Ablaufplanungsprobleme mit beschränkter Verfügbarkeit der Prozessoren und beschränkter Anzahl der Präemptionen

Dissertation
zur Erlangung des Grades eines
Doktors der Wirtschaftswissenschaft
(doctor rerum oeconomicarum)
der Rechts- und Wirtschaftswissenschaftlichen Fakultät
der Universität des Saarlandes

vorgelegt von
Dipl.-Inform. Oliver Braun
Saarbrücken 2002

Tag der Disputation:	04. März 2002
Dekan:	Prof. Dr. Klaus Grupp
Erstberichterstatter:	Prof. Dr. Günter Schmidt
Zweitberichterstatter:	Prof. Dr. Jürgen Zimmermann

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Ablaufplanungsprobleme treten in allen Wissensgebieten und in allen Lebensbereichen auf und bestehen im wesentlichen darin, eine endliche Menge von Aufträgen einer endlichen Anzahl von Prozessoren so zuzuordnen, daß unter Einhaltung zu beachtender Nebenbedingungen bestimmte Optimierungsziele möglichst gut erreicht werden.

In der vorliegenden Arbeit werden die beiden klassischen Ablaufplanungsprobleme $F2 \mid pmtn \mid C_{max}$ und $P \mid pmtn \mid C_{max}$ betrachtet, deren Optimierungsziel jeweils in der Minimierung der Planlänge, d.h. in der Minimierung des maximalen Fertigstellungszeitpunkts eines Auftrags (C_{max}) besteht. Die Aufträge stehen zu Beginn zur Bearbeitung bereit und unterliegen keinen Vorrangbeziehungen. Präemptionen ($pmtn$) sind in der Hinsicht zulässig, daß Aufträge zu jedem beliebigen Zeitpunkt unterbrochen und auf demselben oder einem anderen Prozessor sofort oder zu einem späteren Zeitpunkt ohne Zeitstrafe weiterbearbeitet werden können. Bei den Prozessoren handelt es sich zum einen um spezialisierte Prozessoren (Flowshop mit zwei Prozessoren, $F2$) und zum anderen um parallele, identisch qualifizierte Prozessoren (P).

Zur Lösung der beiden Probleme $F2 \mid pmtn \mid C_{max}$ und $P \mid pmtn \mid C_{max}$ können die in den 50er Jahren veröffentlichten Algorithmen von Johnson [Joh54] und von McNaughton [McN59] verwendet werden. Der Algorithmus von Johnson wurde ursprünglich zur Lösung von $F2 \mid\mid C_{max}$ erstellt, löst aber auch das Problem $F2 \mid pmtn \mid$

C_{max} optimal [GS78]. Die Algorithmen von Johnson und von McNaughton erzeugen in jeweils (in der Anzahl der zu verplanenden Aufträge) polynomialer Zeit optimale Lösungen. Nachdem eine grundlegende Analyse und eine komplexitätstheoretische Einordnung der beiden Probleme mit der Entwicklung der Komplexitätstheorie in den 70er Jahren möglich wurde, gewinnen in letzter Zeit immer mehr aus der Praxis motivierte Fragestellungen an Bedeutung. Zwei dieser Fragestellungen werden in der vorliegenden Arbeit aufgegriffen und analysiert.

Zum einen wird sowohl für das Problem $F2 \mid pmtn \mid C_{max}$ als auch für das Problem $P \mid pmtn \mid C_{max}$ davon ausgegangen, daß alle Prozessoren während des gesamten Planungszeitraums kontinuierlich für produktive Tätigkeiten zur Verfügung stehen. Diese Annahme ist allerdings bei real auftretenden Ablaufplanungsproblemen häufig nicht gerechtfertigt. Beispielsweise fallen Maschinen wegen möglicher Defekte oder wegen Wartungsarbeiten aus, und Arbeiter stehen wegen Krankheit oder Urlaubstagen nicht zur Verfügung. Häufig tritt in praktischen Anwendungen auch das Problem auf, daß zwei Mengen von Aufträgen bearbeitet werden sollen, von denen die Aufträge der einen Menge festen Zeitintervallen zugeordnet sind und die Aufträge der anderen Menge in den verbleibenden freien Prozessorintervallen verplant werden sollen. Eine solche Konstellation entsteht beispielsweise aufgrund von Vorbelegungen bestimmter Zeitintervalle einzelner Prozessoren. So besteht ein Ziel der vorliegenden Arbeit darin, die beiden Ablaufplanungsprobleme $F2 \mid pmtn \mid C_{max}$ und $P \mid pmtn \mid C_{max}$ hinsichtlich beschränkter Verfügbarkeit der Prozessoren zu untersuchen.

Zum anderen sind die beiden Probleme $F2 \mid pmtn \mid C_{max}$ und $P \mid pmtn \mid C_{max}$ so formuliert, daß zu ihrer Lösung Ablaufpläne mit einer beliebigen Anzahl von Präemptionen zugelassen sind. Dabei wird vorausgesetzt, daß mit einer Präemption keine oder lediglich vernachlässigbar geringe zeitlichen Kosten (wie beispielsweise Transport- oder Rüstkosten) auftreten. Nichtsdestotrotz entstehen in praktischen Anwendungen Kosten wie Lagerhaltungskosten oder Kosten für zusätzliche Produktionsfaktoren (z.B. menschliche Arbeitskräfte, Maschinen, Werkstoffe, Energie), die benötigt werden, wenn ein Auftrag von einem Prozessor genommen und auf einen anderen Prozes-

sor wieder aufgesetzt wird. Weitere Argumente für eine möglichst geringe Anzahl von Präemptionen sind, daß in eine laufende Produktion weniger häufig eingegriffen wird, daß die Aufträge seltener transportiert werden müssen, die Prozessoren seltener zur Bearbeitung eines anderen Auftrags hergerichtet (z.B. gereinigt) werden müssen und daß die neben den Prozessoren zur Bearbeitung der Aufträge benötigten Betriebsmittel seltener gewechselt werden müssen. Generell vereinfacht sich die Produktionsplanung, wenn wenige Präemptionen zugelassen sind, und in der Regel hat eine geringe Anzahl von Präemptionen auch Qualitätsverbesserungen zur Folge. So besteht ein weiteres Ziel darin, die beiden Ablaufplanungsprobleme $F2 \mid pmtn \mid C_{max}$ und $P \mid pmtn \mid C_{max}$ hinsichtlich einer beschränkten Anzahl von Präemptionen zu untersuchen.

Zusammenfassend werden in der vorliegenden Arbeit zwei Ziele verfolgt (vgl. Abb. 1.1):

- Zum einen werden die Probleme $F2 \mid pmtn \mid C_{max}$ und $P \mid pmtn \mid C_{max}$ hinsichtlich beschränkter Verfügbarkeit der Prozessoren (NC) untersucht.
- Zum anderen werden die Probleme $F2 \mid pmtn \mid C_{max}$ und $P \mid pmtn \mid C_{max}$ hinsichtlich einer beschränkten Anzahl i der Präemptionen (i-$pmtn$) untersucht.

	$F2 \mid pmtn \mid C_{max}$	$P \mid pmtn \mid C_{max}$
Beschränkte Verfügbarkeit der Prozessoren	$F2, NC \mid pmtn \mid C_{max}$	$P, NC \mid pmtn \mid C_{max}$
Beschränkte Anzahl i der Präemptionen	$F2 \mid i\text{-}pmtn \mid C_{max}$	$P \mid i\text{-}pmtn \mid C_{max}$

Abbildung 1.1: *Übersicht über die in der Arbeit untersuchten Ablaufplanungsprobleme*

Gliederung der Arbeit: Neben einer Einordnung der Produktions- und Ablaufplanung in das betriebswirtschaftliche Umfeld erfolgt in Kapitel 2 eine Beschreibung der

den weiteren Ausführungen zugrundeliegenden Ablaufplanungsmodelle. Zusätzlich werden grundlegende Begriffe aus der Theorie der Berechenbarkeit, soweit sie in der Arbeit zur Modellanalyse und zur komplexitätstheoretischen Einordnung von Ablaufplanungsproblemen benötigt werden, vorgestellt.

Kapitel 3 behandelt die beiden Probleme $F2 \mid pmtn \mid C_{max}$ und $P \mid pmtn \mid C_{max}$ hinsichtlich beschränkt verfügbarer Prozessoren. Während das Problem $F2 \mid pmtn \mid C_{max}$ mit dem Algorithmus von Johnson in polynomialer Zeit optimal gelöst werden kann, wird zur Lösung des gleichen Problems mit beschränkt verfügbaren Prozessoren ein exponentieller Zeitaufwand benötigt. Heuristische Lösungsansätze (Breit [Bre00] und Kubiak *et al.* [KBFBS02]) zeigen, daß auch das Problem mit beschränkt verfügbaren Prozessoren üblicherweise optimal gelöst wird, wenn die Aufträge in der durch den Johnson-Algorithmus ermittelten Reihenfolge eingeplant werden. In Kapitel 3.1 wird daher der Frage nachgegangen, unter welchen Bedingungen eine Johnson-Reihenfolge zur Lösung von $F2 \mid pmtn \mid C_{max}$ optimal bleibt, wenn die beiden Prozessoren beschränkt verfügbar sind. Darauf aufbauend wird ein Algorithmus entwickelt, der für beliebige Probleminstanzen von $F2, NC \mid pmtn \mid C_{max}$ in polynomialer Zeit feststellt, ob eine für das Problem $F2 \mid pmtn \mid C_{max}$ optimale Johnson-Reihenfolge auch im Falle beschränkt verfügbarer Prozessoren optimal bleibt. Das vorgestellte Verfahren erlaubt erstmalig die Lösung großer Probleminstanzen des Problems $F2, NC \mid pmtn \mid C_{max}$ mit bis zu 10000 Aufträgen und 1000 Nichtverfügbarkeitsintervallen.

In Abschnitt 3.2 werden aus der Literatur bekannte Algorithmen zur Lösung von $P, NC \mid pmtn \mid C_{max}$ untersucht. In Abhängigkeit von dem Zeitpunkt, zu dem ein Algorithmus Informationen über die exakten Verfügbarkeiten der Prozessoren erhält, werden die drei Fälle *offline*, *nearly-online* und *online* unterschieden. Sind die Prozessorverfügbarkeiten im voraus bekannt, spricht man vom *offline setting*. Kennt man jeweils den nächsten Zeitpunkt, zu dem sich die Verfügbarkeiten der Prozessoren ändern, spricht man vom *nearly-online setting*. Kann zu jedem beliebigen Zeitpunkt ein Prozessor ausfallen oder wieder eingesetzt werden, spricht man vom *online setting*. Drei Algorithmen, die die jeweiligen Planungssituationen berücksichtigen, werden verglei-

chend untersucht. Ziel war eine Vereinheitlichung der Darstellung, eine detaillierte und vergleichende Ausarbeitung der Algorithmen und eine Analyse der im *worst-case* erzeugten Anzahl der Präemptionen des *nearly-online* Algorithmus. Weiterhin wurde die bislang unbewiesene Vermutung untersucht, ob es stets bezüglich der Planlänge optimale *Online*-Algorithmen gibt, falls die Anzahl der Prozessoren kleiner als die Anzahl der Aufträge ist.

In Kapitel 4 werden die beiden Probleme $F2 \mid pmtn \mid C_{max}$ und $P \mid pmtn \mid C_{max}$ hinsichtlich einer beschränkten Anzahl von Präemptionen untersucht. In Abschnitt 4.1 wird das Problem $F2 \mid i\text{-}pmtn \mid C_{max}$ kurz angesprochen. Da der Johnson-Algorithmus sowohl das Problem $F2 \mid pmtn \mid C_{max}$ als auch das Problem $F2 \mid\mid C_{max}$ in polynomialer Zeit optimal löst, gilt das gleiche für das Problem $F2 \mid i\text{-}pmtn \mid C_{max}$ für alle $i \geq 0$.

Abschnitt 4.2 behandelt das Problem $P \mid i\text{-}pmtn \mid C_{max}$. Es ist bekannt, daß optimale nichtpräemptive Pläne (i=0 erlaubte Präemptionen) gegenüber optimalen präemptiven Plänen ($m-1$ oder mehr erlaubte Präemptionen) eine Gütegarantie von $C_{max}^{np^*} \leq (2 - 2/(m+1))C_{max}^{p^*}$ besitzen [HL92]. Zunächst wird gezeigt, daß für mit Hilfe der *LPT-Regel* erstellte Ablaufpläne die gleiche Güte garantiert werden kann. Daran anschließend wird das Problem untersucht, wie sich die Planlängen optimaler i-präemptiver Pläne, in denen die Anzahl der Präemptionen auf maximal i beschränkt ist, zu Planlängen optimaler präemptiver Pläne verhalten. Ziel war es, eine obere Schranke anzugeben, die eine Aussage darüber trifft, um wieviel schlechter im *worst-case* die Planlängen optimaler *i-präemptiver* Pläne im Vergleich zu Planlängen optimaler *präemptiver* Pläne sein können. Es wird bewiesen, daß $C_{max}^{ip^*} \leq (2 - 2/(m/(i+1)+1))C_{max}^{p^*}$ für $i \leq m-1$ und $C_{max}^{ip^*} = C_{max}^{p^*}$ für $i \geq m$ scharfe obere Schranken darstellen.

Schließlich erfolgt in Kapitel 5 eine Zusammenfassung der Ergebnisse und ein Ausblick auf weitere Forschungsmöglichkeiten.

Kapitel 2

Grundlagen

Aus betriebswirtschaftlicher Sicht ist die Ablaufplanung Teil der Produktionsplanung. Eine entsprechende Einordnung wird in Abschnitt 2.1 vorgenommen. Daran anschließend erfolgt in Abschnitt 2.2 eine Beschreibung des den weiteren Ausführungen zugrundeliegenden Entscheidungsmodells. Schließlich werden in Abschnitt 2.3 grundlegende Begriffe zur Modellanalyse und zur komplexitätstheoretischen Einordnung von Ablaufplanungsproblemen beschrieben. Zur Produktions- und Ablaufplanung sind eine Vielzahl von Publikationen erschienen. Grundlegende Einführungen sind [BEPSW01], [Bru01], [Pin01], [Neu96], [TSS94], [Hoi93], [KS93], [GGR92], [Fre82], [Bak74] und [CMM67].

2.1 Produktions- und Ablaufplanung

Ökonomischer Zweck aller wirtschaftlicher Bemühungen ist die *Produktion* von Gütern und Dienstleistungen. Unter Produktion versteht man den zielgerichteten Einsatz von Sachgütern und Dienstleistungen (*Input, Produktionsfaktoren*) zur Transformation in andere (wertgesteigerte) Sachgüter und Dienstleistungen (*Output, Produkte*).

Zu den Produktionsfaktoren (Einsatzfaktoren) werden üblicherweise nicht die *dispositiven Faktoren*, die alle Tätigkeiten zur Planung, Steuerung und Überwachung von Prozessen beinhalten, gezählt. Bei den Produktionsfaktoren wird zwischen Nutzungs-

faktoren und Verbrauchsfaktoren unterschieden (vgl. Tab. 2.1).

Arbeit (-skräfte)	Betriebsmittel	Zusatzfaktoren	Werkstoffe	Energie
	Grundstücke Gebäude Maschinen Patente Lizenzen Information	Fremde Dienstleistungen Infrastruktur Umwelt	Rohstoffe Vorprodukte Hilfsstoffe Betriebsstoffe	Strom Wasser Gas
Nutzungsfaktoren			Verbrauchsfaktoren	

Tabelle 2.1: *Produktionsfaktoren* ([Hoi93], Abb. 3 auf S. 4)

Nutzungsfaktoren (Arbeitskräfte und Betriebsmittel) stellen ihr Leistungsvermögen langfristig zur Verfügung. Verbrauchsfaktoren (Werkstoffe, Energie) werden bei ihrem Einsatz im Produktionsprozeß sofort verbraucht und stehen danach nicht mehr zur Verfügung. Neben diesen Produktionsfaktoren treten noch Zusatzfaktoren auf, zu denen man beispielsweise fremdbezogene Dienstleistungen (z.B. von Banken und Versicherungen), begleitende Infrastruktur oder auch die Beanspruchung der Umwelt zählen kann. Eine Einordnung der Produktion in den gesamten betrieblichen Kreislauf zeigt Abb. 2.1.

Produktion vollzieht sich nicht beliebig, sondern planvoll, dem ökonomischen Prinzip folgend. In der *Produktionsplanung* werden geeignete organisatorische Einheiten gebildet und ihr Zusammenwirken geregelt. Traditionell wird die Produktionsplanung nach zeitlichen Gesichtspunkten in lang-, mittel- und kurzfristige Produktionsplanung unterschieden (vgl. hier und im folgenden [Sch02]).

Die *langfristige* Produktionsplanung (*Prozeßplanung*) umfaßt Entscheidungen, die auf der *Typebene* von Prozessen zu treffen sind. Den Prozeßtyp erhält man dabei durch eine allgemeine Beschreibung des Prozesses.

Langfristige (strategische) Entscheidungen beziehen sich auf die Vorgabe globaler Rahmenbedingungen und umfassen unter anderem die Festlegung des Produktionspro-

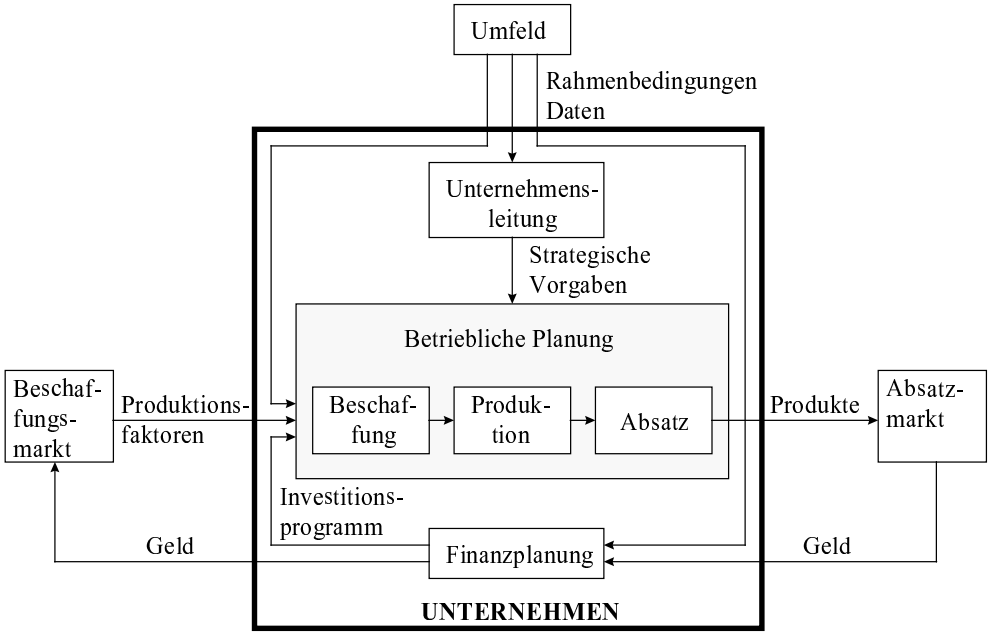

Abbildung 2.1: *Betrieblicher Kreislauf* ([KS93] Erster Teil, Abschnitt 2.1)

gramms, die langfristige Bereitstellung der benötigten Produktionsfaktoren und Überlegungen zu ihrem wirtschaftlichen Einsatz.

Mittel- und kurzfristige Produktionsplanung (*Auftragsplanung*) erfolgen auf der *Ausprägungsebene* von Prozessen. Die Ausprägung eines Prozesses ist dabei die Realisierung eines Prozesses im Rahmen einer Anwendung.

Mittelfristige (taktische) Entscheidungen beziehen sich auf die grobe Festlegung der Produktionsstruktur, der zu berücksichtigenden Mengen bzw. Termine und der zur Ausführung benötigten Ressourcen. Das Ergebnis läßt Spielraum für die *kurzfristige* Produktionsplanung.

Kurzfristige (operative) Entscheidungen beziehen sich im Detail auf die Nutzung der vorhandenen Prozessoren und zusätzlicher Ressourcen. Schnittstelle zur taktischen Ebene ist die Freigabe der Aufträge aus zeitlicher Sicht. Zur operativen Auftragsplanung gehören die *Systeminitialisierung*, der *Systembetrieb* und die *Systemüberwachung*.

Die *Systeminitialisierung* hat die Aufgabe, die Arbeitssysteme so einzurichten, daß alle Verrichtungen eines Auftrags ausgeführt werden können (Auftragsbildung). Aus-

gehend von der Auftragsbildung sind die verfügbaren Prozessoren zu gruppieren und ihre Funktionalität ist festzulegen.

Ausgangspunkt des *Systembetriebs* sind die Ergebnisse der Systeminitialisierung. Vor der Einschleusung der Aufträge müssen die Reihenfolge, in der die Aufträge das System betreten, und die Zeitpunkte, zu denen die Aufträge in das System eintreten, festgelegt werden. Routenwahl bedeutet, aus einer Menge möglicher Prozessoren einen für den nächsten Bearbeitungsschritt auszuwählen. Die Prozessorbelegung hat die Aufgabe, die Reihenfolge der Bearbeitung der Aufträge durch den jeweiligen Prozessor festzulegen und darüber hinaus Bearbeitungsbeginn und -ende jeder Verrichtung zeitlich zu fixieren.

Mit Hilfe der *Systemüberwachung* wird der geplante Zustand des Systems ständig mit dem aktuellen Zustand des Systems verglichen (Betriebsdatenerfassung und Soll-Ist-Vergleich).

Tabelle 2.2 faßt die Aufgaben der kurzfristigen Produktionsplanung zusammen.

Systeminitialisierung	Systembetrieb	Systemüberwachung
Auftragsbildung	Einschleusung	Betriebsdatenerfassung
Prozessorengruppierung	Routenwahl	Soll-Ist-Vergleich
Funktionszuordnung	Prozessorbelegung	

Tabelle 2.2: *Aufgaben der kurzfristigen Produktionsplanung*

Auf operativer Ebene erfolgt eine Trennung von *Offline Planung* und *Online Steuerung*. Die Systeminitialisierung ist Bestandteil der Offline Planung und die Systemüberwachung ist Bestandteil der Online Steuerung. Der Systembetrieb ist sowohl Gegenstand der Offline Planung als auch der Online Steuerung (vgl. Abb. 2.2).

Bindeglied zwischen Systeminitialisierung, -betrieb und -überwachung ist die *Ablaufplanung*, d.h. die zeitliche Zuordnung von Aufträgen zu Prozessoren.

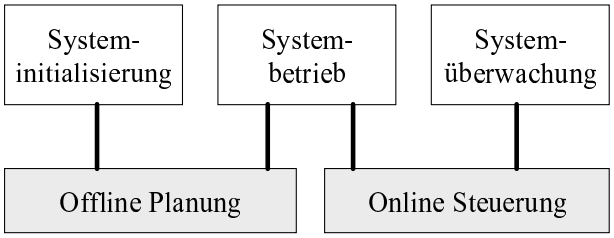

Abbildung 2.2: *Offline Planung und Online Steuerung*

Ein *Ablaufplanungsproblem* ist ein *kombinatorisches Optimierungsproblem*, das durch das Vorhandensein einer Alternativenmenge gekennzeichnet ist, aus der ein rational handelnder Entscheidungsträger eine Auswahl treffen muß. Ein Entscheidungsträger handelt dann rational, wenn er ein gesetztes Ziel bzw. einen zukünftigen, gegenüber dem gegenwärtigen im allgemeinen veränderten, erstrebenswerten Zustand, zu erreichen sucht (vgl. [DK96], S. 15).

Ablaufplanungsprobleme können aus deterministischer und aus stochastischer Sicht formuliert werden. Ein Problem ist *deterministisch*, wenn jeder Problemparameter mit Wahrscheinlichkeit 1 eintritt. Der deterministische Aspekt einer Problemmodellierung weist den Nachteil auf, daß durch eventuelle Störungen des Systems, mögliche Nacharbeiten oder sonstige unvorhersehbare Ereignisse die deterministische Problemformulierung durch stochastische Aspekte überlagert werden kann. In *stochastischen* Problemen erfolgt die Beschreibung der Parameter durch Verteilungsfunktionen.

Die hier betrachteten Ablaufplanungsprobleme sind sowohl *statischer* als auch *dynamischer* Natur, da die Problemdaten entweder gleich bleiben oder sich auch im Zeitverlauf ändern können.

Grundsätzlich wird angenommen, daß in einem Ablaufplanungsproblem n Aufträge $J_1, J_2, \ldots, J_n$ auf m Prozessoren $P_1, P_2, \ldots, P_m$ so zu verplanen sind, daß bestimmte Optimierungsziele unter Einhaltung zu beachtender Nebenbedingungen möglichst gut erreicht werden. Der Einfachheit halber schreibt man häufig anstelle von *Auftrag* J_j lediglich *Auftrag* j. Ebenso schreibt man häufig anstelle von *Prozessor* P_i lediglich *Prozessor* i.

Ergebnis der Ablaufplanung ist bei unseren Betrachtungen immer ein *Ablaufplan*, der häufig als *Gantt-Chart* dargestellt wird. In Abb. 2.3 werden fünf Aufträge $J_1, \ldots, J_5$ auf drei Prozessoren P_1, P_2, P_3 verplant.

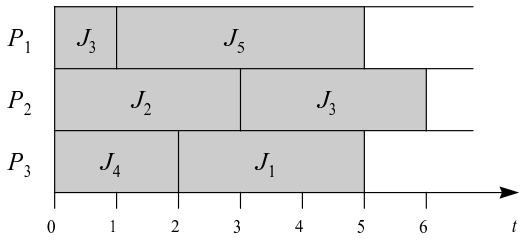

Abbildung 2.3: *Gantt-Chart*

Ablaufpläne können als *Systeme* aufgefaßt werden. Ein System (zum Systembegriff vergleiche [Sch99]) läßt sich beschreiben durch eine Menge von Elementen sowie eine Menge von Relationen, die die Beziehungen der Elemente angibt und eine Teilmenge des kartesischen Produkts der Basismenge der Elemente ist. Eine Klassifikation von Systemen unterscheidet natürliche und künstliche, statische und dynamische, geschlossene und offene sowie deterministische und stochastische Systeme. Ein *Ablaufplan* kann als ein künstliches System, das sowohl statisch als auch dynamisch, geschlossen oder offen und deterministisch oder stochastisch sein kann, aufgefaßt werden. Input sind Aufträge, Prozessoren, Nebenbedingungen und Optimierungsziele. Output ist bei den in der vorliegenden Arbeit betrachteten Ablaufplänen ein Gantt-Chart. Systemelemente sind Input und Output, Systembeziehungen sind im Gantt-Chart dargestellt. Die Systemgrenze ist der Anwendungsbereich, Systemziel ist die Erstellung eines, bezüglich der Optimierungsziele, optimalen Ablaufplans unter Berücksichtigung der einzuhaltenden Nebenbedingungen.

Ein Ablaufplan heißt *zulässig*, wenn keine Nebenbedingung verletzt wird. Ein zulässiger Ablaufplan erfüllt die beiden folgenden Bedingungen:

1. Ein Prozessor kann zu einem bestimmten Zeitpunkt nur einen Auftrag bearbeiten.
2. Ein Auftrag kann zu einem bestimmten Zeitpunkt nur von einem Prozessor bearbeitet werden.

Ein zulässiger Ablaufplan heißt *unverzögert*, wenn kein Prozessor Leerzeiten hat, obwohl ein Auftrag bearbeitet werden könnte. Unverzögerte Ablaufpläne haben keine unerzwungenen Leerzeiten. Optimale präemptive Ablaufpläne sind stets unverzögert. Alle im Rahmen der vorliegenden Arbeit betrachteten Ablaufplanungsprobleme haben optimale Lösungen, die unverzögert sind. Ein zulässiger Ablaufplan heißt *semiaktiv*, wenn kein Auftrag früher fertiggestellt werden kann, ohne daß die Bearbeitungsreihenfolge auf einem der zur Verfügung stehenden Prozessoren geändert wird.

Im nächsten Abschnitt wird das den weiteren Ausführungen zugrundeliegende Modell definiert.

2.2 Modelldefinition

Basierend auf einer Notation von Graham *et al.* [GLLRK79] schlagen Błażewicz *et al.* [BEPSW01] ein Klassifizierungsschema für Ablaufplanungsprobleme vor, das auch in der vorliegenden Arbeit verwendet wird. Darin werden Ablaufplanungsprobleme durch ein Tripel $\alpha \mid \beta \mid \gamma$ beschrieben, wobei

- α die Prozessoren,
- β die Aufträge und
- γ die Zielkriterien

beschreiben. Im folgenden werden zum einen die Parameter vorgestellt, die im weiteren Verlauf der Arbeit benötigt werden, zum anderen wird die Notation erweitert.

2.2.1 Prozessoren

Das erste Feld $\alpha = \alpha_1\alpha_2$ beschreibt die Prozessoren. Ein Prozessor kann dabei eine benötigte Maschine, aber auch benötigtes Personal, Raum, Material, Energie, ein Transportmittel oder ein bestimmtes Werkzeug sein.

2.2.1.1 Das Feld α_1

α_1 beschreibt die Art der Prozessoren. Wir betrachten zwei grundlegende Mehrprozessormodelle:

$\alpha_1 = F2$:

Zum einen werden spezialisierte Prozessoren vom Typ *Zwei-Prozessoren-Flow-Shop* betrachtet. Alle Aufträge durchlaufen dabei die beiden Prozessoren in der gleichen Reihenfolge (*Reihenfertigung*). Die Aufträge *fließen* sozusagen von Prozessor zu Prozessor. Dazu wird jeder Auftrag in zwei Verrichtungen zerlegt, die von den beiden Prozessoren nacheinander bearbeitet werden müssen (vgl. Abb. 2.4).

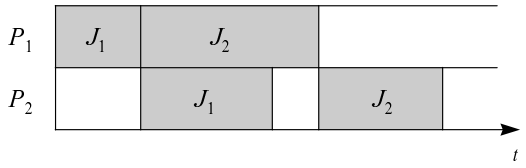

Abbildung 2.4: *Beispiel für einen Zwei-Prozessoren-Flow-Shop*

Bei Warenprozessen findet man *Flow Shops* in der Fließband-, Transferstraßen- und in der Reihenfertigung, bei Informationsprozessen in der seriellen Workflow-Verarbeitung. Voraussetzung für eine Prozeßorganisation als *Flow Shop* sind mittel- bis langfristig stabile Prozeßausprägungen. Ein Nachteil liegt in der mangelnden Flexibilität (vgl. [Sch02], S. 12ff.).

$\alpha_1 = P$:

Neben spezialisierten Prozessoren werden parallele, identisch qualifizierte Prozessoren betrachtet. Jeder Auftrag J_j kann von jedem der Prozessoren mit der gleichen Geschwindigkeit bearbeitet werden. Im Gantt-Chart in Abb. 2.3 werden drei parallele, identisch qualifizierte Prozessoren dargestellt. Das abstrakte Modell paralleler, identisch qualifizierter Prozessoren ist auf viele reale Anwendungen übertragbar. Beispiele aus der industriellen Fertigung sind ein Maschinenpark mit mehreren identisch qualifizierten Maschinen oder eine Gruppe von identisch qualifizierten Arbeitern. Weitere Anwendungen in flexiblen Fertigungssystemen werden in [BFHS88] und in [CS90] beschrieben.

2.2.1.2 Das Feld α_2

α_2 beschreibt die Art der Nichtverfügbarkeitsintervalle der Prozessoren und besteht aus lediglich einem Eintrag $\alpha_2 = NC_p^t$, wobei mit t der Zeitpunkt, zu dem ein Algorithmus Informationen über die Verfügbarkeiten der Prozessoren erhält und mit p das Verfügbarkeitsmuster der Prozessoren bezeichnet wird.

Generell sind, in Abhängigkeit von der zeitlichen Kenntnis der Verfügbarkeiten der Prozessoren, die folgenden Planungssituationen zu unterschieden:

1. $t = \textit{offline}$

   Die Verfügbarkeiten aller Prozessoren sind im voraus bekannt.

2. $t = \textit{nearly-online}$

   Häufig ist der nächste Zeitpunkt, zu dem sich die Anzahl der verfügbaren Prozessoren ändert, bekannt. Das ist zum Beispiel dann der Fall, wenn ein Prozessor (beispielsweise eine Maschine) gewartet werden soll, oder wenn ein Prozessor nach einer Reparatur wieder zur Verfügung steht. Geht man davon aus, daß die Menge der verfügbaren Prozessoren in der Zeit bis zur Wartung des Prozessors bzw. bis zum Wiedereinsatz desselben gleich bleibt, so hat man eine Vorausschau von einem Zeitintervall.

3. $t = \textit{online}$

   Zu jedem beliebigen Zeitpunkt kann ein Prozessor ausfallen oder wieder eingesetzt werden. Lediglich die aktuell verfügbaren Prozessoren sind bekannt.

In [Sch84] und in [LS95] werden verschiedene Verfügbarkeitsmuster von Prozessoren besprochen.

1. $p = \textit{const}$ (*constant pattern, konstantes Muster*)

   In diesem Fall stehen alle Prozessoren kontinuierlich zur Verfügung.

2. $p = zz$ (*zigzag pattern, zick-zack Muster*)

   In diesem Fall stehen stets entweder c oder $c - 1$ Prozessoren zur Verfügung.

3. $p = inc$ (*increasing pattern, zunehmendes Muster*)

   In diesem Fall gilt: $m_{k+1} \geq m_k, 1 \leq k \leq S-1$, mit S=Anzahl der Zeitpunkte, zu denen sich die Menge der verfügbaren Prozessoren ändert, und $m_k \leq m$ die Anzahl der Prozessoren (höchstens m) in einem Intervall, in dem sich die Menge der verfügbaren Prozessoren nicht ändert (Systemintervall).

4. $p = dec$ (*decreasing pattern, abnehmendes Muster*)

   In diesem Fall gilt: $m_{k+1} \leq m_k, 1 \leq k \leq S-1$

5. $p = sc$ (*staircase pattern, Treppenmuster*)

   In diesem Fall kann ein Prozessor P_i in einem bestimmten Systemintervall nur dann verfügbar sein, wenn auch Prozessor P_{i+1} verfügbar ist (vgl. Abb. 2.5).

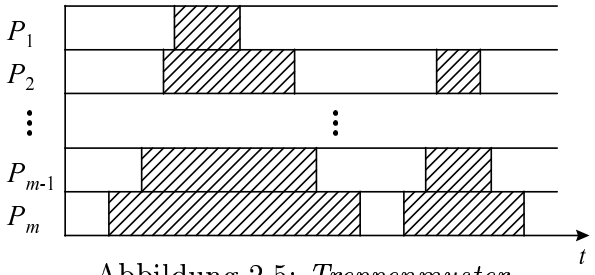

Abbildung 2.5: *Treppenmuster*

   Liegt ein Treppenmuster vor, so gilt $PC_i^r \leq PC_{i-1}^r$ und $PC_i \leq PC_{i-1}$ für alle $1 < i \leq m$, wobei mit PC_i^r die Länge des r-ten Verfügbarkeitsintervalls auf Prozessor P_i bezeichnet wird.

6. $p = \emptyset$ oder $p = win$ (*beliebiges Muster*)

   In diesem Fall stellen die Prozessoren ein beliebiges Verfügbarkeitsmuster dar.

2.2.2 Aufträge

Das zweite Feld $\beta \in \{\emptyset, pmtn, i\text{-}pmtn\}$ beschreibt die Eigenschaften der Aufträge. $\emptyset$ bezeichnet ein leeres Symbol, das man bei der Problemklassifikation wegläßt.

Ablaufplanungsprobleme können hinsichtlich der Zulässigkeit von Präemptionen unterschieden werden in solche Probleme, die keine Präemptionen zulassen und in solche, die Präemptionen erlauben. Die Art und Weise, wie dabei auf eine Präemption reagiert

wird, kann unterschiedlich geregelt sein. Beispielsweise kann angenommen werden, daß ein Auftrag nach einer Präemption wieder von vorne bearbeitet werden muß oder nur mit einer Zeitstrafe weiterbearbeitet werden darf (*non-resumable, semi-resumable*). Im allgemeinen und auch in der vorliegenden Arbeit wird davon ausgegangen, daß ein Auftrag zu jedem beliebigen Zeitpunkt unterbrochen werden darf und

- sofort oder zu einem späteren Zeitpunkt
- auf dem gleichen oder einem anderen Prozessor
- ohne Zeitstrafe

weiterbearbeitet werden kann (*resumable*).

Mit *i-präemptiven Plänen* werden Ablaufpläne bezeichnet, in denen maximal i Präemptionen zugelassen sind (Notation: $\beta = i\text{-}pmtn$). Pläne, in denen beliebig viele Präemptionen zugelassen sind, heißen *präemptive Pläne* (Notation: $\beta = pmtn$). Pläne, in denen keine Präemptionen zugelassen sind, heißen *nichtpräemptive Pläne* (Notation: $\beta = \emptyset$).

2.2.3 Zielkriterien

Das dritte Feld γ beschreibt die *Zielfunktion* bzw. das *Planungs-* oder auch *Optimierungsziel*. Bei den hier betrachteten Ablaufplanungsproblemen bildet eine *Zielfunktion* z eine Alternativenmenge X in die Menge der rationalen Zahlen ab. Die Alternativenmenge X ist endlich und besteht hierbei in allen zulässigen Ablaufplänen. Die Zielfunktion z ist zu minimieren, insofern ist das Optimierungsziel ein Minimierungsziel. Eine Alternative $x^* \in X$ mit dem Zielfunktionswert $z(x^*)$ erreicht das gesteckte Optimierungsziel, wenn keine andere Alternative $x' \in X$ existiert, die zu einem niedrigeren Zielfunktionswert als $z(x^*)$ führt. Alle Alternativen $x' \in X$ mit dieser Eigenschaft stellen *optimale Lösungen* dar und werden zu einer optimalen Lösungsmenge X^* zusammengefaßt. Es gilt

$$X^* := \{x^* \in X \mid \text{ es existiert kein } x' \in X \text{ mit } z(x') < z(x^*)\}\,.$$

Zusammen mit einer Minimierungsvorschrift bilden die Alternativenmenge X und die Zielfunktion z ein elementares *deterministisches Entscheidungsmodell*

$$min\,\{z(x) \mid x \in X\}\,.$$

Da hier lediglich eine Zielfunktion verwendet wird, spricht man auch von einem *skalaren Entscheidungsmodell*. Muß ein Entscheidungsträger eine Entscheidung auf der Basis unterschiedlicher Ziele treffen, liegt häufig ein Zielkonflikt vor, der Gegenstand der Darstellung und Analyse *vektorieller Entscheidungsmodelle* ist (vgl. [DK96], Kapitel 1 und 2).

Als Optimierungsziel wird in der vorliegenden Arbeit die Minimierung der Planlänge bzw. die Minimierung des Fertigstellungszeitpunkts des Auftrags, der zuletzt fertiggestellt wird, betrachtet. Die Planlänge stellt zugleich die Zeitspanne dar, in der sämtliche Aufträge im betrachteten Planungshorizont abgeschlossen sind. Für die Planlänge C_{max} gilt

$$C_{max} = \max_{j=1,\ldots,n} C_j$$

Die Minimierung der Planlänge ist eine *reguläre* Zielfunktion, da die Planlänge eine in den Fertigstellungszeitpunkten $C_j, j = 1, \ldots, n$, der Aufträge monoton wachsende Funktion ist.

In der vorliegenden Arbeit wird ein Modell betrachtet, in dem der Fertigstellungszeitpunkt eines Auftrags die Summe aus ablaufbedingten Wartezeiten vor und zwischen Bearbeitungen und aus seiner Bearbeitungszeit ist. Rüst-, Kontroll- und Transportzeiten mit den damit verbundenen Wartezeiten entfallen bzw. sind vernachlässigbar gering. In diesem Falle ist der Fertigstellungszeitpunkt eines Auftrages gleich der Durchlaufzeit des Auftrages. Bei der Bemessung der Durchlaufzeit eines Auftrags werden in der allgemeinsten Form sämtliche Arbeitsvorgänge, die zur Erledigung des betreffenden Auftrags zu verrichten sind, berücksichtigt (vgl. Abb. 2.6).

Eine kleine Planlänge bedeutet überlicherweise eine hohe Auslastung des Arbeitssystems. Allerdings sind durchlaufzeitbedingte Zielfunktionen und auf Kostenminimie-

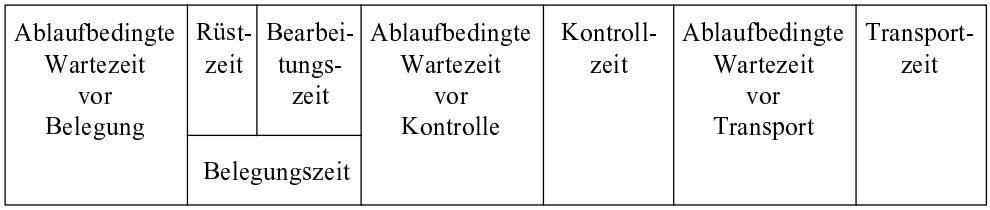

Abbildung 2.6: *Durchlaufzeitkomponenten* ([GGR92], Abb. 20 auf S. 142)

rung ausgerichtete Zielfunktionen wie Kapitalbindungs- oder auch Lagerhaltungskosten nicht notwendigerweise äquivalente Zielfunktionen. So steht das zur Bearbeitung der Aufträge benötigte Kapital nicht notwendigerweise bereits zu Beginn des Planungszeitraumes bereit. Umgekehrt erfolgt auch die Freisetzung des gebundenen Kapitals nicht notwendigerweise erst nach Bearbeitung sämtlicher Aufträge. Bei gegebener Betriebsbereitschaft der Prozessoren fallen die mit diesen Prozessoren verbundenen Bereitschafts- bzw. Dispositionskosten unabhängig von der jeweiligen Kapazitätsauslastung in konstanter Höhe an. Ein Einfluß der Kapazitätsauslastung auf Opportunitätskosten in Form entgangener Gewinne besteht nur dann, wenn zusätzliche Aufträge zur Bearbeitung vorliegen und die freien Kapazitäten genutzt werden können (vgl. [GGR92], S. 190).

2.3 Modellanalyse

Grundlegende Einführungen in die Theorie der Berechenbarkeit stellen [PS98], [Pap93], [GJ79] und [Kar72] dar. An dieser Stelle werden die wichtigsten Ergebnisse, soweit sie im weiteren Verlauf der Arbeit benötigt werden, zusammenfassend dargestellt. Ablaufplanungsprobleme sind *kombinatorische Optimierungsprobleme* (I, A), wobei I eine Probleminstanz und A eine Problemlösung derart ist, daß der Wert einer bestimmten Zielfunktion optimiert wird. Ein *Algorithmus* ist ein Verfahren, das so präzise formuliert ist, daß es von einem mechanisch oder elektronisch arbeitenden Gerät durchgeführt werden kann. Der Name *Algorithmus* leitet sich aus dem Namen des persisch-arabischen Mathematikers Ibn Mûsâ Al-Chwârismî ab, der im 9. Jahrhundert ein Buch über die

„Regeln der Wiedereinsetzung und Reduktion“ schrieb. Ein Algorithmus realisiert eine *berechenbare Funktion*. Funktionen, die nicht berechenbar sind, können nicht durch Algorithmen und somit auch nicht durch Programme einer Rechenanlage beschrieben werden. Die *Komplexität* eines Algorithmus ist der zu seiner Ausführung erforderliche Aufwand an Betriebsmitteln wie Rechenzeit oder Speicherplatz (Rechenaufwand) innerhalb eines bestimmten Berechnungsmodells. Der Rechenaufwand wird dabei in der Regel in Abhängigkeit von den Eingabewerten, meist in Abhängigkeit von der *Eingabelänge* gemessen. Hier und im folgenden wird angenommen, daß eine Probleminstanz I so kodiert ist, daß die Eingabelänge k nicht exponentiell im Vergleich zu anderen möglichen Kodierungen wächst. Die *Laufzeit* eines Algorithmus ist die Anzahl der Rechenschritte, die bei seiner Durchführung für eine bestimmte Eingabe gemacht werden. Die *worst-case* Laufzeit ist die schlechtest mögliche Laufzeit eines Algorithmus für eine Eingabe einer bestimmten Länge. Bei der Untersuchung der *worst-case* Laufzeit greift man also für jede Eingabelänge k diejenige Eingabe heraus, für die der Algorithmus die längste Laufzeit benötigt. Alle in der vorliegenden Arbeit vorgenommenen Laufzeitabschätzungen sind *worst-case* Abschätzungen.

Definition 2.1 *Mit $O(g(k))$ wird die Klasse aller Funktionen f bezeichnet mit der Eigenschaft: $\exists c > 0$ und $\exists k_0 > 0$, so daß $\forall k \geq n_0$ gilt: $f(k) \leq cg(k)$.*

Ein Algorithmus hat *polynomiale Laufzeit* $O(g(k)$, wenn die Anzahl der elementaren Rechenschritte, die der Algorithmus ausführt, für alle Eingabelängen k ein Polynom in k ist, die Funktion g also ein Polynom ist. Entsprechend spricht man von *exponentieller Laufzeit*, wenn die Anzahl der elementaren Rechenschritte des Algorithmus exponentiell mit der Eingabelänge k wächst.

Die beiden Forderungen an einen Algorithmus, einerseits möglichst wenig Speicherplatz zu verbrauchen und andererseits eine möglichst geringe Laufzeit zu besitzen, sind nicht immer gleichzeitig zu erfüllen. Im allgemeinen konkurrieren diese beiden Forderungen sogar miteinander (*Time-space-trade-off*).

Das zu einem Optimierungsproblem gehörende *Entscheidungsproblem* beantwortet die Frage, ob ein bestimmter Zielfunktionswert erreichbar oder nicht erreichbar ist. Das Entscheidungsproblem liefert als Antwort also lediglich *ja* oder *nein* und ist daher nicht schwieriger als das zugehörige Optimierungsproblem zu lösen. Wenn das Entscheidungsproblem aber schon schwierig ist, dann ist das Optimierungsproblem auch schwierig.

Zur komplexitätstheoretischen Einordnung der untersuchten Ablaufplanungsprobleme werden folgende Definitionen 2.2 und 2.3 benötigt.

Definition 2.2 *Die Klasse P besteht aus allen Entscheidungsproblemen, die von einer Deterministischen Turing-Maschine (einem Computer) in (in der Eingabelänge) polynomialer Zeit gelöst werden können. Die Klasse NP besteht aus allen Entscheidungsproblemen, die von einer Nichtdeterministischen Turing-Maschine in (in der Eingabelänge) polynomialer Zeit gelöst werden können.*

Es ist leicht zu sehen, daß $P \subseteq NP$. Offen ist die Frage, ob auch $NP \subseteq P$ und damit $P = NP$ gilt. Es wird angenommen, daß dies nicht der Fall ist, ein Beweis dafür konnte bis jetzt aber noch nicht erbracht werden.

Definition 2.3 *Ein Problem* Π *ist polynomial transformierbar auf ein Problem* Π'*, wenn aus einem Polynomialzeitalgorithmus für* Π' *auch ein Polynomialzeitalgorithmus für* Π *erzeugt werden kann. Wenn also kein Polynomialzeitalgorithmus für* Π *existiert, existiert auch kein Polynomialzeitalgorithmus für* Π'*. Genauer gilt: Sei* X *ein Alphabet, und* $L_1, L_2 \subseteq X^*$ *seien Sprachen.* L_1 *ist polynomial auf* L_2 *transformierbar, wenn es einen polynomial-zeitbeschränkten Algorithmus gibt, der eine Funktion* $f : X^* \longrightarrow X^*$ *berechnet, für die gilt:*

$$w \in L_1 \text{ dann und nur dann, wenn } f(w) \in L_2$$

für alle $w \in X^*$*. Für Probleme anstelle von Sprachen wird die Transformierbarkeit analog definiert. Ein Entscheidungsproblem heißt NP-vollständig, wenn*

1. $\Pi \in NP$

2. *Für jedes Problem* $\Pi' \in NP$ *gilt:* Π' *ist polynomial transformierbar auf* Π

Einige *NP*-vollständigen Entscheidungsprobleme lassen sich in *pseudo-polynomialer Zeit* lösen, wenn die Eingabe unär kodiert wird. Probleme, für die kein in der Länge der unär kodierten Eingabe polynomialer Algorithmus existiert, heißen *unär NP-vollständig* oder auch *NP-vollständig im strengen Sinne.*

Ein grundlegendes *NP*-vollständiges Problem ist PARTITION, das wie folgt formuliert werden kann: Gegeben seien positive ganze Zahlen $a_1, \ldots, a_t$ und $b = (1/2)\sum_{j=1}^{t} a_j$. Frage: Gibt es zwei disjunkte Teilmengen S_1 und S_2, so daß $\sum_{j \in S_1} a_j = \sum_{j \in S_2} a_j = b$?

Ein grundlegendes im strengen Sinne *NP*-vollständiges Problem ist 3-PARTITION, das wie folgt formuliert werden kann: Gegeben seien positive ganze Zahlen $a_1, \ldots, a_{3t}$ und $b = (1/t)\sum_{j=1}^{3t} a_j$ mit $b/4 < a_j < b/2, j = 1, \ldots, 3t$. Frage: Gibt es zwei disjunkte Teilmengen S_1 und S_2, so daß $\sum_{j \in S_1} a_j = \sum_{j \in S_2} a_j = b$? Gibt es t paarweise disjunkte Teilmengen $S_1, S_2, \ldots, S_t$, so daß $\sum_{j \in S_1} a_j = \sum_{j \in S_2} a_j = \ldots = \sum_{j \in S_t} a_j = b$?

Ein Optimierungsproblem nennt man *NP-schwer*, wenn sein entsprechendes Entscheidungsproblem *NP-vollständig* ist. Um zu zeigen, daß ein Optimierungsproblem *NP*-schwer ist, kann man zum Beispiel wie folgt vorgehen (*Beweis durch Restriktion*):

1. Zeige, daß das zu dem Optimierungsproblem zugehörige Entscheidungsproblem zur Klasse *NP* gehört.
2. Zeige, daß PARTITION ein Spezialfall des zu dem Optimierungsproblem zugehörigen Entscheidungsproblem ist.

Entsprechend kann man vorgehen, wenn gezeigt werden soll, daß ein Problem *NP-schwer im strengen Sinne* ist (dabei wird PARTITION durch 3-PARTITION ersetzt).

Nicht für jedes Ablaufplanungsproblem ist ein Algorithmus bekannt, der das Problem in polynomialer Zeit optimal löst. Ist man bei praktischen Anwendungen gezwungen, Lösungsalgorithmen für *NP*-schwere Ablaufplanungsprobleme zu entwickeln, so behilft

man sich häufig mit *Heuristiken*, da exakte Algorithmen wegen der exponentiellen Laufzeit kaum eingesetzt werden können.

Von *Approximierungsalgorithmen* spricht man, wenn man für eine Heuristik eine *Gütegarantie* bezüglich der optimalen Lösung angeben kann. Eine Gütegarantie für einen Approximationsalgorithmus A zur Lösung eines Ablaufplanungsproblems ist eine Aussage darüber, wie der Algorithmus A im ungünstigsten Fall gegenüber einem optimalen Algorithmus abschneidet. Ist das Optimierungsziel ein Minimierungsziel, so lautet die Aussage: *Für alle Probleminstanzen I gilt:* $A(I) \leq R_A(I) \cdot OPT(I)$, wobei $A(I)$ der Zielfunktionswert ist, der mit dem Algorithmus A gefunden wurde und $OPT(I)$ der minimale Zielfunktionswert ist. Ziel ist, den kleinstmöglichen Faktor $r = \min\{R_A(I)\}$ zu bestimmen, d.h. die bestmögliche Gütegarantie. Die Frage, ob nicht eventuell noch bessere Gütegarantien (Schranken) erreicht werden können, kann man beantworten, indem man Probleminstanzen angibt, für die der Algorithmus die Gütegarantie genau erreicht.

Für einige *NP*-schwere Probleme existiert ein *polynomiales Approximationsschema*, welches eine Menge von Algorithmen $\{A_\epsilon\}$ ist, so daß für jedes $\epsilon > 0$ ein Polynomialzeitalgorithmus A_ϵ mit Gütegarantie $r = (1 + \epsilon)$ existiert. Dabei kann die Laufzeit von $\{A_\epsilon\}$ zusätzlich zur Eingabelänge auch noch von ϵ abhängen. Wenn die Laufzeit polynomial in der Eingabelänge und in $1/\epsilon$ ist, heißt die Menge der Approximationsalgorithmen ein *vollständiges polynomiales Approximationsschema*. Für *im strengen Sinne NP-schwere Probleme* kann gezeigt werden, daß kein vollständig polynomiales Approximationsschema existiert, es sei denn $P = NP$.

Kapitel 3

Beschränkte Verfügbarkeit der Prozessoren

In klassischen Modellen der Ablaufplanung wird davon ausgegangen, daß alle Prozessoren während des gesamten Planungszeitraums kontinuierlich zur Verfügung stehen. Diese Annahme ist allerdings häufig nicht gerechtfertigt. Durch zufällige Ausfälle, Wartung, Reparatur oder sonstige Ereignisse können die Prozessoren lediglich in bestimmten Zeitintervallen zur Bearbeitung von Aufträgen bereitstehen. Im vorliegenden Kapitel wird dieses Problem aufgegriffen und für spezialisierte Prozessoren (Flowshop mit zwei Prozessoren) sowie für parallele, identisch qualifizierte Prozessoren untersucht. Beschränkungen der Verfügbarkeit der Prozessoren treten beispielsweise auf Grund von Vorbelegungen der Prozessoren aus vorhergehenden Planungsintervallen oder auf Grund von plötzlich auftretenden Prozessorausfällen und damit verbundenen Wartungsarbeiten auf. Beschränkt verfügbare Prozessoren können als partiell erneuerbare Ressourcen betrachtet werden (vgl. [Zim01]). Weitere Anwendungsbeispiele, die eine Untersuchung von Ablaufplanungsproblemen mit beschränkt verfügbaren Prozessoren rechtfertigen, findet man u.a. in [Sch00] und in [SS98].

In Abschnitt 3.1 wird ein Ablaufplanungsproblem in einem Flowshop mit zwei Prozessoren hinsichtlich beschränkter Verfügbarkeit der Prozessoren untersucht. Sind beide Prozessoren während des gesamten Planungshorizonts verfügbar, läßt sich das Problem

mit Hilfe des Algorithmus von Johnson [Joh54] ohne Präemptionen und (bezüglich der Planlänge) optimal lösen (Abschnitt 3.1.1). Der Algorithmus erzeugt eine Permutation der Aufträge (*Johnson-Permutation*) und verplant die Aufträge in dieser Reihenfolge auf die beiden Prozessoren. In [GS78] wird gezeigt, daß eine Johnson-Permutation, die das Problem $F2 \mid\mid C_{max}$ optimal löst, auch das Problem $F2 \mid pmtn \mid C_{max}$ optimal löst. Eine Johnson-Permutation ist nicht mehr notwendigerweise optimal, wenn die Prozessoren nicht kontinuierlich zur Verfügung stehen. Der obere Teil der Abb. 3.1 zeigt eine Einplanung der Aufträge in der Johnson-Permutation (J_1, J_2, J_3). Im unteren Teil werden die Aufträge in einer optimalen Reihenfolge (J_3, J_2, J_1) eingeplant. Das Nichtverfügbarkeitsintervall $[4, 6]$ auf Prozessor P_1 ist schraffiert dargestellt. Die Intervalle, in denen Aufträge eingeplant sind, sind hellgrau hinterlegt.

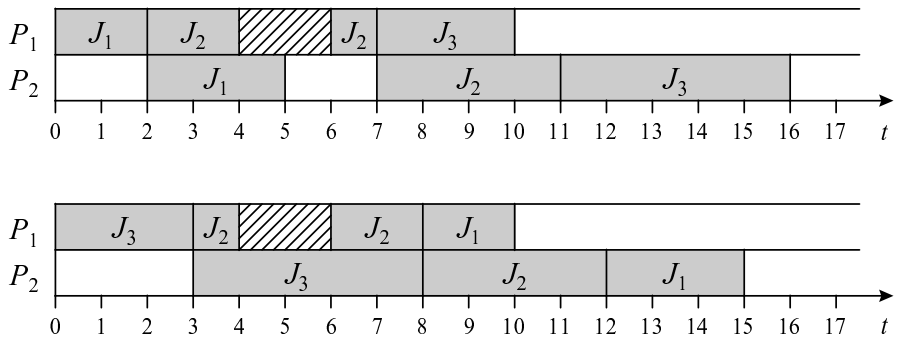

Abbildung 3.1: *Mit dem Algorithmus von Johnson erzeugter nicht optimaler Plan, wenn die Prozessoren nicht kontinuierlich zur Verfügung stehen.*

Experimente haben gezeigt, daß Johnson-Permutationen häufig auch im Falle beschränkt verfügbarer Prozessoren optimal bleiben (vgl. Breit [Bre00] und Kubiak *et al.* [KBFBS01]). In den Abschnitten 3.1.2 bis 3.1.4 (vgl. Braun *et al.* [BLSS02] und [BSS00]) wird daher der Frage nachgegangen, ob man auf Grund einer Information über die Lage der Nichtverfügbarkeitsintervalle eine Aussage darüber treffen kann, unter welchen hinreichenden Bedingungen eine Johnson-Permutation auch im Falle beschränkt verfügbarer Prozessoren optimal bleibt. Genauer wird eine *Stabilitätsanalyse* für mit Hilfe des Algorithmus von Johnson erstellte Ablaufpläne für das Problem $F2, NC^{offline} \mid pmtn \mid C_{max}$ durchgeführt. Basierend auf diesen Ergebnissen

wird ein Algorithmus, der (für Probleminstanzen von $F2 \mid pmtn \mid C_{max}$) optimale Johnson-Permutationen daraufhin überprüft, ob sie auch für Probleminstanzen von $F2, NC \mid pmtn \mid C_{max}$ (mit ansonsten gleichen Eingabedaten) optimal bleiben, entworfen. Die Leistungsfähigkeit des Algorithmus wurde für eine große Anzahl von Testinstanzen überprüft. Abschnitt 3.1.4 beinhaltet eine Diskussion der erhaltenen Ergebnisse.

Liegt ein konstantes Verfügbarkeitsmuster der Prozessoren vor, kann das Problem $P, NC^{offline}_{const} \mid pmtn \mid C_{max}$ mit dem Algorithmus von McNaughton [McN59] in (in der Anzahl der Aufträge) polynomialer Zeit und bezüglich der Planlänge optimal gelöst werden (Abschnitt 3.2.1). Der Algorithmus von McNaughton ist nicht mehr notwendigerweise optimal, wenn beliebige Verfügbarkeitsmuster der Prozessoren vorliegen, da dann möglicherweise nicht zulässige Pläne erzeugt werden. Im linken Teil der Abb. 3.2 ist ein nicht zulässiger Plan abgebildet, im rechten Teil ein zulässiger. Das Nichtverfügbarkeitsintervall auf Prozessor P_2 ist schraffiert dargestellt.

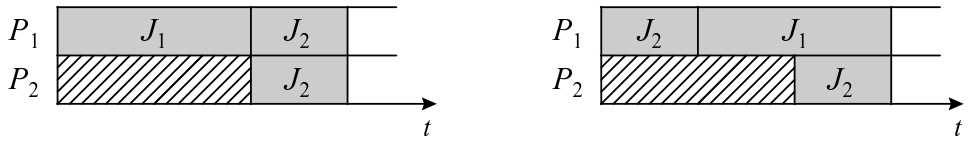

Abbildung 3.2: *Mit dem Algorithmus von McNaughton erzeugter nicht zulässiger Plan, wenn die Prozessoren nicht kontinuierlich zur Verfügung stehen.*

In Abhängigkeit von den Zeitpunkten, zu denen ein Algorithmus Informationen über die Verfügbarkeitsintervalle der einzelnen Prozessoren erhält, werden in den Abschnitten 3.2.2 bis 3.2.4 aus der Literatur bekannte Algorithmen für die Probleme

- $P, NC^{offline} \mid pmtn \mid C_{max}$
- $P, NC^{nearly\text{-}online} \mid pmtn \mid C_{max}$
- $P, NC^{online} \mid pmtn \mid C_{max}$

in einer vereinheitlichten Notation beschrieben. Darüberhinaus erfolgt jeweils eine Analyse der *worst-case* Laufzeiten und der Anzahl der im *worst-case* erzeugten Präemptionen. In Abschnitt 3.1.4 wird zusätzlich die Vermutung untersucht, ob es stets bezüglich der Planlänge optimale *Online*-Algorithmen gibt, falls die Anzahl der Prozessoren kleiner als die Anzahl der Aufträge ist.

3.1 $F2, NC \mid pmtn \mid C_{max}$

Das Problem $F2, NC \mid pmtn \mid C_{max}$ läßt sich wie folgt beschreiben. Es sollen n Aufträge $J_1, \ldots, J_n$ auf zwei parallelen, identisch qualifizierten Prozessoren P_1, P_2 so verplant werden, daß die Planlänge C_{max} minimiert wird. Dabei besteht jeder Auftrag J_j aus zwei Verrichtungen $T_{1,j}$ und $T_{2,j}$ mit nichtnegativen Bearbeitungszeiten $p_{1,j}$ und $p_{2,j}, j \in \{1, \ldots, n\}$. Wenn eine Verrichtung unterbrochen wird, darf sie zum gleichen oder zu einem späteren Zeitpunkt auf demselben oder auf dem anderen Prozessor ohne Zeitstrafe weiterbearbeitet werden. Die Prozessoren können in bestimmten Zeitintervallen nicht zur Verfügung stehen.

Verwandte Ergebnisse

Stehen beide Prozessoren kontinuierlich zur Verfügung, läßt sich das Problem $F2, NC \mid pmtn \mid C_{max}$ optimal mit dem Algorithmus von Johnson [Joh54] lösen. Das gleiche Problem mit nicht kontinuierlich verfügbaren Prozessoren wurde erstmals von Lee [Lee97] aufgegriffen. In zahlreichen weiteren Artikeln wird dieses Problem seitdem untersucht (vgl. Schmidt [Sch00], Kapitel 5 und 6). Lee [Lee97] zeigt, daß das Problem $F2, NC \mid pmtn \mid C_{max}$ *NP*-schwer ist, selbst wenn nur ein Nichtverfügbarkeitsintervall auftritt. Dazu schränkt er das allgemeine Problem auf das Problem mit nur einem Nichtverfügbarkeitsintervall auf dem ersten Prozessor ein und zeigt, daß dieses Problem PARTITION als Spezialfall enthält. Darüberhinaus gibt er ein pseudopolynomiales *Dynamisches Programm* an, das das Problem mit nur einem Nichtverfügbarkeitsintervall optimal löst. Weiter beschreibt Lee eine Heuristik, die für das Problem mit nur einem

Nichtverfügbarkeitsintervall einen Plan erzeugt, der höchstens (3/2)-mal so lang ist wie ein optimaler Plan, wenn sich das Nichtverfügbarkeitsintervall auf dem ersten Prozessor befindet. Befindet sich das Nichtverfügbarkeitsintervall auf dem zweiten Prozessor, gibt Lee eine Heuristik an, die höchstens (4/3)-mal so lange Pläne erzeugt wie die von einem optimalen Verfahren erzeugten Pläne. Cheng und Wang [CW00] zeigen, daß die Schranke von (3/2) für die Situation, daß sich ein Nichtverfügbarkeitsintervall auf dem ersten Prozessor befindet, scharf ist. Für das Problem mit beliebig vielen Nichtverfügbarkeitsintervallen und den Spezialfall, daß ein Nichtverfügbarkeitsintervall auf einem Prozessor unmittelbar gefolgt wird von einem Nichtverfügbarkeitsintervall auf dem anderen Prozessor, geben Cheng und Wang in demselben Artikel eine Heuristik an, die das Problem mit einer *worst-case* Schranke von (5/3) löst.

Kubiak *et al.* [KBFBS02] zeigen, daß das Problem *NP*-schwer im strengen Sinne wird, wenn auf beiden Prozessoren beliebig viele Nichtverfügbarkeitsintervalle auftreten können, und geben einen *Branch-And-Bound*-Algorithmus an. Breit [Bre00] und Kubiak *et al.* [KBFBS01] untersuchen die Leistungsfähigkeit verschiedener Heuristiken zur Lösung des Problems $F2, NC \mid pmtn \mid C_{max}$(mit maximal zehn Nichtverfügbarkeitsintervallen). Resultat ist, daß innerhalb der Zeitschranke von 1000 Sekunden fast alle *einfachen* Probleminstanzen optimal gelöst werden, aber lediglich 41% der *schweren* Probleminstanzen.

Sotskov [Sot91] untersucht für ein Jobshop-Problem mit $m \geq 2$ Prozessoren den Einfluß möglicher Veränderungen der Bearbeitungszeiten der Aufträge auf die Optimalität eines Ablaufplans. Dabei bezeichnet er mit dem Begriff *Stabilitätsradius* die größtmöglichen Veränderungen der Bearbeitungszeiten der Aufträge, so daß ein gegebener optimaler Ablaufplan optimal bleibt. Lai und Sotskov [LS99] und Sotskova [Sot01] untersuchen Jobshop-Probleme, in denen u.a. die Bearbeitungszeiten der Verrichtungen zwischen einer unteren und einer oberen Schranke variieren können. Kravchenko *et al.* geben in [KSW95] notwendige und hinreichende Bedingungen dafür an, daß ein Ablaufplan für einen Jobshop einen unendlich großen Stabilitätsradius hat. Im gleichen Artikel wird auch gezeigt, daß es keine Pläne mit unendlich großem Stabilitätsradi-

us für das Flowshop-Problem mit mehr als zwei Prozessoren und mit mehr als zwei zu verplanenden Aufträgen geben kann. Sotskov *et al.* [SSW97] zeigen, daß optimale Ablaufpläne bezüglich hinreichend kleinen Veränderungen der Bearbeitungszeiten der Aufträge stabil sind.

Verwendete Notation

In dem vorliegenden Kapitel wird die in Tab. 3.1 zusammengefaßte Notation verwendet. Dabei wird i zur Indizierung der Prozessoren ($i \in \{1, 2\}$) und j zur Indizierung der Aufträge ($j \in \{1, \ldots, n\}$) benutzt.

$p_{i,j}$	Bearbeitungszeit von Verrichtung $T_{i,j}$ von Auftrag j auf Prozessor i
w_i	Anzahl der Nichtverfügbarkeitsintervalle auf Prozessor i
w	Gesamtanzahl der Nichtverfügbarkeitsintervalle: $w = w_1 + w_2$
$N_{i,k}$	k-tes Nichtverfügbarkeitsintervall auf Prozessor i
$s(N_{i,k})$	Startzeitpunkt des k-ten Nichtverfügbarkeitsintervalls auf Prozessor i
$f(N_{i,k})$	Endzeitpunkt des k-ten Nichtverfügbarkeitsintervalls auf Prozessor i
$h(N_{i,k})$	Länge des k-ten Nichtverfügbarkeitsintervalls auf Prozessor i, $h(N_{i,k}) := f(N_{i,k}) - s(N_{i,k})$
$r_{i,j}$	maximal mögliche Verlängerung der Bearbeitungszeit von Auftrag j auf Prozessor i, so daß eine Johnson-Permutation unverändert bleibt
ρ_i	Stabilitätsradius einer Johnson-Permutation auf Prozessor i
s_i	frühester Startzeitpunkt eines Auftrags auf Prozessor i
c_i	spätester Fertigstellungszeitpunkt eines Auftrags auf Prozessor i in einem semiaktiven Ablaufplan
$d_{i,j}$	wegen Nichtverfügbarkeitsintervallen maximal mögliche Verlängerung der Bearbeitungszeit von Auftrag j auf Prozessor i
δ_i	Verlängerungsradius der Bearbeitungszeiten auf Prozessor i
$s_{i,j}(\sigma)$	Startzeitpunkt der Verrichtung $T_{i,j}$ in dem durch die Permutation σ definierten semiaktiven Ablaufplan
$C_{i,j}(\sigma)$	Fertigstellungszeitpunkt der Verrichtung $T_{i,j}$ in dem durch die Permutation σ definierten semiaktiven Ablaufplan

Tabelle 3.1: *Notation (Flowshop-Problem)*

Für das Problem $F2, NC^{offline} \mid pmtn \mid C_{max}$ wird angenommen, daß alle Nichtverfügbarkeitsintervalle im voraus bekannt sind. Prozessor P_i ist nicht verfügbar vom Zeitpunkt $s(N_{i,k})$ bis zum Zeitpunkt $f(N_{i,k}) = s(N_{i,k}) + h(N_{i,k}),\ k \in \{1, 2, \ldots, w_i\}$. Verrichtung $T_{i,j}$, mit deren Bearbeitung vor dem Zeitpunkt $s(N_{i,k})$ begonnen wurde und deren Bearbeitung zum Zeitpunkt $s(N_{i,k})$ noch nicht abgeschlossen wurde, wird für $h(N_{i,k})$ Zeiteinheiten unterbrochen. Danach wird die Bearbeitung vom Zeitpunkt $f(N_{i,k}) = s(N_{i,k}) + h(N_{i,k})$ an wieder aufgenommen.

Im nächsten Abschnitt wird der Algorithmus von Johnson beschrieben, der ursprünglich für das Problem $F2 \mid\mid C_{max}$ formuliert wurde. Gonzales und Sahni [GS78] haben allerdings gezeigt, daß der Johnson-Algorithmus auch das Problem $F2 \mid pmtn \mid C_{max}$ optimal löst.

3.1.1 $F2 \mid pmtn \mid C_{max}$ (Algorithmus von Johnson)

Das Problem $F2 \mid pmtn \mid C_{max}$ gehört zu den klassischen Ablaufplanungsproblemen. Ein Algorithmus zur Lösung des Problems ist der von Johnson [Joh54] veröffentlichte Algorithmus 3.1, der ohne Präemptionen auskommt. Ist also die Anzahl der Nichtverfügbarkeitsintervalle $w = 0$, kann ein bezüglich der Planlänge optimaler Ablaufplan für das Problem $F2, NC \mid pmtn \mid C_{max}$ mit Algorithmus 3.1 wie folgt erstellt werden:

Algorithmus 3.1: *Algorithmus von Johnson*

begin

1. Teile die Menge der Aufträge in zwei Teilmengen wie folgt:
   S_1 := Menge der Aufträge mit $p_{1,j} \leq p_{2,j}$;
   S_2 := Menge der Aufträge mit $p_{1,j} > p_{2,j}$;
2. Verplane die Aufträge aus der Menge S_1 in nichtfallender (SPT) Reihenfolge ihrer Bearbeitungszeiten $p_{1,j}$;
3. Verplane die Aufträge aus der Menge S_2 in nichtsteigender (LPT) Reihenfolge ihrer Bearbeitungszeiten $p_{2,j}$;

end;

Die mit Algorithmus 3.1 umgeordnete Auftragsmenge wird als *Johnson-Permutation* oder auch *Johnson-Reihenfolge* σ bezeichnet. Satz 3.1 trifft eine Aussage über Laufzeit und Korrektheit des Algorithmus:

Satz 3.1 *Der Algorithmus von Johnson erzeugt für die Probleme* $F2 \mid pmtn \mid C_{max}$ *und* $F2 \mid\mid C_{max}$ *in Zeit* $O(n \log n)$ *einen zulässigen und bezüglich der Planlänge optimalen Ablaufplan.*

Beweis: Der Beweis für $F2 \mid\mid C_{max}$ erfolgt in [Joh54], *Theorem* 1 auf S. 63. Daß ein beliebiger präemptiver Plan zur Lösung des Problems $F2 \mid\mid C_{max}$ immer in einen nichtpräemptiven Plan mit der gleichen Planlänge transformiert werden kann, folgt aus [GS78], Lemma 3 auf S. 40ff. . □

Mit Präemptionen kann also keine Verbesserung bezüglich der Planlänge erreicht werden. Daher haben gleiche Instanzen der Probleme $F2 \mid pmtn \mid C_{max}$ und $F2 \mid\mid C_{max}$ die gleiche durch Algorithmus 3.1 ermittelte optimale Lösung. Folgendes Beispiel verdeutlicht die Vorgehensweise von Algorithmus 3.1.

Beispiel: Drei Aufträge J_1, J_2, J_3, die aus jeweils zwei Verrichtungen $T_{i,j}$ bestehen, sollen in einem Flowshop mit zwei Prozessoren P_1 und P_2 so verplant werden, daß die Planlänge minimiert wird. Die Verrichtungen haben folgende Bearbeitungszeiten (vgl. Tab. 3.2):

P_1	P_2
$T_{1,1}$ mit $p_{1,1} = 3$	$T_{2,1}$ mit $p_{2,1} = 5$
$T_{1,2}$ mit $p_{1,2} = 4$	$T_{2,2}$ mit $p_{2,2} = 1$
$T_{1,3}$ mit $p_{1,3} = 7$	$T_{2,3}$ mit $p_{2,3} = 2$

Tabelle 3.2: *Bearbeitungszeiten der Verrichtungen für das Beispielproblem*

In Abb. 3.3 ist ein von Algorithmus 3.1 ermittelter optimaler Ablaufplan, in dem die Aufträge in der Johnson-Reihenfolge $\sigma = (J_1, J_3, J_2)$ eingeplant werden, abgebildet. □

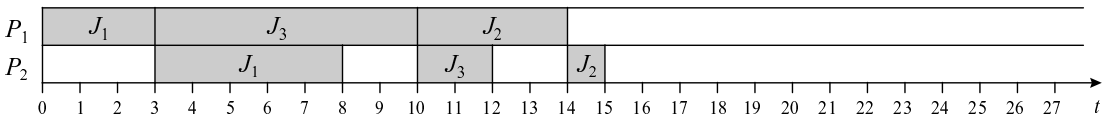

Abbildung 3.3: *Optimale Johnson-Permutation* σ *für das Beispielproblem*

Eine Johnson-Permutation σ ist optimal, wenn beide Prozessoren P_1 und P_2 während des Planungshorizonts kontinuierlich zur Verfügung stehen, d.h. für das Problem $F2 \mid pmtn \mid C_{max}$. In den beiden nächsten Abschnitten werden hinreichende Bedingungen dafür anzugeben, daß σ auch für das Problem $F2, NC^{offline} \mid pmtn \mid C_{max}$ optimal ist, d.h. für den Fall, daß die beiden Prozessoren in $w \geq 1$ Intervallen nicht verfügbar sind und ansonsten die gleichen Eingabedaten wie für das Problem $F2 \mid pmtn \mid C_{max}$ vorliegen. Das obige Beispielproblem wird dazu wie folgt erweitert:

Beispiel: Wir betrachten die in Tab. 3.2 beschriebene Probleminstanz mit einer Ausnahme: Bei ansonsten gleichen Eingabedaten treten nun auf beiden Prozessoren P_1 und P_2 Nichtverfügbarkeitsintervalle auf (vgl. Abb. 3.4). □

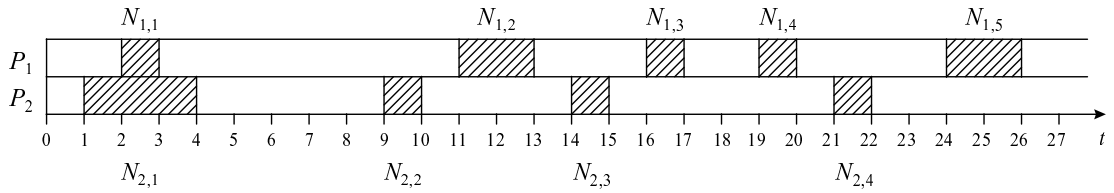

Abbildung 3.4: *Beispielproblem mit* w=9 *Nichtverfügbarkeitsintervallen* $N_{1,1}, \ldots, N_{1,5}$ *und* $N_{2,1}, \ldots, N_{2,4}$

In den folgenden Abschnitten wird gezeigt, wie die Werte $r_{i,j}, \rho_i, d_{i,j}, \delta_i, s_i$ und c_i (vgl. Tab. 3.1) für unser Beispiel berechnet werden können. Dabei wird sich herausstellen, daß der Ablaufplan, der die Aufträge in der Johnson-Reihenfolge $\sigma = (J_1, J_3, J_2)$ einplant, optimal bleibt, wenn die Prozessoren (wie abgebildet) nicht kontinuierlich zur Verfügung stehen.

3.1.2 Stabilitäts- und Verlängerungsradien

Die Idee für eine Stabilitätsanalyse einer Johnson-Permutation σ besteht im wesentlichen darin, ein Nichtverfügbarkeitsintervall $N_{i,k}$ auf Prozessor P_i als einen zusätzlichen Teil der Bearbeitungszeit der Verrichtung $T_{i,j}$ anzusehen, wenn die Bearbeitung von $T_{i,j}$ durch das Nichtverfügbarkeitsintervall $N_{i,k}$ unterbrochen wird. Zunächst wird gezeigt, wie der *Stabilitätsradius* ρ_i von Prozessor $i = 1, 2$ berechnet wird. ρ_i ist die kleinste

aller maximal möglichen Verlängerungen $r_{i,j}$ von Bearbeitungszeiten von Aufträgen j auf Prozessor i, so daß eine Johnson-Permutation gleich bleibt. Dann wird gezeigt, wie der *Verlängerungsradius* δ_i von Prozessor $i = 1, 2$ berechnet wird. δ_i ist die größte aller durch Nichtverfügbarkeitsintervalle verursachten maximal möglichen Verlängerungen $d_{i,j}$ von Bearbeitungszeiten von Aufträgen auf Prozessor i. Eine Johnson-Permutation σ bleibt dann optimal, wenn gilt:

$$d_{i,j} \leq r_{i,j}$$

für Aufträge $j = 1, 2, \ldots, n$ und Prozessoren $i = 1, 2$.

3.1.2.1 Stabilitätsradius für Johnson-Reihenfolge

Zur allgemeinen Definition des Begriffs *Stabilitätsradius* eines optimalen Ablaufplans sei auf Sotskov [Sot91] verwiesen. Mit *Stabilitätsradius* bezeichnet Sotskov die maximal möglichen Veränderungen der Bearbeitungszeiten der Verrichtungen, so daß ein gegebener optimaler Ablaufplan optimal bleibt. Die folgende Definition 3.1 unterscheidet sich davon in zwei Punkten:

1. Zum einen wird in Definition 3.1 angenommen, daß die Bearbeitungszeiten der Verrichtungen sich aufgrund von Nichtverfügbarkeitsintervallen nur erhöhen, aber nie verringern können.
2. Zum anderen wird in Definition 3.1 die Optimalität einer Permutation außer acht gelassen. Es geht nur darum, festzustellen, ob eine bestimmte Permutation nach Verlängerungen einzelner Bearbeitungszeiten noch eine Johnson-Permutation bleibt oder nicht. Zu beachten ist, daß die Optimalität einer Permutation σ nicht impliziert, daß σ eine Johnson-Permutation ist. Das gilt insbesondere auch für die Probleme $F2 \mid pmtn \mid C_{max}$ und $F2 \mid\mid C_{max}$.

Definition 3.1 *Der Stabilitätsradius ρ_i einer Johnson-Permutation σ auf Prozessor P_i ist wie folgt definiert: ρ_i ist das Minimum aller maximal möglichen Verlängerungen $r_{i,j}$ von Verrichtungen $T_{i,j}$ auf Prozessoren P_i, $i \in \{1, 2\}, j \in \{1, 2, \ldots, n\}$, so daß eine*

Permutation σ eine Johnson-Permutation für die modifizierten Bearbeitungszeiten ist. Die Polyeder

$$\Gamma_i := \{x = (x_{i,1}, x_{i,2}, \ldots, x_{i,n}) : p_{i,j} \leq x_{i,j} \leq p_{i,j} + r_{i,j}, i \in \{1,2\}, j \in \{1,2,\ldots,n\}\}$$

im Raum $\mathbb{R}^n$ von nichtnegativen, n-dimensionalen reellen Vektoren heißen Stabilitätspolyeder der Johnson-Permutation.

Stabilitätspolyeder Γ_i definieren alle maximalen Verlängerungen von Bearbeitungszeiten von Verrichtungen $T_{i,j}$ auf Prozessor P_i, so daß eine gegebene Permutation σ von n Aufträgen eine Johnson-Permutation bleibt. Lemma 3.1 trifft eine Aussage über die Laufzeit zur Berechnung der Stabilitätspolyeder und der Stabilitätsradien:

Lemma 3.1 *Die Stabilitätsradien ρ_i und die Stabilitätspolyeder $\Gamma_i, i \in \{1,2\}$, können mit einem Zeitaufwand von $O(n)$ berechnet werden.*

Beweis: Sei σ eine Johnson-Permutation von n Aufträgen. O.B.d.A. nehmen wir an, daß gilt: $\sigma = (J_1, J_2, \ldots, J_n)$. Weiter nehmen wir an, daß die ersten k Aufträge zur Menge S_1 gehören und daß die restlichen $n-k$ Aufträge zur Menge S_2 gehören. Es gilt also $S_1 = \{1,2,\ldots,k\}$ und $S_2 = \{k+1, k+2, \ldots, n\}$. Zur Berechnung von $r_{i,j}$ werden zwei Fälle unterschieden: Entweder gehört Auftrag j zur Menge S_1 oder zur Menge S_2.

Im Falle, daß j zur Menge S_1 gehört, gilt die Ungleichung $p_{1,j} \leq p_{2,j}$, und die maximal möglichen Verlängerungen $r_{1,j}$ sind jeweils das Minimum zweier Werte $a_{1,j}$ und $b_{1,j}$. $a_{1,j} := p_{2,j} - p_{1,j}$ stellt den maximalen Wert dar, den man zur Bearbeitungszeit $p_{1,j}$ addieren kann, so daß Auftrag j in der Menge S_1 verbleibt. Der Wert

$$b_{1,j} := \begin{cases} p_{1,j+1} - p_{1,j}, & j = 1,2,\ldots,k-1, \\ \infty, & j = k, \end{cases}$$

stellt den maximal möglichen Wert dar, den man zur Bearbeitungszeit $p_{2,j}$ hinzuaddieren kann, so daß die SPT-Reihenfolge innerhalb der Menge S_1 erhalten bleibt.

Ähnliche Argumente gelten für den Fall, daß j zur Menge S_2 gehört, d.h. wenn die Ungleichung $p_{1,j} > p_{2,j}$ erfüllt ist. Jede Berechnung eines Wertes $r_{i,j}$ kann in konstanter

Zeit erfolgen, so daß die Stabilitätspolyeder Γ_i und die Stabilitätsradien $\rho_i, i \in \{1,2\}$, in Zeit $O(n)$ berechnet werden können. □

Beispiel: In Tab. 3.3 sind die Resultate der Berechnungen für die in Tab. 3.2 und Abb. 3.4 beschriebene Instanz des Problems $F2, NC^{offline} \mid pmtn \mid C_{max}$ abgebildet.

P_1	
$p_{1,1} = 3$	$r_{1,1} = \min\{2, \infty\} = 2$
$p_{1,2} = 4$	$r_{1,2} = \infty$
$p_{1,3} = 7$	$r_{1,3} = \infty$
	$\rho_1 = 2$

P_2	
$p_{2,1} = 5$	$r_{2,1} = \infty$
$p_{2,2} = 1$	$r_{2,2} = \min\{3, 1\} = 1$
$p_{2,3} = 2$	$r_{2,3} = \min\{5, \infty\} = 5$
	$\rho_2 = 1$

Tabelle 3.3: *Stabilitätsradien*

Falls also keine Bearbeitungszeit einer von Prozessor P_1 zu bearbeitenden Verrichtung um mehr als zwei Zeiteinheiten und falls keine Bearbeitungszeit einer von Prozessor P_2 zu bearbeitenden Verrichtung um mehr als eine Zeiteinheit verlängert wird, bleibt die Permutation $\sigma = (J_1, J_3, J_2)$ eine Johnson-Permutation. □

3.1.2.2 Verlängerungsradius

Falls die Anzahl der Nichtverfügbarkeitsintervalle w=0 ist, werden alle Verrichtungen $T_{i,j}$ ohne Präemptionen bearbeitet und die Länge des Intervalls, in dem $T_{i,j}$ bearbeitet wird, ist gleich $p_{i,j}$. Falls $w \geq 1$, kann die *Belegungszeit*, d.h. die Differenz zwischen Bearbeitungsende und Startzeitpunkt einer Verrichtung $T_{i,j}$ durch ein oder mehrere Nichtverfügbarkeitsintervalle verlängert werden.

Beispiel: Die Belegungszeit der Verrichtungen $T_{1,1}$ und $T_{1,2}$ der Permutation $\sigma = (J_1, J_3, J_2)$ werden durch Nichtverfügbarkeitsintervalle verlängert (vgl. Abb. 3.5). □

Im folgenden wird gezeigt, wie die durch Nichtverfügbarkeitsintervalle maximal möglichen Verlängerungen der Belegungszeiten der Verrichtungen (unabhängig von ei-

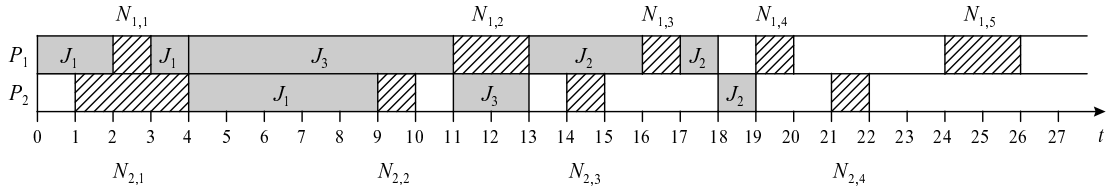

Abbildung 3.5: *Verlängerung von Belegungszeiten durch beschränkt verfügbare Prozessoren*

nem konkreten Ablaufplan) bestimmt werden können. Zunächst werden die Begriffe *Verlängerungsradius* und *Verlängerungspolytop* definiert.

Definition 3.2 *Der Verlängerungsradius δ_i der Verrichtungen auf Prozessor P_i ist wie folgt definiert: δ_i ist das Maximum aller möglichen Verlängerungen $d_{i,j}$ von Belegungszeiten von Verrichtungen $T_{i,j}$, die durch Nichtverfügbarkeitsintervalle auf ProzessorP_i, $i \in \{1,2\}, j \in \{1,\ldots,n\}$, verursacht werden können. Die Polytope*

$$\Delta_i = \{x = (x_{i,1}, x_{i,2}, \ldots, x_{i,n}) : p_{i,j} \leq x_{i,j} \leq p_{i,j} + d_{i,j}\}$$

im Raum $\mathbb{R}^n$ der nichtnegativen n-dimensionalen reellen Vektoren heißen Verlängerungspolytope der Verrichtungen.

Die Verlängerungsradien geben den maximalen Wert an, um den das Bearbeitungsende einer Verrichtung durch ein oder mehrere Nichtverfügbarkeitsintervalle verzögert werden kann. Lemma 3.2 trifft eine Aussage über die Laufzeit zur Berechnung der Verlängerungspolytope und der Verlängerungsradien:

Lemma 3.2 *Verlängerungsradien δ_i und Verlängerungspolytope Δ_i, $i \in \{1,2\}$, können in Zeit $O(w^2 + n \log n)$ berechnet werden.*

Beweis: Wir zeigen zunächst, wie δ_1 und Δ_1 berechnet werden können. Dazu berechnen wir die Summen D_l und E_l (für alle $l = 1,\ldots,w_1$) wie folgt:

Die Summe D_l der Längen von $1 \leq l \leq w_1$ aufeinanderfolgenden Nichtverfügbarkeitsintervallen auf Prozessor P_1 beträgt maximal

$$D_l = \max_{a=1}^{w_1+1-l} \left\{ \sum_{k=a}^{a+l-1} h(N_{1,k}) \right\}.$$

Die Summen E_l werden wie folgt berechnet: Zu den Nichtverfügbarkeitsintervallen werden jeweils die entsprechenden Verfügbarkeitsintervalle auf Prozessor P_1 bestimmt. Mit $A_{1,k}$ wird das k-te Verfügbarkeitsintervall auf Prozessor P_1 bezeichnet. Mit $s(A_{1,k})$, $f(A_{1,k})$ und $h(A_{1,k})$ werden Startzeitpunkt, Endzeitpunkt und Länge des Intervalls $A_{1,k}$ bezeichnet. O.B.d.A. nehmen wir an, daß $s(N_{1,1}) > 0$ ist. Wenn $s(N_{1,1}) = 0$ ist, können wir den frühest möglichen Startzeitpunkt $s_1 = 0$ auf Prozessor P_1 durch $s_1 = f(N_{1,1})$ ersetzen. Die optimale Permutation bleibt die gleiche. Daher gilt $s(A_{1,1}) = 0$, $f(A_{1,1}) = s(N_{1,1})$ und $h(A_{1,1}) = f(A_{1,1}) - s(A_{1,1}) > 0$. Für $k = 2, 3, \ldots, w_1$ gilt $s(A_{1,k}) = f(N_{1,k-1})$, $f(A_{1,k}) = s(N_{1,k})$ und $h(A_{1,k}) = f(A_{1,k}) - s(A_{1,k}) > 0$.

E_0 wird auf $E_0 := 0$ gesetzt und die Summe E_l der Längen von l aufeinanderfolgenden Verfügbarkeitsintervallen $A_{1,k}$ wird wie folgt berechnet:

$$E_l = \min_{a=1}^{w_1+1-l} \left\{ \sum_{k=a}^{a+l-1} h(A_{k,1}) \right\}.$$

Diese Berechnungen benötigen Zeit $O(w^2)$.

Mit Hilfe der Werte D_l und E_l für $l = 1, \ldots, w_1$ kann nun die maximal mögliche Verlängerung der Belegungszeit einer Verrichtung $T_{1,j}$ bestimmt werden. Um Verrichtung $T_{1,j}$ zu bearbeiten, werden höchstens k Verfügbarkeitsintervalle benutzt, wenn $E_{k-1} \leq p_{1,j} < E_k$. Diese k Verfügbarkeitsintervalle hängen mit k Nichtverfügbarkeitsintervallen zusammen. Daher wird die Belegungszeit von Verrichtung $T_{1,j}$ um höchstens D_k verlängert.

Ähnliche Argumente gelten für Prozessor P_2. Insgesamt wurde gezeigt, daß die Berechnung der Verlängerung jedes Auftrags Zeit $O(w + n \log n)$ kostet. Zur Berechnung der Verlängerungsradien δ_i und der Verlängerungspolytope Δ_i wird also insgesamt Zeit $O(w^2 + n \log n)$ benötigt. □

Beispiel: In Tab. 3.4 sind die Resultate der Berechnungen für die in Tab. 3.2 und Abb. 3.4 beschriebene Instanz des Problems $F2, NC^{offline} \mid pmtn \mid C_{max}$ abgebildet. Zu beachten ist, daß aus Berechnungen, die später in Lemma 3.3 beschrieben werden, folgt, daß die letztmöglichen Fertigstellungszeitpunkte eines Auftrags $c_1 = 18$ auf Prozessor P_1 und $c_2 = 27$ auf Prozessor P_2 betragen.

P_1	
$p_{1,1} = 3$	$d_{1,1} = 2$
$p_{1,2} = 4$	$d_{1,2} = 3$
$p_{1,3} = 7$	$d_{1,3} = 3$
	$\delta_1 = 3$

P_2	
$p_{2,1} = 5$	$d_{2,1} = 2$
$p_{2,2} = 1$	$d_{2,2} = 1$
$p_{2,3} = 2$	$d_{2,3} = 1$
	$\delta_2 = 2$

Tabelle 3.4: *Verlängerungsradien*

In Abb. 3.6 ist das Verlängerungspolytop Δ_1 der Verrichtungen auf Prozessor P_1 abgebildet.

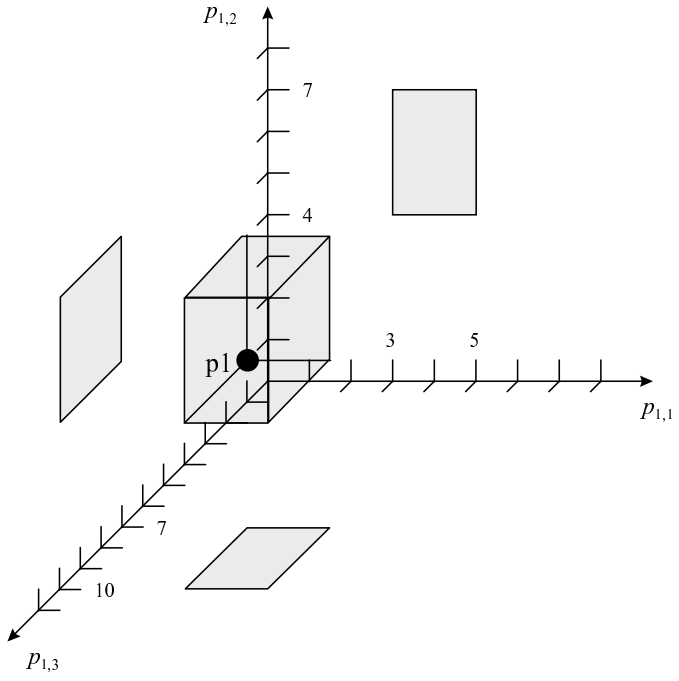

Abbildung 3.6: *Verlängerungspolytop der Verrichtungen auf Prozessor P_1*

Der Punkt p1 wird bestimmt durch die in Tab. 3.2 vorgegebenen Bearbeitungszeiten

$p_{1,j}$ der Aufträge $j \in \{1,2,3\}$ auf Prozessor P_1. Resultat der Berechnungen ist zusammenfassend, daß das Bearbeitungsende einer Verrichtung auf Prozessor P_1 (P_2) durch Nichtverfügbarkeitsintervalle um nicht mehr als um drei (zwei) Zeiteinheiten verzögert werden kann. □

Mit Hilfe von Lemma 3.1 und Lemma 3.2 wird nun folgender Satz 3.2 bewiesen:

Satz 3.2 *Eine für das Problem* $F2 \mid pmtn \mid C_{max}$ *definierte Johnson-Permutation* σ *bleibt optimal für das Problem* $F2, NC^{offline} \mid pmtn \mid C_{max}$, *falls*

$$d_{i,j} \leq r_{i,j}, \text{ für } i \in \{1,2\} \text{ und } j \in \{1,2,\ldots,n\}. \tag{3.1}$$

Der Test von Bedingung (3.1) kostet Zeit $O(w^2 + n \log n)$.

Beweis: Der Beweis erfolgt durch Widerspruch. Sei τ eine Permutation der n Aufträge mit

$$C_{max}(\tau) < C_{max}(\sigma) \tag{3.2}$$

für das Problem $F2, NC^{offline} \mid pmtn \mid C_{max}$. Wir definieren eine Modifikation $(P*)$ des Problems mit den folgenden Bearbeitungszeiten

$$\hat{p}_{i,j} := \begin{cases} C_{i,j}(\tau) - s_{i,j}(\tau) + h(N_{i,k}), & \text{falls ein } N_{kj} \text{ existiert mit } f(N_{k,j}) = s_{i,j}(\tau), \\ C_{i,j}(\tau) - s_{i,j}(\tau), & \text{sonst.} \end{cases}$$

Aus Definition 3.2 folgt, daß gilt: $\hat{p}_{i,j} \leq p_{i,j} + d_{i,j}$. Aus Definition 3.1 und den Ungleichungen (3.1) folgt, daß die Permutation σ eine Johnson-Permutation für Problem $(P*)$ bleibt. Solange aber für Problem $(P*)$ die Anzahl der Nichtverfügbarkeitsintervalle $w = 0$ ist, bleibt die Johnson-Permutation σ optimal. Daher gilt $C_{max}(\sigma) \leq C_{max}(\tau)$, was ein Widerspruch zu Ungleichung (3.2) ist. Der Zeitaufwand zur Berechnung der Werte $r_{i,j}$ und $d_{i,j}$ aus Lemma 3.1 und Lemma 3.2 bestimmt den Zeitaufwand, der erforderlich ist, um die Ungleichungen (3.1) zu testen. □

Beispiel: Durch Vergleichen der Einträge der Tabellen 3.3 und 3.4 sieht man, daß die Ungleichungen (3.1) von Satz 3.2 alle erfüllt sind. Die für die Probleminstanz des Problems $F2 \mid pmtn \mid C_{max}$ optimale Johnson-Permutation $\sigma = (J_1, J_3, J_2)$ löst also auch die Probleminstanz des Problemes $F2, NC^{offline} \mid pmtn \mid C_{max}$ optimal, wenn die Prozessoren (wie in Abb. 3.4 angegeben) lediglich beschränkt verfügbar sind. □

Im nächsten Abschnitt werden weitere hinreichende Bedingungen dafür angegeben, daß eine Johnson-Permutation auch im Falle beschränkt verfügbarer Prozessoren optimal bleibt.

3.1.3 Hinreichende Bedingungen für die Stabilität einer Johnson-Permutation

In Satz 3.3 werden weitere hinreichende Bedingungen für die Stabilität einer Johnson-Permutation angegeben. Die Bedingungen basieren auf Informationen über die Lage der Nichtverfügbarkeitsintervalle und der eingeplanten Verrichtungen in einem konkreten Ablaufplan.

Satz 3.3 *Sei I_1 eine Probleminstanz des Problems $F2 \mid pmtn \mid C_{max}$ und I_2 eine Probleminstanz des Problems $F2, NC \mid pmtn \mid C_{max}$ mit den gleichen Bearbeitungszeiten wie I_1. Sei σ eine zur Lösung von I_1 erstellte Johnson-Permutation und $s(\sigma)$ ein Ablaufplan für I_2, in dem die Aufträge in der durch σ vorgegebenen Reihenfolge eingeplant werden. Sei ferner $C_{i,j_k}(s(\sigma))$ der Fertigstellungszeitpunkt von Verrichtung $T_{i,j_k}, 1 \leq k \leq n$ im Ablaufplan $s(\sigma)$. Dann wird I_2 optimal durch σ gelöst, wenn es einen Zeitpunkt $t = C_{1,j_k}(s(\sigma))$, $1 \leq k \leq n$, im Ablaufplan $s(\sigma)$ gibt, so daß*

(i) die k kleinsten Verrichtungen auf Prozessor P_1 und Nichtverfügbarkeitsintervalle auf Prozessor P_1 das Intervall $[0, t]$ komplett ausfüllen,

(ii) die $n + 1 - k$ kleinsten Verrichtungen auf Prozessor P_2 und Nichtverfügbarkeitsintervalle auf Prozessor P_2 das Intervall $[t, C_{2,j_n}(s(\sigma))]$ komplett ausfüllen.

Beweis: Im Ablaufplan $s(\sigma)$ gibt es keine Leerzeiten auf Prozessor P_1 im Intervall $[0, t]$ und auf Prozessor P_2 im Intervall $[t, C_{2,j_k}(s(\sigma))]$. Daher kann der Wert $C_{\max}(s(\sigma))$ nicht durch ein Vertauschen der Aufträge innerhalb der Mengen $j_1, \ldots, j_k$ und $j_k, \ldots, j_n$ verringert werden.

In einem beliebigen semiaktiven Ablaufplan s gibt es auf Prozessor P_1 keine Leerzeiten innerhalb des Intervalls $[0, C_{1,j_n}]$. Daher füllen die Verrichtungen $T_{1,t}, k+1 \leq t \leq n$, und Nichtverfügbarkeitsintervalle auf Prozessor P_1 das Intervall $[t, C_{1,i_n}(s(\sigma))]$ komplett aus. Aus Bedingung (ii) folgt, daß im Ablaufplan s auf Prozessor P_2 lediglich im Intervall $[0, t]$ Leerzeiten vorhanden sein können. Sei $l(s(\sigma))$ die Gesamtlänge dieser Leerzeiten. Wenn $l(s(\sigma)) = 0$ ist, dann stellt der Ablaufplan $s(\sigma)$ eine optimale Lösung für das Problem $F2, NC^{offline} \mid pmtn \mid C_{max}$ dar. Wenn $l(s(\sigma)) > 0$ gilt, dann führt eine Verringerung von $C_{\max}(s(\sigma)) = C_{2,j_n}(s(\sigma))$ zu einer Verringerung von $l(s(\sigma))$.

Wir betrachten jetzt einen semiaktiven Ablaufplan $s(\sigma')$, der aus dem Ablaufplan $s(\sigma)$ durch einen Vertauschung eines Auftrags $j_y, 1 \leq y \leq k$ mit einem Auftrag $j_z, k+1 \leq z \leq n$ erstellt wird. Aus Bedingung (i) folgt, daß Prozessor $P_1(P_2)$ im Intervall $[0, t]$ höchstens $k(k-1)$ Verrichtungen vollständig bearbeiten kann. Aus Bedingung (ii) folgt, daß im Ablaufplan $s(\sigma')$ die gesamte Leerzeit auf Prozessor P_2 im Intervall $[0, t]$ wenigstens $l(s(\sigma))$ ist. Daher gilt: $C_{\max}(s(\sigma)) \leq C_{\max}(s(\sigma'))$. Durch ein Vertauschen von Aufträgen kann also keine Verbesserung der Planlänge erzielt werden. □

Satz 3.3 gilt für beliebige Permutationen σ. Insbesondere ist es nicht notwendig, daß σ eine Johnson-Permutation ist. Das gleiche gilt für Korollar 3.1 und 3.2:

Korollar 3.1 *Wenn die kleinste Verrichtung auf Prozessor P_2 und Nichtverfügbarkeitsintervalle auf P_2 das Intervall $[c_{1,i_n}(s(\sigma)), c_{2,i_n}(s(\sigma))]$ komplett ausfüllen, dann stellt der Ablaufplan $s(\sigma)$ eine optimale Lösung für das Problem $F2, NC^{offline} \mid pmtn \mid C_{max}$ dar.*

Korollar 3.2 *Wenn $p_{1,i_1} := \min\{p_{1,i_k}$ für $1 \leq k \leq n\}$ und auf Prozessor P_2 im Intervall $[c_{1,i_1}(s(\sigma)), c_{2,i_n}(s(\sigma))]$ keine Leerzeiten auftreten, dann stellt der Ablaufplan $s(\sigma)$ eine optimale Lösung für das Problem $F2, NC^{offline} \mid pmtn \mid C_{max}$ dar.*

Zum Beweis von Korollar 3.1 (Korollar 3.2) ist die Beobachtung hinreichend, daß Bedingungen (i) und (ii) auch für $k = n$ (bzw. $k = 1$) gelten. Der Test der Bedingung in Korollar 3.1 benötigt Zeit $O(w_2)$, der Test der Bedingung in Korollar 3.2 benötigt Zeit $O(w_2 + n)$.

Beispiel: In Abb. 3.5 sieht man, daß für die durch Tab. 3.2 und Abb. 3.4 beschriebene Instanz des Problems $F2, NC^{offline} \mid pmtn \mid C_{max}$ sowohl Satz 3.3 als auch Korollar 3.1 erfüllt sind. □

Die Anzahl der in Satz 3.2 und Satz 3.3 zu betrachtenden Nichtverfügbarkeitsintervalle kann wie folgt eingeschränkt werden:

Lemma 3.3 *Sei $s(\sigma)$ ein semiaktiver Ablaufplan zur Lösung einer Instanz des Problems $F2, NC \mid pmtn \mid C_{max}$. Der frühest mögliche Startzeitpunkt s_i und der spätest mögliche Fertigstellungszeitpunkt c_i eines Auftrags auf Prozessor P_i in $s(\sigma)$ können in Zeit $O(n + w)$ bestimmt werden.*

Beweis: Auf Prozessor P_1 ist der frühest mögliche Startzeitpunkt eines Auftrags

$$s_1 := \max \begin{cases} f(N_{1,1}), & \text{wenn } s(N_{1,1}) = 0, \\ 0, & \text{sonst.} \end{cases}$$

Der spätest mögliche Fertigstellungszeitpunkt eines Auftrags auf Prozessor P_1 ist $c_1 = p_1 + a + b$. p_1, a und b können wie folgt bestimmt werden: $p_1 := \sum p_{1,j}$ und $a := \sum_{i=1}^{k} h(N_{1,j})$ mit $s(N_{1,k}) < p_1$ für k maximal. b wird zunächst auf $b := 0$ gesetzt, k (maximal) wird aus der vorhergehenden Berechnung übernommen. b wird um $h(N_{1,k+1})$ erhöht und k wird um 1 erhöht, solange $s(N_{1,k+1}) < p_1 + a + b$. Die Berechnung von p_1 benötigt $O(n)$ Zeit, die Berechnung von a und b benötigt $O(w)$ Zeit. Insgesamt kann also der spätest mögliche Fertigstellungszeitpunkt eines Auftrags auf Prozessor P_1 in Zeit $O(n + w)$ bestimmt werden.

Auf Prozessor P_2 ist der frühest mögliche Startzeitpunkt eines Auftrags das Bearbeitungsende der ersten Verrichtung auf P_1. Sei $p_{min}(1)$ die kleinste Bearbeitungszeit einer Verrichtung, die von Prozessor P_1 zu bearbeiten ist und sei $d := p_{min}(1) + c$. Die Werte c und d werden zu Beginn auf 0 gesetzt. Solange $s(N_{1,k+1}) < p_{min}(1) + c$ ist, werden c um $h(N_{1,k+1})$ und k um 1 erhöht. Dann gilt:

$$s_2 = \max \begin{cases} f(N_{2,k}), & \text{wenn es ein Nichtverfügbarkeitsintervall } N_{2,k} \text{ gibt} \\ & \text{mit } s_{2,k} \leq d \text{ und } c_{2,k} \geq d, \\ d, & \text{sonst.} \end{cases}$$

Die obige Berechnung besitzt einen Zeitaufwand in Höhe von $O(n + w)$.

Der spätest mögliche Fertigstellungszeitpunkt eines Auftrags auf Prozessor P_2 wird durch die Schranke $c_2 = c_1 + \sum p_{2,j} + d$ nach oben beschränkt. Der Wert k ist der größte Index mit $s(N_{2,k}) \leq c_1$. Der Wert d wird zunächst auf $\max\{f(N_{2,k}) - c_1, 0\}$ gesetzt. Solange $s(N_{2,k+1}) < c_1 + \sum p_{2,j} + d$ ist, werden d um $h(2, N_{k+1})$ und k um 1 erhöht. Diese Berechnung erfordert ebenfalls einen Zeitaufwand in Höhe von $O(n+w)$. □

Beispiel: Abb. 3.7 zeigt die Ergebnisse der beschriebenen Berechnungen für die in Tab. 3.2 und Abb. 3.4 beschriebene Instanz des Problems $F2, NC^{offline} \mid pmtn \mid C_{max}$.

□

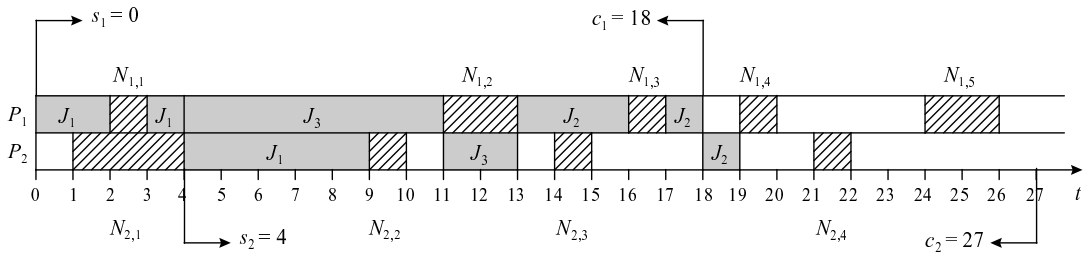

Abbildung 3.7: *Frühest mögliche Startzeitpunkte und spätest mögliche Endzeitpunkte von Verrichtungen auf P_1 und P_2*

Der im nächsten Abschnitt formulierte Algorithmus basiert auf den in diesem und dem vorigen Abschnitt erzielten Ergebnissen.

3.1.4 Experimentelle Ergebnisse

Die im letzten Abschnitt beschriebene Stabilitätsanalyse wird in Algorithmus 3.2 formalisiert. Um die Laufzeit so gering wie möglich zu halten, werden die verschiedenen Bedingungen in (in der Komplexität) aufsteigender Reihenfolge getestet.

Algorithmus 3.2: *Stability Test*

Input:
I_1 := Instanz des Problems $F2 \mid\mid C_{max}$;
I_2 := Instanz des Problems $F2, NC \mid pmtn \mid C_{max}$(bis auf die Nichtverfügbarkeitsintervalle gleiche Instanz wie I_1);
begin
1. Berechne eine Johnson-Permutation σ für das Problem $F2 \mid\mid C_{max}$;
2. **if** (σ erfüllt Korollar 3.1) **then output** „Permutation σ ist optimal für die Instanz I_2 des Problems $F2, NC \mid pmtn \mid C_{max}$“; **stop;**
3. **if** (σ erfüllt Korollar 3.2) **then then output** „Permutation σ ist optimal für die Instanz I_2 des Problems $F2, NC \mid pmtn \mid C_{max}$“; **stop;**
4. Erzeuge $\lambda \leq \min\{\lambda^*, 2^k\}$ Johnson-Permutationen $\sigma_1, \sigma_2, \ldots, \sigma_\lambda$;
5. **for** ($i := 1, \ldots, \lambda$) **do**
6. **begin**
7. Setze $\sigma := \sigma_i$;
8. **if** (σ erfüllt Korollar 3.1) **then output** „Permutation σ ist optimal für die Instanz I_2 des Problems $F2, NC \mid pmtn \mid C_{max}$“; **stop;**
9. **if** (σ erfüllt Korollar 3.2) **then output** „Permutation σ ist optimal für die Instanz I_2 des Problems $F2, NC \mid pmtn \mid C_{max}$“; **stop;**
10. **if** (σ erfüllt Satz 3.3) **then output** „Permutation σ ist optimal für die Instanz I_2 des Problems $F2, NC \mid pmtn \mid C_{max}$“; **stop;**
11. **if** (σ erfüllt Satz 3.2) **then output** „Permutation σ ist optimal für die Instanz I_2 des Problems $F2, NC \mid pmtn \mid C_{max}$“; **stop;**
12. **end;**
13. **output** „Optimalität einer der Permutationen $\sigma_i, i = 1, 2, \ldots, \lambda$, zur Lösung der Instanz I_2 des Problems $F2, NC^{offline} \mid pmtn \mid C_{max}$ konnte nicht bewiesen werden“;

end;

Zeilen 5ff. werden durchgeführt, da es mehr als eine optimale Johnson-Reihenfolge für eine Instanz des Problems $F2 \mid\mid C_{max}$ geben kann. Genauer gilt: Wenn k Bearbeitungszeiten auf einem Prozessor gleich sind, kann es 2^k verschiedene Johnson-Permutationen geben. In der Implementierung wurde die maximale Anzahl der betrachteten Johnson-

Permutationen auf $\min\{\lambda^* := 1024, 2^k\}$ Johnson-Permutationen beschränkt.

Die Ergebnisse von Lemma 3.3 wurden bei der Erzeugung der Probleminstanzen wie folgt eingesetzt: Die Nichtverfügbarkeitsintervalle wurden so ausgewählt, daß tatsächlich jedes der jeweils w Intervalle gezählt wurde. Wenn also c_1 und c_2 die Bearbeitungsenden der letzten (im Ablaufplan mit den Nichtverfügbarkeitsintervallen) auf Prozessor P_1 und P_2 verplanten Verrichtung sind, wurde jedes Nichtverfügbarkeitsintervall entweder im Intervall $[0, c_1]$ auf Prozessor P_1 oder im Intervall $[\min(p_1, j), c_2]$ auf Prozessor P_2 plaziert.

Der Algorithmus wurde in der Programmiersprache C implementiert. Für die Tests stand ein Rechner mit einem AMD 1200 MHz Prozessor und mit 1024 MB Hauptspeicher zur Verfügung.

Die experimentelle Performance-Analyse von Algorithmus 3.2 wurde für zwei Testsettings durchgeführt. Das erste Testsetting bestand aus Probleminstanzen, die (bis auf die Bearbeitungs- und Nichtverfügbarkeitszeiten, die jeweils gleichverteilt aus dem Intervall $[1, 1000]$ stammen, aber nicht notwendigerweise identisch sind) ähnlich den von Kubiak *et al.* [KBFBS02] betrachteten Probleminstanzen waren. Da sich herausgestellt hat, daß Algorithmus 3.2 eine sehr geringe Laufzeit besitzt, wurden in einem weiteren Testsetting Probleminstanzen untersucht, die weitaus mehr Aufträge und Nichtverfügbarkeitsintervalle beinhalten. Die Ergebnisse der Experimente werden entsprechend in den beiden folgenden Abschnitten *Kleines Testsetting* und *Großes Testsetting* beschrieben.

3.1.4.1 Kleines Testsetting

Zunächst wurden Probleminstanzen aus einem ähnlich dem von Kubiak *et al.* [KBFBS02] betrachteten Testsetting untersucht, d.h. mit

- $5, 10, 15, \ldots, 100$ Aufträgen mit ganzzahligen Bearbeitungszeiten, die uniform im Intervall $[1, 1000]$ verteilt sind und mit
- $1, 2, 3, \ldots, 10$ Nichtverfügbarkeitsintervallen auf beiden Prozessoren mit ganzzahligen Bearbeitungszeiten, die uniform im Intervall $[1, 1000]$ verteilt sind.

In Kubiak *et al.* wird ein *Branch-And-Bound*-Algorithmus entwickelt und zur Erzeugung optimaler Ablaufpläne für Instanzen des Problems $F2, NC^{offline} \mid pmtn \mid C_{max}$ eingesetzt. Wegen eines vorgegebenen Zeitlimits $T := 1000$ Sekunden für jede Instanz konnten nicht alle Instanzen innerhalb des Zeitlimits gelöst werden.

In unseren Experimenten wurden für jedes Problem jeweils 10000 Probleminstanzen erzeugt (Kubiak *et al.* jeweils zehn). Für jede Probleminstanz wurde wenigstens eine Johnson-Permutation σ bestimmt, und es wurde mit Hilfe von Algorithmus 3.2 jeweils die Frage beantwortet, ob σ die (bis auf die Nichtverfügbarkeitsintervalle) gleiche Instanz des Problems $F2, NC^{offline} \mid pmtn \mid C_{max}$ optimal löst. Dabei wurden lediglich die in den vorigen Abschnitten beschriebenen hinreichenden Bedingungen getestet. Insbesondere garantiert Algorithmus 3.2 keine optimale Lösung. Unsere Ergebnisse zeigen jedoch, daß für die meisten Probleminstanzen die Johnson-Permutation optimal bleibt.

Die Prozentzahlen in den nachfolgenden Tabellen geben an, für wieviel Prozent aller erzeugten Probleminstanzen durch Algorithmus 3.2 nachgewiesen werden konnte, daß eine Johnson-Permutation die jeweilige Instanz des Problemes $F2, NC^{offline} \mid pmtn \mid C_{max}$ optimal löst (Prozentzahl optimal gelöster Instanzen). Die Laufzeiten sind nicht aufgeführt, da die Laufzeiten alle äußerst gering waren: Die *maximale* Laufzeit über alle Probleminstanzen der Tabellen 3.5-3.9 betrug 0.000125 Sekunden und die aller Probleminstanzen der Tabellen 3.10-3.12 betrug 0.000731 Sekunden.

Tab. 3.5 zeigt die Ergebnisse für die zu Beginn dieses Abschnitts beschriebenen Probleme aus dem *kleinen Testsetting* und entspricht dem Testsetting aus Kubiak *et al.*. Die Prozentzahl der gelösten Instanzen war kleiner für n=5, n=10 und n=15, aber fast gleich für $20 \leq n \leq 100$. Der *Branch-And-Bound*-Algorithmus hat allerdings zum Teil eine wesentlich größere Laufzeit benötigt (1000 Sekunden waren nicht hinreichend, um einige Instanzen zu lösen).

Kubiak *et al.* zeigen, daß Instanzen mit Werten $p_{2,j} = 2p_{1,j}$ wesentlich schwieriger als Instanzen mit für beide Prozessoren uniform erzeugten Bearbeitungszeiten zu lösen sind. Unsere Berechnungen haben diese Vermutung für kleine Probleminstanzen bestätigt

Anzahl der Aufträge	Anzahl der Nichtverfügbarkeitsintervalle									
	1	2	3	4	5	6	7	8	9	10
5	74.2%	72.1%	72.3%	72.7%	72.3%	73.4%	74.1%	74.6%	75.1%	76.2%
10	87.5%	85.0%	84.6%	83.8%	83.5%	83.4%	83.5%	83.1%	84.4%	83.5%
15	91.1%	90.7%	90.0%	89.9%	89.6%	89.0%	88.7%	87.9%	88.5%	88.5%
20	93.4%	93.2%	92.5%	92.3%	92.1%	91.9%	91.7%	91.3%	90.8%	91.3%
25	94.5%	94.1%	94.3%	93.6%	93.3%	93.7%	93.8%	93.0%	93.2%	92.4%
30	95.1%	94.9%	95.0%	95.1%	94.7%	94.5%	93.9%	94.2%	94.0%	94.4%
35	95.5%	95.7%	95.5%	95.9%	95.4%	95.3%	95.5%	95.4%	95.1%	94.9%
40	96.0%	96.2%	96.1%	96.3%	96.1%	96.1%	95.9%	95.8%	95.5%	95.6%
45	96.6%	96.9%	96.6%	96.4%	96.2%	96.5%	96.3%	96.2%	96.1%	96.1%
50	97.2%	97.0%	96.8%	96.6%	96.7%	96.7%	96.9%	96.6%	96.9%	96.4%
55	97.2%	97.2%	97.2%	97.5%	96.8%	97.1%	97.2%	97.1%	96.8%	96.8%
60	97.7%	97.4%	97.6%	97.2%	97.3%	97.3%	97.2%	97.1%	97.2%	97.1%
65	97.6%	97.4%	97.7%	97.7%	97.3%	97.3%	97.3%	97.2%	97.3%	97.4%
70	97.8%	97.5%	97.9%	98.0%	97.5%	97.9%	97.7%	97.2%	97.8%	97.5%
75	97.7%	98.1%	97.9%	98.2%	97.8%	97.7%	97.8%	97.6%	97.8%	97.9%
80	98.1%	98.0%	98.1%	97.9%	98.1%	98.2%	98.0%	97.9%	97.6%	97.9%
85	98.1%	98.3%	98.2%	98.2%	98.3%	98.0%	98.2%	98.1%	98.0%	98.0%
90	98.5%	98.5%	98.3%	98.2%	98.3%	97.9%	98.1%	98.3%	98.2%	98.0%
95	98.3%	98.3%	98.4%	98.2%	98.1%	98.3%	98.2%	98.3%	98.2%	98.2%
100	98.5%	98.5%	98.5%	98.5%	98.4%	98.4%	98.3%	98.3%	98.2%	98.3%

Tabelle 3.5: *Prozentzahlen der gelösten Instanzen mit* $w_1 > 0$ *und* $w_2 > 0$

(vgl. Tabelle 3.5 mit 3.6). Aus einem Vergleich von Tabelle 3.6 mit Tabelle 3 aus Kubiak *et al.* folgt, daß unser Ansatz (mit Ausnahme von $n = 5, 10$ und $w \leq 7$, in denen der *Branch-And-Bound*-Algorithmus jeweils alle zehn Probleminstanzen optimal löst, eine ähnliche Prozentzahl erzielt.

Anzahl der Aufträge	Anzahl der Nichtverfügbarkeitsintervalle									
	1	2	3	4	5	6	7	8	9	10
5	64.5%	50.6%	44.4%	42.1%	40.0%	38.2%	37.7%	37.8%	36.5%	36.0%
10	66.6%	54.2%	47.4%	42.5%	40.3%	36.8%	35.6%	34.6%	33.9%	33.1%
15	68.0%	55.0%	49.2%	45.0%	42.2%	39.1%	37.5%	35.9%	35.6%	34.1%
20	69.4%	56.7%	50.8%	46.7%	43.9%	41.6%	39.9%	38.6%	36.6%	35.2%
25	68.4%	58.2%	52.2%	47.8%	45.8%	42.2%	40.2%	39.0%	37.4%	37.5%
30	68.8%	58.5%	52.7%	48.9%	45.4%	44.2%	42.4%	40.9%	39.8%	38.0%
35	68.6%	58.7%	53.4%	50.0%	46.2%	44.9%	43.0%	41.7%	40.3%	39.4%
40	69.7%	58.2%	52.6%	49.1%	47.2%	45.1%	43.0%	41.1%	41.8%	39.6%
45	69.9%	59.2%	53.2%	50.0%	47.6%	44.9%	43.7%	42.3%	41.2%	41.1%
50	70.3%	60.1%	54.5%	49.5%	47.6%	47.0%	44.5%	44.8%	43.3%	41.0%
55	68.9%	59.1%	54.1%	50.5%	48.0%	46.5%	46.8%	43.6%	42.1%	42.1%
60	70.0%	59.5%	53.8%	50.6%	49.2%	47.4%	46.0%	43.8%	43.7%	42.2%
65	70.4%	60.0%	53.7%	51.5%	49.1%	47.7%	46.5%	44.8%	43.4%	42.7%
70	70.1%	60.4%	54.7%	50.9%	49.5%	48.2%	47.7%	45.5%	43.6%	42.6%
75	68.8%	60.7%	54.9%	51.5%	49.5%	48.4%	46.1%	45.5%	44.3%	43.7%
80	70.0%	60.3%	54.2%	51.5%	50.3%	48.3%	47.0%	45.0%	44.4%	44.6%
85	70.8%	60.5%	55.1%	52.1%	50.4%	48.2%	46.6%	45.5%	44.7%	45.0%
90	70.2%	60.8%	55.4%	52.3%	50.9%	48.0%	47.2%	46.7%	45.5%	43.9%
95	71.2%	60.0%	55.5%	52.5%	51.0%	47.7%	47.5%	46.5%	45.9%	44.6%
100	70.4%	60.8%	57.2%	53.1%	50.8%	48.6%	47.1%	46.9%	45.9%	45.7%

Tabelle 3.6: *Prozentzahlen der gelösten Instanzen mit* $p_{2,j} = 2p_{1,j}$, $w_1 > 0$ *und* $w_2 > 0$

Für Bearbeitungs- und Nichtverfügbarkeitszeiten aus dem Intervall $[0, 100]$ wurden vergleichbare Ergebnisse erzielt (vgl. Tab. 3.7 und Tab. 3.8).

Anzahl der Aufträge	Anzahl der Nichtverfügbarkeitsintervalle									
	1	2	3	4	5	6	7	8	9	10
5%	75.5%	72.7%	72.3%	72.4%	73.5%	74.5%	75.2%	75.4%	76.6%	76.4%
10%	88.0%	85.9%	85.1%	85.1%	83.9%	84.6%	84.7%	84.8%	84.3%	84.7%
15%	92.0%	91.2%	90.6%	90.6%	89.9%	89.7%	89.8%	89.3%	88.9%	89.0%
20%	94.1%	93.7%	93.8%	92.7%	92.9%	93.0%	92.5%	92.2%	92.7%	92.0%
25%	95.0%	94.5%	95.0%	94.7%	94.5%	94.3%	93.8%	93.9%	94.1%	93.8%
30%	96.1%	96.0%	95.9%	95.5%	95.2%	95.2%	95.3%	94.6%	94.8%	94.9%
35%	96.5%	96.4%	96.5%	96.2%	96.4%	96.0%	96.0%	96.0%	95.8%	95.8%
40%	97.2%	97.0%	96.8%	96.7%	96.9%	96.5%	96.6%	96.3%	96.7%	96.3%
45%	97.5%	97.1%	97.4%	97.1%	97.2%	97.3%	97.2%	96.4%	96.7%	96.8%
50%	97.8%	97.6%	97.8%	97.6%	97.4%	97.4%	97.5%	97.1%	97.1%	97.3%
55%	98.1%	97.9%	97.8%	98.0%	97.8%	97.6%	97.7%	97.4%	97.4%	97.6%
60%	98.3%	98.2%	97.9%	97.6%	97.8%	97.9%	97.8%	97.8%	97.8%	97.6%
65%	98.4%	98.4%	98.2%	98.3%	98.0%	98.0%	97.7%	98.0%	97.9%	98.0%
70%	98.3%	98.4%	98.3%	98.2%	98.6%	98.3%	98.4%	98.2%	98.4%	98.2%
75%	98.5%	98.7%	98.4%	98.4%	98.6%	98.4%	98.4%	98.4%	98.3%	98.5%
80%	98.4%	98.8%	98.8%	98.5%	98.7%	98.7%	98.3%	98.5%	98.4%	98.4%
85%	98.6%	98.7%	98.6%	98.6%	98.7%	98.9%	98.8%	98.6%	98.7%	98.4%
90%	98.8%	98.9%	98.9%	98.6%	98.8%	98.9%	98.8%	98.7%	98.7%	98.8%
95%	99.0%	98.9%	98.9%	99.0%	98.9%	98.9%	98.8%	98.9%	98.9%	99.0%
100%	99.0%	98.9%	98.8%	98.8%	98.9%	98.9%	98.7%	99.0%	98.8%	98.8%

Tabelle 3.7: *Prozentzahlen der gelösten Instanzen mit* $w_1 > 0$ *und* $w_2 > 0$, *Werte aus dem Intervall* $[1, 100]$

Anzahl der Aufträge	Anzahl der Nichtverfügbarkeitsintervalle									
	1	2	3	4	5	6	7	8	9	10
5%	64.9%	51.5%	45.8%	43.2%	40.2%	38.8%	38.9%	37.8%	37.2%	36.4%
10%	67.6%	55.6%	48.4%	44.3%	41.4%	39.2%	38.7%	36.6%	35.5%	34.8%
15%	69.7%	58.3%	51.8%	49.0%	45.4%	43.5%	40.8%	39.8%	37.7%	37.5%
20%	71.0%	60.9%	55.0%	51.4%	48.7%	46.0%	44.4%	42.3%	41.0%	39.5%
25%	72.7%	62.4%	57.4%	53.4%	50.2%	48.1%	47.0%	45.1%	44.3%	43.0%
30%	74.5%	64.4%	58.9%	55.4%	52.7%	50.0%	49.6%	47.9%	47.5%	45.3%
35%	74.8%	66.8%	61.9%	57.6%	55.8%	53.6%	52.4%	51.1%	49.4%	47.5%
40%	75.9%	68.3%	64.1%	59.6%	57.7%	55.8%	55.4%	53.5%	51.9%	50.2%
45%	76.5%	68.9%	66.1%	62.3%	59.4%	58.0%	56.9%	56.1%	55.2%	52.6%
50%	78.2%	71.2%	66.7%	64.5%	62.0%	59.8%	60.0%	57.5%	57.0%	56.2%
55%	78.9%	73.2%	68.5%	66.2%	63.0%	62.6%	60.8%	60.8%	59.5%	58.3%
60%	81.2%	74.4%	70.5%	68.5%	66.3%	64.4%	63.3%	62.0%	61.6%	60.5%
65%	82.2%	75.6%	72.3%	70.1%	68.1%	66.9%	65.8%	64.1%	63.2%	61.5%
70%	83.3%	77.4%	74.7%	71.2%	69.6%	69.5%	67.1%	66.4%	65.1%	65.6%
75%	85.0%	78.6%	75.5%	73.7%	71.9%	70.5%	69.7%	68.7%	67.7%	67.3%
80%	85.0%	79.9%	77.0%	75.0%	74.0%	72.4%	70.2%	70.2%	69.6%	68.6%
85%	86.4%	81.2%	78.6%	76.4%	75.3%	73.5%	73.0%	72.5%	70.6%	70.7%
90%	86.9%	82.2%	79.9%	78.3%	77.4%	75.6%	74.6%	74.1%	72.8%	72.3%
95%	88.1%	83.9%	81.3%	79.5%	78.6%	77.2%	76.3%	75.0%	74.7%	74.6%
100%	89.2%	84.4%	82.5%	80.6%	79.1%	78.4%	77.4%	76.8%	76.6%	75.9%

Tabelle 3.8: *Prozentzahlen der gelösten Instanzen mit* $p_{2,j} = 2p_{1,j}$, $w_1 > 0$ *und* $w_2 > 0$, *Werte aus dem Intervall* $[1, 100]$

Neben dem Fall, daß auf beiden Prozessoren Nichtverfügbarkeitsintervalle auftreten ($w_1 > 0$ und $w_2 > 0$), wurden die Fälle getestet, in denen auf einem der beiden Prozessoren keine Nichtverfügbarkeitsintervalle vorkommen (w_1=0 oder w_2=0). Theoretisch scheint der Fall mit $w_1 = 0$ schwieriger als der Fall $w_2 = 0$, da Nichtverfügbarkeitsintervalle auf P_1 Leerzeiten auf P_2 hervorrufen können (während das Gegenteil nicht der Fall ist). Die von uns durchgeführten Tests (vgl. Tab. 3.9 mit $w = w_1$ und Tab. 3.10 mit $w = w_2$) haben allerdings ergeben, daß die beiden Probleme etwa gleich schwer sind. Nur für $n = 5, 10$ wurden im Fall $w = w_1$ schlechtere Ergebnisse als im Fall $w = w_2$ erzielt.

Anzahl der Aufträge	Anzahl der Nichtverfügbarkeitsintervalle									
	1	2	3	4	5	6	7	8	9	10
5	71.4%	67.4%	67.0%	69.0%	69.6%	71.0%	72.5%	73.7%	73.8%	76.4%
10	86.2%	83.3%	82.2%	80.9%	81.2%	81.0%	81.2%	82.6%	81.6%	82.2%
15	91.2%	89.7%	89.5%	88.1%	87.7%	87.6%	87.1%	87.1%	88.0%	88.1%
20	93.0%	92.9%	92.0%	91.7%	90.6%	90.7%	90.5%	90.0%	90.5%	90.2%
25	94.1%	93.8%	93.8%	93.1%	92.8%	92.8%	92.5%	92.2%	92.4%	92.1%
30	95.1%	94.9%	94.2%	95.2%	94.7%	94.1%	94.2%	93.8%	93.9%	93.5%
35	96.1%	95.8%	95.5%	95.3%	94.9%	94.4%	95.1%	94.7%	94.3%	94.3%
40	96.5%	96.1%	95.8%	95.8%	95.9%	95.5%	95.2%	95.6%	95.1%	95.2%
45	97.0%	96.6%	96.4%	96.5%	96.1%	96.2%	95.7%	95.9%	95.9%	95.9%
50	96.9%	96.6%	96.4%	96.7%	96.9%	96.7%	96.5%	96.5%	96.5%	96.2%
55	97.4%	97.2%	96.7%	96.9%	96.9%	97.1%	97.0%	97.0%	96.8%	96.6%
60	97.5%	97.7%	97.2%	97.0%	97.3%	97.0%	97.1%	96.8%	97.1%	97.0%
65	97.9%	97.5%	97.5%	97.6%	97.6%	97.3%	97.4%	97.3%	97.2%	97.2%
70	97.5%	97.7%	98.0%	97.9%	97.8%	97.5%	97.6%	97.1%	97.5%	97.8%
75	97.9%	98.0%	97.9%	97.8%	97.7%	97.8%	97.8%	97.6%	97.5%	97.7%
80	98.2%	98.4%	98.0%	98.0%	98.0%	97.6%	97.8%	97.5%	97.8%	97.7%
85	98.2%	98.2%	98.3%	97.8%	98.0%	98.1%	98.1%	97.7%	97.9%	97.8%
90	98.3%	98.1%	98.4%	98.0%	98.2%	98.3%	98.2%	98.0%	98.2%	97.7%
95	98.5%	98.4%	98.4%	98.2%	98.3%	98.3%	98.1%	97.9%	98.3%	98.1%
100	98.4%	98.3%	98.3%	98.4%	98.5%	98.4%	98.4%	98.2%	98.1%	98.3%

Tabelle 3.9: *Prozentzahlen der gelösten Instanzen mit* $w = w_1$

Die schwierigsten Probleme (mit den meisten negativen Antworten) für Algorithmus 3.2 stellte die Kombination der beiden oben genannten Schwierigkeiten dar. Tab. 3.11 zeigt die Ergebnisse für Probleme mit $w = w_1$ und $p_{2,j} = 2p_{1,j}$. Dabei wurden die schlechtesten Ergebnisse für eine kleine Anzahl (n=5, n=10, n=15 und n=20) von Aufträgen und eine große Anzahl von Nichtverfügbarkeitsintervallen erzielt (vgl. rechte obere Ecke in Tab. 3.11). Zur Lösung von Probleminstanzen mit $p_{2,j} = 2p_{1,j}$ und $w = w_1 > n$ ist Algorithmus 3.2 also schlecht geeignet.

Anzahl der Aufträge	Anzahl der Nichtverfügbarkeitsintervalle									
	1	2	3	4	5	6	7	8	9	10
5	76.7%	75.0%	74.5%	75.2%	76.6%	77.3%	77.9%	77.6%	79.6%	80.3%
10	87.9%	86.4%	85.2%	84.4%	83.1%	83.7%	85.0%	84.5%	85.4%	85.6%
15	91.7%	91.1%	90.4%	88.9%	89.5%	88.9%	89.2%	88.9%	89.2%	88.9%
20	93.6%	92.6%	92.9%	92.2%	92.5%	91.5%	91.7%	91.0%	90.4%	90.5%
25	94.5%	94.5%	94.3%	93.5%	93.7%	92.9%	92.6%	93.2%	92.7%	92.6%
30	95.3%	95.0%	94.6%	94.5%	94.0%	94.1%	94.4%	94.0%	94.0%	93.7%
35	96.0%	95.8%	95.9%	95.7%	95.3%	95.4%	94.8%	94.5%	94.7%	94.8%
40	96.3%	96.4%	95.9%	95.8%	95.8%	96.2%	95.5%	96.0%	95.8%	95.4%
45	96.5%	96.8%	96.5%	96.5%	96.0%	96.4%	96.2%	96.0%	95.7%	96.0%
50	97.2%	96.8%	96.8%	96.7%	96.6%	96.8%	96.8%	96.5%	96.4%	96.0%
55	97.1%	97.4%	97.3%	97.2%	97.0%	96.9%	96.5%	96.8%	96.6%	96.8%
60	97.4%	97.4%	97.5%	97.5%	96.9%	97.1%	97.0%	96.9%	97.0%	96.8%
65	97.5%	97.7%	97.5%	97.5%	97.4%	97.3%	97.2%	97.4%	97.5%	96.8%
70	98.0%	98.0%	97.9%	97.7%	98.0%	97.7%	97.6%	97.4%	97.8%	97.5%
75	98.1%	98.0%	97.8%	97.7%	97.7%	97.8%	97.8%	97.8%	97.8%	97.9%
80	98.2%	98.2%	98.2%	98.0%	98.1%	97.9%	97.9%	97.8%	97.6%	97.6%
85	98.1%	98.3%	98.2%	98.2%	98.0%	97.8%	98.1%	98.1%	97.9%	98.1%
90	98.4%	98.2%	98.3%	98.1%	98.3%	98.2%	98.0%	98.1%	98.2%	98.0%
95	98.6%	98.4%	98.3%	98.4%	98.1%	98.2%	98.3%	98.2%	98.0%	97.8%
100	98.3%	98.3%	98.7%	98.5%	98.4%	98.2%	98.2%	98.2%	98.3%	98.2%

Tabelle 3.10: *Prozentzahlen der gelösten Instanzen mit* $w = w_2$

Anzahl der Aufträge	Anzahl der Nichtverfügbarkeitsintervalle									
	1	2	3	4	5	6	7	8	9	10
5	27.4%	16.2%	9.5%	5.0%	3.0%	1.4%	0.7%	0.5%	0.2%	0.1%
10	33.0%	26.1%	20.8%	15.3%	11.4%	7.5%	5.3%	3.0%	1.9%	0.8%
15	35.0%	31.6%	27.8%	23.3%	19.6%	16.1%	13.8%	10.7%	8.2%	6.1%
20	36.2%	34.0%	29.9%	27.1%	26.6%	23.0%	20.3%	17.6%	15.2%	13.3%
25	36.9%	35.4%	32.5%	30.3%	28.2%	25.9%	24.2%	21.9%	19.6%	17.9%
30	38.4%	35.8%	33.7%	32.9%	30.2%	29.0%	26.6%	25.6%	23.8%	22.6%
35	38.4%	35.9%	35.6%	32.9%	31.9%	30.7%	30.0%	27.1%	26.2%	24.8%
40	38.9%	37.5%	35.6%	33.9%	33.0%	31.8%	30.7%	29.7%	28.7%	26.8%
45	38.9%	38.0%	36.1%	36.1%	34.0%	32.9%	32.9%	30.5%	29.9%	27.5%
50	39.5%	38.4%	36.5%	36.3%	35.1%	34.4%	33.1%	31.8%	30.7%	30.1%
55	39.5%	38.7%	37.5%	36.8%	36.0%	34.7%	33.8%	33.2%	33.1%	30.7%
60	40.7%	38.6%	37.9%	37.3%	36.8%	34.8%	33.8%	34.1%	32.8%	31.7%
65	40.7%	39.6%	38.8%	37.4%	36.9%	36.6%	35.7%	34.2%	33.3%	32.6%
70	39.8%	40.6%	39.2%	38.0%	37.7%	36.5%	36.3%	34.4%	34.6%	33.7%
75	41.2%	39.7%	39.3%	38.0%	38.4%	36.7%	36.6%	35.6%	35.0%	34.2%
80	40.6%	39.9%	40.4%	39.2%	38.7%	37.5%	36.6%	36.1%	36.1%	35.9%
85	41.5%	40.1%	40.2%	39.4%	38.0%	38.7%	37.4%	36.6%	35.5%	34.6%
90	41.2%	39.6%	40.4%	39.5%	39.4%	38.4%	37.2%	37.4%	37.1%	37.0%
95	41.6%	40.6%	40.2%	40.2%	38.9%	38.8%	37.9%	38.3%	35.8%	36.6%
100	41.3%	40.6%	40.0%	39.3%	39.3%	39.3%	38.2%	38.5%	37.8%	36.5%

Tabelle 3.11: *Prozentzahlen der gelösten Instanzen mit* $p_{2,j} = 2p_{1,j}$ *und* $w = w_1$

Die besten Resultate wurde für Probleme mit $w = w_2$ und $p_{2,j} = 2p_{1,j}$ erzielt: Alle zwei Millionen Probleminstanzen wurden optimal gelöst (vgl. Tab. 3.12).

Anzahl der Aufträge	Anzahl der Nichtverfügbarkeitsintervalle									
	1	2	3	4	5	6	7	8	9	10
5	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
10	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
15	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
20	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
25	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
30	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
35	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
40	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
45	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
50	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
55	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
60	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
65	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
70	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
75	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
80	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
85	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
90	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
95	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
100	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

Tabelle 3.12: *Prozentzahlen der gelösten Instanzen mit* $p_{2,j} = 2p_{1,j}$ *und* $w = w_2$

3.1.4.2 Großes Testsetting

Aufgrund der sehr geringen Laufzeit von Algorithmus 3.2 wurde ein Testsetting mit größeren Probleminstanzen (als die im vorigen Abschnitt betrachteten) formuliert:

- $1000, 2000, \ldots, 10000$ Aufträge mit ganzzahligen Bearbeitungszeiten, die uniform aus dem Intervall $[1, 1000]$ verteilt sind,
- $10, 100, 500, 1000$ Nichtverfügbarkeitsintervallen mit ganzzahligen Längen, die uniform im Intervall $[1, 1000]$ verteilt sind.

Die Ergebnisse der Experimente sind in Tab. 3.13-3.18 angegeben. Je größer die Anzahl der Aufträge und die Anzahl der Nichtverfügbarkeitsintervalle ist, desto größer ist die Anzahl der gelösten Probleminstanzen.

Anzahl der Aufträge	Anzahl der Nichtverfügbarkeitsintervalle			
	10	100	500	1000
1000	99.9% (0.008 s)	100.0% (0.008 s)	99.9% (0.010 s)	99.8% (0.027 s)
2000	100.0% (0.029 s)	100.0% (0.029 s)	100.0% (0.031 s)	100.0% (0.037 s)
3000	100.0% (0.065 s)	100.0% (0.065 s)	99.8% (0.101 s)	100.0% (0.073 s)
4000	100.0% (0.116 s)	100.0% (0.116 s)	100.0% (0.118 s)	100.0% (0.123 s)
5000	100.0% (0.180 s)	100.0% (0.180 s)	100.0% (0.182 s)	100.0% (0.188 s)
6000	100.0% (0.259 s)	100.0% (0.259 s)	100.0% (0.261 s)	100.0% (0.266 s)
7000	100.0% (0.352 s)	100.0% (0.352 s)	100.0% (0.354 s)	100.0% (0.359 s)
8000	100.0% (0.459 s)	100.0% (0.459 s)	100.0% (0.461 s)	100.0% (0.466 s)
9000	100.0% (0.580 s)	100.0% (0.580 s)	100.0% (0.582 s)	100.0% (0.588 s)
10000	100.0% (0.715 s)	100.0% (0.716 s)	100.0% (0.717 s)	100.0% (0.723 s)

Tabelle 3.13: *Durchschnittliche Prozentzahlen und Laufzeiten der gelösten Instanzen mit* $w_1 > 0$ *und* $w_2 > 0$

Anzahl der Aufträge	Anzahl der Nichtverfügbarkeitsintervalle			
	10	100	500	1000
1000	100.0% (0.008 s)	99.9% (0.008 s)	99.3% (0.020 s)	99.8% (0.073 s)
2000	100.0% (0.029 s)	100.0% (0.029 s)	100.0% (0.032 s)	100.0% (0.041 s)
3000	99.9% (0.067 s)	100.0% (0.066 s)	100.0% (0.068 s)	100.0% (0.077 s)
4000	100.0% (0.116 s)	100.0% (0.116 s)	100.0% (0.119 s)	100.0% (0.127 s)
5000	100.0% (0.180 s)	99.9% (0.187 s)	100.0% (0.183 s)	100.0% (0.191 s)
6000	100.0% (0.259 s)	100.0% (0.259 s)	100.0% (0.262 s)	100.0% (0.270 s)
7000	100.0% (0.352 s)	100.0% (0.352 s)	100.0% (0.354 s)	100.0% (0.363 s)
8000	100.0% (0.459 s)	100.0% (0.459 s)	100.0% (0.461 s)	100.0% (0.470 s)
9000	100.0% (0.580 s)	100.0% (0.580 s)	100.0% (0.583 s)	100.0% (0.591 s)
10000	100.0% (0.716 s)	100.0% (0.716 s)	100.0% (0.719 s)	100.0% (0.727 s)

Tabelle 3.14: *Durchschnittliche Prozentzahlen und Laufzeiten der gelösten Instanzen mit* $w = w_2$

Anzahl der Aufträge	Anzahl der Nichtverfügbarkeitsintervalle			
	10	100	500	1000
1000	99.8% (0.008 s)	99.9% (0.008 s)	99.7% (0.017 s)	100.0% (0.019 s)
2000	99.9% (0.030 s)	99.8% (0.031 s)	99.8% (0.041 s)	100.0% (0.041 s)
3000	99.9% (0.070 s)	100.0% (0.065 s)	100.0% (0.068 s)	100.0% (0.077 s)
4000	100.0% (0.116 s)	100.0% (0.116 s)	100.0% (0.118 s)	100.0% (0.127 s)
5000	100.0% (0.180 s)	100.0% (0.180 s)	100.0% (0.183 s)	100.0% (0.191 s)
6000	100.0% (0.259 s)	100.0% (0.259 s)	100.0% (0.262 s)	100.0% (0.270 s)
7000	100.0% (0.351 s)	100.0% (0.352 s)	100.0% (0.354 s)	100.0% (0.363 s)
8000	100.0% (0.459 s)	100.0% (0.459 s)	100.0% (0.461 s)	100.0% (0.470 s)
9000	100.0% (0.580 s)	100.0% (0.580 s)	100.0% (0.583 s)	100.0% (0.591 s)
10000	100.0% (0.715 s)	100.0% (0.716 s)	100.0% (0.719 s)	100.0% (0.727 s)

Tabelle 3.15: *Durchschnittliche Prozentzahlen und Laufzeiten der gelösten Instanzen mit* $w = w_1$

Anzahl der Aufträge	Anzahl der Nichtverfügbarkeitsintervalle			
	10	100	500	1000
1000	80.5% (0.032 s)	75.9% (0.036 s)	60.3% (0.175 s)	48.9% (1.329 s)
2000	95.4% (0.074 s)	92.3% (0.084 s)	85.1% (0.155 s)	73.5% (0.802 s)
3000	99.2% (0.137 s)	97.6% (0.147 s)	90.4% (0.233 s)	85.4% (0.602 s)
4000	99.3% (0.238 s)	98.9% (0.245 s)	93.4% (0.338 s)	88.0% (0.696 s)
5000	99.9% (0.359 s)	99.5% (0.368 s)	96.2% (0.454 s)	91.7% (0.752 s)
6000	100.0% (0.514 s)	99.6% (0.526 s)	95.6% (0.664 s)	91.3% (0.984 s)
7000	99.9% (0.703 s)	99.3% (0.728 s)	96.6% (0.853 s)	92.1% (1.221 s)
8000	100.0% (0.912 s)	99.6% (0.934 s)	97.1% (1.081 s)	93.4% (1.444 s)
9000	99.9% (1.168 s)	99.3% (1.210 s)	98.3% (1.286 s)	93.9% (1.740 s)
10000	100.0% (1.447 s)	99.7% (1.472 s)	97.4% (1.678 s)	93.0% (2.209 s)

Tabelle 3.16: *Durchschnittliche Prozentzahlen und Laufzeiten der gelösten Instanzen mit* $p_{2,j} = 2p_{1,j}$, $w_1 > 0$ *und* $w_2 > 0$

Anzahl der Aufträge	Anzahl der Nichtverfügbarkeitsintervalle			
	10	100	500	1000
1000	79.4% (0.032 s)	69.6% (0.042 s)	37.4% (0.457 s)	0.3% (4.968 s)
2000	95.0% (0.075 s)	89.7% (0.093 s)	69.5% (0.352 s)	46.6% (2.856 s)
3000	99.1% (0.136 s)	96.2% (0.158 s)	83.5% (0.360 s)	66.1% (2.054 s)
4000	99.4% (0.237 s)	98.6% (0.248 s)	86.5% (0.498 s)	72.8% (1.937 s)
5000	99.7% (0.363 s)	98.3% (0.393 s)	89.0% (0.661 s)	79.6% (1.797 s)
6000	99.8% (0.520 s)	97.8% (0.581 s)	90.8% (0.854 s)	83.2% (1.858 s)
7000	99.9% (0.703 s)	98.1% (0.778 s)	92.7% (1.049 s)	83.0% (2.246 s)
8000	99.8% (0.923 s)	99.3% (0.950 s)	92.8% (1.350 s)	87.8% (2.181 s)
9000	100.0% (1.161 s)	99.1% (1.224 s)	93.5% (1.653 s)	89.0% (2.469 s)
10000	99.9% (1.455 s)	99.1% (1.524 s)	94.5% (1.956 s)	89.3% (2.899 s)

Tabelle 3.17: *Durchschnittliche Prozentzahlen und Laufzeiten der gelösten Instanzen mit* $p_{2,j} = 2p_{1,j}$, $w = w_1$

Anzahl der Aufträge	Anzahl der Nichtverfügbarkeitsintervalle			
	10	100	500	1000
1000	100.0% (0.008 s)	99.9% (0.008 s)	99.3% (0.020 s)	99.8% (0.073 s)
2000	100.0% (0.029 s)	100.0% (0.029 s)	100.0% (0.032 s)	100.0% (0.041 s)
3000	99.9% (0.067 s)	100.0% (0.066 s)	100.0% (0.068 s)	100.0% (0.077 s)
4000	100.0% (0.116 s)	100.0% (0.116 s)	100.0% (0.119 s)	100.0% (0.127 s)
5000	100.0% (0.180 s)	99.9% (0.187 s)	100.0% (0.183 s)	100.0% (0.191 s)
6000	100.0% (0.259 s)	100.0% (0.259 s)	100.0% (0.262 s)	100.0% (0.270 s)
7000	100.0% (0.352 s)	100.0% (0.352 s)	100.0% (0.354 s)	100.0% (0.363 s)
8000	100.0% (0.459 s)	100.0% (0.459 s)	100.0% (0.461 s)	100.0% (0.470 s)
9000	100.0% (0.580 s)	100.0% (0.580 s)	100.0% (0.583 s)	100.0% (0.591 s)
10000	100.0% (0.716 s)	100.0% (0.716 s)	100.0% (0.719 s)	100.0% (0.727 s)

Tabelle 3.18: *Durchschnittliche Prozentzahlen und Laufzeiten der gelösten Instanzen mit* $p_{2,j} = 2p_{1,j}$, $w = w_2$

Für jedes der untersuchten Probleme hat sich die Anzahl der gelösten Probleminstanzen üblicherweise erhöht mit einer Abnahme der Differenz $n - w$. In den durchgeführten Experimenten hat sich die Schwierigkeit der betrachteten Probleme wie folgt dargestellt:

$$[(p_{2,j} = 2p_{1,j})\&(w = w_2)] \rightarrow [(w_1 > 0)\&(w_2 > 0)] \rightarrow [(w = w_2)] \rightarrow [(w = w_1)] \rightarrow$$

$$[(p_{2,j} = 2p_{1,j})\&(w_1 > 0)\&(w_2 > 0)] \rightarrow [(p_{2,j} = 2p_{1,j})\&(w = w_1)].$$

Die schlechtesten Ergebnisse wurden für Probleme mit $p_{2,j} = 2p_{1,j}$ und $w = w_1$ erzielt, die besten für Probleme mit $p_{2,j} = 2p_{1,j}$ und $w = w_2$.

3.2 $P, NC \mid pmtn \mid C_{max}$

Das Problem $P, NC \mid pmtn \mid C_{max}$ läßt sich wie folgt beschreiben: Es sollen n Aufträge $J_1, \ldots, J_n$ mit nichtnegativen Bearbeitungszeiten $p_1, \ldots, p_n$ auf m parallelen, identisch qualifizierten Prozessoren $P_1, \ldots, P_m$ so verplant werden, daß die Planlänge C_{max} minimiert wird. Wenn ein Auftrag unterbrochen wird, darf er zum gleichen oder zu einem späteren Zeitpunkt auf demselben oder auf einem anderen Prozessor ohne Zeitstrafe weiterbearbeitet werden. Die Prozessoren können Nichtverfügbarkeitsintervalle haben, die dem Ablaufplaner entweder *offline*, *nearly-online* oder erst *online* bekannt sind.

Verwandte Ergebnisse

Liegt ein *konstantes* Verfügbarkeitsmuster der Prozessoren vor, so läßt sich das Problem $P, NC^{offline}_{const} \mid pmtn \mid C_{max}$ optimal mit dem Algorithmus von McNaughton [McN59] lösen.

Kellerer [Kel98] gibt einen dualen Approximationsalgorithmus an, der das Problem $P, NC^{offline}_{inc} \mid pmtn \mid C_{max}$ mit einer Gütegarantie von $C^{LPT}_{max} \leq (5/4)C^{np^*}_{max}$ löst. Lee [Lee91] und Lee *et al.* [LHT00] beweisen, daß die *LPT-Regel* eine Gütegarantie von $C^{LPT}_{max} \leq (3/2 - 1/2m)C^{np^*}_{max}$ und daß eine modifizierte *MLPT-Regel* eine Gütegarantie von $C^{MLPT}_{max} \leq (4/3)C^{np^*}_{max}$ besitzen. He [He98] gibt für die *LPT-Regel* eine Gütegarantie von $C^{LPT}_{max} \leq (3/2 - 1/(2m-2))C^{np^*}_{max}$, für $k = 1$ und von $C^{LPT}_{max} \leq (3/2 - 1/2m)C^{np^*}_{max}$ für $k \geq 2$ an. Dabei bezeichnet er mit k die Anzahl der Aufträge auf dem Prozessor, auf dem der Auftrag, der die Planlänge bestimmt, verplant wird.

Lee [Lee96] untersucht für das Problem $P, NC^{offline}_{win} \mid pmtn \mid C_{max}$ den Spezialfall, daß jeder Prozessor nur ein Nichtverfügbarkeitsintervall hat und daß ein Prozessor während des gesamten Planungshorizonts verfügbar ist. Er nimmt weiter an, daß in dem Fall, daß ein Auftrag durch ein Nichtverfügbarkeitsintervall unterbrochen wird, derselbe Auftrag auf demselben Prozessor weiterbearbeitet werden muß, wenn dieser wieder verfügbar ist. Ein Auftrag darf also nur durch ein Nichtverfügbarkeitsintervall unterbrochen werden (*task preemption, t-pmtn*). Lee zeigt, daß die *LPT-Regel*, die

die Aufträge in nicht steigender Reihenfolge ihrer Bearbeitungsdauern auf den jeweils nächsten freien Prozessor verteilt, bezüglich der Planlänge beliebig schlechte Pläne erzeugen kann (vgl. Abb. 3.8, das Nichtverfügbarkeitsintervall auf P_2 ist schraffiert).

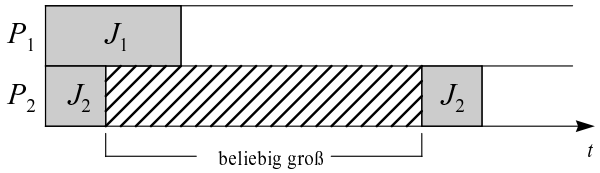

Abbildung 3.8: *LPT mit task-preemption erzeugt beliebig schlechte Pläne*

Modifiziert man die *LPT-Regel* allerdings so, daß der nächste Auftrag aus der nichtsteigend sortierten Liste der Aufträge so verplant wird, daß das Bearbeitungsende dieses Auftrags kleinstmöglich ist, erhält man als Gütegarantie $C_{max}^{LPT2} \leq (3/2 - 1/(2m))C_{max}^{t\text{-}pmtn^*}$, wobei C_{max}^{LPT2} die mit der modifizierten *LPT-Regel* erzielte Planlänge ist und $C_{max}^{t\text{-}pmtn^*}$ die unter den erwähnten Nebenbedingungen optimale Planlänge ist. Einen Überblick über weitere Ergebnisse mit beschränkt verfügbaren, parallelen, identischen Prozessoren gibt Schmidt [Sch00].

Verwendete Notation

Im weiteren wird die in Tab. 3.19 angeführte Notation verwendet. Dabei wird i zur Indizierung der Prozessoren ($i \in \{1, \ldots, m\}$) und j zur Indizierung der Aufträge ($j \in \{1, \ldots, n\}$) benutzt. $p_{max} := \max\{p_1, p_2, p_3, \ldots, p_n\}$ sei die maximale Bearbeitungszeit und $\sum p_j := \sum_{j=1}^{n} p_j$ die Summe der Bearbeitungszeiten der Aufträge. Ein Intervall $[B_i^r, F_i^r)$, in dem Prozessor P_i permanent verfügbar ist, heißt ein *Prozessorintervall*. Genauer ist $[B_i^r, F_i^r)$ das r-te Prozessorintervall von Prozessor P_i. Die Bearbeitungskapazität (*processing capacity*) des Intervalls $[B_i^r, F_i^r)$ wird mit PC_i^r bezeichnet und beträgt $PC_i^r := F_i^r - B_i^r$. $Q := \sum_{i=1}^{m} N(i)$ ist die Gesamtzahl an Prozessorintervallen, wobei $N(i)$ die Anzahl der Prozessorintervalle von Prozessor P_i darstellt. Es gilt $0 \leq r \leq N(i)$. PC_i ist die Bearbeitungskapazität von Prozessor P_i und beträgt $PC_i := \sum_{r=1}^{N(i)} PC_i^r$. Es wird angenommen, daß es höchstens $S \in \mathbb{N}$ Zeitpunkte

p_j	Bearbeitungszeit von Auftrag J_j
p_{max}	maximale Bearbeitungszeit eines Auftrags, $p_{max} := \max\{p_1, \ldots, p_n\}$
$\sum p_j$	Summe der Bearbeitungszeiten der Aufträge, $\sum p_j := \sum_{j=1}^{n} p_j$
B_i^r	Startzeitpunkt des r-ten Verfügbarkeitsintervalls auf Prozessor P_i
F_i^r	Endzeitpunkt des r-ten Verfügbarkeitsintervalls auf Prozessor P_i
$[B_i^r, F_i^r)$	r-tes Prozessorintervall (Verfügbarkeitsintervall) auf Prozessor P_i
PC_i^r	Bearbeitungskapazität des r-ten Prozessorintervalls auf Prozessor P_i, $PC_i^r := F_i^r - B_i^r$
$N(i)$	Gesamtanzahl der Prozessorintervalle auf Prozessor P_i
PC_i	Bearbeitungskapazität von Prozessor P_i, $PC_i := \sum_{r=1}^{N(i)} PC_r^i$
Q	Gesamtanzahl der Prozessorintervalle, $Q := \sum_{i=1}^{m} N(i)$
t_k	Zeitpunkt, zu dem sich die Menge der verfügbaren Prozessoren ändert
S_k	k-tes Systemintervall, $S_k := [t_k, t_{k+1})$
δ	Länge eines Systemintervalls S_k, $\delta := t_{k+1} - t_k$
S	Gesamtanzahl der Systemintervalle
m_k	Anzahl der im k-ten Systemintervall verfügbaren Prozessoren

Tabelle 3.19: *Notation (Problem mit parallelen, identisch qualifiz. Prozessoren)*

$t_1, t_2, \ldots, t_S$ gibt, zu denen sich die Menge der verfügbaren Prozessoren (das Prozessorsystem) ändert. Ein Intervall $[t_k, t_{k+1})$ heißt *Systemintervall*. Zu beachten ist, daß ein Systemintervall $[t_k, t_{k+1})$ nicht notwendigerweise einem Prozessorintervall $[B_i^r, F_i^r)$ entspricht. $\delta := t_{k+1} - t_k$ ist die Länge eines Systemintervalls, t_{S+1} wird zur Vereinfachung der Notation auf ∞ gesetzt. $m_k, 1 \leq k \leq S$, ist die Anzahl der verfügbaren Prozessoren im k-ten Systemintervall. m_k kann möglicherweise 0 sein, so daß in höchstens S Systemintervallen Prozessoren verfügbar sind.

Zur Bestimmung des Verhältnisses zwischen der Anzahl S der Systemintervalle und der Anzahl Q der Prozessorintervalle kann man folgende Überlegungen anstellen. Gegeben seien $Q = mh, h \geq 1$, Prozessorintervalle. Jedes Prozessorintervall hat einen Beginn- und einen Endzeitpunkt (eventuell ist der Endzeitpunkt ∞). Q Prozessorintervalle haben also $2Q$ Beginn- und Endzeitpunkte. **(a)** Wieviele Systemintervalle S können bei gegebener Anzahl Q von Prozessorintervallen höchstens auftreten? Je mehr der $2Q$ Beginn- und Endzeitpunkte der Prozessorintervalle verschieden sind, desto häufiger

ändert sich die Menge der verfügbaren Prozessoren und desto mehr Systemintervalle gibt es. Wenn alle $2Q$ Beginn- und Endzeitpunkte aller Prozessorintervalle verschieden sind, ändert sich die Menge der verfügbaren Prozessoren $2Q$-mal (vgl. Abb. 3.9).

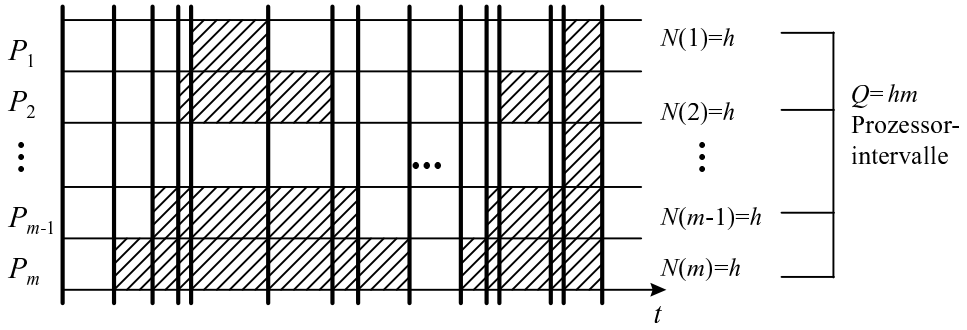

Abbildung 3.9: *Es gibt höchstens* $2Q+1$ *Systemintervalle* $(S-1 \leq 2Q)$

Wenn zu Beginn keine Prozessoren zur Verfügung stehen, gibt es ein zusätzliches Systemintervall. Also gibt es höchstens $S \leq 2Q+1$ Systemintervalle. **(b)** Wieviele Systemintervalle S können bei gegebener Anzahl Q von Prozessorintervallen wenigstens auftreten? Je weniger der $2Q$ Beginn- und Endzeitpunkte der Prozessorintervalle verschieden sind, desto weniger häufig ändert sich die Menge der verfügbaren Prozessoren, desto weniger Systemintervalle gibt es. Von den insgesamt $2Q$ Beginn- und Endzeitpunkten aller Prozessorintervalle können höchstens $2Q/m$ Beginn- und Endzeitpunkte gleich sein. In diesem Fall ändert sich die Menge der verfügbaren Prozessoren $(2Q/m)$-mal (vgl. Abb. 3.10). Wenn alle Endzeitpunkte der jeweils letzten Prozessorintervalle eines Prozessors ∞ sind, gibt es ein Systemintervall weniger. Also gibt es wenigstens $S \geq 2Q/m - 1$ Systemintervalle.

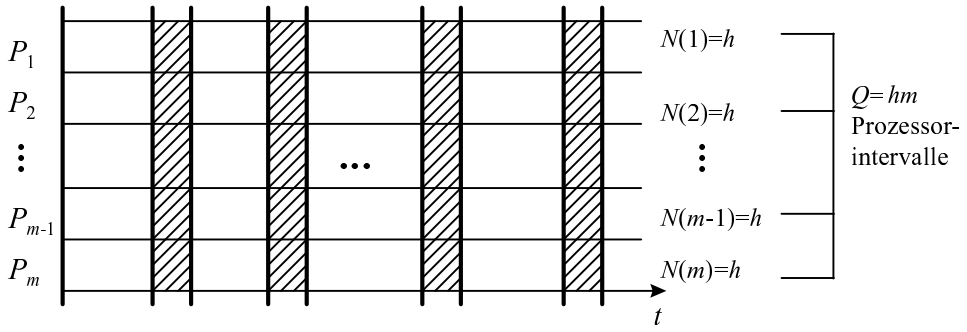

Abbildung 3.10: *Es gibt wenigstens* $2Q/m-1$ *Systemintervalle* $(2Q \leq (S+1)m)$

Die Überlegungen führen zu Ungleichung (3.3):

$$S - 1 \leq 2Q \leq (S + 1)m. \tag{3.3}$$

Sind die Verfügbarkeiten der Prozessoren beschränkt, so muß darüber Buch geführt werden, d.h. ein Algorithmus muß Informationen darüber erhalten, welche Prozessoren wann verfügbar sind. Im *offline setting* erhält ein Algorithmus die Verfügbarkeiten aller Prozessoren zu Beginn als Eingabe, kann also sein Vorgehen danach ausrichten. Anders ist der Fall im *online setting* und im *(nearly-)online setting*. Vereinbarungsgemäß erhält ein *Online*-Algorithmus seine Eingabe mit der Zeit und muß auch nach und nach seine Ausgabe erzeugen. Also erhält ein *Online*-Algorithmus immer dann, wenn sich Verfügbarkeiten von Prozessoren ändern, einen weiteren Teil der Gesamteingabe. Im *nearly-online setting* erhält der Algorithmus zusätzlich noch jeweils den Zeitpunkt als Eingabe, zu dem sich das nächste mal die Verfügbarkeiten der Prozessoren ändern. Eine mögliche Vorgehensweise zur Verwaltung der Eingabe wird im folgenden beschrieben. Die Verfügbarkeiten werden jeweils als Vektor (t_i, t_{i+1}, m_i, M_i) abgespeichert mit $m_i :=$ Anzahl der verfügbaren Prozessoren zum Zeitpunkt t_i und $M_i :=$ Menge der verfügbaren Prozessoren zum Zeitpunkt t_i. Beispielsweise bedeutet $(12, 17, 3, \{P_2, P_4, P_5\})$, daß die drei Prozessoren P_2, P_4 und P_5 vom Zeitpunkt 12 bis zum Zeitpunkt 17 zur Verfügung stehen. Mit Hilfe dieser Konstruktion braucht kein Algorithmus zusätzliche Informationen einzuholen. Immer wenn sich die Verfügbarkeit ändert, bekommt der Algorithmus einen neuen Vektor als Eingabe. Im *offline setting* erhält ein Algorithmus alle Vektoren gleich zu Beginn als Eingabe, im *nearly-online* und im *online setting* zu Beginn jedes Systemintervalls. Im *online setting* entfällt jeweils der nächste Zeitpunkt t_{i+1}.

Beispiel: Abb. 3.11 zeigt einen Ablauf für Updates der Prozessorverfügbarkeiten. Es gibt S=3 Zeitpunkte t_1=0, t_2=5 und t_3=10, zu denen sich die Menge der verfügbaren Prozessoren ändert und S=3 Systemintervalle $[0, 5), [5, 10)$ und $[10, \infty)$. Zum Zeitpunkt t_1=0 stehen die m_1=4 Prozessoren P_1, P_2, P_3, P_4 zur Verfügung, zum Zeitpunkt t_2=5 die m_2=2 Prozessoren P_1, P_2 und zum Zeitpunkt t_3=10 die m_3=4 Pro-

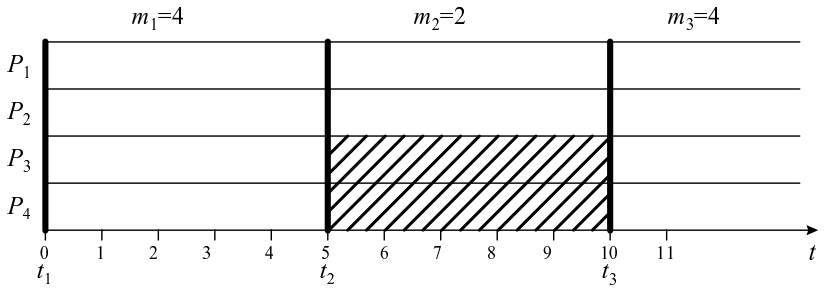

Abbildung 3.11: *Update der Prozessorverfügbarkeiten*

zessoren P_1, P_2, P_3, P_4. Der *Offline*-Algorithmus erhält die gesamte Eingabe vor dem Start in der Form (t_i, t_{i+1}, m_i, M_i), also $(0, 5, 4, \{P_1, P_2, P_3, P_4\})$, $(5, 10, 2, \{P_1, P_2\})$, $(10, \infty, \{P_1, P_2, P_3, P_4\})$. Der *Nearly-Online*-Algorithmus erhält zum Zeitpunkt t_i jeweils einen Vektor (t_i, t_{i+1}, m_i, M_i). Der *Online*-Algorithmus erhält zum Zeitpunkt t_i jeweils einen Vektor $(t_i, -, m_i, M_i)$. □

3.2.1 $P \mid pmtn \mid C_{max}$ (Algorithmus von McNaughton)

Das Problem $P \mid pmtn \mid C_{max}$ gehört zu den klassischen Ablaufplanungsproblemen. Ein Algorithmus zur Lösung des Problems wurde von McNaughton [McN59] veröffentlicht. Dieser erzeugt auch optimale Lösungen für das Problem $P, NC^{offline}_{const} \mid pmtn \mid C_{max}$, in dem die Prozessoren ein konstantes Verfügbarkeitsmuster besitzen. Zur Erzeugung eines optimalen präemptiven Ablaufplans benutzt McNaughton zwei untere Schranken bezüglich der optimalen präemptiven Planlänge $C^{p^*}_{max}$:

$$C^{p^*}_{max} \geq \frac{\sum_{j=1}^{n} p_j}{m} \tag{3.4}$$

$$C^{p^*}_{max} \geq p_j \text{ für alle Aufträge } j \tag{3.5}$$

Die beiden Schranken besagen, daß die Planlänge eines zulässigen Ablaufplans wenigstens so lang ist wie die durchschnittliche Last der Prozessoren (3.4) und wie die Länge eines beliebigen Auftrags (3.5).

Es gilt also

$$C_{max}^{p^*} = \max \left\{ \frac{\sum p_j}{m} , p_{max} \right\}.$$

Algorithmus 3.3 berechnet $C_{max}^{p^*}$ und verplant die Aufträge in beliebiger Reihenfolge mit der *Wrap-Around-Regel.*

Algorithmus 3.3: *Algorithmus von McNaughton*

begin

1. Berechne $C_{max}^{p^*} := \max \left\{ (\sum_{j=1}^{n} p_j)/m , p_{max} \right\}$ und setze $i := 1$;
2. Wähle einen beliebigen Auftrag und verplane den zum Zeitpunkt 0 auf P_i ;
3. Wähle einen beliebigen noch nicht verplanten Auftrag und verplane den so früh wie möglich auf P_i ;
4. Wiederhole den Schritt in Zeile 3 solange bis alle Aufträge verplant wurden oder bis die Last von P_i größer als $C_{max}^{p^*}$ ist;
5. Verplane die Last, die über den Wert $C_{max}^{p^*}$ hinausgeht, auf dem nächsten Prozessor P_{i+1} ;
6. Setze $i := i + 1$ und gehe zu Zeile 3;

end;

Im allgemeinen gibt es mehrere bezüglich der Planlänge optimale Ablaufpläne für eine Instanz des Problems $C_{max}^{p^*}$. Eine Aussage über Korrektheit, Laufzeit und Anzahl der Präemptionen, die von Algorithmus 3.3 im *worst-case* erzeugt werden, trifft Satz 3.4:

Satz 3.4 *Der Algorithmus von McNaughton erzeugt für das Problem* $P \mid pmtn \mid C_{max}$ *mit einem Zeitaufwand von* $O(n)$ *einen zulässigen und bezüglich der Planlänge optimalen Ablaufplan mit höchstens* $m - 1$ *Präemptionen.*

Beweis: Die Aufträge werden in beliebiger Reihenfolge verplant. Von einem Auftrag j werden entweder $0 < t \leq C_{max}^{p^*}$ Zeiteinheiten auf Prozessor P_i oder $0 < t \leq C_{max}^{p^*}$ Zeiteinheiten auf Prozessor P_i und $C_{max}^{p^*} - t$ Zeiteinheiten auf Prozessor P_{i+1} verplant. Es gibt höchstens m $C_{max}^{p^*}$ Zeiteinheiten zu verplanen. Daher wird jeder Auftrag verplant.

Da $C_{max}^{p^*} - t \geq p_j - t$ für jedes beliebige t, wird ein Auftrag zu jedem beliebigen Zeitpunkt auf höchstens einem Prozessor bearbeitet. Die Anzahl der unterbrochenen Aufträge ist demnach höchstens $m-1$. Die Berechnung von $C_{max}^{p^*}$ kostet Zeit $O(n)$. Danach werden die Aufträge der Reihe nach verplant, so daß sich insgesamt eine Laufzeit von $O(n)$ ergibt. □

In den nächsten Abschnitten werden die Fälle betrachtet, in denen die Prozessoren nicht kontinuierlich zur Verfügung stehen.

3.2.2 Offline

Schmidt [Sch84] beschreibt einen Algorithmus *Stair*, der einen zulässigen Plan für das Problem $P, NC_{sc}^{offline} \mid pmtn \mid C_{max}$ erzeugt, wenn die Prozessoren ein *Treppenmuster* bilden. Zur Lösung des Problems mit beliebigem Verfügbarkeitsmuster der Prozessoren kann Algorithmus 3.4 ausgeführt werden.

Algorithmus 3.4: *Offline*

1. Erzeuge aus dem vorgegebenen beliebigen Verfügbarkeitsmuster ein Treppenmuster (Algorithmus *Reorder*);
2. Bestimme $C_{max}^{p^*}$;
3. Bestimme die Prozessorkapazitäten PC_i für $i = 1, \ldots, m$;
4. Führe Algorithmus *Stair* aus;

Der Kern des Algorithmus ist Zeile 4, in dem der Algorithmus *Stair* ausgeführt wird (vgl. Algorithmus 3.5).

Stair verplant jeden Auftrag J_j auf höchstens zwei Prozessoren P_k und P_l mit Hilfe von fünf Regeln. P_k und P_l sind so gewählt, daß die Prozessorkapazität von P_k nicht kleiner ist als p_j und die Prozessorkapazität von P_l kleiner ist als p_j. ϕ_k^a, ϕ_k^b und ϕ_k^c stellen die Bearbeitungskapazität von Prozessor P_k in den Intervallen $[B_k^1, B_l^1]$, $[B_l^1, F_l^{N(l)}]$ und $[F_l^{N(l)}, F_k^{N(k)}]$ dar. Es gilt also $PC_k = \phi_k^a + \phi_k^b + \phi_k^c$.

Algorithmus 3.5: *Stair*

```
begin
 1.  Sortiere die m größten Aufträge in nicht steigender Reihenfolge;
 2.  for (j := 1, ..., n) do
 3.  begin
 4.      if (j < m and p_j > min_i{PC_i}) then
 5.      begin
 6.          Finde Prozessor P_l mit PC_l = max_i{PC_i | PC_i < p_j} und
             finde Prozessor P_k mit PC_k = min_i{PC_i | PC_i ≥ p_j};
 7.          if (PC_k = p_j) then
 8.              call Regel 1;
 9.          else
 10.         begin
 11.             Berechne φ_k^a, φ_k^b, φ_k^c;
 12.             if (p_j − PC_l > max{φ_k^a, φ_k^c}) then
 13.                 if (p_j − φ_k^b ≥ min{φ_k^a, φ_k^c}) then
 14.                     call Regel 2;
 15.                 else
 16.                     call Regel 3;
 17.             else
 18.                 call Regel 4;
 19.         end;
 20.     end;
 21.     else
 22.         call Regel 5;
 23. end;
end;
```

Die fünf Regeln, nach denen einzuplanen ist, werden in [Sch84] auf S. 156 ff. beschrieben. Im folgenden werden die Regeln an einem Beispiel dargestellt. Die Nichtverfügbarkeitsintervalle der Prozessoren sind schraffiert, verplante Zeitintervalle sind jeweils hellgrau hinterlegt.

Beispiel: Eine der Regeln 1-4 wird angewendet, wenn $1 \leq j < m, p_j > \min_i\{PC_i\}$ und falls für zwei Prozessoren P_k und P_l $(k, l \in \{1, \ldots, m\})$ gilt: $PC_k = \min_i\{PC_i \mid PC_i \geq p_j\}$ und $PC_l = \max_i\{PC_i \mid PC_i < p_j\}$. Regel 5 wird aufgerufen, wenn $m \leq j \leq n$ oder $p_j \leq \min_i\{PC_i\}$. Im folgenden wird für jede der fünf Regeln ein Beispiel angegeben.

Falls J_j exakt auf P_k paßt, d.h. falls $p_j = PC_k$, wird Regel 1 angewendet:

- Fülle P_k komplett mit J_j.

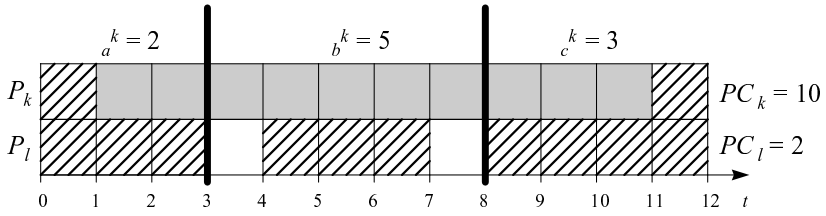

Abbildung 3.12: *Regel 1 ($p_j = 10$)*

Falls J_j nicht auf P_l und $\max\{\phi_k^a, \phi_k^c\}$ paßt und der Rest von J_j nicht auf ϕ_k^b paßt, d.h. falls $p_j - PC_l > \max\{\phi_k^a, \phi_k^c\}$ und $p_j - \phi_k^b \geq \min\{\phi_k^a, \phi_k^c\}$, wird Regel 2 angewendet:

- Falls $\phi_k^a \leq \phi_k^c$, fülle P_k von links nach rechts (ansonsten von rechts nach links).

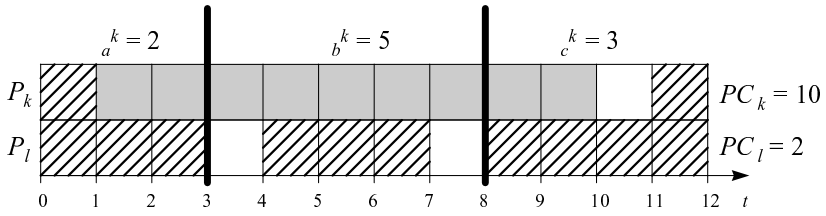

Abbildung 3.13: *Regel 2 ($p_j = 9$)*

Falls J_j nicht auf P_l und $\max\{\phi_k^a, \phi_k^c\}$ paßt und der Rest von J_j auf ϕ_k^b paßt, d.h. falls $p_j - PC_l > \max\{\phi_k^a, \phi_k^c\}$ und $p_j - \phi_k^b < \min\{\phi_k^a, \phi_k^c\}$, wird Regel 3 angewendet:

- Fülle P_k in den Intervallen von P_l.
- Falls $\phi_k^a \leq \phi_k^c$, fülle P_k von links nach rechts (ansonsten von rechts nach links).
- Verschiebe die nach t (im Falle $\phi_k^a \leq \phi_k^c$ vor t) verplanten Zeiteinheiten von J_j von P_k auf P_l. t ist dabei der Zeitpunkt bis zu dem (im Falle $\phi_k^a \leq \phi_k^c$ nach dem) Auftrag J_j kontinuierlich auf Prozessor P_k verplant wird (Intervalle, in denen P_k nicht verfügbar ist, sind nicht berücksichtigt).

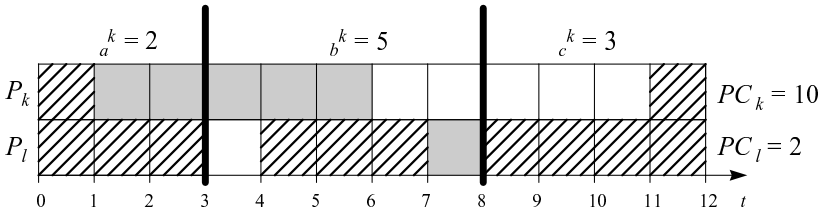

Abbildung 3.14: *Regel 3 ($p_j = 6$)*

Falls J_j auf P_l und $\max\{\phi_k^a, \phi_k^c\}$ paßt, wird Regel 4 angewendet:

- Fülle P_l komplett.
- Falls $\phi_k^a \leq \phi_k^c$, fülle P_k von rechts nach links (ansonsten von links nach rechts).

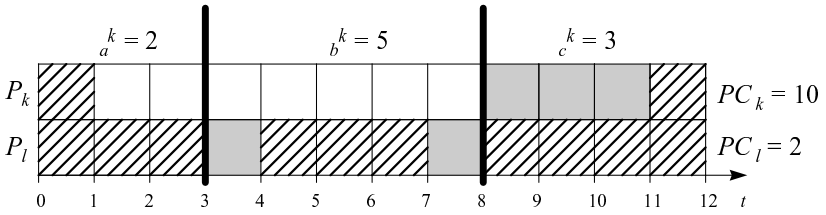

Abbildung 3.15: *Regel 4 ($p_j = 5$)*

In allen anderen Fällen wird nach Regel 5 verplant:

- Verplane die Aufträge in beliebiger Reihenfolge in den verbliebenen freien Bearbeitungsintervallen von links nach rechts ein. Beginne mit Prozessor P_l und wechsle zu einem Prozessor P_i, $i < l$ nur dann, wenn der Prozessor P_{i+1} vollständig belegt ist.

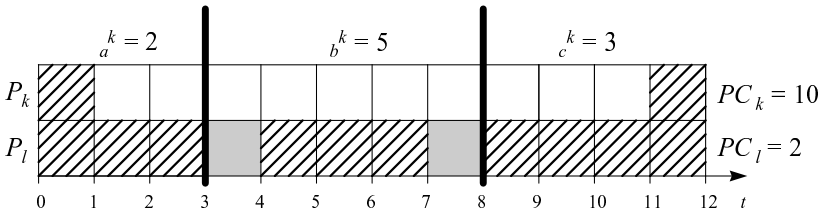

Abbildung 3.16: *Regel 5 ($p_j = 2$)*

Im Anschluß an jede der Regeln 1-4 werden die Prozessoren P_k und P_l jeweils zu einem *virtuellen Prozessor* P_k verbunden, der in den verbleibenden freien Verfügbarkeitsintervallen der ursprünglichen Prozessoren P_k und P_l verfügbar ist. □

Eine Aussage über Korrektheit, Laufzeit und Anzahl der Präemptionen, die von Algorithmus 3.5 im *worst-case* erzeugt werden, trifft Satz 3.5:

Satz 3.5 *Der Algorithmus Offline erzeugt für das Problem $P, NC^{offline} \mid pmtn \mid C_{max}$ mit einem Zeitaufwand von $O(Sm + n + m \log m)$ einen zulässigen und bezüglich der Planlänge optimalen Ablaufplan mit höchstens $\sum_{k=1}^{S-1} m_k + (Q-1) = O(Sm)$ Präemptionen.*

Der Beweis erfolgt mittels der drei folgenden Lemmas.

Lemma 3.4 *Der Algorithmus Offline erzeugt für das Problem $P, NC^{offline} \mid pmtn \mid C_{max}$ einen zulässigen und bezüglich der Planlänge optimalen Ablaufplan.*

Beweis: Die Korrektheit von Algorithmus 3.5 folgt aus dem Korrektheitsbeweis von *Stair* [Sch84], S. 157 ff. Darin wird gezeigt, daß *Stair* einen zulässigen Plan dann und nur dann erzeugt, wenn die folgenden m Bedingungen erfüllt sind:

$$\sum_{j=1}^{k} p_j \leq \sum_{i=1}^{k} PC_i \qquad \mathcal{P}(k = 1, \ldots, m-1)$$

$$\sum_{j=1}^{n} p_j \leq \sum_{i=1}^{m} PC_i \qquad \mathcal{P}(m)$$

mit $p_1 \geq p_2 \geq \ldots \geq p_n$ und $PC_1 \geq PC_2 \geq \ldots \geq PC_m$.

Wenn *Offline* terminiert, wurden alle Aufträge eingeplant. Aus den fünf Regeln, nach denen die Aufträge eingeplant werden, ist ersichtlich, daß kein Auftrag so eingeplant wird, daß er gleichzeitig von zwei Prozessoren bearbeitet wird. Die Bestimmung von $C_{max}^{p^*}$ erfolgt so, daß das Gesamtaufkommen an Bearbeitungszeit, das eingeplant wird, nicht größer als die verfügbare Kapazität ist. Wenn $C_{max}^{p^*}$ minimal ist, folgt daraus die Optimalität. □

Die Laufzeit ergibt sich aus den einzelnen Schritten des Algorithmus wie folgt:

Lemma 3.5 *Der Algorithmus Offline besitzt eine Laufzeit von* $O(n + Sm + m \log m)$. *Die einzelnen Schritte des Algorithmus besitzen jeweils folgende Laufzeiten:*

1. *Die Laufzeit zur Umwandlung eines beliebigen Verfügbarkeitsmusters in ein Treppenmuster beträgt* $O(Sm)$.
2. *Die Laufzeit zur Berechnung von* $C_{max}^{p^*}$ *beträgt* $O(n + Sm)$.
3. *Die Laufzeit zur Bestimmung der* $PC_i, i = 1, \ldots, m$, *beträgt* $O(Sm)$.
4. *Die Laufzeit von Stair beträgt* $O(n + m \log m)$ *unter der Annahme, daß die Summen der Bearbeitungskapazitäten* PC_i *für jeden Prozessor* P_i *bekannt sind und daß als Verfügbarkeitsmuster der Prozessoren ein Treppenmuster vorliegt.*

Beweis:

1. [SS98], S. 798.
2. folgt aus dem Algorithmus zur Berechnung unterer Schranken für $C_{max}^{p^*}$ in [SS98], S. 805 ff.
3. folgt aus der Beobachtung, daß zur Bestimmung der PC_i alle Q Prozessorintervalle betrachtet werden müssen. Mit Ungleichung (3.3) auf S. 60 ergibt sich ein Zeitaufwand in Höhe von $O(Q) = O(Sm)$.
4. [Sch84], S. 161. □

Eine Aussage über die Anzahl der erzeugten Präemptionen trifft folgendes Lemma:

Lemma 3.6 *Der Algorithmus Offline erzeugt nicht mehr als* $\sum_{k=1}^{S-1} m_k + (Q - 1) = O(Sm)$ *Präemptionen.*

Beweis: Nach der Umordnung der Prozessoren in ein *Treppenmuster* besteht ein zusammengesetzter Prozessor $P_i', 1 \leq i \leq m$, aus maximal S Prozessorenstücken. Im ungünstigsten Fall erhält man $\sum_{k=1}^{S-1} m_k \leq Sm$ Präemptionen durch Prozessorenwechsel auf zusammengesetzten Prozessoren. Hinzu kommen maximal $Q - 1$ Präemptionen

durch den Algorithmus *Stair*, da jeder Auftrag auf höchstens zwei Prozessoren verplant wird (vgl. [Sch84], S. 160). Mit $2Q \leq (S+1)\ m$ (vgl. Ungleichung (3.3) auf S. 60) ergeben sich so insgesamt maximal $O(Sm)$ Präemptionen. □

Folgendes Beispiel verdeutlicht, daß die Schranke scharf ist:

Beispiel: Es sollen vier Aufträge J_1, J_2, J_3, J_4 mit Beearbeitungszeiten von p_1=7, p_2=4, p_3=4 und p_4=3 Zeiteinheiten mit dem Algorithmus *Offline* verplant werden.

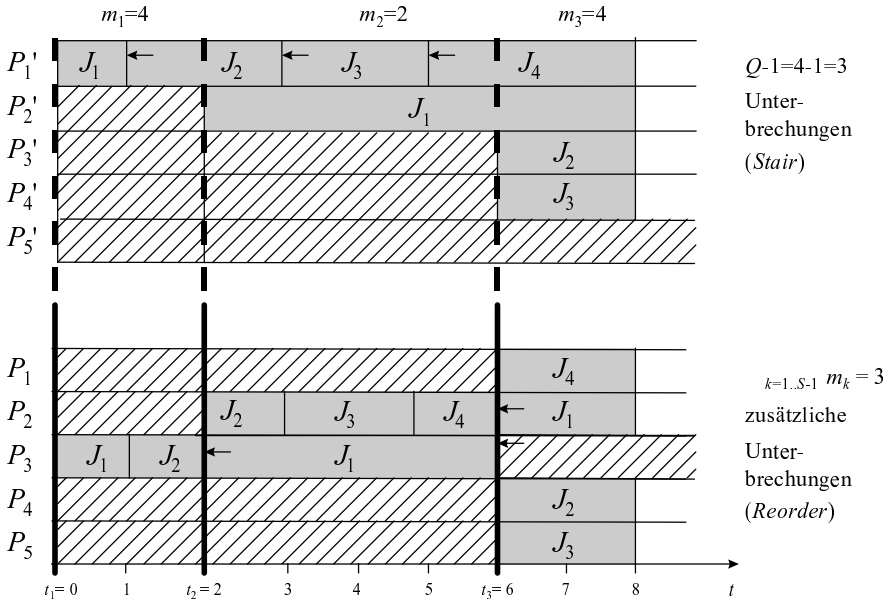

Abbildung 3.17: *Offline erzeugt im worst-case* $\sum_{k=1}^{S-1} m_k + (Q-1)$ *Präemptionen*

In unserem Beispiel gibt es S=3 Systemintervalle, d.h. S=3 Zeitpunkte t_1=0, t_2=2 und t_3=6, zu denen sich die Menge der verfügbaren Prozessoren ändert. Im ersten Systemintervall ist lediglich P_3 verfügbar, im zweiten sind P_2 und P_3 verfügbar und im dritten sind P_1, P_2, P_4 und P_5 verfügbar. Nach der Umordnung der Prozessoren in ein Treppenmuster (*Reorder*), der Bestimmung von $C_{max}^{p^*}$=8 und der Prozessorkapazitäten PC'_1=8, PC'_2=6, PC'_3=2 und PC'_4=2, können die Aufträge mit dem Algorithmus *Stair* auf die Prozessoren verplant werden. Im Beispiel wird die *worst-case*-Schranke erreicht: Die Anzahl der Präemptionen beträgt $(\sum_{k=1}^{S-1} m_k) + (Q-1) = 6$ (vgl. Abb. 3.17, die Präemptionen sind durch Pfeile markiert). □

3.2.3 Nearly-Online

Liu und Sanlaville geben in [LS95] einen Algorithmus für das *nearly-online setting* an, der die Aufträge mit Hilfe der *LPT-Regel* verplant. Der Algorithmus erzeugt optimale Pläne für beliebige Verfügbarkeitsmuster. Albers und Schmidt beschreiben in [AS01] einen Algorithmus *Lookahead* (vgl. Algorithmus 3.6), der auf der gleichen Idee basiert wie der von Liu und Sanlaville.

Algorithmus 3.6: *Lookahead*

begin
1. $t := 0$;
2. **for** $(j := 1, \ldots, n)$ **do** $r_j := p_j$;
3. **while** (es gibt noch Aufträge mit positiver Restbearbeitungszeit) **do**
4. **begin**
5. t' := nächster Zeitpunkt, zu dem sich die Menge der verfügbaren Prozessoren verändert; $\delta := t' - t$;
6. m_a := Anzahl der im Intervall $[t, t+\delta)$ verfügbaren Prozessoren;
7. **for** $(j := 1, \ldots, n)$ **do** Bestimme $T_j := \sum_{k=j}^{n} \max\{0, r_k - (r_j - \delta)$ (Algorithmus 3.7);
8. $j := 1$;
9. **while** $(j \leq n$ **and** $T_j < m_a\delta$ **and** $r_j \geq \delta)$ **do**
10. **begin**
11. Verplane δ Zeiteinheiten von J_j in $[t, t+\delta)$;
12. $r_j := r_j - \delta$; $m_a := m_a - 1$; $j := j + 1$;
13. **end;**
14. **if** $(j \leq n)$ **then**
15. **begin**
16. Berechne das größte ϵ, $\epsilon \leq \min\{\delta, r_j\}$, so daß $\sum_{k=j}^{n} \max\{0, r_k - (r_j - \epsilon)\} \leq m_a\delta$ (Algorithmus 3.8);
17. **for** $(k := i, \ldots, n)$ **do** Verplane mit McNaughtons Algorithmus $\max\{0, r_k - (r_j - \epsilon)\}$ Zeiteinheiten von J_k in $[t, t+\delta)$ und setze $r_k := \min\{r_k, r_j - \epsilon\}$;
18. **end;**
19. $t := t'$;
20. **end;**

end;

Die Algorithmen 3.7 und 3.8, die von Albers und Schmidt nicht beschrieben werden,

können wie folgt formuliert werden:

Algorithmus 3.7: *Berechnung aller T_j*

begin
1. $l_0 := 0$;
2. **for** $(j := 1, \ldots, n)$ **do**
3. **begin**
4. $\quad k := l_{j-1} + 1$;
5. $\quad$ **while** $((r_k - (r_j - \delta)) \geq 0)$ **do** $k := k + 1$;
6. $\quad l_j := k$;
7. **end;**
8. $T_0 := \delta$;
9. **for** $(j := 0, \ldots, n-1)$ **do**

$\quad T_{j+1} := T_j - \delta + (l_j - j)(r_j - r_{j+1}) + \sum_{k=l_j+1}^{l_{j+1}} (r_k - (r_{j+1} - \delta))$;

end;

Algorithmus 3.8: *Berechnung von ϵ*

begin
1. $U_j[k-1] := 0$;
2. **for** $(k := j, \ldots, n)$ **do** $U_j[k] := U_j[k-1] + \epsilon - (r_j - r_k)$;
3. **for** $(k := n, \ldots, j)$ **do**
4. **begin**
5. $\quad$ Berechne ϵ mit $U_j[k] = m_a \delta$;
6. $\quad$ **if** $(\epsilon \geq (r_j - r_k))$ **then goto** 8.;
7. **end;**
8. **if** $(\epsilon > r_j)$ **then** $\epsilon := r_j$;

end;

Lookahead sortiert die Aufträge in nichtaufsteigender Reihenfolge (LPT) und verplant in jedem Systemintervall Teile von Aufträgen, so daß möglichst viel von den längeren Aufträgen verplant wird, ohne daß die LPT-Reihenfolge der Aufträge verändert wird. In Zeile 9 von Algorithmus 3.6 wird für einen Auftrag J_j untersucht, ob es möglich ist, δ Zeiteinheiten von ihm einzuplanen, ohne daß die Invariante $r_1 \geq r_2 \geq \ldots \geq r_n$ verletzt wird. Dazu muß eine Summe $T_j := \sum_{k=1}^{n} \max\{0, r_k - (r_j - \delta)\}$ berechnet werden. T_j gibt an, wieviel Bearbeitungskapazität in dem aktuellen Systemintervall

vorhanden sein muß, so daß nach Einplanung von δ Zeiteinheiten von Auftrag J_j noch genügend Bearbeitungskapazität übrig bleibt, um Teile von allen anderen Aufträgen so einzuplanen, daß die Invariante erhalten bleibt. Die Summen T_j in Zeile 9 werden mit Algorithmus 3.7 berechnet. Zur Berechnung von T_j wird zunächst der größte Auftragsindex l_j bestimmt, so daß $r_{l_j} - (r_j - \delta) \geq 0$. l_{j+1} läßt sich leicht aus l_j berechnen, indem man, startend bei $k := l_j$, solange durch die Terme $r_k - (r_{j+1} - \delta)$ geht, bis $r_k - (r_{j+1} - \delta) \geq 0$. Die Berechnung von ϵ in Zeile 16 von Algorithmus 3.6 erfolgt mit Algorithmus 3.8. Zu beachten ist, daß man zur Berechnung von ϵ folgendes überprüfen muß: 1. $\epsilon \geq (r_j - r_k)$, da ϵ größer oder gleich $r_j - r_k$ sein muß, wenn der k-te Term in der Summe sein soll. 2. $\epsilon \leq r_j$, da man nicht mehr Last als vorhanden ist verplanen kann.

Das folgende Beispiel illustriert die Vorgehensweise von Algorithmus 3.6.

Beispiel: Es sollen n=6 Aufträge J_j, j=1, ... , 6 mit den Bearbeitungsdauern p_j=14, 12, 7, 5, 3, 1 auf den m=4 Prozessoren P_i, $i = 1, \ldots, 4$ so eingeplant werden, daß die Planlänge minimiert wird. Präemptionen der Aufträge sind erlaubt, bezüglich der Verfügbarkeiten der Prozessoren kennt man allerdings lediglich den nächsten Zeitpunkt t', zu dem sich die Menge der verfügbaren Prozessoren ändert, also das jeweils nächste Systemintervall $[t, t')$. Zu Beginn stehen alle m_a=4 Prozessoren genau δ=5 Zeiteinheiten zur Verfügung. Abb. 3.18 zeigt die Ausgangssituation.

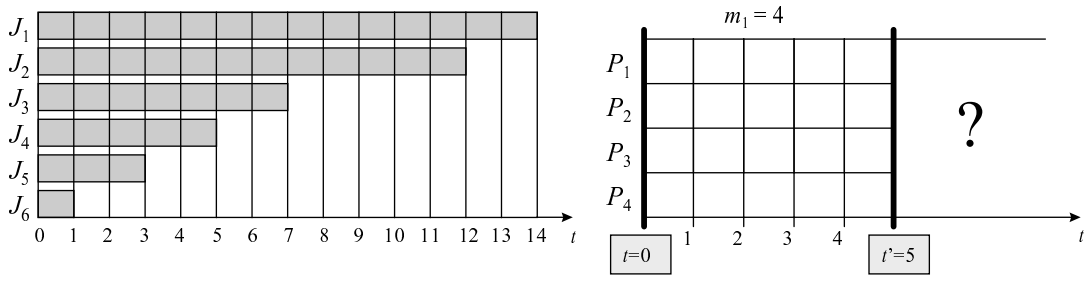

Abbildung 3.18: *Beispiel für Lookahead: Erstes Systemintervall*

Nach Initialisierung von t und der r_j geht Algorithmus 3.6 wie folgt vor:

3. **while** (es gibt noch Aufträge mit positiver Restbearbeitungszeit) **do**
4. **begin**
5. $t' := 5$ = nächster Zeitpunkt, zu dem sich die Menge der verfügbaren Prozessoren verändert; $\delta := t' - t = 5 - 0 = 5$;
6. $m_a := 4$ = Anzahl der im Intervall $[t, t+\delta)$ verfügbaren Prozessoren;

Da zu Beginn alle Aufträge eine positive Restbearbeitungszeit haben, wird t' auf den nächsten Zeitpunkt, zu dem sich die Menge der verfügbaren Prozessoren ändert, gesetzt und δ und m_a bestimmt. In Zeile 7 werden daraufhin mit Algorithmus 3.7 die Summen T_j berechnet. In unserem Beispiel ist $l_j = 2, 3, 5, 6, -, -(j = 1, \ldots, 6)$ (zu beachten ist, daß die Berechnung von l_j nur im Falle $r_j \geq \delta$ erfolgt). Weiter werden $l_0 := r_0 := 0$ und $T_0 := \delta = 5$ gesetzt. In Tab. 3.20 ist die Berechnung aller $T_j, j = 1, \ldots, n$, schematisch dargestellt. Dabei bedeutet $(a, b) := \max\{a, b\}$.

k	j=1	j=2	j=3	j=4	j=5	j=6
	$(0, r_k - (r_j -$ $))$	$(0, r_k - (r_j -$ $))$	$(0, r_k - (r_j -$ $))$	$(0, r_k - (r_j -$ $))$	$(0, r_k - (r_j -$ $))$	$(0, r_k - (r_j -$ $))$
1	(0 , 14-(14-5)) = 5					
2	(0 , 12-(14-5)) = 3	(0 , 12-(12-5)) = 5				
3	(0 , 7-(14-5)) = 0	(0 , 7-(12-5)) = 0	(0 , 7-(7-5)) = 5			
4	(0 , 5-(14-5)) = 0	(0 , 5-(12-5)) = 0	(0 , 5-(7-5)) = 3	(0 , 5-(5-5)) = 5		
5	(0 , 3-(14-5)) = 0	(0 , 3-(12-5)) = 0	(0 , 3-(7-5)) = 1	(0 , 3-(5-5)) = 3	-	
6	(0 , 1-(14-5)) = 0	(0 , 1-(12-5)) = 0	(0 , 1-(7-5)) = 0	(0 , 1-(5-5)) = 1	-	-
	$T_1 = 8$	$T_2 = 5$	$T_3 = 9$	$T_4 = 9$		
	r_j				$r_j <$	

Tabelle 3.20: *Berechnung der Summen T_j*

Beispielsweise ergibt sich T_2 aus T_1 wie folgt:

$$
\begin{aligned}
T_2 &= T_1 \\
&\quad -\delta \text{ (dunkelgrau schraffierte Fläche bei } j = 1 \text{ und } k = 1) \\
&\quad +(l_j - j)(r_j - r_{j+1}) \text{ (hellgrau schraffierte Fläche)} \\
&\quad +\sum_{k=l_j+1}^{l_{j+1}} (r_k - (r_{j+1} - \delta)) \text{ (dunkelgrau schraffierte Fläche bei } j = 2 \text{ und } \\
&\qquad\qquad k = 3) \\
&= 8 - 5 + 2 + 0 = 5.
\end{aligned}
$$

Nach Berechnung der T_j werden die Aufträge der Reihe nach eingeplant. Beispielsweise geht Algorithmus 3.6 bei der Einplanung von $J_{j=1}$ wie folgt vor:

```
9.    while ((j = 1) ≤ (6 = n) and
          (T_1 = 8) < (20 = m_a δ) and
          (r_1 = 14) ≥ (5 = δ)) do
10.   begin
11.       Verplane (δ = 5) Zeiteinheiten von J_1 in [t, t + δ) = [0, 5);
12.       r_1 := r_1 − δ = 14 − 5 = 9;  m_a := m_a − 1 = 3;  j := j + 1 = 2;
13.   end;
```

Für Auftrag J_1 wird untersucht, ob es möglich ist, δ Zeiteinheiten von ihm einzuplanen, ohne daß die Invariante $r_1 \geq r_2 \geq \ldots \geq r_n$ verletzt wird. Da alle Bedingungen in Zeile 9 erfüllt sind, werden von Algorithmus 3.6 in Zeile 11 $\delta = 5$ Zeiteinheiten von J_1 auf Prozessor P_1 eingeplant, und in Zeile 12 werden r_1, m_a und j aktualisiert. Ähnliche Berechnungen werden für die Aufträge $J_{j=2}$, $J_{j=3}$ und $J_{j=4}$ durchgeführt (vgl. Abb. 3.19-3.22).

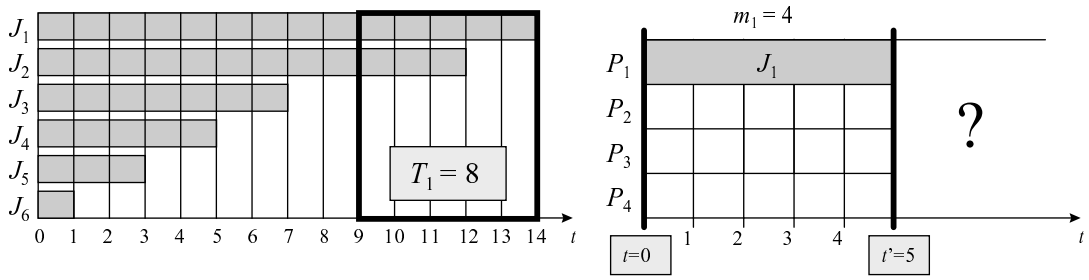

Abbildung 3.19: *Einplanung auf Prozessor* P_1

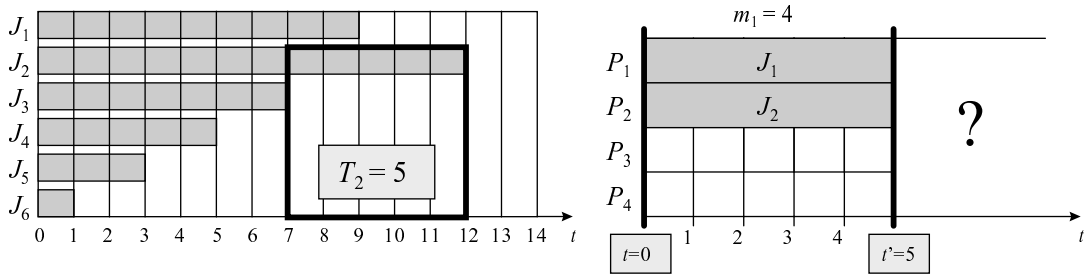

Abbildung 3.20: *Einplanung auf Prozessor* P_2

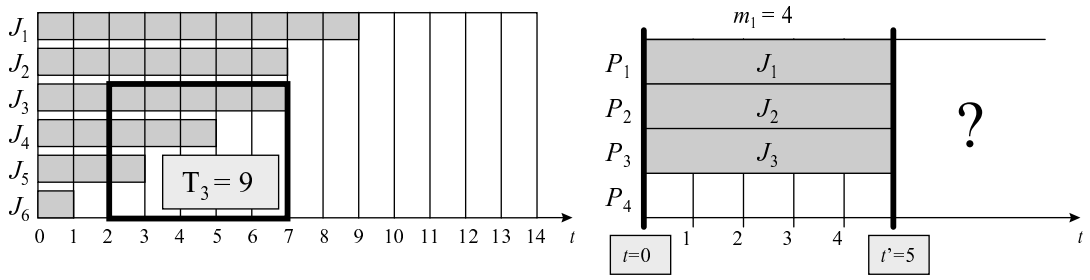

Abbildung 3.21: *Einplanung auf Prozessor* P_3

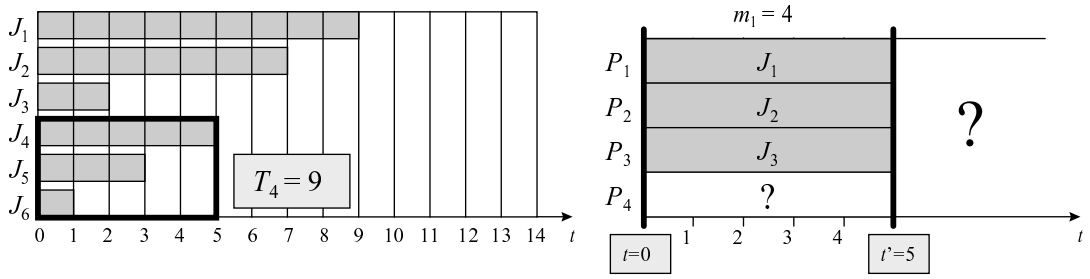

Abbildung 3.22: *Einplanung auf Prozessor* P_4

Während alle Bedingungen in Zeile 9 von Algorithmus 3.6 für j=1, j=2 und j=3 erfüllt sind, ist für j=4 die Bedingung $T_4 < m_a\delta$ nicht erfüllt. Es ist nicht möglich, $\delta = 5$ Zeiteinheiten von Auftrag $J_{i=4}$ einzuplanen, ohne die Invariante $r_1 \geq r_2 \geq \ldots \geq r_n$ zu verletzen, da T_4=9 Zeiteinheiten benötigt würden, um die Invariante einzuhalten, aber lediglich $m_a \cdot \delta = 5$ Zeiteinheiten zur Verfügung stehen. Daher wird mit Algorithmus 3.8 berechnet, wieviele Zeiteinheiten von Auftrag $J_{j=4}$ maximal verplant werden dürfen, so daß noch genügend Restkapazität übrig bleibt, um die Invariante einzuhalten. Algorithmus 3.8 berechnet das größte $\epsilon \leq \min\{\delta, r_4\}$, so daß $\sum_{k=(j=4)}^{n=6} \max\{0, r_k - (r_4 - \epsilon)\} \leq m_a\delta$. Nach Durchlaufen der Schleife in Zeile 2 von Algorithmus 3.8 gilt in unserem Beispiel:

k	4	5	6
$U_j[k]$	$1\epsilon - 0$	$2\epsilon - 2$	$3\epsilon - 6$

In der darauffolgenden Schleife ergibt sich $\epsilon = 11/3$ für $k = n = 6$, was aber nicht größer oder gleich $r_{j=4} - r_{k=6} = 4$ ist. Dann erhält man $\epsilon = 3.5$ für $k = 5$, was größer oder gleich $r_{i=4} - r_{k=6} = 2$ ist. Also hat man $\epsilon = 3.5$ gefunden.

Nun werden in Zeile 17 von Algorithmus 3.6 von den verbleibenden Aufträgen $J_k, k = 4, \ldots, 6$ jeweils $\max\{0, r_k - (r_4 - \epsilon)\}$ Zeiteinheiten mit dem Algorithmus von McNaughton eingeplant. In unserem Beispiel also 3.5 Zeiteinheiten von Auftrag 4, 1.5 Zeiteinheiten von Auftrag 5 und 0 Zeiteinheiten von Auftrag 6. Abb. 3.23 zeigt die Situation nach der Einplanung der Aufträge 4 und 5 auf Prozessor P_4.

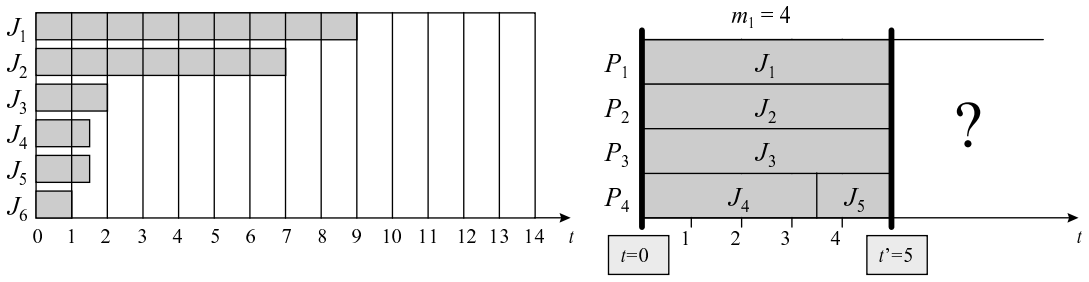

Abbildung 3.23: *Situation nach Einplanung der Aufträge im 1. Systemintervall*

In Zeile 12 wird schließlich noch t auf $t' = 5$ gesetzt. Damit ist das erste Systemintervall vollständig verplant und der Algorithmus springt zurück in Zeile 3. Ähnlich ist die Vorgehensweise im zweiten Systemintervall $[5, 15]$. Der gesamte Ablaufplan ist in Abb. 3.24 abgebildet.

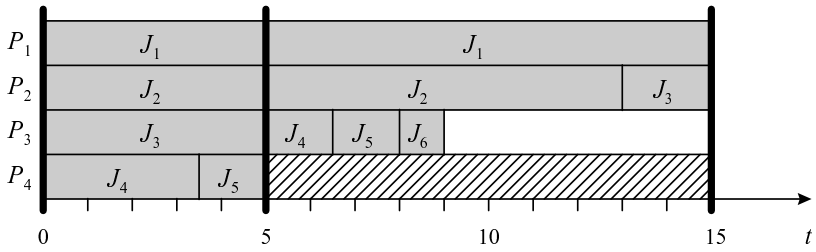

Abbildung 3.24: *Situation nach Einplanung aller Aufträge*

Der Algorithmus terminiert, weil alle Aufträge verplant wurden und die Restbearbeitungszeiten 0 sind. □

Satz 3.6 enthält eine Aussage über Korrektheit, Laufzeit und Anzahl der Präemptionen, die von Algorithmus *Lookahead* im *worst-case* erzeugt werden.

Satz 3.6 *Der Algorithmus Lookahead erzeugt für das Problem $P, NC^{offline} \mid pmtn \mid C_{max}$ mit einem Zeitaufwand von $O(Sn + n \log n)$ einen zulässigen und bezüglich der Planlänge optimalen Ablaufplan mit höchstens $\sum_{k=1}^{S-1} m_k + (S-1)(n-1) + (m-1) = O(Sm + Sn)$ Präemptionen.*

Die Korrektheit von Satz 3.6 ergibt sich aus den folgenden Lemmas 3.7-3.9.

Lemma 3.7 *Lookahead erzeugt für das Problem $P, NC^{nearly\text{-}online} \mid pmtn \mid C_{max}$ einen zulässigen und bezüglich der Planlänge optimalen Ablaufplan.*

Beweis: [AS01], S. 92 und *Theorem* 2 auf S. 93. □

Im Lemma 3.8 wird die Laufzeit von *Lookahead* abgeschätzt. Dabei werden die Prozessorenupdates als Teil der Eingabe betrachtet.

Lemma 3.8 *Der Algorithmus Lookahead besitzt eine Laufzeit von $O(Sn + n \log n)$.*

Beweis: Zu Beginn werden die Aufträge in nichtsteigender Reihenfolge sortiert (Zeitaufwand: $O(n \log n)$). In S Systemintervallen werden dann die Aufträge verplant. Dabei wird die Laufzeit durch die Berechnung der T_j und des jeweiligen ϵ bestimmt. Geht man naiv vor, so erhält man zur Berechnung von T_j eine Laufzeit von $O(n)$. Mit der direkten Berechnung von T_j für einen zu verplanenden Auftrag J_i käme man so auf eine Laufzeit von $O(n^2)$ für die Berechnung aller Summen $T_j, j = 1, \ldots, n$. Anstatt jedes T_{j+1} neu zu berechnen, berechnet man T_{j+1} aus T_j mit Algorithmus 3.7. Die Vorberechnung der l_i dauert Zeit $O(n)$, da $r_{i+1} \geq r_i$ auch $l_{i+1} \geq l_i$ impliziert. Zu beachten ist, daß l_i nur im Falle $r_i \geq \delta$ berechnet wird. Dabei ist l_i der jeweils größte Index k mit $r_k - (r_{i+1} - \delta) \geq 0$. Da jedes T_{j+1} in konstanter Zeit aus T_j berechnet werden kann, ergibt sich ein Zeitbedarf zur Berechnung aller Summen T_j in Höhe von $O(n)$. Die Berechnung des größten ϵ, $\epsilon \leq \min\{\delta, r_j\}$, so daß $\sum_{k=j}^{n} \max\{0, r_k - (r_j - \epsilon)\} \leq m_a \delta$, erfolgt mit Algorithmus 3.8 in Zeit $O(n)$. Die Einplanung der Aufträge in S Systemintervallen erfolgt also nach der anfänglichen Sortierung in Zeit $O(Sn)$. □

In [AS01] wird keine Aussage über die Anzahl der Präemptionen, die *Lookahead* im *worst-case* erzeugt, getroffen. Lemma 3.9 analysiert die Anzahl der Präemptionen.

Lemma 3.9 *Der Algorithmus Lookahead erzeugt höchstens* $\sum_{k=1}^{S}(m_k-1)+(S-1)n = O(Sm+Sn)$ *Präemptionen.*

Beweis: Im ungünstigsten Fall werden in jedem der maximal S Systemintervalle Teile von jedem der n Aufträge mit dem Algorithmus von McNaughton verplant ($\sum_{k=1}^{S}(m_k-1)$ Präemptionen). Werden in $S-1$ Intervallen jeweils alle n Aufträge weiterbearbeitet, so erzeugt der Algorithmus *Lookahead* insgesamt maximal $\sum_{k=1}^{S}(m_k-1)+(S-1)n = \sum_{k=1}^{S-1} m_k + (S-1)(n-1)+(m-1) = O(Sm+Sn)$ Präemptionen. □

Folgendes Beispiel zeigt, daß die angegebene *worst-case* Schranke bzgl. der Anzahl der Präemptionen scharf ist.

Beispiel: Fünf Aufträge $J_1,\ldots,J_5$ mit Bearbeitungszeiten von je acht Zeiteinheiten sollen mit dem Algorithmus *Lookahead* verplant werden. Zu S=3 Zeitpunkten (t_1=0, t_2=5 und t_3=10) ändert sich die Menge der verfügbaren Prozessoren, d.h. es gibt $S=3$ Systemintervalle $[0,5),[5,10)$ und $[10,\infty)$. Im ersten Intervall sind $m_1=4$, im zweiten $m_2=2$ und im dritten sind m_3=4 Prozessoren verfügbar (vgl. Abb. 3.25).

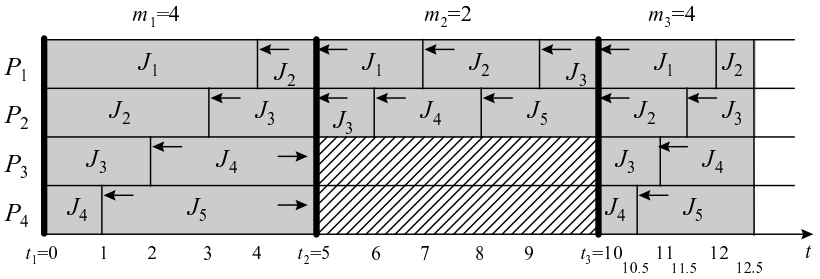

Abbildung 3.25: *Präemptionen von Lookahead*

Lookahead verplant in jedem der S=3 Systemintervalle Teile aller n=5 Aufträge mit dem Algorithmus von McNaughton. Im zweiten und im dritten Intervall werden jeweils alle n=5 Aufträge weiterbearbeitet. Die Anzahl der Präemptionen beträgt $(S-1)n+\sum_{k=1}^{S}(m_k-1) = 10+(3+1+3) = 17$. Das Beispiel zeigt, daß die *worst-case* Schranke erreicht werden kann. □

3.2.4 Online

Online-Algorithmen zur Lösung des Problems $P, NC^{online} \mid pmtn \mid C_{max}$ kennen nur die aktuell verfügbaren Prozessoren. Bezüglich der Planlänge optimale *Online*-Algorithmen lassen sich immer dann erzeugen, wenn entweder nur ein Prozessor zur Verfügung steht ($m = 1$) oder wenn nur ein Auftrag zu bearbeiten ist ($n = 1$). *Theorem* 1 auf S. 3 in [AS01] besagt:

> *No online algorithm can, in general, construct optimal schedules.*

In Abschnitt 3.2.4.1 wird zunächst gezeigt, daß der Beweis nur für Instanzen mit $m = n$ und $n \geq 2$ gilt. Die sich daran anschließende Frage, ob stets optimale *Online*-Algorithmen für den üblichen Fall $m < n$ erstellt werden können, wird im Anschluß untersucht. Dabei wird gezeigt, daß es Probleminstanzen mit $m \geq 3$ und $n \geq 2$ gibt, für die kein optimaler *Online*-Algorithmus erstellt werden kann. Daran anschließend wird in Abschnitt 3.2.4.2 der Algorithmus *Online*(ϵ) von Albers und Schmidt [AS01] beschrieben, der Ablaufpläne mit einem maximalen Fehler von ϵ erstellt. Im folgenden wird mit C_{max}^{off} die beste durch einen *Offline*-Algorithmus erreichbare Planlänge und mit C_{max}^{on} die beste durch einen *Online*-Algorithmus erreichbare Planlänge bezeichnet.

3.2.4.1 Güte von *Online*-Algorithmen

Lemma 3.10 *Für bestimmte Probleminstanzen I von $P, NC \mid pmtn \mid C_{max}$ mit $m = n$ und $n \geq 2$ gibt es keinen Online-Algorithmus ON mit $C_{max}^{on}(I) = C_{max}^{off}(I)$.*

Beweis: Der Beweis aus [AS01], *Theorem* 1, wird hier ausgeführt. Zum Zeitpunkt 0 steht nur einer der m Prozessoren zur Verfügung. Alle n Aufträge haben eine Bearbeitungszeit von 1 Zeiteinheit. Es wird angenommen, daß $m = n$ gilt. Zum Zeitpunkt 0 beginnt ein optimaler Algorithmus ON mit der Bearbeitung von Auftrag J_j. Sei t' der erste Zeitpunkt, zu dem ON die Bearbeitung von J_j das erste mal unterbricht oder zu

dem ON die Bearbeitung von J_j abschließt. Zum Zeitpunkt t' werden alle Prozessoren verfügbar (vgl. Abb. 3.26).

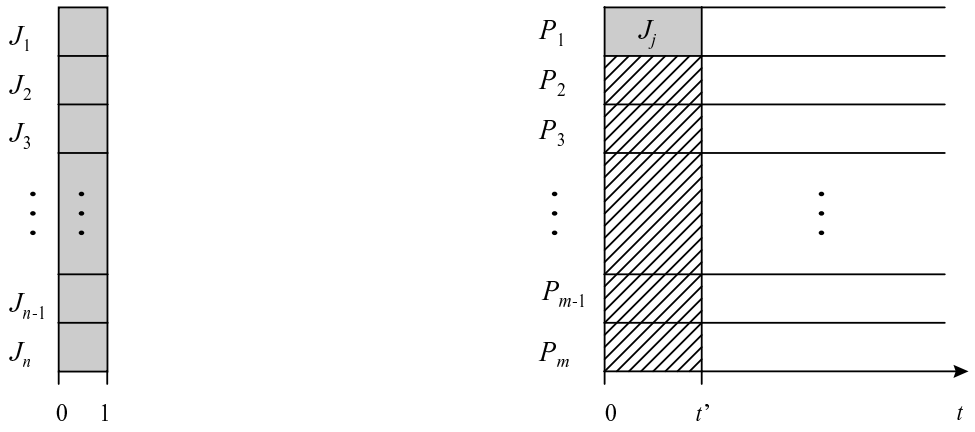

Abbildung 3.26: *Problemstellung* I

Für den von ON erzeugten Plan gilt $C_{max}^{on} \geq t' + 1$, da zum Zeitpunkt t' noch keiner der Aufträge $J_k, k \neq i$ bearbeitet wurde. Genauer gilt: Bis zum Zeitpunkt t' wird nur Auftrag J_i bearbeitet. Nach dem Zeitpunkt t' werden die Aufträge mit dem Algorithmus von McNaughton [McN59] verplant und man erhält als Planlänge:

$$\begin{aligned} C_{max}^{on} &= t' + \max\left\{ \frac{\sum_{j=1}^{n} r_j}{m} \quad , \quad \max\{r_j, 1 \leq j \leq n\} \right\} \\ &= t' + \max\left\{ \frac{n - t'}{m} \quad , \quad 1 \right\} \\ &\geq t' + 1. \end{aligned} \tag{3.6}$$

Ein optimaler *Offline*-Algorithmus teilt das Intervall von 0 bis t' zu gleichen Teilen unter allen n Aufträgen auf. Nach dem Zeitpunkt t' werden die Aufträge mit dem Algorithmus von McNaughton [McN59] verplant und man erhält als Planlänge:

$$\begin{aligned} C_{max}^{off} &= t' + \max\left\{ \frac{\sum_{j=1}^{n} r_j}{m} \quad , \quad \max\{r_j, 1 \leq j \leq n\} \right\} \\ &= t' + \max\left\{ \frac{n - t'}{m} \quad , \quad 1 - \frac{t'}{n} \right\} \end{aligned}$$

$$= t' + \begin{cases} \frac{n-t'}{m} & , m \leq n \\ \frac{n-t'}{n} & , m \geq n \end{cases} \quad (3.7)$$

$$< t' + 1 \text{ für } m \geq n \quad (3.8)$$

$$\overset{(3.6)}{\leq} C_{max}^{on} \text{ für } m \geq n.$$

zu (3.7): $\frac{n-t'}{m} \geq \frac{n-t'}{n} \overset{t' \leq 1 \leq n}{\Longrightarrow} n \geq m$

zu (3.8): $C_{max}^{off} < t' + 1$ gilt nur für $m \geq n$, da:

Fall $m < n$: $C_{max}^{off} = t' + \frac{n-t'}{m} \overset{m \leq n-1, t' \leq 1 \leq n}{\geq} t' + \frac{n-t'}{n-1} \overset{t' \leq 1}{\geq} t' + 1$

Fall $m \geq n$: $C_{max}^{off} = t' + \frac{n-t'}{n} = t' + 1 - \frac{t'}{n} \overset{t' > 0}{<} t' + 1.$

□

Beispiel: m=2 Prozessoren stehen zur Bearbeitung von n=2 Aufträgen J_1, J_2 bereit. J_1 und J_2 haben jeweils eine Bearbeitungszeit von einer Zeiteinheit. Zu Beginn steht lediglich Prozessor P_1 zur Verfügung. Nach t'=2 Zeiteinheiten sind beide Prozessoren verfügbar (vgl. Abb. 3.27). Der *Online*-Algorithmus erzielt eine Planlänge von $C_{max}^{on} =$

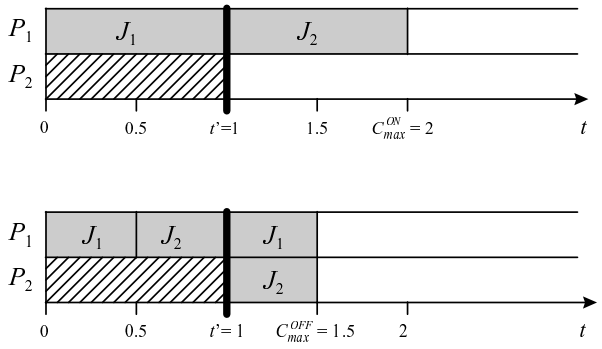

Abbildung 3.27: $C_{max}^{on} > C_{max}^{off}$ *für* $m \geq n$

t'+1 = 1+1 = 2 Zeiteinheiten. Ein optimaler *Offline*-Algorithmus erzielt eine Planlänge von $C_{max}^{off} = t' + 1 - (t'/n) = 1 + 1 - (1/2) = 1.5$ Zeiteinheiten.

Der Beweis gilt nicht, wenn $m < n$ gilt. Folgendes Beispiel verdeutlicht dies. $m = 2$ Prozessoren stehen zur Bearbeitung von $n = 4$ Aufträgen $J_1, \ldots, J_4$ bereit. Alle Aufträge haben eine Bearbeitungszeit von 1 Zeiteinheit. Zu Beginn steht lediglich Prozessor P_1 zur Verfügung. Nach $t' = 1$ Zeiteinheiten sind alle Prozessoren verfügbar (vgl. Abb. 3.28).

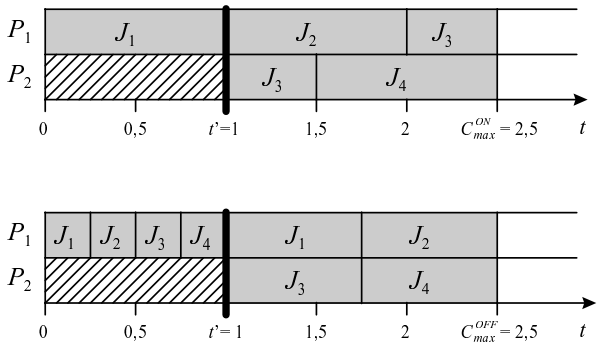

Abbildung 3.28: $C_{max}^{on} = C_{max}^{off}$ *für* $m < n$

Der *Online*-Algorithmus erzielt eine Planlänge von $C_{max}^{on} = t' + (n - t')/m = 1 + (4 - 1)/2 = 2.5$ Zeiteinheiten. Ein optimaler *Offline*-Algorithmus erzielt ebenfalls eine Planlänge von $C_{max}^{off} = t' + (n - t')/m = 1 + (4 - 1)/2 = 2.5$ Zeiteinheiten. □

Da der Fall $m \geq n$ ein Spezialfall ist, kann man sich fragen, ob für den allgemeineren Fall, daß weniger Prozessoren als Aufträge zur Verfügung stehen, stets optimale *Online*-Algorithmen formuliert werden können. Das folgende Lemma 3.11 beweist das Gegenteil.

Lemma 3.11 *Für bestimmte Probleminstanzen I von $P, NC \mid pmtn \mid C_{max}$ mit $m \geq 3$ und $n \geq 2$ gibt es keinen Online-Algorithmus ON mit $C_{max}^{on}(I) = C_{max}^{off}(I)$.*

Beweis: Zum Zeitpunkt 0 steht nur einer der m Prozessoren zur Verfügung. Zwei Aufträge (J_1 und J_2) haben eine Bearbeitungszeit von n Zeiteinheiten, die restlichen

$n-2$ Aufträge $J_3, \ldots, J_n$ haben eine Bearbeitungszeit von einer Zeiteinheit. Es wird angenommen, daß $m \geq 3$ und $n \geq 2$ gilt. Zum Zeitpunkt 0 beginnt ein optimaler Algorithmus ON mit der Bearbeitung von Auftrag J_j. Sei t' der erste Zeitpunkt, zu dem ON die Bearbeitung von J_j das erste mal unterbricht oder zu dem ON die Bearbeitung von J_j abschließt. Zum Zeitpunkt t' werden alle Prozessoren verfügbar (vgl. Abb. 3.29).

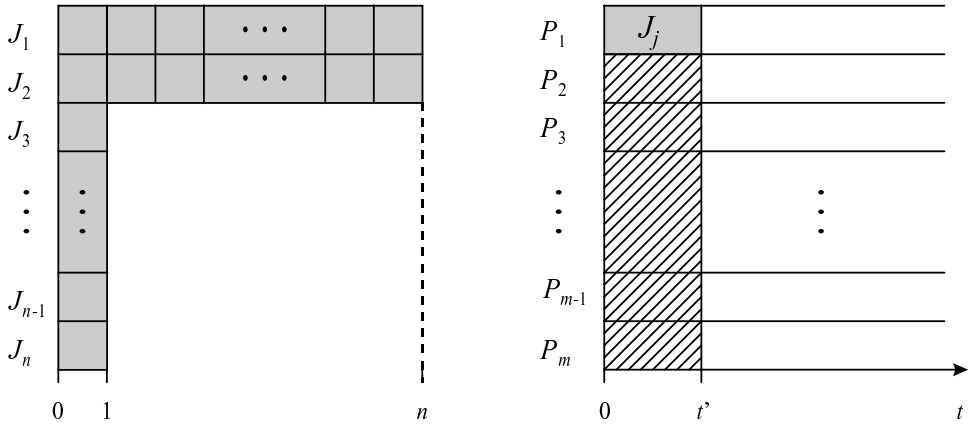

Abbildung 3.29: *Problemstellung* II

Für den von ON erzeugten Plan gilt $C_{max}^{on} = t' + n$, da zum Zeitpunkt t' wenigstens einer der beiden Aufträge J_1 oder J_2 noch nicht bearbeitet wurde. Genauer gilt: Bis zum Zeitpunkt t' wird nur Auftrag J_i bearbeitet. Nach dem Zeitpunkt t' werden die Aufträge mit dem Algorithmus von McNaughton [McN59] verplant und man erhält als Planlänge:

$$\begin{aligned} C_{max}^{on} &= t' + \max\left\{ \frac{\sum_{j=1}^{n} r_j}{m} \quad , \quad \max\{r_j, 1 \leq j \leq n\} \right\} \\ &= t' + \max\left\{ \frac{2n + (n-2) - t'}{m} \quad , \quad n \right\} \\ &\geq t' + n. \end{aligned} \tag{3.9}$$

Ein optimaler *Offline*-Algorithmus teilt das Intervall von 0 bis t' zu gleichen Teilen unter den beiden ersten Aufträgen J_1 und J_2 auf. Nach dem Zeitpunkt t' werden die Aufträge mit dem Algorithmus von McNaughton [McN59] verplant und man erhält als Planlänge:

$$
\begin{aligned}
C_{max}^{off} &= t' + \max\left\{ \frac{\sum_{j=1}^{n} r_j}{m} \quad , \quad \max\{r_j, 1 \le j \le n\} \right\} \\
&= t' + \max\left\{ \frac{2n + (n-2) - t'}{m} \quad , \quad n - \frac{t'}{2} \right\} \\
&= t' + \max\left\{ \frac{3n - (t'+2)}{m} \quad , \quad n - \frac{t'}{2} \right\} \\
&= t' + \begin{cases} \frac{3n-(t'+2)}{m} & , m \le 2 + \frac{2n-4}{2n-t'} \\ n - \frac{t'}{2} & , m \ge 2 + \frac{2n-4}{2n-t'} \end{cases} \qquad (3.10) \\
&< t' + n \text{ für } m \ge 3 \qquad (3.11) \\
&\overset{(3.9)}{<} C_{max}^{on} \text{ für } m \ge 3.
\end{aligned}
$$

zu (3.10): $\frac{3n-(t'+2)}{m} \ge n - \frac{t'}{2} \Longrightarrow m \le \frac{3n-t'-2}{(2n-t')/2} = \frac{6n-2t'-4}{2n-t'} = \frac{4n-2t'+2n-4}{2n-t'} = 2 + \frac{2n-4}{2n-t'}$

zu (3.11): $C_{max}^{off} < t' + n$ gilt für alle $m \ge 3$, da:

Fall $m \le 2 + \frac{2n-4}{2n-t'}$: $C_{max}^{off} = t' + \frac{3n-(t'+2)}{m} \overset{m \ge 3}{<} t' + \frac{3n-(t'+2)}{3} = t' + n - \frac{t'+2}{3} \overset{t'>0}{<} t' + n$

Fall $m \ge 2 + \frac{2n-4}{2n-t'}$: $C_{max}^{off} = t' + n - \frac{t'}{2} \overset{t'>0}{<} t' + n$.

□

Beispiel: m=3 Prozessoren stehen zur Bearbeitung von n=10 Aufträgen $J_1, \ldots, J_{10}$ zur Verfügung. J_1 und J_2 haben jeweils eine Bearbeitungszeit von $n = 10$ Zeiteinheiten, alle anderen $n - 2$ Aufträge haben jeweils eine Bearbeitungszeit von 1 Zeiteinheit. Zu Beginn steht lediglich Prozessor P_1 zur Verfügung. Nach $t' = 2$ Zeiteinheiten sind alle Prozessoren verfügbar (vgl. Abb. 3.30). Der *Online*-Algorithmus erzielt eine Planlänge von $C_{max}^{on} = t' + n = 2 + 10 = 12$ Zeiteinheiten. Ein optimaler *Offline*-Algorithmus erzielt eine Planlänge von $C_{max}^{off} = t' + n - \frac{t'}{2} = 2 + 10 - 1 = 11$ Zeiteinheiten. □

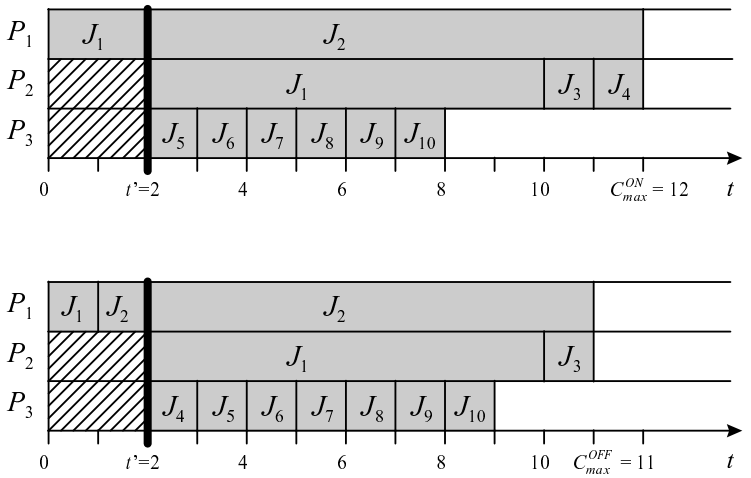

Abbildung 3.30: $C_{max}^{on} < C_{max}^{off}$ *für* $m = 3$ *und* $n = 10,\ t' = 2$

3.2.4.2 Der Algorithmus *Online(ϵ)*

Zur Lösung des Problems $P, NC^{online} \mid pmtn \mid C_{max}$ geben Albers und Schmidt in [AS01] einen Algorithmus *Online(ϵ)* an (vgl. Algorithmus 3.8).

Algorithmus 3.8: *Online(ϵ)*

begin
1. $t := 0$; $\delta = \epsilon/n^2$;
2. **for** $(j := 1, \ldots, n)$ **do** $r_j := p_j$;
3. **while** (es gibt noch Aufträge mit positiver Restbearbeitungszeit) **do**
4. **begin**
5. $m_k :=$ Anzahl der zum Zeitpunkt t verfügbaren Prozessoren;
6. $n_k :=$ Anzahl der Aufträge mit positiver Restbearbeitungszeit;
7. $S :=$ Menge der $\min\{m_k, n_k\}$ Aufträge mit den größten Restbearbeitungszeiten;
8. Verplane die Aufträge $J_j, j \in S$ auf den verfügbaren Prozessoren;
9. **if** (Menge der verfügbaren Prozessoren ändert sich zu einem Zeitpunkt $t + \delta', \delta' < \delta$)
10. **then**
11. Setze $r_j := \max\{0, r_j - \delta'\}$ für $j \in S$; $t := t + \delta'$;
12. **else**
13. Setze $r_j := \max\{0, r_j - \delta\}$ für $j \in S$; $t := t + \delta$;
14. **end;**

end;

Algorithmus 3.8 erstellt Ablaufpläne mit einer Planlänge von $C_{max}^{off} + \epsilon$ für beliebi-

ges $\epsilon > 0$, wenn zu jedem Zeitpunkt wenigstens ein Prozessor verfügbar ist Dazu bestimmt der Algorithmus jeweils zu einem Zeitpunkt t_k die m_k Aufträge mit den größten Restbearbeitungszeiten und verplant sie auf den verfügbaren Prozessoren im Intervall $[t_k, t_k + \delta)$. Ändert sich zu einem Zeitpunkt $t_k + \delta', \delta' < \delta$ die Menge der verfügbaren Prozessoren, unterbricht *Online(ϵ)* die Bearbeitung der Aufträge und berechnet einen neuen Plan für die nächsten δ Zeiteinheiten. Ansonsten berechnet *Online(ϵ)* erst zum Zeitpunkt $t_k + \delta$ einen neuen Teilplan (vgl. Abb. 3.31).

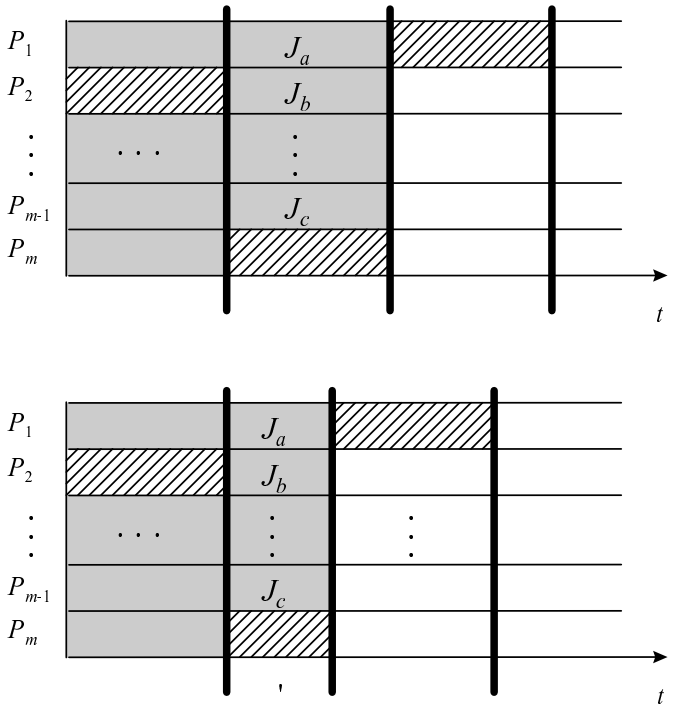

Abbildung 3.31: *Online-Algorithmus*

Im oberen Teil der Abb. 3.31 können alle ausgewählten Aufträge δ Zeiteinheiten verplant werden, da sich in diesem Zeitraum die Menge der verfügbaren Prozessoren nicht verändert. Anders ist die Situation im unteren Teil der Abbildung. Dort fällt nach $\delta' < \delta$ Zeiteinheiten Prozessor P_1 aus und Prozessor P_m wird verfügbar.

Korrektheit, Laufzeit und Anzahl der Präemptionen, die von Algorithmus 3.8 im *worst-case* erzeugt werden, werden in folgendem Satz 3.7 bestimmt und mit Hilfe der Lemmas 3.12-3.14 bewiesen:

Satz 3.7 *Sei $\delta = \epsilon n^2$ die Intervallänge, in der der Algorithmus Online(ϵ) versucht, die Aufträge einzuplanen. Online(ϵ) erzeugt für das Problem $P, NC^{online} \mid pmtn \mid C_{max}$ mit einem Zeitaufwand von $O(Sm \log n + (p_{avg}/\delta)n \log n)$ einen zulässigen Ablaufplan mit einer Planlänge von höchstens $C_{max}^{off} + \epsilon$ und mit höchstens $\sum_{k=1}^{S-1} m_k + (p_{avg}/\delta)n = O(Sm + (p_{avg}/\delta)n)$ Präemptionen.*

Lemma 3.12 *Der Algorithmus Online(ϵ) erzeugt für das Problem $P, NC^{offline} \mid pmtn \mid C_{max}$ einen zulässigen und bezüglich der Planlänge fast optimalen Ablaufplan mit einer Planlänge von höchstens $C_{max}^{off} + \epsilon$.*

Beweis: [AS01], *Theorem* 3 auf S. 98–99. □

Lemma 3.13 *Der Algorithmus Online(ϵ) besitzt eine Laufzeit von $O(Sm \log n + (p_{avg}/\delta)n \log n)$.*

Beweis: [AS01], S. 95, mit $p_{avg} = \sum p_j/n$ und $\delta = \epsilon n^2$. Da angenommen wird, daß ein Algorithmus die Informationen über die Verfügbarkeiten als zusätzliche Eingabe erhält, entfallen Update-Zeiten. □

Lemma 3.14 *Der Algorithmus Online(ϵ) erzeugt höchstens $\sum_{k=1}^{S-1} m_k + \lfloor \sum p_j/\delta \rfloor = O(Sm + (p_{avg}/\delta)n)$ Präemptionen.*

Beweis: In [AS01], S. 95, wird angemerkt, daß *Online(ϵ)* $\sum_{k=1}^{S-1} m_k + (\sum_{j=1}^{n} p_j)n^2/\epsilon, \delta = \epsilon/n^2$ Präemptionen erzeugt. Aus der Analyse des Algorithmus ergibt sich, daß $\epsilon = \delta n^2$ gilt, wobei δ die Länge der Intervalle ist, in denen *Online(ϵ)* die Aufträge jeweils verplant. Mit p_{avg} := die durchschnittliche Bearbeitungszeit aller Aufträge, ergeben sich $O(Sm + /p_{avg}/\delta)n)$ Präemptionen. Die Aufträge werden jeweils in Blöcken von δ Zeiteinheiten verplant, es sei denn, daß sich die Menge der verfügbaren Prozessoren ändert. Wenn alle Änderungen der Menge der verfügbaren Prozessoren nach jeweils einem Vielfachen von δ Zeiteinheiten erfolgen, erhält man genau $\lfloor \sum p_j/\delta \rfloor$ Präemptionen. Im anderen Fall erhält man zusätzliche Präemptionen. Wenn alle Änderungen

der Menge der verfügbaren Prozessoren zu Zeiten erfolgen, daß alle Aufträge zusätzlich unterbrochen werden müssen, kommen $\sum_{k=1}^{S-1} m_k$ Präemptionen hinzu. In diesem Fall werden allerdings immer auch Teile von Aufträgen verplant. Daher wird angenommen, daß ein Wechsel der Prozessorverfügbarkeiten lediglich $\alpha > 0$ Zeiteinheiten nach einer kompletten Einplanung von Aufträgen über δ Zeiteinheiten erfolgt. Die dazu benötigte Prozessorkapazität kann bei hinreichend kleinem α vernachlässigt werden. Außerdem wird angenommen, daß kein Auftrag vor dem letzten Systemintervall fertig wird, im letzten Systemintervall also noch β Zeiteinheiten zur Verplanung der Reste aller Aufträge benötigt werden. So wird sichergestellt, daß in den Systemintervallen $[t_1, t_2), \ldots, (t_{S-1}, t_S]$ echte Präemptionen gezählt werden. Die dazu benötigte Prozessorkapazität kann bei hinreichend kleinem β vernachlässigt werden. Mit $\sum p_j = \sum_{j=1}^{n} p_j = p_{avg} n$ erhält man so maximal $\sum_{k=1}^{S-1} m_k + \lfloor \sum p_j / \delta \rfloor = O(Sm + (p_{avg}/\delta)n)$ Präemptionen. □

Folgendes Beispiel verdeutlicht, daß die Schranke scharf ist:

Beispiel: Es sollen fünf Aufträge J_1, J_2, J_3, J_4, J_5 mit Bearbeitungszeiten von je 4.03 Zeiteinheiten mit dem Algorithmus *Online(ϵ)* verplant werden. Es gibt $S = 5$ Zeitpunkte $t_1 = 0, t_2 = 2.01, t_3 = 4.02, t_4 = 6.03$ und $t_3 = 8.04$, zu denen sich die Menge der verfügbaren Prozessoren ändert, und es gibt $S = 5$ Systemintervalle $[0, 2.01)$, $[2.01, 4.02)$, $[4.02, 6.03)$, $[6.03, 8.04)$ und $[8.04, \infty)$. Im ersten, dritten und fünften Intervall sind $m_1 = m_3 = m_5 = 3$ Prozessoren verfügbar, im zweiten und vierten Intervall sind $m_2 = m_4 = 2$ Prozessoren verfügbar (vgl. Abb. 3.32). Der Algorithmus *Online(ϵ)*

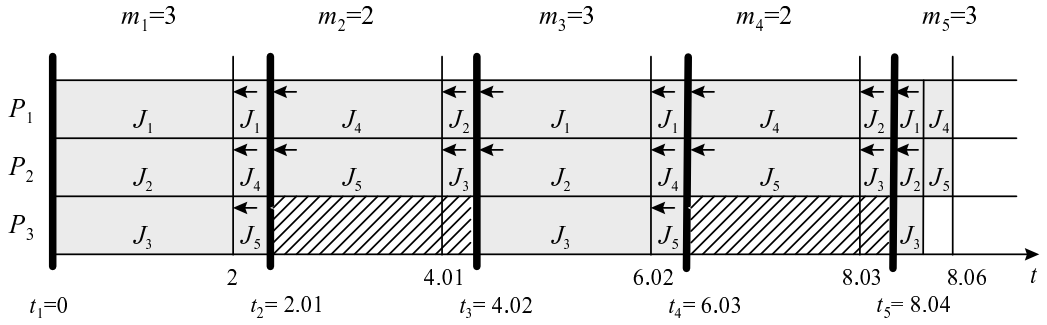

Abbildung 3.32: *Präemptionen von Online(ϵ)*

versucht jeweils, $\delta = 2$ Zeiteinheiten der Aufträge mit der größten Restbearbeitungszeit zu verplanen. Ändert sich die Menge der verfügbaren Prozessoren, wird ein neuer Teilplan erzeugt. Ein Wechsel der Prozessorverfügbarkeiten erfolgt im Beispiel lediglich $\alpha = 0.01$ Zeiteinheiten nach einer kompletten Einplanung von Aufträgen über δ Zeiteinheiten. Im letzten Systemintervall werden noch $\beta = 0.02$ Zeiteinheiten zur Verplanung der Reste aller Aufträge benötigt. Die Anzahl der Präemptionen beträgt $\lfloor P/\delta \rfloor + \sum_{k=1}^{S-1} m_k = 20$. Das Beispiel zeigt, daß die *worst-case* Schranke erreicht werden kann. □

Im vorliegenden Kapitel wurden die beiden klassischen Ablaufplanungsprobleme $P \mid pmtn \mid C_{max}$ und $F2 \mid pmtn \mid C_{max}$ hinsichtlich beschränkt verfügbarer Prozessoren untersucht. Das nächste Kapitel beschäftigt sich mit dem Problem, die beiden klassischen Ablaufplanungsprobleme in der Hinsicht zu erweitern, daß lediglich Ablaufpläne mit einer beschränkten Anzahl von Präemptionen zugelassen sind.

Kapitel 4

Beschränkte Anzahl der Präemptionen

In der Theorie der Ablaufplanung spricht man von *präemtiven Ablaufplänen* (*lat. prae+emere = etwas (weg-)nehmen, bevor es jemand anderes tut*), wenn der Ablaufplaner das Recht hat, einem Prozessor die Bearbeitung eines Auftrags zu jedem beliebigen Zeitpunkt zu entziehen und den Rest des Auftrags sofort oder zu einem späteren Zeitpunkt auf demselben oder auf einem anderen Prozessor einzuplanen. Eine Beschränkung der Anzahl der Präemptionen ist aus mehreren Gründen sinnvoll. Wenn auch keine direkten zeitlichen Kosten (beispielsweise Rüstkosten) mit einer Präemption verbunden sind, so entstehen doch Kosten für zusätzliche Betriebsmittel, die benötigt werden, um einen Auftrag von einem Prozessor zu nehmen, zwischenzulagern und wieder auf einem anderen Prozessor aufzusetzen. Neben diesen Kosten spielen Qualitätsverbesserungen, weniger Eingriffe in eine laufende Produktion und generell eine vereinfachte Produktionsplanung eine Rolle, wenn Ablaufpläne mit möglichst wenig Präemptionen erstellt werden sollen.

In Abschnitt 4.1 wird ein Ablaufplanungsproblem in einem Flowshop mit zwei Prozessoren hinsichtlich einer beschränkten Anzahl der Präemptionen untersucht. Es ist klar, daß das Problem $F2 \mid i\text{-}pmtn \mid C_{max}$ für alle $i \geq 0$ stets eine optimale Lösung besitzt, da das Problem $F2 \mid\mid C_{max}$ mit Hilfe des Johnson-Algorithmus (vgl. Abschnitt 3.1.1)

in polynomialer Zeit optimal gelöst werden kann und Präemptionen keinen Vorteil bringen (vgl. [GS78]).

In Abschnitt 4.2 wird ein Ablaufplanungsproblem mit parallelen, identisch qualifizierten Prozessoren hinsichtlich einer Beschränkung der Anzahl der Präemptionen untersucht. Sind $i = m - 1$ oder mehr Präemptionen zugelassen, kann das Problem $P \mid i\text{-}pmtn \mid C_{max}$ mit dem Verfahren von McNaughton (vgl. Abschnitt 3.2.1) in polynomialer Zeit optimal gelöst werden. Sind keine ($i = 0$) Präemptionen zugelassen, ist man zur Lösung des Problems $P \mid i\text{-}pmtn \mid C_{max}$ auf Approximationsalgorithmen angewiesen, da das Problem mit $i = 0$ *NP-schwer* für bereits $m = 2$ Prozessoren ist (Restriktion auf PARTITION [Kar72]). In Abschnitt 4.2.1 werden die beiden Approximationsalgorithmen von Graham (*List Scheduling* und *LPT-Regel*) vorgestellt und gezeigt, daß *LPT* eine Gütegarantie von $2-2/(m+1)$ bezüglich der optimalen präemptiven Planlänge besitzt.

In Abschnitt 4.2.2 wird die Länge von optimalen präemptiven und i-präemptiven Ablaufplänen verglichen und die folgende grundlegende Frage beantwortet: Um wieviel, bezüglich der Planlänge, besser können optimale präemptive Pläne im Vergleich zu optimalen i-präemptiven Plänen sein? Oder, genauer, was ist für alle Instanzen I die kleinste obere Schranke für den Quotienten $C^{ip^*}_{max}(I)/C^{p^*}_{max}(I)$, wobei $C^{ip^*}_{max}$ und $C^{p^*}_{max}$ die optimalen i-präemptiven und die optimalen präemptiven Planlängen bezeichnen.

4.1 $F2 \mid i\text{-}pmtn \mid C_{max}$

Das Problem $F2 \mid i\text{-}pmtn \mid C_{max}$ läßt sich wie folgt beschreiben. Es sollen n Aufträge $J_1, \ldots, J_n$ auf zwei parallelen, identisch qualifizierten Prozessoren P_1, P_2 so verplant werden, daß die Planlänge C_{max} minimiert wird. Dabei besteht jeder Auftrag J_j aus zwei Verrichtungen $T_{1,j}$ und $T_{2,j}$ mit nichtnegativen Bearbeitungszeiten $p_{1,j}$ und $p_{2,j}, j \in \{1, \ldots, n\}$. Wenn eine Verrichtung unterbrochen wird, darf sie zum gleichen oder zu einem späteren Zeitpunkt auf demselben oder auf einem anderen Prozessor ohne Zeitstrafe weiterbearbeitet werden. Ein i-präemptiver Ablaufplan enthält höchstens i Präemptionen.

Satz 4.1 *Für alle Instanzen I der Ablaufplanungsprobleme $F2 \mid i\text{-}pmtn \mid C_{max}$, $F2, NC \mid pmtn \mid C_{max}$ und $F2 \mid pmtn \mid C_{max}$ gilt bezüglich der optimalen Planlängen stets:*

$$C_{max}^{ip^*} = C_{max}^{np^*} = C_{max}^{p^*}.$$

Beweis: Aus [Joh54], *Theorem* 1, S. 63, folgt, daß das Problem $F2 \mid\mid C_{max}$ optimal durch nichtpräemptive Pläne gelöst wird. In [GS78], *Lemma* 3, S. 40–41, wird gezeigt, daß eine Johnson-Permutation, die das Problem $F2 \mid\mid C_{max}$ optimal löst, auch das Problem $F2 \mid pmtn \mid C_{max}$ optimal löst. □

4.2 $P \mid i\text{-}pmtn \mid C_{max}$

Das Problem $P \mid i\text{-}pmtn \mid C_{max}$ läßt sich wie folgt beschreiben. Es sollen n Aufträge $J_1, \ldots, J_n$ mit nichtnegativen Bearbeitungszeiten $p_1, \ldots, p_n$ auf m parallelen, identisch qualifizierten Prozessoren $P_1, \ldots, P_m$ so verplant werden, daß die Planlänge C_{max} minimiert wird. Wenn ein Auftrag unterbrochen wird, darf er zum gleichen oder zu einem späteren Zeitpunkt auf demselben oder auf einem anderen Prozessor ohne Zeitstrafe weiterbearbeitet werden. Ein i-präemptiver Ablaufplan enthält höchstens i Präemptionen. O.B.d.A. sind Präemptionen nur auf den Prozessoren $P_2, \ldots, P_{i+1}$ zulässig.

Verwandte Ergebnisse

Präemptive und nichtpräemptive Ablaufplanungsalgorithmen zur Lösung des Problems, n Aufträge auf m parallelen, identisch qualifizierten Prozessoren so zu verplanen, daß die Planlänge minimiert wird, gehören zu den am besten untersuchten Ablaufplanungsalgorithmen (vgl. [BEPSW01], [GJ79], [Cof76], [Ull76]).

Mit dem Verfahren von McNaughton [McN59], vgl. Abschnitt 3.2.1, lassen sich optimale präemptive Pläne in (in der Anzahl der Aufträge) polynomialer Zeit erzeugen. Das entsprechende nichtpräemptive Problem ist *NP-schwer* für $m = 2$ Prozessoren [Kar72] und sogar *NP-schwer im strengen Sinne* für beliebig viele Prozessoren [GJ79].

Approximationsalgorithmen liefern mit moderatem Rechenaufwand brauchbare Ergebnisse bezüglich der Lösungsgüte. Graham zeigt in [Gra66], daß *List Scheduling*, in der die Aufträge der Reihe nach so verplant werden, daß immer der Prozessor, der gerade die wenigste Last hat, neu belegt wird, eine Lösungsgüte von $2 - 1/m$ besitzt. Werden die Aufträge nach dem Verplanen mit *List Scheduling* paarweise solange vertauscht, bis keine Verbesserung der Planlänge mehr möglich ist, zeigen Graham *et al.* [GLLRK79], daß die Schranke sich lediglich auf $2 - 2/(m+1)$ verbessert. Der Approximationsalgorithmus *LPT-Regel*, in dem die Aufträge in nichtsteigender Reihenfolge ihrer Bearbeitungszeiten verplant werden, erreicht eine Schranke von $4/3 - 1/3m$ [Gra69]. Goldberg und Shapiro [GS01] zeigen, daß neben *LPT* auch die Algorithmen *Best Fit*, *First Fit*, *Random Fit* und *Greedy* zu einer Klasse von Algorithmen gehören, die alle die Schranke von $4/3 - 1/3m$ erreichen.

Diese *worst-case* Ergebnisse sind jedoch pessimistisch und nicht aussagekräftig bezüglich der Güte der beiden Approximationsalgorithmen in der Praxis. Achugbue und Chin [AC81] untersuchen die Güte von *List Scheduling* in Abhängigkeit von $\rho = p_{max}/p_{min}$. Für $\rho \leq 2$ ist die Schranke $3/2$ für $m = 2, 3$ und $5/3 - 1/(3\lfloor m/2 \rfloor)$ für $m \geq 4$. Für $\rho \leq 2$ ist die Schranke $5/3$ für $m = 3, 4$, $17/10$ für $m = 5$ und $2 - 1/(3\lfloor m/2 \rfloor)$ für $m \geq 4$.

Unter der Annahme, daß die Bearbeitungszeiten p_j unabhängige, identisch verteilte nichtnegative Zufallsvariablen sind, die aus einer uniformen Verteilung aus $(0, 1]$ gezogen werden, zeigen Coffman *et al.* [CFL84], daß die *LPT-Regel* asymptotisch optimal ist. Darüberhinaus zeigen Frenk und Rinnooy Kan [FRK87], daß die *LPT-Regel* fast sicher optimal ist, falls die p_j eine Verteilungsfunktion mit endlichem Mittelwert besitzen. Kedia [Ked70] kommt zum gleichen Resultat auf Grund von Simulationsstudien.

Ibarra und Kim [IK77] untersuchen die Güte von *LPT* in Abhängigkeit von $\rho = p_{max}/p_{min}$. Sie zeigen, daß für $\rho \leq n/2(m-1)$ gilt: $C_{max}^{LPT} \leq ((1+2(m-1)/n)C_{max}^{np^*}$. Allerdings ist diese Schranke asymptotisch nicht besser als die von Graham (vgl. [CS90]). Dobson [Dob84] zeigt, daß für Bearbeitungszeiten aus dem Intervall $[0, (1/k)C_{max}^{np^*}]$ gilt: $C_{max}^{LPT} \leq ((k+3)/(k+2))C_{max}^{np^*}$. Kao und Elsayed [KE88] zeigen, daß für Bearbeitungs-

zeiten $p_i \leq (m/(m-1))p_{i-1}$ für $i = 2, \ldots, n$ gilt: $C_{max}^{LPT} \leq (1 + (m-1)/n)C_{max}^{np^*}$.

Coffman und Sethi [CS76] verallgemeinern die Graham-Schranke in der Hinsicht, daß die Anzahl k der Aufträge, die auf dem die Planlänge bestimmenden Prozessor verplant werden, berücksichtigt wird. Sie zeigen, daß gilt: $C_{max}^{LPT} \leq (1+1/k-1/km)C_{max}^{np^*}$. Chen [Che93] zeigt, daß genauer gilt: $C_{max}^{LPT} = C_{max}^{np^*}$ für $k = 1$, $C_{max}^{LPT} \leq 4/3 - 1/3(m-1)$ für $k = 2$ und $C_{max}^{LPT} \leq (1 + 1/k - 1/km)C_{max}^{np^*}$ für $k \geq 3$. Blocher und Chand [BC91] verbessern die Coffman/Sethi-Schranke, indem sie zusätzlich die Länge des mit *LPT* erzeugten Plans bei der Schrankenberechnung berücksichtigen.

Greenberg [Gre72] stellt eine Heuristik vor, die auf der *LPT-Regel* basiert. Coffman *et al.* [CGJ78] geben einen Algorithmus *MULTIFIT* an, der *Bin Packing* in Verbindung mit binärer Suche benutzt. Sie zeigen daß *MULTIFIT* eine Gütegarantie von 1.22 besitzt. Friesen [Fri84] verbessert die Schranke auf $1.20 + (1/2)^k$, wobei k die Anzahl der Iterationen der binären Suche ist. Yue [Yue90] verbessert die Schranke auf 13/11. Lee und Massey [LM88] geben einen Algorithmus *COMBINE* an, der *LPT* und *MULTIFIT* verbindet. Sie zeigen, daß die Schranke von *COMBINE* niemals schlechter ist als die von *MULTIFIT*. Elmaghraby und Elimam [EE80] geben einen Algorithmus *KOMP* an, der auf einer Lösung des Rucksackproblems beruht, und zeigen, daß *KOMP* bezüglich ihres Testsettings bessere Resultate als *LPT* und *MULTIFIT* liefert, allerdings mit einem größeren Rechenaufwand. Hochbaum und Shmoys [HS87] geben ein polynomiales Approximationsschema, d.h. einen $(1+\epsilon)$ Approximationsalgorithmus an, der das Problem für jedes beliebige $\epsilon > 0$ löst. Dabei erhalten sie Algorithmen mit einer Güte von schlechtestenfalls 7/6. Da $P \mid\mid C_{max}$ ein im strengen Sinne *NP*-schweres Problem ist [GJ79], gibt es kein vollständig polynomiales Approximationsschema, es sei denn $P = NP$.

Für ein gegebenes System von Aufträgen mit Vorrangbeziehungen hat Liu in [Liu72] die Behauptung aufgestellt, das Verhältnis der optimalen nichtpräemptiven Planlänge zur optimalen präemptiven Planlänge betrage $2 - 2/(m+1)$. Coffman und Garey [CG93] haben die Vermutung von Liu für ein System mit lediglich zwei Prozessoren bewiesen. Hong und Leung [HL92] haben die Korrektheit der Schranke sowohl für Unit

Execution Time (UET) als auch für baumartig strukturierte Vorrangbeziehungen der Aufträge bewiesen. Aus ihrem Beweis folgt die Gültigkeit der $2 - 2/(m+1)$-Schranke für Aufträge ohne Vorrangbeziehungen und damit für $C_{max}^{np^*} \leq (2 - 2/(m+1))C_{max}^{p^*}$.

Im nächsten Abschnitt werden die beiden für die weiteren Ausführungen grundlegenden Ergebnisse von Graham zur Erstellung optimaler nichtpräemptiver Pläne für Instanzen des Problems $P \mid\mid C_{max}$ vorgestellt. Darüberhinaus erfolgt eine Analyse der Gütegarantie von mit Hilfe der *LPT-Regel* erstellten Plänen gegenüber optimalen präemptiven Plänen.

4.2.1 Keine Präemptionen

Wenn man das Problem $P \mid\mid C_{max}$ betrachtet, ist intuitiv klar, daß die Planlänge dann minimal ist, wenn jeder Prozessor die gleiche Last hat. Ein einfacher und schneller Algorithmus, der versucht, die Last möglichst gleichmäßig auf die Prozessoren zu verteilen, ist *List Scheduling*. Dazu werden die Aufträge in beliebiger Reihenfolge in einer Liste angeordnet und der Reihe nach auf dem jeweils ersten verfügbaren Prozessor eingeplant (vgl. Algorithmus 4.1).

Algorithmus 4.1: *List Scheduling*

begin
1. **for** $(i := 1, \ldots, m)$ **do** $F_i := 0$;
2. Finde den Prozessor P_i mit dem kleinsten Bearbeitungsende F_i;
3. Verplane den nächsten Auftrag J_j aus der Liste auf Prozessor P_i im Intervall $[F_i, F_i + p_j]$ und setze $F_i := F_i + p_j$;

end;

In Lemma 4.1 wird eine Aussage bezüglich der Laufzeit von Algorithmus 4.1 getroffen, und in Lemma 4.2 wird die Gütegarantie von *List Scheduling* zu optimalen nichtpräemptiven Plänen angegeben.

Lemma 4.1 *Die Laufzeit von List Scheduling beträgt $O(n)$.*

Beweis: Von m Prozessoren $P_1, \ldots, P_m$ muß man sich jeweils die aktuelle Last $F_i, i = 1, \ldots, m$ merken. Als einzige Operationen werden `ExtractMin()` und `Insert(`F_i`)` benötigt. Wird als Datenstruktur ein *Fibonacci Heap* benutzt, ergeben sich jeweils Kosten $O(1)$ und Gesamtkosten in Höhe von $O(n)$. □

Lemma 4.2 *Sei C_{max}^{LS} die durch den Algorithmus List Scheduling erzeugte Planlänge. Dann gilt bezüglich der Güte von C_{max}^{LS} im Vergleich zur Planlänge $C_{max}^{np^*}$ optimaler nichtpräemptiver Abpaufpläne:*

$$C_{max}^{LS} \leq (2 - 1/m) C_{max}^{np^*}.$$

Beweis: [Gra66], *Theorem* 1 auf S. 1570ff. □

Folgendes Beispiel zeigt, daß die in Lemma 4.2 angegebene Schranke $C_{max}^{LS} \leq \left(2 - \frac{1}{m}\right) C_{max}^{np^*}$ scharf ist:

Beispiel: n Aufträge sollen auf m Prozessoren bearbeitet werden mit $n = 2m - 1$ und folgenden Bearbeitungszeiten: $p_j = m-1$ für $1 \leq j \leq m-1$, $p_j = 1$ für $m \leq j \leq 2m-2$, $p_j = m$ für $j = 2m - 1$ (vgl. Abb. 4.1).

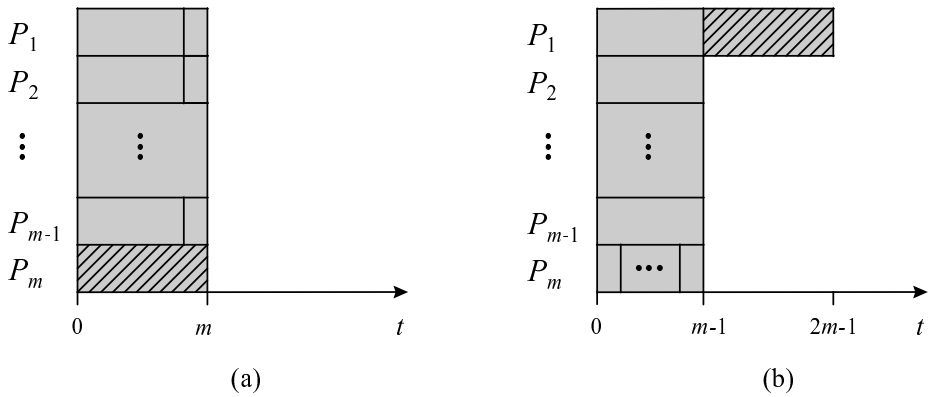

Abbildung 4.1: *Optimaler nichtpräemptiver vs. mit List Scheduling erstellter Ablaufplan*

Alle verplanten Aufträge sind hellgrau gekennzeichnet, der Auftrag J_k, der die Planlänge bestimmt, ist zusätzlich schraffiert. In Abb. (a) werden die Aufträge optimal nichtpräemptiv verplant, und in Abb. (b) werden die Aufträge mit *List Scheduling* verplant. Der optimale nichtpräemptive Ablaufplan hat Länge m, wohingegen der mit *List Scheduling* erzeugte Ablaufplan Länge $2m-1$ hat. Insgesamt ergibt sich

$$\frac{C_{max}^{LS}}{C_{max}^{np^*}} = \frac{2m-1}{m} = 2 - \frac{1}{m}.$$

□

In Abb. 4.1(b) sieht man, daß der schlechteste Fall auftritt, weil der längste Auftrag als letzter verplant wird. Werden die Aufträge vorher in nichtaufsteigender Reihenfolge ihrer Bearbeitungszeiten sortiert, wird genau dieser Fall vermieden (vgl. Algorithmus 4.2).

Algorithmus 4.2: *LPT*

begin
1. Sortiere die Aufträge in *LPT* Reihenfolge, d.h. in nichtaufsteigender Reihenfolge ihrer Bearbeitungszeiten;
2. Wende Algorithmus *List Scheduling* auf die umgeordnete Auftragsmenge an;

end;

In den beiden folgenden Lemmas werden Laufzeit und Güte von *LPT* abgeschätzt:

Lemma 4.3 *Die Laufzeit von LPT beträgt* $O(n \log n)$.

Beweis: Im Vergleich zu *List Scheduling* wird als einzige zusätzliche Operation die Sortierung von n Aufträgen (Zeit $O(n \log n)$) benötigt. □

Lemma 4.4 *Sei* C_{max}^{LPT} *die durch den Algorithmus LPT erzeugte Planlänge. Dann gilt bezüglich der Güte von* C_{max}^{LPT} *im Vergleich zur Planlänge* $C_{max}^{np^*}$ *optimaler nichtpräemptiver Abpaufpläne:*

$$C_{max}^{LPT} \leq (4/3 - 1/3m) C_{max}^{np^*}.$$

Beweis: [Gra69], *Theorem* 2 auf S. 421 ff. □

Um zu zeigen, daß die Schranke $C_{max}^{LPT} \leq \left(\frac{4}{3} - \frac{1}{3m}\right) C_{max}^{np^*}$ scharf ist, betrachten wir folgendes Beispiel.

Beispiel: n Aufträge sollen auf m Prozessoren bearbeitet werden mit $n = 2m - 1$ und folgenden Bearbeitungszeiten: $p_{2j-1} = p_{2j} = 2m - j$ für $1 \leq j \leq m$, $p_{2m+1} = m$ (vgl. Abb. 4.2).

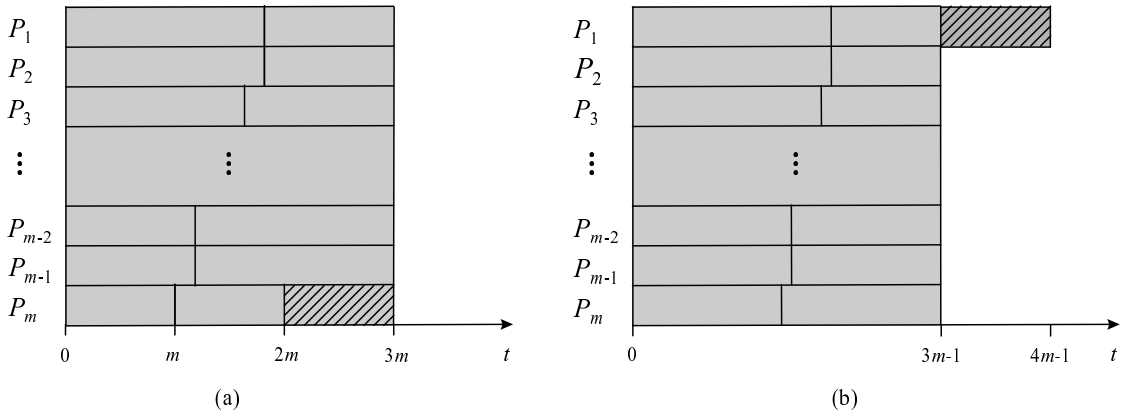

Abbildung 4.2: *Optimaler nichtpräemptiver vs. mit LPT erstellter Ablaufplan*

Der Auftrag J_k, der die Planlänge bestimmt, ist jeweils schraffiert. In Abb. 4.2(a) werden die Aufträge nichtpräemptiv optimal verplant und in Abb. 4.2(b) mit Hilfe der *LPT-Regel*. Der optimale nichtpräemptive Ablaufplan hat Länge $4m - 1$, wohingegen der mit LPT erzeugte Ablaufplan Länge $3m$ hat. Insgesamt ergibt sich

$$\frac{C_{max}^{LPT}}{C_{max}^{np^*}} = \frac{4m-1}{3m} = \frac{4}{3} - \frac{1}{3m}.$$

□

Die Gütegarantie von LPT zu optimalen nichtpräemptiven Plänen ist also bekannt. In Satz 4.2 wird gezeigt, daß das Verhältnis der Länge eines mit LPT erzeugten Ablaufplans zur Länge eines optimalen präemptiven Ablaufplans nach oben durch $2-2/(m+1)$ beschränkt ist.

Satz 4.2 *Sei C_{max}^{LPT} die durch den Algorithmus LPT erzeugte Planlänge. Dann gilt bezüglich der Güte von C_{max}^{LPT} im Vergleich zur Planlänge $C_{max}^{p^*}$ optimaler präemptiver Abpaufpläne:*

$$C_{max}^{LPT} \leq \left(2 - \frac{2}{m+1}\right) C_{max}^{p^*}.$$

Beweis: Sei J_k der Auftrag, der die Planlänge des mit der *LPT-Regel* erzeugten Planes bestimmt. Wir beginnen mit zwei Beobachtungen. Die erste Beobachtung folgt direkt aus [McN59] und liefert untere Schranken für $C_{max}^{p^*}$ (vgl. Abschnitt 3.2.1):

$$C_{max}^{p^*} = \max\left\{ \frac{\sum p_j}{m} \quad , \quad p_{max} \right\}$$

Zweite Beobachtung: Kein Prozessor kann vor s_k verfügbar sein, da sonst die Planlänge durch Verschieben von J_k kleiner würde oder gleich bliebe (vgl. Abb. 4.3, J_k ist schraffiert).

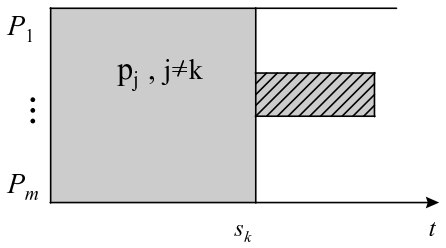

Abbildung 4.3: *s_k ist nie größer als $\left(\sum p_j, j \neq k\right)/m$*

Daher gilt

$$s_k \leq \frac{\sum p_j, j \neq k}{m}.$$

Falls $p_k \leq (m/(m+1))C_{max}^{p^*}$, gilt für die Planlänge des mit Hilfe der *LPT-Regel* erstellten Ablaufplans:

$$C_{max}^{LPT} = s_k + p_k$$

$$\leq \frac{\sum p_j - p_k}{m} + p_k$$

$$
\begin{aligned}
&= \frac{\sum p_j}{m} + \left(1 - \frac{1}{m}\right) p_k \\
&\leq C^{p^*}_{max} + \left(1 - \frac{1}{m}\right) C^{p^*}_{max} \left(\frac{m}{m+1}\right) \\
&= \left(1 + \frac{m-1}{m+1}\right) C^{p^*}_{max} \\
&= \left(1 + \frac{m+1-2}{m+1}\right) C^{p^*}_{max} \\
&= \left(2 - \frac{2}{m+1}\right) C^{p^*}_{max}.
\end{aligned}
$$

Falls $p_k > (m/(m+1))C^{p^*}_{max}$, gilt:

$$
\begin{aligned}
& & p_k &> C^{p^*}_{max} \left(\frac{m}{m+1}\right) \\
\Longrightarrow & & p_k &> \left(\frac{\sum p_j}{m}\right) \left(\frac{m}{m+1}\right) \\
\Longrightarrow & & p_k &> \frac{\sum p_j}{m+1} \\
\Longrightarrow & p_k(m+1) &> \sum p_j \\
\Longrightarrow & p_k m &> \sum p_j - p_k \\
\Longrightarrow & & p_k &> \frac{\sum p_j, j \neq k}{m} \geq s_k \\
\Longrightarrow & & s_k &< p_k.
\end{aligned}
$$

Da der Plan mit Hilfe der *LPT-Regel* erzeugt wird, ist s_k nur dann kleiner als p_k, falls $s_k = 0$ ist. Daher gilt $C^{LPT}_{max} = C^{p^*}_{max}$, falls $p_k > (m/(m+1))C^{p^*}_{max}$. □

Um zu zeigen, daß die Schranke $C_{max}^{LPT} \leq (2 - 2/(m+1))C_{max}^{p^*}$ scharf ist, betrachten wir folgendes Beispiel:

Beispiel: n Aufträge sollen auf m Prozessoren bearbeitet werden mit $n = m+1$ und $p_j = c, j = 1, \ldots, n$ (vgl. Abb. 4.4).

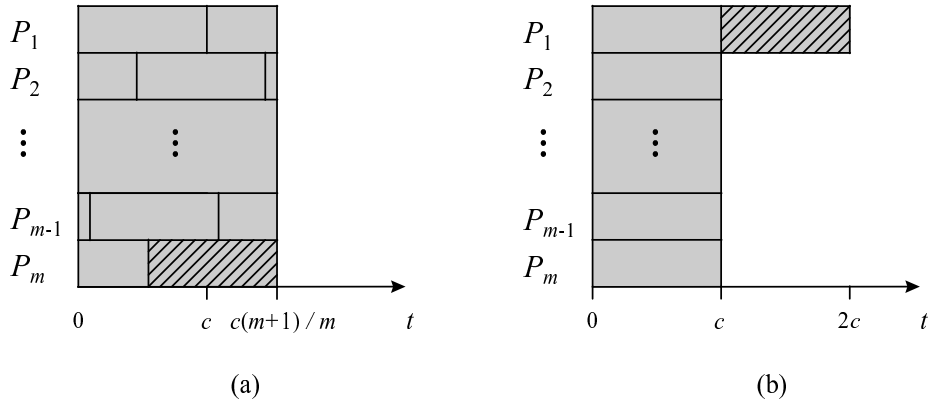

Abbildung 4.4: *Optimaler präemptiver vs. optimaler mit LPT erzeugter Ablaufplan für $m+1$ Aufträge der Längen c*

Der Auftrag J_k mit $C_k = C_{max}^{ip^*}$ ist jeweils grau markiert. In Abb. (a) werden die Aufträge mit McNaughtons Algorithmus optimal präemptiv verplant, und in Abb. (b) werden die Aufträge mit der *LPT-Regel* verplant. Der optimale präemptive Ablaufplan hat Länge $c(m+i+1)/m$, wohingegen der mit LPT erzeugte Ablaufplan Länge $2c$ hat. Insgesamt ergibt sich

$$\frac{C_{max}^{LPT}}{C_{max}^{p^*}} = \frac{2c}{c(m+1)/m} = \frac{2m}{m+1}.$$

□

4.2.2 Beschränkte Anzahl der Präemptionen

Intuitiv ist klar, daß ein optimaler prämptiver Plan im allgemeinen kürzer ist als ein optimaler nichtpräemptiver Plan (vgl. Abb. 4.5 aus [Set76]).

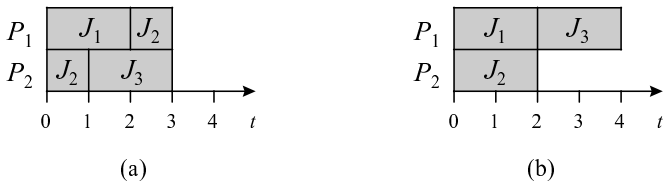

Abbildung 4.5: *Präemptiver* (a) *und nichtpräemptiver* (b) *Ablaufplan für drei Aufträge*

Aus [McN59] folgt, daß, falls i=m-1 oder mehr Präemptionen zugelassen sind, stets bezüglich der Planlänge optimale Ablaufpläne erstellt werden können. Es gilt also $C_{max}^{ip^*} = C_{max}^{p^*}$, für alle $i \geq m-1$. Daher wird im folgenden lediglich der Fall $0 \leq i \leq m-1$ untersucht und gezeigt, daß in diesem Fall $C_{max}^{ip^*} \leq (2 - 2/(m/(i+1)+1))C_{max}^{p^*}$ gilt.

Satz 4.3 *Sei S ein optimaler i-präemptiver Ablaufplan mit Planlänge $C_{max}^{ip^*}$ und $0 \leq i \leq m-1$ die maximale Anzahl zugelassener Präemptionen. Dann gilt für alle Instanzen des Problems $P \mid i\text{-}pmtn \mid C_{max}$*

$$C_{max}^{ip^*} \leq \left(2 - \frac{2}{\left(\frac{m}{i+1}\right)+1}\right) C_{max}^{p^*}.$$

Beweis: Falls $p_k \geq \sum p_j/m$ ist, gilt $C_{max}^{ip^*} = C_{max}^{p^*}$, da ein Ablaufplan nicht kürzer sein kann als p_k. Daher wird im folgenden angenommen, daß $p_k < \sum p_j/m$ gilt. In Lemma 4.5 wird gezeigt, daß es in einem optimalen i-präemptiven Ablaufplan S stets einen Auftrag J_k mit Bearbeitungszeit p_k und Bearbeitungsende $C_k = C_{max}^{ip^*}$ gibt, der nicht unterbrochen wird. O.B.d.A. wird angenommen, daß Auftrag J_k entweder auf Prozessor P_{i+1} oder auf Prozessor P_{i+2} bearbeitet wird (vgl. Abb. 4.6). Die Bearbeitung von J_k beginnt zum Zeitpunkt s_k und endet zum Zeitpunkt C_k. Zu beachten ist, daß lediglich auf den Prozessoren $P_2, \ldots, P_{i+1}$ Präemptionen erlaubt sind.

In Lemma 4.6 wird gezeigt, daß der Startzeitpunkt von Auftrag J_k nicht größer ist als $(\sum p_j - (i+1)p_k)/m$. In Lemma 4.7 und in Lemma 4.8 wird gezeigt, daß im Falle $p_k \leq (m/(m+i+1))C_{max}^{p^*}$ und $p_k > (m/(m+i+1))C_{max}^{p^*}$ die Ungleichung $C_{max}^{ip^*} \leq (2 - 2/(m/(i+1)+1))C_{max}^{p^*}$ erfüllt ist. □

Im folgenden werden die in Satz 4.2 benötigten Lemmas 4.5-4.8 bewiesen.

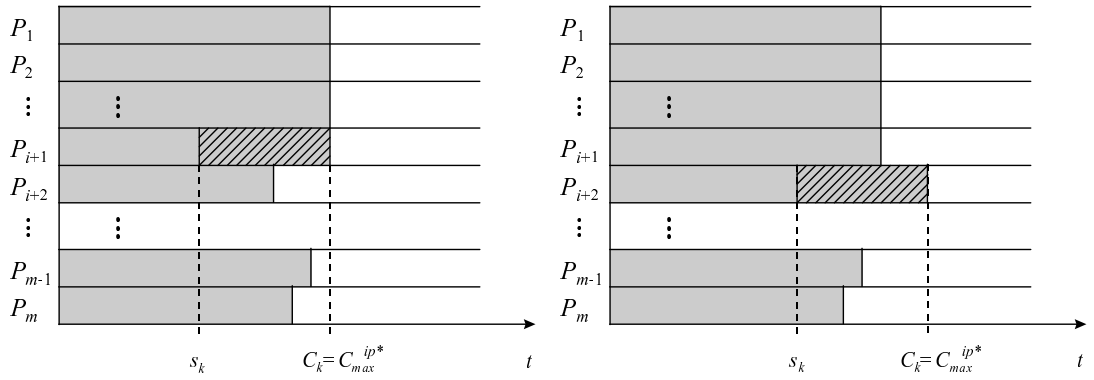

Abbildung 4.6: *Präemptionen sind o.B.d.A. lediglich auf den i Prozessoren $P_2, \ldots, P_{i+1}$ erlaubt. Auftrag J_k, der die Planlänge C_{max}^{ip*} bestimmt, wird o.B.d.A. entweder auf Prozessor P_{i+1} oder auf Prozessor P_{i+2} bearbeitet. Die Bearbeitung von J_k beginnt zum Zeitpunkt s_k und endet zum Zeitpunkt C_k.*

Lemma 4.5 *Sei J_k ein Auftrag mit Bearbeitungszeit $p_k < \sum p_j/m$ und Bearbeitungsende $C_k = C_{max}^{ip*}$, mit $0 \leq i \leq m-1$ die maximal mögliche Anzahl von Präemptionen. Es gibt immer einen optimalen i-präemptiven Plan S, in dem J_k nicht unterbrochen wird.*

Beweis: Der Beweis erfolgt durch Widerspruch. Es wird gezeigt, daß aus einem i-präemptiven Ablaufplan S, in dem J_k unterbrochen wird, stets ein Ablaufplan S' mit der gleichen Planlänge und der zusätzlichen Eigenschaft, daß J_k nicht unterbrochen ist, erzeugt werden kann. Da ein Auftrag höchstens einmal unterbrochen wird und da Präemptionen lediglich auf den i Prozessoren $P_2, \ldots, P_{i+1}$ erlaubt sind, muß der erste Teil von J_k auf einem der Prozessoren $P_2, \ldots, P_{i+1}$ bearbeitet werden (o.B.d.A. auf Prozessor P_2). Da auf den letzten $m-(i+1)$ Prozessoren $P_{i+2}, \ldots, P_m$ nur nichtunterbrochene Aufträge verplant werden, muß der zweite Teil von J_k auf einem der ersten $i+1$ Prozessoren $P_1, \ldots, P_{i+1}$ (o.B.d.A. auf Prozessor P_{i+1}) bearbeitet werden. S' kann dann aus S wie folgt erstellt werden (vgl. Abb. 4.7):

- Verschiebe den auf Prozessor P_2 verplanten Teil von J_k unmittelbar vor den auf Prozessor P_{i+1} verplanten Teil von J_k.

- Verplane die verbleibende Last der ersten $i+1$ Prozessoren mit dem Algorithmus von McNaughton.

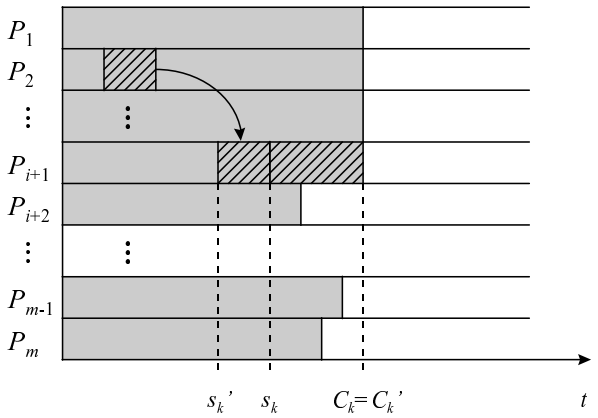

Abbildung 4.7: *In einem optimalen Ablaufplan S gibt es stets einen Auftrag J_k, der nicht unterbrochen wird.*

Resultat ist ein Ablaufplan S' mit den folgenden Eigenschaften:

1. Das Bearbeitungsende der Prozessoren $P_1, \ldots, P_{i+1}$ bleibt bei höchstens C_k, und es werden höchstens i Präemptionen erzeugt.
2. Die Bearbeitungsenden der Prozessoren $P_{i+2}, \ldots, P_m$ bleiben unverändert.

S' erzeugt also die gleiche Planlänge wie S und der Auftrag J_k, der die Planlänge bestimmt, ist nicht unterbrochen. □

Lemma 4.6 *Sei J_k ein Auftrag mit Bearbeitungszeit $p_k < \sum p_j/m$ und Bearbeitungsende $C_k = C_{max}^{ip^*}$ mit $0 \leq i \leq m-1$ die maximal mögliche Anzahl von Präemptionen. Dann gilt für den Startzeitpunkt s_k von J_k*

$$s_k \leq \left(\sum p_j - (i+1)p_k \right) / m$$

Beweis: In Lemma 4.5 wurde gezeigt, daß J_k in einem optimalen i-präemptiven Ablaufplan S nicht unterbrochen wird. Wir unterscheiden zwei Fälle:

Fall 1: In einem optimalen i-präemptiven Ablaufplan S wird J_k auf einem der ersten $i+1$ Prozessoren $P_1, \ldots, P_{i+1}$ (o.B.d.A. auf Prozessor P_{i+1}) bearbeitet.

Zunächst wird gezeigt, daß in einem optimalen i-präemptiven Ablaufplan S die Last der ersten $i+1$ Prozessoren $P_1, \ldots, P_{i+1}$ mindestens $(i+1)C_k$ beträgt. Der Beweis erfolgt durch Widerspruch. Es wird gezeigt, daß aus einem i-präemptiven Ablaufplan S, in dem die Last der ersten $i+1$ Prozessoren $P_1, \ldots, P_{i+1}$ kleiner ist als $(i+1)C_k$, ein Ablaufplan S' mit einer kleineren Planlänge erzeugt werden kann. S' kann aus S wie folgt erstellt werden (o.B.d.A. sei P_1 ein Prozessor mit einem Bearbeitungsende $F_1 < C_k$, vgl. Abb. 4.8):

- Verplane die Last der ersten $i+1$ Prozessoren mit dem Algorithmus von McNaughton.

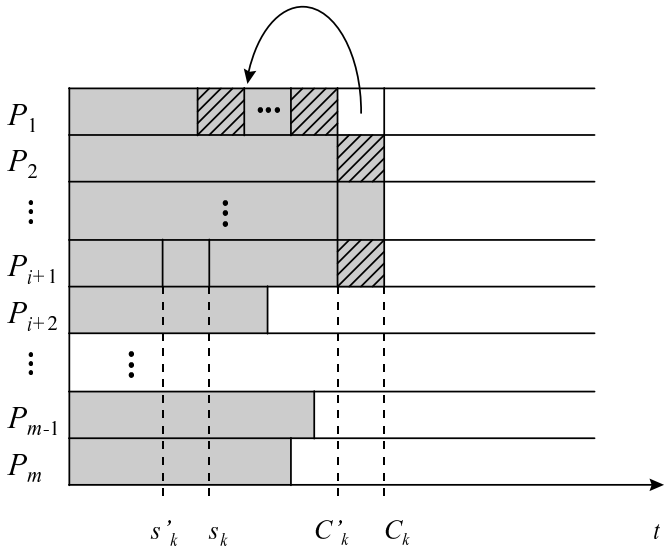

Abbildung 4.8: *Fall 1: Last der ersten $i+1$ Prozessoren ist wenigstens $(i+1)C_k$, wenn J_k auf P_{i+1} verplant wird.*

Resultat ist ein Ablaufplan S' mit den folgenden Eigenschaften:

1. Die Gesamtlast der Prozessoren $P_1, \ldots, P_{i+1}$ bleibt unverändert. Da es erlaubt ist, diese Last mit dem Algorithmus von McNaughton zu verplanen und da es keinen Auftrag J_j gibt mit $p_j = C_{max}^{ip^*}$, ist das Bearbeitungsende der Prozessoren $P_1, \ldots, P_{i+1}$ von Ablaufplan S' kleiner als das Bearbeitungsende C_k von Ablaufplan S und es werden höchstens i Präemptionen erzeugt.
2. Die Bearbeitungsenden der Prozessoren $P_{i+2}, \ldots, P_m$ bleiben unverändert.

Daher kann man einen Ablaufplan S' mit einer kleineren Planlänge als die durch den Ablaufplan S erzeugte Planlänge C_k erstellen und es ist bewiesen, daß die Last der ersten $i+1$ Prozessoren wenigstens $(i+1)C_k$ beträgt.

Im folgenden wird gezeigt, daß in einem optimalen i-präemptiven Ablaufplan S die Last der letzten $m-(i+1)$ Prozessoren $P_{i+2}, \ldots, P_m$ wenigstens $(m-(i+1))s_k$ beträgt. Der Beweis erfolgt durch Widerspruch. Es wird gezeigt, daß aus einem i-präemptiven Ablaufplan S, in dem die Last der letzten $m-(i+1)$ Prozessoren $P_{i+2}, \ldots, P_m$ kleiner ist als $(m-(i+1))s_k$, ein Ablaufplan S' mit einer kleineren Planlänge erzeugt werden kann. S' kann aus S wie folgt erstellt werden (o.B.d.A. sei P_m ein Prozessor mit einem Bearbeitungsende $F_m < s_k$, vgl. Abb. 4.9):

- Verschiebe J_k von Prozessor P_{i+1} auf Prozessor P_m.
- Verplane die verbleibende Last der ersten $i+1$ Prozessoren mit dem Algorithmus von McNaughton.

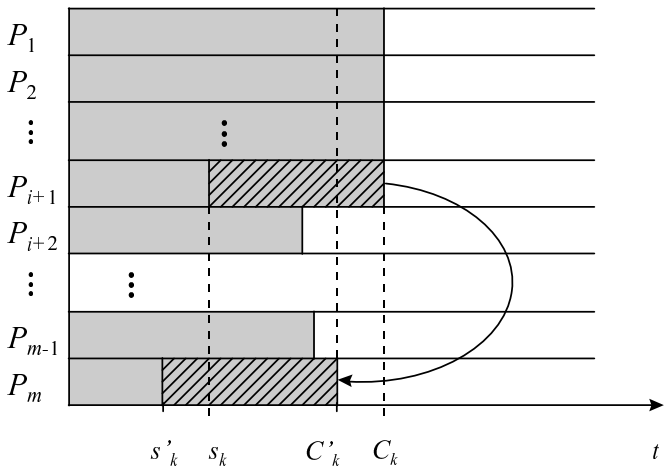

Abbildung 4.9: *Fall 1: Last der letzten $m-(i+1)$ Prozessoren ist wenigstens $m-(i+1)s_k$, wenn J_k auf P_{i+1} verplant wird.*

Resultat ist ein Ablaufplan S' mit den folgenden Eigenschaften:

1. Die Last der ersten $i+1$ Prozessoren $P_1, \ldots, P_{i+1}$ vermindert sich um p_k. Da es erlaubt ist, diese Last mit dem Algorithmus von McNaughton zu verplanen

und da es keinen Auftrag J_j gibt mit $p_j = C_{max}^{ip^*}$, ist das Bearbeitungsende der Prozessoren $P_1, \ldots, P_{i+1}$ von Ablaufplan S' kleiner als das Bearbeitungsende C_k von Ablaufplan S und es werden höchstens i Präemptionen erzeugt.

2. Die Bearbeitungsenden der Prozessoren $P_{i+2}, \ldots, P_m$ bleiben unverändert.
3. Das Bearbeitungsende von Prozessor P_m ist kleiner als C_k, da wegen $F_m < s_k$ gilt $F_m + p_k < s_k + p_k$.

Daher kann man einen Ablaufplan S' mit einer kleineren Planlänge als die durch den Ablaufplan S erzeugte Planlänge C_k erstellen und es ist bewiesen, daß die Last der letzten $m-(i+1)$ Prozessoren wenigstens $(m-(i+1))s_k$ beträgt. Insgesamt ergibt sich

$$\begin{aligned} \sum p_j &\geq (i+1)C_k \quad + \quad (m-(i+1))s_k \\ &\geq (i+1)(s_k+p_k) \quad + \quad (m-(i+1))s_k \\ &= (i+1)p_k + ms_k \end{aligned}$$

oder äquivalent

$$s_k \leq \frac{\sum p_j - (i+1)p_k}{m}$$

Fall 2: In einem optimalen i-präemptiven Ablaufplan S wird J_k auf einem den letzten $m-(i+1)$ Prozessoren $P_{i+2}, \ldots, P_m$ (o.B.d.A. auf Prozessor P_{i+2}) bearbeitet.

Zunächst wird gezeigt, daß in einem optimalen i-präemptiven Ablaufplan S die Last der ersten $i+1$ Prozessoren $P_1, \ldots, P_{i+1}$ mindestens $(i+1)C_k - p_k$ beträgt. Der Beweis erfolgt durch Widerspruch. Es wird gezeigt, daß aus einem i-präemptiven Ablaufplan S, in dem die Last der ersten $i+1$ Prozessoren $P_1, \ldots, P_{i+1}$ kleiner ist als $(i+1)C_k - p_k$, ein Ablaufplan S' mit einer kleineren Planlänge erzeugt werden kann. S' kann aus S wie folgt erstellt werden (o.B.d.A. sei P_{i+1} ein Prozessor mit einem Bearbeitungsende $F_{i+1} < C_k - p_k$, vgl. Abb. 4.10):

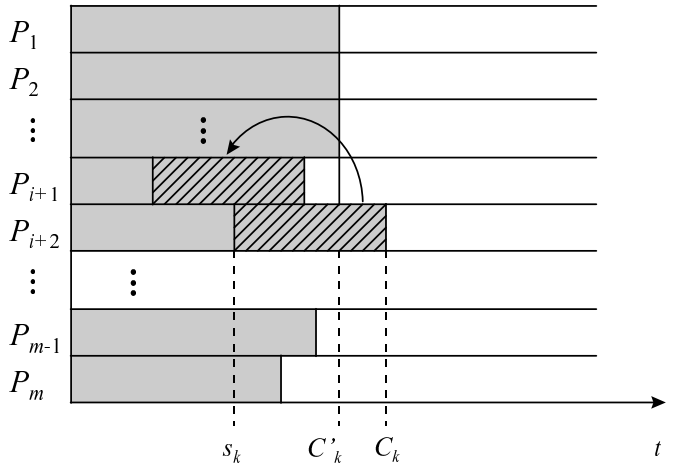

Abbildung 4.10: *Fall 2: Last der ersten $i+1$ Prozessoren ist wenigstens $(i+1)C_k - p_k$, wenn J_k auf P_{i+2} verplant wird.*

- Verschiebe J_k von Prozessor P_{i+2} auf Prozessor P_{i+1}.

Resultat ist ein Ablaufplan S' mit den folgenden Eigenschaften:

1. Die Bearbeitungsenden der Prozessoren $P_1, \ldots, P_i$ bleiben unverändert.
2. Das Bearbeitungsende von Prozessor P_{i+1} ist kleiner als C_k, da wegen $F_{i+1} < C_k - p_k$ gilt $F_{i+1} + p_k < C_k$.
3. Das Bearbeitungsende von Prozessor P_{i+2} vermindert sich um p_k.
4. Die Bearbeitungsenden der Prozessoren $P_{i+3}, \ldots, P_m$ bleiben unverändert.

Daher kann man einen Ablaufplan S' mit einer kleineren Planlänge als die durch den Ablaufplan S erzeugte Planlänge C_k erstellen und es ist bewiesen, daß die Last der ersten $i+1$ Prozessoren wenigstens $(i+1)C_k - p_k$ beträgt.

Im folgenden wird gezeigt, daß in einem optimalen i-präemptiven Ablaufplan S die Last der letzten $m-(i+1)$ Prozessoren $P_{i+2}, \ldots, P_m$ wenigstens $(m-(i+1))s_k + p_k$ beträgt. Der Beweis erfolgt durch Widerspruch. Es wird gezeigt, daß aus einem i-präemptiven Ablaufplan S, in dem die Last der letzten $m-(i+1)$ Prozessoren $P_{i+2}, \ldots, P_m$ kleiner ist als $(m-(i+1))s_k + p_k$, ein Ablaufplan S' mit einer kleineren Planlänge erzeugt werden kann. S' kann aus S wie folgt erstellt werden (o.B.d.A. sei P_m ein Prozessor mit einem Bearbeitungsende $F_m < s_k$, vgl. Abb. 4.11):

- Verschiebe J_k von Prozessor P_{i+2} auf Prozessor P_m.

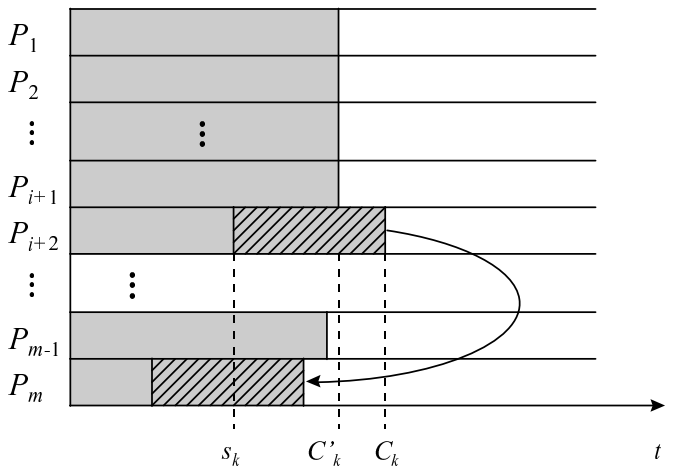

Abbildung 4.11: *Fall 2: Last der letzten $m-(i+1)$ Prozessoren ist wenigstens $m-(i+1)s_k+p_k$, wenn J_k auf P_{i+2} verplant wird.*

Resultat ist ein Ablaufplan S' mit den folgenden Eigenschaften:

1. Die Bearbeitungsenden der Prozessoren $P_1, \ldots, P_{i+1}$ bleiben unverändert.
2. Das Bearbeitungsende von Prozessor P_{i+2} vermindert sich um p_k.
3. Die Bearbeitungsenden der Prozessoren $P_{i+3}, \ldots, P_{m-1}$ bleiben unverändert.
4. Das Bearbeitungsende von Prozessor P_m ist kleiner als C_k, da wegen $F_m < s_k$ gilt $F_m + p_k < s_k + p_k$.

Daher kann man einen Ablaufplan S' mit einer kleineren Planlänge als die durch den Ablaufplan S erzeugte Planlänge C_k erstellen und es ist bewiesen, daß die Last der letzten $m-(i+1)$ Prozessoren wenigstens $(m-(i+1))s_k+p_k$. beträgt. Insgesamt ergibt sich

$$\begin{aligned}\sum p_j &\geq (i+1)C_k - p_k \quad + \quad (m-(i+1))s_k + p_k \\ &= (i+1)(s_k+p_k) - p_k \quad + \quad (m-(i+1))s_k + p_k \\ &= (i+1)p_k + ms_k\end{aligned}$$

oder äquivalent

$$s_k \leq \frac{\sum p_j - (i+1)p_k}{m}. \qquad \square$$

Lemma 4.7 *Sei S ein optimaler i-präemptiver Ablaufplan mit Planlänge $C_{max}^{ip^*}$ und $0 \leq i \leq m-1$ die maximale Anzahl zugelassener Präemptionen. Sei J_k ein Auftrag mit Bearbeitungsende $C_k = C_{max}^{ip^*}$ und mit Bearbeitungszeit $p_k < \sum p_j/m$ und*

$$p_k \leq C_{max}^{p^*} \left(\frac{m}{m+i+1} \right). \tag{4.1}$$

Dann gilt für alle Instanzen des Problems $P \mid i\text{-}pmtn \mid C_{max}$

$$C_{max}^{ip^*} \leq \left(2 - \frac{2}{\left(\frac{m}{i+1}\right)+1} \right) C_{max}^{p^*}$$

Beweis: Aus Lemma 4.5 folgt, daß Auftrag J_k nicht unterbrochen wird. Daher gilt für die Planlänge C_k des optimalen i-präemptiven Ablaufplans

$$C_{max}^{ip^*} = s_k + p_k.$$

Mit dem Ergebnis von Lemma 4.6 folgt

$$\begin{aligned} C_{max}^{ip^*} &\leq \frac{\sum p_j - (i+1)p_k}{m} + p_k \\ &= \frac{\sum p_j}{m} + \frac{m-(i+1)}{m} p_k. \end{aligned}$$

McNaughtons Schranke (3.4) und Annahme (4.1) führen schließlich zu

$$\begin{aligned} C_{max}^{ip^*} &\leq C_{max}^{p^*} + \frac{m-(i+1)}{m} \, \frac{m}{m+(i+1)} C_{max}^{p^*} \\ &= \left(\frac{m+(i+1)+m-(i+1)}{m+(i+1)} \right) C_{max}^{p^*} \\ &= \left(\frac{2m}{m+(i+1)} \right) C_{max}^{p^*} \\ &= \left(2 - \frac{2}{\left(\frac{m}{i+1}\right)+1} \right) C_{max}^{p^*}. \qquad \square \end{aligned}$$

Lemma 4.8 *Sei S ein optimaler i-präemptiver Ablaufplan mit Planlänge $C_{max}^{ip^*}$ und $0 \leq i \leq m-1$ die maximale Anzahl zugelassener Präemptionen. Sei J_k ein Auftrag mit Bearbeitungsende $C_k = C_{max}^{ip^*}$ und mit Bearbeitungszeit $p_k < \sum p_j/m$ und*

$$p_k \quad > \quad C_{max}^{p^*} \left(\frac{m}{m+i+1} \right). \tag{4.2}$$

Dann gilt für alle Instanzen des Problems $P \mid i\text{-pmtn} \mid C_{max}$

$$C_{max}^{ip^*} \leq \left(2 - \frac{2}{\left(\frac{m}{i+1}\right)+1} \right) C_{max}^{p^*}$$

Beweis: Der Beweis erfolgt durch Widerspruch. Es wird gezeigt, daß aus einem i-präemptiven Ablaufplan S, der eine Planlänge $C_{max}^{ip^*} > (2 - 2/(m/(i+1)+1))C_{max}^{p^*}$ hat, ein Ablaufplan S' mit einer kleineren Planlänge erzeugt werden kann.

Zunächst wird gezeigt, daß unter der Annahme $C_{max}^{ip^*} > (2 - 2/(m/(i+1)+1))C_{max}^{p^*}$ stets ein Prozessor mit einem Bearbeitungsende von höchstens $(m/(m+i+1))C_{max}^{p^*}$ existiert. Wenn es keinen Prozessor gäbe mit einem Bearbeitungsende von höchstens $(m/(m+i+1))C_{max}^{p^*}$, müßte gelten:

$$\begin{aligned}
\sum p_j \quad &> \quad (i+1)C_{max}^{ip^*} \quad + \quad (m-(i+1)) \left(\frac{m}{m+i+1} \right) C_{max}^{p^*} \\
&> \quad (i+1) \left(2 - \frac{2}{\left(\frac{m}{i+1}\right)+1} \right) C_{max}^{p^*} \quad + \quad (m-(i+1)) \left(\frac{m}{m+i+1} \right) C_{max}^{p^*} \\
&= \quad (i+1) \left(\frac{2m+2(i+1)-2(i+1)}{m+i+1} \, \frac{m}{m+i+1} \right) C_{max}^{p^*} \quad + \\
&\qquad (m-(i+1)) \left(\frac{m}{m+i+1} \right) C_{max}^{p^*} \\
&= \quad (2(i+1)+m-(i+1)) \left(\frac{m}{m+i+1} \right) C_{max}^{p^*} \\
&= \quad mC_{max}^{p^*}.
\end{aligned}$$

Das wäre ein Widerspruch zur Schranke von McNaughton (3.4). Daher gibt es im Falle $C_{max}^{ip^*} > (2-2/(m/(i+1)+1))C_{max}^{p^*}$ stets einen Prozessor mit einem Bearbeitungsende von höchstens $(m/(m+i+1))C_{max}^{p^*}$. O.B.d.A. sei dies Prozessor P_m mit Bearbeitungsende F_m, und für F_m gilt wegen $F_m \leq (m/(m+i+1))C_{max}^{p^*} < p_k$. Wir unterscheiden zwei Fälle:

Fall 1: In einem optimalen i-präemptiven Ablaufplan S wird J_k auf einem der ersten $i+1$ Prozessoren $P_1, \ldots, P_{i+1}$ (o.B.d.A. auf Prozessor P_{i+1}) bearbeitet.

Der Beweis erfolgt durch Widerspruch. Es wird gezeigt, daß aus einem i-präemptiven Ablaufplan S unter der Annahme $C_{max}^{ip^*} > (2-2/(m/(i+1)+1))C_{max}^{p^*}$ ein Ablaufplan S' mit einer kleineren Planlänge erzeugt werden kann. S' kann aus S wie folgt erstellt werden (vgl. Abb. 4.12):

- Vertausche J_k mit den Aufträgen, die auf Prozessor P_m verplant werden.

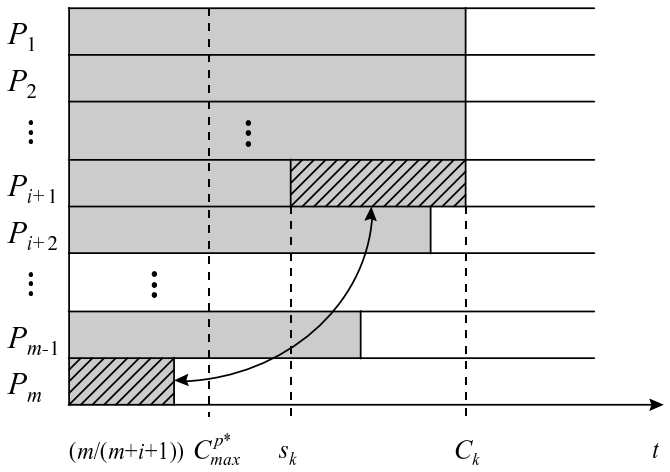

Abbildung 4.12: *Fall 1: Es gibt einen besseren Ablaufplan als S, wenn J_k auf P_{i+1} verplant wird.*

Resultat ist ein Ablaufplan S' mit den folgenden Eigenschaften:

1. Die Last der ersten $i+1$ Prozessoren $P_1, \ldots, P_{i+1}$ vermindert sich um $p_k - F_m > 0$, da $F_m < p_k$. Da es erlaubt ist, diese Last mit dem Algorithmus von

McNaughton zu verplanen und da es keinen Auftrag J_j gibt mit $p_j = C_{max}^{ip^*}$, ist das Bearbeitungsende der Prozessoren $P_1, \ldots, P_{i+1}$ von Ablaufplan S' kleiner als das Bearbeitungsende C_k von Ablaufplan S und es werden höchstens i Präemptionen erzeugt.

2. Die Bearbeitungsenden der Prozessoren $P_{i+2}, \ldots, P_m$ bleiben unverändert.
3. Das Bearbeitungsende von Prozessor P_m ist $p_k < C_k$.

Daher kann man einen Ablaufplan S' mit einer kleineren Planlänge als die durch den Ablaufplan S erzeugte Planlänge C_k erstellen und somit kann S nicht optimal sein.

Fall 2: In einem optimalen i-präemptiven Ablaufplan S wird J_k auf einem der letzten $m-(i+1)$ Prozessoren $P_{i+2}, \ldots, P_m$ (o.B.d.A. auf Prozessor P_{i+2}) bearbeitet.

Der Beweis erfolgt durch Widerspruch. Es wird gezeigt, daß aus einem i-präemptiven Ablaufplan S unter der Annahme $C_{max}^{ip^*} > (2 - 2/(m/(i+1)+1))C_{max}^{p^*}$ ein Ablaufplan S' mit einer kleineren Planlänge erzeugt werden kann. S' kann aus S wie folgt erstellt werden (vgl. Abb. 4.13):

- Vertausche J_k mit den Aufträgen, die auf Prozessor P_m verplant werden.

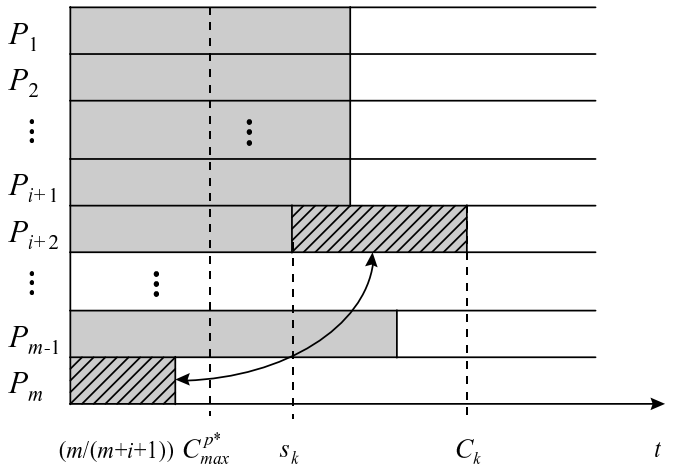

Abbildung 4.13: *Fall 2: Es gibt einen besseren Ablaufplan als S, wenn J_k auf P_{i+2} verplant wird.*

Resultat ist ein Ablaufplan S' mit den folgenden Eigenschaften:

1. Die Bearbeitungsenden der Prozessoren $P_1, \ldots, P_{i+1}$ bleiben unverändert.
2. Das Bearbeitungsende von Prozessor P_{i+2} ist $s_k + F_m < C_k$, da $F_m < p_k$.
3. Das Bearbeitungsende von Prozessor P_m ist $p_k < C_k$.

Daher kann man einen Ablaufplan S' mit einer kleineren Planlänge als die durch den Ablaufplan S erzeugte Planlänge C_k erstellen und somit kann S nicht optimal sein. □

Um zu zeigen, daß die Schranke $C_{max}^{ip^*} \leq (2-2/(m/(i+1)+1))C_{max}^{p^*}$ scharf ist, betrachten wir folgendies Beispiel:

Beispiel: n Aufträge sollen auf m Prozessoren bearbeitet werden mit $n = m+i+1$ und $p_j = c, j = 1, \ldots, n$. Der optimale i-präemptive Ablaufplan hat Länge $2c$, wohingegen der optimale präemptive Ablaufplan Länge $c(m+i+1)/m$ hat (vgl. Abb. 4.14). Die Aufträge J_k mit $C_k = C_{max}^{p^*}$ bzw. $C_k = C_{max}^{ip^*}$ sind jeweils schraffiert. In Abb. 4.14(a) werden die Aufträge mit McNaughtons Algorithmus verplant. In Abb. 4.14(b) und (c) sind optimale i-präemptive Pläne abgebildet.

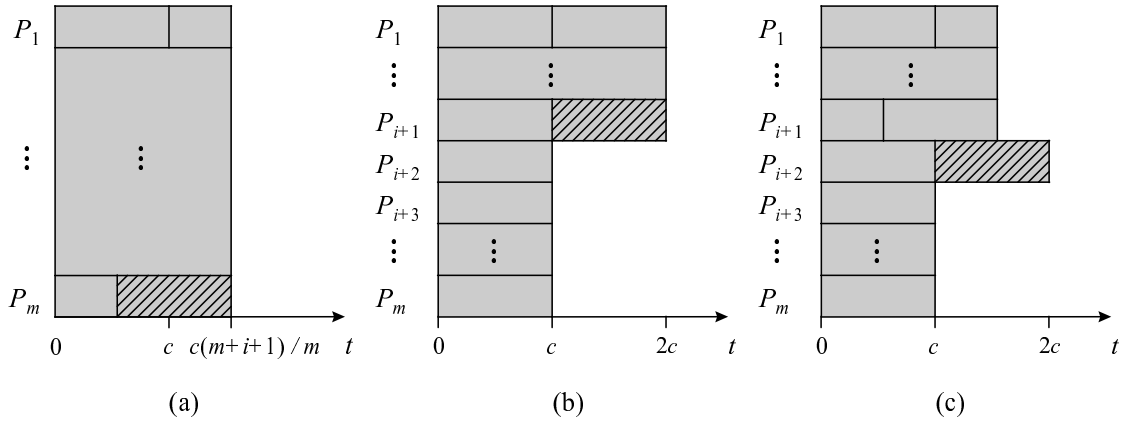

Abbildung 4.14: *Optimaler präemptiver und i-präemptive Ablaufpläne für $m+i+1$ Aufträge der Längen c*

Im Abb. 4.14(b) wird Auftrag J_k von einem der ersten i Prozessoren $P_2, \ldots, P_{i+1}$ (o.B.d.A. von P_{i+1}), auf denen Präemptionen erlaubt sind, bearbeitet. Im Abb. 4.14(c)

wird Auftrag J_k von einem der letzten $m - (i + 1)$ Prozessoren $P_{i+2}, \ldots, P_m$ (o.B.d.A. von P_{i+2}), auf denen keine Präemptionen erlaubt sind, bearbeitet. In beiden Fällen und muß ein Prozessor existieren, der zwei Aufträge ohne Unterbrechung bearbeitet. Insgesamt ergibt sich

$$\frac{C_{max}^{ip^*}}{C_{max}^{p^*}} = \frac{2c}{c(m+i+1)/m} = \frac{2m}{m+(i+1)} = \left(2 - \frac{2}{\left(\frac{m}{i+1}\right)+1}\right).$$

□

Kapitel 5

Zusammenfassung der Ergebnisse

Klassische Modelle der Theorie der Ablaufplanung sind gerade ihrer Abstraktheit wegen auf viele aus der Praxis motivierte Fragestellungen anwendbar, decken allerdings nicht jede Fragestellung ab. So wird in klassischen Modellen gewöhnlich davon ausgegangen, daß alle Prozessoren während des gesamten Planungszeitraums kontinuierlich für produktive Tätigkeiten zur Verfügung stehen. Diese Annahme ist allerdings bei real auftretenden Ablaufplanungsproblemen häufig nicht gerechtfertigt. Beispielsweise fallen Maschinen wegen möglicher Defekte oder wegen Wartungsarbeiten aus, und Arbeiter stehen wegen Krankheit oder Urlaubstagen nicht zur Verfügung. Zum anderen sind viele klassische Modelle der Ablaufplanung so formuliert, daß zu ihrer Lösung Ablaufpläne mit einer beliebigen Anzahl von Präemptionen zugelassen sind. Dabei wird vorausgesetzt, daß mit einer Präemption keine oder lediglich vernachlässigbar geringe zeitlichen Kosten auftreten. Nichtsdestotrotz entstehen in praktischen Anwendungen Kosten wie Lagerhaltungskosten oder Kosten für zusätzliche Produktionsfaktoren (z.B. menschliche Arbeitskräfte, Maschinen, Werkstoffe, Energie), die benötigt werden, wenn ein Auftrag von einem Prozessor genommen und auf einen anderen Prozessor wieder aufgesetzt wird.

So bestanden die beiden Ziele der vorliegenden Arbeit darin, die beiden klassischen Ablaufplanungsprobleme $F2 \mid pmtn \mid C_{max}$ und $P \mid pmtn \mid C_{max}$ hinsichtlich Beschränkungen der Verfügbarkeit der Prozessoren und hinsichtlich Beschränkungen der

Anzahl der zugelassenen Präemptionen zu untersuchen. Dabei wurden im wesentlichen folgende Ergebnisse erzielt (vgl. Abb. 5.1).

	$F2 \mid pmtn \mid C_{max}$	$P \mid pmtn \mid C_{max}$
Beschränkte Verfügbarkeit der Prozessoren	Kapitel 3.1 $F2, NC \mid pmtn \mid C_{max}$	Kapitel 3.2 $P, NC \mid pmtn \mid C_{max}$
	• Algorithmus, der die Optimalität von *Johnson-Permutationen* im Falle beschränkt verfügbarer Prozessoren testet	• *Worst-case*-Analyse der Laufzeiten und der Anzahl der Präemptionen der Algorithmen *Offline*, *Nearly-Online*, *Online* in einer vereinheitlichten Notation
Beschränkte Anzahl i der Präemptionen	Kapitel 4.1 $F2 \mid i\text{-}pmtn \mid C_{max}$	Kapitel 4.2 $P \mid i\text{-}pmtn \mid C_{max}$
	• Gütegarantie von i-präemptiven Plänen zu präemptiven Plänen: $C_{max}^{ip^*} = C_{max}^{p^*}$	• Gütegarantie von *LPT*-Plänen zu präemptiven Plänen: $C_{max}^{ip^*} \leq (2 - 2/(m+1))\, C_{max}^{p^*}$ • Gütegarantie von i-präemptiven Plänen zu präemptiven Plänen: $C_{max}^{LPT} \leq (2 - 2/(m/(i+1)+1))\, C_{max}^{p^*}$

Abbildung 5.1: *Übersicht über die in der Arbeit erzielten Ergebnisse*

5.1 Beschränkte Verfügbarkeit der Prozessoren

In Abschnitt 3.1 wurde ein Algorithmus entwickelt, der durch Überprüfung hinreichender Bedingungen feststellen kann, ob eine für eine Probleminstanz von $F2 \mid pmtn \mid C_{max}$ optimale Johnson-Permutation auch im Falle beschränkt verfügbarer Prozessoren optimal bleibt.

Dazu wurde gezeigt, wie *Stabilitätsradien* ρ_i mit einem Zeitaufwand von $O(n)$ und wie *Verzögerungsradien* δ_i mit einem Zeitaufwand von $O(w^2 + n \log n)$ berechnet werden können. Hierbei bezeichnen n die Anzahl der Aufträge, w die Anzahl der Nichtverfügbarkeitsintervalle und ρ_i das Minimum über alle maximal möglichen Verlänge-

rungen $r_{i,j}$ von Auftrag J_j auf Prozessor P_i, so daß eine Permutation der Aufträge die Johnson-Eigenschaft behält. δ_i bezeichnet das Maximum aller möglichen durch Nichtverfügbarkeitsintervalle verursachten Verlängerungen $d_{i,j}$ von Auftrag J_j auf Prozessor P_i. Es wurde gezeigt, daß eine Johnson-Permutation optimal bleibt, solange $d_{i,j} \leq r_{i,j}$ für alle $i \in \{1, 2\}$ und $j \in \{1, 2, \ldots, n\}$.

Anschließend wurden weitere hinreichende Bedingungen dafür angegeben, daß eine Johnson-Permutation auch im Falle nicht kontinuierlich verfügbarer Prozessoren optimal bleibt. Dazu wurde die Tatsache verwendet, daß es keine bezüglich der Planlänge besseren Pläne geben kann, wenn auf Prozessor P_1 die k (auf P_1) kleinsten Aufträge das Intervall $[0, t]$ komplett ausfüllen und wenn auf Prozessor P_2 die (auf P_2) kleinsten $n+1-k$ Aufträge das Intervall $[t, cmax]$ komplett ausfüllen. $cmax$ bezeichnet dabei das Bearbeitungsende eines mit Hilfe einer Johnson-Permutation erzeugten Ablaufplanes für eine Instanz des Problems $F2, NC^{offline} \mid pmtn \mid C_{max}$.

Basierend auf diesen Ergebnissen wurde der Algorithmus *Stability Test* erstellt und implementiert (vgl. `http://data.itm.uni-sb.de/~ob/html/f2.html`). Es wurde gezeigt, daß *Stability Test* in (in der Anzahl n der Aufträge und in der Anzahl w der Nichtverfügbarkeitsintervalle) polynomialer Zeit läuft, und es wurde eine experimentelle Performance-Analyse des Algorithmus durchgeführt. Dabei hat sich herausgestellt, daß die von Algorithmus *Stability Test* durchgeführte Stabilitätsanalyse sowohl geeignet ist für kleine Testsettings ($n \leq 100$ und $w \leq 10$) als auch insbesondere für große Testsettings ($1000 \leq n \leq 10000$ und $10 \leq w \leq 1000$). Ein Vorteil des entwickelten Algorithmus besteht darin, daß auch die großen Probleminstanzen schnell gelöst werden können. So betrug die maximale Laufzeit für eine Probleminstanz mit 10000 Aufträgen und 1000 Nichtverfügbarkeitsintervallen (mit Bearbeitungszeiten und Intervallängen aus dem Intervall $[1, 1000]$) 2.899 Sekunden. Im Vergleich dazu sind mit dem Branch-And-Bound Verfahren aus Kubiak *et al.* [KBFBS02] lediglich Probleminstanzen mit maximal 100 Aufträgen und 10 Nichtverfügbarkeitsintervallen (mit Bearbeitungszeiten und Intervallängen aus dem Intervall $[1, 1000]$) gelöst worden (mit einer Zeitschranke von 1000 Sekunden). Die Begrenzung der Anzahl der betrachteten Nichtverfügbarkeits-

intervalle rührt daher, daß die Laufzeit des Verfahrens exponentiell mit der Anzahl der Nichtverfügbarkeitsintervalle wächst.

Für die meisten der zufällig generierten Probleminstanzen hat der Algorithmus festgestellt, daß eine für eine Probleminstanz des Problems $F2 \mid pmtn \mid C_{max}$ optimale Johnson-Permutation auch die (bis auf die Nichtverfügbarkeitsintervalle) gleiche Probleminstanz des Problems $F2, NC \mid pmtn \mid C_{max}$ optimal löst. Die einzige Ausnahme waren Klassen von Probleminstanzen, in denen die Bearbeitungszeiten auf dem ersten Prozessor doppelt so groß waren wie die auf dem zweiten Prozessor, die Anzahl der Aufträge kleiner als die Anzahl der Nichtverfügbarkeitsintervalle waren und der zweite Prozessor kontinuierlich verfügbar war. Für eine solche Klasse von Probleminstanzen müssen andere hinreichende Optimalitätsbedingungen entwickelt werden.

Die in Abschnitt 3.1 entwickelte Stabilitätsanalyse kann ebenso für andere Ablaufplanungsprobleme mit beschränkter Verfügbarkeit der Prozessoren benutzt werden, wenn das Problem mit kontinuierlich verfügbaren Prozessoren durch Prioritätsregeln wie *SPT, LPT*, usw. gelöst werden kann. Ebenso können die erzielten Resultate für ein *online setting* beschränkter Prozessorverfügbarkeiten benutzt werden, in denen es keine Informationen über die genaue Lage der Nichtverfügbarkeitsintervalle gibt, aber die Werte $d_{i,j}$, bzw. $\delta_i, i \in \{1,2\}, j \in \{1,\ldots,n\}$ a priori bekannt sind.

In Abschnitt 3.2 wurden aus der Literatur bekannte Algorithmen zur Lösung von $P, NC \mid pmtn \mid C_{max}$ mit beschränkt verfügbaren parallelen Prozessoren untersucht. In Abhängigkeit von dem Zeitpunkt, zu dem ein Algorithmus Informationen über die exakten Verfügbarkeiten der Prozessoren erhält, werden die drei Fälle *offline-setting*, *nearly-online-setting* und *online-setting* unterschieden. Drei Algorithmen, die die jeweiligen Planungssituationen berücksichtigen, wurden vergleichend untersucht. Neben einer Vereinheitlichung der Darstellung, einer detaillierten und vergleichenden Ausarbeitung der Algorithmen und einer Analyse der im *worst-case* erzeugten Anzahl der Präemptionen des *Nearly-Online*-Algorithmus, wurde die Vermutung untersucht, ob es stets bezüglich der Planlänge optimale *Online*-Algorithmen gibt, falls die Anzahl der

Prozessoren kleiner als die Anzahl der Aufträge ist.

Es wurde gezeigt, daß im *worst-case* in jeder der drei Planungssituationen wenigstens $M := \sum_{k=1}^{S-1} m_k = O(Sm)$ Präemptionen auf Grund von wechselnden Prozessorverfügbarkeiten erzeugt werden. In den beiden Tabellen 5.1 und 5.2 werden die erzielten Ergebnisse zusammengefaßt. n bezeichnet dabei die Anzahl der Aufträge, m die Anzahl der Prozessoren und S die Anzahl der Systemintervalle, d.h. die Anzahl der Zeitpunkte, zu denen sich die Menge der verfügbaren Prozessoren (das Prozessorsystem) ändert. Q ist die Anzahl der Prozessorintervalle, d.h der Intervalle, in denen ein Prozessor verfügbar ist. p_{avg} ist die durchschnittliche Bearbeitungszeit eines Auftrags und $\delta = \epsilon n^2$ ist die Schrittweite des *Online*(ϵ) Algorithmus, der eine Planlänge von maximal $C^*_{max} + \epsilon$ erzeugt.

Muster	Planungssituation	Algorithmus	Laufzeit
konstant	beliebig	*McNaughton*	$O(n)$
beliebig	Offline	*Offline*	$O(Sm + n + m \log m)$
	Nearly-Online	*Lookahead*	$O(Sn + n \log n)$
	Online	*Online*	$O(Sm \log n + (p_{avg}/\delta) n \log n)$

Tabelle 5.1: $P, NC \mid pmtn \mid C_{max}$ *(Laufzeit)*

Muster	Planungssituation	Algorithmus	Präemptionen
konstant	beliebig	*McNaughton*	$m - 1 =$ $O(m)$
beliebig	Offline	*Offline*	$M + Q - 1 =$ $O(Sm)$
	Nearly-Online	*Lookahead*	$M + (S-1)(n-1) + (m-1) =$ $O(Sm + Sn)$
	Online	*Online*	$M + (p_{avg}/\delta)n =$ $O(Sm + (p_{avg}/\delta)n)$

Tabelle 5.2: $P, NC \mid pmtn \mid C_{max}$ *(Präemptionen)*

Bei der Analyse der Laufzeiten sieht man, daß der *Offline*-Algorithmus auch im Falle $S = 1$ auf Grund der Sortierung der Aufträge Zeit $O(n + m \log m)$ benötigt, also

langsamer ist als der Algorithmus von McNaughton. So kann das Ziel weiterer Untersuchungen darin bestehen, einen *Offline*-Algorithmus mit Laufzeit $O(Sm + n)$ zu entwickeln.

Analysiert man die Anzahl der im *worst-case* verursachten Präemptionen, bestätigt sich die Vermutung, daß für weniger Informationen bezüglich der Verfügbarkeiten der Prozessoren mit mehr Präemptionen bezahlt werden muß. Umgekehrt sieht man aber auch, daß unter der Voraussetzung, daß $(p_{avg}/\delta) < S$ gilt, der *Online*-Algorithmus im *worst-case* weniger Präemptionen erzeugt als der *Nearly-Online*-Algorithmus. Der Term (p_{avg}/δ) ist dann sehr klein, wenn die durchschnittliche Bearbeitungszeit sehr klein ist oder wenn δ sehr groß ist, man also mit einem großen Fehler $\epsilon = \delta n^2$ einverstanden ist.

Die Vermutung, daß es stets bezüglich der Planlänge optimale *Online*-Algorithmen gibt, falls die Anzahl der Prozessoren kleiner als die Anzahl der Aufträge ist, wurde widerlegt, indem gezeigt wurde, daß es Probleminstanzen von $P, NC \mid pmtn \mid C_{max}$ mit $m \geq 3$ Prozessoren und $n \geq 2$ Aufträgen gibt, für die keine optimalen *Online*-Algorithmen erstellt werden können. Offen bleibt die Frage, ob es für den Fall $m = 2$ und $n \geq 2$ stets optimale *Online*-Algorithmen gibt.

5.2 Beschränkte Anzahl der Präemptionen

Das Problem $F2 \mid i\text{-}pmtn \mid C_{max}$ kann mit Hilfe des Algorithmus von Johnson ohne Präemptionen optimal gelöst werden. Abschnitt 4.1 enthält das entsprechende Ergebnis.

In Abschnitt 4.2 wurde gezeigt, daß das Verhältnis optimaler i-präemptiver Planlängen $C_{max}^{ip^*}$ zu optimalen präemptiven Planlängen $C_{max}^{p^*}$ durch $C_{max}^{ip^*} \leq (2 - 2/(m/(i+1) + 1))C_{max}^{p^*}$ nach oben beschränkt ist. Im grau hinterlegten Bereich in Abb. 5.2 befinden sich alle möglichen Kombinationen von (ganzzahligen) Präemptionen und (rationalen) Planlängen. Darüberhinaus wurde gezeigt, daß mit der *LPT-Regel* erzeugte Pläne die

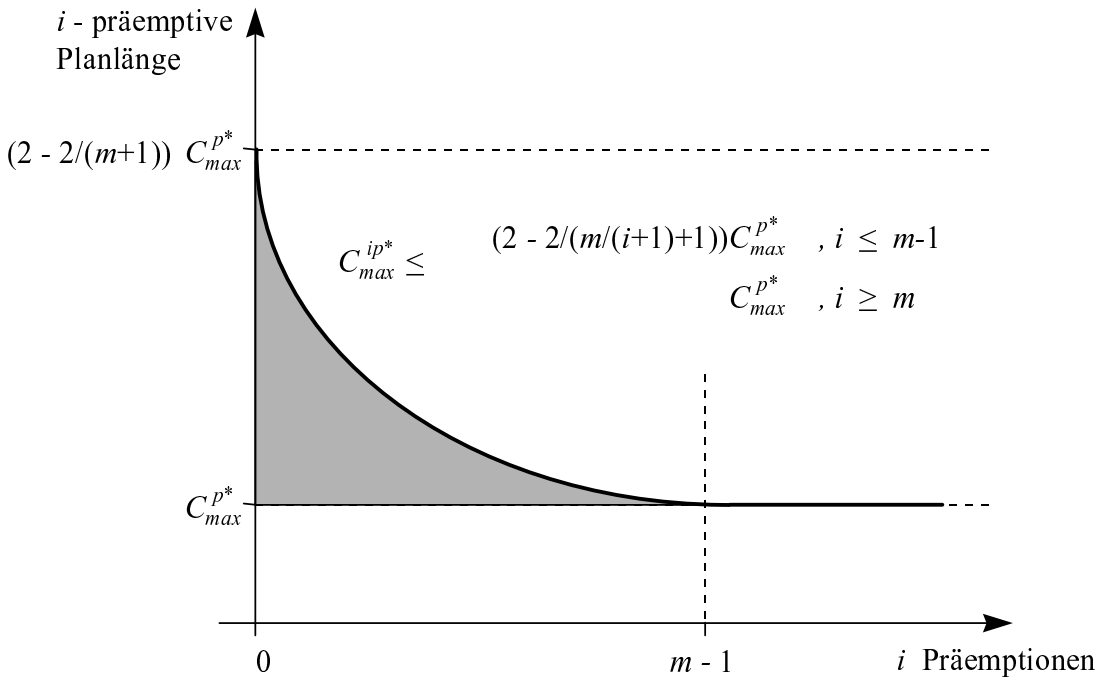

Abbildung 5.2: *i-präemptive vs. präemptive Ablaufpläne*

gleiche Gütegarantie (mit $i = 0$) haben. Mit Hilfe eines *worst-case* Beispiels wurde gezeigt, daß die Schranken scharf sind.

Die Resultate von Abschnitt 4.2 können als Basis für weitergehende Untersuchungen bezüglich des Problems der *Minimierung von Präemptionen* dienen. Dabei sollen zwei Ziele verfolgt werden:

- Minimierung der Planlänge (Zeitziel) und
- Minimierung der Anzahl der Präemptionen (Kostenziel).

Da hier zwei Ziele (Zeit- und Kostenziel) optimiert werden sollen, spricht man von einem *Bicriterion Scheduling*-Problem ([VWB82]). Solche Probleme lassen sich grundsätzlich lösen, indem man für ein Kriterium (*hard criterion*) einen Schwellenwert einführt, der nicht überschritten werden darf , und versucht, das andere Kriterium (*soft criterion*) zu optimieren. Entsprechend kann man zwei Fälle unterscheiden:

Fall 1: Als zusätzliche Eingabe erhalten die Algorithmen eine maximal erlaubte Anzahl

i von Präemptionen, und das Ziel besteht darin, Pläne zu erzeugen, die den beiden folgenden Kriterien genügen:

1. Die Anzahl der Präemptionen beträgt maximal i.
2. Die Planlänge wird minimiert.

Wenn $i \geq m - 1$, kann ein bezüglich der Planlänge optimaler Ablaufplan mit dem Algorithmus von McNaughton erstellt werden. Jedoch wird das Problem für jeden Wert von i zwischen 0 und $m - 2$ *NP*-schwer.

Fall 2: Als zusätzliche Eingabe erhalten die Algorithmen eine maximal erlaubte Planlänge $C_{max}^{allowed}$. Das Ziel besteht darin, Pläne zu erzeugen, die den beiden folgenden Kriterien genügen:

1. Die Planlänge beträgt maximal $C_{max}^{allowed}$.
2. Die Anzahl der Präemptionen wird minimiert.

Wenn $C_{max}^{allowed} < C_{max}^{p^*}$, kann kein zulässiger Ablaufplan existieren. Eine Planlänge $C_{max}^{allowed} \geq 2 - (2/(m+1))C_{max}^{p^*}$ kann mit Hilfe eines mit der *LPT-Regel* erstellten Ablaufplanes ohne Präemptionen erreicht werden (vgl. Satz 4.2). Jedoch wird das Problem für jeden Wert zwischen diesen beiden Werten *NP*-schwer.

Beide Probleme sind verwandt mit den *NP*-schweren Problemen PARTITION und SUBSET SUM, sind in ihrer Komplexität aber noch schwieriger. Für m=2 kann ein Algorithmus, der auf *Dynamischer Programmierung* beruht und das PARTITION-Problem löst, entwickelt werden (vgl. `http://data.itm.uni-sb.de/~ob/html/min.html`). Ein anderer Ansatz besteht darin, Heuristiken zu entwickeln, die auf der *LPT-Regel* basieren.

Literaturverzeichnis

[AC81] Achugbue, J.O., Chin, F.Y., Bounds on schedules for independent tasks with similar execution times, *Journal of the Association for Computing Machinery* 28, 81–99, 1981

[AS01] Albers, S., Schmidt, G., Scheduling with unexpected machine breakdowns, *Discrete Applied Mathematics* 110, 85–99, 2001

[Bak74] Baker, K.R., *Introduction to sequencing and scheduling*, John Wiley, New York, 1974

[BC91] Blocher, J.D., Chand, S., Scheduling on parallel processors: a posterior bound on LPT sequencing and a two-step algorithm, *Naval Research Logistics Quarterly* 38, 273–287, 1991

[BFHS88] Błażewicz, J., Finke, G., Haupt, R., Schmidt, G., New trends in machine scheduling, *European Journal of Operational Research* 37, 303–317, 1988

[BEPSW01] Błażewicz, J., Ecker, K.H., Pesch, E., Schmidt, G., Węglarz, J., *Scheduling computer and manufacturing processes, 2. Auflage*, Springer, Berlin, 2001

[BLSS02] Braun, O., Lai, T.-C., Schmidt, G., Sotskov, Y.N., Stability of Johnson's schedule with respect to limited machine availability, *International Journal of Production Research*, angenommen, 2002

[Bre00] Breit, J., Heuristische Ablaufplanungsverfahren für Flowshops und Openshops mit beschränkt verfügbaren Prozessoren, *Dissertation*, Universität des Saarlandes, Saarbrücken, 2000

[Bru01] Brucker, P., *Scheduling algorithms, 3. Auflage*, Springer, Berlin, 2001

[BSS00] Braun, O., Schmidt, G., Sotskov, Y.N., Two-machine n-job flow-shop problem with limited machine availability, *Proceedings of the 14th workshop on discrete optimization*, TU Freiberg, Germany, 70–73, 2000

[CFL84] Coffman, Jr., E.G., Frederickson, M.L., Lueker, G.S., A note on expected makespan for largest-fit sequences of independent tasks on two processors, *Mathematics of Operations Research* 9, 260–266, 1984

[CG93] Coffman, Jr., E.G., Garey, M.R., Proof of the 4/3 conjecture for preemptive vs. nonpreemptive two-processor scheduling, *Journal of the Association for Computing Machinery* 20, 991–1018, 1993

[CGJ78] Coffman, Jr., E.G., Garey, M.R., Johnson, D.S., *An application of bin-packing to multiprocessor scheduling*, *SIAM Journal on Computing* 7, 1–17, 1978

[Che93] Chen, B., A note on LPT scheduling, *Operations Research Letters* 14, 139–142, 1993

[CMM67] Conway, R.W., Maxwell, W.L., Miller, L.W., *Theory of scheduling*, Addison-Wesley, Reading, 1967

[Cof76] Coffman, Jr., E.G., *Scheduling in computer and job shop systems*, John Wiley, New York, 1976

[CS76] Coffman, Jr., E.G., Sethi, R., A generalized bound on LPT sequencing, *RAIRO-Informatique* 10, 17–25, 1976

[CS90] Cheng, T.C.E., Sin, C.C.S., A state-of-the-art review of parallel-machine scheduling research, *European Journal of Operational Research* 47, 271–292, 1990

[CW99] Cheng, T.C.E., Wang, G., Two-machine flowshop scheduling with consecutive availability constraints, *Information Processing Letters* 71, 49–54, 1999

[CW00] Cheng, T.C.E., Wang, G., An improved heuristic for two-machine flowshop scheduling with an availability constraint, *Operations Research Letters* 26, 223–229, 2000

[Dob84] Dobson, G., Scheduling independent tasks on uniform processors, *SIAM Journal on Computing* 13, 705–716, 1984

[DK96] Dinkelbach, W., Kleine, A., *Elemente einer betriebswirtschaftlichen Entscheidungslehre*, Springer, Berlin, 1996

[EE80] Elmaghraby, S.E., Elimam, A.A., Knapsack-based approaches to the makespan problem on multiple processors, *AIIE Transactions* 12, 87–96, 1980

[Fre82] French, S., *Sequencing and scheduling: an introduction to the mathematics of the job shop*, John Wiley, New York, 1982

[Fri84] Friesen, D.K., Tighter bounds for the MULTIFIT processor scheduling algorithm, *SIAM Journal on Computing* 13, 35–59, 1984

[FRK87] Frenk, J.B.G., Rinnoy Kan, A.H.G., The asymptotic optimality of the LPT rule, *Mathematics of Operations Research* 12, 241–254, 1987

[GGJ78] Garey, M.R., Graham, R.L., Johnson, D.S., Performance guarantees for scheduling algorithms, *Operations Research* 26, 3–21, 1978

[GGR92] Glaser, H., Geiger W., Rohde V., *PPS - Produktionsplanung und -steuerung; Grundlagen - Konzepte - Anwendungen, 2. Auflage*, Gabler, Wiesbaden, 1992

[GJ79] Garey, M.R., Johnson, D.S., *Computers and intractability: a guide to the theory of NP-completeness*, W.H. Freeman, San Francisco, 1979

[GJ80] Gonzales, T., Johnson, D.B., A new algorithm for preemptive scheduling of trees, *Journal of the Association for Computing Machinery* 27, 287–312, 1980

[GLLRK79] Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnoy Kan, A.H.G., Optimization and approximation in deterministic sequencing and scheduling: a survey, *Annals of Discrete Mathematics* 5, 287–326, 1979

[Gra66] Graham, R.L., Bounds for certain multiprocessing anomalies, *Bell System Tech. Journal* 45, 1563–1581, 1966

[Gra69] Graham, R.L., Bounds on multiprocessing timing anomalies, *SIAM Journal on Applied Mathematics* 17, 416-429, 1969

[Gre72] Greenberg, I., Applications of the loading algorithms to balance workloads, *AIIE Transactions* 4, 337–339, 1972

[GS78] Gonzales, T., Sahni, S., Flowshop and jobshop schedules: complexity and approximation, *Operations Research* 26, 36–52, 1978

[GS01] Goldberg, R.R., Shapiro, J., Extending Graham's result on scheduling to other heuristics, *Operations Research Letters* 29, 149–153, 2001

[He98] He, Y., Parametric LPT-bound on parallel machine scheduling with non-simultaneous machine available time, *Acia-Pacific Journal of Operational Research* 15, 29–36, 1998

[HL92] Hong, K.S., Leung, J.Y.-T., Some results on Liu's conjecture, *SIAM Journal on Discrete Mathematics* 5, 500-523, 1992

[Hoi93] Hoitsch, H.-J., *Produktionswirtschaft: Grundlagen einer industriellen Betriebswirtschaftslehre, 2. Auflage*, Vahlen, München, 1993

[Hor74] Horn, W.A., Some simple scheduling algorithms, *Naval Research Logistics Quarterly* 21, 177–185, 1974

[HS87] Hochbaum, D.S., Shmoys, D.B., Using dual approximation algorithms for scheduling nonidentical processors, *Journal of the Association for Computing Machinery* 23, 317–327, 1987

[IK77] Ibarra, O.H., Kim, C.E., Heuristic algorithms for scheduling independent tasks on nonidentical processors, *Journal of the Association for Computing Machinery* 24, 280–289, 1977

[Joh54] Johnson, S.M., Optimal two- and three-stage production schedules with setup times included, *Naval Research Logistics Quarterly* 1, 69–81, 1954

[Kar72] Karp, R.M., Reducibility among combinatorial problems, in Miller, R.R., Thatcher, J.W. (Eds.), *Complexity of Computer Computations*, Plenum Press, New York, 85–104, 1972

[KBFBS01] Kubiak, W., Błażewicz, J., Formanowicz, P., Breit, J., Schmidt, G., Heuristic algorithms for the two-machine flow shop with limited machine availability, *Omega* 29, 599-608, 2001

[KBFBS02] Kubiak, W., Błażewicz, J., Formanowicz, P., Breit, J., Schmidt, G., Two-machine flow shops with limited machine availability, *European Journal of Operational Research* 136, 528–540, 2002

[KBFS97] Kubiak, W., Błażewicz, J., Formanowicz, P., Schmidt, G., A branch and bound algorithm for two machine flow shops with limited machine availability, *Research Report RA-001/97*, Institute of Computing Science, Poznan University of Technology, Poznan, 1997

[KE88] Kao, T.Y., Elsayed, E.A., Performance of the LPT algorithm in multiprocessor scheduling, *IE working paper*, Rutgers University, 88–109, 1988

[Ked70] Kedia, S.K., A job scheduling problem with parallel processors, *Unpublished Report*, Department of Industrial Engineering, University of Michigan, 1970

[Kel98] Kellerer, H., Algorithms for multiprocessor scheduling with machine release times, *IIE Transactions* 30, 991–999, 1998

[KS93] Kistner, K.P., Steven, M., *Produktionsplanung, 3. Auflage*, Physica, Heidelberg, 1993

[KSW95] Kravchenko, S.A., Sotskov, Y.N., Werner, F., Optimal schedules with infinitely large stability radius, *Optimization* 33, 271–280, 1995.

[Lee91] Lee, C.Y., Parallel machines scheduling with nonsimultaneous machine available time, *Discrete Applied Mathematics* 30, 53–61, 1991

[Lee96] Lee, C.Y., Machine scheduling with an availability constraint, *Journal of Global Optimization* 9, 363–384, 1996

[Lee97] Lee, C.Y., Minimizing the makespan in the two-machine flowshop scheduling problem with an availability constraint, *Operations Research Letters* 20, 129–139, 1997

[LHT00] Lee, C.Y., He, Y., Tang, G., A note an parallel machine scheduling with non-simultaneous machine available time, *Discrete Applied Mathematics* 100, 133–135, 2000

[LHYL97] Lin, G., He, Y., Yao, Y., Lu, H., Exact bounds of the modified LPT algorithm applying to parallel machines scheduling with nonsimultaneous machine available times, *Applied Mathematics Journal of the Chinese University* B12, 109–116, 1997

[Liu72] Liu, C.L., Optimal scheduling on multi-processor computing systems, in: *Proceedings of the 13th Ann. Sympos. on Switching and Automata Theory*, IEEE Computer Society, New York, 155–160, 1972

[LM88] Lee, C.Y., Massey, J.D., Multiprocessor scheduling: Combining LPT and MULTIFIT, *Discrete Applied Mathematics* 20, 233-242, 1988

[LS77] Lam, S., Sethi, R., Worst case analysis of two scheduling algorithms, *SIAM Journal on Computing* 6, 518–536, 1977

[LS95] Liu, Z., Sanlaville, E., Preemptive scheduling with variable profile, precedence constraints and due dates, *Discrete Applied Mathematics* 58, 253–280, 1995

[LS99] Lai, T.-C., Sotskov, Y.N., Sequencing with uncertain numerical data for makespan minimization, *Journal of the Operational Research Society* 50, 230–243, 1999

[MC69] Muntz, R., Coffman, Jr., E.G., Optimal preemptive scheduling of two-processor systems, *IEEE Transactions on Computers* C-18, 1014-1029, 1969

[McN59] McNaughton, R., Scheduling with deadlines and loss functions, *Management Science* 6, 1–12, 1959

[Neu96] Neumann, K., *Produktions- und Operationsmanagement*, Springer, Berlin, 1996

[Pap93] Papadimitriou, C.H., *Computational complexity*, Addison-Wesley, Reading, 1993

[Pin01] Pinedo, M., *Scheduling: theory, algorithms, and systems. 2nd edition*, Prentice-Hall, Englewood Cliffs, 2001

[PS98] Papadimitriou, C.H., Steiglitz, K., *Combinatorial optimization: algorithms and complexity*, Dover Publications, Mineola, 1998

[San95] Sanlaville, E., Nearly on line scheduling of preemptive independent tasks, *Discrete Applied Mathematics* 57, 229–241, 1995

[Sch84] Schmidt, G., Scheduling on semi-identical processors, *Zeitschrift für Operations Research* 28, 153–162, 1984

[Sch99] Schmidt, G., *Informationsmanagement: Modelle, Methoden, Techniken, 2. Auflage*, Springer, Berlin, 1999

[Sch00] Schmidt, G., Scheduling with limited machine availability, *European Journal of Operational Research* 121, 1–15, 2000

[Sch02] Schmidt, G., *Prozeßmanagement: Modelle und Methoden, 2. Auflage*, Springer, Berlin, 2002

[Set76] Sethi, R., Algorithms for minimal-length schedules, in: Coffman, Jr., E.G. (Ed.), *Scheduling in Computer and Job Shop Systems*, John Wiley, New York, 1976

[Sot91] Sotskov, Y.N., Stability of an optimal schedule, *European Journal of Operational Research* 55, 91–102, 1991

[Sot01] Sotskova, N.Y., Optimal scheduling with uncertainty in the numerical data on the basis of a stability analysis, *Dissertation*, Otto-von-Guericke-Universität, Magdeburg, 2001

[SS98] Sanlaville, E., Schmidt, G., Machine scheduling with availability constraints, *Acta Informatica* 35, 795–811, 1998

[SSW97] Sotskov, Y.N., Sotskova, N.Y., Werner, F., Stability of an optimal schedule in a job shop, *Omega* 25, 397–414, 1997

[TSS94] Tanaev, V.S., Sotskov, Y.N., Strusevich, V.A., *Scheduling Theory. Multi-Stage Systems*, Kluwer, Dordrecht, 1994

[Ull75] Ullman, J.D., NP-complete scheduling problems, *Journal of Computer and System Sciences* 10, 384–393, 1975

[Ull76] Ullman, J.D., *Complexity of sequencing problems, in:* Coffman, Jr., E.G. (Ed.), *Scheduling in Computer and Job Shop Systems*, John Wiley, New York, 1976

[VWB82] Van Wassenhove, L.N., Baker, K.R., A bicriterion approach to time/cost trade-offs in sequencing, *European Journal of Operational Research* 11, 48–54, 1982

[Yue90] Yue, M., On the exact upper bound for the MULTIFIT processor scheduling algorithm, *Annals of Operations Research* 24, 233–259, 1990

[Zim01] Zimmermann, J., *Ablauforientiertes Projektmanagement: Modelle, Verfahren und Anwendungen*, Gabler, Wiesbaden, 2001