Technische Universität Ilmenau
Fakultät für Wirtschaftswissenschaften
Fachgebiet Wirtschaftsinformatik 1
Prof. Dr. P. Gmilkowsky

Konzeption und prototypische Implementierung einer verteilten Shifting-Bottleneck-Heuristik

Freie wissenschaftliche Arbeit
zur Erlangung des akademischen Grades
„Diplom-Wirtschaftsinformatiker“

an der Fakultät für Wirtschaftswissenschaften
der Technischen Universität Ilmenau

Referent:	Prof. Dr. Peter Gmilkowsky
Betreuer:	Dr. Lars Mönch
Bearbeiter:	René Drießel
Bearbeitungszeit:	6 Monate
Abgabetermin:	19.04.2004

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ilmenau, den 19. April 2004

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Innerhalb des zwanzigsten Jahrhunderts erfuhr die industrielle Fertigung eine enorme Produktivitätssteigerung. Der technologische Fortschritt, der dies möglich machte, führte gleichzeitig zu wesentlich komplexeren Fertigungsabläufen. Außerdem bedingt die komplizierte Fertigungstechnik im Allgemeinen eine Erhöhung der Fixkosten.

Des Weiteren sorgte die erhöhte Produktion für einen Wandel von Verkäufer- zu Käufermärkten. Dies verringerte die Absatzchancen für ein einzelnes Unternehmen. Obgleich die variablen Kosten auf der anderen Seite reduziert wurden, sorgt diese Entwicklung und der hohe Fixkostenblock für einen immer schärferen Wettbewerb.

Durch den Käufermarkt wird es immer schwieriger, exakte Nachfrageprognosen aufzustellen – die Planungsschwierigkeiten für die Fertigung nehmen zu. Gleichzeitig fordert der Markt allerdings eine schnellere Reaktionszeit sowie ein differenzierteres Angebot.

Insbesondere in der Halbleiterindustrie tritt dieses Problem auf. Eine Chipfabrik kostet inzwischen bis zu drei Milliarden Dollar [Review 2003]. Gleichzeitig zeichnet sich die Produktion durch eine hohe Komplexität aus. Bei Bearbeitung eines einzelnen Loses treten für die Steuerung verschiedene Schwierigkeiten, wie beispielsweise Batch-Maschinen oder zyklische Arbeitsgangfolgen, auf.

Verfahren zur Optimierung von fertigungsspezifischen Kenngrößen, wie Durchlaufzeiten oder Verspätungen werden als „Scheduling-Verfahren“ bezeichnet. Die weiter oben angeführten Probleme sorgen dafür, dass die Maschinenbelegungsprobleme (Scheduling-Probleme) sehr schwierig zu lösen sind. In vielen Fällen ist keine exakte Lösung innerhalb eines vertretbaren Zeitrahmens möglich. Vielmehr wird mittels Heuristiken versucht, den Suchraum von vornherein einzuschränken. Einen vielversprechenden Ansatz bieten so genannte Dekompositionsheuristiken, bei denen das Hauptproblem in kleinere, leichter lösbare Unterprobleme aufgeteilt wird. Ein Vertreter dieser Dekompositionsansätze ist die Shifting-Bottleneck-Heuristik, die 1988 von Adams, Balas und Zawack [Adams u. a. 1988] entwickelt wurde.

Innerhalb des Shifting-Bottleneck-Ansatzes wird das Scheduling-Problem für das gesamte Fertigungssystem in dynamische Belegungsprobleme für Maschinengruppen zerlegt. Die globale Sicht auf das Gesamtproblem erfolgt mittels eines disjunkten Graphen, der Abschätzungen für die frühesten Start- und spätesten Fertigstellungstermine der Lose an die Maschinengruppen liefert.

1.2 Zielstellung

Der Graph (und damit die Heuristik) kann verteilt werden, wenn für das gesamte Fertigungssystem andere Verfahren die frühesten Start- und spätesten Endtermine bereitstellen. In dieser Arbeit soll jedem Bereich (Zusammenfassung mehrerer räumlich beieinanderliegender Maschinengruppen) ein Shifting-Bottleneck-Algorithmus zugeordnet werden. Die Bereiche können Informationen über Start- und Fertigstellungstermine austauschen und auf diese Weise bessere Scheduling-Entscheidungen treffen. Ziel dieser Arbeit ist der Entwurf und die Implementation eines Koordinationsmechanismus zwischen den

Bereichen. Des Weiteren soll der Koordinationsmechanismus und die verteilte Shifting-Bottleneck-Heuristik in das Multiagentensystem FABMAS [Mönch u. a.] integriert werden. Dieses System soll eine hierarchische Steuerung einer Halbleiterfabrik ermöglichen [Zimmermann 2003].

Es ergeben sich also die folgenden Aufgaben:

- Entwurf eines Koordinationsmechanismus zum Austausch von Start- und Fertigstellungszeiten von Losen für die Shifting-Bottleneck-Heuristik in unterschiedlichen Bereichen.
- Prototypische Umsetzung des Entwurfs in C++.
- Bewertung des implementierten Verfahrens unter Verwendung des Simulationstools AutoSched AP.
- Prototypische Umsetzung von Teilen des Verfahrens im Multi-Agenten-System FABMAS [Mönch u. a.] unter Verwendung von C#.

1.3 Aufbau der Arbeit

Im zweiten Kapitel wird ein grober Überblick über die bestehenden Verfahren gegeben, um Job-Shop-Scheduling-Probleme zu lösen. Das dritte Kapitel gibt eine Einführung in die verwendete Notation und erläutert die verwendete Shifting-Bottleneck-Heuristik. Der Koordinationsmechanismus für die verteilte Shifting-Bottleneck-Heuristik ist ebenfalls Thema dieses Kapitels.

Das vierte Kapitel geht auf die Implementation des Algorithmus näher ein. Zunächst erfolgt eine prototypische Implementation in der Programmiersprache C++. Danach wird der Algorithmus in das Multiagentensystem FABMAS integriert.

Im fünften Kapitel erfolgt eine Überprüfung der Leistungsfähigkeit des Algorithmus. Mit Hilfe des Simulators AutoSched AP [Brooks Automation 2001b] werden verschiedene Halbleiterfabrikmodelle getestet. Die Ergebnisse dieser Simulationsexperimente werden hier behandelt. Zum Schluss erfolgt noch ein Ausblick auf weitere mögliche Forschungsarbeiten.

2 Lösungsansätze für das deterministische Job-Shop-Scheduling-Problem

2.1 Begriffsklärung

Innerhalb der Literatur sind die verwendeten Begriffe nicht immer eindeutig. Um eine Verwirrung zu vermeiden, erfolgt an dieser Stelle zunächst eine kurze Erläuterung der in dieser Arbeit verwendeten Begriffe:

Arbeitsplan
: Ein Arbeitsplan (Route, Technologie) beschreibt die zur Herstellung eines Produktes nötigen Arbeitsschritte. Er legt die Reihenfolge der einzelnen Arbeitsschritte auf den jeweiligen Maschinengruppen fest.

Arbeitsschritt
: Ein Arbeitsschritt (Step) ist ein Teil eines Arbeitsplans und beschreibt die Bearbeitung eines Loses auf der zugehörigen Maschine.

Los
: Ein Los (Lot) umfasst mehrere Siliziumscheiben (Wafer), welche in der Fabrik bearbeitet werden sollen. In dieser Arbeit werden die Begriffe Job, Los, Auftrag und Fertigungsauftrag synonym verwendet. Im Rahmen einer objektorientierten Betrachtung ist ein Los die konkrete Umsetzung (Instanz) eines Arbeitsplanes.

Arbeitsgang
: Der Arbeitsgang (Task) gehört im Unterschied zum Arbeitsschritt zu einem Los und nicht zu einer Route. Er ist – äquivalent zur Los-Arbeitsplan-Beziehung – eine Instanz eines Arbeitsschritts.

Maschine
: Eine Maschine (Station, Tool) ist die einzuplanende Ressource innerhalb der Fabrik.

Maschinengruppe
: Eine Maschinengruppe (Stationfamily, Toolgroup) besteht aus mehreren gleichartigen Maschinen, welche die selben Arbeitsschritte durchführen können.

Fertigungsbereich
: Ein Fertigungsbereich fasst mehrere Maschinengruppen logisch und physisch zu einem Bereich zusammen.

Bereichsabschnitt
: Ein Bereichsabschnitt (Operation) ist ein Teil eines Arbeitsplanes, der innerhalb eines Bereiches liegt. In der Halbleiterfertigung wird ein Bereichsabschnitt häufig als Operation bezeichnet. Auf einen zweiten Begriff zur Instanzunterscheidung wird an dieser Stelle verzichtet. Stattdessen wird immer von der Operation eines Loses oder von der Operation eines Arbeitsplans gesprochen.

Belegungsplan
Der Belegungsplan (Schedule) beinhaltet die Reihenfolge der Abarbeitung der Arbeitsgänge auf den einzelnen Maschinen. Im Rahmen des Scheduling wird dieser Plan im Voraus bestimmt und über Dispatchregeln auf die Maschinen umgesetzt (vgl. Kapitel 3.7).

Für die Instanzbegriffe (Los, Bereichsabschnitt und Arbeitsgang) ergeben sich mindestens vier wichtige Zeitbegriffe. An gegebener Stelle werden diese weiter verfeinert.

Verfügbarkeitsdatum
Das Verfügbarkeitsdatum (release date) ist der Zeitpunkt, an dem ein Arbeitsgang (bzw. das Los/die Operation) verfügbar ist, das heißt, abgearbeitet werden kann. Für eine Maschine existiert ebenfalls ein Verfügbarkeitsdatum. Dieses bezeichnet den Zeitpunkt, zu dem die Maschine freigeworden und für die nächsten Arbeitsgänge verfügbar ist.

Fälligkeitsdatum
Das Fälligkeitsdatum (due date) ist der Zeitpunkt, zu dem eine Fertigung abgeschlossen sein sollte. Dieser Begriff kann sich auch auf Operationen, Lose oder Arbeitsgänge beziehen.

Startdatum
Das Startdatum eines Arbeitsganges ist der Zeitpunkt, an dem der Arbeitsgang für die Bearbeitung vorgesehen ist. Dieser ist immer größer oder gleich dem Verfügbarkeitsdatum. Auch hier gilt wieder, dass es auch ein Startdatum für eine Operation oder einen Job geben kann.

Fertigstellungsdatum
Der Fertigstellungstermin eines Arbeitsganges ergibt sich aus dem Starttermin zuzüglich der Rüstzeit und der kompletten Bearbeitungszeit dieses Arbeitsganges. Das Fertigstellungsdatum einer Operation oder eines Loses ist das Fertigstellungsdatum des letzten Arbeitsganges.

2.2 Job-Shop-Scheduling-Probleme

Ein Job-Shop-Scheduling-Problem ist durch mehrere Maschinen und mehrere Lose (Jobs) mit unterschiedlichen Arbeitsplänen (Routen) gekennzeichnet. Es existieren noch weitere Schwierigkeiten, wie Batchmaschinen, zyklische Arbeitspläne, reihenfolgeabhängige Rüstzeiten, Maschinenausfälle, etc. Bei einer Batchmaschine werden mehrere Arbeitsgänge verschiedener Lose zu einem Batch zusammengefasst und gleichzeitig bearbeitet. Man spricht von einem zyklischen Arbeitsplan, wenn er eine Ressource im Zeitablauf mehrmals benötigt. Die meisten Job-Shop-Scheduling-Probleme sind zu komplex (NP-Hart), als dass sie in kurzer Zeit optimal gelöst werden können. Aus diesem Grund werden häufig Heuristiken eingesetzt.

In diesem Kapitel werden einige allgemeine Verfahrensansätze zum Lösen von Job-Shop-Scheduling-Problemen vorgestellt. Am Ende des Kapitels erfolgt eine Zusammenfassung der Anforderungen an ein Scheduling-Verfahren, das in realistischen Anwendungsszenarien zum Einsatz kommen kann.

2.3 Exakte Verfahren für Job-Shop-Scheduling-Probleme

In der Literatur gibt es verschiedene exakte Verfahrensansätze für die einzelnen Scheduling-Probleme. Allen diesen Verfahren ist gemeinsam, dass im Voraus ein Plan (Schedule) bestimmt wird, der die optimale Belegungsplanung der einzelnen Maschinen beinhaltet.

Beispiele für exakte Verfahren sind die sogenannte lineare Optimierung und die dynamische Programmierung. An dieser Stelle sollen beide Methoden kurz erläutert werden.

2.3.1 Lineare Optimierung

Die lineare Optimierung beschäftigt sich mit der Lösung von allgemeinen Optimierungsproblemen. Ein Problem wird hierbei als lineares Ungleichungssystem dargestellt:

minimiere:

$$c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$$

unter den Nebenbedingungen:

$$\begin{array}{rcl} a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n & \leq & b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n & \leq & b_2 \\ & \vdots & \\ a_{m,1}x_1 + a_{m,2}x_2 + \ldots + a_{m,n}x_n & \leq & b_m \\ \\ x_j & \geq & 0 \quad \forall j = 1 \ldots n \end{array}$$

Der Vektor $c_1, \ldots, c_n$ ist der Kostenvektor. Die Vektoren $a_{1,j}, \ldots, a_{m,j}$ bezeichnet man als Aktivitätsvektoren. Die Variable x_j steht allgemein für die Intensität der jeweiligen Aktivität. Der Vektor $b_1, \ldots, b_m$ ist der Ressourcenvektor, der die Beschränkungen für die Aktivitäten widerspiegelt.

Das Ziel ist das Finden eines Variablenvektors $x_1, \ldots, x_n$ bei dem die Kosten, unter Berücksichtigung der Nebenbedingungen, minimal werden.

Für die lineare Optimierung existieren effiziente und exakte Algorithmen, allerdings lassen sich die meisten Scheduling-Probleme nicht dadurch abbilden [Pinedo 2002]. Erweiterungen die eine Abbildung der meisten Scheduling-Probleme ermöglichen sind die ganzahlige lineare Optimierung und die dijunktive Optimierungen.

ganzzahlige lineare Optimierung (integer programming)

Die ganzzahlige lineare Optimierung ist eine Erweiterung der linearen Optimierung.

Im Unterschied zur linearen Optimierung existiert bei der ganzzahligen linearen Optimierung die Restriktion, dass die Variablen $x_1, \ldots, x_n$ ganze Zahlen (Integer) sein müssen. Viele Scheduling-Probleme können als ganzzahlige lineare Optimierungsprobleme formuliert werden. Im Unterschied zur Linearen Programmierung existieren für die Integer Programmierung keine effizienten und exakten Algorithmen [Pinedo 2002].

Disjunktive Programmierung (disjunctive programming)

Die Nebenbedingungen in dem oben vorgestellten linearen Ungleichungssystem sind alle konjunkt, dass heißt sie müssen alle gleichzeitig erfüllt sein. In vielen mathematischen Programmen können die Nebenbedingungen (constraints) allerings in konjunkte und disjunkte Bedingungen unterteilt werden.

Wenn alle Bedingungen erfüllt sein müssen, nennt man die Bedingungsmenge konjunkt. Eine Bedingungsmenge ist disjunkt, wenn nur eine Bedingung erfüllt sein muss.

Auf beide Formulierungen lassen sich dieselben Techniken zur Lösung anwenden. Ein in der Literatur häufig verwendeter Ansatz ist die so genannte „Branch and Bound“ Technik. Das Branching bezieht sich hierbei auf eine Aufteilung des Lösungsraums. Das Bounding auf die Begrenzung des Zielkriteritums. Ist eine Lösung eines Teils des Problemraums schlechter als der Grenzwert, kann der entsprechende Problemraum verworfen werden.

Im folgenden Beispiel sind die Bedingungen für ein Job-Shop-Scheduling-Problem mit m Maschinen und n Jobs dargestellt [Pinedo 2002]. Das Ziel ist die Reduzierung der maximalen Zykluszeit. Das Problem lässt sich sehr schön mittels eines gerichteten Graphen G darstellen (Abbildung 2/1). Dieser Graph ist definiert als $G = (N, E_c, E_d)$. N bezieht sich hierbei auf die Menge der Knoten. Jeder Knoten repräsentiert eine Operation (i, j) des Jobs j, die auf der Maschine i abgearbeitet wird. Zusätzlich existieren noch ein künstlicher Startknoten und ein künstlicher Endknoten, um die Berechnung der Zykluszeit zu erleichtern.

Die Kantenmenge E_c enthält alle konjunkten Kanten, die die Losroute repräsentieren. Alle Kanten der Form $(i, j) \rightarrow (k, j)$ sind konjunkt. Dies bedeutet, dass der Job j auf der Maschine i bearbeitet werden muss, bevor er auf Maschine k bearbeitet werden kann. Die Kantenmenge E_d enthält alle disjunkten Kanten. Alle Kanten der Form $(i, l) - (i, j)$ sind disjunkte Kanten. Der Job l kann entweder vor oder nach dem Job j auf der Maschine i abgearbeitet werden. Die disjunkten Kanten bilden m Cliquen, wobei m die Anzahl der Maschinen darstellt. Eine Clique ist ein Begriff aus der Graphentheorie, der einen Graphen beschreibt, bei dem alle Knoten untereinander verbunden sind.

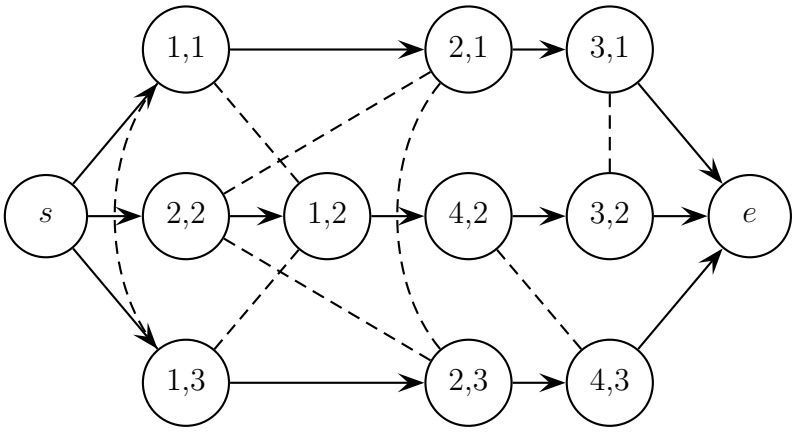

Abb. 2/1: Graphenrepräsentation für 3 Lose auf 4 Maschinen

Für die Minimierung der Zykluszeit (C_{max}) kann man folgendes disjunkte Programm formulieren ($y_{i,j}$ stellt den Startzeitpunkt der Operation (i,j) dar):

minimiere C_{max}

unter den Nebenbedingungen:

$$
\begin{array}{lcll}
y_{k,j} - y_{i,j} & \leq & p_{i,j} & \forall (i,j) \rightarrow (k,j) \in E_c \\
C_{max} - y_{i,j} & \leq & p_{i,j} & \forall (i,j) \in N \\
y_{i,j} - y_{i,l} & \leq & p_{i,l} \text{ oder } y_{i,l} - y_{i,j} \leq p_{i,j} & \forall (i,l) \text{ und } (i,j), i = 1, \ldots, m \\
y_{i,j} & \geq & 0 & \forall (i,j) \in N
\end{array}
$$

Die erste Bedingung stellt sicher, dass die Operation (k,j) nicht vor dem Fertigstellen der Operation (i,j) begonnen wird. Die dritte Gleichung repräsentiert die disjunktive Beziehung innerhalb des Problems. Sie stellt sicher, dass die Operationen auf der Maschine i in einer bestimmten Reihenfolge bearbeitet werden müssen. Aufgrund dieser Bedingung wird dieses Programm als disjunktives Programm bezeichnet.

2.3.2 Dynamische Programmierung

Ein anderer häufig verwendeter Ansatz ist die sogenannte dynamische Programmierung (dynamic programming) [Pinedo 2002]. Die dynamische Programmierung verfolgt im Grundansatz eine vollständige Enumeration. Die Menge der Berechnungen wird über einen „Teile und Herrsche“ Ansatz minimiert. Die dynamische Programmierung gliedert sich prinzipiell in drei Teile.

- eine Initialbedingung
- eine rekursive Relation
- eine Zielfunktion

Innerhalb der rekursiven Relation löst der Algorithmus eine Reihe von Unterproblemen, bis die Lösung für das Originalproblem gefunden ist. Der Algorithmus ermittelt die optimale Lösung für jedes Unterproblem und den Beitrag des Unterproblems zur global optimalen Lösung. Bei jeder Iteration wird ein Unterproblem gelöst, das größer ist als alle vorher gelösten Unterprobleme. Die Lösung für das aktuelle Unterproblem wird mit Hilfe der Lösungen der vorher gelösten Probleme gefunden.

Aufgrund ihrer Komplexität sind exakte Verfahren nur für kleine Probleme und zur mathematischen Beschreibung von Job-Shop-Scheduling-Problemen geeignet. In der Praxis werden deswegen heuristische Verfahren angewendet, die zwar nicht das optimale Ergebnis garantieren können, jedoch optimierenden Charakter haben.

2.4 Heuristische Verfahren für Job-Shop-Scheduling-Probleme

2.4.1 Dispatchregeln zur Steuerung

Dispatch- bzw. Prioritätsregeln sind die einfachste Art der Steuerung. Innerhalb der Massenfertigung liefern Dispatchregeln häufig gute Ergebnisse bei gleichzeitig kurzer Ausfüh-

rungszeit. Innerhalb der Serien- und Einzelfertigung ergeben die Anwendung von Prioritätsregeln allerdings häufig verbesserungswürdige Resultate.

Die Maschinenbelegung mittels Prioritätsregeln erfolgt folgendermaßen: Für die einzelnen Arbeitsgänge, die sich in der Warteschlange einer Maschine bzw. einer Maschinengruppe befinden, wird eine Prioritätszahl berechnet. Anhand dieser Prioritätszahlen wird entschieden, welcher der Arbeitsgänge als nächstes auf der Maschine abgearbeitet wird.

Prioritätsregeln werden in der Anwendung komplizierter, sobald Batchentscheidungen mitgetroffen werden müssen. Dabei muss vor der Auswahl des zu bearbeitenden Batches eine Zusammenfassung der einzelnen Arbeitsgänge zu einem Batch erfolgen. Hierbei müssen die jeweiligen Batchkriterien sowie die Beladegrößen der einzelnen Maschinen berücksichtigt werden. Die einzelnen Arbeitsgänge werden hierzu nach einer einfachen Dispatchregel sortiert. Nachdem die Batches gebildet wurden, kann über eine weitere Dispatchregel der zu bearbeitende Batch ausgewählt werden. Nachfolgend sind einige Prioritätsregeln näher aufgeführt:

EDD (Earliest Due Date)

Die Arbeitsgänge werden nach dem frühesten Fälligkeitstermins (des Auftrages) sortiert.

$$EDD_{i,j} = d_j$$

mit: d_j = geplanter Fertigstellungstermin des Auftrages j.

CR (Critical Ratio)

Die Sortierung erfolgt nach der zur Verfügung stehenden Zeitreserve (Slack). Dieser Slack berechnet sich aus dem Fertigstellungstermin, dem aktuellen Zeitpunkt und der verbleibenden Restprozesszeit.

$$CR_{i,j} = (d_j - t)/(\textstyle\sum_{k=i}^{m_j} p_{kj})$$

mit: t = aktueller (Entscheidungs-)Zeitpunkt.
i = aktueller Arbeitsschritt des Auftrages j.
m_j = Anzahl der Arbeitsschritte des Auftrages j.
$p_{k,j}$ = Bearbeitungszeit des Arbeitsschrittes k des Auftrages j.

S/RPT (Slack Per Remaining Processing Time – Schlupfzeitregel)

Die Sortierung erfolgt nach dem Verhältnis zwischen dem Slack und der Summe der verbleibenden Prozesszeit.

$$SRPT_{i,j} = \frac{d_j - (t + \sum_{k=i}^{m_j} p_{kj})}{\sum_{k=i}^{m_j} p_{kj}}$$

WSPT (Weighted Shortest Processing Time – gewichtete kürzeste Prozesszeit)

Die Sortierung erfolgt nach dem Quotienten der Priorität des Auftrages und der Prozesszeit des Arbeitsschrittes.

$$WSPT_{i,j} = \frac{w_j}{p_{ij}}$$

mit: w_j = Priorität des Auftrages j.
$p_{i,j}$ = Bearbeitungszeit des Arbeitsschrittes i des Auftrages j.

ATC (Apparent Tardiness Cost)

Die ATC-Regel gehört zu den komplexeren zusammengesetzten Prioritätsregeln. Sie unterteilt sich in zwei Teile – einen WSPT Term und einer slackähnliche Regel. Die hier gezeigte ATC-Regel ist nur eine einfache Variante. Es existieren in der Literatur und in der Praxis andere und komplexere Varianten.

$$ATC_{i,j} = \frac{w_j}{p_{ij}} \cdot e^{-\frac{max(d_{i,j}-p_{i,j}-t,0)}{K \cdot \bar{p}}}$$

mit: $\bar{p}$ = Durchschnittliche Prozesszeit aller Arbeitsschritte auf der Maschine i
K = Skalierungsparameter

Innerhalb der Literatur existieren noch eine Fülle weiterer Dispatchregeln. Häufig existieren auch verschiedene Varianten ein und derselben Regel. Es existieren auch Studien über die Vor- und Nachteile einiger Dispatchregeln in bestimmten Fabrikumgebungen. In [Rose 2003] erfolgt zum Beispiel eine Untersuchung der Performance von verschiedenen CR-Regeln.

Problematisch sind die räumlich und zeitlich lokalen Entscheidungen der Prioritätsregeln zu sehen. Häufig existieren mehrere Einstellparameter, mit denen diese Regeln an die jeweilige Fabriksituation angepasst werden können. Dazu sind allerdings aufwendige Simulationsstudien notwendig.

2.4.2 Dekompositionsansätze

Dekompsitionsansätze verfolgen einen Mittelweg zwischen den lokalen Dispatchregeln auf der einen Seite und der Komplettlösung des Problems auf der anderen Seite. Das globale Problem wird in kleinere, leichter lösbare Unterprobleme aufgeteilt.

Es existieren prinzipiell zwei Arten von Dekomposition. Die zeitliche Dekomposition macht sich den Umstand zu nutze, dass die Ressourcen zu unterschiedlichen Zeiten benötigt werden. Es ist jedoch auch möglich, die Dekomposition anhand der verschiedenen Scheduling-Entitäten vorzunehmen.

2.4.2.1 Zeitliche Dekomposition

Zeitliche Dekomposition basiert auf der Tatsache, dass die Ressourcen zu unterschiedlichen Zeiten benötigt werden. So braucht man beispielsweise einen Arbeitsgang, der erst in zwei Tagen verfügbar ist nicht in einer Entscheidung für die nächsten zwei Stunden zu berücksichtigen. Man kann die zeitliche Dekomposition wiederum in die lineare und hierarchische zeitliche Dekomposition unterteilen [Ovacik und Uzsoy 1997].

lineare zeitliche Dekomposition

Bei der linearen zeitlichen Dekomposition erfolgen die Scheduling-Entscheidungen zu bestimmten Zeitpunkten. Der Scheduling-Horizont wird in mehrere kleinere Zeitintervalle aufgeteilt. Zu Beginn eines jeden Zeitintervalls erfolgt eine einzelne Scheduling-Entscheidung. Diese Art der Vorgehensweise wird häufig als „Rolling Horizon Procedure“ (RHP) bezeichnet [Morton 1981].

Die weiter oben erwähnten Dispatchregeln sind ein Beispiel für die zeitliche, lineare Dekomposition. Zu jeder Zeit, wenn eine Maschine frei wird, wird eine Scheduling-Entscheidung gefällt.

hierarchische zeitliche Dekomposition

Bei der hierarchischen, zeitlichen Dekomposition werden Scheduling-Entscheidungen auf verschiedenen zeitlichen Ebenen getroffen. Bei den höheren Ebenen ist die Frequenz der Zeitpunkte, zu denen Scheduling-Entscheidungen getroffen werden, niedriger als bei den unteren Ebenen.

Die höhere und langfristigere Ebene gibt hierbei die Randbedingungen für die niedrigere und operativere Ebene vor. Es ist allerdings schwierig, sicherzustellen, dass die Vorgaben, die die obere Ebene macht, in der unteren auch anwendbar sind.

Ein Beispiel für diesen hierarchischen Ansatz ist das FABMAS System ([Mönch u. a.] und [Zimmermann 2003]), in dessen Rahmen diese Arbeit zu sehen ist.

2.4.2.2 Auf Entitäten basierende Dekomposition

Bei der auf Entitäten basierenden Dekomposition werden die verschiedenen Operationen, welche eingeplant werden sollen, zu Gruppen zusammengefasst, die gemeinsam geplant werden. Die Gruppierung der Operationen kann auf verschiedene Arten erfolgen. Eine Variante ist die Gruppierung nach Losen (Jobs), eine andere nach Maschinengruppen. Es ist leicht ersichtlich, dass die Gruppierung auch auf höheren Ebenen erfolgen kann, wie zum Beispiel Losgruppen oder Fertigungsbereichen.

Losbasierte Dekomposition

Bei der Los basierten Dekomposition werden alle Operationen eines Loses eingeplant. Die Lose werden üblicherweise nacheinander in den Schedule eingebracht. Hier muss der Unterproblemlöser sicherstellen, dass der Schedule gültig bleibt.

Maschinengruppenbasierte Dekomposition

Statt einzelne Jobs nacheinander einzuplanen, erfolgt hier das Scheduling auf Basis der einzelnen Maschinengruppen. Ein Beispiel für diesen Ansatz ist die in dieser Arbeit verwendete Shifting-Bottleneck-Heuristik.

2.5 Anforderungen an Lösungsverfahren für Job-Shop-Scheduling-Probleme in realistischen Anwendungsszenarien

In heutigen Halbleiterfabriken erfolgt keine vollständig automatisierte Steuerung. Weder über zentrale noch dezentrale Ansätze kann eine Fabrik ohne manuellen Eingriff gesteuert werden. Alle Algorithmen, die in akzeptabler Zeit ein Ergebnis liefern, bieten nur Näherungslösungen für das formulierte Problem. Ein weiteres Problem, welches bei Algorithmen auftritt, die Schedules bilden, ist die Rescheduling Problematik. Wenn eine Maschine ausfällt, dann ist der gebildete Schedule obsolet. Das Problem kann entweder durch ein Rescheduling oder eine Dispatchregel kompensiert werden. Ob und wann dieses Rescheduling allerdings genau durchzuführen ist, ist schwer zu bestimmen.

Einerseits liefern einfache Dispatchregeln aufgrund ihrer räumlich und zeitlich lokalen Sichtweise häufig unbefriedigende Ergebnisse. Andererseits sind global optimale Scheduling-Algorithmen aufgrund ihrer hohen Laufzeiten nicht einsetzbar. Die heutigen Rechnergenerationen erlauben die Verwendung von höherwertigen Algorithmen. Eine Verbesserung der Ergebnisse, bei gleichzeitiger Kontrolle der Laufzeit, ist mit höherwertige Heuristiken möglich.

Dekompositionsansätze, wie die in dieser Arbeit verwendete Shifting-Bottleneck-Heuristik, bieten die Möglichkeit, das globale Scheduling-Problem durch Dekomposition in kleinere Aufgaben zu lösen. Die Unterproblemlöser können aufgrund des hierarchischen Ansatzes von Dekompositionsalgorithmen globale Informationen in ihre Entscheidungen mit einbeziehen. Hierdurch lässt sich eine gute Balance zwischen Laufzeiteffizienz und der Scheduling-Qualität erreichen.

3 Konzept für eine verteilte Shifting Bottleneck Heuristik

3.1 Notation

Die Anzahl der Jobs (Fertigungsaufträge, Lose) wird durch n repräsentiert und die Maschinengruppenanzahl durch m. Jeder Job j ist eine Instanz einer Route (Arbeitsplan) r. Das Paar (i, r) stellt einen einzelnen Arbeitsschritt der Route r auf der Maschinengruppe i dar. Äquivalent zur Job-Route-Beziehung ist das Paar (i, j) eine Instanz von (i, r). Das Paar (i, j) repräsentiert also einen Arbeitsgang des Jobs j auf der Maschinengruppe i. Die folgenden Variablen dienen zur näheren Beschreibung eines Jobs j:

Freigabezeitpunk $\mathbf{r_j}$
: Dies ist die Ankunftszeit des Jobs im System. Der Job kann frühestens zu diesem Zeitpunkt bearbeitet werden.

Fälligkeitszeitpunkt $\mathbf{d_j}$
: Dies ist der Zeitpunkt, an dem der Job fertig abgearbeitet sein sollte. Eine Verfrühung oder Verspätung ist erlaubt.

Priorität $\mathbf{w_j}$
: Die Priorität bzw. das Gewicht des Jobs.

Fertigstellungszeitpunkt $\mathbf{C_j}$
: Der Fertigstellungszeitpunkt des Jobs j.

Verspätung $\mathbf{T_j}$
: T_j ist definiert als $max(C_j - d_j, 0) = max(L_j, 0)$ und repräsentiert die Verspätung des Jobs j im Bezug auf seinen Fälligkeitszeitpunkt.

Die folgenden Variablen beschreiben einen Arbeitsgang (i, j) des Jobs j:

Prozesszeit $\mathbf{p_{i,j}}$
: Die Prozesszeit $p_{i,j}$ repräsentiert die durchschnittliche Bearbeitungszeit des Arbeitsgangs (i, j) des Jobs j auf der Maschinengruppe i.

Verfügbarkeitstermin $\mathbf{r_{i,j}}$
: Dies ist die voraussichtliche Ankunftszeit des Arbeitsgangs (i, j). Dieser Zeitpunkt kann über eine Abschätzung der Wartezeit gewonnen werden.

Fälligkeitstermin $\mathbf{d_{i,j}}$
: Dies ist der Zeitpunkt, an dem der Arbeitsgang fertig abgearbeitet sein sollte. Eine Verfrühung oder Verspätung ist erlaubt. Wie der Verfügbarkeitstermin kann dieser Wert über eine Abschätzung der Wartezeit gewonnen werden.

Wenn in der Losroute Zyklen auftreten, das heißt eine Maschinengruppe von einem Job mehrmals besucht wird, dann kann der Jobteil des Paares (i, j) durch Verwendung eines

Buchstaben eindeutig gemacht werden. So kennzeichnet beispielsweise $(1, 2a)$ die erste Bearbeitung des Jobs 2 auf der Maschinengruppe 1 und $(1, 2b)$ die zweite Bearbeitung [Oey und Mason 2001].

Die Maschinengruppe i besteht typischerweise aus mehreren einzelnen Anlagen a, welche die gleichen Arbeitsgänge durchführen können. Innerhalb der Literatur werden solche Maschinengruppen auch als parallele Maschinen bezeichnet [Pinedo 2002]. Die folgenden Variablen beschreiben eine Maschine, beziehungsweise eine Maschinengruppe näher:

Beladezeit $\mathbf{l_a}$
Die Beladezeit einer Maschine a.

Durchschnittliche Beladezeit $\mathbf{\bar{l}_i}$
Die durchschnittliche Beladezeit der Maschinengruppe i, gebildet als arithmetisches Mittel der Beladezeiten der zugehörigen Maschinen.

Entladezeit $\mathbf{u_a}$
Die Entladezeit einer Maschine a.

Durchschnittliche Entladezeit $\mathbf{\bar{u}_i}$
Die durchschnittliche Entladezeit der Maschinengruppe i, gebildet als arithmetisches Mittel der Entladezeiten der zugehörigen Maschinen.

Rüstzeit $\mathbf{s^t_{j,k}}$
Die Umrüstzeit, welche beim Umrüsten von Job j auf Job k auf der Maschine a benötigt wird.

Verfügbarkeit $\mathbf{r_a}$
Die früheste Verfügbarkeit der Maschine. Im Unterschied zu den Verfügbarkeitsdaten der Arbeitsgänge muss diese nicht geschätzt werden. Sie kann aus den Betriebsdaten ermittelt werden.

Der Einfachheit halber werden die einzelnen Maschinen einer Gruppe als identisch angesehen. Das heißt, dass die Belade-, Entlade-, Rüst- und Prozesszeiten für alle Maschinen innerhalb einer Gruppe gleich lang sind. Mit diesem Wissen kann man die Beschreibung der Arbeitsgänge noch weiter verfeinern:

Starttermin $\mathbf{b^t_{i,j}}$
Dies ist die Startzeit des Arbeitsgangs (i, j) auf der Maschine a. Dieser Zeitpunkt ist erst bekannt, wenn das Scheduling-Problem gelöst ist.

eingeplante Maschine $\mathbf{tool_{i,j}}$
Dies ist die Maschine des Arbeitsgangs (i, j). Diese Information ist ebenfalls erst bekannt, wenn das Scheduling-Problem gelöst ist.

3.2 Klassifikation von Scheduling-Problemen

Um die Diskussion über Scheduling-Probleme zu erleichtern, ist eine Klassifikation nötig. Innerhalb der Literatur hat sich die Systematik von Graham [Graham u. a. 1979] durchgesetzt. Hierbei wird das Einplanungsproblem (Reihenfolge-, Scheduling-Problem) durch ein Tripel $\alpha|\beta|\gamma$ klassifiziert (vgl. [Graham u. a. 1979] und [Pinedo 2002]). Die Bedeutung der einzelnen Felder ist im folgenden aufgeführt:

α-Feld

Dieses Feld beschreibt die Maschinenumgebung durch einen einzelnen Eintrag. Mögliche Einträge sind:

Flow-Shop (Fm) Für ein Flow-Shop Problem sind n Lose und m Maschinen gegeben. Alle Lose werden nach demselben Arbeitsplan hergestellt.

Job-Shop (Jm) Bei einem Job-Shop-Problem liegen ebenfalls n Lose und m Maschinen vor. Im Unterschied zu einem Flow-Shop-Problem haben die einzelnen Jobs unterschiedliche Arbeitspläne.

β-Feld

Dieses Feld enthält die Prozesscharakteristik. Es kann aus einem oder mehreren Einträgen bestehen. Diese sind unter anderem:

$\mathbf{r_j}$ Die einzelnen Jobankünfte sind dynamisch.

$\mathbf{s_{jk}}$ Innerhalb der Algorithmen müssen reihenfolgeabhängige Rüstzeiten für die einzelnen Maschinen mit berücksichtigt werden.

batch Innerhalb des Prozesses werden Batchmaschinen verwendet.

batch, incompatible Die Batches können nur innerhalb inkompatibler Batchfamilien bearbeitet werden.

recr Die Arbeitsgangfolge (Losroute) kann Zyklen enthalten. Das heißt, das eine Maschine mehrmals von einem Los passiert werden kann.

γ-Feld

In diesem Feld steht, welche Prozessmerkmale optimiert, das heißt minimiert oder maximiert werden sollen. Häufig enthält das γ-Feld nur einen einzigen Eintrag. Im Folgenden sind einige mögliche Zielfunktionen aufgezählt:

$\mathbf{C_{max}}$ Die Minimierung des spätesten Fertigstellungszeitpunkts stellt eine möglichst (zeit-)effiziente Ressourcennutzung in den Vordergrund.

$\sum_{\mathbf{j=1}}^{\mathbf{n}} \mathbf{w_j T_j}$ Für die terminorientierte Fertigung ist die absolute gewichtete Verspätung (Total Weighted Tardiness) allerdings eine bessere Zielfunktion. Die Termintreue ist hier (im Vergleich zu C_{max}) in der Formel direkt enthalten.

3.3 Shifting-Bottleneck-Heuristik für komplexe Produktionssysteme

3.3.1 Voraussetzungen

Innerhalb der Halbleiterfertigung ist die termingerechte Fertigstellung wegen der verschärften Wettbewerbssituation von entscheidender Bedeutung. Dieses Zielkriterium bildet das Leistungsmaß der gewichteten Verspätung (*Total-Weighted-Tardiness* - **TWT**) gut ab. Die Zielfunktion ist wie folgt definiert:

$$\sum_{j=1}^{n} w_j T_j \tag{1}$$

Das Scheduling-Problem innerhalb einer Halbleiterfabrik kann man wie folgt klassifizieren [Mönch und Rose 2003]:

$$Jm|r_j, s_{jk}, batch, incompatible, rectr| \sum w_j T_j \tag{2}$$

Jm steht hierbei für einen Job Shop, durch $batch, incompatible$ ist die Batchbearbeitung mit inkompatiblen Familien bezeichnet. Alle Arbeitsgänge einer Familie können zur Bildung eines Batches herangezogen werden. s_{jk} bezeichnet reihenfolgeabhängige Umrüstzeiten und r_j dynamische Jobankünfte. Die Zyklen innerhalb der Losrouten sind durch den Term $rectr$ gekennzeichnet.

Das Problem 2 hat sich durch Rückführung auf das einfachere Problem $1||\sum w_j T_j$ als NP-vollständig erwiesen [JK u. a. 1977]. Aus diesem Grund werden Heuristiken zur Lösung von 2 eingesetzt.

3.3.2 Problemrepräsentation

Für die Abbildung des Job-Shop-Problems ist – wie in Kapitel 2.3 bereits erwähnt – eine Graphendarstellung vorteilhaft. Dieser Graph ist definiert als:

$$G = (V, E_c, E_d) \tag{3}$$

Die einzelnen Komponenten des Graphen sind im Folgenden erläutert:

Knoten V

Innerhalb des in dieser Arbeit verwendeten Graphen existieren vier Arten von Knoten:

Arbeitsgangknoten

Diese Knoten repräsentieren einen Arbeitsgang (i, j) des Jobs j auf einer Maschine i.

Jobstartknoten

Der Graph wird innerhalb der Literatur häufig durch einen globalen künstlichen Startknoten a vervollständigt. In dieser Arbeit wird allerdings jedem Job ein Startknoten $(s_1, \ldots, s_n)$ zugeordnet, da dies die Behandlung innerhalb des Koordinationsmechanismus erleichtert.

Jobendknoten

Des Weiteren erhält jeder Job einen künstlichen Endknoten $(e_1, \ldots, e_n)$, der die Aussteuerung eines Loses aus der Fertigung symbolisiert. Ist die Zielfunktion die Minimierung von C_{max}, so kann ein weiterer künstlicher Endknoten eingefügt werden, der die Berechnung von C_{max} erleichtert.

Maschinenstartknoten

Um die Abbildung der Verfügbarkeitszeiten der Maschinen zu erleichtern, existiert für jede Maschine ein künstlicher Startknoten, welcher den Verfügbarkeitstermin für die Maschine enthält.

Alle Kanten, die von den genannten künstlichen Knoten ausgehen, haben das Gewicht 0. Sie dienen einzig und allein der leichteren Berechnung des Graphen.

Konjunkte Kanten E_c

Eine konjunkte Kante ist eine gerichtete Kante $e = (u, v)$ zwischen zwei Knoten $u \in V$ und $v \in V$. Hierdurch wird eine Reihenfolge in der Jobabarbeitung beschrieben. Die Arbeitspläne (Losrouten) werden durch konjunkte Kanten modelliert. Die späteren Scheduling-Entscheidungen werden ebenfalls auf diese Weise im Graphen eingetragen.

Das Gewicht einer konjunkten Kante (u, v) repräsentiert die Dauer der Bearbeitung eines Arbeitsgangs (i, j) wobei u den Arbeitsgang (i, j) repräsentiert. Hierbei werden die Rohprozesszeit $p_{i,j}$, die Be- und Entladezeiten der Maschine (l_a und u_a) sowie die Rüstzeiten $s_{k,j}$ berücksichtigt. Die reihenfolgeabhängigen Rüstzeiten können allerdings erst nach der Belegung einer Maschine innerhalb des Graphen abgebildet werden. Das Kantengewicht der disjunkten Kanten, die diese Belegung repräsentieren, wird hierbei um die Rüstzeiten erhöht.

Das Kantengewicht $k(u, v)$ ist demnach wie folgt definiert:

$$k(u,v) = \begin{cases} \bar{l_i} + p_{i,j} + \bar{u_i} & \text{wenn } u \text{ nicht eingeplant} \\ l_a + p_{i,j} + u_a + s_{k,j} & \text{wenn } u \text{ eingeplant.} \end{cases} \tag{4}$$

Da alle ausgehenden Kanten eines Knotens u das selbe Gewicht haben, kann man $k(u) = k(u, v) \forall (u, v) \in dest_u$ setzen. $dest_u$ bezeichnet hierbei die Menge aller ausgehenden Kanten des Knotens u.

Disjunkte Kanten E_d

Disjunkte Kanten sind ungerichtete Kanten $e = (u, v)$ zwischen zwei Knoten $u \in V$ und $v \in V$, die zur gleichen Maschinengruppe gehören. Zwischen zwei Knoten, die denselben Job beschreiben, werden keine disjunkten Kanten abgetragen, da die Reihenfolgeentscheidung durch den Arbeitsplan vorweggenommen wurde.

Disjunkte Kanten beschreiben noch nicht vorgenommene Scheduling-Entscheidungen. Eine disjunkte Kante kann den Graphen – nach der Scheduling-Entscheidung – auf drei verschiedene Weisen ändern:

1. Es existiert keine Kante im finalen Graphen.

2. Es existiert eine konjunkte Kante (u, v).
3. Es existiert eine konjunkte Kante (v, u).

Disjunkte Kanten werden manchmal auch als zwei konjunkte Kanten abgetragen, von denen eine oder beide im Verlauf des Algorithmus entfernt werden. (vgl. [Pinedo 2002, 160 ff.]).

Ein Beispiel für einen solchen disjunkten Graph zeigt die Abbildung 3/1. In diesem Beispiel werden drei Jobs mit unterschiedlichen Arbeitsplänen (Losrouten) auf vier verschiedenen Maschinengruppen bearbeitet. Die Arbeitspläne der drei Lose zeigt Tabelle 1. Die Daten der Maschinen sind in Tabelle 2 gegeben. Die Maschinengruppe 1 besteht aus einer Batch Maschine, mit der maximal drei Arbeitsgänge verschiedener Lose gemeinsam verarbeitet werden können. Die anderen Maschinen sind normale Maschinen. Die Maschinengruppe 4 besteht aus zwei identischen Maschinen.

Die Abbildung zeigt nur den Initialgraphen. Wie ein Schedule implementiert werden kann, zeigt der nächste Abschnitt.

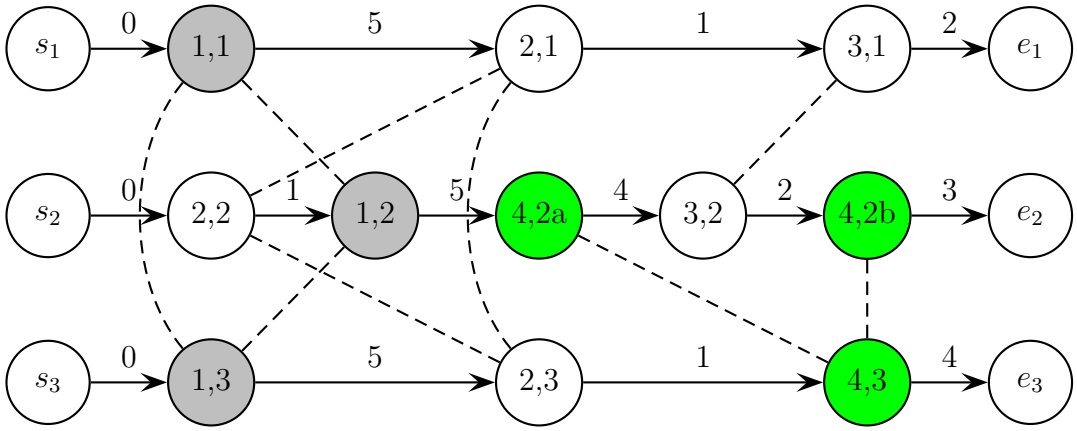

Abb. 3/1: Zyklischer Graph für 3 Lose auf 4 Maschinengruppen

Job	Arbeitsplan	Prozesszeiten
1	$1, 2, 3$	$p_{1,1} = 3; p_{2,1} = 1; p_{3,1} = 2$
2	$2, 1, 4, 3, 4$	$p_{2,2} = 1; p_{1,2} = 3; p_{4,2a} = 2; p_{3,2} = 2; p_{4,2b} = 1$
3	$1, 2, 4$	$p_{1,3} = 3; p_{2,3} = 1; p_{4,3} = 2$

Tabelle 1: Arbeitspläne (Maschinenfolgen)

3.3.3 Implementierung eines Schedules

Abhängig von der Maschinenumgebung existieren mehrere Möglichkeiten, einen einzelnen Schedule für eine Maschinengruppe innerhalb des Graphen zu implementieren:

Maschine	Beladezeit	Entladezeit	Batchgröße
1	$l_1 = 1$	$ul_1 = 1$	3
2	$l_2 = 0$	$ul_2 = 0$	1
3	$l_3 = 0$	$ul_3 = 0$	1
4_1	$l_4^1 = 1$	$ul_4^1 = 1$	1
4_2	$l_4^2 = 1$	$ul_4^2 = 1$	1

Tabelle 2: Arbeitspläne (Maschinenfolgen)

parallele Nicht-Batchmaschinen

Die Schedules für Nicht Batchmaschinen lassen sich ohne Umstände in der Graphen implementieren. Zwischen den Arbeitsgängen, die auf einer einzelnen Maschine bearbeitet werden, werden Kanten in der Reihenfolge der Bearbeitung implementiert.

parallele Batchmaschinen

Batchmaschinen erfordern eine besondere Behandlung innerhalb des Graphen. Bei der Berücksichtigung von Batchmaschinen existieren zwei Möglichkeiten der Implementierung eines Schedules:

künstliche Batchknoten

: Bei dieser Lösung werden künstliche Batchknoten in den Graphen eingefügt ([Oey und Mason 2001] und [Pabst 2003]). Alle Knoten, die zu einem Batch gehören, werden mit diesem Batchknoten über Kanten der Länge 0 verbunden. Von dem Batchknoten gehen wiederum Kanten zu den nachfolgenden Knoten der jeweiligen Losrouten, sowie zum nächsten Batch. Das Gewicht (Länge) dieser Kanten beinhaltet nun die korrekten Rüstzeiten und die korrekten Be- und Entladezeiten der eingeplanten Maschine. Ein Beispiel für diese Variante der Implementierung liefert Abbildung 3/2.

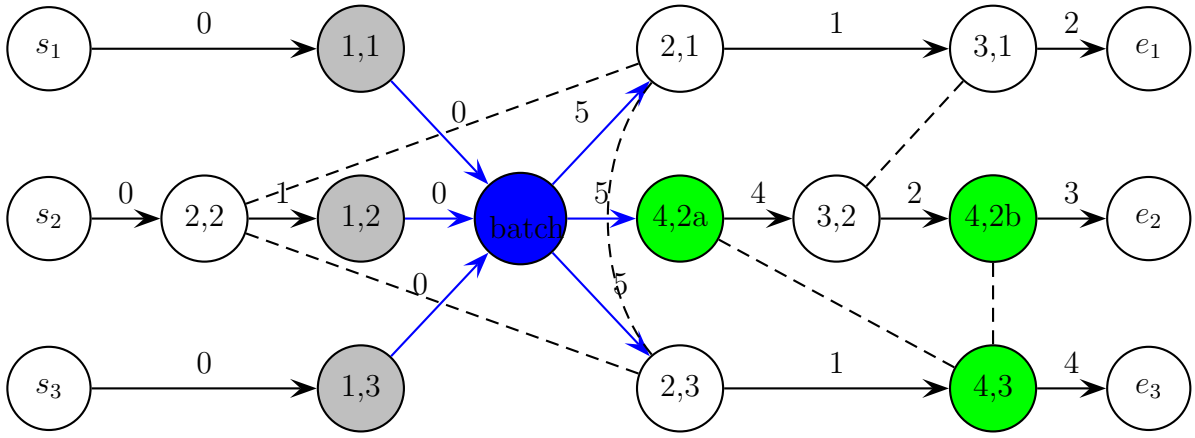

Abb. 3/2: Batchknoten für 3 Lose auf Maschinengruppe 1

Batchkanten

: Die Verwendung von Batchkanten entwickelt die Implementierung von einfa-

chen parallelen Maschinen weiter. Für jeden Knoten des Batches werden zusätzlich Kanten zu den Nachfolgeknoten der anderen Batchknoten eingefügt. Auch in diesem Fall besitzen die ausgehenden Kanten die korrekten Gewichte (Belade-, Entlade- und Rüstzeiten). Abbildung 3/3 verdeutlicht diese Vorgehensweise.

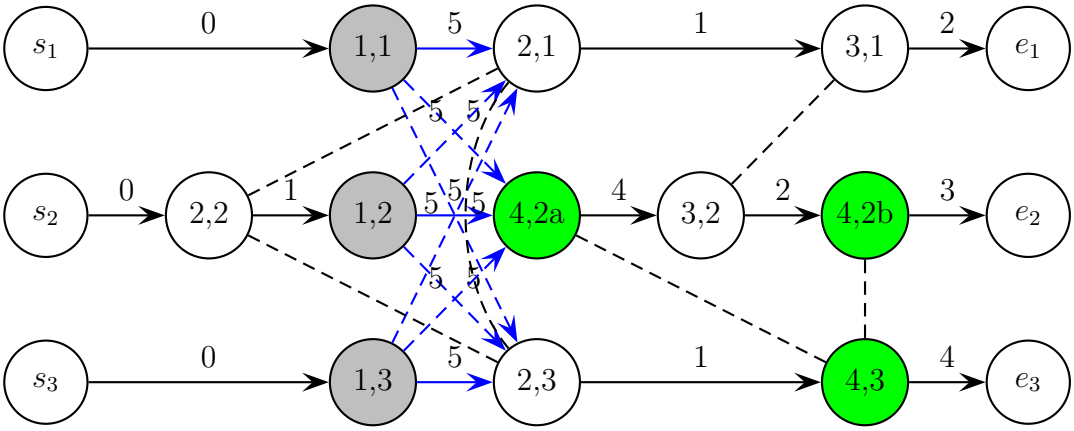

Abb. 3/3: Batchkanten für 3 Lose auf Maschinengruppe 1

Innerhalb dieser Arbeit wurde die zweite Variante gewählt. Der Vorteil besteht in einer einfacheren Architektur des Datenmodells und des Algorithmus (einfache Maschinen werden als einfacher Spezialfall von Batchmaschinen behandelt). Der Nachteil besteht in einer etwas langsameren Berechnung der frühesten Verfügbarkeitstermine und der geplanten Fälligkeitstermine im Vergleich zu der ersten Variante.

Um die Maschinenverfügbarkeitszeiten korrekt mit zu berücksichtigen, müssen noch Kanten von den Maschinenstartknoten zum Anfang des Schedules in den Graphen eingefügt werden.

3.3.4 Algorithmus

Wie bereits erwähnt, ist die Shifting-Bottleneck-Heuristik eine Dekompositionsheuristik. Der ursprüngliche Algorithmus wurde 1998 von Adams et. al [Adams u. a. 1988] für Probleme des Typs $Jm||C_{max}$ entworfen. Seit der ersten Veröffentlichung wurde diese Heuristik kontinuierlich weiterentwickelt und an andere Problemstellungen angepasst. So wurden Untersuchungen zur Maschinenkritikalität [Lee und Pinedo 1997] durchgeführt. Die Berücksichtigung von Batchmaschinen wurde eingebaut [Oey und Mason 2001]. In [Pabst 2003] wiederum erfolgte eine Behandlung von zyklischen Abhängigkeiten. Es hat sich gezeigt, dass die ursprünglich für statische Probleme entwickelte Heuristik auch für komplexere Probleme gute Ergebnisse liefern kann [Mönch und Rose 2003].

Das Gesamtproblem wird hierbei als eine Folge von Einzelmaschinenproblemen $Pm|r_j|w_jT_j$ gesehen, die sukzessive gelöst werden. Um die Reihenfolge bestimmen zu können, in der die Unterprobleme letztendlich in den Graphen implementiert werden, ist die Berechnung der Kritikalität (engl. criticality) des jeweiligen Unterproblems nötig. Im Folgenden bezeichnet $K(i)$ die Kritikalität der Maschinengruppe i. $K_{max}(k)$ sagt aus,

dass die Maschinengruppe k die höchste Kritikalität K_{max} hat. M ist die Menge aller Maschinengruppen und M_0 ist die Menge aller bereits eingeplanten Maschinen. Der in dieser Arbeit verwendete Algorithmus untergliedert sich in die folgenden fünf Schritte:

Schritt 1 (Initialbedingungen)
Setze $M_0 = \emptyset$; der Graph G enthält alle konjunkten Kanten (Auftragsrestriktionen) und keine disjunkten Kanten

Schritt 2 (Analyse der Maschinen, welche eingeplant werden sollen)
$\forall i \in \{M - M_0\}$:
erzeuge Instanz $Pm|r_j|w_jT_j$ mit

Schritt 3 (Engpass Auswahl und Einplanung)
Setze $K_{max}(k) = max(K(i)) \; \forall i \in \{M - M_0\}$;
Einplanung der Maschine k in der Reihenfolge, wie Schritt 2 sie ergeben hat;
Setze $M_0 = M_0 \cup \{k\}$

Schritt 4 (Erneutes Optimieren und Einplanen aller schon eingeplanten Maschinen)

Schritt 5 (Abbruchkriterium)
Wenn $M_0 = M$ dann Stopp, sonst gehe zu Schritt 2

Wenn alle Maschinen eingeplant, das heißt ihre Schedules in den Graphen implementiert sind, kann man aus dem Graphen die Startzeiten der Arbeitsgänge auf den einzelnen Maschinen durch eine Vorwärtskalkulation des Graphen ermitteln. Die folgenden Pseudocodes verdeutlichen die Herangehensweise des Algorithmus in detaillierterer Form. Die Algorithmen verwenden einige Funktionen, die an dieser Stelle nur informell beschrieben werden sollen:

Schedule Maschinengruppe
Die Methode löst das Unterproblem für die Maschinengruppe i. Hierbei wird angenommen, dass dem Unterproblemlöser alle zugehörigen Knoten bekannt sind. Jeder Knoten wird einer Maschine der Maschinengruppe i in einer bestimmten Reihenfolge zugeordnet. Die Unterproblemlöser werden im nachfolgenden Kapitel näher behandelt.

Implementiere Schedule
Mittels dieser Prozedur wird der Schedule der Maschinengruppe i in den Graphen G implementiert.

entferne Schedule
Diese Methode ist das Gegenstück zu der vorherigen Methode. Der Schedule wird aus dem Graphen entfernt.

Speichere Schedule
Die Methode dient zum zwischenzeitlichen Speichern des aktuellen Schedules einer Maschinengruppe i.

Ersetze Schedule

Das Rücksetzen eines Schedules erfolgt mit dieser Methode.

Shifting-Bottleneck-Heuristik

$M_0 \leftarrow \emptyset$
repeat
    Kritische Maschinengruppe $\leftarrow 0$
    K $\leftarrow 0$
    for all $i \in M \setminus M_0$ **do**
        SCHEDULE MASCHINENGRUPPE(i)
        IMPLEMENTIERE SCHEDULE(i)
        K[i] $\leftarrow \sum_{j=1}^{n} w_j \cdot T_j$
        ENTFERNE SCHEDULE(i)
        if $K[i] > K$ **then**
            K $\leftarrow$ K[i]
            Kritische Maschinengruppe $\leftarrow$ i
        end if
    end for
    IMPLEMENTIERE SCHEDULE(Kritische Maschinengruppe)
    REOPTIMIERE MASCHINENGRUPPEN(Kritische Maschinengruppe)
    $M_0 \leftarrow M_0 \cup \{KritischeMaschinengruppe\}$
until $M_0 = M$

Der Algorithmus zur Reoptimierung ist im Folgenden aufgeführt. Die Variable *max_reopt* enthält die maximalen Reoptimierungsschritte, die durchgeführt werden sollen.

Reoptimierung der Maschinengruppen

reopt_schritt $\leftarrow 0$
graph_geaendert $\leftarrow WAHR$
min_zielwert $\leftarrow \sum_{j=1}^{n} w_j \cdot T_j$
while graph_geaendert && reopt_schritt < max_reopt **do**
    graph_geaendert $\leftarrow FALSCH$
    for all $i \in M_0$ **do**
        Schedule $\leftarrow$ SPEICHER SCHEDULE(i)
        ENTFERNE SCHEDULE(i)
        SCHEDULE MASCHINENGRUPPE(i)
        IMPLEMENTIERE SCHEDULE(i)
        K[i] $\leftarrow \sum_{j=1}^{n} w_j \cdot T_j$
        if $K[i] < min\_zielwert$ **then**
            min_zielwert $\leftarrow$ K[i]
            graph_geaendert $\leftarrow WAHR$
        else
            ENTFERNE SCHEDULE(i)
            ERSETZE SCHEDULE(i, Schedule)
            IMPLEMENTIERE SCHEDULE(i)
        end if

end for
reopt_schritt $\leftarrow$ reopt_schritt + 1
until $M_0 = M$

3.3.5 Unterproblemlöser

Es hat sich gezeigt, dass für die Effizienz des Algorithmus die richtige Behandlung der Unterprobleme wichtig ist. Innerhalb der Literatur wird leider sehr wenig auf die Unterproblembehandlung eingegangen. An dieser Stelle soll deshalb näher darauf eingegangen werden.

3.3.5.1 Knotenbasierte Kennzahlen

Die Datenversorgung der Unterproblemlöser ist von entscheidender Bedeutung für die Qualität der Ergebnisse. An dieser Stelle sollen die Kennzahlen für einen Knoten aufgeführt werden. Hiermit lassen sich die folgenden Werte für einen Knoten u ermitteln:

frühester Verfügbarkeitstermin $\mathbf{r(u)}$
Der früheste Verfügbarkeitstermin (engl. earliest ready date) kann über den Graphen ermittelt werden. Dieser Termin ist rekursiv definiert:

$$r(u) = \begin{cases} r_j & \text{wenn } u = s_j - u \text{ ist Startknoten von Job } j \\ r_a & \text{wenn } u = s_a - u \text{ ist Startknoten von Maschine } a \\ \max\limits_{(v,u) \in source_u}(r(v) + k(v,u)) & \text{sonst} \end{cases} \tag{5}$$

Zur Erinnerung: r_j repräsentiert das Verfügbarkeitsdatum des Jobs j, r_a stellt das Verfügbarkeitsdatum der Maschine a dar und $k(v,u)$ ist das Gewicht der Kante zwischen den Knoten $v \in V$ und $u \in V$. Die Menge $source_u$ beinhaltet alle eingehenden Kanten des Knotens u.

Der früheste Verfügbarkeitstermin ist also der längste Pfad, ausgehend von den Startknoten zum jeweiligen Knoten u. Wenn alle Maschinen eingeplant sind, dann entspricht der früheste Verfügbarkeitstermin $r(u)$ des Knotens dem Starttermin des zugehörigen Arbeitsgangs (i,j), wenn der Knoten kein künstlicher Knoten war.

spätester Fälligkeitstermin $\mathbf{d(u)}$
Äquivalent zum frühesten Verfügbarkeitstermin kann auch der späteste Fälligkeitstermin (engl. latest due date) über den Graphen ermittelt werden. Auch dieser Term ist rekursiv definiert:

$$d(u) = \begin{cases} d_j & \text{wenn } u = e_j - u \text{ ist Endknoten von Job } j \\ \min\limits_{(u,v) \in dest_u}(d(v) - k(v)) & \text{sonst} \end{cases} \tag{6}$$

Hierbei repräsentiert d_j das Verfügbarkeitsdatum des Jobs j und $k(v)$ ist das Gewicht der ausgehenden Kanten des Knotens v. Die Menge $dest_u$ beinhaltet alle ausgehenden Kanten des Knotens u.

Auch der späteste Fälligkeitstermin ist also über den längsten Pfad des Knotens u zu den Endknoten des Graphen definiert.

Wartedauer $\mathbf{w_u}$
Diese Variable repräsentiert die Verzögerung zwischen dem Verfügbarkeitstermin $r(u)$ und dem realisierten Starttermin innerhalb des Schedules. Wenn der Knoten noch nicht eingeplant ist, dann ist $w_u = 0$.

3.3.5.2 Vorgängerbehandlung

Innerhalb des Graphen können während der sukzessiven Lösung der Unterprobleme Zyklen (Abbildung 3/4) auftreten, wenn die entstehenden Vorgänger nicht entsprechend behandelt werden. Wenn der Schedule einer Maschinengruppe in den Graphen implementiert wird, schränkt dies die Rahmenbedingungen (engl. constraints) für die anderen Maschinengruppen ein.

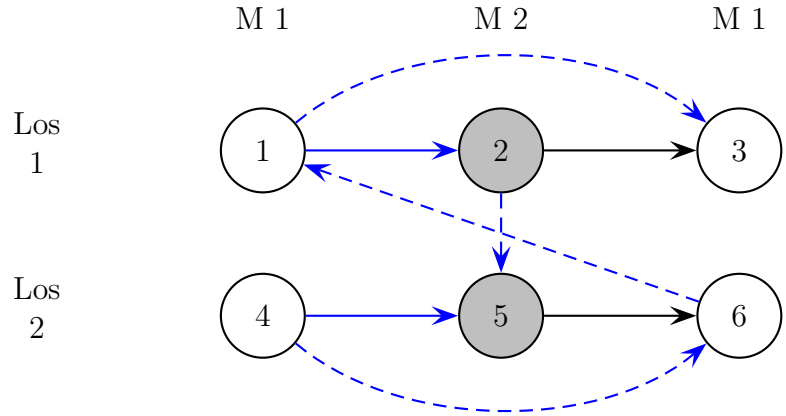

Abb. 3/4: Zyklus innerhalb eines Graphen (M2 wurde vor M1 eingeplant)

Durch die zusätzliche Berücksichtigung von Vorgängerbeziehungen innerhalb der Unterproblemlöser kann man das Auftreten von Zyklen verhindern. Prinzipiell lassen sich Vorgängerbeziehungen wie folgt klassifizieren (vgl. [Pabst 2003]):

konjunkte Vorgängerbeziehungen
Konjunkte Vorgängerbeziehungen ergeben sich aus dem Arbeitsplan des jeweiligen Loses. Ein Beispiel ist ebenfalls in Abbildung 3/4 zu finden. Der Knoten 1 ist Vorgänger des Knotens 3.

disjunkte Vorgängerbeziehungen
Disjunkte Vorgängerbeziehungen entstehen bei der Implementierung von Schedules in den Graphen. In Abbildung 3/4 ist der Knoten 1 ein disjunkter Vorgänger zum Knoten 6. Das Ignorieren dieser Vorgangsbeziehung führte zum Zyklus in Abbildung 3/4.

Neben der Zyklenvermeidung existiert noch ein weiterer wichtiger Aspekt der Vorgängerbehandlung. Die frühesten Verfügbarkeitstermine werden auf Basis des längsten Pfades zwischen zwei Knoten bestimmt. Wenn es beim Einplanen eines Knotens eine Verzögerung (Wartezeit) gibt, so muss diese bei allen nachfolgenden Knoten berücksichtigt werden. Wird diese Verzögerung nicht berücksichtigt, so kann der Scheduler einen Knoten einplanen, obwohl er noch nicht verfügbar ist.

Auf Basis dieser Wartezeitbetrachtung hat Papst [Pabst 2003] noch eine weitere Klassifikationsmöglichkeit eingeführt. Diese orientiert sich an der „Bedeutung“ der Wartezeit für die Shifting-Bottleneck-Heuristik. Abbildung 3/5 zeigt alle Arten dieser Vorgängerbeziehungen:

direkte Vorgängerbeziehungen

Jeder Pfad zwischen zwei Knoten, der nur Knoten passiert, die zu unterschiedlichen Unterproblemen gehören, ist eine direkte Vorgängerbeziehung. Für die Verfügbarkeitsverzögerung sind diese Beziehungen am wichtigsten. Die Knotenpaare $1 \rightarrow 8$, $6 \rightarrow 8$, $8 \rightarrow 5$ und $8 \rightarrow 10$ fallen in diese Klasse.

semidirekte Vorgängerbeziehungen

Eine semidirekte Vorgängerbeziehung entsteht aus der Berücksichtigung von zwei direkten Vorgängerbeziehungen. Hierbei gehören die zwei Knoten zu ein und demselben Los. Aufgrund dieser engen Beziehung ist hier die korrekte Berücksichtigung der Verfügbarkeitsverzögerung ebenfalls sehr wichtig.Das Knotenpaar $1 \rightarrow 5$ fällt in diese Kategorie.

indirekte Vorgängerbeziehungen

Eine indirekte Vorgängerbeziehung besteht aus zwei oder mehreren semidirekten oder direkten Vorgängerbeziehungen. Da die Beziehung hier nicht so eng ist, ist der Einfluss einer Verzögerung nicht so groß. Trotzdem sollten sie mit berücksichtigt werden.

Die Knotenpaare $1 \rightarrow 10$, $6 \rightarrow 5$ und $6 \rightarrow 10$ gehören zu dieser Klasse.

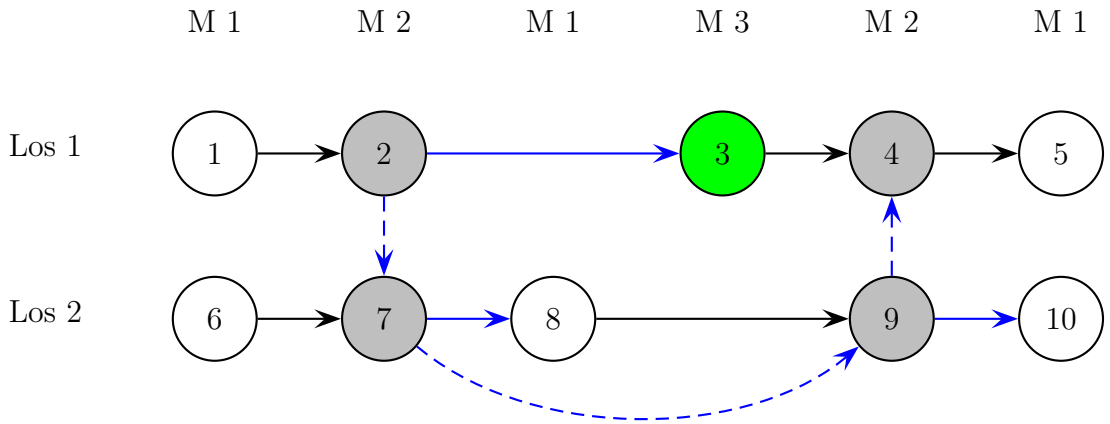

Abb. 3/5: Vorgängerbeziehungen auf Maschine 1 (aus [Pabst 2003])

In Abbildung 3/6 ist die Auswirkung einer Verfügbarkeitsverzögerung dargestellt. Verwendet wird wieder das Eingangs erwähnte Beispiel aus 4 Maschinengruppen und 3 Jobs.

Der eingeplante Schedule von Maschine 2 sorgt für eine Vorgängerabhängigkeit zwischen den Knoten $(1,3)$ und $(1,2)$ sowie zwischen $(1,1)$ und $1,2$ für die Batchmaschine 1. In dieser Situation lassen sich mehrere Konsequenzen aufgrund der Vorgängerinformationen ableiten:

- Die drei Knoten $(1,1)$, $(1,2)$ und $(1,3)$ können nicht zu einem Batch zusammengefasst werden (Maschine 1 war eine Batchmaschine). Nur die Arbeitsgänge $(1,1)$ und $(1,3)$ können miteinander gebatched werden. Diese Besonderheit von Batchmaschinen ist auch der Grund, weswegen bei Nichtbeachtung der Vorgängerabhängigkeiten Zyklen durch Batchmaschinen häufiger auftreten als bei einfachen Maschinen.
- Wenn der Schedule für Maschine 2 implementiert ist, können durch Vorwärtskalkulation die frühesten Verfügbarkeitsdaten (Formel 5) ermittelt werden. Für die Knoten $(1,1)$, $(1,2)$ und $(1,3)$ ergeben sich dann $r_{1,1} = 0$, $r_{1,2} = 7$ und $r_{1,3} = 0$. Nehmen wir nun an, dass die Maschine 1 frühestens zum Zeitpunkt 1 verfügbar ist. Es ergibt sich also eine Verzögerung um eine Zeiteinheit für den ersten Batch. Wenn diese Verzögerung nicht bei den nachfolgenden Knoten berücksichtigt wird, wird der Schedule, den das Unterproblem liefert, auf Basis von falschen Vorgabedaten gebildet (Abbildung 3/7). Durch die Berücksichtigung (Abbildung 3/8) kann der Unterproblemlöser bessere Entscheidungen treffen.

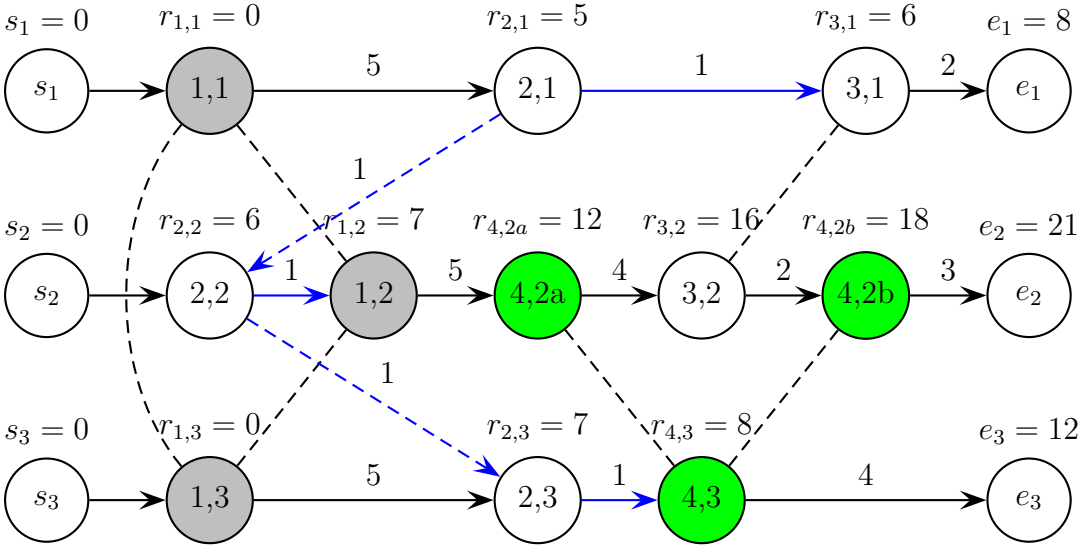

Abb. 3/6: Vorgängerbeziehungen aufgrund der Einplanung von Maschine 2

Um die vorhanden Vorgängerbeziehungen in den Unterproblemen berücksichtigen zu können, benötigt man pro Knoten eine Liste mit den entsprechenden Vorgängerknoten. Die Menge der Vorgänger eines Knotens kann man rekursiv definieren:

$$pred(u) = pred(v) \cup \{v\}; \forall (v,u) \in source_u \tag{7}$$

Die Menge der Vorgänger $pred(u)$ eines Knotens u ist die Vereinigungsmenge der Knoten v der eingehenden Kanten (v,u) und Knoten v. Wenn die Vorgängerbeziehungen gegeben

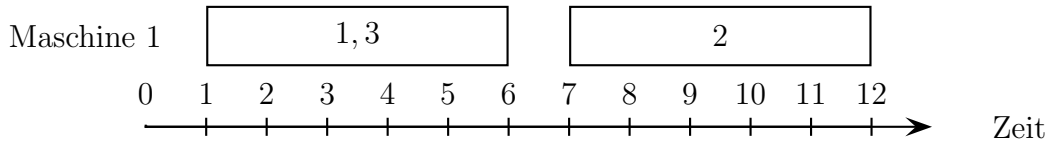

Abb. 3/7: Lokaler Schedule, Werte aus 3/6

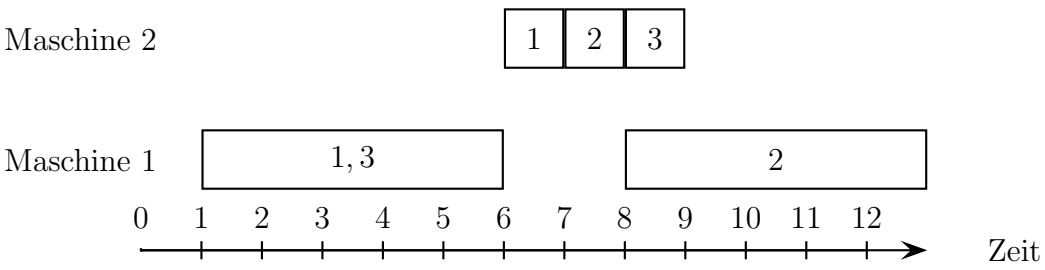

Abb. 3/8: Globaler Schedule bei Beachtung von Vorgängerbeziehungen

sind, kann die Ankunftszeit (engl. avail time) eines Knotens besser bestimmt werden (Formel 8):

$$a(u) = r(u) + max(w_v); \forall v \in pred(u) \tag{8}$$

Wie Formel 5, so impliziert auch Formel 7 die Verwendung einer Breitensuche zur Traversierung des Graphen. Allerdings lässt sich auch eine Tiefensuche zur Vorgängersuche einsetzen wie [Pabst 2003] zeigt. Innerhalb dieser Diplomarbeit wurde die Vorgängersuche jedoch als Breitensuche implementiert, da sie so sehr leicht in die normale Vorwärtskalkulation des Graphen eingebettet werden kann. Um die Performance des Algorithmus zu verbessern, kann man die Vorgängersuche auf das aktuelle Unterproblem beschränken. Zusammenfassend kann ein Algorithmus zur Vorwärtsberechnung und Vorgängersuche des Graphen wie folgt definiert werden:

Algorithmus zur Vorwärtskalkulation und Vorgängersuche
{*Initialisierung*}
Besuchte Kanten $\leftarrow \emptyset$
Knotenstapel $\leftarrow \{s_j; j = 1 \ldots n\}$
for all $u \in V$ **do**
if $u) = s_j$ **then**
$r(u) \leftarrow r_j$ **else**
if $u) = s_a$ **then**
$r(u) \leftarrow r_j$ **else**$r(u) \leftarrow 0$
end if**end if**
end for
{*Traversierung*}
while Knotenstapel $\neq \emptyset$ **do**

```
    u ← POP(Knotenstapel)
    for all (v, u) ∈ source_u do
        r(u) ← max(r(v) + k(v, u))
    end for
    for all (u, v) ∈ destination_u do
        Besuchte Kanten ← Besuchte Kanten ∪ {(u, v)}
        pred(v) ← pred(u) ∪ {u}
        if source_v \ Besuchte Kanten = ∅ then
            PUSH(Knotenstapel, v)
        end if
    end for
end while
```

3.3.5.3 Kritikalitätsbestimmung

Für die Bestimmung der Kritikalität einer Unterproblemlösung existieren verschiedene Datenquellen:

- Die Kritikalität der Maschinengruppen wird durch externe Daten vorgegeben. Diese externen Daten können beispielsweise durchschnittliche Kapazitätsauslastungen oder die Häufigkeit der Maschinenausfälle sein.
- Die Kritikalität wird nur lokal durch Berücksichtigung der Arbeitsgänge des jeweiligen Unterproblems ermittelt.
- Die Kritikalität wird global über den Graphen bestimmt. Der Schedule wird in den Graphen implementiert und die Kritikalität anhand der Veränderung der Fertigstellungszeiten der Jobs gemessen.

Es ist ersichtlich, dass die letzte Art die besten Ergebnisse liefert. Allerdings leidet die Laufzeit des Algorithmus darunter, da der Graph bei jeder Kritikalitätsberechnung traversiert werden muss.

Für die Kritikalitätsberechnung einer Maschinengruppe wird die letzte Variante gewählt. Um die Kritikalität einer Lösung innerhalb eines Unterproblems zu bestimmen, ist allerdings die zweite Variante ausreichend.

Innerhalb dieser Arbeit ist das Kritikalitätsmaß einer Lösung die gewichtete Verspätung der berücksichtigten Arbeitsgänge. Für einen Unterproblemlöser (Algorithmus) a der Maschinengruppe i wird das Maß demnach wie folgt berechnet (Formel 9):

$$K(a) = \sum_{j=1}^{n} w_j \cdot T_{i,j} \tag{9}$$

Hierbei wird $T_{i,j}$ als die absolute Verspätung des Arbeitsgangs (i, j) definiert (Formel 10):

$$T_{i,j} = s_j^i + k_{i,j} - d_{i,j} \tag{10}$$

In Formel 10 ist s_j^i der Startzeitpunkt des Arbeitsgangs, wie ihn der Schedule liefert. $k_{i,j}$ ist die Dauer des Arbeitsgangs, wie in Formel 4 definiert und $d_{i,j}$ ist der geschätzte Fälligkeitstermin des Arbeitsgangs (i, j).

Für die Kritikalität einer Maschinengruppe an sich existieren verschiedene Formen von Kritikalitätsmaßen. Die üblichsten sind an dieser Stelle kurz aufgeführt (vgl. [AYTUG u. a. 2002]):

Maschinenauslastung
Diese Maß berechnet sich aus der Summe der Prozesszeiten aller Arbeitsgänge, die auf dieser Maschinegruppe i bearbeitet werden (Formel 11).

$$K(i) = \sum_{j=1}^{n} p_{i,j} \tag{11}$$

Durchschnittlich verbleibende Arbeitsgänge zur Bearbeitung
Dieses Maß berücksichtigt die verbleibende Arbeitsgänge für jeden einzelnen Job nachdem er auf der Maschine bearbeitet wurde. Grundidee ist hierbei, dass Maschinen, welche früher benötigt werden auch einen höheren Einfluss auf das Zielkriteritum haben (Formel 12).

$$K(i) = \frac{\sum_{x=i+1}^{m_j} 1}{n} \tag{12}$$

Hierbei bezeichnet m_j den letzten Arbeitsgang des Loses j, i ist der aktuelle Index in der Losroute und n ist die Anzahl der Lose, die auf der Maschine verarbeitet wurden.

Mittlere verbleibende Prozesszeit
Hier wird die mittlere verbleibende Prozesszeit der Jobs, die auf dieser Maschine bearbeitet wurden als Maß verwendet. Es stellt eine gewichtete Variante des zweiten Maßes dar (Formel 13).

$$K(i) = \frac{\sum_{x=i+1}^{m_j} p_{xj}}{n} \tag{13}$$

Die einzige Änderung zur Formel 12 besteht darin, dass hier die Prozesszeit p_{xj} des Arbeitsganges (x, j) mit berücksichtigt wird.

Gewichtete Verspätung über alle Lose Auch für die Kritikalitätsbestimmung der Maschinengruppe ist es möglich die gewichtete Verspätung zu verwenden. Hierbei ist jedoch zu beachten, dass sich Formel 14 auf alle Lose und nicht nur auf die jeweiligen Arbeitsgänge bezieht. Aus diesem Grund ist wie weiter oben erwähnt eine Implementierung des Schedules in den Graphen Vorraussetzung.

$$K(i) = \sum_{j=1}^{n} w_j \cdot T_j \tag{14}$$

In dieser Arbeit wurde als Kritikalitätsmaß für die Maschinengruppe die letzte Variante gewählt.

3.3.5.4 Arten von Unterproblemlösern

Welche Algorithmen als Unterproblemlöser verwendet werden, ist entscheidend für die Qualität des Gesamtschedules. Des weiteren hat der Unterproblemlöser eine große Auswirkung auf die Laufzeit des Gesamtalgorithmus. Innerhalb dieser Arbeit wurden im wesentlichen zwei Arten von Unterproblemlösern implementiert. Beiden Arten ist gemeinsam, dass die Reihenfolge des Schedules durch die dynamische Priorität der Knoten bestimmt wird.

- Die erste Variante eines Unterproblemlösers basiert auf einer deterministischen Vorwärtssimulation. Hierbei werden sukzessive die freiwerdenden Maschinen mit den einzuplanenden Knoten belegt. Falls mehrere Knoten zur gleichen Zeit verfügbar sind, werden die Knoten mit der besten dynamischen Priorität verwendet.
- Der andere Ansatz ist unter dem Namen „listenbasiertes Scheduling“ bekannt. Hierbei werden die Knoten zunächst anhand ihrer dynamischen Priorität sortiert und danach auf die jeweiligen Maschinen eingeplant.

Innerhalb eines Unterproblems lassen sich mehrere Algorithmen einsetzen, um dieses Problem zu lösen. Der beste Schedule wird am Ende genommen. Ein generischer Unterproblemlöser – für eine Maschinengruppe i – kann also wie folgt formuliert werden:

Generischer Unterproblemlöser
SUCHE VORGAENGER(i)
Schedule $\leftarrow \emptyset$
K $\leftarrow \infty$
for all $s \in SSP[i]$ **do**
    INITIALISIERE PROBLEM(a, i)
    Schedule[a] = LÖSE PROBLEM(a, i)
    K[a] $\leftarrow \sum_{j=1}^{n} w_j \cdot T_{i,j}$ **if** K[a] < K **then**
        K $\leftarrow$ K[a]
        Schedule $\leftarrow$ Schedule[a]
    end if
end for
IMPLEMENTIERE SCHEDULE(Schedule, i)
K[i] = $\sum_{j=1}^{n} w_j \cdot T_j$
ENTFERNE SCHEDULE(Schedule, i)

Selbstverständlich sind weitere Unterproblemlöser, wie beispielsweise genetische oder auch Branch & Bound Ansätze, denkbar.

3.4 Verteilungsansatz für die Shifting-Bottleneck-Heuristik

Wenn die einzelnen Maschinengruppen verschiedenen Bereichen zugeordnet sind, dann lässt sich die Shifting-Bottleneck-Heuristik verteilen (engl. distributed Shifting-Bottleneck-Heuristik – DSBH). Der grundlegende Ansatz dahinter ist, dass pro Bereich ein disjunkter Graph erzeugt wird, auf dem dann die Shifting-Bottleneck-Heuristik angewandt wird. Dieser Ansatz hat prinzipiell zwei Vorteile:

- Durch die Verteilung ist es möglich, das Gesamtproblem auf mehrere Rechner zu verteilen.
- Durch die Verkleinerung der Problemgröße ist die Berechnung der Schedules nicht mehr so rechenintensiv wie eine globale Berechnung.

Gleichzeitig hat dieser Ansatz allerdings auch prinzipbedingte Nachteile.

- Die Aufteilung der Fabrik in mehrere Bereiche bedingt wiederum die Zergliederung der einzelnen Losrouten in Operationen. Hieraus resultieren Probleme mit der Abschätzung für die Eintrittszeitpunkte der Lose in die jeweiligen Bereiche. Um das Problem zu verdeutlichen, stellt Abbildung 3/12 einen vereinfachten Arbeitsplan innerhalb einer Halbleiterfabrik dar.
- Da die Graphen nur ein kleineres Problem abbilden, können zunächst die globalen Auswirkungen der lokalen Schedules nicht überprüft werden.

Um diese Probleme zu lösen, existieren zwei Möglichkeiten. Zunächst können über einen Grobscheduler Eckplandaten für die einzelnen Eintrittszeitpunkte ermittelt werden. Nachdem die Schedules für die einzelnen Bereiche ermittelt wurden, können die errechneten Zeiten allerdings von den tatsächlichen Zeiten abweichen. Hier ist die Verwendung eines Koordinationsmechanismus angebracht. Im Folgenden sollen beide Möglichkeiten näher behandelt werden.

3.5 Zwei-Ebenen-Verfahren zur Produktionssteuerung

Wie bereits weiter oben erwähnt, wird die Losroute in verschiedene Bereichsabschnitte „zerteilt“. Die Abschätzung der Eintrittszeitpunkte in die einzelnen Bereiche erfolgt mit Hilfe eines globalen Grobschedulers (obere Ebene, Fabrikscheduler). Auf der unteren Ebene (Bereichsscheduler, Feinscheduler) erfolgt mit Hilfe der verteilten Shifting-Bottleneck-Heuristik das Feinscheduling. Die obere Ebene muss der unteren Ebene für den betrachteten Scheduling-Horizont Verfügbarkeits- und Fälligkeitsdaten für die einzelnen Operationen eines Jobs j vorgeben ($r_{i,j}$ und $d_{i,j}$). Abbildung 3/9 verdeutlicht die Idee.

Beide Ebenen beziehen ihre Informationen aus einer Datenschicht (Blackboard), welches die aktuellen Informationen über das zu schedulende System enthält. Dieses Blackboard wird synchron mit den Daten des zu schedulenden Systems gehalten. Die Schedules, welche sowohl der Grobscheduler als auch der Feinscheduler liefern, werden ebenfalls

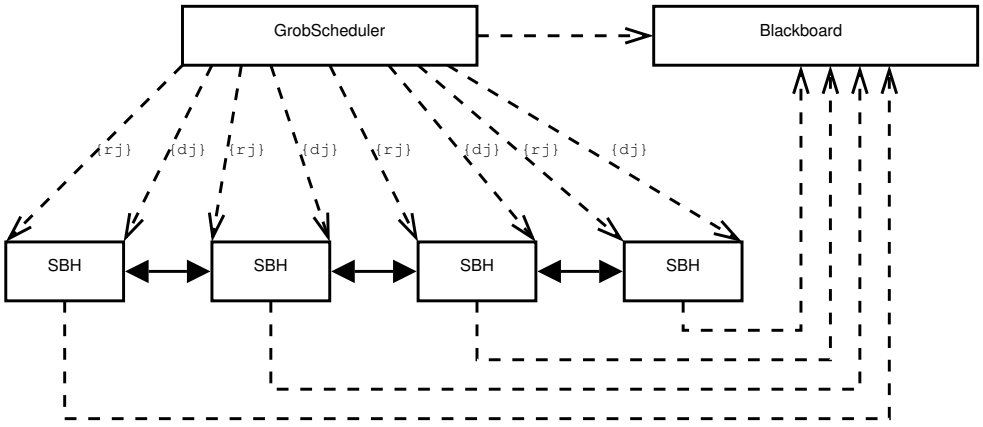

Abb. 3/9: Architektur der Verteilten Shifting-Bottleneck-Heuristik

innerhalb des Blackboards gespeichert und über dieses auf das zu schedulende System angewendet.

Folgende Daten werden für den Bereichsscheduler aus dem Blackboard benötigt:

Maschinendaten
: Zu den benötigten Maschinendaten gehören die minimalen und maximalen Batchgrößen, die Be- und Entladezeiten sowie die Rüstmatrix für die reihenfolgeabhängigen Rüstzeiten.

Arbeitsplan
: Der Arbeitsplan muss die Reihenfolge der Arbeitsschritte sowie die benötigten Prozesszeiten enthalten.

Losdaten
: Zu den Losdaten gehören Losgröße, der aktuelle Arbeitsgang, die Start- und Fälligkeitsdaten sowie das Verfügbarkeitsdatum des aktuellen Arbeitsgangs.

Zeitabschätzung
: Der Grobscheduler liefert auf Basis der oben genannten Daten Abschätzungen für die Verfügbarkeits- und Fälligkeitsdaten für die einzelnen Operationen der verschiedenen Lose.

Auch der verwendete Grobsscheduler hat einen großen Einfluss auf die Scheduling-Qualität. Es existieren verschiedene Ansätze unterschiedlicher Komplexität. An dieser Stelle sollen zwei einfache Ansätze genannt werden.

Hotfactor
: Hier erfolgt die Abschätzung an Hand eines dynamisch ermittelten Flussfaktors. Die verbleibende Zykluszeit wird hierbei durch die restliche Prozesszeit geteilt:

$$HF_j = max(\frac{d_j - max(t, r_j)}{\sum_{k=i}^{m_j} p_{kj}}, 1.0) \tag{15}$$

Hierbei ist d_j das Fälligkeitsdatum des Jobs j, a der aktuelle Zeitpunkt, r_j der Einsteuertermin des Loses j. i stellt an dieser Stelle den aktuellen Arbeitsgang des Auftrages j dar. m_j enthält die Anzahl der Arbeitsgänge des Loses j. $p_{k,j}$ letztendlich repräsentiert die Bearbeitungszeit des Arbeitsgangs k des Auftrages j

Die erwartete Durchlaufzeit $ect_{i,j} = HF_j \cdot p_{i,j}$ kann nun zur Abschätzung der Verfügbarkeits- und Fälligkeitstermine der einzelnen Arbeitsgänge und damit der Operationen verwendet werden.

Gleitender Flussfaktor

Einen etwas anderen Ansatz verfolgt die Methode des gleitenden Flussfaktors. Hierbei wird für, jeden Arbeitsschritt eines Arbeitsplanes ein mittlerer Flussfaktor aus jüngsten Vergangenheitswerten ermittelt. Damit lässt sich nun ebenfalls eine Abschätzung durchführen.

Problematisch an beiden Ansätzen ist die fehlende Kapazitätsbetrachtung im Planungsansatz. Es können Vorgaben gemacht werden, die nicht zu halten sind. Allerdings haben beide Ansätze den Vorteil, dass sie kaum Rechenzeit in Anspruch nehmen können. Des Weiteren sollte der Koordinationsmechanismus die Problematik der falschen Vorgabezeiten etwas entschärfen.

3.6 Konzeption eines Koordinierungsmechanismus zur Erhöhung der Leistungsfähigkeit der verteilten Shifting-Bottleneck-Heuristik

Der zu entwickelnde Koordiniationsalgorithmus soll die Probleme von nicht passenden Vorgabedaten lösen. Da die Koordinierung sehr teuer werden kann, werden an dieser Stelle mehrere Verfahren vorgeschlagen. Damit kann man die Laufzeit des Algorithmus unter Kontrolle halten.

3.6.1 Modifikation des disjunkten Graphen

Da ein Los einen Bereich mehrmals durchlaufen kann (Abbildung 3/10), muss die Behandlung der Vorgängerbeziehungen etwas erweitert werden. Zwischen den einzelnen Bereichsdurchläufen eines Loses müssen die Vorgängerbeziehungen mitgeschrieben werden. Dies geschieht durch das Einfügen von virtuellen Kanten innerhalb des Graphen.

Jeder Knoten u enthält eine Menge seiner Vorgängerknoten $virtual\_edges_u$ innerhalb der Losroute (Abbildung 3/10)). Diese virtuellen Kanten müssen allerdings noch in dem Vorgängerbestimmungsalgorithmus berücksichtigt werden.

Erweiterung der Vorgängersuche innerhalb der DSBH

{*Initialisierung*}
Besuchte Kanten $\leftarrow \emptyset$
Knotenstapel $\leftarrow \{s_j; j = 1 \ldots n\}$
for all $u \in V$ **do**
if $u) = s_j$ **then**
$r(u) \leftarrow r_j$ **else**

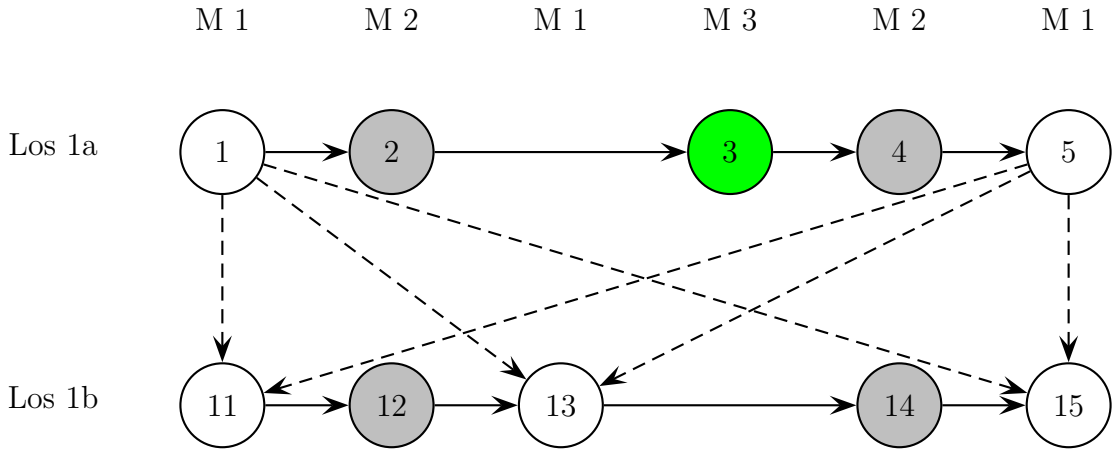

Abb. 3/10: Zusätzliche Vorgängerbeziehungen zwischen Los 1a und Los 1b für Maschine 1

if $u) = s_a$ **then**
$r(u) \leftarrow r_j$ **else**$r(u) \leftarrow 0$
end if**end if**
for all $v \in virtual\_edges_u$ **do**
$pred(u) \leftarrow pred(u) \cup \{v\}$
end for
end for
while Knotenstapel $\neq \emptyset$ **do** $\{Traversierung\}$
$u \leftarrow$ POP(Knotenstapel)
for all $(v, u) \in source_u$ **do**
$r(u) \leftarrow max(r(v) + k(v, u))$
end for
for all $(u, v) \in destination_u$ **do**
Besuchte Kanten $\leftarrow$ Besuchte Kanten $\cup \{(u, v)\}$
$pred(v) \leftarrow pred(u) \cup \{u\}$
if $source_v \setminus$ Besuchte Kanten $= \emptyset$ **then**
PUSH(Knotenstapel, v)
end if
end for
end while

Plausible Verfügbarkeitszeiten sollten bereits durch die obere Planungsebene geliefert werden, so dass man hierauf keine gesonderte Rücksicht nehmen muss.

3.6.2 Verfahren

Die grundlegende Idee für die Koordination der Bereiche ist wiederum ein Shifting-Bottleneck-Ansatz. In diesem Fall sind die Bereiche die Maschinen. Allerdings sind die Bereiche nur über die Losrouten verbunden. Es existiert kein disjunkter Graph auf Fabrikebene.

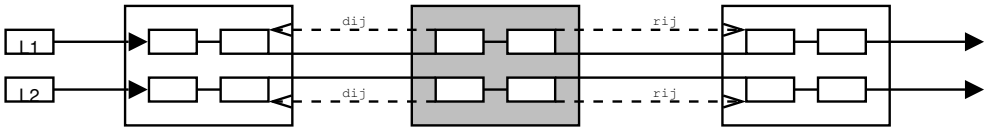

Abb. 3/11: Synchronisation zwischen 3 Bereichen

Zunächst werden für alle Bereiche Schedules mit Hilfe der Shifting-Bottleneck-Heuristik ermittelt. Wenn dies geschehen ist, dann werden die Zeiten an den Grenzen von zwei Bereichen mit ziemlicher Wahrscheinlichkeit nicht mehr konsistent sein. Bei der Betrachtung eines Loses muss die Fertigstellung eines Bereichsabschnitts gleich dem frühesten Eintrittszeitpunkt in dem nachfolgenden Bereich sein. Gleichzeitig kann man die Forderung aufstellen, dass der späteste Eintrittzeitpunkt eines Bereichs gleich dem spätesten Fälligkeitstermin des Vorgängerbereichs entsprechen muss. Beide Werte lassen sich durch eine einfache Vorwärts- bzw. Rückwartskalkulation ermitteln.

Um herauszufinden, in welcher Reihenfolge die Bereiche zu synchronisieren sind, muss für jeden Bereich eine Kritikalität bestimmt werden. Aufbauend auf diesen Überlegungen lassen sich zwei Koordinationsalgorithmen formulieren.

3.6.2.1 Koordinierung des kritischen Bereichs

Hierbei erfolgt nur die Auswahl eines kritischen Bereichs, an den sich die anderen Bereiche anpassen müssen. Der Algorithmus ist sehr einfach:

Koordinierung des kritischen Bereichs innerhalb der DSBH
Kritischer Graph $\leftarrow 0$
K $\leftarrow 0$
for all $G \in GRAPHS$ **do**
    SCHEDULE GRAPH(G)
    AKTUALISIERE NACHBARGRAPHEN(G)
    K[G] $\leftarrow \sum_{j=1}^{n} w_j \cdot T_j$
    SETZE NACHBARGRAPHEN ZURÜCK(G)
    if $K[G] > K$ **then**
        K $\leftarrow$ K[G]
        Kritischer Graph $\leftarrow$ G
    end if
end for
AKTUALISIERE NACHBARGRAPHEN(Kritischer Graph)
for all $G \in GRAPHS \setminus \{KritischerGraph\}$ **do**

aktualisiere Nachbargraphen(G)
end for

3.6.2.2 Dynamische Koordinierung Dieser Ansatz ähnelt eher dem der Shifting-Bottleneck-Heuristik. Der Algorithmus ist wie folgt definiert:

Dynamische Koordinierung innerhalb der DSBH
Geschedulte Graphen $\leftarrow \emptyset$
repeat
  Kritischer Graph $\leftarrow 0$
  K $\leftarrow 0$
  for all $G \in GRAPHS \setminus \{GeschedulteGraphen\}$ **do**
    schedule Graph(G)
    aktualisiere Nachbargraphen(G)
    K[G] $\leftarrow \sum_{j=1}^{n} w_j \cdot T_j$
    setze Nachbargraphen zurück(G)
    if $K[G] > K$ **then**
      K $\leftarrow$ K[G]
      Kritischer Graph $\leftarrow$ G
    end if
  end for
  Geschedulte Graphen $\leftarrow$ Geschedulte Graphen $\cup\{KritischerGraph\}$
  aktualisiere Nachbargraphen(Kritischer Graph)
until Geschedulte Graphen = GRAPHS

Auf die Einführung einer Reoptimierung wie bei der originalen Shifting-Bottleneck-Heuristik wurde in dieser Arbeit verzichtet, da dies die Laufzeitersparnis wieder zunichte gemacht hätte.

3.6.3 Aktualisierung der Nachbargraphen

Die hauptsächliche Koordinierungsarbeit leistet die Methode zum aktualisieren der Nachbargraphen. An dieser Stelle soll darauf näher eingegangen werden. Es existieren aufgrund der zyklischen Abhängigkeiten zwischen den einzelnen Bereichen und den betrachteten Planungshorizont mehrere Ausbaustufen dieser Methode. Da die Koordinierung nur auf Losebene stattfindet, genügt es, wenn die folgenden Betrachtungen sich auf ein Los j beziehen.

Koordinierung ohne Zyklenberücksichtigung

Dies ist der einfachere und häufiger auftretende Fall. Der betrachtete Planungshorizont schränkt das Problem meistens soweit ein, dass Zyklen zwischen den Bereichen nicht mehr berücksichtigt werden müssen.

Für den Vorgängerbereich eines Loses müssen in diesem Fall die spätesten Fälligkeitstermine angepasst werden. Betrachten werden zwei Knoten $u \in V_1$ und $v \in V_2$

der Graphen 1 und 2, wobei u ein direkter Vorgänger von v in der Losroute des Jobs j ist. Der Knoten v wird auf der Maschine a der Maschinengruppe i abgearbeitet.

$$d_u = d_{i,j} - (l_a + p_{i,j} + u_a + s_{k,j}) \tag{16}$$

Im nachfolgenden Bereich eines Loses müssen die frühesten Verfügbarkeitstermine angepasst werden. Betrachtet werden zwei Knoten $u \in V_2$ und $v \in V_3$ der Graphen 2 und 3, wobei u ein direkter Vorgänger von v in der Losroute des Jobs j ist. Der Knoten u wird auf der Maschine a der Maschinengruppe i abgearbeitet.

$$r_v = r_{i,j} + (l_a + p_{i,j} + u_a + s_{k,j}) \tag{17}$$

Koordinierung mit Zyklenberücksichtigung

Wie bereits erwähnt, reicht bei einem kleinen Planungshorizont diese Art der Koordinierung schon aus. Bei einem größeren Planungshorizont kann eine rekursive Behandlung der Zyklen erforderlich sein. Folgender Pseudocode verdeutlicht die Vorgehensweise.

Aktualisiere Nachfolger
VORWAERTSKALKULATION(G)
zu aktualisierende Graphen $\leftarrow \emptyset$
for all $j \in 1 \ldots n$ **do**
    $u \leftarrow$ SUCHE LETZTEN KNOTEN(G, j)
    $v \leftarrow$ SUCHE NACHFOLGEKNOTEN(j, u)
    if $v \neg 0$ **then**
        $r_v = r_{i,j} + (l_a + p_{i,j} + u_a + s_{k,j})$
        $G_2 \leftarrow$ SUCHE GRAPH(v)
        zu aktualisierende Graphen $\leftarrow$ zu aktualisierende Graphen $\cup\{G_2\}$
    end if
end for
for all *graph* $\in$ zu aktualisierende Graphen **do**
    AKTUALISIERE NACHFOLGER(*graph*)
end for

Die Aktualisierung der Vorgänger erfolgt ähnlich:

Aktualisiere Vorgaenger
RUECKWAERTSKALKULATION(G)
Zu aktualisierende Graphen $\leftarrow \emptyset$
for all $j \in 1 \ldots n$ **do**
    $v \leftarrow$ SUCHE ERSTEN KNOTEN(G, j)
    $u \leftarrow$ SUCHE VORGAENGERKNOTEN(j, u)
    if $v \neg 0$ **then**
        $d_u = d_{i,j} - (l_a + p_{i,j} + u_a + s_{k,j})$

$G_2 \leftarrow$ Suche Graph(u)
zu aktualisierende Graphen $\leftarrow$ zu aktualisierende Graphen $\cup\{G_2\}$
end if
end for
for all *graph* $\in$ zu aktualisierende Graphen **do**
Aktualisiere Vorgaenger(*graph*)
end for

Rückmeldung an die obere Ebene

Eine weitere Möglichkeit ist die Rückmeldung der Änderungen an die obere Ebene und das Neubestimmen der Verfügbarkeitsdaten mithilfe des oberen Planungsalgorithmus.

Die rekursive Behandlung der Vorgängerbeziehungen kann sehr teuer werden, da die Graphen unter Umständen mehrmals neu berechnet werden müssen – in Abhängigkeit vom Planungshorizont. In vielen Fällen dürften die Puffer, welche die obere Ebene vorgibt oder ein kleinerer Planungshorizont die Rekursion zu vermeiden helfen.

3.6.4 Kritikalitätsmaße für die verteilte Shifting-Bottleneck-Heuristik

Wie bei den Kritikalitätsbetrachtungen zu den einzelnen Maschinengruppen, kann man auch bei der Bereichskritikalität zwischen mehreren Optionen wählen:

- Die Kritikalität der Bereiche wird durch externe Daten vorgegeben. Diese externen Daten können beispielsweise durchschnittliche Kapazitätsauslastungen oder die Häufigkeit der Maschinenausfälle sein.
- Die Kritikalität wird nur lokal durch Berücksichtigung der Arbeitsgänge des jeweiligen Bereichs ermittelt.
- Die Kritikalität wird global bestimmt. Der Schedule des aktuellen Graphen wird festgehalten und die Nachbargraphen mit den neuen Zeiten aktualisiert. Am Ende wird die Kritikalität global für alle Lose bestimmt.

Innerhalb dieser Arbeit wird die letzte Variante verwendet, da sie die besten Ergebnisse liefern dürfte (Durch Berücksichtigung globaler Kriterien wird das Ziel eher erreicht als durch lokale Kriterien). Die Laufzeitkomplexität ist bei dieser Variante natürlich am größten. Jedoch kann dies in Kauf genommen werden. Größeren Einfluss hat das Verfahren der Koordinierung.

Die Kritikalität ist auch hier die absolute gewichtete Verspätung der einzelnen Jobs:

$$K(a) = \sum_{j=1}^{n} w_j \cdot T_j \tag{18}$$

Innerhalb der Experimente hat sich herausgestellt, dass es auch in diesem Fall von Vorteil ist, wenn immer der kritischste Bereich eingeplant wird. Eine Umkehrung der Logik hat zu schlechteren Ergebnissen geführt.

3.7 Umsetzung der Schedules auf die Maschinen

Ein Problem, welches bisher noch nicht behandelt wurde, ist die Art und Weise der endgültigen Umsetzung der Schedules in der Fabrik – oder im Fall dieser Arbeit in dem Simulator. Der Bereichsscheduler speichert die Schedules pro Maschine als eine Folge von Batches. Zu jedem Batch wird zusätzlich die berechnete Startzeit auf der Maschine gespeichert. Wenn eine Maschine frei wird, dann wird eine Dispatchregel aufgerufen. Diese bestimmt anhand des Schedules der aktuellen Maschine, welche Lose aus der Warteschlange bearbeit werden sollen.

Es kann allerdings sein, dass die berechneten Startzeitpunkte nicht mit den Ankunftszeitpunkten der Lose an den Maschinen synchronisieren. Es kann hierfür verschiedene Gründe geben. So ist es beispielsweise möglich, dass eine Maschine ausgefallen ist. Der Scheduling-Horizont könnte allerdings auch zu knapp gewählt worden sein und demzufolge bilden die Schedules nicht alles ab.

Die Beschäftigung mit diesem Problem wirft zwei prinzipielle Fragen auf:

- Wie stellt man fest, dass es eine relevante Abweichung gegeben hat? Das Schwierige an dieser Frage ist der Punkt der Relevanz. Herauszufinden, ob es eine Abweichung von den Planvorgaben gibt, ist relativ einfach. Problematisch ist es allerdings, herauszufinden, ob diese Abweichung relevant war, so dass man zur zweiten Frage kommen kann.
- Wie reagiert man auf diese Abweichung? Wenn man festgestellt hat, dass die Abweichung relevant war, muss man sich überlegen, was man tun kann, um den Plan wieder mit dem Prozess zu synchronisieren.

Innerhalb dieser Arbeit wurde ein sehr einfacher Umsetzungsmechanismus gewählt. Insgesamt kann es dabei mehrere Möglichkeiten geben:

1. Die Warteschlange der Maschinengruppe enthält wartende Lose.
    a) Die Maschine verfügt über einen Schedule.
        i. Die Startzeit des ersten Batches im Schedule ist kleiner oder gleich der aktuellen Zeit – der Batch wird gebildet und auf die Maschine gestellt.
        ii. Die Startzeit des ersten Batches ist größer als die aktuelle Zeit – alles ist in Ordnung. Die Maschine muss noch warten.
        iii. Der erste Batch enthält Lose, die nicht in der Warteschlange vorkommen und die Startzeit ist kleiner als die aktuelle Zeit
            A. Die Lose werden auf einer anderen Maschine abgearbeitet – alles in Ordnung
            B. Die Lose sind in den Schedules für diese Maschinengruppe nicht richtig berücksichtigt. – FIFO Rückfall.
    b) Der Schedule der Maschine ist leer.
        i. Die Lose werden auf einer anderen Maschine abgearbeitet – alles in Ordnung

ii. Die Lose sind in den Schedules für diese Maschinengruppe nicht richtig berücksichtigt. – FIFO Rückfall.

2. Die Warteschlange der Maschinengruppe ist leer.

Man kann diese Dispatchregel noch weiter verfeinern. Die Auswirkungen des FIFO-Rückfalls auf die zukünftigen Schedules sind zum Beispiel nicht berücksichtigt. Des weiteren ist ein Rescheduling nicht vorgesehen.

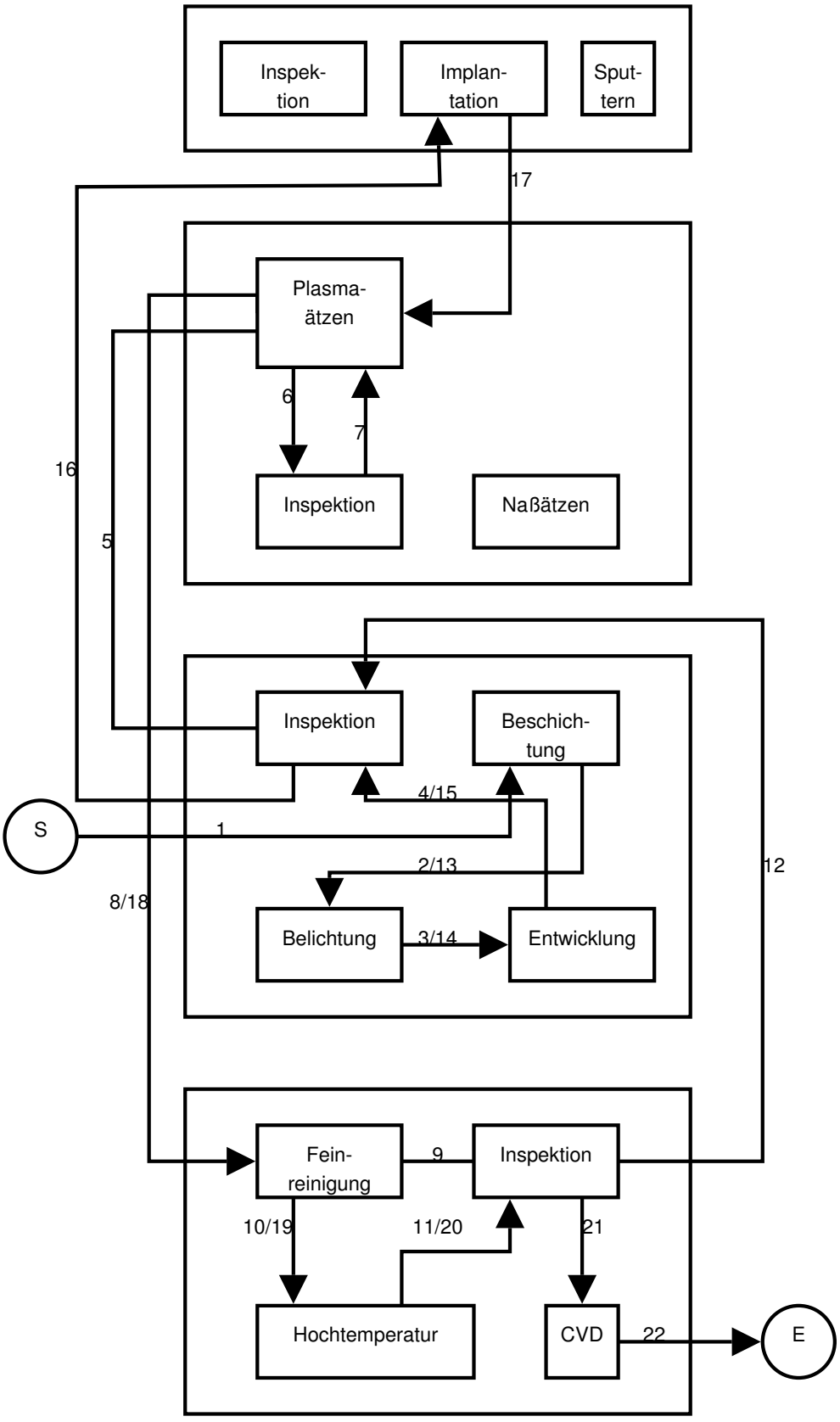

Abb. 3/12: Losfragmentierung bei der Verteilten Shifting-Bottleneck-Heuristik

4 Prototypische Umsetzung einer verteilten Shifting-Bottleneck-Heuristik

Dieses Kapitel beschäftigt sich mit der Implementation der verteilten Shifting-Bottleneck-Heuristik. Zur Überprüfung der Leistungsfähigkeit wird die Heuristik zunächst mit Hilfe der Programmiersprache C++ implementiert. Bei dieser Umsetzung erfolgt zunächst nur eine rein fachliche Verteilung, um die Brauchbarkeit des Ansatzes zu überprüfen. Die Überprüfung der Ergebnisse und die Datenbereitstellung erfolgt mit Hilfe eines Balckboards, welches in [Mönch u. a. 2002] vorgestellt wurde. In einem zweiten Schritt wird die Heuristik danach in Teilen in das Multiagentensystem FABMAS integriert. Auf die Besonderheiten beider Implementationen soll im Folgenden eingegangen werden.

4.1 Implementation in der Programmiersprache C++

Der Hauptgrund für die Wahl der Programmiersprache C++ liegt in der Tatsache begründet, dass die Schnittstellenbibliotheken des Simulators [Brooks Automation 2001a] als auch das verwendete Blackboard in dieser Sprache implementiert sind. Außerdem erweitert C++ die objektorientierte Programmierung durch generische Ansätze in Form von Templates. Stroustrup [Stroustrup 2000] bietet eine gute Einführung in die Sprache C++ und in die Verwendung von Templates. In vielen Fällen sind Implementationen in C++ gleich schnell wie eine entsprechende Implementation in C [Stroustrup 2000]. Der Aspekt der Geschwindigkeit und die Verfügbarkeit einer Template Bibliothek (Standard Template Library - STL) machen C++ ideal für die Implementierung von Scheduling-Algorithmen. Als Entwicklungsumgebung wurde Microsoft Visual Studio 6.0 verwendet.

4.1.1 Verwendete Bibliotheken, Datenstrukturen und Algorithmen

Aus den zuvor aufgerührten Gründen wird in dieser C++ Implementation häufiger Gebrauch von den Standarddatenstrukturen und den Standardalgorithmen der STL gemacht. Hierzu zählen unter anderen [Stroustrup 2000]:

std::set
: Ein set verwaltet einer Menge von Objekten. Innerhalb dieses Containers werden die einzelnen Objekte sortiert. Dadurch hat der lesende und schreibende Zugriff ein logarithmisches Zeitverhalten, da für beides eine binäre Suche verwendet werden kann. Gleichzeitig ist garantiert, dass ein Objekt nur einmal in einem set vorkommen kann. Diese Datenstruktur wird verwendet, um die mathematischen Mengen aus dem vorigen Kapitel abzubilden.

std::list
: Diese Datenstruktur ist nicht sortiert. Sie stellt eine doppelt verkettete Liste zur Verfügung. Das Einfügen und Entfernen von Elementen ist sehr effizient, wohingegen nur eine lineare Suche möglich ist. Die Schedules werden mit Hilfe von Listen abgebildet.

std::map
: Dieser assoziative Container dient zum Speicher von Schlüssel-Wert-Paaren. Die Schlüsselmenge verhält sich wie ein set, das heißt die Schlüssel sind sortiert und ein Schlüssel kann nur einmal vorkommen. Mit einer map lassen sich Klassifizierungen abbilden, wenn der Werttyp ein Container ist. Innerhalb der Unterproblemlöser dienen maps zum Beispiel dazu, die Arbeitsgänge nach Batchfamilien zu gruppieren.

std::hashset
: Ein hashset bietet die selben Schnittstellen wie ein normales set. Allerdings ist der Container nicht sortiert, sondern die Positionen der Objekte werden über eine Hashfunktion berechnet. Dadurch sind Einfüge-, Lösch- und Suchoperationen sehr schnell. Allerdings ist der Speicherbedarf eines Hashes im Schnitt größer als der eines normalen sets. Bei großen Datenmengen bringen hashsets eine starke Verbesserung der Laufzeitperformance. Innerhalb des Schedulers werden hashsets dort eingesetzt, wo häufige Einfüge- und Löschoperationen nötig sind.

Die STL bietet für verschiedene häufig auftretende Probleme Standardalgorithmen (in Form von Templatefunktionen) an. Die verwendeten Algorithmen und ihre Komplexität werden im Folgenden kurz erläutert.

std::sort
: Diese Templatefunktion dient zum Sortieren des übergebenen Containers. Wonach sortiert werden soll, kann als optionaler Parameter angegeben werden. Nach [Stroustrup 2000] sollte diese Methode dem normalen qsort der Programmiersprache C vorgezogen werden, da sie häufig effizienter und – im Bezug auf die verwendeten Datentypen – allgemeingültiger implementiert ist.

std::find
: Mit find gibt es eine generische Funktion zum Suchen innerhalb eines Containers. Sortierte Container, wie set und map, implementieren eigene find Methoden, die die Vorteile der Sortierung nutzen.

std::set_difference
: Für die Arbeit mit Mengen bietet die STL verschiedene Algorithmen an. Die Funktion set_difference bildet die Schnittmenge aus zwei gegebenen Mengen.

Die von Microsoft mitgelieferte STL beinhaltete leider keine hashset Implementation, da diese nur eine SGI-Erweiterung [STL a] der STL ist. Da die Verwendung dieser Datenstukturen allerdings Performancegewinne versprechen wurden verschiedene STL Implementationen untersucht. Die Wahl fiel letztendlich auf die STLPort Bibliothek [STL b]. Die Gründe sollen an dieser Stelle zusammengefasst werden:

- Die Verwendung dieser Software ist kostenlos. Da es sich nur um einen Prototyp handelt macht eine kostenpflichtige STL-Implementation keinen Sinn.
- Die Bibliothek ist Freie Software. Eine Erläuterung der Vorteile Freier Software findet sich in [Stallman 2002].

- Wie der Name STLPort bereits andeutet, ist die Bibliothek auf einer Reihe von Plattformen mit verschieden Compilern verfügbar. Des Weiteren ist die Implementation konform zum STL Standard (die Microsoft STL ist dies nicht). Eine Portierung auf andere Plattformen, Compiler oder gar andere STL-Bibliotheken ist somit leicht möglich.
- Andere freie STL Implementationen – wie die GNU Implementation [GNU] – haben Probleme mit den proprietären Microsoft Visual Studio und den proprietären ASAP Bibliotheken.
- Die Entwicklergemeinde dieser Bibliothek ist sehr groß. Die Weiterentwicklung und der Support sind damit sichergestellt. Desweiteren existiert eine Firma (STLport Consulting), welche gegebenenfalls kostenpflichtigen Support anbieten kann.
- Auf Grund der Lizenz ist auch eine Verwendung in proprietärer Software – wie dieser Implementation – möglich.

Da diesen Vorteilen keine offensichtlichen Nachteile gegenüber stehen, wurde die Entscheidung für STLPort getroffen.

4.1.2 Klassenmodelle

In diesem Abschnitt erfolgt eine Erläuterung der implementierten Datenstrukturen. Die Art der Daten kann man grob in statische und dynamische Daten unterteilen. Abbildung 4/1 zeigt alle Klassen, welche zu den statischen Daten gehören. Hier sind der Scheduler an sich, die Bereiche in Form der Klasse Graph sowie die Maschinengruppen (Klasse ToolGroup) und die Maschinen (Klasse Tool) zu erwähnen.

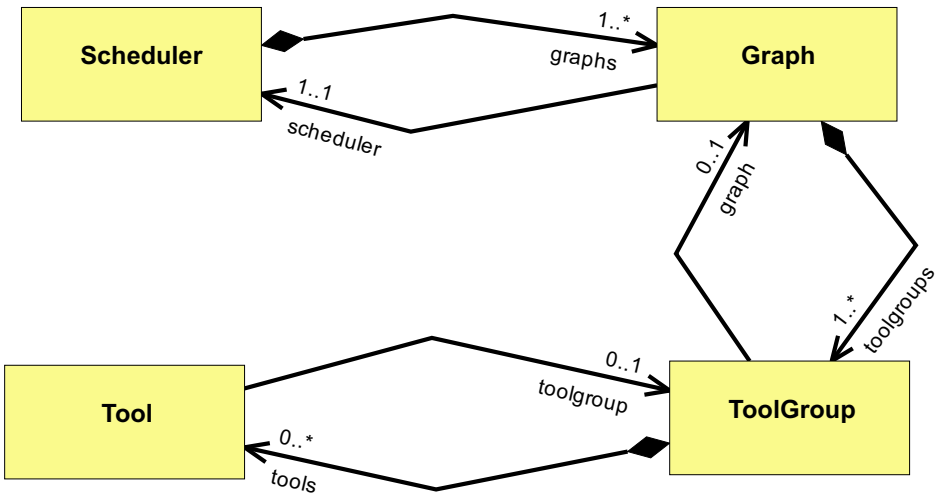

Abb. 4/1: Statisches Datenmodell in der DSBH

Abbildung 4/2 zeigt die Erweiterung des statischen Modells um dynamische Elemente. Hierzu gehören die Lose an sich (Klasse Job), die Bereichabschnitte der einzelnen Lose (Klasse Operation) und die verschiedenen Knotenarten.

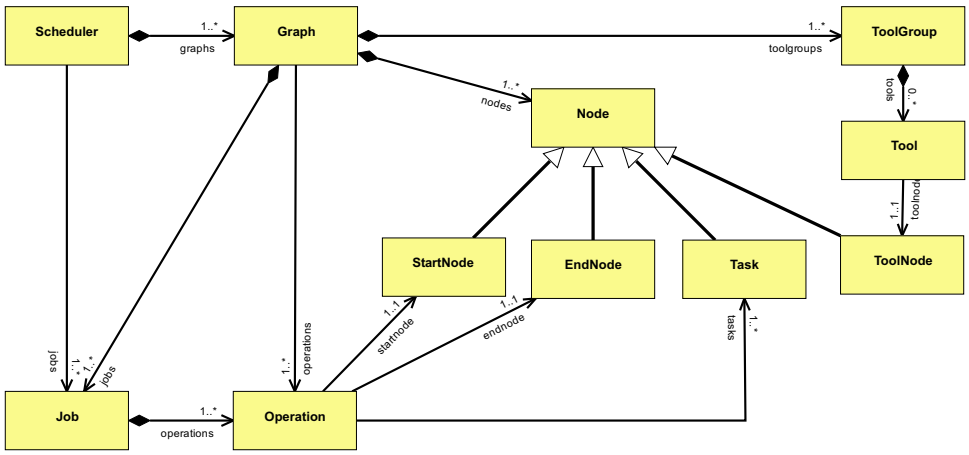

Abb. 4/2: Dynamisches Datenmodell in der DSBH

Im Folgenden werden die einzelnen Klassen detaillierter erläutert:

4.1.2.1 Klasse Scheduler

Diese Klasse verwaltet die Erzeugung und Zerstörung der einzelnen Bereichsgraphen. Die Speicherung der Schedules in das Blackbord und die Koordinierung der Bereichsgraphen erfolgt ebenfalls in dieser Klasse. Es existiert während der Simulation nur eine Instanz dieser Klasse.

4.1.2.2 Klasse Graph

Diese Klasse repräsentiert einen Bereich in der Halbleiterfabrik. Die Fabrik in Abbildung 3/12 beispielsweise zeigt vier verschiedene Bereiche. Für jeden von ihnen wird eine Instanz dieser Klasse angelegt. Ein Graph Objekt verwaltet weiterhin Mengen der enthaltenen Maschinengruppen und Maschinen. Die einzelnen Losabschnitte, die den jeweiligen Graphen durchlaufen sind ebenfalls mitgeführt.

4.1.2.3 Klasse ToolGroup

Ein ToolGroup-Objekt repräsentiert eine einzelne Maschinengruppe innerhalb eines Bereichs. Ein Objekt dieser Klasse führt Listen über die zugehörigen Maschinen (Tools), die zu bearbeitenden Arbeitsgänge und verwaltet die Unterproblemlöser für die entsprechende Maschinengruppe.

4.1.2.4 Klasse Tool

Eine Maschine wird durch die Klasse Tool abgebildet. Sie verwaltet alle wichtigen Informationen für eine Maschine. Die Lade- und Entladezeiten, Rüstinformationen und die Schedules einer Maschine werden in Toolobjekten gespeichert. Die Schedules werden als eine Liste von Batches gespeichert (Abbildung 4/3). Diese Batches fassen wiederum mehrere Tasks zusammen.

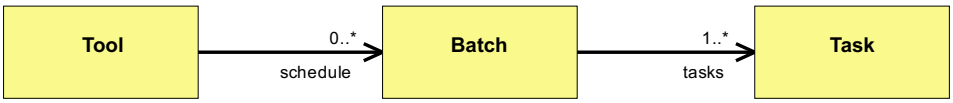

Abb. 4/3: Scheduleabbildung in der DSBH

4.1.2.5 Klasse Job

Diese Klasse repräsentiert ein komplettes Los, welches in allen Bereichen abgearbeitet wird. Ein Los besteht aus mehreren Bereichsabschnitten (Operationen). Diese wiederum bestehen aus mehreren Arbeitsgängen (Tasks). Neben Listen zu den Operationen und den Tasks enthält ein Job auch ein Verfügbarkeitsdatum (engl. release date) und ein Fälligkeitsdatum (engl. due date).

4.1.2.6 Klasse Operation

Wie bereits weiter oben erläutert, repräsentiert eine Instanz der Klasse Operation einen Teil der Losroute, dessen Grenzen durch einen Bereich bestimmt sind. Eine Operation enthält Daten, den geplanten Eintrittszeitpunkt und den geplanten Austrittszeitpunkt, aus dem jeweiligen Bereich. Diese Informationen werden von der oberen Planungsebene bezogen.

4.1.2.7 Klasse Node

Ein Knoten innerhalb des Graphen wird durch diese Klasse abgebildet. Es existieren weitere Spezialisierungen für Start-, End-, Maschinen- und Arbeitsgangknoten. Ein Knoten besitzt ein – über den Graphen berechnetes – frühestes Verfügbarkeitsdatum sowie einen spätesten Fälligkeitstermin. Mit Hilfe der mitgeführten Vorgängerknoten kann, wie im vorherigen Kapitel erläutert, das tatsächliche Verfügbarkeitsdatum innerhalb eines Unterproblems besser abgeschätzt werden. Ein Knoten verwaltet weiterhin seine Vorgänger- und Nachfolgekanten.

4.1.2.8 Klasse Edge

Diese Klasse repräsentiert eine Kante innerhalb des Graphen. Eine Kante besteht aus

einem Quellknoten, einem Zielknoten sowie einer Kantenlänge (Kantengewicht). Im Laufe des Algorithmus wird dieses Gewicht nach Formel 4 verändert, beziehungsweise neue Kanten in den Graphen eingefügt oder entfernt. Eine Änderung an den Kanten macht eine Neuberechnung des Graphen nötig.

4.1.2.9 Klasse Task

Diese Klasse ist von Node abgeleitet und stellt einen normalen Arbeitsgang eines Jobs dar. Die Eigenschaften eines Knotens werden durch Setup-, Prozesszeit sowie Be- und Entladeinformationen erweitert.

4.1.2.10 Klasse StartNode

Ein Objekt dieser Klasse repräsentiert einen künstlichen Startknoten eines Loses. Das früheste Verfügbarkeitsdatum dieses Knotens ist das Verfügbarkeitsdatum des zugehörigen Jobs.

4.1.2.11 Klasse EndNode

Äquivalent zum Startknoten stellt diese Klasse einen Endknoten dar. Der späteste Fälligkeitstermin ist der Fälligkeitstermin des zugehörigen Jobs.

4.1.2.12 Klasse ToolNode

Um die Maschinenverfügbarkeitszeiten abbilden zu können, wird ein künstlicher Maschinenknoten an den Anfang jedes zu implementierenden Schedules gestellt. Das früheste Verfügbarkeitsdatum dieses Knotens ist das Verfügbarkeitsdatum der zugehörigen Maschine.

4.1.2.13 Klasse SSP

Diese Klasse stellt eine abstrakte Schnittstelle für die jeweiligen Unterproblemlöser zur Verfügung. Im Wesentlichen enthält sie zwei Schnittstellen, welche alle Unterproblemlöser implementieren müssen:

initialize_problem
: Diese Methode dient zum Initialisieren des Problems. Meistens erfolgt hier eine Sortierung der Arbeitsgänge, um den Suchaufwand innerhalb des Algorithmus geringer zu halten.

solve_problem
: Hier erfolgt die Lösung des Unterproblems. Im folgenden Abschnitt werden einige Unterproblemlöser vorgestellt.

4.1.3 Unterproblemlöser

In dieser Arbeit wurden zwei Arten von Unterproblemlösern implementiert. Der eine Ansatz ist ein Dispatch Ansatz. Hierbei werden die einzelnen Arbeitsgänge nach ihrer Verfügbarkeit auf die einzelnen Maschinen eingeplant. Der zweite Ansatz ist ein listenbasierter Ansatz (engl. list based scheduling). Hierbei erfolgt eine Einplanung nur anhand der errechneten Priorität eines Arbeitsgangs.

Sowohl der Dispatch Ansatz als auch der Listenbasierte Ansatz errechnen eine dynamische Priorität für die einzelnen Arbeitsgänge, wonach sie dann eingeplant werden. Der im Rahmen dieser Arbeit mitgelieferte Datenträger enthält eine Implementation dieser zwei Unterproblemlöseransätze in den FIFO, CR, Slack und ATC-Varianten.

4.2 Implementation innerhalb des Multiagentensystems FABMAS

Die Implementation innerhalb des Multiagentensystems orientiert sich an der C++ Implementation. Da es sich im Unterschied zur vorherigen Implementation um ein tatsächlich verteiltes System handelt, müssen einige Abwandlungen vorgenommen werden. Eine vollständige Erläuterung des FABMAS Systems würde an dieser Stelle zu weit führen. Es sei deswegen auf die Arbeiten [Mönch u. a.] und [Zimmermann 2003] verwiesen. Eine gute Einführung in die Agententechnologie bietet [Kirn 2001].

Die Datenbereitstellung erfolgt in diesem System ebenfalls wieder über ein Blackboard. Dieses ist in Art und Struktur dem ähnlich, welches in der C++ Implementation verwendet wurde (vgl. [Mönch u. a. 2002]). Die einzelnen Entitäten der Halbleiterfabrik werden allerdings zusätzlich als Agenten abgebildet. Im Folgenden sind die wichtigsten Agentenarten kurz aufgeführt (vgl. [Mönch u. a.] und [Zimmermann 2003]):

Fabrikagent
: Der Fabrikagent verwaltet die gesamte Fabrik. Er ist gleichzeit der einzige Entscheideragent innerhalb des System. Er trifft die Entscheidung, wann ein Scheduling durchzuführen ist und ob das Ergebniss verwendet wird.

Bereichsagent
: Der Bereichsagent verwaltet einen einzelnen Bereich der Fabrik. das Multiagentensystem ist so konzipiert, das jeder einzelnen Bereich auf einem separatem Rechner laufen kann.

Maschinengruppenagent
: Dieser Agent verwaltet eine einzelne Maschinengruppe. Die Maschinen innerhalb dieser Gruppe werden als Objekte repräsentiert.

Losagent
: Dieser Agent verwalten ein einzelnes Los. Losagenten sind die einzigen mobilen Agent innerhalb dieses Systems. Dies ist nötig, da sie zwischen den einzelnen Bereichen wandern können.

Dienstagent
: Ein Dienstagent implementiert einen Scheduling-Algorithmus. Im Rahmen dieser Diplomarbeit ist jedem Bereich ein entsprechender SBH Dienstagent zugeordnet.

Bei der Implementierung in das Multiagentensystem musste der Algorithmus nur minimal angepasst werden. Die bedeutendste Einschränkung ist der Wegfall der Klasse Scheduler. Diese geht in dem Fabrikenagenten auf. Anstatt die Koordination durch direkten Zugriff auf die Objekte durchzuführen, kommuniziert der Fabrikagent mit den einzelnen Dienstagenten über Nachrichten. Ebenfalls von der Änderung betroffen ist die Klasse Job. Da die Losroute über mehrere Bereiche verteilt ist existieren in dem entsprechenden Dienstagenten nur noch Fragmente der globalen Losroute – repräsentiert durch die Klasse Operation. Abbildung 4/4 zeigt das Klassenmodell für einen Dienstagenten.

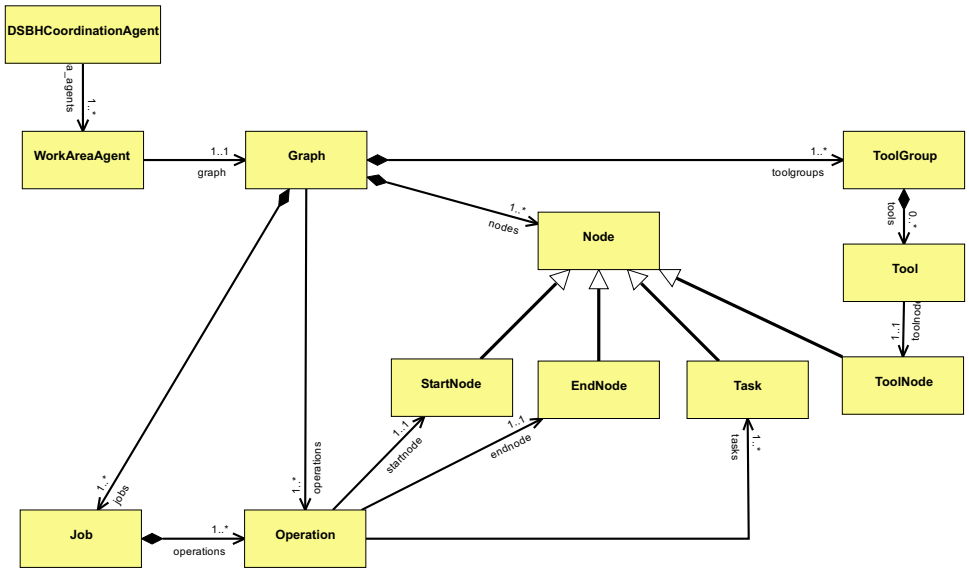

Abb. 4/4: Datenmodell eines Bereichsagenten in der DSBH

5 Simulationsbasierte Leistungsbewertung

5.1 Architektur zur Leistungsbewertung

Um die Leistungsfähigkeit der verteilten Shifting-Bottleneck-Heuristik zu messen, wurden verschiedene Simulationsstudien durchgeführt. Die verwendete Software ist AutoSched AP 7.1 [Brooks Automation 2001b]. AutoSched AP ist ein diskreter ereignisorientierter Simulator, der speziell die Simulation von Halbleiterfabriken unterstützt. Im Umfeld des Halbleiterscheduling wird dieses Programm häufig verwendet, da es gute Modellierungsmöglichkeiten für diese Domäne anbietet.

Der Simulator bietet verschiedene Möglichkeiten der Erweiterung [Brooks Automation 2001a]. So ist es beispielsweise möglich, eigene Attribute und Aktionslisten für die einzelnen Modellelemente zu definieren. Weiterhin bietet AutoSched AP eine C++ Bibliothek, an mit deren Hilfe u.a. höhere Scheduling-Algorithmen implementiert werden können. Für die Leistungsanalyse wurde ein Blackboard verwendet, das in [Mönch u. a. 2002] beschrieben wurde. Mit Hilfe dieses Blackboard wird nicht nur die Datenversorgung des Algorithmus sichergestellt, sondern auch die Werte für die Leistungsmaße ermittelt.

5.2 Versuchsplanung zur Leistungsbewertung

5.2.1 Verwendete Fabrikumgebungen

In den Simulationen wurden fünf verschiedene Modelle von Halbleiterfabriken getestet. Im Folgenden sind die Modelle der Größe nach geordnet aufgeführt:

SmallFab

Dies ist das kleinste Modell (Abbildung 5/1). Es umfasst drei Maschinengruppen mit insgesamt sechs Maschinen. Eine dieser Maschinengruppen besteht aus Batchmaschinen. Des Weiteren besitzt dieses Modell drei Arbeitspläne mit jeweils sechs Schritten. Außerdem ist in den Arbeitsplänen ein einfacher Zyklus (d.h. alle Maschinen werden bei der Bearbeitung eines Loses zweimal in Anspruch genommen) eingebaut [Adl u. a. 1996]. Dieses Modell ist hauptsächlich für Entwicklungszwecke interessant, da es die wichtigsten Kriterien abbildet. Für realistische Untersuchungen ist es jedoch weniger geeignet.

AreaFab

Um den Koordinationsmechanismus in einer einfachen Umgebung zu testen wurde das SmallFab Modell vergrößert. Die ursprüngliche Fabrik wurden verdoppelt, so dass zwei Bereiche mit je drei Maschinengruppen entstanden. Die Routen wurden entsprechend auf zwölf Schritte erweitert. In diesem Modell (Abbildung 5/2) lies sich die Koordination nach dem kritischen Bereich testen.

AreaFab2

Für eine Testmöglichkeit der dynamischen Koordination sind mindestens drei Bereiche nötig. Das vorherige AreaFab Modell wurde um einen zusätzlichen Bereich

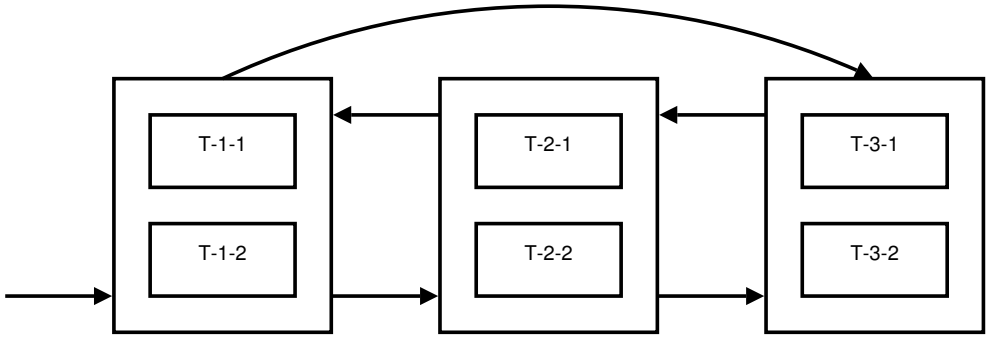

Abb. 5/1: Arbeitspläne im Modell SmallFab

erweitert. Die Arbeitspläne umfassen hier 24 Schritte und der erste Bereich wird in diesem Modell zweimal durchlaufen. Mit diesem Modell (Abbildung 5/7) lassen sich alle Varianten der DSBH testen.

QuarterFab

Da die AreaFab Modelle lediglich akademischer Natur sind, ist es nötig die Performance des Algorithmus an einem realistischeren Modell zu testen. Um die Simulationszeiten zu verkürzen wurde das vorhandene SemiFab Modell verkleinert. Dies geschah durch das Halbieren der Arbeitspläne. Dieses Modell umfasst 146 Maschinen in vier Bereichen und zwei Routen mit 100 beziehungsweise 103 Schritten.

SemiFab

Das SemiFab Modell ist ein Modell einer Halbleiterfabrik. Es handelt sich dabei um das „MIMAC Testbed Data Set 1“, welches öffentlich zugänglich ist und in zahlreichen Simulationsstudien Verwendung findet [MIM].

Für die Verwendung mit den Blackboard hat es – wie die anderen Modelle auch – einige Veränderungen erfahren. So mussten beispielsweise die einzelnen Bereiche hinzugefügt und die kritischen Maschinen innerhalb der Bereiche markiert werden.

Dieses Modell umfasst 270 Maschinen in fünf Bereichen und zwei Routen mit 210 beziehungsweise 245 Schritten.

SemiFab stellt das (idealisierte) Modell einer mittelgroßen Halbleiterfabrik dar. Die Aussagen auf Basis dieser Simulationsstudien sind insoweit brauchbar, dass sie eine Tendenz für größer werdende Systeme erkennen lassen.

Um noch weitere Aussagen über das Verhalten in verschiedenen Umgebungen erhalten zu können wurden die folgenden Parameter variiert. Allen dieses Parametern ist gemeinsam, dass sie die Schwierigkeit der Systemumgebung beeinflussen:

Auslastung

Die Auslastung der Fabrik lässt sich über die Menge der eingesteuerten Lose regulieren. Je höher die Auslastung ist desto schwieriger wird die Systemumgebung. In den AreaFab2, QuarterFab und SemiFab Modellen wurden zwei Auslastungsszenarien analysiert: HOCH und MITTEL.

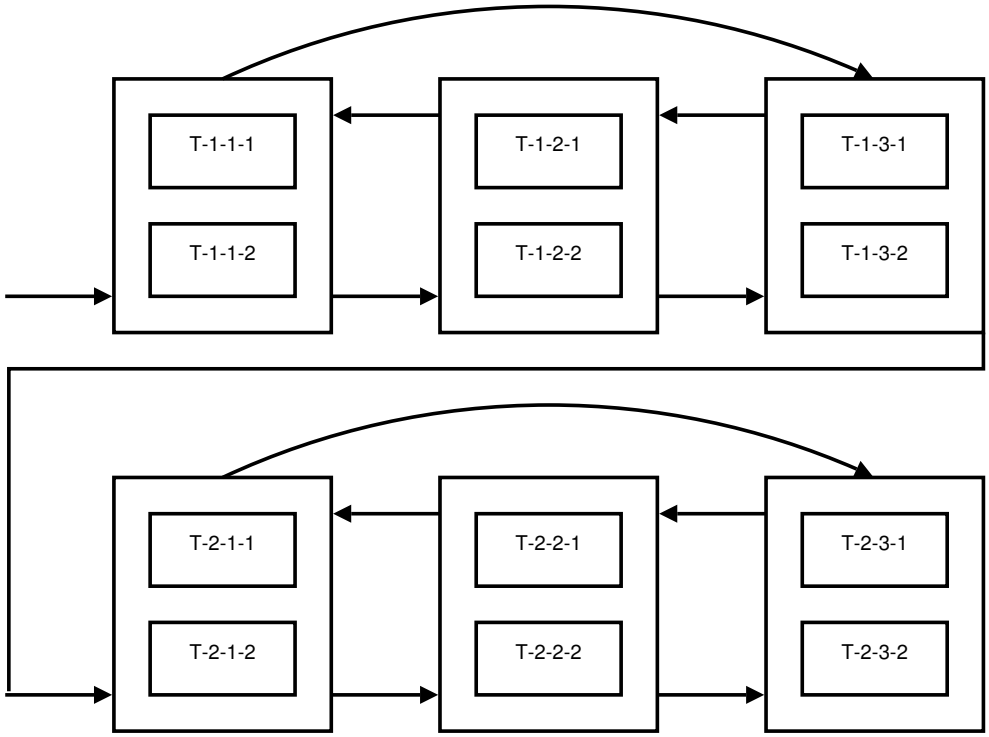

Abb. 5/2: Arbeitspläne im Modell AreaFab

Prioritätsverteilung

Die Prioritätsverteilung hat ebenfalls Einfluss auf den Schwierigkeitsgrad der Umgebung. Bei vielen höher priorisierten Losen ist die Umgebung schwerer einzustufen als bei wenigen priorisierten Losen. In den Modellen wurden AreaFab2, QuarterFab und SemiFab die zwei diskrete Verteilungen D1 und D2 getestet (Tabelle 3). Die Verteilung D1 repräsentiert also ein schwierigere Problemgröße als die Verteilung D2.

Verteilung	Wahrscheinlichkeit	Priorität
$D1$	$0,50$	1
	$0,35$	5
	$0,15$	10
$D2$	$0,50$	1
	$0,45$	2
	$0,05$	10

Tabelle 3: Verwendete Prioritätsverteilungen

Flussfaktor

Über den Flussfaktor werden die Fälligkeitsdaten der einzelnen Lose berechnet. Hierbei wird zum Verfügbarkeitsdatum die erwartete Durchlaufzeit hinzugefügt. Diese erwartete Durchlaufzeit eines Loses wird als Vielfaches (der Flussfaktor) der

Rohprozesszeit angegeben (Formel 19). Ein Flussfaktor von $2,0$ gibt also an, dass das Los zusätzlich zur reinen Rohprozesszeit dieselbe Zeit mit Warten im System verbringt. Je kleiner der Flussfaktor ist, um so schwieriger wird die Systemumgebung. Einen Flussfaktor kleiner $1,0$ kann es nicht geben. In den kleineren Modellen (SmallFab, AreaFab) wurden Flussfaktoren zwischen $1,0$ und $2,0$ getestet. In den AreaFab2, QuarterFab und SemiFab Modellen wurden nur die Flussfaktoren $1,6$ und $2,0$ untersucht.

$$FF_j = \frac{d_j - r_j}{p_j} \tag{19}$$

Auch die Größe des Modells hat Einfluss auf den Schwierigkeitsgrad der Umgebung. Je größer ein Modell ist, umso schwieriger ist das Problem. Dies zeigt sich vor allem dadurch, dass in den größeren Modellen die Unterschiede zwischen den einzelnen Verfahren deutlicher zu Tage treten.

Um eine leichtere Vergleichbarkeit bei geringeren Simulationsaufwand zu erhalten wurde der Zufall durch explizites Initialisieren des Zufallszahlengenerators ausgeschaltet.

Im Folgenden wird für die Beschreibung der Systemumgebung eine Kurzschreibweise in der Form Auslastung-Prioritätsverteilung-Flussfaktor eingeführt. So repräsentiert HOCH-D1-1,6 eine Umgebung mit hoher Auslastung, der Prioritätsverteilung D1 und einem Flussfaktor von 1,6.

5.2.2 Verwendete Einstellungen der Algorithmen

Der verteilte Algorithmus besitzt eine Reihe von Parameter mit denen er eingestellt werden kann. Im Folgenden sind die einzelnen untersuchten Parameter aufgeführt.

Aufrufintervall der oberen Ebene

Die Aufrufhäufigkeiten der oberen Ebene sind bei den größeren Modelle (AreaFab2, QuarterFab und SemiFab) $2h$, $4h$, $5h$, $6h$ und $8h$. Bei den anderen Modellen variieren die Aufrufzeiten zwischen $1h$ und $9h$. Allerdings erfolgt der Aufruf des DSBH Schedulers dort synchron mit der oberen Ebene.

Aufrufintervall der DSBH

Wie die Aufrufintervalle der oberen Ebene variieren die Intervalle der DSBH bei den Modellen AreaFab2, QuarterFab und SemiFab zwischen $2h$, $4h$, $5h$, $6h$ und $8h$. Bei den kleineren Modellen ist der Aufruf, wie bereits erwähnt, unmittelbar nach der oberen Ebene.

zusätzlicher Planungshorizont

Bei den großen Modellen wird noch ein zusätzlicher Planungshorizont (Forecast) von $0h$, $2h$ und $4h$ mit berücksichtigt.

Koordinationsmechanismus

Für die Wahl des Koordinationsmechanismus werden drei Ausprägungen variiert:

- Zunächst wird keine Koordinierung durchgeführt. Die Idee besteht darin, zu überprüfen inwieweit ein Einhalten der Vorgabezeiten ausreicht, um ein gutes Ergebnis zu erzielen.
- Als nächstes wird immer nur der kritischste Bereich mit dem angrenzenden Bereich koordiniert.
- Bei größeren Aufrufintervallen kann eine dynamische Koordination vorteilhaft sein. Hier wird nur die nicht rekursive Variante getestet.

Unterproblemlöser

Für die kleineren Modelle wurden verschiedene Unterproblemlöser getestet. Bei den größeren Modellen wurden für kritische Maschinengruppen die ATC-Regel verwendet. Bei den nicht kritischen Maschinengruppen wurde zwischen ATC und einer FIFO-Regel variiert.

Die Informationen, welche Maschinengruppen als kritisch angesehen werden, wurden extern vorgegeben. Hierbei wurden die Auslastungen der Maschinengruppen in Betracht gezogen.

In den meisten Experimenten wurde als Terminierungsalgorithmus, der im Kapitel 3.5 erwähnte Hotfactor Ansatz gewählt. An den Stellen bei denen ein anderer Ansatz verwendet wurde, wird explizit darauf hingewiesen.

In vorherigen Untersuchungen [Mönch und Rose 2003] hatte sich eine zweifache Reoptimierung für die globale SBH als günstig erwiesen. Aus diesem Grund wird in der verteilten SBH auch eine zweifache Reoptimierung verwendet.

Auch für die Algorithmusparameter wird eine Kurzschreibweise eingeführt. Die Form ist hierbei Heuristik-Aufrufintervall_1-Aufrufintervall_2-Unterproblemlöser-Forecast. So stellt DSBH-4h-2h-ATC-0h die Heuristik DSBH bei einem Aufrufintervall von 4 Stunden für die obere Ebene und einen Aufrufintervall von 2 Stunden für die untere Ebene dar. Der verwendete Unterproblemlöser ist ATC und der zusätzliche Forecast beträgt 0 Stunden. Bei Heuristiken, die die anderen Felder nicht benötigen, steht anstelle der Ausprägung ein NONE. So bezeichnet FIFO-NONE-NONE-NONE-NONE eine normale FIFO Regel. Um die Übersicht zu wahren wird diese Schreibweise zu FIFO abgekürzt.

Die Kurzschreibweisen der Umgebung und die Kurzschreibweisen der Algorithmuseinstellungen lassen sich kombinieren. So drückt HOCH-D1-1,6-FIFO einen kompletten Testdatensatz aus.

5.2.3 Untersuchte Zielgrößen

Das hauptsächliche Zielkriterium ist die gewichtete Verspätung aller Lose (Formel 20 – TWT). Diese setzt sich aus der Summe der gewichteten Verspätungen der einzelnen Lose (T_j) zusammen.

$$TWT := \sum_{j=0}^{n} w_j \cdot T_j \tag{20}$$

Neben diesem Kriterium, welches die Zielfunktion des Algorithmus bildet, existieren noch andere Kriterien, die untersucht werden. Ein weiteres Kriterium ist die durchschnittliche gewichtete Verspätung, eines einzelnen Jobs (n bezeichnet die Anzahl der fertiggestellten Lose):

$$T_{avg} := \frac{\sum_{j=0}^{n} w_j \cdot T_j}{n} \tag{21}$$

Der Durchsatz der Fabrik ist ein weiterer wichtiger Indikator. Hierbei wird zwischen dem normalen Durchsatz (der Anzahl der fertig gestellten Lose – TP) und dem gewichteten Durchsatz (die Summe der Gewichte der fertig gestellten Lose – WTP) unterschieden.

Außerdem ist die durchschnittliche Zykluszeit pro Produkt interessant. Auch hier wird wieder zwischen mit ($WCT(P)$) und ohne Gewichtung ($CT(P)$) unterschieden.

Weiterhin wird die reine Anzahl der frühen ($C_j < d_j$) und späten ($C_j > d_j$) Lose betrachtet. Auf eine Gewichtung wird bei diesen Kennzahlen allerdings verzichtet.

5.2.4 Verwendete Hardware und Anzahl der Experimente

Die Simulationen wurden größtenteils auf Rechnern durchgeführt, die 1,5 bis 2 GHz Prozessortakt und zwischen 256MB und 1GB RAM hatten.

Tabelle 4 fasst die verwendeten Parameter und ihre untersuchten Ausprägungen noch einmal zusammen. Zusätzlich zu diesen Ausprägungen wurden während der Entwicklung Untersuchungen zur Auswirkung der Flussfaktoren, der Aufrufintervalle und der Koordinationsmechanismen durchgeführt. Insgesamt wurden ungefähr 6000 Experimente mit den verschiedenen Modellen durchgeführt.

Faktor	Ausprägung
SSP-Mix	Engpass: ATC, Nichtengpass: FIFO, ATC
Horizonte	Aufrufe: 2,4,5,6,8, Forecast: 0, 2
Flussfaktor	1,6 und 2,0
Prioritätsverteilung	D1, D2
Systemlast	Mittel, hoch
Aufrufhäufigkeit der oberen Ebene	4, 8, synchron
Koordinationsmechanismus	Statisch, Kritisch, Dynamisch

Tabelle 4: Arbeitspläne (Maschinenfolgen)

Alle Modelle wurden 50 Tage simuliert. Um die Simulationen sinnvoll bewerten zu können ist eine entsprechende Einschwingphase notwendig. Diese kann nicht in die Untersuchung mit einbezogen werden. Um diese Einschwingzeit zu reduzieren war die Fabrik anfangs nicht leer, sondern es fanden WIP-Lose Verwendung. Weiterhin wurden nur neu eingesteuerte Lose berücksichtigt. Hierdurch war es möglich, die Einschwingphase auf fünf Tage zu reduzieren.

5.3 Analyse der Experimente

5.3.1 Ergebnisse während der Entwicklung

Die Untersuchungen der Modelle SmallFab, AreaFab und in Teilen AreaFab2 dienten der Überprüfung des Algorithmus. Hier wurden, wie weiter oben erwähnt, nur die Aufrufintervalle, die Flussfaktoren, die Unterproblemlöser und Koordinationsmechanismen (AreaFab und AreaFab2) variiert. Die Ergebnisse bieten einen guten Überblick über die Auswirkungen der einzelnen Parameter.

Die Auswirkung der Flussfaktoren lassen sich in Abbildung 5/3 sehr gut ablesen. Das betrachtete Modell ist das SmallFab Modell. Hier wurde nur der Flussfaktor variiert und alle anderen Parameter gleich gehalten. Es wird ersichtlich, wie die Qualität der Ergebnisse der einzelnen Verfahren von dem Flussfaktor abhängt.

Die Gründe für das relativ schlechte Abscheiden der SBH bei weit gesetzten Fälligkeitsterminen (großen Flussfaktoren) liegen darin, dass durch das Zielkriterium TWT eine Umsortierung der Lose stattfindet. Des Weiteren bleibt dem Algorithmus auch weniger Spielraum zur Optimierung. Ein multikriterielles Zielkritierium könnte hier Verbesserungen bringen.

Wenn beispielsweise kein Los im FIFO Fall Verspätung hat, so macht eine Optimierung im Bezug auf die Termintreue keinen Sinn mehr. An dieser Stelle würden andere Optimierungskriterien sinnvoller sein, wie der Durchsatz der Fabrik oder die Durchlaufzeit der einzelnen Lose.

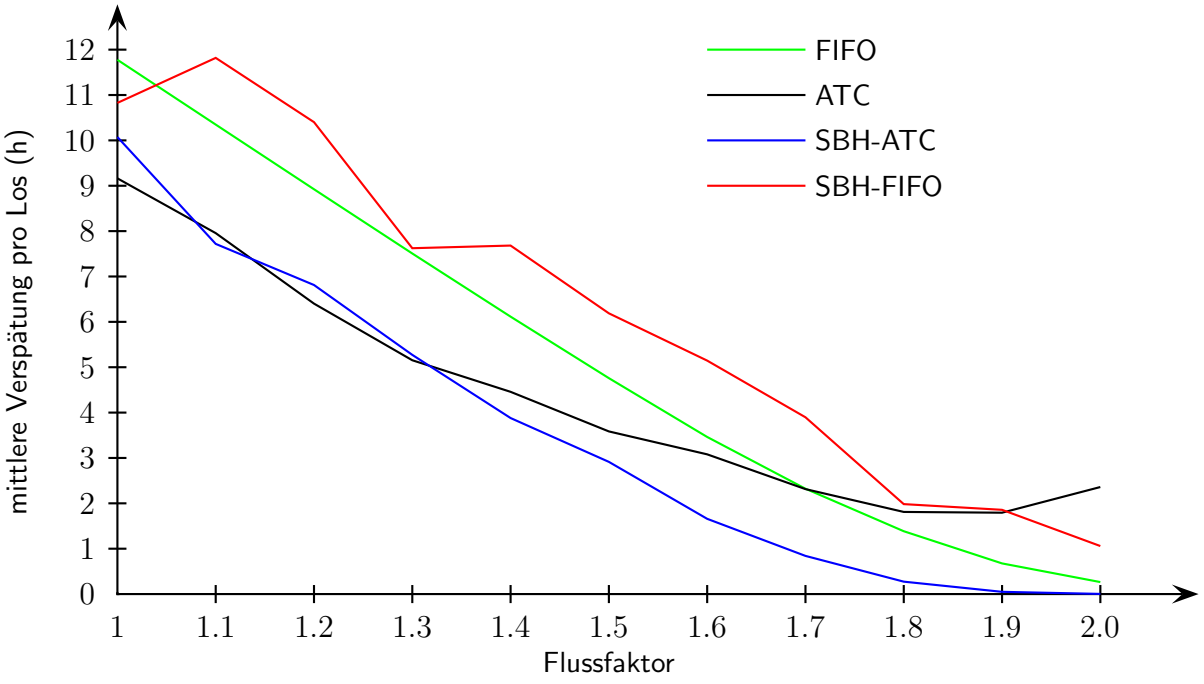

Abb. 5/3: Ergebnisse SmallFab

Ein anderes Beispiel (Abbildung 5/4) veranschaulicht die Bedeutung der Aufrufintervalle und der Koordinierung. Es wurde hier das AreaFab Modell mit zwei Bereichen betrachtet.

Die verteilte Heuristik liefert in einem Bereich von 5 bis 6 Stunden die besten Ergebnisse. Wohingegen die nicht verteilte Heuristik ihre besten Ergebnisse im Bereich von 2 bis 3 Stunden liefert. Dies deckt sich auch mit Untersuchungen aus [Mönch und Rose 2003].

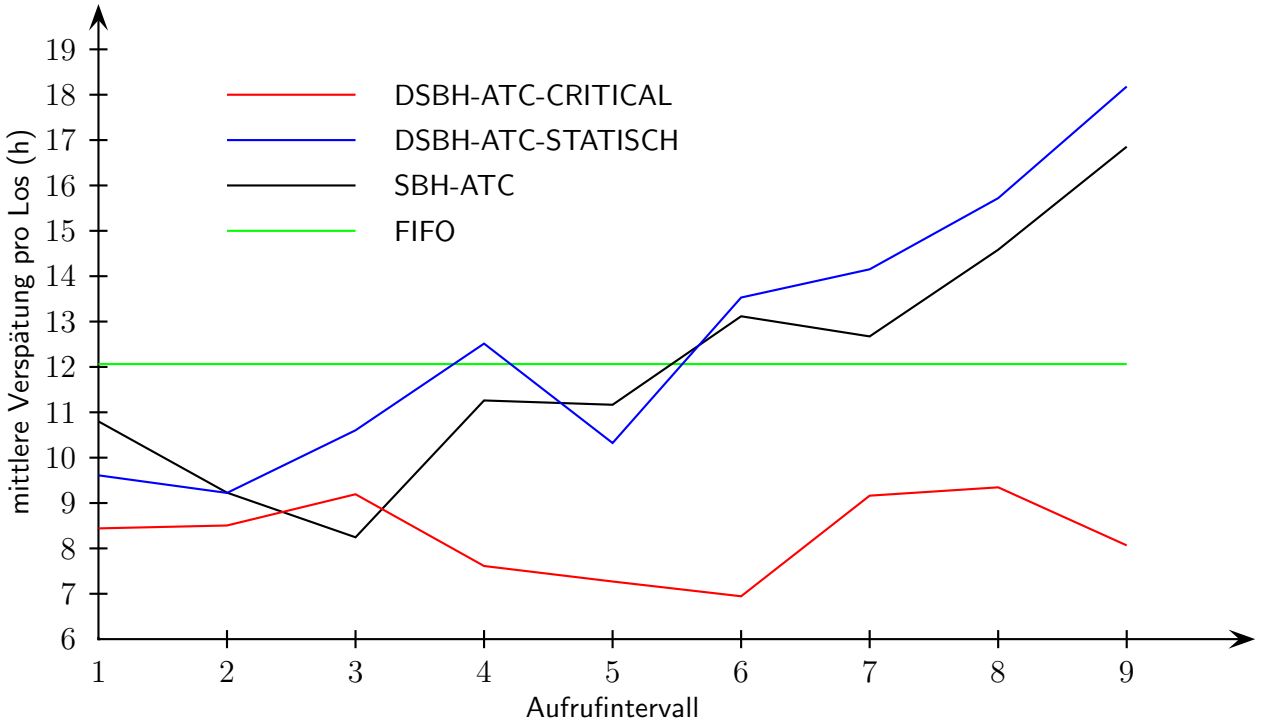

Abb. 5/4: Ergebnisse AreaFab

Hier sieht man außerdem die Bedeutung der Koordinierung. Obwohl oder gerade weil das Modell (AreaFab) nur zwei Bereiche besitzt, ist die Auswirkung der Koordination besonders bei großen Aufrufintervallen sehr groß.

Ein weiterer Aspekt ist die Wahl der Unterproblemlöser für die SBH. Bei der normalen SBH ist die Wahl eines guten Unterproblemlösers von entscheidender Bedeutung für die Qualität der Ergebnisse (vgl. Abbildung 5/3). Auch wenn bei der Verteilung der Heuristik, die Koordination die Auswirkungen eines schlechteren Unterproblemlösers mildert, so spielt er zumindest für die Enpassmaschinen eine wichtige Rolle.

5.3.2 Ergebnisse des Modells AreaFab2

Das Modell AreaFab2 ist zwar immer noch ein akademisches Modell, jedoch ist es größer und damit realistischer als die im vorigen Abschnitt behandelten Modelle. Getestet wurden bei diesem Modell eine normale FIFO-Regel, die normale SBH mit ATC-Unterproblemlöser sowie die verteilte SBH mit verschiedenen Parametern. Wie bereits weiter oben angedeutet lassen sich die Umgebungen grob anhand ihres Schwierigkeitsgrades unterteilen. Abbildung 5/5 zeigt die Performance der einzelnen Verfahren in Bezug auf den Schwierigkeitsgrad der Umgebungen. Das untersuchte Leistungsmaß war die mittlere gewichtete Verspätung. Diesem Diagramm kann man mehrere Schlussfolgerungen entnehmen:

- In „schwierigen“ Umgebungen erbringen die SBH Varianten deutlich bessere Leistungen als entsprechende Dispatchregeln.

- In „weniger schwierigen“ Umgebungen scheinen die SBH Varianten prinzipielle Probleme zu haben. Als besonders entscheidend ist hierbei die Abhängigkeit zu den Flussfaktoren zu sehen. Dies wurde bereits in [Mönch und Rose 2003] herausgefunden.

  Der Grund hierfür scheint darin zu liegen, dass die Unterproblemlöser der SBH verhältnismäßig schlechte Ergebnisse liefern. Alle Prioritätsregeln, die sich am Fälligkeitsdatum orientieren nehmen eine entsprechende Umsortierung der Warteschlangen vor. Dies liefert allerdings nicht in allen Fällen gute Ergebnisse. So hat auch [Rose 2003] in entsprechenden Untersuchungen zu CR-Regeln auf diesen Fakt hingewiesen.

- Die verteilte Heuristik scheint gegenüber der normalen SBH tendenziell bessere Ergebnisse zu liefern.

  Der Grund hierfür liegt wahrscheinlich darin begründet, dass durch die Aufsplittung der Losrouten bessere Fälligkeitsdaten ermittelt werden können. Diese Interpretation geht mit den obigen Betrachtungen zum Einfluss der Flussfaktoren einher.

- Die Verwendung einer Koordinierung zwischen den Bereichen scheint ebenfalls bessere Ergebnisse erzielen zu können.

  Auch hier lässt sich argumentieren, dass durch eine bessere Koordinierung bessere Eckdaten für die einzelnen Losabschnitte ermittelt werden können. Es ist allerdings auch zu sehen, dass eine Koordinierung nicht immer bessere Ergebnisse erzielt. Es hängt vom Zusammenspiel aller Parameter ab.

Tabelle 5 stellt die einige Ergebnisse der DSBH für die Umgebung HOCH-D1-1,6 den Ergebnissen der SBH und der FIFO-Regel gegenüber. Neben den vom Optimierungskriterium direkt beeinflussten Wert der durchschnittlichen gewichteten Verspätung sind die anderen drei Kriterien interessant. Der reine Durchsatz der FIFO Regel ist zwar am höchsten, allerdings zeigt eine Betrachtung des gewichteten Durchsatzes, dass bei der DSBH die höher priorisierten Lose eher fertig werden. Interessant ist auch, dass mit einer Verbesserung der Verspätung nicht unbedingt eine Minimierung der mittleren gewichteten Zykluszeit einhergeht.

Die Tabelle lässt auch erkennen, dass die Verwendung eines FIFO Unterprblemlösers nicht wesentlich schlechtere Ergebnisse liefert als ein höherwertiger Unterproblemlöser. Dies ist eine Möglichkeit die Laufzeiten der Heuristik zu reduzieren.

In Tabelle 6 finden sich einige Ergebnisse für die Umgebung MITTEL-D1-1,6. Hier zeigt sich wieder, dass die Ergebnisse der verschiedene SBH Varianten gegenüber von FIFO nachlassen. Jedoch liegen auch hier die „besten“ Aufrufintervalle bei vier bis fünf Stunden.

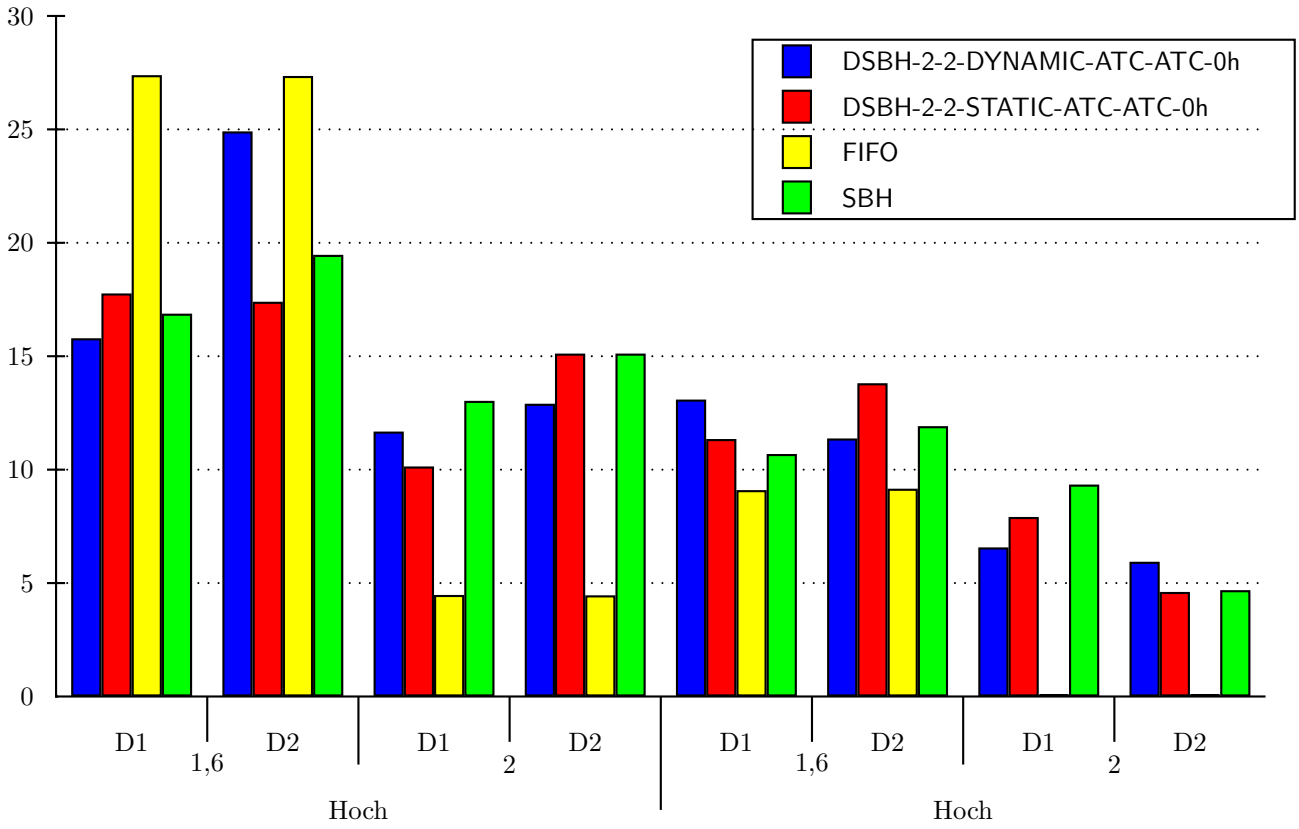

Abb. 5/5: Performance in Abhängigkeit vom Schwierigkeitsgrad der Umgebung

5.3.3 Ergebnisse des Modells QuarterFab

Die Ergebnisse des Modells QuarterFab bestätigen die Ergebnisse aus den vorherigen „akademischen“ Modellen. Die Tendenz, dass die DSBH in schwierigeren Umgebungen bessere Ergebnisse liefert bestätigen sich auch in diesem Modell. Zwar liegen die Ergebnisse der DSBH und der SBH in Abbildung 5/6 relativ dicht beieinander, jedoch erreicht die DSBH ihre Ergebnisse mit einem mehr als doppelt so großem Aufrufintervall (fünf Stunden gegenüber zwei Stunden). Auch in diesem Beispiel wird die große Bedeutung der Flussfaktoren und der Auslastung deutlich. Die Auswirkungen der Prioritätsverteilungen sind dagegen nicht so gravierend. Tabelle 7 zeigt einige ausgewählte Ergebnisse für die Umgebung HOCH-D1-1,6. Auf Basis dieser Daten lassen sich einige Aussagen für die Parametereinstellungen der DSBH ableiten:

- Ein zusätzlicher Planungshorizont ist, zumindest bei den verwendeten SBH Varianten, nicht nötig. Es hat sich vielmehr herausgestellt, dass ein zu großer zusätzlicher Planungshorizont sehr schlechte Ergebnisse liefert
- Die Heuristik läuft optimal, wenn die Aufrufintervalle synchron geschaltet sind und das Intervall bei vier bis fünf Stunden liegt.
- Der Qualitätsverlust durch das Verwenden einer einfachen FIFO-Regel bei nicht

Algorithmus	Durchschnittliche gewichtete Verspätung	Durchschnittliche gewichtete Zykluszeit	Durchsatz (Fertiggestellte Lose)	Gewichteter Durchsatz
DSBH-4h-4h-STATIC-ATC-ATC-2h	12,6907	103,87	630	2544
DSBH-2h-2h-STATIC-ATC-FIFO-0h	12,9753	101,801	601	2485
DSBH-6h-6h-STATIC-ATC-FIFO-2h	16,5424	108,485	637	2519
DSBH-2h-2h-CRITICAL-ATC-FIFO-2h	13,1187	101,551	601	2516
DSBH-8h-8h-DYNAMIC-ATC-ATC-0h	33,2209	125,274	616	2499
SBH	16,8769	106,74	585	2469
FIFO	27,3907	119,444	647	2481

Tabelle 5: Ergebnisse des Models AreaFab2 für die Umgebung HOCH-D1-1,6

kritischen Maschinen ist vertretbar. Gleichzeitig wird die Geschwindigkeit des Algorithmus dadurch erhöht.

- Eine Koordinierung ist in den Fällen nicht nötig, bei denen die Aufrufintervalle synchron geschaltet werden. Ist das Aufrufintervall der Terminierungsebene größer als das Aufrufintervall des Feinschedulers, dann erbringt eine Koordinierung einen zusätzlichen Qualitätsgewinn.

Um die Auswirkung des oberen Terminierungsalgorithmus auf die Scheduling-Qualität zu untersuchen, wurden einige Experimente mit einem Terminierungsalgorithmus durchgeführt, der auf einem gleitendenden Flussfaktor beruht. Dieser wurde in Kapitel 3.5 näher beschrieben. Tabelle 8 stellt einige dieser Ergebnisse für die Umgebung HOCH-D2-2 dar. Es zeigt sich, dass dieser Ansatz in weniger schwierigen Umgebungen bessere Ergebnisse liefern kann als der Hotfactor Ansatz. In schwierigen Umgebungen sind die Ergebnisse allerdings deutlich schlechter, wie Tabelle 9 zeigt. Dies liegt darin begründet, dass der Ansatz des gleitenden Flussfaktors ein adaptiver Ansatz ist. Im Gegensatz zum Hotfactor Ansatz berücksichtigt er in gewisser Weise die Kapazitätsauslastung der einzelnen Maschinen. Dies sorgt nun dafür, dass den einzelnen Arbeitsgängen zu viel Freiraum gegeben wird. Dies bedingt wiederum ein schlechteres Ergebnis der DSBH. So schaukeln sich beide Algorithmen gegenseitig auf.

Algorithmus	Durchschnittliche gewichtete Verspätung	Durchschnittliche gewichtete Zykluszeit	Durchsatz (Fertiggestellte Lose)	Gewichteter Durchsatz
DSBH-2h-2h-CRITICAL-ATC-FIFO-2h	10,489	99,2147	627	2467
DSBH-5h-5h-DYNAMIC-ATC-ATC-2h	9,98415	101,794	631	2440
DSBH-4h-4h-STATIC-ATC-ATC-2h	9,51878	100,603	625	2437
DSBH-8h-8h-DYNAMIC-ATC-ATC-0h	31,1738	123,227	617	2410
SBH	10,6905	99,6434	619	2446
FIFO	9,09828	100,713	642	2450

Tabelle 6: Ergebnisse des Models AreaFab2 für die Umgebung MITTEL-D1-1,6

5.3.4 Ergebnisse des Modells SemiFab

Die Ergebnisse, die das große Modell lieferte waren den vorherigen Ergebnissen vergleichbar. In Tabelle 10 werden einige Ergebnisse für die Umgebung HOCH-D1-1,6 aufgeführt. Die DSBH lieferte im Vergleich zu der SBH bessere Ergebnisse. Gleichzeitig sind jedoch größere Aufrufintervalle beziehungsweise kürzere Laufzeiten möglich.

Die Tabellen 11, 12 und 13 stellen einige Ergebnisse aus den Simulationen für alle Modelle dar. Die betrachtete Umgebung war HOCH-D1-1.6. Interessant ist in diesem Fall die durchschnittliche Zeit, welche pro Scheduleraufruf benötigt wird. Die Zeiten liegen alle unterhalb einer Minute. Im Vergleich mit der normalen SBH erreicht man einen fünffachen Geschwindigkeitszuwachs. Des Weiteren ist erkennbar, dass die DSBH auch besser skaliert als die SBH. Diese Effekte resultieren hauptsächlich aus den kleineren Bereichsgraphen. Weiterhin ist zu erwähnen, dass sich diese Ergebnisse auf die rein fachliche Verteilung der DSBH beziehen. Das heißt alle Bereiche wurden nacheinander auf der selben Maschine gerechnet. Im Falle einer physischen Verteilung wie im FABMAS System dürfte die Laufzeit, wenn auf Koordinierung verzichtet wird, um ein fünftel kleiner sein.

5.3.5 Einfluss der einzelnen Faktoren

In den vorherigen Abschnitten wurde bereits an gegebener Stelle auf den Einfluss der einzelnen Umgebungs- und Algorithmusparameter eingegangen. An dieser Stelle sollen die Erkenntnisse noch einmal zusammengefasst werden, sowie entsprechende Verbesserungsvorschläge für die Heuristik formuliert werden.

Verfahren	Durchschnittliche gewichtete Verspätung	Verspätete Lose	Verfrühte Lose
DSBH-5h-5h-STATIC-ATC-ATC-0h	3,2966	397	397
DSBH-8h-8h-DYNAMIC-ATC-ATC-4h	22,333	722	28
DSBH-8h-2h-STATIC-ATC-ATC-0h	4,13	297	499
DSBH-8h-2h-DYNAMIC-ATC-ATC-0h	3,59	298	492
DSBH-5h-5h-STATIC-ATC-FIFO_DISPATCH-0h	3,85191	425	370
FIFO	9,79858	358	425
SBH	3,59124	373	418

Tabelle 7: Ergebnisse QuarterFab für die Umgebung HOCH-D1-1,6

5.3.5.1 Einfluss des Flussfaktors

Bei eng gesetzten Fälligkeitsdaten ist die Umgebung als schwieriger einzustufen als bei weiten. Wie bei der globalen SBH liefert die DSBH bei eng gesetzten Fälligkeitsdaten bessere Ergebnisse als Prioritätsregeln. Bei weit gesetzten Fälligkeitsdaten liefert FIFO allerdings bessere Ergebnisse. Der Grund hierfür scheint der verwendete ATC-Unterproblemlöser zu sein. Dieser benötigt ein gewisses Optimierungspotential um gute Ergebnisse zu bringen. Bei großen Flussfaktoren ist dieses nicht gegeben. Die Umsortierung durch die ATC sorgt hierbei für schlechtere Ergebnisse.

5.3.5.2 Einfluss der Auslastung

Neben dem Flussfaktor ist der Auslastungsgrad der zweitwichtigste Umgebungsparameter für die Heuristik. Bei einer hohen Auslastung ist die Umgebung als schwieriger einzustufen als bei einer geringeren.

Die Beobachtungen beim Auslastungsgrad gehen mit den Flussfaktorbetrachtungen einher. Je höher die Auslastung ist umso bessere Ergebnisse liefert die DSBH. Bei geringeren Auslastungen liegen die Ergebnisse der SBH und der DSBH näher beieinander, wenngleich die DSBH auch hier tendenziell bessere Ergebnisse liefert.

Wird die Auslastung zu gering, dann liefert die SBH Algorithmen schlechtere Ergebnisse

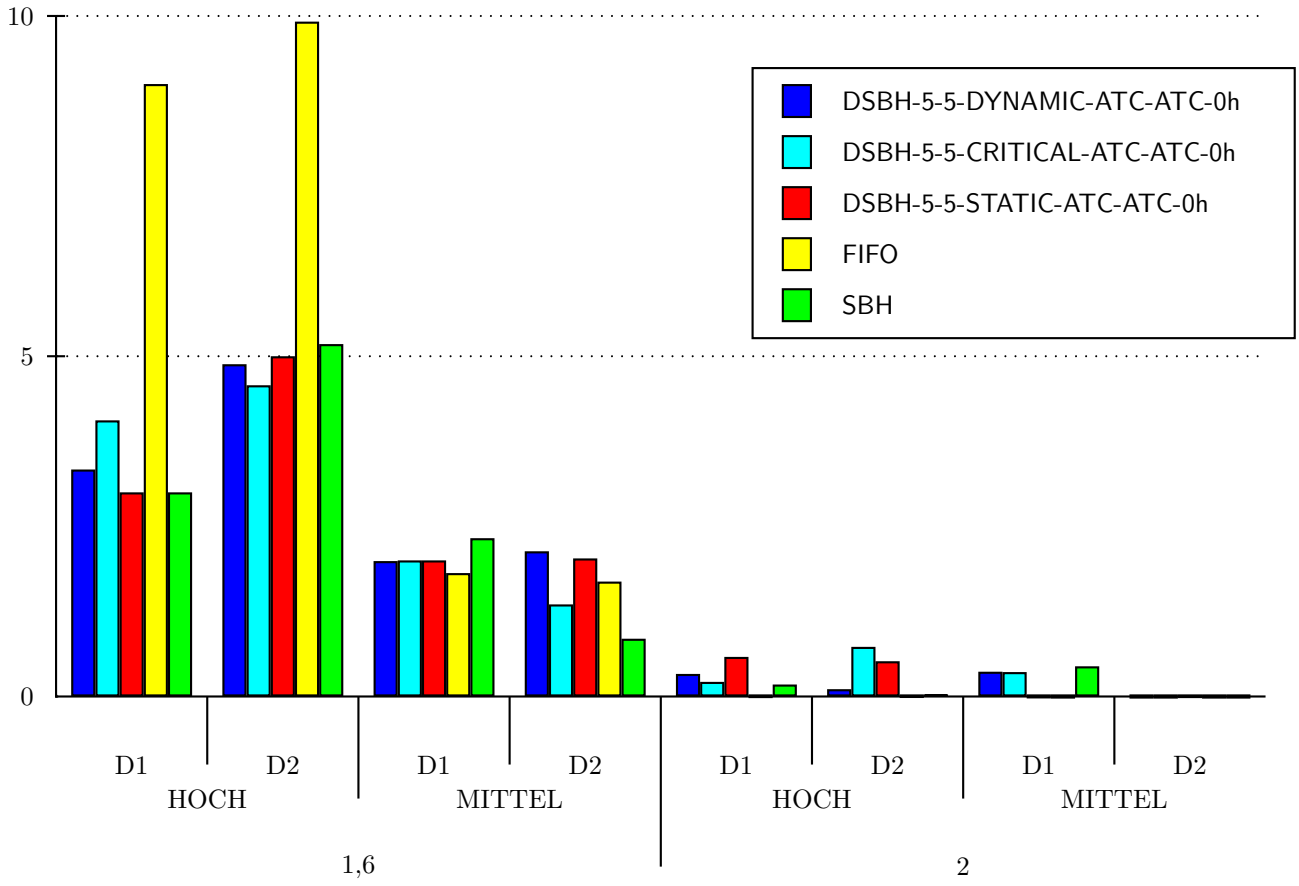

Abb. 5/6: Ergebnisse QuarterFab

als eine einfache FIFO-Regel.

5.3.5.3 Einfluss der Prioritätenverteilung

Die Prioritätsverteilung hat ebenfalls Einfluss auf die Schwierigkeit der Umgebung. Bei vielen höher priorisierten Losen ist die Umgebung schwieriger als bei einer gleichförmigeren Verteilung. Im Unterschied zu den anderen Kriterien beeinflusst die Prioritätsverteilung die Ergebnisse am wenigsten.

Je mehr höher priorisierte Lose im System (Verteilung D1) sind, umso besser ist die DSBH. Je gleichförmiger die Verteilung ist umso näher liegen die Ergebnisse der einzelnen Verfahren.

5.3.5.4 Einfluss der Modellgröße

Je größer die Modelle werden umso größer werden die Unterschiede, die durch die einzelnen Verfahren erzielt werden können. Auch dies geht mit der Klassifikation nach schwierigen und nichtschwierigen Umgebungen konform. Bei einem größeren Modell sind die

Terminierung	Verfahren	absolute gewichtete Verspätung	Verspätete Lose	Verfrühte Lose
Gl. Flussfaktor	DSBH-2h-2h-STATIC-ATC-ATC-0h	0	0	801
Gl. Flussfaktor	DSBH-4h-4h-STATIC-ATC-ATC-0h	0	0	790
Gl. Flussfaktor	DSBH-5h-5h-STATIC-ATC-ATC-0h	116,488	13	771
Gl. Flussfaktor	DSBH-2h-2h-CRITICAL-ATC-ATC-0h	0	0	803
Gl. Flussfaktor	DSBH-4h-4h-CRITICAL-ATC-ATC-0h	0	0	794
Gl. Flussfaktor	DSBH-5h-5h-CRITICAL-ATC-ATC-0h	0	0	787
Hotfactor	DSBH-5h-5h-STATIC-ATC-FIFO-0h	46,86	10	781
Hotfactor	SBH	48,3081	6	783
	FIFO	12,9319	5	778

Tabelle 8: Einfluss des Terminierungsalorithmus für die Umgebung HOCH-D2-2,0 (Model QuarterFab)

Abhängigkeiten zwischen den einzelnen Elementen komplexer, wodurch höherwertige Verfahren, die diese Abhängigkeiten berücksichtigen bessere Ergebnisse liefern können.

5.3.5.5 Einfluss der Aufrufintervalle

Die DSBH scheint bei einem Aufrufintervall von vier bis fünf Stunden die besten Ergebnisse zu liefern. Bei dem verwendeten Terminierungsalgorithmus wurden die besten Ergebnisse erreicht, wenn er synchron mit der DSBH aufgerufen wurde. In diesen Fällen wurde auch meistens eine Koordinierung nicht benötigt. Wenn die Aufrufintervalle des Terminierungsalgorithmus größer waren als die der DSBH brachte eine Koordinierung eine Verbesserung.

Es ist zu hoffen, dass man mit einem besseren Planungsalgorithmus auch eine Vergrößerung der Aufrufintervalle erzielen kann.

Terminierung	Verfahren	Durchschnittliche gewichtete Verspätung	Verspätete Lose	Verfrühte Lose
Gl. Flussfaktor	DSBH-2h-2h-STATIC-ATC-ATC-0h	6,38	452	327
Gl. Flussfaktor	DSBH-4h-4h-STATIC-ATC-ATC-0h	14,70	579	189
Gl. Flussfaktor	DSBH-2h-2h-CRITICAL-ATC-ATC-0h	10,65	506	268
Gl. Flussfaktor	DSBH-4h-4h-CRITICAL-ATC-ATC-0h	35,26	718	37
Hotfactor	DSBH-8h-4h-STATIC-ATC-ATC-0h	4,14	333	463
Hotfactor	DSBH-4h-4h-CRITICAL-ATC-ATC-0h	4,51	384	407
Hotfactor	SBH	5,17	353	436
	FIFO	9,91	358	425

Tabelle 9: Einfluss des Terminierungsalorithmus für die Umgebung HOCH-D2-1,6 (Model QuarterFab)

5.3.5.6 Einfluss der verwendeten Unterproblemlöser

Es ist so, dass „bessere“ Unterproblemlöser auch bessere Ergebnisse liefern. Es hat sich allerdings gezeigt, dass durch die Verwendung einer FIFO-Regel für nichtkritische Maschinen keine gravierende Verschlechterung bringt. Im Gegenzug sorgt die Verwendung einer FIFO-Regel allerdings für eine Beschleunigung des Algorithmus.

5.3.5.7 Einfluss der Koordinierung

Wie oben erwähnt wurden bei den größeren Modellen die besten Ergebnisse erreicht, wenn der Terminierungsalgorithmus synchron mit der DSBH aufgerufen wurde. In diesen Fällen wurde auch meistens eine Koordinierung nicht benötigt. Wenn die Aufrufintervalle des Terminierungsalgorithmus größer waren als die der DSBH, brachte eine Koordinierung eine Verbesserung. Weiterhin scheint die Koordinierung in nicht schwierigen Umgebungen nötig zu sein. Bei schwierigen Umgebungen hat der obere Planungsalgorithmus hier weniger Spielraum und kann demzufolge weniger Fehler machen. Die Vorgabedaten sind dadurch tendenziell besser.

Verfahren	Durchschnittliche Verspätung pro Los	Maximale Verspätung	Maximale Verfrühung
DSBH-2h-2h-STATIC-ATC-ATC-0h	6,80	181,93	78,37
DSBH-2h-2h-CRITICAL-ATC-ATC-0h	7,12	142,72	67,50
DSBH-2h-2h-DYNAMIC-ATC-ATC-0h	6,19	157,08	65,19
DSBH-4h-4h-STATIC-ATC-FIFO-0h	9,64	156,42	59,21
SBH	7,78	158,21	73,26
TSZ-6h-6h	19,97	42,57	0
FIFO	47,61	116,88	0

Tabelle 10: Ausgewählte Ergebnisse des Modells SemiFab

Wie in Abbildung 5/4 zu sehen ist, sorgt die Koordination unter Umständen dafür, dass die Aufrufintervalle größer gewählt werden können.

5.3.5.8 Einfluss des zusätzlichen Planungshorizontes

In den Ergebnissen wurde deutlich, das ein zu großer Planungshorizont die Ergebnisse verschlechtert. Dies liegt vermutlich daran, dass bei zu großen Planungshorizonten zu sehr auf Lose gewartet wird, welche eine hohe Priorität haben.

Bei einem kleinen zusätzlichen Planungshorizont ergaben sich die besten Ergebnisse.

5.3.6 Zusammenfassung

In vielen Fällen liefert die DSBH bessere Ergebnisse als die SBH. Der Grund hierfür ist in der zeitlichen Dekomposition der Losrouten zu suchen. Die Kombination aus zeitlicher Dekomposition und entitätenbasierter Dekomposition scheint hier tendenziell besser Ergebnisse bei kleineren Laufzeiten zu liefern. Durch den oberen Terminierungsalgorithmus erfolgt eine besseres Bestimmung der Fälligkeitsdaten für die einzelnen Losabschnitte.

In weniger schwierigen Umgebungen lässt die Ergebnisqualität der SBH und DSBH allerdings nach und andere Verfahren liefern bessere Ergebnisse.

Weiterhin hat sich gezeigt, dass für Maschinen, die keinen Enpass darstellen, die Verwendung von FIFO Unterproblemlösern häufig ausreichend ist. Dies führt zu einer erheblich schnelleren Heuristik, da auf aufwändige Unterproblemlöser verzichtet werden kann.

Verfahren	Durchschnittliche Zeit pro Scheduleraufruf (s)	Durchschnittliche Verspätung pro Los	Maximale Verspätung	Maximale Verfrühung
DSBH-2h-2h-STATIC-ATC-ATC-0h	1,39	17,99	412,28	18,58
DSBH-2h-2h-CRITICAL-ATC-ATC-0h	1,74	16,44	421,63	18,50
DSBH-2h-2h-DYNAMIC-ATC-ATC-0h	2,16	16,71	471,56	18,70
SBH	4,74	17.18	472.33	18.92

Tabelle 11: Untersuchung des Zeitverhaltens für Model AreaFab2

Verfahren	Durchschnittliche Zeit pro Scheduleraufruf (s)	Durchschnittliche Verspätung pro Los	Maximale Verspätung	Maximale Verfrühung
DSBH-2h-2h-STATIC-ATC-ATC-0h	6,99	3.78	111.35	58.51
DSBH-2h-2h-CRITICAL-ATC-ATC-0h	7,21	4.46	157.37	57.05
DSBH-2h-2h-DYNAMIC-ATC-ATC-0h	6,81	3,58	122,07	52,82
SBH	15,97	4,17	118,18	52,38

Tabelle 12: Untersuchung des Zeitverhaltens für Model QuarterFab

Verfahren	Durchschnittliche Zeit pro Scheduleraufruf (s)	Durchschnittliche Verspätung pro Los	Maximale Verspätung	Maximale Verfrühung
DSBH-2h-2h-STATIC-ATC-ATC-0h	8,62	6,80	181,93	78,37
DSBH-2h-2h-CRITICAL-ATC-ATC-0h	9,56	7,12	142,72	67,50
DSBH-2h-2h-DYNAMIC-ATC-ATC-0h	10,75	6,19	157,08	65,19
SBH	51,98	7,78	158,21	73,26

Tabelle 13: Untersuchung des Zeitverhaltens für Model SemiFab

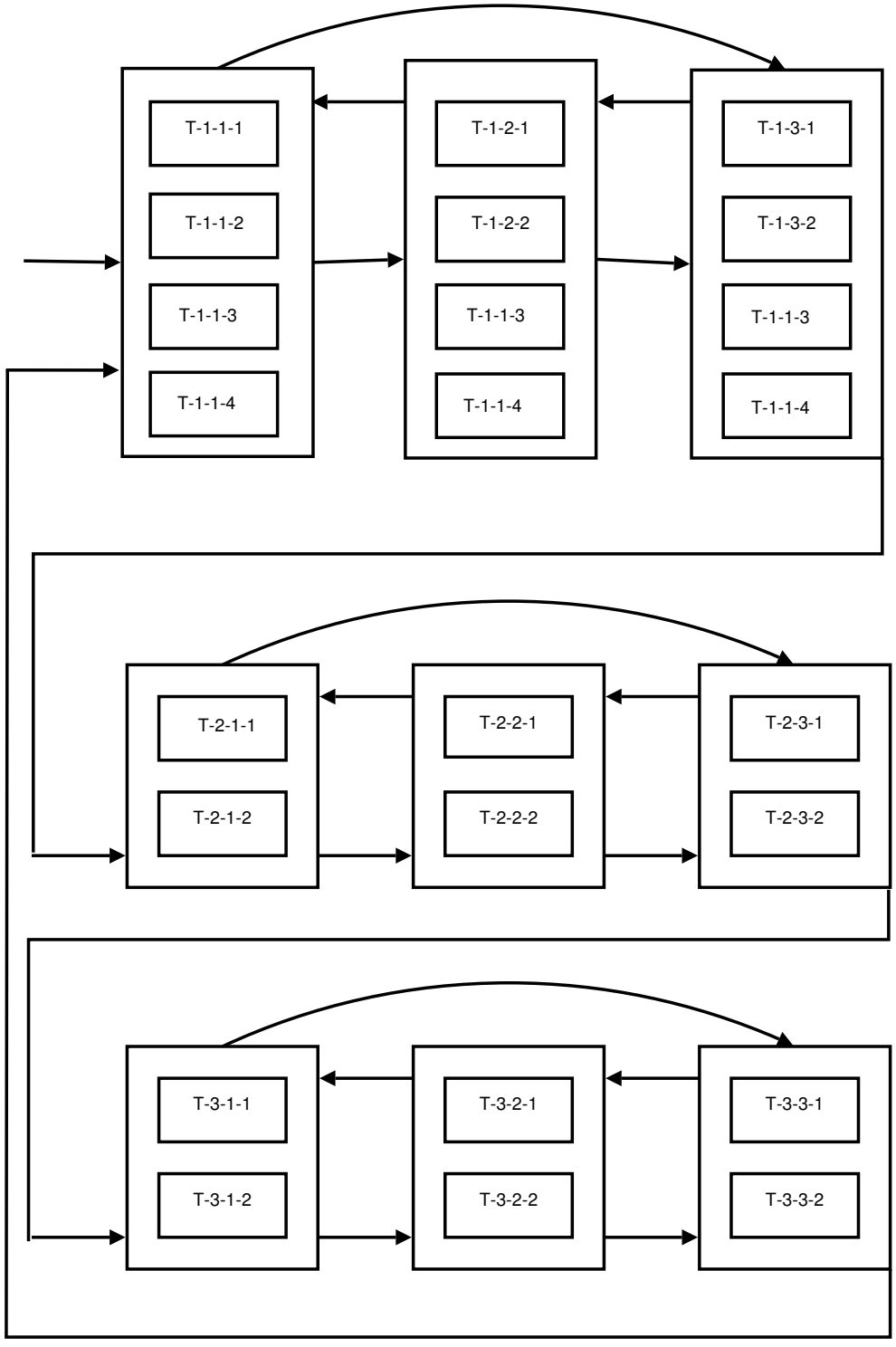

Abb. 5/7: Arbeitspläne im Modell AreaFab2

6 Schlussbemerkung

6.1 Zusammenfassung

Die entworfene und implementierte Verteilung der Shifting-Bottleneck-Heuristik zeigte eine bessere Performance als die ursprüngliche Heuristik. Die Gründe sind hauptsächlich durch die besseren Vorgabedaten der oberen Planungsebene und der Verschränkung von zeitlicher und entitätenbasierter Dekomposition zu erklären.

Die Heuristik wurde ebenfalls innerhalb des Agentensystems FABMAS als Dienstagent implementiert. Der Entwurf und die Umsetzung eines vollständigen Kommunikationsszenarios steht noch aus.

6.2 Ausblick auf weitere Forschungsaufgaben

Neben der weiteren Integration in das FABMAS System existieren noch verschiedene Aspekte, welche zu einer Erhöhung der Leistungsfähigkeit der Heuristik beitragen können. Einige Ideen sollen an dieser Stelle kurz zusammengetragen werden:

- Die Verwendung eines besseren Planungsalgorithmus (Grobschedulers) sollte die Ergebnisqualität steigern. Die Terminierung erfolgte in den bisherigen Fällen nur mit Hilfe eines Algorithmus, der die Kapazitäten der Maschinengruppen unberücksichtigt lässt. Wenn innerhalb der Terminierung Kapazitäten berücksichtigt werden, ist eine weitere Leistungssteigerung in Form von größeren Aufrufintervallen und besseren Ergebnissen zu erwarten.

- Wenn Zyklen zwischen den Bereichen auftreten ist der jetzige Koordinierungsmechanismus suboptimal. Eine Verbesserung kann erzielt werden, wenn mit Hilfe des obere Terminierungsalgorithmus die Verfügbarkeits und Fälligkeitsdaten erneut ermittelt werden.

  Dies bedingt jedoch eine entsprechende Designerweiterung der oberen Planungsebene.

- Wie bei der „normalen“ Shifting-Bottleneck-Heuristik ist die Wahl der „richtigen“ Unterproblemlöser ein weiteres Forschungsfeld von dem entsprechende Leitungssteigerungen zu erwarten sind. Im Folgenden sind einige Ideen zusammengetragen.

  - Die momentan verwendete ATC Regel erlangt ihre Leistung aus der Iteration über mehrere K Parameter. Da diese Parameter Umgebungsabhängig sind gilt es zu überlegen ob mit Hilfe von maschinellen Lernverfahren diese K Parameter besser gesetzt werden können. Dies würde zu einem leistungsfähigeren und schnelleren Unterproblemlöser führen.
  - In anderen Untersuchungen haben sich Verbesserungen durch die Verwendung eines genetischen Algorithmus gezeigt. Diese Arbeiten bezogen sich auf die ursprüngliche globale Shifting-Bottleneck-Heuristik. Es ist zu überprüfen ob dies ebenfalls zu Verbesserungen in der DSBH führen kann.

- In dieser Arbeit wurde gezeigt, dass unterschiedliche Unterproblemlöser für Engpass- und Nichtengpassmaschinen zu Geschwindigkeitssteigerungen bei vertretbaren Ergebnisqualitätseinbußen, führen. Diese Informationen wurden allerdings extern vorgegeben. Wünschenswert wäre eine automatische Bestimmung. Denkbar ist auch die Verwendung einer feineren Differenzierung.

- Des Weiteren wäre eine Untersuchung interessant, in wie weit eine Variation der Zielfunktionen bessere Ergebnisse in den weniger schwierigen Umgebungen liefert. Wenn beispielsweise kein Los im FIFO Fall Verspätung hat, so macht eine Optimierung im Bezug auf die Termintreue keinen Sinn mehr. An dieser Stelle würden andere Optimierungskriterien sinnvoller sein, wie der Durchsatz der Fabrik oder die Durchlaufzeit der einzelnen Lose.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[Adams u. a. 1988] Adams, J. ; Balas, E. ; Zawack, D.: The shifting bottleneck procedure for job shop scheduling. In: *Managment Science* 34 (1988), S. 391–401

[Adl u. a. 1996] Adl, M. K. E. ; Rodriguez, Armando A. ; Tsakalis, Konstantinos S.: Hierarchical Modeling and Control of Re-entrant Semiconductor Manufacturing Facilities. In: *Proceedings of the 35th Conference on Decision and Control, Kobe, Japan*, 1996

[AYTUG u. a. 2002] AYTUG, H. ; KEMPF, K. ; UZSOY, R.: Measures of subproblem criticality in decomposition algorithms for shop scheduling. In: *International Journal of Production Research* 41 (2002), S. 865–882

[Brooks Automation 2001a] Brooks Automation, Inc.: *AutoSched AP Customization Guide v7.0.* Utah : Brooks Automation, Inc., 2001a

[Brooks Automation 2001b] Brooks Automation, Inc.: *AutoSched AP User's Guide v7.0.* Utah : Brooks Automation, Inc., 2001b

[GNU] *GNU Standard C++ Library.* http://gcc.gnu.org/onlinedocs/libstdc++/documentation.html. – Abgerufen am 24.02.2004

[Graham u. a. 1979] Graham, R. L. ; Lawler, E. L. ; Lenstra, J. K. ; Kann, A. H. G. R.: Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. In: *In: Annals of Discrete Mathematics* (1979), S. 343–362

[JK u. a. 1977] JK, Lenstra ; AHG, Rinnooy K. ; P., Brucker: Complexity of machine scheduling problems. In: *Annals of Discrete Mathematics*, 1977, S. 1:343–362

[Kirn 2001] Kirn, Prof. Dr. Stefan: *Multiagentensysteme.* Addison-Wesley, München, 2001

[Lee und Pinedo 1997] Lee, YH ; Pinedo, ML: Scheduling jobs on parallel machines with sequence-dependent setup times. In: *European Journal of Operational Research* 100 (1997), S. 464–474

[MIM] *MASM Test Data Sets.* ftp://ftp.eas.asu.edu/pub/centers/masmlab/factory_datasets/. – Abgerufen am 24.02.2004

[Morton 1981] Morton, T.E.: Forward Algorithms for Forward-Thinking-Managers. In: *Applications of Management* (1981), S. 1–55

[Mönch und Rose 2003] Mönch, Lars ; Rose, Oliver: Shifting-Bottleneck-Heuristik für komplexe Produktionssysteme softwaretechnische Realisierung und Leistungsbewertung. (2003)

[Mönch u. a. 2002] Mönch, Lars ; Rose, Oliver ; Sturm, Roland: Framework for the perfromance assessment of Shop-Floor Control Systems. (2002)

[Mönch u. a.] Mönch, Lars ; Stehli, Marcel ; Zimmermann, Jens: FABMAS: an Agent-Based System for Production Control of Semiconductor Manufacturing Processes

[Oey und Mason 2001] Oey, Kasin ; Mason, Scott J.: Scheduling batch processing machines in complex job shops. In: *Proceedings of the 2001 Winter Simulation Conference*, 2001, S. 1200–1207

[Ovacik und Uzsoy 1997] Ovacik, Irfan M. ; Uzsoy, Reha: *Decomposition methods for complex Factory Scheduling Problems*. 1. Auflage. Boston/Dordrecht/London : Kluwer Academic Publishers, 1997

[Pabst 2003] Pabst, Detlef: *Behandlung von Vorgängerbeziehungen in der Shifting-Bottleneck-Heuristik angewendet in einem dynamischen Halbleiterumfeld*, Bayrische Julius Maximians Universität, Würzburg, Diplomarbeit, 2003

[Pinedo 2002] Pinedo, M.: *Scheduling: Theory, Algorithm, and Systems*. Second Edition. Prentice Hall, 2002

[Review 2003] Review, Technology: Moores Gesetz. In: *Technology Review* 10 (2003), S. 16–24

[Rose 2003] Rose, Oliver: Comparison of Due-date Oriented Dispatch Rules in Semiconductor Manufacturing. (2003)

[Stallman 2002] Stallman, Richard M.: *Free Software Free Society selected essays of Richard M. Stallman*. 59 Temple Place Boston, MA : Free Software Foundation, 2002

[STL a] *Standard Template Library Programmer's Guide*. `http://www.sgi.com/tech/stl/`. a. – Abgerufen am 24.02.2004

[STL b] *STLPort Webpräsenz*. `http://www.stlport.org/`. b. – Abgerufen am 24.02.2004

[Stroustrup 2000] Stroustrup, Bjarne: *Die C++-Programmiersprache*. 4. Auflage, Deutsche Übersetzung von Nicolai Josuttis und Achim Lörke. München : Addison-Wesley, 2000

[Zimmermann 2003] Zimmermann, Jens: *Konzeption und prototypische Implementierung eines hierarchisch organisierten Multi-Agenten-Systems zur Steuerung einer Halbleiterfabrik*, Technische Universität Ilmenau, Diplomarbeit, 2003