Hybrid Flow-Shop Scheduling mit verschiedenen Restriktionen: Heuristische Lösung und LP-basierte untere Schranken

Von der Fakultät für Mathematik der Universität Duisburg-Essen
zur Erlangung des akademischen Grades einer
Doktorin der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation

von

Verena Gondek

aus Viersen

Referent: Prof. Dr. Günter Törner
Korreferent: Prof. Dr. Rainer Leisten

Datum der mündlichen Prüfung: 21. April 2011

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Danksagung

Mein Dank gilt an erster Stelle meinem Betreuer Herrn Prof. Dr. Günter Törner, ohne den die Umsetzung meines Promotionsvorhabens nicht möglich gewesen wäre. Außerdem danke ich der ThyssenKrupp Steel Europe AG, insbesondere Frau Dr. Susanne van den Bosch und Herrn Dr. Udo Paul, für die Möglichkeit, meine Resultate im Rahmen eines Kooperationsprojektes auch in der Praxis zu erproben. Die Ergebnisse dieses Projektes bilden als Fallstudie einen wichtigen Bestandteil meiner Dissertation. Hierbei gebührt mein Dank des Weiteren Henning Kerstan für die Übernahme umfangreicher Programmierarbeiten.

Ich danke außerdem meinen Kolleginnen Bettina Rösken und Miriam Dieter sowie unseren studentischen Mitarbeitern/innen Britta Berndtsen, Carina Hüsken und Sebastian Feldhaus für die angenehme und gute Zusammenarbeit. Mein besonderer Dank gilt außerdem Herrn Prof. Dr. Rüdiger Schultz für seine Unterstützung bei Fragen und Problemen jeder Art.

Abschließend danke ich meiner Familie für ihren Rückhalt und ihr Verständnis auch in schwierigen Phasen meines Studiums und meiner Promotionszeit.

Köln, im Januar 2011

Zusammenfassung

Während der Herstellung von Stahl ist es erforderlich, kontinuierlich dessen Qualität zu überwachen. Aus diesem Grund werden an verschiedenen Positionen in einem Stahlwerk fortlaufend Produktproben entnommen und analysiert. Ein großer deutscher Stahlerzeuger betreibt zu diesem Zweck ein vollautomatisiertes Labor. Die Proben werden per Rohrpost in dieses Labor gesendet und dort mit Hilfe verschiedener Maschinen untersucht. Notwendige Transporte zwischen diesen Maschinen werden unter Verwendung mehrerer Roboter durchgeführt. Die Belegungsplanung der Maschinen sowie das entsprechende Routing der Roboter bilden ein komplexes Scheduling-Problem. Dabei soll eine möglichst geringe Aufenthaltsdauer der Proben im Labor realisiert werden. Insgesamt kann diese Aufgabe als dynamisches Hybrid Flow-Shop-Problem mit Transporten und Minimierung der gewichteten Gesamtfertigstellungszeit (resp. gewichtete Gesamtflusszeit) klassifiziert werden, da die Ankunftszeit der Proben a priori nicht bekannt ist. Weil die Analyse einer Probe im Labor zudem maximal wenige Minuten dauern darf, steht nur eine sehr geringe Rechenzeit zur Lösung dieses Scheduling-Problems zur Verfügung.

Die Entwicklung eines neuen Entscheidungssystems zur Optimierung der Arbeitsabläufe in einem solchen Labor ist ein Bestandteil der vorliegenden Dissertation. Dazu wird ein mehrstufiges heuristisches Lösungsverfahren entwickelt, welches auf einem Dekompositionsansatz, (engpass-orientierten) Prioritätsregeln und einer job-orientierten List Scheduling Strategie basiert. Die Arbeitsweise des Verfahrens für das Labor wird im Rahmen einer Fallstudie simuliert und die erzielten Lösungen mit dem Ist-Zustand des Labors verglichen. In der entsprechenden Analyse kann ein enormes Verbesserungspotential gegenüber dem derzeit verwendeten Planungstool nachgewiesen werden.

Neben diesem anwendungsorientierten Teil der Arbeit wird die Performance des vorgestellten Verfahrens auch für allgemeinere Situationen empirisch untersucht. Zur Auswertung der erzielten Lösungen für verschiedene zufällig generierte Datensätze (insgesamt 1500 Probleminstanzen), werden zwei LP-basierte untere Schranken verwendet, welche auf einer zeit-indizierten gemischt-ganzzahligen Modellierung des Problems beruhen. Darüber hinaus werden diese Schranken auch auf theoretischer Ebene analysiert und mit weiteren in der Literatur gebräuchlichen Schranken verglichen.

Abstract

During the manufacture of steel, its quality has to be monitored continuously. Therefore, samples are taken at several stages of the production process and their chemical composition is analyzed. A big German steel producer uses an automatic laboratory to perform this task. The samples are sent to this laboratory under usage of a pneumatic post system and afterwards they are processed by different machines. Arising transportation tasks between those operations are managed by a fleet of robots. The timetabling of the several machines as well as the related routing of the robots is a complex scheduling problem. Therein the flow time of the samples should be minimized. Altogether, this task can be classified as a dynamic hybrid flow-shop scheduling problem with transportation and total weighted completion time or total weighted flow time objective, because the arrival times of the samples are not known in advance. Because the analysis of one sample should at most last a few minutes, the available computational time to perform the required real-time optimization is strictly limited.

The development of a decision support system to optimize the workflow in such a laboratory is one part of this dissertation. Therefore, a multi-stage heuristic algorithm is designed, which is based on a decomposition approach, (bottleneck related) dispatching rules as well as a job-oriented list scheduling strategy. The performance of this method in case of the laboratory is simulated and compared to the current control system. It can be shown that the new approach is able to reduce the total weighted flow time significantly.

Beneath this application part of the thesis, the performance of the method is further evaluated in a more theoretical fashion. Therefore, an extensive empirical analysis is performed, where lower bounds are used to benchmark the heuristic solutions to 1500 random problem instances under consideration. These bounds are based on the lp relaxation of two time-indexed mixed-integer formulations of the problem. Furthermore, they are also compared to different other bounds introduced in literature.

1. Einführung

Die wachsende Komplexität und Dynamik von Produktionssystemen sowie der Übergang zur teilweisen oder sogar vollständigen Automatisierung ganzer Fertigungseinheiten führt dazu, dass diese manuell oftmals nicht mehr effizient steuerbar sind, sondern vielschichtige Entscheidungs- und Steuerungssysteme erfordern, um dem wachsenden Wettbewerbsdruck standhalten zu können. Ein entsprechendes Werkzeug muss in der Lage sein, immer größer werdende Datenmengen zu verarbeiten, gleichzeitig werden jedoch immer kürzere Reaktionszeiten verlangt. Die Entwicklung generischer Tools ist in diesem Zusammenhang nur bedingt möglich, da jedes Planungsproblem in der Praxis durch eine Vielzahl individueller Merkmale und Restriktionen charakterisiert wird.

Ein Versuch, solche Probleme dennoch anhand ihrer Rahmenbedingungen zu klassifizieren und anschließend Lösungsverfahren für eine ganze Problemklasse zu konzipieren, wird in der Scheduling-Theorie unternommen. Es handelt sich dabei um eine verhältnismäßig junge[1] mathematische bzw. betriebswirtschaftliche Disziplin, die zahlreiche Gemeinsamkeiten mit der kombinatorischen Optimierung aufweist und auf der Ebene kurzfristiger Entscheidungsprozesse angesiedelt werden kann (vgl. Kapitel 2). Bei der Lösung eines realistischen Planungsproblems kann eine Scheduling-Strategie in den meisten Fällen aber wohl nur als Grundlage für die Konstruktion eines problemspezifischeren Verfahrens dienen. Gegenstand dieser Arbeit ist zunächst die Entwicklung und Analyse eines (allgemeinen) heuristischen Verfahrens zur Lösung sogenannter Hybrid Flow-Shop-Probleme unter Berücksichtigung verschiedener Nebenbedingungen. Anschließend wird die entstandene Methode zur Lösung eines Anwendungsproblems aus der Stahlindustrie modifiziert und spezialisiert.

Während der Produktion von Stahl ist es erforderlich, fortlaufend dessen chemische Zusammensetzung zu überprüfen, um so den notwendigen Qualitätsstandard zu sichern. Die ThyssenKrupp Steel Europe AG betreibt zur Untersuchung entsprechender Produktproben ein vollautomatisiertes Labor, welches im weiteren Verlauf dieser Arbeit auch als *Prozesslabor* bezeichnet wird. Die Ablaufsteuerung dieses Labors ist ein komplexes Planungsproblem, welches im Rahmen der vorliegenden Dissertation als praktische Fallstudie behandelt wird. Entsprechende Arbeiten wurden teilweise im Rahmen eines Kooperationsprojektes mit dem oben genannten Industriepartner durchgeführt (vgl. insbesondere Kapitel 7).

[1] Erste Publikationen sind in den 1950er Jahren zu finden (vgl. u. a. Johnson, 1954, [60]).

Die zu analysierenden Proben werden an verschiedenen Positionen im Stahlwerk entnommen und per Rohrpost an das Prozesslabor gesendet. Dort eingetroffen werden sie von Robotern in Empfang genommen, welche auch alle weiteren notwendigen Transporte innerhalb des Labors durchführen. Jede Probe wird unter Verwendung mehrerer Maschinen untersucht. Die Menge erforderlicher Bearbeitungsschritte kann dabei zwar variieren, die Reihenfolge, in welcher diese durchgeführt werden, ist aus physikalischen Gründen jedoch festgelegt. Ferner ist jede Maschine im Labor in mehrfacher Ausführung vorhanden, um mehr Arbeitsgänge zur selben Zeit ausführen zu können. Dies erhöht jedoch gleichermaßen die Komplexität der Ablaufplanung, da zusätzlich entschieden werden muss, welche dieser parallelen (aber nicht notwendigerweise gleichwertigen) Maschinen zur Bearbeitung einer bestimmten Probe ausgewählt werden sollen.

Neben der Organisation anfallender Transportaufgaben, welche natürlich zusätzlich mit der Belegung der verschiedenen Maschinen zu synchronisieren ist, spielen bei der Entwicklung eines Planungssystems weitere Nebenbedingungen eine Rolle. So kann eine Probe zwischen zwei Bearbeitungsgängen beispielsweise nur bedingt zwischengelagert werden. Im Kontext des Prozesslabors bedeutet dies, dass eine Probe nach dem Abschluss eines Arbeitsgangs solange in der betreffenden Maschine verweilt und diese damit blockiert, bis sie zur nächsten Maschine transportiert wird. Bevor eine Probe in die Rohrpost übergeben wird, ist im Labor des Weiteren weder bekannt wann, noch wo diese im Werk entnommen wird. Infolgedessen kann eine entsprechende Ablaufsteuerung nur dynamisch vonstattengehen. Darüber hinaus durchlaufen die Proben das Labor innerhalb weniger Minuten. Ein Planungssystem, welches beim Eintreffen einer Probe entscheidet, wann diese auf welchen Maschinen bearbeitet werden soll und wie die zugehörigen Transporte zu organisieren sind, muss demnach extrem schnell reagieren können.

Das Ziel eines neuen Steuerungssystems ist es, die Durchlaufzeit der Proben durch das Prozesslabor gegenüber dem Ist-Zustand zu verringern. Dabei muss berücksichtigt werden, dass die Proben unterschiedlichen Prioritäten unterliegen, was jedoch durch eine geeignete Gewichtung einer entsprechenden Zielfunktion abgebildet werden kann. Aus scheduling-theoretischer Sicht kann man die Ablaufplanung des Prozesslabors im Wesentlichen als dynamisches Hybrid Flow-Shop-Problem mit Transporten beschreiben, in welchem die mittlere gewichtete Fertigstellungszeit bzw. Flusszeit der Proben minimiert werden soll. Für detaillierte Definitionen dieser Fachtermini sei auf die Kapitel 2, 3 und 7 verwiesen.

Insgesamt ist die vorliegende Dissertation wie folgt gegliedert: In Kapitel 2 werden grundlegende Begriffe und Methoden der Scheduling-Theorie vorgestellt. Zusätzlich benötigte Basiskenntnisse aus dem Bereich der (kombinatorischen) Optimierung werden jedoch vorausgesetzt. Für entsprechende Informationen sei beispielsweise auf die Standardwerke von Nemhauser und Wolsey (1998) [96] sowie Kallrath (2002) [65] verwiesen. Der aktuelle Stand der Forschung im Zusammenhang mit (heuristischen) Lösungsmethoden für verschiedene Hybrid Flow-Shop-Probleme wird in Kapitel 3 zusammengestellt. Dabei

wird insbesondere die Relevanz der vorgestellten Methoden in Bezug auf das zugrundeliegende Anwendungsproblem diskutiert. Kapitel 4 ist der mathematischen Modellierung und LP-basierten Schrankenbildung für die untersuchten Scheduling-Probleme gewidmet. In Kapitel 5 werden die entwickelten Lösungsverfahren präsentiert. Die zugehörige empirische Performance-Analyse ist anschließend in Kapitel 6 dargestellt. Die notwendigen Anpassungen der entwickelten Methode, um diese zur Steuerung des Prozesslabors verwenden zu können, werden in Kapitel 7 vorgestellt. Dieses enthält zudem die Ergebnisse einer entsprechenden Ablaufsimulation im Vergleich zum Ist-Zustand des Labors. Eine abschließende Bewertung der Resultate dieser Dissertation kann Kapitel 8 entnommen werden.

2. Grundbegriffe der Scheduling-Theorie

In diesem Kapitel werden die grundlegenden Begriffe der Scheduling-Theorie eingeführt. Sofern nicht anders angegeben, basieren die Ausführungen dabei im Wesentlichen auf den Büchern von Pinedo (2008) [106] sowie Leung (2004) [80].

Wörtlich übersetzt heißt *Scheduling* soviel wie Zeitplanerstellung oder Ablaufplanung. Das Grundproblem der Scheduling-Theorie ist daher im Kern das Folgende: Es steht eine begrenzte Anzahl von *Ressourcen* bzw. *Maschinen* zur Verfügung und mit deren Hilfe ist eine Menge von *Aufträgen* (oder *Jobs*) möglichst kostengünstig[1] zu bearbeiten (vgl. dazu auch Jungwattanakit et al., 2008, [62]: *Production scheduling can be defined as the allocation of available production resources over time to perform a collection of tasks*). Man spricht in diesem Zusammenhang auch von einem *Maschinenbelegungsproblem*, wobei der Begriff „Maschine“ auch im übertragenen Sinn[2] verstanden werden kann. Scheduling ist damit insbesondere ein wichtiger Bestandteil kurzfristiger Produktions- oder Terminplanung (vgl. Quadt & Kuhn, 2007, [110]), beispielsweise in der Stahl- und Automobilindustrie oder dem Management eines Flughafens (vgl. Bianco et al., 2006, [19]), um nur wenige der zahlreichen Anwendungsgebiete zu nennen. Eine mögliche Lösung eines Scheduling-Problems bezeichnet man zumeist als *Terminplan* oder *Schedule*.

In einem Produktionsprozess kann ein Job aus einer einzelnen Operation oder aus einer Menge von Operationen bestehen, die durchaus auf verschiedenen Maschinen bearbeitet werden können oder sogar müssen. Grundsätzlich gehen wir von m vorhandenen Maschinen ($I = \{1,\dots,m\}$) und n zu bearbeitenden Jobs ($J = \{1,\dots,n\}$) aus. Erfordert ein Job j mehrere Arbeitsgänge, also verschiedene Operationen, dann repräsentiert das Paar (i, j) einen Bearbeitungsschritt des Jobs j auf der Maschine i. Die Reihenfolge, in welcher diese Operationen in einem Schedule abgearbeitet werden, nennt man *Arbeitsgangfolge* des Jobs j.

2.1. Notation und Klassifikation von Scheduling-Problemen

Jedes Scheduling-Problem wird durch verschiedene grundlegende Merkmale charakterisiert. Dazu gehört die jeweilige Zielfunktion genauso wie die Art der Maschinen. Zur

[1] Wie in diesem Kontext „kostengünstig“ verstanden werden muss, ist von der Zielfunktion des jeweiligen Problems abhängig und kann nur diesbezüglich spezifiziert werden.

[2] D. h. verschiedene Arten von Ressourcen, z. B. auch Personal, können als Maschine bezeichnet werden.

Spezifikation eines Scheduling-Problems hat sich nach einer Arbeit von Graham et al. (1979) [46] ein formales 3-Feld-Klassifikationsschema etabliert, welches im Folgenden vorgestellt wird. Um die verschiedenen Zielfunktionen etc. zu beschreiben, wird dazu zunächst die in der Literatur gebräuchliche Notation der verwendeten Parameter eingeführt. Da der größte Teil scheduling-theoretischer Literatur in englischer Sprache verfasst ist, werden zusätzlich auch immer die entsprechenden englischen Bezeichnungen angegeben.

2.1.1. Problemparameter

Es gibt zahlreiche Parameter, die in einem Scheduling-Problem eine Rolle spielen können. Wir beschränken uns daher auf diejenigen, welche im Rahmen dieser Arbeit benötigt werden. Die nachfolgende Liste enthält zunächst verschiedene Input-Parameter.

- $I = \{1, \ldots, m\}$ - Menge der Maschinen (machines)
- $J = \{1, \ldots, n\}$ - Menge der Jobs (jobs)
- p_j, p_{ij} - Bearbeitungszeit eines Jobs j bzw. einer Operation (i, j) (processing time)
- w_j - Gewicht (bzw. Priorität) eines Jobs j (weight)
- r_j - Freigabezeit eines Jobs j (release date)
- d_j - Liefertermin eines Jobs j (due date)

Darüber hinaus werden einige, vom Schedule abhängige (Output-)Parameter zur Definition der Zielfunktionen benötigt. Gebräuchlich sind dabei unter anderem:

- C_j - Fertigstellungszeit eines Jobs j (completion time)
- $F_j := C_j - r_j$ - Flusszeit eines Jobs j (flow time)
- $L_j := C_j - d_j$ - Terminabweichung eines Jobs j (lateness)
- $T_j := \max\{0, L_j\}$ - Verspätung eines Jobs j (tardiness)
- $E_j := \max\{0, -L_j\}$ - Verfrühung eines Jobs j (earliness)

2.1.2. Ein Klassifikationsschema für Scheduling-Probleme

In der Grundversion des 3-Feld-Klassifikationsschemas nach Graham et al. (1979) [46] verwendet man ein Tripel $\alpha|\beta|\gamma$ um die entscheidenden Merkmale eines Scheduling-Problems zu charakterisieren. Im Einzelnen sind dies die Maschinenumgebung (α), die Randbedingungen des Problems (β) und die untersuchten Zielfunktionen (γ). Für bestimmte Typen von Scheduling-Problemen wurde dieses Schema später noch verfeinert, darauf wird jedoch erst an entsprechender Stelle im weiteren Verlauf dieser Arbeit eingegangen.

2.1.2.1. Maschinenumgebungen (α)

Vereinfacht dargestellt, werden drei Grundtypen von Maschinenumgebungen unterschieden:

1. 1-Maschinen-Probleme
2. Parallelmaschinen-Probleme
3. Shop-Scheduling-Probleme

Die Verplanung einer einzelnen Maschine bezeichnet man als *1-Maschinen-Problem* und das α-Feld enthält in diesem Fall den Eintrag „1“. Bei *Parallelmaschinen-Problemen* unterscheidet man verschiedene Ausprägungen. Im einfachsten Fall gibt es mehrere *identische* Maschinen, die parallel arbeiten, d. h. jeder Job muss nur von einer der Maschinen bearbeitet werden und jede der Maschinen benötigt dazu dieselbe Geschwindigkeit. Es werden in der Scheduling-Theorie aber auch Situationen mit *parallelen Maschinen unterschiedlicher Geschwindigkeit* behandelt. Die folgende Liste enthält die Notation zur Beschreibung der Grundtypen paralleler Maschinenumgebungen im α-Feld:

Pm (parallel and identical machines): Es gibt m parallele Maschinen derselben Geschwindigkeit. Jeder Job besteht aus einer einzelnen Operation, die auf jeder der m Maschinen bearbeitet werden kann.

Qm (uniform machines): Es gibt m parallele Maschinen unterschiedlicher Geschwindigkeit. Besitzt Maschine i die Geschwindigkeit v_i, so ergibt sich die Verweildauer eines Jobs j auf der Maschine i als $\frac{p_j}{v_i}$.

Rm (unrelated machines): Es gibt m parallele Maschinen unterschiedlicher Geschwindigkeit, wobei diese vom jeweils bearbeiteten Job abhängt. Besitzt eine Maschine i bei der Bearbeitung des Jobs j die Geschwindigkeit v_{ij}, so ergibt sich dessen Verweildauer auf der Maschine i demnach als $\frac{p_j}{v_{ij}}$.

Shop-Scheduling-Probleme beinhalten Maschinenumgebungen mit komplexerer Struktur. In einem *Flow-Shop* gibt es beispielsweise m verschiedene Maschinen und jeder Job muss auf jeder dieser Maschinen bearbeitet werden.[3] Alle Jobs durchlaufen die Maschinen dabei in derselben Reihenfolge, weshalb man diese Maschinenumgebung auch häufig als Fertigungsstraße oder -linie (flow line) bezeichnet. Bei einem *Job-Shop-Problem* gibt es dagegen für jeden Job eine individuelle Reihenfolge, in welcher der Maschinenpark passiert werden muss. Die folgende Liste enthält die Notation zur Beschreibung der Grundtypen von Shop-Scheduling-Problemen im α-Feld:

[3] Falls nicht jeder Job alle Maschinen durchlaufen soll, wählt man die Bearbeitungszeit 0.

Fm (flow shop): Es gibt m verschiedene Maschinen in Serie, welche von den Jobs in derselben vorgegebenen Reihenfolge durchlaufen werden, d. h. für jeden Job ist dieselbe Arbeitsgangfolge vorgegeben. Jeder Job kann dabei zur gleichen Zeit nur von einer Maschine bearbeitet werden, genauso wie jede Maschine zum selben Zeitpunkt nur einen Job bearbeiten kann.

Jm (job shop): Es gibt wie im Flow-Shop m verschiedene Maschinen, aber für jeden Job ist eine individuelle Arbeitsgangfolge vorgegeben.

Om (open shop): Es gibt wie im Flow-Shop m verschiedene Maschinen, auf denen jeder Job bearbeitet werden muss. Die Arbeitsgangfolge der Jobs ist dabei jedoch beliebig.

Des Weiteren können auch Kombinationen aus Parallelmaschinen- und Shop-Scheduling-Problemen auftreten. Gibt es z. B. in einem Flow-Shop nicht m einzelne Maschinen sondern c verschiedene Maschinenstufen, wobei jede Stufe aus mehreren parallelen Maschinen besteht und jeder Job nur von einer Maschine jeder Stufe bearbeitet werden muss, so spricht man von einem *Hybrid Flow-Shop-Problem*.[4] Probleme dieses Typs sind Gegenstand der vorliegenden Dissertation und werden daher in Kapitel 3 detailliert diskutiert. Darüber hinaus gibt es noch viele andere Ausprägungen der drei Grundtypen von Maschinenumgebungen. Da sie für diese Arbeit jedoch nicht von Bedeutung sind, verzichten wir hier auf weitere Ausführungen.

2.1.2.2. Zusätzliche Restriktionen (β)

In der Praxis sind vielfältige Nebenbedingungen denkbar, die bei der Lösung eines Scheduling-Problems zusätzlich berücksichtigt werden müssen. Solche Restriktionen werden im β-Feld notiert, wobei hier multiple Einträge β_1, β_2 usw. möglich sind. Wir listen an dieser Stelle erneut nur die wichtigsten, für uns relevanten Möglichkeiten auf:

prec (precedence constraints): Bei der Verplanung der Jobs sind Reihenfolgebedingungen zu berücksichtigen. Diese werden meist in Gestalt eines gerichteten Graphen definiert, wobei eine Kante der Form $j \rightarrow k$ beschreibt, dass Job j vor Job k zu bearbeiten ist.

no-wait (no waiting time): Zwischen der Bearbeitung mehrerer Operationen desselben Jobs darf keine Wartezeit entstehen.

block (blocking constraints): Falls zwischen der Bearbeitung mehrerer Operationen desselben Jobs auf verschiedenen Maschinen Wartezeit entsteht, blockiert der entsprechende Job die vorangehende Maschine solange, bis seine Bearbeitung auf der nächsten beginnt.

[4] Analog definiert man ein Hybrid Job-Shop-Problem.

buffer *(limited buffer capacity):* In einem Shop-Scheduling-Problem steht zwischen den verschiedenen Bearbeitungsstufen nur eine eingeschränkte Möglichkeit zur Zwischenlagerung von Jobs zur Verfügung, insbesondere handelt es sich bei No-Wait- und Blocking-Constraints also um Spezialfälle dieser Restriktion. Liegt keine der Nebenbedingungen vor, spricht man von einem Problem mit unbegrenztem Zwischenspeicher (unlimited intermediate buffer).

sds *(sequence-dependent setup times)*: Zwischen der Bearbeitung zweier Jobs j und k (in dieser Reihenfolge) auf derselben Maschine entsteht eine Rüstzeit, die von der Sequenz, also der Reihenfolge $j \rightarrow k$, abhängt. Gelegentlich findet man statt *sds* auch die Notation s_{jk} im β-Feld.

r_j *(release dates):* Die Jobs stehen nicht alle zum gleichen Zeitpunkt zur Bearbeitung an, sondern ein Job j kann erst ab dem Zeitpunkt seiner Freigabe r_j bearbeitet werden. Falls diese Restriktion nicht im β-Feld notiert ist, wird für alle Jobs $r_j = 0$ vorausgesetzt.

prmp *(preemptions):* Die Bearbeitung von Jobs darf unterbrochen und zu einem späteren Zeitpunkt (oder zum gleichen Zeitpunkt auf einer geeigneten anderen Maschine) fortgesetzt werden. Gelegentlich findet man für diese Situation auch die alternative Notation *pmnt*.

prmu *(permutations):* Diese Restriktion bezieht sich speziell auf Flow-Shop-Probleme und kennzeichnet, dass die Jobs auf jeder Bearbeitungsstufe in derselben Reihenfolge bearbeitet werden müssen, d. h. die Lösung des Problems reduziert sich auf die Bestimmung einer einzigen Jobsequenz. Man spricht in diesem Fall auch von einem *Permutation Flow-Shop-Problem*.

time-lags *(delays):* Diese Restriktion kennzeichnet, dass die Ausführung zweier aufeinander folgender Operationen desselben Jobs zeitlich verzögert werden muss. Dies ist in verschiedenen Situationen denkbar, z. B. wenn ein Job in einem zusätzlichen Arbeitsgang von einer Maschine zur nächsten transportiert werden muss. In diesem Fall findet man in der Literatur auch häufig die Kennzeichnung $transport$ anstelle von $time-lags$ im β-Feld.

2.1.2.3. Zielfunktionen (γ)

Im γ-Feld werden ein oder mehrere Zielkriterien notiert, bezüglich derer ein Schedule optimiert werden soll. Im Wesentlichen unterscheidet man dabei zwei Typen von Funktionen, die *Min-Max-* und die *Min-Sum-Ziele*. Erstere beschäftigen sich mit der Minimierung eines Maximalwertes, wie z. B. dem Maximum der Fertigstellungszeiten $\max_j C_j$ der Jobs, also gerade dem Zeitpunkt, zu welchem die Bearbeitung sämtlicher Jobs abgeschlossen ist. Diesen nennt man auch *Zykluszeit* oder *Makespan*. Die Minimierung der

Zykluszeit kennzeichnet man im γ-Feld mit dem Eintrag $C_{\max}$. Es handelt sich dabei um die in der Scheduling-Theorie mit Abstand am häufigsten diskutierte Zielfunktion. In der Praxis ist sie beispielsweise dann von Nutzen, wenn die Kapazitätsauslastung eines Systems maximiert werden soll. In einem liefertermin-orientierten Szenario ist dagegen auch $L_{\max} := \max_j L_j$, d. h. die Minimierung der maximalen Terminabweichung der Jobs, ein naheliegendes Optimierungskriterium.

Min-Sum-Ziele werden für die Modellierung von Situationen verwendet, in denen sich jeder einzelne Job auf die Qualität eines Schedules auswirkt. Hier ist unter anderem die *Summe der Fertigstellungszeiten* der einzelnen Jobs ($\sum C_j$) zu nennen. Sind die zu verplanenden Jobs prioritätsbehaftet, so kann man dies ebenfalls in die Zielfunktion einfließen lassen, indem man die einzelnen Summanden unterschiedlich gewichtet, also beispielsweise die Funktion $\sum w_j C_j$, d. h. *die gewichtete Summe der Fertigstellungszeiten*, minimiert. Scheduling-Probleme mit dieser Zielfunktion sind Gegenstand dieser Arbeit und werden im weiteren Verlauf als WCT-Scheduling-Probleme[5] bezeichnet. Analog definiert man die Minimierung der Gesamtverspätung ($\sum T_j$) bzw. der gewichteten Gesamtverspätung ($\sum w_j T_j$). Neben den genannten sind natürlich viele alternative Zielfunktionen denkbar; für eine umfassende Übersicht sei auf Pinedo (2008) [106] verwiesen.

2.2. Lösungsverfahren

Es gibt nur wenige Scheduling-Probleme, die effizient lösbar sind. In den meisten Fällen sind sie zumindest *NP*-schwer[6], insbesondere, wenn es sich um praktisch motivierte Probleme handelt. Im Fall realistischer Aufgaben spielen zudem oftmals weitere Restriktionen eine Rolle, die diese zusätzlich verkomplizieren.

Zur optimalen Lösung kleiner oder mittlerer Instanzen eines Scheduling-Problems kommen daher häufig Branch & Bound Verfahren, dynamische Programmierung oder Ähnliches zum Einsatz (vgl. Pinedo, 2008, [106]), sofern der jeweilige Kontext entsprechend viel Rechenzeit zulässt. Der wesentlich größere Teil scheduling-theoretischer Literatur beschäftigt sich dagegen mit der Entwicklung und Analyse von Heuristiken und Approximationsalgorithmen. Auf die grundlegenden Konzepte solcher Verfahren wird in den folgenden Abschnitten eingegangen. Dabei wird kein Anspruch auf Vollständigkeit erhoben, die Ausführungen bieten lediglich eine kompakte Übersicht der für diese Arbeit relevanten Methoden.

[5] WCT steht dabei für Weighted Completion Time.

[6] Eine umfassende Auflistung von Scheduling-Problemen und ihrer Komplexität ist unter `http://www.mathematik.uni-osnabrueck.de/research/OR/class/` zu finden.

2.2.1. Prioritätsregeln und List Scheduling

Die Begriffe *Prioritätsregel* und *List Scheduling Verfahren* werden in der Literatur häufig vermischt oder in einem Atemzug genannt. Sie stehen insofern in engem Zusammenhang, als dass die Verwendung einer Prioritätsregel zumeist auch die Anwendung eines List Scheduling Verfahrens impliziert und umgekehrt. In diesem Abschnitt werden die Grundideen beider Ansätze präzisiert.

2.2.1.1. Prioritätsregeln

Die einfachste und zugleich am häufigsten verwendete heuristische Vorgehensweise zur Lösung von Scheduling-Problemen jeder Art ist die Verwendung sogenannter Prioritätsregeln[7] (vgl. Blażewicz et al., 1996, [21]). Dabei wird im statischen[8] Fall mit Hilfe einer oftmals sehr einfachen „Faustregel“ ein Prioritätsindex für jeden Job berechnet und mit dessen Hilfe eine Jobsequenz erstellt. Die Jobs werden anschließend in dieser Reihenfolge auf geeignete Weise verplant. Zumeist wird dazu der nächste anstehende Job aus der Liste eingeplant, sobald eine entsprechende Maschine frei wird, um Leerlaufzeiten der Maschinen zu vermeiden. Diese Vorgehensweise bezeichnet man auch *Graham's non-idling* oder *Standard List Scheduling Verfahren* (vgl. dazu auch Abschnitt 2.2.1.2).

Im dynamischen Fall kann man nicht unbedingt mit einer starren Liste arbeiten, sondern die Priorität von Jobs kann sich im Laufe der Planungsperiode in Abhängigkeit des aktuellen Systemstatus' verändern (vgl. dazu auch Pinedo, 2008, [106]). Sowohl zeitabhängige als auch zeitunabhängige Verfahren weisen in vielen Fällen ein sehr gutes Laufzeitverhalten[9] auf, zumeist kann man jedoch nicht ohne Weiteres eine Aussage über die Qualität einer so erzielten Lösung treffen (vgl. dazu Abschnitt 2.2.3). Für manche Scheduling-Probleme konnten hingegen auch Prioritätsregeln entwickelt werden, deren Verwendung sogar eine optimale Lösung liefert.

Beispiel 2.2.1 *Ein klassisches Beispiel ist das Problem* $1||\sum w_j C_j$. *Für dieses Problem kann man mit Hilfe der sogenannten WSPT-Regel*[10]*, auch bekannt unter dem Namen Smith's Rule, eine optimale Jobsequenz bestimmen. Dazu berechnet man für jeden Job den Quotienten aus seiner Bearbeitungszeit und seinem Gewicht* ($\frac{p_j}{w_j}$) *und bearbeitet die Jobs dann in aufsteigender Reihenfolge bezüglich dieser Quotienten. Im Fall* $w_j = 1$ *für alle Jobs* j *(d. h. im ungewichteten Fall) spricht man auch von der SPT-Regel.*

Scheduling-Probleme in der Praxis sind natürlich zumeist wesentlich komplexer, so dass auch entsprechende Heuristiken an die Komplexität der Aufgabenstellung ange-

[7] Engl. priority rules oder dispatching rules

[8] Statisch bedeutet dabei, dass die verwendete Regel nicht zeitabhängig ist, sondern nur auf den Input-Daten basiert.

[9] Dieses hängt zumeist wesentlich von der verwendeten Methode zur Berechnung der Prioritätsindizes ab.

[10] WSPT steht für Weighted Shortest Processing Time first.

passt werden müssen. Einfache Prioritätsregeln liefern dabei jedoch oftmals eine adäquate Grundlage. So werden zur Bearbeitung vielschichtiger Probleme beispielsweise häufig mehrere einfache Prioritätsregeln kombiniert (vgl. Pinedo, 2008, [106]).

2.2.1.2. List Scheduling

List Scheduling ist ein weit verbreitetes Lösungskonzept im Bereich der Scheduling-Theorie. Ein solches Verfahren berechnet für eine gegebene Sequenz[11] von Jobs einen korrespondierenden Schedule. Dazu wird ein Job nach dem anderen in der vorgegebenen Reihenfolge betrachtet und basierend auf dem bis dahin erstellten partiellen Schedule verplant. Zumindest im statischen Fall werden dabei Entscheidungen zu einem späteren Zeitpunkt normalerweise nicht mehr rückgängig gemacht, d. h. wenn ein Job einmal verplant ist, wird er im weiteren Verlauf des Verfahrens nicht mehr neu verplant (vgl. Hurink und Knust, 2001, [58]).

Im Wesentlichen unterscheidet man hierbei die beiden Grundprinzipien, *maschinenorientiertes* und *job-orientiertes List Scheduling*. Im ersten Fall wird üblicherweise wie folgt vorgegangen: Immer wenn eine Maschine einen Job fertiggestellt hat, so wird der nächste Job in der Liste dieser Maschine zugeordnet. Mit anderen Worten, in der Reihenfolge der zugrundeliegenden Jobsequenz wird ein Job immer so früh wie möglich bearbeitet. In diesem Zusammenhang sprechen wir im Folgenden auch von der Standard List Scheduling Strategie. Diese Methode geht auf eine Arbeit von Graham (1969) [45] zurück. Intuitiv ist klar, dass es auf diese Weise nicht möglich ist, eine Maschine leer laufen zu lassen, obwohl eigentlich ein Job zur Bearbeitung ansteht. Daher wird diese Vorgehensweise auch als Graham's non-idling bezeichnet. Sie eignet sich in erster Linie für kapazitäts-orientierte Zielkriterien, wie z. B. die Minimierung der Zykluszeit (vgl. Queyranne und Schulz, 2006, [113]).

Job-orientierte List Scheduling Verfahren lassen dagegen auch dann Leerlaufzeiten der Maschinen zu, wenn eigentlich Jobs zur Bearbeitung anstehen. Genau wie beim maschinen-orientierten List Scheduling geht man von einer zugrundeliegenden Jobsequenz aus und betrachtet die einzelnen Jobs in der Reihenfolge dieser Liste. Diese werden dann jeweils an eine zulässige Stelle im partiellen Schedule eingefügt. Dies muss jedoch, anders als im obigen Fall, nicht unbedingt der frühestmögliche Zeitpunkt sein, d. h. es wird hierbei für einen Job nicht zwingend diejenige Maschine ausgewählt, welche als erstes zur Verfügung steht. Diese Vorgehensweise ist eher für die Minimierung job-bezogener Zielkriterien (z. B. $\sum C_j$) geeignet (vgl. Queyranne und Schulz, 2006, [113]).

Die Möglichkeiten für die Konstruktion von List Scheduling Verfahren sind vielfältig, insbesondere in Bezug auf die Berechnung initialer Jobsequenzen. Auf konkrete Beispiele, insbesondere für Probleme des Typs Hybrid Flow-Shop, wird in den Kapiteln 3 und 5

[11] Diese wurde vorab z. B. mit einer einfachen Prioritätsregel berechnet.

eingegangen.

2.2.2. Metaheuristiken

Als *Metaheuristik* bezeichnet man ein generisches Prinzip zur Entwicklung eines heuristischen Verfahrens. Ein solches ist üblicherweise noch nicht problemspezifisch, sondern muss zunächst an eine konkrete Aufgabe angepasst werden. In der Literatur basieren Metaheuristiken zumeist auf dem Prinzip der *lokalen Suche (Local Search)*, welches im Wesentlichen auf die iterative Verbesserung einer geeigneten Startlösung abzielt (vgl. Voß, 2009, [146]). Gemäß Morton und Pentico (1993) [92] ist ein Local Search Verfahren in vier Schritte gegliedert:

1. Bestimmung einer geeigneten Startlösung (oftmals mit Hilfe einer einfachen Prioritätsregel)
2. Definition einer Lösungsnachbarschaft (Menge „nahe gelegener“ Lösungen)
3. Festlegung eines Kriteriums zur Auswahl einer benachbarten Lösung
4. Definition eines Abbruchkriteriums

In der Grundversion eines solchen Verfahrens werden neue Lösungen normalerweise nur dann akzeptiert, wenn sie besser sind als die vorangehende (vgl. Quadt und Kuhn, 2007, [110]). Da es auf diese Weise häufig zu einer Terminierung des Verfahrens in einem lokalen Optimum kommt, haben sich im Lauf der Zeit verschiedene Ausprägungen lokaler Suche etabliert, welche versuchen, diesen Nachteil zu umgehen. Im Folgenden sind nun nur die beiden Varianten aufgeführt, welche in der Scheduling-Theorie am häufigsten zum Einsatz kommen, insbesondere in Bezug auf Hybrid Flow-Shop-Probleme (vgl. Ruiz und Vázquez-Rodríguez, 2010, [118]).

- *Tabu Search*: Bei der sogenannten Tabu Suche wird für gewöhnlich in jedem Schritt der bestmögliche Nachbar ausgewählt, auch wenn dieser den aktuellen Zielfunktionswert verschlechtert. Um zu verhindern, dass das Verfahren auf diese Weise in einen Kreis gerät, wird der Übergang zu einer (noch zu spezifizierenden) Menge bereits besuchter benachbarter Lösungen verboten (Tabu Liste). Darüber hinaus können auch zwischenzeitlich untersuchte lokale Optima gespeichert werden. Als Abbruchkriterium kann beispielsweise die Anzahl der Iterationen oder auch das Erreichen eines vorab festgelegten Zeitlimits dienen.
- *Simulated Annealing*: Bei einem solchen Verfahren wird der Abkühlungsprozess eines heißen Materials (z. B. Metall) nachempfunden. Ähnlich wie bei der Tabu Suche werden auch hier Verschlechterungen des Zielfunktionswertes beim Übergang zu einer Nachbarlösung zugelassen. Dieser wird hier jedoch randomisiert, d. h. jede

der Nachbarlösungen wird mit einer festgelegten Wahrscheinlichkeit angenommen, welche wiederum mit jeder Iteration sinkt (Abkühlung), damit das Verfahren terminiert.

Neben Local Search Verfahren kommen zur Lösung komplexer Scheduling-Probleme auch immer mehr *Genetische Algorithmen* zum Einsatz, deren Vorbild in der biologischen Evolution liegt.[12] In einem solchen Algorithmus wird eine Population zulässiger Lösungen gleichzeitig behandelt, um so verschiedene Bereiche des Suchraums simultan abarbeiten zu können. Dabei liegt das Darwinistische Prinzip *survival of the fittest* zugrunde (Jungwattanakit et al., 2008, [62]). In jeder Generation werden mit Hilfe der besten Lösungen (most fit individuals) neue Lösungen (children) generiert, indem man die besten Eigenschaften der Eltern kombiniert oder mutiert (Morton & Pentico, 1993, [92]).

Natürlich gibt es noch viele weitere Metaheuristiken, die ebenfalls bei der Lösung von Scheduling-Problemen zum Einsatz kommen können, wie z. B. *Ant Colony Optimization* (ACO), *Neuronale Netze* (NN) oder *Memetische Algorithmen*. Da dies jedoch im Rahmen dieser Arbeit keine Rolle spielt, sei für weitere Informationen beispielsweise auf das Buch von Pinedo (2008) [106] sowie Rixen (1997) [117] verwiesen.

2.2.3. Performance-Analyse

Für heuristische Verfahren ist natürlich die Frage nach der Qualität einer erzeugten Lösung naheliegend. In der Scheduling-Theorie werden in diesem Zusammenhang vornehmlich zwei Arten der Performance-Analyse verwendet. Mehr von akademischem Interesse ist die *Worst Case Analyse* eines Algorithmus', also eine Angabe, wie stark eine approximative Lösung im schlechtesten Fall vom optimalen Zielfunktionswert abweicht.

Definition 2.2.2 *Man bezeichnet einen Algorithmus A als* ρ-Approximation *für ein Optimierungsproblem, wenn der Zielfunktionswert der mit A erzeugten Lösung* ($ZF(A)$) *höchstens* ρ*-mal schlechter ist als der Zielfunktionswert einer optimalen Lösung, d. h. es gilt:*

$$ZF(A) \leq \rho \cdot ZF(OPT).$$

Man spricht in diesem Zusammenhang auch von einer Worst Case Performance Schranke[13].

Besonders interessant sind dabei natürlich möglichst gute Schranken. Bestenfalls kann man eine Probleminstanz konstruieren, für welche der berechnete Wert ρ tatsächlich angenommen wird. In diesem Fall bezeichnet man die Schranke als *strikt*. Grundsätzlich ist eine Worst Case Analyse zwar eher von theoretischem Interesse, da eine schlechte

[12] Genetische Algorithmen gehören daher zur Klasse der sogenannten *Evolutionären Algorithmen*.
[13] Engl. Worst Case Error Bound

Schranke nicht zwangsläufig Rückschlüsse auf eine schlechte Performance eines Verfahrens in der Praxis zulässt. Auf der anderen Seite ist eine gute Worst Case Schranke jedoch zumeist ein Indiz dafür, dass ein Verfahren auch in der Praxis brauchbare Resultate liefert.

In der Existenz einer Worst Case Schranke besteht der Unterschied zwischen approximativen und heuristischen Verfahren (Savelsbergh et al., 2005, [121]). Approximationsalgorithmen besitzen polynomiale Laufzeit und es besteht die Möglichkeit, Worst Case Performance Schranken zu berechnen. Heuristiken weisen ebenfalls polynomiale Laufzeit auf, aber in den meisten Fällen ist keine Worst Case Analyse möglich bzw. kann die heuristische Lösung grundsätzlich beliebig schlecht werden.

Die Güte einer Heuristik wird dann anhand ihrer durchschnittlichen Performance in der Praxis untersucht (vgl. auch Lee & Chen, 2001, [75]). Um diese zu beurteilen, führt man in der Regel eine *empirische Analyse* durch. Für kleinere Probleminstanzen vergleicht man dazu die heuristischen Lösungen meist mit den entsprechenden optimalen Lösungen. Oftmals kommen aber auch (möglichst gute) Schranken als Benchmark zum Einsatz, falls nicht in angemessener Zeit optimale Lösungen berechnet werden können (vgl. Wang, 2005, [148]). Im Rahmen einer empirischen Analyse können unter anderem die folgenden Fragestellungen von Interesse sein:

- Wie groß ist die mittlere/maximale relative Abweichung einer heuristischen Lösung von einer optimalen Lösung/Schranke?
- Welche Rechenzeit (in s/ms) wird benötigt?

Beim Vergleich mehrerer Heuristiken kann es zudem sinnvoll sein, folgende Aspekte zu untersuchen (vgl. Leisten, 1990, [78]):

- Wie groß ist die mittlere/maximale relative Abweichung einer Heuristik von der besten heuristischen Lösung?
- Wie oft liefert eine Heuristik die beste Lösung im Vergleich mit den anderen Heuristiken?
- Wie oft ist die Lösung einer Heuristik maximal um 1% schlechter als die der besten Heuristik?

2.3. Aktive Schedules und reguläre Ziele

Unter gewissen Voraussetzungen müssen bei der Suche nach einem optimalen Schedule nicht sämtliche Möglichkeiten berücksichtigt werden. Von großer Bedeutung ist in dieser Hinsicht die Klasse der regulären Zielfunktionen.

Definition 2.3.1 *Eine Zielfunktion f heißt* regulär*, falls sie mit Hilfe der Fertigstellungszeiten $C_1,\ldots,C_n$ definiert wird und bezüglich dieser nicht monoton fallend ist, genauer: Seien X,Y zwei zulässige Schedules und für alle Jobs j gelte $C_j(X) \leq C_j(Y)$, dann gilt auch:*

$$f(C_1(X),\ldots,C_n(X)) \leq f(C_1(Y),\ldots,C_n(Y)).$$

Falls in einem Scheduling-Problem ein reguläres Ziel minimiert wird, so kann die Suche nach einem optimalen Schedule auf die Klasse sogenannter (semi-)aktiver Schedules beschränkt werden.

Definition 2.3.2 *Ein Schedule heißt* semi-aktiv*, falls eine Verbesserung des Zielfunktionswertes nur durch Vertauschung der Jobreihenfolge auf einer Maschine möglich ist. Ein semi-aktiver Schedule heißt* aktiv*, falls die terminliche Verbesserung eines Jobs nur durch Verzögerung eines anderen realisiert werden kann.*

Für diese Klassen von Schedules gilt nun die folgende Aussage:

Satz 2.3.3 *Für Scheduling-Probleme mit regulärer Zielfunktion gibt es eine optimale Lösung die semi-aktiv/aktiv ist.*

Beweis: Siehe French (1982) [41] bzw. Rixen (1997) [117]. ∎

2.4. On-line Scheduling

On-line Scheduling-Probleme zeichnen sich dadurch aus, dass nicht alle Input-Daten a priori vorliegen, sondern zum Teil erst während des Planungsprozesses bekannt werden. In entsprechenden Lösungsverfahren werden demnach Entscheidungen nur auf der Basis eines Teils der notwendigen Informationen getroffen. Man unterscheidet in der Scheduling-Theorie im Wesentlichen drei verschiedene On-line Paradigmen (vgl. Sgall, 1998, [129] oder Pruhs et al., 2004, [108]):

1. *One-by-one Scheduling*: In diesem Fall sind die Jobs in einer Liste angeordnet. Ein one-by-one Algorithmus berücksichtigt bei einer Entscheidung ausschließlich den ersten Job dieser Liste, für welchen sämtliche Daten bereits vorliegen. Sobald er zur Verplanung des nächsten Jobs in der Liste übergeht, können vorgängig getroffene Entscheidungen nicht mehr revidiert werden.

2. *Unknown processing times*: Hier sind die Bearbeitungszeiten der Jobs nicht vorab festgelegt. Einem entsprechenden Algorithmus ist nur bekannt, ob ein Job sich gerade in Bearbeitung befindet oder nicht. Im Gegensatz zu den anderen Paradigmen kann hier jedoch gegebenenfalls die gesamte Menge zu bearbeitender Jobs bereits zu Beginn berücksichtigt werden.

3. *Jobs arrive over time*: In dieser Situation unterliegen die Jobs individuellen Freigabezeiten. Sämtliche Parameter eines Jobs werden aber erst bei dessen Ankunft im System bekannt. Man spricht diesbezüglich auch von einem dynamischen Scheduling-Problem (vgl. Kadipasaoglu et al., 1997, [64]).

Zur Lösung eines On-line Problems wird häufig eine List Scheduling Strategie eingesetzt, denn bei der Verplanung der Jobs werden diese basierend auf einer Liste einzeln betrachtet und es werden nicht unbedingt Informationen über die anderen (später eintreffenden) Jobs benötigt. Auf Methoden zur Lösung von On-line Scheduling-Problemen, die für das Prozesslabor relevant sind, wird im weiteren Verlauf dieser Arbeit eingegangen.

Analog zum vorangehend eingeführten Begriff der ρ-Approximation misst man die Performance eines On-line Algorithmus' mit Hilfe des sogenannten *Competitive Ratio*.[14] Genauer heißt ein On-line Algorithmus σ-*competitive*, wenn der damit ermittelte Zielfunktionswert für jede beliebige Probleminstanz höchstens σ-mal so groß ist wie der optimale Zielfunktionswert.

2.5. Ergänzende Bemerkungen

In der klassischen Scheduling-Theorie werden zumeist einige Grundannahmen unterstellt, die für praktische Anwendungen eher unrealistisch sind (vgl. Soukhal & Martineau, 2005, [132]). So wird beispielsweise bei Shop-Scheduling-Problemen in den meisten Fällen von unbegrenzter Speicherkapazität zwischen verschiedenen Bearbeitungsstufen ausgegangen, was bei der Produktionsplanung in einer Fabrik oder Ähnlichem offensichtlich eher selten zutreffend ist. Genauso werden nur vereinzelt Beschränkungen von Ankunfts- oder Abfertigungszonen berücksichtigt. Darüber hinaus werden in den wenigsten Arbeiten anfallende Transportzeiten zwischen verschiedenen Maschinen, entsprechende Ladezeiten etc. einbezogen. Da diese Arbeit durch ein konkretes Anwendungsproblem in der Stahlindustrie motiviert ist, besteht ein Ziel darin, einen Beitrag zur Annäherung zwischen Theorie und Praxis in der Scheduling-Theorie zu leisten.

[14] Hier gibt es kein deutsches Äquivalent. Wörtlich übersetzt bedeutet Competitive Ratio soviel wie Konkurrenzfaktor.

3. Hybrid Flow-Shop-Probleme in der Literatur

Unter einem Hybrid Flow-Shop-Problem - in der englischen Literatur auch synonym als *flexible flow shop*, *multiprocessor flow shop* oder *flexible flow line* bezeichnet - versteht man eine Fertigungsstraße mit mehreren parallelen Maschinen auf einer oder mehreren Bearbeitungsstufen, d. h. die verschiedenen Bearbeitungsstufen sind in Serie geschaltet und alle Jobs besitzen dieselbe Arbeitsgangfolge (vgl. Kapitel 2). Auf jeder der Stufen wird eine Maschine zur Bearbeitung eines Jobs ausgewählt. Eine schematische Darstellung des Problems nach Quadt und Kuhn (2007) [110] zeigt Abbildung 3.1.

In der Grundform des Problems sind alle Jobs von Beginn an bekannt und sämtliche Parameter deterministisch. Die parallelen Maschinen jeder Stufe sind zudem identisch und jede Maschine kann zu jedem Zeitpunkt nur einen Job bearbeiten, genauso wie sich jeder Job zu jedem Zeitpunkt nur auf einer Maschine in Bearbeitung befinden darf. Des Weiteren werden keine Rüstzeiten berücksichtigt, es sind keine Job-Unterbrechungen zugelassen und zwischen sämtlichen Bearbeitungsstufen steht unbegrenzter Zwischenspeicher zur Verfügung (vgl. Ruiz & Vázquez-Rodríguez, 2010, [118]). Falls auf jeder der Bearbeitungsstufen nur eine einzelne Maschine vorhanden ist, spricht man von einem *Standard Flow-Shop-Problem*.[1]

Ein Hybrid Flow-Shop-Problem wurde erstmals von Salvador (1973) [119] untersucht. Der Autor hat in seiner Arbeit ein Branch & Bound Verfahren zur Minimierung der Zykluszeit entwickelt (vgl. z. B. Quadt & Kuhn, 2007, [110]). Seitdem wurde es in der Literatur im Gegensatz zu vielen anderen Scheduling-Problemen schon recht häufig behandelt. Allerdings beschränken sich die meisten Arbeiten auf das Problem mit nur zwei Bearbeitungsstufen (vgl. Linn & Zhang, 1999, [84]). Darüber hinaus wird meist die Minimierung der Zykluszeit als Zielkriterium betrachtet. Die Minimierung der (gewichteten) Gesamtfertigstellungszeit oder Gesamtflusszeit hingegen wird wesentlich seltener diskutiert. Meist untersucht man dabei sogar nur die ungewichteten Varianten $\sum C_j$ bzw. $\sum F_j$ und Jobgewichte, die in vielen Anwendungsproblemen eine große Rolle spielen, werden gar nicht erst berücksichtigt (vgl. Tang & Xuan, 2006, [137]). Neben Ruiz und Vázquez-Rodríguez (2010) [118] bieten die Arbeiten von Linn und Zhang (1999) [84], Wang (2005) [148], Quadt und Kuhn (2007) [110], Ribas et al. (2010) [116] sowie die Dissertation von Urlings (2010) [142] weitere umfassende Übersichten zu den behandelten Problemvarianten und möglichen Lösungsverfahren.

[1] Vereinzelt findet man in der Literatur diesbezüglich auch den Begriff *regulärer Flow-Shop*.

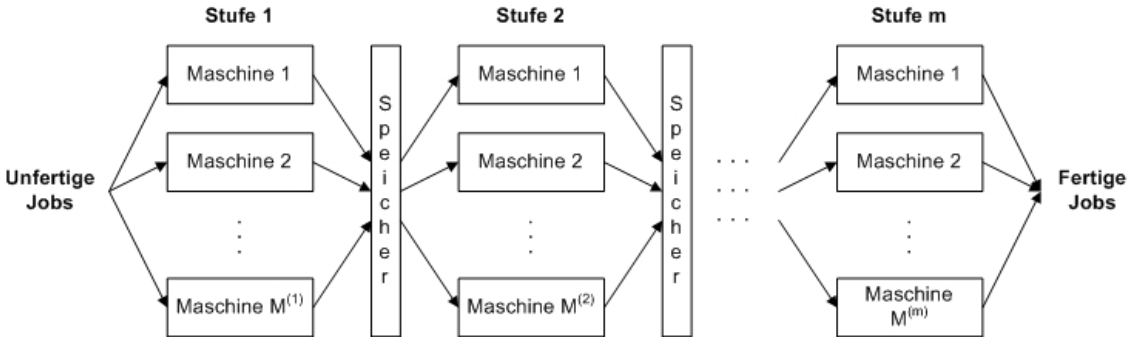

Abbildung 3.1.: Schematische Darstellung eines Hybrid Flow-Shop-Problems

Motiviert durch das eingangs skizzierte Anwendungsproblem steht im Rahmen dieser Arbeit die Minimierung der gewichteten Gesamtfertigstellungszeit (WCT) bzw. der gewichteten Gesamtflusszeit (WFT) im Vordergrund. Im Fall von $r_j = 0$ sind diese beiden Zielfunktionen offensichtlich identisch. Andernfalls sind sie zwar äquivalent bezüglich der Optimalität eines Schedules, jedoch muss ein effektiver Algorithmus zur Minimierung ersterer nicht unbedingt auch zielführend bei der Minimierung letzterer sein und umgekehrt. Beispielsweise in der Arbeit von Savelsbergh et al. (2005) [121] wird dieser Aspekt genauer untersucht.

Im Folgenden diskutieren wir nun die im Kontext dieser Dissertation relevante Literatur mit Fokus auf WCT- bzw. WFT-Minimierung. Dazu wird im nächsten Abschnitt die Verallgemeinerung des 3-Feld-Klassifikationsschemas für Hybrid Flow-Shop-Probleme nach Vignier et al. (1999) [145] vorgestellt. Anschließend werden in Abschnitt 3.2 verschiedene Dekompositionsansätze diskutiert. Die Abschnitte 3.3 und 3.4 widmen sich der Analyse bekannter Lösungsverfahren, wobei der Schwerpunkt im Hinblick auf das hier untersuchte Anwendungsproblem auf heuristischen Ansätzen liegt. Dabei werden insbesondere auch Verfahren für verwandte WCT-Probleme, die beispielsweise als Teilproblem eines HFS-Problems auftreten können, untersucht.

3.1. Notation und Klassifikation

In der Arbeit von Vignier et al. (1999) [145] wird das Klassifikationsschema nach Graham et al. (1979) [46] speziell für Hybrid Flow-Shop-Probleme erweitert, um mehr problemspezifische Merkmale wie z. B. die Anzahl zur Verfügung stehender Maschinen pro Stufe erfassen zu können. Der Arbeit von Ruiz und Vázquez-Rodriguez (2010) [118] kann eine kompakte Zusammenfassung dieser Erweiterung entnommen werden, welche im Folgenden wiedergegeben wird. Im Wesentlichen ersetzt man das α-Feld wie folgt durch ein Quadrupel $(\alpha_1, \ldots, \alpha_4)$.

- α_1 gibt analog zu α den Typ der Maschinenumgebung an, in diesem Fall HF oder

auch FH für Hybrid Flow-Shop.[2]

- α_2 bezeichnet die Anzahl vorhandener Bearbeitungsstufen.
- $\alpha_3 \in \{\emptyset, P, Q, R\}$ gibt die Art der Maschinen auf einer Bearbeitungsstufe an, analog zu Abschnitt 2.1.2.1 steht hier beispielsweise P für identisch parallele Maschinen. Mit $\emptyset$ wird eine einzelne Maschine auf einer Stufe gekennzeichnet.
- α_4 repräsentiert die Anzahl paralleler Maschinen auf einer Stufe. α_4 Maschinen des Typs α_3 auf der k-ten Stufe notiert man dementsprechend mit $(\alpha_3\alpha_4)^k$.

Die Grundform eines Hybrid Flow-Shop-Problems mit Minimierung der Zykluszeit wird dementsprechend als $FHm, (PM^{(k)})_{k=1}^{m}||C_{\max}$ notiert (vgl. dazu auch Ribas et al., 2010, [116]). Für das Standard Flow-Shop-Problem mit nur einer Maschine auf jeder der m Bearbeitungsstufen behalten wir im Folgenden die Notation Fm im α-Feld bei.

3.2. Problemdekomposition

Aufgrund der Komplexität von Hybrid Flow-Shop-Problemen besteht eine häufig praktizierte Vorgehensweise darin, das behandelte Problem nach dem Prinzip *Teile und Herrsche*[3] in mehrere (zumeist zwei) Teilprobleme zu zerlegen und diese konsekutiv zu lösen. Man bezeichnet dies auch als *Dekomposition* des Problems oder hierarchischen Lösungsansatz. Demgegenüber stehen integrierte[4] Ansätze, in welchen die verschiedenen Aspekte eines Problem simultan bearbeitet werden (vgl. Mati et al., 2001, [90]; Sawik, 2000, [123]).

Das wohl gebräuchlichste Dekompositionskonzept wird beispielsweise von Linn und Zhang (1999) oder Wittrock (1988) [154] beschrieben. Man entscheidet dabei im ersten Schritt, welcher Job auf welcher Maschine einer Stufe bearbeitet werden soll. Anschließend bestimmt man im zweiten Schritt die Reihenfolge der zu bearbeitenden Jobs für jede einzelne Maschine. Die Lösungsmöglichkeiten für die erzeugten Teilprobleme sind dabei vielfältig und hängen natürlich von der betrachteten Zielfunktion ab (vgl. dazu auch Baykasoğlu et al., 2004, [15] oder Schuster, 2006, [128]). Quadt und Kuhn (2007) [110] unterscheiden für HFS-Probleme drei grundlegende Arten der Dekomposition:

1. *Stufen-orientierte Dekomposition:* In diesem Fall wird das Problem stufenweise behandelt, d. h. die verschiedenen Bearbeitungsstufen werden eine nach der anderen verplant bzw. für jede muss (basierend auf dem Schedule bereits verplanter Stufen)

[2] Im Französischen entspricht Hybrid Flow-Shop dem Begriff Flow-Shop hybride, was die umgekehrte Anordnung der Buchstaben erklärt (vgl. Vignier et al., 1999, [145]).

[3] Engl. *Divide and Conquer*

[4] Engl. integrated/monolithic approach

ein Parallelmaschinen-Problem gelöst werden. Dabei muss nicht unbedingt von vorne nach hinten vorgegangen werden, sondern die Verplanung kann beispielsweise auch bei einer Stufe begonnen werden, die besonders stark ausgelastet ist und damit einen Flaschenhals des Systems bildet (vgl. Abschnitt 3.4.1.4). Viele stufenorientierte Ansätze verwenden List Scheduling und Prioritätsregeln zur Lösung der einzelnen Parallelmaschinen-Probleme (vgl. Quadt & Kuhn, 2007, [110]).

2. *Job-orientierte Dekomposition:* Bei einer job-orientierten Dekomposition werden die Jobs einzeln und nacheinander betrachtet, d. h. in jedem Schritt wird ein Job ausgewählt und auf geeignete Weise verplant. So kann simultan die Sequenzierung der Jobs auf den Maschinen bestimmt werden.

3. *Problem-orientierte Dekomposition:* Analog zu dem zu Beginn dieses Abschnitts beschriebenen Ansatz, wird ein HFS-Problem im Fall problem-orientierter Dekomposition in Subprobleme zerlegt, die basierend auf der Lösung des vorangehenden konsekutiv gelöst werden. Dabei sind natürlich auch andere Zerlegungen als die oben genannte denkbar. Genauso können die Teilprobleme auch in umgekehrter Reihenfolge bearbeitet werden, d. h. im ersten Schritt wird entschieden, in welcher Reihenfolge die Jobs auf den Stufen bearbeitet werden sollen und darauf basierend werden die Jobs erst konkreten Maschinen zugeordnet. Wie bei allen Arten der Dekomposition besteht auch beim problem-orientierten Ansatz die Schwierigkeit darin, die Zusammenhänge zwischen den verschiedenen Teilproblemen bei der Lösung geeignet zu berücksichtigen.

Als Beispiel für einen stufen-orientierten Dekompositionsansatz ist die Arbeit von Kochhar und Morris (1987) [70] zu nennen. Die Autoren behandeln darin das Problem $FHm,(PM^{(k)})_{k=1}^{m}||\sum w_jC_j$. Zur Lösung werden im ersten Schritt die Startzeiten bzw. die Reihenfolge der Jobs auf der ersten Stufe[5] festgelegt. Anschließend wird für die übrigen Stufen wie folgt vorgegangen: Sobald eine Maschine frei wird, wählt man aus der Menge der Jobs, die auf der vorangehenden Stufe bereits fertiggestellt sind, (auf geeignete Weise) einen aus und bearbeitet diesen auf der frei gewordenen Maschine.

Einen problem-orientierten Ansatz stellen Guinet und Solomon (1996) [48] vor. Im ersten Schritt wird das HFS-Problem auf ein Standard Flow-Shop-Problem reduziert, indem man die Bearbeitungszeiten der Jobs auf den einzelnen Stufen durch die jeweilige Anzahl zur Verfügung stehender paralleler Maschinen dividiert. Die Lösung dieses Problems dient der Sequenzierung der Jobs auf den Bearbeitungsstufen. Basierend darauf erfolgt im zweiten Schritt die Zuordnung der Jobs zu konkreten Maschinen einer Bearbeitungsstufe mit Hilfe von Prioritätsregeln. Ziel des Algorithmus' von Guinet und Solomon ist die Minimierung der Zykluszeit sowie der maximalen Verspätung, falls Liefertermine für die Jobs vorliegen.

[5] In diesem Zusammenhang spricht man auch von Entry Point Sequencing (EPS).

Linn und Zhang (1999) [84] verweisen auf eine alternative problem-orientierte Strategie, welche jedoch nur verwendet werden kann, falls auf jeder Bearbeitungsstufe dieselbe Anzahl paralleler Maschinen zur Verfügung steht. In diesem Fall kann ein HFS-Problem in entsprechend viele Standard Flow-Shop-Probleme zerlegt werden. Jedes wird dann als einzelne Maschine interpretiert, so dass man insgesamt eine Umgebung mit (identisch) parallelen Maschinen erhält. Für diese muss nun entschieden werden, welcher Job auf welcher Maschine, also in welchem der „parallelen Flow-Shops“, bearbeitet werden soll. Die Bearbeitungszeiten der Jobs ergeben sich dabei als Summe der Bearbeitungszeiten auf den verschiedenen Stufen. Im zweiten Schritt wird dann die konkrete Jobsequenzierung für die ausgewählten „Maschinen“ entschieden.

Dekompositionsansätze kommen insbesondere auch häufig im Rahmen anwendungsorientierter Scheduling-Probleme zum Einsatz, da hier meist noch viele weitere Aspekte eine Rolle spielen, welche die Lösung einer Aufgabe zusätzlich erschweren. Beispielsweise Quadt und Kuhn (2005) [109] bearbeiten ein praktisches HFS-Problem aus dem Bereich der Halbleiterherstellung. In diesem sind zum Einen verschiedene Zielfunktionen relevant und zum Anderen ist neben dem eigentlichen Scheduling der Aufträge auch die Losgrößenoptimierung wesentlich. Die Autoren präsentieren ein dreistufiges engpassorientiertes Verfahren. In der ersten Phase werden Produktfamilien konsolidiert, um anfallende Rüstzeiten zu reduzieren. Des Weiteren werden die Losgrößen optimiert sowie die Engpassstufe verplant. Im zweiten Schritt erfolgt die Verplanung der Produktfamilien über die übrigen Bearbeitungsstufen. Erst in der dritten Phase wird die exakte Maschinenauswahl festgelegt sowie ein finaler Schedule innerhalb der Produktfamilien bestimmt.

3.3. Lösungsverfahren für verwandte Probleme

Im Rahmen der hier behandelten HFS-Probleme werden wir zwar nicht direkt mit einem 1-Maschinen- oder Parallelmaschinen-Problem konfrontiert, dennoch sind für uns auch verschiedene Ergebnisse für diese Problemtypen von Bedeutung, da sie beispielsweise bei Verwendung eines Dekompositionsansatzes als Subprobleme entstehen können. Bevor wir daher Lösungsverfahren für HFS-Probleme diskutieren, geben wir an dieser Stelle zunächst einen kompakten Abriss vorhandener Literatur zu 1-Maschinen- und Parallelmaschinen-WCT-Problemen unter besonderer Berücksichtigung von Freigabezeiten. Darüber hinaus behandeln wir in einem zusätzlichen Abschnitt den Stand der Forschung in Bezug auf dynamische WCT-Probleme dieser Gestalt.

3.3.1. WCT-Probleme ohne zusätzliche Restriktionen

Für das Problem $1||\sum w_jC_j$ liefert bekanntermaßen die WSPT-Regel eine optimale Lösung (vgl. Abschnitt 2.2.1.1). Diese einfache Prioritätsregel ist ein wichtiger Ausgangspunkt zur Entwicklung heuristischer Lösungsverfahren für viele komplexere WCT-Pro-

bleme. Für das Problem $Pm||\sum w_jC_j$ kann man beispielsweise mit Hilfe von WSPT eine initiale Jobsequenz bestimmen und diese anschließend unter Anwendung der Standard List Scheduling Strategie verplanen. Für diese Vorgehensweise konnten Kawaguchi und Kyan (1986) [67] folgende Worst Case Performance Schranke nachweisen:

$$\frac{\sum w_jC_j(WSPT)}{\sum w_jC_j(OPT)} < \frac{1}{2}(1+\sqrt{2}).$$

Auch Leung et al. (2006) [81] entwickeln verschiedene heuristische List Scheduling Strategien zur Minimierung von $\sum w_jC_j$ bei identisch parallelen Maschinen, allerdings unter der zusätzlichen Annahme, dass ein Job aus mehr als einer Operation bestehen kann. Da es sich bei $Pm||\sum w_jC_j$ um einen Spezialfall dieses allgemeineren Problems handelt, sind die vorgestellten Verfahren natürlich trotzdem verwendbar. Neben der WSPT-Regel werden hier zwei weitere Prioritätsregeln im Zusammenhang mit verschiedenen maschinen-orientierten List Scheduling Strategien analysiert:

WSLM[6] Hierbei wird zunächst mit Hilfe der sogenannten *LPT-Regel*[7] ein Hilfsschedule erzeugt. Anschließend werden die Jobs in aufsteigender Reihenfolge gemäß der Koeffizienten C^j_{LPT}/w_j sequenziert, wobei C^j_{LPT} die Fertigstellungszeit des Jobs j im zuvor generierten LPT-Hilfsschedule bezeichnet.

MULTIFIT: Die zweite zusätzliche Sequenzierungsstrategie arbeitet ähnlich wie die WSLM-Regel, verwendet zur Erzeugung des Hilfsschedules aber die sogenannte MULTIFIT-Regel. Hierbei handelt es sich um eine iterative Prozedur, deren Lösungsqualität insbesondere von der Anzahl durchgeführter Iterationen abhängig ist. Für detaillierte Informationen sei auf Leung et al. (2006) [81] verwiesen.

Für die WSPT-basierten Verfahren können die Autoren eine gute (für manche Fälle sogar strikte) Worst Case Schranke von $2-\frac{1}{m}$ nachweisen.[8] Des Weiteren zeigen sie, dass die anderen, nicht WSPT-basierten Ansätze zu keiner konstanten Performance Schranke führen können. Die ebenfalls durchgeführte empirische Analyse deutet jedoch darauf hin, dass diese in der Praxis dennoch keine schlechten Ergebnisse liefern. Dies belegt, dass die theoretische Worst Case Schranke nicht grundsätzlich einen Rückschluss auf die tatsächliche Performance eines Verfahrens in der Praxis zulässt. In einer späteren Arbeit erweitern Leung et al. (2008) [82] ihre Ergebnisse, z. B. für Situationen, in denen Jobs nur von bestimmten Maschinen bearbeitet werden können. Außerdem werden Verfahren konstruiert, welche die Jobsequenz nicht vollständig a priori sondern dynamisch bestimmen.

Ein weiteres Beispiel für die erfolgreiche Anwendung der (W)SPT-Regel, hier allerdings für den ungewichteten Fall, präsentieren Yang und Posner (2005) [156]. Für zwei

[6] WSLM steht für Weighted Shortest LPT Makespan first.

[7] LPT steht für Largest Processing Time first, d. h. die Jobs werden absteigend gemäß der Länge ihrer Bearbeitungszeiten sortiert.

[8] Diese Schranke ist schlechter als die von Kawaguchi und Kyan (1986) [67], da hier der allgemeinere Fall, in dem Jobs aus mehreren Operationen bestehen können, untersucht wird.

parallele Maschinen entwickeln die Autoren mit Hilfe einer Kombination aus SPT-, LPT-Regel und List Scheduling approximative Verfahren mit Worst Case Schranken von $\frac{6}{5}$ bzw. $\frac{9}{7}$. Die WSPT-Regel konnte jedoch nicht nur im Rahmen von List Scheduling Strategien geeignet adaptiert werden. So stellen beispielsweise Belouadah und Potts (1994) [17] ein Branch & Bound Verfahren zur Lösung von $Pm||\sum w_jC_j$ vor. Darin werden untere Schranken mit Hilfe einer Lagrange-Relaxation bestimmt, wobei die Berechnung der Lagrange-Multiplikatoren auf einer Anwendung der WSPT-Regel basiert.

3.3.2. WCT-Probleme mit Freigabezeiten

Scheduling-Probleme, die vermeintlich einfacher Natur sind, werden durch das Auftreten von Freigabezeiten häufig *NP*-schwer. Über entsprechende Approximationsalgorithmen ist nur wenig bekannt. So ist beispielsweise schon das 1-Maschinen-Problem $1|r_j|\sum w_jC_j$ nicht mehr effizient lösbar.[9] Zur Approximation dieses Problems zeigt jedoch die Verwendung eines sogenannten α-Schedulers vielversprechende Erfolge. Einen solchen definiert man wie folgt:

Definition 3.3.1 *Es sei S ein Schedule, in welchem Unterbrechungen von Jobs enthalten sein dürfen. Des Weiteren bezeichne $C_j^S(\alpha)$ für $0 < \alpha \leq 1$ den Zeitpunkt, zu dem ein Anteil von α der Bearbeitungszeit eines Jobs j, also gerade $\alpha \cdot p_j$ im Schedule S, fertiggestellt wird. Unter einem α-Scheduler versteht man nun ein Verfahren, welches in Abhängigkeit eines vorgegebenen Wertes für α mit Hilfe von List Scheduling aus dem Schedule S einen Schedule ohne Job-Unterbrechungen generiert. Genauer werden die Jobs in aufsteigender Reihenfolge der Koeffizienten $C_j^S(\alpha)$ sequenziert und anschließend mit Hilfe eines List Scheduling Verfahrens verplant.*

Beispielsweise Chekuri et al. (2001) [33] erhalten für das Problem $1|r_j|\sum C_j$ mit Hilfe eines α-Schedulers mit zufällig gewähltem α eine konstante Worst Case Performance Schranke. Dabei ist die Wahrscheinlichkeitsverteilung, mit deren Hilfe α bestimmt wird, entscheidend für die Berechnung der Schranke. Die Autoren zeigen, dass eine Verteilung existiert, unter deren Verwendung die approximierte Lösung im schlechtesten Fall um einen Faktor von ca. $1,58$ größer ist als der optimale Zielfunktionswert. Auch Savelsbergh et al. (2005) [121] analysieren in ihrer Arbeit verschiedene α-Scheduler Varianten.

Das von Phillips et al. (1998) [104] untersuchte Verfahren entspricht einem α-Scheduler mit $\alpha = 1$. Für das Problem $1|r_j|\sum w_jC_j$ zeigen die Autoren, dass sich der Zielfunktionswert eines fraktionalen Schedules bei der entsprechenden Konvertierung in einen Schedule ohne Unterbrechungen maximal verdoppelt bzw. im Fall des Parallelmaschinen-Problems $Pm|r_j|\sum w_jC_j$ höchstens verdreifacht.[10] Chakrabarti et al. (1996) [32] konnten letztere Aussage bereits verbessern und erhalten im Fall des Parallelmaschinen-Problems

[9] Sogar $1|r_j, prmp|\sum w_jC_j$ ist bereits *NP*-schwer (vgl. u. a. Pinedo, 1983, [105]).
[10] Für das 1-Maschinen-Problem ist diese Schranke sogar strikt.

eine Worst Case Schranke von $\frac{7}{3}$ unter Voraussetzung eines optimalen Schedules für das Problem mit Unterbrechungen. Die Güte einer approximativen Lösung dieser Art hängt demnach von der Qualität des zugrundeliegenden fraktionalen Schedules ab. Es kann zur Lösung eines Problems also durchaus sinnvoll sein, einen möglichst guten Schedule mit Job-Unterbrechungen zu konstruieren und diesen geeignet zu modifizieren.

Zur Generierung eines fraktionalen Schedules für das Problem $Pm|r_j|\sum w_jC_j$ verwenden Phillips et al. (1998) [104] beispielsweise die Lösung der LP-Relaxation einer gemischt-ganzzahligen Formulierung des Problems und erhalten so insgesamt eine $24+\varepsilon$-Approximation des Problems.[11] Diese Schranke ist zwar konstant, aber dennoch nicht sehr vielversprechend. Mit Hilfe von LP-Relaxationen geeigneter Formulierungen konnten im Lauf der 90er Jahre jedoch schon weitaus striktere Worst Case Performance Schranken erzielt werden. Beispielsweise verwendet Schulz (1996) [126] eine (aufwendige) LP-Relaxation, mit deren Hilfe er für das Problem $Pm|r_j|\sum w_jC_j$ eine Schranke von $4-\frac{1}{m}$ herleitet. Für genauere Informationen über die von ihm verwendete Formulierung sei an dieser Stelle auf Kapitel 4 verwiesen.

Ein Approximationsverfahren, welches auch ohne die Lösung einer aufwendigen LP-Relaxation vielversprechende Ergebnisse liefert, stellen Chekuri et al. (2001) [33] vor. Darin wird das Problem $Pm|r_j|\sum w_jC_j$ auf ein 1-Maschinen-Problem relaxiert, indem man die Bearbeitungszeiten der Jobs durch m dividiert. Liegt eine ρ-Approximation für dieses 1-Maschinen-Problem vor, erhält man mit Hilfe einer List Scheduling Strategie für das Parallelmaschinen-Problem eine Worst Case Performance Schranke von $(1+\beta)\rho+(1+1/\beta)$ für das Ausgangsproblem, wobei $\beta>0$ ein wählbarer Parameter aus dem verwendeten List Scheduling Verfahren ist.

3.3.3. Dynamische WCT-Probleme

Die vorangehend genannten Lösungsverfahren beziehen sich sämtlich auf die statische Version der jeweiligen Scheduling-Probleme. Mit anderen Worten, die Jobs unterliegen zwar gegebenenfalls Freigabezeiten, dennoch ist von Beginn an bekannt, dass diese zur Bearbeitung anstehen. Die gebräuchlichste und durchaus erfolgreiche Methode, ein solches Off-line Verfahren in einen On-line Algorithmus[12] umzuwandeln ist denkbar einfach: Alle Entscheidungen werden nur in Bezug auf die bereits freigegebenen Jobs getroffen (vgl. Phillips et al., 1995, [103]). Kommen zu einem späteren Zeitpunkt weitere Jobs hinzu, werden diese nachträglich in die Planung einbezogen, d. h. alle Jobs, die zwar verplant, aber noch nicht in Bearbeitung sind, werden bei der Erstellung eines Schedules für die neu eingetroffenen Jobs erneut berücksichtigt (vgl. dazu auch Huo & Leung, 2005, [57]). In diesem Zusammenhang sprechen wir im Folgenden von der *dynamischen* oder *On-line Version* eines Verfahrens. Diese Vorgehensweise wird auch von Hall et al.

[11] Im Fall des 1-Maschinen-Problems $1|r_j|\sum w_jC_j$ wird analog eine $16+\varepsilon$-Approximation erzielt.
[12] Wir beziehen uns hier nur auf dynamische Probleme mit jobs arriving over time (vgl. Abschnitt 2.4).

(1997) [49] als vielversprechend bewertet. Es gelingt den Autoren auf diese Weise, unter anderem ein On-line Verfahren zur Lösung des Problems $Pm|r_j|\sum w_jC_j$ zu entwickeln, welches einen Competetive Ratio von $4+\varepsilon$ aufweist.

Auch Megow und Schulz (2004) [91] beschäftigen sich mit der dynamischen Version des Problems $Pm|r_j|\sum w_jC_j$. Ein WSPT-basiertes Verfahren liefert dabei für den Fall mit Job-Unterbrechungen einen Competitive Ratio von 2, bzw. $2+\frac{1}{\alpha_m}$ ohne Unterbrechungen.[13] Megow und Schulz setzen die WSPT-Regel jedoch nicht direkt in eine On-line Version um, sondern arbeiten zusätzlich mit verzögerten Jobfreigabezeiten, d. h. die Bearbeitung eines Jobs darf nicht direkt nach seiner Ankunft im System beginnen, sondern es muss noch eine gewisse Zeitspanne abgewartet werden. Auf diese Weise soll die Wahrscheinlichkeit erhöht werden, dass Jobs mit höherer Priorität, die möglicherweise kurze Zeit später eintreffen, trotzdem zuerst bearbeitet werden können (vgl. dazu auch Chekuri et al., 2001, [33]; Anderson & Potts, 2004, [9]; Megow & Schulz, 2004, [91]).

Auch ohne die Berücksichtigung oben genannter Verzögerungszeiten konnten mit Hilfe von Prioritätsregeln bereits Erfolge bei der Lösung dynamischer Scheduling-Probleme erzielt werden. So erhalten beispielsweise Chakrabarti et al. (1996) [32] mit Hilfe der On-line SPT-Regel ein 2-competitive Verfahren für das Problem $P|r_j,prmp|\sum C_j$ und damit (durch Konvertierung des fraktionalen Schedules, siehe oben) eine 6-competitive Strategie für das dynamische Problem ohne Unterbrechungen.

Unter zusätzlicher Beachtung von Jobgewichten weisen Chou et al. (2006) [35] die asymptotische Optimalität der On-line WSPT-Regel (in dieser Arbeit als WSPR für *weighted shortest processing requirement* bezeichnet) für das Problem $Pm|r_j|\sum w_jC_j$ mit dynamisch eintreffenden Jobs nach. Mit anderen Worten, sobald eine Maschine frei wird, wählt man denjenigen verfügbaren Job, der zu diesem Zeitpunkt den kleinsten WSPT-Koeffizienten besitzt. Megow und Schulz (2004) [91] untersuchen darüber hinaus die Performance der WSPT-Regel für das Problem $Pm|r_j,prmp|\sum w_jC_j$, d. h. es werden Unterbrechungen der Jobs zugelassen. Bei Verwendung der sogenannten *preemptive WSPT-Regel* werden in jedem Zeittakt diejenigen verfügbaren Jobs mit dem kleinsten Quotienten aus verbleibender Bearbeitungszeit und Gewicht als nächstes eingeplant. Jobs, die bereits in Bearbeitung sind, dürfen dazu unterbrochen werden. Sowohl im statischen als auch im dynamischen Fall erhält man auf diese Weise eine 2-Approximation bzw. einen Competetive Ratio von 2.

3.4. Lösungsverfahren für HFS-Probleme

In diesem Abschnitt werden nun bekannte Lösungsansätze für das HFS-Problem mit Minimierung der gewichteten Gesamtfertigstellungszeit diskutiert. Aufgrund des zu Beginn

[13] Dabei ist $\alpha_m = \frac{1-m+\sqrt{16m^2+(m-1)^2}}{4m}$ und das Minimum wird für $m \to \infty$ bei ca. 3,28 angenommen.

beschriebenen Anwendungsbezugs dieser Arbeit stehen dabei approximative bzw. heuristische Ansätze im Vordergrund. Vorhandene optimale Lösungsansätze für HFS-Probleme zielen des Weiteren fast ausschließlich auf die Minimierung der Zykluszeit ab und zumeist handelt es sich dabei um Branch & Bound Verfahren (vgl. Quadt & Kuhn, 2007, [110]; Ruiz & Vázquez-Rodríquez, 2010, [118]). Ferner kommen aber auch lineare oder dynamische Programmierung zum Einsatz (vgl. Kadipasaoglu et al., 1997, [64]).

Eine umfassende Übersicht vorhandener exakter Lösungsverfahren bietet die Arbeit von Kis und Pesch (2005) [69], wobei die Zielfunktionen $C_{\max}$ und $\sum C_j$ berücksichtigt werden, erstere jedoch klar im Vordergrund steht. Des Weiteren werden ausschließlich Probleme mit identisch parallelen Maschinen auf den Bearbeitungsstufen diskutiert, da den Autoren für andere Maschinenumgebungen keine exakten Verfahren bekannt sind. Beispiele für häufig in der Literatur zitierte Ansätze zur Minimierung der Zykluszeit sind die Branch & Bound Verfahren von Brah und Hunsucker (1991) [25] oder Moursli und Pochet (2000) [93]. Ersteres konnte von Azizoğlu et al. (2001) [11] erfolgreich zur Minimierung der Funktion $\sum C_j$ adaptiert werden. Rajendran und Chaudhuri (1992) [115] entwickelten ebenfalls ein Branch & Bound Verfahren zur Minimierung der gesamten Bearbeitungszeit ($\sum C_j$) in einer HFS-Umgebung ohne zusätzliche Restriktionen.

Gemäß Ribas et al. (2010) [116] sind die besten bekannten Branch & Bound Algorithmen lediglich in der Lage, Instanzen mit weniger als 15 oder 20 Jobs und maximal 5 Bearbeitungsstufen effizient zu lösen. Im Allgemeinen wird das Problem mit steigender Stufenzahl oder wenn kein eindeutiger Flaschenhals identifiziert werden kann schwerer. Die Berechnung exakter Lösungen spielt im Rahmen der vorliegenden Arbeit aus diesem Grund keine Rolle. Die oben genannten Branch & Bound Verfahren reichen als Einblick in die Thematik daher an dieser Stelle aus.

Auch in der Literatur konzentriert man sich bei der Entwicklung praxisrelevanter Lösungsansätze vorwiegend auf heuristische Verfahren, wie die zu Beginn dieses Kapitels genannten Übersichtsarbeiten unterstreichen. Beispielsweise diskutiert Wang (2005) [148] neben Branch & Bound Verfahren verschiedene Heuristiken und Metaheuristiken, darunter: Neuronale Netze, Genetische Algorithmen und Tabu Search. Auch in der Arbeit von Quadt und Kuhn (2007) [110] liegt der Fokus auf der Untersuchung heuristischer Methoden. Dabei klassifizieren die Autoren die berücksichtigten Arbeiten nach der Art des darin verwendeten Dekompositionsansatzes. Ruiz und Vázquez-Rodríguez (2010) [118] setzen diese Arbeit fort und heben dabei hervor, dass der größte Teil aus der Literatur bekannter Heuristiken für bestimmte Problemtypen maßgeschneidert ist[14], z. B. in Bezug auf eine festgelegte Stufenzahl, eine bestimmte Anzahl von Maschinen auf einer oder mehreren Bearbeitungsstufen oder sogar beides.[15] Ferner wird als Zielkriterium in den meisten Fällen lediglich die Minimierung der Zykluszeit untersucht. Nur 2% der mehr

[14] Engl. tailored heuristics

[15] Wie eingangs bereits erwähnt, beziehen sich die meisten Heuristiken lediglich auf das zwei- (oder auch das drei-) stufige Problem sowie die Minimierung der Zykluszeit.

als 200 von Ruiz und Vázquez-Rodríguez berücksichtigten Arbeiten beschäftigen sich dagegen mit der WCT- oder WFT-Zielfunktion. Im ungewichteten Fall steigt der Anteil zumindest auf 11%. Auch Ribas et al. (2010) [116] stellen eine umfangreiche Übersicht zu Lösungsverfahren für HFS-Probleme bereit. Darunter sind nur vier Literaturverweise zu finden, die sich auf die Minimierung der gewichteten Gesamtfertigstellungszeit beziehen.

Jungwattanakit et al. (2009) [63] vergleichen in ihrer etwas spezieller ausgerichteten Arbeit verschiedene Lösungsstrategien für Hybrid Flow-Shop-Probleme mit unabhängigen parallelen Maschinen auf den Bearbeitungsstufen. Zudem werden Rüstzeiten berücksichtigt. Als Zielfunktion verwenden die Autoren eine Konvexkombination aus der Zykluszeit sowie der Anzahl verspäteter Jobs. Diesbezüglich untersuchen sie sowohl einfache Prioritätsregeln als auch Metaheuristiken wie Simulated Annealing und Genetische Algorithmen.

Im Folgenden diskutieren wir nun verschiedene heuristische Lösungsstrategien im Hinblick auf Hybrid Flow-Shop-Probleme. Dabei liegt der Fokus aufgrund der stark begrenzten Rechenzeit im Kontext des Anwendungsproblems auf einfachen Prioritätsregeln bzw. List Scheduling Ansätzen. In diesem Zusammenhang spielen des Weiteren Transportrestriktionen, begrenzter Zwischenspeicher und dynamisch eintreffende Jobs eine wichtige Rolle. Mithin ist diesen Aspekten anschließend jeweils ein gesonderter Abschnitt gewidmet. Metaheuristische Verfahren und weitere heuristische Strategien werden dagegen nur skizziert, da diese im weiteren Verlauf dieser Arbeit keine Rolle mehr spielen.

3.4.1. Einfache Heuristiken

Dispatching rules are particularly suitable to deal with complex, dynamic, and unpredictable environments and hence their popularity in practice.

(Ruiz & Vázquez-Rodríguez, 2010, [118])

Die Korrektheit dieser Aussage belegen zahlreiche Veröffentlichungen, die sich mit der Lösung konkreter Anwendungsprobleme vom Typ Hybrid Flow-Shop oder Job-Shop beschäftigen. Beispielsweise Adler et al. (1993) [2] untersuchen entsprechende Verfahren für ein Problem in der Verpackungsindustrie. Die Arbeit von Alvarez-Valdez et al. (2005) [8] ist einem Anwendungsproblem in der Glasherstellung gewidmet. Voß und Witt (2007) [147] präsentieren darüber hinaus ein Problem aus der Stahlindustrie und lösen dies mit Hilfe verschiedener Prioritätsregeln, um nur wenige Beispiele aus verschiedenen Anwendungsbereichen zu nennen.

Auch Brah (1996) [24] betont die Stärke einfacher Prioritätsregeln bei der Entwicklung von Entscheidungssystemen in der Praxis. Oftmals ist eine so erzielte Lösung zwar abstrakt und mitunter relativ weit von den tatsächlichen Gegebenheiten entfernt, dennoch

kann sie als Ausgangspunkt zur erfolgreichen Steuerung eines Systems sehr hilfreich sein. Der Autor untersucht in seiner Arbeit zehn einfache Prioritätsregeln und deren Wirksamkeit bei der Lösung des Problems $FHm,(PM^{(k)})_{k=1}^{m}||\sum T_j$.

Im Zusammenhang mit Hybrid Flow-Shop-Problemen wurden in der Literatur bereits diverse Prioritätsregeln in Kombination mit List Scheduling Verfahren diskutiert. Bevor wir uns der Analyse der entsprechenden Quellen widmen, geben wir zunächst eine Auflistung und kurze Beschreibung dieser Regeln. Dabei wird kein Anspruch auf Vollständigkeit erhoben, es werden lediglich häufig genannte Varianten aufgegriffen. Dabei stehen solche, die in erster Linie auf die Minimierung der gesamten (gewichteten) Fertigstellungs- oder Flusszeit abzielen, im Vordergrund.

(W)SPT (Weighted shortest processing time): Die Jobs werden in aufsteigender Reihenfolge ihrer (gewichteten) Bearbeitungszeiten sequenziert (vgl. dazu auch Abschnitt 2.2.1.1), z. B. in: [11, 56, 64, 71, 142, 115].

LPT (Largest processing time): Die Jobs werden in absteigender Reihenfolge ihrer Bearbeitungszeiten sequenziert, z. B. in: [56, 64, 142, 154].

MD (Minimum Deviation): Diese Regel bezieht sich in erster Linie auf das zweistufige Problem, Jobs werden in absteigender Reihenfolge ihrer Differenzen zwischen den Bearbeitungszeiten auf den ersten beiden Stufen sequenziert, d. h. in absteigender Reihenfolge der Koeffizienten MD_j, wobei

$$MD_j = |p_{2j} - p_{1j}| \quad \text{für alle } j,$$

z. B. in: [64].

GCMD (Generalized cumulative minimum deviation): Diese Regel bezieht sich analog zu MD vor allem auf das zweistufige Problem. Jobs mit kleinster Erhöhung der Summe aus Leerlaufzeiten und Wartezeiten erhalten die höchste Priorität, so sollen Warte- und Leerlaufzeiten reduziert werden, z. B. in: [64].

LWR (Least work remaining): Dieser Ansatz kann stufenweise verwendet werden. Für die erste Stufe werden die Jobs in aufsteigender Reihenfolge bzgl. ihrer (verbleibenden) Gesamtbearbeitungszeit sequenziert, entweder diese Sequenz wird beibehalten (dann handelt es sich um eine Verallgemeinerung der SPT-Regel) oder für die übrigen Stufen werden die Jobs mit Hilfe dieser Regel neu sequenziert, z. B. in: [56, 142].

MWR (Most work remaining): Analog zu LWR werden die Jobs in absteigender Reihenfolge bezüglich ihrer (verbleibenden) Gesamtbearbeitungszeit sequenziert, z. B. in: [56, 142].

Diese Liste könnte durch Hinzunahme liefertermin-orientierter Prioritätsregeln (u. a. ATC oder EDD[16]) noch erweitert werden, dazu sei an dieser Stelle jedoch lediglich auf die Arbeiten von Jayamohan und Rajendran (2000) [59] oder Voß und Witt (2007) [147] verwiesen.

Unter den oben genannten Prioritätsregeln werden die SPT- und LPT-Regel mit Abstand am häufigsten adaptiert. Beispielsweise Rajendran und Chaudhuri (1992) [115] entwickeln eine SPT-basierte Heuristik für das zweistufige Problem zur Minimierung der gesamten Flusszeit. Das entwickelte Verfahren wird empirisch analysiert und weist in den durchgeführten Testrechnungen eine durchschnittliche Abweichung von der optimalen Lösung von weniger als 2% auf. Wittrock (1988) [154] präsentiert ein Dekompositionsverfahren zur Lösung von $FHm,(PM^{(k)})_{k=1}^{m}||C_{\max}$, in welchem im ersten Schritt die Maschinenzuordnung für jede Stufe mit Hilfe von LPT vorgenommen wird. Die Abweichung dieser heuristischen von einer optimalen Lösung liegt für das gesamte Verfahren bei 8-17%.

Azizoğlu et al. (2001) [11] stellen in Ihrer Arbeit zwei verschiedene SPT-basierte Heuristiken für das Hybrid Flow-Shop-Problem mit beliebig vielen Stufen vor. In der ersten Variante bestimmt man zunächst für jede einzelne Bearbeitungsstufe die SPT-Sequenz. Anschließend wählt man diejenige der so ermittelten Jobsequenzen aus, welche für das gesamte System (bei Übertragung auf sämtliche Bearbeitungsstufen) den geringsten Zielfunktionswert liefert. Im zweiten der vorgestellten Verfahren wird für die erste Stufe mit Hilfe der SPT-Regel und dem Standard List Scheduling Verfahren ein Schedule bestimmt. Basierend auf den entsprechenden Fertigstellungszeiten verplant man die Jobs nun erneut mit SPT für die nächste Stufe usw. Eine dritte Strategie arbeitet nach demselben Grundprinzip, allerdings wird hier eine andere Prioritätsregel verwendet. Und zwar sequenziert man die Jobs in absteigender Reihenfolge gemäß der Koeffizienten $\sum_{i=1}^{m}(m-i+1)p_{ij}$. Die von den Autoren durchgeführte empirische Analyse liefert für alle drei Verfahren vielversprechende Ergebnisse, insgesamt schneidet die dritte Variante jedoch am besten ab.

Kyparisis und Koulamas (2001) [71] gelingt es, mit Hilfe eines WSPT-basierten Algorithmus' (im Folgenden mit WSPT' bezeichnet) folgende Worst Case Performance Schranke für das Problem $FHm,(PM^{(k)})_{k=1}^{m}||\sum w_jC_j$ zu berechnen:

$$\frac{\sum w_jC_j(WSPT')}{\sum w_jC_j(OPT)} \leq \left(\frac{\sqrt{2}+1}{2}\right)\left\lceil\frac{\sum_k M^{(k)}}{\min_k M^{(k)}}\right\rceil.$$

Es liegt dabei ein Verfahren zugrunde, welches zunächst die Anzahl der Maschinen einer Stufe für alle Maschinentypen auf das Minimum, also die geringste vorhandene Anzahl $\min_k M^{(k)}$, beschränkt und danach entsprechend viele Standard Flow-Shop-Probleme behandelt. Stufen, auf denen mehr als diese minimale Anzahl von Maschinen zur Verfügung steht, werden anschließend geeignet modifiziert. In einer späteren Arbeit (Kyparisis

[16] ATC steht für Apparent Tardiness Costs und EDD für Earliest Due Date.

& Koulamas, 2006, [72]) erweitern die Autoren ihre Ergebnisse für Hybrid Flow-Shop-Probleme mit parallelen Maschinen unterschiedlicher Geschwindigkeit und Minimierung der Zykluszeit.

Eine vergleichende empirische Analyse verschiedener Prioritätsregeln für das HFS-Problem mit identisch parallelen Maschinen bieten beispielsweise die Arbeiten von Hunsucker und Shah (1994) [56], Kadipasaoglu et al. (1997) [64] sowie Brah und Wheeler (1998) [27]. In der erstgenannten werden insbesondere SPT, LPT, LWR und MWR untersucht. Die Autoren berücksichtigen dabei verschiedene Zielkriterien. Für die Minimierung der Zykluszeit sowie der mittleren Flusszeit liefert die SPT-Regel in dieser Studie mit Abstand die besten Ergebnisse. Bei der Minimierung der maximalen Flusszeit kann dagegen kein eindeutiger „Sieger“ ausgemacht werden. Lediglich die LPT-Regel zeigt in allen Fällen keine zufriedenstellende Performance. Allerdings besitzt die SPT-Regel trotz guter Ergebnisse den grundsätzlichen Nachteil, dass manche Jobs unter Umständen sehr lange im System verweilen, da so der Zielfunktionswert insgesamt verbessert wird. In der Praxis ist dies aber möglicherweise nicht tragbar.

Kadipasaoglu et al. (1997) [64] beschränken sich in ihrer Untersuchung auf das zweistufige Problem und berücksichtigen darin neben verschiedenen liefertermin-orientierten Strategien die Prioritätsregeln SPT, LPT, MD und GCMD. Dabei werden unter anderem die Zielfunktionen $\sum C_j$ und $C_{\max}$ untersucht. Die LPT-Regel liefert auch in diesem Fall keine adäquaten Resultate. SPT und MD dagegen führen für das zweistufige Problem fast immer zum selben (guten) Schedule. Im Hinblick auf die Gesamtfertigstellungszeit schneidet die SPT-Regel sogar noch etwas besser ab. Die oben bereits erwähnte Schwäche dieser Regel tritt hier jedoch ebenfalls auf, was insbesondere für liefertermin-orientierte Ziele ein Problem darstellen kann.

Brah und Wheeler (1998) [27] kommen zu ähnlichen Ergebnissen wie die Autoren der beiden vorgängig genannten Arbeiten. Erneut erweist sich die SPT-Regel für viele Instanzen als die beste Wahl im Hinblick auf die Minimierung von $\sum C_j$ oder $C_{\max}$, während die LPT-Regel am schlechtesten abschneidet. Darüber hinaus heben die Autoren jedoch hervor, dass neben der Wahl der Prioritätsregel auch die Charakteristika der jeweiligen Probleminstanz einen nicht zu vernachlässigenden Einfluss auf die Qualität einer Lösung haben können (vgl. dazu auch Wang, 2005, [148]). Dies sollte bei der empirischen Performance-Analyse von Prioritätsregeln demnach auf jeden Fall berücksichtigt werden und motiviert die Entwicklung von *Hybridverfahren*, welche sich die Vorteile mehrerer einfacher Heuristiken gleichzeitig zunutze machen.

Neben der gewählten Prioritätsregel ist für die Lösung eines HFS-Problems auch immer von Bedeutung, auf welche Weise diese angewendet werden soll. Dabei spielen insbesondere folgende Fragestellungen eine Rolle:

- Soll jede Bearbeitungsstufe getrennt mit Hilfe der Regel verplant werden oder bestimmt man nur für die erste (oder eine andere) Stufe einen Schedule und verwendet

die zugehörige Jobsequenz auch für die verbleibenden Stufen?

- Wie soll die Übertragung einer Sequenz dabei vonstattengehen? Soll beispielsweise für jede Stufe exakt die gleiche Sequenz verwendet werden oder arbeitet man möglicherweise nach dem FIFO- oder LIFO[17]-Prinzip (vgl. dazu auch Ding & Kittichartphayak, 1994, [38] oder Hunsucker & Shah, 1994, [56])?

- Soll bei getrennter Verplanung für jede der Stufen dieselbe Prioritätsregel verwendet werden?

Der letzten Frage gehen beispielsweise Jayamohan und Rajendran (2000) [59] auf den Grund. In ihrer Arbeit wird erneut die Effektivität von Prioritätsregeln für das HFS-Problem mit Minimierung von $\sum C_j$ oder $\sum T_j$ untersucht, wobei wiederum die SPT-Regel berücksichtigt wird. Insgesamt kommen die Autoren zu dem Schluss, dass eine individuelle Verplanung der Stufen die besten Ergebnisse liefert. Dies ist nicht weiter verwunderlich, da auf diese Weise natürlich unterschiedliche Charakteristika der Stufen besser berücksichtigt werden können. Je nach Problemstellung kann aber nicht ohne Weiteres jede Stufe autark verplant werden, wie beispielsweise bei No-Wait-Constraints.

Neben den vorangehend diskutierten, sehr einfachen Prioritätsregeln werden zur Lösung von Hybrid Flow-Shop-Problemen auch immer wieder einige elaboriertere heuristische Strategien angewendet. Eine Übersicht und Analyse dieser Ansätze bietet die Arbeit von Brah und Loo (1999) [26]. Darin werden insgesamt fünf populäre Heuristiken zur Minimierung der Zykluszeit sowie der mittleren Fertigstellungszeit untersucht. Die besten Resultate liefern dabei der Algorithmus von Ho sowie die NEH-Heuristik. Beide Verfahren werden daher in den nun folgenden Abschnitten vorgestellt. Im Anschluss daran diskutieren wir engpass-orientierte Methoden sowie heuristische Verfahren, die auf einer Relaxation des Ausgangsproblems basieren.

3.4.1.1. NEH-Heuristik

Die sogenannte NEH-Heuristik[18] geht zurück auf eine Arbeit von Nawaz et al. (1983) [95]. Bei der darin entwickelten Grundversion handelt es sich um eine Erweiterung der LPT-Regel zur Lösung eines Standard Flow-Shop-Problems. Im ersten Schritt werden die Jobs zunächst absteigend in Bezug auf ihre Gesamtbearbeitungszeit ($\sum_i p_{ij}$) sequenziert. Die ersten beiden Jobs dieser Sequenz bestimmen den initialen Schedule. Anschließend wird der dritte Job der Sequenz so in den initialen Schedule eingefügt, dass der Zielfunktionswert minimiert wird.[19] Genauso verfährt man nun mit den verbleibenden Jobs

[17] Diese Abkürzungen stehen für First In First Out bzw. Last In First Out.

[18] Dieses Verfahren ist benannt nach den Namen der Autoren: Nawaz, M.; Enscore, E. und Ham, I.

[19] In der genannten Arbeit wird die Zykluszeit minimiert. Das Verfahren kann aber natürlich auch für andere Zielkriterien adaptiert werden (vgl. z. B. Wang & Tang, 2009, [149]).

der LPT-Sequenz. Dabei bleibt die relative Position der bereits verplanten Jobs in jedem Schritt unverändert.

Urlings (2010) [142] vergleicht in seiner Dissertation heuristische Lösungsverfahren für ein Hybrid Flow-Shop-Problem unter Berücksichtigung verschiedener zusätzlicher Restriktionen, darunter insbesondere sequenz-abhängige Rüstzeiten und Reihenfolgebedingungen. In der Klasse der einfachen Heuristiken erzielt die NEH-Heuristik dabei mit Abstand die besten Ergebnisse. Diese weist aber auch den größten Rechenaufwand auf, da beim Einfügen neuer Jobs in eine partielle Sequenz alle möglichen Positionen überprüft werden müssen. Dennoch ist auch ein NEH-Schedule oftmals noch weit von einer optimalen Lösung entfernt. Aus diesem Grund untersucht Urlings in der genannten Arbeit zusätzlich verschiedene Metaheuristiken. In der Studie von Brah und Loo (1999) [26] liefert neben dem Verfahren von Ho ebenfalls die NEH-Heuristik die besten Resultate. Doch auch hier weisen die Autoren auf den deutlich erhöhten Rechenaufwand gegenüber anderen einfachen Heuristiken hin.

Im Allgemeinen besitzen einfache Heuristiken bereits im Hinblick auf ein Standard (Permutation) Flow-Shop-Problem eine sehr schlechte Worst Case Performance, sofern überhaupt eine solche Schranke ermittelt werden kann (vgl. Smutnicki, 1998, [131]). Auch in dieser Hinsicht schneidet die NEH-Heuristik bei der Minimierung der Zykluszeit im Vergleich am besten ab.

3.4.1.2. Verfahren von Ho

Das Verfahren von Ho (1995) [52] dient in seiner ursprünglichen Version der Lösung des Problems $Fm|prmu|\sum C_j$ und basiert auf der Sequenzierung der Jobs mit Hilfe eines sogenannten Slope-Index'[20], welcher wie folgt definiert ist:

$$S_H(j) = \sum_{i=1}^{m} (m-i+1)p_{ij}.$$

Die Jobs werden zur Bestimmung einer initialen Jobsequenz in absteigender Reihenfolge dieser Koeffizienten sortiert. Diese wird dazu im Anschluss mit einer Prozedur verbessert, die BubbleSort[21] ähnelt. Zwei benachbarte Jobs werden genau dann vertauscht, wenn dadurch eine Verbesserung des Zielfunktionswertes erreicht wird. Diese Verbesserungsphase bricht ab, sobald keine Reduktion der Gesamtflusszeit mehr erreicht werden kann. Auf diese Weise wird ein lokales Optimum erzielt, welches in der dritten Phase des Verfahrens durch Vertauschung nicht benachbarter Jobs weiter verbessert wird. Für weitere Details sei auf die oben genannte Arbeit verwiesen.

[20] In Anlehnung an den ähnlich definierten Slope-Index von Palmer (1965) [100]: $S_P(j) = -\sum_{i=1}^{m} (m-(2i-1))\, p_{ij}$. In Palmers Heuristik zur Lösung von $Fm|prmu|C_{\max}$ werden die Jobs in absteigender Reihenfolge bzgl. dieses Index' sortiert.

[21] Für die Definition dieses Sortierverfahrens sei auf Cormen et al. (2004) [36] verwiesen.

In der vergleichenden Studie von Brah und Loo (1999) [26] liefert die Methode von Ho mit Abstand die besten Resultate, allerdings zieht sie einen noch wesentlich höheren Rechenaufwand als die NEH-Heuristik nach sich.[22]

3.4.1.3. Verwendung von Problemrelaxationen

Aus der Literatur sind zahlreiche heuristische Verfahren bekannt, welche auf einer Vereinfachung des Ausgangsproblems basieren. Entweder wird dabei aus der Lösung einer geeigneten Relaxation direkt eine zulässige Lösung für das Ausgangsproblem konstruiert oder man verwendet die relaxierte Lösung zur Erzeugung einer initialen Jobsequenz für ein List Scheduling Verfahren. Letzteres kommt häufig im Zusammenhang mit der Berechnung von Worst Case Performance Schranken zum Einsatz, wobei entsprechende Verfahren aufgrund hoher Rechenzeiten meist nicht zur Lösung praktischer Scheduling-Probleme geeignet sind. Insgesamt hängt der Rechenaufwand aber natürlich wesentlich von der Art der verwendeten Relaxation ab.

Beispielsweise Liu et al. (2005) [86] relaxieren das Problem $Fm|r_j|\sum w_jC_j$ auf ein verwandtes 1-Maschinen-Problem. Darin wird als Bearbeitungszeit der Jobs deren durchschnittliche Bearbeitungszeit über sämtliche Stufen gewählt. Für dieses 1-Maschinen-Problem wird mit Hilfe der On-line Version der WSPT-Regel[23] eine Jobsequenz bestimmt. Diese Jobfolge definiert nun einen Permutation Schedule für das Ausgangsproblem. Die Autoren weisen in ihrer Arbeit die asymptotische Optimalität (in Bezug auf die Jobanzahl) des Verfahrens nach, allerdings konvergiert die Lösung unter Umständen sehr langsam. Darüber hinaus verschlechtert sich die Performance mit wachsender Anzahl der Bearbeitungsstufen.

Neben einer solchen, sogenannten kombinatorischen Schranke kommt ebenso häufig die LP-Relaxation einer geeigneten gemischt-ganzzahligen Formulierung des Problems zum Einsatz. Mit Hilfe der entsprechenden Lösung wird dann eine initiale Jobsequenz für ein List Scheduling Verfahren konstruiert. Auf diese Weise gelingt beispielsweise Schulz (1996) [126] die Herleitung einer Worst Case Performance Schranke für das Problem $FHm,(PM^{(k)})_{k=1}^{m}|r_j|\sum w_jC_j$. Im Fall des Standard Flow-Shop-Problems präsentiert er ähnliche Schranken unter zusätzlicher Berücksichtigung von Reihenfolgebedingungen. In Kapitel 4 der vorliegenden Arbeit, welches der LP-basierten Berechnung unterer Schranken für verschiedene WCT-Probleme gewidmet ist, werden die verwendeten Formulierungen erneut aufgegriffen, weshalb wir an dieser Stelle nicht weiter ins Detail gehen. Insgesamt ist die beschriebene Vorgehensweise auf Grund des hohen Rechenaufwands allerdings eher von akademischem Interesse und zur Lösung eines Anwendungsproblems mit großen Datenmengen nicht praktikabel.

[22] In der genannten Arbeit wächst die durchschnittliche Rechenzeit gegenüber der NEH-Heuristik etwa um den Faktor zehn.

[23] Die On-line Version der WSPT-Regel ist asymptotisch optimal für das Problem $1|r_j|\sum w_jC_j$ (vgl. Chou et al., 2006, [35]).

Vereinzelt kommen in der Literatur auch Lagrange-Relaxationen (LR) zur Lösung von HFS-Problemen zum Einsatz. In Verbindung mit List Scheduling liefern diese oftmals bessere Approximationen als einfachere Strategien zur Sequenzierung der Jobs, haben auf der anderen Seite aber natürlich dementsprechend höhere Laufzeiten. Beispielsweise in den Arbeiten von Tang et al. (2002, 2006) [139, 138] werden drei LR-basierte Verfahren zur Minimierung der gewichteten Gesamtfertigstellungszeit in einem Hybrid Flow-Shop-Problem vorgestellt.

3.4.1.4. Engpass-orientierte Verfahren

In einem Produktivsystem bestehen oftmals erhebliche Schwankungen im Hinblick auf den Auslastungsgrad der verschiedenen Bearbeitungsstufen. Dies kann beispielsweise durch eine geringe Anzahl zur Verfügung stehender Maschinen eines bestimmten Typs bedingt werden oder durch lange Bearbeitungsgänge auf einer oder mehreren Bearbeitungsstufen. In den vorgängig beschriebenen heuristischen Verfahren werden diese Aspekte bislang nicht berücksichtigt. Verschiedene Studien belegen jedoch, dass es durchaus sinnvoll sein kann, bei der Verplanung der Maschinen nicht einfach der Reihe nach vorzugehen, sondern sich zunächst auf solche zu konzentrieren, bei denen es zu Engpässen[24] (auch *Flaschenhälse* genannt) kommen kann und darauf basierend die übrigen Maschinen zu verplanen (vgl. Azizoğlu et al., 2001, [11] sowie Quadt & Kuhn, 2007, [110]).

Das wohl bekannteste Verfahren dieser Art ist die von Adams et al. (1988) [1] entwickelte *Shifting Bottleneck Heuristik* zur Lösung des Problems $Jm||C_{\max}$. Die darin enthaltene Methode zur Identifizierung und Verplanung von Engpässen in einem System erzielt zwar in der Praxis gute Erfolge (vgl. dazu auch Pinedo, 2008, [106]), basiert jedoch auf der wiederholten optimalen Lösung NP-schwerer Probleme des Typs $1|r_j|L_{\max}$ und ist daher sehr zeitaufwendig. Für weitere Informationen verweisen wir an dieser Stelle auf das Buch von Pinedo (2008) [106].

Paternina-Arboleda et al. (2008) [102] konstruieren eine engpass-orientierte Heuristik zur Minimierung der Zykluszeit in einer HFS-Umgebung. Als Flaschenhals des Systems wird diejenige Bearbeitungsstufe gewählt, welche in Bezug auf die zu bearbeitenden Jobs die höchste relative Gesamtauslastung besitzt. Diese wird definiert als Summe der Bearbeitungszeiten der Jobs bezüglich einer Stufe, dividiert durch die Anzahl der darauf zur Verfügung stehenden parallelen Maschinen (vgl. dazu auch Wittrock, 1988, [154]). Für die so identifizierte Engpassstufe wird ein Schedule bestimmt und darauf basierend werden die übrigen Stufen des Systems verplant. Das entwickelte Verfahren ist wesentlich einfacher als die Shifting Bottleneck Heuristik und arbeitet dementsprechend schneller, liefert aber dennoch ähnlich gute Ergebnisse.

Dass engpass-orientierte Ansätze auch in der Praxis erfolgreich eingesetzt werden können, zeigt beispielsweise die Arbeit von Adler et al. (1993) [2]. Das darin untersuchte

[24] Engl. bottleneck

HFS-Problem mit verschiedenen Zielkriterien (u. a. $\sum w_j T_j$) ist durch eine Anwendung bei der Herstellung von Papiersäcken zur Verpackung von Zement und Ähnlichem motiviert. Hier wird zunächst die Engpassstufe mit Hilfe eines List Scheduling Ansatzes verplant. Die dabei zugrundeliegende initiale Jobsequenz[25] wird in den meisten Fällen für die übrigen Bearbeitungsstufen übernommen. Ob für eine Stufe eine andere Sortierung der Jobs verwendet werden soll, wird mit Hilfe eines lokalen Optimalitätskriteriums entschieden.

3.4.2. Begrenzter Zwischenspeicher

Unter dem Oberbegriff *begrenzter Zwischenspeicher* fassen wir im Wesentlichen drei Ausprägungen dieser zusätzlichen Restriktion zusammen (vgl. dazu auch Kapitel 2). Im Fall der *No-Wait-Constraints* ist gar kein Zwischenspeicher vorhanden, zwischen den einzelnen Arbeitsgängen eines Jobs darf darüber hinaus aber auch keine Wartezeit entstehen. Wenn zumindest letzteres gestattet ist, steht zwar keine Puffermöglichkeit zur Verfügung, ein Job kann eine Maschine nach seiner dortigen Fertigstellung jedoch bis zu seiner Bearbeitung auf der nächsten Stufe blockieren. In dieser Situation spricht man von *Blocking-Constraints*. Sind zwischen den Bearbeitungsstufen kapazitätsbeschränkte Puffer gegeben, liegt ein Problem mit *endlichem Zwischenspeicher* vor. Dabei wird gelegentlich noch zwischen *lokalem* oder *globalem Speicher* unterschieden, je nachdem ob zwischen jeweils zwei Stufen ein eigener Puffer vorhanden ist oder ob ein (einziger) Puffer für das gesamte System zur Verfügung steht. In der Praxis treten Probleme ohne oder mit begrenztem Puffer beispielsweise bei der Stahlerzeugung (vgl. Wang & Tang, 2009, [149]) oder in der chemischen Industrie auf (z. B. zur Vermeidung der Kontamination von Produkten, vgl. Liu & Karimi, 2008, [87]).

In Bezug auf Hybrid Flow-Shop-Probleme mit begrenztem Zwischenspeicher gibt es nur wenige Resultate in der Literatur (vgl. Liu & Karimi, 2008, [87]). Ribas et al. (2010) [116] nennen in ihrer umfassenden Übersichtsarbeit keine einzige Quelle, die ein HFS-Problem mit No-Wait-Constraints behandelt, und dabei mehr als zwei Bearbeitungsstufen berücksichtigt. Des Weiteren wird in den bekannten Arbeiten fast ausschließlich die Minimierung der Zykluszeit als Zielfunktion betrachtet (vgl. Brucker et al., 2006, [28]). Beispielsweise stellt die Arbeit von Leisten (1990) [78] eine Übersicht von Formulierungsmöglichkeiten für Standard Flow-Shop-Probleme mit begrenztem Zwischenspeicher und $C_{\max}$-Minimierung sowie entsprechenden heuristischen Lösungsansätzen bereit. Dabei ist erwähnenswert, dass unter den berücksichtigten Verfahren auch hier die NEH-Heuristik zwar die besten Ergebnisse liefert, aber auch mit Abstand den größten Rechenaufwand aufweist. Hall und Sriskandarajah (1996) [50] präsentieren ferner eine Zusammenfassung bekannter Komplexitätsuntersuchungen[26] und Worst Case Analysen solcher und anderer

[25] Die Sequenzierung der Jobs basiert hier auf der sogenannten ATC-Regel, vgl. dazu Pinedo (2008) [106].
[26] Bereits die Probleme $F2|no-wait|\sum C_j$ und $F3|block|C_{\max}$ sind NP-vollständig.

Standard Shop-Scheduling-Probleme.[27] Im Zusammenhang mit HFS-Problemen werden aber auch hier nur zweistufige Szenarien diskutiert.

Als Literaturbeispiele für die heuristische Lösung von HFS-Problemen mit beliebig vielen Bearbeitungsstufen und Minimierung der Zykluszeit sind die Arbeiten von Sawik (1995) [122], Thornton und Hunsucker (2004) [140] sowie Wardono und Fathi (2004) [150] zu nennen. In erstgenannter wird ein iterativer (step-by-step) Ansatz zur Behandlung von $FHm,(PM^{(k)})_{k=1}^{m}|block|C_{\max}$ vorgestellt. Darin wird der jeweils nächste zu verplanende Job mit Hilfe einer einfachen Greedy-Strategie[28] ausgewählt und in den aktuellen Schedule integriert. Thornton und Hunsucker (2004) [140] bearbeiten dasselbe Problem und entwickeln zu dessen Lösung eine List Scheduling Strategie. Zur Sequenzierung der Jobs werden dabei mehrere einfache Prioritätsregeln simultan berücksichtigt und der beste so erzeugte Schedule ausgewählt (Hybridverfahren). Die Autoren der dritten Arbeit konstruieren darüber hinaus ein Tabu Search Verfahren zur Lösung dieses Problems.

Die einzigen uns bekannten Quellen, worin in Verbindung mit begrenzter Puffermöglichkeit die Zielfunktion $\sum w_j C_j$ untersucht wird, sind die Arbeiten von Tang und Xuan (2006) [137] sowie Wang und Tang (2009) [149]. In ersterer wird das Problem mit endlichem Zwischenspeicher auf ein Hybrid Flow-Shop-Problem mit Blocking-Constraints reduziert, indem man die Puffer als Maschinen modelliert und für die Jobs darauf jeweils die Bearbeitungszeit Null festlegt. Das entstehende Modell wird dann mit Hilfe einer Lagrange-Relaxation gelöst. Die Autoren des zweiten Papers entwickeln ein Tabu Search Verfahren zur Lösung desselben Problems. Darin wird die NEH-Heuristik zur Berechnung der Startlösung verwendet.

3.4.3. Transportrestriktionen

Bereits das zweistufige Standard Flow-Shop-Problem mit Minimierung der Zykluszeit wird unter Transportrestriktionen *NP*-schwer. Soukhal et al. (2005) [133] und Yuan et al. (2007) [158] beweisen darüber hinaus, dass einige Spezialfälle dieses Problems ebenfalls bereits *NP*-schwer sind. Für weitere Literaturverweise zu verschiedenen Scheduling-Problemen mit Transporten, insbesondere zweistufigen Standard Flow-Shop-Problemen, sei auf die Arbeit von Lee und Chen (2001) [75] verwiesen.

Quellen, die HFS-Probleme mit Transportrestriktionen behandeln, werden in den zu Beginn dieses Kapitels genannten Übersichtsarbeiten entweder gar nicht oder wie bei Ribas et al. (2010) [116] nur am Rande erwähnt. Neben der vorgängig bereits diskutierten Arbeit von Tang et al. (2002) [139] (LR-basiertes Verfahren) wird hier lediglich noch auf Xuan und Tang (2007) [155] sowie Botta-Genoulaz (2000) [23] verwiesen. In dem zweiten Paper wird das Problem $FHm,(PM^{(k)})_{k=1}^{m}|time-lags|L_{\max}$ untersucht. Dabei kann

[27] Dabei werden verschiedene Restriktionen und Zielkriterien berücksichtigt.

[28] Der nächste Job wird so ausgewählt, dass die gesamte Leerlaufzeit der Maschinen im partiellen Schedule minimal ist.

die Restriktion $time - lags$ genauso Transporte wie auch Rüstzeiten repräsentieren. Die Autoren evaluieren zur Lösung dieses Problems sechs liefertermin-orientierte Heuristiken. Xuan und Tang (2007) [155] dagegen modellieren das m-stufige Problem mit einer allgemeinen Zielfunktion, die von den Fertigstellungszeiten der Jobs abhängen muss, als ganzzahliges Optimierungsproblem. Dieses wird mit Hilfe einer Lagrange-Relaxation und dynamischer Programmierung für die entstehenden Subprobleme gelöst.

Ruiz und Vázquez-Rodríguez (2010) [118] verweisen ferner auf eine Arbeit von Naderi et al. (2009) [94]. Darin wird basierend auf Simulated Annealing eine Heuristik zur Lösung eines HFS-Problems mit $\sum C_j / \sum T_j$-Minimierung sowie sequenz-abhängigen Rüstzeiten und Transporten entwickelt. Die Transportzeiten hängen dabei nur von den jeweiligen Bearbeitungsstufen ab, zwischen denen transportiert wird, nicht aber von den konkreten Maschinen. D. h. unabhängig von den ausgewählten Maschinen zweier aufeinander folgender Stufen bleibt die dazwischen anfallende Transportzeit unverändert. Naderi et al. beziehen sich in ihrer Literaturanalyse lediglich auf Quellen, die sich mit Rüstzeiten beschäftigen; es werden keine Referenzen zu HFS-Problemen mit Transporten genannt.

Neben den genannten Quellen ist uns lediglich eine weitere Arbeit bekannt, die das fragliche Problem behandelt. Gröflin et al. (2009) [47] entwickeln ein Tabu Search Verfahren zur Lösung eins Hybrid Job-Shop-Problems mit Blocking-Constraints und Transporten. Dieses basiert auf einer Verallgemeinerung des Konzepts disjunktiver Graphen.[29] Ein (Standard) Flow-Shop-Problem mit Blocking-Constraints und (Roboter-)Transporten wird in der Literatur auch als *Robotic Cell*[30] bezeichnet (vgl. z. B. Dawande et al., 2005, [37]).

3.4.3.1. Robotic Cells

Eine umfassende Literaturübersicht sowie ein Klassifikationsschema für Robotic Cells beinhaltet die Arbeit von Dawande et al. (2005) [37]. Die meisten Autoren, die sich mit diesem Thema beschäftigen, beziehen sich allerdings auf die Bestimmung und Optimierung zyklischer Schedules.[31] Ein solcher wird über eine wiederholbare Sequenz von Roboterbewegungen zum Transport aller Jobs durch die Robotic Cell definiert. Zumeist soll dabei die Gesamtlänge einer solchen Bewegungsfolge bzw. eines solchen Zyklus' minimiert werden.[32] Wir werden auf die entsprechende Literatur daher an dieser Stelle nicht genauer eingehen. Für weitere Informationen sei neben der genannten Arbeit auch auf

[29] Da dieses Konzept im Rahmen der vorliegenden Arbeit nicht verwendet wird, sei für genauere Informationen z. B. auf das Buch von Pinedo (2008) [106] verwiesen.

[30] Bereits eine Robotic Cell mit nur drei Maschinen und einem einzigen Roboter ist streng *NP*-schwer (vgl. Soukhal & Martineau, 2005, [132]).

[31] Engl. cyclic schedules

[32] Dieses Optimierungsziel ist nicht zu verwechseln mit der Minimierung der schon häufig genannten Zykluszeit ($C_{\max}$), d. h. der Gesamtlänge eines Schedules.

Agnetis und Pacciarelli (2000) [3] sowie Brucker und Kampmeyer (2008) [29] verwiesen.

Das Paper von Carlier et al. (2010) [31] ist nach Angaben der Autoren die erste Arbeit zum Thema Robotic Cells, welche sich nicht auf die Suche nach zyklischen Lösungen beschränkt. Das darin vorgestellte Lösungsverfahren basiert auf einem Dekompositionsansatz mit zwei Subproblemen, die konsekutiv behandelt werden. Im ersten Schritt wird ein Standard Flow-Shop-Problem mit Blocking-Constraints gelöst. Die Autoren berücksichtigen bereits die notwendigen Transporte, setzen dabei allerdings voraus, dass der Transportroboter immer zur Verfügung steht, sobald er benötigt wird.[33] Erst im zweiten Schritt wird dann die tatsächliche Route für den Roboter durch Lösung eines entsprechenden 1-Maschinen-Problems mit Reihenfolgebedingungen und Verzögerungen bestimmt. Ziel ist dabei die Minimierung der Zykluszeit. Beide Teilprobleme werden im vorgestellten Verfahren mit Hilfe von Branch & Bound Strategien optimal gelöst.

3.4.4. Dynamische Probleme

Auch über dynamische Hybrid Flow-Shop-Probleme wurde bisher nur sehr wenig Literatur publiziert. Die Übersichtsarbeiten von Pruhs et al. (2004) [108] und Sgall (1998) [129] zum Thema On-line Scheduling liefern diesbezüglich gar keine Referenzen. Erst den aktuellen Arbeiten von Ribas et al. (2010) [116] sowie Ruiz und Vázquez-Rodríguez (2010) [118] können wenige Verweise auf entsprechende Veröffentlichungen entnommen werden. Beispielsweise entwickeln Havill und Mao (1998) [51] einen Algorithmus zur Lösung des Problems $FHm,(QM^{(k)})_{k=1}^{m}||C_{\max}$ mit dynamisch eintreffenden Jobs. Für jeden neuen Job wird als nächstes die schnellste verfügbare Maschine ausgewählt. Die Methode wird anschließend für das zweistufige Problem weiter verfeinert um diesbezüglich noch eine bessere Performance zu erzielen.

Yang et al. (2004) [157] bearbeiten das Problem $FHm,(PM^{(k)})_{k=1}^{m}||\sum T_j$ und konzipieren ein Tabu Search Verfahren, welches zur Bestimmung einer initialen Lösung die EDD-Regel[34] verwendet. Die Methode soll nach Angabe der Autoren zukünftig auch zur Echtzeitsteuerung eines Systems in der Praxis ausgebaut werden. In der durchgeführten Simulation wird die Effektivität des Verfahrens neben der oben genannten Zielfunktion auch für die Kriterien $\sum F_j$ und $C_{\max}$ untersucht. Tang et al. (2005) [136] beschäftigen sich mit demselben Problemtyp, hier soll lediglich anstelle der Zykluszeit der Anteil verspäteter Jobs minimiert werden. Dazu entwickeln die Autoren ein Neuronales Netz und trainieren dieses mit Hilfe verschiedener Simulationen. Des Weiteren werden einfache Prioritätsregeln untersucht, wobei erneut die SPT-Regel mit Abstand die besten Ergebnisse bei Minimierung der Gesamtflusszeit ($\sum F_j$) liefert.

[33] In dieser Arbeit wird von einem einzigen Transportroboter ausgegangen.

[34] EDD steht für Earliest Due Date First, d. h. die Jobs werden in aufsteigender Reihenfolge ihrer Liefertermine sequenziert.

Auch Takaku und Yura (2005) [135] behandeln in ihrer Arbeit die On-line Version eines Hybrid Flow-Shop-Problems, wobei liefertermin-orientierte Ziele im Vordergrund stehen. Hier bedeutet „On-line“ aber nicht nur, dass die Jobs kontinuierlich eintreffen, sondern dass sämtliche Scheduling-Entscheidungen für einen Job immer erst dann getroffen werden, wenn er die jeweilige Bearbeitungsstufe erreicht. Dazu wird eine einfache Greedy-Strategie in Kombination mit einer Prioritätsregel verwendet. Entscheidungen, die einmal getroffen sind, werden zudem grundsätzlich nicht rückgängig gemacht.

Zur Lösung komplexer dynamischer Scheduling-Probleme werden oftmals einfache und möglichst schnelle Heuristiken konstruiert, die bei Ereignissen wie dem Eintreffen neuer Jobs oder dem Ausfall von Maschinen immer wieder neu ausgeführt werden können (Kochhar & Morris, 1987, [70]). In diesem Sinn werden auch viele der vorangehend in Abschnitt 3.4.1 diskutierten einfachen Heuristiken dynamisiert (Voß & Witt, 2007, [147]). In Abhängigkeit des jeweiligen Problems können dabei bereits getroffene Entscheidungen möglicherweise rückgängig gemacht werden (Re-Scheduling). Beispiele für eine solche Vorgehensweise beinhalten die nachfolgend aufgeführten Arbeiten.

Vestjens (1997) [144] untersucht in seiner Dissertation einen Greedy-Algorithmus zur Minimierung der Zykluszeit in einem zweistufigen dynamischen Problem. Kadipasaoglu et al. (1997) [64] analysieren die Performance der in Abschnitt 3.4.1 bereits diskutierten Prioritätsregeln auch für das dynamische Szenario. Auch hier erweist sich die LPT-Regel als eher schwach, während SPT und MD bessere Ergebnisse liefern. Insgesamt sind die zu verzeichnenden Unterschiede zwischen den verschiedenen Regeln aber nicht so deutlich wie im statischen Fall. Auf ähnliche Weise untersucht auch Lee (2009) [76] die Prioritätsregeln EDD, SPT und FCFS[35]. Dabei werden jedoch komplexere als die bisher berücksichtigten Zielfunktionen untersucht. Für nähere Informationen sei auf die genannte Arbeit verwiesen.

Schließlich beschreiben Liu et al. (2005) [86] neben der für das Problem $Fm|r_j|\sum w_jC_j$ in Abschnitt 3.4.1.3 beschriebenen Variante der WSPT-Regel auch die folgenden beiden dynamischen Heuristiken: In der ersten Version wird in jedem Schritt unter den bereits eingetroffenen Jobs als nächstes derjenige verplant, welcher die geringste gewichtete Bearbeitungszeit bezüglich der ersten Bearbeitungsstufe ($\frac{p_{1j}}{w_j}$) besitzt. In der zweiten Strategie wird der nächste zu verplanende Job dagegen gemäß der kleinsten gewichteten Gesamtbearbeitungszeit ($\frac{\sum_i p_{ij}}{w_j}$) ausgewählt. Beide Herangehensweisen sind asymptotisch optimal.

Ein reales Anwendungsbeispiel für die beschriebene Vorgehensweise liefert die Arbeit von Agnetis et al. (1997) [4]. Konkret geht es darin um die Durchsatzsteigerung einer Automobilfabrik. Es handelt sich um ein komplexes HFS-Problem mit verschiedenen Restriktionen, unter anderem Transportzeiten und begrenztem Zwischenspeicher. Des Weiteren treffen die Jobs dynamisch ein. Zur Lösung wird eine problemspezifische Heuristik

[35] First Come First Serve, einfache Greedy-Strategie, analog zu FIFO

konstruiert, wobei neben der Minimierung der Anzahl verspäteter Jobs die Durchsatzmaximierung im Vordergrund steht. Immer wenn ein Job auf einer Maschine fertiggestellt wird, kommt eine Prioritätsregel zur Auswahl der Maschine auf der nächsten Bearbeitungsstufe zum Einsatz.

3.4.5. Metaheuristiken

Da Metaheuristiken im Rahmen dieser Arbeit keine Rolle spielen werden, verzichten wir an dieser Stelle auf eine detaillierte Diskussion der entsprechenden Literatur und verweisen auf die zu Beginn dieses Kapitels genannten Übersichtsarbeiten. Ribas et al. (2010) widmen dem Thema ein eigenes Unterkapitel. Auch bei Ruiz und Vázquez-Rodríguez sowie Quadt und Kuhn (2007) [110] sind ausführliche Informationen über Metaheuristiken zu finden. In der zweiten Arbeit wird außerdem hervorgehoben, dass integrierte Verfahren zur Lösung von Hybrid Flow-Shop-Problemen oftmals metaheuristischer Natur sind.

Häufig zitiert werden unter anderem die folgenden Arbeiten, um nur wenige Beispiele für verschiedenartige Metaheuristiken zu nennen: Kochhar und Morris (1987) [70] bearbeiten das Problem $FHm,(PM^{(k)})_{k=1}^{m}||\sum C_j$ mit Hilfe einer zweistufigen Dekomposition. Das erste der entstehenden Teilprobleme wird mit Hilfe Lokaler Suche gelöst. Al-Anzi und Allahverdi (2002) [5] beschäftigen sich mit einem konkreten Problem, das bei der Optimierung von Internetdiensten mit parallelen Servern entsteht und entwerfen ein Lösungsverfahren, welches unter anderem einen Genetischen Algorithmus beinhaltet. Eine Methode auf der Basis von Simulated Annealing verwenden Naderi et al. (2009) [94] zur Lösung eines HFS-Problems mit sequenz-abhängigen Rüstzeiten und Transporten sowie Minimierung der Gesamtfertigstellungszeit oder Gesamtverspätung.

4. LP-basierte untere Schranken für WCT-Scheduling-Probleme

Zielfunktionen der Gestalt „Min-Sum“, die in der Regel von den einzelnen Jobs abhängen, spielen für viele (praktisch motivierte) Scheduling-Probleme eine große Rolle. Wie vorangehend bereits diskutiert, ist die Minimierung der Gesamtfertigstellungszeit ($\sum C_j$) der Jobs oder allgemeiner deren gewichteter Gesamtfertigstellungszeit ($\sum w_j C_j$) ein häufig untersuchtes Kriterium. Nun sind diese WCT-Scheduling-Probleme, mit Ausnahme des einfachsten Falls $1||\sum w_j C_j$ sowie einiger Spezialfälle, *NP*-schwer (vgl. z. B. Lenstra et al., 1977, [79]). Infolgedessen können für größere Probleminstanzen bekanntlich zumeist nur mit großem Aufwand - wenn überhaupt - optimale Lösungen bestimmt werden. In diesem Kapitel werden daher nun verschiedene Methoden zur Berechnung unterer Schranken für WCT-Probleme vorgestellt und entwickelt. Im Vordergrund steht dabei die Bereitstellung geeigneter Werkzeuge, um auch ohne Kenntnis optimaler Lösungen für die im nachfolgenden Kapitel präsentierten heuristischen Lösungsverfahren eine aussagekräftige empirische Analyse durchführen zu können. Der Fokus liegt in diesem Zusammenhang auf der Untersuchung von LP-Relaxationen verschiedener gemischt-ganzzahliger linearer Modellierungsansätze.

Ein Optimierungsproblem zur Modellierung des einfachsten WCT-Problems $1||\sum w_j C_j$ hat folgende grundlegende Gestalt:

$$\sum_{j \in J} w_j C_j \rightarrow \text{ Min!}$$

so dass

$$C_j \geq p_j \qquad \textit{für alle } j \in J \tag{4.1}$$

$$C_k \geq C_j + p_k \text{ oder } C_j \geq C_k + p_j \qquad \textit{für alle } j,k \in J,\ j \neq k \tag{4.2}$$

Die Schwierigkeit besteht hier natürlich in einer geeigneten Formulierung der disjunktiven Bedingungen (4.2). Eine umfassende Übersicht entsprechender Ansätze am Beispiel des Problems $1|r_j|\sum w_j C_j$ bieten beispielsweise die Arbeiten von Queyranne und Schulz (1994) [112] oder Keha et al. (2009) [68]. In Abhängigkeit der jeweils verwendeten Variablen werden im Wesentlichen vier Varianten unterschieden.

Fertigstellungszeitvariablen (completion time variables): In einer fertigstellungszeitbezogenen (auch als natürlich[1] bezeichneten) Formulierung werden als Variablen wie im obigen Modell die Fertigstellungszeiten der Jobs (C_j) gewählt. Zur Darstellung der disjunktiven Bedingungen kommen dabei üblicherweise problemspezifische Klassen gültiger (Un-)Gleichungen zum Einsatz, welche die Menge der zulässigen Fertigstellungszeitvektoren mehr oder weniger gut beschreiben. Erste Untersuchungen der Polyederstruktur solcher Modelle gehen auf eine Arbeit von Queyranne (1993) [111] zurück. Auf entsprechende Ergebnisse wird im weiteren Verlauf dieses Kapitels detaillierter eingegangen.

Zeit-indizierte Variablen (time-indexed variables): Eine zeit-indizierte Formulierung basiert auf einer Diskretisierung der Zeit bzw. der Planungsperiode $0,\ldots,T$. Dabei ist T eine geeignete obere Schranke (in Abhängigkeit der Problemstellung) für den Planungshorizont. Zumeist werden binäre Variablen der Gestalt x_{jt} verwendet, die sich entweder auf ein Zeitintervall $[t,t+1]$ und den Bearbeitungszeitraum eines Jobs oder dessen Start- bzw. Fertigstellungszeit beziehen. Im ersten Fall nimmt eine Variable x_{jt} den Wert 1 an, falls der Job j im Intervall $[t,t+1]$ bearbeitet wird, und ansonsten 0. Bei der zweiten Möglichkeit gilt $x_{jt}=1$, falls der Job j zum Zeitpunkt t gestartet bzw. fertiggestellt wird, und 0 sonst (vgl. z. B. Voß & Witt, 2007, [147]). Da diese Methoden im weiteren Verlauf des Kapitels eingehend diskutiert werden, verzichten wir an dieser Stelle auf eine genauere Darstellung des Modells.

Die Anzahl der Variablen in einer zeit-indizierten Formulierung hängt wesentlich von der Länge der Planungsperiode ab, ist demnach nur pseudopolynomial und kann je nach Art der Eingabedaten extrem groß werden. Es ist jedoch bekannt, dass die LP-Relaxationen derartiger Modelle im Allgemeinen starke untere Schranken für das jeweils betrachtete Problem liefern (vgl. beispielsweise Savelsbergh et al., 2005, [121]). Möglicherweise können aber selbst diese bereits schwer zu lösen sein. Daher wurde insbesondere im vergangenen Jahrzehnt zunehmend Aufwand in die Entwicklung schnellerer modellspezifischer Lösungsmethoden investiert, um auch größere Probleminstanzen lösen zu können. Hierbei steht Spaltengenerierung basierend auf Dantzig-Wolfe-Dekomposition im Vordergrund (vgl. u. a. van den Akker et al., 2000, [143]; als Weiterentwicklung dieser Arbeit Bigras et al., 2008, [20] sowie Bülbül et al., 2004, [22]). Mithilfe dieser Ansätze werden zeit-indizierte Modelle auch zur Schrankengenerierung im Rahmen einer Branch & Bound Strategie nutzbar.

Zeit-indizierte Formulierungen für WCT-Scheduling-Probleme wurden erstmalig von Dyer und Wolsey (1990) [39] untersucht. Für das 1-Maschinenproblem $1|r_j|\sum w_jC_j$ zeigen die Autoren, dass untere Schranken, welche auf der LP-Relaxation dieser Modelle basieren, stärker sind als andere Ansätze (beispielsweise bestimmte Formulierungen mit Ordnungsvariablen, siehe unten).

[1] Anstelle von *completion time variables* findet man in der Literatur auch häufig den Begriff *natural variables*.

Darüber hinaus spielen zeit-indizierte Formulierungen auch bei der Entwicklung von Approximationsalgorithmen eine große Rolle. Um deren Lösung zu erleichtern, geht man in diesem Zusammenhang auch zu einer Formulierung mit intervall-indizierten Variablen (interval-indexed variables) und Intervallen mit wachsender Länge über (vgl. u. a. Hall et al., 1997, [49]).[2] Auf diese Weise erhält man eine polynomiale Formulierung des relaxierten Problems, jedoch auch schwächere untere Schranken. Im Rahmen dieser Arbeit steht jedoch die Berechnung möglichst guter Schranken als Referenz zur Auswertung heuristischer Lösungen im Vordergrund und weniger der notwendige Rechenaufwand. Aus diesem Grund sehen wir von einer intervall-basierten Modellierung ab.

Ordnungsvariablen (linear ordering variables): Als lineare Ordnungsvariablen bezeichnet man binäre Variablen der Gestalt δ_{kj}, die den Wert 1 annehmen, falls der Job k vor dem Job j bearbeitet wird und 0 sonst. Diese Art der Formulierung wird bereits seit den 50er Jahren untersucht (vgl. Schulz, 1996, [126]). Das Problem $1||\sum w_j C_j$ kann man wegen $C_j = \sum_{k \in J, k \neq j} p_k \delta_{kj} + p_j$ unter Verwendung von Ordnungsvariablen mit Hilfe des folgenden gemischt-ganzzahligen linearen Programms modellieren (vgl. u. a. Keha et al., 2009, [68]):

$$\sum_{j,k \in J,\, j \neq k} w_j p_k \delta_{kj} + \sum_{j \in J} w_j p_j \rightarrow \text{ Min!}$$

so dass

$$\begin{aligned}
\delta_{jk} + \delta_{kj} &= 1 && \text{für alle } 1 \leq j < k \leq n && (4.3)\\
\delta_{jk} + \delta_{kl} + \delta_{lj} &\leq 2 && \text{für alle } j,k,l \in J, j \neq k \neq l \neq j && (4.4)\\
\delta_{jk} &\in \{0,1\} && \text{für alle } j,k \in J, j \neq k && (4.5)
\end{aligned}$$

Für die Einbeziehung von Jobfreigabezeiten ($r_j \neq 0$) reicht das obige Modell für sich noch nicht aus. Es wurden daher verschiedene Ansätze zur Konstruktion gültiger linearer Ungleichungen entwickelt (für Details vgl. Dyer & Wolsey, 1990, [39] oder Nemhauser & Savelsbergh, 1992, [97]), mit deren Hilfe dies zusätzlich realisiert werden kann.

Keha et al. (2009) [68] weisen in einer empirischen Untersuchung der einzelnen Formulierungsmöglichkeiten am Beispiel eines 1-Maschinen-Problems nach, dass die LP-Relaxation des obigen Modells genau wie die zeit-indizierte Variante gute untere Schranken liefert. Dabei werden verschiedene Zielkriterien sowie Probleme mit und ohne Freigabezeiten der Jobs berücksichtigt. Um den Ansatz auf eine komplexere Situation wie z. B. ein Parallelmaschinen-Problem mit identischen Maschinen zu übertragen, ist jedoch zusätzlich eine geeignete Modellierung der Maschinenzuordnung erforderlich. Die obigen Restriktionen (4.3) und (4.4) müssen dann in Abhängigkeit dieser Zuordnung formuliert

[2] Die vollständige Darstellung eines solchen Modells kann ebenfalls Hall et al. (1997) [49] entnommen werden.

werden, d. h. sie müssen nur dann erfüllt sein, wenn die betreffenden Jobs sämtlich auf derselben Maschine bearbeitet werden. Wir definieren zu diesem Zweck eine weitere Variable y_{ikj}, die den Wert 1 annimmt, falls die Jobs j und k beide auf derselben Maschine i bearbeitet werden, und 0 sonst. Die obigen Variablen δ_{jk} versehen wir bei gleicher Definition mit einem zusätzlichen Index i, um den Maschinenbezug abzubilden. Somit kann die Gleichung (4.3) durch die folgenden beiden logischen Restriktionen dargestellt werden:

$$\delta_{ijk} + \delta_{ikj} \leq 2 - y_{ikj} \qquad \textit{für alle } i = 1,\ldots,m, k, j \in J, k \neq j \tag{4.6}$$

$$\delta_{ijk} + \delta_{ikj} \geq y_{ikj} \qquad \textit{für alle } i = 1,\ldots,m, k, j \in J, k \neq j \tag{4.7}$$

In der LP-Relaxation des Problems ist eine derartige Modellierung des Variablenzusammenhangs jedoch nicht mehr bindend und die entsprechende Schranke ist gleichermaßen schwach (vgl. u. a. Bülbül et al., 2004, [22]; Kis & Pesch, 2005, [69] oder Jungwattanakit et al., 2008, [62]). Aus diesem Grund werden wir eine Formulierung der verschiedenen Probleme mit Hilfe von Ordnungsvariablen im weiteren Verlauf nicht berücksichtigen.

Positions- und Zuordnungsvariablen (positional date and assignment variables): In einer positionsbezogenen Formulierung verwendet man erneut binäre Variablen, die sich nun aber auf die Position eines Jobs im Schedule beziehen. Im Fall eines 1-Maschinen-Problems wählt man $u_{jk} = 1$, falls der Job j an k-ter Stelle bearbeitet wird, und 0 sonst. Des Weiteren bezeichne $\gamma_k \geq 0$ die Fertigstellungszeit desjenigen Jobs, welcher sich an der k-ten Position befindet. Die Menge zulässiger Jobanordnungen kann demnach durch die folgenden Restriktionen beschrieben werden:

$$\sum_{k \in J} u_{jk} = 1 \qquad \textit{für alle } j \in J \tag{4.8}$$

$$\sum_{j \in J} u_{jk} = 1 \qquad \textit{für alle } k \in J \tag{4.9}$$

$$\gamma_k \geq \gamma_{k-1} + \sum_{j \in J} p_j u_{jk} \qquad \textit{für alle } k = 2,\ldots,n \tag{4.10}$$

$$\gamma_1 \geq \sum_{j \in J} p_j u_{j1} \tag{4.11}$$

Dieser Ansatz geht zurück auf eine Arbeit von Lasserre und Queyranne (1992) [73] (vgl. auch Schulz, 1996, [126]). Die Bedingungen (4.8) und (4.9) stellen die eindeutige Zuordnung der Jobs zu den Positionen sicher, während (4.10) und (4.11) die Fertigstellungszeiten in Abhängigkeit der Jobanordnung abbilden. Freigabezeiten der Jobs können zudem mit Hilfe der folgenden Ungleichung einbezogen werden:

$$\gamma_k \geq \sum_{j \in J} (p_j + r_j) u_{jk} \qquad \textit{für alle } k \in J \tag{4.12}$$

Um die Zielfunktion $\sum w_j C_j$ mit Hilfe von Positionsvariablen darstellen zu können, muss ferner die Beziehung zwischen den job- und den positionsbezogenen Fertigstellungszeiten modelliert werden. Die Ungleichung

$$C_j \geq \gamma_k - M(1 - u_{jk}) \qquad \textit{für alle } j,k \in J \tag{4.13}$$

stellt diesen Zusammenhang (für genügend großes M) her, ist jedoch im Fall einer LP-Relaxation nicht mehr bindend. Keha et al. (2009) [68] entwickeln aus diesem Grund eine neue Klasse gültiger Ungleichungen, die ohne eine solche *Big-M-Formulierung* auskommt und können so die zugehörigen unteren Schranken verbessern.

Dennoch gilt für Formulierungen mit Positions- und Zuordnungsvariablen Ähnliches wie im Fall der Ordnungsvariablen. Die Methode kann relativ leicht für Probleme verallgemeinert werden, in welchen die Lösung lediglich von einer einzigen Jobsequenz abhängt, wie bei einem 1-Maschinen-Problem oder einem Permutation Flow-Shop-Problem (vgl. Schulz, 1996, [126]; Hoogeveen & van de Velde, 1995, [53]; Hoogeveen et al., 2006, [54]). Für Situationen mit verschiedenen Jobsequenzen (auf verschiedenen Maschinen), insbesondere bei mehreren parallelen Maschinen, sind die obigen Restriktionen jedoch erneut abhängig von der jeweiligen Maschinenzuordnung. Die Verknüpfung der verschiedenen Variablenarten erfordert, wie im Fall der Ordnungsvariablen bereits beschrieben, die Verwendung logischer Restriktionen (vgl. auch Soukhal & Martineau, 2005, [132]), welche in der LP-Relaxation nicht mehr bindend sind. Im weiteren Verlauf dieses Kapitels werden wir infolgedessen auch von der Verwendung einer positionsbezogenen Modellierung absehen.

Aus den oben genannten Gründen liegt der Schwerpunkt bei der Entwicklung und Analyse unterer Schranken in dieser Arbeit ähnlich wie bei Pan und Shi (2007) [101] sowie Savelsbergh et al. (2005) [121] auf der Verwendung natürlicher und zeit-indizierter Modellierungsansätze. Sofern in der Literatur nicht bereits geschehen, werden dazu nun vorhandene Formulierungen für 1-Maschinen-WCT-Probleme zunächst für Parallelmaschinen- und anschließend für (Hybrid) Flow-Shop-Probleme verallgemeinert. Darüber hinaus beinhaltet dieses Kapitel auch einen theoretischen Vergleich der verschiedenen Schranken. Diesbezüglich schon bekannte Resultate, insbesondere aus den Arbeiten von Dyer und Wolsey (1990) [39], Goemans (1996) [43], Queyranne und Schulz (1994) [112] sowie Schulz (1995, 1996) [125, 126], werden dazu an den entsprechenden Stellen zitiert und auf komplexere Umgebungen übertragen. Die Performance der untersuchten zeit-indizierten Formulierungen in der Praxis wird zudem in Kapitel 6 empirisch analysiert.

In den weiteren Abschnitten verwenden wir dazu die folgenden gebräuchlichen Notationen:

$$p(S) := \sum_{j \in S} p_j, \quad p^2(S) := \sum_{j \in S} p_j^2, \quad r(S) := \max_{j \in S} r_j.$$

Dabei bezeichne $S \subseteq J$ eine beliebige Teilmenge der Jobmenge J.

4.1. 1-Maschinen-Probleme

In diesem Abschnitt fassen wir bekannte und für uns relevante Ergebnisse über untere Schranken für das 1-Maschinen-Problem $1|r_j|\sum w_j C_j$ zusammen. Dazu zitieren wir zu-

nächst die folgende Definition:

Definition 4.1.1 *Es sei N eine endliche Menge. Eine Mengenfunktion $f : \mathscr{P}(N) \longmapsto \mathbb{R}$ heißt* supermodular*, falls für alle $S,T \subseteq N$ gilt:*

$$f(S \cup T) + f(S \cap T) \quad \geq \quad f(S) + f(T)$$

Des Weiteren nennt man ein Polyeder $P(f) = \{x \in \mathbb{R}^N : x(A) \geq f(A)$ für alle $A \subseteq N\}$ supermodular, genau dann wenn f eine supermodulare Funktion ist. Dabei definiert man für $x \in \mathbb{R}^N$ und $A \subseteq N$ üblicherweise $x(A) := \sum_{j \in A} x_j$.

In der linearen Optimierung spielen supermodulare Polyeder eine große Rolle, denn sobald der zulässige Bereich eines linearen Minimierungsproblems als supermodulares Polyeder dargestellt werden kann, wissen wir, dass dieses mit Hilfe des Greedy-Algorithmus' für supermodulare Polyeder in $O(n \log n)$ Zeit gelöst werden kann.[3] Eine erste wichtige Aussage über die Polyederstruktur des 1-Maschinenproblems $1||\sum w_j C_j$ stammt aus einer Arbeit von Queyranne (1993) [111].

Satz 4.1.2 (Queyranne, 1993) *Die konvexe Hülle zulässiger Fertigstellungszeitvektoren $C \in \mathbb{R}^n$ für das Problem $1||\sum w_j C_j$ wird vollständig beschrieben durch die folgende Menge linearer Ungleichungen. Dabei sei $g(S) := \frac{1}{2}(p^2(S) + (p(S))^2)$.*

$$\sum_{j \in S} p_j C_j \quad \geq \quad g(S) \qquad\qquad \text{für alle } S \subseteq J$$

Insbesondere ist die Funktion g supermodular. Das zugehörige Optimierungsproblem ist also in polynomialer Zeit lösbar. Daraus folgt zudem, dass es sich bei der WSPT-Regel um einen Spezialfall des Greedy-Algorithmus' für supermodulare Polyeder handelt.

Durch „Shiften" der Funktion g kann man für das 1-Maschinen-WCT-Problem mit Freigabezeiten $(1|r_j|\sum w_j C_j)$ eine ähnliche Klasse gültiger Ungleichungen bestimmen, beispielsweise durch Verwendung der Funktion $l(S) := p(S)r(S) + \frac{1}{2}(p^2(S) + (p(S))^2)$ (vgl. Goemans, 1996, [43]). Diese ist allerdings nicht supermodular. Goemans gelingt jedoch durch den Übergang zur oberen Dilworth-Trunkierung von $l(S)$ die Entwicklung einer supermodularen und damit in Polynomzeit lösbaren Darstellung.[4] Bevor wir den entsprechenden Satz zitieren, geben wir zunächst die Definition der oberen Dilworth-Trunkierung an.

Definition 4.1.3 *Es sei $f : \mathscr{P}(N) \longmapsto \mathbb{R}$ eine Mengenfunktion. Für beliebiges $S \subseteq N$ bezeichnet man den folgenden Ausdruck als obere Dilworth-Trunkierung der Funktion f:*

$$g(S) \quad = \quad \max \left\{ \sum_i f(S_i) : \{S_1, \ldots, S_k\} \text{ Partition von } S \right\}$$

[3] Eine Beschreibung des Algorithmus' kann Goemans (1996) [43] entnommen werden.

[4] Lovász (1983) [88] zeigte zuvor, dass die obere Dilworth-Trunkierung einer Mengenfunktion supermodular ist.

Satz 4.1.4 (Goemans, 1996) *Eine optimale Lösung des folgenden linearen Programms ist eine untere Schranke für die optimale Lösung des Problems* $1|r_j|\sum w_jC_j$.

$$(1a) \qquad \sum_{j\in J} w_jC_j \to \text{Min!}$$

so dass

$$\sum_{j\in S} p_jC_j \;\geq\; h(S) \qquad \text{für alle } S\subseteq J \tag{4.14}$$

Dabei ist h die obere Dilworth-Trunkierung der Funktion

$$l(S) = p(S)r(S) + \frac{1}{2}(p^2(S) + (p(S))^2) \tag{4.15}$$

und damit insbesondere supermodular.

Im Folgenden bezeichne $z^{(1a)}$ den optimalen Zielfunktionswert der LP-Relaxation der Formulierung $(1a)$. Mit Hilfe der Supermodularität der Funktion h kann Goemans (1996) [43] die Äquivalenz dieser unteren Schranke mit der LP-Relaxation der folgenden zeitindizierten Formulierung aus der Arbeit von Dyer und Wolsey (1990) [39] nachweisen.

Satz 4.1.5 (Dyer/Wolsey, 1990) *Eine optimale Lösung des folgenden linearen Modells ist eine untere Schranke für das Problem* $1|r_j|\sum w_jC_j$. *Dabei ist* y_{jt} *eine binäre Variable, welche den Wert 1 annimmt, falls Job* j *im Intervall* $[t,t+1]$ *bearbeitet wird, und* 0 *sonst. Außerdem ist* $y_{jt} = 0$ *für* $t < r_j$ *und* $t = T$.

$$(1b) \qquad \sum_{j\in J} w_jC_j \to \text{Min!}$$

so dass

$$\sum_{j\in J} y_{jt} \;\leq\; 1 \qquad \text{für alle } t = 0,\ldots,T \tag{4.16}$$

$$\sum_{t=0}^{T} y_{jt} \;=\; p_j \qquad \text{für alle } j\in J \tag{4.17}$$

$$C_j \;=\; \frac{p_j}{2} + \frac{1}{p_j}\sum_{t=0}^{T}\left(t+\frac{1}{2}\right)y_{jt} \qquad \text{für alle } j\in J \tag{4.18}$$

Es kann dabei $T \geq \max_j r_j + \sum_j p_j$ *als obere Schranke für den Planungshorizont gewählt werden.*

Die optimale Lösung dieses Modells ist lediglich eine untere Schranke des Problems, da sie nicht unbedingt die zusammenhängende Bearbeitung der Jobs gewährleistet. Im Folgenden bezeichne nun $z^{(1b)}$ den optimalen Zielfunktionswert der LP-Relaxation der obigen Formulierung. Dann gilt die folgende Beziehung:

Satz 4.1.6 (Goemans, 1996) $z^{(1a)} = z^{(1b)}$.

Dies ist insofern bemerkenswert, als dass fertigstellungszeitbezogene Modelle im Allgemeinen schwächere untere Schranken liefern als zeit-indizierte Formulierungen. Im nächsten Abschnitt werden wir jedoch sehen, dass die Aussage nicht ohne Weiteres für Parallelmaschinen-Probleme verallgemeinert werden kann (auch nicht für den einfachsten Fall $Pm||\sum w_jC_j$); zumindest nicht unter Verwendung einer gebräuchlichen Klasse gültiger Ungleichungen aus einer Arbeit von Schulz (1996) [126] (vgl. Satz 4.2.2).

Neben der obigen Darstellung präsentieren Dyer und Wolsey (1990) [39] die folgende (stärkere) zeit-indizierte Formulierung für das Problem:

Satz 4.1.7 (Dyer/Wolsey, 1990) *Eine optimale Lösung des folgenden linearen Modells ist eine optimale Lösung für das Problem* $1|r_j|\sum w_jC_j$. *Dabei ist* x_{jt} *eine binäre Variable, welche den Wert 1 annimmt, falls Job* j *zum Zeitpunkt* t *gestartet wird, und* 0 *sonst. Außerdem ist* $x_{jt} = 0$ *für* $t < r_j$ *und* $t > T - p_j$.

$$(1c) \qquad \sum_{j\in J} w_jC_j \to \text{Min!}$$

so dass

$$\sum_{t=0}^{T} x_{jt} = 1 \qquad \text{für alle } j \in J \tag{4.19}$$

$$\sum_{j\in J}\sum_{s=t-p_j+1}^{t} x_{js} \leq 1 \qquad \text{für alle } t = 0,\ldots,T \tag{4.20}$$

$$C_j = \sum_{t=0}^{T} t x_{jt} + p_j \qquad \text{für alle } j \in J \tag{4.21}$$

Im Folgenden bezeichne nun $z^{(1c)}$ den optimalen Zielfunktionswert der LP-Relaxation des Modells $(1c)$. Zwischen den beiden zeit-indizierten Formulierungen $(1b)$ und $(1c)$ besteht der folgende Zusammenhang:

Satz 4.1.8 (Dyer/Wolsey, 1990) $z^{(1b)} \leq z^{(1c)}$.

Wir werden sehen, dass analoge Aussagen auch für Parallelmaschinen- und (Hybrid) Flow-Shop-Probleme hergeleitet werden können.

Neben diesem Resultat können Dyer und Wolsey (1990) [39] zeigen, dass beide zeit-indizierten Formulierungen stärker sind als verschiedene andere Ansätze, darunter eine Variante mit Ordnungsvariablen. Dieses Resultat veranlasste Sousa und Wolsey (1992) [134] dazu, zeit-indizierte Formulierungen unter dem Gesichtspunkt des Job-Splittings weiter zu untersuchen, d. h. ein Job j wird gegebenenfalls in mehrere kürzere Jobs j_i

aufgespalten und auch sein Gewicht w_j wird geeignet auf die Teiljobs verteilt[5], sodass $\sum w_{j_i} = w_j$ gilt. Da das Problem dabei weiter relaxiert wird, ist eine solche Vorgehensweise nur dann sinnvoll, wenn für die ursprüngliche Relaxation in angemessener Zeit keine Lösung gefunden werden kann (vgl. dazu auch Belouadah et al., 1992, [16]). Wir werden diese Methode mithin nicht weiter verfolgen.

4.2. Identisch parallele Maschinen

Eine Übersicht zu Modellierungsansätzen und Schranken für Parallelmaschinen-Probleme kann der aktuellen Arbeit von Li und Yang (2009) [83] entnommen werden.[6] Hier werden im Wesentlichen erneut die eingangs vorgestellten Modellierungsmethoden (lineare Ordnung, Fertigstellungszeitvariablen, zeit-indizierte und intervall-basierte Formulierung) diskutiert; in diesem Fall lediglich mit direktem Bezug auf Parallelmaschinen-Probleme. Dabei sollte erwähnt werden, dass der vermeintlich einfache Fall $P2||\sum w_jC_j$ dieser Problemklasse bereits *NP*-schwer ist. Dies gilt selbst für eine Relaxation des Problems, in welcher Job-Unterbrechungen zugelassen sind (vgl. Bruno et al., 1974, [30]).

Eine der ersten unteren Schranken für das WCT-Problem mit identisch parallelen Maschinen und ohne zusätzliche Restriktionen ($Pm||\sum w_jC_j$) stammt aus einer Arbeit von Eastman et al. (1964) [40] und hat folgende Gestalt:

Satz 4.2.1 (Eastman et al., 1964) *Es sei C^* ein optimaler Fertigstellungszeitenvektor für das Problem $Pm||\sum w_jC_j$. Des Weiteren bezeichne C^1 eine optimale Lösung des zugehörigen 1-Maschinen-Problems[7] (in polynomialer Zeit mit WSPT-Regel bestimmbar). Dann ist durch folgende Ungleichung eine untere Schranke für den optimalen Zielfunktionswert des Problems gegeben:*

$$\sum_{j\in J} w_jC_j^* \;\geq\; \frac{1}{m}C^1 + \frac{m-1}{2m}\sum_{j\in J} w_jp_j.$$

Diese entspricht der schwächsten unteren Schranke aus einer Arbeit von Webster (1992) [151]. Der Autor entwickelt darin eine Kette verschiedener Schranken für das Problem $Pm||\sum w_jC_j$, welche auf Job-Splitting-Ansätzen basieren. Diese setzen jedoch sämtlich die Optimalität der WSPT-Regel für das Problem $1||\sum w_jC_j$ voraus. Aus diesem Grund sind sie zwar einerseits in polynomialer Zeit berechenbar, können auf der anderen Seite aber nicht ohne Weiteres für die Situation mit Jobfreigabezeiten verallgemeinert werden, da das zugehörige 1-Maschinen-Problem $1|r_j|\sum w_jC_j$ in diesem Fall *NP*-schwer ist.

[5] Die Art der Aufteilung der Gewichte beeinflusst dabei maßgeblich die Qualität der entstehenden Schranke.

[6] In der genannten Arbeit steht der allgemeinere Fall mit nicht identisch parallelen Maschinen im Vordergrund.

[7] Das zugehörige 1-Maschinen-Problem bezeichnet dabei ein 1-Maschinen-Problem $1||\sum w_jC_j$ mit derselben zugrundeliegenden Jobmenge.

Uma et al. (2006) [141] weisen darauf hin, dass die obige Aussage 4.2.1 mit der nachfolgenden von Schulz (1996) [126] entwickelten unteren Schranke übereinstimmt. Es handelt sich dabei um eine Weiterentwicklung der Ergebnisse von Queyranne (1993) [111] für das Parallelmaschinen-WCT-Problem (vgl. Satz 4.1.2).

Satz 4.2.2 (Schulz, 1996) *Für jeden zulässigen Fertigstellungszeitenvektor C des Problems $Pm||\sum w_jC_j$ ist die folgende Klasse von Ungleichungen gültig:*

$$\sum_{j\in S} p_jC_j \geq \frac{1}{2}\sum_{j\in S} p_j^2 + \frac{1}{2m}\left(\sum_{j\in S} p_j\right)^2 \qquad \text{für alle } S\subseteq J \tag{4.22}$$

Damit liefert die LP-Relaxation des folgenden linearen Programms eine untere Schranke für das Problem $Pm|r_j|\sum w_jC_j$:

$$(Pa) \qquad \sum_{j\in J} w_jC_j \to \text{Min!}$$

so dass

$$\sum_{j\in S} p_jC_j \geq \frac{1}{2}\sum_{j\in S} p_j^2 + \frac{1}{2m}\left(\sum_{j\in S} p_j\right)^2 \qquad \text{für alle } S\subseteq J \tag{4.23}$$

$$C_j \geq r_j + p_j \qquad \text{für alle } j\in S \tag{4.24}$$

Im Allgemeinen liefert die LP-Relaxation einer fertigstellungszeitbasierten Formulierung eine verhältnismäßig schwache untere Schranke. Zahlreiche Versuche, diese Modelle durch zusätzliche Klassen gültiger Ungleichungen zu verbessern, konnten dieses Problem bis dato nicht wirklich beheben (vgl. Pan & Shi, 2007, [101]). Beispielsweise Queyranne und Sviridenko (2002) [114] verbessern die obige Schranke im Hinblick auf Shop-Scheduling-Probleme durch Hinzufügen einer neuen job-bezogenen Klasse gültiger Ungleichungen. Diese ist aber eigentlich nur für Open-Shop-Probleme von Interesse, da sie sich für den Flow-Shop- und Job-Shop-Fall bereits implizit aus der vorgegebenen Arbeitsgangfolge der Jobs ergibt. Insbesondere wird auf diese Weise für das Parallelmaschinen-Problem keine Verbesserung erzielt.

Schulz (1996) [126] gelingt der Beweis, dass es sich bei der oben vorgestellten unteren Schranke um eine $1/(4-\frac{1}{m})$-Relaxation des Ausgangsproblems handelt (vgl. Satz 4.2.4). Der Begriff der ρ-Relaxation ist dabei wie folgt definiert:

Definition 4.2.3 *Unter einer* ρ-Relaxation *eines Problems, versteht man eine Relaxation, deren Zielfunktionswert schlechtestenfalls um einen Faktor ρ vom Wert einer optimalen Lösung abweicht, d. h. es gilt:*

$$ZF(approx.) \geq \rho \cdot ZF(opt.).$$

Das genannte Resultat bleibt darüber hinaus auch richtig, wenn man neben Jobfreigabezeiten auch Reihenfolgebedingungen und Verzögerungszeiten zwischen den Jobs zulässt:

Satz 4.2.4 (Schulz, 1996) *Es sei C^{LP} eine optimale Lösung der in 4.2.2 vorgestellten LP-Relaxation für das Problem $Pm|r_j|\sum w_jC_j$ sowie C^* eine optimale Lösung für dieses Scheduling-Problem. Dann gilt:*

$$\sum_{j\in J} w_jC_j^* \;\leq\; \left(4-\frac{1}{m}\right)\cdot\sum_{j\in J} w_jC_j^{LP}.$$

Auch die im vorangehenden Abschnitt vorgestellten zeit-indizierten Formulierungen kann man für das Parallelmaschinen-WCT-Problem mit Freigabezeiten verallgemeinern.

Korollar 4.2.5 *Ersetzt man im Problem (1b) die Restriktion (4.16) durch*

$$\sum_{j\in J} x_{jt} \;\leq\; m \qquad \text{für alle } t=0,\ldots,T \tag{4.25}$$

so ist die optimale Lösung des entstehenden Optimierungsproblems (Pb) eine untere Schranke für das Problem $Pm|r_j|\sum w_jC_j$ mit m identisch parallelen Maschinen.

Dies folgt aus der einfachen Tatsache, dass in einem Zeitintervall $[t,t+1]$ maximal m Jobs gleichzeitig bearbeitet werden können. Wir bezeichnen nun mit $z^{(Pa)}$ und $z^{(Pb)}$ die optimalen Zielfunktionswerte der LP-Relaxationen der entsprechenden Modelle. Anders als im Fall des 1-Maschinen-Problems gilt hier jedoch nicht unbedingt die Gleichheit, wie wir in 4.2.7 und 4.2.8 nachweisen werden. Zum Beweis des Satzes 4.2.7 benötigen wir außerdem den folgenden Spezialfall der Cauchy-Schwarz'schen Ungleichung (vgl. dazu auch Eastman et al., 1964, [40]).

Lemma 4.2.6 *Für jede Menge von n reellen Zahlen $x_1,\ldots,x_n$ gilt: $\frac{1}{n}\left(\sum_i x_i\right)^2 \leq \sum_i x_i^2$.*

Es gilt nun die folgende Beziehung:

Satz 4.2.7 $z^{(Pa)} \leq z^{(Pb)}$.

Beweis:

Wir müssen zeigen, dass jeder zulässige Fertigstellungszeitenvektor für (Pb) auch zulässig für (Pa) ist, die Gleichheit der Zielfunktionswerte ergibt sich dann automatisch.

Es sei C also ein zulässiger Fertigstellungszeitenvektor für (Pb). Die zugehörigen Werte der binären Variablen y_{jt} in der Lösung von (Pb) beschreiben einen fraktionalen Schedule für das Ausgangsproblem. D. h. innerhalb eines Intervalls $[t,t+1]$ werden darin y_{jt} Einheiten des Jobs j bearbeitet. Wir zeigen nun zunächst, dass C die Ungleichung (4.24) erfüllt. Im zu C gehörigen Schedule ist $y_{jt}=0$ für $r_j<t$ und $t=T$. Daraus ergibt sich:

$$\begin{aligned} C_j &= \frac{p_j}{2} + \frac{1}{p_j} \sum_{t=r_j}^{T-1} \left(t + \frac{1}{2}\right) y_{jt} \\ &\geq \frac{p_j}{2} + \frac{1}{p_j} \sum_{t=r_j}^{r_j+p_j-1} \left(t + \frac{1}{2}\right) \\ &= \frac{p_j}{2} + \frac{1}{p_j} \left(r_j p_j + (0+1+\ldots+(p_j-1)) + \frac{p_j}{2}\right) \\ &= r_j + \frac{p_j}{2} + \frac{1}{p_j} \left(\frac{(p_j-1)p_j}{2} + \frac{p_j}{2}\right) \\ &= r_j + p_j \end{aligned}$$

Dabei gilt die erste Abschätzung, da der Wert der Summe kleiner wird, je früher der zugehörige Job eingeplant wird.

Um nun zu zeigen, dass auch (4.23) erfüllt ist, gehen wir im Folgenden o.B.d.A. davon aus, dass möglichst viele Einheiten eines Jobs derselben Maschine zugeordnet werden. Dennoch kann die Situation auftreten, dass verschiedene Teile eines Jobs in dem zu C gehörenden fraktionalen Schedule von verschiedenen Maschinen bearbeitet werden müssen. Im Folgenden betrachten wir nun alle Jobs j, die im obigen Schedule nicht zusammenhängend bearbeitet werden können, als mehrere Jobs j_k mit der Gesamtbearbeitungszeit $\sum_k p_{j_k} = p_j$. Auf diese Weise entsteht aus der ursprünglichen Menge der Jobs J eine Menge von Hilfsjobs J' mit $|J'| > |J|$, aber $\sum_{j\in J} p_j = \sum_{j\in J'} p_j$.

In dem zu C gehörigen Schedule kann jeder Job aus der Menge J' eindeutig einer Maschine zugeordnet werden. Mithilfe von J' weisen wir nun die Zulässigkeit von C für die Bedingung (4.23) des Problems (Pa) nach. Dazu sei $S \subseteq J$ beliebig. Dann gilt:

$$\begin{aligned} \sum_{j\in S} p_j C_j &= \sum_{j\in S} p_j \left(\frac{p_j}{2} + \frac{1}{p_j} \sum_t \left(t + \frac{1}{2}\right) y_{jt}\right) \\ &= \sum_{j\in S} \frac{p_j^2}{2} + \sum_{j\in S} \sum_t \left(t + \frac{1}{2}\right) y_{jt} \end{aligned}$$

Mit S' bezeichnen wir analog zu J' die Menge der zu S gehörenden Hilfsjobs. Aufgrund der Definition von S' gilt: $\sum_{j\in S} p_j^2 \geq \sum_{j\in S'} p_j^2$. Wir setzen daher im Folgenden: $\frac{1}{2}\sum_{j\in S} p_j^2 = \frac{1}{2}\sum_{j\in S'} p_j^2 + R$ für $R \geq 0$ geeignet. Außerdem gilt $\sum_{j\in S}\sum_t \left(t+\frac{1}{2}\right) y_{jt} = \sum_{j\in S'}\sum_t \left(t+\frac{1}{2}\right) y_{jt}$, da sich die Anzahl von Jobs, die zu einem Zeitpunkt in Bearbeitung sind, durch das Aufspalten der Jobs nicht verändert.

Im Folgenden bezeichne nun $S'_i \subseteq S'$ (für $i = 1,\ldots,m$) die Menge der Jobs aus S', welche im betrachteten Schedule auf der Maschine i bearbeitet werden. Da sich S' nach Definition als disjunkte Vereinigung der S'_i darstellen lässt, ergibt sich insgesamt:

$$
\begin{aligned}
\sum_{j\in S} p_j C_j &= R + \sum_{j\in S'} \left(\frac{p_j^2}{2} + \sum_t \left(t + \frac{1}{2} \right) y_{jt} \right) \\
&= R + \sum_{i=1}^{m} \sum_{j\in S'_i} \left(\frac{p_j^2}{2} + \sum_t \left(t + \frac{1}{2} \right) y_{jt} \right) \\
&\overset{(4.1.6)}{\geq} R + \sum_{i=1}^{m} \frac{1}{2} \left(\sum_{j\in S'_i} p_j^2 + \left(\sum_{j\in S'_i} p_j \right)^2 \right) \\
&= R + \frac{1}{2} \sum_{j\in S'} p_j^2 + \frac{1}{2} \sum_{i=1}^{m} \left(\sum_{j\in S'_i} p_j \right)^2 \\
&= \frac{1}{2} \sum_{j\in S} p_j^2 + \frac{1}{2} \sum_{i=1}^{m} \left(\sum_{j\in S'_i} p_j \right)^2 \\
&\overset{(4.2.6)}{\geq} \frac{1}{2} \sum_{j\in S} p_j^2 + \frac{1}{2m} \left(\sum_{j\in S'} p_j \right)^2 \\
&= \frac{1}{2} \sum_{j\in S} p_j^2 + \frac{1}{2m} \left(\sum_{j\in S} p_j \right)^2 .
\end{aligned}
$$

■

Die Umkehrung dieser Aussage gilt im Allgemeinen nicht, wie das folgende Beispiel belegt. Wir konstruieren darin eine Probleminstanz, die eine zulässige Lösung für (Pa) besitzt, welche nicht zulässig für (Pb) ist und einen geringeren Zielfunktionswert aufweist als die optimale Lösung für (Pb).

Beispiel 4.2.8 *Wir betrachten zwei identisch parallele Maschinen und eine Menge von Jobs* $J = \{1,2,3\}$ *mit den folgenden Gewichten, Freigabe- und Bearbeitungszeiten:*

Jobs	1	2	3
r_j	1	1	0
p_j	1	2	2
w_j	1	1	1

Der in Abbildung 4.1 (links) dargestellte Schedule ist optimal für (Pa). Darin ergeben sich die Fertigstellungszeiten:

$$C_1 = 2, C_2 = 3 \text{ und } C_3 = 2.$$

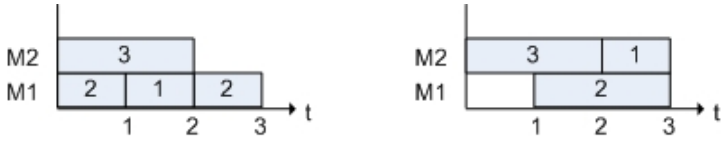

Abbildung 4.1.: Optimale Schedules für (Pa) und (Pb) in Beispiel 4.2.8

Mit anderen Worten, Ungleichung (4.24) *ist erfüllt. Auch* (4.23) *ist erfüllt, denn beispielsweise für* $S = \{1,2,3\}$ *ergibt sich*

$$\sum_{j\in S} p_j C_j \;=\; 12 \;>\; \frac{43}{4} \;=\; \frac{1}{2}\sum_{j\in S} p_j^2 + \frac{1}{4}\left(\sum_{j\in S} p_j\right)^2$$

Analog überprüft man die Gleichung für die anderen Teilmengen von J. Diese Lösung hat den Zielfunktionswert 7. *Der Schedule ist jedoch wegen* $y_{20} = 1$ *für* $0 < r_2 = 1$ *nicht zulässig für* (Pb). *Eine optimale Lösung für dieses Problem wird durch den in Abbildung 4.1 (rechts) dargestellten Schedule repräsentiert. Sie besitzt den Zielfunktionswert* 8.

Analog zu 4.2.5 kann man auch das Modell $(1c)$ für identisch parallele Maschinen verallgemeinern. Die entsprechende Aussage beinhaltet der folgende Satz.

Satz 4.2.9 *Ersetzt man im Problem (1c) die Ungleichung (4.20) durch*

$$\sum_{j\in J}\sum_{s=t-p_j+1}^{t} x_{js} \;\leq\; m \qquad \text{für alle } t = 0,\ldots,T \tag{4.26}$$

so ist die optimale Lösung des entstehenden Optimierungsproblems (Pc) eine optimale Lösung für das Problem $Pm|r_j|\sum w_j C_j$ *mit m identisch parallelen Maschinen.*

Beweis:

Wir zeigen lediglich die Korrektheit der Bedingung (4.26). Der Rest des Beweises ergibt sich direkt aus Satz 4.1.7. Angenommen, in einer Lösung wird ein Job k zum Zeitpunkt t_0 gestartet, d. h. es gilt $x_{kt_0} = 1$. Um zu gewährleisten, dass die Kapazität von m Maschinen nicht überschritten wird, dürfen höchstens $m-1$ weitere Jobs $j_1,\ldots,j_{m-1}$ (wobei $j_i \neq k$ für $i = 1,\ldots,m-1$) innerhalb des Intervalls $[t_0 - p_{j_i} + 1, t_0]$ gestartet werden, denn diese Jobs sind dann sämtlich zum Zeitpunkt t_0 in Bearbeitung, es gilt also

$$\sum_{j\in J, j\neq k}\sum_{t=t_0-p_j+1}^{t_0} x_{jt} \leq m-1 \quad \text{bzw.} \quad \sum_{j\in J}\sum_{t=t_0-p_j+1}^{t_0} x_{jt} \leq m$$

Dies ist für alle Zeitpunkte $t_0 = 0,\ldots,T$ richtig, mithin ist Ungleichung (4.26) gültig. Dass umgekehrt keine zulässige Lösung existiert, welche diese Bedingung nicht erfüllt, ergibt sich mit Hilfe derselben Argumentation. ∎

Man kann nun entsprechend der Beweisführung von Dyer und Wolsey (1990) [39] und mit Hilfe der obigen Verallgemeinerung eine analoge Aussage zu Satz 4.1.8 für das Parallelmaschinen-Problem $Pm|r_j|\sum w_j C_j$ beweisen. Dazu bezeichne $z^{(Pc)}$ den optimalen Zielfunktionswert der LP-Relaxation des Modells (Pc). Dann gilt:

Satz 4.2.10 $z^{(Pb)} \leq z^{(Pc)}$.

Beweis:

Der Beweis ergibt wie bei Dyer und Wolsey [39] durch Verwendung der Substitution $y_{jt} := \sum_{t-p_j<s\leq t} x_{js}$. ∎

Die in 4.2.4 zitierte Worst Case Abschätzung gilt damit natürlich insbesondere auch für die LP-Relaxationen der beiden Formulierungen (Pb) und (Pc).

4.3. Hybrid Flow-Shop-Probleme

Im Folgenden verallgemeinern wir nun zunächst die vorangehend vorgestellten Ergebnisse für das Hybrid Flow-Shop-Problem mit Freigabezeiten $FHm,(PM^{(k)})_{k=1}^{m}|r_j|\sum w_jC_j$. Auf die zusätzliche Berücksichtigung von Blocking-Constraints sowie Transportrestriktionen gehen wir in den anschließenden Abschnitten ein. Das fertigstellungszeitbasierte Modell (Pa) wurde bereits von Schulz (1996) [126] wie folgt auf das HFS-Problem übertragen:

Satz 4.3.1 (Schulz, 1996) *Die LP-Relaxation des folgenden linearen Programms ist eine untere Schranke für das Problem $FHm,(PM^{(k)})_{k=1}^{m}|r_j|\sum w_jC_j$.*

$$(HFSa) \qquad \sum_{j\in J} w_jC_{mj} \rightarrow \text{Min!}$$

so dass

$$\sum_{j\in S} p_{ij}C_{ij} \geq \frac{1}{2}\sum_{j\in S} p_{ij}^2 + \frac{1}{2M^{(i)}}\left(\sum_{j\in S} p_{ij}\right)^2 \qquad \text{für alle } S\subseteq J, i\in I \tag{4.27}$$

$$C_{(i+1)j} \geq C_{ij} + p_{(i+1)j} \qquad \text{für alle } i\in I\setminus\{m\},\ j\in J \tag{4.28}$$

$$C_{1j} \geq r_j + p_{1j} \qquad \text{für alle } j\in J \tag{4.29}$$

Darüber hinaus weist Schulz in seiner Arbeit das folgende Worst Case Resultat für die obige Schranke nach.

Satz 4.3.2 (Schulz, 1996) *Es sei C^{LP} ein optimaler Fertigstellungszeitenvektor in einer Lösung der LP-Relaxation der Formulierung aus 4.3.1 sowie C^* ein optimaler Fertigstellungszeitenvektor für das Ausgangsproblem $FHm,(PM^{(k)})_{k=1}^{m}|r_j|\sum w_jC_j$. Dann gilt:*

$$\sum_{j=1}^{n} w_jC_j^* \leq (3m+1)\sum_{j=1}^{n} w_jC_j^{LP}$$

Analog zu 4.3.1 können auch die beiden zeit-indizierten Formulierungen (Pb) und (Pc) für das Hybrid Flow-Shop-Problem verallgemeinert werden.

Satz 4.3.3 *Die optimale Lösung des folgenden linearen Programms ist eine untere Schranke für das Problem* $FHm,(PM^{(k)})_{k=1}^{m}|r_j|\sum w_jC_j$. *Analog zu 4.1.5 ist darin* y_{ijt} *eine binäre Variable, welche den Wert 1 annimmt, falls Job* j *im Intervall* $[t,t+1]$ *auf einer beliebigen Maschine der Stufe* i *bearbeitet wird, und ansonsten* 0. *Es ist dabei* $y_{ijt}=0$ *für* $t<r_j+\sum_{l=1}^{i-1}p_{lj}$ *und* $t\geq T-\sum_{l=i+1}^{m}p_{lj}$. *Des Weiteren bezeichne* C_{ij} *die Fertigstellungszeit eines Jobs* j *auf einer Bearbeitungsstufe* i.

$$(HFSb)\qquad \sum_{j\in J} w_jC_{mj}\rightarrow \text{Min!}$$

so dass

$$\sum_{j\in J} y_{ijt} \leq M^{(i)} \qquad \text{für alle } i\in I, t=0,\ldots,T \tag{4.30}$$

$$\sum_{i\in I} y_{ijt} \leq 1 \qquad \text{für alle } j\in J, t=0,\ldots,T \tag{4.31}$$

$$\sum_{t=0}^{T} y_{ijt} = p_{ij} \qquad \text{für alle } i\in I,\ j\in J \tag{4.32}$$

$$C_{ij} \overset{(=)}{\leq} C_{(i+1)j}-p_{(i+1)j} \qquad \text{für alle } i\in I\setminus\{m\},\ j\in J \tag{4.33}$$

$$r_j \leq C_{1j}-p_{1j} \qquad \text{für alle } j\in J \tag{4.34}$$

$$C_{ij} = \frac{p_{ij}}{2}+\frac{1}{p_{ij}}\sum_{t=0}^{T}\left(t+\frac{1}{2}\right)y_{ijt} \qquad \text{für alle } i\in I,\ j\in J \tag{4.35}$$

Beweis:

Die Ungleichungen (4.30) modellieren die Maschinenkapazität, während (4.31) gewährleistet, dass sich jeder Job zu einem Zeitpunkt nur auf einer einzigen Bearbeitungsstufe befindet. Die Restriktionen (4.33) und (4.34) beschreiben analog zu 4.3.1 die vorgegebene Arbeitsgangfolge für die verschiedenen Operationen eines Jobs. Daher weisen wir analog zum Beweis von 4.1.5 bei Dyer und Wolsey (1990) [39] lediglich die Gültigkeit der Gleichungen (4.35) nach, alles Weitere folgt direkt aus den vorangehenden Ausführungen.

Angenommen, ein Job j wird im Intervall $[t_0,t_0+p_{ij}]$ auf einer Bearbeitungsstufe i bearbeitet. D. h. es gilt $y_{ijt}=1$ für $t=t_0,\ldots,t_0+p_{ij}-1$ und $C_{ij}=t_0+p_{ij}$. Einsetzen dieser Werte in (4.35) liefert:

$$\begin{aligned}
C_{ij} &= \frac{p_{ij}}{2}+\frac{1}{p_{ij}}\sum_{t=0}^{T}\left(t+\frac{1}{2}\right)y_{ijt}\\
&= \frac{p_{ij}}{2}+\frac{1}{p_{ij}}\sum_{t=t_0}^{t_0+p_{ij}-1}\left(t+\frac{1}{2}\right)\\
&= \frac{p_{ij}}{2}+\frac{1}{p_{ij}}\left(\frac{1}{2}p_{ij}+t_0p_{ij}+\frac{(p_{ij}-1)p_{ij}}{2}\right)\\
&= t_0+p_{ij}
\end{aligned}$$

■

Durch Ersetzen von „$\leq$“ durch „$=$“ in (4.33) kann in diesem Modell zudem ein Verbot von Wartezeiten der Jobs zwischen den einzelnen Bearbeitungsschritten (No-Wait-Constraints) realisiert werden.

Die obige Formulierung ist aufgrund der Bedingung (4.35) nicht geeignet, um eine Situation zu modellieren, in welcher ein Job j eine Bearbeitungsstufe i überspringen kann, d. h. in der $p_{ij} = 0$ möglich ist. Um auch für diesen Fall untere Schranken zu erhalten, kann man folgende Verallgemeinerung der zweiten zeit-indizierten Formulierung (Pc) verwenden. Dazu müssen analog zu 4.3.3 Restriktionen hinzugefügt werden, welche die Einhaltung der vorgegebenen Arbeitsgangfolgen gewährleisten und die Aufspaltung von Jobs über mehrere Bearbeitungsstufen vermeiden. Insgesamt ergibt sich so der folgende Satz:

Satz 4.3.4 *Die optimale Lösung des folgenden linearen Programms ist eine optimale Lösung für das Problem $FHm,(PM^{(k)})_{k=1}^{m}|r_j|\sum w_jC_j$. Analog zu 4.1.7 ist darin x_{ijt} eine binäre Variable, welche den Wert 1 annimmt, falls Job j zum Zeitpunkt t auf Stufe i gestartet wird, und 0 sonst. Außerdem ist $x_{ijt} = 0$ für $t < r_j + \sum_{l=1}^{i-1} p_{lj}$ und $t > T - \sum_{l=i}^{m} p_{lj}$.*

$(HFSc)$ $\quad \sum_{j\in J} w_jC_{mj} \to \text{Min!}$

so dass

$$\sum_{t=0}^{T} x_{ijt} = 1 \qquad \text{für alle } i \in I,\ j \in J \tag{4.36}$$

$$\sum_{j\in J}\sum_{s=t-p_{ij}+1}^{t} x_{ijs} \leq M^{(i)} \qquad \text{für alle } i \in I, t = 0,\ldots,T \tag{4.37}$$

$$\sum_{i\in I}\sum_{s=t-p_{ij}+1}^{t} x_{ijs} \leq 1 \qquad \text{für alle } j \in J, t = 0,\ldots,T \tag{4.38}$$

$$C_{ij} = \sum_{t=0}^{T} t x_{ijt} + p_{ij} \qquad \text{für alle } i \in I,\ j \in J \tag{4.39}$$

$$C_{ij} \overset{(=)}{\leq} C_{(i+1)j} - p_{(i+1)j} \qquad \text{für alle } i \in I \setminus \{m\},\ j \in J \tag{4.40}$$

$$r_j \leq C_{1j} - p_{1j} \qquad \text{für alle } j \in J \tag{4.41}$$

Beweis:

Wir weisen die Korrektheit der Restriktion (4.38) zur Vermeidung der Aufspaltung von Jobs über mehrere Stufen nach, der Rest des Beweises folgt direkt aus 4.2.9. Angenommen, ein Job j wird zum Zeitpunkt t_0 auf einer Stufe i gestartet, d. h. es gilt $x_{ijt_0} = 1$. Dann darf j im Intervall $[t_0 - p_{lj} + 1, t_0]$ auf keiner anderen Stufe l gestartet werden, d. h. für alle Stufen $l \neq i$ gilt:

$$\sum_{t=t_0-p_{lj}+1}^{t_0} x_{ljt} = 0 \quad \text{, falls } x_{ijt_0} = 1$$

Daraus folgt insbesondere:

$$\sum_{l \in I, l \neq i} \sum_{t=t_0-p_{lj}+1}^{t_0} x_{ljt} \quad = \quad 0 \quad \text{, falls } x_{ijt_0} = 1$$

Da dies für alle Zeitpunkte $t_0 = 0, \ldots, T$ erfüllt sein muss, ergibt sich so insgesamt die Gültigkeit der Ungleichung (4.38). Dass keine zulässige Lösung existiert, welche diese Bedingung nicht erfüllt, ergibt sich mit Hilfe derselben Argumentation und damit folgt die Behauptung. ∎

Auch in dieser Formulierung können durch Ersetzen von „$\leq$“ durch „$=$“ in (4.40) No-Wait-Constraints anstelle von unbegrenztem Zwischenspeicher realisiert werden. Des Weiteren kann man analog zu Satz 4.2.7 und 4.2.10 die optimalen Zielfunktionswerte der LP-Relaxationen der drei Modelle $(HFSa)$, $(HFSb)$ und $(HFSc)$ vergleichen.

Satz 4.3.5 *Es bezeichne $z^{(HFSi)}$ den optimalen Zielfunktionswert der LP-Relaxation des Modells $(HFSi)$ für $i = a, b, c$. Dann gilt:*

$$z^{(HFSa)} \leq z^{(HFSb)} \leq z^{(HFSc)}.$$

Beweis:

Die erste Beziehung ergibt sich direkt aus Satz 4.2.7. Der zweite Teil ergibt sich fast vollständig aus 4.2.10, es muss nur noch nachgewiesen werden, dass eine zulässige Lösung der LP-Relaxation von $(HFSc)$ die Bedingung (4.31) aus $(HFSb)$ erfüllt. Dies kann aber sofort unter erneuter Verwendung der Substitution $y_{ijt} = \sum_{t-p_{ij}<s\leq t} x_{ijs}$ gezeigt werden. ∎

Damit gilt natürlich die in 4.3.2 zitierte Worst Case Schranke auch für die LP-Relaxationen der Modelle $(HFSb)$ und $(HFSc)$. Aufgrund des Resultats 4.3.5 werden wir im weiteren Verlauf dieser Arbeit, insbesondere zur Auswertung des im nächsten Kapitel vorgestellten heuristischen Lösungsverfahrens, nur noch die beiden Modelle $(HFSb)$ und $(HFSc)$ berücksichtigen. Im Folgenden widmen wir uns nun der Erweiterung der beiden Modelle zur Berücksichtigung zusätzlicher Restriktionen, welche für die im Rahmen dieser Arbeit untersuchte Anwendungsaufgabe eine besondere Rolle spielen: Blocking-Constraints und Transportrestriktionen.

4.3.1. Blocking-Constraints

Die im vorangehenden Abschnitt vorgestellten Formulierungen $(HFSb)$ und $(HFSc)$ gestatten sowohl die Modellierung von HFS-Problemen mit unbegrenztem Zwischenspeicher als auch mit No-Wait-Constraints. Für die Praxis sind jedoch auch Situationen mit Blocking-Constraints bzw. endlichem Zwischenspeicher von großer Bedeutung. Im Allgemeinen ist es ausreichend, sich auf den Fall mit Blocking-Constraints (also keine Zwischenlagerung außerhalb von Maschinen) zu beschränken, da auf diese Weise auch jedes

Problem mit endlichem Zwischenspeicher modelliert werden kann (vgl. Abschnitt 3.4.2 oder Tang & Xuan, 2006, [137] bzw. Wang & Tang, 2009, [149]).

Die meisten vorhandenen Modelle für Flow-Shop- oder Job-Shop-Probleme, welche Blocking-Constraints berücksichtigen, beinhalten als Zielfunktion die Minimierung der Zykluszeit. Beispielsweise Sawik (2002) [124] entwirft für dieses Problem ein Modell mit Zuordnungs- und Ordnungsvariablen, wobei die Berechnung exakter Lösungen im Vordergrund steht. Die entsprechende LP-Relaxation ist jedoch, wie eingangs bereits diskutiert, zur Berechnung starker Schranken nicht geeignet.

Eine andere, liefertermin-orientierte Zielfunktion berücksichtigen Gomes et al. (2005) [44]. Die Autoren entwickeln ein Modell für das Hybrid Job-Shop-Problem, welches die beiden bisher verwendeten Arten zeit-indizierter Variablen kombiniert. Es ist allerdings auf eine geeignete Modellierung endlicher Zwischenspeicher ausgelegt und kann nicht direkt auf Blocking-Constraints übertragen werden. Dazu müsste eine Speicherkapazität von Null vorausgesetzt werden, unter dieser Annahme reduziert sich die verwendete Formulierung jedoch auf die Situation mit No-Wait-Constraints. Man kann die Modelle $(HFSb)$ und $(HFSc)$ aber unter Verwendung zusätzlicher binärer bzw. diskreter Variablen der folgenden Art zur Abbildung von Blocking-Constraints erweitern:

- b_{ijt} nimmt analog zu y_{ijt} den Wert 1 an, falls eine Maschine der Stufe i im Intervall $[t,t+1]$ durch den Job j blockiert wird, und ansonsten 0.
- $B_{ij} = \sum_{t=0}^{T} b_{ijt}$ repräsentiert dann die Gesamtdauer, in welcher eine Maschine des Typs i durch den Job j blockiert wird.

Das Modell $(HFSb)$ kann dann folgendermaßen modifiziert werden:

$$(HFSb-block) \qquad \sum_{j\in J} w_j C_{mj} \rightarrow \text{Min!}$$

so dass

$$\sum_{j\in J}(y_{ijt}+b_{ijt}) \leq M^{(i)} \qquad \textit{für alle } i\in I, t=0,\ldots,T \tag{4.42}$$

$$\sum_{i\in I}(y_{ijt}+b_{ijt}) \leq 1 \qquad \textit{für alle } j\in J, t=0,\ldots,T \tag{4.43}$$

$$\sum_{t=0}^{T} y_{ijt} = p_{ij} \qquad \textit{für alle } i\in I,\ j\in J \tag{4.44}$$

$$C_{(i+1)j} - C_{ij} = p_{(i+1)j} + B_{ij} \qquad \textit{für alle } i\in I\setminus\{m\},\ j\in J \tag{4.45}$$

$$C_{1j} - p_{1j} \geq r_j \qquad \textit{für alle } j\in J \tag{4.46}$$

$$C_{ij} = \frac{p_{ij}}{2} + \frac{1}{p_{ij}} \sum_{t=0}^{T} \left(t+\frac{1}{2}\right) y_{ijt} \qquad \textit{für alle } j\in J \tag{4.47}$$

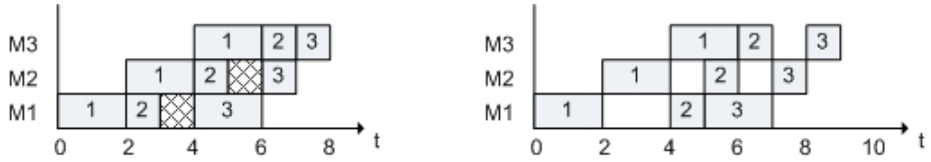

Abbildung 4.2.: Lösung mit Blocking- und No-Wait-Constraints in Beispiel 4.3.6

Für ein Flow-Shop-Problem ist die Menge zulässiger Schedules unter Berücksichtigung von Blocking-Constraints offenkundig wesentlich größer als für den Spezialfall mit No-Wait-Constraints. Dies gilt formal natürlich auch für die LP-Relaxationen der Modelle $(HFSb)$ und $(HFSb-block)$. Der tatsächliche Effekt des Übergangs zu No-Wait-Constraints ist allerdings verhältnismäßig gering, da die Differenz der Zielfunktionswerte beider Formulierungen hier stark von dem für T gewählten Wert abhängt (vgl. Beispiel 4.3.6). Je größer T, desto größer ist auch der „Spielraum“, eine (optimale) fraktionale Lösung für $(HFSb-block)$ zu konstruieren, welche auch No-Wait-Constraints genügt, wie das nächste Beispiel verdeutlicht.

Beispiel 4.3.6 *Wir betrachten das Standard Flow-Shop-Problem $F3|r_j|\sum w_jC_j$ mit drei Jobs und den folgenden Daten:*

Job j	r_j	w_j	p_{1j}	p_{2j}	p_{3j}
1	0	7	2	2	2
2	2	3	1	1	1
3	4	1	2	1	1

In Abbildung 4.2 sind Lösungen für diese Probleminstanz unter Berücksichtigung von Blocking-Constraints (links) und No-Wait-Constraints (rechts) dargestellt. Diese sind sowohl für das ursprüngliche Problem als auch für das obige Modell $(HFSb-block)$ optimal. Im ersten Fall ist $C_1=6$, $C_2=7$, $C_3=8$, bei einem ZF-Wert von 71, wogegen unter No-Wait-Constraints $C_1=6$, $C_2=7$ und $C_3=9$ gilt, bei einem ZF-Wert von 72.

Man kann daher bei der Lösung der LP-Relaxation $T=8$ wählen. Die dargestellte Lösung für das Problem mit Blocking-Constraints ist auch in der LP-Relaxation optimal. Unter No-Wait-Constraints ist in der LP-Relaxation eine fraktionale Lösung mit den Fertigstellungszeiten $C_1=6.125$, $C_2=6.75$ und $C_3=8$ sowie einem ZF-Wert von 71.125 optimal. Erhöht man jedoch T lediglich um 1 also $T=9$, so gibt es auch für No-Wait-Constraints eine fraktionale Lösung mit dem ZF-Wert 71.

Dieses Beispiel unterstreicht, wie stark die Qualität der LP-Relaxation bei No-Wait-Constraints von der gewählten oberen Schranke T für die Planungperiode abhängt. Da im Allgemeinen die Zykluszeit einer optimalen Lösung natürlich nicht im Voraus bekannt ist, sind in der Praxis bei der Unterscheidung von No-Wait- und Blocking-Constraints

höchstens marginale Unterschiede in den ZF-Werten der LP-Relaxationen zu erwarten. Dies gilt genauso für den Vergleich der Situationen mit unbegrenztem Zwischenspeicher und Blocking-Constraints.

Auf ähnliche Weise kann man prinzipiell auch das Modell $(HFSc)$ zur Berücksichtigung von Blocking-Constraints modifizieren. Wir verwenden dazu erneut die obige Definition der Variablen b_{ijt} und B_{ij} und erhalten insgesamt die folgende Formulierung:

$$(HFSc-block) \qquad \sum_{j\in J} w_j C_{mj} \to \text{Min!}$$

so dass

$$\sum_{t=0}^{T} x_{ijt} = 1 \qquad \text{für alle } i\in I,\ j\in J \tag{4.48}$$

$$\sum_{j\in J}\left(b_{ijt} + \sum_{s=t-p_{ij}+1}^{t} x_{ijs}\right) \le M^{(i)} \qquad \text{für alle } i\in I, t=0,\dots,T \tag{4.49}$$

$$\sum_{i\in I}\left(b_{ijt} + \sum_{s=t-p_{ij}+1}^{t} x_{ijs}\right) \le 1 \qquad \text{für alle } j\in J, t=0,\dots,T \tag{4.50}$$

$$C_{ij} - p_{ij} = \sum_{t=0}^{T} t x_{ijt} \qquad \text{für alle } i\in I,\ j\in J \tag{4.51}$$

$$C_{(i+1)j} - C_{ij} - B_{ij} = p_{(i+1)j} \qquad \text{für alle } i\in I\setminus\{m\},\ j\in J \tag{4.52}$$

$$C_{1j} - p_{1j} \ge r_j \qquad \text{für alle } j\in J \tag{4.53}$$

Auf diese Weise ist auch hier die Behandlung von Situationen mit $p_{ij}=0$ möglich. In diesem Fall muss jedoch auch $b_{ijt}=0$ für alle $t=0,\dots,T$ gewährleistet sein. Dies kann durch Hinzufügen der Restriktion

$$\sum_{t=0}^{T} b_{ijt} \le M p_{ij} \qquad \text{für alle } i\in I,\ j\in J \tag{4.54}$$

für genügend großes M realisiert werden.

Einen ähnlichen Ansatz zur Modellierung von Blocking-Constraints präsentieren Tang und Xuan (2006) [137] . Darin wird auf eine zusätzliche Art von Variablen zur Abbildung der Blockungszeiten verzichtet. Stattdessen verwenden die Autoren die zeit-indizierten Variablen aus Formulierung $(HFSb)$ mit modifizierter Definition. y_{ijt} nimmt nun den Wert 1 an, falls Job j im Intervall $[t,t+1]$ auf der Stufe i bearbeitet wird oder diese blockiert, und 0 sonst. Darüber hinaus werden neben den Fertigstellungszeiten C_{ij} der Jobs auch deren Startzeiten S_{ij} auf den Stufen als Entscheidungsvariablen definiert. Die Restriktionen entsprechen dann im Wesentlichen denen aus $(HFSb)$, es wird lediglich eine weitere Bedingung zur Abbildung der Blocking-Constraints benötigt:

$$\sum_{t=0}^{T} y_{ijt} = S_{(i+1)j} - S_{ij} \qquad \text{für alle } i\in I,\ j\in J \tag{4.55}$$

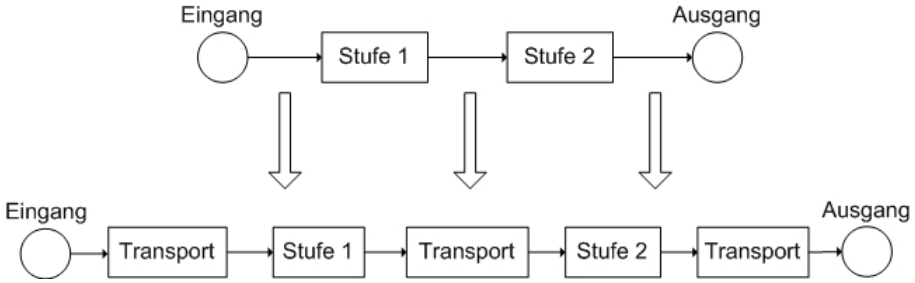

Abbildung 4.3.: Hinzufügen von Transportstufen

Das so entstehende Problem wird in der genannten Arbeit unter Verwendung einer Lagrange-Relaxation behandelt.

4.3.2. Transportrestriktionen

(Hybrid) Flow-Shop-Probleme mit Transporten wurden in der Literatur bereits in verschiedenen Ausprägungen diskutiert. Dabei stehen jedoch oftmals die Behandlung praxisbezogener Details des Problems sowie die Maximierung des Durchsatzes im Vordergrund (vgl. Abschnitt 3.4.3). Shop-Scheduling-WCT-Probleme mit Transporten haben in der Literatur dagegen bisher kaum Beachtung gefunden. Beispielsweise Xuan und Tang (2007) [155] präsentieren ein zeit-indiziertes Modell, welches Transportzeiten zwischen den Bearbeitungsstufen berücksichtigt. Insgesamt wird das Problem darin allerdings stark vereinfacht, da die Kapazität der Transportmittel nicht beschränkt ist.

Unter der Voraussetzung, dass die notwendige Transportzeit zwischen zwei Bearbeitungsstufen für jeden Job bereits im Voraus bekannt ist, können die zeit-indizierten Ansätze $(HFSb)$ und $(HFSc)$ relativ einfach zur Berücksichtigung von Transporten mit beschränkter Kapazität der Transportmittel modifiziert werden. Zur Modellierung fügen wir jeweils zwischen zwei aufeinander folgenden Bearbeitungsstufen sowie vor die erste und im Anschluss an die letzte Stufe eine zusätzliche *Transportstufe* ein (vgl. Abbildung 4.3). Die Bearbeitungszeit der Jobs auf jeder dieser Stufen entspricht dann der jeweiligen Transportzeit. Außerdem bestimmt die Anzahl zur Verfügung stehender Transportmittel bzw. Fahrzeuge die Kapazität jeder dieser Transportstufen. Insgesamt ergibt sich so ein Modell mit $2m+1$ Bearbeitungsstufen.

Falls sämtliche Transportaufgaben aus einer gemeinsamen Flotte von Transportmitteln bedient werden, muss natürlich außerdem sichergestellt sein, dass sich zu einem Zeitpunkt insgesamt nur so viele Jobs gleichzeitig auf den verschiedenen Transportstufen befinden, wie Fahrzeuge zur Verfügung stehen. Um dies zu modellieren, verwenden wir einen zusätzlichen binären Parameter z_i für jede Maschinenstufe i. Dieser nimmt den Wert 1 an, falls es sich bei i um eine Transportstufe handelt, und 0 sonst. Die Kapazitätsbeschränkung der Transportmittel wird im Modell $(HFSb)$ dann durch folgende zusätzliche Ne-

benbedingung realisiert:

$$\sum_{j\in J}\sum_{i\in I,z_i=1} y_{ijt} \leq n_R \qquad \textit{für alle } t=0,\ldots,T \tag{4.56}$$

Dabei bezeichnet n_R die Anzahl zur Verfügung stehender Fahrzeuge. Eine analoge Restriktion kann man auch in der Formulierung $(HFSc)$ ergänzen:

$$\sum_{j\in J}\sum_{i\in I,z_i=1}\sum_{s=t-p_{ij}+1}^{t} x_{ijs} \leq n_R \qquad \textit{für alle } t=0,\ldots,T \tag{4.57}$$

Alternativ zu diesem Ansatz könnte man auch nur eine einzelne zusätzliche Transportstufe betrachten, die dann mehrfach besucht[8] wird (vgl. z. B. Soukhal & Martienau, 2005, [132]). Der Preis für die geringere Stufenzahl ist dann jedoch ein komplexeres Modell, in dem zusätzlich eine Stufe mit Rückführung formuliert werden muss.

4.4. Ergänzende Bemerkungen

Die Entwicklung (starker) unterer Schranken spielt nicht nur bei der Evaluation heuristischer Ansätze zur Lösung *NP*-schwerer (Scheduling-)Probleme eine große Rolle. Vielmehr sind untere Schranken auch ein wichtiges Werkzeug zur Konstruktion von Approximationsalgorithmen (vgl. z. B. Webster, 1992, [151]). Oftmals erlauben List Scheduling Verfahren, deren Sequenzierungsregeln auf der Auswertung geeigneter unterer Schranken basieren, im Gegensatz zu anderen derartigen Ansätzen Aussagen über die Qualität der erzielten Näherungslösung (vgl. dazu auch Abschnitt 3.4.1.3). Beispielsweise in den Arbeiten von Schulz (1996) [126] oder Hall et al. (1997) [49] sowie Queyranne und Schulz (2006) [113] werden zu diesem Zweck die bereits vorgestellten fertigstellungszeitbezogenen Ansätze verwendet. Des Weiteren entwickeln Schulz und Skutella (1997) [127] Approximationsalgorithmen, welche auf der stärkeren zeit-indizierten Formulierung basieren und für die eine Performance Schranke berechnet werden kann.

Die eingangs vorgestellten Möglichkeiten zur Modellierung von (Scheduling-)Problemen werden des Weiteren natürlich nicht ausschließlich getrennt voneinander berücksichtigt; genauso sind in der Literatur auch Mischformen zu finden. Z. B. Sawik (2000) [123] präsentiert ein Modell mit Zuordnungs- und linearen Ordnungsvariablen. Liu und Karimi (2008) [87] verwenden ferner zwei Modelle mit Positions- und linearen Ordnungsvariablen. Diese werden in der genannten Arbeit für ein HFS-Problem mit identisch parallelen Maschinen untersucht, allerdings ohne dabei die konkrete Maschinenauswahl abzubilden. Beide Modelle benötigen jedoch wie bereits beschrieben eine Darstellung des Variablenzusammenhangs in Form logischer Restriktionen, was die LP-Relaxation der Formulierung stark abschwächt. Die Autoren bestimmen mit Hilfe ihrer Modelle optimale Lösungen für dreistufige Probleme, wobei eine einigermaßen gleichmäßige Auslastung der

[8] Engl. recirculation, vgl. z. B. Sawik (2002) [124]

Stufen Voraussetzung ist.[9] Darüber hinaus beinhalten die untersuchten Datensätze maximal elf Jobs. Eine optimale Lösung kann mithin nur für sehr kleine Probleminstanzen bestimmt werden.

Des Weiteren ist der Vollständigkeit halber folgende alternative Formulierungsmethode zu nennen, die im Rahmen dieser Arbeit jedoch keine Anwendung findet: Pan und Shi (2007) [101] modellieren ein Problem der Gestalt $1||\sum f_j(C_j)$ ohne zusätzliche Restriktionen unter Verwendung eines zeit-indizierten linearen Programms als Transportproblem mit der Zielfunktion $\sum c_{jt}y_{jt}$. Dabei ist y_{jt} eine zeit-indizierte binäre Variable, die den Wert 1 annimmt, falls Job j im Intervall $[t,t+1]$ bearbeitet wird, und 0 sonst. Unter gewissen Voraussetzungen für die Koeffizienten c_{jt} erhält man so eine untere Schranke für das Ausgangsproblem. Durch Übergang zum dualen Problem optimieren die Autoren die Wahl dieser Multiplikatoren. Bei Verwendung bestmöglicher Koeffizienten können sie zudem die Äquivalenz des Modells zur stärkeren zeit-indizierten Formulierung nachweisen. Die dabei präsentierte Vorgehensweise kann man jedoch nicht ohne Weiteres auf (Hybrid) Flow-Shop-Probleme übertragen.

Nicht nur die LP-Relaxation eines geeigneten Modells ist bei der Berechnung (unterer) Schranken von Bedeutung. Im Folgenden geben wir daher eine kompakte Übersicht möglicher Alternativen, die in der Literatur bei der Schrankenbildung für Scheduling-Probleme verwendet werden. Eine Variante sind sogenannte kombinatorische Ansätze. Darunter versteht man Schranken, deren Berechnung auf einer inhaltlichen Relaxation (im Gegensatz zur LP-Relaxation einer mathematischen Formulierung) des Problems basiert, beispielsweise auf der Zulassung von Job-Unterbrechungen. Dennoch stehen LP-basierte und kombinatorische Schranken oftmals in enger Beziehung. So gelang es Uma et al. (2006) [141], Zusammenhänge zwischen kombinatorischen und LP-basierten Schranken für verschiedene WCT-Scheduling-Probleme nachzuweisen. Im Einzelnen untersuchen die Autoren die Probleme $1|r_j|\sum w_jC_j$, $Pm||\sum w_jC_j$ und $Pm|r_j|\sum w_jC_j$. Sie zeigen, dass es für jede dieser Aufgaben eine kombinatorische Schranke gibt, die für das 1-Maschinen-Problem mit der LP-Relaxation der schwächeren zeit-indizierten Formulierung übereinstimmt, bzw. im Fall der Parallelmaschinen-Probleme einer Formulierung mit Fertigstellungszeitvariablen entspricht. Die Autoren betrachten insgesamt zwei kombinatorische Relaxationen, die auf Job-Splitting basieren. Die Teiljobs werden in beiden Fällen unter Verwendung der preemptive WSPT-Regel berechnet. Die beiden Ansätze unterscheiden sich lediglich in der Aufteilung der Jobgewichte auf die einzelnen Fragmente.

In den Arbeiten von Kaminsky und Simchi-Levi (1998) [66] sowie Bai und Tang (2007) [13] werden kombinatorische Schranken für Flow-Shop-Probleme untersucht. In der erstgenannten kann bereits eine gute asymptotische Performance der entwickelten Schranke nachgewiesen werden; in der zweiten Arbeit werden diese Ergebnisse weiter verbessert. Eine Übertragung der Ansätze auf das Hybrid Flow-Shop-Problem ist zwar grundsätz-

[9] D. h. es ist kein eindeutiger Engpass identifizierbar.

lich möglich, macht die Schranke jedoch sehr schwach. Beispielsweise könnte ein Hybrid Flow-Shop-Problem durch Division der Bearbeitungszeiten der Jobs durch die jeweils zur Verfügung stehende Anzahl von Maschinen auf den einzelnen Stufen auf ein Standard Flow-Shop-Problem reduziert werden. So wird eine (möglicherweise bereits schwache) Relaxation jedoch nochmals sehr stark abgeschwächt. Ferner gelten die in den obigen Arbeiten präsentierten Beweise nur für Permutation Schedules sowie Probleme mit unbegrenztem Zwischenspeicher, können also nicht auf Situationen mit Blocking-Constraints etc. übertragen werden. Auch Soukhal und Martineau (2005) [132] entwickeln eine kombinatorische Schranke für das Standard Flow-Shop-Problem mit Minimierung der Zykluszeit, welche jedoch nicht auf das Hybrid Flow-Shop-Problem übertragen werden kann.

Neben LP-Relaxationen und kombinatorischen Ansätzen basiert Schrankenbildung für (Hybrid) Flow-Shop-Probleme häufig auf einer geeigneten Problemdekomposition (vgl. dazu auch Abschnitt 3.2). So werden Schranken beispielsweise Stufe für Stufe berechnet und zwar so, dass diese anschließend zu einer Schranke für das Gesamtproblem kombiniert werden können. In entsprechenden Arbeiten steht jedoch für gewöhnlich die Minimierung der Zykluszeit im Vordergrund. Des Weiteren werden diese Schranken meist im Hinblick auf ein Branch & Bound Verfahren konstruiert und nicht um besonders starke Schranken für ein Problem zu berechnen (vgl. dazu u. a. Brah & Hunsucker, 1991, [25]; Rajendran & Chaudhuri, 1992, [115]; Santos et al., 1995, [120]; Portmann et al., 1998, [107]).

Sowohl bei der Problemdekomposition als auch bei der Entwicklung von Branch & Bound Verfahren spielen Ansätze, die auf einer Lagrange-Relaxation basieren eine wichtige Rolle.[10] In den bekannten Ansätzen werden dazu verschiedene Restriktionen relaxiert. Oftmals sind dies die Kapazitätsrestriktionen, in den letzten Jahren wird aber auch vermehrt die Reihenfolge der Maschinenstufen relaxiert. Für nähere Informationen über die einzelnen Methoden sei an dieser Stelle auf die Arbeiten von Hoogeveen und van den Velde (1995) [55], Liu und Chang (2000) [85], Hoogeveen et al. (2006) [54], Tang et al. (2006) [138] sowie Li und Yang (2009) [83] verwiesen.

Verschiedenartige untere Schranken für das Problem $Pm|r_j|\sum w_jC_j$ untersuchen Baptiste et al. (2008) [14]. Dazu wird dieses zunächst auf mehrere Weisen kombinatorisch relaxiert, z. B. werden die Freigabezeiten der Jobs vernachlässigt (entweder vollständig oder jeweils für Jobgruppen). Darüber hinaus wird das Problem auch hier mit Hilfe von Job-Splitting (in Teiljobs der Länge 1) auf ein Flussproblem reduziert. Zusätzlich untersuchen die Autoren die zeit-indizierte Formulierung des Problems mit Startzeitvariablen. Dabei wird die LP-Relaxation des Modells genauso berücksichtigt wie LR-basierte Strategien. Diese beiden Vorgehensweisen liefern in der genannten Arbeit insgesamt die besten unteren Schranken. Der LR-Ansatz schneidet dabei zwar etwas besser ab, der entstehende Rechenaufwand ist aber sogar noch größer als bei der Lösung der LP-Relaxation.

[10] Diese wird oftmals im Zusammenhang mit dynamischer Programmierung eingesetzt.

5. Heuristische Lösung von HFS-Problemen mit verschiedenen Restriktionen

Motiviert durch das eingangs beschriebene Anwendungsproblem wird in diesem Kapitel ein heuristisches Lösungsverfahren zur Minimierung der gewichteten Gesamtfertigstellungszeit in einer Hybrid Flow-Shop-Umgebung entwickelt. Neben Freigabezeiten der Jobs werden dabei verschiedene zusätzliche Restriktionen berücksichtigt. Zunächst untersuchen wir die Probleme $FHm,(PM^{(k)})_{k=1}^{m}|r_j|\sum w_jC_j$ mit unbegrenztem Zwischenspeicher sowie $FHm,(PM^{(k)})_{k=1}^{m}|r_j,no-wait|\sum w_jC_j$ mit No-Wait-Constraints (vgl. Abschnitt 5.2). Darüber hinaus wird das in diesem Zusammenhang entwickelte Lösungsverfahren zur Behandlung von Transportrestriktionen erweitert (vgl. auch Abschnitt 5.4). Abschließend modifizieren wir die Methoden zur Behandlung der entsprechenden Situationen mit dynamisch eintreffenden Jobs (vgl. Abschnitt 5.5).

In sämtlichen hier untersuchten Problemklassen werden identisch parallele Maschinen auf den verschiedenen Bearbeitungsstufen vorausgesetzt. Unter dieser Annahme ist die konkrete Allokation der Maschinen einer Stufe grundsätzlich beliebig, sofern dabei die Gesamtkapazität der Stufe zu keinem Zeitpunkt überschritten wird (vgl. dazu auch Liu & Karimi, 2008, [87]). Infolgedessen wird der Umfang einer Problemstellung unter Umständen erheblich reduziert. Im Hinblick auf das in dieser Arbeit betrachtete Anwendungsproblem ist die Voraussetzung identischer Maschinen jedoch nur bedingt zutreffend (vgl. Kapitel 7). Aus diesem Grund wird im Folgenden zusätzlich ein heuristischer Ansatz zur Lösung des Problems mit konkreter Maschinenauswahl entwickelt (vgl. Abschnitt 5.3).

Zur Lösung des Anwendungsproblems ist ein echtzeitfähiges Verfahren mit sehr geringer Rechenzeit erforderlich. Für genauere Informationen sei an dieser Stelle auf Kapitel 7 verwiesen. Aus der in Kapitel 3 vorgestellten Literaturrecherche geht hervor, dass eine adäquate Methode, welche diesen Anforderungen genügt, auf geeigneten einfachen Prioritätsregeln (insbesondere der WSPT-Regel) und einer List Scheduling Strategie basieren kann. Dieser Aspekt sowie der im folgenden Abschnitt dargestellte Dekompositionsansatz bilden die Grundlage für das im Rahmen der vorliegenden Arbeit entwickelte Verfahren. Dieses dient zunächst der Lösung der statischen Versionen obiger Problemstellungen, kann aufgrund seiner geringen Rechenzeit jedoch auch als On-line Verfahren bei dynamisch eintreffenden Jobs eingesetzt werden (siehe Abschnitt 5.5). Eine ähnliche Strategie verfolgen beispielsweise auch Alvarez-Valdes et al. (2005) [8] bei der Bearbeitung eines Anwendungsproblems. Auch hier ist ein echtzeitfähiges Verfahren erforder-

lich, welches die Autoren mit Hilfe einer schnellen Heuristik realisieren. Diese basiert auf einer zweistufigen Problemdekomposition und generiert unter Verwendung von Prioritätsregeln zunächst einen möglichst kompakten Schedule mit hoher Maschinenauslastung. Anschließend wird eine Local Search Methode zur Verbesserung dieser initialen Lösung implementiert.

5.1. Dekompositionsansatz

Die eingangs genannten Scheduling-Probleme sind sämtlich *NP*-schwer. Besonders komplex sind jedoch ohne Zweifel diejenigen Aufgaben, in welchen zusätzlich Transportrestriktionen eine Rolle spielen. Bei der heuristischen Lösung der betreffenden Problemstellungen liegt daher ein zwei- bzw. sogar dreistufiger Dekompositionsansatz zugrunde.

Prinzipiell besteht die Möglichkeit, die vorhandenen Transportmittel (ähnlich wie bei der Bestimmung von unteren Schranken für dieses Szenario in Abschnitt 4.3.2) als weitere Bearbeitungsstufe(n) zu interpretieren und so insgesamt ein Hybrid Flow-Shop-Problem mit höherer Maschinenzahl, jedoch ohne Transportrestriktionen zu lösen. Im Fall des hier behandelten Anwendungsproblems besitzen die Transportmittel jedoch vollkommen andere Eigenschaften als die übrigen Maschinen und unterliegen somit auch anderen Restriktionen. Berücksichtigt man nun beide „Arten von Maschinen“ simultan, so wird das Problem für beide unnötig komplex, da schlimmstenfalls sämtliche Randbedingungen für alle Stufen berücksichtigt werden müssen. Darüber hinaus bestimmen die anfallenden Transportzeiten im untersuchten Problem nur einen Bruchteil der gesamten Flusszeit eines Jobs. Mit anderen Worten, die Verplanung der „echten“ Maschinen ist ausschlaggebend für die Performance des gesamten Systems.

Aus den oben genannten Gründen werden die betrachteten Scheduling-Probleme mit Transporten in zwei Teilprobleme zerlegt, die konsekutiv behandelt werden. Im ersten Schritt lösen wir das jeweilige Hybrid Flow-Shop-Problem ohne Transporte. Dieses Teilproblem wird im weiteren Verlauf als *Sequencing-Problem* bezeichnet. Bei dessen Lösung ist zusätzlich entscheidend, ob die Allokation der Maschinen einer Stufe grundsätzlich beliebig ist oder ebenfalls optimiert werden soll. Im zweiten Fall wird die konkrete Maschinenauswahl als weiteres separates Teilproblem betrachtet. In Abschnitt 3.2 haben wir diesen gebräuchlichen problem-orientierten Dekompositionsansatz bereits vorgestellt. Basierend auf der Lösung des Sequencing-Problems verplanen wir nun die vorhandenen Transportmittel zur Umsetzung der daraus resultierenden Transportaufträge. Dieses Teilproblem wird im Folgenden als *Routing-Problem* bezeichnet.

Das in den nächsten Abschnitten präsentierte heuristische Lösungsverfahren ist gemäß der beschriebenen Dekomposition in drei Phasen gegliedert. In *Phase 1* werden die Jobs für jede Bearbeitungsstufe sequenziert, d. h. wir lösen das zugrundeliegende Hybrid Flow-Shop-Problem zunächst ohne konkrete Allokation der Maschinen. Diese erfolgt in *Phase*

2 des Algorithmus', basierend auf einem Schedule für die Maschinenstufen gemäß Phase 1. Falls bei der Lösung des Problems zudem Transportrestriktionen berücksichtigt werden sollen, so wird eine Lösung für das Routing-Problem basierend auf der Lösung des Sequencing-Problems in *Phase 3* des Verfahrens ermittelt.[1]

5.2. Lösung des Sequencing-Problems ohne Maschinenauswahl (Phase 1)

Das Sequencing-Problem ohne konkrete Maschinenauswahl kann, wie zu Beginn dieses Kapitels diskutiert, als Scheduling-Problem des Typs $FHm,(PM^{(k)})_{k=1}^{m}|r_j|\sum w_jC_j$ oder $FHm,(PM^{(k)})_{k=1}^{m}|r_j,no-wait|\sum w_jC_j$ klassifiziert werden, je nachdem von welcher Art des Zwischenspeichers wir ausgehen. Das entsprechend entwickelte heuristische Lösungsverfahren ist eine job-orientierte List Scheduling Strategie, bei deren Umsetzung mehrere initiale Jobsequenzen berücksichtigt werden. Diese werden unter Verwendung verschiedener Prioritätsregeln generiert. Nach Anwendung des List Scheduling Verfahrens auf jede dieser Jobsequenzen wird letztlich diejenige mit dem besten zugehörigen Zielfunktionswert im Hinblick auf das Sequencing-Problem ausgewählt. Auf diese Weise entsteht ein hinreichend schnelles Hybridverfahren, welches die Vorteile mehrerer einfacher Prioritätsregeln ausnutzt. Wir stellen nun in Abschnitt 5.2.1 zunächst den verwendeten List Scheduling Ansatz vor und gehen in Abschnitt 5.2.2 detailliert auf die Bestimmung der initialen Jobsequenzen ein.

5.2.1. Eine job-orientierte List Scheduling Strategie

Die im Rahmen dieser Arbeit untersuchte Zielfunktion $\sum w_jC_j$ hängt von der Fertigstellungszeit jedes einzelnen Jobs ab; wie von Queyranne und Schulz (2006) [113] hervorgehoben, sollte eine entsprechende List Scheduling Strategie daher job-orientiert sein. In unserem Fall wird jeder Job so in den bestehenden partiellen Schedule eingefügt, dass dieser auf jeder Bearbeitungsstufe so früh wie möglich gestartet werden kann. Dabei muss natürlich die zur Verfügung stehende Kapazität jeder Stufe berücksichtigt werden. Bei dieser Vorgehensweise ist es grundsätzlich möglich, dass eine Maschine leer läuft, obwohl eigentlich ein oder mehrere Jobs zur Bearbeitung anstehen. Demnach handelt es sich um einen job-orientierten Ansatz. Im Fall von No-Wait-Constraints reduziert sich die Suche nach den Startzeitpunkten der einzelnen Operationen eines Jobs auf die erste Bearbeitungsstufe, da sich die Startzeiten auf den folgenden Stufen daraus bereits implizit ergeben. Im Folgenden algorithmisieren wir nun die dynamische Variante dieses Verfahrens, d. h. sobald ein neuer Job eintrifft, wird der bereits bestehende partielle Schedule reoptimiert. Dabei werden allerdings nur die Jobs berücksichtigt, welche sich zum be-

[1] Eine ähnliche Dekomposition wird auch von Jungwattanakit et al. (2008) [62] verwendet.

treffenden Zeitpunkt noch nicht in Bearbeitung befinden. Im statischen Fall verläuft das Verfahren analog, es werden lediglich alle Jobs gleichzeitig berücksichtigt. Zur Beschreibung des Algorithmus' verwenden wir die folgenden zusätzlichen Notationen:

- $J(t)$ - Menge der Jobs j, die zum Zeitpunkt t bereits freigegeben aber noch nicht fertiggestellt sind, d. h. $r_j \leq t < C_j$
- $L(t)$ - Menge der Jobs $j \in J(t)$, die zum Zeitpunkt t freigegeben aber noch nicht in Bearbeitung sind
- $cap(i,t) \leq M^{(i)}$ - Auslastung der Bearbeitungsstufe i zum Zeitpunkt t; zusätzlich verwenden wir den Parameter $cap'(i,t)$, um die temporäre Auslastung der Stufen während der Anwendung des List Scheduling Verfahrens für verschiedene Jobsequenzen zu kennzeichnen
- s_{ij} - Startzeit des Jobs j auf Stufe i; die Startzeiten auf den Stufen i mit $i > 1$ ergeben sich unter No-Wait-Constraints dabei implizit aus s_{1j}
- H - Menge der verwendeten Prioritätsregeln zur Generierung initialer Jobsequenzen

Die oben beschriebene Vorgehensweise kann für No-Wait-Constraints nun insgesamt wie folgt formalisiert werden (vgl. dazu auch Anhang A):

Algorithmus 5.2.1 ***(List Scheduling - No-Wait (dynamisch)):***

Zu Beginn ist noch kein Job verplant und wir setzen $cap(i,t) = 0$ für alle i,t. Wird zum Zeitpunkt r_j ein Job j freigegeben, verfahren wir wie folgt:

1. *Setze $t_0 = r_j$ und $s_{ij} = T$ für alle Stufen $i = 1,\ldots,m$.*
2. *Füge j zu $J(t_0)$ hinzu und setze $L(t_0) := \{k \in J(t_0) | r_k \leq t_0 \leq s_{1k}\}$.*
   Für alle $k \in L(t_0)$ und alle Stufen $i = 1,\ldots,m$ setze $s_{ik} = T$.
3. *Für alle Jobs in $J(t_0)$:*
   Falls $s_{1j} < T$
   Setze $cap(i,t) = cap(i,t) - 1$ für $t = s_{ij},\ldots,s_{ij} + p_{ij} - 1$ für alle Stufen $i \in I$.
4. *Für jede Heuristik $h \in H$:*
   Wende h auf $L(t_0)$ an. Die resultierende Jobsequenz wird dann mit $j_1^h,\ldots,j_r^h$ bezeichnet.
   Für alle Jobs $j = j_1^h,\ldots,j_r^h$:
   a) Setze $t_{j_k}^h = t_0$.
   b) Setze $cap'(i,t) = cap(i,t)$.

c) Falls $cap'(i,t) = M^{(i)}$ *für mindestens eine Stufe* i *und ein* $t = t^h_{j_k} + \sum_{l=1}^{i-1} p_{l\,j^h_k}, \ldots, t^h_{j_k} + \sum_{l=1}^{i} p_{l\,j^h_k} - 1$

Setze $t^h_{j_k} = t^h_{j_k} + 1$ *und überprüfe die obige Bedingung erneut.*

Andernfalls

Setze $s^h_{1\,j_k} = t^h_{j_k}$ *und* $s^h_{i\,j_k} = s^h_{i-1\,j_k} + p_{i-1\,j^h_k}$ *für* $i = 2, \ldots, m$.

Setze $C^h_{j_k} = s^h_{1\,j_k} + \sum_{i=1}^{m} p_{i\,j^h_k}$.

Setze $cap'(i,t) = cap'(i,t) + 1$ *für* $i = 1, \ldots, m$.

Setze $t = s^h_{i\,j_k}, \ldots, s^h_{i\,j_k} + p_{i\,j^h_k} - 1$.

d) Berechne den zugehörigen Zielfunktionswert $z^h := \sum_{j \in L(t_0)} w_j C^h_j$.

5. Wähle diejenige Jobsequenz $j^{h^*}_1, \ldots, j^{h^*}_r$ *mit*

$$z^{h^*} := \min_{h \in H} z^h.$$

6. Für alle Jobs in $L(t_0)$ *und alle Stufen* $i = 1, \ldots, m$*:*

Setze $s_{1\,j_1} = s^{h^*}_{1\,j}$ *und* $s_{ij} = s^{h^*}_{ij}$.

Setze $cap(i,t) = cap(i,t) + 1$ *für* $i = 1, \ldots, m$ *und* $t = s_{ij}, \ldots, s_{ij} + p_{ij} - 1$.

Die Komplexität dieses Algorithmus' ist $O(np^2)$ wobei $p := \sum p_{ij}$, d. h. die Methode ist lediglich pseudopolynomial. Beim Einfügen eines neuen Jobs in den aktuellen partiellen Schedule muss dessen Startzeit auf der ersten Maschine bestimmt werden. Aus diesem Grund muss für jede mögliche Startzeit, beginnend bei t_0 (vgl. Schritt 4a), überprüft werden, ob die korrespondierenden Maschinenkapazitäten zur Verfügung stehen oder nicht (vgl. Schritt 4c). Beides ist von der Zeit (also insbesondere T) abhängig, wobei die Summe sämtlicher Bearbeitungszeiten als obere Schranke für T gewählt werden kann. In der Praxis müssen natürlich meist wesentlich weniger Startzeitpunkte überprüft werden und die Summe der Bearbeitungszeiten wird für gewöhnlich nicht beliebig groß. Die Laufzeit des Verfahrens genügt daher dennoch den beschriebenen Anforderungen (vgl. dazu auch Kapitel 7).

Algorithmus 5.2.1 kann in ähnlicher Form auch für das Problem mit unbegrenztem Zwischenspeicher verwendet werden. Dazu müssen wir lediglich Schritt 4 wie folgt anpassen:

Algorithmus 5.2.2 ***(List Scheduling - Unlimited (dynamisch)):***

Wir verwenden die Schritte 1 - 3 sowie 5 - 7 aus obigem Verfahren und ersetzen 4 durch:

4.' Für jede Heuristik $h \in H$*:*

Wende h *auf* $L(t_0)$ *an. Die resultierende Jobsequenz wird dann mit* $j^h_1, \ldots, j^h_r$ *bezeichnet.*

Für alle Jobs $j = j_1^h, \ldots, j_r^h$*:*

a) Setze $t^h_{0j_k} = t_0$.

b) Setze $cap'(i,t) = cap(i,t)$.

c) Für alle Maschinentypen $i = 1, \ldots, m$*:*

Beginne mit $t^h_{ij_k} = s^h_{(i-1)j_k} + p_{(i-1)j^h_k}$*, dabei sei* $p_{0j^h_k} := 0$ *und* $s^h_{0j_k} := t^h_{0j_k}$.

Falls $cap'(i,t) = M^{(i)}$ *für mindestens ein* $t = t^h_{ij_k}, \ldots, t^h_{ij_k} + p_{ij^h_k} - 1$ *gilt*

Setze $t^h_{ij_k} = t^h_{ij_k} + 1$ *und prüfe die Bedingung erneut.*

Andernfalls

Setze $s^h_{ij} = t^h_{ij_k}$.

Setze $C^h_{ij_k} = s^h_{ij_k} + p_{ij_k}$.

Setze $cap'(i,t) = cap'(i,t) + 1$ *für* $t = s^h_{ij_k}, \ldots, s^h_{ij_k} + p_{ij_k} - 1$.

d) Berechne den zugehörigen Zielfunktionswert $z^h = \sum_{j \in L(t_0)} w_j C^h_j$.

Der nun folgende Abschnitt ist der Erzeugung initialer Jobsequenzen gewidmet, welche im Zusammenhang mit beiden List Scheduling Ansätzen verwendet werden können.

5.2.2. Bestimmung initialer Jobsequenzen

Zur Bestimmung geeigneter Sequenzierungsregeln sind verschiedene Strategien denkbar. Wie vorangehend bereits diskutiert, sind hier im Hinblick auf die erforderliche Rechenzeit vor allem einfache Prioritätsregeln naheliegend. In der Literatur konnte in diesem Zusammenhang oftmals die WSPT-Regel erfolgreich adaptiert werden (vgl. Abschnitt 3.4.1), weshalb diese auch im Folgenden eine große Rolle spielt. Des Weiteren erscheint die Verwendung einer engpass-orientierten Strategie sinnvoll (vgl. Abschnitt 3.4.1.4), wobei hierbei mehrere Varianten zur Definition entsprechender Prioritätsregeln in Frage kommen. Die Verwendung einer LP-basierten Problemrelaxation ist an dieser Stelle dagegen eher von theoretischem Interesse und wegen der hohen zu erwartenden Rechenzeit nicht für die Lösung des praktischen Anwendungsproblems geeignet (vgl. auch Leung et al., 2006, [81]). Insgesamt werden im Folgenden mit Hilfe der genannten Werkzeuge fünf verschiedene initiale Jobsequenzen generiert.

1. Verwendung der WSPT-Regel

Für ein Hybrid Flow-Shop-Problem kann die WSPT-Regel nicht direkt angewendet werden, da sich jeder Job aus mehreren Operationen zusammensetzt. Man kann die Jobs alternativ aber in Bezug auf ihre Gesamtbearbeitungszeit ($\sum_i p_{ij}$) untersuchen und diesbezüglich gemäß der WSPT-Regel sortieren. In diesem Zusammenhang spricht man in der Literatur oftmals auch von der WSTP[2]-Regel. Wir erhalten so folgende Strategie:

[2]WSTP steht dabei für Weighted Shortest Total Processing Time first.

(1) Die Jobs werden in aufsteigender Reihenfolge ihrer gewichteten Gesamtbearbeitungszeit sequenziert, d. h. wir sortieren die Jobs in der Reihenfolge $j_1, \ldots, j_n$, falls

$$\frac{\sum_{i \in I} p_{ij_1}}{w_{j_1}} \leq \frac{\sum_{i \in I} p_{ij_2}}{w_{j_2}} \leq \ldots \leq \frac{\sum_{i \in I} p_{ij_n}}{w_{j_n}}.$$

2. Engpass-orientierte Strategien

Den Nutzen engpass-orientierter Methoden haben wir in Abschnitt 3.4.1.4 bereits eingehend diskutiert. Wie wichtig eine gute Verplanung der Engpassstufe für das gesamte System sein kann, wird zudem bei Kochhar und Morris (1987) [70] hervorgehoben. Der Engpass eines Systems kann dabei auf verschiedene Weisen definiert werden. Wir wählen hier als Flaschenhals analog zu Paternina-Arboleda et al. (2008) [102] diejenige Bearbeitungsstufe mit der größten relativen Auslastung, d. h. unter allen Stufen $i \in I$, wählen wir diejenige i^* mit

$$w(i^*) = \max_{i \in I} w(i), \text{ wobei } w(i) := \frac{\sum_{j \in J} p_{ij}}{M^{(i)}} \text{ für alle } i \in I.$$

Basierend auf dieser Definition des Flaschenhalses gibt es nun ebenfalls verschiedene Möglichkeiten, die Jobs zu sortieren. Wir interpretieren die Maschinen des Typs i^* dazu als Parallelmaschinen-Problem und bestimmen diesbezüglich eine Jobfolge, die dann durch Anwendung der List Scheduling Strategie auf das gesamte Problem übertragen wird. Dies kann vollkommen unabhängig von den übrigen Bearbeitungsstufen erfolgen, alternativ kann man aber auch Informationen über andere Maschinentypen in geeigneter Form mit in das betrachtete Parallelmaschinen-Problem einbeziehen. Wir betrachten zunächst für die Engpassstufe i^* das Problem $PM^{(i^*)}||\sum w_j C_j$ und sequenzieren die Jobs zu dessen Lösung mit Hilfe der WSPT-Regel, d. h.:

(2.1) Die Jobs werden in der Reihenfolge $j_1, \ldots, j_n$ sortiert, falls

$$\frac{p_{i^* j_1}}{w_{j_1}} \leq \frac{p_{i^* j_2}}{w_{j_2}} \leq \ldots \leq \frac{p_{i^* j_n}}{w_{j_n}}.$$

Auf diese Weise werden jedoch sämtliche zur Verfügung stehenden Informationen über die Bearbeitungszeiten der Jobs auf den übrigen Stufen nicht berücksichtigt, was möglicherweise eine insgesamt ungünstige Sortierung der Jobs zur Folge hat. Aus diesem Grund bestimmen wir in Analogie zu einer Arbeit von Bertel und Billaut (2004) [18] die frühestmögliche Startzeit aller Jobs auf der Engpassstufe i^* in Abhängigkeit der Bearbeitungszeiten auf den vorangehenden Stufen. Diese fiktiven Freigabezeiten $r_{i^* j}$ ergeben sich also wie folgt:

$$r_{i^* j} := \sum_{h=1}^{i^*} p_{(h-1)j}, \text{ wobei } p_{0j} := \max\{t_0, r_j\} \text{ für alle } j \in J.$$

So entsteht für i^* ein Parallelmaschinen-Problem des Typs $PM^{(i^*)}|r_j|\sum w_j C_j$. Dieses ist, wie in Abschnitt 3.3.2 beschrieben, nicht einfach zu approximieren. Die besten Resultate

basieren auf einer geeigneten Relaxation. Wir vereinfachen das Problem hier, indem wir Unterbrechungen der Jobs zulassen. Das daraus resultierende Parallelmaschinen-Problem $PM^{(i^*)}|r_j, prmp|\sum w_j C_j$ behandeln wir mit Hilfe der preemptive WSPT-Regel: Besitzt ein Job j die Bearbeitungszeit p_{i^*j} auf Maschinentyp i^*, besteht diese im relaxierten Problem aus p_{i^*j} Bearbeitungsschritten der Länge 1. Zu jedem Zeitpunkt verplant man nun maximal $M^{(i^*)}$ dieser Blöcke auf die vorhandenen Maschinen. Diese werden wie folgt bestimmt: Unter allen Jobs wählt man diejenigen mit den $M^{(i^*)}$ kleinsten Quotienten aus verbleibender Bearbeitungszeit und Gewicht (entsprechend weniger, wenn nicht genügend Jobs zur Bearbeitung anstehen) und verplant diese auf die vorhandenen Maschinen. Mit anderen Worten, sobald ein neuer Job eintrifft oder eine Maschine einen Job fertiggestellt hat, wird unter den zu verplanenden Jobs in $J(t)$ derjenige mit kleinster gewichteter verbleibender Bearbeitungszeit ausgewählt und verplant. Gegebenenfalls wird dazu ein Job, der sich bereits in Bearbeitung befindet, nun aber einen größeren Quotienten aufweist, unterbrochen.

Wenn so sämtliche Jobs j vollständig verplant sind, ergibt sich für diese eine Fertigstellungszeit C_j^* in dem erstellten Hilfsschedule mit Unterbrechungen. Mit Hilfe dieser Zeiten sequenzieren wir die Jobs nun auf drei verschiedene Arten:

(2.2) Wir sortieren die Jobs in der Reihenfolge $j_1, \ldots, j_n$, falls

$$C_{j_1}^* \leq C_{j_2}^* \leq \ldots \leq C_{j_n}^*.$$

Bei dieser Anordnung der Jobs fallen unterschiedlich lange Bearbeitungszeiten der Jobs auf der Engpassstufe i^* allerdings nicht ins Gewicht, was unter Umständen zu einer ungünstigen Jobsequenz führen kann (vgl. dazu Queyranne & Schulz, 2006, [113]). Um diesen Aspekt bei der Verplanung ebenfalls einzubeziehen, verwenden wir zusätzlich die von den genannten Autoren vorgeschlagene Alternative:

(2.3) Wir setzen $M_j^* := C_j^* - \frac{p_{i^*j}}{2}$ (mittlere Fertigstellungszeit) und erhalten die Sortierung $j_1, \ldots, j_n$, falls

$$M_{j_1}^* \leq M_{j_2}^* \leq \ldots \leq M_{j_n}^*.$$

Zusätzlich zu dieser Methode verwenden wir einen α-Scheduler zur Sequenzierung der Jobs:

(2.4) Wir wählen $\alpha = \frac{1}{2}$, definieren also $C_j^*(\frac{1}{2})$ als denjenigen Zeitpunkt, zu dem $\frac{p_{i^*j}}{2}$ Einheiten des Jobs j fertiggestellt sind.[3] Wir verwenden dann die Sequenz $j_1, \ldots, j_n$, falls

$$C_{j_1}^*(\tfrac{1}{2}) \leq C_{j_2}^*(\tfrac{1}{2}) \leq \ldots \leq C_{j_n}^*(\tfrac{1}{2}).$$

[3] Da der Hilfsschedule Unterbrechungen enthält, können (2.3) und (2.4) unterschiedliche Jobsequenzen liefern.

Falls bei der Durchführung von Phase 1 nur wenige Jobs zur Bearbeitung anstehen, ist es nicht sinnvoll, ausschließlich diese Prioritätsregeln zu verwenden, da die Überprüfung aller möglichen Jobsequenzen keinen Mehraufwand bedeutet. Aus diesem Grund verwenden wir die oben genannten Prioritätsregeln im Folgenden nur, falls mindestens vier Jobs gleichzeitig zur Bearbeitung anstehen, d. h. falls $|L(t)| \geq 4$.

Die Komplexität der obigen Prioritätsregeln entspricht grundsätzlich der eines (schnellen) Sortierverfahrens, also $O(n \log n)$. Lediglich die Anwendung der preemptive WSPT-Regel bei der Berechnung der Sequenzen (2.2) - (2.4) hat einen Aufwand von $O(n^2 \log n)$, da jedes Mal, wenn ein weiterer Job eintrifft, erneut sortiert werden muss. Wegen $n \leq p$ ist die Gesamtkomplexität des List Scheduling Verfahrens unter Berücksichtigung der fünf initialen Listen also $O(np^2)$. Wir veranschaulichen die Arbeitsweise der gesamten dynamischen Methode nun anhand des folgenden Beispiels:

Beispiel 5.2.3 *Wir betrachten fünf Jobs sowie drei Bearbeitungsstufen mit den Kapazitäten* $M^{(1)} = 2$, $M^{(2)} = 1$, $M^{(3)} = 3$ *und den folgenden Daten:*

Job	*1*	*2*	*3*	*4*	*5*
r_j	*8*	*2*	*15*	*13*	*16*
w_j	*3*	*5*	*5*	*1*	*2*
p_{1j}	*1*	*8*	*6*	*6*	*7*
p_{2j}	*2*	*10*	*3*	*2*	*9*
p_{3j}	*8*	*2*	*9*	*10*	*3*

Wir wenden nun das dynamische List Scheduling Verfahren für das Problem mit No-Wait-Constraints an, beginnen also mit dem Zeitpunkt $t = 2$. *Wir haben* $J(2) = L(2) = \{2\}$ *mit* $s_2 = 2$ *und* $C_2 = 22$. *Zum Zeitpunkt* $t = 8$ *trifft Job* 1 *ein, aber da Job* 2 *bereits in Bearbeitung ist, haben wir nur* $L(8) = \{1\}$ *mit* $s_1 = 19$ *und* $C_1 = 30$. *Im nächsten Schritt erhalten wir* $L(13) = \{1,4\}$, *da Job* 1 *zu dieser Zeit noch nicht bearbeitet wird. Wegen* $|L(13)| < 4$ *berücksichtigen wir beide möglichen Sortierungen der Jobs. Die Folge* $1 \rightarrow 4$ *liefert den kleineren Zielfunktionswert. Deshalb erhalten wir* $s_1 = 19, C_1 = 30$ *sowie* $s_4 = 16, C_4 = 34$. *Die gleiche Situation tritt zum Zeitpunkt* $t = 15$ *mit* $L(15) = \{1,3,4\}$ *ein. Die Untersuchung aller sechs Sortierungen liefert als beste Liste* $1 \rightarrow 3 \rightarrow 4$ *mit* $s_1 = 19, C_1 = 30$, $s_3 = 16, C_3 = 34$ *und* $s_4 = 20, C_4 = 38$.

Zum Zeitpunkt $t = 16$ *trifft Job* 5 *ein und wir haben* $L(16) = \{1,3,4,5\}$. *Wir verwenden nun also die obigen fünf Prioritätsregeln zur Bestimmung einer Jobsequenz. Als Engpassstufe ergibt sich Maschinentyp* $i^* = 2$ *mit einer relativen Auslastung von* $w(2) = 21$. *Wir erhalten daher die fiktiven Freigabezeiten* $r_{21} = 17$, $r_{23} = 22$, $r_{24} = 22$ *und* $r_{25} = 23$. *Die Anwendung der preemptive WSPT-Regel liefert folgenden Hilfsschedule für Bearbeitungsstufe* 2*: Job* 1 *wird im Intervall* $[17,19]$ *vollständig bearbeitet, Job* 3 *kann* $[22,25]$ *zugeordnet werden sowie Job* 4 *dem Intervall* $[25,27]$ *und Job* 5 *dem Slot* $[27,36]$. *Die Verwendung dieses Schedules liefert die folgenden Prioritätsindizes und korrespondierenden Jobsequenzen:*

	1	*3*	*4*	*5*	*Sequenz*	*ZF-Wert*
(1)	*3.67*	*3.6*	*18*	*9.5*	$3 \rightarrow 1 \rightarrow 5 \rightarrow 4$	*386*
(2.1)	*0.67*	*0.6*	*2*	*4.5*	$3 \rightarrow 1 \rightarrow 4 \rightarrow 5$	*380*
(2.2)	*19*	*25*	*27*	*36*	$1 \rightarrow 3 \rightarrow 4 \rightarrow 5$	*380*
(2.3)	*18*	*23.5*	*26*	*31.5*	$1 \rightarrow 3 \rightarrow 4 \rightarrow 5$	*380*
(2.4)	*18*	*23.5*	*26*	*31.5*	$1 \rightarrow 3 \rightarrow 4 \rightarrow 5$	*380*

Wir erhalten in diesem Beispiel also denselben Zielfunktionswert für die Prioritätsregeln (2.1) - (2.4), können also beliebig eine der zugehörigen Joblisten auswählen. Beispielsweise (2.2) liefert die Sortierung $(2 \rightarrow)1 \rightarrow 3 \rightarrow 4 \rightarrow 5$ *mit einem zugehörigen Gesamtzielfunktionswert von* $380 + 5 \cdot 22 = 490$.

5.2.3. Ergänzende Bemerkungen

Falls basierend auf einer initialen Jobliste ein Schedule generiert werden soll, der Blocking-Constraints anstelle von No-Wait-Constraints oder unbegrenztem Zwischenspeicher genügt, so könnte man dazu den von Wang und Tang (2009) [149] entwickelten Greedy-Algorithmus verwenden. Zur Bestimmung der initialen Jobsequenz nutzen die Autoren in der genannten Arbeit die NEH -Heuristik bzw. deren Verallgemeinerung für die WCT-Zielfunktion. Mit Hilfe der dritten Phase unseres Verfahrens kann jedoch auch die Lösung des Sequencing-Problems mit No-Wait-Constraints in eine Lösung des gesamten Problems mit Blocking-Constraints und Transporten transformiert werden.

5.3. Maschinenauswahl (Phase 2)

Für den Fall, dass wir ein Hybrid Flow-Shop-Problem mit identisch parallelen Maschinen auf jeder Bearbeitungsstufe betrachten, ist die Auswahl der einzelnen Maschinen für jeden Job grundsätzlich beliebig, solange er in dem ihm zugewiesenen Zeitintervall bearbeitet werden kann. Die Start- und Fertigstellungszeiten der Jobs auf jeder Stufe wurden in Phase 1 bereits bestimmt. Da hierbei auch die zugehörigen Kapazitäten berücksichtigt wurden, sind die Existenz und Zulässigkeit einer entsprechenden Maschinenauswahl gewährleistet. Die Allokation konkreter Maschinen erfolgt formal in der im weiteren Verlauf beschriebenen initialen Maschinenauswahl. Falls es sich um ein Problem mit parallelen Maschinen unterschiedlicher „Qualität“ handelt, kann man diese initiale Zuordnung anschließend noch mit Hilfe eines einfachen Greedy-Algorithmus' (vgl. Abschnitt 5.3.2) verbessern.

5.3.1. Initiale Zuordnung

Zur Bestimmung einer initialen Maschinenauswahl für einen Maschinentyp i sortieren wir zunächst alle zu berücksichtigenden Jobs j aufsteigend gemäß ihrer Startzeiten s_{ij},

welche aus Phase 1 resultieren. Die Maschinen numerieren wir in beliebiger Reihenfolge. Nun wird den Jobs der Reihe nach eine Maschine zugewiesen und zwar jeweils diejenige mit kleinster Nummer, welche im entsprechenden Zeitraum zur Verfügung steht. Diese Vorgehensweise kann man wie folgt formalisieren:

Algorithmus 5.3.1 ***(Initiale Maschinenauswahl für Typ i (statisch))***

Input:

In Phase 1 bestimmte Jobsequenz $j_1,\ldots,j_n$ mit den Start- und Fertigstellungszeiten $(s_{ij_1},C_{ij_1}),\ldots,(s_{ij_n},C_{ij_n})$.

Output:

Geordnete Jobliste S_{il} für jede Maschine l des Typs i und für jedes Paar aus Job j und Bearbeitungsstufe i ein Parameter $m(i,j)=l$, falls Job j auf Maschine l des Typs i bearbeitet wird.

Algorithmus:

1. *Sortiere die Jobs j_k in aufsteigender Reihenfolge ihrer Startzeiten s_{ij_k}. So ergibt sich die Sequenz $j'_1,\ldots,j'_n$ mit $s_{ij'_k}\leq s_{ij'_{k+1}}$ für $k=1,\ldots,n-1$.*
2. *Für diese Jobs wähle nun wie folgt eine Maschine des Typs i aus:*

   Für alle $k=1,\ldots,n$:

   Beginne mit $l=1$.

   Falls ein Index r existiert, mit $m(i,j'_r)=l$ und $C_{ij'_r}>s_{ij'_k}$ sowie $s_{ij'_r}\leq C_{ij'_k}$ dann setze $l=l+1$.

   Sonst

   Setze $m(i,j'_k)=l$.

   $S_{il}\leftarrow j'_k$, d. h. hänge j'_k an das Ende der Liste S_{il}.

Hierbei kann l nie größer werden als $M^{(i)}$, denn durch die Lösung aus Phase 1 ist gewährleistet, dass die Anzahl verfügbarer Maschinen eines Typs zu keinem Zeitpunkt überschritten wird. Die entstehende Maschinenbelegung ist also in jedem Fall zulässig. Die Komplexität des obigen Algorithmus' beträgt $O(n^2\cdot M^{(i)})$, da für jeden der n Jobs maximal alle $M^{(i)}$ Maschinen des Typs i durchlaufen werden. Während in der dynamischen Version von Phase 1 Jobs, die bereits in Bearbeitung sind, bei einer Reoptimierung nicht berücksichtigt werden, so können Operationen eines solchen Jobs, deren Bearbeitung selbst aber noch nicht begonnen hat, in einer dynamischen Variante der Phase 2 natürlich in eine Reoptimierung einbezogen werden. Ein ausführliches Beispiel für den Ablauf der initialen und verbesserten Maschinenauswahl ist am Ende des nachfolgenden Abschnitts zu finden.

5.3.2. Verbesserte Zuordnung

Im Fall von parallelen Maschinen unterschiedlicher Qualität[4] beginnen wir ebenfalls mit der oben beschriebenen initialen Maschinenauswahl, verbessern diese jedoch anschließend mit Hilfe eines einfachen Greedy-Algorithmus'. Im Wesentlichen vertauschen wir dazu paarweise die initiale Maschinenauswahl der Jobs, wenn möglich so, dass anschließend solche mit größerem Gewicht den besseren Maschinen zugeordnet sind. D. h. bei der intialen Auswahl $m(i,j) = l$ und $m(i,k) = l'$ erhalten wir in der verbesserten Auswahl $m(i,j) = l'$ und $m(i,k) = l$, falls $w_j < w_k$ und die Maschine l' besser ist als Maschine l.

Da bei der Vertauschung für zwei beliebige Jobs der Schedule gegebenenfalls seine Zulässigkeit verliert, gehen wir stattdessen wie folgt vor: Zunächst wird der initiale Schedule für jede Bearbeitungsstufe in Zeitintervalle zerlegt. Jobs, die innerhalb eines Intervalls derselben Maschine zugeordnet sind, werden zu einer Einheit zusammengefasst. Die Grenzen der Intervalle werden dabei so definiert, dass die Maschinenzuordnung für die Einheiten innerhalb eines Intervalls beliebig vertauscht werden darf. Um dies zu gewährleisten, definieren wir einen Zeitpunkt t_i als eine Nahtstelle zwischen zwei Intervallen, wenn für alle Maschinen des Typs mindestens eine der folgenden drei Bedingungen gilt (jedoch nicht für alle Maschinen Bedingung 3):

1. Zum Zeitpunkt t_i endet die Bearbeitung eines Jobs.
2. Zum Zeitpunkt t_i beginnt die Bearbeitung eines Jobs.
3. Zum Zeitpunkt t_i wird gar kein Job bearbeitet.

Wir bezeichnen diese Intervallzerlegung im Folgenden auch als Zerlegung des Schedules in Blöcke. Ein Block entspricht dann genau dem Zeitintervall zwischen zwei Nahtstellen $[t_i, t_{i+1}]$. Da bei der Bestimmung der Nahtstellen jeder Zeitpunkt für jede Bearbeitungsstufe genau einmal überprüft wird, besitzt diese den Aufwand $O(pM^{(i)})$. Abbildung 5.1 zeigt die Blockzerlegung der initialen Maschinenauswahl exemplarisch für einen Maschinentyp i der Kapazität $M^{(i)} = 3$ und acht Jobs.

Darüber hinaus wird jeder Maschine l des Typs i entsprechend der jeweiligen Problemstellung eine Güteklasse q_{il} zugewiesen. Je höher die Güteklasse, desto „besser“ ist die jeweilige Maschine. Bei der Verbesserung der initialen Maschinenauswahl versuchen wir daher, für jeden Block eine Zielfunktion der Struktur $\sum w_j q_{il}$ zu maximieren.

Alle Jobs, die in der initialen Maschinenauswahl innerhalb eines Blocks $[t_k, t_{k+1}]$ derselben Maschine l eines Typs i zugeordnet sind, werden im Folgenden als Einheit betrachtet und mit $U(i,l,k)$ bezeichnet. Wir erhalten gemäß der Intervallzerlegung demnach höchstens $M^{(i)}$ Einheiten für jeden Block des Maschinentyps i. Innerhalb eines Blocks werden

[4] Die Qualität einer Maschine kann sich dabei u. a. auf deren Geschwindigkeit beziehen; falls Transportrestriktionen berücksichtigt werden, aber auch auf deren Position und den daraus resultierenden Anfahrtsweg.

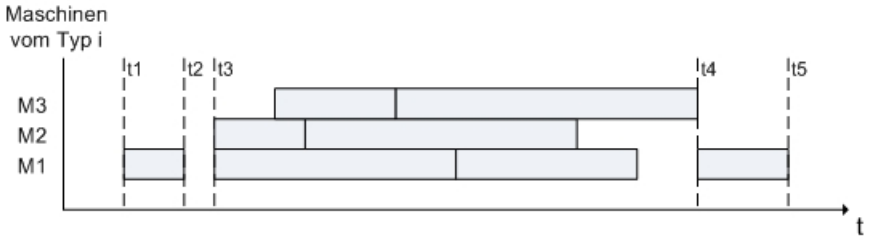

Abbildung 5.1.: Zerlegung eines Schedules aus Phase 1 in Blöcke

die Einheiten nun so vertauscht, dass obige Zielfunktion bezüglich dieser maximal wird. Eine entsprechende Zuordnung liefert ein einfacher Greedy-Algorithmus: Für die Einheit mit höchster Gesamtpriorität (d. h. mit größter Summe der zugehörigen Jobgewichte, $w_{U(i,l,k)} := \sum_{j \in U(i,l,k)} w_j$) wird diejenige Maschine mit größter Güteklasse ausgewählt. Dieses Vorgehen kann wie folgt formalisiert werden:

Algorithmus 5.3.2 ***(Verbesserte Maschinenzuordnung für Typ*** i ***(statisch))***

Zunächst wird die aktuelle Maschinenauswahl aufgehoben:

Für alle Intervalle $[t_k, t_{k+1}]$ *und für alle* $l = 1, \ldots, M^{(i)}$:[5]

Für alle $j \in U(i,l,k)$:

Setze $m(i,j) = 0$.[6]

Entferne j *aus* S_{il}.

Markiere die Maschine l *im Intervall* $[t_k, t_{k+1}]$ *als frei.*

Anschließend wird die neue verbesserte Maschinenbelegung ermittelt:

Für alle Intervalle $[t_k, t_{k+1}]$:

Solange noch unverplante Einheiten $U(i,l,k)$ *vorhanden sind:*

1. *Wähle unter den unverplanten Einheiten diejenige* $U(i,l^*,k)$ *mit maximalem Gewicht* $w_{U(i,l^*,k)}$.
2. *Wähle für diese Einheit nun diejenige freie Maschine* l *mit größter Güteklasse* q_{il}.
3. *Markiere* $U(i,l^*,k)$ *als verplant und die Maschine* l *für das Intervall* $[t_k, t_{k+1}]$ *als belegt.*
4. *Für alle Jobs* j *in* $U(i,l^*,k)$ *setze* $m(i,j) = l$.

[5] Wir gehen an dieser Stelle der Einfachheit halber davon aus, dass sämtliche Operationen berücksichtigt werden dürfen. Arbeitsgänge, die im dynamischen Fall nicht neu verplant werden dürfen, kann man in diesem Zusammenhang als blockiert markieren und die jeweilige Maschinenauswahl beibehalten. Für eine detailliertere Beschreibung sei auf Anhang A verwiesen.

[6] 0 bezeichnet hier den Default-Wert, d. h. $m(i,j) = 0$, falls noch keine Maschine ausgewählt wurde.

5. Füge j zu S_{il} hinzu.

Nachdem wie beschrieben für jede Jobeinheit eine Maschine ausgewählt wurde, erhalten wir als Resultat der zweiten Phase für jede Maschine l eines Typs i die zugeordnete Sequenz S_{il} von Jobs. Diese stellt gleichzeitig die Eingabe für die nächste Phase des Verfahrens, die Planung anfallender Transportaufgaben, dar. Eine vollständige formale Beschreibung der hier zum Teil nur verbal vorgestellten Vorgehensweise zur Maschinenauswahl kann Anhang A entnommen werden (insbesondere für den dynamischen Fall). Der Aufwand der verbesserten Maschinenauswahl für einen Maschinentyp i beträgt $O(pM^{(i)})$, denn insgesamt werden maximal p Intervalle untersucht und für jedes dieser Intervalle werden maximal $M^{(i)}$ Jobs neu zugeordnet. Bevor wir im nächsten Abschnitt auf die Lösung des Routing-Problems eingehen, wird der Ablauf von Phase 2 zunächst anhand des folgenden Beispiels veranschaulicht:

Beispiel 5.3.3 *Wir betrachten erneut die Probleminstanz aus Beispiel 5.2.3. Zusätzlich legen wir für die sechs Maschinen die folgenden Güteklassen fest:* $q_{11} = 1$, $q_{12} = 2$, $q_{21} = 1$, $q_{31} = 1$, $q_{32} = 2$ *und* $q_{33} = 3$. *Basierend auf der im letzten Schritt bestimmten Sequenzierung der Jobs führen wir nun die initiale und verbesserte Maschinenauswahl durch. Die Start- und Fertigstellungszeiten des dabei zugrundeliegenden Schedules sind in der folgenden Tabelle zusammengefasst:*

Job j	s_{1j}	C_{1j}	s_{2j}	C_{2j}	s_{3j}	C_{3j}
2	*2*	*10*	*10*	*20*	*20*	*22*
1	*19*	*20*	*20*	*22*	*22*	*30*
3	*16*	*22*	*22*	*25*	*25*	*34*
4	*20*	*26*	*26*	*28*	*28*	*38*
5	*22*	*29*	*29*	*38*	*38*	*41*

Zur initialen Maschinenauswahl verplanen wir die Jobs in der Reihenfolge ihrer Startzeiten und wählen für jeden Job diejenige freie Maschine mit kleinster Nummer. Auf diese Weise erhalten wir für unser Beispiel die in Abbildung 5.2 dargestellte initiale Maschinenbelegung für die drei Maschinentypen.

Der entstandene Schedule wird nun für die Maschinentypen 1 *und* 3 *in Blöcke zerlegt (vgl. Abbildung 5.3). Für Maschinentyp* 1 *erhalten wir die Liste* $\delta_1 = (2, 10, 16, 29)$ *von Nahtstellen sowie* $\delta_3 = (20, 22, 38, 41)$ *für den Typ* 3. *Ein Block entspricht genau dem Zeitintervall zwischen zwei Nahtstellen.*

Wir beginnen mit der Verbesserung der initialen Belegung für Maschinentyp 1. *Im Intervall* $[2, 10]$ *befindet sich lediglich Job* 2, *den wir daher auf die Maschine mit der größeren Güteklasse, hier also Maschine* 2, *verschieben. Im Intervall* $[16, 29]$ *haben wir vier Aufgaben, wobei die Jobs* 3 *und* 5 *sowie die Jobs* 1 *und* 4 *jeweils eine Einheit* $U(1, l, 3)$ *(*$l = 1, 2$*) bilden mit* $U(1, 1, 3) = \{3, 5\}$ *und* $U(1, 2, 3) = \{1, 4\}$ *sowie* $w_{U(1,1,3)} = w_3 + w_5 = 7$ *und*

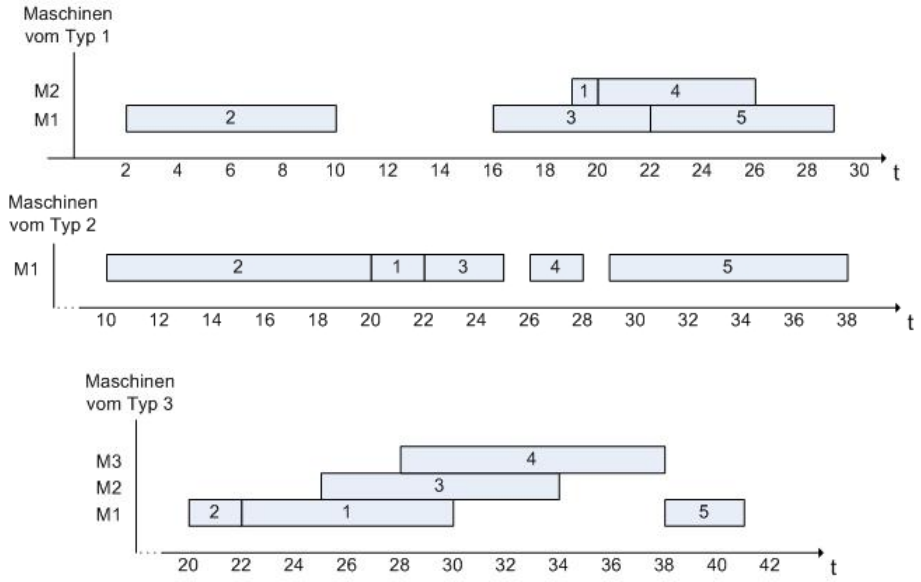

Abbildung 5.2.: Initiale Maschinenbelegung für Beispiel 5.3.3

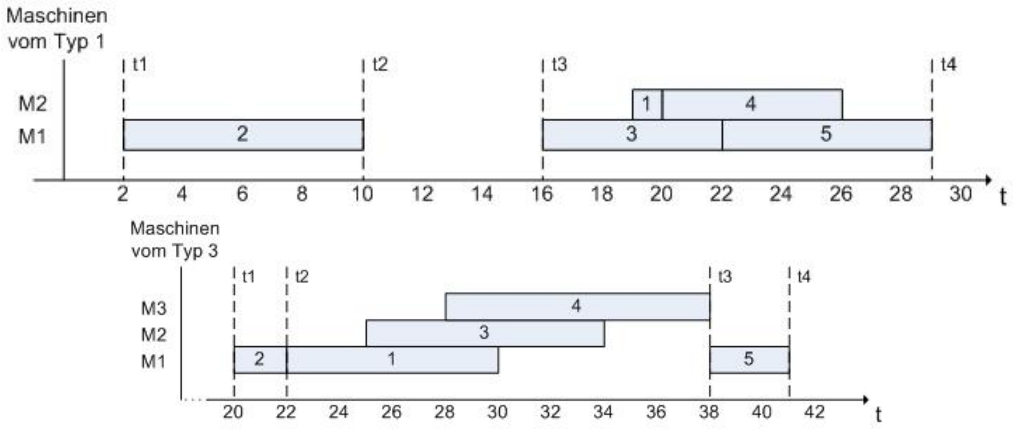

Abbildung 5.3.: Zerlegung der initialen Belegung in Blöcke für Beispiel 5.3.3

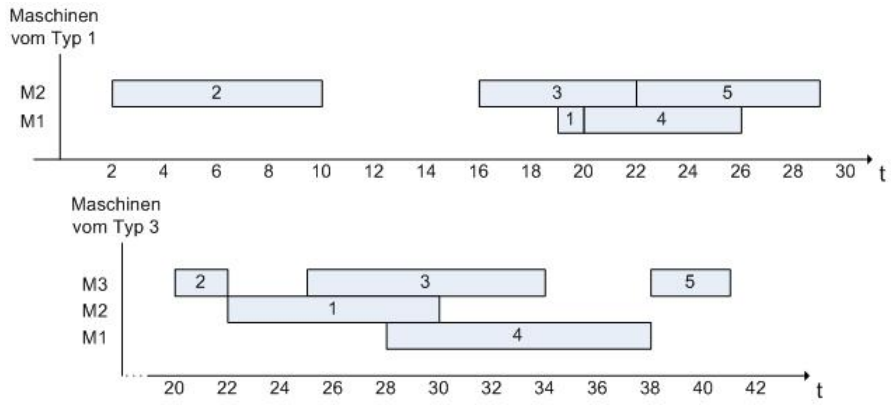

Abbildung 5.4.: Verbesserte Maschinenbelegung für Beispiel 5.3.3

$w_{U(1,2,3)} = w_1 + w_4 = 4$. *Die Anordnung der Einheiten muss also vertauscht werden, da* $q_{11} < q_{12}$. *Für Maschinentyp* 3 *verfahren wir analog und erhalten insgesamt die in Abbildung 5.4 dargestellte verbesserte Maschinenauswahl. Für Maschinentyp* 2 *ist eine Verbesserung der initialen Maschinenauswahl nicht notwendig, da hier nur eine einzige Maschine zur Verfügung steht.*

5.4. Lösung des Routing-Problems (Phase 3)

In der Praxis können Transportzeiten zwischen Bearbeitungsstufen vielfältige Gestalt haben. Sie können sowohl durch die räumliche Anordnung von Maschinen bestimmt werden, als auch durch die Art der eingesetzten Transportmittel. Beispielsweise kann die Transportzeit zwischen zwei Bearbeitungsstufen fest sein, unabhängig davon, welche der parallelen Maschinen zur Bearbeitung eines Jobs ausgewählt wurden. Genauso können sich die einzelnen Maschinen derselben Bearbeitungsstufe an unterschiedlichen Positionen befinden. Letzteres ist für die hier untersuchte Anwendungsaufgabe zutreffend und offensichtlich spielt in diesem Fall bereits die Maschinenauswahl in Phase 2 eine Rolle für die Organisation der entstehenden Transportaufgaben. Des Weiteren muss berücksichtigt werden, wie die zur Verfügung stehenden Transportmittel beschaffen sind: Sind diese schienengebunden oder frei beweglich? Falls mehrere Transportmittel zur Verfügung stehen, sind diese unabhängig voneinander oder kann es zu Kollisionen kommen? Diese und viele andere Fragen müssen bei der Lösung eines anwendungsorientierten Scheduling-Problems mit Transportrestriktionen natürlich beachtet werden.

In Anlehnung an das zugrundeliegende Praxisproblem (vgl. Kapitel 7) untersuchen wir im Rahmen dieser Arbeit folgende allgemeine Situation: Wir betrachten eine Menge von Transportmitteln, von denen grundsätzlich jedes für sich jeden der notwendigen Transporte durchführen kann. Die jeweils anfallende Transportzeit ist für jede Aufgabe eindeutig

bestimmt und hängt nicht von der Wahl des Transportmittels ab. Prinzipiell kann man also die vorhandenen Transportmittel als *identisch parallele Maschinen* interpretieren, die Transportaufträge entsprechen dann der Menge der zu verplanenden Jobs.

Darüber hinaus untersuchen wir einige zusätzliche Restriktionen. Die Menge der Jobs wird bestimmt durch die Lösung des Sequencing-Problems. Dementsprechend müssen bei deren Abarbeitung *Reihenfolgebedingungen* beachtet werden. Diese werden einerseits durch die vorgegebene Arbeitsgangfolge innerhalb der Flow-Shop-Umgebung definiert und andererseits durch die in Phase 1 und 2 ermittelten Jobsequenzen für jede Maschine festgelegt. Hierbei müssen gegebenenfalls Verzögerungszeiten (delays) zwischen aufeinander folgenden Transporten berücksichtigt werden: Angenommen, ein Job wird von Maschine 1 zu Maschine 2 und anschließend weiter zu Maschine 3 transportiert. Bevor der zweite Transport beginnen kann, muss die Bearbeitung des Jobs auf Maschine 2 natürlich abgeschlossen sein.

Ferner ist es möglich, dass bei der Abarbeitung der Aufgaben Leerfahrten der Transportmittel notwendig werden. Ist es beispielsweise günstig mit demselben Fahrzeug zunächst einen Job von A nach B und direkt im Anschluss einen anderen Job von C nach D zu transportieren, so ist das Fahrzeug bei der Fahrt von B nach C dann möglicherweise leer. Diese Situation kann man im Rahmen eines Scheduling-Problems als *sequenzabhängige Rüstzeit* auffassen. Schließlich übertragen sich mögliche Freigabezeiten der ursprünglichen Jobs auch auf die jeweils zugehörigen Transportaufträge. Infolgedessen kann man das Routing-Problem aus scheduling-theoretischer Sicht als Problem des Typs $Pm|r_j, sds, prec, delay|\sum w_j C_j$ klassifizieren. Dabei stehen Reihenfolgebedingungen und Rüstzeiten im Vordergrund. Freigabe- und Verzögerungszeiten können auch implizit durch einen geeigneten Reihenfolgegraphen abgebildet werden (vgl. Abschnitt 5.4.2.1). Bevor wir im weiteren Verlauf nun eine Heuristik zur Lösung des Routing-Problems präsentieren, diskutieren wir zunächst die in Abschnitt 3.3 bisher nicht berücksichtigte Literatur über WCT-Probleme mit Reihenfolgebedingungen und sequenz-abhängigen Rüstzeiten.

5.4.1. Theoretischer Hintergrund

Im Hinblick auf das Problem $Pm|sds, prec|\sum w_j C_j$ liegen nach unserem derzeitigen Kenntnisstand bislang gar keine Ergebnisse in der Literatur vor (vgl. dazu auch Allahverdi et al., 2008, [7]). Uns ist lediglich eine einzige Arbeit bekannt, welche sequenz-abhängige Rüstzeiten und Reihenfolgebedingungen als simultane Nebenbedingungen in einem Parallelmaschinen-Problem behandelt. Hurink und Knust (2001) [58] verfolgen dabei als Zielfunktion jedoch die Minimierung der Zykluszeit. Zudem steht in der genannten Arbeit der Beweis von Komplexitätsaussagen für Spezialfälle des Problems im Vordergrund. Im Folgenden fassen wir daher zunächst die relevante Literatur in Bezug auf Reihenfolgebedingungen zusammen und behandeln anschließend das vorhandene Material über sequenz-abhängige Rüstzeiten.

5.4.1.1. Reihenfolgebedingungen

Durch Hinzunahme von Reihenfolgebedingungen wird bereits das Problem $1||\sum w_j C_j$ im Allgemeinen *NP*-schwer. Für spezielle Strukturen von Reihenfolgegraphen kann man jedoch mit Hilfe einer Verallgemeinerung der WSPT-Regel auf einfache Weise eine optimale Lösung bestimmen. Z. B. für lineare Jobketten als Reihenfolgebedingungen[7] kann das Problem mit Hilfe des sogenannten ρ-*Faktors* gelöst werden.[8] Dieses Konzept geht auf eine Arbeit von Sidney (1975) [130] zurück. Diese beschäftigt sich grundsätzlich mit beliebigen Strukturen des Reihenfolgegraphen. Zur Lösung des Problems $1|prec|\sum w_j C_j$ wird dieser in sogenannte maximale ρ-minimale Teilgraphen zerlegt, man spricht in diesem Zusammenhang auch von der *Sidney-Dekomposition* eines Reihenfolgegraphen.[9] Für die erzeugten Teilgraphen kann man mit Hilfe der WSPT-Regel jeweils eine optimale Lösung ermitteln und anschließend die Teillösungen zu einer optimalen Lösung des Gesamtsystems zusammensetzen. Sidney beweist, dass ein Schedule für das $1|prec|\sum w_j C_j$-Problem genau dann optimal ist, wenn er konsistent mit seinem Verfahren ist. Zudem ist eine Gesamtlösung genau dann optimal, wenn ihre Einzelteile optimal in Bezug auf das entsprechende Teilproblem sind. Dies ist grundsätzlich ein vielversprechendes Resultat auch im Hinblick auf andere Scheduling-Probleme, welche Reihenfolgebedingungen unterliegen. Die Schwierigkeit besteht jedoch in einer effizienten Implementierung des Verfahrens, zumal die Sidney-Dekomposition eines Reihenfolgegraphen nicht eindeutig bestimmt ist. Bisher ist dies nur für spezielle Arten von Graphen gelungen (vgl. dazu u. a. Lawler, 1978, [74]). Margot et al. (2003) [89] versuchen in diesem Zusammenhang die feinste[10] aller Sidney-konsistenten Zerlegungen des Graphen zu bestimmen.

Die obigen Ausführungen verdeutlichen, dass es grundsätzlich sinnvoll ist, die Struktur vorhandener Reihenfolgebedingungen zu untersuchen und ein Verfahren daran anzupassen, um so gegebenenfalls bessere Lösungen zu erhalten als unter allgemeinen Restriktionen. Baev et al. (2002) [12] beschäftigen sich in diesem Zusammenhang vornehmlich mit linearen Ketten und Baumstrukturen,[11] genauer mit Bäumen vom Typ In-Tree bzw. Out-Tree.[12] Die von den Autoren vorgeschlagenen Verfahren basieren auf List Scheduling Strategien. Zunächst wird das ρ-Faktor-Konzept von Sidney (1975) [130] in Kombination mit dem Standard List Scheduling Verfahren adaptiert, welches für Ketten und Bäume im Fall einer einzelnen Maschine die optimale Lösung liefert. In einem zweiten Ansatz verknüpfen Baev et al. (2002) [12] die aus einer LP-Relaxation resultierende Jobfolge mit ei-

[7] D. h. jeder Job hat im Graphen höchstens einen Vorgänger und höchstens einen Nachfolger.

[8] Für detaillierte Informationen verweisen wir auf Pinedo (2008) [106].

[9] Eine exakte Beschreibung des Verfahrens führt an dieser Stelle zu weit. Wir verweisen daher auf die genannte Arbeit von Sidney (1975) [130].

[10] Eine Zerlegung in möglichst viele Teilgraphen wird als „fein“ bezeichnet.

[11] Unter einem Reihenfolgebaum versteht man einen Reihenfolgegraphen, in dem es zwischen jedem Paar von Jobs genau einen Weg aus Reihenfolgebeziehungen gibt, insbesondere treten also keine Kreise auf.

[12] D. h. es handelt sich um einen Reihenfolgebaum, in dem jeder Job maximal einen Nachfolger bzw. maximal einen Vorgänger hat.

nem α-Scheduler. Zur Evaluation der beiden Strategien wird eine umfassende empirische Studie durchgeführt, jedoch nur für Probleminstanzen mit den genannten speziellen Reihenfolgestrukturen. Insgesamt liefert diese zufriedenstellende Ergebnisse, lediglich die direkte Anwendung der WSPT-Regel (ohne Berücksichtigung des ρ-Faktors) führt in einigen Fällen zu einer hohen Abweichung von bis zu 30% im Vergleich zum optimalen Zielfunktionswert bzw. zur unteren Schranke. Zusätzlich wird auch die Qualität der durch die LP-Relaxation ermittelten unteren Schranke für das Problem ausgewertet, mit dem Ergebnis, dass diese in den meisten Fällen weit weniger als 1% von der optimalen Lösung abweicht.

Hall et al. (1997) [49] dagegen verwenden die LP-Relaxation geeigneter Modelle zur Bestimmung initialer Joblisten und wenden darauf das Standard List Scheduling Verfahren an, um so Probleme des Typs $1|prec|\sum w_jC_j$ sowie $1|r_j,prec|\sum w_jC_j$ zu bearbeiten. Auf diese Weise gelingt ihnen eine 2-Approximation der Variante mit Reihenfolgebedingungen sowie eine 3-Approximation bei zusätzlicher Berücksichtigung von Freigabezeiten. Auch im Hinblick auf Parallelmaschinen-Probleme ist eine solche Strategie gebräuchlich. In der genannten Arbeit erzielen die Autoren auf diese Weise eine 7-Approximation des Problems $Pm|r_j,prec|\sum w_jC_j$. Queyranne und Schulz (2006) [113] berücksichtigen ebenfalls simultan Freigabezeiten und Reihenfolgebedingungen. Dabei steht die Berechnung von Worst Case Performance Schranken im Vordergrund. Der verwendete Algorithmus basiert erneut auf einer LP-Relaxation des Problems. Unter Verwendung eines joborientierten List Scheduling Verfahrens werden die Jobs in der Reihenfolge ihrer mittleren Fertigstellungszeiten in der Lösung der Relaxation verplant (vgl. Jobsequenz (2.3) in Phase 1 unseres Verfahrens). Auf diese Weise kann eine 4-Approximation für das Problem $Pm|prec,delay|\sum w_jC_j$ ermittelt werden.

Chekuri et al. (2001) [33] verfeinern ihr in Abschnitt 3.3.2 bereits vorgestelltes List Scheduling Verfahren auch zur Lösung des Problems $Pm|r_j,prec|\sum w_jC_j$. Erneut kommt dabei eine Relaxation des Ausgangsproblems auf ein 1-Maschinen-Problem mit Job-Unterbrechungen zum Einsatz. Für dieses wird eine (nicht notwendig optimale) Jobsequenz erzeugt, welche den Reihenfolgebedingungen genügt. Anschließend wird diese jedoch nicht mit Hilfe der Standard List Scheduling Strategie verplant, sondern unter bestimmten Bedingungen werden Jobs vorgezogen, die gemäß der Sequenz eigentlich noch nicht zur Bearbeitung anstehen. Dahinter steckt der Versuch, die Balance zwischen unnötiger Verzögerung durch die Reihenfolgebedingungen und notwendiger Leerzeit von Maschinen herzustellen. Unter der Voraussetzung, dass eine optimale Lösung für das zugrundeliegende 1-Maschinen-Problem vorliegt, liefert dieses Verfahren eine 4-Approximation des Ausgangsproblems. Für Reihenfolgebedingungen mit In-Tree-Struktur kann dies sogar auf eine Schranke von 2 verbessert werden. Die Performance Schranke für das Problem $Pm|r_j,prec|\sum w_jC_j$ beträgt 5,83 bei beliebigen Reihenfolgebedingungen, wenn eine geeignete Approximation des 1-Maschinen-Problems verwendet wird. Diese Schranke ist zwar schlechter als die anderer bekannter Näherungslösungen, jedoch basieren bessere

Schranken auf der Lösung aufwendiger linearer Programme, während das hier vorgestellte Verfahren effizient implementiert werden kann und daher möglicherweise auch für praktische Anwendungen von Interesse ist.

5.4.1.2. Sequenz-abhängige Rüstzeiten

Parallelmaschinen-Probleme mit Minimierung der gewichteten Gesamtfertigstellungszeit haben im Zusammenhang mit (sequenz-abhängigen) Rüstzeiten bisher nur sehr wenig Beachtung in der Literatur gefunden. Dies unterstreichen auch die umfassenden Übersichtsarbeiten von Allahverdi et al. (1999, 2008) [6, 7]. Darin werden verschiedene Formen von Rüstzeiten, insbesondere antizipatorischer und nicht-antizipatorischer Natur, unterschieden. Im ersten Fall kann der Rüstvorgang einer Maschine zur Bearbeitung eines Jobs durchgeführt werden, bevor dieser tatsächlich zur Bearbeitung ansteht, im zweiten Fall ist dies nicht möglich. Für das Routing-Problem ist die erste Variante zutreffend. Die Rüstzeit eines Fahrzeugs ist darin gleichzusetzen mit einer notwendigen Leerfahrt zwischen zwei aufeinander folgenden Transportaufträgen. Diese kann natürlich durchgeführt werden, bevor der nächste zu bearbeitende Transport überhaupt freigegeben ist.

In Bezug auf das Problem $Pm|sds|\sum w_jC_j$ enthalten die Arbeiten von Allahverdi et al. nur wenige für uns relevante Referenzen: Ghosh (1994) [42] konstruiert mit Hilfe dynamischer Programmierung einen polynomialen Algorithmus zur Lösung des Problems $Pm|sds|\sum C_j$, während Lee und Pinedo (1997) [77] eine dreistufige Heuristik zur Behandlung des Problems $Pm|sds|\sum w_jT_j$ entwickeln. Innerhalb der ersten beiden Stufen wird dabei mit Hilfe einer Prioritätsregel eine geeignete Startlösung ermittelt, welche anschließend mit Hilfe von Simulated Annealing wenn möglich weiter verbessert werden kann. Ferner entwerfen Chen und Powell (2003) [34] ein Branch & Bound Verfahren zur Lösung von Problemen des Typs $Pm|sds|\sum w_jC_j$ oder $Pm|sds|\sum w_jU_j$ und mittlerer Größenordnung (mit maximal 40 Jobs und vier Bearbeitungsstufen).

Vor dem Hintergrund der zu behandelnden Anwendungsaufgabe weisen die Ansätze der drei vorangehend genannten Quellen sämtlich einen zu hohen Rechenaufwand auf. Demgegenüber erscheinen in diesem Zusammenhang die Strategien, welche Weng et al. (2001) [152] vorstellen, sehr interessant. In dieser Arbeit werden insgesamt sieben verschiedene Heuristiken zur Lösung des Problems $Rm|sds|\sum w_jC_j$ präsentiert. Sechs davon basieren auf einer List Scheduling Strategie, wobei die Ausgangssequenzen der Jobs durch verschiedene Verallgemeinerungen der WSPT-Regel bestimmt werden. Keine dieser Varianten bezieht allerdings bei der Erstellung der initialen Liste die anfallenden Rüstzeiten mit ein, sondern es werden nur die (durchschnittlichen) Bearbeitungszeiten berücksichtigt. Die Rüstzeiten spielen dann erst beim eigentlichen List Scheduling eine Rolle. Dies ist wohl auch der Grund, warum diese Ansätze zumindest für den Fall unabhängiger Maschinen unterschiedlicher Geschwindigkeit (*Rm*) keine zufriedenstellenden Ergebnisse liefern. Die berechneten Lösungen werden schlechter, je größer die Rüstzeiten im Verhält-

nis zu den Bearbeitungszeiten sind. Dies lässt natürlich keinen unmittelbaren Rückschluss auf die Lösungsgüte für den Fall identisch paralleler Maschinen zu, dennoch handelt es sich um ein Indiz, dass auch diesbezüglich die Vernachlässigung der Rüstzeiten bei der Sequenzierung der Jobs nicht ratsam ist.

Die siebte und vielversprechendste Heuristik aus der Arbeit von Weng et al. (2001) [152] berücksichtigt dagegen die Rüstzeiten in jeder Entscheidung, die zur Verplanung der Jobs getroffen wird. Es handelt sich allerdings nicht um eine List Scheduling Strategie im herkömmlichen Sinn, sondern die Jobs werden in folgender Weise verplant: Ein Job j wird zum Zeitpunkt t auf einer Maschine i gestartet, falls der Ausdruck

$$t + \frac{s_{l_t(i)j} + p_j}{w_j}$$

in diesem Moment minimal wird. Dabei bezeichnet $l_t(i)$ denjenigen Job, der in Bezug auf den Zeitpunkt t zuletzt auf Maschine i bearbeitet wurde, und $s_{l_t(i)j}$ repräsentiert die entsprechende Rüstzeit zwischen $l_t(i)$ und j. Falls j der erste Job auf Maschine i ist, fällt je nach Problemkontext entweder keine oder eine initiale Rüstzeit an. Im Verhältnis zu den sechs WSPT- und List Scheduling-basierten Strategien liefert dieses Verfahren aussichtsreiche Ergebnisse.[13] Aufgrund der extrem kurzen Rechenzeit bildet es daher die Grundlage für die im Folgenden vorgestellte Heuristik zur Lösung des Routing-Problems.

Neben diesen heuristischen Ansätzen wird in der Arbeit von Nessah et al. (2007) [98] ein notwendiges Optimalitätskriterium für eine Lösung des Problems $Pm|r_j, sds|\sum C_j$ hergeleitet. Dieses wird darin zwar noch nicht zur Konstruktion eines Lösungsverfahrens verwendet, ist aber grundsätzlich im Rahmen eines Local Search Verfahrens oder zur Einschränkung des Suchraums bei einem Branch & Bound Verfahren vorstellbar. Zudem ist das Kriterium auch für die Zielfunktion $\sum w_j C_j$ erweiterbar. Für weitere Informationen sei auf Abschnitt 5.4.3 verwiesen.

5.4.2. Heuristische Lösung des Routing-Problems

Die vorangehend präsentierte Übersicht zeigt, dass eine simultane Behandlung von Reihenfolgebedingungen und Rüstzeiten in einer Parallelmaschinen-Umgebung mit Minimierung der Zielfunktion $\sum w_j C_j$ in der Literatur bisher nicht berücksichtigt wurde. Beide Arten von Randbedingungen wurden bislang nur getrennt voneinander behandelt, wobei für Probleme mit Reihenfolgebedingungen vornehmlich Relaxationsansätze als Basis für ein List Scheduling Verfahren verwendet werden und im Fall von sequenz-abhängigen Rüstzeiten erneut auch einfache Prioritätsregeln eine Rolle spielen. Die Motivation für das im weiteren Verlauf vorgestellte Verfahren besteht in dem Versuch, diese Ansätze geeignet zu adaptieren, um beide Arten von Randbedingungen gleichzeitig bearbeiten zu

[13] Die Lösungen der siebten Heuristik werden als Benchmark verwendet, da sie über alle Testinstanzen die besten sind.

können. Wir skizzieren nun zunächst die dazu notwendige Konstruktion eines Reihenfolgegraphen.

5.4.2.1. Konstruktion des Reihenfolgegraphen (Phase 3, 1. Teil)

Aus der in Phase 2 getroffenen Maschinenauswahl resultiert eine eindeutig bestimmte Liste zu bearbeitender Transportaufträge. Jeder dieser Aufträge wird durch einen Knoten im Graphen repräsentiert, genauer: Für jeden Transportauftrag der Gestalt *„Transportiere Job j von Maschine l_1 des Typs i_1 zur Maschine l_2 des Typs i_2“* fügen wir einen Knoten der Form $(j, i_1(l_1) \rightarrow i_2(l_2))$ in den Graphen ein. Zwischen sämtlichen Knoten müssen grundsätzlich zwei Arten von Reihenfolgebeziehungen berücksichtigt werden.

Durch die Maschinenumgebung des Typs Flow-Shop ist die Arbeitsgangfolge für jeden Job bereits vorgegeben. Dieser Restriktion unterliegen natürlich auch die korrespondierenden Transportaufträge. Daraus ergibt sich die erste Art von Kanten, welche wir im Folgenden als *starke Reihenfolgebedingungen* bezeichnen werden. Angenommen, ein Job j wird in einem Transportauftrag k zunächst von einer Maschine der Stufe $i-1$ zu einer der Stufe i transportiert und anschließend von dort in einem Auftrag k' weiter zur nächsten Bearbeitungsstufe $i+1$ befördert. Für die Startzeit des zweiten Auftrag k' muss dann gelten: $s_{k'} \geq C_k + p_{ij}$. Mit anderen Worten, der Auftrag k' kann erst gestartet werden, wenn der Auftrag k abgeschlossen ist und der Job j auf der Maschinenstufe i fertiggestellt wurde. Der zweite Summand korrespondiert damit zu den im Routing-Problem anfallenden Verzögerungszeiten.[14] Für die beschriebene Situation fügen wir eine entsprechende *starke Kante* $k \rightarrow k'$ in den Reihenfolgegraphen ein.

In Phase 1 und 2 des Verfahrens wurde für jede Maschine des Systems eine Jobfolge bestimmt. Aus dieser ergeben sich weitere, sogenannte *schwache Reihenfolgebedingungen* für die Transportaufträge. Um die vorgegebene Jobsequenz für eine Maschine einzuhalten, muss ein Job diese Maschine verlassen oder sogar bereits verlassen haben, sobald der nächste Job dort eintrifft. Wir betrachten eine Maschine l, auf der zunächst Job j und anschließend Job j' bearbeitet werden soll. Angenommen, in einem Transportauftrag k wird der Job j' zur Maschine l transportiert und in einem weiteren Auftrag k' wird der Job j von Maschine l zur nächsten befördert. Für diese beiden Aufträge muss dann die folgende Beziehung gelten: $s'_k \leq C_k$. Falls das gleichzeitige Aufnehmen und Ablegen eines Jobs aufgrund technischer Restriktionen nicht möglich sein sollte, kann man diese Bedingung durch $s'_k \leq C_k - 1$ ersetzen. Mit anderen Worten, ein Job darf erst auf einer Maschine eintreffen, wenn sein Vorgänger diese bereits verlassen hat. Für diese Situation fügen wir dem Graphen eine entsprechende *schwache Kante* hinzu. Die Konstruktion des gesamten Graphen mit schwachen und starken Kanten verdeutlichen wir nun anhand des folgenden Beispiels:

[14] Die zusätzliche Restriktion *delay* des Routing-Problems wird also mit Hilfe des Reihenfolgegraphen abgebildet.

Nr.	Transportauftrag	Vorgänger (stark)	Vorgänger (schwach)
1	$(1,0 \to 1(1))$	-	$(2,1(1) \to 2(3))$
2	$(1,1(1) \to 2(2))$	$(1,0 \to 1(1))$	$(3,2(2) \to -1)$
3	$(1,2 \to -1)$	$(1,1 \to 2)$	-
4	$(2,0 \to 1(1))$	-	$(3,1(1) \to 2(2))$
5	$(2,1(1) \to 2(3))$	$(2,0 \to 1(1))$	-
6	$(2,2(3) \to -1)$	$(2,1(1) \to 2(3))$	-
7	$(3,0 \to 1(1))$	-	-
8	$(3,1(1) \to 2(2))$	$(3,0 \to 1(1))$	-
9	$(3,2(2) \to -1)$	$(3,1(1) \to 2(2))$	-

Tabelle 5.1.: Liste der Transportaufträge für Beispiel 5.4.1

Beispiel 5.4.1 *Wir betrachten drei Jobs und zwei Bearbeitungsstufen mit* $M^{(1)} = 1$ *und* $M^{(2)} = 2$*. Alle Jobs müssen zunächst auf Stufe* 1 *und anschließend auf Stufe* 2 *bearbeitet werden. Folgende Sequenzen für die drei Maschinen seien gemäß der Lösung des Sequencing-Problems gegeben:*

Maschinennummer	*Bearbeitungsstufe*	*Jobfolge*
1	1	3,2,1
2	2	3,1
3	2	2

Da oftmals auch für die Ein- und Ausgabe in bzw. aus dem System eine Transportaufgabe anfällt, fügen wir zusätzlich zu den drei Maschinen noch die beiden Dummy-Maschinen „0“ (Eingabeposition) und „-1“ (Ausgabeposition) jeweils ohne Kapazitätsbeschränkung hinzu. Insgesamt erhält man so die in Tabelle 5.1 aufgelisteten Transportaufträge zwischen den Knoten.

Die erste Zeile der Tabelle enthält die Vorgänger für den Knoten $(1,0 \to 1(1))$*. Es handelt sich um die erste Operation des ersten Jobs, in der Arbeitsgangfolge des Jobs gibt es also keine Vorgänger und dementsprechend auch keinen Vorgänger mit starker Reihenfolgebeziehung. In der Jobfolge für Maschine* 1 *steht Job* 1 *an letzter Stelle mit Job* 2 *als Vorgänger. Um diese Situation zu modellieren, fügen wir die schwache Kante* $(2,1(1) \to 2(3)) \to (1,0 \to 1(1))$ *in den Graphen ein. Für die übrigen Knoten verfahren wir auf dieselbe Weise und erhalten unter Verwendung der in Tabelle 5.1 angegebenen Nummerierung der Aufträge den in Abbildung 5.5 dargestellten Reihenfolgegraphen.*

In dem so entstehenden Graphen hat jeder Knoten maximal einen starken und einen schwachen Vorgänger und genauso auch maximal einen starken und einen schwachen Nachfolger. Insgesamt besitzt also jeder Knoten maximal zwei Vorgänger und Nachfolger.

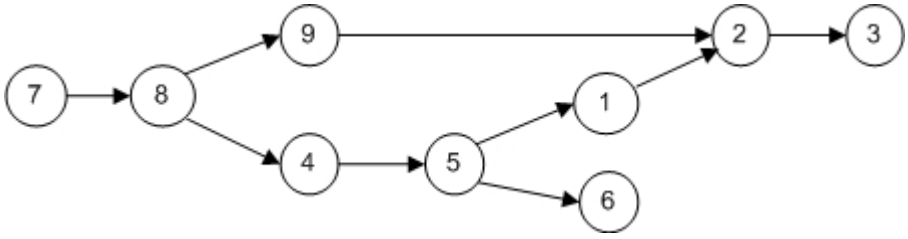

Abbildung 5.5.: Reihenfolgegraph für Beispiel 5.4.1

Darüber hinaus weist dieser Graph jedoch keine spezielle Struktur auf, welche die Lösung des Problems möglicherweise vereinfacht, wie z. B. die eines Baumes der Gestalt In-Tree oder Out-Tree. Das nachfolgend vorgestellte heuristische Lösungsverfahren bezieht sich daher auf allgemeine Reihenfolgerestriktionen. Der Aufwand zur Generierung des Reihenfolgegraphen beträgt im schlechtesten Fall $O(n\sum_i M^{(i)})$, da zur Kantengenerierung höchstens alle $\sum_i M^{(i)}$ Joblisten der maximalen Länge n durchlaufen werden müssen.

5.4.2.2. Heuristische Routenplanung (Phase 3, 2. Teil)

Zur Lösung des Routing-Problems greifen wir erneut auf eine List Scheduling Strategie zurück. Dabei orientieren wir uns an dem von Weng et al. (2001) [152] vorgestellten Ansatz. Wie in Abschnitt 5.4.1.2 bereits diskutiert, kommen die Autoren zu dem Schluss, dass eine Heuristik, in der nicht eine a priori erstellte Jobsequenz konsequent abgearbeitet wird, sondern in welcher die Jobs dynamisch priorisiert werden, für das Problem mit Rüstzeiten bessere Ergebnisse liefert. Unter zusätzlicher Berücksichtigung von Reihenfolgerestriktionen ist eine solche Vorgehensweise erst recht sinnvoll, da nicht sämtliche Jobs von Beginn an bearbeitet werden können. Ein initiale Jobliste, die auf einer vorderen Position einen Job enthält, welcher im Reihenfolgegraphen viele Vorgänger besitzt, liefert möglicherweise einen ungünstigen Schedule.

In dem nun präsentierten Verfahren werden daher in jedem Schritt nur für die aktuell verplanbaren Jobs, d. h. nur solche, deren Vorgänger sämtlich eingeplant sind, Prioritätsindizes berechnet. Diese Menge bezeichnen wir im weiteren Verlauf mit L. Darunter wählen wir denjenigen Job mit dem höchsten entsprechenden Wert aus, verplanen diesen geeignet und aktualisieren L. Zu Beginn enthält L natürlich genau diejenigen Transportaufträge, welche im Reihenfolgegraphen keine Vorgänger besitzen.

Eine Iteration des Verfahrens verläuft im Wesentlichen wie folgt: Im ersten Schritt bestimmen wir Freigabezeiten für die Transportaufträge in L, welche aus der bisher berechneten Teillösung resultieren und somit den Reihenfolgebedingungen genügen. Anschließend ermitteln wir im zweiten Schritt für jede vorhandene Maschine (bzw. für jedes vorhandene Transportmittel) r und jeden Transportauftrag $k \in L$ den frühestmöglichen Zeitpunkt $f_k(r)$, zu welchem r mit der Bearbeitung von k beginnen könnte. Für jedes mögliche Paar (k,r) aus verfügbarem Auftrag in L und vorhandenem Transportmittel werden

im dritten Schritt Prioritätsindizes κ_{kr} berechnet (siehe unten). In Abhängigkeit der zuvor berechneten Werte $f_k(r)$, verwenden wir dabei (analog zu Weng et al. (2001) [152]) erneut eine modifizierte Variante der WSPT-Regel, mit welcher auch Rüstzeiten berücksichtigt werden können. Anschließend wählen wir dasjenige Paar (k,r) mit kleinstem Index κ_{kr} aus und verplanen das Fahrzeug r zur Durchführung des Transportauftrags k.

Im Folgenden formalisieren wir die einzelnen Schritte des Verfahrens. Dazu verwenden wir die nachfolgend aufgelisteten zusätzlichen Notationen:

- V - Menge der Transportaufträge (bzw. Menge der Jobs); jeder Auftrag $k \in V$ wird mindestens durch die drei nachfolgenden Parameter charakterisiert:
  - $j(k)$ - Job, dem der Transportauftrag k zugeordnet werden kann
  - $o(k)$ - Startposition des Auftrags k
  - $d(k)$ - Zielposition des Auftrags k
- R - Menge der Transportmittel
- A - Kantenmenge des Reihenfolgegraphen; insbesondere kann man wie folgt die Menge der direkten Vorgänger ($Pred(k)$) und Nachfolger ($Succ(k)$) eines Knoten k im Graphen beschreiben:
  - $Pred(k) := \{x \in V | (x,k) \in A\}$
  - $Succ(k) := \{x \in V | (k,x) \in A\}$

  Die Kantenmenge A ist dabei die disjunkte Vereinigung der beiden Teilmengen A_1 (starke Kanten) und A_2 (schwache Kanten)
- $D = (V,A)$ - Gerichteter Reihenfolgegraph
- $L \subseteq V$ - Menge der verplanbaren Jobs (d. h. L enthält während dieser Phase des Verfahrens immer die Jobs $k \in V$, deren Vorgänger bereits abgearbeitet sind)
- $\bar{r}_k$ - Freigabezeit des Auftrags $k \in V$ in Abhängigkeit der Reihenfolgebedingungen; entsprechende Werte werden beim Hinzufügen eines Jobs in die Menge L berechnet
- $f_k(r)$ - frühestmögliche Startzeit des Auftrags $k \in V$, falls Fahrzeug $r \in R$ für den Transport k ausgewählt wird
- s_k - Startzeit des Auftrags $k \in V$; hier ist der Zeitpunkt gemeint, zu dem ein ausgewähltes Transportmittel mit der Anfahrt zum Startort $o(k)$ beginnt[15]
- $\bar{s}_k$ - tatsächliche Startzeit des Auftrags $k \in V$, also der Zeitpunkt, zu dem der echte Transport des Auftrags beginnt

[15] Die hier betrachteten Leerfahrten bzw. Rüstzeiten sind also antizipatorisch, da sie schon vor Beginn des eigentlichen Auftrags durchgeführt werden können.

- C_k - Fertigstellungszeit des Auftrags $k \in V$, also der Zeitpunkt, zu dem der zu Auftrag k gehörige Job $j(k)$ sein Ziel $d(k)$ erreicht
- d_k - Verzögerungszeit, die nach dem Auftrag $k \in V$ entsteht, wenn im Anschluss an k ein Auftrag k' ausgeführt wird, welcher zum selben Job gehört, d. h. $j(k) = j(k')$; falls ein Auftrag k als Ziel eine Maschine des Typs i hat, muss vor dem Transport des Jobs $j(k)$ zu einer Maschine des Typs $i+1$ eine Verzögerung von $d_k = p_{ij(k)}$ abgewartet werden (abgebildet durch starke Kanten)
- p_k - Bearbeitungszeit des Auftrags $k \in V$ bzw. notwendige Transportzeit zwischen Start- und Zielort des Auftrags

5.4.2.3. Bestimmung der Freigabezeiten

Für einen verfügbaren Auftrag $k \in L$ ergibt sich die Freigabezeit $\bar{r}_k$ aus der partiellen Lösung, je nachdem welche Gestalt $Pred(k)$ aufweist. Falls k keine Vorgänger besitzt, entspricht $\bar{r}_k$ der Freigabezeit des zugehörigen Jobs ($r_{j(k)}$). Falls k einen starken Vorgänger k_1 hat (also $(k_1,k) \in A_1$), welcher also denselben zugehörigen Job besitzt ($j(k) = j(k_1)$), so muss k_1 vollständig abgearbeitet sein und die Verzögerungszeit d_{k_1} am Zielort muss verstrichen sein, bevor Auftrag k gestartet werden kann (d. h. $\bar{s}_k \geq C_{k_1} + d_{k_1}$).

Falls k im Graphen einen schwachen Vorgänger k_2 hat, also $(k_2,k) \in A_2$ und damit $j(k_2) \neq j(k)$, so muss die entsprechende schwache Reihenfolgebeziehung berücksichtigt werden: Ein Auftrag k darf erst fertiggestellt werden, nachdem die Bearbeitung des Vorgängers begonnen hat (d. h. $\bar{s}_{k_2} \leq \bar{s}_k + p_k$). Für einen verfügbaren Auftrag $k \in L$ wird die Freigabezeit demnach durch den folgenden Ausdruck beschrieben:

$$\bar{r}_k := \max\{t_0, r_{j(k)}, \max_{k_1 \in Pred(k),(k_1,k)\in A_1}(C_{k_1} + d_{k_1}), \max_{k_2 \in Pred(k),(k_2,k)\in A_2}(\bar{s}_{k_2} - p_k)\}$$

Hierbei bezeichnet t_0 den Beginn der aktuellen Planungsperiode. Dieser Parameter ist nur für die Dynamisierung des Verfahrens von Bedeutung (vgl. Abschnitt 5.5).

5.4.2.4. Bestimmung der frühestmöglichen Startzeiten

Die frühestmögliche Startzeit $f_k(r)$ eines Auftrags k unter der Verwendung eines Fahrzeugs r ergibt sich als derjenige Zeitpunkt, zu dem das Transportmittel r den Auftrag k frühestmöglich an seinem Startort $o(k)$ aufnehmen und mit dem Transport beginnen kann (d. h. nach erfolgter Routenplanung gilt $f_k(r) \leq \bar{s}_k$). Der entsprechende Wert kann dabei natürlich nie die Freigabezeit $\bar{r}_k$ des Auftrags unterschreiten. Wenn ein Fahrzeug zum Zeitpunkt der Freigabe eines Auftrags gerade beschäftigt ist, muss natürlich außerdem gewartet werden, bis dieses wieder verfügbar ist. Anschließend muss es vom Zielort des zuletzt fertiggestellten Auftrags zum Startort des Auftrags k fahren und erst dann kann mit dem eigentlichen Transport begonnen werden. In diesem Fall ergibt sich die frühestmögliche Startzeit also aus der Summe des Zeitpunktes a_r, zu dem das Fahrzeug frei wird,

und der anfallenden Leerfahrtzeit (bzw. Rüstzeit) $s_{p_r(a_r)o(k)}$ zum Startort $o(k)$ des Auftrags. Dabei bezeichne $p_r(a_r)$ die Position des Fahrzeugs r zum Zeitpunkt a_r. Ist r zum Freigabezeitpunkt des Auftrags k bereits frei, so kann gegebenenfalls schon früher mit der Anfahrt zum Startort $o(k)$ des Auftrags k begonnen werden. Bestenfalls erreicht das Fahrzeug r diesen schon zum Zeitpunkt $\bar{r}_k$. Die Werte $f_k(r)$ ergeben sich demnach für alle $k \in L$ und $r \in R$ wie folgt:

$$f_k(r) := \max(a_r, \bar{r}_k) - \min(\max(\bar{r}_k - a_r, 0), s_{p_r(a_r)o(k)}) + s_{p_r(a_r)o(k)}$$

Bei der Berechnung von $f_k(r)$ kann darüber hinaus die Situation auftreten, dass das Transportmittel r den Auftrag k schon früher bearbeiten kann, als andere bereits verplante Aufträge. D. h. es ist möglich, dass eine (geeignete) Leerlaufphase eines Transportmittels unter Berücksichtigung der Reihenfolgevorgaben zur vorgezogenen Durchführung eines Auftrags genutzt werden kann. Sofern ein Leerlaufintervall ausreichend lang ist, ergibt sich der Wert von $f_k(r)$ in diesem Fall aber ebenfalls wie oben beschrieben. Lediglich die algorithmische Umsetzung dieser Situation erfordert aufgrund der Suche nach geeigneten Leerlaufintervallen verschiedene zusätzliche Notationen und ist daher nur in Anhang A detailliert beschrieben.

5.4.2.5. Bestimmung der Prioritätsindizes

Für sämtliche Paare (k,r) verfügbarer Aufträge $k \in L$ und vorhandener Fahrzeuge $r \in R$ bestimmen wir nun basierend auf den frühestmöglichen Startzeiten $f_k(r)$ den Prioritätsindex

$$\kappa_{kr} := f_k(r) + \frac{s_{p_r(a_r)o(k)} + p_k}{w_{j(k)}}.$$

Insgesamt ergibt sich so der folgende Algorithmus zur heuristischen Lösung des Routing-Problems. Die Beschreibung ist an dieser Stelle zur besseren Übersicht informell gehalten. Für eine detaillierte formale Beschreibung sämtlicher Schritte sei auf Anhang A verwiesen.

Algorithmus 5.4.2 ***(Heuristische Routenplanung (statisch))***

Generiere zunächst basierend auf der Lösung des Sequencing-Problems den Reihenfolgegraphen D und setze

$$L = \{k \in V | Pred(k) = \emptyset\}.$$

Verfahre nun wie folgt:

Solange $L \neq \emptyset$*:*

1. *Für alle* $k \in L$ *und* $r \in R$*:*

   Bestimme $f_k(r)$.

Bestimme folgenden Prioritätsindex:

$$\kappa_{kr} := f_k(r) + \frac{s_{pr(a_r)o(k)} + p_k}{w_{j(k)}}.$$

2. *Wähle das Paar (k,r), für welches der Koeffizient κ_{kr} minimal wird, und verplane das Fahrzeug r zur Durchführung des Transports k.*
3. *Aktualisiere L.*

D. h. nach der Berechnung sämtlicher Koeffizienten in Schritt 1 wählen wir das Paar (k,r) aus Transportauftrag und Transportmittel, für welches der Index κ_{kr} minimal wird und verplanen r zur Durchführung des Auftrags k. Basierend auf dieser Entscheidung werden nun gegebenenfalls neue Transportaufträge (Nachfolger von k) zur Menge der verfügbaren Aufträge L hinzugefügt und das Verfahren beginnt von vorne. Der Aufwand dieses Algorithmus' beträgt im schlechtesten Fall $O(n^2 \cdot m^2 \cdot n_R)$, da maximal $n(m+1) = nm+n$ Transportaufträge generiert und höchstens entsprechend viele Iterationen durchgeführt werden. In jeder Iteration werden dann maximal $n(m+1)n_R$ der Werte $f_k(r)$ bzw. κ_{kr} berechnet. Ein vollständiges Beispiel zur Arbeitsweise der heuristischen Routenplanung ist sehr umfangreich und kann Anhang B entnommen werden.

5.4.3. Ergänzende Bemerkungen

Wie in Abschnitt 5.4.1.2 bereits angedeutet, können bei der Lösung von Scheduling-Problemen auch Überlegungen zur notwendigen Struktur einer optimalen Lösung eine Rolle spielen. Ein entsprechendes Dominanzkriterium für eine Lösung des Problems $Pm||\sum w_jC_j$ präsentieren Azizoglu und Kirca (1999) [10]. Dieses wird anschließend im Rahmen eines Branch & Bound Verfahrens genutzt, um so im Branching-Schritt die Anzahl möglicher Schedules zu begrenzen. Die Autoren erweitern diesen Ansatz ferner zur Verplanung paralleler Maschinen unterschiedlicher Geschwindigkeit ($Qm||\sum w_jC_j$).

In der Arbeit von Nessah et al. (2007) [98] wird ein notwendiges Optimalitätskriterium für eine Lösung des Problems $Pm|r_j,sds|\sum C_j$ bewiesen. Die Aussagen lassen sich für den gewichteten Fall $Pm|sds,r_j|\sum w_jC_j$ verallgemeinern. Dazu verwenden wir in Anlehnung an die genannte Arbeit folgende Notationen: Angenommen, zwei Jobs i und j werden nacheinander auf einer Maschine bearbeitet, die ab dem Zeitpunkt $t = C_k$ zur Verfügung steht, wobei k der zuletzt auf dieser Maschine bearbeitete Job ist. Nun bezeichne C^w_{ij} die gewichtete Summe der Fertigstellungszeiten der Jobs i und j, wenn zuerst i und direkt im Anschluss j bearbeitet wird. Wir untersuchen nun, unter welcher Bedingung $C^w_{ij} \leq C^w_{ji}$ gilt, und erhalten in diesem Zusammenhang die folgende Aussage:

Satz 5.4.3 *Es seien i und j zwei Jobs, die nacheinander auf derselben Maschine bearbeitet werden. Diese steht ab dem Zeitpunkt $t = C_k$ zur Verfügung, k ist der zuletzt darauf bearbeitete Job. Dann gilt $C^w_{ij} \leq C^w_{ji}$ genau dann, wenn $f^w_{ij}(k) \leq f^w_{ji}(k)$, wobei*

$$f_{ij}^{w}(k) := \max \left\{ \begin{array}{l} (w_i + w_j)\max(C_k + s_{ki}, r_i) + w_j(p_i + s_{ij}), \\ w_i \max(C_k + s_{ki}, r_i) + w_j \max(C_k + s_{ki}, r_j) \end{array} \right\}$$

Beweis:

Es gilt:

$$\begin{aligned} C_{ij}^{w} &= w_i(\max(C_k + s_{ki}, r_i) + p_i) + w_j(\max(\max(C_k + s_{ki}, r_i) + p_i + s_{ij}, r_j) + p_j) \\ &= w_i \max(C_k + s_{ki}, r_i) + w_j \max(\max(C_k + s_{ki}, r_i) + p_i + s_{ij}, r_j) + w_i p_i + w_j p_j \\ &= w_i \max(C_k + s_{ki}, r_i) + w_j \max(\max(C_k + s_{ki}, r_i) + p_i + s_{ij}, \max(C_k + s_{ki}, r_j)) \\ &\quad + w_i p_i + w_j p_j \\ &= \max \left\{ \begin{array}{l} (w_i + w_j)\max(C_k + s_{ki}, r_i) + w_j(p_i + s_{ij}), \\ w_i \max(C_k + s_{ki}, r_i) + w_j \max(C_k + s_{ki}, r_j) \end{array} \right\} + w_i p_i + w_j p_j \\ &= f_{ij}^{w}(k) + w_i p_i + w_j p_j \end{aligned}$$

Analog erhalten wir:

$$C_{ji}^{w} = f_{ji}^{w}(k) + w_j p_j + w_i p_i$$

und damit gilt $C_{ij}^{w} \leq C_{ji}^{w}$ genau dann, wenn $f_{ij}^{w}(k) \leq f_{ji}^{w}(k)$. ∎

Bemerkung 5.4.4 *Die analoge Aussage gilt auch, wenn wir „$\leq$“ durch „$<$“ ersetzen.*

Nessah et. al. (2007) [98] definieren in ihrer Arbeit den Begriff eines *FS-aktiven* Schedules. Wir verwenden im Folgenden nun eine analoge Begriffsbildung für das gewichtete Problem.

Definition 5.4.5 *Es seien S ein aktiver Schedule sowie x und y zwei beliebige Jobs, die direkt nacheinander auf derselben Maschine bearbeitet werden. Zuvor werde auf dieser Maschine der Job k bearbeitet. Falls für jedes beliebige Paar solcher Jobs x und y mindestens eine der nachfolgenden Bedingungen gilt, nennen wir den Schedule S* FSw-aktiv.

1. $\max(C_k + s_{kx}, r_x) < \max(C_k + s_{ky}, r_y) + s_{yx}$
2. $f_{xy}^{w}(k) \leq f_{yx}^{w}(k)$

Basierend auf dem Begriff der FSw-Aktivität kann man nun folgendes notwendige Kriterium für die Optimalität eines Schedules für das Problem $Pm|sds, r_j|\sum w_j C_j$ nachweisen:

Satz 5.4.6 *Falls die sequenz-abhängigen Rüstzeiten die Dreiecksungleichung erfüllen, ist jeder optimale Schedule für das Problem $Pm|sds, r_j|\sum w_j C_j$ FSw-aktiv.*

Beweis: (analog zu Nessah et. al., 2007, [98])

Angenommen, es gäbe einen optimalen Schedule für das Problem $Pm|sds,r_j|\sum w_jC_j$, der nicht aktiv ist. Dann gäbe es in diesem Schedule einen Job, der früher gestartet werden könnte, ohne die anderen Jobs zu verzögern. Im betrachteten Scheduling-Problem, würde dieser Job dann jedoch auch früher fertiggestellt werden und das ist ein Widerspruch zur vorausgesetzten Optimalität.

Angenommen, es gäbe einen optimalen Schedule S, der nicht FSw-aktiv ist. In diesem Schedule gibt es dann mindestens zwei aufeinander folgende Jobs x und y auf derselben Maschine i, für welche keine der beiden Bedingungen erfüllt ist, d. h. es gilt:

(1) $max(C_k + s_{kx}, r_x) \geq \max(C_k + s_{ky}, r_y) + s_{yx}$ und

(2) $f^w_{xy}(k) > f^w_{yx}(k)$.

Wir bezeichnen im Folgenden nun für eine Teilmenge $A \subseteq J$ von Jobs die gewichtete Fertigstellungszeit dieser Jobs mit $WCT(A) := \sum_{j\in A} w_jC_j$. Des Weiteren sei:

- J^S_i - die geordnete Menge der Jobs auf Maschine i im Schedule S
- A^S_i - die geordnete Menge der Jobs auf Maschine i im Schedule S, die vor x bearbeitet werden
- B^S_i - die geordnete Menge der Jobs auf Maschine i im Schedule S, die nach y bearbeitet werden, z sei der erste Job in B^S_i

Aus Satz 5.4.3 wissen wir, dass wegen (2)

$$WCT(A^S_i) + C^w_{xy} > WCT(A^S_i) + C^w_{yx} \qquad (*)$$

gilt. Wir betrachten nun den Schedule S', welcher aus S durch Vertauschung von x und y entsteht. z ist derjenige Job, welcher im Schedule S im Anschluss an x und y bearbeitet wird. Des Weiteren bezeichne $t_S(z)$ die frühestmögliche Startzeit des Jobs z im Schedule S, $t_{S'}(z)$ sei analog definiert. Da es sich bei $\sum w_jC_j$ um eine reguläre Zielfunktion handelt und wegen $(*)$ gilt nun $WCT(J^{S'}_i) \leq WCT(J^S_i)$, falls $t_{S'}(z) \leq t_S(z)$. Wir zeigen nun also $t_{S'}(z) \leq t_S(z)$. Da wegen (1) insbesondere $r_y \leq \max(C_k + s_{kx}, r_x)$ richtig ist, gilt:

$$\begin{aligned} t_S(z) &= \max(r_z, \max(r_y, \max(C_k + s_{kx}, r_x) + p_x + s_{xy}) + p_y + s_{yz}) \\ &= \max(r_z, \underbrace{\max(C_k + s_{kx}, r_x) + p_x + p_y + s_{xy} + s_{yz})}_{a} \end{aligned}$$

sowie

$$t_{S'}(z) = \max(r_z, \underbrace{\max(r_x, \max(C_k + s_{ky}, r_y) + p_y + s_{yx}) + p_x + s_{xz})}_{b}.$$

Es ist also noch $b \leq a$ zu zeigen.

1. Fall: Falls $\max(r_x, \max(C_k + s_{ky}, r_y) + p_y + s_{yx}) = r_x$, so ist $b = r_x + p_x + s_{xz}$.
Außerdem gilt:

$$\begin{aligned} b - a &= r_x + p_x + s_{xz} - \max(C_k + s_{kx}, r_x) - p_x - p_y - s_{xy} - s_{yz} \\ &= \underbrace{r_x - \max(C_k + s_{kx}, r_x)}_{\leq 0} + \underbrace{(s_{xz} - s_{xy} - s_{yz})}_{\leq 0} - p_y \leq 0, \end{aligned}$$

da die betrachteten Rüstzeiten nach Voraussetzung die Dreiecksungleichung erfüllen.

2. Fall: $b = \max(C_k + s_{ky}, r_y) + p_y + s_{yx} + p_x + s_{xz}$.
Dann ist:

$$\begin{aligned} b - a &= \max(C_k + s_{ky}, r_y) + p_y + s_{yx} + p_x + s_{xz} \\ &\quad -(\max(C_k + s_{kx}, r_x) + p_x + p_y + s_{xy} + s_{yz}) \\ &= \underbrace{\max(C_k + s_{ky}, r_y) + s_{yx} - \max(C_+ s_{kx}, r_x)}_{\leq 0} + \underbrace{(s_{xz} - s_{xy} - s_{yz})}_{\leq 0} \leq 0, \end{aligned}$$

da für die Rüstzeiten die Dreiecksungleichung erfüllt ist und wegen (1).

Insgesamt folgt $t_{S'}(z) \leq t_S(z)$. ∎

Für den Spezialfall $r_j = 0$, für alle Jobs j, ergibt sich insbesondere:

Korollar 5.4.7 *Für das Problem $Pm|sds|\sum w_j C_j$ gilt $C^w_{ij} \leq C^w_{ji}$ genau dann, wenn*

$$\frac{p_i + s_{ij}}{w_i} \leq \frac{p_j + s_{ji}}{w_j}.$$

Die Aussagen 5.4.6 und 5.4.7 könnten zur schrittweisen Verbesserung einer Startlösung verwendet werden, insbesondere einer Lösung, die mit Hilfe von Phase 3 unseres Verfahrens generiert wurde. Zudem ist deren Einsatz im Rahmen eines Branch & Bound Verfahrens analog zu der eingangs genannten Methode von Azizoglu und Kirca (1999) [10] denkbar. Eine ähnliche Argumentation ist auch bei Jouglet et. al. (2004) [61] zu finden. Die Autoren entwickeln in der genannten Arbeit ein Branch & Bound Verfahren zur Lösung des Problems $1|r_j|\sum w_j T_j$.

5.5. Dynamisierung des Verfahrens

Grundsätzlich werden alle drei Phasen des Verfahrens immer dann durchgeführt, sobald ein neuer Job eintrifft. Genauso kann reoptimiert werden, falls ein anderes unvorhergesehenes Ereignis, wie der Ausfall einer Maschine, eintritt. Natürlich kann nicht bei jeder Reoptimierung die gesamte bisher berechnete Lösung verworfen werden. Alle Transportaufträge oder Jobs, die sich zu dem jeweiligen Zeitpunkt bereits in Bearbeitung befinden, werden nicht neu verplant. Dies gilt ebenfalls für solche, deren Terminplan aufgrund der Reihenfolgebedingungen nicht mehr verändert werden kann, ohne die Zulässigkeit des Schedules zu verletzen. In Bezug auf die drei Phasen des Verfahrens bedeutet dies im Einzelnen:

Phase 1: Bei der Sequenzierung der Jobs werden im Rahmen einer Reoptimierung nur solche Jobs berücksichtigt, die sich noch gar nicht in Bearbeitung befinden. D. h. auch spätere Operationen eines Jobs werden bei der Lösung des Sequencing-Problems nicht neu sortiert, auch wenn sie zum Zeitpunkt der Reoptimierung noch nicht in Bearbeitung sind. Lediglich im Zuge des List Schedulings werden solche Operationen gegebenenfalls neu terminiert.

Phase 2: Im Rahmen der Maschinenauswahl werden sämtliche Operationen, die noch nicht in Bearbeitung sind, neu zugeordnet. Falls sich die ersten Operationen eines Jobs bereits auf einer Maschine befinden oder gerade ein zugehöriger Transport durchgeführt wird, kann die Maschinenauswahl für spätere Operationen desselben Jobs dennoch reoptimiert werden, sofern die Reihenfolgebedingungen dies gestatten.

Phase 3: Ähnlich wie bei der Maschinenauswahl werden auch hier (fast) alle Transportaufträge, die zum Zeitpunkt einer Reoptimierung noch nicht begonnen haben, bei der erneuten Lösung des Routing-Problems berücksichtigt. Es ist dabei aber die Situation möglich, dass in diesem Zusammenhang ein Auftrag neu verplant werden müsste, für den ein Nachfolger im Reihenfolgegraph jedoch nicht neu terminiert werden darf. Um dies zu verhindern, dürfen sämtliche Vorgänger eines solchen „blockierten“ Transports bei der Reoptimierung ebenfalls nicht berücksichtigt werden.

Zur Algorithmisierung dieser Vorgehensweise arbeiten wir mit einem binären Parameter $b(k)$ bzw. $b(i,j)$, der den Wert 1 annimmt, falls ein Transportauftrag k oder eine Operation (i,j) bei der Reoptimierung nicht berücksichtigt werden darf, also blockiert ist, und 0 sonst. In Abhängigkeit dieser Parameter können sämtliche der oben beschriebenen Methoden für den dynamischen Fall formalisiert werden (vgl. Anhang A). Eine detaillierte und formale Beschreibung sämtlicher Arbeitsschritte sowie insbesondere die Synchronisation der drei Phasen des dynamischen Verfahrens kann ebenfalls Anhang A entnommen werden. Ein entsprechendes ausführliches Beispiel ist in Anhang B zu finden.

6. Empirische Performance-Analyse

In diesem Kapitel wird das vorangehend vorgestellte Verfahren zunächst mit Hilfe zufällig generierter Testdaten ausgewertet. Die Analyse der Performance für das zugrundeliegende Anwendungsproblem erfolgt in Kapitel 7. Dennoch orientieren wir uns bei der Erzeugung der verschiedenen Szenarien bereits teilweise an den Rahmenbedingungen des Prozesslabors. So spielt die Größenordnung der darin auftretenden Transportzeiten im Vergleich zu den Bearbeitungszeiten der Proben auf den Maschinen nur eine untergeordnete Rolle. Mit anderen Worten, es gilt $s_{jk} << p_{ij}$ und aus diesem Grund ist die Qualität der ersten Phase des Verfahrens maßgebend für die Performance des gesamten Verfahrens. Diesen Aspekt berücksichtigen wir bei der Wahl der Intervalle zur Generierung der einzelnen Problemparameter (vgl. Abschnitt 6.1).

Neben der empirischen Analyse des gesamten Verfahrens führen wir daher ferner eine separate Evaluation für Phase 1 durch. Dabei untersuchen wir sowohl die Situation mit unbegrenztem Zwischenspeicher als auch mit No-Wait-Constraints. Als Benchmark für die Performance verwenden wir in allen Fällen die LP-Relaxationen der in Kapitel 4 vorgestellten zeit-indizierten Formulierungen des Problems (siehe Abschnitt 6.2). Unter zusätzlicher Berücksichtigung von Transportrestriktionen wird dabei die in Abschnitt 4.3.2 vorgeschlagene Modifikation der Modelle eingesetzt.

6.1. Datengenerierung

In der Literatur gibt es im Zusammenhang mit Hybrid Flow-Shop-Problemen keine standardisierte Menge von Benchmark-Instanzen, nicht einmal für die am häufigsten untersuchte Grundform des Problems $(FHm, (PM^{(k)})_{k=1}^{m}||C_{\max})$. Auch eine einheitliche Vorgehensweise zur randomisierten Generierung von Testdaten ist nicht erkennbar (vgl. Ruiz & Vázquez-Rodríguez, 2010, [118]). Aus diesen Gründen sind empirische Analysen in der Literatur zum Teil nur schwer vergleichbar. Des Weiteren erklärt dies, warum verschiedene Arbeiten bei der Evaluation derselben Heuristik zum Teil gegenteilige Schlüsse ziehen, denn die Charakteristika einer Probleminstanz, d. h. die Anzahl der Jobs und Maschinen sowie die Werte der Problemparameter, können einen nicht zu vernachlässigenden Einfluss auf die Qualität einer heuristischen Lösung besitzen (vgl. Brah, 1996, [24] sowie Brah & Wheeler, 1998, [27]).

In den meisten Fällen werden für alle benötigten Parameter Intervalle festgelegt und die Datensätze diesbezüglich unter Verwendung der Gleichverteilung zufällig generiert. Dabei stellt sich natürlich die Frage, welche Intervalle zu diesem Zweck geeignet sind.

	Datensatz 1	**Datensatz 2**	**Datensatz 3**
p_j	$U[1,99]$	$U[20,40]$	$U[10,50]$
w_j	$U[1,5]$	$U[1,5]$	$U[1,5]$
r_j	$U[1,100]$	$U[1,50]$	$U[1,200]$
$M^{(i)}$	$U[1,4]$	$U[1,4]$	$U[1,4]$

Tabelle 6.1.: Intervalle für Testinstanzen ohne Transporte

In der Literatur sind in diesem Zusammenhang erneut vielfältige Varianten zu finden. Darüber hinaus besteht die Möglichkeit, alternative Verteilungen zur Gewinnung der Daten zu verwenden. So werden insbesondere bei der Analyse dynamischer Verfahren die Freigabezeiten der Jobs gelegentlich unter Verwendung einer Exponentialverteilung erzeugt (vgl. beispielsweise Lee, 2009, [76]). Savelsbergh et al. (2005) [121] vergleichen daher verschiedene Verteilungen zur Konstruktion von Testdaten, kommen aber zu dem Schluss, dass die jeweilige Wahl sich nicht messbar auf die Performance der untersuchten Verfahren auswirkt. Wir beschränken uns daher im Folgenden auf die gleichverteilte Generierung sämtlicher Problemparameter. Bei der Wahl der Intervalle orientieren wir uns dabei zum Einen an dem zumindest teilweise auszumachenden Konsens in der Literatur. Zum Anderen konstruieren wir Szenarien, die für das Prozesslabor repräsentativ sind. Darüber hinaus setzen wir in sämtlichen Testrechnungen identisch parallele Maschinen auf den verschiedenen Bearbeitungsstufen voraus.

6.1.1. Szenarien ohne Transporte

Bei der Generierung der Datensätze müssen zunächst die zu berücksichtigenden Dimensionen festgelegt werden. In der Literatur werden zumeist zehn bis höchstens 100 Jobs je Probleminstanz (vgl. u. a. Baptiste et al., 2008, [14]), oftmals sogar nicht mehr als 50 Jobs pro Datensatz untersucht (z. B. bei Brah, 1996, [24]; Brah & Loo, 1999, [26]). Im Fall des Prozesslabors sind eher kleinere Jobzahlen realistisch. Daher beschränken wir uns im Folgenden auf eine Anzahl von $n \in \{10,15,20,30,50\}$. Darüber hinaus werden meistens nicht mehr als zehn Bearbeitungsstufen berücksichtigt (siehe Baptiste et al., 2008, [14]), in vielen Fällen sogar nur fünf (Carlier et al., 2010, [31]). Die Analyse einer Probe im Prozesslabor beinhaltet selten mehr als fünf verschiedene Arbeitsgänge, weshalb wir $m \in \{2,3,5\}$ wählen. Für jede mögliche Kombination der Werte für n und m generieren wir 30 zufällige Probleminstanzen. Dabei werden die Eckdaten für die zugrundeliegenden Problemparameter auf drei verschiedene Weisen gewählt (vgl. Tabelle 6.1) und jeweils zehn der 30 Datensätze beziehen sich auf dasselbe Szenario. Auf diese Weise ergeben sich insgesamt 450 verschiedene Testinstanzen. Für jede untersuchen wir sowohl die Situation mit unbegrenztem Zwischenspeicher als auch mit No-Wait-Constraints, sodass in der empirischen Studie für Phase 1 insgesamt 900 Testläufe durchgeführt werden.

	Datensatz 4	**Datensatz 5**	**Datensatz 6**
p_j	$U[20,40]$	$U[20,40]$	$U[20,60]$
w_j	$U[1,5]$	$U[1,5]$	$U[1,5]$
r_j	$U[1,50]$	$U[1,100]$	$U[1,100]$
$M^{(i)}$	$U[1,4]$	$U[1,4]$	$U[1,4]$
s_{jk}	$U[1,5]$	$U[1,5]$	$U[1,5]$

Tabelle 6.2.: Intervalle für Instanzen mit Transporten

Die in Tabelle 6.1 angegebenen Intervalle orientieren sich an entsprechenden Referenzen in der Literatur sowie den Rahmenbedingungen des Prozesslabors. Im Einzelnen berücksichtigen wir maximal vier parallele Maschinen pro Bearbeitungsstufe. Da die zu untersuchenden Proben im Prozesslabor zudem in wenige Prioritätsklassen gruppiert werden können, wählen wir die Gewichte der Jobs in jeder Testreihe aus dem Bereich $[1,5]$. Bei der Generierung der Bearbeitungszeiten sind viele Möglichkeiten denkbar. Sehr häufig wird dazu in der Literatur das Intervall $[1,99]$ bzw. $[1,100]$ verwendet (Testreihe 1, vgl. u. a. Al-Anzi & Allahverdi, 2002, [5]; Lee, 2009, [76]; Baptiste et al., 2008, [14]; Carlier et al., 2010, [31] und viele mehr). Genauso gebräuchlich ist auch die Verwendung der Intervalle $[1,50]$ oder $[10,50]$ (Testreihe 3, vgl. beispielsweise Jayamohan & Rajendran, 2000, [59]; Kadipasaoglu et al., 1997, [64]). Darüber hinaus verwenden wir zusätzlich das Intervall $[20,40]$ (Testreihe 2), um zu untersuchen, wie gut die Heuristik bei weniger stark schwankenden Bearbeitungszeiten arbeitet (vgl. dazu auch Oğuz et al., 2003, [99]).

Bei der gleichverteilten Generierung der Freigabezeiten gibt es grundsätzlich zwei verschiedene Möglichkeiten: Wählt man die Größe des entsprechenden Intervalls in Abhängigkeit der Problemdimension, um so für jede Jobanzahl eine vergleichbare Auslastung der Bearbeitungsstufen zu erreichen (Baptiste et al., 2008, [14]), stellt sich die Frage, inwiefern sich die Problemgröße in diesem Fall noch auf die Performance eines heuristischen Verfahrens auswirkt. Verschiedene Autoren kommen zu dem Schluss, dass die Anzahl der Jobs und Maschinen sowie die Anzahl paralleler Maschinen auf einer Stufe entscheidenden Einfluss auf die Qualität einer Heuristik besitzen können (z. B. Brah, 1996, [24] oder Brah & Wheeler, 1998, [27]). Daher ist es sinnvoll, diesen Effekt bei einer empirischen Untersuchung nicht durch eine angepasste Wahl der Freigabezeiten abzuschwächen, sondern stattdessen ein festes Intervall zur Generierung dieser Daten für alle betrachteten Problemdimensionen zu wählen. Anstelle verschiedener Intervalle innerhalb einer Testreihe simulieren wir eine unterschiedlich starke Auslastung des Maschinenparks durch die drei verschiedenen Testreihen (vgl. Tabelle 6.1).

6.1.2. Szenarien mit Transporten

Bei der Generierung von Testdatensätzen für das Problem mit Transporten sind ähnliche Überlegungen ausschlaggebend. Auch hier müssen zunächst die Dimension der Instanzen sowie die Intervalle für die zu erzeugenden Parameter festgelegt werden. Darüber hinaus spielt hierbei jedoch eine große Rolle, wie sich Transportzeiten und Bearbeitungszeiten zueinander verhalten, d. h., ob diese von derselben Größenordnung sind oder nicht. Im Fall des Prozesslabors bestimmen die Transporte nur einen marginalen Anteil der Gesamtdurchlaufzeit einer Probe, mit anderen Worten, die entsprechenden Zeiten sind wesentlich geringer als die Bearbeitungszeiten auf den einzelnen Maschinen. Diesen Aspekt berücksichtigen wir nun bei der Konstruktion der Testinstanzen für das Problem mit Transporten. Darüber hinaus stellt sich die Frage, ob die vorhandenen Transportmittel unabhängig voneinander eingesetzt werden dürfen oder ob gegebenenfalls Konflikte entstehen können. In Anlehnung an den Aufbau des Prozesslabors werden wir im Folgenden nur unabhängige Transportmittel untersuchen.

In dieser zweiten empirischen Studie berücksichtigen wir erneut drei verschiedene Szenarien. Referenzen zur Erzeugung geeigneter Datensätze sind in der Literatur nahezu nicht vorhanden. Lediglich Soukhal und Martineau (2005) untersuchen ein ähnliches Problem und betrachten dabei Instanzen mit wenigstens drei und maximal 50 Jobs sowie drei bis zehn Bearbeitungsstufen. Auch in der genannten Arbeit werden die Transportzeiten stark durch die Bearbeitungszeiten dominiert.[1]

Wir wählen die in Tabelle 6.2 angegebenen Parameter zur Generierung der Daten. Da die Lösung der erweiterten Modelle mit Transporten wesentlich zeitintensiver ist, als für das HFS-Problem ohne Transporte, beschränken wir zudem die untersuchte Anzahl von Jobs auf 30 je Datensatz. Des Weiteren betrachten wir lediglich Instanzen mit zwei oder drei Maschinenstufen sowie zwei oder drei Transportmitteln. Für jede mögliche Kombination dieser Parameter generieren wir erneut zehn zufällige Datensätze und erhalten so insgesamt eine Anzahl von 600 verschiedenen Testinstanzen.

6.2. Auswertung der Ergebnisse

Zur Auswertung des heuristischen Verfahrens wurde dieses in C++ implementiert[2] und für die zuvor generierten Datensätze ausgeführt. Darüber hinaus lösen wir die LP-Relaxationen der beiden zeit-indizierten Formulierungen des Problems (vgl. Kapitel 4). Dazu wurden diese in OPL implementiert und unter Verwendung von Ilog OPL Studio 3.7 gelöst. Die entsprechenden Schranken werden im Folgenden mit LB1 und LB2 bezeichnet.

[1] Hier wird $p_{ij} \in [1,100]$ und $s_{jk} \in [1,10]$ gewählt.

[2] Der entsprechende Quellcode ist im Rahmen des eingangs genannten Kooperationsprojektes mit der ThyssenKrupp Steel Europe AG entstanden.

Wir vergleichen die jeweils resultierenden Zielfunktionswerte mit Hilfe des auftretenden relativen Fehlers:

$$\frac{\sum w_j C_j(Heuristik) - \sum w_j C_j(LB)}{\sum w_j C_j(LB)}.$$

Für sämtliche Berechnungen wurde ein PC mit Pentium IV Prozessor (2,8 GHz) und 1 GB RAM verwendet.

Um nicht nur die Performance des dynamischen Verfahrens sondern speziell für Phase 1 auch die Effektivität der verschiedenen Sequenzierungsregeln analysieren zu können, wurde dieses für das Problem ohne Transporte zusätzlich in der statischen Variante durchgeführt. Dazu werden sämtliche Jobs eines Datensatzes gleichzeitig betrachtet und mit Hilfe eines einzigen Durchlaufs der ersten Phase verplant. Ein Vergleich der so erzielten Zielfunktionswerte ermöglicht einen Rückschluss auf die Qualität der verschiedenen Sequenzierungsregeln.

Die Berechnung der zweiten Schranke (LB2) erfordert grundsätzlich einen höheren Zeitaufwand als die der ersten (LB1). Aus Kapitel 4 ist jedoch bekannt, dass LB1 von LB2 dominiert wird (vgl. Satz 4.3.5). Um daher die Berechnung von LB2 zu beschleunigen, ermitteln wir zuerst LB1 und verwenden den entsprechenden Zielfunktionswert als untere Schranke bei der Bestimmung von LB2. Darüber hinaus untersuchen wir, wie stark sich die beiden Schranken in der Praxis tatsächlich unterscheiden. Im Folgenden präsentieren wir nun sämtliche Ergebnisse zunächst für das Problem ohne Transporte (vgl. 6.2.1 und 6.2.2) sowie anschließend unter Hinzunahme dieser Nebenbedingung (vgl. 6.2.3).

6.2.1. HFS mit unbegrenztem Zwischenspeicher

Der durchschnittliche relative Fehler der zehn Instanzen jeder Dimension eines Szenarios ist in den Tabellen 6.3 bis 6.8 dargestellt. In der ersten Spalte ist die jeweils betrachtete Problemgröße angegeben. Die zweite bis sechste Spalte enthalten den mittleren relativen Fehler der einzelnen Sequenzierungsregeln (1) - (2.4) bei einmaliger Verwendung von Phase 1 zur Lösung des statischen Problems $FHm,(PM^{(k)})_{k=1}^{m}|r_j|\sum w_j C_j$. Die durchschnittliche Abweichung von der unteren Schranke bei der Wahl der jeweils besten Sequenz (Hybridverfahren) für das statische Problem zeigt die siebte Spalte (best). Die vorletzte Spalte (dyn) beinhaltet den mittleren relativen Fehler der dynamischen Lösung im Vergleich mit der deterministischen unteren Schranke. Ferner kann die dynamische Lösung natürlich auch als Lösung des statischen Problems interpretiert werden. Bezieht man diese als weitere Alternative in das Hybridverfahren für das statische Problem mit ein, so erhält man diesbezüglich die in der letzten Spalte (best_dyn) angegebene durchschnittliche Abweichung von der unteren Schranke. Die mittleren Rechenzeiten für die statische Version des Problems sind für alle untersuchten Dimensionen in Tabelle 6.9 angegeben.

Dimension	statisch						dynamisch	
$n \times m$	**(1)**	**(2.1)**	**(2.2)**	**(2.3)**	**(2.4)**	**best**	**dyn**	**best_dyn**
10×2	12,96	12,87	11,14	9,54	7,77	6,91	10,47	5,92
10×3	9,30	13,89	11,34	9,78	9,78	7,03	11,73	6,25
10×5	12,69	16,19	9,61	9,12	8,76	7,70	16,78	7,39
15×2	12,42	12,89	9,49	9,09	9,06	7,71	11,92	6,96
15×3	12,13	15,33	10,59	8,94	8,31	7,87	12,88	6,69
15×5	13,25	16,34	10,00	9,82	9,56	9,23	15,29	8,16
20×2	12,70	13,16	9,20	8,55	8,55	7,80	16,26	7,04
20×3	16,94	19,92	18,13	17,17	16,89	14,81	15,41	13,25
20×5	17,73	20,28	14,89	14,38	14,36	11,87	19,10	11,74
30×2	12,08	14,58	12,02	10,58	10,58	9,39	11,98	8,39
30×3	15,00	19,78	15,35	14,89	15,56	10,73	13,02	10,01
30×5	17,13	20,11	14,53	14,29	13,71	11,93	17,75	11,56
50×2	10,18	15,55	14,90	15,15	15,08	9,00	9,88	7,92
50×3	15,90	20,97	20,29	19,53	19,62	13,26	15,56	13,06
50×5	14,35	17,36	25,19	24,53	25,63	12,89	19,77	12,40
Gesamt (∅)	**14,52**	**16,61**	**13,78**	**13,03**	**12,88**	**9,88**	**14,52**	**9,12**

Tabelle 6.3.: HFS mit unbegrenztem Zwischenspeicher vs. LB1 (Datensatz 1)

Dimension	statisch						dynamisch	
$n \times m$	**(1)**	**(2.1)**	**(2.2)**	**(2.3)**	**(2.4)**	**best**	**dyn**	**best_dyn**
10×2	9,26	9,14	7,47	5,97	4,26	3,42	6,87	2,47
10×3	6,87	11,32	8,84	7,30	7,30	4,63	9,09	3,85
10×5	10,67	14,14	7,67	7,19	6,84	5,79	14,70	5,48
15×2	9,45	9,92	6,60	6,22	6,19	4,88	8,96	4,15
15×3	9,45	12,55	7,93	6,32	5,71	5,28	10,13	4,12
15×5	-	-	-	-	-	-	-	-
20×2	10,10	10,53	6,64	6,01	6,01	5,28	13,56	4,54
20×3	13,70	16,60	14,87	13,93	13,65	11,64	12,22	10,12
20×5	-	-	-	-	-	-	-	-
30×2	10,43	12,88	10,35	8,93	8,93	7,77	10,32	6,78
30×3	-	-	-	-	-	-	-	-
30×5	-	-	-	-	-	-	-	-
50×2	-	-	-	-	-	-	-	-
50×3	-	-	-	-	-	-	-	-
50×5	-	-	-	-	-	-	-	-
Gesamt (∅)	**9,99**	**12,13**	**8,80**	**7,73**	**7,36**	**6,08**	**10,73**	**5,19**
Ref_LB1	**12,65**	**14,85**	**11,44**	**10,35**	**9,96**	**8,65**	**13,43**	**7,73**

Tabelle 6.4.: HFS mit unbegrenztem Zwischenspeicher vs. LB2 (Datensatz 1)

Dimension	statisch						dynamisch	
$n \times m$	**(1)**	**(2.1)**	**(2.2)**	**(2.3)**	**(2.4)**	**best**	**dyn**	**best_dyn**
10×2	8,60	10,17	3,46	3,04	3,10	3,04	5,18	2,80
10×3	5,77	5,55	3,49	2,84	2,50	2,47	5,00	1,75
10×5	3,81	6,13	2,02	2,02	2,02	1,87	6,40	1,87
15×2	7,03	7,67	3,06	2,94	2,94	2,78	4,30	2,27
15×3	5,41	5,44	2,89	2,88	2,65	2,65	6,25	2,15
15×5	5,75	5,49	3,38	3,38	3,38	3,38	4,73	2,89
20×2	7,68	7,70	3,75	3,44	3,44	3,25	5,30	2,91
20×3	4,49	5,45	2,01	2,07	2,07	1,89	6,43	1,78
20×5	6,69	6,27	2,33	2,43	2,43	2,33	4,99	2,22
30×2	5,93	5,22	2,65	2,66	2,66	2,64	3,64	1,97
30×3	6,07	5,41	3,72	3,64	3,64	3,25	4,90	2,88
30×5	6,92	5,49	2,48	2,48	2,48	2,48	6,27	2,46
50×2	5,47	4,92	2,65	2,60	2,60	2,54	3,14	2,31
50×3	4,88	5,01	1,97	2,03	2,03	1,97	2,91	1,71
50×5	7,21	8,37	6,82	6,81	6,81	6,25	6,20	5,39
Gesamt (∅)	**6,05**	**6,42**	**3,04**	**2,89**	**2,84**	**2,72**	**5,21**	**2,33**

Tabelle 6.5.: HFS mit unbegrenztem Zwischenspeicher vs. LB1 (Datensatz 2)

Dimension	statisch						dynamisch	
$n \times m$	**(1)**	**(2.1)**	**(2.2)**	**(2.3)**	**(2.4)**	**best**	**dyn**	**best_dyn**
10×2	6,42	7,97	1,38	0,97	1,03	0,97	3,04	0,74
10×3	4,48	4,25	2,21	1,58	1,24	1,21	3,70	0,50
10×5	2,61	4,92	0,85	0,85	0,85	0,70	5,20	0,70
15×2	5,53	6,17	1,61	1,49	1,49	1,33	2,82	0,82
15×3	4,10	4,13	1,62	1,60	1,37	1,37	4,94	0,89
15×5	4,55	4,30	2,21	2,21	2,21	2,21	3,55	1,73
20×2	6,22	6,25	2,34	2,04	2,04	1,86	3,87	1,52
20×3	3,42	4,37	0,96	1,03	1,03	0,85	5,35	0,74
20×5	-	-	-	-	-	-	-	-
30×2	4,95	4,24	1,71	1,71	1,71	1,69	2,69	1,03
30×3	5,26	4,61	2,93	2,86	2,86	2,47	4,11	2,10
30×5	-	-	-	-	-	-	-	-
50×2	-	-	-	-	-	-	-	-
50×3	-	-	-	-	-	-	-	-
50×5	-	-	-	-	-	-	-	-
Gesamt (∅)	**4,76**	**5,12**	**1,78**	**1,63**	**1,58**	**1,47**	**3,93**	**1,08**
Ref_LB1	**6,09**	**6,67**	**2,92**	**2,74**	**2,67**	**2,57**	**5,31**	**2,19**

Tabelle 6.6.: HFS mit unbegrenztem Zwischenspeicher vs. LB2 (Datensatz 2)

Dimension	statisch						dynamisch	
$n \times m$	**(1)**	**(2.1)**	**(2.2)**	**(2.3)**	**(2.4)**	**best**	**dyn**	**best_dyn**
10×2	6,49	5,76	3,45	3,43	3,43	3,03	5,91	2,70
10×3	3,87	6,76	3,17	3,13	2,90	2,30	5,50	2,28
10×5	4,82	3,78	2,23	2,17	2,03	2,03	6,12	2,03
15×2	7,80	6,21	3,58	3,47	3,00	2,96	3,87	2,37
15×3	10,20	10,13	3,80	3,76	3,59	3,39	7,31	2,86
15×5	9,11	10,65	4,98	5,08	5,10	4,12	10,31	3,76
20×2	10,72	9,26	6,11	4,99	4,71	4,71	7,91	3,84
20×3	10,13	11,53	6,87	5,97	6,29	5,65	8,27	4,74
20×5	12,70	11,98	6,43	6,50	6,40	6,20	14,85	5,20
30×2	11,12	11,79	7,02	6,25	6,19	5,80	9,68	4,00
30×3	13,36	13,53	6,59	6,55	6,18	6,11	12,75	5,30
30×5	10,18	10,67	5,61	5,51	5,32	5,22	9,52	5,14
50×2	12,42	11,10	5,74	5,32	5,25	5,20	5,48	4,18
50×3	12,33	11,01	6,88	6,88	6,49	6,25	10,56	4,88
50×5	11,99	9,00	4,47	4,21	3,96	3,96	6,08	3,89
Gesamt (∅)	**9,82**	**9,54**	**5,13**	**4,88**	**4,72**	**4,46**	**8,27**	**3,81**

Tabelle 6.7.: HFS mit unbegrenztem Zwischenspeicher vs. LB1 (Datensatz 3)

Dimension	statisch						dynamisch	
$n \times m$	**(1)**	**(2.1)**	**(2.2)**	**(2.3)**	**(2.4)**	**best**	**dyn**	**best_dyn**
10×2	4,71	3,99	1,75	1,73	1,73	1,34	4,12	1,01
10×3	2,64	5,48	1,95	1,92	1,68	1,09	4,23	1,07
10×5	3,78	2,76	1,22	1,16	1,02	1,02	5,06	1,02
15×2	6,31	4,72	2,11	2,00	1,55	1,50	2,40	0,92
15×3	8,51	8,45	2,21	2,17	2,00	1,81	5,68	1,28
15×5	7,86	9,36	3,76	3,85	3,87	2,91	9,04	2,56
20×2	9,16	7,72	4,62	3,52	3,24	3,24	6,34	2,37
20×3	8,66	10,04	5,44	4,54	4,87	4,23	6,81	3,33
20×5	-	-	-	-	-	-	-	-
30×2	9,49	10,16	5,45	4,70	4,64	4,25	8,08	2,48
30×3	12,01	12,18	5,33	5,28	4,92	4,86	11,41	4,05
30×5	-	-	-	-	-	-	-	-
50×2	-	-	-	-	-	-	-	-
50×3	-	-	-	-	-	-	-	-
50×5	-	-	-	-	-	-	-	-
Gesamt (∅)	**7,31**	**7,49**	**3,38**	**3,09**	**2,95**	**2,63**	**6,32**	**2,01**
Ref_LB1	**8,76**	**8,94**	**4,78**	**4,48**	**4,34**	**4,01**	**7,76**	**3,39**

Tabelle 6.8.: HFS mit unbegrenztem Zwischenspeicher vs. LB2 (Datensatz 3)

Die Berechnung der zweiten unteren Schranke ist wie angenommen wesentlich aufwendiger als im Fall der ersten (vgl. Tabelle 6.9). Aus diesem Grund verzichten wir für die größeren Probleminstanzen auf die Bestimmung der entsprechenden Werte. Um die Performance der beiden Schranken dennoch vergleichbar zu machen, ist in der jeweils letzten Zeile (Ref_LB1) der Tabellen 6.4, 6.6 und 6.8 der durchschnittliche relative Fehler im Vergleich mit LB1 für diejenigen Instanzen angegeben, für welche auch LB2 berechnet wurde. Für den ersten Datensatz beträgt die durchschnittliche Abweichung der beiden Schranken 2,4%. Im Fall der Datensätze 2 und 3 ist diese mit 1,0% bzw. 1,2% sogar noch geringer. Darüber hinaus ist die Differenz scheinbar unabhängig von der Größenordnung des Datensatzes und variiert über die verschiedenen Probleminstanzen selten sehr stark. So beträgt die Standardabweichung der entsprechenden Werte für Datensatz 1 gerade einmal 1,3% sowie 0,9% und 0,8% für die Datensätze 2 und 3. Ferner liegt die maximale Abweichung der beiden Schranken für den ersten Datensatz bei 6,7% sowie 4,7% bzw. 5,1% für die Datensätze 2 und 3. Insgesamt lässt dies auch bei den größeren Instanzen auf ein ähnliches Verhältnis der beiden Schranken zueinander schließen. Zur empirischen Analyse eines heuristischen Verfahrens ist die Verwendung der ersten Schranke in diesem Zusammenhang daher ausreichend.

Die Performance des dynamischen Verfahrens ist vielversprechend, insbesondere im Fall der Datensätze 2 und 3. So beträgt der mittlere relative Fehler im Vergleich mit LB1 weniger als 15% für das erste Szenario sowie etwa 5,0% und 8,3% für die Szenarien 2 und 3. Dies erlaubt den Schluss, dass sich mit geringerer Varianz der Bearbeitungszeiten auf den verschiedenen Maschinenstufen die Ergebnisse der Heuristik verbessern. Für den statischen Fall sinkt die durchschnittliche Abweichung noch einmal (Spalte best) und liegt bei weniger als 10% für Datensatz 1, unter 3% für Datensatz 2 und bei etwa 4,5% für den dritten Datensatz (bzw. 9,1% (DS 1), 2,5% (DS 2) und 3,8% (DS 3), falls man die dynamische Lösung als weitere Lösung des statischen Problems interpretiert, Spalte best_dyn).

Die verschiedenen Sequenzierungsregeln zeigen für alle drei Testreihen ein ähnliches Verhalten. So liefern die Methoden (1) und (2.1) in allen Fällen die im Mittel schlechtesten Ergebnisse, wobei (2.1) meist noch etwas schlechter abschneidet. Bei dieser Beobachtung muss natürlich berücksichtigt werden, dass wir die Regeln anhand des statischen Verfahrens vergleichen. Im Gegensatz zu (2.2) - (2.4) werden bei den Regeln (1) und (2.1) die Freigabezeiten der Jobs bei der Sequenzierung nicht berücksichtigt. Mithin sind diese beiden Methoden zur Bearbeitung des statischen Problems nur bedingt geeignet.

Dennoch legen die Resultate den Schluss nahe, dass die Methoden (1) und (2.1) für ein erfolgreiches Hybridverfahren möglicherweise vernachlässigt werden könnten. Dass diese Folgerung nicht zutreffend ist, zeigt Tabelle 6.10. In der jeweils ersten Zeile (best) für jeden Datensatz ist darin die Anzahl von Instanzen notiert, für welche die entsprechende Sequenzierung die beste Lösung liefert. Da diese in vielen Fällen von mehr als einer Regel bestimmt wird, ist darüber hinaus in der jeweils zweiten Zeile (best_unique) angegeben,

Dimension	Heuristik (ms)			LB1 (s)			LB2 (s)		
$n \times m$	**DS 1**	**DS 2**	**DS 3**	**DS 1**	**DS 2**	**DS 3**	**DS 1**	**DS 2**	**DS 3**
10×2	53,0	32,7	35,7	1,0	0,5	0,5	37,1	18,0	11,4
10×3	46,6	32,6	37,3	3,6	1,3	0,8	133,8	56,2	28,8
10×5	76,3	49,7	51,4	12,0	3,8	2,0	1818,2	863,4	206,4
15×2	101,6	65,5	52,6	4,9	2,6	1,6	356,0	157,7	31,7
15×3	114,1	70,2	73,3	12,6	4,1	2,4	1470,1	505,5	149,6
15×5	154,8	70,4	89,0	31,0	16,3	7,9	-	4162,7	1898,3
20×2	131,3	81,1	99,8	9,3	3,3	2,4	875,6	147,7	130,1
20×3	150,3	100,1	98,5	44,2	11,7	6,5	3061,4	3017,3	806,4
20×5	221,8	142,0	134,2	232,6	41,5	20,6	-	-	-
30×2	201,5	131,4	182,9	48,2	25,5	12,8	3066,0	1013,5	1075,4
30×3	284,3	157,8	211,1	243,1	49,4	31,5	-	3820,7	3272,5
30×5	382,7	303,3	234,7	1529,0	400,2	161,5	-	-	-
50×2	498,3	450,0	329,9	442,4	200,4	184,9	-	-	-
50×3	502,8	534,4	429,9	3124,9	619,0	568,5	-	-	-
50×5	756,3	542,3	594,0	12160,6	6538,4	4872,8	-	-	-

Tabelle 6.9.: Rechenzeiten bei unbegrenztem Zwischenspeicher

Datensatz		(1)	(2.1)	(2.2)	(2.3)	(2.4)
1	best	50	14	44	61	69
(150 Instanzen)	best-unique	50	14	14	2	11
	Diff $\leq 1\%$	10	8	25	21	25
2	best	9	30	103	115	116
(150 Instanzen)	best-unique	7	14	11	0	2
	Diff $\leq 1\%$	24	7	36	28	29
3	best	13	9	84	98	117
(150 Instanzen)	best-unique	8	1	13	3	26
	Diff $\leq 1\%$	10	13	34	33	24

Tabelle 6.10.: Anzahl bester/guter Lösungen bei unbegrenztem Zwischenspeicher

für wie viele Instanzen eine Regel alleine die beste Lösung liefert. Dabei wird insbesondere für die Anordnung der Jobs gemäß der verallgemeinerten WSPT-Regel (1) deutlich, dass eine beste Lösung, welche durch diese Methode bestimmt wird, selten auch von einer anderen erzielt werden kann. Demnach darf gerade die Bedeutung von Regel (1) für den Erfolg des Hybridverfahrens grundsätzlich nicht unterschätzt werden. Der Effekt der Variante (2.1) ist zwar schwächer, aber auch hier sind einige Datensätze zu verzeichnen, für welche alleine diese Regel die beste Jobsequenz liefert. Die entsprechende Anzahl für Strategie (2.3) ist dagegen wesentlich geringer (DS 1: 2, DS 2: 0, DS 3: 3). Somit könnte man zur Beschleunigung der ersten Phase des Verfahrens gegebenenfalls Sequenz (2.3) vernachlässigen.

In der jeweils dritten Zeile (Diff $\leq 1\%$) der Tabelle 6.10 ist zudem für jedes Szenario die Anzahl der Instanzen angegeben, für welche die entsprechende Sequenzierungsregel zwar nicht die beste Lösung liefert, der zugehörige Zielfunktionswert aber um höchstens 1% von dem der besten Jobliste abweicht (vgl. Abschnitt 2.2.3). Die vermerkten Anzahlen unterstreichen in erster Linie die bessere Performance der Methoden (2.2) - (2.4). Insbesondere für die Datensätze 2 und 3 kann man mit Hilfe der Regeln (2.3) und (2.4) bereits für fast alle der insgesamt 150 Testinstanzen eine Jobsequenz bestimmen, die entweder die beste Lösung liefert oder nur geringfügig von dieser abweicht. Sollte also eine Situation auftreten, welche ein beschleunigtes Verfahren erfordert, so könnte man dieses ausschließlich auf Basis der Regel (2.4) konstruieren. Dabei ist aber natürlich mit einer (geringfügigen) Verschlechterung der Performance zu rechnen.

6.2.2. HFS mit No-Wait-Constraints

Zur Untersuchung von Phase 1 für das Problem mit No-Wait-Constraints verwenden wir dieselben Testinstanzen wie im vorangehenden Abschnitt. Darüber hinaus evaluieren wir die Ergebnisse unter Verwendung der zuvor berechneten unteren Schranken für das Problem mit unbegrenztem Zwischenspeicher. Dabei ist natürlich mit einer Verschlechterung des durchschnittlichen relativen Fehlers zu rechnen. Grundsätzlich kann man die Modelle durch Ersetzen von „$\leq$“ durch „$=$“ in den Bedingungen (4.33) und (4.40) zwar leicht für No-Wait-Constraints anpassen, geht man aber zur LP-Relaxation der Formulierungen über, so ist diese Einschränkung nicht mehr bindend. Die sich ergebende Lösung hat zwar möglicherweise eine andere Struktur, der Zielfunktionswert bleibt jedoch bis auf wenige Ausnahmen gleich (vgl. Beispiel 4.3.6). Aus diesem Grund greifen wir an dieser Stelle auf die bereits vorliegenden Schranken zurück.

Die Ergebnisse der Testreihen für das Problem $FHm,(PM^{(k)})_{k=1}^{m}|r_j,no-wait|\sum w_jC_j$ sind in den Tabellen 6.11 bis 6.16 dargestellt. Diese sind analog zu den entsprechenden Resultaten im vorangehenden Abschnitt strukturiert. Die zugehörigen Rechenzeiten können Tabelle 6.17 entnommen werden. Grundsätzlich weist das heuristische Verfahren im Fall von No-Wait-Constraints eine höhere Laufzeit auf. Dies ist auf die Variante des ent-

Dimension	statisch						dynamisch	
$n \times m$	**(1)**	**(2.1)**	**(2.2)**	**(2.3)**	**(2.4)**	**best**	**dyn**	**best_dyn**
10×2	13,93	13,69	11,93	10,12	8,47	7,52	8,80	6,80
10×3	10,77	13,06	13,02	10,95	10,95	8,11	8,09	6,66
10×5	15,74	16,04	12,80	11,65	11,45	10,87	15,89	10,78
15×2	15,06	17,01	12,82	12,13	12,10	11,20	13,87	10,25
15×3	13,46	18,07	12,23	10,59	9,84	8,83	14,57	7,86
15×5	14,96	19,96	12,93	12,18	12,17	11,70	15,77	10,82
20×2	13,82	15,74	12,24	11,61	11,61	10,35	14,30	9,57
20×3	19,90	24,77	22,22	22,26	21,98	19,38	20,10	17,87
20×5	23,21	26,82	25,14	23,75	23,24	19,30	22,78	18,10
30×2	14,08	19,62	16,46	15,57	15,57	12,55	15,77	11,98
30×3	22,12	26,04	23,21	22,74	22,24	18,44	21,62	16,82
30×5	21,16	24,73	22,08	20,74	20,03	17,10	20,31	16,65
50×2	15,75	24,95	24,41	23,49	23,52	14,98	16,58	14,24
50×3	25,49	34,13	32,30	32,98	33,07	24,10	23,29	22,27
50×5	21,15	24,00	35,41	32,33	32,61	20,51	23,24	19,89
Gesamt (∅)	**17,37**	**21,24**	**19,28**	**18,20**	**17,92**	**14,33**	**17,00**	**13,37**

Tabelle 6.11.: HFS mit No-Wait-Constraints vs. LB1 (Datensatz 1)

Dimension	statisch						dynamisch	
$n \times m$	**(1)**	**(2.1)**	**(2.2)**	**(2.3)**	**(2.4)**	**best**	**dyn**	**best_dyn**
10×2	10,22	9,96	8,24	6,53	4,94	4,02	5,28	3,33
10×3	5,64	8,28	10,53	10,48	8,43	8,43	5,67	4,25
10×5	13,66	13,98	10,80	9,67	9,48	8,90	13,83	8,81
15×2	12,04	13,94	9,84	9,18	9,15	8,28	10,87	7,34
15×3	10,74	15,24	9,54	7,93	7,20	6,23	11,79	5,27
15×5	-	-	-	-	-	-	-	-
20×2	11,20	13,06	9,59	8,98	8,98	7,77	11,63	7,01
20×3	16,59	21,32	18,84	18,88	18,61	16,08	16,78	14,61
20×5	-	-	-	-	-	-	-	-
30×2	12,40	17,84	14,73	13,86	13,86	10,88	14,06	10,33
30×3	-	-	-	-	-	-	-	-
30×5	-	-	-	-	-	-	-	-
50×2	-	-	-	-	-	-	-	-
50×3	-	-	-	-	-	-	-	-
50×5	-	-	-	-	-	-	-	-
Gesamt (∅)	**11,89**	**14,48**	**11,51**	**10,43**	**10,08**	**8,48**	**11,24**	**7,62**
Ref_LB1	**14,59**	**17,25**	**14,21**	**13,11**	**12,74**	**11,10**	**13,92**	**10,22**

Tabelle 6.12.: HFS mit No-Wait-Constraints vs. LB2 (Datensatz 1)

Dimension	statisch						dynamisch	
$n \times m$	**(1)**	**(2.1)**	**(2.2)**	**(2.3)**	**(2.4)**	**best**	**dyn**	**best_dyn**
10×2	7,43	8,94	3,64	3,26	3,32	3,25	3,70	2,92
10×3	6,84	6,67	4,30	3,45	3,04	3,02	4,79	2,05
10×5	4,76	6,63	2,61	2,61	2,61	2,60	5,13	2,41
15×2	7,68	8,33	3,99	3,93	3,93	3,71	4,69	3,15
15×3	5,74	5,78	3,26	3,23	3,12	3,12	5,10	2,44
15×5	5,41	6,13	4,40	4,40	4,40	4,26	5,14	3,70
20×2	7,99	8,77	5,22	4,81	4,81	4,50	5,68	3,68
20×3	5,20	6,30	3,08	2,96	2,96	2,91	4,46	2,80
20×5	7,14	6,83	4,20	4,43	4,43	3,81	5,23	3,77
30×2	6,47	6,02	3,43	3,42	3,42	3,38	4,03	2,78
30×3	7,27	6,99	5,29	5,28	5,28	4,97	5,08	4,10
30×5	8,02	7,13	5,25	5,25	5,25	4,99	5,48	4,36
50×2	6,03	5,33	3,34	3,47	3,47	3,31	4,36	3,01
50×3	5,49	5,43	3,34	3,33	3,33	3,09	3,23	2,52
50×5	11,70	12,91	11,66	11,44	11,44	10,99	12,01	10,63
Gesamt (∅)	**6,88**	**7,21**	**4,47**	**4,35**	**4,32**	**4,13**	**5,21**	**3,62**

Tabelle 6.13.: HFS mit No-Wait-Constraints vs. LB1 (Datensatz 2)

Dimension	statisch						dynamisch	
$n \times m$	**(1)**	**(2.1)**	**(2.2)**	**(2.3)**	**(2.4)**	**best**	**dyn**	**best_dyn**
10×2	5,26	6,75	1,55	1,18	1,24	1,18	1,61	0,85
10×3	5,53	5,36	3,01	2,17	1,78	1,75	3,49	0,80
10×5	3,55	5,41	1,44	1,44	1,44	1,43	3,95	1,23
15×2	6,17	6,82	2,53	2,46	2,46	2,25	3,20	1,70
15×3	4,43	4,46	1,97	1,95	1,84	1,83	3,80	1,17
15×5	4,22	4,93	3,22	3,22	3,22	3,09	3,95	2,52
20×2	6,54	7,31	3,79	3,40	3,40	3,08	4,25	2,27
20×3	4,14	5,21	2,03	1,92	1,92	1,86	3,40	1,75
20×5	-	-	-	-	-	-	-	-
30×2	5,48	5,04	2,48	2,47	2,47	2,42	3,08	1,84
30×3	6,46	6,18	4,48	4,48	4,48	4,17	4,28	3,31
30×5	-	-	-	-	-	-	-	-
50×2	-	-	-	-	-	-	-	-
50×3	-	-	-	-	-	-	-	-
50×5	-	-	-	-	-	-	-	-
Gesamt (∅)	**5,18**	**5,75**	**2,65**	**2,47**	**2,42**	**2,31**	**3,50**	**1,74**
Ref_LB1	**5,77**	**6,17**	**3,31**	**3,15**	**3,09**	**3,01**	**4,07**	**2,48**

Tabelle 6.14.: HFS mit No-Wait-Constraints vs. LB2 (Datensatz 2)

Dimension	**statisch**						**dynamisch**	
$n \times m$	**(1)**	**(2.1)**	**(2.2)**	**(2.3)**	**(2.4)**	**best**	**dyn**	**best_dyn**
10×2	6,78	6,00	3,05	3,03	3,03	3,02	3,76	2,58
10×3	4,62	7,08	3,46	3,42	3,86	2,93	3,61	2,40
10×5	5,84	5,73	3,10	2,99	2,77	2,73	4,20	2,22
15×2	7,79	6,51	3,86	3,75	3,44	3,07	3,07	2,35
15×3	10,51	10,67	4,49	4,34	4,17	4,07	4,91	3,58
15×5	11,43	12,59	6,70	6,71	6,77	6,28	6,87	5,13
20×2	10,23	9,77	6,82	5,90	5,62	5,49	4,83	3,97
20×3	12,17	14,80	10,46	9,55	9,99	8,51	7,66	7,27
20×5	13,70	14,50	9,43	9,49	9,78	9,25	10,64	8,17
30×2	12,61	13,85	8,32	7,39	7,61	7,20	5,71	5,01
30×3	12,86	14,35	8,90	8,94	8,94	8,14	7,13	6,55
30×5	12,38	13,42	10,19	10,46	9,51	8,83	8,91	6,98
50×2	13,07	13,61	8,37	8,30	8,31	7,91	6,96	6,73
50×3	14,83	16,04	12,95	12,38	11,92	10,66	9,60	8,25
50×5	14,95	13,86	10,71	10,56	10,72	10,22	9,09	8,68
Gesamt (∅)	**10,92**	**11,52**	**7,39**	**7,15**	**7,10**	**6,55**	**6,46**	**5,32**

Tabelle 6.15.: HFS mit No-Wait-Constraints vs. LB1 (Datensatz 3)

Dimension	**statisch**						**dynamisch**	
$n \times m$	**(1)**	**(2.1)**	**(2.2)**	**(2.3)**	**(2.4)**	**best**	**dyn**	**best_dyn**
10×2	5,00	4,22	1,36	1,34	1,34	1,33	2,05	0,89
10×3	3,37	5,78	2,23	2,19	2,62	1,70	2,38	1,19
10×5	4,79	4,69	2,07	1,97	1,75	1,71	3,17	1,21
15×2	6,30	5,02	2,37	2,27	1,98	1,62	1,60	0,91
15×3	8,82	8,97	2,88	2,74	2,57	2,48	3,30	1,99
15×5	10,14	11,27	5,45	5,46	5,52	5,03	5,64	3,90
20×2	8,66	8,22	5,31	4,41	4,13	4,00	3,36	2,50
20×3	10,68	13,27	8,98	8,08	8,52	7,06	6,22	5,84
20×5	-	-	-	-	-	-	-	-
30×2	10,96	12,20	6,74	5,82	6,05	5,64	4,16	3,47
30×3	11,52	13,00	7,61	7,65	7,65	6,86	5,86	5,30
30×5	-	-	-	-	-	-	-	-
50×2	-	-	-	-	-	-	-	-
50×3	-	-	-	-	-	-	-	-
50×5	-	-	-	-	-	-	-	-
Gesamt (∅)	**8,02**	**8,66**	**4,50**	**4,19**	**4,21**	**3,74**	**3,78**	**2,72**
Ref_LB1	**9,48**	**10,14**	**5,91**	**5,60**	**5,62**	**5,14**	**5,17**	**4,11**

Tabelle 6.16.: HFS mit No-Wait-Constraints vs. LB2 (Datensatz 3)

Dimension	Heuristik (ms)			LB1 (s)			LB2 (s)		
$n \times m$	DS 1	DS 2	DS 3	DS 1	DS 2	DS 3	DS 1	DS 2	DS 3
10×2	81,2	40,4	49,8	1,0	0,5	0,5	37,1	18,0	11,4
10×3	59,0	46,7	49,8	3,6	1,3	0,8	133,8	56,2	28,8
10×5	251,6	142,0	113,9	12,0	3,8	2,0	1818,2	863,4	206,4
15×2	136,0	85,9	73,2	4,9	2,6	1,6	356,0	157,7	31,7
15×3	206,2	143,6	114,0	12,6	4,1	2,4	1470,1	505,5	149,6
15×5	504,7	115,7	223,3	31,0	16,3	7,9	-	4162,7	1898,3
20×2	271,7	112,3	132,6	9,3	3,3	2,4	875,6	147,7	130,1
20×3	343,6	230,9	143,7	44,2	11,7	6,5	3061,4	3017,3	806,4
20×5	899,4	381,3	406,3	232,6	41,5	20,6	-	-	-
30×2	317,3	213,8	329,6	48,2	25,5	12,8	3066,0	1013,5	1075,4
30×3	690,5	324,9	590,6	243,1	49,4	31,5	-	3820,7	3272,5
30×5	2456,1	1403,3	828,2	1529,0	400,2	161,5	-	-	-
50×2	1034,4	661,1	535,9	442,4	200,4	184,9	-	-	-
50×3	1297,2	1356,3	1337,4	3124,9	619,0	568,5	-	-	-
50×5	4886,2	1396,9	1715,8	12160,6	6538,4	4872,8	-	-	-

Tabelle 6.17.: Rechenzeiten bei No-Wait-Constraints

haltenen List Scheduling Verfahrens zurückzuführen, denn im Allgemeinen müssen mehr mögliche Startzeitpunkte für einen Job überprüft werden, bis ein solcher unter No-Wait-Constraints gefunden werden kann. Die angegebenen Rechenzeiten für die Bestimmung der unteren Schranken entsprechen natürlich denen in Tabelle 6.9.

Im Wesentlichen zeigt unsere heuristische Methode für No-Wait-Constraints dasselbe Verhalten wie im Fall von unbegrenztem Zwischenspeicher. Lediglich der durchschnittliche relative Fehler im Vergleich zu den unteren Schranken ist wie erwartet etwas größer. So weicht die dynamische Lösung für Datensatz 1 im Mittel um 17% von dem Wert der ersten Schranke ab sowie etwa um 5,2% und 6,5% für die Datensätze 2 und 3. Auch hier sind die entsprechenden Werte im Fall des statischen Verfahrens (Spalte best) mit ca. 14,3% (DS 1), 4,1% (DS 2) bzw. 6,5% (DS 3) geringer.

Auch der Vergleich der einzelnen Sequenzierungsregeln führt zu denselben Schlussfolgerungen. Lediglich die Anzahlen der Instanzen, für die eine Regel die (alleinige) beste Jobanordnung liefert (vgl. Tabelle 6.18), sind etwas moderater verteilt als für das Problem mit unbegrenztem Zwischenspeicher. Die Tendenz bleibt jedoch unverändert. Grundsätzlich erweisen sich die Strategien (2.2) - (2.4) als effizienter im Vergleich zu (1) oder (2.1). Dennoch liefert insbesondere Regel (1) häufig als einzige die beste Jobsequenz (vgl. Datensatz 1). Insgesamt schneidet aber Methode (2.4) auch hier am besten ab. Gerade bei einer starken Schwankung in den Eingabedaten (z. B. Datensatz 1 mit $p_{ij} \in U[1,99]$) erweist sich aber die Verwendung eines Hybridansatzes auf der Basis verschiedener Se-

Datensatz		(1)	(2.1)	(2.2)	(2.3)	(2.4)
1	best	63	11	42	56	61
(150 Instanzen)	best-unique	63	11	13	0	7
	Diff $\leq 1\%$	13	12	16	14	10
2	best	14	32	100	107	107
(150 Instanzen)	best-unique	13	15	13	0	1
	Diff $\leq 1\%$	24	16	34	32	34
3	best	23	11	83	89	102
(150 Instanzen)	best-unique	17	4	16	4	20
	Diff $\leq 1\%$	7	11	27	34	21

Tabelle 6.18.: Anzahl bester/guter Instanzen bei No-Wait-Constraints

quenzierungsregeln als sinnvoll.

6.2.3. Dynamisches HFS mit Transporten und Blocking-Constraints

Um die Performance des gesamten dynamischen Verfahrens zu evaluieren, wurde dieses ebenfalls in C++ implementiert[3] und für die in Abschnitt 6.1 beschriebenen Datensätze 4 - 6 ausgeführt. Als Benchmark verwenden wir die unteren Schranken, welche sich aus der LP-Relaxation der in Absatz 4.3.2 beschriebenen Modifikation der zeit-indizierten Formulierungen zur Berücksichtigung von Transportrestriktionen ergeben. Der mittlere relative Fehler sowie die Berechnungszeit der unteren Schranken (in Sekunden) für die jeweils zehn zufälligen Instanzen jeder untersuchten Dimension sind in den Tabellen 6.19 bis 6.21 dargestellt. Die jeweils vorletzte Zeile (Gesamt ($\emptyset$)) enthält ferner den durchschnittlichen relativen Fehler im Vergleich mit LB1 und LB2 über alle betrachteten Instanzen eines Szenarios.

Auch für das Problem mit Transporten ist die Berechnung der zweiten Schranke natürlich wesentlich zeitaufwendiger als für die erste. Daher beschränken wir uns im Fall der Datensätze 4 - 6 ebenfalls auf die Verwendung von LB1 als Benchmark für die größeren Probleminstanzen. Analog zu den vorangehend beschriebenen Testreihen ist als Referenz für LB2 zudem die mittlere Abweichung der kleineren Instanzen mit 5 bis 15 Jobs in Bezug auf LB1 angegeben (letzte Zeile, Ref_LB1). Eine geringere Anzahl verfügbarer Fahrzeuge erhöht den Wert der Schranke für dieselbe Instanz nur minimal bzw. in den meisten Fällen gar nicht. Im Allgemeinen verringert sich lediglich die Rechenzeit, je mehr Fahrzeuge zur Verfügung stehen. Aus diesem Grund verwenden wir den Wert von LB2 für drei Transportmittel darüber hinaus auch als Schranke für dieselbe Probleminstanz mit nur zwei vorhandenen Fahrzeugen. Der Vergleich der Schranken zeigt auch hier, dass LB1 in der Praxis von LB2 leicht dominiert wird. In diesem Zusammenhang ist der

[3] Auch hier ist der Quellcode im Rahmen des genannten Projekts entstanden.

Dimension	**2 Fahrzeuge**				**3 Fahrzeuge**			
$n \times m$	**LB1**	**t(s)**	**LB2**	**t(s)**	**LB1**	**t(s)**	**LB2**	**t(s)**
5×2	6,36	0,6	4,91	16,4	4,43	0,5	3,03	16,4
5×3	11,25	0,9	9,37	29,6	9,13	0,9	7,27	29,6
10×2	13,03	2,9	10,94	151,4	8,51	2,9	6,50	151,4
10×3	18,32	8,2	17,01	1373,2	10,88	7,2	9,65	1373,2
15×2	18,62	13,0	16,93	536,3	9,73	11,9	8,18	536,3
15×3	20,80	25,0	19,42	4481,9	12,73	19,7	11,43	4481,9
20×2	22,20	33,0	-	-	12,48	32,7	-	-
20×3	25,46	111,8	-	-	14,79	101,0	-	-
30×2	20,64	120,4	-	-	12,48	93,6	-	-
30×3	29,41	603,6	-	-	17,57	511,2	-	-
Gesamt (∅)	**18,61**	**91,9**	**13,10**	**1098,1**	**11,11**	**78,2**	**7,68**	**1098,1**
Ref_LB1			**14,73**	**8,4**			**9,24**	**7,2**

Tabelle 6.19.: Dynamisches HFS mit Transporten und Blocking (Datensatz 4)

erforderliche Mehraufwand zur Bestimmung von LB2 jedoch unverhältnismäßig groß.

Für das Hybrid Flow-Shop-Problem mit Transporten werten wir an dieser Stelle nur die dynamische Version des gesamten Verfahrens aus, da eine weitere Untersuchung der verschiedenen Sequenzierungsregeln dabei keine weiteren Einsichten erwarten lässt und die Lösung des statischen Problems demnach nicht benötigt wird. In Anlehnung an das zugrundeliegende Anwendungsproblem untersuchen wir hier zudem lediglich das Problem mit Blocking-Constraints. Zu diesem Zweck ist in Phase 1 nur eine Verwendung des List Scheduling Verfahrens für No-Wait-Constraints (anstelle von unbegrenztem Zwischenspeicher) sinnvoll. In Phase 3 des Verfahrens werden, basierend auf der Lösung des Sequencing-Problems, die notwendigen Transporte zwischen zwei Arbeitsgängen desselben Jobs hinzugefügt, wobei gegebenenfalls auch Wartezeiten des Jobs auf einer Maschine entstehen können. Insgesamt wird die ursprüngliche Lösung mit No-Wait-Constraints aus Phase 1 und 2 dabei also in eine Lösung transformiert, welche Blocking-Constraints genügt. Für weitere Einzelheiten sei auf Anhang A verwiesen. Als Benchmark verwenden wir hier jedoch erneut die Modelle mit unbegrenztem Zwischenspeicher, da die in 4.3.1 vorgestellte Modifikation der Formulierungen für Blocking-Constraints nur bei sehr genau gewähltem Wert für T eine (geringe) Verbesserung verspricht. Für beliebige Datensätze ist eine solche Begrenzung der Planungsperiode im Vorhinein jedoch schwer möglich und deshalb verzichten wir bei der Schrankenbildung auf die Lösung des komplexeren Modells.

Des Weiteren ist es im Fall des dynamischen Verfahrens wenig sinnvoll die Rechenzeiten genau zu analysieren, da im Wesentlichen die Dauer jedes einzelnen Durchlaufs entscheidend ist. Eine Reoptimierung hängt aber zumeist nur von einer relativ geringen

Dimension	**2 Fahrzeuge**				**3 Fahrzeuge**			
$n \times m$	**LB1**	**t(s)**	**LB2**	**t(s)**	**LB1**	**t(s)**	**LB2**	**t(s)**
5×2	6,62	0,3	5,35	9,7	5,02	0,3	3,76	9,7
5×3	6,18	0,8	5,20	27,3	3,89	0,7	2,94	27,3
10×2	12,67	1,9	10,88	213,4	6,96	1,7	5,24	213,4
10×3	15,96	5,2	14,03	1209,5	10,92	4,6	9,08	1209,5
15×2	21,88	7,7	19,87	162,4	12,58	7,4	10,71	162,4
15×3	22,14	16,2	20,60	3831,7	12,06	14,9	10,65	3831,7
20×2	27,62	36,3	-	-	14,77	31,3	-	-
20×3	25,95	96,6	-	-	17,17	66,9	-	-
30×2	23,66	221,4	-	-	12,09	163,2	-	-
30×3	25,38	493,5	-	-	17,56	353,3	-	-
Gesamt (∅)	**18,81**	**88,0**	**12,65**	**909,0**	**11,30**	**64,4**	**7,06**	**909,0**
Ref_LB1			**14,24**	**5,4**			**8,57**	**5,0**

Tabelle 6.20.: Dynamisches HFS mit Transporten und Blocking (Datensatz 5)

Dimension	**2 Fahrzeuge**				**3 Fahrzeuge**			
$n \times m$	**LB1**	**t(s)**	**LB2**	**t(s)**	**LB1**	**t(s)**	**LB2**	**t(s)**
5×2	7,14	0,5	5,61	14,7	5,88	1,6	4,37	14,7
5×3	6,64	1,1	5,04	46,6	4,36	1,1	2,79	46,6
10×2	14,81	3,2	11,56	236,6	11,73	3,1	8,55	236,6
10×3	16,17	8,1	14,57	949,2	12,58	6,3	11,02	949,2
15×2	18,55	16,6	15,87	876,3	13,21	14,4	10,65	876,3
15×3	17,87	34,6	16,30	6911,3	12,97	30,2	11,47	6911,3
20×2	15,39	44,5	-	-	12,94	51,7	-	-
20×3	22,48	82,1	-	-	15,82	72,9	-	-
30×2	22,45	202,5	-	-	13,81	135,5	-	-
30×3	23,76	646,2	-	-	18,29	382,7	-	-
Gesamt (∅)	**16,53**	**103,9**	**11,49**	**1505,8**	**12,16**	**70,0**	**8,14**	**1505,8**
Ref_LB1			**13,53**	**10,7**			**10,12**	**9,5**

Tabelle 6.21.: Dynamisches HFS mit Transporten und Blocking (Datensatz 6)

Jobanzahl ab und die entsprechende Laufzeit ist demnach nicht sehr aussagekräftig. Aus diesem Grund verzichten wir an dieser Stelle auf eine ausführliche Darstellung.[4] Stattdessen werden wir im nächsten Kapitel bei der Performance-Analyse des Verfahrens für das Prozesslabor detaillierter auf diesen Aspekt eingehen.

Insgesamt sind die in den Tabellen 6.19 bis 6.21 dargestellten Ergebnisse vielversprechend. Das entwickelte heuristische Verfahren weist eine sehr geringe Rechenzeit auf und liefert dennoch gute Lösungen für ein Scheduling-Problem mit sehr hoher Komplexität und großer praktischer Relevanz. Die erzielten Resultate sind in der Summe für alle drei Testreihen ähnlich. Die Datensätze 4 und 5 unterscheiden sich zwar nicht in ihrer grundlegenden Struktur, es wurde bei der Generierung der Daten jedoch das Intervall für die Freigabezeiten der Jobs variiert, um so eine unterschiedlich starke Auslastung des Systems zu simulieren. Die sich ergebenden Werte für den relativen Fehler im Vergleich mit LB1 sind für beide Datensätze nahezu gleich. So liegt dieser bei zwei Fahrzeugen für beide Datensätze bei etwas weniger als 19% und im Fall von drei Transportmitteln bei etwas mehr als 11%. Entsprechend geringer sind diese Werte im Vergleich zu LB2. Das Verfahren liefert demnach bei unterschiedlich ausgeprägtem Jobaufkommen vergleichbare Resultate. Noch etwas bessere Ergebnisse liegen, mit einer Abweichung von etwa 16,5% bei zwei Fahrzeugen und ca. 12% bei drei Fahrzeugen (durchschnittlicher relativer Fehler gegenüber LB1), für Datensatz 6 vor. Hier wurden bei der Datenerzeugung die Bearbeitungszeiten im Verhältnis zu den Transportzeiten etwas vergrößert. Mit anderen Worten, das Verfahren scheint für geringere Transportzeiten etwas besser zu arbeiten.

Bei einer Bewertung der obigen Resultate darf nicht übersehen werden, dass in der als Benchmark verwendeten unteren Schranke neben dem Verzicht auf die Ganzzahligkeit auch die Transportrestriktionen nur in einer vereinfachten Form einbezogen wurden. So werden gegebenenfalls notwendige Leerfahrten nicht berücksichtigt, sondern es wird angenommen, dass sich ein Fahrzeug, welches frei ist, auch bereits am „richtigen“ Ort befindet. Gegenüber der Analyse von Phase 1 sind die Schranken bei der Auswertung also noch etwas abgeschwächt.

6.3. Ergänzende Bemerkungen

Für die in diesem Kapitel verwendeten theoretischen Probleminstanzen ist eine gesonderte Evaluation der zweiten Phase des Verfahrens (Maschinenauswahl) nicht erforderlich, da in allen Datensätzen identisch parallele Maschinen auf den Bearbeitungsstufen vorausgesetzt wurden. Die verbesserte Maschinenauswahl kommt lediglich bei der Lösung des Anwendungsproblems zum Einsatz (vgl. Kapitel 7). Grundsätzlich könnte man zur Analyse von Phase 2 aber natürlich ebenfalls mit Hilfe eines entsprechenden Optimie-

[4] Die Rechenzeit beträgt für das gesamte Verfahren (Phasen 1-3) ebenfalls nur einen Sekundenbruchteil für kleine Instanzen und maximal wenige Sekunden für die größten Instanzen.

rungsproblems eine Referenz konstruieren. Basierend auf einer Bewertung der parallelen Maschinen mit Hilfe von Güteklassen wie in Phase 2 könnte ein solches beispielsweise die unten dargestellte Form besitzen. Zur Formulierung des Modells legen wir zudem folgende Bezeichnungen und Variablen fest:

- a_{ijt} - binärer Parameter, der angibt, ob Job j im Intervall $[t,t+1]$ auf Maschinenstufe i bearbeitet wird, oder nicht (ergibt sich aus Phase 1 des Verfahrens)
- q_{il} - Güteklasse der Maschine l des Typs i
- m_{ilj} - nimmt den Wert 1 an, falls Maschine l des Typs i für die Bearbeitung des Jobs j ausgewählt wurde, und 0 sonst
- x_{iljt} - nimmt den Wert 1 an, falls $a_{ijt} = 1$ und $m_{ilj} = 1$, und 0 sonst

Unter Verwendung dieser Notationen kann man mit Hilfe eines wie folgt strukturierten Modells eine optimale[5] Maschinenauswahl für eine Bearbeitungsstufe i bestimmen:

$$\sum_{l,j} w_j q_{il} m_{ilj} \rightarrow \text{Max!}$$

so dass

$$\sum_{l} m_{ilj} = 1 \qquad \textit{für alle } j \tag{6.1}$$

$$x_{iljt} \leq a_{ijt} \qquad \textit{für alle } l,j,t \tag{6.2}$$

$$x_{iljt} \leq m_{ilj} \qquad \textit{für alle } l,j,t \tag{6.3}$$

$$\sum_{l,t} x_{iljt} = p_{ij} \qquad \textit{für alle } j \tag{6.4}$$

$$\sum_{j} x_{iljt} \leq 1 \qquad \textit{für alle } l,t \tag{6.5}$$

Die Zielfunktion ordnet Jobs mit großen Gewichten Maschinen mit großer Güteklasse zu. Dabei garantiert (6.1), dass für jeden Job j genau eine Maschine des Typs i ausgewählt wird. Die verbleibenden vier Restriktionen (6.2) - (6.5) modellieren des Weiteren die Zulässigkeit der Auswahl in Abhängigkeit des zuvor in Phase 1 bestimmten Schedules.

Wir verzichten an dieser Stelle jedoch auf eine explizite Analyse der zweiten Phase, da sich die Maschinen einer Stufe im Prozesslabor lediglich in ihrer Position unterscheiden, grundsätzlich aber gleich schnell arbeiten. Da die Transportzeiten zudem nur einen kleinen Teil der Durchlaufzeit einer Probe bestimmen, hat demnach auch die Maschinenauswahl nur eine geringe Auswirkung auf den Zielfunktionswert.

Von einer mathematischen Formulierung des Routing-Problems und einer gesonderten empirischen Evaluation der dritten Phase des Verfahrens sehen wir daher ebenfalls

[5] Optimal bezieht sich hier auf die in Phase 2 zugrundeliegenden Annahmen.

ab. Grundsätzlich ist die Entwicklung eines entsprechenden Modells natürlich möglich, da im Rahmen des Routing-Problems jedoch auch Rüstzeiten der Maschinen bzw. Leerfahrten der Transportmittel berücksichtigt werden müssen, ist die Verwendung eines LP-basierten Ansatzes in diesem Zusammenhang nur bedingt geeignet. Rüstzeiten fallen jeweils zwischen der Bearbeitung zweier verschiedener Jobs auf derselben Maschine an. In der Lösung einer LP-Relaxation wird ein Job aber gegebenenfalls in sehr viele Teiljobs aufgespalten und es stellt sich die Frage, wie in einem solchen Fall sinnvoll sequenzabhängige Rüstzeiten abgebildet werden können. Alternativ könnte man die Rüstzeiten antizipieren, also in die Bearbeitungszeiten einbeziehen, und so das Routing-Problem auf ein Parallelmaschinen-Problem mit Reihenfolgebedingungen, Freigabezeiten und Verzögerungszeiten reduzieren. Aus den oben genannten Gründen gehen wir an dieser Stelle jedoch nicht weiter darauf ein, sondern untersuchen im nächsten Kapitel die Arbeitsweise des Verfahrens am Beispiel des zugrundeliegenden Anwendungsproblems.

7. Fallstudie: Qualitätssicherung in einem Stahlwerk

Die in den vorangehenden Kapiteln untersuchten Hybrid Flow-Shop-Probleme, insbesondere im Zusammenhang mit Transportrestriktionen, sind nicht nur von theoretischem Interesse, sondern treten auch in der Praxis auf. Im Folgenden beschreiben wir nun zunächst ein konkretes Anwendungsproblem dieses Typs in der Stahlindustrie (Abschnitt 7.1). Anschließend wird das in Kapitel 5 konstruierte Verfahren zur Lösung dieses speziellen Problems angepasst (vgl. Abschnitt 7.2). In Abschnitt 7.3 untersuchen wir schließlich die Effektivität der Methode anhand entsprechender Echtdaten. Insgesamt basiert der Inhalt dieses Kapitels auf den Ergebnissen eines Kooperationsprojektes mit der ThyssenKrupp Steel Europe AG.

7.1. Problembeschreibung und scheduling-theoretische Analyse

Während des Herstellungsprozesses in einem Stahlwerk ist es notwendig, zu mehreren Zeitpunkten die chemische Zusammensetzung verschiedener End- und Zwischenprodukte zu überprüfen, um die erforderliche Qualität des produzierten Stahls zu gewährleisten. Zu diesem Zweck wurde bei der ThyssenKrupp Steel Europe AG im Jahr 2005 ein vollautomatisiertes Prozesslabor in Betrieb genommen. Darin werden Konverter[1]-, Stahl-, Roheisen- und Schlackeproben mit Hilfe verschiedener Maschinen bearbeitet und untersucht. Diese Proben werden vorab an den entsprechenden Positionen im Stahlwerk von einem Arbeiter entnommen und von diesem per Rohrpost an das Labor versendet. Dort angekommen, werden sie von Robotern in Empfang genommen, welche ebenfalls für den Transport der Proben zwischen den einzelnen Analysestationen zuständig sind. Die Dauer der Untersuchung im Labor muss an den Produktionsprozess angepasst sein und je nach Priorität einer Probe entsprechend „schnell“ vonstattengehen. Beispielsweise sollte die Analyse einer Konverterprobe höchstens drei Minuten beanspruchen, während für eine Roheisenprobe fünf und eine Schlackeprobe acht Minuten zur Verfügung stehen.

Wann die Proben im Werk entnommen werden, ist a priori nicht festgelegt. Dass eine Probe im Labor eintreffen wird, ist bekannt, sobald diese in das Rohrpostsystem übergeben wird. Ab diesem Zeitpunkt stehen sämtliche Informationen über eine Probe zur

[1] In einem Konverter wird durch Zuführung von Sauerstoff überschüssiger Kohlenstoff aus dem Roheisen herausoxidiert.

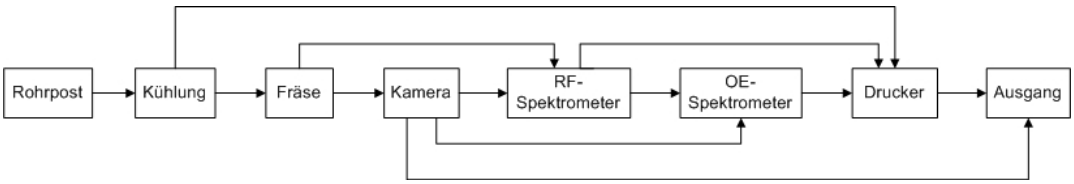

Abbildung 7.1.: Schematische Arbeitsgangfolge einer Stahlprobe

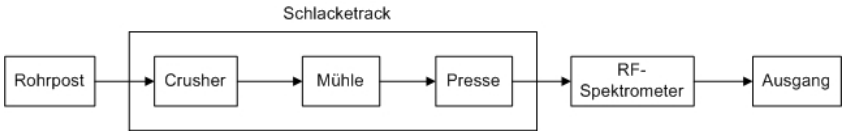

Abbildung 7.2.: Schematische Arbeitsgangfolge einer Schlackeprobe

Verfügung. Mit ihrer Eingabe in die Rohrpost wird dem Labor ein elektronisches *Worksheet* übermittelt, welches die durchzuführende Liste von Arbeitsgängen sowie die Priorität der Probe beinhaltet. Der Transport ins Labor kann wenige Sekunden oder bis zu einer Minute dauern, je nachdem an welcher Position im Werk diese entnommen wurde. Die Art der vorgegebenen Arbeitsgänge legt dabei in jedem Fall eindeutig die Reihenfolge fest, in welcher diese abgearbeitet werden müssen. Auch wenn sich die Arbeitsschritte für verschiedene Proben unterscheiden können, durchlaufen diese die Stationen im Labor sämtlich in derselben Reihenfolge, es werden lediglich manche Arbeitsgänge übersprungen. In den Abbildungen 7.1 und 7.2 ist exemplarisch die schematische Arbeitsgangfolge einer Stahl- bzw. einer Schlackeprobe ohne Berücksichtigung sämtlicher Feinheiten dargestellt (für eine detailliertere Beschreibung vgl. Witte, 2008, [153]). Dies zeigt, dass das Labor insgesamt als Flow-Shop strukturiert ist, in welchem jeder Job bestimmte Bearbeitungsstufen überspringt.

Abbildung 7.3 skizziert den Aufbau des Prozesslabors. Der linke vertikale Abschnitt wird als *Rohrposttrack* bezeichnet, da er die Rohrposteingänge enthält. Außerdem sind darin drei Kühlstationen (unten links) angesiedelt. Zwei Roboter sorgen für den Transport der Proben. Diese können sich innerhalb des Tracks weitestgehend unabhängig voneinander bewegen, d. h. sie können zu jeder Zeit aneinander vorbeifahren, müssen dabei aber gegebenenfalls eine Sicherheitsposition einnehmen.[2] Keiner der Roboter kann seinen Track jedoch verlassen. Der rechte vertikale Abschnitt zeigt den sogenannten *Schlacketrack*. Hier werden alle schlackespezifischen Arbeitsschritte durchgeführt. Die enthaltenen Crusher, Mühlen und Pressen (jeweils zwei) werden von einem einzelnen Roboter bedient. Das Herzstück der Anlage ist der horizontal dargestellte *Haupttrack*. Dieser beinhaltet wie der Rohrposttrack zwei Roboter, die neben der Bedienung der vorhandenen Fräsen und Spektrometer die Schlackeproben aus dem Rohrposttrack in den Schlacketrack

[2]Bei zwei Robotern im selben Track ist einer am Boden und der andere an der Decke angebracht, so dass sie in diesem Fall ebenfalls aneinander vorbeifahren können.

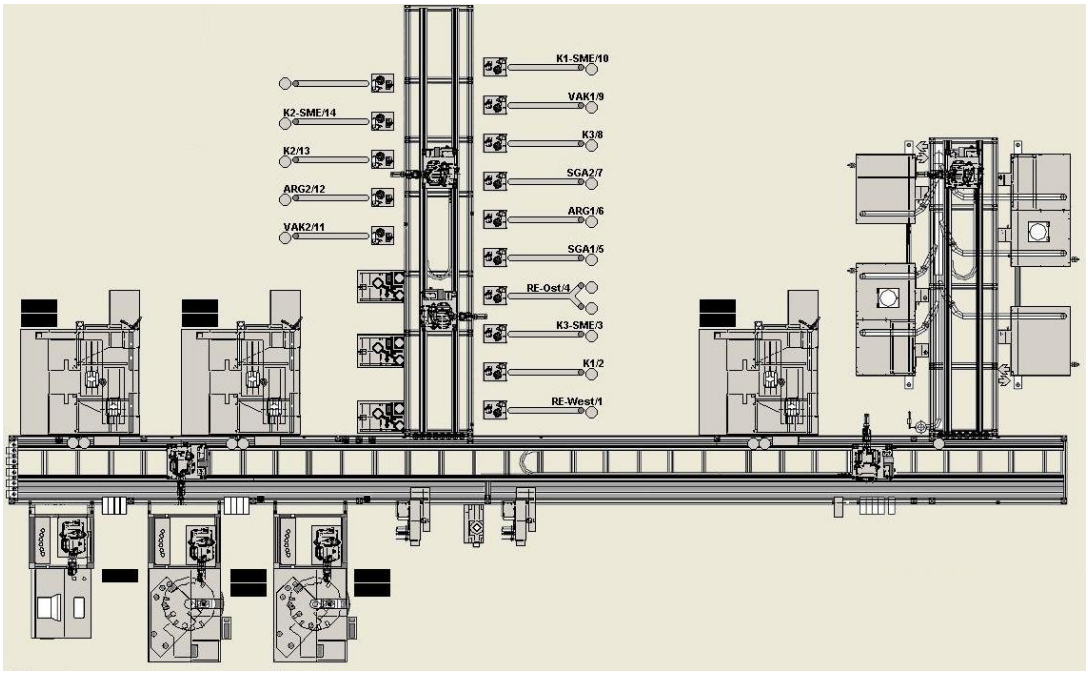

Abbildung 7.3.: Schematische Darstellung des Prozesslabors

transportieren sowie die Proben nach der Analyse aus dem System (z. B. in das Archiv) ausschleusen. Insgesamt sind im Labor im Wesentlichen die folgenden Maschinen bzw. Positionen vorhanden:

- 15 Rohrposteingänge
- 3 Kühlstationen (Rohrposttrack, links)
- 3 Fräsen und Kameras (Haupttrack, oben)
- 1 OE-Spektrometer (Haupttrack, unten links)
- 2 TEAMWORKS (OE- und XRF-Spektrometer in einer Bearbeitungseinheit, Haupttrack, unten)
- 2 Label-Drucker (Haupttrack, unten mittig)
- jeweils 2 Crusher, Mühlen, Pressen (Schlacketrack)

Da jeweils mehrere parallele Maschinen eines Typs zur Verfügung stehen, kann die Struktur des Labors als Hybrid Flow-Shop mit Transporten interpretiert werden. Die vorgegebenen Arbeitsgangfolgen erfordern ferner die Bearbeitung von Proben auf Maschinen, die sich innerhalb verschiedener Tracks befinden. Um den Wechsel zwischen den Tracks zu organisieren, sind Übergabepositionen (24 zwischen Rohrpost- und Haupttrack sowie acht zwischen Haupt- und Schlacketrack) vorhanden, an denen ein Roboter des einen

Tracks eine Probe ablegt und diese von einem Roboter des anderen Tracks wieder aufgenommen wird. Diese Positionen können als weitere Bearbeitungsstufe modelliert werden, wobei sämtliche Jobs, für die eine Übergabe notwendig ist, darauf die kleinste mögliche positive Bearbeitungszeit (hier eine Sekunde) besitzen. Dies ist erforderlich, da gleichzeitiges Ablegen durch einen und Aufnehmen durch einen anderen Roboter (bzw. eine direkte Übergabe von Roboter zu Roboter) technisch nicht möglich ist.

Zwischen den einzelnen Arbeitsgängen besteht im Labor für die Proben nur eine begrenzte Zwischenlagermöglichkeit. Es sind zwar wenige Parkpositionen (am linken Ende des Haupttracks) vorhanden, jedoch ist ein zusätzlicher Transport notwendig, um eine Probe dort abzulegen, sowie ein weiterer, um diese dort wieder abzuholen. Deshalb sollte eine Zwischenspeicherung der Proben nur im Notfall erfolgen. Aus diesem Grund werden im weiteren Verlauf die Parkpositionen bei der Planung nicht mit einbezogen und lediglich die Lagerung von Jobs auf Maschinen im Anschluss an die Bearbeitung zugelassen (Blocking-Constraints). Für detailliertere Informationen über die Abläufe im Labor verweisen wir an dieser Stelle auf Witte (2008) [153].

Das Ziel bei der Ablaufplanung des Labors ist es, die durchschnittliche gewichtete Durchlaufzeit der Proben im Vergleich mit dem derzeitigen Planungssystem zu verringern. Dementsprechend beinhaltet das Prozesslabor Optimierungspotential in zweierlei Hinsicht. Zum Einen stellt sich die Frage, in welcher Reihenfolge die eingehenden Proben abgearbeitet werden sollen und für welche Probe welche Maschine zum Einsatz kommt. Die meisten Maschinen sind mehrfach vorhanden, unterscheiden sich aber im erforderlichen Anfahrtsweg (Sequencing-Problem). Zum Anderen müssen die Transporte der Proben durch die Roboter organisiert werden, d. h. es ist zu entscheiden, welcher Roboter welche Transportaufgaben übernimmt (Routing-Problem). Der in Abschnitt 5.1 vorgestellte Dekompositionsansatz ist mithin auch als Grundlage für die Ablaufoptimierung des Prozesslabors geeignet. Man könnte die Roboter zwar (ähnlich wie bei der Schrankenbildung in Kapitel 4) als weitere Bearbeitungsstufe interpretieren, dies ist jedoch wenig sinnvoll, da die Roboter vollkommen anderen Restriktionen unterliegen als die Analysemaschinen. Im Folgenden diskutieren wir nun die Einzelheiten der beiden Teilprobleme am Beispiel des Prozesslabors sowie die Anforderungen an ein entsprechendes Lösungsverfahren.

7.1.1. Das Sequencing-Problem

Die angestrebte Optimierung der Abläufe ist zeitbezogen, d. h. die Proben sollen das Prozesslabor möglichst schnell durchlaufen. Jeder Probe wird bei der Übergabe in das Rohrpostsystem eine individuelle Priorität zugeordnet, welche wir mit Hilfe von Jobgewichten abbilden. Wir verwenden also grundsätzlich die Zielfunktion $\sum w_j F_j$. Im Fall einer dynamischen Optimierung, die in jedem Schritt nur bereits eingetroffene Proben einbezieht, entspricht dies bei jeder Reoptimierung der Minimierung der Funktion $\sum w_j C_j$. Wie in

Kapitel 5 bereits diskutiert, ist bei der Lösung des Sequencing-Problems die Berücksichtigung von No-Wait-Constraints sinnvoll, falls das gesamte Problem Blocking-Constraints unterliegt. Darüber hinaus ist die Ankunft einer Probe im Prozesslabor streng genommen nicht erst bei ihrem tatsächlichen Eintreffen bekannt, sondern bereits bei ihrer Übergabe in das Rohrpostsystem. Die dazwischen liegende Transportzeit kann bereits zur Ablaufplanung für die entsprechende Probe genutzt werden. Dabei muss dann aber eine (geringe) Freigabezeit ($r_j > 0$) berücksichtigt werden.[3] Insgesamt kann das Sequencing-Problem im Fall des Prozesslabors als $FH, (PM^{(k)})_{k=1}^{m} | (r_j), no-wait | \sum w_j C_j$ klassifiziert werden.

Die parallelen Maschinen eines Typs sind im Prozesslabor eigentlich identisch parallel, dennoch ergeben sich in Abhängigkeit ihrer Position „Qualitätsunterschiede“. Diese müssen bei der konkreten Maschinenauswahl für die Proben berücksichtigt werden. Dabei ist es jedoch wenig sinnvoll bereits die tatsächlich entstehenden Transportzeiten einzubeziehen, da die Maschinenbelegung im Rahmen des Sequencing-Problems gar nicht auf den tatsächlichen Start- und Fertigstellungszeiten der einzelnen Operationen basiert (an dieser Stelle wurden noch keine Transportzeiten berücksichtigt). Mithin wäre es unverhältnismäßig, bereits an dieser Stelle enormen Aufwand in eine möglichst günstige Maschinenauswahl zu investieren, obwohl noch gar nicht klar ist, ob diese sich auch nach erfolgter Routenplanung als beste Wahl erweisen wird.

7.1.2. Das Routing-Problem

Wie in Abschnitt 5.1 bereits beschrieben, werden die vorhandenen Transportmittel (hier also die Menge der Roboter) als identisch parallele Maschinen interpretiert. Grundsätzlich ist diese Vorgehensweise auch für das Prozesslabor zutreffend, da sämtliche Roboter dieselbe Geschwindigkeit besitzen. Allerdings kann nicht jeder Transport von jedem Roboter durchgeführt werden, da diese an ihren Track gebunden sind. Bei der Planung eines Auftrags muss also zusätzlich berücksichtigt werden, ob ein Roboter diesen überhaupt durchführen kann. Wir notieren diese Einschränkung im β-Feld des Klassifikationsschemas mit dem Symbol M_j, wobei dieses die Menge der Maschinen bezeichnet, welche zur Bearbeitung des Jobs j verwendet werden können.

Jeder Transportauftrag besitzt zudem einen individuellen Start- und Zielort. Bearbeitet ein Roboter also nacheinander zwei Transportaufträge j und k, muss er sich zunächst vom Zielort $d(j)$ des ersten zum Startort $o(k)$ des zweiten bewegen. Es sind demnach auch hier sequenz-abhängige Rüstzeiten zu berücksichtigen.

Aus der Lösung des Sequencing-Problems resultieren Jobfolgen für jede Maschine. Zusammen mit den zugrundeliegenden Arbeitsgangfolgen der Jobs ergeben sich so wie bereits beschrieben Reihenfolgebedingungen für das Routing-Problem, welche man in Form

[3] In gewissem Sinn kann man hier also analog zu der Strategie mit verzögerten Freigabezeiten (vgl. Abschnitt 3.3.3) arbeiten.

eines gerichteten Graphen modellieren kann. Es kommt im Fall des Prozesslabors jedoch erschwerend hinzu, dass eine direkt aufeinander folgende Bearbeitung von Transportaufträgen $j \to k$ mit $d(j) = o(k)$ durch denselben Roboter physikalisch nicht realisierbar ist. Gerade Reihenfolgen dieser Gestalt erscheinen zur Minimierung von Leerfahrten zwar naheliegend, die verschiedenen Analysestationen im Labor besitzen jedoch zumeist nur eine Übergabeposition für Proben. Genauso kann jeder Roboter auch zu einem Zeitpunkt nur eine Probe transportieren. Demzufolge ist es für diesen nicht möglich, eine Probe an einer Maschine abzuliefern und gleichzeitig eine fertig bearbeitete aufzunehmen. Durch die Berücksichtigung schwacher Kanten im Graphen (mit der Bedingung $C_k \geq s_j + 1$ anstelle von $C_k \geq s_j$) ist es jedoch möglich, dieser Schwierigkeit Rechnung zu tragen.

Neben Reihenfolgebedingungen und Rüstzeiten müssen natürlich auch für das Routing-Problem die Freigabezeiten der Proben berücksichtigt werden. Außerdem können zwei Transportaufträge, die zur selben Probe gehören, nicht direkt nacheinander ausgeführt werden, sondern es muss abgewartet werden, bis die Probe auf der jeweiligen Analysestation bearbeitet wurde und erst anschließend kann sie weiter transportiert werden (Verzögerungszeiten, starke Kanten des Reihenfolgegraphen).

Da jeder Transport eindeutig einer Probe zugeordnet ist, kann man gemäß ihrer Priorität auch eine Gewichtung des Transportauftrags vornehmen. Aus diesem Grund betrachten wir auch für das Routing-Problem eine Zielfunktion der Gestalt $\sum w_j C_j$, wobei hier im Wesentlichen nur der Transport einer Probe zum Systemausgang entscheidend ist. Man kann die Verplanung der Roboter also als $Pm|sds, prec, r_j, delay, M_j|\sum w_j C_j$-Problem klassifizieren. Da die Roboter schienengebunden sind und sich daher nur linear bewegen können, ergeben sich die sequenz-abhängigen Rüstzeiten als

$$s_{jk} = \frac{|d(j) - o(k)|}{v_R},$$

wobei v_R die Geschwindigkeit aller Roboter (bzw. die Maschinengeschwindigkeit) bezeichnet.

7.1.3. Anforderungen an ein Lösungsverfahren

Bei durchschnittlicher Auslastung des Prozesslabors ist mit 400 - 600 eingehenden Proben pro Tag zu rechnen. Ferner arbeitet das Labor 24 Stunden täglich an sieben Tagen in der Woche. Da die Jobs dort dynamisch eintreffen, ist ein echtzeitfähiges Planungssystem erforderlich. Durch den Projektpartner ist in diesem Zusammenhang eine maximale Berechnungsdauer von drei Sekunden vorgegeben, wenn möglich weniger.[4] Die zur Verfügung stehende Rechenzeit ist also erheblich beschränkt.

Grundsätzlich ist es natürlich denkbar, auch statistische Beobachtungen über Probenhäufigkeit und Ankunftszeiten in die Planung einzubeziehen. Im Vordergrund bei der

[4] Diese Angabe bezieht sich auf einen wenige Jahre alten Standard PC mit Pentium IV 2,8 GHz Prozessor und 1 GB RAM.

Optimierung steht hier jedoch eine kurzfristige und ereignisbezogene Steuerung des Labors und vor diesem Hintergrund erscheint ein deterministisches Verfahren, welches nur bereits bekannte Proben berücksichtigt, als geeigneter. Sobald eine neue Probe eintrifft, muss diese nachträglich in die Optimierung einbezogen werden.

Neben den vorab nicht bekannten Ankunftszeiten der Jobs spielen bei der Planung des Labors weitere Unsicherheitsfaktoren eine Rolle. So ist der plötzliche Ausfall einer Maschine genauso möglich, wie die Situation, dass ein Arbeitsgang nicht erfolgreich war und aus diesem Grund wiederholt werden muss.[5] Letzteres kann z. B. beim Fräsen einer Probe auftreten. Bei der OE-Spektroskopie ist dies sogar die Regel, da eine Probe für gewöhnlich mehrfach angefunkt werden muss, bis die entsprechende Analyse abgeschlossen ist. Zumeist wird diese Prozedur zwei- bis dreimal wiederholt. Im Vorhinein ist jedoch nicht bekannt, wie viele Funkvorgänge für eine Probe tatsächlich durchgeführt werden müssen. Des Weiteren können Proben nicht nur über die Rohrposteingänge in das Labor geschleust werden, sondern alternativ auch über manuelle Eingabepositionen dorthin gelangen oder sogar direkt von außen in eine Maschine eingesetzt werden (beispielsweise zu deren Rekalibrierung). Falls das Labor jedoch mit Hilfe eines schnellen ereignisbezogenen Entscheidungssystems gesteuert wird, können solche Vorkommnisse bei ihrem Eintreten genau wie das Eintreffen neuer Proben durch eine entsprechende Reoptimierung des gesamten Ablaufplans einbezogen werden (vgl. dazu auch Wittrock, 1988, [154]).

7.2. Anpassung des Lösungsverfahrens

Die vorangehenden Ausführungen belegen, dass das in Kapitel 5 entwickelte Lösungsverfahren (Phasen 1 - 3) grundsätzlich zur Steuerung des Prozesslabors geeignet ist. Es sind lediglich einige geringfügige Modifikationen erforderlich, um alle relevanten Aspekte dieses Anwendungsproblems erfassen zu können. Die notwendigen Veränderungen und Verfeinerungen werden im Folgenden erläutert.

Um für das Routing-Problem die zusätzliche Restriktion, dass jede Transportaufgabe nur von einer bestimmten Menge von Robotern bedient werden kann, abzubilden, kennzeichnen wir sämtliche Maschinen zusätzlich mit der Bezeichnung des Tracks, in welchem sie angesiedelt sind. Aus der Angabe, ich welchem Track sich die einzelnen Roboter befinden, ergibt sich demnach die entsprechende Teilmenge M_j, die zur Durchführung des Auftrags j verwendet werden kann. Ansonsten unterscheidet sich der Ablauf der dritten Phase des Verfahrens nicht von der Beschreibung in Kapitel 5. Bei der Berechnung der Prioritätsindizes werden lediglich nur die zulässigen Roboter anstelle sämtlicher Transportmittel berücksichtigt.

[5] Laut Ribas et al. (2010) [116] wurde eine solche Situation im Zusammenhang mit Hybrid Flow-Shop-Problemen in der Literatur bislang noch nicht untersucht.

Die Übergabe von Proben zwischen verschiedenen Tracks modellieren wir wie oben bereits angedeutet als Arbeitsgang der Länge 1 auf einer zusätzlichen Bearbeitungsstufe mit 24 bzw. acht parallelen Maschinen. Diese müssen jedoch anders als die übrigen Maschinen jeweils zwei verschiedenen Tracks zugeordnet werden.

Im Prozesslabor werden die Proben nicht auf sämtlichen vorhandenen Maschinentypen bearbeitet, sondern es können Stufen übersprungen werden. Diese Situation kann durch eine Bearbeitungszeit der Länge Null ($p_{ij} = 0$) modelliert werden, muss aber natürlich bei der Generierung der Transportaufträge gesondert behandelt werden. Bei der Lösung des Sequencing-Problems können solche Operationen ohne Schwierigkeit wie alle anderen betrachtet werden. Wie man dies geeignet in die Formulierung des Algorithmus' einbeziehen kann, zeigt die vollständige formelle Beschreibung des modifizierten Verfahrens in Anhang A.

Grundsätzlich wird die Priorität einer Probe durch das mitgelieferte Worksheet festgelegt. Nun kann prinzipiell aber die Situation auftreten, dass nach einer Probe mit geringem Gewicht, viele Proben mit höherer Priorität im Labor eintreffen. In diesem Fall muss gewährleistet werden, dass die erste Probe trotzdem zeitnah bearbeitet wird und nicht liegen bleibt bis sämtliche „wichtigeren“ Proben untersucht wurden. Dies kann man mit Hilfe von Jobgewichten realisieren, die zunehmen, je länger eine Probe auf ihre Bearbeitung wartet.

7.3. Simulation für das Prozesslabor

Um die Arbeitsweise des angepassten Verfahrens für das Prozesslabor zu simulieren, haben wir anhand aufgezeichneter Ablaufprotokolle verschiedene Testdatensätze generiert. Jeweils über den Zeitraum einer Stunde werden sämtliche eingegangenen Proben mit der entsprechenden Ankunftszeit sowie den jeweils durchgeführten Arbeitsschritten als Eingabedaten für das Verfahren verwendet. Nach genauerer Beschreibung dieses Datenmaterials im nachfolgenden Abschnitt stellen wir zunächst die verwendete Modellierung des Prozesslabors vor, bevor wir in Abschnitt 7.3.3 die diesbezüglich erzielten Ergebnisse mit den tatsächlichen Abläufen im Labor vergleichen.

7.3.1. Verwendetes Datenmaterial

Zur Generierung der Datensätze wurde das Probenaufkommen über einen Zeitraum von 24 Stunden (7. Dezember 2009) untersucht. Aus den diesbezüglich protokollierten Abläufen ergeben sich 24 Datensätze, die jeweils den eingegangenen Proben einer Stunde entsprechen. Dabei werden manuell eingesetzte Proben, Proben zur Rekalibrierung der Spektrometer sowie sogenannte Doppelproben, die lediglich aus dem System ausgeschleust werden müssen, ebenfalls als Probeneingänge interpretiert. Die Prioritäten der Proben wurden aus den zugehörigen Worksheets übernommen. Für jeden der Datensätze

wird das angepasste dynamische Verfahren (Phasen 1 - 3) durchgeführt, wobei Proben, die zum Zeitpunkt einer Iteration noch nicht der Rohrpost übergeben bzw. eingetroffen sind, im Verfahren natürlich auch noch nicht berücksichtigt werden.

7.3.2. Modellierung des Prozesslabors

Um die Arbeitsweise des Verfahrens für die Echtdaten zu simulieren, haben wir die Abläufe im Prozesslabor zunächst etwas vereinfacht modelliert. So wird für alle Arbeitsschritte mit den mittleren Laufzeiten[6] der verschiedenen Programme eines Maschinentyps gearbeitet (vgl. Tabelle 7.1), die wir zudem auf volle Sekunden runden. Dazu verwenden wir die entsprechenden Werte aus der Arbeit von Witte (2008) [153]. Insbesondere für die Bearbeitung einer Probe im OE-Spektrometer wird von durchschnittlich drei Funkvorgängen (Programm 3) ausgegangen. Auch bei der Verplanung der Roboter werden lediglich mittlere Transportzeiten berücksichtigt.

Darüber hinaus werden auch die Abläufe innerhalb der Maschinen (z. B. die Behandlung einer Probe durch einen internen Roboter in TEAMWORKS) zu einem einzigen Arbeitsschritt zusammengefasst. Insbesondere wird bei der Simulation der konkrete Ablauf innerhalb der Schlackelinie nicht explizit geplant. Schlackeproben stellen den kleineren Anteil im Probenaufkommen dar und haben zudem geringere Priorität als die übrigen Proben. Aus diesem Grund wird die Schlackelinie als eine „Bearbeitungsstufe“ mit einer Kapazität von zwei Maschinen interpretiert, die eine durchschnittliche Bearbeitungszeit von 400 Sekunden aufweist. Dabei sind gegebenenfalls notwendige Spülproben[7] nicht berücksichtigt. Insbesondere ist es auf diese Weise nicht erforderlich, die Übergabe von Proben zwischen Haupt- und Schlacketrack zu modellieren. Eine vollständige Liste der verwendeten durchschnittlichen Bearbeitungszeiten sind Tabelle 7.1 zu entnehmen (basierend auf Witte, 2008, [153]).

In Abhängigkeit der tatsächlichen Positionen der Maschinen im Labor sowie der durchschnittlichen Transportzeiten zwischen zwei Positionen innerhalb eines Tracks (vgl. Witte, 2008, [153]) wurden die Maschinenstandorte zur Simulation so gewählt, dass die Entfernung zwischen zwei Positionen in Metern ungefähr der dazwischen notwendigen Transportzeit (inklusive Aufnehmen und Ablegen der Probe) in Sekunden entspricht. Dadurch kann die Robotergeschwindigkeit (v_R) bei der Berechnung gleich 1 gesetzt werden. Die verwendeten Maschinenstandorte bzw. Systemeingänge und -ausgänge in Rohrpost- und Haupttrack können den Tabellen 7.2 und 7.3 entnommen werden. Darin ist zudem der gewählte Wert für die Güteklassen der verschiedenen Maschinen einer Stufe angegeben. Zur Verarbeitung dieser Daten in der Implementierung des Verfahrens muss formal auch

[6] Rein theoretisch hat ein Programm auf einer Bearbeitungsstufe immer dieselbe Laufzeit. In der Praxis treten aber geringfügige Schwankungen auf.

[7] Eine Spülprobe ist notwendig, um eine Maschine der Schlackelinie zu „reinigen“, bevor eine andersartige Probe bearbeitet wird. Sie entspricht im Wesentlichen also einer Rüstzeit der Maschine.

Maschinentyp	Programm	Laufzeit (s)	Näherung (s)
Kühlung (CUH)	1	26,8	27
	3	27,8	28
	6	24,2	24
	9	41,3	41
Fräse (VMC)	1	34,8	35
	3	28,4	28
	6	27,2	27
	9	58	58
	14	53,3	53
	99	7,2	7
Kamera (VISION)	1	6	6
OE-Spektrometer (OES)	1	25,5	26
	2	73,6	74
	3	115	115
	4	158,4	158
	5	206	206
	6	228,5	229
XRF-Spektrometer	1	68,5	69
Label-Printer (LPR)	1	6	6
Schlackelinie	1	400	400

Tabelle 7.1.: Durchschnittliche Laufzeiten der verschiedenen Arbeitsgänge

Maschine	Güteklasse	Position	genäherte Position
manuelle Eingänge (1-3)	1	6,5	6
manuelle Eingänge (4-6)	1	13,0	13
Speicher	1	5,6	6
Archiv	1	7,9	8
manuelle Ausgänge (1-5)	1	2,1	2
manuelle Ausgänge (6-9)	1	13,4	13
Übergabe Rohrposttrack	1	7,9	8
VMC1	1	4,0	4
VISION1	1	4,5	5
VMC2	3	10,7	10
VISION2	3	11,2	11
VMC3	2	13,7	13
VISION3	2	14,2	14
TEAMWORKS1	3	9,9	10
TEAMWORKS2	2	12,2	12
OES3	1	14,6	15
LPR1	1	7,3	7
LPR2	1	8,5	9
Schlackelinie1	1	1,1	1
Schlackelinie2	1	1,1	1

Tabelle 7.2.: Positionen im Haupttrack

Maschine	Güteklasse	genäherte Position
Rohrposteingang 1	1	6
Rohrposteingang 2	1	6
Rohrposteingang 3	1	6
Rohrposteingang 4	1	6
Rohrposteingang 5	1	6
Rohrposteingang 6	1	2
Rohrposteingang 7	1	1
Rohrposteingang 8	1	1
Rohrposteingang 9	1	0
Rohrposteingang 10	1	0
Rohrposteingang 11	1	2
Rohrposteingang 12	1	1
Rohrposteingang 13	1	1
Rohrposteingang 14	1	0
Rohrposteingang 15	1	0
CUH1	3	4
CUH2	2	4
CUH3	1	4
Übergabe Haupttrack	1	8

Tabelle 7.3.: Positionen im Rohrposttrack

den Ein- und Ausgängen eine Güteklasse zugewiesen werden, welche auf den Ablauf der Optimierung aber natürlich keinen Einfluss besitzt.

7.3.3. Ergebnisse

Die Ergebnisse für die 24 Datensätze sind zusammenfassend in den Tabellen 7.4 und 7.5 dokumentiert. Auf der linken Seite ist jeweils die Nummer des Datensatzes, die Anzahl der in dieser Zeitspanne eingegangenen Proben sowie der betrachtete Zeitraum angegeben. Der mittlere Block beider Tabellen bezieht sich auf die Zielfunktion $\sum w_j C_j$, d. h. die Summe der gewichteten Fertigstellungszeiten der Proben. Die Spalte WCT (Ist) enthält dabei den Wert dieser Funktion im tatsächlichen Ablauf. WCT (H) dagegen beinhaltet den Wert der Zielfunktion nach Simulation des Ablaufs mit Hilfe unseres Verfahrens. Die dritte Spalte des Blocks (Diff (%)) gibt die relative Abweichung der beiden Werte an, mit anderen Worten die potentielle Einsparung in Bezug auf das Zielkriterium $\sum w_j C_j$.

Die Fertigstellungszeit C_j einer Probe bezieht sich natürlich immer auf den Beginn der jeweiligen Planungsperiode und lässt keinen Rückschluss über den Zeitpunkt ihrer

Datensatz			gewichtete Fertigstellungszeit			gewichtete Durchlaufzeit		
Nr.	# Proben	Zeitraum	WCT (Ist)	WCT (H)	Diff (%)	WFT (Ist)	WFT (H)	Diff (%)
1	40	0-1 Uhr	740919	679267	8,32	116534	54882	52,90
2	44	1-2 Uhr	752335	695586	7,54	122017	65268	46,51
3	46	2-3 Uhr	762182	709526	6,91	112885	60229	46,65
4	41	3-4 Uhr	719644	667444	7,25	108104	55904	48,29
5	43	4-5 Uhr	751632	704550	6,26	104004	56922	45,27
6	37	5-6 Uhr	601387	564603	6,12	85531	48747	43,01
7	45	6-7Uhr	823305	761573	7,50	124329	62597	49,65
8	40	7-8Uhr	591574	549489	7,11	102905	60820	40,90
9	42	8-9 Uhr	864388	822821	4,81	98328	56761	42,27
10	38	9-10 Uhr	929765	881463	5,20	113394	65092	42,60
11	57	10-11 Uhr	652765	615147	5,76	103013	65395	36,52
12	41	11-12 Uhr	614298	550354	10,41	116428	52448	54,95
13	72	12-13 Uhr	959836	908626	5,34	121239	70029	42,24
14	54	13-14 Uhr	992417	947546	4,52	116446	71575	38,53
15	61	14-15 Uhr	864716	812428	6,05	117535	65247	44,49
16	46	15-16 Uhr	872717	828324	5,09	113417	69024	39,14
17	58	16-17 Uhr	861397	823445	4,41	112484	74532	33,74
18	43	17-18 Uhr	699517	666795	4,68	82795	50073	39,52
19	71	18-19 Uhr	798951	740855	7,27	134685	76589	43,13
20	66	19-20 Uhr	1246414	1192782	4,30	153366	99734	34,97
21	60	20-21 Uhr	834188	792235	5,03	98132	56179	42,75
22	44	21-22 Uhr	1077195	1028732	4,50	123488	75025	39,25
23	51	22-23 Uhr	990021	955576	3,48	98788	64343	34,87
24	39	23-0 Uhr	630679	591235	6,25	86725	47281	45,48
Mittlere Einsparung (%)					**6,00**			**42,82**

Tabelle 7.4.: Ist-Zustand vs. Simulation mit 3x OES (bzgl. Daten vom 7.12.2009)

Ankunft im Labor zu. Aufgrund der sekundengenauen Modellierung der Abläufe ergeben sich jedoch verhältnismäßig hohe Werte für die Freigabezeiten der Proben. Daher ist die gewichtete Gesamtfertigstellungszeit zur Auswertung des Verfahrens im Fall des Prozesslabors nur bedingt aussagekräftig. Viel bedeutsamer ist dagegen die Summe der gewichteten Durchlaufzeiten ($\sum w_j F_j$) der Proben. Die diesbezüglich erzielten Resultate können dem jeweils rechten Block der Tabellen entnommen werden. Die darin verwendeten Bezeichnungen sind analog zum mittleren Block gewählt. Die jeweils letzte Zeile einer Tabelle gibt ferner die durchschnittliche Abweichung der Zielfunktionswerte über alle 24 Datensätze an.

Bei den Berechnungen bezüglich Tabelle 7.4 wurden für jede Probe, welche eine OE-Spektroskopie erfordert, drei Funkvorgänge mit einer durchschnittlichen Gesamtbearbeitungszeit von 115 Sekunden veranschlagt. Dies entspricht natürlich nicht der tatsächlichen

Datensatz			gewichtete Fertigstellungszeit			gewichtete Durchlaufzeit		
Nr.	# Proben	Zeitraum	WCT (Ist)	WCT (H)	Diff (%)	WFT (Ist)	WFT (H)	Diff (%)
1	40	0-1 Uhr	740919	677414	8,57	116534	53029	54,49
2	44	1-2 Uhr	752335	683487	9,15	122017	53169	56,42
3	46	2-3 Uhr	762182	701569	7,95	112885	52272	53,69
4	41	3-4 Uhr	719644	653737	9,16	108104	42197	60,97
5	43	4-5 Uhr	751632	695106	7,52	104004	47478	54,35
6	37	5-6 Uhr	601387	555161	7,69	85531	39305	54,05
7	45	6-7 Uhr	823305	750302	8,87	124329	51326	58,72
8	40	7-8 Uhr	591574	543385	8,15	102905	54716	46,83
9	42	8-9 Uhr	864388	810067	6,28	98328	44007	55,24
10	38	9-10 Uhr	929765	870139	6,41	113394	53768	52,58
11	57	10-11 Uhr	652765	602463	7,71	103013	52711	48,83
12	41	11-12 Uhr	614298	543235	11,57	116428	45329	61,07
13	72	12-13 Uhr	959836	901399	6,09	121239	62802	48,20
14	54	13-14 Uhr	992417	941825	5,10	116446	65854	43,45
15	61	14-15 Uhr	864716	805086	6,90	117535	57905	50,73
16	46	15-16 Uhr	872717	812624	6,89	113417	53324	52,98
17	58	16-17 Uhr	861397	807244	6,29	112484	58331	48,14
18	43	17-18 Uhr	699517	657625	5,99	82795	40903	50,60
19	71	18-19 Uhr	798951	739750	7,41	134685	75484	43,96
20	66	19-20 Uhr	1246414	1170904	6,06	153366	77856	49,24
21	60	20-21 Uhr	834188	783237	6,11	98132	47181	51,92
22	44	21-22 Uhr	1077195	1017657	5,53	123488	63950	48,21
23	51	22-23 Uhr	990021	944186	4,63	98788	52953	46,40
24	39	23-0 Uhr	630679	585736	7,13	86725	41782	51,82
Mittlere Einsparung (%)					**7,21**			**51,79**

Tabelle 7.5.: Ist-Zustand vs. Simulation mit tats. OES (bzgl. Daten vom 7.12.2009)

Bearbeitungsdauer. Da aber vorab nicht bekannt ist, wie viele Funkvorgänge zur Analyse einer Probe erforderlich werden, muss bei der Planung von einem solchen Näherungswert ausgegangen werden. Um jedoch auch die Arbeitsweise des Verfahrens in Bezug auf die tatsächliche Anzahl von Funkvorgängen im OE-Spektrometer zu untersuchen, wurden alle Testläufe mit den entsprechenden Werten wiederholt, d. h. unter der Annahme, dass die notwendige Anzahl von Funkvorgängen für eine Probe bereits vorab bekannt wäre. Die Resultate dieser zweiten Testreihe sind in Tabelle 7.5 zusammengefasst.

Die Ergebnisse der Simulationen zeigen, dass bei der Planung des Prozesslabors in Bezug auf die gewichteten Durchlaufzeiten der Proben enormes Verbesserungspotential besteht. Unter der Annahme, dass für jede Probe, sofern OE-Spektroskopie erforderlich ist, genau drei Funkvorgänge durchgeführt werden müssen, weist die Simulation einen um durchschnittlich 42,82% geringeren Zielfunktionswert auf, als der tatsächlich erfolgte

Ablauf. Dieses Ergebnis ist auch für die Praxis realistisch, da in den meisten Fällen eher zwei als drei Funkvorgänge notwendig sind, bei der Simulation also im Schnitt von einer größeren Bearbeitungszeit auf dieser Maschinenstufe ausgegangen wurde. Die Annahme, die Anzahl der Funkvorgänge sei bereits im Voraus bekannt, ist zwar grundsätzlich nicht realistisch, die entsprechenden Resultate implizieren jedoch, dass auch beim Einsatz des Verfahrens in der Praxis eine erhebliche Einsparung möglich ist. Natürlich wurde bei der Simulation - nicht nur in Bezug auf die OE-Spektroskopie - lediglich mit Näherungswerten gearbeitet. Doch selbst wenn die entsprechenden Werte in der Realität größer werden sollten, ist weiterhin mit einer enormen Einsparung zu rechnen. Insgesamt konnte in der durchgeführten Simulation die Laufzeit jeder Probe verkürzt werden.

Auch die benötigte Rechenzeit wird den Anforderungen des Prozesslabors gerecht (vgl. Tabelle 7.6). In den ersten beiden Spalten ist hier erneut die Nummer des Datensatzes sowie die Anzahl der enthaltenen Proben angegeben. Der dritten Spalte kann außerdem die Gesamtanzahl zugehöriger Transportaufträge entnommen werden. Die im nächsten Block vermerkte Rechenzeit bezieht sich jeweils auf die Dauer einer Reoptimierung, wobei zuerst die durchschnittliche Länge eines Durchlaufs notiert ist, gefolgt von der Dauer des längsten Durchlaufs für diesen Datensatz (jeweils in ms). Die letzten beiden Spalten der Tabelle geben an, wie viele Proben bei der Resequenzierung der Jobs (in Phase 1) innerhalb eines Durchlaufs im Mittel bzw. höchstens berücksichtigt wurden. Die untersuchten Datensätze beziehen sich auf einen Zeitraum von 24 Stunden, in welchem die Auslastung des Labors sehr hoch war. Dies ist insofern bemerkenswert, als das bei der Resequenzierung nur in zwei Fällen überhaupt vier Proben gleichzeitig berücksichtigt wurden. Mit anderen Worten, trotz hoher Auslastung des Labors kommen nur zweimal bei insgesamt 1158 Reoptimierungsschritten die Prioritätsregeln zur Jobsequenzierung (in Phase 1) zum Einsatz. In den übrigen Durchläufen (mit höchsten drei Proben) werden dagegen sämtliche möglichen Jobsequenzen ausgewertet. Die Wahl der Sequenzierungsregeln hat demnach im Fall des Prozesslabors nahezu keinen Einfluss auf die Performance des Verfahrens. Trotz der geringen Probenzahl darf die Komplexität einer Reoptimierung nicht unterschätzt werden. Für viele Proben, die zwar nicht resequenziert werden, müssen dennoch viele Transportaufträge und Arbeitsgänge bei der Erstellung des neuen Schedules berücksichtigt werden (vgl. dazu auch Anhang A). Hier ergibt sich also eine wesentlich höhere Anzahl wiederholt zu verplanender Operationen.

Die enorme Abweichung von 1179 Proben innerhalb des betrachteten Zeitraums gegenüber den vorangehend erwarteten 400 - 600 Proben pro Tag lässt sich dadurch erklären, dass zur Simulation auch jede manuell eingesetzte Probe als eingehende Probe interpretiert wurde. Das angenommene Probenaufkommen bezieht sich aber lediglich auf die tatsächlich aus dem Werk (per Rohrpost) gesendeten Proben. Für die Planung müssen aber natürlich alle zu bearbeitenden Aufträge berücksichtigt werden, unabhängig davon, auf welchem Weg diese das Labor erreichen.

Datensatz Nr.	Anzahl Proben	Anzahl Transporte	Mittlere Laufzeit (ms)	Maximale Laufzeit (ms)	Mittlere Probenzahl	Maximale Probenzahl
1	40	198	459	1235	1,50	3
2	44	231	632	1516	1,35	4
3	46	233	624	1437	1,24	3
4	41	215	543	1281	1,32	2
5	43	228	620	1531	1,19	2
6	37	202	507	1172	1,32	3
7	45	240	645	1438	1,16	3
8	40	215	665	1578	1,10	2
9	42	217	571	1563	1,10	2
10	38	208	503	1141	1,13	2
11	57	226	783	1734	1,36	3
12	41	202	528	1407	1,15	2
13	72	287	823	2578	1,14	3
14	54	246	742	1719	1,24	3
15	61	253	665	2125	1,12	2
16	46	221	589	1437	1,22	2
17	58	271	948	2625	1,35	4
18	43	221	531	1563	1,05	2
19	71	292	966	2343	1,17	3
20	66	305	982	2875	1,31	3
21	60	280	909	3250	1,38	3
22	44	231	641	1750	1,20	3
23	51	238	571	1781	1,10	2
24	39	202	528	1391	1,15	2
Gesamt	**1179**	**5662**	**694**	**3250**	**1,22**	**4**

Tabelle 7.6.: Rechenzeiten (bzgl. Daten vom 7.12.2009)

In den durchgeführten Testläufen wurde keine wartezeitbezogene Anpassung der Probengewichte vorgenommen. Dies ist aufgrund des beschränkten Planungshorizonts von jeweils einer Stunde nicht erforderlich, zumal die Prioritäten der eingehenden Proben produktionsbedingt variieren. Insbesondere treffen in den betrachteten Datensätzen nie viele aufeinander folgende Proben von besonders hoher Priorität ein. Will man die Abläufe im Prozesslabor nun aber kontinuierlich und nicht nur für den Zeitraum einer Stunde planen, wird eine Prioritätsteigerung mit wachsender Wartezeit erforderlich, um die zeitnahe Bearbeitung von Proben geringer Ausgangspriorität zu gewährleisten (vgl. Anhang A).

8. Abschließende Bemerkungen und Ausblick

Ziel dieser Arbeit war die Entwicklung eines echtzeitfähigen heuristischen Verfahrens zur Lösung von (dynamischen) Hybrid Flow-Shop-Problemen mit Minimierung der gewichteten Gesamtfertigstellungszeit bzw. der gewichteten Gesamtflusszeit unter Berücksichtigung verschiedener zusätzlicher Restriktionen. Die Performance dieses Verfahrens sollte im Rahmen einer umfassenden empirischen Untersuchung sowohl auf theoretischer Ebene als auch im Kontext eines konkreten Anwendungsproblems in der Stahlindustrie, der Ablaufsteuerung eines sogenannten Prozesslabors, untersucht werden.

Basierend auf einem Dekompositionsansatz wurde daher in Kapitel 5 eine dreistufige Heuristik entwickelt, welche sich zur Lösung von Hybrid Flow-Shop-Problemen mit No-Wait-Constraints und den oben genannten Zielfunktionen genauso eignet wie zur Behandlung eines solchen Problems unter Berücksichtigung von Blocking-Constraints und Transportrestriktionen. Grundsätzlich ist die Methode zwar zur Lösung des statischen Problems konzipiert, aufgrund der extrem geringen Rechenzeit kann sie jedoch auch im Fall einer Situation mit dynamisch eintreffenden Jobs eingesetzt werden. Sobald ein neuer Job freigegeben wird oder ein anderes unvorhergesehenes Ereignis, wie der Ausfall einer Maschine, eintritt, kann der gesamte Schedule unter erneuter Ausführung des Verfahrens reoptimiert werden.

Die Ergebnisse der empirischen Untersuchung in Kapitel 6 sind vielversprechend. Für die Probleme $FHm,(PM^{(k)})_{k=1}^{m}|r_j|\sum w_jC_j$ und $FHm,(PM^{(k)})_{k=1}^{m}|r_j,no-wait|\sum w_jC_j$ wurden jeweils drei Testreihen mit verschiedenen Rahmenbedingungen durchgeführt. Als Benchmark für die approximativen Lösungen kamen dabei die in Kapitel 4 diskutierten LP-basierten unteren Schranken $(HFSb)$ und $(HFSc)$ zum Einsatz. Bei unbegrenztem Zwischenspeicher und dynamisch eintreffenden Jobs liegt der mittlere relative Fehler im Vergleich mit der ersten Schranke je nach Szenario bei 5 bis 15% und im statischen Fall sinken diese Werte sogar auf 3,8 bis 9,1%. Nur geringfügig schlechtere Resultate liefert die Methode unter No-Wait-Constraints. Zur Untersuchung des dynamischen Problems mit Transportrestriktionen wurden die entsprechend modifizierten Varianten der beiden Schranken (vgl. Abschnitt 4.3.2) verwendet. Für die diesbezüglich betrachteten Szenarien ergibt sich im Vergleich mit der ersten Variante ein durchschnittlicher relativer Fehler von 11 bis 19%, je nach Anzahl zur Verfügung stehender Transportmittel.

Bei der Simulation des Verfahrens als neues Steuerungssystem für das Prozesslabor konnte ein enormes Einsparpotential gegenüber der derzeitigen Ablaufplanung prognostiziert werden. So wurde in den diesbezüglich generierten Schedules die Durchlaufzeit jeder einzelnen Probe erheblich verkürzt. Insgesamt konnte im Vergleich zum Ist-Zustand

des Labors sogar eine durchschnittliche Verringerung der gewichteten Probenlaufzeiten von mehr als 40% erzielt werden.

Die als Benchmark verwendeten unteren Schranken basieren auf der LP-Relaxation zweier zeit-indizierter gemischt-ganzzahliger Formulierungen des Problems. In Kapitel 4 konnte zum Einen gezeigt werden, dass die erste Schranke in jedem Fall von der zweiten dominiert wird. Zum Anderen wurde nachgewiesen, dass beide Modelle stärkere untere Schranken liefern, als die LP-Relaxation einer in der Literatur gebräuchlichen Formulierung mit Fertigstellungszeitvariablen. Mithin gelten diesbezüglich bekannte Worst Case Resultate genauso für die hier verwendeten Schranken.

In Kapitel 6 wurde die Abweichung der beiden verwendeten Schranken voneinander zwar auch in der Praxis evaluiert, zusätzlich wäre aber eine weitergehende empirische Analyse der Schranken im Verhältnis zur optimalen Lösung der betrachteten Probleminstanzen von Interesse. Die in Kapitel 4 zitierte Performance Schranke hängt von der Anzahl der Bearbeitungsstufen des jeweiligen Hybrid Flow-Shop-Problems ab und kann daher je nach Szenario nur bedingt Aufschluss über die tatsächliche Qualität der Schranken geben. Die Bestimmung optimaler Lösungen für sämtliche der hier untersuchten 1500 Probleminstanzen hätte jedoch den Rahmen dieser Arbeit gesprengt, da der Fokus mehr auf der Performance-Analyse des entwickelten Lösungsverfahrens liegt. Zu diesem Zweck ist die Verwendung der unteren Schranken als Benchmark vollkommen ausreichend.

Genauso wäre auch die Untersuchung weiterer Problemkonstellationen mit anderen als den in den Datensätzen 1 - 6 betrachteten Rahmenbedingungen von Interesse. Eine solche Evaluation ist vor allem dann sinnvoll, wenn ein konkreter Anwendungsbezug vorliegt, da ansonsten unbegrenzt viele Möglichkeiten zur Generierung von Probleminstanzen berücksichtigt werden müssten. Die in der vorliegenden Dissertation analysierten Datensätze wurden daher zum größten Teil in Anlehnung an die Gegebenheiten des Prozesslabors erzeugt. Darüber hinaus erfolgt eine Auswertung anhand von Echtdaten. Insgesamt sind die gewählten Testinstanzen dem Kontext dieser Arbeit daher angemessen, im Hinblick auf andere praktische Problemstellungen können sie aber ausgeweitet werden. Insbesondere ist das entwickelte Verfahren so allgemein gehalten, dass es leicht zur Lösung ähnlicher Situationen adaptiert werden kann.

Bei der Konstruktion der heuristischen Methode in Kapitel 5 lag der Fokus vornehmlich auf der Behandlung dynamischer Probleme bzw. auf der Echtzeitsteuerung des Prozesslabors, grundsätzlich basiert sie jedoch auf einem Algorithmus zur Lösung des statischen Problems. Bei der Auswertung in Kapitel 6 wurde daher auch die Performance des Verfahrens im statischen Fall untersucht. Im Zuge einer Vertiefung der Ergebnisse dieser Arbeit wäre es daher von Interesse, die Qualität statischer Lösungen weiter zu verbessern. Anders als bei der Steuerung des Prozesslabors ist in einer entsprechenden Situation in der Praxis keine Beschränkung der Rechenzeit auf weniger als drei Sekunden zu erwar-

ten. Schon bei geringfügiger Verlängerung der möglichen Berechnungsdauer wäre eine Verbesserung der Lösungsgüte vorstellbar. Hierzu könnte die heuristische Lösung beispielsweise als Ausgangspunkt für die Konstruktion eines metaheuristischen Verfahrens dienen (vgl. Kapitel 2).

A. Formale Ablaufbeschreibung des gesamten Verfahrens

In diesem Kapitel erfolgt eine vollständige formale Beschreibung des in Kapitel 5 vorgestellten heuristischen Verfahrens zur Lösung von Hybrid Flow-Shop-Problemen mit Transportrestriktionen. Insbesondere werden dabei die drei Phasen der Methode synchronisiert. Wir nehmen ferner bei sämtlichen Ausführungen Bezug auf die Rahmenbedingungen des Prozesslabors.

A.1. Vorbereitung und Initialisierung

Im Folgenden werden zunächst sämtliche verwendeten Parameter aufgelistet und definiert. Dabei geben wir zum Einen deren erstmalige Initialisierung an, die beim Start des Verfahrens benötigt wird, und legen zum Anderen fest, wie sie zu Beginn einer Reoptimierung in Abhängigkeit des aktuellen Systemstatus' reinitialisiert werden müssen.

A.1.1. Eingabedaten

Die folgenden Parameter werden als Eingabe für das Verfahren benötigt:

- $J = \{1, \ldots, n\}$ - Menge der Jobs[1]
- a_j - Ankunftszeit des Jobs $j \in J$ (ab diesem Zeitpunkt ist ein Job bekannt und kann bei der Planung berücksichtigt werden)[2]
- r_j - Freigabezeit des Jobs $j \in J$ (ab diesem Zeitpunkt kann die Bearbeitung des Jobs begonnen werden)[3], es gilt also $a_j \leq r_j$
- w_j - Gewicht des Jobs $j \in J$ (zur Darstellung der Prioritäten)
- $f(t, w_j)$ - Zeitabhängige Updatefunktion für das Gewicht eines Jobs $j \in J$[4]

[1] In der Praxis ist diese Menge natürlich nicht vorab bekannt. Sie wird hier lediglich zur Simulation des Verfahrens benötigt.

[2] Im Fall des Prozesslabors entspricht dies dem Zeitpunkt der Übergabe einer Probe in das Rohrpostsystem bzw. der manuellen Eingabe einer Probe an einer dazu vorgesehenen Position im Labor.

[3] Im Fall des Prozesslabors entspricht dies dem Zeitpunkt der Ankunft einer Probe in der Rohrpostempfangsstation bzw. der manuellen Eingabe einer Probe.

[4] Diese Funktion muss je nach Situation vom Anwender spezifiziert werden.

- $J(t) := \{j \in J | a_j \leq t < C_j\}$ - Menge der Jobs, die zu einem Zeitpunkt t bereits eingetroffen aber noch nicht fertiggestellt sind, also bei der Planung berücksichtigt werden können
- $m(0, j)$ - Eingangsposition eines Jobs j in das System
- $m(-1, j)$ - Ausgangsposition eines Jobs j aus dem System
- $I = \{1, \ldots, m\}$ - geordnete Menge der Maschinentypen (bzw. Bearbeitungsstufen), jeder Job durchläuft diese Stufen in derselben Reihenfolge $1 \to \ldots \to m$, das Überspringen einer Stufe ist dabei jedoch möglich
- $M^{(i)}$ - Anzahl der Maschinen des Typs i
- q_{il} - Güteklasse einer Maschine l des Typs i
- T_{il} - Menge der Tracks, auf denen eine Maschine l des Typs i angesiedelt ist[5]
- p_{ij} - Bearbeitungszeit des Jobs $j \in J$ auf Bearbeitungsstufe $i \in I$, falls eine Stufe übersprungen werden soll, wird $p_{ij} = 0$ gesetzt
- $R = \{1, \ldots, n_R\}$ - Menge der Fahrzeuge bzw. Transportroboter
- v_R - Robotergeschwindigkeit (ist für alle Roboter identisch)
- t_r - Track, auf dem sich Roboter r befindet
- $P = \{0, \ldots, n_P\}$ - Menge der Haltepositionen für die Roboter im System, dabei setzt sich eine Position p aus dem zugehörigen Track sowie ihrer Position innerhalb des Tracks zusammen, d. h. $p = (tr, pos) \in P$.
- $s_{ab} = \frac{|pos_a - pos_b|}{v_R}$ - Fahrtzeit zwischen zwei Positionen $a = (tr, pos_a)$ und $b = (tr, pos_b)$ innerhalb desselben Tracks[6]
- $p(r)$ - Ausgangsposition des Roboters $r \in R$
- $t = t_0, \ldots, T$ - Zeitindex (im Fall des Prozesslabors Sekunden), T ist dabei eine beliebige obere Schranke für den Planungshorizont

[5] Grundsätzlich befindet sich eine Maschine natürlich nur in einem Track. Da im Fall des Prozesslabors jedoch die Übergabeposition zwischen Tracks, z. B. Übergabe zwischen Rohrposttrack und Haupttrack, als Maschinen modelliert werden, kann auch der Fall auftreten, dass eine Maschine mehreren Tracks zugeordnet ist.

[6] Diese Definition bezieht sich speziell auf das Prozesslabor, kann aber natürlich situationsabhängig angepasst werden.

A.1.2. Verwendete Parameter

Die folgenden Parameter werden während des Verfahrens in Abhängigkeit des Startzeitpunktes t_0 der jeweiligen Planungsperiode gesetzt und dienen unter anderem der Ausgabe einer ermittelten Lösung.

- s_{ij} - Startzeitpunkt des Jobs j auf Bearbeitungsstufe i
- C_{ij} - Fertigstellungszeitpunkt des Jobs j auf Bearbeitungsstufe i
- $\overline{p}_{ij}$ - Bearbeitungszeit des Jobs j auf Stufe i; dieser Parameter hat den Wert p_{ij}, falls $b(i,j) = 0$ sowie den Wert $\overline{p}_{ij} = 0$, sobald $b(i,j) = 1$ gesetzt wird (zur Definition von $b(i,j)$ siehe unten)
- $cap(i,t) \leq M^{(i)}$ - Kapazitätsauslastung der Stufe i im Zeitintervall $[t,t+1]$ (d. h. Anzahl der Jobs, die in diesem Intervall auf Maschinentyp i bearbeitet werden)
- $m(i,j) = l$, falls Job j von Maschine l des Typs i bearbeitet wird
- $y(i,l,j,t) = 1$, falls $m(i,j) = l$ und $t = s_{ij}, \ldots, C_{ij} - 1$ und 0 sonst
- S_{il} - Jobliste für die Maschine l des Typs i

Basierend auf den Werten der obigen Parameter wird im Rahmen der Routenplanung der Roboter ein Reihenfolgegraph für die durchzuführenden Transportaufträge erstellt. Auf Einzelheiten werden wir in der detaillierten Ablaufbeschreibung der dritten Phase des Verfahrens eingehen. An dieser Stelle listen wir aber bereits alle dazu verwendeten Parameter auf.

- M_j - Maschinenfolge für jeden Job j (hier sind nicht die Maschinentypen, sondern die konkreten Einzelmaschinen gemeint)
- $V = \{1, \ldots, v\}$ - Liste von Transportaufträgen, diese entspricht der Knotenmenge des Reihenfolgegraphen (wird zu Beginn von Phase 3 generiert), insbesondere kann jeder Auftrag $k \in V$ durch die nachfolgenden vier Parameter charakterisiert werden:
  - $j(k) \in J$ - Job, dem der Transportauftrag k zugeordnet werden kann
  - $o(k) \in P$ - Startposition des Auftrags k
  - $d(k) \in P$ - Zielposition des Auftrags k
  - $i(k) \in I$ - Maschinentyp am Zielort des Auftrags[7]
- T_k - Track, in dem ein Transportauftrag durchgeführt werden muss; entspricht dem Track der Start- und Zielposition des Auftrags

[7] Dieser Parameter wird zur Synchronisation der drei Phasen benötigt, hat ansonsten aber keine direkte Auswirkung auf deren Ablauf.

- $R_k = \{r \in R | t_r \in T_k\}$ - Menge der für einen Auftrag k zur Verfügung stehenden Roboter
- $A \subseteq V \times V$ - Kantenmenge des Reihenfolgegraphen, enthält die Reihenfolgebeziehungen zwischen den Transportaufträgen, insbesondere kann man wie folgt die Menge der direkten Vorgänger ($Pred(k)$) und Nachfolger ($Succ(k)$) eines Knoten k im Graphen beschreiben:
  - $Pred(k) := \{j \in V | (j,k) \in A\}$
  - $Succ(k) := \{j \in V | (k,j) \in A\}$
- $D = (V,A)$ - Gerichteter Reihenfolgegraph

Nachfolgend sind die zur Routenplanung der Roboter verwendeten Parameter aufgelistet.

- $p_r(t)$ - Position des Roboters r zum Zeitpunkt t
- $E_r = \{[a_1,b_1],[a_2,b_2],\ldots,[a_s,T]\}$ - Menge der Leerlaufintervalle des Roboters r, also gerade der Zeitintervalle, in denen der Roboter r noch nicht verplant ist
- s_k - Startzeit des Auftrags k, hier ist der Zeitpunkt gemeint, zu dem ein zugeordneter Roboter mit der Anfahrt zum Startort $o(k)$ des Auftrags k beginnt
- $\bar{s}_k$ - Tatsächliche Startzeit des Auftrags k, also der Zeitpunkt, zu dem der wirkliche Transport des Auftrags beginnt
- C_k - Fertigstellungszeit des Auftrags k, also der Zeitpunkt, zu dem der zu Auftrag k gehörige Job $j(k)$ das Ziel $d(k)$ erreicht
- $m(k) = r$, falls Roboter $r \in R_k$ die Durchführung des Transports k übernimmt

Zur Organisation einer Reoptimierung werden des Weiteren die folgenden beiden Parameter benötigt:

- $b(i,j) \in \{0,1\}$ - Binärer Parameter, der angibt, ob die Operation (i,j) blockiert ist; in diesem Fall wird der Wert 1 gesetzt, oder nicht (Wert 0). Dies ist für die Initialisierung der Daten zu Beginn einer Reoptimierung von Bedeutung, denn blockierte Operationen, werden dabei nicht mehr berücksichtigt
- $b(k) \in \{0,1\}$ - Analog definierter Parameter, der angibt, ob ein Transportauftrag k blockiert ist, oder nicht

A.1.3. Erstmalige Initialisierung

Zu Beginn einer Simulation bzw. vor dem erstmaligen Start des Verfahrens werden die oben definierten Parameter wie folgt initialisiert:

- $t_0 := 0$
- $J(t_0) := \{j \in J | a_j = 0\}$
- $s_{ij} := T$ für alle i, j
- $C_{ij} := T$ für alle i, j
- $\overline{p}_{ij} := p_{ij}$ für alle i, j
- $cap(i,t) = 0$ für alle i,t
- $m(i, j) = 0$ für alle i, j (Die Auswahl 0 gibt an, dass für die Operation (i, j) noch keine Maschine ausgewählt wurde.)
- $y(i,l, j,t) = 0$ für alle i,l, j,t
- $S_{il} := \emptyset$ für alle i,l
- $M_j := \emptyset$ für alle j
- $V := \emptyset$
- $A := \emptyset$
- $p_r(t) := p(r)$ für alle t,r
- $E_r := \{[0,T]\}$ für alle r
- $b(i, j) = 0$ für alle i, j

Die entsprechenden Parameter für die Transporte müssen hier noch nicht gesetzt werden, da zu Beginn noch keine Transportaufträge generiert wurden.

A.1.4. Initialisierung zu Beginn einer Reoptimierung

Es wird reoptimiert, wenn einer neuer Job j eintrifft[8], also für alle Jobs zum Zeitpunkt a_j. In diesem Fall wird zunächst der Startwert aktualisiert und die blockierten Operationen und Transportaufträge, für welche der Schedule bei der Reoptimierung nicht verändert werden darf, werden festgelegt.

[8] Falls aus einem anderen Grund reoptimiert werden muss, setzt man t_0 entsprechend. Zur Simulation gehen wir hier der Einfachheit halber jedoch davon aus, dass nur beim Eintreffen neuer Jobs eine Reoptimierung gestartet wird.

- $t_0 := a_j$
- Für alle Jobs j setze $w_j = f(t_0, w_j)$.
- $J(t_0) := \{j \in J | a_j \leq t_0 < C_j\}$
- Für alle Transportaufträge k:

  Falls $s_k \leq t_0$, dann setze $b(k) = 1$ und $b(i(k), j(k)) = 1$.
- Für alle $k \in V$ mit $b(k) = 1$:

  Setze nun für alle $k' \in Prec(k)$ ebenfalls $b(k') = 1$ und $b(i(k'), j(k')) = 1$ (Die Vorgänger eines blockierten Auftrags müssen ebenfalls blockiert werden.)
- Für alle $j \in J(t_0)$, alle Stufen i mit $b(i, j) = 1$ und diejenige Maschine l des Typs i mit $m(i, j) = l$ setze auch für alle Vorgänger $k \in S_{il}$ von j: $b(i,k) = 1$ (d. h. Vorgänger blockierter Jobs in der Jobfolge einer Maschine müssen ebenfalls geblockt werden).

Nach Festlegung der blockierten Operationen können auch die übrigen Parameter entsprechend aktualisiert werden.

- Parameter der Operationen oder Transportaufträge, die nicht blockiert sind, werden zurückgesetzt, d. h. für alle Operationen (i, j) mit $b(i, j) = 0$ sowie alle Transportaufträge k mit $b(k) = 0$ setze:
  - $s_{ij} = C_{ij} = T$
  - $s_k = \overline{s}_k = C_k = T$
  - $m(k) = 0$
  - $\overline{p}_{ij} = p_{ij}$

  Für alle Operationen (i, j) und Maschinen l des Typs i mit $b(i, j) = 0$ und $m(i, j) = l$ setze:
  - $y(i, l, j, t) = 0$ für alle $t \geq t_0$
  - $m(i, j) = 0$
  - $j \leftarrow S_{il}$, entferne j aus der Jobliste für Maschine l des Typs i
- Falls eine Operation blockiert ist, darf sie im List Scheduling Algorithmus am Ende der ersten Phase nicht mit verplant werden, deshalb setze für alle Operationen mit $b(i, j) = 1$:
  - $\overline{p}_{ij} = 0$
- Die Parameter, welche den Auslastungsgrad der Maschinen angeben, müssen dementsprechend angepasst werden:

- Setze zunächst $cap(i,t) = 0$ für alle $t \geq t_0$ und alle $i \in I$
- Für alle Operationen (i,j) mit $b(i,j) = 1$ setze anschließend $cap(i,t) = cap(i,t) + 1$ für $t = \max(s_{ij}, t_0), \ldots, C_{ij} - 1$

- Abschließend müssen noch die Leerlaufzeiten und Positionen der Roboter im System aktualisiert werden. Dazu werden die Roboter vom Zeitpunkt t_0 an zunächst vollständig freigegeben, wir beginnen also mit $E_r = \{[t_0, T]\}$, und anschließend werden die Belegungsintervalle für die blockierten Aufträge, welche dem Roboter r zugeordnet sind (d. h. $b(k) = 1, m(k) = r$) aus $[t_0, T]$ „herausgeschnitten“ und E_r entsprechend aktualisiert. Algorithmisch kann man diese Vorgehensweise wie folgt beschreiben:

  Für alle Roboter $r \in R$:

  Setze $E_r := \{t_1^r\}$ mit $t_1^r = [a_{t_1^r}, b_{t_1^r}] := [t_0, T]$.

  Dabei ist $p_r(t_0) := d(k^*)$,

  wobei k^* derjenige Auftrag mit $s_{k^*} : \max_{k \in V} \{s_k | m(k) = r, b(k) = 1, s_k \leq t_0\}$.[9]

  Für alle Aufträge $k \in V$ mit $m(k) = r$ und $b(k) = 1$:

  Falls $C_k \leq t_0$, setze $k = k + 1$.[10]

  Falls $s_k \leq t_0 < C_k$, setze $a_{t_1^r} := C_k$ mit $p_r(C_k) = d(k)$.

  Falls $s_k > t_0$

  Bestimme in E_r dasjenige Intervall $t_i^r = [a_{t_i^r}, b_{t_i^r}]$ mit $a_{t_i^r} \leq s_k$ und $C_k \leq b_{t_i^r}$.

  Entferne t_i^r aus E_r.

  Falls $s_k - a_{t_i^r} \geq 3$, füge $[a_{t_i^r}, s_k]$ zu E_r hinzu mit $p_r(s_k) := p(a_{t_i^r})$.

  Falls $b_{t_i^r} - C_k \geq 3$, füge $[C_k, b_{t_i^r}]$ zu E_r hinzu mit $p_r(C_k) := d(k)$.

A.2. Phase 1: Sequenzierung der Jobs

Bei der (Re-)Sequenzierung werden nur diejenigen Jobs $j \in J(t_0)$ berücksichtigt, für die noch keine Operation blockiert ist, also nur Jobs mit $b(i,j) = 1$ für alle Bearbeitungs-

[9] Die Position des Roboters zum Zeitpunkt t_0 muss definiert werden. Hier wird diese als Zielort des letzten Auftrags, den dieser Roboter durchgeführt hat, gewählt. Dabei kann natürlich die Situation auftreten, dass der Roboter zum Zeitpunkt t_0 gerade mit der Durchführung eines Auftrags beschäftigt ist (also $s_k \leq t_0 < C_k$), in diesem Fall wird die Position von r zum Zeitpunkt t_0 trotzdem schon auf $d(k)$ gesetzt. Das ist zwar vorübergehend falsch, wird durch das Ersetzen von t_0 durch C_k im zweiten Fall (vgl. nächster Schritt) aber sofort wieder behoben und ist deshalb nicht von Bedeutung.

[10] Für Aufträge die in der Vergangenheit liegen passiert natürlich gar nichts. Dieser Fall kann dennoch auftreten, da auch diese Aufträge noch mitgeführt werden, falls der zum jeweiligen Auftrag gehörige Job noch nicht fertiggestellt ist.

stufen $i \in I$. Für die übrigen Jobs wird die bisherige Sequenzierung beibehalten. Im Folgenden bezeichne nun $j_1, \ldots, j_r$ die bereits bestehende (blockierte) Jobsequenz und $\overline{J(t_0)} := \{j_i | i = r+1, \ldots n\}$ die Menge der (neu) zu sequenzierenden Jobs. Die nun folgenden Sequenzierungsmethoden werden nur dann durchgeführt, wenn mindestens vier Jobs zur Bearbeitung anstehen, d. h. wenn $n - r \geq 4$. Für drei oder weniger Jobs werden stattdessen alle Möglichkeiten, diese zu sortieren, berücksichtigt.

A.2.1. Sortierung der Jobs auf verschiedene Weisen

1. **WSTP-Sequenz:**

   (1) Wähle die Sortierung $j_{r+1}, \ldots, j_n$, falls
   $$\frac{\sum_{i \in I} p_{ij_{r+1}}}{w_{j_{r+1}}} \leq \frac{\sum_{i \in I} p_{ij_{r+2}}}{w_{j_{r+2}}} \leq \ldots \leq \frac{\sum_{i \in I} p_{ij_n}}{w_{j_n}}.$$

2. **Engpass-Sequenzen:**

   Lege Engpassstufe $i^* \in I$ fest, falls $w(i^*) = \max_{i \in I} w(i)$, wobei $w(i) := \frac{\sum_{j \in \overline{J(t_0)}} p_{ij}}{M^{(i)}}$.

   (2.1) Wähle die Sortierung $j_{r+1}, \ldots, j_n$, falls
   $$\frac{p_{i^* j_{r+1}}}{w_{j_{r+1}}} \leq \frac{p_{i^* j_{r+2}}}{w_{j_{r+2}}} \leq \ldots \leq \frac{p_{i^* j_n}}{w_{j_n}}.$$

   Berechne für die Engpassstufe i^* folgende zusätzlichen Werte:

   $$r_{i^* j} := \sum_{h=1}^{i^*} p_{(h-1)j}, \text{ wobei } p_{0j} := \max(t_0, r_j) \text{ für alle } j \in \overline{J(t_0)}.$$

   Setze außerdem $L_{i^*}(t) := \{j \in \overline{J(t_0)} | r_{i^* j} \leq t,\ p_{i^* j} > 0\}$.

   Erstelle nun wie folgt einen Hilfsschedule[11]:

   - Beginne für $t = t_0$.
   - Setze $p'_{i^* j} := p_{i^* j}$.
   - Für alle $j \in L_{i^*}(t)$ berechne
     $$k_j := \frac{p'_{i^* j}}{w_j}$$
     und wähle (maximal) diejenigen $M^{(i)}$ Jobs j mit den kleinsten Koeffizienten k_j aus und weise ihnen das Intervall $[t, t+1]$ zu. Setze anschließend
     $$p'_{i^* j} := p'_{i^* j} - 1 \text{ sowie } t = t + 1$$
     und beginne von vorne.

[11] Dieser wird für die Jobs in $L_{i^*}(t)$ mit Hilfe der preemptive WSPT-Regel erstellt.

- Das Verfahren bricht ab, sobald alle Jobs vollständig verplant sind, also wenn $p'_{i^*j} = 0$ für alle $j \in J(t_0)$ gilt.
- Bestimme für jeden Job j den Zeitpunkt C^*_j, zu dem er in dem so erzeugten Hilfsschedule fertiggestellt wird.

Verwende nun die folgenden drei Methoden zur Sortierung der Jobs:

(2.2) Wir erhalten die Sortierung $j_{r+1}, \ldots, j_n$, falls

$$C^*_{j_{r+1}} \leq C^*_{j_{r+2}} \leq \ldots \leq C^*_{j_n}.$$

(2.3) Wir setzen $M^*_j := C^*_j - \frac{1}{2} p_{i^*j}$ und erhalten die Sortierung $j_{r+1}, \ldots, j_n$, falls

$$M^*_{j_{r+1}} \leq M^*_{j_{r+2}} \leq \ldots \leq M^*_{j_n}.$$

(2.4) Wir wählen $C^*_j(\frac{1}{2})$ als denjenigen Zeitpunkt, zu dem $\frac{1}{2} p_{i^*j}$ Zeiteinheiten des Jobs j fertiggestellt sind. Wir erhalten dann die Sortierung $j_{r+1}, \ldots, j_n$, falls

$$C^*_{j_{r+1}}(\tfrac{1}{2}) \leq C^*_{j_{r+2}}(\tfrac{1}{2}) \leq \ldots \leq C^*_{j_n}(\tfrac{1}{2}).$$

A.2.2. List Scheduling (unter No-Wait-Constraints)

Wende folgendes Verfahren auf sämtliche der oben erzeugten Sortierungen der Jobs an.

Input:

- Jobsequenz $j_1, \ldots, j_r, j_{r+1}, \ldots, j_n$ (Hier müssen alle Jobs in $J(t_0)$ berücksichtigt werden!)

Output:

- $\overline{s}_{ij}$ - (Vorläufiger) Startzeitpunkt des Jobs j auf Stufe i
- $cap(i,t)$ - Auslastung der Bearbeitungsstufe i zum Zeitpunkt t (entspricht der Anzahl der Jobs, die im Zeitintervall $[t, t+1]$ auf Stufe i bearbeitet werden)

Algorithmus:

1. Für alle Jobs $j = j_1, \ldots, j_n$ beginne mit[12]:

$$\begin{aligned} t_{j_k} \quad := \quad & \max\{\max(t_0, r_{j_k}) + \max_{j \in J(t_0)} \{\max_{i \in I}\{C_{ij} \cdot b(i,j)\} - \sum_{i \in I} p_{ij} \cdot b(i,j) - r_j\}, \\ & \max_{j \in J(t_0)} \{\max_{i \in I}\{C_{ij} \cdot b(i,j)\}\}\} \end{aligned}$$

[12] Die frühestmögliche Startzeit eines Jobs muss an den bisherigen partiellen Schedule angeglichen werden, der bei einer Reoptimierung blockierte Transportaufträge enthalten kann. Aus diesem Grund verschieben wir die frühestmögliche Startzeit t_{j_k} eines Jobs j_k um die maximale Gesamttransport- und Wartezeit, die für einen Job bisher eingefügt wurde: $\max_{j \in J(t_0)}\{\max_{i \in I}\{C_{ij} \cdot b(i,j)\} - \sum_{i \in I} p_{ij} \cdot b(i,j) - r_j\}$. Die übrigen Operationen eines Jobs, der blockierte Operationen enthält, dürfen natürlich auch erst nach Fertigstellung der vorangehenden Operationen gestartet werden, d. h. $t_{j_k} \geq \max_{j \in J(t_0)}\{\max_{i \in I}\{C_{ij} \cdot b(i,j)\}$.

und überprüfe für alle Bearbeitungsstufen i, ob für ein beliebiges t in der Zeitspanne $t = t_{j_k} + \sum_{l=1}^{i-1} \overline{p}_{l j_k}, \ldots, t_{j_k} + \sum_{l=1}^{i} \overline{p}_{l j_k} - 1$ die Gleichung $cap(i,t) = M^{(i)}$ erfüllt ist[13].

- Falls ja, setze $t_{j_k} := t_{j_k} + 1$.
- Falls nein, setze $\overline{s}_{1 j_k} = t_{j_k}$ und $\overline{s}_{i_k} = \overline{s}_{i-1, j_k} + \overline{p}_{i-1, j_k}$ für $i = 2, \ldots, m$. Setze zudem $cap(i,t) = cap(i,t) + 1$ für $t = \overline{s}_{i j_k}, \ldots, \overline{s}_{i j_k} + \overline{p}_{i j_k} - 1$.
- Falls $b(i,j) = 0$, setze $s_{i j_k} = \overline{s}_{i j_k}$.

Berechne für alle fünf Sequenzen S den Zielfunktionswert $ZF(S) = \sum_{j \in \overline{J(t_0)}} w_j C_j(S)$, wobei $C_j(S)$ den Fertigstellungszeitpunkt von Job j bei Anwendung des obigen Algorithmus' auf Sequenz S bezeichnet.

Die Sequenz S^* mit kleinstem Zielfunktionswert $ZF(S^*)$ wird zur Weiterverarbeitung ausgewählt und die diesbezüglich ermittelten Startzeiten s^*_{ij} für sämtliche Jobs und Bearbeitungsstufen übernommen.

A.3. Phase 2: Maschinenauswahl

Für alle Maschinentypen $i \in I$ und alle Operationen (i,j) mit $b(i,j) = 0$ wird zunächst eine initiale Maschinenbelegung bestimmt.

A.3.1. Initiale Maschinenauswahl für Maschinentyp i:

Input:

- In Phase 1 bestimmte Jobsequenz S^* (alle Jobs aus $J(t_0)$ werden berücksichtigt)
- $(s_{i j_1}, C_{i j_1}), \ldots, (s_{i j_n}, C_{i j_n})$ (Start- und Fertigstellungszeiten der Jobs auf Bearbeitungsstufe i, dabei ist $C_{i j_k} := s_{i j_k} + p_{i j_k}$)

Output:

- $m(i, j_k) = l$, falls Job j_k von Maschine l des Typs i bearbeitet wird
- $y(i,l,j_k,t) = 1$, falls $m(i,j_k) = l$ und $t = s_{i j_k}, \ldots, C_{i j_k} - 1$ und $y(i,l,j_k,t) = 0$, sonst
- S_{il} - Jobliste für jede Maschine l des Typs i

Algorithmus:

1. Sortiere die Jobs j_k in aufsteigender Reihenfolge ihrer Startzeiten $s_{i j_k}$, so ergibt sich die Sequenz[14] $j_1, \ldots, j_n$ mit $s_{i j_k} \leq s_{i j_{k+1}}$ für $k = 1, \ldots, n-1$.

[13] Hier ist der Übergang zu $\overline{p}_{ij}$ erforderlich, da für blockierte Operationen $\overline{p}_{ij} = 0$ gilt.

[14] Die Indizierung der Jobs behalten wird der Einfachheit halber bei.

2. Den Jobs $j_1,\ldots,j_n$ mit $b(i,j_k)=0$ weisen wir nun wie folgt eine Maschine der Stufe i zu (für die übrigen Jobs besteht die Maschinenzuweisung bereits)

   Für alle $k=1,\ldots,n$:

   Falls $b(i,j_k)=1$ oder $p_{ij_k}=0$, setze $k=k+1$.

   Sonst

   Beginne mit $l=1$.

   Falls ein Index r existiert mit $m(i,j_r)=l$ und $C_{ij_r}>s_{ij_k}$ sowie $s_{ij_r}\leq C_{ij_k}$, setze $l=l+1$.[15]

   Sonst

   Setze $m(i,j_k)=l$.

   Setze $y(i,l,j_k,t)=1$ für $t=s_{ij_k},\ldots,s_{ij_k}+p_{ij_k}-1$.

   $S_{il}\leftarrow j_k$, d. h. hänge j_k an das Ende der Liste S_{il}.

Die so bestimmte Maschinenbelegung wird nun wenn möglich verbessert.

A.3.2. Verbesserung der initialen Maschinenauswahl für Typ i

Zur Verbesserung der oben getroffenen Maschinenauswahl zerlegen wir den entsprechenden Schedule zunächst in Blocks. Dazu werden Nahtstellen zwischen den einzelnen Blocks definiert und in einer Liste $\delta_i=(t_1,\ldots,t_{a_i})$ gespeichert.

Bestimmung der Nahtstellen für Stufe i:

Für alle $t=T_{\min}:=\max\{t_0,\min\limits_{k=1,\ldots n} s_{ij_k}\},\ldots,T_{\max}:=\max\limits_{k=1,\ldots,n} C_{ij_k}$

Falls für jedes $l=1,\ldots,M^{(i)}$ mindestens eine der folgenden drei Bedingungen gilt (aber nicht für alle die Bedingungen 1 und 2), dann markiere t bzw. hänge t an das Ende der Liste δ_i.

1. $t\neq t_0$ und $y(i,l,j_k,t-1)=0$ für alle $k=1,\ldots,n$
2. $t\neq T_{\max}$ und $y(i,l,j_k,t)=0$ für alle $k=1,\ldots,n$
3. $t\notin\{t_0,T_{\max}\}$ und $y(i,l,j_k,t-1)=1$ für ein $k\in\{1,\ldots,n\}$ sowie $y(i,l,j_r,t)=1$ für ein $r\in\{1,\ldots,n\}$, wobei $r\neq k$.

Sei nun $\delta_i=(t_1,\ldots,t_{a_i})$ die Liste der Markierungen aus obiger Schleife. Wir fassen nun alle Jobs, die auf derselben Maschine l des Typs i und innerhalb desselben Intervalls $[t_k,t_{k+1}]$ bearbeitet werden, zu Jobeinheiten $U(i,l,k)$ zusammen.

[15] Hier kann l nie größer werden als $M^{(i)}$, denn durch die Lösung aus Phase 1 ist gewährleistet, dass die Anzahl verfügbarer Maschinen eines Typs zu keinem Zeitpunkt überschritten wird. Operationen mit der Bearbeitungszeit Null wird keine Maschine zugewiesen.

Definition von Jobeinheiten für jedes Intervall $[t_k, t_{k+1}]$ und der Stufe i:

Für alle Listen S_{il} ($l = 1, \ldots, M^{(i)}$):

Initialisiere $U(i,l,k) := \emptyset$ und $w_{U(i,l,k)} := 0$ für alle $k = 1, \ldots, a_i - 1$

Für alle Intervalle $[t_k, t_{k+1}]$ ($k = 1, \ldots, a_i - 1$):

Für alle Jobs j in S_{il}:

Falls $t_k < C_{ij} \leq t_{k+1}$

$j \to U(i,l,k)$ (Job j zur Einheit $U(i,l,k)$ hinzufügen)

Falls $b(i,j) = 1$, markiere die Einheit $U(i,l,k)$ als verplant und die Maschine l für das Intervall $[t_k, t_{k+1}]$ als belegt (d. h. Einheiten, die eine blockierte Operation enthalten, dürfen keiner anderen Maschine zugewiesen werden).

$w_{U(i,l,k)} = w_{U(i,l,k)} + w_j$

Gehe zum nächsten Job in S_{il}.

In jedem der Intervalle werden die so definierten Einheiten nun gegebenenfalls anderen Maschinen zugeordnet, so dass solche mit großem Gewicht von Maschinen mit hoher Güteklasse bearbeitet werden.

Neue Maschinenzuordnung für Bearbeitungsstufe i:

Bevor unverplante (nicht markierte) Einheiten neu verplant werden können, müssen zunächst die in der initialen Maschinenbelegung ausgewählten Maschinen wieder freigegeben werden:

Für alle Intervalle $[t_k, t_{k+1}]$ ($k = 1, \ldots, a_i - 1$) und für alle $l = 1, \ldots, M^{(i)}$:

Falls $U(i,l,k)$ nicht als verplant markiert, dann

Für alle $j \in U(i,l,k)$:

Setze $m(i,j) = 0$ und $y(i,l,j,t) = 0$ für $t = s_{ij}, \ldots, s_{ij} + p_{ij} - 1$.

Entferne j aus S_{il}.

Markiere die Maschine l im Intervall $[t_k, t_{k+1}]$ als frei.

Anschließend werden die unverplanten (nicht markierten) Einheiten erneut den Maschinen zugeordnet.

Für alle Intervalle $[t_k, t_{k+1}]$ ($k = 1, \ldots, a_i - 1$):

Solange noch unverplante (nicht markierte) Einheiten $U(i,l,k)$ vorhanden sind:

1. Wähle unter den unverplanten Einheiten diejenige $U(i,l^*,k)$ mit maximalem Gewicht $w_{U(i,l^*,k)}$.
2. Wähle für diese Einheit nun diejenige freie Maschine l mit größter Güteklasse q_{il}.
3. Markiere $U(i,l^*,k)$ als verplant und l für das Intervall $[t_k,t_{k+1}]$ als belegt.
4. Für alle Jobs j in $U(i,l^*,k)$ setze $m(i,j)=l$ sowie $y(i,l,j,t)=1$ für $t = s_{ij},\ldots,s_{ij}+p_{ij}-1$.
5. Füge j zu S_{il} hinzu.

Aus dieser finalen Maschinenauswahl ergibt sich nun direkt die Maschinenfolge M_j für jeden Job j als:

$$M_j : m(i_1,j) \to m(i_2,j) \to \ldots \to m(i_{m_j},j).$$

Dabei werden Stufen mit $m(i,j)=0$ nicht berücksichtigt[16], es gilt also $\{i_1,\ldots,i_{m_j}\} \subseteq I$.

A.4. Phase 3: Routing der Transportaufträge

A.4.1. Erstellung des Reihenfolgegraphen

Eingabe:

- Jobfolge S_{il} für alle Maschinen $l = 1,\ldots,M^{(i)}$ eines Typs $i \in I$
- Maschinenfolge M_j für jeden Job $j \in J(t_0)$
- Liste von Transportaufträgen $V \ni k = (j(k),i(k),o(k) \to d(k))$: Die Transportaufträge ergeben sich für jeden Job j direkt aus der Maschinenfolge M_j, hinzu kommen außerdem jeweils ein Auftrag zur Beförderung des Jobs von seinem Eingang in das System (Dummymaschine, Typ 0) zur ersten Maschine sowie einer zur Beförderung des Jobs aus dem System, also von der letzten Maschine zu seinem Ausgang (Dummymaschine, Typ -1). D. h. für einen Job j erhalten wir die folgende Menge von Transportaufträgen:

$$\begin{aligned} k_1 &= (j,i_1,m(0,j) \to m(i_1,j)) \\ k_2 &= (j,i_2,m(i_1,j) \to m(i_2,j)) \\ &\vdots \\ k_{m_j+1} &= (j,-1,m(i_{m_j},j) \to m(-1,j)) \end{aligned}$$

[16] Da für $p_{ij}=0$ ein Job eine Stufe überspringt, beinhaltet die Maschinenfolge nur die tatsächlich ausgewählten Maschinen.

Konstruktion des Graphen:

1. Für jeden Transportauftrag k jedes Jobs j wird ein Knoten in den Graphen eingefügt, unabhängig davon, ob k blockiert ist oder nicht.

2. Für sämtliche Paare von Transportaufträgen a,b mit $j(a) = j(b)$ und $d(a) = o(b)$ wird eine gerichtete Kante $a \to b$ in den Graphen eingefügt.

3. Basierend auf den Jobfolgen S_{il} für alle Maschinen l des Typs i werden des Weiteren wie folgt gerichtete Kanten gebildet: Angenommen, j ist direkter Vorgänger von k in S_{il} und es gelte $(m(i_s, j) = m(i_t, k) = l)$ mit $i_s = i_t$ und j ist direkter Vorgänger von k in S_{il}. Außerdem seien für die Jobs j und k folgende Maschinenfolgen vorgegeben:

$$M_j : \ldots \to l \to m(i_{s+1}, j) \to \ldots$$
$$M_k : \ldots \to m(i_{t-1}, k) \to l \to \ldots$$

Daraus ergibt sich im Reihenfolgegraphen die Kante:

$$(j, i_{s+1}, l \to m(i_{s+1}, j)) \longrightarrow (k, i_t, m(i_{t-1}, k) \to l).$$

A.4.2. Verplanung der Transportaufträge

Eingabe:

- p_k - Transportzeit für Auftrag $k \in V$, diese entspricht hier $s_{o(k)d(k)} = \frac{|o(k)-d(k)|}{v_R}$, zur Vereinfachung der Darstellung verwenden wir im Folgenden jedoch die Bezeichnung p_k

- d_k - Verzögerungszeit, die durch die Bearbeitung des beförderten Jobs am Zielort des Transportauftrags entsteht, d. h. es gilt $d_k = p_{i(k)j(k)}$, falls $i(k) = -1$, setzen wir $d_k = 0$

Zusätzliche Parameter:

- L - Menge der verplanbaren Jobs (d. h. L enthält während dieser Phase des Verfahrens immer die Jobs, deren Vorgänger bereits verplant sind)

- L^* - Menge der bereits verplanten Jobs

- $\bar{r}_k$ - Freigabezeit des Auftrags k in Abhängigkeit der Reihenfolgebedingungen; die entsprechenden Werte werden während des Verfahrens berechnet, sobald ein neuer Job zu L hinzugefügt wird

- $f_k(r)$ - Frühestmögliche Startzeit des Auftrags k, falls Roboter $r \in R_k$ für den Transport k ausgewählt wird

Initialisierung:

- $L^* := \{k \in V | b(k) = 1\}$
- $L := \{k \in V \setminus L^* | Pred(k) \subseteq L^*\}$
  (falls $L^* = \emptyset$, sind dies gerade alle Auträge k mit $Pred(k) = \emptyset$)
- Solange es ein $k \in L$ mit $p_k = 0$ gibt:[17]

  $L \longrightarrow k$ (entferne k aus L)

  $k \longrightarrow L^*$ (füge k zu L^* hinzu)

  Setze $s_k = \bar{s}_k = C_k = s_{i(k),j(k)}$.

  Für alle $k' \in Succ(k)$:

  Falls $Prec(k') \subseteq L^*$, dann $k' \longrightarrow L$ (füge k' zu L hinzu).
- $\bar{r}_k := T$ für alle $k \in L$
- $f_k(r) := T$ für alle $k \in L$, $r \in R_k$

Freigabezeiten der Transportaufträge in L:

Für einen verfügbaren Auftrag $k \in L$ ergibt sich die Freigabezeit aus der partiellen Lösung, je nachdem welche Gestalt $Pred(k)$ besitzt.

- 1. Fall: $Pred(k) = \emptyset$, dann ist $\bar{r}_k = \max(t_0, r_{j(k)})$.
- 2. Fall: Es gibt ein $k' \in Pred(k)$ mit $j(k) = j(k')$, dann ist $\bar{r}_k \geq C_{k'} + d_{k'}$
- 3. Fall: Es gibt ein $k' \in Pred(k)$ mit $j(k) \neq j(k')$, dann ist im Fall des Prozesslabors $\bar{r}_k \geq \bar{s}_{k'} + 1 - p_k$ (ansonsten gilt $\bar{r}_k \geq \bar{s}_{k'} - p_k$).

Insgesamt erhält man also:

$$\bar{r}_k := \max\{t_0, r_{j(k)}, \max_{j \in Pred(k), j(j)=j(k)}(C_j + d_j), \max_{j \in Pred(k), j(j)\neq j(k)}(\bar{s}_j + 1 - p_k)\}$$

[17] Transportaufträge mit $d(k) = o(k)$ also $p_k = 0$ müssen generiert werden, um die Reihenfolgebeziehungen der übrigen Transportaufträge sowie die in Phase 1 und 2 bestimmten Maschinenfolgen richtig abzubilden. Da hier aber eigentlich gar nichts transportiert werden muss, dürfen diese „Dummyaufträge“ bei der Routenplanung nicht berücksichtigt werden, sondern werden direkt in die Menge der verplanten Aufträge verschoben. Dabei muss zur Wahrung der Reihenfolge auch eine Start- und Endzeit für den Auftrag gesetzt werden. Diese entspricht dann einfach der Startzeit des transportierten Jobs auf der Zielmaschine. Ein solcher Transportauftrag tritt im Prozesslabor auf, falls eine Probe direkt in eine Maschine eingesetzt oder aus dieser entnommen wird, also kein Transport vom Eingang zur ersten Maschine bzw. von der letzten Maschine zum Ausgang erforderlich ist.

Bestimmung der frühestmöglichen Startzeiten $f_k(r)$:

Wir bestimmen nun für alle Aufträge $k \in L$ und alle Roboter $r \in R_k$ die frühestmögliche Startzeit $f_k(r)$ eines Auftrags k unter der temporären Annahme, dass der Roboter r zur Durchführung von k verwendet wird. Wird ein Roboter r im Laufe des Verfahrens tatsächlich zur Durchführung des Auftrags k ausgewählt, so gilt dann $\bar{s}_k = f_k(r)$.

Dazu wird für jeden Auftrag $k \in L$ und jeden Roboter $r \in R_k$, für die Intervalle in E_r der Reihe nach (beginnend bei dem Intervall mit kleinstem b_{t_r}, so dass $\bar{r}_k < b_{t_r}$) überprüft, ob der Auftrag in diesem Intervall abgearbeitet werden kann[18]. Wenn nicht, wird das nächste Intervall überprüft und wenn ja, wird das entsprechende Intervall ausgewählt bzw. der Wert $f_k(r)$ entsprechend berechnet (siehe unten).

Dabei kann folgende Situation auftreten: Falls $\bar{r}_k > a_{t_r}$, kann der Roboter schon mit der Anfahrt zum Startort $o(k)$ beginnen, bevor der Auftrag k überhaupt freigegeben wird. Natürlich darf der Roboter auch nicht beliebig früh losfahren, mit anderen Worten, er soll den Startort $o(k)$ nicht vor der Freigabe des Auftrags k erreichen, da in diesem Fall möglicherweise mehr „freie Roboterzeit“ beansprucht wird als notwendig. Wenn die Differenz $\bar{r}_k - a_{t_r} > 0$ kleiner ist als die entstehende Anfahrtszeit $s_{p_r(a_{t_r})o(k)}$, so beginnt der Roboter schon zum Zeitpunkt a_{t_r} mit der Anfahrt und nur ein Teil der Anfahrt wird schon vor der Freigabe des Auftrags k erledigt. Der Startzeitpunkt[19] s_k für einen Auftrag k ergibt sich bei vorheriger Wahl des Intervalls $t_r = [a_{t_r}, b_{t_r}]$ also als:

$$s_k = \max(a_{t_r}, \bar{r}_k) - \min(\max(\bar{r}_k - a_{t_r}, 0), s_{p_r(a_{t_r})o(k)}).$$

Algorithmisch kann man die Berechnung der frühestmöglichen Startzeit $f_k(r)$ eines Auftrags k bei Verwendung des Roboters r daher wie folgt beschreiben:

Für alle $k \in L$ und für alle $r \in R_k$:[20]

Für alle Intervalle $t_r := [a_{t_r}, b_{t_r}] \in E_r$ mit $\overline{r_k} < b_{t_r}$:

Falls t_r das letzte Intervall in E_r ist[21]

Setze $f_k(r) := \max(a_{t_r}, \bar{r}_k) - \min(\max(\bar{r}_k - a_{t_r}, 0), s_{p_r(a_{t_r})o(k)}) + s_{p_r(a_{t_r})o(k)}$.

Setze $r := r + 1$.

Sonst

Falls $\max(a_{t_r}, \bar{r}_k) - \min(\max(\bar{r}_k - a_{t_r}, 0), s_{p_r(a_{t_r})o(k)})$
$+ s_{p_r(a_{t_r})o(k)} + p_k + s_{d(k)p_r(b_{t_r})} \leq b_{t_r}$

Setze $f_k(r) := \max(a_{t_r}, \bar{r}_k) - \min(\max(\bar{r}_k - a_{t_r}, 0), s_{p_r(a_{t_r})o(k)}) + s_{p_r(a_{t_r})o(k)}$.

[18] Dies ist der Fall, wenn $s_{p_r(a_{t_r}),o(k)} + p_k + s_{d(k),p_r(b_{t_r})} \leq b_{t_r} - a_{t_r}$ gilt.

[19] Hier ist der Beginn der Anfahrt und nicht der Beginn des tatsächlichen Transports gemeint.

[20] Hier ist die Reihenfolge entscheidend. Die Transportaufträge bilden die äußere Schleife.

[21] D. h. die rechte Seite b_{t_r} entspricht dem Default-Wert T.

Setze $r := r + 1$.

Sonst

Gehe zum nächsten Intervall in E_r.

Setze $k := k + 1$.

Algorithmus mit Berechnung der Kontrollkoeffizienten:

Solange noch verplanbare Jobs vorhanden sind, also $L \neq \emptyset$ gilt, berechnen wir nun in jedem Schritt des Algorithmus' für alle Aufträge $k \in L$ und alle Roboter $r \in R_k$ Kontrollkoeffizienten κ_{kr}. Die Kombination aus Roboter und Transportauftrag, für welche das Minimum dieser Koeffizienten angenommen wird, wählen wir zur entsprechenden Verplanung aus. Anschließend wird dieser Auftrag aus L entfernt und ggf. weitere Knoten, deren Vorgänger nun abgearbeitet sind, zu L hinzugefügt. Formal kann man diese Vorgehensweise wie folgt beschreiben:

Solange $L \neq \emptyset$:

Für alle $k \in L$ und $r \in R_k$:

Bestimme $f_k(r)$.

Bestimme folgenden Kontrollkoeffizienten:[22]

$$\kappa_{kr} := f_k(r) + \frac{s_{p_r(a_{t_r})o(k)} + p_k}{w_{j(k)}}.$$

Wähle das Paar (k, r), für welches der Koeffizient κ_{kr} minimal wird, und verplane den Roboter r zur Durchführung des Transports k (siehe unten).

Verplanung eines Roboters r zum Transport eines Auftrags k:

Wird ein Roboter r zur Bearbeitung eines Auftrags k ausgewählt, so müssen alle entsprechenden Parameter aktualisiert werden:

- $m(k) = r$
- $k \to L^*$ (k zur Menge der verplanten Aufträge hinzufügen)
- $L \to k$ (k aus der Menge der verplanbaren Aufträge entfernen)
- $L := \{k \in V \setminus L^* | Pred(k) \subseteq L^*\}$ (d. h. alle Nachfolger von k werden zu L hinzugefügt, sofern auch deren übrige Vorgänger bereits zu L^* gehören), also:

  Für alle $k' \in Succ(k)$:

[22] Bei der Berechnung der Kontrollkoeffizienten κ_{kr} wird die nötige Anfahrtszeit berücksichtigt, d. h. a_{t_r} bezeichnet hier erneut die linke Seite des Intervalls aus E_r, welches bei der Berechnung von $f_k(r)$ ausgewählt wurde.

Falls $Pred(k') \subseteq L^*$, dann $k' \rightarrow L$ (füge k' zu L hinzu).

- $s_k := f_k(r) - s_{p_r(a_{t_r})o(k)}$, falls $t_r = [a_{t_r}, b_{t_r}]$ das zur Bearbeitung von k ausgewählte Leerlaufintervall des Roboters r ist
- $\bar{s}_k = f_k(r)$
- $C_k := \bar{s}_k + p_k$
- $E_r \rightarrow t_r = [a_{t_r}, b_{t_r}]$, für dasjenige t_r mit $[s_k, C_k] \subseteq [a_{t_r}, b_{t_r}]$ (das benötigte Zeitintervall wird entfernt)
- Falls $b_{t_r} - C_k \geq 3$, dann $[C_k, b_{t_r}] \rightarrow E_r$ mit $p_r(C_k) := d(k)$
- Falls $s_k - a_{t_r} \geq 3$, dann $[a_{t_r}, s_k] \rightarrow E_r$ mit $p_r(s_k) := p_r(a_{t_r})$
- $s_{i(k)j(k)} := C_k$ (Startzeit der anschließenden Operation, muss gemäß der Verplanung angepasst werden)

B. Beispiel zum Ablauf des gesamten dynamischen Verfahrens (Phasen 1-3)

Im Folgenden veranschaulichen wir den Ablauf des gesamten dynamischen Verfahrens am Beispiel eines fiktiven Datensatzes. Bei der Lösung des Sequencing-Problems setzen wir dabei No-Wait-Constraints voraus. In Analogie zu den Rahmenbedingungen des Prozesslabors betrachten wir zudem Maschinen, die entlang eines linearen Tracks angesiedelt sind und von Transportrobotern bedient werden. Alle Jobs werden dabei über einen einzigen Eingang in das System geschleust und verlassen dieses genauso über einen einzigen Ausgang. Bei der Darstellung der einzelnen Rechenschritte liegt der Fokus in erster Linie auf der dritten Phase des Verfahrens sowie der Vorgehensweise zu Beginn einer Reoptimierung. Die Phasen 1 und 2 wurden bereits in Kapitel 5 anhand ausführlicher Beispiele veranschaulicht, entsprechende Lösungen werden hier daher nur in kompakter Form angegeben. Wir verwenden bei sämtlichen Ausführungen die Notationen aus Anhang A.

B.1. Beispielinstanz

Wir betrachten fünf Jobs und drei Bearbeitungsstufen ($M^{(1)} = 2, M^{(2)} = 1, M^{(3)} = 3$) sowie zwei Transportroboter. Alle Jobs durchlaufen die verschiedenen Stufen in der Reihenfolge $1 \rightarrow 2 \rightarrow 3$. Die Anordnung der einzelnen Maschinen entlang des Tracks sowie die Ausgangspositionen der beiden Roboter werden in Abbildung B.1 skizziert. Die Transport- oder Leerfahrtzeit zwischen zwei Maschinen bzw. Positionen ergibt sich aus der Differenz der Positionsnummern dividiert durch die Robotergeschwindigkeit v_R, welche wir hier der Einfachheit halber als $v_R = 1$ wählen. Die Nummer der Maschine eines Typs entspricht gleichzeitig deren Güteklasse. Es liegen ferner die in Tabelle B.1 angegebenen Jobdaten zugrunde.

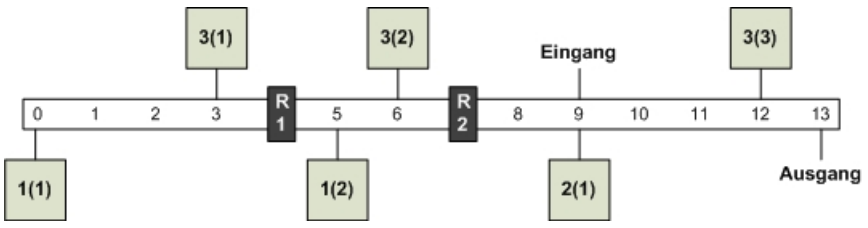

Abbildung B.1.: Maschinenanordnung der Beispielinstanz

Job	**1**	**2**	**3**	**4**	**5**
p_{1j}	1	8	6	6	7
p_{2j}	2	10	3	2	9
p_{3j}	8	2	9	10	3
w_j	3	5	5	1	2
r_j	8	2	15	13	16

Tabelle B.1.: Jobspezifische Daten der Beispielinstanz

B.2. Ablauf des Verfahrens

1. Durchlauf:

$t = 2$, $J(2) = \{2\}$. Die Start- und Fertigstellungszeiten des Jobs 2 in Phase 1 ergeben sich unter Anwendung des List Scheduling Algorithmus' für No-Wait-Constraints als: $s_{12} = 2, C_{12} = s_{22} = 10, C_{22} = s_{32} = 20, C_{32} = 22$. Des Weiteren werden für die drei Operationen die Maschinen $1(2), 2(1)$ und $3(3)$ mit höchster Güteklasse ausgewählt und wir erhalten den in Abbildung B.2 dargestellten Reihenfolgegraphen. Für alle Transportaufträge wird nun Roboter 2 ausgewählt, da dieser den kürzeren Anfahrtsweg von seiner Ausgangsposition zum Startort des ersten Auftrags besitzt und anschließend genauso für die verbleibenden Aufträge (vgl. Abbildung B.3).

2. Durchlauf:

$t = 8$, $J(8) = \{1\}$. Zum Zeitpunkt $t = 8$ ist Job 2 in der Lösung des Routing-Problems bereits in Bearbeitung, d. h. er wird bei der anstehenden Resequenzierung in Phase 1 nicht berücksichtigt. Blockiert wird jedoch lediglich die Operation $(1,2)$, weshalb die beiden anderen Operationen des Jobs 2 innerhalb des List Scheduling Verfahrens genauso wie in den Phasen 2 und 3 erneut berücksichtigt werden.

Wir übernehmen für Job 2 zunächst die nun bekannte tatsächliche Startzeit auf der ersten Stufe ($s_{12} = 8, C_{12} = 16$) und aktualisieren entsprechend die Maschinenkapazitäten. Für alle anderen Operationen wird basierend auf der Jobsequenz $2 \rightarrow 1$ nun der List Scheduling Algorithmus durchgeführt. Dieser startet jedoch nicht mit dem Wert $t = 8$, sondern der Beginn wird noch um die bisher größte Transport- und Wartezeit eines Jobs verschoben. Für Job 2 haben wir insgesamt $t - r_j = 8 - 2 = 6$ Zeiteinheiten eingefügt, in denen

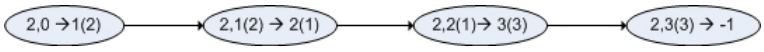

Abbildung B.2.: Reihenfolgegraph zum Zeitpunkt $t = 2$

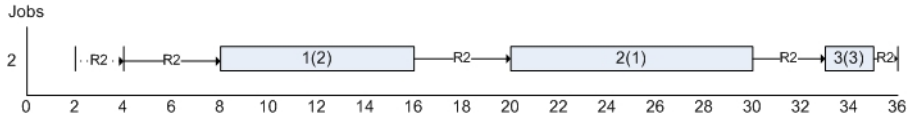

Abbildung B.3.: Routing zum Zeitpunkt $t = 2$

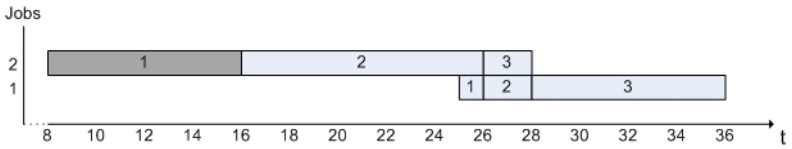

Abbildung B.4.: Resequenzierung zum Zeitpunkt $t = 8$

der Job wartet oder transportiert wird. Demnach startet der List Scheduling Algorithmus nun mit dem Wert $t = 8+6 = 14$. Auf diese Weise ergibt sich die in Abbildung B.4 dargestellte Lösung des Sequencing-Problems. Blockierte Operationen sind darin dunkelgrau markiert.

Es werden zu keiner Zeit zwei Jobs auf derselben Stufe bearbeitet, daher werden alle Operationen jeweils der Maschine mit höchster Güteklasse zugeordnet. Wir wählen also für beide Jobs die Maschinenfolge $1(2), 2(1), 3(3)$, wobei jede dieser Maschinen zuerst Job 2 und dann Job 1 bearbeitet. Im Folgenden nummerieren wir die entstehenden Transportaufgaben gemäß Tabelle B.2 und erhalten so den in Abbildung B.5 dargestellten Reihenfolgegraphen. Transportauftrag 1 ist zum Zeitpunkt $t = 8$ bereits bearbeitet und wird beim nun anstehenden Rerouting nicht mehr berücksichtigt. Da Job 2 zu diesem Zeitpunkt jedoch noch nicht vollständig abgeschlossen ist, verbleiben alle zugehörigen (blockierten) Transportaufträge im Reihenfolgegraphen, da die entsprechenden Reihenfolgebedingungen für nachfolgende Aufträge natürlich trotzdem eingehalten werden müssen. Blockierte Transporte werden in Phase 3 daher von Beginn an in der Menge L^* der verplanten Aufträge geführt (vgl. Abbildung B.5).

Im Teilgraphen der unbearbeiteten Aufträge gibt es nur einen Knoten ohne Vorgänger, wir beginnen bei der Routenplanung daher mit $L = \{2\}$ und $L^* = \{1\}$, wobei $s_1 = 2$, $\bar{s}_1 = 4$ und $C_1 = 8$. Der Auftrag 2 hat die Freigabezeit $r_2 = C_1 + d_1 = 8+8 = 16$. Des Weiteren haben wir die Roboterbelegung $E_1 = E_2 = \{[8, T]\}$, mit den Roboterpositionen $p_1(8) = 5$,

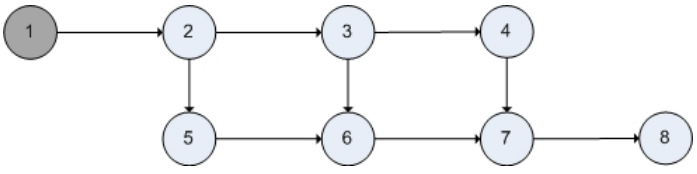

Abbildung B.5.: Reihenfolgegraph zum Zeitpunkt $t = 8$

Nr.	Transportauftrag
1	$(2,0(1) \rightarrow 1(2))$
2	$(2,1(2) \rightarrow 2(1))$
3	$(2,2(1) \rightarrow 3(3))$
4	$(2,3(3) \rightarrow -1(1))$
5	$(1,0(1) \rightarrow 1(2))$
6	$(1,1(2) \rightarrow 2(1))$
7	$(1,2(1) \rightarrow 3(3))$
8	$(1,3(3) \rightarrow -1(1))$

Tabelle B.2.: Transportaufträge zum Zeitpunkt $t = 8$

$p_2(8) = 4$. Mit Hilfe dieser Daten erhalten wir die folgenden frühestmöglichen Startzeiten für den Auftrag 2 unter Verwendung der beiden Roboter:

$$f_2(1) = 16 - \min(16-8,1) + 1 = 16$$
$$f_2(2) = 16 - \min(16-8,0) + 0 = 16$$

Wir können nun also die Prioritätsindizes bestimmen und erhalten:

$$\kappa_{2,1} = 16 + \tfrac{1+4}{5} = 17,\ \kappa_{2,2} = 16 + \tfrac{0+4}{5} = 16\tfrac{4}{5}$$

Dabei ist $\kappa_{2,2} < \kappa_{2,1}$, wir wählen daher Roboter 2 zur Durchführung des Transports 2. Daraus ergeben sich die folgenden neuen Werte für die verschiedenen Parameter der dritten Phase:

$L = \{3,5\}$, $L^* = \{1,2\}$,

$s_2 = 16$, $\bar{s}_2 = 16$, $C_2 = 20$,

$E_1 = \{[8,T]\}$, $p_1(8) = 4$

$E_2 = \{[8,16],[20,T]\}$, $p_2(8) = p_2(16) = 4$, $p_2(20) = 9$

Wir verzichten an dieser Stelle auf die Darstellung der weiteren Rechenschritte, da der größte Teil der hier berechneten Lösung des Routing-Problems (vgl. Abbildung B.6) bei der nächsten Reoptimierung wieder verworfen wird. Wir werden jedoch in einer späteren Iteration detailliert auf sämtliche notwendigen Rechenschritte eingehen.

3./4. Durchlauf:

$t = 13$, $J(13) = \{4\}$. Zum Zeitpunkt $t = 13$ hat die Anfahrt für den ersten Transport des Jobs 1 bereits begonnen. Dieser wird bei der Resequenzierung in Phase 1 also nicht

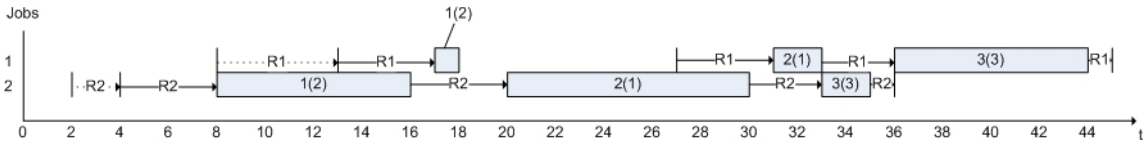

Abbildung B.6.: Routing zum Zeitpunkt $t = 8$

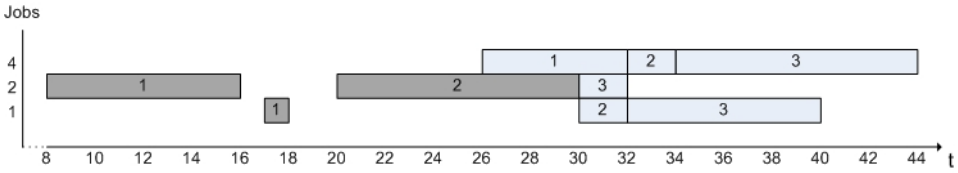

Abbildung B.7.: Resequenzierung zum Zeitpunkt $t = 13$

erneut betrachtet. Genauso befinden sich die Transporte 1 und 5 zu diesem Zeitpunkt schon in Bearbeitung. Aus diesem Grund werden beide Aufträge sowie der Transport 2 (als Vorgänger von 5) blockiert und nicht mehr neu verplant.

Wir wenden nun das List Scheduling Verfahren auf die Sequenz $2 \rightarrow 1 \rightarrow 4$ an, wobei zuvor die Operationen $(1,1)$, $(2,1)$ und $(1,2)$ mit ihren tatsächlichen Startzeiten übernommen werden. Der Ausgangswert für das List Scheduling Verfahren zur Bestimmung der Startzeiten der übrigen Operationen der Jobs 1 und 2 sowie des Jobs 4 ergibt sich nun als Summe aus dem aktuellen Zeitpunkt $t = 13$ und der maximal eingefügten Transport- und Wartezeit eines Jobs. Für Job 1 ist bisher lediglich ein Transportauftrag (Nr. 5 mit $s_5 = 8$, $C_5 = 18$) und daher eine Transport- und Wartezeit von neun Zeiteinheiten eingeplant. Für Job 2 sind es zwei Aufträge (Nr. 1 und 2 mit $s_1 = 2$, $C_1 = 8$, $s_2 = 16$, $C_2 = 20$). Dieser Job wartet bislang also in zehn Zeiteinheiten bzw. wird darin transportiert. D. h. wir starten das List Scheduling Verfahren mit dem Wert 23. Auf diese Weise erhalten wir in Phase 1 den in Abbildung B.7 dargestellten Schedule. Bei der Maschinenauswahl wird erneut jede Operation derjenigen Maschine mit größter Güteklasse zugewiesen, mit Ausnahme der Bearbeitungsstufe 3. In diesem Fall werden die Jobs 1 und 2 von Maschine 3(3) bearbeitet und Job 4 wird der Maschine 3(2) zugeordnet.

Da zu den Zeitpunkten $t = 15$ und $t = 16$ bereits die Jobs 3 und 5 eintreffen und in jedem Schritt das gesamte System mit Ausnahme der bisher verplanten Operationen und Transportaufträge reoptimiert werden muss, verzichten wir an dieser Stelle auf eine ausführliche Darstellung der entsprechenden Zwischenschritte und gehen sofort zum Zeitpunkt $t = 16$ über.

5. Durchlauf:

$t = 16$, $J = \{3,4,5\}$. Bei der Resequenzierung besteht dieselbe Ausgangssituation wie zum Zeitpunkt $t = 13$, d. h. die Operationen $(1,1)$, $(2,1)$ und $(1,2)$ werden erneut mit

Sequenz	C_{33}	C_{34}	C_{35}	**ZF-Wert**
$3 \rightarrow 4 \rightarrow 5$	44	47	51	369
$3 \rightarrow 5 \rightarrow 4$	44	57	47	371
$4 \rightarrow 3 \rightarrow 5$	46	44	51	376
$4 \rightarrow 5 \rightarrow 3$	54	44	45	404
$5 \rightarrow 3 \rightarrow 4$	54	57	45	417
$5 \rightarrow 4 \rightarrow 3$	56	54	45	424

Tabelle B.3.: Ergebnisse des List Scheduling Verfahrens zum Zeitpunkt $t = 16$

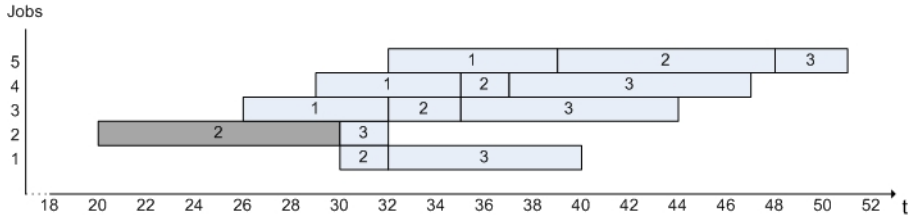

Abbildung B.8.: Resequenzierung zum Zeitpunkt $t = 16$

ihren tatsächlichen Startzeiten übernommen und die übrigen Operationen der Jobs 1 und 2 entsprechend nach hinten verschoben. Der Startwert für das List Scheduling Verfahren ergibt sich dabei erneut als Summe aus dem aktuellen Zeitpunkt und der maximal eingefügten Transport- und Wartezeit von zehn Zeiteinheiten, hier also 26. Es sind nur drei Jobs neu zu sequenzieren, d. h. auf alle sechs möglichen Sequenzen wird der List Scheduling Algorithmus angewendet und die Sortierung mit kleinstem Zielfunktionswert wird ausgewählt. Es ergeben sich dabei die in Tabelle B.3 aufgelisteten Daten. Es wird demnach die Reihenfolge $3 \rightarrow 4 \rightarrow 5$ verwendet. Die zugehörige Lösung aus Phase 1 ist in Abbildung B.8 dargestellt.

Des Weiteren zeigt Abbildung B.9 die Maschinenauswahl für die Stufen 1 und 3.

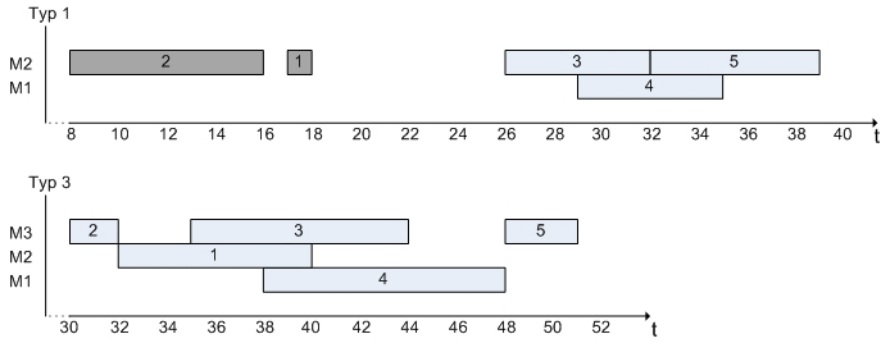

Abbildung B.9.: Maschinenauswahl zum Zeitpunkt $t = 16$

Nr.	Transportauftrag
1	$(2,0(1) \rightarrow 1(2))$
2	$(2,1(2) \rightarrow 2(1))$
3	$(2,2(1) \rightarrow 3(3))$
4	$(2,3(3) \rightarrow -1(1))$
5	$(1,0(1) \rightarrow 1(2))$
6	$(1,1(2) \rightarrow 2(1))$
7	$(1,2(1) \rightarrow 3(2))$
8	$(1,3(2) \rightarrow -1(1))$
9	$(3,0(1) \rightarrow 1(2))$
10	$(3,1(2) \rightarrow 2(1))$
11	$(3,2(1) \rightarrow 3(3))$
12	$(3,3(3) \rightarrow -1(1))$
13	$(4,0(1) \rightarrow 1(1))$
14	$(4,1(1) \rightarrow 2(1))$
15	$(4,2(1) \rightarrow 3(1))$
16	$(4,3(1) \rightarrow -1(1))$
17	$(5,0(1) \rightarrow 1(2))$
18	$(5,1(2) \rightarrow 2(1))$
19	$(5,2(1) \rightarrow 3(3))$
20	$(5,3(3) \rightarrow -1(1))$

Tabelle B.4.: Transportaufgaben zum Zeitpunkt $t = 16$

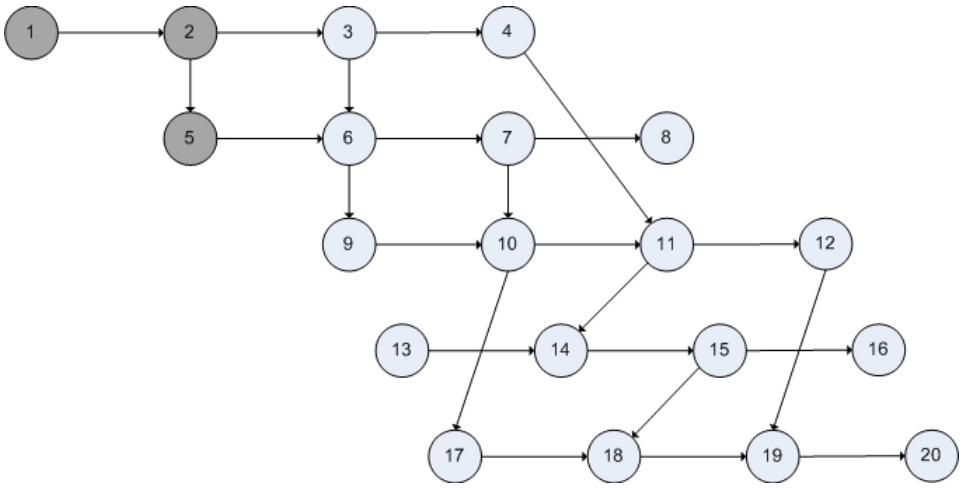

Abbildung B.10.: Reihenfolgegraph zum Zeitpunkt $t = 16$

Die grau markierten Operationen werden dabei nicht neu zugewiesen und sind nur der Vollständigkeit halber abgebildet. In Tabelle B.4 sind die entsprechend zu verplanenden Transportaufgaben aufgelistet. Der resultierende Reihenfolgegraph ist in Abbildung B.10 skizziert. Wir führen nun die Routenplanung (Phase 3) durch und geben dabei die Zwischenergebnisse jeder Iteration an:

1. Iteration

$L = \{3, 13\}$
$E_1 = \{[17, T]\} \quad p_1(17) = 5$
$E_2 = \{[20, T]\} \quad p_2(20) = 9$

Freigabezeiten:

$\overline{r}_3 = 30$, $\overline{r}_{13} = 16$

Frühestmögliche Startzeiten:

$f_3(1) = 30 - \min(30-17, 4) + 4 = 30 \quad f_{13}(1) = 17 + 4 = 21$
$f_3(2) = 30 - \min(30-20, 0) + 0 = 30 \quad f_{13}(2) = 20 + 0 = 20$

Prioritätsindizes:

$\kappa_{3,1} = 30 + \frac{4+3}{5} = 31\frac{2}{5} \quad \kappa_{13,1} = 21 + \frac{4+9}{1} = 34$
$\kappa_{3,2} = 30 + \frac{0+3}{5} = 30\frac{3}{5} \quad \boxed{\kappa_{13,2} = 20 + \frac{0+9}{1} = 29}$

Wir wählen also Roboter 2 zur Durchführung des Transports 13 und erhalten so:

$s_{13} = 20$, $\overline{s}_{13} = 20$, $C_{13} = 29$.

2. Iteration

$L = \{3\}$
$E_1 = \{[17, T]\} \quad p_1(17) = 5$
$E_2 = \{[29, T]\} \quad p_2(29) = 0$

$\overline{r}_3 = 30$

$f_3(1) = 30 - \min(30-17, 4) + 4 = 30$
$f_3(2) = 30 - \min(30-29, 9) + 9 = 38$

$\boxed{\kappa_{3,1} = 30 + \frac{4+3}{5} = 31\frac{2}{5}}$
$\kappa_{3,2} = 38 + \frac{9+3}{5} = 40\frac{2}{5}$

3. Iteration

$L = \{4,6\}$
$E_1 = \{[17,26],[33,T]\}$ $p_1(17) = p_1(26) = 5$, $p_1(33) = 12$
$E_2 = \{[29,T]\}$ $p_2(29) = 0$

$\bar{r}_4 = C_3 + d_3 = 35$ $\bar{r}_6 = \max(C_5 + d_5, \bar{s}_3 + 1 - p_6) = 27$

$f_4(1) = 35 - \min(35-33,0) + 0 = 35$ $f_6(1) = 33 + 7 = 40$
$f_4(2) = 35 - \min(35-29,12) + 12 = 41$ $f_6(2) = 29 + 5 = 34$

$\boxed{\kappa_{4,1} = 35 + \frac{0+1}{5} = 35\frac{1}{5}}$ $\kappa_{6,1} = 40 + \frac{7+4}{3} = 43\frac{2}{3}$
$\kappa_{4,2} = 41 + \frac{12+1}{5} = 43\frac{3}{5}$ $\kappa_{6,2} = 34 + \frac{5+4}{3} = 37$

4. Iteration

$L = \{6\}$
$E_1 = \{[17,26],[36,T]\}$ $p_1(17) = p_1(26) = 5$, $p_1(36) = 13$
$E_2 = \{[29,T]\}$ $p_2(29) = 0$

$\bar{r}_6 = 27$

$f_6(1) = 36 + 8 = 44$
$f_6(2) = 29 + 5 = 34$

$\kappa_{6,1} = 44 + \frac{8+4}{3} = 48$
$\boxed{\kappa_{6,2} = 34 + \frac{5+4}{3} = 37}$

5. Iteration

$L = \{7,9\}$
$E_1 = \{[17,26],[36,T]\}$ $p_1(17) = p_1(26) = 5$, $p_1(36) = 13$
$E_2 = \{[38,T]\}$ $p_2(38) = 9$

$\bar{r}_7 = C_6 + d_6 = 40$ $\bar{r}_9 = \bar{s}_6 + 1 - p_9 = 31$

$f_7(1) = 40 - \min(40-36,4) + 4 = 40$ $f_9(1) = 36 + 4 = 40$
$f_7(2) = 40 - \min(40-38,0) + 0 = 40$ $f_9(2) = 38 + 0 = 38$

$\kappa_{7,1} = 40 + \frac{4+3}{3} = 42\frac{1}{3}$ $\kappa_{9,1} = 40 + \frac{4+4}{5} = 41\frac{3}{5}$
$\kappa_{7,2} = 40 + \frac{0+3}{3} = 41$ $\boxed{\kappa_{9,2} = 38 + \frac{0+4}{5} = 38\frac{4}{5}}$

6. Iteration

$L = \{7\}$
$E_1 = \{[17,26],[36,T]\}$ $p_1(17) = p_1(26) = 5$, $p_1(36) = 13$
$E_2 = \{[42,T]\}$ $p_2(42) = 5$

$\bar{r}_7 = 40$

$f_7(1) = 40 - \min(40-36,4) + 4 = 40$
$f_7(2) = 42 + 4 = 46$

$\boxed{\kappa_{7,1} = 40 + \frac{4+3}{3} = 42\frac{1}{3}}$
$\kappa_{7,2} = 46 + \frac{4+3}{3} = 48\frac{1}{3}$

7. Iteration

$L = \{8,10\}$
$E_1 = \{[17,26],[43,T]\}$ $p_1(17) = p_1(26) = 5$, $p_1(43) = 6$
$E_2 = \{[42,T]\}$ $p_2(42) = 5$

$\bar{r}_8 = C_7 + d_7 = 51$ $\bar{r}_{10} = \max(C_9 + d_9, \bar{s}_7 + 1 - p_{10}) = 48$

$f_8(1) = 51 - \min(51-43,0) + 0 = 51$ $f_{10}(1) = 48 - \min(48-43,1) + 1 = 48$
$f_8(2) = 51 - \min(51-42,1) + 1 = 51$ $f_{10}(2) = 48 - \min(48-42,0) + 0 = 48$

$\kappa_{8,1} = 51 + \frac{0+7}{3} = 53\frac{1}{3}$ $\kappa_{10,1} = 48 + \frac{1+4}{5} = 49$
$\kappa_{8,2} = 51 + \frac{1+7}{3} = 53\frac{2}{3}$ $\boxed{\kappa_{10,2} = 48 + \frac{0+4}{5} = 48\frac{4}{5}}$

8. Iteration

$L = \{8,11,17\}$
$E_1 = \{[17,26],[43,T]\}$ $p_1(17) = p_1(26) = 5$, $p_1(43) = 6$
$E_2 = \{[42,48],[52,T]\}$ $p_2(42) = p_2(48) = 5$, $p_2(52) = 9$

$\bar{r}_8 = 51$ $\bar{r}_{11} = \max(C_{10} + d_{10}, \bar{s}_4 + 1 - p_{11}) = 55$ $\bar{r}_{17} = \max(r_5, \bar{s}_{10} + 1 - p_{17}) = 45$

$f_8(1) = 51$ $f_{11}(1) = 55 - \min(55-43,3) + 3 = 55$
$f_8(2) = 52 + 3 = 55$ $f_{11}(2) = 55 - \min(55-52,0) + 0 = 55$
$f_{17}(1) = 45 - \min(45-43,3) + 3 = 46$
$f_{17}(2) = 52 + 0 = 52$

$\kappa_{8,1} = 51 + \frac{0+7}{3} = 53\frac{1}{3}$ $\kappa_{11,1} = 55 + \frac{3+3}{5} = 56\frac{1}{5}$ $\boxed{\kappa_{17,1} = 46 + \frac{3+4}{2} = 49\frac{1}{2}}$
$\kappa_{8,2} = 55 + \frac{3+7}{3} = 58\frac{1}{3}$ $\kappa_{11,2} = 55 + \frac{0+3}{5} = 55\frac{3}{5}$ $\kappa_{17,2} = 52 + \frac{0+4}{2} = 54$

9. Iteration

$L = \{8, 11\}$
$E_1 = \{[17,26],[50,T]\} \quad p_1(17) = p_1(26) = 5,\ p_1(50) = 5$
$E_2 = \{[42,48],[52,T]\} \quad p_2(42) = p_2(48) = 5,\ p_2(52) = 9$

$\bar{r}_8 = 51 \quad \bar{r}_{11} = 55$

$f_8(1) = 51 - \min(51-50,1) + 1 = 51 \quad f_{11}(1) = 55 - \min(55-50,4) + 4 = 55$
$f_8(2) = 52 + 3 = 55 \quad f_{11}(2) = 55 - \min(55-52,0) + 0 = 55$

$\boxed{\kappa_{8,1} = 51 + \frac{1+7}{3} = 53\frac{2}{3}} \quad \kappa_{11,1} = 55 + \frac{4+3}{5} = 56\frac{2}{5}$
$\kappa_{8,2} = 55 + \frac{3+7}{3} = 58\frac{1}{3} \quad \kappa_{11,2} = 55 + \frac{0+3}{5} = 55\frac{3}{5}$

10. Iteration

$L = \{11\}$
$E_1 = \{[17,26],[58,T]\} \quad p_1(17) = p_1(26) = 5,\ p_1(58) = 13$
$E_2 = \{[42,48],[52,T]\} \quad p_2(42) = p_2(48) = 5,\ p_2(52) = 9$

$\bar{r}_{11} = 55$

$f_{11}(1) = 58 + 4 = 62$
$f_{11}(2) = 55 - \min(55-52,0) + 0 = 55$

$\kappa_{11,1} = 62 + \frac{4+3}{5} = 63\frac{2}{5}$
$\boxed{\kappa_{11,2} = 55 + \frac{0+3}{5} = 55\frac{3}{5}}$

11. Iteration

$L = \{12, 14\}$
$E_1 = \{[17,26],[58,T]\} \quad p_1(17) = p_1(26) = 5,\ p_1(58) = 13$
$E_2 = \{[42,48],[52,55],[58,T]\} \quad p_2(42) = p_2(48) = 5,\ p_2(52) = p_2(55) = 9,\ p_2(58) = 12$

$\bar{r}_{12} = C_{11} + d_{11} = 67 \quad \bar{r}_{14} = \max(C_{13} + d_{13}, \bar{s}_{11} + 1 - p_{14}) = 47$

$f_{12}(1) = 67 - \min(67-58,1) + 1 = 67 \quad f_{14}(1) = 58 + 13 = 71$
$f_{12}(2) = 67 - \min(67-58,0) + 0 = 67 \quad f_{14}(2) = 58 + 12 = 70$

$\kappa_{12,1} = 67 + \frac{1+1}{5} = 67\frac{2}{5} \quad \kappa_{14,1} = 71 + \frac{13+9}{1} = 93$
$\boxed{\kappa_{12,2} = 67 + \frac{0+1}{5} = 67\frac{1}{5}} \quad \kappa_{14,2} = 70 + \frac{12+9}{1} = 91$

Zur Vereinfachung der Lesbarkeit werden in den nachfolgenden Iterationen nur noch die jeweils relevanten freien Zeitintervalle der Roboter in E_1 und E_2 aufgeführt.

12. Iteration

$L = \{14\}$
$E_1 = \{[58,T]\}$ $\quad p_1(58) = 13$
$E_2 = \{[58,67],[68,T]\}$ $\quad p_2(58) = p_2(67) = 12,\ p_2(68) = 13$

$\bar{r}_{14} = 47$

$f_{14}(1) = 58 + 13 = 71$
$f_{14}(2) = 68 + 13 = 81$

$\boxed{\kappa_{14,1} = 71 + \frac{13+9}{1} = 93}$
$\kappa_{14,2} = 81 + \frac{13+9}{1} = 103$

13. Iteration

$L = \{15\}$
$E_1 = \{[80,T]\}$ $\quad p_1(80) = 9$
$E_2 = \{[68,T]\}$ $\quad p_2(68) = 13$

$\bar{r}_{15} = C_{14} + d_{14} = 82$

$f_{15}(1) = 82 - \min(82-80,0) + 0 = 82$
$f_{15}(2) = 82 - \min(82-68,4) + 4 = 82$

$\boxed{\kappa_{15,1} = 82 + \frac{0+6}{1} = 88}$
$\kappa_{15,2} = 82 + \frac{4+6}{1} = 92$

14. Iteration

$L = \{16,18\}$
$E_1 = \{[88,T]\}$ $\quad p_1(88) = 3$
$E_2 = \{[68,T]\}$ $\quad p_2(68) = 13$

$\bar{r}_{16} = C_{15} + d_{15} = 98$ $\quad \bar{r}_{18} = \max(C_{17} + d_{17}, \bar{s}_{15} + 1 - p_{18}) = 79$

$f_{16}(1) = 98 - \min(98-88,0) + 0 = 98$ $\qquad f_{18}(1) = 88 + 2 = 90$
$f_{16}(2) = 98 - \min(98-68,10) + 10 = 98$ $\quad f_{18}(2) = 79 - \min(79-68,8) + 8 = 79$

$\kappa_{16,1} = 98 + \frac{0+10}{1} = 108$ $\qquad \kappa_{18,1} = 90 + \frac{2+4}{2} = 93$
$\kappa_{16,2} = 98 + \frac{10+10}{1} = 118$ $\quad \boxed{\kappa_{18,2} = 79 + \frac{8+4}{2} = 85}$

15. Iteration

$L = \{16, 19\}$
$E_1 = \{[88, T]\} \quad p_1(88) = 3$
$E_2 = \{[83, T]\} \quad p_2(83) = 9$

$\bar{r}_{16} = 98 \quad \bar{r}_{19} = \max(C_{18} + d_{18}, \bar{s}_{12} + 1 - p_{19}) = 92$

$f_{16}(1) = 98 - \min(98 - 88, 0) + 0 = 98 \quad f_{19}(1) = 92 - \min(92 - 88, 6) + 6 = 94$
$f_{16}(2) = 98 - \min(98 - 83, 6) + 6 = 98 \quad f_{19}(2) = 92 - \min(92 - 83, 0) + 0 = 92$

$\kappa_{16,1} = 98 + \frac{0+10}{1} = 108 \quad \kappa_{19,1} = 94 + \frac{6+3}{2} = 98\frac{1}{2}$
$\kappa_{16,2} = 98 + \frac{6+10}{1} = 114 \quad \boxed{\kappa_{19,2} = 92 + \frac{0+3}{2} = 93\frac{1}{2}}$

16. Iteration

$L = \{16, 20\}$
$E_1 = \{[88, T]\} \quad p_1(88) = 3$
$E_2 = \{[95, T]\} \quad p_2(95) = 12$

$\bar{r}_{16} = 98 \quad \bar{r}_{20} = C_{19} + d_{19} = 98$

$f_{16}(1) = 98 - \min(98 - 88, 0) + 0 = 98 \quad f_{20}(1) = 98 - \min(98 - 88, 9) + 9 = 98$
$f_{16}(2) = 98 - \min(98 - 95, 9) + 9 = 104 \quad f_{20}(2) = 98 - \min(98 - 95, 0) + 0 = 98$

$\kappa_{16,1} = 98 + \frac{0+10}{1} = 108 \quad \kappa_{20,1} = 98 + \frac{9+1}{2} = 103$
$\kappa_{16,2} = 104 + \frac{9+10}{1} = 123 \quad \boxed{\kappa_{20,2} = 98 + \frac{0+1}{2} = 98\frac{1}{2}}$

17. Iteration

$L = \{16\}$
$E_1 = \{[88, T]\} \quad p_1(88) = 3$
$E_2 = \{[99, T]\} \quad p_2(99) = 13$

$\bar{r}_{16} = 98$

$f_{16}(1) = 98 - \min(98 - 88, 0) + 0 = 98$
$f_{16}(2) = 99 + 10 = 109$

$\boxed{\kappa_{16,1} = 98 + \frac{0+10}{1} = 108}$
$\kappa_{16,2} = 109 + \frac{10+10}{1} = 129$

Der auf diese Weise konstruierte Schedule ist in Abbildung B.11 dargestellt. Des Weiteren sind in Tabelle B.5 die entsprechenden Start- und Fertigstellungszeiten der Transportaufträge aufgelistet.

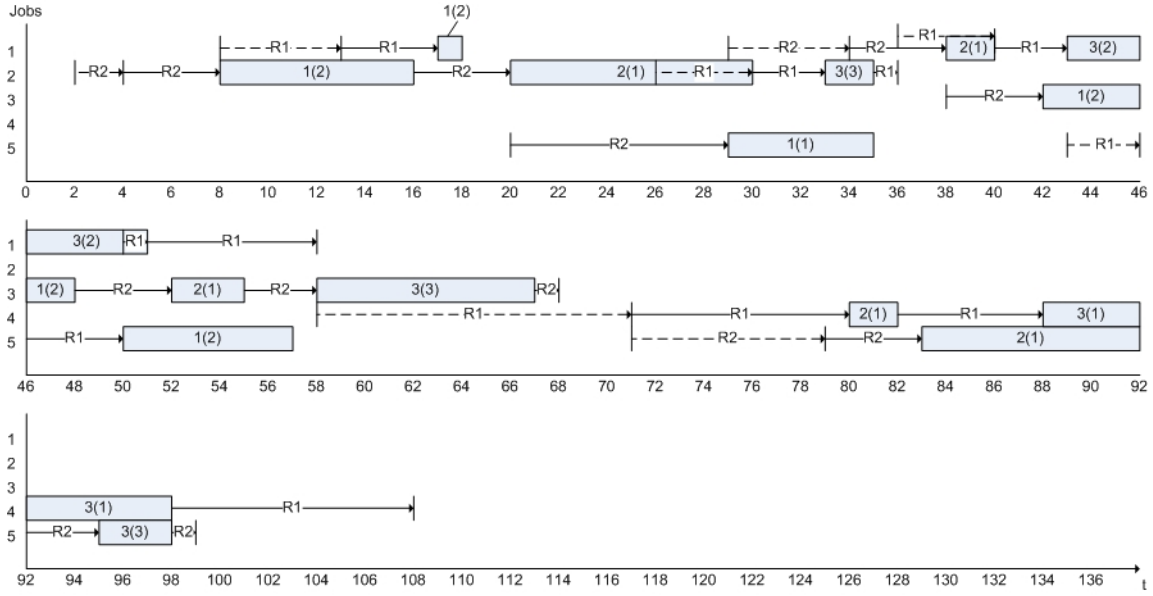

Abbildung B.11.: Finaler Schedule für die Beispielinstanz

Nr.	Transportauftrag	s_k	$\bar{s}_k$	C_k	Roboter
1	$(2,0(1) \to 1(2))$	2	4	8	2
2	$(2,1(2) \to 2(1))$	16	16	20	2
3	$(2,2(1) \to 3(3))$	26	30	33	1
4	$(2,3(3) \to -1(1))$	35	35	36	1
5	$(1,0(1) \to 1(2))$	8	13	17	1
6	$(1,1(2) \to 2(1))$	29	34	38	2
7	$(1,2(1) \to 3(2))$	36	40	43	1
8	$(1,3(2) \to -1(1))$	50	51	58	1
9	$(3,0(1) \to 1(2))$	38	38	42	2
10	$(3,1(2) \to 2(1))$	48	48	52	2
11	$(3,2(1) \to 3(3))$	55	55	58	2
12	$(3,3(3) \to -1(1))$	67	67	68	2
13	$(4,0(1) \to 1(1))$	20	20	29	2
14	$(4,1(1) \to 2(1))$	58	71	80	1
15	$(4,2(1) \to 3(1))$	82	82	88	1
16	$(4,3(1) \to -1(1))$	98	98	108	1
17	$(5,0(1) \to 1(2))$	43	46	50	1
18	$(5,1(2) \to 2(1))$	71	79	83	2
19	$(5,2(1) \to 3(3))$	92	92	95	2
20	$(5,3(3) \to -1(1))$	98	98	99	2

Tabelle B.5.: Lösung des Routing-Problems zum Zeitpunkt $t = 16$

Literaturverzeichnis

[1] ADAMS, J. ; BALAS, E. ; ZAWACK, D.: The shifting bottleneck procedure for job shop scheduling. In: *Management Science* 34 (1988), Nr. 3, S. 391 – 401

[2] ADLER, L. ; FRAIMAN, N. ; KOBACKER, E. ; PINEDO, M. ; PLOTNICOFF, J.C. ; WU, T.P.: BPSS: A scheduling support system for the packing industry. In: *Operations Research* 41 (1993), Nr. 4, S. 641 – 648

[3] AGNETIS, A. ; PACCIARELLI, D.: Part sequencing in three-machine no-wait robotic cells. In: *Operations Research Letters* 27 (2000), S. 185 – 192

[4] AGNETIS, A. ; PACIFICI, A. ; ROSSI, F. ; LUCERTINI, M. ; NICOLETTI, S. ; NICOLÒ, F. ; ORIOLO, G. ; PACCIARELLI, D. ; PESARO, E.: Scheduling of flexible flow lines in an automobile assembly plant. In: *European Journal of Operational Research* 97 (1997), S. 348 – 362

[5] AL-ANZI, F.S. ; ALLAHVERDI, A.: Scheduling in flexible multi-server internet services. In: *6th Saudi Engineering Conference, KFUPM* 4 (2002), S. 41 – 53

[6] ALLAHVERDI, A. ; GUPTA, J.N.D. ; ALDOWAISAN, T.: A review of scheduling research involving setup considerations. In: *OMEGA - The International Journal of Management Sciences* 27 (1999), S. 219 – 239

[7] ALLAHVERDI, A. ; NG, C.T. ; CHENG, T.C.E. ; KOVALYOV, M.Y.: A survey of scheduling problems with setup times or costs. In: *European Journal of Operational Research* 187 (2008), Nr. 3, S. 985 – 1032

[8] ALVAREZ-VALDES, R. ; FUERTES, A. ; TAMARIT, J.M. ; GIMÉNEZ, G. ; RAMOS, R.: A heuristic to schedule flexible job-shop in a glass factory. In: *European Journal of Operational Research* 165 (2005), S. 525 – 534

[9] ANDERSON, E.J. ; POTTS, C.N.: Online Scheduling of a single machine to minimize total weighted completion time. In: *Mathematics of Operations Research* 29 (2004), Nr. 3, S. 686 – 697

[10] AZIZOGLU, M. ; KIRCA, O.: On the minimization of total weighted flow time with identical and uniform parallel machines. In: *European Journal of Operational Research* 113 (1999), S. 91 – 100

[11] AZIZOĞLU, M. ; ÇAKMAK, E. ; KONDAKCI, S.: A flexible flow shop problem with total flowtime minimization. In: *European Journal of Operational Research* 132 (2001), Nr. 3, S. 528 – 538

[12] BAEV, I.D. ; MELEIS, W.M. ; EICHENBERGER, A.: An experimental study of algorithms for weighted completion time scheduling. In: *Algorithmica* 33 (2002), S. 34 – 51

[13] BAI, D. ; TANG, L.: Worst case analysis of a new lower bound for flow shop weighted completion time problem. In: *Lecture Notes in Computer Science* 4616 (2007), S. 191 – 199

[14] BAPTISTE, P. ; JOUGLET, A. ; SAVOUREY, D.: Lower bounds for parallel machine scheduling problems. In: *International Journal of Operational Research* 3 (2008), S. 643 – 664

[15] BAYKASOĞLU, A. ; ÖZBAKIR, L. ; SÖNMEZ, A.I.: Using multiple objective tabu search and grammars to model and solve multi-objective flexible job shop scheduling problems. In: *Journal of Intelligent Manufacturing* 15 (2004), S. 777 – 785

[16] BELOUADAH, H. ; POSNER, M.E. ; POTTS, C.N.: Scheduling with release dates on a single machine to minimize total weighted completion time. In: *Discrete Applied Mathematics* 36 (1992), S. 213 – 231

[17] BELOUADAH, H. ; POTTS, C.N.: Scheduling identical parallel machines to minimize total weighted completion time. In: *Discrete Applied Mathematics* 48 (1994), S. 201 – 218

[18] BERTEL, S. ; BILLAUT, J.-C.: A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation. In: *European Journal of Operational Research* 159 (2004), S. 651 – 662

[19] BIANCO, L. ; DELL'OLMO, P. ; GIORDANI, S.: Scheduling models for air traffic control in terminal areas. In: *Journal of Scheduling* 9 (2006), S. 223 – 253

[20] BIGRAS, L.-P. ; GAMACHE, M. ; SAVARD, G.: Time-indexed formulations and the total weighted tardiness problem. In: *INFORMS Journal on Computing* 20 (2008), Nr. 1, S. 133 – 142

[21] BLAŻEWICZ, J. ; DOMSCHKE, W. ; PESCH, E.: The job shop scheduling problem: Conventional and new solution techniques. In: *European Journal of Operational Research* 93 (1996), S. 1 – 33

[22] BÜLBÜL, K. ; KAMINSKY, P. ; YANO, C.: Flow shop scheduling with earliness, tardiness, and intermediate inventory holding costs. In: *Naval Research Logistics* 51 (2004), S. 407 – 445

[23] BOTTA-GENOULAZ, V.: Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness. In: *International Journal of Production Economics* 64 (2000), S. 101 – 111

[24] BRAH, S.A.: A comparative analysis fo due date based job sequencings rules in a flow shop with multiple processors. In: *Production Planning & Control* 7 (1996), Nr. 4, S. 362 – 373

[25] BRAH, S.A. ; HUNSUCKER, J.L.: Branch and bound algorithm for the flow shop with multiple processors. In: *European Journal of Operational Research* 51 (1991), S. 88 – 99

[26] BRAH, S.A. ; LOO, L.L.: Heuristics for scheduling in a flow shop with multiple processors. In: *European Journal of Operational Research* 113 (1999), S. 113 – 122

[27] BRAH, S.A. ; WHEELER, G.E.: Comparison of Scheduling Rules in a Flow Shop with Multiple Processors: A Simulation. In: *Simulation* 71 (1998), Nr. 5, S. 302 – 311

[28] BRUCKER, P. ; HEITMANN, S. ; HURINK, J. ; NIEBERG, T.: Job-shop scheduling with limited capacity buffers. In: *OR Spectrum* 28 (2006), S. 151 – 176

[29] BRUCKER, P. ; KAMPMEYER, T.: A general model for cyclic machine scheduling problems. In: *Discrete Applied Mathematics* 156 (2008), S. 2561 – 2572

[30] BRUNO, J. ; COFFMAN, E. G. Jr. ; SETHI, R.: Scheduling independent tasks to reduce mean finishing time. In: *Communications of the ACM* 17 (1974), Nr. 7, S. 382 – 387

[31] CARLIER, J. ; HAOUARI, M. ; KHARBECHE, M. ; MOUKRIM, A.: An optimization-based heuristic for the robotic cell problem. In: *European Journal of Operational Research* 202 (2010), S. 636 – 645

[32] CHAKRABARTI, S. ; PHILLIPS, C.A. ; SCHULZ, A.S. ; SHMOYS, D.B. ; STEIN, C. ; WEIN, J.: Improved Scheduling Algorithms for Minsum Criteria. In: *Lecture Notes in Computer Science* (1996), S. 646 – 657

[33] CHEKURI, C. ; MOTWANI, R. ; NATARAJAN, B. ; STEIN, C.: Approximation techniques for average completion time scheduling. In: *SIAM Journal on Computing* 31 (2001), S. 146 – 166

[34] CHEN, T.-L. ; POWELL, W.B.: Exact algorithms for scheduling multiple families of jobs on parallel machines. In: *Naval Research Logistics* 50 (2003), S. 823 – 840

[35] CHOU, M.C. ; QUEYRANNE, M. ; SIMCHI-LEVI, D.: The asymptotic performance ratio of an on-line algorithm for uniform parallel machine scheduling with release dates. In: *Mathematical Programming* 106 (2006), S. 137 – 157

[36] CORMEN, T.H. ; LEISERSON, C.E. ; RIVEST, R. ; STEIN, C.: *Algorithmen - Eine Einführung*. 2.Auflage. München : Oldenbourg Wissenschaftsverlag, 2004

[37] DAWANDE, M. ; GEISMAR, H.N. ; SETHI, S.P. ; SRISKANDARAJAH, C.: Sequencing and scheduling in robotic cells: Recent developments. In: *Journal of Scheduling* 8 (2005), S. 387 – 426

[38] DING, F.-Y. ; KITTICHARTPHAYAK, D.: Heuristics for scheduling flexible flow lines. In: *Computers and Industrial Engineering* 26 (1994), Nr. 1, S. 27 – 34

[39] DYER, M.E. ; WOLSEY, L.A.: Formulating the single machine sequencing problem with release dates as a mixed integer program. In: *Discrete Applied Mathematics* 26 (1990), S. 255 – 270

[40] EASTMAN, W. L. ; EVEN, S. ; ISAACS, I. M.: Bounds for the optimal scheduling of *n* jobs on *m* processors. In: *Management Science* 11 (1964), Nr. 2, S. 268 – 279

[41] FRENCH, S.: *Sequencing and Scheduling - An Introduction to the Mathematics of the Job-Shop*. West-Sussex : Ellis Horwood Limited. John Wiley & Sons, 1982

[42] GHOSH, J.B.: Batch scheduling to minimize total weighted completion time. In: *Operations Research Letters* 16 (1994), S. 271 – 275

[43] GOEMANS, M.X.: A supermodular relaxation for scheduling with release dates, in. In: QUEYRANNE, M. (Hrsg.): *Integer Programming and Combinatorial Optimization*. Heidelberg : Springer, 1996, S. 288 – 300

[44] GOMES, M.C. ; BARBOSA-PÓVOA, A.P. ; NOVAIS, A.Q.: Optimal Scheduling for Flexible Job Shop Operation. In: *International Journal of Production Research* 43 (2005), Nr. 11, S. 2323 – 2353

[45] GRAHAM, R.L.: Bounds on multiprocessing timing anomalies. In: *SIAM Journal on Applied Mathematics* 17 (1969), S. 416 – 429

[46] GRAHAM, R.L. ; LAWLER, E.L. ; LENSTRA, J.K. ; RINNOOY KAN, A.: Optimization and approximation in deterministic sequencing and scheduling: A survey. In: *Annals of Discrete Mathematics* 5 (1979), S. 287 – 326

[47] GRÖFLIN, H. ; PHAM, D.N. ; BÜRGY, R.: The flexible blocking job shop with transfer and set-up times. In: *Journal of Combinatorial Optimization* (online first) (2009)

[48] GUINET, A. ; SOLOMON, M.: Scheduling hybrid flowshops to minimize maximum tardiness or maximum completion time. In: *International Journal of Production Research* 34 (1996), Nr. 6, S. 1643 – 1654

[49] HALL, L.A. ; SCHULZ, A.S. ; SHMOYS, D.B. ; WEIN, J.: Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. In: *Mathematics of Operations Research* 22 (1997), Nr. 2, S. 513 – 544

[50] HALL, N.G. ; SRISKANDARAJAH, C.: A survey of machine scheduling problems with blocking and no-wait in progress. In: *Operations Research* 44 (1996), Nr. 3, S. 510 – 525

[51] HAVILL, J.T. ; MAO, W.: On-line algorithms for hybrid flow shop scheduling. In: *Proceedings of the Fourth International Conference on Computer Science and Informatics* (1998), S. 134 – 137

[52] HO, J.C.: Flowshop sequencing with mean flowtime objective. In: *European Journal of Operational Research* 81 (1995), S. 571 – 578

[53] HOOGEVEEN, H. ; VAN DE VELDE, S.: Formulating a scheduling problem with almost identical jobs by using positional completion times. In: BALAS, E. (Hrsg.) ; CLAUSEN, J. (Hrsg.): *Integer Programming and Combinatorial Optimization, Proceedings of the 4th International IPCO Conference*. Heidelberg : Springer, 1995, S. 292 – 306

[54] HOOGEVEEN, H. ; VAN NORDEN, L. ; VAN DEN VELDE, S.: Lower bounds for minimizing total weighted completion time in a two-machine flow shop. In: *Journal of Scheduling* 9 (2006), S. 559 – 568

[55] HOOGEVEEN, J.A. ; VAN DE VELDE, S.L.: Stronger Lagrangian bounds by use of slack variables: Applications to machine scheduling problems. In: *Mathematical Programming* 70 (1995), S. 173 – 190

[56] HUNSUCKER, J.L. ; SHAH, J.R.: Comparative performance analysis of priority rules in a constrained flow shop with multiple processors environment. In: *European Journal of Operational Research* 72 (1994), S. 102 – 114

[57] HUO, Y. ; LEUNG, J. Y.-T.: Online scheduling of precedence constrained tasks. In: *SIAM Journal on Computing* 34 (2005), Nr. 3, S. 743 – 762

[58] HURINK, J. ; KNUST, S.: List scheduling in a parallel machine environment with precedence constraints and setup times. In: *Operations Research Letters* 29 (2001), S. 231 – 239

[59] JAYAMOHAN, M.S. ; RAJENDRAN, C.A.: A comparative analysis of two different approaches to scheduling in flexible flow shops. In: *Production Planning & Control* 11 (2000), Nr. 6, S. 572 – 580

[60] JOHNSON, S. M.: Optimal two- and three-stage production schedules with setup times included. In: *Naval Research Logistic Quarterly* 1 (1954), S. 61 – 68

[61] JOUGLET, A. ; BAPTISTE, P. ; CARLIER, J.: Branch-and-bound algorithms for total weighted tardiness. In: LEUNG, J. Y.-T. (Hrsg.): *Handbook of Scheduling. Algorithms, Models and Performance Analysis*. Boca Raton : Chapman & Hall/CRC, 2004

[62] JUNGWATTANAKIT, J. ; REODECHA, M. ; CHAOVALITWONGSE, P. ; WERNER, F.: Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria. In: *International Journal of Advanced Manufacturing Technologies* 37 (2008), S. 354 – 370

[63] JUNGWATTANAKIT, J. ; REODECHA, M. ; CHAOVALITWONGSE, P. ; WERNER, F.: A comparison of scheduling algorithms for flexible flow shop problems with unrelated parallel machines, setup times and dual criteria. In: *Computers & Operations Research* 36 (2009), S. 358 – 378

[64] KADIPASAOGLU, S.N. ; XIANG, W. ; KHUMAWALA, B.M.: A comparison of sequencing rules in static and dynamic hybrid flow systems. In: *International Journal of Production Research* 35 (1997), Nr. 5, S. 1359 – 1384

[65] KALLRATH, J.: *Gemischt-ganzzahlige Optimierung: Modellierung in der Praxis*. Braunschweig : vieweg, 2002

[66] KAMINSKY, P. ; SIMCHI-LEVI, D.: Probalistic analysis and practical algorithms for the flow shop weighted completion time problem. In: *Operations Research* 46 (1998), Nr. 6, S. 872 – 882

[67] KAWAGUCHI, T. ; KYAN, S.: Worst case bound of a LRF for the mean weighted flowtime problem. In: *SIAM Journal on Computing* 15 (1986), S. 1119 – 1129

[68] KEHA, A.B. ; KHOWALA, K. ; FOWLER, J.W.: Mixed integer programming formulations for single machine scheduling problems. In: *Computers & Industrial Engineering* 56 (2009), Nr. 1, S. 357 – 367

[69] KIS, T. ; PESCH, E.: A review of exact solution methods for the non-preemptive multiprocessor flowshop problem. In: *European Journal of Operational Research* 164 (2005), S. 592 – 608

[70] KOCHHAR, S. ; MORRIS, R.J.T.: Heuristic methods for flexible flow line scheduling. In: *Journal of Manufacturing Systems* 6 (1987), S. 299 – 314

[71] KYPARISIS, G.J. ; KOULAMAS, C.: A note on weighted completion time minimization in a flexible flow shop. In: *Operations Research Letters* 29 (2001), S. 5 – 11

[72] KYPARISIS, G.J. ; KOULAMAS, C.: Flexible flow shop scheduling with uniform parallel machines. In: *European Journal of Operational Research* 168 (2006), S. 985 – 997

[73] LASSERRE, J.-B. ; QUEYRANNE, M.: Generic scheduling polyhedra and a new mixed-integer formulation for single-machine scheduling. In: BALAS, E. (Hrsg.) ; CORNUÉJOLS, G. (Hrsg.) ; KANNAN, R. (Hrsg.): *Integer Programming and Combinatorial Optimization, Proceedings of the 2nd International IPCO Conference*, Carnegie Mellon University, 1992, S. 136 – 149

[74] LAWLER, E.L.: Sequencing jobs to minimize total weighted completion time subject to precedence constraints. In: *Annals of Discrete Mathematics* 2 (1978), S. 75 – 90

[75] LEE, C.-Y. ; CHEN, Z.-L.: Machine scheduling with transportation considerations. In: *Journal of Scheduling* 4 (2001), S. 3 – 24

[76] LEE, G.-C.: Estimating order lead times in hybrid flowshops with different scheduling rules. In: *Computers & Industrial Engineering* 56 (2009), S. 1668 – 1674

[77] LEE, Y.H. ; PINEDO, M.: Scheduling jobs on parallel machines with sequence-dependent setup times. In: *European Journal of Operational Research* 100 (1997), S. 464 – 474

[78] LEISTEN, R.: Flowshop sequencing problems with limited buffer storage. In: *International Journal of Production Research* 28 (1990), Nr. 11, S. 2085 – 2100

[79] LENSTRA, J.K. ; RINNOOY KAN, A.H.G. ; BRUCKER, P.: Complexity of machine scheduling problems. In: *Annals of Discrete Mathematics* 1 (1977), S. 343 – 362

[80] LEUNG, J. Y.-T.: Introduction and Notation. In: LEUNG, J. Y.-T. (Hrsg.): *Handbook of Scheduling. Algorithms, Models and Performance Analysis*. Boca Raton : Chapman & Hall/CRC, 2004

[81] LEUNG, J. Y.-T. ; LI, H. ; PINEDO, M.: Approximation algorithms for minimizing total weighted completion time of orders on identical machines in parallel. In: *Naval Research Logistics* 53 (2006), S. 243 – 260

[82] LEUNG, J. Y.-T. ; LI, H. ; PINEDO, M.: Scheduling orders on either dedicated or flexible machines in parallel to minimize total weighted completion time. In: *Annals of Operations Research* 159 (2008), S. 107 – 123

[83] LI, K. ; YANG, S.-L.: Non-identical parallel-machine scheduling research with minimizing total weighted completion times: Models, relaxations and algorithms. In: *Applied Mathematical Modelling* 33 (2009), S. 2145 – 2158

[84] LINN, R. ; ZHANG, W.: Hybrid flow shop scheduling: A survey. In: *Computers & Industrial Engineering* 37 (1999), S. 57 – 61

[85] LIU, C.-Y. ; CHANG, S.-C.: Scheduling flexible flow shops with sequence-dependent setup effects. In: *IEEE Transactions on Robotics and Automation* 16 (2000), Nr. 4, S. 408 – 419

[86] LIU, H. ; QUEYRANNE, M. ; SIMCHI-LEVI, D.: On the asymptotic optimality of algorithms for the flow shop problem with release dates. In: *Naval Research Logistics* 52 (2005), S. 232 – 242

[87] LIU, Y. ; KARIMI, I.A.: Scheduling multistage batch plants with parallel units and no interstage storage. In: *Computers and Chemical Engineering* 32 (2008), S. 671 – 693

[88] LOVÁSZ, L.: Submodular functions and convexity. In: BACHEM, A. (Hrsg.) ; GRÖTSCHEL, M. (Hrsg.) ; KORTE, B. (Hrsg.): *Mathematical Programming: The State of the Art*. Bonn : Springer, 1983, S. 235 – 257

[89] MARGOT, F. ; QUEYRANNE, M. ; WANG, Y.: Decompositions, network flows, and a precedence constrained single-machine scheduling problem. In: *Operations Research* 51 (2003), Nr. 6, S. 981 – 992

[90] MATI, Y. ; REZG, N. ; XIE, X.: An integrated greedy heuristic for a flexible job shop scheduling problem. In: *IEEE International Conference on Systems, Man, and Cybernetics* 4 (2001), S. 2534 – 2539

[91] MEGOW, N. ; SCHULZ, A.S.: On-line scheduling to minimize average completion time revisited. In: *Operations Research Letters* 32 (2004), S. 485 – 490

[92] MORTON, T.E. ; PENTICO, D.W.: *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management*. New York : Wiley-Interscience, 1993

[93] MOURSLI, Y. ; POCHET, Y.: A branch-and-bound algorithm for the hybrid flow-shop. In: *International Journal of Production Economics* 64 (2000), S. 113 – 125

[94] NADERI, B. ; ZANDIEH, M. ; KHALEGI GHOSHE BALAGH, A. ; ROSHANAEI, V.: An improved simulated annealing for hybrid flowshops with sequence-dependent setup und transportation times to minimize total completion time and total tardiness. In: *Expert Systems with Applications* 36 (2009), S. 9625 – 9633

[95] NAWAZ, M. ; ENSCORE, E. ; HAM, I.: A heuristic algorithm for the *m* machine, *n* job flow-shop sequencing problem. In: *OMEGA* 11 (1983), Nr. 1, S. 91 – 95

[96] NEMHAUSER, G. ; WOLSEY, L.: *Integer and combinatorial optimization*. New York : Wiley-Interscience, 1988

[97] NEMHAUSER, G.L. ; SAVELSBERGH, M.W.P: A cutting plane algorithm for the single machine scheduling problem with release dates. In: *Combinatorial Optimization: New Frontiers in the Theory and Practice, NATO ASI Series F: Computer and Systems Sciences 82*. Berlin : Springer, 1992, S. 63 – 84

[98] NESSAH, R. ; CHU, C. ; YALAOUI, F.: An exact method for $Pm|sds, r_i|\sum_{i=1}^{n} C_i$ problem. In: *Computers & Operations Research* 34 (2007), S. 2840 – 2848

[99] OĞUZ, C. ; ERCAN, M.F. ; CHENG, T.C.E. ; FUNG, Y.F.: Heuristic algorithms for multiprocessor task scheduling in a two-stage hybrid flow-shop. In: *European Journal of Operational Research* 149 (2003), S. 390 – 403

[100] PALMER, D.S.: Sequencing jobs through a multi-stage process in the minimum total time - A quick method of obtaining a near optimum. In: *Operational Research Quarterly* 16 (1965), Nr. 2, S. 101 – 107

[101] PAN, Y. ; SHI, L.: On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single machine scheduling problems. In: *Mathematical Programming* 110 (2007), S. 543 – 559

[102] PATERNINA-ARBOLEDA, C.D. ; MONTOYA-TORRES, J.R. ; ACERO-DOMINGUEZ, M.J. ; HERRERA-HERNANDEZ, M.C.: Scheduling jobs on a k-stage flexible flow-shop. In: *Annals of Operations Research* 164 (2007), Nr. 1, S. 29 – 40

[103] PHILLIPS, C. ; STEIN, C. ; WEIN, J.: Scheduling jobs that arrive over time. In: *Lecture Notes in Computer Science* 955 (1995), S. 86 – 97

[104] PHILLIPS, C. ; STEIN, C. ; WEIN, J.: Minimizing average completion time in the presence of release dates. In: *Mathematical Programming* 82 (1998), S. 199 – 223

[105] PINEDO, M.: Stochastic scheduling with release dates and due dates. In: *Operations Research* 31 (1983), Nr. 3, S. 559 – 572

[106] PINEDO, M.: *Scheduling: Theory, Algorithms and Systems*. 3. Auflage. New York : Springer, 2008

[107] PORTMANN, M.-C. ; VIGNIER, A. ; DARDILHAC, D. ; DEZALAY, D.: Branch and bound crossed with GA to solve hybrid flowshops. In: *European Journal of Operational Research* 107 (1998), S. 389 – 400

[108] PRUHS, K. ; SGALL, J. ; TORNG, E.: Online Scheduling. In: LEUNG, J. Y.-T. (Hrsg.): *Handbook of Scheduling. Algorithms, Models and Performance Analysis*. Boca Raton : Chapman & Hall/CRC, 2004

[109] QUADT, D. ; KUHN, H.: Conceptual framework for lot-sizing and scheduling of flexible flow lines. In: *International Journal of Production Research* 43 (2005), Nr. 11, S. 2291 – 2308

[110] QUADT, D. ; KUHN, H.: A taxonomy of flexible flow line scheduling procedures. In: *European Journal of Operational Research* 178 (2007), Nr. 3, S. 686 – 698

[111] QUEYRANNE, M.: Structure of a simple scheduling polyhedron. In: *Mathematical Programming* 58 (1993), Nr. 2, S. 263 – 285

[112] QUEYRANNE, M. ; SCHULZ, A.S.: Polyhedral Approaches to Machine Scheduling / Technische Universität Berlin. 1994. – Technischer Report

[113] QUEYRANNE, M. ; SCHULZ, A.S.: Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. In: *SIAM Journal on Computation* 35 (2006), Nr. 5, S. 1241 – 1253

[114] QUEYRANNE, M. ; SVIRIDENKO, M.: Approximation algorithms for shop scheduling problems with minsum objective. In: *Journal of Scheduling* 5 (2002), S. 287 – 305

[115] RAJENDRAN, C. ; CHAUDHURI, D.: A multi-stage parallel-processor flowshop problem with minimum flowtime. In: *European Journal of Operational Research* 57 (1992), S. 111 – 122

[116] RIBAS, I. ; LEISTEN, R. ; FRAMIÑAN, J.M.: Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. In: *Computers & Operations Research* 37 (2010), S. 1439 – 1454

[117] RIXEN, I.: *Maschinenbelegungsplanung mit Evolutionären Algorithmen*. Wiesbaden : Deutscher Universitäts-Verlag, 1997

[118] RUIZ, R. ; VÁZQUEZ-RODRÍGUEZ, J.A.: The hybrid flow shop scheduling problem. In: *European Journal of Operational Research* 205 (2010), S. 1 – 18

[119] SALVADOR, M.S.: A solution to a special case of flow shop scheduling problems. In: ELMAGHRABY, S.E. (Hrsg.): *Symposium of the Theory of Scheduling and Applications*, Springer, 1973, S. 83 – 91

[120] SANTOS, D.L. ; HUNSUCKER, J.L. ; DEAL, D.E.: Global lower bounds for flow shops with multiple processors. In: *European Journal of Operational Research* 80 (1995), S. 112 – 120

[121] SAVELSBERGH, M.W.P. ; UMA, R.N. ; WEIN, J.: An experimental study of LP-based approximation algorithms for scheduling problems. In: *INFORMS Journal on Computing* 17 (2005), Nr. 1, S. 123 – 136

[122] SAWIK, T.: Scheduling flexible flow lines with no in-process buffer. In: *International Journal of Production Research* 33 (1995), Nr. 5, S. 1357 – 1367

[123] SAWIK, T.: Simultaneous versus sequential loading and scheduling of flexible assembly systems. In: *International Journal of Production Research* 38 (2000), Nr. 14, S. 3267 – 3282

[124] SAWIK, T.: An exact approach for batch scheduling in flexible flow lines with limited intermediate buffers. In: *Mathematical and Computer Modelling* 36 (2002), Nr. 4/5, S. 461 – 471

[125] SCHULZ, A.S.: *Polytopes and Scheduling*, Technische Universität Berlin, Diss., 1995

[126] SCHULZ, A.S.: Scheduling to minimize total weighted completion time: performance guarantees of LP-based heuristics and lower bounds. In: CUNNINGHAM, W.H. (Hrsg.) ; QUEYRANNE, M. (Hrsg.) ; MCCORMICK, S.T. (Hrsg.): *Integer Programming and Combinatorial Optimization*. Heidelberg : Springer, 1996, S. 301 – 315

[127] SCHULZ, A.S. ; SKUTELLA, M.: Random-based scheduling: New approximations and LP lower bounds. In: *RANDOM, Lecture Notes in Computer Science*, Springer, 1997, S. 119 – 133

[128] SCHUSTER, C.J.: No-wait job shop scheduling: tabu search and complexity of subproblems. In: *Mathematical Methods of Operations Research* 63 (2006), S. 473 – 491

[129] SGALL, J.: On-line Scheduling. In: FIAT, A. (Hrsg.) ; WOEGINGER, G.J. (Hrsg.): *Online Algorithms: The State of the Art*. Berlin : Springer, 1998 (Lecture Notes in Computer Science 1442), S. 196 – 231

[130] SIDNEY, J.B.: Decompostion algorithms for single-machine sequencing with precedence relations and deferral costs. In: *Operations Research* 23 (1975), S. 283 – 298

[131] SMUTNICKI, C.: Some results of the worst-case analysis for flow shop scheduling. In: *European Journal of Operational Research* 109 (1998), S. 66 – 87

[132] SOUKHAL, A. ; MARTINEAU, P.: Resolution of a scheduling problem in a flowshop robotic cell. In: *European Journal of Operational Research* 161 (2005), S. 62 – 72

[133] SOUKHAL, A. ; OULAMARA, A. ; MARTINEAU, P.: Complexity of flow shop scheduling problems with transportation constraints. In: *European Journal of Operational Research* 161 (2005), S. 32 – 41

[134] SOUSA, J.P. ; WOLSEY, L.A.: A time-indexed formulation of non-preemptive single machine scheduling problems. In: *Mathematical Programming* 36 (1992), S. 353 – 367

[135] TAKAKU, K. ; YURA, K.: Online scheduling aiming to satisfy due date for flexible flow shops. In: *JSME International Journal Series C - Mechanical Systems Machine Elements and Manufacturing* 48 (2005), Nr. 1, S. 21 – 25

[136] TANG, L. ; LIU, W. ; LIU, J.: A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment. In: *Journal of Intelligent Manufacturing* 16 (2005), S. 361 – 370

[137] TANG, L. ; XUAN, H.: Lagrangian relaxation methods for real-time hybrid flowshop scheduling with finite intermediate buffers. In: *Journal of the Operational Research Society* 57 (2006), S. 316 – 324

[138] TANG, L. ; XUAN, H. ; LIU, J.: A new Lagrangian relaxation algorithm for the hybrid flowshop scheduling problem to minimize total weighted completion time. In: *Computers & Operations Research* 33 (2006), S. 3344 – 3359

[139] TANG, L.X. ; LUH, P.B. ; LIU, J.Y. ; FANG, L.: Steel-making process scheduling using Lagrangian relaxation. In: *International Journal of Production Research* 40 (2002), Nr. 1, S. 55 – 70

[140] THORNTON, H.W. ; HUNSUCKER, J.L.: A new heuristic for minimal makespan in flow shops with multiple processors and no intermediate storage. In: *European Journal of Operational Research* 152 (2004), S. 96 – 114

[141] UMA, R. N. ; WEIN, J. ; WILLIAMSON, David P.: On the relationship between combinatorial and LP-based lower bounds for NP-hard scheduling problems. In: *Theoretical Computer Science* 361 (2006), Nr. 2, S. 241 – 256

[142] URLINGS, T.: *Heuristics and metaheuristics for heavily constrained hybrid flowshop problems*, Universidad Politecnia de Valencia, Diss., 2010

[143] VAN DEN AKKER, J.M. ; HURKENS, C.A.J. ; SAVELSBERGH, M.W.P.: Time-indexed formulations for machine scheduling problems: Column generation. In: *INFORMS Journal on Computing* 12 (2000), Nr. 2, S. 111 – 124

[144] VESTJENS, A.P.A.: *On-line machine scheduling*, Institute for Business Engineering and Technology Application - Eindhoven University of Technology, Diss., 1997

[145] VIGNIER, A. ; BILLAUT, J.-C. ; PROUST, C.: Les problèmes d'ordonnancement de type flow-shop hybride : État de l'art. In: *RAIRO: Recherche opérationelle* 33 (1999), Nr. 2, S. 117 – 183

[146] VOSS, S.: In: KUBEL, K. (Hrsg.) ; BECKER, J. (Hrsg.) ; GRONAU, N. (Hrsg.) ; SINZ, E. (Hrsg.) ; SUHL, L. (Hrsg.): *Enzyklopädie der Wirtschaftsinformatik. On-line Lexikon*. http://www.enzyklopaedie-der-wirtschaftsinformatik.de : Oldenbourg Wissenschaftsverlag, 2009

[147] VOSS, S. ; WITT, A.: Hybrid flow shop scheduling as a multi-mode multi-project scheduling problem with batching requirements: A real-world application. In: *International Journal of Production Economics* 105 (2007), Nr. 2, S. 445 – 458

[148] WANG, H.: Flexible flow shop scheduling: Optimum, heuristics and artificial intelligence solutions. In: *Expert Systems* 22 (2005), Nr. 2, S. 78 – 85

[149] WANG, X. ; TANG, L.: A tabu search heuristic for the hybrid flowshop scheduling with finite intermediate buffers. In: *Computers & Operations Research* 36 (2009), Nr. 3, S. 907 – 918

[150] WARDONO, B. ; FATHI, Y.: A tabu search algorithm for the multi-stage parallel machine problem with limited buffer capacities. In: *European Journal of Operational Research* 155 (2004), Nr. 2, S. 380 – 401

[151] WEBSTER, S.: New bounds for the identical parallel processor weighted flow time problem. In: *Management Science* 38 (1992), Nr. 1, S. 124 – 136

[152] WENG, M.X. ; LU, J. ; REN, H.: Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective. In: *International Journal of Production Economics* 70 (2001), S. 215 – 226

[153] WITTE, M.: *Optimierung der Robotersteuerung in einem Prozesslabor für die Stahlherstellung*, Technische Universität Dortmund, Diplomarbeit, 2008

[154] WITTROCK, R.J.: An adaptable scheduling algorithm for flexible flow lines. In: *Operations Research* 36 (1988), Nr. 3, S. 445 – 453

[155] XUAN, H. ; TANG, L.: Scheduling a hybrid flowshop with batch production at the last stage. In: *Computers & Operations Research* 34 (2007), Nr. 5, S. 2718 – 2733

[156] YANG, J. ; POSNER, M.E.: Scheduling parallel machines for the customer order problem. In: *Journal of Scheduling* 8 (2005), S. 49 – 74

[157] YANG, T. ; KUO, Y. ; CHANG, I.: Tabu-search simulation optimization approach for flow-shop scheduling with multiple processors - A case study. In: *International Journal of Production Research* 42 (2004), Nr. 19, S. 4015 – 4030

[158] YUAN, J. ; SOUKHAL, A. ; CHEN, Y. ; LU, L.: A note on the complexity of flow shop scheduling with transportation constraints. In: *European Journal of Operational Research* 178 (2007), S. 918 – 925