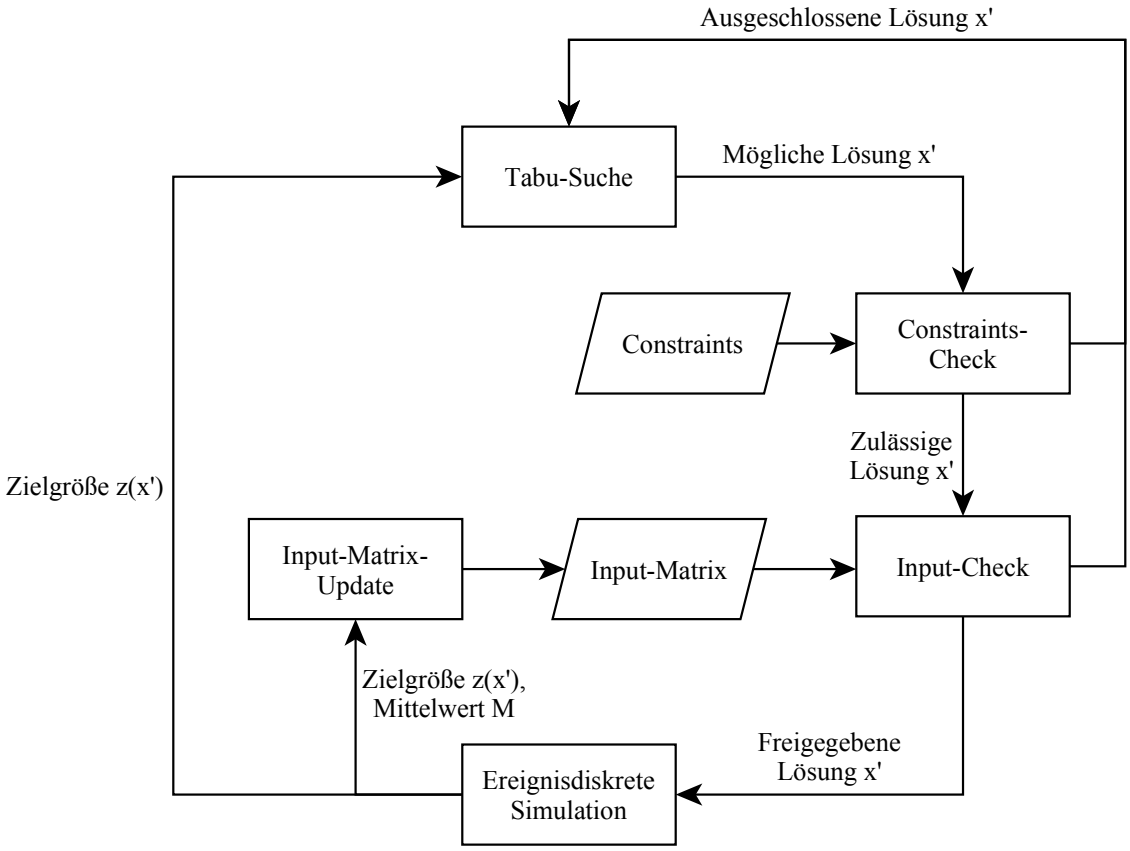

Abbildung 36: Konzept mit aufgelöstem Filter

Die Tabu-Suche generiert durch die Anwendung der spezifischen Transformationsregeln eine mögliche Lösung x'. Die Lösung x' wird basierend auf den formulierten Constraints des Job-Shop-Problems auf die Zulässigkeit überprüft. Ist die Lösung unzulässig, wird sie ausgeschlossen und auf die Tabu-Liste gesetzt. Andernfalls wird sie dem Input-Check

übergeben. Wird innerhalb des Input-Checks mit Hilfe der Input-Matrix festgestellt, dass die Lösung voraussichtlich gut ist, wird sie zur Simulation freigegeben. Sollte die Input-Matrix Schlüsse zulassen, die darauf deuten, dass die Lösung wahrscheinlich schlecht ist, wird sie ebenfalls ausgeschlossen und auf die Tabu-Liste gesetzt. Nachdem die Simulation mit der Stellgröße der Lösung x' ausgeführt wurde, werden die Zielgröße $z(x')$ und der Mittelwert M dem Input-Matrix-Update übergeben, um die Input-Matrix zu aktualisieren. Die Zielgröße $z(x')$ wird auch der Tabu-Suche übermittelt und der zugehörigen Lösung x' zugeordnet. Die Vorgehensweise wiederholt sich so lange, bis das Abbruchkriterium der Tabu-Suche erfüllt wird.

Die folgende Validierung des Konzeptes soll nun zeigen, wie und durch welche Komponenten des neuen Konzeptes die funktionalen Anforderungen aus Kapitel 3.5.6 in Abbildung 26 unter (2) erfüllt werden.

4.3.1 Integration der ereignisdiskreten Simulation in den Lösungsprozess

Betrachtet man die Konzepte, die in Kapitel 3.5.5 vorgestellt wurden, ist zu erkennen, dass die Optimierungsmethode dort nur für die Lösungssuche und die Simulation lediglich für die Zielgrößenberechnung verantwortlich ist. Es existiert demnach eine imaginäre Grenze zwischen den beiden Komponenten, denn jede Komponente nimmt nur die eigenen Aufgaben wahr. Vergleicht man nun das neue Konzept (s. Abbildung 36) mit Abbildung 27, fällt auf, dass die ereignisdiskrete Simulation nun nicht mehr ausschließlich die Zielgrößen für die Tabu-Suche berechnet. Die statistischen Daten $(z(x'), M)$ des Simulationsdurchlaufs werden auch dem Input-Matrix-Update übermittelt und ermöglichen so die Qualitätsbeurteilung der Lösungen. In Abhängigkeit der Lösungsqualität wird die Input-Matrix entsprechend der vorgestellten Regeln aktualisiert und beeinflusst im Verlauf der simulationsbasierten Optimierung den Ausschluss oder die Freigabe der Lösungen im Input-Check. Somit lässt die ereignisdiskrete Simulation durch die Bereitstellung der statistischen Daten für das Input-Matrix-Update eine Unterscheidung von voraussichtlich guten und schlechten Lösungen im Input-Check zu (s. Abbildung 33). Auf diese Weise wird die Simulation indirekt am Lösungsprozess beteiligt. Das Zusammenwirken der drei Komponenten (Input-Matrix-Update, Input-Matrix, Input-Check) ermöglicht folglich das Zusammenwirken der ereignisdiskreten Simulation mit der Tabu-Suche.

4.3.2 Identifikation einflussreicher Stellgrößen

Eine weitere Anforderung, die mit Hilfe des neuen Konzeptes erfüllt werden soll, ist die Identifikation von einflussreichen Stellgrößen. Hierfür muss zuerst die Frage beantwortet

werden, welche Stellgrößen im Rahmen des Job-Shop-Problems als einflussreich zu bezeichnen sind. Die simulationsbasierte Optimierung des Job-Shop-Problems hat das Ziel, eine Lösung mit einer möglichst kurzen Gesamtzykluszeit zu finden. Stellgrößen sind in diesem Zusammenhang einflussreich, wenn sie große Auswirkungen auf die Länge der Gesamtzykluszeit haben. In dieser Arbeit werden deshalb alle Stellgrößen als einflussreich bezeichnet, die nicht zu einer kurzen Gesamtzykluszeit führen. Damit eine Unterscheidung in lange und kurze Gesamtzykluszeiten überhaupt möglich ist, wird im Input-Matrix-Update eine Grenze definiert (s. Kapitel 4.2.2.3). So ist die Identifikation von Stellgrößen, die zu langen Gesamtzykluszeiten (größer als die definierte Grenze) führen, möglich. Im Rahmen des Input-Matrix-Updates wird mit der Input-Matrix nach Gemeinsamkeiten in solchen Stellgrößen gesucht, um immer wieder vorkommende Strukturen und Elemente zu identifizieren (s. Kapitel 4.2.2.4). Da sich Stellgrößen, die im Lösungsraum weit voneinander entfernt sind, stark unterscheiden, wird die Input-Matrix kontinuierlich durch schlechte Stellgrößen aktualisiert. Auf diese Weise wird die Identifikation von einflussreichen Stellgrößen zu jedem Zeitpunkt der simulationsbasierten Optimierung sichergestellt.

4.3.3 Vermeidung unnötiger Simulationsdurchläufe

In Abbildung 36 ist zu erkennen, dass das neue Konzept zur Kopplung von Simulation und Optimierung auch die Möglichkeit bietet, Lösungen von der weiteren Betrachtung auszuschließen. Werden Lösungen vom Lösungsprozess ausgeschlossen, werden sie nicht zur Simulation freigegeben und ihnen wird folglich keine Zielgröße zugewiesen. Ziel ist es, die Lösungen auszuschließen, die von vornherein nicht für die optimale Lösung in Frage kommen und bei denen die Berechnung ihrer Zielgröße mit der Simulation somit unnötig wäre. Um zu verhindern, dass die optimale Lösung oder die Lösungen, die zu ihr führen würden, ausgeschlossen werden, müssen die Lösungen sorgfältig ausgewählt werden. Für diesen Zweck wurden der Constraints- und Input-Check in das Konzept integriert. Im Constraints-Check werden alle Lösungen ausgeschlossen, die für das vorliegende Problem unzulässig sind (s. Kapitel 4.2.2.1). Das stellt zwar keine Besonderheit dar, dient aber trotzdem dem Ziel, unnötige Simulationsdurchläufe zu vermeiden. Der Input-Check hingegen ist eine elementare Komponente des neuen Konzepts und steht im direkten Zusammenhang mit der Identifikation von einflussreichen Stellgrößen (s. Kapitel 4.3.2.). Innerhalb des Input-Checks werden mit der Input-Matrix Lösungen, die aufgrund der Ergebnisse vorheriger Simulationsdurchläufe voraussichtlich schlechte Lösungen sind, identifiziert (s. Kapitel 4.2.2.4). Der Ausschluss dieser Lösungen im Input-Check verhindert somit einen unnötigen Simulationsdurchlauf. Durch die direkte Rückführung der ausgeschlossenen Lösung an die Tabu-Suche (s. Abbildung 36) wird die eingesparte Zeit genutzt, um den Lösungsraum nach weiteren Lösungen zu durchsuchen.

Aus diesem Grund wurde in Kapitel 3.3 die Tabu-Suche von Nowicki und Smutnicki (1996) dem Verfahren von Balas und Vazacopoulos (1998) vorgezogen, da die eingesparte Zeit durch die hohe Lösungsgeschwindigkeit der Tabu-Suche effizienter genutzt werden kann und die Wahrscheinlichkeit somit ansteigt, die optimale Lösung zu finden.

Die Validierung des Konzeptes zeigt, dass die funktionalen Anforderungen, die in Kapitel 3.5.6 an das neue Konzept zur Kopplung von Simulation und Optimierung gestellt wurden, erfüllt werden können. Tabelle 8 gibt einen abschließenden Überblick über die Anforderungen und die Komponenten, die zu der Erfüllung dieser beitragen.

Tabelle 8: Übersicht Anforderungen – Komponenten

Anforderungen	Verantwortliche Komponenten
Zusammenwirken der Simulation & Optimierung	- Input-Matrix-Update - Input-Matrix - Input-Check
Identifikation einflussreicher Stellgrößen	- Input-Matrix-Update - Input-Matrix
Vermeidung unnötiger Simulationsdurchläufe	- Constraints-Check - Input-Check

4.3.4 Mögliche Schwachstellen des neuen Konzeptes

Auch wenn die Validierung zeigt, dass die funktionalen Anforderungen, die in Kapitel 3.5.6 an das neue Konzept gestellt wurden, erfüllt werden, kann das Konzept nicht ohne Weiteres in der Praxis angewandt werden. Vielmehr sollte das Konzept als Grundlage für die zukünftige Forschung im Rahmen der simulationsbasierten Optimierung dienen. Da das Risiko besteht, während des Lösungsprozesses einige Lösungen fälschlicherweise auszuschließen (z.B. die optimale oder Lösungen, die zur optimalen Lösung geführt hätten), sollte der Fokus hauptsächlich auf den Komponenten, die für die Identifikation von einflussreichen Stellgrößen und den Ausschluss von Lösungen verantwortlich sind, liegen (s. Tabelle 8). Das in dieser Arbeit entwickelte Konzept stellt die für die zukünftige Forschung benötigten Parameter bereit (s. Kapitel 4.2.3), sodass das Verhalten des Gesamtkonzeptes mit verschiedenen Parameterkombinationen untersucht werden kann. Die folgende Tabelle zeigt die möglichen Schwachstellen der einzelnen Komponenten auf und identifiziert die Parameter, die diese beeinflussen. Des Weiteren werden Fragestellungen hervorgebracht, die in Zukunft weiter erforscht werden sollten. Der Constraints-

Check ist an dieser Stelle nicht mit aufgeführt, da die Einhaltung der Nebenbedingungen in jedem Fall gesichert werden muss.

Tabelle 9: Mögliche Schwachstellen des Konzeptes

Input-Matrix-Update		
Schwachstelle	Fragestellung	Parameter
Grenze zur Beurteilung der Qualität der Lösungen	- Eignet sich der Mittelwert zur Beurteilung der Qualität der Lösungen? - Hat die Toleranz Einfluss auf das Ergebnis? Wenn ja: - Wie groß sollte der Toleranzwert sein?	- M - t
Input-Matrix		
Schwachstelle	Fragestellung	Parameter
Finden von Gemeinsamkeiten in den Stellgrößen	- Werden genügend Gemeinsamkeiten in den Stellgrößen schlechter Lösungen gefunden? Wenn ja: - Wie hoch sollte die Anzahl an Übereinstimmungen sein, bevor die Lösung ausgeschlossen wird? - Existiert dabei ein optimales Verhältnis von $\widehat{s_{IM}}$ zur Dimension der Stellgröße? - Passt sich die Input-Matrix der Position im Lösungsraum ausreichend an?	- s_{IM} - $\widehat{s_{IM}}$
Input-Check		
Schwachstelle	Fragestellung	Parameter
Ausschluss von Lösungen	- Kann trotz des Ausschlusses von Lösungen eine optimale Lösung erzielt werden? - Kann die Lösungszeit durch den Ausschluss von Lösungen reduziert werden? - Welche Anforderung stellt das Ausschließen von Lösungen an die Länge der Tabu-Liste? - Hat die Dauer der anfänglichen Inaktivität des Input-Checks Einfluss auf das Ergebnis? Wenn ja: - Wie oft sollte die Input-Matrix aktualisiert werden, bevor der Input-Check aktiviert wird?	- d - g - l

Für die Untersuchung der verschiedenen Fragestellungen in Tabelle 9 empfiehlt der Autor die von Vaessens, R. J. M. et al. (1996) als schwer bezeichneten Benchmarkprobleme (s.

Kapitel 3.3). Jedes dieser Probleme wurde bereits optimal gelöst, sodass überprüft werden kann, ob die Integration der ereignisdiskreten Simulation in den Lösungsprozess und der damit verbundene Ausschluss von Lösungen überhaupt eine optimale Lösung der Benchmarkprobleme ermöglicht. Des Weiteren erlaubt der umfassende Performancevergleich von Jain und Meeran (1999) eine Bewertung der verschiedenen Parameterkombinationen bzgl. der Lösungsperformance (z.B. Abweichung vom Optimum und Lösungsgeschwindigkeit). Unabhängig von den Resultaten für die 13 Benchmarkprobleme sollte auch die Fähigkeit zur Lösung von großen praxisrelevanten Problemen untersucht werden. Erst die Ergebnisse von Benchmarktests mit unterschiedlichen Parameterkombinationen für kleine und große Probleme lassen eine Beurteilung zu, ob und in welchem Rahmen sich das Konzept in der Praxis anwenden lässt.

5 Zusammenfassung

Die Betrachtung der Ablaufplanung im produktionswirtschaftlichen Kontext zeigt, dass sie ein Aufgabengebiet der Produktionssteuerung ist. In diesem Rahmen hat sie die Aufgabe, verschiedene Ablaufplanungsprobleme unter Berücksichtigung einer oder mehrerer Zielfunktionen zu lösen. Solche Ablaufplanungsprobleme sind aber von hoher Komplexität gekennzeichnet. Die meisten von ihnen sind sogar NP-schwer, sodass für die Lösung exakte Verfahren, wie das „Branch & Bound"-Verfahren oder die gemischt-ganzzahlige Programmierung, nur bis zu einer bestimmten Problemgröße in Frage kommen. Probleme in praxisrelevanten Größen sind nur unter Zuhilfenahme von Heuristiken zu lösen. Das Job-Shop-Problem stellt in diesem Zusammenhang ein besonders schweres Ablaufplanungsproblem dar und ist zugleich Grundlage für viele komplizierte Probleme der Praxis (vgl. Schuster, 2003, Brinkkötter und Brucker, 1999). Aus diesem Grund dient das Job-Shop-Problem zur Auswahl einer Heuristik, die im Rahmen dieser Arbeit mit der Simulation zu koppeln ist. Die Analyse der Ergebnisse mehrerer Benchmarktests zeigt, dass die Tabu-Suche von Nowicki und Smutnicki (1996) das Job-Shop-Problem in einer angemessenen Qualität und vor allem mit einer sehr hohen Geschwindigkeit löst. Im neuen Konzept zur Kopplung von Simulation und Optimierung übernimmt sie deshalb die Aufgabe der Lösungssuche.

Im Rahmen der Produktion und Logistik ist die ereignisdiskrete Simulation ein wichtiges Werkzeug, mit dem das Verhalten von komplexen Systemen untersucht und beurteilt werden kann. Deshalb wird die Simulation auch in der Ablaufplanung für verschiedene Aufgaben eingesetzt. Hier dient sie zur simulationsbasierten Optimierung von Ablaufplänen. In diesem Zusammenhang berechnet die Simulation aber lediglich die Zielgröße für jede von der Optimierungsmethodik vorgeschlagene Lösung. Dies steht aber in einem Widerspruch zu den kommerziellen Anwendungen, in denen nicht die Simulation, sondern die Optimierungsmethodik eine untergeordnete Rolle spielt. Um diese Gegensätzlichkeit zu vermeiden, fordert Fu (2002) von zukünftigen Konzepten zur simulationsbasierten Optimierung ein besseres Zusammenwirken von Simulation und Optimierung. Um die Effizienz der simulationsbasierten Optimierung zu steigern, soll demnach die Identifikation von einflussreichen Stellgrößen und vor allem die Vermeidung von unnötigen Simulationsdurchläufen ermöglicht werden.

Basierend auf diesen Anforderungen ist ein neues Konzept zur Kopplung von Simulation und Optimierung entwickelt worden. Während die derzeitigen Konzepte zur simulationsbasierten Optimierung direkt über die Stell- und Zielgrößen miteinander kommunizieren, geschieht dies im neuen Konzept ausschließlich über den entwickelten Filter. Der Filter besteht aus vier wesentlichen Komponenten, die den Prozess der simulationsbasierten Optimierung maßgeblich steuern.

1. Constraints-Check: Der Constraints-Check dient zur Überprüfung der Lösung auf ihre Zulässigkeit. Verletzt die von der Tabu-Suche vorgeschlagene Lösung die Nebenbedingungen des Job-Shop-Problems, ist die Lösung unzulässig und wird von der weiteren Betrachtung ausgeschlossen.
2. Input-Check: Ist die Lösung zulässig, wird unter Zuhilfenahme der Input-Matrix überprüft, ob es sich um eine voraussichtlich gute oder schlechte Lösung handelt. Ist die Lösung vermutlich schlecht, wird sie ebenfalls vom Lösungsprozess ausgeschlossen.
3. Input-Matrix-Update: Dem Input-Matrix-Update werden nach jedem Simulationsdurchlauf statistische Daten übermittelt, damit die Qualität der Lösung festgestellt werden kann. Ist die Lösung schlecht, nimmt die Stellgröße Einfluss auf die Input-Matrix, damit ähnliche Stellgrößen in Zukunft im Input-Check identifiziert und ausgeschlossen werden können.
4. Input-Matrix: Die Input-Matrix enthält Informationen zu Stellgrößen, die in der Vergangenheit zu schlechten Lösungen geführt haben. Je mehr Gemeinsamkeiten eine Stellgröße mit der Input-Matrix hat, desto größer ist die Wahrscheinlichkeit, dass es sich um eine voraussichtlich schlechte Lösung handelt. Auf diese Weise kann die Input-Matrix die Identifikation von voraussichtlich schlechten Lösungen im Input-Check ermöglichen.

Die vier Komponenten des Filters lassen die Identifikation von einflussreichen bzw. schlechten Stellgrößen sowie die Vermeidung unnötiger Simulationsdurchläufe durch den Ausschluss schlechter Lösungen zu. Durch die Übermittlung der dafür notwendigen statistischen Daten zur Qualitätsbeurteilung im Input-Matrix-Update dient die ereignisdiskrete Simulation im neuen Konzept somit nicht nur zur Berechnung der Zielgröße, sondern hat auch indirekten Einfluss auf den Lösungsprozess, sodass die Effizienz durch das optimale Zusammenwirken von Simulation und Optimierung gesteigert werden kann.

Das neue Konzept zur Kopplung von Simulation und Optimierung kann nicht ohne Weiteres in der Praxis angewandt werden. Vielmehr dient es als Grundlage für die zukünftige Forschung im Rahmen der simulationsbasierten Optimierung. Hier ist an Benchmarkproblemen, für die optimale Lösungen existieren, zu prüfen, ob durch den Ausschluss von Lösungen überhaupt optimale Lösungen erzielt werden können. Des Weiteren muss analysiert werden, ob ausreichend Gemeinsamkeiten in den Stellgrößen schlechter Lösungen festgestellt werden können, sodass zuverlässig zwischen guten und schlechten Lösungen unterschieden werden kann. Auch die Grenze, die innerhalb des Input-Matrix-Updates zur Qualitätsbeurteilung der Lösungen genutzt wird, muss eingehend auf deren Eignung hin untersucht werden. Die vielfältigen Parameterkom-

binationen des Filters ermöglichen die dafür notwendigen Tests, sodass die Zweckmäßigkeit des neuen Konzeptes für die unterschiedlichsten Anwendungsfälle in der Praxis überprüft werden kann.

6 Literaturverzeichnis

Aarts, B.: A parallel local search algorithm for the job shop scheduling problem. Eindhoven, Eindhoven University of Technology, Department of Mathematics and Computing Science, Masterthesis, 1996.

Achterberg, T.: Constraint integer programming. Berlin, Technische Universität Berlin, Fakultät II, Mathematik und Naturwissenschaften, Dissertation, 2007.

Acker, I.: Methoden der mehrstufigen Ablaufplanung in der Halbleiterindustrie. Wiesbaden: Gabler 2011.

Adam, D.: Produktionsplanung bei Sortenfertigung: Ein Beitrag zur Theorie der Mehrproduktunternehmung. Wiesbaden: Gabler 1969.

Adam, D.: Planung und Entscheidung: Modelle - Ziele - Methoden. Wiesbaden: Gabler 1996.

Adams, J.; Balas, E.; Zawack, D.: The shifting bottleneck procedure for job shop scheduling. Management Science 34, 1988, 3, S. 391–401.

Andersen, E.; Glass, C.A.; Potts, C.N.: Maschine scheduling. In: Aarts, E.; Lenstra, J.K. (Hrsg.): Local search in combinatorial optimization: John Wiley & Sons 1997, S. 361–414.

Andradottir, S.: A review of simulation optimization techniques. In: Carson, J.; Manivannan, M.; Medeiros, D.J.; Watson, E. (Hrsg.): Proceedings of the 1998 Winter Simulation Conference, Washington, USA, 1998, S. 151–158.

Applegate, D.; Cook, W.: A computational study of the job-shop scheduling problem. ORSA Journal on Computing 3, 1991, 2, S. 149–156.

April, J.; Glover, F.; Kelly, J.P.; Laguna, M.: Practical introduction to simulation optimization. In: Ferrin, D.; Morrice, D.; Sanchez, P.; Chick, S. (Hrsg.): Proceedings of the 2003 Winter Simulation Conference, New Orleans, USA, 2003, S. 71–78.

Baker, K.R.: Introduction to sequencing and scheduling. New York, Toronto: John Wiley & Sons 1974.

Balas, E.; Vazacopoulos, A.: Guided local search with shifting bottleneck for job shop scheduling. Management Science 44, 1998, 2, S. 262–275.

Banks, J.; Carson, J.; Nelson, B.L.; Nicol, D.: Discrete-event system simulation. Upper Saddle River: Prentice Hall 2014.

Barton, R.: Simulation metamodels. In: Carson, J.; Manivannan, M.; Medeiros, D.J.; Watson, E. (Hrsg.): Proceedings of the 1998 Winter Simulation Conference, Washington, USA, 1998, S. 167–174.

Benker, H.: Mathematische Optimierung mit Computeralgebrasystemen: Einführung für Ingenieure, Naturwissenschaftler und Wirtschaftswissenschaftler unter Anwendung von MATHEMATICA, MAPLE, MATHCAD, MATLAB und EXCEL. Berlin [u.a.]: Springer 2003.

Błażewicz, J.; Domschke, W.; Pesch, E.: The job shop scheduling problem: conventional and new solution techniques. European Journal of Operational Research 93, 1996, 1, S. 1–33.

Blum, C.: Ant colony optimization: introduction and recent trends. Physics of Life Reviews 2, 2005, 4, S. 353–373.

Bonabeau, E.; Dorigo, M.; Theraulaz, G.: Swarm intelligence: from natural to artificial intelligence. New York: Oxford University Press 1999.

Bonabeau, E.; Dorigo, M.; Theraulaz, G.: Inspiration for optimization from social insect behaviour. Nature 406, 2000, 6791, S. 39–42.

Brah, S.A.; Wheeler, G.E.: Comparison of scheduling rules in a flow shop with multiple processors: a simulation. SIMULATION 71, 1998, 5, S. 302–311.

Brinkkötter, W.; Brucker, P.: Solving open benchmark problems for the job shop problem. Osnabrück: Fachbereich Mathematik/Informatik, Universität Osnabrück 1999.

Brinkkötter, W.; Brucker, P.: Solving open benchmark instances for the job-shop problem by parallel head-tail adjustments. Journal of Scheduling 4, 2001, 1, S. 53–64.

Brucker, P.: Scheduling algorithms. Berlin, Heidelberg: Springer 2004.

Brucker, P.; Jurisch, B.; Sievers, B.: A branch and bound algorithm for the job-shop scheduling problem. Discrete Applied Mathematics 49, 1994, 1, S. 107–127.

Brucker, P.; Knust, S.: Complex scheduling. Berlin, Heidelberg, New York: Springer 2006.

Brüggemann, W.: Ausgewählte Probleme der Produktionsplanung: Modellierung, Komplexität und neuere Lösungsmöglichkeiten. Heidelberg: Physica-Verlag 1995.

Carson, Y.; Maria, A.: Simulation optimization: methods and applications. In: Andradóttir, S.; Healy, K.J.; Withers, D.H.; Nelson, B.L. (Hrsg.): Proceedings of the 1997 Winter Simulation Conference, Atlanta, USA, 1997, S. 118–126.

Caseau, Y.; Laburthe, F.: Improving branch and bound for jobshop scheduling with constraint propagation. In: Goos, G.; Hartmanis, J.; Leeuwen, J.; Deza, M.; Euler, R.; Manoussakis, I. (Hrsg.): Combinatorics and computer science. Berlin, Heidelberg: Springer 1996, S. 129–149.

Černý, V.: Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. Journal of Optimization Theory and Applications 45, 1985, 1, S. 41–51.

Demirkol, E.; Mehta, S.; Uzsoy, R.: A computational study of shifting bottleneck procedures for shop scheduling problems. Journal of Heuristics 3, 1997, 2, S. 111–137.

Demirkol, E.; Mehta, S.; Uzsoy, R.: Benchmarks for shop scheduling problems. European Journal of Operational Research 109, 1998, 1, S. 137–141.

Domschke, W.; Drexl, A.: Einführung in Operations Research. Berlin [u.a.]: Springer 2005.

Domschke, W.; Scholl, A.: Heuristische Verfahren. Jena: Wirtschaftswissenschaftliche Fakultät, Friedrich-Schiller-Universität Jena 2006.

Domschke, W.; Scholl, A.; Voß, S.: Produktionsplanung: Ablauforganisatorische Aspekte. Berlin, Heidelberg, New York: Springer 1997.

Dorigo, M., 2014: Ant colony optimization. Unter Mitarbeit von Romain Hendrickx und Leonardo Bezerra, Université Libre de Bruxelles Belgium. Online verfügbar unter http://www.aco-metaheuristic.org, zuletzt geprüft am 01.04.15.

Dorigo, M.; Blum, C.: Ant colony optimization theory: a survey. Theoretical Computer Science 344, 2005, 2, S. 243–278.

Dorigo, M.; Di Caro, G.: The ant colony optimization metaheuristic. In: Corne, D.; Glover, F.; Dorigo, M. (Hrsg.): New ideas in optimisation. London: McGraw-Hill 1999, S. 11–32.

Dorigo, M.; Di Caro, G.; Gambardella, L.M.: Ant algorithms for discrete optimization. Artificial Life 5, 1999, 2, S. 137–172.

Dorigo, M.; Maniezzo, V.; Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics. Part B (Cybernetics) 26, 1996, 1, S. 29–41.

Dorigo, M.; Stützle, T.: Ant colony optimization. Cambridge: MIT Press 2004.

Dowsland, K.A.: Simulated annealing. In: Reeves, C.R. (Hrsg.): Modern heuristic techniques for combinatorial problems. New York: John Wiley & Sons 1993, S. 20–69.

Dueck, G.; Scheuer, T.; Wallmeier, H.-M.: Toleranzschwelle und Sintflut: Neue Ideen zur Optimierung. Spektrum der Wissenschaft, 1993, 3, S. 42–51.

Fandel, G.: Optimale Entscheidung bei mehrfacher Zielsetzung. Berlin, New York: Springer 1972.

Fisher, H.; Thompson, G.L.: Probabilistic learning combinations of local job-shop scheduling rules. [s.l.]: [s.n.] 1963.

Fowler, J.; Mönch, L.; Rose, O.: Scheduling and simulation. In: Herrmann, J.W. (Hrsg.): Handbook of production scheduling. New York: Springer 2006, S. 109–133.

Fritzsche, A.: Heuristische Suche in komplexen Strukturen: Zur Verwendung Genetischer Algorithmen bei der Auftragseinplanung in der Automobilindustrie. Wiesbaden: Gabler 2009.

Fu, M. (Hg.): Handbook of simulation optimization. New York, Heidelberg, Dordrecht, London: Springer 2015.

Fu, M.C.: Optimization for simulation: theory vs. practice. INFORMS Journal on Computing 14, 2002, 3, S. 192–215.

Fu, M.C.; Glover, F.W.; April, J.: Simulation optimization: a review, new developments, and applications. In: Armstrong, F.; Joines, J.; Steiger, N.; Kuhl, M. (Hrsg.): Proceedings of the 2005 Winter Simulation Conference, Orlando, USA, 2005, S. 83–95.

Garey, M.R.; Johnson, D.S.; Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research 1, 1976, 2, S. 117–129.

Garfinkel, R.S.; Nemhauser, G.L.: Integer programming. New York: John Wiley & Sons 1972.

Geiger, M.J.: Multikriterielle Ablaufplanung. Wiesbaden: Deutscher Universitäts-Verlag 2005.

Giffler, B.; Thompson, G.L.: Algorithms for solving production-scheduling problems. Operations Research 8, 1960, 4, S. 487–503.

Glover, F.: Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 13, 1986, 5, S. 533–549.

Glover, F.: Tabu Search - Part I. INFORMS Journal on Computing 1, 1989, 3, S. 190–206.

Glover, F.: Tabu Search - Part II. INFORMS Journal on Computing 2, 1990, 1, S. 4–32.

Glover, F.; Laguna, M.: Tabu search. In: Reeves, C.R. (Hrsg.): Modern heuristic techniques for combinatorial problems. New York: John Wiley & Sons 1993, S. 70–150.

Goldberg, D.E.: Genetic algorithms in search, optimization, and machine learning. Boston: Addison-Wesley 1989.

Graham, R.L.; Lawler, E.L.; Lenstra, J.K.; Rinnooy Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In: Hammer,

P.L.; Johnson, E.L.; Korte, B.H. (Hrsg.): Discrete Optimization II: Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium. Amsterdam, New York, Oxford: North-Holland Publishing Company 1979, S. 287–326.

Gutenberg, E.: Grundlagen der Betriebswirtschaftslehre: Die Produktion. Berlin: Springer 1983.

Hansen, P.: The steepest ascent mildest descent heuristic for combinatorial programming. In: [s.n.] (Hrsg.): Proceedings of the Congress on Numerical Methods in Combinatorial Optimization, 1986.

Hansmann, K.-W.: Industrielles Management. München [u.a.]: Oldenbourg 2001.

Heib, C.; Nickel, S.: Performancevergleich zwischen simulationsbasierter Online- und Offline-Optimierung anhand von Scheduling-Problemen. In: März, L.; Krug, W.; Rose, O.; Weigert, G. (Hrsg.): Simulation und Optimierung in Produktion und Logistik: Praxisorientierter Leitfaden mit Fallbeispielen. Berlin: Springer 2011, S. 205–214.

Heistermann, J.: Genetische Algorithmen: Theorie und Praxis evolutionärer Optimierung. Stuttgart: Teubner 1994.

Henning, A.: Praktische Job-Shop Scheduling-Probleme. Jena, Friedrich-Schiller-Universität Jena, Fakultät für Mathematik und Informatik, Dissertation, 2002.

Hillier, F.S.: Efficient heuristic procedures for integer linear programming with an interior. Operations Research 17, 1969, 4, S. 600–637.

Hoitsch, H.-J.: Produktionswirtschaft: Grundlagen einer industriellen Betriebswirtschaftslehre. München: Vahlen 1993.

Holland, J.H.: Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. Ann Arbor: University of Michigan Press 1975.

Horn, S.: Simulationsgestützte Optimierung von Fertigungsabläufen in der Produktion elektronischer Halbleiterspeicher. Dresden, Technische Universität Dresden, Dissertation, 2008.

Iltzsche, L.; Schmidt, P.-M.; Völker, S.: Simulationsbasierte Optimierung der Einsteuerungsreihenfolge für die Automobil-Endmontage. In: März, L.; Krug, W.; Rose, O.; Weigert, G. (Hrsg.): Simulation und Optimierung in Produktion und Logistik: Praxisorientierter Leitfaden mit Fallbeispielen. Berlin: Springer 2011, S. 117–132.

Jacob, H.; Adam, D.: Industriebetriebslehre: Handbuch für Studium und Prüfung. Wiesbaden: Gabler 1990.

Jaehn, F.; Pesch, E.: Ablaufplanung: Einführung in Scheduling. Berlin, Heidelberg: Springer Gabler 2014.

Jain, A.S.; Meeran, S.: Deterministic job-shop scheduling: past, present and future. European Journal of Operational Research 113, 1999, 2, S. 390–434.

Kallrath, J.: Gemischt-ganzzahlige Optimierung: Modellierung in der Praxis. Wiesbaden: Springer 2013.

Kern, W.: Industrielle Produktionswirtschaft. Stuttgart: Poeschel 1992.

Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P.: Optimization by simulated annealing. Science 220, 1983, 4598, S. 671–680.

Kistner, K.-P.; Steven, M.: Produktionsplanung. Heidelberg: Physica-Verlag 2001.

Klemmt, A.: Ablaufplanung in der Halbleiter- und Elektronikproduktion: Hybride Optimierungsverfahren und Dekompositionstechniken. Wiesbaden: Springer Vieweg 2012.

Knust, S., 2009: Complexity results for scheduling problems. Unter Mitarbeit von Sigrid Knust. Hg. v. P. Brucker, Fakultät Informatik, Universität Osnabrück. Online verfügbar unter http://www2.informatik.uni-osnabrueck.de/knust/class/, zuletzt geprüft am 31.03.2015.

Krug, W.; Schwope, M.: Simulationsbasierte Reihenfolgeoptimierung in der Produktionsplanung und -steuerung. In: März, L.; Krug, W.; Rose, O.; Weigert, G. (Hrsg.): Simulation und Optimierung in Produktion und Logistik: Praxisorientierter Leitfaden mit Fallbeispielen. Berlin: Springer 2011, S. 105–116.

Krumke, S.O.; Noltemeier, H.: Graphentheoretische Konzepte und Algorithmen. Wiesbaden: Springer Vieweg 2012.

Kurbel, K.; Meynert, J.: Flexibilität und Planungsstrategien für interaktive PPS-Systeme: Arbeitsberichte des Lehrstuhls für Wirtschaftsinformatik. Dortmund: Universität Dortmund 1987.

Kurbel, K.; Rohmann, T.: Ein Vergleich von Verfahren zur Maschinenbelegungsplanung: Simulated Annealing, Genetische Algorithmen und mathematische Optimierung. Wirtschaftsinformatik 37, 1995, 6, S. 581–593.

Laguna, M.; Marti, R.: Neural network prediction in a system for optimizing simulations. IIE Transactions 34, 2002, 3, S. 273–282.

Law, A.M.: Simulation modeling and analysis. Boston: McGraw-Hill 2007.

Lawrence, S.: Supplement to resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. Pittsburgh, Carnegie-Mellon University, Dissertation, 1984.

Lenstra, J.K.; Rinnooy Kan, A.: Computational complexity of discrete optimization problems. In: Hammer, P.L.; Johnson, E.L.; Korte, B.H. (Hrsg.): Discrete Optimization I: Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium. Amsterdam, New York, Oxford: North-Holland Publishing Company 1979, S. 121–140.

Lenstra, J.K.; Rinnooy Kan, A.; Brucker, P.: Complexity of machine scheduling problems. In: Hammer, P.L.; Johnson, E.L.; Korte, B.H.; Nemhauser, G.L. (Hrsg.): Studies in integer programming. Amsterdam, New York, Oxford: North-Holland Publishing Company 1977, S. 343–362.

Littger, K.: Optimierung: Eine Einführung in rechnergestützte Methoden. Berlin, Heidelberg: Springer 1992.

Manne, A.S.: On the job-shop scheduling problem. Operations Research 8, 1960, 2, S. 219–223.

Martin, P.; Shmoys, D.: An new approach to computing optimal schedules for the job-shop scheduling problem. In: Cunningham, W.H.; McCormick, S.Thomas; Queyranne, M. (Hrsg.): Integer programming and combinatorial optimization: Proceedings of the 5th International IPCO Conference, Vancouver, Canada, 1996, S. 389–403.

Martin, P.D.: A time-oriented approach to computing optimal schedules for the job-shop scheduling problem. Ithaca, Cornell University, School of Operations Research and Industrial Engineering, Dissertation, 1996.

März, L.; Krug, W.: Kopplung von Simulation und Optimierung. In: März, L.; Krug, W.; Rose, O.; Weigert, G. (Hrsg.): Simulation und Optimierung in Produktion und Logistik: Praxisorientierter Leitfaden mit Fallbeispielen. Berlin: Springer 2011, S. 41–45.

März, L.; Weigert, G.: Simulationsgestützte Optimierung. In: März, L.; Krug, W.; Rose, O.; Weigert, G. (Hrsg.): Simulation und Optimierung in Produktion und Logistik: Praxisorientierter Leitfaden mit Fallbeispielen. Berlin: Springer 2011, S. 3–12.

Matsuo, H.; Suh, C.J.; Sullivan, R.S.: A controlled search simulated annealing method for the general job-shop scheduling system: Working Paper. Austin: Graduate School of Business, Departement of Management, University of Texas 1988.

Metropolis, N.; Rosenbluth, A.W.; Rosenbluth, M.N.; Teller, A.H.; Teller, E.: Equation of state calculations by fast computing machines. The Journal of Chemical Physics 21, 1953, 6, S. 1087.

Mitchell, M.: An introduction to genetic algorithms. Cambridge: MIT Press 1996.

Müller-Merbach, H.: Optimale Reihenfolgen. Berlin, Heidelberg: Springer 1970.

Nakano, R.; Yamada, T.: Conventional genetic algorithm for job shop problems. In: Kenneth, M.K.; Booker, L.B. (Hrsg.): Proceedings of the 4th International Conference on Genetic Algorithms and their Applications, 1991, S. 474–479.

Nowicki, E.; Smutnicki, C.: A fast taboo search algorithm for the job shop problem. Management Science 42, 1996, 6, S. 797–813.

Nuijten, W.; Le Pape, C.: Constraint-based job-shop scheduling with ILOG scheduler. Journal of Heuristics 3, 1998, 4, S. 271–286.

Pesch, E.; Voß, S.: Strategies with memories: local search in an application oriented environment. OR Spektrum 17, 1995, 2-3, S. 55–66.

Pidd, M.: Computer simulation in management science. Chichester, New York, Weinheim, Brisbane, Singapore, Toronto: John Wiley & Sons 1998.

Pinedo, M.: Scheduling: theory, algorithms, and systems. New York: Springer 2012.

Pirlot, M.: General local search methods. European Journal of Operational Research 92, 1996, 3, S. 493–511.

Preßmar, D.B.: Produktions- und kostentheoretische Modellierung der Kuppelproduktion. In: Keuper, F.; Layer, M. (Hrsg.): Produktion und Controlling: Festschrift für Manfred Layer zum 65. Geburtstag. Wiesbaden: Deutscher Universitäts-Verlag 2002, S. 83–101.

Rabe, M.; Spieckermann, S.; Wenzel, S.: Verifikation und Validierung für die Simulation in Produktion und Logistik: Vorgehensmodelle und Techniken. Berlin, Heidelberg: Springer 2008.

Reeves, C. (Hg.): Modern heuristic techniques for combinatorial problems. London, New York: McGraw Hill Book Co. 1995.

Reeves, C.R.; Beasley, J.E.: Introduction. In: Reeves, C.R. (Hrsg.): Modern heuristic techniques for combinatorial problems. New York: John Wiley & Sons 1993, S. 1–19.

Rieck, J.: Tourenplanung mittelständischer Speditionsunternehmen: Modelle und Methoden. Wiesbaden: Gabler 2009.

Rinnooy Kan, A. H. G.: Machine scheduling problems: classification, complexity and computations. The Hague: Nijhoff 1976.

Rixen, I.: Maschinenbelegungsplanung mit evolutionären Algorithmen. Wiesbaden: Deutscher Universitäts-Verlag 1997.

Rose, O.; März, L.: Simulation. In: März, L.; Krug, W.; Rose, O.; Weigert, G. (Hrsg.): Simulation und Optimierung in Produktion und Logistik: Praxisorientierter Leitfaden mit Fallbeispielen. Berlin: Springer 2011, S. 13–19.

Roy, B.; Sussman, B.: Les problèmes d'ordonnancement avec contraintes disjonctives: Note de travail no. 9, Direction Scientifique. Paris: [s.n.] 1964.

Scheer, A.-W.: Wirtschaftsinformatik: Referenzmodelle für industrielle Geschäftsprozesse. Berlin, Heidelberg, New York, Barcelona, Budapest, Hongkong, London, Mailand, Paris, Santa Clara, Singapur, Tokio: Springer 1997.

Schuster, C.J.: No-wait Job-Shop-Scheduling: Komplexität und Local Search. Duisburg, Universität Duisburg-Essen, Fakultät der Naturwissenschaften, Dissertation, 2003.

Schutten, J.: Practical job shop scheduling. Annals of Operations Research 83, 1998, S. 161–178.

Schwartz, F.: Störungsmanagement in Produktionssystemen. Aachen: Shaker 2004.

Schwede, C.; Klingebiel, K.; Pauli, T.; Wagenitz, A.: Simulationsgestützte Optimierung für die distributionsorientierte Auftragsreihenfolgeplanung in der Automobilindustrie. In: März, L.; Krug, W.; Rose, O.; Weigert, G. (Hrsg.): Simulation und Optimierung in Produktion und Logistik: Praxisorientierter Leitfaden mit Fallbeispielen. Berlin: Springer 2011, S. 151–170.

Seelbach, H.A.: Ablaufplanung. Würzburg, Wien: Physica-Verlag 1975.

Seelbach, H.A.: Ablaufplanung. In: Wittmann, W.; Kern, W.; Köhler, R.; Küpper, H.-U.; Wysocki, K. von (Hrsg.): Handwörterbuch der Betriebswirtschaft. Stuttgart: Schäffer-Poeschel 1993, S. 1–15.

Siegel, T.: Optimale Maschinenbelegungsplanung: Zweckmäßigkeit der Zielkriterien und Verfahren zur Lösung des Reihenfolgeproblems. Berlin: E. Schmidt 1974.

Sotskov, Y.; Shakhlevich, N.V.: NP-hardness of shop-scheduling problems with three jobs. Discrete Applied Mathematics 59, 1995, 3, S. 237–266.

Storer, R.H.; Wu, S.D.; Vaccari, R.: New search spaces for sequencing problems with application to job shop scheduling. Management Science 38, 1992, 10, S. 1495–1509.

Streim, H.: Heuristische Lösungsverfahren: Versuch einer Begriffsklärung. Zeitschrift für Operations Research 19, 1975, 5, S. 143-162.

Taillard, E.: Parallel taboo search technique for the jobshop scheduling problem: Internal research report, Département de Mathématiques, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 1989.

Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 1993, 2, S. 278–285.

Taillard, E.: Parallel taboo search techniques for the job shop scheduling problem. INFORMS Journal on Computing 6, 1994, 2, S. 108–117.

Teich, T.: Optimierung von Maschinenbelegungsplänen unter Benutzung heuristischer Verfahren. Lohmar, Köln: Eul 1998.

Thomsen, S.: Metaheuristics combined with branch and bound: Technical Report, Copenhagen Business School, Copenhagen, Denmark, 1997.

Timkovsky, V.G.: A polynomial-time algorithm for the two-machine unit-time release-date job-shop schedule-length problem. Discrete Applied Mathematics 77, 1997, 2, S. 185–200.

Ting, C.-K.; Li, S.-T.; Lee, C.: On the harmonious mating strategy through tabu search. Information Sciences 156, 2003, 3-4, S. 189–214.

Troßmann, E.: Ablaufplanung bei Einzel- und Serienproduktion. In: Kern, W.; Schröder, H.-H.; Weber, J. (Hrsg.): Handwörterbuch der Produktionswirtschaft. Stuttgart: Schäffer-Poeschel 1996, S. 11–26.

Unger, T.: Lineare Optimierung. Wiesbaden: Teubner 2007.

Vaessens, R. J. M.; Aarts, E. H. L.; Lenstra, J.K.: Job shop scheduling by local search. INFORMS Journal on Computing 8, 1996, 3, S. 302–317.

Van Dyke Parunak, H.: Characterizing the manufacturing scheduling problem. Journal of Manufacturing Systems 10, 1991, 3, S. 241–259.

van Laarhoven, Peter J. M.; Aarts, Emile H. L.; Lenstra, J.K.: Job shop scheduling by simulated annealing. Operations Research 40, 1992, 1, S. 113–125.

van Wassenhove, L.N.; Gelders, L.F.: Solving a bicriterion scheduling problem. European Journal of Operational Research 4, 1980, 1, S. 42–48.

Verein Deutscher Ingenieure 3633 Blatt 1: Simulation von Logistik-, Materialfluss- und Produktionssystemen: Grundlagen. Berlin: Beuth, 2014.

Weigert, G.; Klemmt, A.; Horn, S.: Design and validation of heuristic algorithms for simulation-based scheduling of a semiconductor backend facility. International Journal of Production Research 47, 2009, 8, S. 2165–2184.

Weigert, G.; Rose, O.: Stell- und Zielgrößen. In: März, L.; Krug, W.; Rose, O.; Weigert, G. (Hrsg.): Simulation und Optimierung in Produktion und Logistik: Praxisorientierter Leitfaden mit Fallbeispielen. Berlin: Springer 2011, S. 29–39.

Wennink, M.: Algorithmic support for automated planning boards. Eindhoven, Technische Universiteit Eindhoven, Department of Mathematics and Computing Science, Ph.D. Thesis, 1995.

Wenzel, S.: Referenzmodelle für die Simulation in Produktion und Logistik. Delft, Erlangen, Ghent, San Diego: Society for Computer Simulation International 2000.

Wolpert, D.H.; Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1, 1997, 1, S. 67–82.

Wolpert, D.H.; Macready, W.G.: Coevolutionary free lunches. IEEE Transactions on Evolutionary Computation 9, 2005, 6, S. 721–735.

Woolsey, R. E. D.: Production scheduling quick and dirty methods for parallel machines. Production and Inventory Management 31, 1990, 3, S. 84–87.

Yamada, T.; Nakano, R.: A genetic algorithm applicable to large-scale job-shop problems. In: Männer, R.; Manderick, B. (Hrsg.): Proceedings of the 2nd International Workshop on Parallel Problem Solving from Nature, Brussels, Belgium, 1992, S. 281–290.

Yamada, T.; Nakano, R.: Job-shop scheduling by simulated annealing combined with deterministic local search. In: Osman, I.H.; Kelly, J.P. (Hrsg.): Meta-heuristics: theory and applications. Boston: Springer 1996, S. 237–248.

Zanakis, S.H.; Evans, J.R.; Vazacopoulos, A.A.: Heuristic methods and applications: a categorized survey. European Journal of Operational Research 43, 1989, 1, S. 88–110.

Zäpfel, G.: Produktionswirtschaft: Operatives Produktions-Management. Berlin [u.a.]: De Gruyter 1982.

Zäpfel, G.: Grundzüge des Produktions- und Logistikmanagement. Berlin, New York: De Gruyter 1996.

Zäpfel, G.: Strategisches Produktions-Management. München, Wien: Oldenbourg 2000.

Zäpfel, G.; Braune, R.: Moderne Heuristiken der Produktionsplanung: am Beispiel der Maschinenbelegung. München: Vahlen 2005.

Zelewski, S.; Hohmann, S.; Hügens, T.: Produktionsplanungs- und -steuerungssysteme: Konzepte und exemplarische Implementierungen mithilfe von SAP R/3. München: Oldenbourg 2008.

Anhang A1

Tabelle 10: Ergebnisse für die Benchmarkprobleme von Taillard (1993) (Jain und Meeran, 1999)

Problem	Beste obere Grenze	Als erstes erreicht durch
(15×15)		
TD1	1231	Taillard (1994)
TD2	(1244)	Nowicki und Smutnicki (1996)
TD3	(1218)	Balas und Vazacopoulos (1998)
TD4	(1175)	Wennink (1995)
TD5	(1228)	Wennink (1995)
TD6	(1240)	Wennink (1995)
TD7	(1228)	Taillard (1994)
TD8	(1217)	Balas und Vazacopoulos (1998)
TD9	(1274)	Balas und Vazacopoulos (1998)
TD10	1241	Balas und Vazacopoulos (1998)
(20×15)		
TD11	(1364)	Balas und Vazacopoulos (1998)
TD12	(1367)	Balas und Vazacopoulos (1998)
TD13	(1350)	Balas und Vazacopoulos (1998)
TD14	1345	Nowicki und Smutnicki (1996)
TD15	(1342)	Vaessens, R. J. M. et al. (1996)
TD16	(1368)	Aarts (1996)
TD17	(1478)	Balas und Vazacopoulos (1998)
TD18	(1396)	Balas und Vazacopoulos (1998)
TD19	(1341)	Balas und Vazacopoulos (1998)
TD20	(1353)	Wennink (1995)
(20×20)		
TD21	(1647)	Aarts (1996)
TD22	(1603)	Balas und Vazacopoulos (1998)
TD23	(1558)	Balas und Vazacopoulos (1998)
TD24	(1651)	Aarts (1996)
TD25	(1598)	Taillard (1995)

Fortsetzung Tabelle 10		
Problem	**Beste obere Grenze**	**Als erstes erreicht durch**
TD26	(1655)	Wennink (1995)
TD27	(1689)	Balas und Vazacopoulos (1998)
TD28	(1615)	Balas und Vazacopoulos (1998)
TD29	(1625)	Aarts (1996)
TD30	(1596)	Vaessens, R. J. M. et al. (1996)
(30×15)		
TD31	(1766)	Nowicki und Smutnicki (1996)
TD32	(1803)	Balas und Vazacopoulos (1998)
TD33	(1796)	Balas und Vazacopoulos (1998)
TD34	(1832)	Balas und Vazacopoulos (1998)
TD35	2007	Taillard (1994)
TD36	(1823)	Balas und Vazacopoulos (1998)
TD37	(1784)	Balas und Vazacopoulos (1998)
TD38	(1681)	Balas und Vazacopoulos (1998)
TD39	(1798)	Balas und Vazacopoulos (1998)
TD40	(1686)	Balas und Vazacopoulos (1998)
(30×20)		
TD41	(2026)	Balas und Vazacopoulos (1998)
TD42	(1967)	Balas und Vazacopoulos (1998)
TD43	(1881)	Balas und Vazacopoulos (1998)
TD44	(2004)	Balas und Vazacopoulos (1998)
TD45	(2008)	Balas und Vazacopoulos (1998)
TD46	(2040)	Balas und Vazacopoulos (1998)
TD47	(1921)	Balas und Vazacopoulos (1998)
TD48	(1982)	Balas und Vazacopoulos (1998)
TD49	(1994)	Balas und Vazacopoulos (1998)
TD50	(1951)	Aarts (1996)
(50×15)		
TD51	2760	Taillard (1994)

Fortsetzung Tabelle 10		
Problem	**Beste obere Grenze**	**Als erstes erreicht durch**
TD52	2756	Taillard (1994)
TD53	2717	Taillard (1994)
TD54	2839	Taillard (1994)
TD55	2679	Nowicki und Smutnicki (1996)
TD56	2781	Taillard (1994)
TD57	2953	Taillard (1994)
TD58	2885	Taillard (1994)
TD59	2655	Taillard (1994)
TD60	2723	Taillard (1994)
(50×20)		
TD61	2868	Nowicki und Smutnicki (1996)
TD62	(2900)	Balas und Vazacopoulos (1998)
TD63	2755	Nowicki und Smutnicki (1996)
TD64	2702	Nowicki und Smutnicki (1996)
TD65	2725	Nowicki und Smutnicki (1996)
TD66	2845	Nowicki und Smutnicki (1996)
TD67	(2826)	Balas und Vazacopoulos (1998)
TD68	2784	Nowicki und Smutnicki (1996)
TD69	3071	Nowicki und Smutnicki (1996)
TD70	2995	Nowicki und Smutnicki (1996)
(100×20)		
TD71	5464	Taillard (1994)
TD72	5181	Taillard (1994)
TD73	5568	Taillard (1994)
TD74	5339	Taillard (1994)
TD75	5392	Taillard (1994)
TD76	5342	Taillard (1994)
TD77	5436	Taillard (1994)
TD78	5394	Taillard (1994)

Fortsetzung Tabelle 10		
Problem	**Beste obere Grenze**	**Als erstes erreicht durch**
TD79	5358	Taillard (1994)
TD80	5183	Nowicki und Smutnicki (1996)

Werte in () beste erreichte obere Grenze

Werte ohne () optimale Lösung, d.h. Problem gelöst

Tabelle 11: Ergebnisse für die Benchmarkprobleme von Demirkol et al. (1998) (Jain und Meeran, 1999)

Problem	**Beste obere Grenze**	**Als erstes erreicht durch**
(20×15)		
DMU1	(2607)	Nowicki und Smutnicki (1996)
DMU2	(2760)	Nowicki und Smutnicki (1996)
DMU3	(2813)	Nowicki und Smutnicki (1996)
DMU4	(2691)	Nowicki und Smutnicki (1996)
DMU5	(2791)	Nowicki und Smutnicki (1996)
(20×20)		
DMU6	(3316)	Nowicki und Smutnicki (1996)
DMU7	(3127)	Nowicki und Smutnicki (1996)
DMU8	(3234)	Nowicki und Smutnicki (1996)
DMU9	(3185)	Nowicki und Smutnicki (1996)
DMU10	(3022)	Nowicki und Smutnicki (1996)
(30×15)		
DMU11	(3539)	Nowicki und Smutnicki (1996)
DMU12	(3605)	Nowicki und Smutnicki (1996)
DMU13	(3725)	Nowicki und Smutnicki (1996)
DMU14	(3439)	Nowicki und Smutnicki (1996)
DMU15	(3413)	Nowicki und Smutnicki (1996)
(30×20)		
DMU16	(3882)	Nowicki und Smutnicki (1996)
DMU17	(4009)	Nowicki und Smutnicki (1996)
DMU18	(3939)	Nowicki und Smutnicki (1996)
DMU19	(3967)	Nowicki und Smutnicki (1996)
DMU20	(3833)	Nowicki und Smutnicki (1996)
(40×15)		
DMU21	4380	Nowicki und Smutnicki (1996)
DMU22	(4769)	Demirkol et al. (1997)

Fortsetzung Tabelle 11		
Problem	**Beste obere Grenze**	**Als erstes erreicht durch**
DMU23	(4695)	Demirkol et al. (1997)
DMU24	4648	Nowicki und Smutnicki (1996)
DMU25	(4168)	Nowicki und Smutnicki (1996)
(40×20)		
DMU26	(4850)	Nowicki und Smutnicki (1996)
DMU27	(4945)	Nowicki und Smutnicki (1996)
DMU28	(4735)	Nowicki und Smutnicki (1996)
DMU29	(4764)	Nowicki und Smutnicki (1996)
DMU30	(4885)	Nowicki und Smutnicki (1996)
(50×15)		
DMU31	(5695)	Nowicki und Smutnicki (1996)
DMU32	5927	Demirkol et al. (1997)
DMU33	5728	Demirkol et al. (1997)
DMU34	5385	Demirkol et al. (1997)
DMU35	5635	Demirkol et al. (1997)
(50×20)		
DMU36	(5705)	Nowicki und Smutnicki (1996)
DMU37	(5940)	Nowicki und Smutnicki (1996)
DMU38	(5806)	Nowicki und Smutnicki (1996)

Werte in () beste erreichte obere Grenze

Werte ohne () optimale Lösung, d.h. Problem ist gelöst

Tabelle 12: Ergebnisse für die 13 schweren Benchmarkprobleme (Jain und Meeran, 1999)

Autor	Benchmarkprobleme													
	FT10	LA2	LA19	LA21	LA24	LA25	LA27	LA29	LA36	LA37	LA38	LA39	LA40	Gelöst
Optimum	930	655	842	1046	935	977	1235	1152	1268	1397	1196	1233	1222	13
Martin (1996)	930	655	842	1046	935	977	1235	1152	1268	1397	1196	1233	1222	13
Balas und Vazacopoulos (1998)	930	655	842	1046	935	977	1235	1157	1268	1397	1196	1233	1224	11
Nuijten und Le Pape (1998)	930	655	842	1046	938	977	1235	1165	1268	1397	1196	1233	1224	10
Yamada und Nakano (1996)	930	655	842	1046	935	977	1235	1154	-	-	1198	-	1228	7
Nowicki und Smutnicki (1996)	930	655	842	1047	939	977	1236	1160	1268	1407	1196	1233	1229	7
Thomsen (1997)	930	655	842	1047	935	977	1235	1157	1268	1397	1196	1233	1224	10

FT - Fisher und Thompson (1963), LA - Lawrence (1984)

Tabelle 13: Abweichung und Dauer der Lösungsverfahren für die 13 schweren Benchmarkprobleme (Jain und Meeran, 1999)

Autor	Summe der mittleren relativen Abweichung (%)	Durchschnittliche Dauer CI-CPU (s)
Martin (1996)	0	97160,4
Balas und Vazacopoulos (1998)	0,60	999,19
Nuijten und Le Pape (1998)	1,61	29335,4
Yamada und Nakano (1996)	0,83	132961
Nowicki und Smutnicki (1996)	2,59	106,89
Thomsen (1997)	0,69	7650,1

CI-CPU - Computer-Independent Central Processing Unit

Eidesstattliche Versicherung

_____ _____

Name, Vorname Matr.-Nr.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem Titel

_____

_____

_____

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

_____ _____

Ort, Datum Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -)

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

_____ _____

Ort, Datum Unterschrift