Bewegungs-minimierung in der Förderband-Flow-Shop-Verarbeitung

Inaugural-Dissertation

zur
Erlangung des Doktorgrades der
Mathematisch-Naturwissenschaftlichen Fakultät
der Heinrich-Heine-Universität Düsseldorf

vorgelegt von

Wolfgang Espelage

aus Vechta

Düsseldorf
2002

Gedruckt mit der Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät
der Heinrich-Heine-Universität Düsseldorf

Referent: Prof. Dr. Egon Wanke
Korreferent: Prof. Dr. Jörg Rothe

Tag der mündlichen Prüfung: 24.01.2002

Vorwort

Die in dieser Arbeit dargestellen Resultate entstanden während meiner Tätigkeit als wissenschaftlicher Mitarbeiter an der Heinrich-Heine-Universität Düsseldorf in einem von der Deutschen Forschungsgemeinschaft geförderten Forschungsprojekt im Rahmen des Schwerpunktprogramms "Effiziente Algorithmen für diskrete Probleme und ihre Anwendungen"[1].

Ich möchte an dieser Stelle Prof. Dr. Egon Wanke meinen besonderen Dank aussprechen. Er stand mir bei zahlreichen Diskussionen stets mit wertvollen Anregungen und Hinweisen zur Seite und begleitete die Arbeit mit großem Interesse. Seine Betreuung hat wesentlich zum Gelingen dieser Arbeit beigetragen.

Mein Dank gilt auch meinen Kollegen Jochen Rethmann, Frank Gurski, Andreas Beck und Prof. Dr. Volker Aurich aus der Abteilung für Informatik der Heinrich-Heine-Universität Düsseldorf für die gute Zusammenarbeit, die freundschaftliche Atmosphäre und das gute Arbeitsklima.

Ferner danke ich der Bertelsmann Distribution GmbH in Gütersloh für die Möglichkeit, uns vor Ort ein genaues Bild von dem Problem der Bewegungsminimierung in der Förderband-Flow-Shop-Verarbeitung machen zu können. Namentlich möchte ich hier Dr. Christian Ewering für seine Unterstützung danken.

Düsseldorf, März 2002 Wolfgang Espelage

[1] DFG-Projekt Wa 674/8-1/2: Entwicklung effizienter Algorithmen für die Minimierung von Arbeiterbewegungen in Flow-Shop-Fertigungssystemen

Inhaltsverzeichnis

Kapitel 1

Einleitung

In dieser Arbeit entwickeln und analysieren wir Algorithmen zur Minimierung der Bewegungen von Arbeitern in geradlinig angeordneten Förderbandsystemen, wie sie insbesondere in so genannten Kommissionieranlagen in der Versandindustrie auftreten. In solchen Systemen werden Aufträge mit einem Förderband von Arbeitsposition zu Arbeitsposition transportiert und dort der Reihe nach in Flow-Shop-Manier bearbeitet. Für die Bearbeitung der Aufträge ist die Anwesenheit eines Arbeiters erforderlich. Da in der Regel weniger Arbeiter als Arbeitspositionen vorhanden sind, müssen die Arbeiter zur Bearbeitung der Aufträge ihre Positionen wechseln. Das Ziel unserer Untersuchungen ist es, Algorithmen bzw. Strategien zur Minimierung dieser Bewegungen zu entwickeln und zu analysieren. Die Ergebnisse dieser Arbeit sind im Bereich der theoretischen Informatik anzusiedeln, das zugrunde liegende Problem ist jedoch stark durch die Praxis motiviert. Um den Praxisbezug des von uns in dieser Arbeit behandelten Grundproblems nachvollziehen zu können, soll im folgenden Abschnitt kurz die grundlegende Funktionsweise einer typischen Kommissionieranlage skizziert werden. Anschließend erläutern wir das theoretische Modell und gehen auf die Komplexität des Problems und die betrachteten Algorithmen ein.

Kommissionieranlagen Die Filialen großer Warenhäuser werden oft von einem Zentrallager aus mit Waren versorgt. Die Waren werden dabei normalerweise in einer Kommissionieranlage in stabile, gleich große Behälter gepackt, welche anschließend nach Zusammengehörigkeit auf Paletten gestapelt und dann zu den einzelnen Filialen transportiert werden. Um die Behälter mit den verschiedenen Waren zu füllen, verfügt die Kommissionieranlage über ein Förderbandsystem, das die einzelnen Kommissionierbereiche miteinander verbindet. Das Förderbandsystem besteht aus mindestens einem Hauptförderband und mehreren Nebenförderbändern, die parallel zum Hauptförderband verlaufen. Das Hauptförderband ist oft zyklisch angelegt, damit Aufträge, die nicht sofort bearbeitet werden können, in einer späteren Runde erledigt werden. Jedes Nebenförderband verbindet einige Kommissionierbereiche miteinander. Muss ein Behälter Waren aus einem solchen Kommissionierbereich aufnehmen, so wird er automatisch vom Hauptförderband auf das entsprechende Nebenförderband ausgeschleust. Dort kann der Behälter an dem Kommissionierbereich gestoppt werden, ohne den Transport der übrigen Behälter auf dem Hauptförderband zu

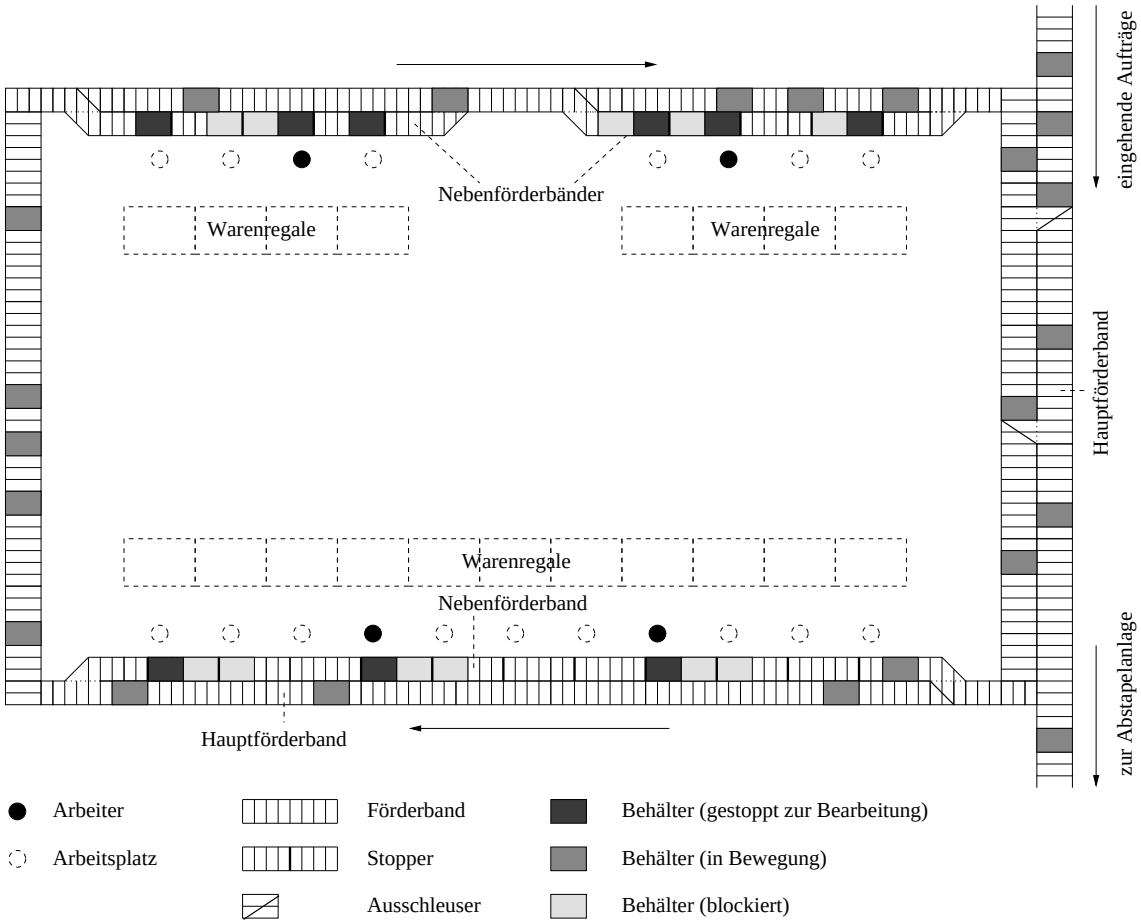


Abbildung 1.1: Schematische Darstellung einer Kommissionieranlage.

behindern. Nachdem der Behälter an den Kommissionierbereichen des Nebenförderbandes mit Waren gefüllt wurde, wird er am Ende des Nebenförderbandes wieder auf das Hauptförderband ausgeschleust. In einigen Anlagen ist es auch möglich, den Behälter an jedem Arbeitsplatz des Nebenförderbandes sofort wieder auf das Hauptförderband auszuschleusen, falls der Behälter keinen weiteren Kommissionierbereich dieses Nebenförderbandes ansteuern muss. In Abbildung 1.1 ist eine solche Anlage skizziert.

Muss ein Behälter an einem Kommissionierbereich mit Waren gefüllt werden, so wird er automatisch auf dem Nebenförderband an der entsprechenden Stelle gestoppt. Den Arbeitern wird über ein Display angezeigt, welche Artikel des Kommissionierbereichs in welcher Menge in den Behälter gepackt werden müssen. Nachdem die Waren in den Behälter gelegt wurden, wird er auf dem Nebenförderband weitertransportiert, um gegebenenfalls an den anderen Kommissionierbereichen weitere Artikel aufzunehmen. Am Ende des Nebenförderbandes wird der Behälter wieder auf das Hauptförderband ausgeschleust. Der Behälter steuert dann weitere Nebenförderbänder an oder verlässt die Kommissionieran-

lage in Richtung Abstapelanlage, falls er bereits alle erforderlichen Waren aufgenommen hat. Kommissionieranlagen dieser Art sind mittlerweile weit verbreitet. Teilweise werden statt Förderbändern andere Transportsysteme wie z.B. Gondelbahnen eingesetzt, jedoch ist die prinzipielle Funktionsweise meistens ähnlich.

Ein Vorteil solcher Kommissionieranlagen liegt darin, dass man auch stark unterschiedliche Auftragsvolumina durch eine entsprechende Anzahl von Arbeitern relativ effizient bearbeiten kann. Die Anzahl der Arbeiter zu variieren, stellt meistens kein Problem dar, da es sich bei ihnen in der Regel um Aushilfskräfte handelt, die nur während ihrer Einsatzzeiten bezahlt werden. Sind sehr viele Aufträge zu bearbeiten, so werden natürlich viele oder gar alle Arbeitsplätze besetzt, um einen möglichst hohen Durchsatz zu erzielen. Im Regelfall werden aber weit weniger Arbeiter eingesetzt, da andernfalls viele von ihnen an ihren Arbeitsplätzen die meiste Zeit nichts zu tun hätten. In diesem Fall müssen die Arbeiter ständig ihre Arbeitspositionen wechseln, um alle Aufträge zu bedienen. Die Zeit, die sie für das Wechseln der Arbeitsposition benötigen, geht natürlich für die Bearbeitung der Aufträge verloren. Bei Messungen hat man festgestellt, dass die Arbeiter teilweise bis zu 50 Prozent der Zeit mit dem Wechseln der Arbeitspositionen verbringen. Daher ist es von entscheidendem Interesse, die Bewegungen der Arbeiter zu minimieren. In dieser Arbeit wollen wir das Kernproblem, welches dieser Problematik zugrunde liegt, analysieren und effiziente Bewegungsstrategien und Algorithmen dafür entwickeln. Wir modellieren dabei die Situation, in der genau ein Arbeiter alle Arbeitsplätze an einem Nebenförderband bedient. Wir halten dieses für ein Basisproblem der Bewegungsminimierung in solchen Förderbandsystemen, welches nach unserem Wissen bislang von niemandem analysiert wurde.

Kommissionieranlagen selbst wurden bereits von anderen Autoren untersucht. In [6] beschreibt de Koster azyklische Kommissionieranlagen als ein Netz verbundener Transportbänder und Arbeitsplätze. Zu einer gegebenen Kommissionieranlage ermittelt er Informationen über den Durchsatz, die Auslastung der Arbeiter und die durchschnittliche Anzahl der im System befindlichen Behälter. Die Analyse beruht auf Jackson Netzwerken; nähere Informationen dazu findet man in [12]. Kommissionieranlagen werden ferner auch in [2, 18] untersucht. Probleme, die beim Abstapelprozess der fertig gepackten Behälter in den Abstapelanlagen auftreten, werden in [21, 19, 20, 22] behandelt.

Theoretisches Modell Wir modellieren das Kernproblem in dieser Arbeit wie folgt. Gegeben sind m linear angeordnete Maschinen (Arbeitsplätze) $P_1, P_2, \ldots, P_m$ sowie eine Folge $J_1, J_2, \ldots, J_n$ von n Jobs (Aufträgen). Für jeden Job ist angegeben, an welchen der Maschinen er bearbeitet werden muss. Diese Information wird durch eine $n \times m$-Matrix mit Einträgen aus $\{0, 1\}$ angegeben. Ein 1-Eintrag in der Matrix bedeutet, dass der zugehörige Job an der entsprechenden Maschine bearbeitet werden muss. Wir sprechen daher für jeden 1-Eintrag der Matrix von einer Task, die ausgeführt werden muss. Die Jobs werden an den Maschinen in Permutations-Flow-Shop-Manier bearbeitet, d.h. die Jobs besuchen die Maschinen in der Reihenfolge $P_1, \ldots, P_m$ und die Maschinen bearbeiten die Jobs in der Reihenfolge $J_1, \ldots, J_n$. An jeder Maschine befindet sich zu jedem Zeitpunkt höchstens ein Job. Ein Job kann eine Maschine nur dann verlassen, wenn die nachfolgende Maschine nicht mehr von einem Job belegt ist. Für die Bearbeitung eines Jobs an einer Maschine ist die

Anwesenheit des Arbeiters erforderlich. Um alle Tasks auszuführen, muss der Arbeiter die Maschinen wiederholt wechseln. Mit den Maschinenwechseln sind Kosten verbunden, die in einer Distanzmatrix angegeben sind. Für die formale Definition des Problems in Kapitel 2 setzen wir lediglich voraus, dass die Distanzfunktion der Dreiecksungleichung genügt und dass ein Verbleiben des Arbeiters an der Maschine mit keinen Kosten verbunden ist. In dieser Arbeit betrachten wir jedoch im Wesentlichen nur zwei verschiedene Distanzfunktionen. Dies sind einerseits die Einheitsdistanzen, bei denen die Distanz zwischen zwei beliebigen verschiedenen Maschinen immer gleich 1 ist, und andererseits die linearen Distanzen, bei denen die Distanz zwischen Maschine P_i und Maschine P_j durch den Wert $|i-j|$ gegeben ist. Eine Verallgemeinerung der linearen Distanzen stellen die additiven Distanzen dar. Dabei ergibt sich die Distanz zwischen zwei Maschinen als die Summe der Distanzen zwischen benachbarten Maschinen, wobei diese Distanzen jedoch unterschiedlich sein können und nicht wie bei linearen Distanzen stets 1 sein müssen. Da die meisten Resultate für lineare Distanzen auch für die additiven Distanzen gelten, betrachten wir oft gleich diese Verallgemeinerung. Das Ziel besteht in jedem Fall darin, eine Bearbeitungsreihenfolge der Tasks (nicht der Jobs!) zu bestimmen, so dass die gesamten Kosten für die Bewegungen des Arbeiters bei der Abarbeitung der Tasks in dieser Reihenfolge minimal sind. Dabei sind natürlich nicht alle Reihenfolgen erlaubt, sondern nur solche, die durch die Gegebenheiten der Flow-Shop-Verarbeitung möglich sind.

Flow-Shop-Verarbeitungen werden oft mit Scheduling-Problemen in Verbindung gebracht, die Gegenstand sehr vieler theoretischer und praxisorientierter Untersuchungen sind, siehe z.B. [3, 5, 17, 26, 27]. Ein wesentlicher Unterschied zwischen Flow-Shop-Scheduling-Problemen und unserem Problem besteht darin, dass bei Scheduling-Problemen die Maschinen gleichzeitig arbeiten können und immer Optimierungsfunktionen betrachtet werden, die in irgendeiner Form zeitabhängig sind. Bei unserem Problem hingegen werden die Tasks sequentiell ausgeführt und die Optimierungsfunktion bezieht sich nur auf die Bewegungen des Arbeiters. Die genauen Bearbeitungszeiten der Tasks spielen bei uns deshalb keine Rolle.

Ferner besteht bei Scheduling-Problemen die Aufgabe meistens darin, eine optimale Jobreihenfolge zu bestimmen. In unserem Problem hingegen sehen wir die Reihenfolge der Jobs als gegeben an, da sie in der Praxis kaum beeinflussbar ist. Dies ist vor allem auf folgende Gründe zurückzuführen. Nicht alle Behälter müssen die gleichen Nebenförderbänder anfahren. Da nicht alle Arbeitsplätze an den Nebenförderbändern besetzt sind, müssen die Behälter des Öfteren warten, bis sie von einem Arbeiter bedient werden. Dadurch stauen sich natürlich auch die nachfolgenden Behälter. Ist der Anfang des Nebenförderbandes von einem Behälter belegt, so kann kein neuer Behälter vom Hauptförderband auf das Nebenförderband ausgeschleust werden. In diesem Fall müssen Behälter, die eigentlich dieses Nebenförderband ansteuern müssten, eine zusätzliche vollständige Runde auf dem Hauptförderband zurücklegen. Alle diese Faktoren bedingen, dass die Reihenfolge der Behälter auf dem Nebenförderband nur sehr bedingt durch die Reihenfolge, in der sie in das System gelangen, beeinflussbar ist. Bei den meisten Flow-Shop-Scheduling-Problemen ist darüber hinaus eine Flow-Shop-Verarbeitung, bei der ein Job einen anderen blockieren kann, wie dies in unserem Modell der Fall ist, ausgeschlossen. Es gibt jedoch

Ausnahmen, in denen auch solche Modelle betrachtet werden, siehe z.B. [14]. Für eine Flow-Shop-Verarbeitung, bei der es vor jeder Maschine einen beschränkten Puffer gibt, der nicht überlaufen darf, wurde das Problem der Minimierung der Maschinenwechsel des Arbeiters bereits in [8] betrachtet.

Ein anderes Problem, welches auf den ersten Blick eine gewisse Verwandtschaft zu unserem Problem aufweist, ist das Robotic-Cell-Problem. Dort findet ebenfalls eine Flow-Shop-Verarbeitung an linear angeordneten Maschinen statt. Ein Roboter transportiert dabei die Jobs von Maschine zu Maschine. Es ist eine Reihenfolge der Jobs und eine Steuerung des Roboters gesucht, so dass die Summe der vom Roboter zurückgelegten Wegstecke minimal wird. Da beim Robotic-Cell-Problem der Transport der Jobs gesteuert werden soll und die Bearbeitung der Jobs an den Maschinen automatisch erfolgt, während bei unserem Problem der Transport der Jobs automatisch erfolgt und die Bewegung des Arbeiters zur Bearbeitung der Jobs gesteuert werden soll, ergeben sich zwei sehr unterschiedliche Probleme. Untersuchungen des Robotic-Cell-Problems findet man beispielsweise in [24, 13, 25].

Zusammenfassung der Ergebnisse Die in dieser Arbeit erzielten Ergebnisse erstrecken sich von Komplexitätsaussagen über Approximations- und Online-Algorithmen bis zu empirischen Auswertungen. Im Kapitel 2 geben wir eine formale Beschreibung unseres Problems an und führen die grundlegenden Begriffe ein. Wir zeigen hier außerdem, dass man die in unserem Modell gültigen Bearbeitungsreihenfolgen durch eine einfache Vorgängerrelation auf der Menge der Tasks beschreiben kann. In Kapitel 3 untersuchen wir die Komplexität des Problems. Hierzu betrachten wir wie üblich die Entscheidungsversion des Problems. Dabei ist zusätzlich zur eigentlichen Instanz noch eine Zahl K gegeben, und es ist zu entscheiden, ob es eine Abarbeitung der Taskmatrix gibt, so dass die vom Arbeiter zurückgelegte Distanz bezüglich der gegebenen Distanzmatrix höchstens K ist. Zunächst weisen wir die NP-Vollständigkeit des Problems für allgemeine Distanzfunktionen nach. Der Beweis beruht auf einer sehr einfachen polynomiellen Transformation von dem Problem 'Gerichteter Hamiltonweg', welches bekanntermaßen NP-vollständig ist, siehe [11]. Wir zeigen dann, dass das Problem für allgemeine Distanzfunktionen in polynomieller Zeit lösbar ist, sofern entweder die Anzahl der Maschinen oder die Anzahl der Jobs durch eine Konstante beschränkt ist. In diesem Fall ist nämlich die Zahl der möglichen Konfigurationen polynomiell beschränkt, und man kann mit Hilfe dynamischer Programmierung in polynomieller Zeit eine minimale Abarbeitung bestimmen. Im dritten Abschnitt von Kapitel 3 betrachten wir dann ausnahmsweise den Fall, dass die Reihenfolge der Jobs nicht fest vorgegeben ist. Es ist dann eine Jobreihenfolge und eine Abarbeitung für diese Jobreihenfolge gesucht, so dass diese Abarbeitung minimal unter allen Abarbeitungen für alle möglichen Jobreihenfolgen ist. Wir beweisen, dass die Entscheidungsversion dieses Problems sowohl für Einheitsdistanzen als auch für lineare Distanzen NP-vollständig ist. Im letzten Abschnitt des Kapitels zeigen wir dann schließlich, dass das Problem der Minimierung der Bewegungen des Arbeiters für Einheitsdistanzen auch bei fest vorgegebener Jobreihenfolge NP-vollständig ist. Leider lässt sich dieser Beweis nicht auf lineare Distanzen übertragen. Für lineare Distanzen bleibt die Komplexität des Problems ungeklärt.

In Kapitel 4 geben wir Approximationsalgorithmen sowohl für Einheitsdistanzen als

auch für additive Distanzen an, deren Laufzeit linear in der Größe der gegebenen Taskmatrix ist. Im Fall der Einheitsdistanzen garantiert der Algorithmus Lösungen, bei denen die zurückgelegte Distanz höchstens zweimal so groß ist wie die einer optimalen Lösung. Bei additiven Distanzen sind die berechneten Lösungen höchstens um den Faktor 3 schlechter als die optimalen Lösungen. Empirische Untersuchungen belegen, dass die Abweichungen der Lösungen vom Optimum bei zufällig gewählten Taskmatrizen in der Regel weit unterhalb der garantierten Faktoren 2 bzw. 3 liegen.

In Kapitel 5 betrachten wir Online-Algorithmen, bei denen nicht die gesamte Jobsequenz bekannt ist. Online-Algorithmen sind für unser Problem aus zweierlei Gründen von besonderem Interesse. Zum einen ist es aus den oben bereits geschilderten Gründen normalerweise nicht vorhersagbar, in welcher Reihenfolge die Jobs an die Maschinen gelangen, so dass die gesamte Jobsequenz nicht von vornherein bekannt ist. Zum anderen sind Online-Algorithmen oft leicht umsetzbar. Im Idealfall sind die Online-Strategien so einfach, dass der Arbeiter sie unmittelbar erlernen und anwenden kann. In unserem Fall bezeichnen wir einen Algorithmus als Online-Algorithmus, wenn er nur Informationen über die Jobs verwendet, die die erste Maschine des Förderbandes bereits erreicht haben.

Bei Online-Algorithmen ist besonders ihre Güte, die man im allgemeinen mit Hilfe des kompetitiven Faktors angibt, von Interesse. Der kompetitive Faktor vergleicht dabei die Lösung des Algorithmus mit einer optimalen Offline-Lösung im Worst-Case. Näheres dazu findet man unter anderem in [4, 10, 15].

Wir untersuchen Online-Algorithmen für lineare bzw. additive Distanzen. Wir betrachten zunächst einige einfache Algorithmen und zeigen, dass sie fast alle einen kompetitiven Faktor in $\Omega(m)$ haben, wobei m die Anzahl der Maschinen ist. Wir untersuchen dann einen Algorithmus, den wir Free-Left nennen und der in experimentellen Auswertungen fast immer die besten Ergebnisse liefert. Abgesehen von den guten Resultaten hat dieser Algorithmus den Vorteil, dass er nur sehr wenig Information verwendet, die zudem unmittelbar verfügbar ist. Wir können für diesen Algorithmus nachweisen, dass er für additive Distanzen einen kompetitiven Faktor in $O(\log m)$ hat. Für eine spezielle Klasse von Taskmatrizen liefert er sogar optimale Abarbeitungen. Außerdem können wir zeigen, dass der Free-Left-Algorithmus in vielen Situationen optimale Entscheidungen trifft, was eine Erklärung für seine guten Ergebnisse liefert. Allerdings ist der kompetitive Faktor von Free-Left keine Konstante. Wir geben Beispiele an, die belegen, dass der kompetitive Faktor von Free-Left selbst für lineare Distanzen in $\Omega\left(\frac{\log m}{\log\log m}\right)$ liegt. In der Praxis scheint der Free-Left-Algorithmus jedoch einen wesentlich geringeren Fehler aufzuweisen als der theoretisch bewiesene Faktor aus $O(\log m)$. Für lineare Distanzen liegt dieser Fehler bei zufällig gewählten Taskmatrizen fast immer unter 20 Prozent, oft sogar deutlich darunter. Dies zeigen empirische Auswertungen, auf die wir im letzten Kapitel kurz eingehen.

Kapitel 2

Grundlagen

Im ersten Abschnitt dieses Kapitels geben wir eine formale Beschreibung des Problems der Bewegungsminimierung in der Förderband-Flow-Shop-Verarbeitung mit einem Arbeiter an und führen einige Bezeichnungen ein. Wir definieren das Problem für eine vorgegebene Jobreihenfolge, da es in dieser Arbeit fast ausschließlich in dieser Form betrachtet wird. Dieses wiederum ist darin begründet, dass die Reihenfolge der Jobs in der Praxis meistens nicht zu beeinflussen ist. Wir nennen das Problem in dieser Form kurz das *Förderband-Flow-Shop-Problem*. Am Ende des ersten Abschnitts geben wir noch eine Variante des Problems an, bei dem die Jobreihenfolge nicht fest vorgegeben ist. Diese Variante wird jedoch im weiteren Verlauf nur noch einmal im Kapitel 3 betrachtet, in dem wir uns mit der Komplexität des Problems befassen. Wir bezeichnen diese Variante als *Förderband-Flow-Shop-Problem mit variabler Jobreihenfolge*. Im zweiten Abschnitt dieses Kapitels zeigen wir einige grundlegende Aussagen.

2.1 Formale Beschreibung des Problems

Eine Instanz des Förderband-Flow-Shop-Problems für $n \geq 1$ *Jobs* $J_1, \ldots, J_n$ und $m \geq 1$ *Maschinen* $P_1, \ldots, P_m$ wird durch ein Tupel $(\mathcal{M}, \mathcal{D})$ beschrieben. Dabei ist

- $\mathcal{M} = (\mu_{j,p})_{1 \leq j \leq n, 1 \leq p \leq m}$ eine $(n \times m)$-*Taskmatrix* mit $\mu_{j,p} \in \{0, 1\}$,
- $\mathcal{D} = (\delta_{p,q})_{0 \leq p,q \leq m}$ eine $((m+1) \times (m+1))$-*Distanzmatrix* mit $\delta_{p,q} \in \mathbb{R}_{\geq 0}$ und $\delta_{p,p} = 0$ für alle $p, q \in \{0, \ldots, m\}$, so dass für alle $0 \leq p, q, r \leq m$, $q \neq 0$ die Dreiecksungleichung $\delta_{p,r} \leq \delta_{p,q} + \delta_{q,r}$ gilt.

Jeder Eintrag $\mu_{j,p}$ der Taskmatrix, für den $\mu_{j,p} = 1$ gilt, definiert eine *Task* $T_{(j,p)}$. Die Menge aller Tasks der Taskmatrix $\mathcal{M}$ bezeichnen wir mit $\mathcal{T}(\mathcal{M})$, d.h.

$$\mathcal{T}(\mathcal{M}) := \{T_{(j,p)} \mid \mu_{j,p} = 1\}.$$

Für eine Task $T = T_{(j,p)}$ nennen wir j den *Jobindex* und p den *Maschinenindex* der Task. Der Job J_j besteht aus allen Tasks mit Jobindex j. Jede Task $T = T_{(j,p)}$ muss an der

Maschine P_p unter Anwesenheit des Arbeiters bearbeitet werden. Die Distanz, die der Arbeiter zurücklegen muss, um von der Maschine P_p zur Maschine P_q zu gelangen, beträgt $\delta_{p,q}$. Die *Startdistanz*, die benötigt wird, wenn die Bearbeitung der Jobsequenz an der Maschine P_p beginnt, beträgt $\delta_{0,p}$, und die *Enddistanz*, die zurückgelegt werden muss, wenn der Arbeiter die Bearbeitung der Jobsequenz an der Maschine P_p beendet, beträgt $\delta_{p,0}$. Sofern nicht ausdrücklich etwas anderes gesagt wird, bezeichnet in der gesamten Arbeit $\mathcal{M}$ immer die gegebene Taskmatrix, n und m sind stets die Anzahl der Jobs bzw. die Anzahl der Maschinen der Taskmatrix, und $\mathcal{D}$ bezeichnet die Distanzmatrix.

Eine Abarbeitung der Jobsequenz (oder der Taskmatrix) wird durch die Reihenfolge beschrieben, in welcher der Arbeiter die Tasks bearbeitet. Aufgrund der Gegebenheiten eines Förderband-Flow-Shops sind natürlich nur bestimmte Ausführungsreihenfolgen möglich. Damit eine Task $T_{(j,p)}$ bearbeitet werden kann, muss sich der zugehörige Job J_j an der Maschine P_p befinden. Um die möglichen Ausführungsreihenfolgen der Tasks zu beschreiben, werden wir die möglichen *Zustände*, die *Positionen der Jobs in einem Zustand* und die in einem Zustand *ausführbaren Tasks* definieren. Bevor wir diese Begriffe präzise einführen, wollen wir ihre Bedeutung zunächst informell beschreiben. Ein Zustand wird durch eine Menge $\mathcal{T} \subseteq \mathcal{T}(\mathcal{M})$ beschrieben, welche die noch zu bearbeitenden Tasks enthält. Die Position eines Jobs ist eine Zahl aus $\{p \in \mathbb{Z} \mid p \leq m\} \cup \{\infty\}$. Eine Position $p \in \{1, \ldots, m\}$ bedeutet, dass sich der Job an der Maschine P_p befindet. Eine Jobposition $p \leq 0$ sagt aus, dass vor dem Job in der Jobsequenz noch $|p|$ andere Jobs sind, die die erste Maschine noch nicht erreicht haben. Die Position '∞' haben diejenigen Jobs, welche die letzte Maschine bereits passiert haben. An dieser speziellen Position können sich mehrere Jobs befinden, während sich an allen anderen Positionen in einem Zustand jeweils höchstens ein Job aufhalten kann. Eine Task $T_{(j,p)}$ ist in einem Zustand ausführbar, wenn sie noch nicht bearbeitet wurde und sich der Job J_j an der Maschine P_p befindet.

Wir geben jetzt die formalen Definitionen an. Für '∞' benutzen wir dabei die üblichen Rechenregeln. Insbesondere bedeutet dies, dass $\infty + \infty = \infty$ sowie $x + \infty = \infty$ und $x < \infty$ für jedes $x \in \mathbb{R}$ gilt. $\infty - \infty$ ist nicht definiert. Außerdem setzen wir, wie ebenfalls gebräuchlich, $\min \emptyset := \infty$.

Wir definieren zunächst die Position eines Jobs. Der Einfachheit halber definieren wir diese für eine beliebige Menge $\mathcal{T} \subseteq \mathcal{T}(\mathcal{M})$ noch zu bearbeitender Tasks, obwohl wir uns eigentlich nur für solche Teilmengen interessieren, die auch gültige Zustände darstellen. Für eine beliebige Teilmenge $\mathcal{T} \subseteq \mathcal{T}(\mathcal{M})$ und für einen Jobindex $j \in \{1, \ldots, n\}$ bezeichne

$$\sigma^{\mathcal{T}}(j) := \min\{p \mid T_{(j,p)} \in \mathcal{T}\}$$

den kleinsten Index einer Maschine, an der der Job J_j noch bearbeitet werden muss, bzw. ∞, falls $\mathcal{T}$ keine zu J_j gehörige Task enthält. Die *Position $\rho^{\mathcal{T}}(j)$ eines Jobs J_j für eine Menge noch zu bearbeitender Tasks $\mathcal{T}$* ist nun wie folgt definiert. Falls $\mathcal{T} = \emptyset$, so ist $\rho^{\mathcal{T}}(j) = \infty$ für alle Jobs J_j, $j = 1, \ldots, n$. Andernfalls sei j_0 der minimale Jobindex, so dass $\mathcal{T}$ eine Task mit Jobindex j_0 enthält. Dann ist $\rho^{\mathcal{T}}(j) = \infty$ für $1 \leq j < j_0$ und für

$j \in \{j_0, j_0+1, \ldots, n\}$ ergibt sich die Position des Jobs J_j rekursiv durch

$$\rho^{\mathcal{T}}(j) = \begin{cases} \sigma^{\mathcal{T}}(j), & \text{falls } j = j_0, \\ \min\{\sigma^{\mathcal{T}}(j), \rho^{\mathcal{T}}(j-1) - 1\}, & \text{falls } j > j_0. \end{cases}$$

Ein Job rückt also bis zu der Position vor, an der er als nächstes bearbeitet werden muss, sofern er nicht an einer Maschine verbleiben muss, weil der vorangehende Job noch die nächste Maschine belegt, siehe auch Abbildung 2.1. Aus der Definition ergeben sich für einen Jobindex j unmittelbar die folgenden Implikationen

$$\rho^{\mathcal{T}}(j) < \infty \quad \Longrightarrow \quad \forall j' \geq j : \rho^{\mathcal{T}}(j') \leq \rho^{\mathcal{T}}(j) - (j' - j) \tag{2.1}$$

und

$$\rho^{\mathcal{T}}(j) \neq \sigma^{\mathcal{T}}(j) \quad \Longrightarrow \quad \exists\, T_{(j',p')} \in \mathcal{T} : \quad \begin{array}{l} j' < j,\ \rho^{\mathcal{T}}(j') = \sigma^{\mathcal{T}}(j') = p' \text{ und} \\ \rho^{\mathcal{T}}(j) = p' - (j - j'). \end{array} \tag{2.2}$$

Wir definieren nun die *Zustände* $\mathcal{T} \subseteq \mathcal{T}(\mathcal{M})$ für eine Taskmatrix $\mathcal{M}$ rekursiv wie folgt.

1. $\mathcal{T} = \mathcal{T}(\mathcal{M})$ ist ein Zustand für $\mathcal{M}$. Wir nennen diesen Zustand den *Anfangszustand*.
2. Falls $\mathcal{T}$ ein Zustand für $\mathcal{M}$ und $T = T_{(j,p)}$ eine Task aus $\mathcal{T}$ mit $\rho^{\mathcal{T}}(j) = p$ ist, so ist $\mathcal{T}' := \mathcal{T} - \{T\}$ ein Zustand für $\mathcal{M}$. $\mathcal{T}'$ ist der *Nachfolgezustand von* $\mathcal{T}$, in den $\mathcal{T}$ *nach Bearbeitung der Task* T *übergeht*. Wir schreiben hierfür $\mathcal{T} \to_T \mathcal{T}'$ oder auch $\mathcal{T} \to_{(j,p)} \mathcal{T}'$.

Für einen Zustand $\mathcal{T}$ nennen wir $\rho^{\mathcal{T}}(j)$ die *Position des Jobs* J_j *im Zustand* $\mathcal{T}$. Die *ausführbaren Tasks in einem Zustand* $\mathcal{T}$ sind die Tasks $T_{(j,p)} \in \mathcal{T}$ mit $\rho^{\mathcal{T}}(j) = p$.

Wir sagen, dass eine Task $T \in \mathcal{T}(\mathcal{M})$ in einem Zustand $\mathcal{T}$ bereits *bearbeitet* oder *ausgeführt* wurde, wenn $T \notin \mathcal{T}$ gilt. Andernfalls ist die Task *noch nicht bearbeitet* oder *ausgeführt*. Unmittelbar aus den Definitionen folgt, dass in einem Zustand $\mathcal{T}$, alle Tasks $T_{(j,p)} \in \mathcal{T}(\mathcal{M})$ mit $p < \rho^{\mathcal{T}}(j)$ bereits bearbeitet wurden, während die Tasks $T_{(j,p)}$ mit $p > \rho^{\mathcal{T}}(j)$ noch nicht ausgeführt wurden. Die Tasks $T_{(j,p)}$ mit $p = \rho^{\mathcal{T}}(j)$ können ausgeführt oder nicht ausgeführt sein. Abbildung 2.1 zeigt eine Taskmatrix und die Positionen der Jobs in einem Zustand, in dem drei Tasks bereits bearbeitet wurden.

Ist eine Task $T_{(j,p)}$ in einem Zustand $\mathcal{T}$ ausführbar, so sagen wir auch, dass *der Job* J_j *im Zustand* $\mathcal{T}$ *an der Maschine* P_p *auf Bearbeitung wartet*. Da es in jedem Zustand $\mathcal{T} \neq \emptyset$ mindestens eine ausführbare Task gibt, besitzt jeder solche Zustand mindestens einen Nachfolgezustand. Daraus folgt, dass auch $\mathcal{T} = \emptyset$ ein Zustand für $\mathcal{M}$ ist. Wir nennen diesen Zustand den *Endzustand*.

Eine *Teilabarbeitung der Taskmatrix* $\mathcal{M}$ besteht aus einer Folge

$$T_1, T_2, \ldots, T_r$$

von $r \geq 0$ paarweise verschiedenen Tasks aus $\mathcal{T}(\mathcal{M})$, so dass $\mathcal{T}_k := \mathcal{T}(\mathcal{M}) - \{T_1, \ldots, T_k\}$ für $k = 0, 1, \ldots, r$ ein Zustand ist und für $k = 0, 1, \ldots, r-1$

$$\mathcal{T}_k \to_{T_{k+1}} \mathcal{T}_{k+1}$$

$$\mathcal{M} = \begin{array}{c|ccccc} & P_1 & P_2 & P_3 & P_4 & P_5 \\ \hline J_1 & 0 & 1 & 1 & 0 & 0 \\ J_2 & 0 & 0 & 0 & 1 & 0 \\ J_3 & 1 & 0 & 0 & 1 & 0 \\ J_4 & 1 & 0 & 0 & 0 & 1 \\ J_5 & 0 & 0 & 1 & 0 & 0 \\ J_6 & 1 & 0 & 0 & 1 & 0 \end{array}$$

$$\mathcal{T} = \{ T_{(2,4)}, T_{(3,4)}, T_{(4,1)}, T_{(4,5)}, T_{(5,3)}, T_{(6,1)}, T_{(6,4)} \}$$

j	1	2	3	4	5	6
$\sigma^{\mathcal{T}}(j)$	∞	4	4	1	3	1

j	1	2	3	4	5	6
$\rho^{\mathcal{T}}(j)$	∞	4	3	1	0	−1

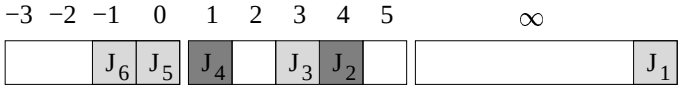


Abbildung 2.1: Taskmatrix für 6 Jobs und 5 Maschinen. Unten sind die Positionen der Jobs im Zustand $\mathcal{T} = \{T_{(2,4)}, T_{(3,4)}, T_{(4,1)}, T_{(4,5)}, T_{(5,3)}, T_{(6,1)}, T_{(6,4)}\}$ dargestellt. Die Tasks $T_{(1,2)}$,$T_{(1,3)}$ und $T_{(3,1)}$ wurden bereits bearbeitet. Die Jobs J_2 und J_4 sind dunkler gezeichnet, um anzuzeigen, dass sie sich an einer Maschine befinden, an der sie noch bearbeitet werden müssen.

gilt. Beinhaltet die Teilabarbeitung alle Tasks aus $\mathcal{T}(\mathcal{M})$, so nennen wir die Teilabarbeitung eine *Abarbeitung der Taskmatrix*. Da jeder Zustand $\mathcal{T} \neq \emptyset$ einen Nachfolgezustand besitzt, lässt sich jede Teilabarbeitung zu einer Abarbeitung fortsetzen.

Für eine Teilabarbeitung $\Lambda = (T_1, T_2, \ldots, T_r)$ der Taskmatrix $\mathcal{M}$ mit $T_i = T_{(j_i,p_i)}$, $i = 1, \ldots, r$, definieren wir $\Delta^{\star\mathcal{D}}(\Lambda) = 0$, falls Λ die leere Folge ist und

$$\Delta^{\star\mathcal{D}}(\Lambda) := \delta_{0,p_1} + \sum_{i=1}^{r-1} \delta_{p_i,p_{i+1}}$$

sonst. Ist Λ eine Abarbeitung der Taskmatrix $\mathcal{M}$, so ist die *Distanz* $\Delta^{\mathcal{D}}(\Lambda)$ der Abarbeitung bezüglich der Distanzmatrix $\mathcal{D}$ definiert durch $\Delta^{\mathcal{D}} = 0$, falls Λ die leere Folge ist und durch

$$\Delta^{\mathcal{D}}(\Lambda) := \Delta^{\star\mathcal{D}}(\Lambda) + \delta_{p_r,0} = \delta_{0,p_1} + \left(\sum_{i=1}^{r-1} \delta_{p_i,p_{i+1}} \right) + \delta_{p_r,0}$$

sonst.

Das Ziel beim Förderband-Flow-Shop-Problem besteht darin, eine Abarbeitung mit minimaler Distanz zu finden. Eine solche Abarbeitung nennen wir eine *minimale Abarbeitung von $\mathcal{M}$ bezüglich $\mathcal{D}$*. Die Distanz einer minimalen Abarbeitung von $\mathcal{M}$ bezüglich $\mathcal{D}$ bezeichnen wir mit $\Delta^{\mathcal{D}}_{\min}(\mathcal{M})$.

In dieser Arbeit interessieren wir uns im Wesentlichen nur für zwei Distanzmaße. Dies sind zum Ersten die *Einheitsdistanzen*, bei denen die Distanzmatrix $\mathcal{D} = (\delta_{p,q})_{0 \le p,q \le m}$ durch

$$\delta_{p,q} = \begin{cases} 1, & p \neq q \text{ und } q \neq 0 \\ 0, & \text{sonst} \end{cases}, \quad 0 \le p, q \le m,$$

definiert ist. Die Kosten für einen Wechsel von einer Maschine zu einer beliebigen anderen Maschine sind bei Einheitsdistanzen also stets 1, während ein Verbleiben an der Maschine keine Kosten verursacht. Die Festlegung der Start- und Enddistanzen ist eher technischer Natur, um spätere Aussagen leichter formulieren zu können. Die hier gewählte Festlegung entspricht dem Fall, dass bei der Aufnahme der Arbeit an einer beliebigen Maschine stets die Kosten 1 anfallen, während bei der Beendigung der Arbeit an einer beliebigen Maschine keine Kosten entstehen. Bei der Betrachtung von Einheitsdistanzen bezeichnen wir die Distanz einer Abarbeitung Λ bzw. die Distanz einer minimalen Abarbeitung einer Taskmatrix $\mathcal{M}$ mit $\Delta^{\mathrm{E}}(\Lambda)$ bzw. $\Delta^{\mathrm{E}}_{\min}(\mathcal{M})$. Falls aus dem Zusammenhang klar ist, dass wir Einheitsdistanzen betrachten, schreiben wir auch nur $\Delta(\Lambda)$ bzw. $\Delta_{\min}(\mathcal{M})$.

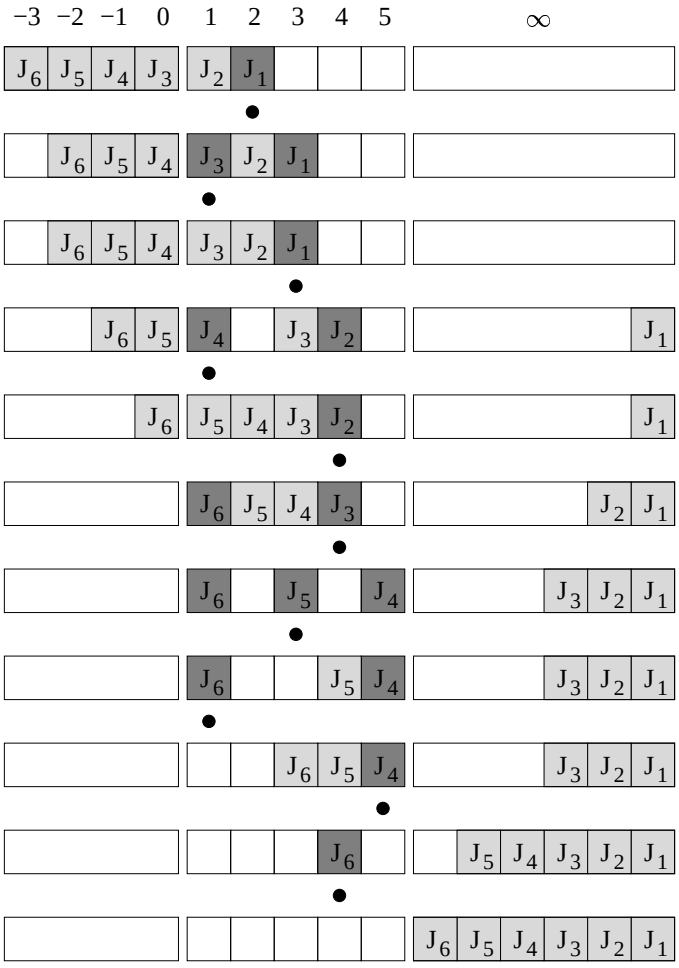
Zum Zweiten interessieren wir uns für die *additiven Distanzen*, wobei das besondere Augenmerk auf dem Spezialfall der *linearen Distanzen* liegt. Wir sprechen von additiven Distanzen, wenn die Distanzmatrix die folgenden drei Bedingungen erfüllt:

1. $\forall\, p, q, r,\ 0 \le p \le q \le r \le m: \quad \delta_{p,r} = \delta_{p,q} + \delta_{q,r} \quad \text{und} \quad \delta_{r,p} = \delta_{r,q} + \delta_{q,p}.$
2. $\delta_{0,1} = \delta_{1,0} = 0.$
3. $\forall p, q \in \{1, \ldots, m\} : \delta_{p,q} = \delta_{q,p}.$

Dies bedeutet, dass die additiven Distanzen bereits vollständig durch die Entfernungen $\delta_{p,p+1}$ und $\delta_{p+1,p}$, $p = 1, \ldots, m-1$, benachbarter Maschinen bestimmt sind. Wir können daher jede Bewegung als eine Hintereinanderreihung von Bewegungen zwischen benachbarten Maschinen auffassen. Die Festlegung der Start- und Enddistanzen $\delta_{0,p}$ bzw. $\delta_{p,0}$ ist wieder eher technischer Natur und entspricht der Annahme, dass der Arbeiter seine Arbeit an der ersten Maschine P_1 beginnt und dorthin nach Bearbeitung der letzten Task zurückkehrt. Da sich der Arbeiter wegen dieser Festlegung in jeder Abarbeitung genauso oft von der Maschine P_p zur Maschine P_{p+1} bewegt wie umgekehrt, bedeutet die dritte Bedingung in der Definition keine wirkliche Einschränkung. Erfüllt die Distanzmatrix $\mathcal{D}$ nämlich nur die beiden ersten Bedingungen und definiert man $\mathcal{D}' = (\delta'_{p,q})$ durch $\delta'_{p,q} := \frac{1}{2}(\delta_{p,q} + \delta_{q,p})$, so erfüllt $\mathcal{D}'$ alle drei Bedingungen der Definition, und es gilt $\Delta^{\mathcal{D}'}(\Lambda) = \Delta^{\mathcal{D}}(\Lambda)$ für alle Abarbeitungen Λ. Als *lineare Distanzen* bezeichnen wir den Spezialfall der additiven Distanzen, bei dem die Abstände $\delta_{p,p+1}$ zwischen zwei benachbarten Maschinen für alle $p = 1, \ldots, m-1$ gleich 1 sind. Bei der Betrachtung von linearen Distanzen bezeichnen wir die Distanz einer Abarbeitung Λ bzw. die Distanz einer minimalen Abarbeitung einer Taskmatrix $\mathcal{M}$ mit $\Delta^{\mathrm{lin}}(\Lambda)$ bzw. $\Delta^{\mathrm{lin}}_{\min}(\mathcal{M})$. Falls aus dem Zusammenhang klar ist, dass wir lineare Distanzen betrachten, schreiben wir auch einfach $\Delta(\Lambda)$ bzw. $\Delta_{\min}(\mathcal{M})$. Abbildung 2.2 zeigt an einem Beispiel die Zustände sowie die zurückgelegten Einheitsdistanzen und linearen Distanzen nach jedem Schritt einer Abarbeitung.

$\mathcal{M} =$

	P_1	P_2	P_3	P_4	P_5
J_1	0	1	1	0	0
J_2	0	0	0	1	0
J_3	1	0	0	1	0
J_4	1	0	0	0	1
J_5	0	0	1	0	0
J_6	1	0	0	1	0



	Δ^E	Δ^{lin}
$T_{(1,2)}$	1	1
$T_{(3,1)}$	2	2
$T_{(1,3)}$	3	4
$T_{(4,1)}$	4	6
$T_{(2,4)}$	5	9
$T_{(3,4)}$	5	9
$T_{(5,3)}$	6	10
$T_{(6,1)}$	7	12
$T_{(4,5)}$	8	16
$T_{(6,4)}$	9	17
	9	20

Abbildung 2.2: Eine Abarbeitung der oben angegebenen Taskmatrix. Die Positionen der Jobs in den einzelnen Zuständen sind graphisch dargestellt. Rechts sind die jeweils bearbeitete Task sowie die aktuellen Einheitsdistanzen und linearen Distanzen angegeben. Die Position, an der die Bearbeitung einer Task durch den Arbeiter erfolgt, ist durch ein '•' gekennzeichnet. Jobs, die auf Bearbeitung warten, sind dunkler gezeichnet. Die Abarbeitung hat die Einheitsdistanz 9 und die lineare Distanz 20. Beides ist nicht minimal, da es Abarbeitungen mit Einheitsdistanz 8 bzw. linearer Distanz 16 gibt.

Im nächsten Kapitel, in dem wir die Problemkomplexität des Förderband-Flow-Shop-Problems untersuchen, betrachten wir noch folgende Variante des soeben beschriebenen Problems, bei dem die Reihenfolge der Jobs gewählt werden kann. Eine Instanz des *Förderband-Flow-Shop-Problems mit variabler Jobreihenfolge* wird genau wie das Förderband-Flow-Shop-Problem mit fester Jobreihenfolge durch ein Tupel $(\mathcal{M}, \mathcal{D})$ beschrieben. Unser Ziel ist jedoch nun ein anderes. Für eine Permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ der Jobindizes sei $\mathcal{M}^\pi$ die Taskmatrix, bei der die $\pi(i)$-ten Zeile mit der i-te Zeile von $\mathcal{M}$ überstimmt. Gesucht ist nun eine Permutation π und eine minimale Abarbeitung Λ der Instanz $(\mathcal{M}^\pi, \mathcal{D})$, so dass $\Delta^{\mathcal{D}}(\Lambda) \leq \Delta^{\mathcal{D}}_{\min}(\mathcal{M}^{\pi'})$ für alle Permutationen π' der Jobindizes gilt.

2.2 Grundlegende Aussagen

Lemma 2.1 *Sei $\mathcal{T} \subseteq \mathcal{T}(\mathcal{M})$ ein Zustand, in dem sich der Job J_j an einer Position $\geq p$ befindet, d.h. $\rho^{\mathcal{T}}(j) \geq p$. Dann sind im Zustand $\mathcal{T}$ alle Tasks $T_{(j',p')}$ mit $j' \leq j$ und $j' + p' < j + p$ bereits ausgeführt.*

Beweis Angenommen, eine Task $T_{(j',p')}$ mit $j' \leq j$ und $j' + p' < j + p$ ist im Zustand $\mathcal{T}$ noch nicht ausgeführt. Dann gilt $\rho^{\mathcal{T}}(j') \leq p' < \infty$ und somit folgt wegen (2.1)

$$\rho^{\mathcal{T}}(j) \leq \rho^{\mathcal{T}}(j') - (j - j') \leq p' - (j - j') < p,$$

was im Widerspruch zur Voraussetzung $\rho^{\mathcal{T}}(j) \geq p$ steht. □

Für die Charakterisierung der Abarbeitungen und Teilabarbeitungen führen wir eine *Vorgängerrelation* '$\prec$' auf der Menge $\mathcal{T}(\mathcal{M})$ der Tasks einer Taskmatrix ein. Diese ist wie folgt für zwei Tasks $T_1 = T_{(j_1,p_1)}$ und $T_2 = T_{(j_2,p_2)}$ definiert:

$$T_1 \prec T_2 : \Longleftrightarrow \; j_1 \leq j_2 \text{ und } j_1 + p_1 < j_2 + p_2. \tag{2.3}$$

Offensichtlich ist '$\prec$' eine irreflexive, antisymmetrische und transitive Relation. Der gerichtete Graph mit den Tasks als Knoten und den gerichteten Kanten $\{(T_1, T_2) \mid T_1 \prec T_2\}$ ist also azyklisch. Falls $T_1 \prec T_2$ gilt, so nennen wir T_1 einen *Vorgänger* oder eine *Vorgängertask* von T_2, bzw. T_2 einen *Nachfolger* oder eine *Nachfolgertask* von T_1. Wir schreiben $T_1 \not\prec T_2$, falls T_1 kein Vorgänger von T_2 ist. Abbildung 2.3 veranschaulicht die Definition der Vorgängerrelation an einer Taskmatrix.

Da wir die Vorgängerrelation immer wieder benutzen werden, ist es hilfreich, sich die Definition anhand der beiden folgenden einfachen Beobachtungen nochmals zu vergegenwärtigen.

Beobachtung 2.1 *Seien $T_1 = T_{(j_1,p_1)}$ und $T_2 = T_{(j_2,p_2)}$ Tasks.*

1. *Falls $p_1 < p_2$, so gilt $T_1 \prec T_2$ genau dann, wenn $j_1 \leq j_2$.*
2. *Falls $p_1 \geq p_2$, so gilt $T_1 \prec T_2$ genau dann, wenn $j_1 + p_1 < j_2 + p_2$.*

	P_1	P_2	P_3	P_4	P_5	P_6
J_1	0	0	0	0	1	1
J_2	1	1	1	0	0	1
J_3	1	1	**1**	1	0	1
J_4	0	0	0	1	0	1
J_5	0	0	1	0	1	1
J_6	1	1	1	0	0	1
J_7	1	1	0	1	0	1

	P_1	P_2	P_3	P_4	P_5	P_6
J_1	0	0	0	0	1	1
J_2	1	1	1	0	0	1
J_3	1	1	1	1	0	1
J_4	0	0	0	1	0	1
J_5	0	0	**1**	0	1	1
J_6	1	1	1	0	0	1
J_7	1	1	0	1	0	1

Abbildung 2.3: Die grau unterlegten Bereiche der Taskmatrizen enthalten die Nachfolger von $T_{(3,3)}$ (linke Taskmatrix) bzw. die Vorgänger von $T_{(5,3)}$ (rechte Taskmatrix).

Lemma 2.2 *Sei $\mathcal{T} \subseteq \mathcal{T}(\mathcal{M})$ ein Zustand und $T \in \mathcal{T}$ eine Task. T ist im Zustand $\mathcal{T}$ genau dann ausführbar, wenn alle Vorgänger von T bereits bearbeitet sind, d.h. wenn $\mathcal{T}$ keinen Vorgänger von T enthält.*

Beweis

'$\Rightarrow$' Falls $T = T_{(j,p)}$ im Zustand $\mathcal{T}$ ausführbar ist, so gilt $\rho^{\mathcal{T}}(j) = p$. Nach Lemma 2.1 sind dann alle Vorgänger von T bereits ausgeführt.

'$\Leftarrow$' Sei $T = T_{(j,p)}$. Da alle Tasks $T_{(j,p')}$ mit $p' < p$ Vorgänger von T sind, sind diese bereits ausgeführt und somit nicht in $\mathcal{T}$ enthalten. Wegen $T \in \mathcal{T}$ muss der Job J_j daher als nächstes an der Maschine P_p bearbeitet werden, d.h. $\sigma^{\mathcal{T}}(j) = p$.

Angenommen, T ist nicht ausführbar. Dann muss $\rho^{\mathcal{T}}(j) < \sigma^{\mathcal{T}}(j) = p$ gelten. Aus Gleichung (2.2) folgt dann, dass es eine noch nicht bearbeitete Task $T' = T_{(j',p')} \in \mathcal{T}$ mit $j' < j$ und $\rho^{\mathcal{T}}(j) = p' - (j - j')$ gibt. Dann ist aber $j' + p' = j + \rho^{\mathcal{T}}(j) < j + p$ und somit ist T' eine Vorgängertask von T, was ein Widerspruch zu der Voraussetzung ist, dass alle Vorgänger von T bereits ausgeführt sind. Also ist T doch ausführbar. □

Wir sagen, dass eine Folge von Tasks $T_1, \ldots, T_r$ *die Vorgängerrelation einhält*, falls die Tasks paarweise verschieden sind und $T_{i_2} \not\prec T_{i_1}$ für alle Indizes i_1, i_2 mit $1 \leq i_1 < i_2 \leq r$ gilt. Eine Taskfolge, die die Vorgängerrelation einhält, nennen wir auch eine *gültige Taskfolge*.

Mit Hilfe von Lemma 2.2 erhalten wir leicht die folgende Charakterisierung der Abarbeitungen und Teilabarbeitungen einer Taskmatrix.

Lemma 2.3 *Sei $\mathcal{M}$ eine Taskmatrix.*

1. *Eine Folge $T_1, \ldots, T_r$ von Tasks aus $\mathcal{T}(\mathcal{M})$ ist genau dann eine Teilabarbeitung, wenn sie die Vorgängerrelation einhält und keine Task $T \in \mathcal{T}(\mathcal{M}) - \{T_1, \ldots, T_r\}$ existiert, die eine Vorgängertask einer der Tasks $T_1, \ldots, T_r$ ist.*

2. *Eine Folge $T_1, \ldots, T_r$ die alle Task aus $\mathcal{T}(\mathcal{M})$ enthält, ist genau dann eine Abarbeitung, wenn sie die Vorgängerrelation einhält.*

Beweis

1. '$\Rightarrow$' Sei $T_1, \ldots, T_r$ eine Teilabarbeitung. Angenommen, die Aussage stimmt nicht. Dann gibt es ein k, $1 \leq k \leq r$, und eine Task $T \in \mathcal{T}(\mathcal{M}) - \{T_1, \ldots, T_k\}$, so dass $T \prec T_k$. Nach Lemma 2.2 wäre dann aber T_k im Zustand $\mathcal{T}(\mathcal{M}) - \{T_1, \ldots, T_{k-1}\}$ nicht ausführbar und $T_1, \ldots, T_r$ könnte keine Teilabarbeitung sein.

   '$\Leftarrow$' Mit Lemma 2.2 folgt durch triviale Induktion, dass $\mathcal{T}_k := \mathcal{T}(\mathcal{M}) - \{T_1, \ldots, T_{k-1}\}$ für $k = 1, \ldots, r$ ein Zustand ist, in dem die Task T_k ausführbar ist. Daraus folgt die Aussage.

2. Folgt unmittelbar aus 1. □

Aufgrund von Lemma 2.3 stellt das Förderband-Flow-Shop-Problem ein so genanntes Sequential Ordering Problem mit einer speziellen Vorgängerrelation dar. Beim Sequential Ordering Problem besteht die Eingabe aus einem vollständigen Graphen mit Kantengewichten sowie einer Vorgängerrelation auf den Knoten. Gesucht ist ein Hamiltonweg mit minimalen Kosten, der die Vorgängerrelation einhält. Das Problem ist offensichtlich NP-vollständig, da es für den Spezialfall der leeren Vorgängerrelation zum gewichteten Hamiltonweg-Problem wird, welches als NP-vollständig bekannt ist. Das Sequential Ordering Problem wurde zum Beispiel in [1] untersucht.

Das folgende Korollar charakterisiert die Zustände eines Förderband-Flow-Shops.

Korollar 2.1 *Sei $\mathcal{M}$ eine Taskmatrix und $\mathcal{T} \subseteq \mathcal{T}(\mathcal{M})$. $\mathcal{T}$ ist genau dann ein Zustand, wenn es keine Tasks T, T' gibt mit $T \in \mathcal{T}$, $T' \in \mathcal{T}(\mathcal{M}) - \mathcal{T}$ und $T \prec T'$.*

Beweis Ein solches Paar T, T' kann offensichtlich in keinem Zustand existieren, da nach Lemma 2.2 die Task T in jeder Teilabarbeitung vor der Task T' bearbeitet werden muss. Existiert andererseits kein solches Paar, so stellt nach Lemma 2.3 eine beliebige Anordnung der Tasks aus $\mathcal{T}(\mathcal{M}) - \mathcal{T}$, welche die Vorgängerrelation einhält, eine Teilabarbeitung dar, die den Anfangszustand in den Zustand $\mathcal{T}$ überführt. Eine solche Anordnung ist natürlich immer möglich, da die Vorgängerrelation azyklisch ist. □

Das folgende Korollar zeigt die auch intuitiv leicht nachvollziehbare Tatsache, dass die Distanz einer minimalen Abarbeitung nicht geringer werden kann, wenn wir Tasks zu unserer Instanz hinzufügen.

Korollar 2.2 *Seien $\mathcal{M} = (\mu_{j,p})_{1 \leq j \leq n, 1 \leq p \leq m}$ und $\mathcal{M}' = (\mu'_{j,p})_{1 \leq j \leq n, 1 \leq p \leq m}$ zwei Taskmatrizen mit $\mu_{j,p} \leq \mu'_{j,p}$ für alle $1 \leq j \leq n$ und $1 \leq p \leq m$. Dann gilt für jede Distanzmatrix $\mathcal{D}$*

$$\Delta^{\mathcal{D}}_{\min}(\mathcal{M}) \leq \Delta^{\mathcal{D}}_{\min}(\mathcal{M}').$$

Beweis Sei Λ' eine minimale Abarbeitung von $\mathcal{M}'$ bezüglich $\mathcal{D}$. Nach Lemma 2.3 hält Λ' die Vorgängerrelation ein. Sei nun Λ die Taskfolge, die man erhält, indem man aus Λ' alle Tasks streicht, die keine Tasks von $\mathcal{M}$ sind. Λ hält dann offensichtlich ebenfalls die Vorgängerrelation ein und ist somit nach Lemma 2.3 eine Abarbeitung von $\mathcal{M}$. Da $\mathcal{D}$ die Dreiecksungleichung einhält, ist offensichtlich $\Delta^{\mathcal{D}}(\Lambda) \leq \Delta^{\mathcal{D}}(\Lambda')$. Daraus folgt die Aussage. □

2.3 Datenstrukturen

Zur Beschreibung von Algorithmen werden wir eine C-ähnliche Syntax verwenden. Dabei bezeichnet '&&' ein logisches Und, '|| ' ein logisches Oder, '!' ein logisches Nicht, '==' den Vergleichsoperator, '=' den Zuweisungsoperator und '++' bzw. '−−' den Inkrement- bzw. Dekrementoperator für Integer-Variablen. Die übrige Syntax sollte ohne weitere Erklärungen verständlich sein. Außerdem benutzen wir die folgenden Felder:

FirstJob[p]	kleinster Jobindex j, so dass $T_{(j,p)}$ eine Task ist, oder $+\infty$, falls kein solcher Index existiert
LastJob[p]	größter Jobindex j, so dass $T_{(j,p)}$ eine Task ist, oder $-\infty$, falls kein solcher Index existiert
NextJob[p][j]	kleinster Jobindex $k > j$, so dass $T_{(k,p)}$ eine Task ist, oder $+\infty$, falls kein solcher Index existiert
PrevJob[p][j]	größter Jobindex $k < j$, so dass $T_{(k,p)}$ eine Task ist, oder $-\infty$, falls kein solcher Index existiert
FirstMachine[j]	kleinster Maschinenindex p, so dass $T_{(j,p)}$ eine Task ist, oder $+\infty$, falls kein solcher Index existiert
LastMachine[j]	größter Maschinenindex p, so dass $T_{(j,p)}$ eine Task ist, oder $-\infty$, falls kein solcher Index existiert
NextMachine[j][p]	kleinster Maschinenindex $q > p$, so dass $T_{(j,q)}$ eine Task ist, oder $+\infty$, falls kein solcher Index existiert
PrevMachine[j][p]	größter Maschinenindex $q < p$, so dass $T_{(j,q)}$ eine Task ist, oder $-\infty$, falls kein solcher Index existiert

Für die Abschätzung der Laufzeit der Algorithmen verwenden wir das Einheitskostenmaß. Da die Werte für die obigen Felder offensichtlich in $O(nm)$ Zeit berechnet werden können und alle von uns betrachteten Algorithmen eine Laufzeit in $\Omega(nm)$ haben, nehmen wir stets an, dass uns die obigen Felder zur Verfügung stehen.

Kapitel 3

Problemkomplexität

In diesem Kapitel untersuchen wir die Komplexität des Förderband-Flow-Shop-Problems. Da NP-Vollständigkeit nur für Entscheidungsprobleme definiert ist, betrachten wir bei NP-Vollständigkeitsbeweisen immer die Entscheidungsversion des Problems, bei der zusätzlich zur Instanz eine Schranke $K \in \mathbb{R}$ gegeben. Die Frage ist dann, ob es eine Abarbeitung mit einer Distanz $\leq K$ gibt.

In Abschnitt 3.1 geben wir zunächst einen sehr einfachen Beweis dafür an, dass das Förderband-Flow-Shop-Problem NP-vollständig ist, wenn wir beliebige Distanzfunktionen zulassen. In Abschnitt 3.2 betrachten wir das Problem für den Fall, dass die Anzahl der Maschinen oder die Anzahl der Jobs durch eine Konstante beschränkt ist. In diesem Fall lässt sich das Problem in polynomieller Zeit lösen. Allerdings geht dabei die (konstante) Job- oder Maschinenanzahl exponentiell in die Laufzeit des Algorithmus ein. In Abschnitt 3.3 betrachten wir ausnahmsweise das Förderband-Flow-Shop-Problem mit variabler Jobreihenfolge. Hier zeigen wir die NP-Vollständigkeit sogar für den Fall, dass wir uns auf Einheitsdistanzen oder lineare Distanzen beschränken. In Abschnitt 3.4 weisen wir schließlich nach, dass das Förderband-Flow-Shop-Problem bei vorgegebener Jobreihenfolge sogar für Einheitsdistanzen NP-vollständig ist. Leider lässt sich der Beweis nicht auf lineare Distanzen übertragen. Für den Fall linearer Distanzen bei vorgegebener Jobreihenfolge bleibt die Komplexität offen.

3.1 Beliebige Distanzfunktionen

Wir zeigen in diesem Abschnitt, dass das Förderband-Flow-Shop-Problem in der Entscheidungsversion NP-vollständig ist, wenn wir als Eingabe beliebige Distanzfunktionen, die die Dreiecksungleichung erfüllen, zulassen. Obwohl dieses Ergebnis auch aus der NP-Vollständigkeit des Problems für Einheitsdistanzen folgt, die wir in Abschnitt 3.4 nachweisen, geben wir hier einen separaten Beweis an, da dieser sehr viel einfacher ist als der Beweis aus Abschnitt 3.4.

Wir zeigen die NP-Vollständigkeit mit Hilfe einer einfachen polynomiellen Reduktion vom Problem 'GERICHTETER HAMILTON-WEG'. Dieses Problem ist wie folgt definiert.

Problem 3.1 (GERICHTETER HAMILTON-WEG)
Gegeben: *Ein gerichteter Graph $G = (V, A)$ mit einer endlichen Menge $V = \{v_1, \ldots, v_n\}$ von Knoten und einer Menge $A \subseteq V \times V$ von gerichteten Kanten, sowie zwei ausgezeichnete Knoten $s, t \in V$.*
Frage: *Gibt es einen einfachen Weg von von s nach t, der jeden Knoten genau einmal durchläuft, d.h., gibt es eine Anordnung $v_{i_1}, \ldots, v_{i_n}$ aller n Knoten, so dass $v_{i_1} = s$, $v_{i_n} = t$, $v_{i_l} \neq v_{i_r}$ für $l \neq r$, und $(v_{i_l}, v_{i_{l+1}}) \in A$ für $l = 1, \ldots, n-1$?*

Das Problem 'GERICHTETER HAMILTON-WEG' ist NP-vollständig, siehe z.B. [11].

Satz 3.1 *Das Förderband-Flow-Shop-Problem (in der Entscheidungsversion) ist NP-vollständig.*

Beweis Es ist offensichtlich, dass das Förderband-Flow-Shop-Problem (in der Entscheidungsversion) in der Klasse NP liegt, da man leicht in polynomieller Zeit verifizieren kann, ob eine gegebene Taskfolge eine gültige Abarbeitung mit einer Distanz $\leq K$ darstellt. Wir zeigen nun die NP-Vollständigkeit, indem wir eine polynomielle Transformation vom Problem 'GERICHTETER HAMILTON-WEG' angeben.

Sei $(G = (V, A), s, t)$ eine Instanz des Problems 'GERICHTETER HAMILTON-WEG' mit $V = \{v_1, \ldots, v_n\}$. Wir konstruieren eine Instanz $(\mathcal{M}, \mathcal{D})$ des Förderband-Flow-Shop-Problems mit n Jobs und n Maschinen, wobei jeder Job genau eine Task hat. Für $j = 1, \ldots, n$ braucht der Job J_j nur an der Maschine P_{n-j+1} bearbeitet werden. Die Taskmatrix $\mathcal{M}$ sieht also wie folgt aus:

$$\mathcal{M} = \left(\begin{array}{c|ccccccc} & P_1 & P_2 & P_3 & \ldots & P_{n-2} & P_{n-1} & P_n \\ \hline J_1 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ J_2 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ J_3 & 0 & 0 & 0 & \cdots & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ J_{n-2} & 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ J_{n-1} & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ J_n & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \end{array}\right)$$

Wir definieren die Distanzmatrix $\mathcal{D} = (\delta_{p,q})_{0 \leq p,q \leq n}$ durch

$$\delta_{p,q} := \begin{cases} 0, & \text{falls } p = q \\ 1, & \text{falls } p = 0, v_q = s \\ 1, & \text{falls } q = 0, v_p = t \\ 1, & \text{falls } p, q \neq 0, (v_p, v_q) \in A \\ 2, & \text{sonst} \end{cases} .$$

Die Distanzen erfüllen offensichtlich die Dreiecksungleichung. Da keine Task ein Vorgänger einer anderen Task ist, können die Tasks der Taskmatrix nach Lemma 2.3 in einer beliebigen Reihenfolge bearbeitet werden.

Es ist leicht zu sehen, dass es genau dann einen gerichteten Hamiltonweg von s nach t in G gibt, wenn es eine Abarbeitung von $\mathcal{M}$ mit einer Distanz $\leq n+1$ gibt. Da die Transformation offenbar in polynomieller Zeit durchgeführt werden kann, beweist dies den Satz. □

3.2 Konstante Maschinen- oder Jobanzahl

Da das Förderband-Flow-Shop-Problem NP-vollständig ist, kann für dieses Problem im allgemeinen Fall kein polynomieller Algorithmus existieren, sofern P $\neq$ NP gilt. In diesem Abschnitt zeigen wir, dass das Problem für beliebige Distanzmatrizen $\mathcal{D}$ in polynomieller Zeit mit Hilfe dynamischer Programmierung gelöst werden kann, falls wir eine konstante Jobanzahl oder eine konstante Maschinenanzahl betrachten.

Jede Teilabarbeitung $\Lambda = (T_1, \ldots, T_r)$ einer Taskmatrix $\mathcal{M}$ definiert einen Zustand $\mathcal{T}_\Lambda := \mathcal{T}(\mathcal{M}) - \{T_1, \ldots, T_r\}$ und eine Arbeiterposition p_Λ. Die Arbeiterposition p_Λ ist der Maschinenindex der zuletzt ausgeführten Task T_r bzw. 0, falls Λ die leere Folge ist. Wir nennen jedes Paar $(\mathcal{T}, p)$, wobei $\mathcal{T}$ ein Zustand und $p \in \{0, 1, \ldots, m\}$ eine Arbeiterposition ist, eine *Konfiguration*, falls es eine Teilabarbeitung Λ mit $(\mathcal{T}, p) = (\mathcal{T}_\Lambda, p_\Lambda)$ gibt. In diesem Fall schreiben wir $(\mathcal{T}, p) = \kappa(\Lambda)$. Für eine Konfiguration $\kappa = (\mathcal{T}, p)$ bezeichnen wir mit $\mathcal{T}_\kappa := \mathcal{T}$ den Zustand und mit $p_\kappa := p$ die Arbeiterposition der Konfiguration. Die Konfiguration $(\mathcal{T}(\mathcal{M}), 0)$ nennen wir die *Startkonfiguration* und jede der höchstens m Konfigurationen κ mit $\mathcal{T}_\kappa = \emptyset$ nennen wir eine *Endkonfiguration*. Wenn κ eine Konfiguration und $T_{(j,p)}$ eine Task ist, die im Zustand $\mathcal{T}_\kappa$ ausführbar ist, so ist offensichtlich $\kappa' = (\mathcal{T}_\kappa - \{T_{(j,p)}\}, p)$ ebenfalls eine Konfiguration. Wir nennen κ' in diesem Fall eine *Nachfolgekonfiguration von* κ.

Für eine Konfiguration κ definieren wir die *minimale Distanz* $\Delta^{\star\mathcal{D}}_{\min}(\kappa)$ *für das Erreichen der Konfiguration* κ durch

$$\Delta^{\star\mathcal{D}}_{\min}(\kappa) := \min\left\{\Delta^{\star\mathcal{D}}(\Lambda) \mid \Lambda \text{ ist Teilabarbeitung mit } \kappa = \kappa(\Lambda)\right\}.$$

Offensichtlich ist $\Delta^{\star\mathcal{D}}_{\min}(\kappa) = 0$, falls κ die Anfangskonfiguration ist, und für alle anderen Konfigurationen κ gilt

$$\Delta^{\star\mathcal{D}}_{\min}(\kappa) = \min\left\{\Delta^{\star\mathcal{D}}_{\min}(\kappa') + \delta_{p_{\kappa'}, p_\kappa} \mid \kappa \text{ ist Nachfolgekonfiguration von } \kappa'\right\}.$$

Wir können daher ausgehend von der Anfangskonfiguration den Wert $\Delta^{\star\mathcal{D}}_{\min}(\kappa)$ für alle Konfigurationen κ bestimmen. Da für die Distanz $\Delta^{\mathcal{D}}_{\min}(\mathcal{M})$ einer minimalen Abarbeitung der Taskmatrix

$$\Delta^{\mathcal{D}}_{\min}(\mathcal{M}) = \min\{\Delta^{\star\mathcal{D}}_{\min}(\kappa) + \delta_{p_\kappa, 0} \mid \kappa \text{ ist Endkonfiguration}\}$$

gilt, können wir auf diese Weise die Distanz einer minimalen Abarbeitung bestimmen.

Betrachten wir zunächst den Fall, dass die Maschinenanzahl m konstant ist. Sei κ eine beliebige Konfiguration und sei j_0 der kleinste Jobindex, mit $\sigma^{\mathcal{T}_\kappa}(j_0) < \infty$, d.h. J_{j_0} ist der

Job mit kleinstem Index, dessen Tasks im Zustand $\mathcal{T}_\kappa$ noch nicht alle bearbeitet wurden. Falls kein solcher Job existiert, sei $j_0 := n+1$. Da die Jobs J_j mit $j \geq j_0 + m$ in der Konfiguration κ die erste Maschine noch nicht erreicht haben, kann die Konfiguration κ offensichtlich eindeutig durch das $m+2$-Tupel

$$Z(\kappa) := (j_0, \sigma^{\mathcal{T}_\kappa}(j_0), \sigma^{\mathcal{T}_\kappa}(j_0+1), \ldots, \sigma^{\mathcal{T}_\kappa}(j_0+m-1), p_\kappa)$$

beschrieben werden. Dabei sei $\sigma^{\mathcal{T}_\kappa}(j_0+i) := \infty$, falls j_0+i größer als die Anzahl n der Jobs ist. Es ist nun leicht zu sehen, dass bezüglich der lexikographischen Ordnung für eine Konfiguration κ und eine Nachfolgekonfiguration κ' von κ stets $Z(\kappa) < Z(\kappa')$ gilt, da sich für κ' entweder der Wert von j_0 oder genau einer der Werte $\sigma^{\mathcal{T}_\kappa}(j_0+i)$, $i = 0, \ldots, m-1$, erhöht, während alle anderen Werte gleich bleiben. Im Folgenden identifizieren die Konfigurationen κ mit dem Tupel $Z(\kappa)$ und sagen dass κ kleiner als κ' ist, falls $Z(\kappa)$ lexikographisch kleiner als $Z(\kappa')$ ist. Wir sprechen daher der Einfachheit halber immer von Konfigurationen κ, auch wenn wir diese durch die Tupel $Z(\kappa)$ konstanter Größe verwalten. Man betrachte nun den folgenden Algorithmus, der eine Menge U mit Paaren (κ, d) verwaltet. Dabei ist κ eine Konfiguration, die Menge U enthält für jede Konfiguration κ höchstens ein Paar (κ, d) und d gibt den bislang kleinsten gefunden Wert $\Delta^{\star\mathcal{D}}(\Lambda)$ für eine Teilabarbeitung Λ mit $\kappa(\Lambda) = \kappa$ an.

Algorithmus 1 (Minimale Distanz)

```
(1)   U = {(T(M), 0), 0)};
(2)   d_min = ∞;
(3)   while (U ≠ ∅) {
(4)     sei (κ, d) ∈ U das Element mit kleinster Konfiguration κ;
(5)     if ( κ ist Endkonfiguration ) {
(6)       if (d + δ_{p_κ,0} < d_min)
(7)         d_min = d + δ_{p_κ,0};
(8)     }
(9)     else { /* keine Endkonfiguration */
(10)      for (jede Nachfolgekonfiguration κ' von κ) {
(11)        if (existiert (κ', d') ∈ U für irgendein d') {
(12)          if (d + δ_{p_κ,p_κ'} < d') ändere (κ', d') zu (κ', d + δ_{p_κ,p_κ'});
(13)        }
(14)        else füge (κ', d + δ_{p_κ,p_κ'}) in U ein;
(15)      }
(16)    }
(17)    entferne (κ, d) aus U;
(18)  } /* while (U ≠ ∅) */
```

Da die Nachfolgekonfigurationen einer Konfiguration κ größer sind als die Konfiguration κ, wird für jede Konfiguration κ nur einmal ein Paar (κ, d) in Zeile 17 aus der Menge

U entfernt. Durch einfache Induktion folgt ferner, dass für jedes Paar (κ, d) mit kleinster Konfiguration κ, welches der Algorithmus in Zeile 4 aus U wählt, $d = \Delta^{\star\mathcal{D}}_{\min}(\kappa)$ gilt. Daraus folgt, dass $d_{\min}$ bei Beendigung des Algorithmus korrekt den Wert $\Delta^{\mathcal{D}}_{\min}(\mathcal{M})$ angibt. Da jede Konfiguration κ nur einmal aus U entfernt wird, ist die Anzahl der Durchläufe durch die while-Schleife offensichtlich so groß wie die Anzahl der möglichen Konfigurationen. Nun ist die Anzahl der möglichen $m+2$-Tupel $Z(\kappa)$ und somit die Anzahl der möglichen Konfigurationen beschränkt durch

$$(n+1)(m+1)^{m+1},$$

was im Falle einer konstanten Maschinenanzahl m linear in der Eingabegröße ist. Sei $|U|_{\max}$ die maximale Anzahl von Elementen, die zu einem Zeitpunkt in U enthalten sind. Da bei konstanter Maschinenanzahl alle Nachfolgekonfigurationen κ' (bzw. deren Darstellung durch das Tupel $Z(\kappa')$) offensichtlich in konstanter Zeit berechnet werden können, ist klar, dass der Algorithmus mit einer Laufzeit von $O(n \cdot |U|_{\max})$ implementiert werden kann. Man betrachte nun die Menge U zu einem beliebigen Zeitpunkt des Algorithmus. Sei (κ, d) das Element mit minimalem κ in U und sei j_0 der Wert der ersten Komponente von $Z(\kappa)$, d.h. der kleinste Index eines Jobs, für den noch eine Task in κ zu bearbeiten ist. Dann ist für alle Paare (κ', d') aus U die Konfiguration κ' eine Nachfolgekonfiguration einer Konfiguration κ'', für die der Index des kleinsten noch nicht vollständig abgearbeiteten Jobs kleiner oder gleich j_0 ist. Nun sei $x \geq j_0 + m$ der kleinste Jobindex, so dass J_x eine Task besitzt. Falls kein solches x existiert, sei $x = n+1$. Dann sind für alle (κ', d') in U die ersten Komponenten von $Z(\kappa')$ offensichtlich $\leq x$. Da die ersten Komponenten ebenfalls alle $\geq j_0$ sind und da es keine Konfigurationen κ' gibt, so dass die erste Komponente von $Z(\kappa')$ aus der Menge $\{j_0 + m, j_0 + m + 1, \ldots, x - 1\}$ ist, folgt, dass es in U höchstens $m+1$ verschiedene Werte für die ersten Komponenten der Tupel $Z(\kappa)$ gibt. Daraus folgt aber unmittelbar, dass die Anzahl der Elemente in U durch eine Konstante beschränkt ist. Also hat der obige Algorithmus lineare Laufzeit.

Für den Fall, dass die Jobanzahl n durch eine Konstante beschränkt ist, kodieren wir die Konfigurationen κ einfach durch das $(n+1)$-Tupel

$$Z(\kappa) := (\sigma^{\mathcal{T}_\kappa}(1), \sigma^{\mathcal{T}_\kappa}(2), \ldots, \sigma^{\mathcal{T}_\kappa}(n), p_\kappa),$$

welches die Konfiguration κ offensichtlich eindeutig beschreibt. Die Anzahl solcher Tupel ist beschränkt durch $(m+1)^{n+1}$, was polynomiell in der Eingabegröße ist, falls n durch eine Konstante beschränkt ist. Es ist außerdem klar, dass bezüglich der lexikographischen Ordnung die Tupel $Z(\kappa')$ für Nachfolgekonfigurationen κ' von κ größer sind als $Z(\kappa)$. Damit ist klar, dass Algorithmus 1 auch in diesem Fall in polynomieller (allerdings nicht linearer) Zeit die Distanz einer minimalen Abarbeitung berechnet. Algorithmus 1 kann leicht so modifiziert werden, dass er auch die minimale Abarbeitung selbst bestimmt. In Kapitel 6, in dem wir einige empirische Auswertungen vornehmen, gehen wir nochmals kurz auf Algorithmus 1 ein. Wir zeigen dort, dass für Einheitsdistanzen und für additive Distanzen häufig nicht alle Nachfolgekonfigurationen einer Konfiguration betrachtet werden müssen. Dadurch lässt sich die Laufzeit des Algorithmus soweit verringern, dass die optimalen Lösungen für nicht allzu große Instanzen noch sehr schnell berechnet werden können.

3.3 Variable Jobreihenfolge

In diesem Abschnitt betrachten wir ausnahmsweise das Förderband-Flow-Shop-Problem mit variabler Jobreihenfolge. Bei diesem Problem suchen wir zu einer gegebenen Taskmatrix $\mathcal{M}$ und einer gegebenen Distanzmatrix $\mathcal{D}$ eine Permutation π der Jobindizes und eine minimale Abarbeitung Λ der Taskmatrix $\mathcal{M}^\pi$, so dass $\Delta^{\mathcal{D}}(\Lambda) \leq \Delta^{\mathcal{D}}_{\min}(\mathcal{M}^{\pi'})$ für alle Permutationen π' der Jobindizes gilt. Dabei bezeichnet $\mathcal{M}^\pi$ die Taskmatrix, deren $\pi(i)$-te Zeile gleich der i-ten Zeile von $\mathcal{M}$ ist. Wir zeigen, dass dieses Problem NP-vollständig ist, auch wenn wir uns auf Einheitsdistanzen oder lineare Distanzen beschränken.

Für jede ganze Zahl $r > 2$ sei $\mathcal{H}(r) = (\eta_{j,p})_{1\leq j\leq r, 1\leq p\leq 2r-2}$ die Taskmatrix für r Jobs und $2r-2$ Maschinen, die definiert ist durch

$$\eta_{j,p} := \begin{cases} 1, & \text{falls } p = 1 \text{ und } j \neq r, \\ 1, & \text{falls } p = r-1 \text{ oder } p = 2r-2, \\ 0, & \text{sonst.} \end{cases}$$

Die Taskmatrix $\mathcal{H}(5)$ hat z.B. folgendes Aussehen:

$$\mathcal{H}(5) = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Die ersten $r-1$ Jobs von $\mathcal{H}(r)$ haben jeweils genau drei Tasks, nämlich an den Maschinen P_1, P_{r-1} und P_{2r-2}. Der letzte Job hat genau zwei Tasks, nämlich an den Maschinen P_{r-1} und P_{2r-2}. Man kann die Tasks der Matrix $\mathcal{H}(r)$ offensichtlich abarbeiten, indem man zunächst alle Tasks an der Maschine P_1, dann alle Tasks an der Maschine P_{r-1} und anschließend alle Tasks an der Maschine P_{2r-2} bearbeitet.

Sei $\mathcal{S} = \{T_{(j_1,p)}, T_{(j_2,p)}, \ldots, T_{(j_k,p)}\}$ eine Menge von Tasks an derselben Maschine P_p. Wir sagen, dass die Tasks von $\mathcal{S}$ *ohne Maschinenwechsel* in einer Abarbeitung bearbeitet werden, wenn zwischen der ersten Bearbeitung einer Task aus $\mathcal{S}$ und der letzten Bearbeitung einer Task aus $\mathcal{S}$ nur Tasks an der Maschine P_p bearbeitet werden.

Lemma 3.1 *Sei $r > 2$ eine ganze Zahl und $\mathcal{M} = (\mu_{j,p})_{1\leq j\leq n, 1\leq p\leq m}$ eine Taskmatrix mit $n \geq r$ und $m \geq 2r-2$. Ferner gebe es r Jobindizes $j_1, \ldots, j_r$ und $2r-2$ aufeinander folgende Maschinenindizes $s+1, \ldots, s+2r-2$, $s \geq 0$, von $\mathcal{M}$, so dass die Matrix $(\mu_{j_k, s+p})_{1\leq k\leq r, 1\leq p\leq 2r-2}$ gleich der Matrix $\mathcal{H}(r)$ ist.*

Sei π eine Jobreihenfolge und Λ eine Abarbeitung der Taskmatrix $\mathcal{M}^\pi$, so dass die Tasks der Menge

$$\mathcal{S} := \left\{T_{(\pi(j_k), s+r-1)} \mid k = 1, 2, \ldots, r\right\} \subseteq \mathcal{T}(\mathcal{M}^\pi)$$

ohne Maschinenwechsel in Λ bearbeitet werden.

Dann gibt es ein $t \geq 0$, so dass $I := \{\pi(j_k) \mid 1 \leq k \leq r\}$ die Menge der aufeinander folgenden Jobindizes $\{t+1, t+2, \ldots, t+r\}$ mit $\pi(j_r) = t+r$ ist.

Beweis Sei $j_{\min}$ der minimale und $j_{\max}$ der maximale Index in I. Angenommen, I bildet keine Menge von aufeinander folgenden Indizes. Dann ist $j_{\max} - j_{\min} \geq r$ und es gilt

$$T_{(j_{\min},s+r-1)} \prec T_{(j_{\min},s+2r-2)} \prec T_{(j_{\max},s+r-1)}.$$

Also können nicht alle Tasks von $\mathcal{S}$ ohne Maschinenwechsel bearbeitet werden. Daher muss I doch eine Menge von aufeinander folgenden ganzen Zahlen $\{t+1, t+2, \ldots, t+r\}$ für ein $t \geq 0$ bilden.

Angenommen, $\pi(j_r) \neq t+r$. Dann ist $T_{(t+r,s+1)}$ eine Task und es gilt

$$T_{(t+1,s+r-1)} \prec T_{(t+r,s+1)} \prec T_{(t+r,s+r-1)}.$$

Dann könnten wiederum nicht alle Tasks von $\mathcal{S}$ ohne Maschinenwechsel bearbeitet werden. Also muss doch $\pi(j_r) = t+r$ gelten. □

Wir zeigen nun, dass das Förderband-Flow-Shop-Problem mit variabler Jobreihenfolge NP-vollständig ist, selbst für den Fall, dass wir nur Einheitsdistanzen oder lineare Distanzen betrachten. Der Beweis beruht auf einer polynomiellen Reduktion vom Problem '3-PARTITION', welches wie folgt definiert ist:

Problem 3.2 (3-PARTITION)
Gegeben: *Eine endliche Menge* $C = \{c_1, \ldots, c_{3k}\}$ *mit* $3k$ *Elementen,* $k \geq 1$, *eine ganze Zahl* $B > 0$ *und eine Funktion* $s : C \to \mathbb{N}$ *mit* $\frac{B}{4} < s(c) < \frac{B}{2}$ *für alle* $c \in C$ *und* $\sum_{c \in C} s(c) = kB$.
Frage: *Gibt es eine Partition von* C *in* k *disjunkte Teilmengen* $C_1, \ldots, C_k$, *so dass*

$$\sum_{c \in C_i} s(c) = B$$

für alle $1 \leq i \leq k$*?*

Die Bedingungen $\frac{B}{4} < s(c) < \frac{B}{2}$ implizieren, dass jedes C_i genau drei Elemente haben muss. Das Problem ist streng NP-vollständig, d.h. das Problem bleibt NP-vollständig, wenn man sich auf Instanzen beschränkt, bei denen die Größe der vorkommenden Zahlen $s(c_i)$ und B polynomiell in der Eingabegröße ist, siehe z.B. [11].

Satz 3.2 *Das Förderband-Flow-Shop-Problem mit variabler Jobreihenfolge (in der Entscheidungsversion) ist NP-vollständig, auch wenn nur Einheitsdistanzen oder lineare Distanzen betrachtet werden.*

Beweis Das Problem gehört offensichtlich zu NP, da man leicht in polynomieller Zeit verifizieren kann, ob eine gegebene Taskfolge eine gültige Abarbeitung mit einer Distanz $\leq K$ für eine gegebene Jobreihenfolge darstellt.

Sei $I = (C, B, s)$ mit $C = \{c_1, \ldots, c_{3k}\}$ eine beliebige Instanz von 3-PARTITION. Wegen der strengen NP-Vollständigkeit von 3-PARTITION können wir davon ausgehen, dass B und alle Zahlen $s(c_i)$ polynomiell durch die Größe der Instanz beschränkt sind.

Weiter nehmen wir ohne Einschränkung an, dass alle Zahlen $s(c_i)$ größer als $3k$ sind. Dieses lässt sich leicht erreichen, indem wir B und alle Werte $s(c_i)$ mit $3k+1$ multiplizieren. Das modifizierte Problem ist wieder eine gültige Instanz von 3-PARTITION, seine Größe ist polynomiell in der Größe des ursprünglichen Problems und es hat offensichtlich genau dann eine Lösung, wenn das ursprüngliche Problem eine Lösung hat.

Wir konstruieren nun eine Instanz $I'(I)$ des Förderband-Flow-Shop-Problems mit variabler Jobreihenfolge für Einheitsdistanzen bzw. lineare Distanzen, welche genau dann eine Lösung hat, wenn die Instanz I von 3-PARTITION eine Lösung besitzt.

Für $i = 1, \ldots, 3k$ sei $\mathcal{H}_{c_i} := \mathcal{H}(s(c_i))$ die Matrix aus Lemma 3.1 und $\mathcal{F}_{c_i}$ die folgende $s(c_i) \times (B-1)$-Matrix:

$$\mathcal{F}_{c_i} := \overbrace{\left.\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}\right\} s(c_i)}^{B-1}$$

Alle Elemente von $\mathcal{F}_{c_i}$ in der letzten Spalte und alle Elemente in der ersten Spalte mit Ausnahme der letzten Zeile sind 1. Alle übrigen Einträge sind 0.

Die Taskmatrix $\mathcal{M}_I$ der Instanz $I'(I)$ besitzt nun $n := kB = \sum_{i=1}^{3k} s(c_i)$ Jobs und

$$m := B - 1 + n + \left(\sum_{i=1}^{3k} (2s(c_i) - 2 + n)\right) + 1$$

Maschinen und hat folgendes Aussehen:

$\overbrace{\quad}^{B-1}$	$\overbrace{\quad}^{n}$	$\overbrace{\quad}^{2s(c_1)-2}$	$\overbrace{\quad}^{n}$	$\overbrace{\quad}^{2s(c_2)-2}$	$\overbrace{\quad}^{n}$		$\overbrace{\quad}^{2s(c_{3k-1})-2}$	$\overbrace{\quad}^{n}$	$\overbrace{\quad}^{2s(c_{3k})-2}$	$\overbrace{\quad}^{n}$	
$\mathcal{F}_{c_1}$	$0\cdots0$	$\mathcal{H}_{c_1}$	$0\cdots0$	$0\cdots0$	$0\cdots0$	$\cdots$	$0\cdots0$	$0\cdots0$	$0\cdots0$	$0\cdots0$	1
$\mathcal{F}_{c_2}$	$0\cdots0$	$0\cdots0$	$0\cdots0$	$\mathcal{H}_{c_2}$	$0\cdots0$	$\cdots$	$0\cdots0$	$0\cdots0$	$0\cdots0$	$0\cdots0$	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathcal{F}_{c_{3k-1}}$	$0\cdots0$	$0\cdots0$	$0\cdots0$	$0\cdots0$	$0\cdots0$	$\cdots$	$\mathcal{H}_{c_{3k-1}}$	$0\cdots0$	$0\cdots0$	$0\cdots0$	1
$\mathcal{F}_{c_{3k}}$	$0\cdots0$	$0\cdots0$	$0\cdots0$	$0\cdots0$	$0\cdots0$	$\cdots$	$0\cdots0$	$0\cdots0$	$\mathcal{H}_{c_{3k}}$	$0\cdots0$	1

Da die Größe von $\mathcal{M}_I$ polynomiell in der Größe von I ist, gibt es offensichtlich einen polynomiellen Algorithmus, der aus einer gegebenen Instanz I von 3-PARTITION die Matrix $\mathcal{M}_I$ konstruiert.

Sei Y die lineare Distanz von der Maschine P_{B-1} zur letzten Maschine von $\mathcal{M}_I$, d.h.

$$Y = m - (B-1) = n + \left(\sum_{i=1}^{3k} (2s(c_i) - 2 + n)\right) + 1.$$

Wir beweisen den Satz nun, indem wir zeigen, dass die Instanz I von 3-PARTITION eine Lösung hat,

1. genau dann, wenn es für $\mathcal{M}_I$ eine Jobreihenfolge π und eine Abarbeitung Λ von $\mathcal{M}_I^\pi$ gibt mit linearer Distanz

$$\Delta^{\text{lin}}(\Lambda) \leq K_I^{\text{lin}} := 2k(B-2) + 2Y, \tag{3.1}$$

bzw.

2. genau dann, wenn es für $\mathcal{M}_I$ eine Jobreihenfolge π und eine Abarbeitung Λ von $\mathcal{M}_I^\pi$ gibt mit Einheitsdistanz

$$\Delta^{\text{E}}(\Lambda) \leq K_I^{\text{E}} := 11k + 1. \tag{3.2}$$

Nehmen wir zuerst an, dass die Instanz I von 3-PARTITION die Lösung $C_1, \ldots, C_k$ mit $C_j = \{c_{i_{j,1}}, c_{i_{j,2}}, c_{i_{j,3}}\}$, $j = 1, \ldots, k$, besitzt. Sei $\mathcal{M}_I^{c_i}$ die Teilmatrix von $\mathcal{M}_I$, die aus den $s(c_i)$ Jobs besteht, die zu c_i gehören, d.h. $\mathcal{M}_I^{c_1}$ besteht aus den ersten $s(c_1)$ Jobs von $\mathcal{M}_I$, $\mathcal{M}_I^{c_2}$ besteht aus den nächsten $s(c_2)$ Jobs von $\mathcal{M}_I$, usw.. Sei π die Permutation der Jobindizes, so dass $\mathcal{M}_I^\pi$ die Matrix ist, die wir erhalten, wenn wir die $\mathcal{M}_I^{c_i}$ in der Reihenfolge

$$\mathcal{M}_I^{c_{i_{1,1}}}, \mathcal{M}_I^{c_{i_{1,2}}}, \mathcal{M}_I^{c_{i_{1,3}}}, \ \mathcal{M}_I^{c_{i_{2,1}}}, \mathcal{M}_I^{c_{i_{2,2}}}, \mathcal{M}_I^{c_{i_{2,3}}}, \ \ldots, \ \mathcal{M}_I^{c_{i_{k,1}}}, \mathcal{M}_I^{c_{i_{k,2}}}, \mathcal{M}_I^{c_{i_{k,3}}}$$

untereinander anordnen. Wir behaupten, dass $\mathcal{M}_I^\pi$ eine Abarbeitung Λ mit linearer Distanz K_I^{lin} und Einheitsdistanz K_I^{E} besitzt.

Da $C_1, \ldots, C_k$ eine Lösung der Instanz I von 3-PARTITION ist, brauchen die Jobs von $\mathcal{M}_I^\pi$ mit den Indizes iB, $1 \leq i \leq k$ nicht an der ersten Maschine bearbeitet werden, d.h. es gibt keine Tasks $T_{(iB,1)}$, $i = 1, \ldots, k$. Seien $\Lambda_{i,1}$ und $\Lambda_{i,B-1}$ die Folgen aller Tasks der Jobs $J_{(i-1)B+1}, J_{(i-1)B+2}, \ldots, J_{iB}$, an den Maschinen P_1 bzw. P_{B-1}, jeweils aufsteigend nach Jobindizes geordnet. Da keine Task $T_{(iB,1)}$, $i = 1, \ldots, k$, existiert, ist für $i = 1, \ldots, k$ keine der Tasks aus $\Lambda_{i,B-1}$ ein Vorgänger einer der Tasks aus $\Lambda_{i,1}$. Da nach der Maschine P_{B-1} n Maschinen folgen, an denen kein Job bearbeitet werden muss, gibt es überhaupt keine Task mit Maschinenindex $\geq B$, die ein Vorgänger irgendeiner Task aus $\Lambda_{i,1}$ oder $\Lambda_{i,B-1}$ für irgendein $i \in \{1, \ldots, k\}$ ist. Daher stellt die Taskfolge

$$\Lambda' := \Lambda_{1,1}, \Lambda_{1,B-1}, \Lambda_{2,1}, \Lambda_{2,B-1}, \ldots, \Lambda_{k,1}, \Lambda_{k,B-1}$$

eine Teilabarbeitung dar, für die

$$\Delta^{\star\text{lin}}(\Lambda') = (2k-1)(B-2) \tag{3.3}$$

und

$$\Delta^{\star\text{E}}(\Lambda') = 2k. \tag{3.4}$$

gilt. Da zwischen den 'Bereichen' zweier benachbarter Matrizen $\mathcal{H}_{c_i}$ jeweils n Maschinen sind, an denen kein Job bearbeitet werden muss und da die Tasks der Matrix $\mathcal{H}_{c_i}$ bearbeitet werden können, indem diejenigen drei Maschinen der Matrix, die Tasks aufweisen, jeweils einmal in aufsteigender Reihenfolge ihrer Maschinenindizes besucht werden, kann der Arbeiter offensichtlich alle Tasks an den Maschinen P_p, $p \geq B$ bearbeiten, indem er jede dieser Maschinen, welche Tasks besitzt, genau einmal in aufsteigender Reihenfolge der Indizes besucht. Einschließlich der Enddistanz ergeben diese Bewegungen eine lineare Distanz von

$$2Y + B - 2$$

und eine Einheitsdistanz von

$$9k + 1.$$

Daher lässt sich die Teilabarbeitung Λ' zu einer Abarbeitung Λ fortsetzen, für die

$$\Delta^{\mathrm{lin}}(\Lambda) = \Delta^{\star\mathrm{lin}}(\Lambda') + 2Y + B - 2 = (2k-1)(B-2) + 2Y + B - 2 = K_I^{\mathrm{lin}}$$

und

$$\Delta^{\mathrm{E}}(\Lambda) = \Delta^{\star\mathrm{E}}(\Lambda') + 9k + 1 = 2k + 9k + 1 = K_I^{\mathrm{E}}$$

wie gewünscht gilt.

Nehmen wir nun andererseits an, dass es für $\mathcal{M}_I$ eine Jobreihenfolge π und eine Abarbeitung Λ für $\mathcal{M}_I^\pi$ gibt mit $\Delta^{\mathrm{lin}}(\Lambda) \leq K_I^{\mathrm{lin}}$ bzw. $\Delta^{\mathrm{E}}(\Lambda) \leq K_I^{\mathrm{E}}$.

Wir bezeichnen die Jobs, die an Maschine P_1 bearbeitet werden müssen, als Jobs vom Typ 1 und alle anderen als Jobs vom Typ 2. Da wir ohne Einschränkung $B > 3k$ angenommen hatten, gibt es

$$kB - 3k > kB - B = (k-1)B > (k-1)(B-1)$$

Jobs vom Typ 1. Da alle diese Jobs auch an der Maschine P_{B-1} bearbeitet werden müssen, kann der Arbeiter offensichtlich höchstens $B-1$ dieser Jobs vom Typ 1 an Maschine P_1 bzw. an Maschine P_{B-1} bearbeiten, ohne zur jeweils anderen Maschine zu wechseln. Daher muss sich der Arbeiter mindestens k-mal zu jeder dieser beiden Maschinen bewegen, um alle Tasks an diesen beiden Maschinen zu bearbeiten. Die lineare Distanz, die hierfür zwischen den Maschinen P_1 und P_{B-1} zurückgelegt werden muss, ist mindestens

$$K'^{\mathrm{lin}} = (2k-1)(B-2) \tag{3.5}$$

und die Einheitsdistanz für die Bearbeitung dieser Tasks (einschließlich der Startdistanz) ist mindestens

$$K'^{\mathrm{E}} = 2k. \tag{3.6}$$

Da jeder Job an der letzten Maschine bearbeitet werden muss, endet jede Abarbeitung unabhängig von der Jobreihenfolge mit der Bearbeitung einer Task an der letzten Maschine. Die Enddistanz ist somit stets $Y+B-2$ bei linearen Distanzen und 0 bei Einheitsdistanzen.

Für die Bearbeitung aller Tasks an den Maschinen $P_B, \ldots, P_m$ verbleibt daher im Fall linearer Distanzen für Bewegungen zwischen den Maschinen P_{B-1} und P_m nur noch die Distanz

$$K_I^{\text{lin}} - K'^{\text{lin}} - (Y + B - 2) = Y \tag{3.7}$$

und im Fall von Einheitsdistanzen nur noch die Distanz

$$K_I^{\text{E}} - K'^{\text{E}} = 9k + 1. \tag{3.8}$$

Im Falle linearer Distanzen muss der Arbeiter daher alle Tasks an den Maschinen $P_B, \ldots, P_m$ bearbeiten, indem er sich ohne 'Rückwärtsbewegungen' einmal von der Maschine P_{B-1} zur Maschine P_m bewegt und dabei jeweils sämtliche Tasks an jeder dieser Maschinen ohne Maschinenwechsel bearbeitet. Da genau $9k + 1$ dieser Maschinen Tasks aufweisen, müssen die Tasks an jeder dieser Maschinen auch im Fall von Einheitsdistanzen ohne Maschinenwechsel bearbeitet werden. Nach Lemma 3.1 folgt in jedem Fall, dass die Jobs die zu einem c_i gehören, für $i = 1, \ldots, 3k$ einen zusammenhängenden Block in $\mathcal{M}_I^\pi$ bilden, wobei der Job vom Typ 2 der letzte Job des Blocks ist. Insbesondere gibt es keine zwei aufeinander folgende Jobs vom Typ 2.

Da die in (3.7) und (3.8) angegebenen Werte offenbar untere Schranken sind, folgt, dass für die Bearbeitung der Tasks an den Maschinen P_1 und P_{B-1} höchstens und daher genau die in (3.5) bzw. (3.6) angegebenen Distanzen zur Verfügung stehen. Insbesondere bedeutet dies, dass der Arbeiter genau k-mal zur Maschine P_{B-1} geht.

Da keine zwei aufeinander folgende Jobs vom Typ 2 existieren, können offensichtlich höchstens B Jobs ohne Maschinenwechsel an der Maschine P_{B-1} bearbeitet werden. Dies kann ferner nur dann geschehen, wenn der letzte dieser B Jobs ein Job vom Typ 2 ist. Da der Arbeiter nur k-mal zur Maschine P_{B-1} geht, muss er also jedesmal genau B Jobs dort bearbeiten. Daher müssen alle Jobs J_{iB}, $i = 1, \ldots, k$, vom Typ 2 sein. Da alle Jobs, die zu einem c_i gehören, einen zusammenhängenden Block bilden, wobei der Job vom Typ 2 der letzte des Blocks ist, folgt, dass die Instanz I von 3-PARTITION eine Lösung besitzt. □

3.4 Einheitsdistanzen

Wir wollen nun die NP-Vollständigkeit des Förderband-Flow-Shop-Problems für Einheitsdistanzen bei vorgegebener Jobreihenfolge beweisen. Dazu zeigen wir zunächst, dass wir die Abarbeitungen durch gewisse *Überdeckungen* der Taskmatrix mit so genannten *Taskblöcken* beschreiben können, so dass die Einheitsdistanz der Abarbeitung der Anzahl der Blöcke in der Überdeckung entspricht. Wir definieren die Begriffe 'Taskblock' und 'Überdeckung' etwas allgemeiner für beliebige Teilmengen der Tasks einer Taskmatrix.

Sei $\mathcal{M} = (\mu_{j,p})_{1 \le j \le n, 1 \le p \le m}$ eine Taskmatrix und $X \subseteq \mathcal{T}(\mathcal{M})$ eine Teilmenge der Tasks der Taskmatrix. Wir nennen eine nicht-leere Folge

$$T_{(j_1,p)}, T_{(j_2,p)}, \ldots, T_{(j_k,p)}, \qquad (1 \le p \le m, 1 \le j_1 < \ldots < j_k \le n)$$

von Tasks aus X mit demselben Maschinenindex p einen *Taskblock* oder kurz *Block* von X, wenn die Folge alle Tasks $T_{(l,p)} \in X$ mit $j_1 \leq l \leq j_k$ enthält. Wir bezeichnen diesen Taskblock mit $\Phi_p^{[j_1,j_k]}(X)$. Wenn klar ist, auf welche Menge X wir uns beziehen, so schreiben wir auch vereinfachend $\Phi_p^{[j_1,j_k]}$. Falls $X = \mathcal{T}(\mathcal{M})$, so benutzen wir auch die Schreibweise $\Phi_p^{[j_1,j_k]}(\mathcal{M})$ und sprechen von einem *Taskblock* oder *Block von* $\mathcal{M}$.

Die *Länge* eines Taskblocks $\Phi_p^{[j,k]}$ ist definiert als $k - j + 1$. Blöcke der Länge $1, 2, 3, \ldots$ bezeichnen wir als *Einerblöcke*, *Zweierblöcke*, *Dreierblöcke*, usw..

Man beachte, dass ein Block der Länge $k > 2$ im allgemeinen weniger als k Tasks enthält. Wenn wir von einem Block $\Phi_p^{[j,k]}(X)$ sprechen, setzen wir aber immer voraus, dass $T_{(j,p)}$ und $T_{(k,p)}$ Tasks aus X sind.

Offensichtlich kann man eine Abarbeitung $T_1, \ldots, T_r$ der Taskmatrix in Taskblöcke von $\mathcal{M}$ unterteilen. Man betrachte zum Beispiel die Taskmatrix aus Abbildung 3.1 (a). Im Folgenden ist eine Abarbeitung sowie eine Unterteilung der Abarbeitung in 15 Blöcke angegeben:

$$
\begin{array}{l}
\underbrace{T_{(1,1)}, T_{(2,1)}, T_{(3,1)}}_{\Phi_1^{[1,3]}}, \underbrace{T_{(1,4)}}_{\Phi_4^{[1,1]}}, \underbrace{T_{(2,2)}}_{\Phi_2^{[2,2]}}, \underbrace{T_{(2,3)}, T_{(3,3)}}_{\Phi_3^{[2,3]}}, \underbrace{T_{(2,5)}}_{\Phi_5^{[2,2]}}, \underbrace{T_{(3,4)}}_{\Phi_4^{[3,3]}}, \underbrace{T_{(4,2)}}_{\Phi_2^{[4,4]}}, \underbrace{T_{(4,3)}}_{\Phi_3^{[4,4]}}, \underbrace{T_{(1,6)}, T_{(4,6)}}_{\Phi_6^{[1,4]}}, \\
\underbrace{T_{(5,1)}, T_{(6,1)}}_{\Phi_1^{[5,6]}}, \underbrace{T_{(5,2)}, T_{(6,2)}, T_{(7,2)}}_{\Phi_2^{[5,7]}}, \underbrace{T_{(5,4)}, T_{(6,4)}}_{\Phi_4^{[5,6]}}, \underbrace{T_{(5,5)}, T_{(6,5)}}_{\Phi_5^{[5,6]}}, \underbrace{T_{(7,3)}}_{\Phi_3^{[7,7]}}, \underbrace{T_{(5,6)}, T_{(7,6)}}_{\Phi_6^{[5,7]}}
\end{array}
\tag{3.9}
$$

In Abbildung 3.1(a) sind die Taskblöcke graphisch dargestellt. Die Abarbeitung könnte natürlich auch in mehr als 15 Blöcke unterteilt werden, da jede einzelne Task einen Block darstellt. Offensichtlich entspricht die Einheitsdistanz der Abarbeitung der minimalen Anzahl von Blöcken, in die die Abarbeitung unterteilt werden kann. Wir sagen, dass eine Abarbeitung *einen Taskblock B enthält*, wenn die Abarbeitung den Block B als zusammenhängende Folge beinhaltet.

Obwohl ein Taskblock B als eine Folge von Tasks definiert ist, werden wir ihn im Folgenden des Öfteren mit der Menge der Tasks, die in B enthalten sind, identifizieren und daher Ausdrücke wie $T \in B$, $B \cap B' = \emptyset$, $B - B'$ oder $|B|$ verwenden. Dies sollte zu keinerlei Missverständnissen führen.

Wir nennen einen Taskblock $\Phi_p^{[j,k]}$ einer Menge X einen *gültigen Block von* X, wenn es keine Task $T \in X - \Phi_p^{[j,k]}$ gibt, so dass $T_{(j,p)} \prec T \prec T_{(k,p)}$. Im Fall $X = \mathcal{T}(\mathcal{M})$ sprechen wir auch von einem *gültigen Block von* $\mathcal{M}$. Abbildung 3.1(b) zeigt einen gültigen Block einer Taskmatrix. Wir nennen einen Taskblock einer Taskmatrix $\mathcal{M}$ *ausführbar*, wenn es eine Abarbeitung von $\mathcal{M}$ gibt, die den Block enthält.

Lemma 3.2 *Ein Block $B = \Phi_p^{[j,j']}(\mathcal{M})$ ist genau dann ausführbar, wenn er ein gültiger Block von $\mathcal{M}$ ist, d.h. wenn es keine Task $T \in \mathcal{T}(\mathcal{M}) - B$ gibt, so dass $T_{(j,p)} \prec T \prec T_{(j',p)}$.*

Beweis '$\Rightarrow$' Falls eine Task $T \in \mathcal{T}(\mathcal{M}) - B$ mit $T_{(j,p)} \prec T \prec T_{(j',p)}$ existiert, so muss diese nach Lemma 2.3 in jeder Abarbeitung nach der Task $T_{(j,p)}$ und vor der Task $T_{(j',p)}$ bearbeitet werden. Dann kann aber $B = \Phi_p^{[j,j']}(\mathcal{M})$ kein ausführbarer Block sein.

	P_1	P_2	P_3	P_4	P_5	P_6
J_1	1	0	0	1	0	1
J_2	1	1	1	0	1	0
J_3	1	0	1	1	0	0
J_4	0	1	1	0	0	1
J_5	1	1	0	1	1	1
J_6	1	1	0	1	1	0
J_7	0	1	1	0	0	1

(a)

	P_1	P_2	P_3	P_4	P_5	P_6	P_7
J_1	0	0	0	0	1	1	1
J_2	1	1	1	1	0	0	1
J_3	1	1	1	0	0	1	1
J_4	0	1	0	1	1	1	1
J_5	1	0	0	1	1	1	1
J_6	1	1	1	0	0	1	1

(b)

Abbildung 3.1: (a) Graphische Darstellung der Unterteilung der Abarbeitung aus (3.9) in Taskblöcke. (b) Taskblock $\Phi_4^{[2,5]}(\mathcal{M})$ ist gültig, da die gestrichelt eingezeichneten Bereiche keine Tasks enthalten.

'$\Leftarrow$' Sei $B = \Phi_p^{[j,j']}(\mathcal{M}) = (T_1, T_2, \ldots, T_k)$ mit $T_1 = T_{(j,p)}$ und $T_k = T_{(j',p)}$ ein gültiger Block. Sei $T'_1, \ldots, T'_q$ eine beliebige gültige, d.h. die Vorgängerrelation einhaltende Anordnung aller Vorgängertasks von $T_{(j',p)}$, die nicht in B enthalten sind. Sei $T''_1, \ldots, T''_r$ eine beliebige gültige Anordnung aller restlichen Tasks, die nicht in B enthalten sind. Da B ein gültiger Block ist, enthält $T'_1, \ldots, T'_q$ keine Nachfolgertask von $T_{(j,p)}$ und somit keine Nachfolgertask irgendeiner Task aus B. Ferner enthält $T''_1, \ldots, T''_r$ keine Vorgängertask T einer Task aus $T'_1, \ldots, T'_q$ oder aus B, da T dann auch Vorgängertask von $T_{(j',p)}$ und somit in $\{T'_1, \ldots, T'_q\}$ wäre. Daher ist die Taskfolge

$$T'_1, \ldots, T'_q, T_1, \ldots, T_k, T''_1, \ldots, T''_r$$

gültig und somit nach Lemma 2.3 eine Abarbeitung, die den Block $\Phi_p^{[j,j']}(\mathcal{M})$ enthält. □

Aus dem Lemma und der Definition der Vorgängerrelation folgt, dass alle Einer- und Zweierblöcke gültig und damit ausführbar sind.

Wir nennen eine Menge $\mathcal{C}$ von Taskblöcken einer Menge $X \subseteq \mathcal{T}(\mathcal{M})$ eine *Blocküberdeckung von* X oder kurz *Überdeckung von* X, wenn jede Task von X in einem Block von $\mathcal{C}$ enthalten ist. Eine Blocküberdeckung von $\mathcal{T}(\mathcal{M})$ nennen wir auch eine *Blocküberdeckung von* $\mathcal{M}$. Ist Y eine Teilmenge von X, so induziert eine Blocküberdeckung von X eine Blocküberdeckung von Y. Die induzierte Blocküberdeckung von Y erhält man, indem man aus allen Blöcken der Blocküberdeckung von X die Tasks, die nicht in Y sind, streicht, und eventuell entstehende leere Blöcke entfernt.

Wir nennen eine Blocküberdeckung $\mathcal{C}$ der Taskmatrix $\mathcal{M}$ *ausführbar*, wenn die Blöcke von $\mathcal{C}$ so angeordnet werden können, dass die daraus resultierende Folge der Tasks eine Abarbeitung der Taskmatrix ist. Die Blöcke in einer ausführbaren Blocküberdeckung müssen offenbar paarweise disjunkt sein.

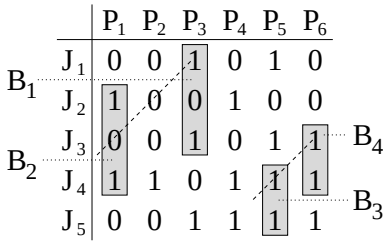


Abbildung 3.2: B_1 und B_2 sowie B_3 und B_4 sind jeweils Paare unverträglicher gültiger Blöcke, da die erste Task des einen Blocks jeweils ein Vorgänger der letzten Task des anderen Blocks ist.

Wenn $T_1, \ldots, T_r$ eine Abarbeitung mit der Einheitsdistanz k ist, so liefert die Unterteilung der Abarbeitung in maximale Blöcke offensichtlich eine ausführbare Blocküberdeckung mit k Blöcken. Ist andererseits $\mathcal{C}$ eine ausführbare Blocküberdeckung mit k Blöcken, so gibt es eine Abarbeitung, deren Einheitsdistanz höchstens k ist.

Eine ausführbare Blocküberdeckung kann offensichtlich nur ausführbare Blöcke enthalten. Diese Bedingung ist jedoch noch nicht hinreichend. Um die ausführbaren Blocküberdeckungen zu charakterisieren, führen wir die Relation '$\prec$' auf Blöcken ein. Es soll $B_1 \prec B_2$ für zwei Blöcke B_1, B_2 gelten, wenn B_1 eine Task enthält, die eine Vorgängertask einer Task von B_2 ist. Man beachte, dass diese Relation - im Gegensatz zu der Relation $\prec$ auf Tasks - im allgemeinen weder transitiv noch azyklisch ist. Wir nennen zwei Blöcke B_1 und B_2 *unverträglich*, wenn sie eine gemeinsame Task enthalten oder wenn sowohl $B_1 \prec B_2$ als auch $B_2 \prec B_1$ gilt. Andernfalls nennen wir die Blöcke *verträglich*. Nach der Definition der Vorgängerrelation für Tasks sind zwei disjunkte Blöcke $\Phi_{p_1}^{[j_1,j_1']}$ und $\Phi_{p_2}^{[j_2,j_2']}$ genau dann unverträglich, wenn sowohl $T_{(j_1,p_1)} \prec T_{(j_2',p_2)}$ als auch $T_{(j_2,p_2)} \prec T_{(j_1',p_1)}$ gilt. Die gültigen Blöcke einer Menge X von Tasks sind genau diejenigen Blöcke B, die mit allen zu B disjunkten Einerblöcken von X verträglich sind. Abbildung 3.2 zeigt Beispiele für unverträgliche, aber gültige Blöcke einer Taskmatrix. Eine Blocküberdeckung, deren Blöcke paarweise verträglich sind, nennen wir eine *gültige Überdeckung*. Da jede Task in einem Block enthalten ist, kann eine gültige Überdeckung offenbar nur gültige Blöcke enthalten. Eine gültige Blocküberdeckung einer Menge X induziert eine gültige Blocküberdeckung für jede Teilmenge von X.

Lemma 3.3 *Sei $\mathcal{X}$ eine nicht-leere Menge von paarweise verträglichen Blöcken einer Taskmatrix $\mathcal{M}$. Dann gibt es einen Block $B_0 \in \mathcal{X}$, so dass kein Block $B \in \mathcal{X} - \{B_0\}$ mit $B \prec B_0$ existiert.*

Beweis Ein Block $\Phi_p^{[j,j']}$ heiße *frei* in $\mathcal{X}$, wenn kein anderer Block aus $\mathcal{X}$ eine Task $T_{(k,q)}$ mit $k \leq j'$ und $q \leq p$ enthält. Da die Blöcke aus $\mathcal{X}$ paarweise disjunkt sind, enthält $\mathcal{X}$ offenbar mindestens einen freien Block, nämlich den Block $\Phi_p^{[j,j']}$ mit minimalem p, für den j (oder j') minimal ist.

Sei $B_0 = \Phi_{p_0}^{[j_0,j_0']}$ der freie Block, für den j_0 minimal ist, siehe auch Abbildung 3.3. Angenommen, es gäbe einen Block $B = \Phi_p^{[j,j']}$ mit $B \prec B_0$. Dann sei B ein solcher Block,

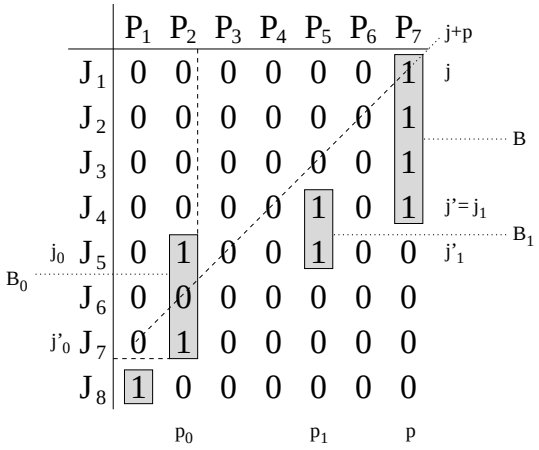


Abbildung 3.3: Zum Beweis von Lemma 3.3. Block $B_0 = \Phi_2^{[5,7]}$ ist frei, da sich im Bereich der Jobs J_1 bis J_7 und der Maschinen P_1, P_2 kein weiterer Block befindet. Der Block B ist kein gültiger Block.

für den p minimal ist, und falls mehrere davon existieren, sei B derjenige von diesen mit minimalem j. Da B_0 frei ist, muss $p > p_0$ gelten. Wäre $j' \geq j_0$, so folgte $B_0 \prec B$ und damit die Unverträglichkeit der Blöcke B_0 und B. Also muss $j' < j_0$ gelten. Nach Wahl von B_0 kann daher B nicht frei sein, d.h. es existiert ein zu B disjunkter Block $B_1 = \Phi_{p_1}^{[j_1,j_1']} \in \mathcal{X}$ mit $p_1 \leq p$ und $j_1 \leq j'$. Dann gilt $B_1 \prec B$ und wegen $j_0 > j'$ ist $B_1 \neq B_0$.

Nach Wahl von B muss $j_1 + p_1 > j + p$ gelten, da andernfalls auch $B_1 \prec B_0$ gilt. Daraus folgt aber $B \prec B_1$ und damit wegen $B_1 \prec B$ die Unverträglichkeit von B und B_1 (es folgt sogar die Ungültigkeit des Blocks B wegen $T_{(j,p)} \prec T_{(j_1,p_1)} \prec T_{(j',p)}$). Das bedeutet einen Widerspruch zur Voraussetzung. Also kann kein Block B mit $B \prec B_0$ existieren. □

Der folgende Satz charakterisiert die ausführbaren Blocküberdeckungen.

Satz 3.3 *Eine Blocküberdeckung $\mathcal{C}$ einer Taskmatrix $\mathcal{M}$ ist genau dann ausführbar, wenn sie gültig ist, d.h., wenn sie kein Paar unverträglicher Taskblöcke enthält.*

Beweis '$\Rightarrow$' Es ist offensichtlich, dass eine ausführbare Blocküberdeckung kein Paar unverträglicher Blöcke enthalten kann.

'$\Leftarrow$' Nach Lemma 3.3 gibt es einen Block B_0 in $\mathcal{C}$, so dass kein Block $B \in \mathcal{C} - \{B_0\}$ mit $B \prec B_0$ existiert. Wiederum nach Lemma 3.3 gibt es dann einen Block B_1 in $\mathcal{C} - \{B_0\}$, so dass kein Block in $B \in \mathcal{C} - \{B_0, B_1\}$ mit $B \prec B_1$ existiert, usw.. Dies ergibt eine Anordnung $B_0, B_1, \ldots, B_k$ aller Blöcke von $\mathcal{C}$ mit der Eigenschaft $B_l \not\prec B_i$ für $l > i$. Hieraus folgt mit Lemma 2.3 unmittelbar, dass die Anordnung $B_0, B_1, \ldots, B_k$ eine Abarbeitung ist. □

Eine gültige Überdeckung heißt *minimale gültige Überdeckung*, wenn sie eine minimale Anzahl von Blöcken enthält. Satz 3.3 zeigt, dass die minimalen gültigen Überdeckungen einer Taskmatrix den Abarbeitungen mit minimaler Einheitsdistanz entsprechen. Es können allerdings verschiedene Abarbeitungen zu derselben Blocküberdeckung führen. Die Beweise

von Satz 3.3 und Lemma 3.3 zeigen überdies, dass man aus einer gültigen Blocküberdeckung auf einfache Weise eine zugehörige Abarbeitung bestimmen kann.

Wir zeigen nun die NP-Vollständigkeit des Problems *'Minimale gültige Blocküberdeckung'*, welches darin besteht zu entscheiden, ob es zu einer Taskmatrix $\mathcal{M}$ und einer ganzen Zahl K eine gültige Überdeckung von $\mathcal{M}$ mit höchstens K Blöcken gibt. Hieraus folgt mit Satz 3.3 leicht die NP-Vollständigkeit des Förderband-Flow-Shop-Problems für Einheitsdistanzen. Zum Beweis der NP-Vollständigkeit des Problems 'Minimale gültige Blocküberdeckung' geben wir eine polynomielle Reduktion vom Problem 3-SAT an, welches als NP-vollständig bekannt ist [11].

Das Problem 3-SAT ist wie folgt definiert. Sei X eine Menge von booleschen Variablen. Für jede Variable $x \in X$ bezeichnen wir x und $\overline{x}$ als Literale von X. Eine *Wahrheitsbelegung für* X ist eine Menge τ von Literalen von X, so dass für jede Variable $x \in X$ genau eines der Literale x oder $\overline{x}$ in τ ist. Wenn x in τ ist, sagen wir, dass x unter der Wahrheitsbelegung τ *wahr* ist, ansonsten ist x *falsch* unter τ. Eine *Klausel* über X ist eine Menge von Literalen von X. Eine Klausel wird von einer Wahrheitsbelegung τ erfüllt, wenn mindestens eines ihrer Literale in τ ist. Das Problem 3-SAT lässt sich nun wie folgt formulieren:

Problem 3.3 (3-SAT)
Gegeben: *Eine Menge* $X = \{x_1, \ldots, x_t\}$ *von booleschen Variablen und eine Menge* $\{C_1, \ldots, C_r\}$ *von Klauseln über* X *mit* $|C_i| = 3$, $i = 1, \ldots, r$.
Frage: *Gibt es eine Wahrheitsbelegung* τ *für* X, *die alle Klauseln erfüllt?*

Wir werden eine polynomielle Transformation angeben, die zu jeder Instanz von 3-SAT eine Taskmatrix $\mathcal{M}$ und eine ganze Zahl K angibt, so dass die Instanz von 3-SAT genau dann eine erfüllende Belegung hat, wenn die Taskmatrix $\mathcal{M}$ eine gültige Blocküberdeckung mit höchstens K Blöcken hat.

Für die Transformation benötigen wir spezielle Mengen von Tasks, die zunächst eingeführt werden sollen. Für $k \geq j$ nennen wir die Menge

$$T_{(j,p)}, T_{(j+1,p)}, \ldots, T_{(k,p)}$$

von Tasks einen *(vertikalen) Weg* der *Länge* $k - j + 1$ und bezeichnen diesen mit $W_p^{[j,k]}$. Eine Menge von Tasks, die aus den beiden vertikalen Wegen $W_p^{[j,k]}$ und $W_{p+1}^{[j-1,k-1]}$ besteht, nennen wir einen *(vertikalen) Doppelweg* der *Länge* $k - j + 1$ und bezeichnen diesen mit $D_p^{[j,k]}$. Abbildung 3.4 zeigt zwei vertikale Doppelwege und jeweils zwei mögliche gültige Blocküberdeckungen der Doppelwege.

Einen Block $\Phi_p^{[j,k]}$ nennen wir *gerade*, falls j gerade ist, und *ungerade* sonst. Wir sprechen von einer *geraden* bzw. *ungeraden Überdeckung*, wenn mit Ausnahme der Einerblöcke alle Blöcke in der Überdeckung gerade bzw. ungerade sind. Für eine beliebige Menge X von Tasks nennen wir die gerade bzw. ungerade Blocküberdeckung, die alle möglichen geraden bzw. ungeraden Zweierblöcke sowie alle dazu disjunkten Einerblöcke enthält, die *gerade* bzw. die *ungerade Zweierüberdeckung* von X. Da zwei gerade bzw. zwei ungerade Zweierblöcke immer verträglich sind, sind die gerade und die ungerade Zweierüberdeckung für jede Menge X gültige Überdeckungen. Wenn die Menge X mindestens einen geraden oder einen

	P_1	P_2	P_3	P_4
J_1	0	0	0	0
J_2	0	0	1	0
J_3	0	1	1	0
J_4	0	1	1	0
J_5	0	1	1	0
J_6	0	1	1	0
J_7	0	1	1	0
J_8	0	1	1	0
J_9	0	1	1	0
J_{10}	0	1	1	0
J_{11}	0	1	0	0
J_{12}	0	0	0	0

	P_1	P_2	P_3	P_4
J_1	0	0	0	0
J_2	0	0	1	0
J_3	0	1	1	0
J_4	0	1	1	0
J_5	0	1	1	0
J_6	0	1	1	0
J_7	0	1	1	0
J_8	0	1	1	0
J_9	0	1	1	0
J_{10}	0	1	1	0
J_{11}	0	1	0	0
J_{12}	0	0	0	0

	P_1	P_2	P_3	P_4
J_1	0	0	0	0
J_2	0	0	1	0
J_3	0	1	1	0
J_4	0	1	1	0
J_5	0	1	1	0
J_6	0	1	1	0
J_7	0	1	1	0
J_8	0	1	1	0
J_9	0	1	1	0
J_{10}	0	1	0	0
J_{11}	0	0	0	0
J_{12}	0	0	0	0

	P_1	P_2	P_3	P_4
J_1	0	0	0	0
J_2	0	0	1	0
J_3	0	1	1	0
J_4	0	1	1	0
J_5	0	1	1	0
J_6	0	1	1	0
J_7	0	1	1	0
J_8	0	1	1	0
J_9	0	1	1	0
J_{10}	0	1	0	0
J_{11}	0	0	0	0
J_{12}	0	0	0	0

Abbildung 3.4: Gerade und ungerade Zweierüberdeckungen der vertikalen Doppelwege $D_2^{[3,11]}$ und $D_2^{[3,10]}$.

ungeraden Zweierblock besitzt, so sind die gerade und die ungerade Zweierüberdeckung von X verschieden. Abbildung 3.4 zeigt die gerade und die ungerade Zweierüberdeckung für einen Doppelweg gerader Länge und einen Doppelweg ungerader Länge.

Lemma 3.4 *Sei $D = D_p^{[j,k]}$ ein vertikaler Doppelweg der Länge $l \geq 2$. Dann hat D genau zwei minimale gültige Überdeckungen, nämlich die gerade und die ungerade Zweierüberdeckung. Beide Überdeckungen haben $l + 1$ Blöcke.*

Beweis Es ist leicht zu sehen, dass ein Doppelweg keinen gültigen Block mit einer Länge > 2 besitzt. Da eine Überdeckung, die nur aus Zweierblöcken besteht, offensichtlich nicht gültig ist, muss jede gültige Überdeckung von D mindestens $l + 1$ Blöcke besitzen. Da die gerade und die ungerade Zweierüberdeckung genau diese Anzahl von Blöcken besitzen, sind es also minimale gültige Überdeckungen. Wegen $l \geq 2$ sind diese beiden Überdeckungen auch verschieden.

Man betrachte nun eine beliebige minimale gültige Überdeckung von D. Da die Überdeckung aus Einer- und Zweierblöcken besteht und genau $l + 1$ Blöcke besitzt, muss sie genau 2 Einerblöcke enthalten. Ist die erste oder die letzte Task eines der beiden vertikalen Wege in einem Zweierblock enthalten, so muss die erste bzw. letzte Task des anderen Weges einen Einerblock bilden, da die Überdeckung sonst unverträgliche Blöcke enthielte. Da genau zwei Einerblöcke vorhanden sind, folgt, dass es sich um die gerade oder die ungerade Zweierüberdeckung handeln muss. □

In der Reduktion des NP-Vollständigkeitsbeweises werden wir für jedes Literal aus jeder Klausel einen solchen vertikalen Doppelweg benutzen, um durch die gerade oder die ungerade Zweierüberdeckung die Wahrheitswerte der Literale in einer erfüllenden Belegung zu definieren. Da gleiche Literale jedoch mit dem gleichen Wert belegt sein müssen und

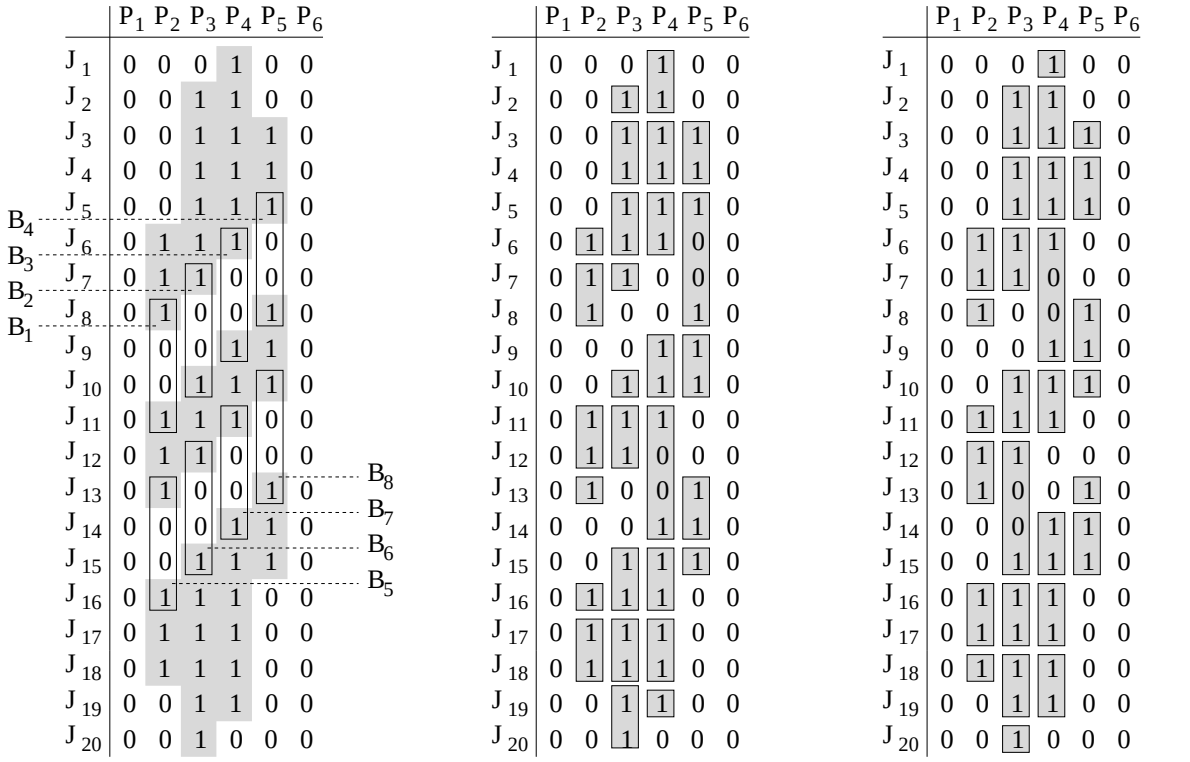


Abbildung 3.5: Unterbrechungselement mit zwei seiner möglichen minimalen gültigen Überdeckungen.

ein negiertes Literal den negierten Wert haben muss, müssen die Wege, die zu denselben Variablen gehören, in geeigneter Weise aneinander gekoppelt sein. Um diese Kopplungen gewährleisten zu können, müssen in die Doppelwege Unterbrechungen eingebaut werden. Man betrachte dazu das *Unterbrechungselement* in Abbildung 3.5, das aus drei zusammenhängenden Komponenten besteht, wobei die Doppelwege am oberen und am unteren Ende beliebig verlängert werden können. Es gilt das folgende Lemma.

Lemma 3.5 *Für eine minimale gültige Überdeckung des Unterbrechungselementes aus Abbildung 3.5 gilt einer der beiden folgenden Fälle:*

1. *Die Überdeckung induziert eine gerade Zweierüberdeckung in allen drei Komponenten des Elementes und sie enthält genau einen der Viererblöcke B_1, B_3 und genau einen der Viererblöcke B_6, B_8 aus Abbildung 3.5.*

2. *Die Überdeckung induziert eine ungerade Zweierüberdeckung in allen drei Komponenten des Elementes und sie enthält genau einen der Viererblöcke B_2, B_4 und genau einen der Viererblöcke B_5, B_7 aus Abbildung 3.5.*

Eine minimale Überdeckung des Unterbrechungselementes aus Abbildung 3.5 hat 28 Blöcke, also 8 Blöcke mehr als die minimale Überdeckung des entsprechenden Doppelweges der

Länge 19.

Beweis Jede der drei Komponenten besteht aus einem Doppelweg, an den auf beiden Seiten zwei einfache Wege der Länge 3 angefügt sind. Mit Hilfe von Lemma 3.4 ist leicht zu verifizieren, dass jede der drei Komponenten genau die geraden und die ungeraden Zweierüberdeckungen als minimale gültige Überdeckungen besitzt. Die einzigen gültigen Blöcke des Unterbrechungselementes, die Tasks aus unterschiedlichen Komponenten enthalten, sind die eingezeichneten Viererblöcke $B_1, \ldots, B_8$. Da die Blöcke $B_1, \ldots, B_4$ und die Blöcke $B_5, \ldots, B_8$ jeweils paarweise unverträglich sind, kann nur je einer der Blöcke $B_1, \ldots, B_4$ und der Blöcke $B_5, \ldots, B_8$ in einer gültigen Überdeckung enthalten sein. Wenn wir in allen drei Komponenten die gerade Zweierüberdeckung wählen, so können wir durch einen der Blöcke B_1, B_3 und durch einen der Blöcke B_6, B_8 jeweils zwei Einerblöcke zu einem Block zusammenfassen. Analog können wir in allen drei Komponenten die ungerade Zweierüberdeckung wählen und jeweils zwei Einerblöcke durch einen der Blöcke B_2, B_4 sowie einen der Blöcke B_5, B_7 zusammenfassen. Die resultierenden Überdeckungen müssen daher minimale gültige Überdeckungen sein. Dies bedeutet aber auch, dass jede minimale gültige Überdeckung in jeder Komponente eine minimale gültige Überdeckung, d.h. eine gerade oder ungerade Zweierüberdeckung induzieren muss, wobei je zwei Einerblöcke aus der induzierten Überdeckung der oberen und mittleren bzw. der mittleren und unteren Komponente einen gemeinsamen Viererblock bilden. Dieses ist nur möglich wenn die induzierten Zweierüberdeckungen entweder alle gerade oder alle ungerade sind. Dass eine minimale gültige Überdeckung 28 Blöcke enthält, kann leicht nachgezählt werden. □

Wegen Lemma 3.5 können wir in einen Doppelweg beliebig viele Unterbrechungselemente einbauen, da sich die gerade oder ungerade Überdeckung in einer minimalen gültigen Überdeckung über die Unterbrechungen hinweg fortsetzt. Wir wollen nun sehen, wie wir diese Unterbrechungen für die Kopplung der Literale verwenden können. Wir betrachten dazu das in Abbildung 3.6 dargestellte *Kopplungselement.*

Lemma 3.6 *Die in Abbildung 3.6 dargestellte gerade und ungerade Überdeckung sind die einzigen minimalen gültigen Überdeckungen des Kopplungselementes und enthalten jeweils 28 Blöcke.*

Beweis Es ist leicht zu verifizieren, dass die dargestellten Überdeckungen gültig sind. Ferner sind für eine gültige Überdeckung der Tasks an den Maschinen P_2 bzw. P_9 jeweils mindestens 3 Blöcke erforderlich. Da die gerade und die ungerade Zweierüberdeckung minimale gültige Überdeckungen des Doppelweges sind, folgt die Minimalität der beiden dargestellten gültigen Überdeckungen. Hieraus folgt wiederum, dass jede minimale gültige Überdeckung sowohl den Doppelweg als auch die Tasks an den Maschinen P_2 bzw. P_9 minimal gültig überdecken muss, d.h. die Überdeckung des Doppelwegs ist die gerade oder ungerade Zweierüberdeckung und die Tasks an den Maschinen P_2 und P_9 sind jeweils mit drei Blöcken überdeckt.

Die Viererblöcke B_1, B_2, B_3, B_4 (siehe Abbildung 3.6) sind die einzigen gültigen Blöcke mit einer Länge > 2. (Die Blöcke $\Phi_2^{[6,8]}$ und $\Phi_9^{[13,15]}$ sind wegen der Tasks $T_{(6,3)}$ bzw. $T_{(15,8)}$

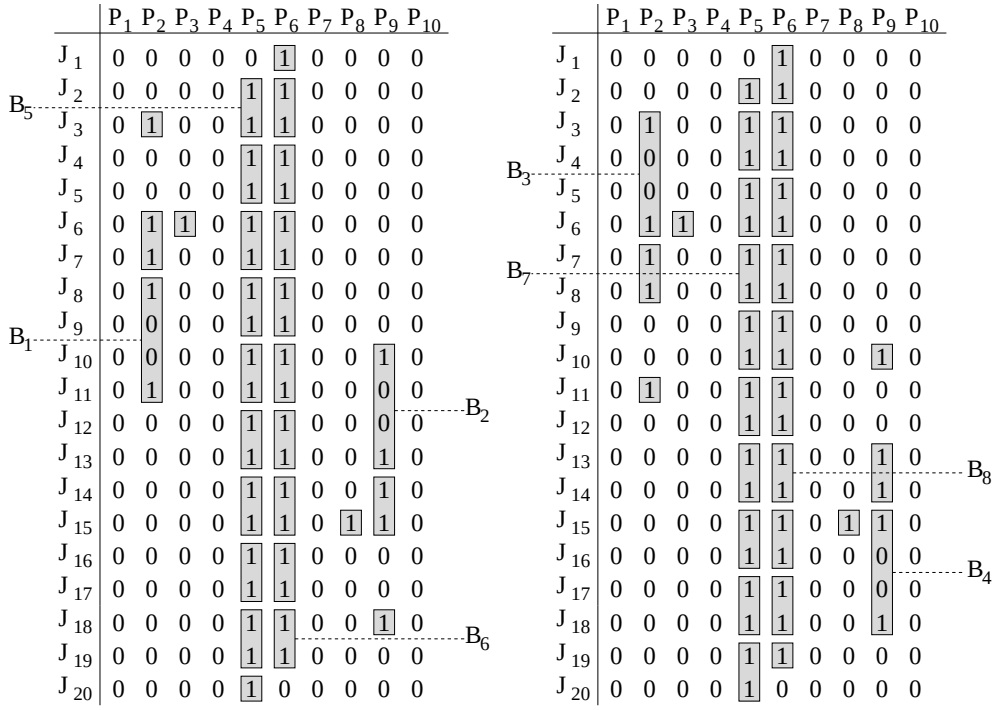


Abbildung 3.6: Gerade und ungerade Überdeckung des Kopplungselementes.

nicht gültig.) Um die Tasks an den Maschinen P_2 bzw. P_9 mit drei gültigen Blöcken zu überdecken, muss daher mindestens einer der Blöcke B_1, B_3 und mindestens einer der Blöcke B_2, B_4 benutzt werden. Da die Blöcke B_1, B_2 mit den Blöcken B_7 bzw. B_8 der ungeraden Zweierüberdeckung des Doppelweges unverträglich sind und da die Blöcke B_3, B_4 mit den Blöcken B_5 bzw. B_6 der geraden Zweierüberdeckung unverträglich sind (siehe Abbildung 3.6), können die Blöcke B_1, B_2 nur mit der geraden Zweierüberdeckung des Doppelweges und die Blöcke B_3, B_4 nur mit der ungeraden Zweierüberdeckung des Doppelweges kombiniert werden. Also muss eine minimale gültige Überdeckung des Kopplungselementes genau eine der beiden in Abbildung 3.6 dargestellten Überdeckungen sein. □

Entscheidend für die Reduktion von 3-SAT ist nun, dass man aus zwei Kopplungselementen und einem Unterbrechungselement eine *Kreuzung* konstruieren kann, so dass sich in einer minimalen gültigen Überdeckung der Kreuzung die geraden bzw. die ungeraden Überdeckungen sowohl in horizontaler Richtung, d.h. vom linken zum rechten Kopplungselement, als auch in vertikaler Richtung, d.h. von der oberen zur unteren Komponente des Unterbrechungselementes, unabhängig voneinander fortsetzen. In Abbildung 3.7 ist eine solche Kreuzung dargestellt.

Lemma 3.7 *Die in den Abbildungen 3.8, 3.9, 3.10 und 3.11 angegebenen vier Überdeckungen sind genau die minimalen gültigen Überdeckungen der Kreuzung.*

Beweis Es ist leicht zu verifizieren, dass die vier Überdeckungen gültig sind. Dass es sich

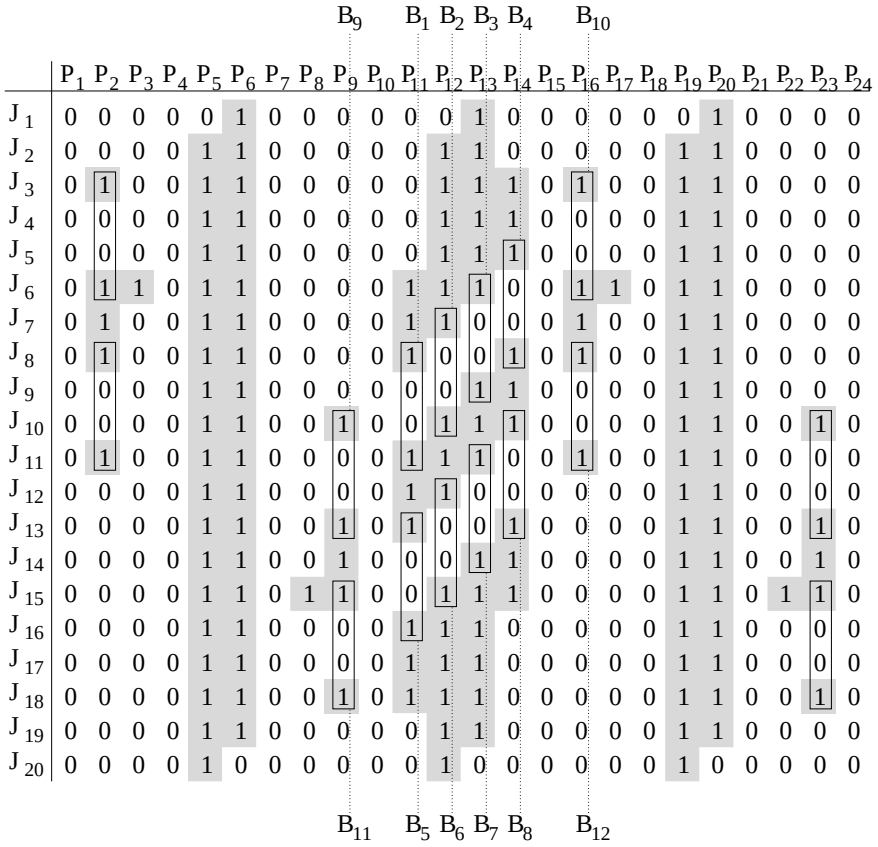


Abbildung 3.7: Kreuzung aus einem Unterbrechungselement und zwei Kopplungselementen.

um minimale gültige Überdeckungen handelt, ist klar, da die Überdeckungen der einzelnen Kopplungselemente und des Unterbrechungselementes nach Lemma 3.5 bzw. Lemma 3.6 minimale gültige Überdeckungen sind.

Um zu sehen, dass es keine weiteren minimalen gültigen Überdeckungen gibt, betrachte man nochmals Abbildung 3.7. Da die angegebenen Überdeckungen die beiden Kopplungselemente und das Unterbrechungselement jeweils minimal gültig überdecken, ist klar, dass dies für jede minimale gültige Überdeckung gelten muss. Nun enthalten die Mengen $\{B_1, B_2, B_3, B_4\}$, $\{B_5, B_6, B_7, B_8\}$, $\{B_1, B_2, B_9\}$, $\{B_3, B_4, B_{10}\}$, $\{B_5, B_6, B_{11}\}$ und $\{B_7, B_8, B_{12}\}$ jeweils paarweise unverträgliche Blöcke. Daraus folgt aber mit Hilfe von Lemma 3.5 und Lemma 3.6, dass die minimalen gültigen Überdeckungen der beiden Kopplungselemente und des Unterbrechungselementes nur auf die vier angegebenen Arten zu einer gültigen Überdeckung der Kreuzung kombiniert werden können. □

In Abbildung 3.12 ist dargestellt, wie die Doppelwege an den Maschinen P_1, P_2 bzw. P_{29}, P_{30} mit Hilfe der Kreuzung über den Doppelweg an den Maschinen P_{15}, P_{16} hinweg aneinander gekoppelt sind. In einer minimalen gültigen Überdeckung müssen der erste und der letzte Doppelweg entweder beide gerade oder beide ungerade überdeckt sein, da die

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}	P_{18}	P_{19}	P_{20}	P_{21}	P_{22}	P_{23}	P_{24}
J_1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
J_2	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0
J_3	0	1	0	0	1	1	0	0	0	0	0	1	1	1	0	1	0	0	1	1	0	0	0	0
J_4	0	0	0	0	1	1	0	0	0	0	0	1	1	1	0	0	0	0	1	1	0	0	0	0
J_5	0	0	0	0	1	1	0	0	0	0	0	1	1	1	0	0	0	0	1	1	0	0	0	0
J_6	0	1	1	0	1	1	0	0	0	0	1	1	1	0	0	1	1	0	1	1	0	0	0	0
J_7	0	1	0	0	1	1	0	0	0	0	1	1	0	0	0	1	0	0	1	1	0	0	0	0
J_8	0	1	0	0	1	1	0	0	0	0	1	0	0	1	0	1	0	0	1	1	0	0	0	0
J_9	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0
J_{10}	0	0	0	0	1	1	0	0	1	0	0	1	1	1	0	0	0	0	1	1	0	0	1	0
J_{11}	0	1	0	0	1	1	0	0	0	0	1	1	1	0	0	1	0	0	1	1	0	0	0	0
J_{12}	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0
J_{13}	0	0	0	0	1	1	0	0	1	0	1	0	0	1	0	0	0	0	1	1	0	0	1	0
J_{14}	0	0	0	0	1	1	0	0	1	0	0	0	1	1	0	0	0	0	1	1	0	0	1	0
J_{15}	0	0	0	0	1	1	0	1	1	0	0	1	1	1	0	0	0	0	1	1	0	1	1	0
J_{16}	0	0	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0
J_{17}	0	0	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0
J_{18}	0	0	0	0	1	1	0	0	1	0	1	1	1	0	0	0	0	0	1	1	0	0	1	0
J_{19}	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0
J_{20}	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0

Abbildung 3.8: Kreuzung mit gerader vertikaler und gerader horizontaler Überdeckung.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}	P_{18}	P_{19}	P_{20}	P_{21}	P_{22}	P_{23}	P_{24}
J_1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
J_2	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0
J_3	0	1	0	0	1	1	0	0	0	0	0	1	1	1	0	1	0	0	1	1	0	0	0	0
J_4	0	0	0	0	1	1	0	0	0	0	0	1	1	1	0	0	0	0	1	1	0	0	0	0
J_5	0	0	0	0	1	1	0	0	0	0	0	1	1	1	0	0	0	0	1	1	0	0	0	0
J_6	0	1	1	0	1	1	0	0	0	0	1	1	1	0	0	1	1	0	1	1	0	0	0	0
J_7	0	1	0	0	1	1	0	0	0	0	1	1	0	0	0	1	0	0	1	1	0	0	0	0
J_8	0	1	0	0	1	1	0	0	0	0	1	0	0	1	0	1	0	0	1	1	0	0	0	0
J_9	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0
J_{10}	0	0	0	0	1	1	0	0	1	0	0	1	1	1	0	0	0	0	1	1	0	0	1	0
J_{11}	0	1	0	0	1	1	0	0	0	0	1	1	1	0	0	1	0	0	1	1	0	0	0	0
J_{12}	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0
J_{13}	0	0	0	0	1	1	0	0	1	0	1	0	0	1	0	0	0	0	1	1	0	0	1	0
J_{14}	0	0	0	0	1	1	0	0	1	0	0	0	1	1	0	0	0	0	1	1	0	0	1	0
J_{15}	0	0	0	0	1	1	0	1	1	0	0	1	1	1	0	0	0	0	1	1	0	1	1	0
J_{16}	0	0	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0
J_{17}	0	0	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0
J_{18}	0	0	0	0	1	1	0	0	1	0	1	1	1	0	0	0	0	0	1	1	0	0	1	0
J_{19}	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0
J_{20}	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0

Abbildung 3.9: Kreuzung mit gerader vertikaler und ungerader horizontaler Überdeckung.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}	P_{18}	P_{19}	P_{20}	P_{21}	P_{22}	P_{23}	P_{24}
J_1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
J_2	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0
J_3	0	1	0	0	1	1	0	0	0	0	0	1	1	1	0	1	0	0	1	1	0	0	0	0
J_4	0	0	0	0	1	1	0	0	0	0	0	1	1	1	0	0	0	0	1	1	0	0	0	0
J_5	0	0	0	0	1	1	0	0	0	0	0	1	1	1	0	0	0	0	1	1	0	0	0	0
J_6	0	1	1	0	1	1	0	0	0	0	0	1	1	0	0	1	1	0	1	1	0	0	0	0
J_7	0	1	0	0	1	1	0	0	0	0	1	1	0	0	0	1	0	0	1	1	0	0	0	0
J_8	0	1	0	0	1	1	0	0	0	0	1	0	0	1	0	1	0	0	1	1	0	0	0	0
J_9	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0
J_{10}	0	0	0	0	1	1	0	0	1	0	0	1	1	1	0	0	0	0	1	1	0	0	1	0
J_{11}	0	1	0	0	1	1	0	0	0	0	1	1	1	0	0	1	0	0	1	1	0	0	0	0
J_{12}	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0
J_{13}	0	0	0	0	1	1	0	0	1	0	1	0	0	1	0	0	0	0	1	1	0	0	1	0
J_{14}	0	0	0	0	1	1	0	0	1	0	0	0	1	1	0	0	0	0	1	1	0	0	1	0
J_{15}	0	0	0	0	1	1	0	1	1	0	0	1	1	0	0	0	0	0	1	1	0	1	1	0
J_{16}	0	0	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0
J_{17}	0	0	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0
J_{18}	0	0	0	0	1	1	0	0	1	0	1	1	1	0	0	0	0	0	1	1	0	0	1	0
J_{19}	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0
J_{20}	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0

Abbildung 3.10: Kreuzung mit ungerader vertikaler und gerader horizontaler Überdeckung.

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}	P_{18}	P_{19}	P_{20}	P_{21}	P_{22}	P_{23}	P_{24}
J_1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
J_2	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0
J_3	0	1	0	0	1	1	0	0	0	0	0	1	1	1	0	1	0	0	1	1	0	0	0	0
J_4	0	0	0	0	1	1	0	0	0	0	0	1	1	1	0	0	0	0	1	1	0	0	0	0
J_5	0	0	0	0	1	1	0	0	0	0	0	1	1	1	0	0	0	0	1	1	0	0	0	0
J_6	0	1	1	0	1	1	0	0	0	0	0	1	1	1	0	0	1	1	0	1	0	0	0	0
J_7	0	1	0	0	1	1	0	0	0	0	1	1	0	0	0	1	0	0	1	1	0	0	0	0
J_8	0	1	0	0	1	1	0	0	0	0	1	0	0	1	0	1	0	0	1	1	0	0	0	0
J_9	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0
J_{10}	0	0	0	0	1	1	0	0	1	0	0	1	1	1	0	0	0	0	1	1	0	0	1	0
J_{11}	0	1	0	0	1	1	0	0	0	0	1	1	1	0	0	1	0	0	1	1	0	0	0	0
J_{12}	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0
J_{13}	0	0	0	0	1	1	0	0	1	0	1	0	0	1	0	0	0	0	1	1	0	0	1	0
J_{14}	0	0	0	0	1	1	0	0	1	0	0	0	1	1	0	0	0	0	1	1	0	0	1	0
J_{15}	0	0	0	0	1	1	0	1	1	0	0	1	1	0	0	0	0	0	1	1	0	1	1	0
J_{16}	0	0	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0
J_{17}	0	0	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	0
J_{18}	0	0	0	0	1	1	0	0	1	0	1	1	1	0	0	0	0	0	1	1	0	0	1	0
J_{19}	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	1	1	0	0	0	0
J_{20}	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0

Abbildung 3.11: Kreuzung mit ungerader vertikaler und ungerader horizontaler Überdeckung.

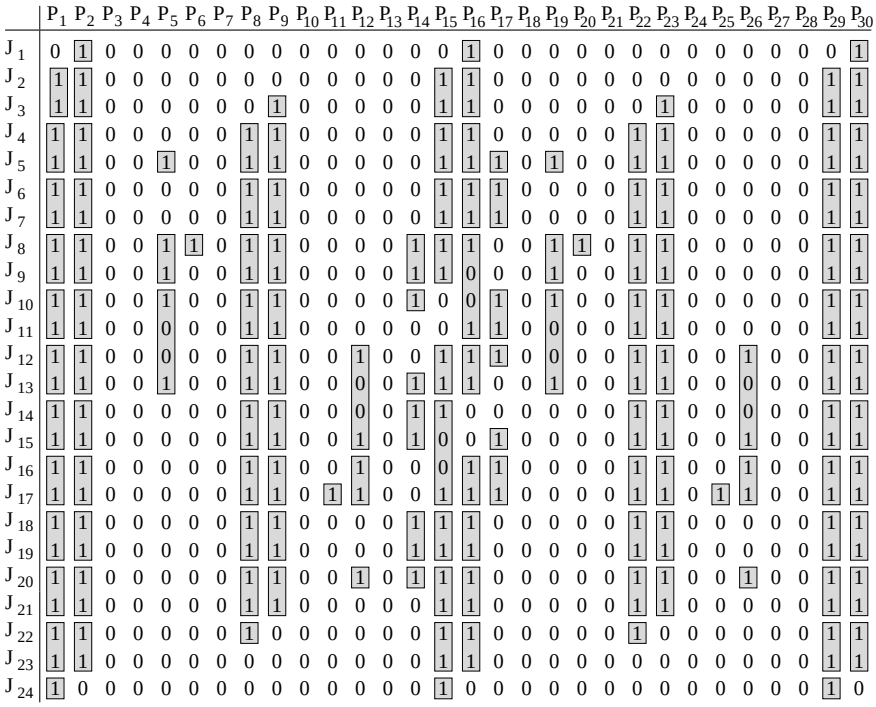


Abbildung 3.12: Kopplung des Doppelweges an den Maschinen P_1, P_2 an den Doppelweg an den Maschinen P_{29}, P_{30}.

geraden bzw. ungeraden Zweierüberdeckungen der Doppelwege nur mit den geraden bzw. ungeraden Überdeckungen der Kopplungselemente kombiniert werden können. Der Doppelweg an den Maschinen P_{15}, P_{16} kann unabhängig davon entweder gerade oder ungerade überdeckt sein. Durch Verwendung mehrerer Kreuzungen kann natürlich eine beliebige Anzahl von vertikalen Doppelwegen überquert werden.

Um ein Literal an das negierte Literal zu koppeln, betrachte man das *invertierende Kopplungselement* aus Abbildung 3.13. Es unterscheidet sich von dem Kopplungselement nur dadurch, dass der Doppelweg in der Mitte durch eine Diagonale unterbrochen ist. Es ist leicht zu sehen, dass die minimalen gültigen Überdeckungen dieses unterbrochenen Doppelweges aus einer Kombination einer geraden bzw. ungeraden Überdeckung des oberen Teils mit einer ungeraden bzw. geraden Überdeckung des unteren Teils bestehen, wobei jeweils zusätzlich ein Einerblock des oberen und des unteren Teils zu einem Dreierblock zusammengefasst sind. Daraus folgt dann auch, dass die beiden in Abbildung 3.13 angegebenen Überdeckungen die einzigen minimalen gültigen Überdeckungen des invertierenden Kopplungselementes sind. Wir haben also das folgende Lemma:

Lemma 3.8 *Die in Abbildung 3.13 dargestellten Überdeckungen sind die einzigen minimalen gültigen Überdeckungen des invertierenden Kopplungselementes und enthalten 27 Blöcke.*

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
J_1	0	0	0	0	0	**1**	0	0	0	0
J_2	0	0	0	0	**1**	**1**	0	0	0	0
J_3	0	**1**	0	0	**1**	**1**	0	0	0	0
J_4	0	0	0	0	**1**	**1**	0	0	0	0
J_5	0	0	0	0	**1**	**1**	0	0	0	0
J_6	0	**1**	**1**	0	**1**	**1**	0	0	0	0
J_7	0	**1**	0	0	**1**	**1**	0	0	0	0
J_8	0	**1**	0	0	**1**	**1**	0	0	0	0
J_9	0	**0**	0	0	**1**	**1**	0	0	0	0
J_{10}	0	**0**	0	0	**1**	0	0	0	**1**	0
J_{11}	0	**1**	0	0	**0**	**1**	0	0	0	0
J_{12}	0	0	0	0	**1**	**1**	0	0	0	0
J_{13}	0	0	0	0	**1**	**1**	0	0	**1**	0
J_{14}	0	0	0	0	**1**	**1**	0	0	**1**	0
J_{15}	0	0	0	0	**1**	**1**	0	**1**	**1**	0
J_{16}	0	0	0	0	**1**	**1**	0	0	**0**	0
J_{17}	0	0	0	0	**1**	**1**	0	0	**0**	0
J_{18}	0	0	0	0	**1**	**1**	0	0	**1**	0
J_{19}	0	0	0	0	**1**	**1**	0	0	0	0
J_{20}	0	0	0	0	**1**	0	0	0	0	0

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}
J_1	0	0	0	0	0	**1**	0	0	0	0
J_2	0	0	0	0	**1**	**1**	0	0	0	0
J_3	0	**1**	0	0	**1**	**1**	0	0	0	0
J_4	0	**0**	0	0	**1**	**1**	0	0	0	0
J_5	0	**0**	0	0	**1**	**1**	0	0	0	0
J_6	0	**1**	**1**	0	**1**	**1**	0	0	0	0
J_7	0	**1**	0	0	**1**	**1**	0	0	0	0
J_8	0	**1**	0	0	**1**	**1**	0	0	0	0
J_9	0	0	0	0	**1**	**1**	0	0	0	0
J_{10}	0	0	0	0	**1**	**0**	0	0	**1**	0
J_{11}	0	**1**	0	0	0	**1**	0	0	**0**	0
J_{12}	0	0	0	0	**1**	**1**	0	0	**0**	0
J_{13}	0	0	0	0	**1**	**1**	0	0	**1**	0
J_{14}	0	0	0	0	**1**	**1**	0	0	**1**	0
J_{15}	0	0	0	0	**1**	**1**	0	**1**	**1**	0
J_{16}	0	0	0	0	**1**	**1**	0	0	0	0
J_{17}	0	0	0	0	**1**	**1**	0	0	0	0
J_{18}	0	0	0	0	**1**	**1**	0	0	**1**	0
J_{19}	0	0	0	0	**1**	**1**	0	0	0	0
J_{20}	0	0	0	0	**1**	0	0	0	0	0

Abbildung 3.13: Die beiden minimalen gültigen Überdeckungen des invertierenden Kopplungselementes.

Beweis Folgt aus dem vorangehenden Text. □

Ersetzt man nun in Abbildung 3.12 eines der Kopplungselemente durch ein invertierendes Kopplungselement, so müssen der erste und der letzte Doppelweg entgegengesetzte Überdeckungen in einer minimalen Überdeckung haben.

Bevor wir die NP-Vollständigkeit des Problems 'Minimale gültige Blocküberdeckung' beweisen, benötigen wir noch ein weiteres Element, um den Wahrheitswert einer Klausel zu überprüfen. Man betrachte dazu das *Klauselelement* aus Abbildung 3.14. Wenn wir zunächst die drei einzelnen Tasks $T_{(11,17)}, T_{(15,13)}$ und $T_{(16,16)}$ vernachlässigen, so besteht das Klauselelement aus drei zusammenhängenden Komponenten. Eine davon ist ein Doppelweg, dessen minimale gültige Überdeckungen die gerade und die ungerade Zweierüberdeckung sind. Die beiden anderen Komponenten sind aus einem vertikalen Doppelweg und einem 'diagonalen Weg' zusammengesetzt. Es ist leicht zu sehen, dass auch diese Komponenten als einzige minimale gültige Überdeckungen die gerade und die ungerade Zweierüberdeckung haben. Die drei einzelnen Tasks können nun mit einer Task aus je einer dieser Komponenten einen gültigen Fünferblock bilden, siehe Abbildung 3.14. Da diese drei Fünferblöcke paarweise unverträglich sind, kann maximal einer von ihnen in einer gültigen Überdeckung vorkommen. Da die Tasks aus den Fünferblöcken, die in den Komponenten liegen, genau in den geraden Überdeckungen in Einerblöcken enthalten sind, folgt, dass man genau dann durch die Verwendung eines Fünferblocks einen Block einsparen kann, wenn mindestens eine der Komponenten gerade überdeckt ist. Also ergibt

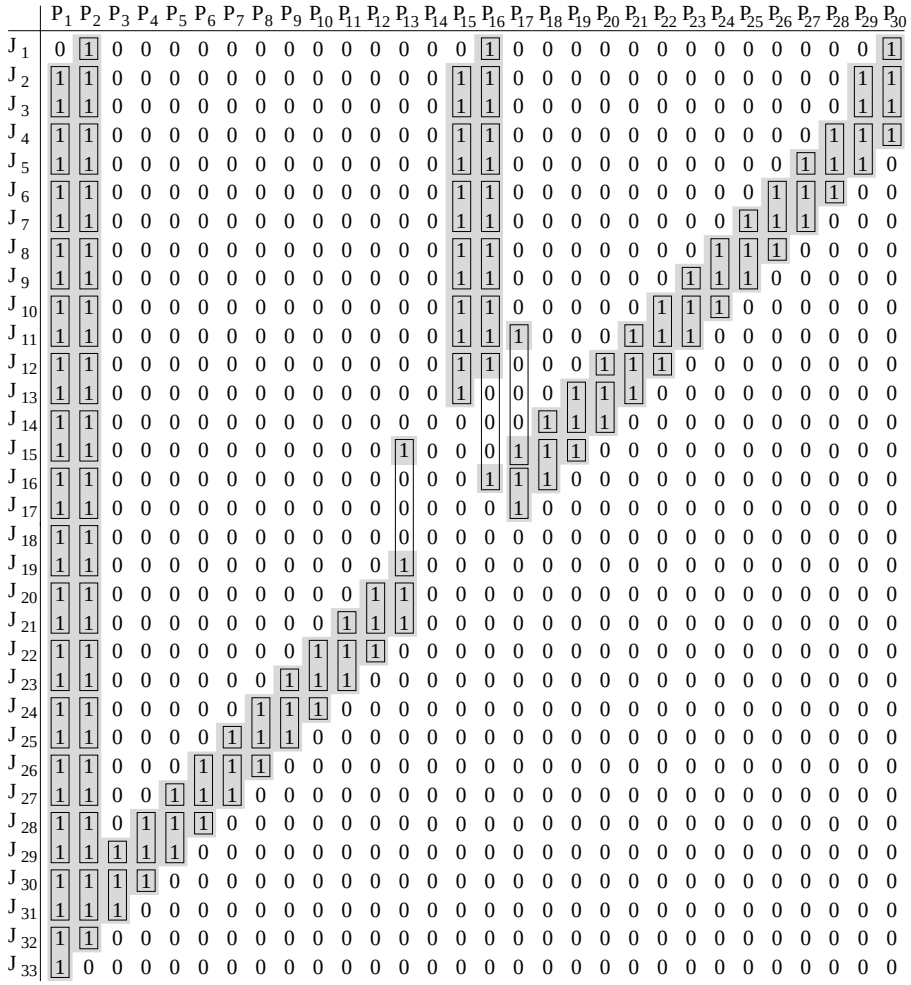


Abbildung 3.14: Klauselelement: Es können genau dann zwei Einerblöcke zu einem Fünferblock zusammengefasst werden, wenn mindestens einer der drei Wege gerade überdeckt ist. Die angegebene Überdeckung ist nicht gültig, da die drei Fünferblöcke paarweise unverträglich sind.

sich das folgende Lemma.

Lemma 3.9 *Jede minimale Überdeckung des Klauselelements aus Abbildung 3.14 induziert in jeder der drei zusammenhängenden Komponenten ohne die drei einzelnen Tasks $T_{(11,17)}, T_{(15,13)}$ und $T_{(16,16)}$ eine gerade oder eine ungerade Überdeckung, wobei mindestens eine der drei Überdeckungen gerade ist. Ferner ist ein Einerblock aus einer der induzierten geraden Überdeckungen mit einer der drei Tasks $T_{(11,17)}, T_{(15,13)}$ oder $T_{(16,16)}$ zu einem Fünferblock zusammengefasst.*

Beweis Folgt aus dem vorangehenden Text. □

Wir können nun die NP-Vollständigkeit des Problems 'Minimale gültige Blocküberdeckung' beweisen.

Satz 3.4 *Das Problem 'Minimale gültige Blocküberdeckung' ist NP-vollständig.*

Beweis Wir beweisen die NP-Vollständigkeit, indem wir eine polynomielle Transformation von 3-SAT angeben. Dazu konstruieren wir zu jeder Instanz I von 3-SAT eine Taskmatrix $\mathcal{M}_I$ und eine ganze Zahl z_I, so dass $\mathcal{M}_I$ genau dann eine gültige Blocküberdeckung mit höchstens z_I Blöcken besitzt, wenn die Instanz von 3-SAT erfüllbar ist.

Sei die Instanz I von 3-SAT durch die Variablenmenge $X = \{x_1, x_2, \ldots, x_t\}$ und die Klauseln $C_1, C_2, \ldots, C_r$ mit $C_i = \{c_{i,1}, c_{i,2}, c_{i,3}\}$, $i = 1, \ldots, r$, gegeben. Unsere Taskmatrix $\mathcal{M}_I$ hat dann $2+14\cdot(3r-1)$ Maschinen und $30\cdot t+33$ Jobs. Für das Literal $c_{i,j}$, $1 \leq i \leq r$, $1 \leq j \leq 3$, ist in den Spalten $1 + 14(3(i-1)+j-1)$ und $2 + 14(3(i-1)+j-1)$ ein möglicherweise mehrfach durch Unterbrechungselemente unterbrochener Doppelweg vorhanden, der am unteren Ende der Matrix in ein Klauselelement mündet. Alle Doppelwege von Literalen, die zu derselben Variablen x_k gehören, werden mit Hilfe von Kopplungselementen, invertierenden Kopplungselementen und Unterbrechungselementen im Bereich der Zeilen $30(k-1)+1$ bis $30k$ aneinander gekoppelt. Die Kopplungselemente und die Unterbrechungselemente werden dazu in der Mitte dieses Bereiches platziert. Auf diese Weise befinden sich zwischen zwei übereinander angeordneten Kopplungselementen aus benachbarten Bereichen mindestens 10 Zeilen, in denen keine Tasks vorkommen. Dies schließt aus, dass Tasks aus verschiedenen Kopplungselementen in einem gemeinsamen gültigen Block enthalten sein können. In den letzten 33 Jobs der Taskmatrix werden nun jeweils die drei zu einer Klausel gehörenden Doppelwege mit denen eines Klauselelements zu durchgehenden Doppelwegen verbunden. Dies vervollständigt die Beschreibung der Taskmatrix $\mathcal{M}_I$. Als Beispiel betrachte man eine Instanz I mit den Variablen $\{x_1, x_2, x_3, x_4\}$ und den Klauseln $C_1 = \{x_1, \overline{x_2}, x_4\}$, $C_2 = \{x_2, \overline{x_3}, \overline{x_4}\}$ und $C_3 = \{\overline{x_1}, \overline{x_2}, \overline{x_3}\}$. In Abbildung 3.15 ist der Aufbau der Matrix für diese Instanz schematisch dargestellt.

Wir müssen jetzt noch die Zahl z_I angeben, so dass $\mathcal{M}_I$ genau dann eine gültige Blocküberdeckung mit höchstens z_I Blöcken besitzt, wenn die Instanz von 3-SAT erfüllbar ist. Seien $n_U, n_K, n_{\overline{K}}$ die Anzahl der verwendeten Unterbrechungs-, Kopplungs- bzw. invertierenden Kopplungselemente. Dann definieren wir

$$z_I := 28n_K + 27n_{\overline{K}} + 8n_U + 90rt + 99r.$$

Es gibt offensichtlich einen polynomiellen Algorithmus, der aus einer gegebenen Instanz I von 3-SAT die Taskmatrix $\mathcal{M}_I$ sowie die Zahl z_I konstruiert. Wir beweisen nun den Satz, indem wir zeigen, dass $\mathcal{M}_I$ genau dann eine gültige Blocküberdeckung mit höchstens z_I Blöcken besitzt, wenn die Instanz von 3-SAT eine erfüllende Belegung hat.

Betrachten wir zunächst die Taskmatrix $\mathcal{M}'_I$, die wir aus $\mathcal{M}_I$ erhalten, indem wir in allen Klauselelementen die drei einzelnen Tasks, die nicht in den vertikalen und diagonalen Wegen enthalten sind, entfernen. Aufgrund der vorangehenden Diskussion ist klar, dass es für jede beliebige Belegung τ der Variablen eine minimale gültige Überdeckung $\mathcal{C}_\tau(\mathcal{M}'_I)$ der Matrix $\mathcal{M}'_I$ gibt, so dass die vertikalen Doppelwege einschließlich der zugehörigen diagonalen Wege in den Klauselelementen genau dann gerade überdeckt sind, wenn das zugehörige Literal wahr ist. Außerdem folgt aus den vorangehenden Betrachtungen, dass jede minimale Überdeckung dieses Aussehen für eine passende Belegung τ hat. Wir wollen

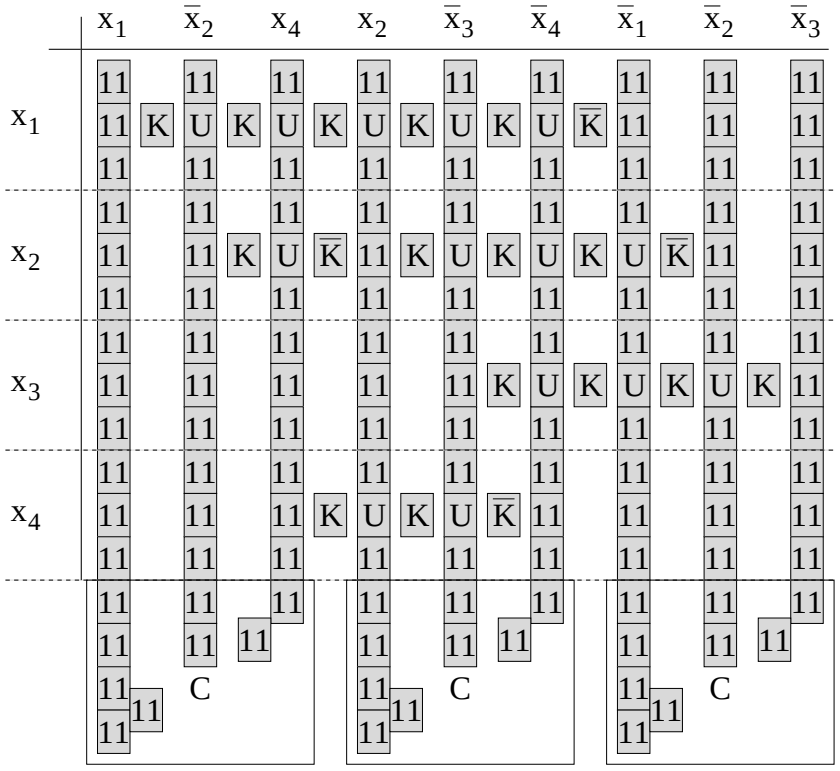


Abbildung 3.15: Aufbau der Matrix für die Beispiel-Instanz $C_1 = \{x_1, \overline{x_2}, x_4\}$, $C_2 = \{x_2, \overline{x_3}, \overline{x_4}\}$ und $C_3 = \{\overline{x_1}, \overline{x_2}, \overline{x_3}\}$. Bausteine '11' kennzeichnen Doppelwege, 'K' Kopplungselemente, '$\overline{\text{K}}$' invertierende Kopplungselemente, 'U' Unterbrechungselemente und 'C' Klauselelemente.

die Anzahl der Blöcke in einer solchen minimalen Überdeckung bestimmen. Für $1 \leq i \leq r$ und $1 \leq j \leq 3$ sei $l_{i,j}$ die Länge des zum Literal $c_{i,j}$ gehörenden Doppelweges einschließlich etwaiger Unterbrechungen und einschließlich der Fortsetzung in das Klauselelement. Da die drei Doppelwege des Klauselelements die Längen 32, 12 bzw 4 haben, ergibt sich also

$$l_{i,j} = \begin{cases} 30t + 32, & \text{falls } j = 1 \mod 3, \\ 30t + 12, & \text{falls } j = 2 \mod 3, \\ 30t + 4, & \text{falls } j = 0 \mod 3. \end{cases}$$

Wären die Doppelwege nicht unterbrochen, so enthielte die Überdeckung nach Lemma 3.4 $\sum_{i=1}^{r} \sum_{j=1}^{3} (l_i + 1) = 90rt + 51r$ Blöcke zur Überdeckung der Doppelwege. Für jedes Unterbrechungselement werden aber nach Lemma 3.5 acht zusätzliche Blöcke benötigt. Für jedes Kopplungselement sind 28 und für jedes invertierende Kopplungselement 27 Blöcke erforderlich. Da für jedes der r Klauselelemente noch 46 Blöcke für die Überdeckung der diagonalen Wege gebraucht werden, ergibt dies insgesamt

$$28n_K + 27n_{\overline{K}} + 8n_U + 46r + 90rt + 51r = z_I - 2r$$

Blöcke für eine minimale Überdeckung $\mathcal{C}_\tau(\mathcal{M}'_I)$ der Matrix $\mathcal{M}'_I$.

Ist τ nun eine erfüllende Belegung, so ist in jedem Klauselelement eine der Komponenten in der Überdeckung $\mathcal{C}_\tau(\mathcal{M}'_I)$ gerade überdeckt. Wir können dann offensichtlich aus $\mathcal{C}_\tau(\mathcal{M}'_I)$ eine gültige Überdeckung für $\mathcal{M}_I$ konstruieren, indem wir in jedem Klauselelement eine der zusätzlichen Tasks mit einem Einerblock einer solchen geraden Komponente zusammenfassen und die beiden anderen Tasks mit je einem Einerblock überdecken. Dies ergibt $2r$ zusätzliche Blöcke und somit hat die Überdeckung die geforderte Anzahl z_I von Blöcken.

Hat andererseits die Matrix $\mathcal{M}_I$ eine gültige Überdeckung $\mathcal{C}$ mit z_I Blöcken, so induziert dies auf der Matrix $\mathcal{M}'_I$ eine gültige Überdeckung, die höchstens $z_I - 2r$ Blöcke hat, da mindestens zwei der Einzeltasks jedes Klauselelementes in einem Einerblock enthalten sind. Da eine minimale gültige Überdeckung von $\mathcal{M}'_I$ ebensoviele Blöcke hat, muss die induzierte Überdeckung genau $z_I - 2r$ Blöcke besitzen und somit minimal sein. Daher definiert diese Überdeckung eine Belegung τ der Variablen, die genau die Literale enthält, deren Doppelwege gerade überdeckt sind. Da die Überdeckung der gesamten Matrix nur $2r$ Blöcke mehr enthält, muss in jedem Klauselelement eine der Einzeltasks in einem Fünferblock enthalten sein. Nach Lemma 3.9 muss dann aber eine der drei Komponenten des Klauselelements gerade belegt sein und somit das entsprechende Literal wahr unter τ sein. Dies bedeutet aber, dass τ eine erfüllende Belegung ist. □

Korollar 3.1 *Das Förderband-Flow-Shop-Problem für Einheitsdistanzen ist NP-vollständig.*

Beweis Wegen Satz 3.3 gibt es genau dann eine Abarbeitung der Taskmatrix $\mathcal{M}$ mit einer Einheitsdistanz $\leq K$, wenn es eine gültige Blocküberdeckung von $\mathcal{M}$ mit höchstens K Blöcken gibt. Daraus folgt unmittelbar das Korollar. □

Kapitel 4

Approximationsalgorithmen

In diesem Kapitel beschäftigen wir uns mit Approximationsalgorithmen für das Förderband-Flow-Shop-Problem. Für einen Approximationsalgorithmus A bezeichne $\Delta_{\mathrm{A}}^{\mathcal{D}}(\mathcal{M})$ die Distanz bezüglich $\mathcal{D}$ der Abarbeitung, die der Algorithmus A für eine Taskmatrix $\mathcal{M}$ liefert. Für $c \geq 1$ nennen wir A einen *c-Approximationsalgorithmus*, falls für alle Taskmatrizen $\mathcal{M}$

$$\Delta_{\mathrm{A}}^{\mathcal{D}}(\mathcal{M}) \leq c \cdot \Delta_{\min}^{\mathcal{D}}(\mathcal{M})$$

gilt. Das Infimum aller c, so dass A ein c-Approximationsalgorithmus ist, nennen wir die *Güte von A*. Wir werden in diesem Kapitel einen 2-Approximationsalgorithmus für Einheitsdistanzen sowie einen 3-Approximationsalgorithmus für additive Distanzen angeben. Für die Abschätzung der Güte bestimmen wir in beiden Fällen zunächst eine untere Schranke für die Distanz einer Abarbeitung. Auszüge der ersten beiden Abschnitte sowie der 3-Approximationsalgorithmus wurden in Kurzform bereits in [7] bzw. in [9] vorgestellt.

4.1 Eine untere Schranke für Einheitsdistanzen

Für die Bestimmung der unteren Schranke benötigen wir die Begriffe aus Abschnitt 3.4, die wir kurz wiederholen. Die *(Task-)Blöcke* $\Phi_p^{[j,j']}$ einer Taskmatrix sind die nicht-leeren Folgen

$$T_{(j_1,p)}, T_{(j_2,p)}, \ldots, T_{(j_k,p)}, \quad 1 \leq j = j_1 < j_2 < \ldots < j_k = j' \leq n$$

von Tasks an einer Maschine P_p, die alle Tasks $T_{(l,p)}$ mit $j \leq l \leq j'$ enthalten. Ein Block $B = \Phi_p^{[j,j']}$ heißt *gültig*, wenn es keine Task $T \in \mathcal{T}(\mathcal{M}) - B$ gibt mit $T_{(j,p)} \prec T \prec T_{(j',p)}$. Die gültigen Blöcke sind genau diejenigen, die in einer Abarbeitung als zusammenhängende Teilfolge enthalten sein können. Zwei Blöcke $\Phi_{p_1}^{[j_1,j_1']}$ und $\Phi_{p_2}^{[j_2,j_2']}$ sind *unverträglich*, wenn sie entweder eine gemeinsame Task enthalten oder wenn sowohl $T_{(j_1,p_1)} \prec T_{(j_2',p_2)}$ als auch $T_{(j_2,p_2)} \prec T_{(j_1',p_1)}$ gilt. Eine Menge von Blöcken nennen wir eine *(Block-)Überdeckung der Taskmatrix*, wenn jede Task in einem Block enthalten ist. Die Überdeckung heißt *gültig*, wenn sie aus paarweise verträglichen Blöcken besteht. Die Blöcke einer Überdeckung können genau dann zu einer Abarbeitung angeordnet werden, wenn die Über-

deckung gültig ist. Daher ist die Anzahl der Blöcke in einer minimalen gültigen Überdeckung gleich der Einheitsdistanz einer minimalen Abarbeitung. Da jede Task in einem Block der Überdeckung enthalten sein muss, kann eine gültige Überdeckung nur gültige Blöcke enthalten. Eine Überdeckung mit gültigen Blöcken ist aber nicht notwendigerweise gültig. Da eine gültige Überdeckung aber nur gültige Blöcke enthält, ist die Anzahl der Blöcke in einer minimalen Überdeckung mit gültigen Blöcken auf jeden Fall eine untere Schranke für die Anzahl der Blöcke in einer gültigen Überdeckung und somit auch eine untere Schranke für die minimale Einheitsdistanz einer Abarbeitung.

Da man zu einer Task $T_{(j,p)}$ leicht den maximalen Index j' bestimmen kann, so dass $\Phi_p^{[j,j']}$ ein gültiger Block ist, lässt sich eine minimale Überdeckung der Taskmatrix mit gültigen Blöcken sehr einfach durch den folgenden Algorithmus bestimmen. Dieser berechnet für jede Maschine P_p, $p = 1, \ldots, m$, 'top-down' eine minimale Überdeckung $\mathcal{C}_p$ der Tasks an der Maschine P_p mit gültigen Blöcken. Die Vereinigung $\mathcal{C}$ aller $\mathcal{C}_p$ bildet dann eine minimale Überdeckung der Taskmatrix mit gültigen Blöcken.

Algorithmus 2 (Top-Down-Überdeckung gültiger Blöcke)

(1) **for** $(p = 1, \ldots, m)$ **{**
(2) $\mathcal{C}_p = \emptyset$;
(3) **while** (es gibt Task $T_{(j,p)}$, die in keinem Block von $\mathcal{C}_p$ enthalten ist) **{**
(4) wähle eine solche Task $T_{(j,p)}$ mit minimalem j;
(5) bestimme maximales $j' \geq j$, so dass $\Phi_p^{[j,j']}$ ein gültiger Block ist;
(6) $\mathcal{C}_p = \mathcal{C}_p \cup \{\Phi_p^{[j,j']}\}$;
(7) **}**
(8) **}**
(9) $\mathcal{C} = \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_m$;

Da jede zusammenhängende Teilfolge eines gültigen Blocks wieder ein gültiger Block ist, ist offensichtlich, dass der Algorithmus für jede Maschine P_p eine minimale Überdeckung $\mathcal{C}_p$ der Tasks an der Maschine P_p mit gültigen Blöcken berechnet. Wir bezeichnen die Anzahl der Blöcke in $\mathcal{C}_p$ mit $\mathrm{LB}_p^{\mathrm{E}}(\mathcal{M})$. Die Vereinigung der Top-Down-Überdeckungen aller Maschinen bildet dann natürlich eine minimale Überdeckung der Taskmatrix mit gültigen Blöcken. Wir bezeichnen diese als *Top-Down-Überdeckung der Taskmatrix mit gültigen Blöcken*. Offensichtlich ist also

$$\mathrm{LB}^{\mathrm{E}}(\mathcal{M}) := \sum_{p=1}^{m} \mathrm{LB}_p^{\mathrm{E}}(\mathcal{M}) \tag{4.1}$$

eine untere Schranke für die Einheitsdistanz einer Abarbeitung der Taskmatrix $\mathcal{M}$ mit m Maschinen.

	P_1	P_2	P_3	P_4	P_5	P_6
J_1	1	0	0	1	0	1
J_2	0	0	1	0	1	0
J_3	1	0	1	1	0	0
J_4	0	1	1	0	0	1
J_5	1	1	0	0	1	1
J_6	1	0	0	1	1	0
J_7	0	0	1	0	0	1
J_8	0	1	0	1	0	1

	P_1	P_2	P_3	P_4	P_5
J_1	0	0	0	0	1
J_2	0	0	0	1	0
J_3	0	0	1	0	0
J_4	0	1	0	0	0
J_5	1	0	0	0	1
J_6	0	0	0	1	0
J_7	0	0	1	0	0
J_8	0	1	0	0	0
J_9	1	0	0	0	0

Abbildung 4.1: Top-Down-Überdeckungen zweier Taskmatrizen mit gültigen Blöcken. Die Überdeckungen sind nicht gültig, da sie unverträgliche Blöcke enthalten.

Abbildung 4.1 zeigt die Top-Down-Überdeckungen zweier Taskmatrizen mit gültigen Blöcken. Da die Überdeckung der linken Taskmatrix 14 Blöcke enthält, beträgt die Einheitsdistanz jeder Abarbeitung dieser Taskmatrix mindestens 14. Die Überdeckung der rechten Matrix besteht aus fünf paarweise unverträglichen gültigen Blöcken. Jede Abarbeitung muss daher mindestens die Einheitsdistanz 9 besitzen, da höchstens ein Block mit zwei Tasks gebildet werden kann. Das Beispiel lässt sich offensichtlich auf eine beliebige Anzahl von m paarweise unverträglichen gültigen Blöcken mit je zwei Tasks verallgemeinern. Dieses zeigt, dass das Verhältnis von minimaler Distanz zur unteren Schranke dem Wert 2 beliebig nahe kommen kann. Im nächsten Abschnitt werden wir sehen, dass dieses auch tatsächlich der schlechteste Fall ist.

4.2 Ein Approximationsalgorithmus für Einheitsdistanzen

Wir stellen in diesem Abschnitt einen Approximationsalgorithmus vor, der Abarbeitungen berechnet, deren Einheitsdistanz höchstens das Zweifache der unteren Schranke $\mathrm{LB}^{\mathrm{E}}(\mathcal{M})$ aus Gleichung 4.1 ist. Wir zeigen zunächst einige benötigte Lemmata.

Lemma 4.1 *Seien $B_1 = \Phi_{p_1}^{[s_1,t_1]}$ und $B_2 = \Phi_{p_2}^{[s_2,t_2]}$ zwei gültige Blöcke einer Taskmatrix mit $p_1 < p_2$. Dann sind B_1 und B_2 genau dann unverträglich, wenn die beiden folgenden Ungleichungen gelten:*

(i) $s_2 < s_1 \leq t_2 < t_1$ *und*

(ii) $s_1 + p_1 \leq s_2 + p_2 < t_1 + p_1 \leq t_2 + p_2$.

Beweis '$\Rightarrow$' Da B_1 und B_2 disjunkt und unverträglich sind, folgt $T_{(s_1,p_1)} \prec T_{(t_2,p_2)}$ und $T_{(s_2,p_2)} \prec T_{(t_1,p_1)}$. Aus der Definition von '$\prec$' folgt daher $s_1 \leq t_2$ und $s_2 + p_2 < t_1 + p_1$.

Nach Voraussetzung ist $p_1 < p_2$. Wäre $s_1 \leq s_2$, so folgte $T_{(s_1,p_1)} \prec T_{(s_2,p_2)}$ und somit wegen $T_{(s_2,p_2)} \prec T_{(t_1,p_1)}$ die Ungültigkeit von B_1. Also gilt $s_2 < s_1$. Wäre $t_1 \leq t_2$, so folgte $T_{(t_1,p_1)} \prec T_{(t_2,p_2)}$ und daraus wegen $T_{(s_2,p_2)} \prec T_{(t_1,p_1)}$ die Ungültigkeit von B_2. Also ist $t_2 < t_1$ und somit gilt Ungleichung (i).

Wäre jetzt $s_2+p_2 < s_1+p_1$, so folgte wegen der Gültigkeit der Ungleichung (i) $T_{(s_2,p_2)} \prec T_{(s_1,p_1)}$. Dann wäre aber B_2 wegen $T_{(s_1,p_1)} \prec T_{(t_2,p_2)}$ ungültig. Genauso führt die Annahme $t_2 + p_2 < t_1 + p_1$ in Verbindung mit der Gültigkeit der Ungleichung (i) zu $T_{(t_2,p_2)} \prec T_{(t_1,p_1)}$ und somit wegen $T_{(s_1,p_1)} \prec T_{(t_2,p_2)}$ zur Ungültigkeit von B_1. Also gilt auch die Ungleichung (ii).

'$\Leftarrow$' Die Ungleichungen (i) und (ii) implizieren offensichtlich $T_{(s_1,p_1)} \prec T_{(t_2,p_2)}$ und $T_{(s_2,p_2)} \prec T_{(t_1,p_1)}$. Also sind B_1 und B_2 unverträglich. □

Lemma 4.2 *Seien $B_1 = \Phi_{p_1}^{[s_1,t_1]}$, $B_2 = \Phi_{p_2}^{[s_2,t_2]}$ und $B_3 = \Phi_{p_3}^{[s_3,t_3]}$ gültige Blöcke mit $p_1, p_2 < p_3$. Wenn sowohl B_1 und B_3 als auch B_2 und B_3 unverträglich sind, so sind B_1 und B_2 ebenfalls unverträglich.*

Beweis Angenommen, B_1, B_3 und B_2, B_3 sind unverträglich. Nach Lemma 4.1 folgen dann die Ungleichungen

$$\begin{aligned} s_3 < s_1 \leq t_3 < t_1, \\ s_3 < s_2 \leq t_3 < t_2, \\ s_1 + p_1 \leq s_3 + p_3 < t_1 + p_1 \leq t_3 + p_3, \\ s_2 + p_2 \leq s_3 + p_3 < t_2 + p_2 \leq t_3 + p_3. \end{aligned}$$

Aus diesen Ungleichungen folgt einerseits $s_1 \leq t_2$ und $s_1+p_1 < t_2+p_2$, d.h. $T_{(s_1,p_1)} \prec T_{(t_2,p_2)}$, und andererseits $s_2 \leq t_1$ und $s_2 + p_2 < t_1 + p_1$, d.h. $T_{(s_2,p_2)} \prec T_{(t_1,p_1)}$. Also sind die Blöcke B_1 und B_2 unverträglich. □

Lemma 4.3 *Seien $B_1 = \Phi_{p_1}^{[s_1,t_1]}$ und $B_2 = \Phi_{p_2}^{[s_2,t_2]}$ zwei gültige unverträgliche Blöcke einer Taskmatrix mit $p_1 < p_2$. Dann kann der Block B_2 so in zwei disjunkte Blöcke $B_{2,1}$ und $B_{2,2}$ aufgeteilt werden, dass $B_{2,1}$ und $B_{2,2}$ zusammen alle Tasks aus B_2 enthalten und die Blöcke $B_1, B_{2,1}, B_{2,2}$ paarweise verträglich sind.*

Beweis Nach Lemma 4.1 gilt $s_2 < s_1 \leq t_2$. Daher sind der größte Jobindex $t_2' < s_1$ und der kleinste Jobindex $s_2' \geq s_1$, so dass $T_{(t_2',p_2)}$ bzw. $T_{(s_2',p_2)}$ Tasks sind, definiert, siehe auch Abbildung 4.2. Die Blöcke $B_{2,1} := \Phi_{p_2}^{[s_2,t_2']}$ und $B_{2,2} := \Phi_{p_2}^{[s_2',t_2]}$ sind offenbar disjunkt, miteinander verträglich und enthalten genau die Tasks aus B_2. Da die Blöcke $B_{2,1}$ und $B_{2,2}$ als Teilblöcke eines gültigen Blocks ebenfalls gültig sind, folgt die Verträglichkeit der beiden Blöcke mit B_1 unmittelbar aus Lemma 4.1 (i). □

Wir können nun offensichtlich eine gültige Überdeckung der Taskmatrix durch folgendes Verfahren bestimmen. Zunächst berechnen wir die Top-Down-Überdeckung der Taskmatrix mit gültigen Blöcken. Dann testen wir der Reihe nach für $p = 2, \ldots, m$, ob die Blöcke auf den Maschinen P_p dieser Überdeckung verträglich mit allen Blöcken auf Maschinen mit

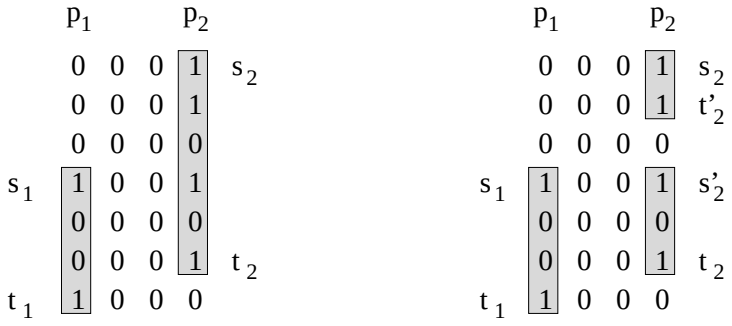


Abbildung 4.2: Aufteilung eines gültigen Blocks zur Entfernung einer Unverträglichkeit.

kleinerem Index sind. Falls dies für einen Block B nicht der Fall ist, kann nach Lemma 4.2 eine Unverträglichkeit mit höchstens einem Block auf einer Maschine mit kleinerem Index vorliegen. Wir beseitigen dann diese Unverträglichkeit, indem wir den Block B wie im Beweis zu Lemma 4.3 in zwei Blöcke aufteilen. Offensichtlich erhalten wir dadurch eine gültige Blocküberdeckung, die höchstens zweimal soviele Blöcke enthält wie die Top-Down-Überdeckung mit gültigen Blöcken, d.h. höchstens zweimal soviele Blöcke wie die untere Schranke $\mathrm{LB}^{\mathrm{E}}(\mathcal{M})$. Dass das Verhältnis der Anzahl der erzeugten Blöcke zur unteren Schranke dem Wert 2 beliebig nahe kommen kann, ist klar, da wir aus dem Beispiel aus Abbildung 4.1 wissen, dass auch das Verhältnis einer optimalen Lösung zur unteren Schranke dem Wert 2 beliebig nahe kommen kann. Dieses Beispiel schließt allerdings noch nicht aus, dass das Verhältnis zu einer optimalen Lösung immer kleiner als c für eine Konstante $c < 2$ sein könnte. Dass dies nicht der Fall ist, zeigt eine Verallgemeinerung des Beispiels aus Abbildung 4.3, bei dem fast jeder Block der Top-Down-Überdeckung gültiger Blöcke aufgeteilt wird, obwohl die Anzahl der Blöcke einer minimalen gültigen Überdeckung gleich der unteren Schranke ist. In diesem Beispiel könnten nach dem Aufteilen der Blöcke wieder Blöcke zusammengefasst werden, um eine gültige Überdeckung mit weniger Blöcken zu erhalten. Der Algorithmus auf der nächsten Seite erzeugt von vornherein nur gültige Überdeckungen, in denen keine Blöcke mehr zusammengefasst werden können. Er arbeitet fast genauso wie der Algorithmus zur Berechnung der Top-Down-Überdeckung mit gültigen Blöcken. Anstelle von maximal gültigen Blöcken bestimmt er jedoch solche maximalen Blöcke, die gültig und zugleich verträglich mit allen bereits berechneten Blöcken sind. Wir nennen den Algorithmus *Top-Down-Cover* und die von ihm berechnete gültige Überdeckung die *Top-Down-Überdeckung der Taskmatrix mit verträglichen Blöcken*. Da Einerblöcke stets gültig und mit allen gültigen Blöcken verträglich sind, ist offensichtlich, dass der Algorithmus eine gültige Blocküberdeckung berechnet. Der folgende Satz zeigt, dass seine Güte ≤ 2 ist.

Satz 4.1 *Die von Algorithmus 3 berechnete Top-Down-Überdeckung einer Taskmatrix $\mathcal{M}$ mit verträglichen Blöcken enthält höchstens $2 \cdot LB^E(\mathcal{M})$ Blöcke und damit höchstens zweimal soviele Blöcke wie eine minimale gültige Überdeckung.*

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
J_1	0	0	0	0	0	0	0	1
J_2	0	0	0	0	0	1	0	0
J_3	0	0	0	1	0	0	0	0
J_4	0	1	0	0	0	0	0	0
J_5	0	0	0	0	0	0	0	1
J_6	0	0	0	0	0	1	0	1
J_7	0	0	0	1	0	1	0	0
J_8	0	1	0	1	0	0	0	0
J_9	1	1	0	0	0	0	0	0
J_{10}	0	0	0	0	0	0	0	1
J_{11}	0	0	0	0	0	1	0	1
J_{12}	0	0	0	1	0	1	0	0
J_{13}	0	1	0	1	0	0	0	0
J_{14}	1	1	0	0	0	0	0	0
J_{15}	0	0	0	0	0	0	0	1
J_{16}	0	0	0	0	0	1	0	1
J_{17}	0	0	0	1	0	1	0	0
J_{18}	0	1	0	1	0	0	0	0
J_{19}	1	1	0	0	0	0	0	0

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
J_1	0	0	0	0	0	0	0	1
J_2	0	0	0	0	0	1	0	0
J_3	0	0	0	1	0	0	0	0
J_4	0	1	0	0	0	0	0	0
J_5	0	0	0	0	0	0	0	1
J_6	0	0	0	0	0	1	0	1
J_7	0	0	0	1	0	1	0	0
J_8	0	1	0	1	0	0	0	0
J_9	1	1	0	0	0	0	0	0
J_{10}	0	0	0	0	0	0	0	1
J_{11}	0	0	0	0	0	1	0	1
J_{12}	0	0	0	1	0	1	0	0
J_{13}	0	1	0	1	0	0	0	0
J_{14}	1	1	0	0	0	0	0	0
J_{15}	0	0	0	0	0	0	0	1
J_{16}	0	0	0	0	0	1	0	1
J_{17}	0	0	0	1	0	1	0	0
J_{18}	0	1	0	1	0	0	0	0
J_{19}	1	1	0	0	0	0	0	0

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
J_1	0	0	0	0	0	0	0	1
J_2	0	0	0	0	0	1	0	0
J_3	0	0	0	1	0	0	0	0
J_4	0	1	0	0	0	0	0	0
J_5	0	0	0	0	0	0	0	1
J_6	0	0	0	0	0	1	0	1
J_7	0	0	0	1	0	1	0	0
J_8	0	1	0	1	0	0	0	0
J_9	1	1	0	0	0	0	0	0
J_{10}	0	0	0	0	0	0	0	1
J_{11}	0	0	0	0	0	1	0	1
J_{12}	0	0	0	1	0	1	0	0
J_{13}	0	1	0	1	0	0	0	0
J_{14}	1	1	0	0	0	0	0	0
J_{15}	0	0	0	0	0	0	0	1
J_{16}	0	0	0	0	0	1	0	1
J_{17}	0	0	0	1	0	1	0	0
J_{18}	0	1	0	1	0	0	0	0
J_{19}	1	1	0	0	0	0	0	0

Abbildung 4.3: Top-Down-Überdeckung gültiger Blöcke (links), gültige Überdeckung des Algorithmus, bei der fast jeder Block aufgeteilt wird (Mitte) und minimale gültige Überdeckung (rechts)

Algorithmus 3 (Top-Down-Überdeckung verträglicher Blöcke)

(1) $\mathcal{C} = \emptyset$;
(2) **for** $(p = 1, \ldots, m)$ **{**
(3) **while** (es gibt Task $T_{(j,p)}$, die in keinem Block von $\mathcal{C}$ enthalten ist) **{**
(4) wähle eine solche Task $T_{(j,p)}$ mit minimalem j;
(5) bestimme maximales $j' \geq j$, so dass $\Phi_p^{[j,j']}$ gültig
und mit allen Blöcken aus $\mathcal{C}$ verträglich ist;
(6) $\mathcal{C} = \mathcal{C} \cup \{\Phi_p^{[j,j']}\}$;
(7) **}**
(8) **}**

	P_1	P_2	P_3
J_1	0	0	1
J_2	0	1	1
J_3	0	1	1
J_4	0	1	1
J_5	1	0	1
J_6	1	0	0
J_7	1	0	0

	P_1	P_2	P_3
J_1	0	0	1
J_2	0	1	1
J_3	0	1	1
J_4	0	1	1
J_5	1	0	1
J_6	1	0	0
J_7	1	0	0

Abbildung 4.4: Taskblock $\Phi_3^{[3,5]}$ aus der Top-Down-Überdeckung gültiger Blöcke (links) wird von Algorithmus 3 zweifach unterteilt (rechts).

Beweis Wir zeigen, dass für jede Maschine P_p die Anzahl der Blöcke an der Maschine P_p in der berechneten Überdeckung höchstens zweimal so groß ist wie die Anzahl $\mathrm{LB}_p^{\mathrm{E}}(\mathcal{M})$ der Blöcke an dieser Maschine in der Top-Down-Überdeckung mit gültigen Blöcken. Seien dazu $B_1, \ldots, B_k$, $k = \mathrm{LB}_p^{\mathrm{E}}(\mathcal{M})$, die Blöcke der Top-Down-Überdeckung gültiger Blöcke und $B'_1, \ldots, B'_r$ die vom Algorithmus berechneten Blöcke an dieser Maschine, jeweils aufsteigend sortiert nach den Jobindizes der ersten Tasks der Blöcke.

Wir behaupten, dass für jedes $j \in \{1, \ldots, k\}$ höchstens die ersten $2j$ der Blöcke $B'_1, \ldots, B'_r$ benötigt werden, um alle Tasks aus den Blöcken $B_1, \ldots, B_j$ zu überdecken. Aus dieser Behauptung folgt unmittelbar der Satz. Angenommen, die Behauptung trifft nicht zu. Dann sei j minimal, so dass die Behauptung für dieses j nicht gilt. Nach Wahl von j werden dann die Blöcke $B_1, \ldots, B_{j-1}$ von den Blöcken $B'_1, \ldots, B'_{2j-2}$ überdeckt. Aus Lemma 4.2 und Lemma 4.3 folgt, dass B_j so in zwei Blöcke $B_{j,1}$ und $B_{j,2}$ aufgeteilt werden kann, dass beide Blöcke mit allen bereits vom Algorithmus berechneten Blöcken auf Maschinen P_q, $q < p$, verträglich sind. Da dies dann auch für alle Teilblöcke von $B_{j,1}$ und von $B_{j,2}$ gilt und da der Algorithmus 'top-down' maximale verträgliche Blöcke berechnet, muss $B_{j,1}$ von $B'_1, \ldots, B'_{2j-1}$ und $B_{j,2}$ von $B'_1, \ldots, B'_{2j}$ überdeckt werden. Daraus ergibt sich ein Widerspruch zu unserer Annahme, dass $B_1, \ldots, B_j$ nicht von $B'_1, \ldots, B'_{2j}$ überdeckt wird. □

Man beachte, dass es vorkommen kann, dass Algorithmus 3 für die Überdeckung eines Blocks aus der Top-Down-Überdeckung gültiger Blöcke drei Blöcke benötigt, siehe Abbildung 4.4. In den meisten Fällen dürften die berechneten Blocküberdeckungen jedoch weniger Blöcke aufweisen als beim einfachen Aufteilen der nicht verträglichen Blöcke. Allerdings hat auch dieser Algorithmus keine Güte < 2. Um dies zu sehen, betrachten wir die Taskmatrizen $\mathcal{M}_{k,r}$, die für alle ganzen Zahlen $k \geq 3$, $r \geq 1$ wie folgt definiert sind. $\mathcal{M}_{k,r}$ hat $k + r(2k-2)$ Maschinen und $k(2k-1)$ Jobs. Jeder Job hat Tasks an genau den Maschinen P_1, P_k und $P_{k+i(2k-2)}$, $i = 1, \ldots, r$. Das obere Bild in Abbildung 4.5 zeigt die Taskmatrix $\mathcal{M}_{3,4}$ mit der Top-Down-Überdeckung verträglicher Blöcke nach Algorithmus 3. Es ist leicht zu sehen, dass der Algorithmus für die Taskmatrix $\mathcal{M}_{k,r}$ an jeder Maschine, die Tasks besitzt, $2k-1$ Blöcke der Länge k liefert. Somit enthält die Überdeckung insge-

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}	P_{18}	P_{19}
J_1	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_2	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_3	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_4	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_5	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_6	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_7	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_8	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_9	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_{10}	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_{11}	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_{12}	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_{13}	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_{14}	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_{15}	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}	P_{18}	P_{19}
J_1	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_2	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_3	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_4	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_5	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_6	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_7	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_8	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_9	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_{10}	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_{11}	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_{12}	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_{13}	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_{14}	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1
J_{15}	1	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1

Abbildung 4.5: Top-Down-Überdeckung verträglicher Blöcke nach Algorithmus 3 (oberes Bild) und minimale gültige Überdeckung (unteres Bild) der Taskmatrix $\mathcal{M}_{3,4}$.

samt $(r+2)(2k-1)$ Blöcke. Andererseits erhält man eine bessere gültige Blocküberdeckung, wenn man die Tasks an den Maschinen $P_{k+i(2k-2)}$, $i = 1, \ldots, r$, jeweils durch k Blöcke der Länge $2k-1$ überdeckt und die Tasks an den Maschinen P_1 und P_k jeweils abwechselnd durch Blöcke der Länge k und $k-1$, siehe unteres Bild in Abbildung 4.5. Auf diese Weise erhält man eine gültige Blocküberdeckung mit $rk + 2(2k) = (r+4)k$ Blöcken. Für $r \to \infty$ strebt das Verhältnis dieser beiden Lösungen gegen den Wert $\frac{2k-1}{k}$. Da $k \geq 3$ beliebig ist, zeigt dies, dass die Güte von Algorithmus 3 nicht kleiner als 2 ist.

Wir wollen nun eine Implementierung von Algorithmus 3 mit einer Laufzeit von $O(nm)$ angeben. Für den Beweis der Korrektheit benötigen wir das folgende Lemma.

Lemma 4.4 *Seien $B_1 = \Phi_{p_1}^{[s_1,t_1]}$ und $B_2 = \Phi_{p_2}^{[s_2,t_2]}$ zwei gültige, unverträgliche Blöcke mit $p_1 < p_2$. Dann existiert keine Task $T_{(s_1,q)}$ mit $p_1 < q < p_2$.*

Beweis Nach Lemma 4.1 wissen wir, dass $s_2 < s_1 \leq t_2$ und $s_2 + p_2 < t_1 + p_1$ gilt. Angenommen, es gäbe eine Task $T_{(s_1,q)}$ mit $p_1 < q < p_2$. Falls $s_1 + q < t_1 + p_1$ gilt, so folgt $T_{(s_1,p_1)} \prec T_{(s_1,q)} \prec T_{(t_1,p_1)}$ und damit die Ungültigkeit von Block B_1. Andernfalls gilt $s_2 + p_2 < t_1 + p_1 \leq s_1 + q$ und es folgt $T_{(s_2,p_2)} \prec T_{(s_1,q)} \prec T_{(t_2,p_2)}$ und damit die Ungültigkeit von B_2. □

Man betrachte nun Algorithmus 4 auf der nächsten Seite, der neben den in Abschnitt 2.3 definierten Feldern noch ein Feld $E[p][j]$ benutzt. Dieses Feld gibt für den Fall, dass bereits ein Block $\Phi_p^{[j,k]}$ für ein $k \geq j$ definiert ist, das Blockende k an. Andernfalls ist $E[p][j] = -\infty$.

Satz 4.2 *Algorithmus 4 berechnet die Top-Down-Überdeckung verträglicher Blöcke in Zeit $O(nm)$.*

Beweis Es ist offensichtlich, dass der Algorithmus eine Laufzeit in $O(nm)$ besitzt, da er die Taskmatrix genau einmal Spalte für Spalte durchläuft und die Felder 'FirstJob[.]', 'NextJob[.][.]', 'NextMachine[.][.]' und 'PrevMachine[.][.]' in einem Vorverarbeitungsschritt in Zeit $O(nm)$ berechnet werden können.

Es ist ferner leicht zu sehen, dass der Algorithmus die Variable 'gültig' in Zeile (14) genau dann auf 'false' setzt, wenn es eine Nachfolgertask von $T_{(\text{blockstart},p)}$ gibt, die eine Vorgängertask jeder Task $T_{(k',p)}$ mit $k' \geq k$ ist, und somit das bisher gefundene Blockende bereits den maximalen gültigen Block definiert. Genauso leicht sieht man, dass in dem Fall, in dem der Algorithmus die Variable 'verträglich' auf 'false' setzt, jeder Block $\Phi_p^{[\text{blockstart},t]}$ mit $t \geq k$ unverträglich mit dem bereits definierten Block $\Phi_q^{[k,E[q][k]]}$ wäre. Da der Algorithmus nur gültige Blöcke erzeugt, kann er andererseits keinen Block $\Phi_p^{[s,t]}$ erzeugen, der unverträglich mit einem bereits berechneten Block $\Phi_q^{[s',t']}$ mit $q < p$ ist. Nach Lemma 4.1 ist dann nämlich $s < s' \leq t$ und der Algorithmus würde wegen Lemma 4.4 in der Zeile (15) die Unverträglichkeit der Blöcke feststellen. □

Eine Abarbeitung zu der Top-Down-Überdeckung verträglicher Blöcke lässt sich ebenfalls in Zeit $O(nm)$ erzeugen. Man betrachte dazu den folgenden Algorithmus, der entsprechend den Beweisen von Lemma 3.3 bzw. Satz 3.3 aus einer gegebenen gültigen Blocküber-

Algorithmus 4 (Top-Down-Überdeckung verträglicher Blöcke)

```
(1)  C = ∅;
(2)  Setze E[p][j] = −∞ für alle Maschinenindizes p und alle Jobindizes j;
(3)  for (p = 1, ..., m) {
(4)      blockstart = FirstJob[p];
(5)      while (blockstart ≤ n) {
(6)          rdiag = blockstart + NextMachine[blockstart][p];
(7)          blockende = blockstart;
(8)          gültig = 'true';
(9)          verträglich = 'true';
(10)         k = blockstart + 1;
(11)         while (k ≤ n && gültig && verträglich) {
(12)             q = PrevMachine[k][p];
(13)             if (k + p > rdiag || k + q > blockstart + p)
(14)                 gültig = 'false';
(15)             else if (q ≥ 1 && q + E[q][k] > blockstart + p)
(16)                 verträglich = 'false';
(17)             else {
(18)                 if (k + NextMachine[k][p] < rdiag)
(19)                     rdiag := k + NextMachine[k][p];
(20)                 if (T_(k,p) ist Task)
(21)                     blockende := k;
(22)                 k := k + 1;
(23)             }
(24)         }
(25)         C := C ∪ Φ_p^[blockstart,blockende];
(26)         E[p][blockstart] := blockende;
(27)         blockstart := NextJob[p][blockende];
(28)     } /* while (blockstart ≤ n) */
(29) } /* for (p = 1, ..., m) */
```

deckung eine Abarbeitung erzeugt, indem er sukzessive jeweils den freien Block $\Phi_p^{[j,j']}$ mit minimalem Index j entfernt. Die Blöcke ergeben dann in der Reihenfolge, in der sie entfernt werden, eine Abarbeitung der Taskmatrix. Der Algorithmus kennzeichnet die Blöcke in einem Feld $B[1\ldots m][1\ldots n]$ wie folgt. Für jeden Block $\Phi_p^{[j,j']}$ wird $B[p][j], B[p][j+1], \ldots, B[p][j'-1]$ auf j' und $B[p][j']$ auf j gesetzt. Alle anderen Einträge sind 0. Die Task $T_{(j,p)}$ ist also genau dann die letzte Task in einem Block, wenn $B[p][j] \neq 0$ und $B[p][j] \leq j$ gilt. Der Algorithmus durchsucht die Taskmatrix zeilenweise nach dem freien Block $\Phi_p^{[j,j']}$ mit minimalem j. In einem Feld $s[.]$ wird die aktuelle Maschinenposition für jeden Zeile

gespeichert. Trifft der Algorithmus in der Zeile j auf einen Block, der nicht in dieser Zeile endet, so fährt er in der Zeile $j+1$ an der Position $s[j+1]$ fort. Trifft er in der Zeile j' auf das Ende eines Blocks $\Phi_p^{[j,j']}$, so wird der Block ausgegeben, die Positionen $s[j], s[j+1], \ldots, s[j']$ werden auf den Wert $p+1$ gesetzt und es wird in der Zeile j an der Stelle $s[j] = p+1$ fortgefahren.

Algorithmus 5 (Abarbeitung aus gültiger Überdeckung $\mathcal{C}$)

```
(1)  Λ = ∅;
(2)  B[p][j] = 0 für alle Maschinenindizes p und alle Jobindizes j;
(3)  s[j] = 1 für alle Jobindizes j;
(4)  for all (Φ_p^[j,j'] ∈ C) { /* Blöcke in B[.][.] markieren */
(5)      for (k = j; k < j'; k++) B[p][k] := j';
(6)      B[p][j'] = j;
(7)  }
(8)  job = 1;
(9)  while (job ≤ n) {
(10)     p = s[job];
(11)     while (p ≤ m && B[p][job] = 0) p = p + 1;
(12)     s[job] = p;
(13)     if (p > m || B[p][job] > job)
(14)         job = job + 1;
(15)     else { /* Blockende eines freien Blocks gefunden */
(16)         j = B[p][job];
(17)         Λ = (Λ, Φ_p^[j,job]);
(18)         for (k = j; k ≤ job; k++) s[k] = p + 1;
(19)         job = j;
(20)     }
(21) } /* while (job ≤ n) */
```

Es ist leicht zu sehen, dass der Algorithmus nacheinander die freien Blöcke $\Phi_p^{[j,j']}$ mit minimalem j (oder gleichbedeutend minimalem j') entfernt. Nach den Beweisen von Lemma 3.3 bzw. Satz 3.3 erzeugt der Algorithmus daher eine Abarbeitung Λ, deren Einheitsdistanz höchstens so groß ist wie die Anzahl der Blöcke in der Blocküberdeckung. Es ist nicht schwer zu sehen, dass die Laufzeit des Algorithmus in $O(nm)$ liegt. Wir haben daher den folgenden Satz.

Satz 4.3 *Eine Abarbeitung einer Taskmatrix $\mathcal{M}$, deren Einheitsdistanz höchstens das Zweifache der unteren Schranke $LB^E(\mathcal{M})$ und somit höchstens das Zweifache der Distanz einer minimalen Abarbeitung beträgt, kann in Zeit $O(nm)$ berechnet werden.*

4.3 Eine untere Schranke für additive Distanzen

Die Distanzmatrix $\mathcal{D} = (\delta_{p,q})_{0 \le p,q \le m}$ beschreibt in diesem Abschnitt immer additive Distanzen. Um die nachfolgenden Definitionen und Beweise zu erleichtern, fügen wir bei additiven Distanzen der Taskmenge $\mathcal{T}(\mathcal{M})$ eine zusätzliche spezielle *Starttask* $T_{(0,0)}$ und eine zusätzliche spezielle *Endtask* $T_{(\infty,0)}$ hinzu. Da für jede andere Task T nach Definition der Vorgängerrelation $T_{(0,0)} \prec T$ und $T \prec T_{(\infty,0)}$ gilt, bewirkt dies lediglich, dass jede Teilabarbeitung mit der speziellen Starttask $T_{(0,0)}$ beginnt und dass jede Abarbeitung mit der speziellen Endtask $T_{(\infty,0)}$ endet. Die Distanz einer Abarbeitung $\Lambda = (T_{(j_1,p_1)}, \ldots, T_{(j_r,p_r)})$ der Taskmatrix verändert sich durch diese Modifikation nicht, lässt sich aber vereinfachend durch

$$\Delta(\Lambda) = \sum_{i=1}^{r-1} \delta_{p_i,p_{i+1}}$$

angeben. In Beispielen und Abbildungen werden wir diese speziellen Tasks in der Regel vernachlässigen.

Sei $1 \le s \le m-1$ ein Maschinenindex. Eine Task $T_{(j,p)}$ *liegt links von* P_s, falls $p \le s$, und *rechts von* P_s andernfalls. Zwei Tasks T_1, T_2 *liegen auf der gleichen Seite von* P_s, wenn beide links von P_s oder beide rechts von P_s liegen. Andernfalls *liegen sie auf entgegengesetzten Seiten von* P_s. Man betrachte eine Teilabarbeitung $\Lambda = (T_1, T_2, \ldots, T_r)$ von $\mathcal{M}$. Wir nennen jedes Paar (T_i, T_{i+1}) aufeinander folgender Tasks in der Teilabarbeitung eine *Bewegung*. Die Bewegung heißt eine *Rechts-Bewegung* bzw. *Links-Bewegung*, falls der Maschinenindex von T_{i+1} größer bzw. kleiner als der Maschinenindex von T_i ist. Falls T_i und T_{i+1} auf entgegengesetzten Seiten von P_s liegen, so nennen wir die Bewegung eine *P_s-Bewegung*. Wir sprechen von einer *Rechts-P_s-Bewegung*, falls T_{i+1} rechts von P_s liegt, und von einer *Links-P_s-Bewegung*, falls T_{i+1} links von P_s liegt.

Für $s = 1, \ldots, m-1$ bezeichnen wir mit $R_s(\Lambda)$ die Anzahl der Rechts-P_s-Bewegungen in der Teilabarbeitung Λ. Ist Λ eine Abarbeitung, so ist die erste Task $T_1 = T_{(0,0)}$ und die letzte Task $T_r = T_{(\infty,0)}$. Somit muss die Anzahl der Links-P_s-Bewegungen der Abarbeitung Λ gleich der Anzahl der Rechts-P_s-Bewegungen sein. Da bei additiven Distanzen für $p < q$ die Entfernung $\delta_{p,q}$ von Maschine P_p zu Maschine P_q gleich der Summe der Entfernungen $\delta_{p,p+1} + \delta_{p+1,p+2} + \ldots, +\delta_{q-1,q}$ ist, folgt wegen der Symmetrie $\delta_{p,q} = \delta_{q,p}$ und wegen $\delta_{0,1} = 0$, dass die Distanz der Abarbeitung Λ durch

$$\Delta(\Lambda) = 2 \sum_{s=1}^{m-1} \delta_{s,s+1} R_s(\Lambda) \tag{4.2}$$

gegeben ist. Wir wollen nun eine untere Schranke für die Anzahl $R_s(\Lambda)$ der Rechts-P_s-Bewegungen in einer Abarbeitung Λ der Taskmatrix bestimmen, um damit aus Gleichung (4.2) eine untere Schranke für die Distanz einer Abarbeitung zu gewinnen.

Eine Folge $T_1, \ldots, T_r$ von Tasks heißt *P_s-alternierend*, wenn die Tasks der Folge abwechselnd links und rechts von P_s liegen. Gilt für eine P_s-alternierende Folge $T_1, \ldots, T_r$, dass $T_1 \prec \ldots \prec T_r$, so ist die Anzahl der Tasks in der Folge, die rechts von P_s liegen,

	P_1	P_2	P_3	P_4	P_5	P_6	P_7
J_1	0	0	0	0	0	0	0
J_2	0	0	0	0	0	**1**	0
J_3	0	0	0	0	0	0	0
J_4	0	0	0	0	0	0	0
J_5	0	0	0	0	0	0	0
J_6	0	0	0	0	0	0	0
J_7	0	0	0	0	0	0	0

	P_1	P_2	P_3	P_4	P_5	P_6	P_7
J_1	0	0	0	0	0	0	0
J_2	0	0	0	0	0	0	0
J_3	0	0	0	0	0	0	0
J_4	0	0	0	0	0	0	0
J_5	0	**1**	0	0	0	0	0
J_6	0	0	0	0	0	0	0
J_7	0	0	0	0	0	0	0

Abbildung 4.6: $R_2^\star$-Task $T_{(2,6)}$ (links) und $L_5^\star$-Task $T_{(5,2)}$ (rechts). Die grau unterlegten Bereiche dürfen keine Tasks enthalten.

offensichtlich eine untere Schranke für die Anzahl der Rechts-P_s-Bewegungen in einer Abarbeitung der Taskmatrix. Um eine möglichst gute untere Schranke zu erhalten, suchen wir natürlich eine solche Folge, die eine maximale Anzahl von Tasks enthält.

Sei $T = T_{(j,p)}$ eine Task von $\mathcal{M}$ und sei s, $1 \leq s \leq m-1$ ein Maschinenindex.

1. Wir nennen T eine *$R_s^\star$-Task* von $\mathcal{M}$, falls T rechts von P_s liegt und es außer T keine weitere Task $T' = T_{(j',p')} \in \mathcal{T}(\mathcal{M})$ rechts von P_s mit $j' \geq j$ und $j' + p' \leq j + p$ gibt.

2. Wir nennen T eine *$L_s^\star$-Task* von $\mathcal{M}$, falls T links von P_s liegt und es außer T keine weitere Task $T' = T_{(j',p')} \in \mathcal{T}(\mathcal{M})$ links von P_s mit $j' \leq j$ und $j' + p' \geq j + p$ gibt.

Abbildung 4.6 verdeutlicht die obige Definition. Die speziellen Tasks $T_{(0,0)}$ und $T_{(\infty,0)}$ sind nach der Definition für jedes $s \in \{1, \ldots, m-1\}$ $L_s^\star$-Tasks.

Seien $T_1 = T_{(j_1,p_1)}, T_2 = T_{(j_2,p_2)}$ Tasks von $\mathcal{M}$ und sei $1 \leq s \leq m-1$.

1. Wir nennen (T_1, T_2) ein *LR_s-Paar von $\mathcal{M}$*, wenn die folgenden beiden Bedingungen erfüllt sind:

   (a) T_1 ist eine $L_s^\star$-Task, T_2 ist eine $R_s^\star$-Task und $T_1 \prec T_2$.

   (b) Es gibt keine Task $T = T_{(j,p)}$ rechts von P_s mit $T_1 \prec T$ und $j + p < j_2 + p_2$.

2. Wir nennen (T_1, T_2) ein *RL_s-Paar von $\mathcal{M}$*, falls die folgenden beiden Bedingungen erfüllt sind:

   (a) T_1 ist eine $R_s^\star$-Task, T_2 ist eine $L_s^\star$-Task und $T_1 \prec T_2$.

   (b) Es gibt keine Task $T = T_{(j,p)}$ links von P_s mit $T_1 \prec T$ und $j < j_2$.

Abbildung 4.7 verdeutlicht die obige Definition. Wenn T eine $L_s^\star$-Task ist, die einen Nachfolger rechts von P_s besitzt, so gibt es offensichtlich eine eindeutig bestimmte Task T'

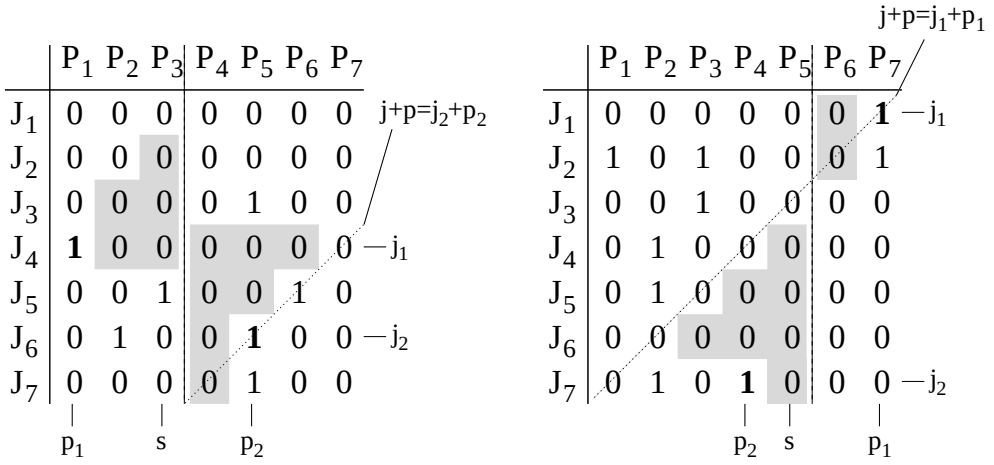


Abbildung 4.7: LR_3-Paar $(T_{(4,1)}, T_{(6,5)})$ (links) und RL_5-Paar $(T_{(1,7)}, T_{(7,4)})$ (rechts). Die grau unterlegten Bereiche dürfen keine Tasks enthalten.

rechts von P_s, so dass (T, T') ein LR_s-Paar ist. $T' = T_{(j',p')}$ ist die Nachfolgertask von T rechts von P_s, für die $j'+p'$ minimal ist, und falls mehrere solche Tasks existieren, diejenige von diesen mit minimalem p'. Ist T eine $\mathrm{R}_s^\star$-Task, die einen Nachfolger links von P_s besitzt, so gibt es offensichtlich eine eindeutig bestimmte Task T' links von P_s, so dass (T, T') ein RL_s-Paar ist. $T' = T_{(j',p')}$ ist die Nachfolgertask von T links von P_s, für die j' minimal ist, und falls mehrere solche Tasks existieren, diejenige von diesen mit maximalem p'.

Für $s \in \{1, \ldots, m-1\}$ nennen wir eine Folge $T_1, \ldots, T_r$ von Tasks aus $\mathcal{T}(\mathcal{M})$ eine *s-Kette von $\mathcal{M}$*, falls die folgenden Bedingungen gelten:

1. T_1 ist eine $\mathrm{L}_s^\star$-Task oder eine $\mathrm{R}_s^\star$-Task.

2. Für $i = 1, \ldots, r-1$ ist T_{i+1} die eindeutig bestimmte Task, so dass (T_i, T_{i+1}) ein LR_s-Paar bzw. ein RL_s-Paar ist.

Wir nennen die eindeutig bestimmte s-Kette $T_1, \ldots, T_r$ mit der maximalen Anzahl von Tasks, die mit der $\mathrm{L}_s^\star$-Task $T_1 = T_{(0,0)}$ beginnt, die *vollständige s-Kette* von $\mathcal{M}$. Da die Endtask $T_{(\infty,0)}$ ein Nachfolger aller anderen Tasks ist, hat die vollständige s-Kette immer eine ungerade Anzahl von Tasks. Die letzte Task in der Kette liegt links von P_s und besitzt keinen Nachfolger rechts von P_s. Es muss sich dabei aber nicht unbedingt um die Endtask $T_{(\infty,0)}$ handeln. Offensichtlich lässt sich die vollständige s-Kette einfach bestimmen. Wir bezeichnen die Anzahl der Tasks in der vollständigen s-Kette von $\mathcal{M}$, die rechts von P_s liegen, mit $R_s^{\min}(\mathcal{M})$. Natürlich ist $R_s^{\min}(\mathcal{M})$ eine untere Schranke für die Anzahl der Rechts-P_s-Bewegungen in jeder Abarbeitung der Taskmatrix. Somit ist

$$\mathrm{LB}^{\mathrm{A}}(\mathcal{M}, \mathcal{D}) := 2 \sum_{s=1}^{m-1} \delta_{s,s+1} R_s^{\min}(\mathcal{M}) \tag{4.3}$$

eine untere Schranke für die Distanz jeder Abarbeitung von $\mathcal{M}$ bezüglich der additiven Distanzmatrix $\mathcal{D} = (\delta_{p,q})$. Im Fall linearer Distanzen bezeichnen wir die untere Schranke mit $\mathrm{LB}^{\mathrm{lin}}(\mathcal{M})$. Diese ergibt sich dann vereinfacht durch

$$\mathrm{LB}^{\mathrm{lin}}(\mathcal{M}) := 2 \sum_{s=1}^{m-1} R_s^{\min}(\mathcal{M}) \tag{4.4}$$

Wir wollen noch kurz zeigen, dass die untere Schranke $R_s^{\min}(\mathcal{M})$ scharf ist, d.h., dass es immer eine Abarbeitung mit genau dieser Anzahl von Rechts-P_s-Bewegungen gibt. Sei $\Gamma_s = (T_1, \ldots, T_r)$ mit $T_i = T_{(j_i,p_i)}$ die vollständige s-Kette von $\mathcal{M}$ für ein $s \in \{1, \ldots, m-1\}$. Wir zeigen durch Induktion, dass die folgende Aussage für $k = 1, \ldots, r$ gilt:

Es gibt eine Teilabarbeitung $\Lambda_k = (T'_1, \ldots, T'_{l_k})$ von $\mathcal{M}$ mit $T'_{l_k} = T_k$, die genau $k-1$ P_s-Bewegungen aufweist und folgende Eigenschaft erfüllt:

1. Falls k ungerade ist, so sind alle Tasks $T_{(j,p)}$ rechts von P_s mit $j < j_k$ in Λ_k enthalten.
2. Falls k gerade ist, so sind alle Tasks $T_{(j,p)}$ links von P_s mit $j + p \leq j_k + p_k$ in Λ_k enthalten.

<u>$k = 1$:</u> Wegen $T_1 = T_{(0,0)}$ erfüllt $\Lambda_1 = (T_1)$ offensichtlich die Behauptung.

<u>$1 < k \leq r$:</u> Wir unterscheiden zwei Fälle.

1. Sei k gerade. Dann ist (T_{k-1}, T_k) ein LR_s-Paar. Nach Induktionsvoraussetzung gibt es eine Teilabarbeitung $\Lambda_{k-1} = (T'_1, \ldots, T'_{l_{k-1}})$ mit $T'_{l_{k-1}} = T_{k-1}$ und genau $k-2$ P_s-Bewegungen, so dass Λ_{k-1} alle Tasks $T_{(j,p)}$ rechts von P_s mit $j < j_{k-1}$ enthält. Da (T_{k-1}, T_k) ein LR_s-Paar ist, existieren keine Tasks $T_{(j,p)}$ rechts von P_s mit $j \geq j_{k-1}$ und $j + p < j_k + p_k$. Dann existieren aber für Tasks $T_{(j,p)}$ links von P_s mit $j + p \leq j_k + p_k$ offensichtlich keine Vorgänger rechts von P_s, die nicht in Λ_{k-1} enthalten sind. Daher können wir Λ_{k-1} mit nur einer zusätzlichen P_s-Bewegung zu der gewünschten Teilabarbeitung Λ_k fortsetzen.
2. Sei k ungerade. Dann ist (T_{k-1}, T_k) ein RL_s-Paar. Nach Induktionsvoraussetzung gibt es eine Teilabarbeitung $\Lambda_{k-1} = (T'_1, \ldots, T'_{l_{k-1}})$ mit $T'_{l_{k-1}} = T_{k-1}$ und genau $k-2$ P_s-Bewegungen, so dass Λ_{k-1} alle Tasks $T_{(j,p)}$ links von P_s mit $j + p \leq j_{k-1} + p_{k-1}$ enthält. Da (T_{k-1}, T_k) ein RL_s-Paar ist, existieren keine Tasks $T_{(j,p)}$ links von P_s mit $j + p > j_{k-1} + p_{k-1}$ und $j < j_k$. Dann existieren aber für Tasks $T_{(j,p)}$ rechts von P_s mit $j < j_k$ offensichtlich keine Vorgänger links von P_s, die nicht in Λ_{k-1} enthalten sind. Daher können wir Λ_{k-1} mit nur einer zusätzlichen P_s-Bewegung zu der gewünschten Teilabarbeitung Λ_k fortsetzen.

Da r ungerade ist, folgt aus der Induktionsbehauptung für $k = r$, dass es eine Teilabarbeitung $\Lambda_r = (T'_1, \ldots, T'_{l_r})$ mit $T'_{l_r} = T_r$ und genau $r-1$ P_s-Bewegungen gibt, so dass Λ_r alle Tasks $T_{(j,p)}$ rechts von P_s mit $j < j_r$ bereits enthält. Da T_r links von P_s liegt und Γ_s die vollständige s-Kette ist, gibt es keine Tasks $T_{(j,p)}$ rechts von P_s mit $j \geq j_r$. Daher kann Λ_r ohne weitere P_s-Bewegung zu einer Abarbeitung fortgesetzt werden und hat dann genau $R_s^{\min}(\mathcal{M})$ Rechts-P_s-Bewegungen.

4.4 Ein Approximationsalgorithmus für additive Distanzen

Wie im vorigen Abschnitt beschreibt die Distanzmatrix $\mathcal{D}$ auch in diesem Abschnitt immer additive Distanzen und wir fügen der Taskmenge wieder die speziellen Tasks $T_{(0,0)}$ und $T_{(\infty,0)}$ hinzu. Wir werden einen Approximationsalgorithmus angeben, der eine Abarbeitung von $\mathcal{M}$ berechnet, deren Distanz höchstens das Dreifache der unteren Schranke $\mathrm{LB}^{\mathrm{A}}(\mathcal{M}, \mathcal{D})$ ist.

Man betrachte eine beliebige Abarbeitung $\Lambda = (T_1, T_2, \ldots, T_r)$ der Taskmatrix $\mathcal{M}$ und einen Maschinenindex $1 \leq s \leq m-1$. Wir unterteilen Λ in maximale nicht-leere Folgen

$$L_0, R_1, L_1, R_2, L_2, \ldots, R_t, L_t,$$

so dass die Teilfolgen L_i, $i = 0, \ldots, t$, nur Tasks links von P_s und die Teilfolgen R_i, $i = 1, \ldots, t$, nur Tasks rechts von P_s enthalten. Wir nennen eine solche Aufteilung eine *RL_s-Aufteilung von* Λ. Man beachte, dass aufgrund der Tasks $T_{(0,0)}$ und $T_{(\infty,0)}$ die erste und die letzte Task der Abarbeitung immer links von P_s liegen. Wir werden den Begriff RL_s-Aufteilung jedoch auch auf Teilfolgen von Abarbeitungen anwenden. In diesem Fall kann die Aufteilung dann auch rechts von P_s beginnen bzw. enden.

Betrachte nun die obige RL_s-Aufteilung und ein $k \in \{1, \ldots, t-1\}$. Falls die Teilfolge R_k keinen Vorgänger einer Task aus L_k enthält, so stellt die Taskfolge, die sich aus der Vertauschung von R_k und L_k ergibt, nach Lemma 2.3 ebenfalls eine Abarbeitung dar. Genauso ist für den Fall, dass die Teilfolge L_k keinen Vorgänger einer Task aus R_{k+1} enthält, die Taskfolge, die aus der Vertauschung von L_k und R_{k+1} resultiert, eine Abarbeitung. Wir nennen jede dieser Vertauschungen einen *RL_s-Austausch*. Man beachte, dass L_0 und L_t nie an einem RL_s-Austausch beteiligt sind. In Abbildung 4.8 ist ein solcher RL_s-Austausch gezeigt. In der Abbildung ist die Abarbeitung durch eine Linie in der Taskmatrix dargestellt, welche die Bewegungen des Arbeiters kennzeichnet. Wir werden diese Darstellung einer Abarbeitung im Folgenden immer wieder benutzen.

Durch einen RL_s-Austausch wird die Distanz der Abarbeitung verringert, da die Anzahl der Rechts-P_s-Bewegungen sich dabei um eins verringert und die Anzahl der Rechts-$P_{s'}$-Bewegungen für $s' \neq s$ sich nicht erhöht. Um dies zu sehen, betrachten wir zunächst den Fall, in dem R_k mit L_k vertauscht wird. Dann ändert sich die Abarbeitung von

$$L_0, R_1, L_1, \ldots, L_{k-1}, R_k, L_k, R_{k+1}, \ldots, R_t, L_t,$$

durch den RL_s-Austausch zur Abarbeitung

$$L_0, R_1, L_1, \ldots, L_{k-1}, L_k, R_k, R_{k+1}, \ldots, R_t, L_t.$$

Es ist offensichtlich, dass sich die Anzahl der Rechts-P_s-Bewegungen dabei um genau eins verringert. Um zu sehen, dass sich die Anzahl der Rechts-$P_{s'}$-Bewegungen für $s' \neq s$ nicht erhöht, betrachten wir die Bewegungen, die in beiden Abarbeitungen unterschiedlich sind.

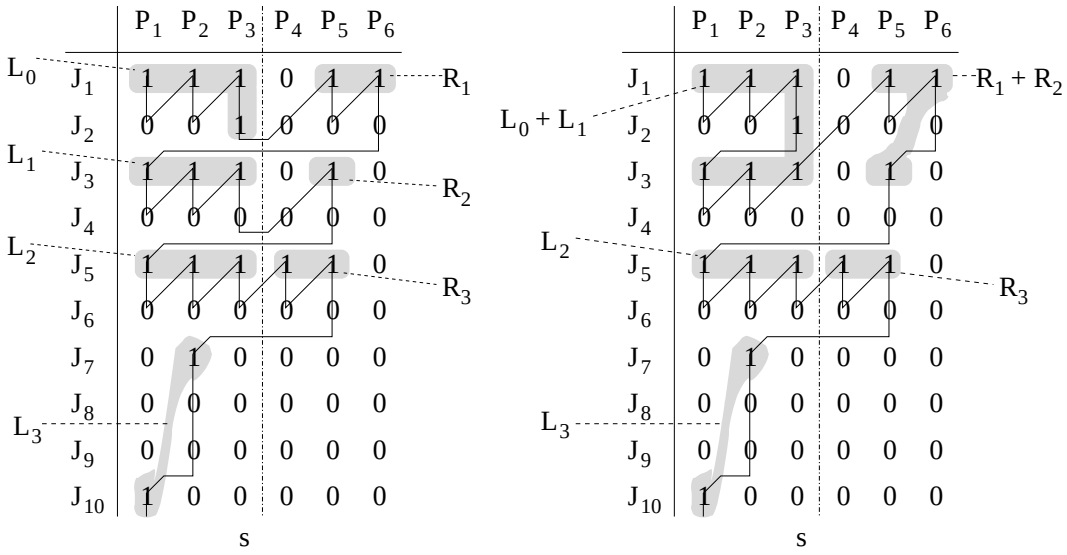


Abbildung 4.8: RL_3-Austausch von R_1 und L_1. Die Linie in der Taskmatrix $\mathcal{M} = (\mu_{j,p})$ zeigt die Bewegungen des Arbeiters bei der Abarbeitung. Ein vertikaler Verlauf zeigt an, dass der Arbeiter an der Maschine verbleibt, während die Jobs soweit wie möglich aufrücken. Ein diagonaler Verlauf durch einen Eintrag $\mu_{j,p}$ bedeutet, dass sich der Arbeiter an Maschine P_p vorbeibewegt, während diese von Job J_j belegt ist. Ein horizontaler Verlauf zwischen den Einträgen $\mu_{j,p}$ und $\mu_{j+1,p}$ besagt, dass der Arbeiter die Maschine P_p passiert, wobei der Job J_j diese Maschine bereits verlassen hat, während der Job J_{j+1} diese Maschine noch nicht erreicht hat.

Seien $T_e^{L_{k-1}}$, $T_e^{L_k}$ und $T_e^{R_k}$ die letzten Tasks von L_{k-1}, L_k bzw. R_k, und seien $T_a^{R_k}$, $T_a^{L_k}$ und $T_a^{R_{k+1}}$ die ersten Tasks von R_k, L_k bzw. R_{k+1}. Dann sind die drei Bewegungen

$$(T_e^{L_{k-1}}, T_a^{R_k}), (T_e^{R_k}, T_a^{L_k}), (T_e^{L_k}, T_a^{R_{k+1}}) \tag{4.5}$$

der ersten Abarbeitung durch die drei Bewegungen

$$(T_e^{L_{k-1}}, T_a^{L_k}), (T_e^{L_k}, T_a^{R_k}), (T_e^{R_k}, T_a^{R_{k+1}}) \tag{4.6}$$

in der zweiten Abarbeitung ersetzt. Wir betrachten zunächst den Fall $1 \leq s' < s$. Ist die erste Bewegung aus (4.6) eine Rechts-$P_{s'}$-Bewegung, so trifft dies offensichtlich auch auf die erste Bewegung aus (4.5) zu. Ist die zweite Bewegung aus (4.6) eine Rechts-$P_{s'}$-Bewegung, so ist die dritte Bewegung aus (4.5) ebenfalls eine Rechts-$P_{s'}$-Bewegung. Die dritte Bewegung aus (4.6) kann wegen $s' < s$ keine Rechts-$P_{s'}$-Bewegung sein. Also hat sich die Anzahl der Rechts-$P_{s'}$-Bewegungen für $1 \leq s' < s$ durch den RL_s-Austausch nicht erhöht. Man betrachte nun den Fall $s < s' \leq m - 1$. Dann kann die erste Bewegung aus (4.6) keine Rechts-$P_{s'}$-Bewegung sein, und falls die zweite bzw. die dritte Bewegung aus (4.6) eine Rechts-$P_{s'}$-Bewegung ist, so ist offensichtlich auch die erste bzw. die dritte Bewegung aus (4.5) eine Rechts-$P_{s'}$-Bewegung. Also hat sich auch für $s < s' \leq m - 1$ die Anzahl der Rechts-$P_{s'}$-Bewegungen nicht erhöht.

Für den Fall des RL_s-Austausches von L_k und R_{k+1} ändert sich die Abarbeitung von

$$L_0, R_1, L_1, \ldots, R_k, L_k, R_{k+1}, L_{k+1}, \ldots, R_t, L_t,$$

durch den RL_s-Austausch zur Abarbeitung

$$L_0, R_1, L_1, \ldots, R_k, R_{k+1}, L_k, L_{k+1}, \ldots, R_t, L_t.$$

Es ist wieder offensichtlich, dass sich die Anzahl der Rechts-P_s-Bewegungen dabei um genau eins verringert. Um zu sehen, dass sich die Anzahl der Rechts-$P_{s'}$-Bewegungen für $s' \neq s$ nicht erhöht, betrachten wir wieder die Bewegungen, die in beiden Abarbeitungen unterschiedlich sind. Seien $T_e^{R_k}$, $T_e^{L_k}$ und $T_e^{R_{k+1}}$ die letzten Tasks von R_k, L_k bzw. R_{k+1}, und seien $T_a^{L_k}$, $T_a^{R_{k+1}}$ und $T_a^{L_{k+1}}$ die ersten Tasks von L_k, R_{k+1} bzw. L_{k+1}. Dann sind die drei Bewegungen

$$(T_e^{R_k}, T_a^{L_k}), (T_e^{L_k}, T_a^{R_{k+1}}), (T_e^{R_{k+1}}, T_a^{L_{k+1}}) \tag{4.7}$$

der ersten Abarbeitung durch die drei Bewegungen

$$(T_e^{R_k}, T_a^{R_{k+1}}), (T_e^{R_{k+1}}, T_a^{L_k}), (T_e^{L_k}, T_a^{L_{k+1}}) \tag{4.8}$$

in der zweiten Abarbeitung ersetzt. Für $1 \leq s' < s$ kann offensichtlich höchstens die dritte Bewegung aus (4.8) eine Rechts-$P_{s'}$-Bewegung sein. Ist dies der Fall, so trifft dies auch auf die zweite Bewegung aus (4.7) zu. Für $s < s' \leq m-1$ kann höchstens die erste Bewegung aus (4.8) eine Rechts-$P_{s'}$-Bewegung sein. Ist dies der Fall, so gilt dies offensichtlich ebenfalls für die zweite Bewegung aus (4.7). Also hat sich durch den RL_s-Austausch die Anzahl der Rechts-$P_{s'}$-Bewegungen für kein s' erhöht.

Unser Approximationsalgorithmus lässt sich nun einfach wie folgt formulieren:

Algorithmus 6 (Approximationsalgorithmus für additive Distanzen)

(1) Starte mit einer beliebigen Abarbeitung $\Lambda = (T_1, \ldots, T_r)$ von $\mathcal{M}$;
(z.B. Tasks lexikographisch nach (Jobindex, Maschinenindex) sortiert)
(2) **while** (RL_s-Austausch für ein $s \in \{1, \ldots, m-1\}$ möglich)
(3) führe RL_s-Austausch durch;

Es ist offensichtlich, dass der Algorithmus terminiert, da sich bei jedem RL_s-Austausch die Distanz der Abarbeitung verringert. Wir werden später zeigen, dass der Algorithmus mit einer Laufzeit von $O(nm)$, d.h. mit einer Laufzeit, die linear in der Größe der Taskmatrix ist, implementiert werden kann. Zunächst wollen wir jedoch die Güte des Algorithmus abschätzen. Dazu benötigen wir einige Lemmata. Das folgende Lemma ist eine einfache Folgerung aus den Definitionen eines LR_s- bzw. RL_s-Paares. Zur Veranschaulichung betrachte man auch Abbildung 4.9.

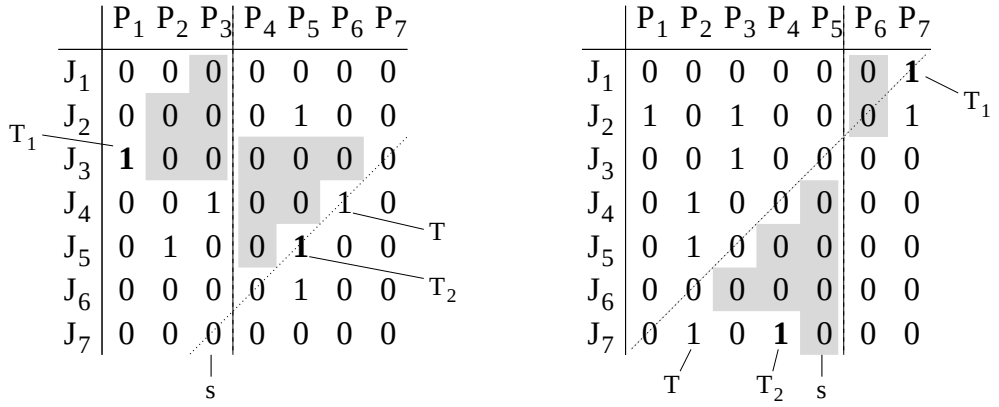


Abbildung 4.9: Zum Beweis von Lemma 4.5. Die grau unterlegten Bereiche enthalten aufgrund der Definition eines LR_s- bzw. RL_s-Paares keine Tasks.

Lemma 4.5 *Sei s, $1 \leq s \leq m-1$, ein Maschinenindex und seien $T_1 = T_{(j_1,p_1)}$, $T_2 = T_{(j_2,p_2)}$ und $T = T_{(j,p)}$ Tasks.*

1. *Falls (T_1, T_2) ein LR_s-Paar von $\mathcal{M}$ und T ein Nachfolger von T_1 ist, der rechts von P_s liegt, so gilt $j + p \geq j_2 + p_2$ und in jeder Abarbeitung von $\mathcal{M}$ sind alle Tasks $T' = T_{(j',p')}$ mit $p' > s$ und $j' + p' < j_2 + p_2$ bereits abgearbeitet, wenn die Task T bearbeitet wird.*

2. *Falls (T_1, T_2) ein RL_s-Paar von $\mathcal{M}$ und T ein Nachfolger von T_1 ist, der links von P_s liegt, so gilt $j \geq j_2$ und in jeder Abarbeitung von $\mathcal{M}$ sind alle Tasks $T' = T_{(j',p')}$ mit $p' \leq s$ und $j' < j_2$ bereits abgearbeitet, wenn die Task T bearbeitet wird.*

Beweis

1. Die Aussage $j + p \geq j_2 + p_2$ folgt direkt aus der Definition eines LR_s-Paares. Sei $T' = T_{(j',p')}$ mit $p' > s$ und $j'+p' < j_2+p_2$. Aus der Definition eines LR_s-Paares folgt, dass T' kein Nachfolger von T_1 ist. Also gilt $j' < j_1$. Wegen $T_1 \prec T$ ist andererseits $j_1 \leq j$ und somit $j' < j$. Wegen $j' + p' < j_2 + p_2 \leq j + p$ ist T' daher ein Vorgänger von T und muss somit in jeder Abarbeitung vor T ausgeführt werden.

2. Die Aussage $j \geq j_2$ folgt direkt aus der Definition eines RL_s-Paares. Sei $T' = T_{(j',p')}$ mit $p' \leq s$ und $j' < j_2$. Aus der Definition eines RL_s-Paares folgt, dass T' kein Nachfolger von T_1 ist. Also gilt $j' + p' \leq j_1 + p_1$. Wegen $T_1 \prec T$ ist andererseits $j_1 + p_1 < j + p$ und somit $j' + p' < j + p$. Wegen $j' < j_2 \leq j$ ist T' daher ein Vorgänger von T und muss somit in jeder Abarbeitung vor T ausgeführt werden. □

Die folgenden beiden Lemmata sind wesentlich für die Abschätzung der Güte des Approximationsalgorithmus.

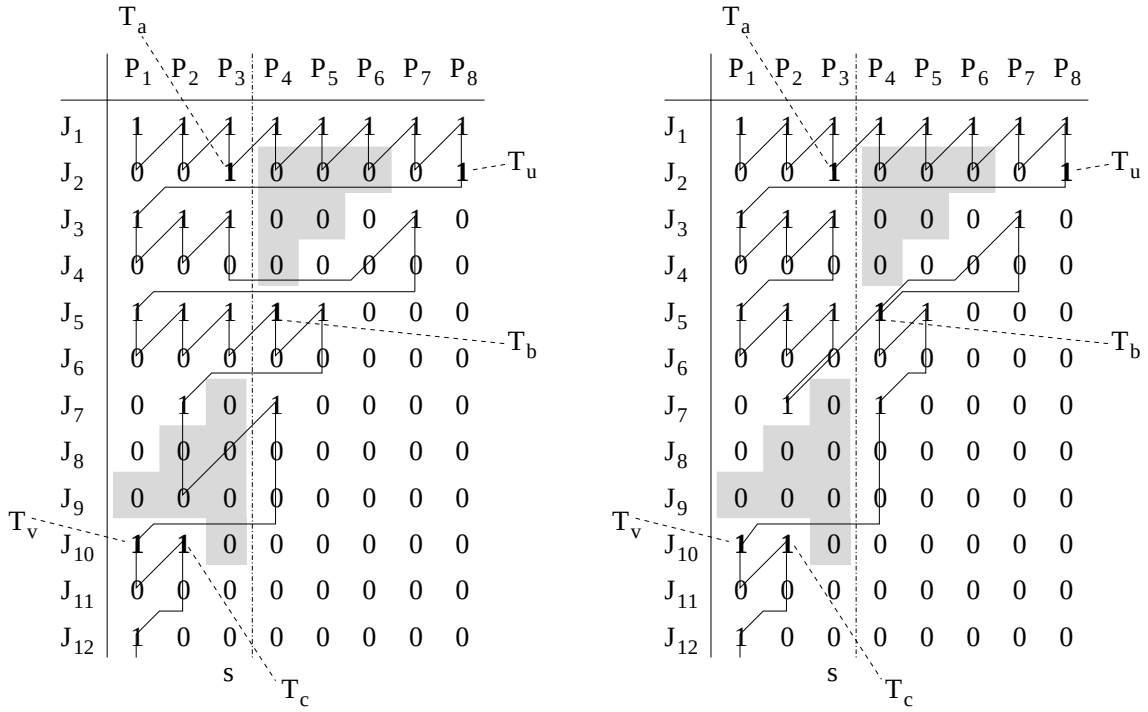


Abbildung 4.10: Die grau unterlegten Bereiche enthalten aufgrund der Definition eines LR_s- bzw. RL_s-Paares keine Tasks. Die Linie kennzeichnet wie in Abbildung 4.8 die Bewegungen des Arbeiters bei der Abarbeitung. Das rechte Bild zeigt die Abarbeitung nach einem zweifachen RL_s-Austausch wie im Beweis des Lemmas 4.6.

Lemma 4.6 *Sei s, $1 \leq s \leq m-1$, ein Maschinenindex und sei $\Lambda = (T_1, T_2, \ldots, T_z)$ eine Abarbeitung von $\mathcal{M}$ mit $T_i = T_{(j_i,p_i)}$, $i = 1, \ldots, z$. Sei (T_a, T_b, T_c), $1 \leq a < b < c \leq z$, eine beliebige s-Kette von $\mathcal{M}$, wobei T_a links von P_s liegt, und seien*

$$u := \min\{k \mid a < k \leq b,\ T_k \text{ rechts von } P_s,\ T_a \prec T_k\},$$

$$v := \min\{k \mid b < k \leq c,\ T_k \text{ links von } P_s,\ T_b \prec T_k\}.$$

(Wegen $T_a \prec T_b \prec T_c$ sind u und v wohldefiniert mit $a < u < v \leq c$.)

Falls es in Λ mehr als drei P_s-Bewegungen zwischen T_u und T_v gibt, so ist in Λ ein RL_s-Austausch möglich. Insbesondere ist in diesem Fall die Abarbeitung Λ nicht minimal.

Beweis Sei (T_a, T_b, T_c) eine s-Kette, wobei T_a links von P_s liegt, und seien u und v wie im Lemma definiert, siehe auch Abbildung 4.10. Ferner gebe es mehr als drei P_s-Bewegungen in der Abarbeitung zwischen T_u und T_v. Wir betrachten die RL_s-Aufteilung

$$R_1, L_1, R_2, L_2, \ldots, R_t, L_t, \tag{4.9}$$

der Teilfolge $T_u, T_{u+1}, \ldots, T_v$ von Λ. Man beachte, dass R_1 und L_t in Λ nicht unbedingt maximale Teilfolgen von Tasks sind, die rechts bzw. links von P_s liegen. Da es mehr als drei P_s-Bewegungen in der Abarbeitung Λ zwischen T_u und T_v gibt, muss $t \geq 3$ gelten.

Da (T_a, T_b) ein LR_s-Paar und T_u eine Task rechts von P_s mit $T_a \prec T_u$ ist, wissen wir nach Lemma 4.5, dass $j_u + p_u \geq j_b + p_b$ gilt und dass alle Tasks $T_{(j,p)}$ rechts von P_s mit $j + p < j_b + p_b$ bereits bearbeitet sind, wenn T_u ausgeführt wird. Also enthalten die Folgen $R_1, \ldots, R_t$ nur Tasks $T_{(j,p)}$ mit $j + p \geq j_b + p_b$.

Da T_v die erste Nachfolgertask von T_b in der Abarbeitung Λ ist, die links von P_s liegt, folgt, dass die Folgen $L_1, L_2, \ldots, L_{t-1}$ nur Tasks $T_{(j,p)}$ mit $j + p \leq j_b + p_b$ enthalten, da sie andernfalls Nachfolger von T_b wären. Damit kann aber keine Task aus $R_1, R_2, \ldots, R_t$ ein Vorgänger einer Task aus $L_1, L_2, \ldots, L_{t-1}$ sein. Wegen $t \geq 3$ ist daher ein RL_s-Austausch von R_2 und L_2 möglich. Tatsächlich können wir die Teilfolge (4.9) in der Abarbeitung Λ sogar durch die Folge

$$R_1, L_1, L_2, \ldots, L_{t-1}, R_2, R_3, \ldots, R_t, L_t,$$

ersetzen, was einem wiederholten RL_s-Austausch entspricht, falls $t > 3$. □

Das nächste Lemma ist analog zum vorigen Lemma und behandelt den Fall, dass die s-Kette mit einer Task rechts von P_s beginnt.

Lemma 4.7 *Sei s, $1 \leq s \leq m-1$, ein Maschinenindex und sei $\Lambda = (T_1, T_2, \ldots, T_z)$ eine Abarbeitung von $\mathcal{M}$ mit $T_i = T_{(j_i,p_i)}$, $i = 1, \ldots, z$. Sei (T_a, T_b, T_c), $1 \leq a < b < c \leq z$, eine beliebige s-Kette von $\mathcal{M}$, wobei T_a rechts von P_s liegt, und seien*

$$u := \min\{k \mid a < k \leq b,\ T_k \text{ links von } P_s,\ T_a \prec T_k\},$$

$$v := \min\{k \mid b < k \leq c,\ T_k \text{ rechts von } P_s,\ T_b \prec T_k\}.$$

(Wegen $T_a \prec T_b \prec T_c$ sind u und v wohldefiniert mit $a < u < v \leq c$.)

Falls es in Λ mehr als drei P_s-Bewegungen zwischen T_u und T_v gibt, so ist in Λ ein RL_s-Austausch möglich. Insbesondere ist in diesem Fall die Abarbeitung Λ nicht minimal.

Beweis Sei (T_a, T_b, T_c) eine s-Kette, wobei T_a rechts von P_s liegt, und seien u und v wie im Lemma definiert, siehe auch Abbildung 4.11. Ferner gebe es mehr als drei P_s-Bewegungen in der Abarbeitung zwischen T_u und T_v. Wir betrachten die RL_s-Aufteilung

$$L_1, R_1, L_2, R_2, \ldots, L_t, R_t, \tag{4.10}$$

der Teilfolge $T_u, T_{u+1}, \ldots, T_v$ von Λ. Da es mehr als drei P_s-Bewegungen in der Abarbeitung zwischen T_u und T_v gibt, muss $t \geq 3$ gelten.

Da (T_a, T_b) ein RL_s-Paar und T_u eine Task links von P_s mit $T_a \prec T_u$ ist, wissen wir nach Lemma 4.5, dass $j_u \geq j_b$ gilt und dass alle Tasks $T_{(j,p)}$ links von P_s mit $j < j_b$ bereits bearbeitet sind, wenn T_u ausgeführt wird. Also enthalten die Folgen $L_1, \ldots, L_t$ nur Tasks $T_{(j,p)}$ mit $j \geq j_b$.

Da T_v die erste Nachfolgertask von T_b in der Abarbeitung Λ ist, die rechts von P_s liegt, folgt, dass die Folgen $R_1, R_2, \ldots, R_{t-1}$ nur Tasks $T_{(j,p)}$ mit $j < j_b$ enthalten, da sie andernfalls Nachfolger von T_b wären. Damit kann aber keine Task aus $L_1, \ldots, L_t$ ein Vorgänger einer Task aus $R_1, R_2, \ldots, R_{t-1}$ sein. Wegen $t \geq 3$ ist daher ein RL_s-Austausch

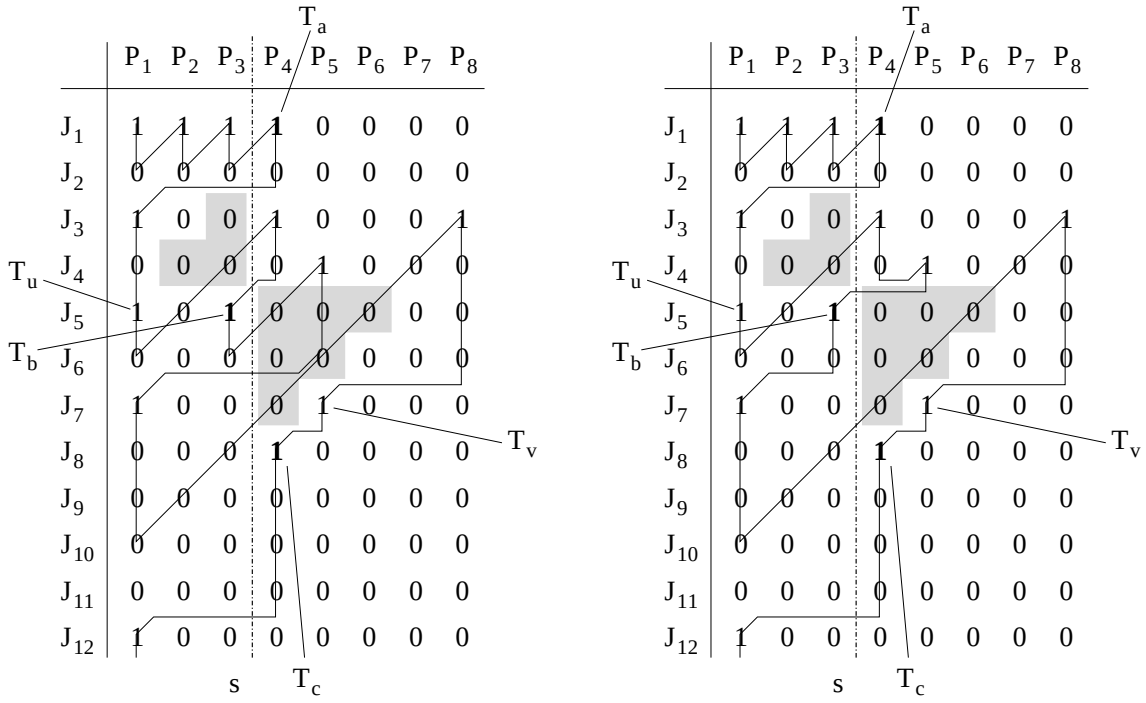


Abbildung 4.11: Zum Beweis von Lemma 4.7. Die grau unterlegten Bereiche enthalten aufgrund der Definition eines LR_s- bzw. RL_s-Paares keine Tasks. Die Linie kennzeichnet wie in Abbildung 4.8 die Bewegungen des Arbeiters bei der Abarbeitung. Das rechte Bild zeigt die Abarbeitung nach einem RL_s-Austausch wie im Beweis des Lemmas.

von L_2 und R_2 möglich. Tatsächlich können wir die die Teilfolge (4.10) in der Abarbeitung Λ sogar durch die Folge

$$L_1, R_1, R_2, \ldots, R_{t-1}, L_2, L_3, \ldots, L_t, R_t,$$

ersetzen, was einem wiederholten RL_s-Austausch entspricht, falls $t > 3$. □

Aus dem folgenden Satz ergibt sich unmittelbar, dass unser Approximationsalgorithmus eine Abarbeitung berechnet, deren Distanz höchstens dreimal so groß ist wie die untere Schranke $\mathrm{LB}^{\mathrm{A}}(\mathcal{M}, \mathcal{D})$ aus (4.3).

Satz 4.4 *Sei Λ eine Abarbeitung von $\mathcal{M}$, so dass für kein $s \in \{1, \ldots, m-1\}$ ein RL_s-Austausch möglich ist. Dann gilt für die Distanz $\Delta^{\mathcal{D}}(\Lambda)$ der Abarbeitung bezüglich jeder additiven Distanzmatrix $\mathcal{D}$*

$$\Delta^{\mathcal{D}}(\Lambda) \leq 3 \cdot LB^{A}(\mathcal{M}, \mathcal{D}),$$

wobei $LB^{A}(\mathcal{M}, \mathcal{D})$ die untere Schranke aus Gleichung (4.3) ist.

Beweis Sei $\Lambda = (T_1, \ldots, T_z)$ mit $T_i = T_{(j_i, p_i)}$, $i = 1, \ldots, z$. Wir beweisen den Satz, indem wir für jedes $s \in \{1, \ldots, m-1\}$ zeigen, dass die Anzahl der P_s-Bewegungen in Λ höchstens

dreimal so groß ist wie die minimale Anzahl $2 \cdot R_s^{\min}(\mathcal{M})$ der P_s-Bewegungen, die jede Abarbeitung benötigt. Sei also $s \in \{1, \ldots, m-1\}$ beliebig und sei

$$\Gamma_s = (T_{k_1}, T_{k_2}, \ldots, T_{k_y}), \quad 1 \leq k_1 < k_2 \ldots, < k_y \leq z,$$

die vollständige s-Kette von $\mathcal{M}$. Dann ist $T_{k_1} = T_1 = T_{(0,0)}$, T_{k_y} ist eine $\mathrm{L}_s^\star$-Task und $y = 2 \cdot R_s^{\min}(\mathcal{M}) + 1$.

Für $r = 1, 2, \ldots, y-1$ sei q_r der kleinste Index, so dass T_{q_r} ein Nachfolger von T_{k_r} ist, der auf der entgegengesetzten Seite von P_s liegt wie T_{k_r}. Offensichtlich sind die Werte q_r wohl definiert mit $q_r \leq k_{r+1}$. Nach Lemma 4.6 und Lemma 4.7 wissen wir, dass es für $1 \leq r \leq y-2$ in Λ zwischen T_{q_r} und $T_{q_{r+1}}$ höchstens drei P_s-Bewegungen gibt. Also gibt es höchstens $3(y-2)$ P_s-Bewegungen in Λ zwischen T_{q_1} und $T_{q_{y-1}}$.

Da T_{q_1} die erste Task in Λ ist, die rechts von P_s liegt, gibt es genau eine P_s-Bewegung vor der Ausführung von T_{q_1}. Wir behaupten, dass höchstens zwei P_s-Bewegungen nach Ausführung von $T_{q_{y-1}}$ auftreten können. Um dies zu sehen, betrachte man die RL_s-Aufteilung

$$L_1, R_1, L_2, R_2, \ldots, L_t, R_t, L_{t+1},$$

der Teilfolge $T_{q_{y-1}}, T_{q_{y-1}+1}, \ldots, T_z$.

Nach Lemma 4.5 wissen wir, dass für die Jobindizes $j_{q_{y-1}}$ von $T_{q_{y-1}}$ und j_{k_y} von T_{k_y} die Ungleichung $j_{q_{y-1}} \geq j_{k_y}$ gilt und dass alle Tasks $T_{(j,p)}$ mit $j < j_{k_y}$, die links von P_s liegen, bereits bearbeitet sind, wenn $T_{q_{y-1}}$ ausgeführt wird. Da Γ_s die vollständige s-Kette ist, gibt es keine Nachfolger von T_{k_y}, die rechts von P_s liegen, d.h. es gibt keine Tasks $T_{(j,p)}$ mit $j \geq j_{k_y}$ rechts von P_s. Dies bedeutet, dass keine der Folgen $L_1, \ldots, L_{t+1}$ einen Vorgänger einer Task aus $R_1, \ldots, R_t$ enthält. Daher wäre für den Fall $t \geq 2$ ein RL_s-Austausch von L_2 und R_2 möglich. Da dies nach Voraussetzung nicht der Fall ist, muss $t \leq 1$ gelten, d.h., es treten höchstens noch zwei P_s-Bewegungen nach $T_{q_{y-1}}$ auf. Somit ist die Anzahl der P_s-Bewegungen in Λ kleiner oder gleich $3(y-2)+1+2 = 3(y-1) = 6 \cdot R_s^{\min}(\mathcal{M})$. □

Wir wollen noch kurz zeigen, dass die Konstante '3' in Satz 4.4 nicht durch einen kleineren Wert ersetzt werden kann. Dazu definieren wir für $k \geq 4$ und $r \geq 1$ die Taskmatrizen $\mathcal{M}_k^r$, die aus r untereinander geschriebenen Kopien einer Taskmatrix $\mathcal{M}_k$ bestehen, siehe Abbildung 4.12. Die Taskmatrix $\mathcal{M}_k$ hat k Jobs, k Maschinen und die acht Tasks $T_{(1,1)}$, $T_{(1,k-1)}$, $T_{(1,k)}$, $T_{(k-1,1)}$, $T_{(k-1,2)}$, $T_{(k-1,k)}$, $T_{(k,2)}$ und $T_{(k,k-1)}$.

$$\Lambda_k := (T_{(1,1)}, T_{(k-1,1)}, T_{(1,k-1)}, T_{(k-1,2)}, T_{(1,k)}, T_{(k-1,k)}, T_{(k,2)}, T_{(k,k-1)})$$

ist eine Abarbeitung von $\mathcal{M}_k$ (ohne Berücksichtigung der speziellen Tasks $T_{(0,0)}$ und $T_{(\infty,0)}$) mit der linearen Distanz $6k-14$. Mit Λ_k^r bezeichnen wir die Abarbeitung von $\mathcal{M}_k^r$, die wir erhalten, wenn wir die Abarbeitungen Λ_k der Kopien von $\mathcal{M}_k$ konkatenieren. Λ_k^r hat die lineare Distanz

$$\Delta^{\mathrm{lin}}(\Lambda_k^r) = r(6k-14).$$

In Abbildung 4.12 ist die Matrix $\mathcal{M}_7^3$ mit der Abarbeitung Λ_7^3 dargestellt. Es ist leicht zu überprüfen, dass die Abarbeitung Λ_k^r keinen RL_s-Austausch erlaubt. Man betrachte

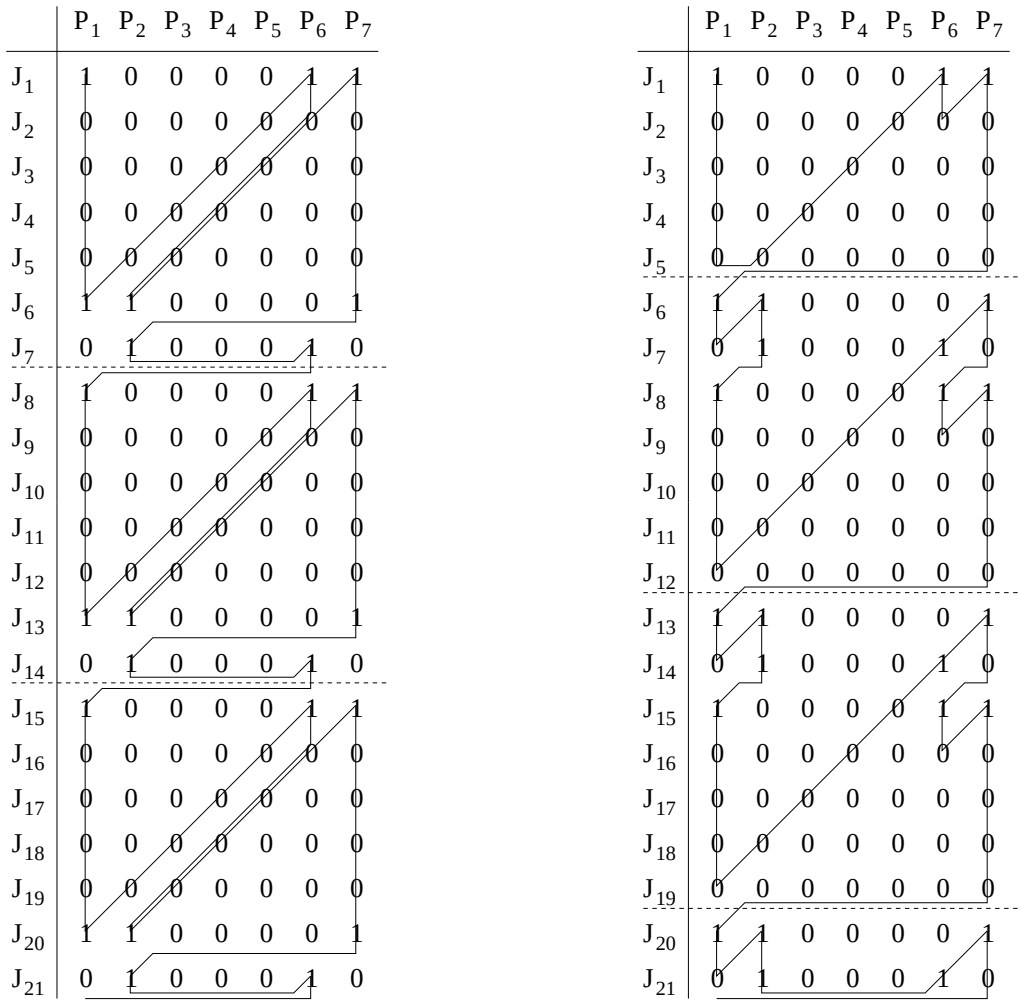


Abbildung 4.12: Links ist die Matrix $\mathcal{M}_7^3$ mit der Abarbeitung Λ_7^3 dargestellt. Λ_7^3 hat die lineare Distanz $3(6\cdot 7-14) = 84$. Rechts ist die Abarbeitung Λ'^3_7 der Matrix $\mathcal{M}_7^3$ mit der linearen Distanz $3(2\cdot 7+2)+2\cdot 7-6 = 56$ dargestellt.

nun die Abarbeitung Λ'^3_7 der Taskmatrix $\mathcal{M}_7^3$, die im rechten Bild von Abbildung 4.12 dargestellt ist. Dieses Beispiel lässt sich offensichtlich einfach auf eine Abarbeitung Λ'^r_k der Matrix $\mathcal{M}_k^r$ übertragen, bei der zunächst die drei Tasks des ersten Jobs bearbeitet werden, dann $r-1$ identische Abarbeitungen der Taskmatrizen folgen, die jeweils aus den Jobs $J_{ik-1},\ldots,J_{(i+1)k-2}$, $i = 1,\ldots,r-1$, bestehen, und abschließend die Tasks der letzten beiden Jobs bearbeitet werden. Die lineare Distanz dieser Abarbeitung beträgt

$$\Delta^{\text{lin}}(\Lambda'^r_k) = 2(k-1)+(r-1)(2k+2)+2(k-1) = r(2k+2)+2k-6.$$

Für $r\to\infty$ folgt daher

$$\lim_{r\to\infty}\frac{\Delta^{\text{lin}}(\Lambda_k^r)}{\Delta^{\text{lin}}(\Lambda'^r_k)} = \lim_{r\to\infty}\frac{r(6k-14)}{r(2k+2)+2k-6} = \frac{6k-14}{2k+2} = \frac{3k-7}{k+1}.$$

Da dieser Wert für $k \to \infty$ gegen 3 konvergiert, zeigt dies, dass die Konstante '3' in Satz 4.4 nicht durch einen kleineren Wert ersetzt werden kann. Dies zeigt auch, dass Algorithmus 6 keine bessere Güte als 3 hat, sofern man mit einer beliebigen Abarbeitung beginnen darf.

Wir wollen jetzt zeigen, wie der Algorithmus mit einer Laufzeit von $O(nm)$ implementiert werden kann. Betrachte dazu Algorithmus 7, der mit der speziellen Abarbeitung startet, in der alle Tasks lexikographisch nach Jobindex und Maschinenindex sortiert sind.

Algorithmus 7 (Approximationsalgorithmus für additive Distanzen)

(1) Starte mit der Abarbeitung Λ von $\mathcal{M}$, in der alle Tasks lexikographisch nach (Jobindex, Maschinenindex) sortiert sind;
(2) **for** $(k = 1, \ldots, m-1)$ **{**
(3) **while** (RL_k-Austausch in Λ möglich)
(4) führe RL_k-Austausch in Λ durch;
(5) **}** ;

Wir behaupten, dass in der von Algorithmus 7 berechneten Abarbeitung für kein $s \in \{1, \ldots, m-1\}$ ein RL_s-Austausch möglich ist. Für ein $s \in \{1, \ldots, m-1\}$ und zwei Indizes $j_1 < j_2$ bezeichnen wir die Folge aller Tasks $T_{(j,p)}$, $j_1 \leq j < j_2$, $p > s$, die lexikographisch nach Jobindex und Maschinenindex geordnet ist, mit Λ_{s,j_1,j_2}.

Lemma 4.8 *Sei $r \in \{1, \ldots, m-1\}$ und sei Λ die Abarbeitung in Algorithmus 7 zu einem beliebigen Zeitpunkt nach dem Initialisierungsschritt (1) und vor der Ausführung des Schleifenrumpfs der for-Schleife für $k = r+1$. Dann erfüllt Λ die folgende Eigenschaft $\mathcal{E}_s$ für alle s mit $r \leq s \leq m-1$.*

$\mathcal{E}_s$: Wenn $L_0, R_1, L_1, \ldots, R_t, L_t$ die RL_s-Aufteilung von Λ ist und j_i, $i = 0, \ldots, t$, den Jobindex der ersten Task von L_i bezeichnet, so gilt $R_i = \Lambda_{s,j_{i-1},j_i}$ für $i = 1, \ldots, t$. Insbesondere sind für $i = 1, \ldots, t$ alle Tasks von R_i Nachfolger der ersten Task von L_{i-1}.

Beweis Es ist leicht zu sehen, dass jede Abarbeitung Λ, die die Eigenschaft $\mathcal{E}_r$ erfüllt, auch die Eigenschaft $\mathcal{E}_s$, $r < s \leq m-1$ erfüllt. Da in der initialen Abarbeitung offenbar $\mathcal{E}_1$ gilt, reicht es zu zeigen, dass die Eigenschaft $\mathcal{E}_r$ nach einem RL_r-Austausch erhalten bleibt. Sei also $r \in \{1, \ldots, m-1\}$ und $L_0, R_1, L_1, \ldots, R_t, L_t$ die RL_r-Aufteilung von Λ. Angenommen, Λ erfüllt $\mathcal{E}_r$. Dann kann ein RL_r-Austausch höchstens ein R_i mit dem nachfolgenden L_i, aber nicht mit dem vorausgehenden L_{i-1} vertauschen. Ein solcher RL_r-Austausch von R_i und L_i ändert Λ zu Λ', wobei

$$\Lambda = L_0, R_1, L_1, \ldots, R_{i-1}, L_{i-1}, R_i, L_i, R_{i+1}, L_{i+1}, \ldots, R_t, L_t,$$

$$\Lambda' = L_0, R_1, L_1, \ldots, R_{i-1}, L_{i-1}, L_i, R_i, R_{i+1}, L_{i+1}, \ldots, R_t, L_t.$$

Da Λ die Eigenschaft $\mathcal{E}_r$ erfüllt, gilt $R_i = \Lambda_{r,j_{i-1},j_i}$ und $R_{i+1} = \Lambda_{r,j_i,j_{i+1}}$. Dann ist aber die Folge R_i, R_{i+1} gleich $\Lambda_{r,j_{i-1},j_{i+1}}$. Das zeigt, dass auch Λ' die Eigenschaft $\mathcal{E}_r$ erfüllt. □

Lemma 4.9 *Sei $r \in \{1, \ldots, m-1\}$ und sei Λ die Abarbeitung in Algorithmus 7 zu einem beliebigen Zeitpunkt nach der Ausführung des Schleifenrumpfs der for-Schleife für $k = r$. Dann erfüllt Λ die folgende Eigenschaft $\mathcal{F}_s$ für alle s, $1 \leq s \leq r$.*

$\mathcal{F}_s$: Wenn $L_0, R_1, L_1, \ldots, R_t, L_t$ die RL_s-Aufteilung von Λ ist, so enthält die Folge R_i, $i = 1, \ldots, t$, eine Task T, die sowohl Nachfolger einer Task aus L_{i-1} als auch Vorgänger einer Task aus L_i ist. Insbesondere ist in Λ kein RL_s-Austausch möglich.

Beweis Betrachte die Abarbeitung Λ unmittelbar nach der Ausführung des Schleifenrumpfs der for-Schleife für $k = r$. Da die while-Schleife im Schleifenrumpf beendet wurde, ist in Λ kein RL_r-Austausch möglich. Daher enthält jede Teilfolge R_i in der RL_r-Aufteilung $L_0, R_1, L_1, \ldots, R_q, L_q$ von Λ eine Task T, die Vorgänger einer Task aus L_i ist. Nach Lemma 4.8 ist T ein Nachfolger der ersten Task von L_{i-1}. Daher erfüllt Λ die die Eigenschaft $\mathcal{F}_r$. Es reicht also zu zeigen, dass die Eigenschaft $\mathcal{F}_r$ nach jedem späteren $\mathrm{RL}_{r'}$-Austausch, $r' > r$, erhalten bleibt.

Betrachte also einen $\mathrm{RL}_{r'}$-Austausch von R'_i und L'_i (nur solch ein Austausch kann nach Lemma 4.8 auftreten) in der $\mathrm{RL}_{r'}$-Aufteilung $L'_0, R'_1, L'_1, \ldots, R'_t, L'_t$ einer Abarbeitung Λ, welche die Eigenschaft $\mathcal{F}_r$, $r < r'$, erfüllt. Das ändert die Abarbeitung Λ zu Λ', wobei

$$\Lambda = L'_0, R'_1, L'_1, \ldots, R'_{i-1}, L'_{i-1}, R'_i, L'_i, R'_{i+1}, L'_{i+1}, \ldots, R'_t, L'_t,$$

$$\Lambda' = L'_0, R'_1, L'_1, \ldots, R'_{i-1}, L'_{i-1}, L'_i, R'_i, R'_{i+1}, L'_{i+1}, \ldots, R'_t, L'_t.$$

Sei $L_0, R_1, L_1, \ldots, R_q, L_q$ die RL_r-Aufteilung von Λ. Dann ist R'_i in einem R_j und R'_{i+1} in einem R_l, $l \geq j$, enthalten. Falls $l = j$, d.h. falls sowohl R'_i als auch R'_{i+1} in R_j enthalten sind, so gilt die Eigenschaft $\mathcal{F}_r$ nach dem $\mathrm{RL}_{r'}$-Austausch offensichtlich weiterhin, da nur Tasks innerhalb von R_j umgeordnet wurden. Andernfalls seien $R_j = X_j, R'_i, Y_j$ und $R_l = X_l, R'_{i+1}, Y_l$. Dann können wir Λ und Λ' wie folgt schreiben:

$$\Lambda = L_0, R_1, L_1, \ldots, L_{j-1}, \underbrace{X_j, R'_i, Y_j}_{R_j}, L_j, \ldots, L_{l-1}, \underbrace{X_l, R'_{i+1}, Y_l}_{R_l}, L_l, \ldots, R_q, L_q,$$

$$\Lambda' = L_0, R_1, L_1, \ldots, L_{j-1}, X_j, Y_j, L_j, \ldots, L_{l-1}, X_l, R'_i, R'_{i+1}, Y_l, L_l, \ldots, R_q, L_q.$$

Da Λ die Eigenschaft $\mathcal{F}_r$ erfüllt, enthält jedes R_u, $u \neq j, l$, offensichtlich eine Task T, die Nachfolger einer Task aus L_{u-1} und Vorgänger einer Task aus L_u ist. Ferner enthält die rechte Teilfolge X_l, R'_i, R'_{i+1}, Y_l der RL_r-Aufteilung von Λ' eine Task T, die Nachfolger einer Task der vorausgehenden linken Teilfolge L_{l-1} und Vorgänger einer Task der nachfolgenden linken Teilfolge L_l ist, weil R_l eine solche Task enthält. Es bleibt zu zeigen, dass die rechte Teilfolge X_j, Y_j der RL_r-Aufteilung von Λ' eine Task T enthält, die Nachfolger einer Task aus L_{j-1} und Vorgänger einer Task aus L_j ist. Da Λ die Eigenschaft $\mathcal{F}_r$ erfüllt, muss R_j eine solche Task T enthalten. Offensichtlich kann T nicht in der Teilfolge R'_i von R_j enthalten sein, da Λ' dann keine gültige Abarbeitung wäre. Also ist T in X_j, Y_j enthalten und Λ' erfüllt somit die Eigenschaft $\mathcal{F}_r$. □

Aus Lemma 4.9 folgt, dass Algorithmus 7 eine Abarbeitung berechnet, in der kein RL_s-Austausch möglich ist. Daher ist die Distanz der berechneten Abarbeitung höchstens das

Dreifache der unteren Schranke. Der Algorithmus hat offensichtlich die Laufzeit $O(nm)$, falls jeder Durchlauf durch die for-Schleife die Laufzeit $O(n)$ hat. Wir beschreiben kurz, wie dies erreicht werden kann. Wir verwalten die aktuelle Abarbeitung in einer doppelt verketteten Liste von Tasks. In einem zweidimensionalen Feld speichern wir für jedes Paar (j,p), für das $T_{(j,p)}$ eine Task ist, einen Pointer auf die entsprechende Task in der doppelt verketteten Liste. Auf diese Weise können wir für jedes solche Paar von Indizes auf die entsprechende Task und auf die in der aktuellen Abarbeitung vorausgehende und nachfolgende Task in konstanter Zeit zugreifen. Diese Datenstrukturen können für die initiale Abarbeitung offensichtlich in Zeit $O(nm)$ aufgebaut werden. Weiterhin benutzen wir die Felder LastMachine[.] und NextMachine[.][.], die im Abschnitt 2.3 definiert sind und die ebenfalls in Zeit $O(nm)$ berechnet werden können.

Betrachte nun die r-te Iteration der for-Schleife in Algorithmus 7. Wegen Lemma 4.8 können sämtliche P_r-Bewegungen leicht in Zeit $O(n)$ bestimmt werden, da jede Links-P_r-Bewegung von einer Task $T_{(j,\text{LastMachine}[j])}$ zu einer Task mit einem Maschinenindex $\leq r$ und jede Rechts-P_r-Bewegung von einer Task mit einem Maschinenindex $\leq r$ zu einer Task $T_{(j,\text{NextMachine}[j][r])}$ führt. Daher können auch die ersten und letzten Tasks aller rechten und linken Teilfolgen in der RL_r-Aufteilung in Zeit $O(n)$ gefunden werden.

Für eine Menge oder Folge S von Tasks sei $\text{MinDiag}(S) := \min\{j+p \mid T_{(j,p)} \in S\}$ und $\text{MaxDiag}(S) := \max\{j+p \mid T_{(j,p)} \in S\}$. Nach Lemma 4.8 wissen wir, dass jeder RL_r-Austausch im Algorithmus nur ein R_i mit dem nachfolgenden L_i in der RL_r-Aufteilung vertauschen kann. Solch eine Vertauschung ist genau dann möglich, wenn

$$\text{MinDiag}(R_i) \geq \text{MaxDiag}(L_i)$$

gilt. Seien j_1, j_2 die Jobindizes der ersten bzw. letzten Task von R_i. Nach Lemma 4.8 kann der Wert $\text{MinDiag}(R_i)$ einfach bestimmt werden durch

$$\text{MinDiag}(R_i) = \min\{j + \text{NextMachine}[j][r] \mid j_1 \leq j \leq j_2\}.$$

Also können diese Werte für alle rechten Teilfolgen der RL_r-Aufteilung in Zeit $O(n)$ berechnet werden. Etwas komplizierter ist die Berechnung der Werte $\text{MaxDiag}(L_i)$. Um alle diese Werte in Zeit $O(n)$ berechnen zu können, sorgen wir dafür, dass der Wert $\text{MaxDiag}(L_i)$ für jede linke Teilfolge L_i der RL_r-Aufteilung nach der r-ten Iteration der for-Schleife in $\text{MaxWert}[j][p]$ gespeichert ist, wobei $T_{(j,p)}$ die letzte Task von L_i ist. Für $r = 1$ kann dies offensichtlich dadurch erreicht werden, dass wir einfach $\text{MaxWert}[j][p] = j + p$ für alle Tasks $T_{(j,p)}$ setzen, welche die letzten Tasks in einer linken Teilfolge der RL_1-Aufteilung sind. Falls $r > 1$ ist, können wir diese Werte für die linken Teilfolgen der RL_{r-1}-Aufteilung für die Berechnung der Werte $\text{MaxDiag}(L_i)$ für eine linke Teilfolge L_i der RL_r-Aufteilung wie folgt benutzen. Seien $T_{(j_1,p_1)}$ und $T_{(j_2,p_2)}$ die erste bzw. die letzte Task von L_i und sei $T_{(j_3,p_3)}$ die erste Task der nächsten linken Teilfolge L_{i+1}. Falls der Maschinenindex p_2 der letzten Task $T_{(j_2,p_2)}$ von L_i gleich r ist, so ist $\text{MaxDiag}(L_i) = j_2 + p_2$, da alle Tasks $T_{(j,p)}$, $p \leq r$, $j + p > j_2 + p_2$, Nachfolger von $T_{(j_2,p_2)}$ sind und somit nicht in L_i enthalten sein können. Andernfalls seien $L'_j, L'_{j+1}, \ldots, L'_k$ die linken Teilfolgen der RL_{r-1}-Aufteilung,

die in L_i enthalten sind, und sei $\mathcal{T}_{r,j_1,j_3} := \{T_{(j,r)} \in \mathcal{T}(\mathcal{M}) \mid j_1 \leq j < j_3\}$. Dann gilt $L_i = \mathcal{T}_{r,j_1,j_3} \cup L'_j \cup \ldots \cup L'_k$. ($L_i$ und $L'_j, \ldots, L'_k$ seien dabei als Mengen aufgefasst). Also gilt

$$\text{MaxDiag}(L_i) = \max\{\text{MaxDiag}(\mathcal{T}_{r,j_1,j_3}), \text{MaxDiag}(L'_j), \ldots, \text{MaxDiag}(L'_k)\}.$$

Falls $k > j$ gilt, so muss eine Task $T_{(j,r)} \in \mathcal{T}_{r,j_1,j_3}$ nach der letzten Task von L'_{k-1} ausgeführt werden. Daher gilt $\text{MaxDiag}(L'_l) \leq j + r \leq \text{MaxDiag}(\mathcal{T}_{r,j_1,j_3})$ für $l = j, \ldots, k-1$. Da $\text{MaxDiag}(L'_k) = \text{MaxWert}[j_2][p_2]$ folgt also

$$\text{MaxDiag}(L_i) = \max\{\text{MaxDiag}(\mathcal{T}_{r,j_1,j_3}), \text{MaxWert}[j_2][p_2]\}.$$

Dieser Wert wird dann als neuer Wert in $\text{MaxWert}[j_2][p_2]$ gespeichert. Auf diese Weise können auch die Werte von $\text{MaxDiag}(L_i)$ für alle linken Teilfolgen der RL_r-Aufteilung in Zeit $O(n)$ berechnet werden.

Nachdem wir alle RL_r-Bewegungen und alle Werte $\text{MinDiag}(R_i)$ und $\text{MaxDiag}(L_i)$ für alle rechten Teilfolgen R_i und für alle linken Teilfolgen L_i der RL_r-Aufteilung berechnet haben, können offensichtlich alle möglichen RL_r-Austausche 'top-down' in Zeit $O(n)$ durchgeführt werden. Wenn wir zwei linken Teilfolgen bei einem RL_r-Austausch vereinigen, müssen wir den Wert MaxWert[.][.] der neuen linken Teilfolge auf das Maximum der Werte der beiden linken Teilfolgen setzen.

Aufgrund der vorausgehenden Diskussion ergibt sich das folgende Korollar zu Satz 4.4.

Korollar 4.1 *Eine Abarbeitung einer Taskmatrix $\mathcal{M}$, deren Distanz bezüglich jeder Distanzmatrix $\mathcal{D}$ höchstens das Dreifache der unteren Schranke $LB^A(\mathcal{M}, \mathcal{D})$ und damit auch höchstens das Dreifache einer minimalen Abarbeitung beträgt, kann in Zeit $O(nm)$ berechnet werden, wobei n die Anzahl der Jobs und m die Anzahl der Maschinen der Taskmatrix ist.*

Zum Abschluss dieses Kapitels wollen wir noch kurz auf die Qualität unserer unteren Schranke eingehen. Sei c^{A}_{LB} das Infimum aller c, so dass für alle Taskmatrizen $\mathcal{M}$ und alle additiven Distanzmatrizen $\mathcal{D}$

$$\Delta^{\mathcal{D}}_{\min}(\mathcal{M}) \leq c \cdot \text{LB}^{\text{A}}(\mathcal{M}, \mathcal{D})$$

gilt, und sei $c^{\text{lin}}_{\text{LB}}$ das Infimum aller c, so dass für alle Taskmatrizen $\mathcal{M}$

$$\Delta^{\text{lin}}_{\min}(\mathcal{M}) \leq c \cdot \text{LB}^{\text{lin}}(\mathcal{M})$$

gilt. Da die linearen Distanzen ein Spezialfall der additiven Distanzen sind, gilt $c^{\text{lin}}_{\text{LB}} \leq c^{\text{A}}_{\text{LB}}$. Nach Korollar 4.1 ist $c^{\text{A}}_{\text{LB}} \leq 3$ und somit auch $c^{\text{lin}}_{\text{LB}} \leq 3$. Der Beweis basierte darauf, dass eine minimale Abarbeitung für jedes $s \in \{1, \ldots, m-1\}$ höchstens dreimal soviele Rechts-P_s-Bewegungen hat wie die minimale erforderliche Anzahl $R^{\min}_s(\mathcal{M})$ solcher Bewegungen. Das Beispiel aus Abbildung 4.13 zeigt, dass es Taskmatrizen $\mathcal{M}$ und minimale Abarbeitungen Λ von $\mathcal{M}$ bezüglich linearer Distanzen gibt, so dass das Verhältnis $\frac{R_s(\Lambda)}{R^{\min}_s(\mathcal{M})}$ dem Wert 3 für einen Maschinenindex s tatsächlich beliebig nahe kommt. Dieses Beispiel zeigt, dass man

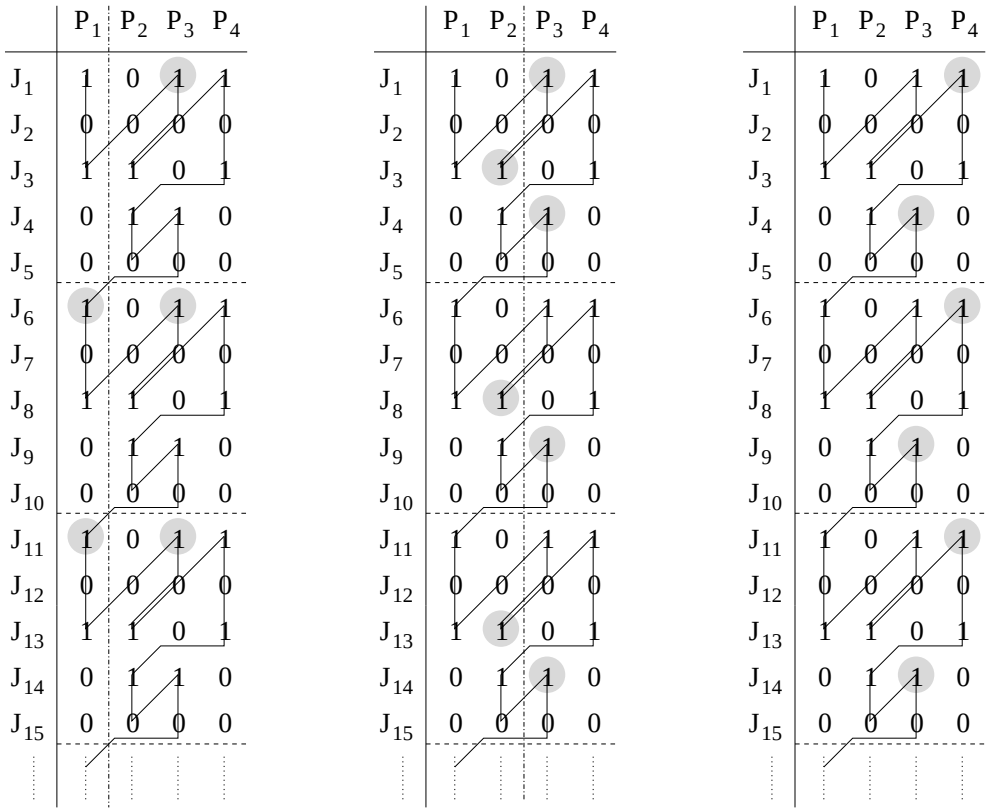


Abbildung 4.13: Die Taskmatrix besteht aus einer beliebig häufigen Wiederholung der ersten 5 Jobs. Die eingezeichnete Abarbeitung ist minimal bezüglich linearer Distanzen. Die grau unterlegten Tasks zeigen die vollständige 1-, 2- bzw. 3-Kette (ohne die Starttask $T_{(0,0)}$). Für $s = 1, 3$ gibt es genau eine, für $s = 2$ (mit Ausnahme der ersten beiden Tasks) genau drei P_s-Bewegungen zwischen zwei aufeinander folgenden Tasks der vollständigen s-Kette. Dies zeigt zum einen, dass für ein einzelnes s das Verhältnis der Anzahl der P_s-Bewegungen in einer minimalen Abarbeitung zu der Anzahl der minimal benötigten P_s-Bewegungen dem Wert 3 beliebig nahe kommen kann. Zum anderen zeigt das Beispiel, dass sich das Verhältnis zwischen der minimalen linearen Distanz und der unteren Schranke dem Wert $\frac{5}{3}$ beliebig annähern kann.

$c^{\text{lin}}_{\text{LB}} < 3$ oder $c^{\text{A}}_{\text{LB}} < 3$ nicht dadurch beweisen kann, dass man für jeden Maschinenindex $s \in \{1, \ldots, m-1\}$ getrennt die minimale Anzahl $R^{\min}_s(\mathcal{M})$ von Rechts-P_s-Bewegungen mit der Anzahl solcher Bewegungen in einer minimalen Abarbeitung vergleicht. (Das Beispiel ist im Übrigen auch nicht dafür geeignet, für allgemeine additive Distanzen $c^{\text{A}}_{\text{LB}} = 3$ dadurch zu zeigen, dass man die Distanz zwischen P_2 und P_3 beliebig erhöht. In diesem Fall ist die angegebene Abarbeitung nämlich nicht mehr minimal.) Die beste uns bekannte untere Schranke für den Wert von $c^{\text{lin}}_{\text{LB}}$ und c^{A}_{LB} ist $\frac{5}{3}$. Diese Schranke belegt ebenfalls das Beispiel aus Abbildung 4.13. Wir haben also die Abschätzungen

$$\frac{5}{3} \leq c^{\text{lin}}_{\text{LB}} \leq c^{\text{A}}_{\text{LB}} \leq 3.$$

Kapitel 5

Online-Algorithmen

In diesem Kapitel untersuchen wir Online-Algorithmen. Die Untersuchungen sind dabei durch den besonders relevanten Fall der linearen Distanzen motiviert. Die meisten hier gezeigten Resultate gelten jedoch auch für beliebige additive Distanzen, so dass wir sie gleich für den allgemeinen Fall formulieren und beweisen. Bei der Untersuchung von unteren Schranken für die Güte von Algorithmen betrachten wir in der Regel aber lineare Distanzen. Natürlich gelten die unteren Schranken für lineare Distanzen auch für den allgemeineren Fall beliebiger additiver Distanzen, allerdings sind sie für diesen Fall oft weniger interessant. Für additive Distanzen, bei denen sich die Entfernungen zwischen verschiedenen benachbarten Maschinenpaaren stark unterscheiden, würde man vermutlich Online-Strategien in Erwägung ziehen, die die Anzahl der besonders kostenträchtigen Übergänge minimieren. Solche Algorithmen werden hier nicht betrachtet, da das Hauptinteresse auf den linearen Distanzen liegt. Teile dieses Kapitels wurden bereits kurz in [7] vorgestellt.

Für das Problem der Bewegungsminimierung in der Förderband-Flow-Shop-Verarbeitung sind Online-Algorithmen besonders interessant. Dies liegt daran, dass sich die Jobreihenfolge, wie in der Einleitung geschildert, erst während der Abarbeitung ergibt. Somit ist nicht die gesamte Jobsequenz bekannt, sondern nur der Teil, der sich gerade auf dem Nebenförderband befindet. Insbesondere sind solche Bewegungsstrategien interessant, bei denen der Arbeiter die nächste auszuführende Task anhand einer einfachen Regel selbst bestimmen kann.

Im ersten Abschnitt machen wir grundlegende Aussagen über Online-Algorithmen für unser Problem und zeigen eine untere Schranke für deren Güte. Im zweiten Abschnitt untersuchen wir einige einfache Online-Algorithmen, deren Güte sich jedoch als unbefriedigend herausstellt. Im dritten Abschnitt betrachten wir einen Algorithmus, den wir Free-Left nennen. Dieser bestimmt die nächste auszuführende Task durch eine sehr einfache Regel. Dennoch liefert er in den meisten Fällen sehr gute Abarbeitungen. Wir können für den Free-Left-Algorithmus beweisen, dass die von ihm berechneten Lösungen höchstens um einen Faktor aus $O(\log m)$ schlechter sind als die optimalen Lösungen, wobei m die Anzahl der Maschinen ist. Es gibt Beispiele, die belegen, dass der Free-Left-Algorithmus in der Tat keinen konstanten Fehler hat. Empirische Untersuchungen zeigen jedoch, dass die von dem Algorithmus berechneten Lösungen für lineare Distanzen in fast allen Fällen um

weniger als 20 Prozent über der optimalen Lösung liegen. Diese guten Ergebnisse sind darauf zurückzuführen, dass der Algorithmus in vielen Fällen optimale Entscheidungen trifft. Im gesamten Kapitel stellt die Distanzmatrix $\mathcal{D} = (\delta_{p,q})$ immer additive Distanzen dar. Für Beweiszwecke fügen wir wie in Abschnitt 4.3 die speziellen Tasks $T_{(0,0)}$ und $T_{(\infty,0)}$ zur Taskmenge hinzu.

5.1 Allgemeine Aussagen über Online-Algorithmen

Wir sprechen von einem *Online-Algorithmus* für das Förderband-Flow-Shop-Problem, falls der Algorithmus für die Entscheidung, welche Task in einem Zustand $\mathcal{T}$ als nächstes ausgeführt wird, keine Informationen über Jobs verwendet, die die erste Maschine noch nicht erreicht haben. Auch die Anzahl der Jobs ist dem Algorithmus nicht bekannt. Erst zu dem Zeitpunkt, an dem die erste Maschine P_1 nicht mehr von einem Job belegt wird, weiß der Algorithmus, dass der letzte Job die erste Maschine bereits passiert hat.

Ein Algorithmus heißt *deterministisch*, wenn er keine zufälligen Entscheidungen trifft, d.h., wenn er bei derselben Eingabe stets dieselbe Abarbeitung berechnet. Sei $\Delta_{\mathrm{A}}^{\mathcal{D}}(\mathcal{M})$ die Distanz der Abarbeitung einer Taskmatrix $\mathcal{M}$, die ein deterministischer Online-Algorithmus A liefert. Der Algorithmus A heißt *c-kompetitiv*, falls es eine Konstante b gibt, so dass für alle Taskmatrizen $\mathcal{M}$

$$\Delta_{\mathrm{A}}^{\mathcal{D}}(\mathcal{M}) \leq c \cdot \Delta_{\min}^{\mathcal{D}}(\mathcal{M}) + b$$

gilt. Der *kompetitive Faktor* oder *die Güte von A* ist das Infimum aller c, so dass A c-kompetitiv ist.

Sei $\Lambda = T_1, T_2, \ldots, T_k$ eine Teilabarbeitung von $\mathcal{M}$. Wir nennen eine Anordnung $\Lambda' = (T'_1, T'_2, \ldots, T'_r)$ der Tasks von $\mathcal{T}(\mathcal{M}) - \{T_1, \ldots, T_k\}$, die die Vorgängerrelation einhält, eine *Fortsetzung von* Λ. In diesem Fall ist $\Lambda \circ \Lambda' := (T_1, \ldots, T_k, T'_1, \ldots, T'_r)$ eine Abarbeitung von $\mathcal{M}$. Wir nennen die Fortsetzung Λ' eine *minimale Fortsetzung von* Λ, wenn die Distanz $\Delta^{\mathcal{D}}(\Lambda \circ \Lambda')$ minimal unter allen Fortsetzungen von Λ ist.

Wir hatten bereits in Kapitel 3 den Begriff einer Konfiguration eingeführt. Durch die Einführung der zusätzlichen Tasks $T_{(0,0)}$ und $T_{(\infty,0)}$ lässt sich die Definition etwas vereinfachen. Wir wiederholen sie daher nochmals in leicht geänderter Form.

Jede nicht-leere Teilabarbeitung $\Lambda = (T_1, \ldots, T_k)$ definiert eine Paar $\kappa(\Lambda) := (\mathcal{T}_\Lambda, p_\Lambda)$, wobei $\mathcal{T}_\Lambda := \mathcal{T}(\mathcal{M}) - \{T_1, \ldots, T_k\}$ der Zustand nach Bearbeitung der Tasks $\{T_1, \ldots, T_k\}$ und p_Λ die letzte Arbeiterposition, d.h. der Maschinenindex der zuletzt ausgeführten Task T_k, ist. Wir nennen jedes Paar $\kappa = (\mathcal{T}, p)$, wobei $\mathcal{T}$ ein Zustand und p eine Arbeiterposition ist, eine *Konfiguration*, falls es eine nicht-leere Teilabarbeitung Λ mit $(\mathcal{T}, p) = \kappa(\Lambda)$ gibt. Die durch die Teilabarbeitung $(T_{(0,0)})$ definierte Konfiguration $(\mathcal{T}(\mathcal{M}) - \{T_{(0,0)}\}, 0)$ heißt *Anfangskonfiguration* und die durch eine beliebige Abarbeitung definierte Konfiguration $(\emptyset, 0)$ heißt *Endkonfiguration*. Für eine Konfiguration $\kappa = (\mathcal{T}, p)$ bezeichnen wir mit $\mathcal{T}_\kappa := \mathcal{T}$ den Zustand und mit $p_\kappa := p$ die Arbeiterposition der Konfiguration. Falls κ nicht die Endkonfiguration ist, so nennen wir eine Anordnung $\Lambda = (T_1, \ldots, T_r)$, $T_i = T_{(j_i,p_i)}$, aller Tasks von $\mathcal{T}_\kappa$ eine *Restabarbeitung für* κ, falls Λ die Vorgängerrelation einhält. Die *Distanz*

$\Delta_\kappa^{\mathcal{D}}(\Lambda)$ *der Restabarbeitung* Λ *bezüglich der Konfiguration* κ ist definiert durch

$$\Delta_\kappa^{\mathcal{D}}(\Lambda) := \delta_{p_\kappa,p_1} + \sum_{i=1}^{r-1} \delta_{p_i,p_{i+1}}.$$

Wir nennen die Restabarbeitung *minimal für* κ, falls ihre Distanz bezüglich κ minimal unter allen Restabarbeitungen für κ ist. Ist Λ' eine Teilabarbeitung, so sind die Fortsetzungen von Λ' natürlich genau die Restabarbeitungen für $\kappa(\Lambda')$. Ist Λ eine solche Restabarbeitung, so gilt für die Distanz der Abarbeitung $\Lambda' \circ \Lambda$

$$\Delta^{\mathcal{D}}(\Lambda' \circ \Lambda) = \Delta^{\star\mathcal{D}}(\Lambda') + \Delta^{\mathcal{D}}_{\kappa(\Lambda')}(\Lambda).$$

Sei $\Lambda = (T_1, \ldots, T_r)$, $T_i = T_{(j_i,p_i)}$, eine Restabarbeitung für die Konfiguration κ. Unter den *Bewegungen der Restabarbeitung* Λ *bezüglich der Konfiguration* κ verstehen wir neben den üblichen Bewegungen (T_i, T_{i+1}), $i = 1, \ldots, r-1$, noch eine zusätzliche Bewegung von der Arbeiterposition p_κ zur Maschine P_{p_1} der ersten Task T_1. Für einen Maschinenindex s, $1 \leq s \leq m-1$, bezeichnen wir mit $R_s(\kappa, \Lambda)$ die Anzahl der Rechts-P_s-Bewegungen von Λ bezüglich κ, wobei wir die Bewegung zur ersten Task im Fall $p_\kappa \leq s < p_1$ hinzuzählen. Offensichtlich gilt für die Distanz der Restabarbeitung Λ bezüglich der Konfiguration κ

$$\Delta_\kappa^{\mathcal{D}}(\Lambda) = \delta_{p_\kappa,0} + 2\sum_{s=1}^{m-1} \delta_{s,s+1} R_s(\kappa, \Lambda). \tag{5.1}$$

Ein Online-Algorithmus sollte in einer Konfiguration κ möglichst als nächstes eine Task ausführen, die die erste Task in einer minimalen Restabarbeitung für κ ist. Wählt ein Algorithmus eine solche Task, so sprechen wir von einer *optimalen Entscheidung*. Wartet ein Job J_j in der Konfiguration κ an der Maschine P_{p_κ} auf Bearbeitung, so ist es offensichtlich eine optimale Entscheidung, die Task $T_{(j,p_\kappa)}$ als nächstes auszuführen, da das Verbleiben des Arbeiters an der augenblicklichen Position keine Kosten verursacht. Dies gilt ganz allgemein für beliebige Distanzfunktionen und lässt sich leicht mit Lemma 2.3 beweisen. Man betrachte nun eine Konfiguration κ, in der kein Job an der aktuellen Arbeiterposition p_κ auf Bearbeitung wartet. Bei additiven Distanzen, die wir hier behandeln, gibt es dann offensichtlich eine minimale Restabarbeitung, in der zuerst entweder die nächste ausführbare Task links von P_{p_κ}, d.h. die ausführbare Task $T_{(j,p)}$ mit maximalem $p < p_\kappa$, oder die nächste ausführbare Task rechts von P_{p_κ}, d.h. die ausführbare Task $T_{(j,p)}$ mit minimalem $p > p_\kappa$, bearbeitet wird. Auch diese Tatsache lässt sich für additive Distanzen leicht mit Lemma 2.3 beweisen.

Wir nennen eine Konfiguration κ eine *Entscheidungskonfiguration (für additive Distanzen)*, falls im Zustand $\mathcal{T}_\kappa$ keine Task an der Maschine P_{p_κ} auf Bearbeitung wartet, aber es ausführbare Tasks sowohl links als auch rechts von P_{p_κ} gibt. Nach den obigen Bemerkungen, muss ein Online-Algorithmus für additive Distanzen nur in Entscheidungskonfigurationen bestimmen, ob der Arbeiter sich zur nächsten ausführbaren Task links oder zur nächsten ausführbaren Task rechts bewegt. Wir beschreiben daher einen Online-Algorithmus, indem wir angeben, welche dieser beiden Tasks der Arbeiter als nächstes bearbeitet. In allen

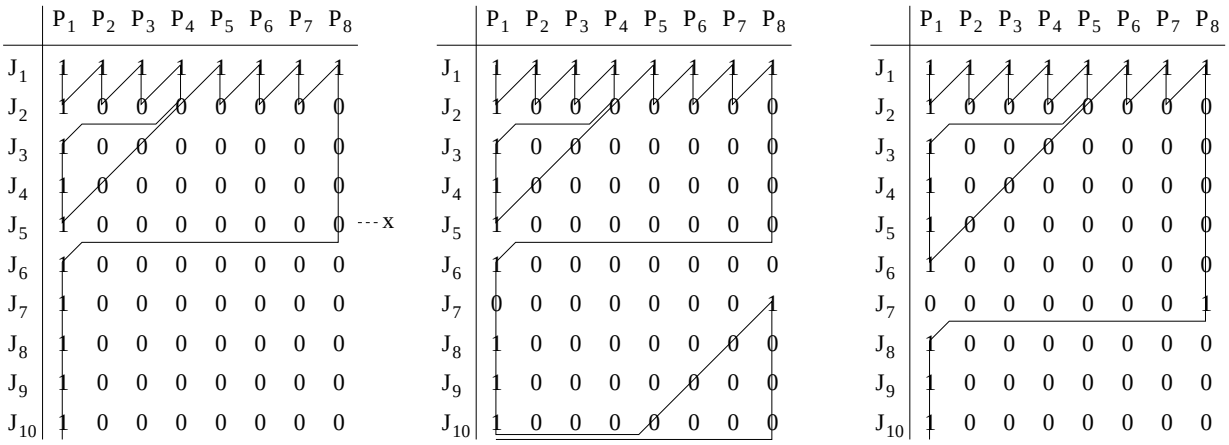


Abbildung 5.1: Zum Beweis von Satz 5.1. Links ist die Taskmatrix $\mathcal{M}$ mit der Abarbeitung eines deterministischen Online-Algorithmus A dargestellt. In der Mitte ist eine mögliche Abarbeitung der modifizierten Matrix $\mathcal{M}'$ durch A gezeigt, welche bis zur Task $T_{(x+1,1)}$ mit der Abarbeitung der Matrix $\mathcal{M}$ übereinstimmen muss. Rechts ist eine minimale Abarbeitung von $\mathcal{M}'$ angegeben.

Nicht-Entscheidungskonfigurationen gehen wir davon aus, dass der Algorithmus immer die offensichtlich mögliche optimale Entscheidung trifft.

Satz 5.1 *Der kompetitive Faktor eines jeden deterministischen Online-Algorithmus für das Förderband-Flow-Shop-Problem mit linearen Distanzen ist mindestens*

$$\phi := \frac{1}{2}\left(1+\sqrt{5}\right) \approx 1.61803\ldots \quad \textit{('goldener Schnitt').}$$

Beweis Sei A ein beliebiger deterministischer Online-Algorithmus. Für ein beliebiges $m > 1$ betrachten wir die Taskmatrix $\mathcal{M}$ mit m Maschinen und $m+2$ Jobs, deren erster Job Tasks an allen Maschinen hat und deren restliche Jobs nur jeweils eine Task an der ersten Maschine haben, siehe Abbildung 5.1. Sei x der maximale Jobindex, so dass der Algorithmus A die Task $T_{(x,1)}$ vor der Task $T_{(1,m)}$ ausführt. Wegen $m > 1$ gilt $T_{(1,1)} \prec T_{(1,m)} \prec T_{(m+1,1)}$ und somit ist x wohldefiniert mit $1 \leq x \leq m$. Da für $x > 1$ die Task $T_{(1,x-1)}$ ein Vorgänger von $T_{(x,1)}$ ist und da die Task $T_{(1,m)}$ nach der Task $T_{(x,1)}$ und vor der Task $T_{(x+1,1)}$ ausgeführt wird, beträgt (auch im Fall $x = 1$) die Distanz der Abarbeitung der Taskmatrix $\mathcal{M}$ durch den Algorithmus A zum Zeitpunkt der Bearbeitung von $T_{(x+1,1)}$ mindestens $2(x-2)+2(m-1) = 2(m+x-3)$. Da eine minimale Abarbeitung von $\mathcal{M}$ offensichtlich die lineare Distanz $2(m-1)$ hat, folgt für den Fall $x \geq (\phi-1)m$

$$\frac{\Delta_{\mathrm{A}}^{\mathrm{lin}}(\mathcal{M})}{\Delta_{\min}^{\mathrm{lin}}(\mathcal{M})} \geq \frac{m+x-3}{m-1} \geq \frac{m+(\phi-1)m-3}{m-1} \underset{m\to\infty}{\longrightarrow} \phi.$$

Falls $x < (\phi-1)m$, so betrachten wir die Taskmatrix $\mathcal{M}'$, die wir aus $\mathcal{M}$ erhalten, indem wir die Task $T_{(x+2,1)}$ entfernen und die Task $T_{(x+2,m)}$ hinzufügen, siehe mittleres Bild in

Abbildung 5.1. Da die ersten $x+1$ Jobs mit denen der Taskmatrix $\mathcal{M}$ identisch sind, verhält sich der Algorithmus A bei der Bearbeitung von $\mathcal{M}'$ bis einschließlich der Ausführung der Task $T_{(x+1,1)}$ offensichtlich genauso wie bei der Bearbeitung von $\mathcal{M}$. Da anschließend noch die Task $T_{(x+2,m)}$ bearbeitet werden muss, benötigt der Algorithmus für die Bearbeitung der gesamten Taskmatrix also mindestens die lineare Distanz $2(m+x-3)+2(m-1) = 2(2m+x-4)$. Andererseits gibt es aber eine Abarbeitung mit der Distanz $2(x-1)+2(m-1) = 2(m+x-2)$, siehe rechtes Bild in Abbildung 5.1. Somit folgt wegen $x < (\phi-1)m$

$$\frac{\Delta_{\mathrm{A}}^{\mathrm{lin}}(\mathcal{M}')}{\Delta_{\mathrm{min}}^{\mathrm{lin}}(\mathcal{M}')} \geq \frac{2m+x-4}{m+x-2} \geq \frac{2m+(\phi-1)m-4}{m+(\phi-1)m-2} \underset{m\to\infty}{\longrightarrow} \frac{1+\phi}{\phi} = \phi.$$

Die letzte Gleichung folgt wegen $\phi^2 = \frac{1}{4}\left(1+\sqrt{5}\right)^2 = \frac{1}{4}\left(6+2\sqrt{5}\right) = 1+\frac{1}{2}\left(1+\sqrt{5}\right) = 1+\phi$. Mit wachsender Maschinenanzahl m kommt das Verhältnis der Distanz einer Abarbeitung durch den Algorithmus A zur minimalen Distanz dem Wert ϕ also entweder für die Matrix $\mathcal{M}$ oder für die Matrix $\mathcal{M}'$ beliebig nahe. Dies zeigt, dass der kompetitive Faktor von A nicht kleiner als ϕ sein kann. □

5.2 Einfache Online-Algorithmen

In diesem Abschnitt wollen wir beispielhaft einige einfache Bewegungsstrategien untersuchen. Alle hier vorgestellten Online-Algorithmen mit Ausnahme des Nearest-First-Algorithmus haben einen kompetitiven Faktor in $\Omega(m)$. Für den Nearest-First-Algorithmus können wir lediglich einen kompetitiven Faktor in $\Omega\left(\frac{m}{\log m}\right)$ nachweisen, was jedoch nur unwesentlich besser ist. Wir verwenden in diesem Abschnitt immer lineare Distanzen.

Betrachten wir zunächst den Algorithmus *Right-First*, der in einer Entscheidungskonfiguration immer die Task rechts von der augenblicklichen Arbeiterposition bearbeitet. In Abbildung 5.2 ist die Abarbeitung einer Taskmatrix mit 13 Maschinen und 12 Jobs durch den Algorithmus Right-First einer minimalen Abarbeitung gegenübergestellt. Das Beispiel lässt sich leicht auf $m = 2k+1$ Maschinen und $n = 2k$ Jobs für beliebiges $k \in \mathbb{N}$ verallgemeinern. Die Abarbeitung durch den Right-First-Algorithmus hat dann eine Distanz von $4k^2$, während eine minimale Abarbeitung eine Distanz von $6k-2$ hat. Dies zeigt, dass der Algorithmus Right-First einen kompetitiven Faktor in $\Omega(m)$ besitzt.

Betrachten wir als nächstes den Algorithmus *Left-First*, der in einer Entscheidungskonfiguration immer die Task links von der augenblicklichen Arbeiterposition bearbeitet. In Abbildung 5.3 ist die Abarbeitung einer Taskmatrix mit 12 Maschinen und 12 Jobs durch den Algorithmus Left-First einer minimalen Abarbeitung gegenübergestellt. Das Beispiel lässt sich leicht auf eine beliebige Anzahl von m Maschinen und m Jobs verallgemeinern. Die Abarbeitung durch den Left-First-Algorithmus hat dann eine Distanz von $2\sum_{i=1}^{m-1} i = m(m-1)$, während eine eine minimale Abarbeitung eine Distanz von $2(m-1)$ hat. Dies zeigt, dass auch der Algorithmus Left-First einen kompetitiven Faktor in $\Omega(m)$ hat.

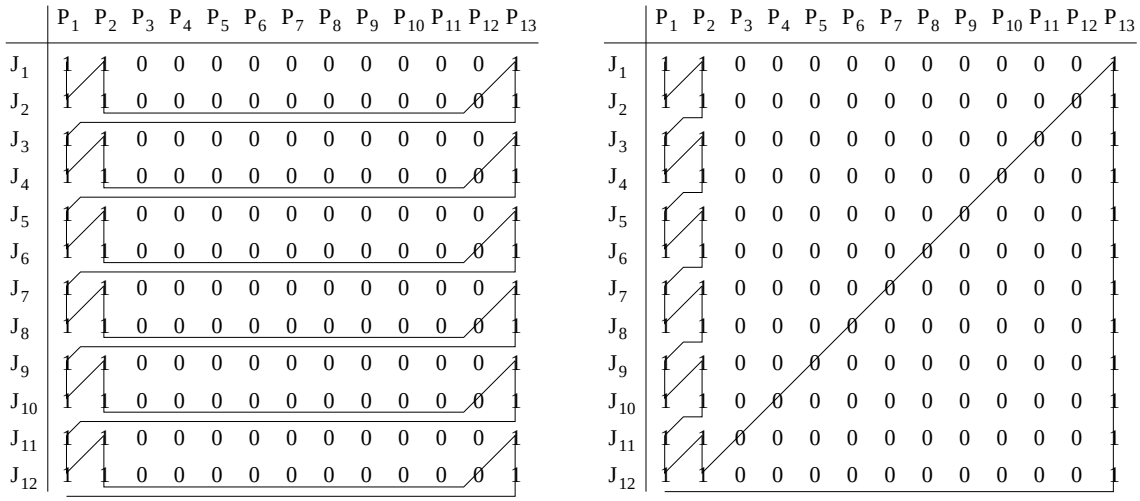


Abbildung 5.2: Links eine Abarbeitung durch den Algorithmus Right-First. Rechts eine minimale Abarbeitung.

Der Algorithmus *Keep-Direction*, bei dem sich der Arbeiter in einer Entscheidungskonfiguration immer in dieselbe Richtung (d.h. nach rechts oder nach links) bewegt wie bei der letzten Bewegung, führt auf den Matrizen aus Abbildung 5.2 zu denselben Abarbeitungen wie der Right-First-Algorithmus und hat daher auch einen kompetitiven Faktor in $\Omega(m)$. Der Algorithmus *Change-Direction*, der sich entgegengesetzt verhält und in einer Entscheidungskonfiguration die entgegengesetzte Richtung der letzten Bewegung wählt, führt auf den Matrizen aus Abbildung 5.3 zu den gleichen Abarbeitungen wie der Left-First-Algorithmus. Wir haben noch eine Reihe weiterer einfacher Bewegungsstrategien untersucht wie z.B. *More-Waiting* und *Less-Waiting*, bei denen sich der Arbeiter in einer Entscheidungskonfiguration zu der Seite bewegt, auf der mehr bzw. weniger Jobs auf Bearbeitung warten, oder *More-Ready* und *Less-Ready*, bei denen der Arbeiter die Seite ansteuert, auf der sich mehr bzw. weniger Jobs befinden, die an ihrer augenblicklichen Position nicht mehr bearbeitet werden müssen. Für alle diese Strategien lassen sich einfach Beispiele angeben, die einen kompetitiven Faktor in $\Omega(m)$ belegen.

Zum Schluss wollen wir noch den Algorithmus *Nearest-First* betrachten. In einer Entscheidungskonfiguration bewegt der Nearest-First-Algorithmus den Arbeiter zu der am nächsten gelegenen ausführbaren Task. Falls zwei ausführbare Tasks die gleiche Entfernung von der augenblicklichen Arbeiterposition haben, so wird eine beliebige dieser Tasks, z.B. immer die linke oder immer die rechte, gewählt. Man könnte zunächst vermuten, dass dieser Algorithmus einen wesentlich besseren kompetitiven Faktor besitzt. Wir zeigen jedoch, dass der kompetitive Faktor von Nearest-First in $\Omega\left(\frac{m}{\log_2 m}\right)$ liegt und somit nicht wesentlich besser als $\Omega(m)$ ist. Man betrachte dazu die Taskmatrizen $\mathcal{M}_k$, $k \geq 1$, die wie folgt definiert sind. $\mathcal{M}_k$ hat $m := 2 + 2^{k+1}$ Maschinen und $n := 1 + 2^k = \frac{m}{2}$ Jobs. Alle Jobs haben Tasks genau an den Maschinen P_1 und P_{2+2^i}, $i = 0, \ldots, k+1$. Abbildung

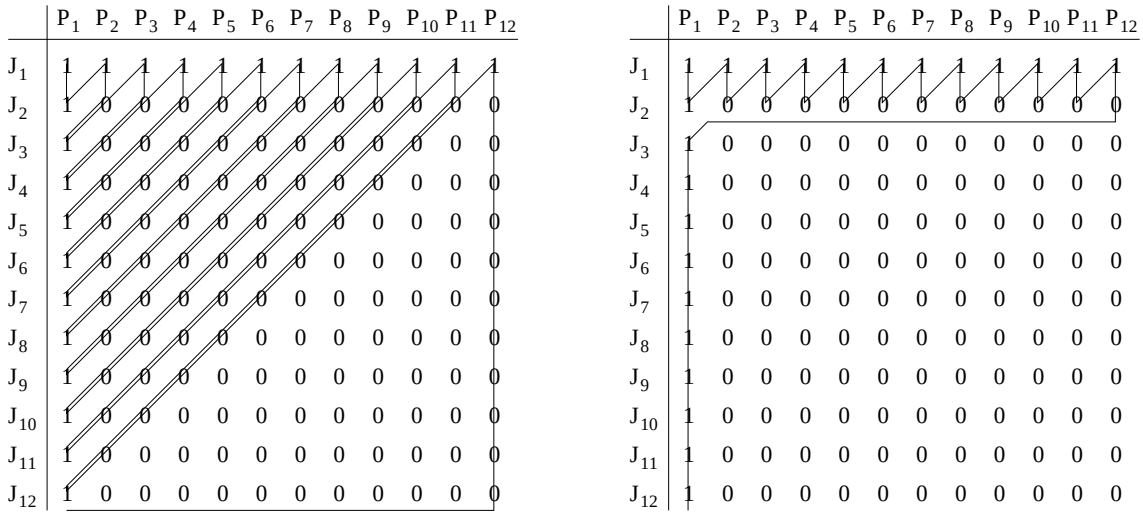


Abbildung 5.3: Links eine Abarbeitung durch den Algorithmus Left-First. Rechts eine minimale Abarbeitung.

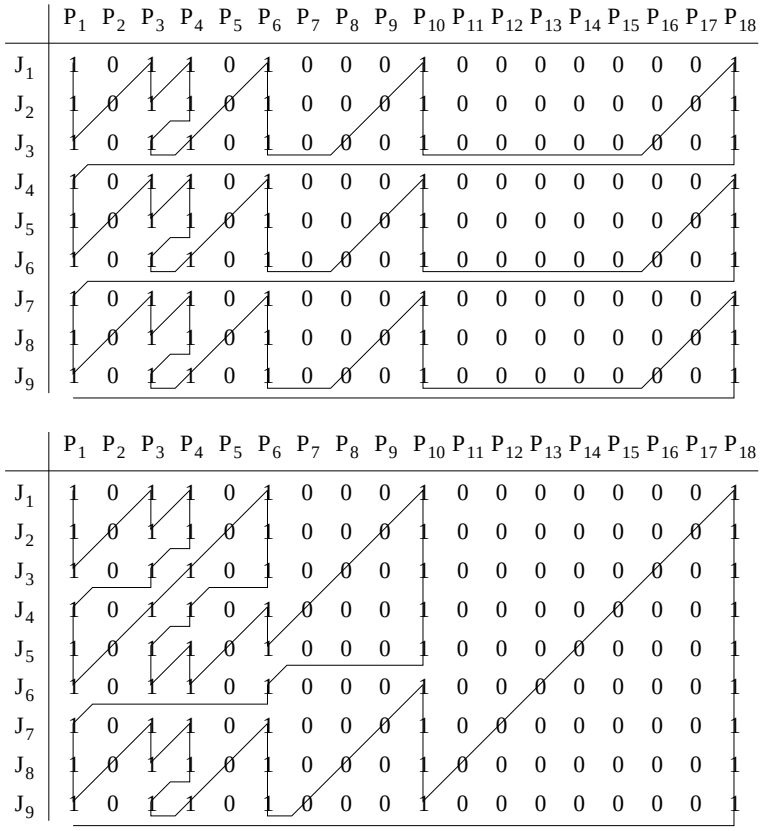


Abbildung 5.4: Oben eine Abarbeitung durch den Algorithmus Nearest-First, unten eine minimale Abarbeitung der Matrix $\mathcal{M}_3$.

5.4 zeigt die Abarbeitung der Matrix $\mathcal{M}_3$ durch den Algorithmus Nearest-First sowie eine minimale Abarbeitung. Es ist leicht zu sehen, dass die Abarbeitung der Matrix $\mathcal{M}_k$ durch den Algorithmus Nearest-First eine Distanz $\geq 2(m-1)\left\lfloor \frac{n}{3} \right\rfloor \in \Omega(m^2)$ hat. Aus Satz 5.2, den wir im nächsten Abschnitt beweisen werden, folgt, dass eine minimale Abarbeitung der Matrix $\mathcal{M}_k$ die Distanz

$$\begin{aligned}\Delta_{\min}(\mathcal{M}_k) &= 2\left(\left\lceil \frac{n}{3} \right\rceil (3-1) + \sum_{i=0}^{k} \left\lceil \frac{n}{2^{i+1}-2^i+1} \right\rceil (2^{i+1}-2^i)\right) \\ &= 2\left(2\left\lceil \frac{m}{6} \right\rceil + \sum_{i=0}^{\log_2(m-2)-1} \left\lceil \frac{m}{2(2^{i+1}-2^i+1)} \right\rceil (2^{i+1}-2^i)\right) \in O(m \log_2 m)\end{aligned}$$

besitzt. Also liegt der kompetitive Faktor von Nearest-First in $\Omega\left(\frac{m}{\log_2 m}\right)$.

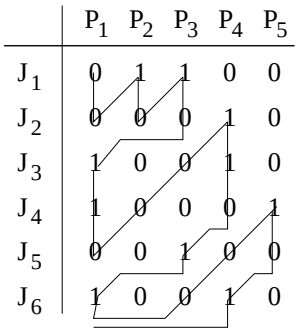
5.3 Der Algorithmus Free-Left

In diesem Abschnitt stellen wir den Online-Algorithmus Free-Left vor, der seine Entscheidungen anhand sehr geringer Information trifft, aber dennoch hervorragende Ergebnisse liefert. Der Algorithmus Free-Left wählt in einer Entscheidungskonfiguration $(\mathcal{T}, p)$ zwischen der nächsten ausführbaren Task links und der nächsten ausführbaren Task rechts nach der folgenden einfachen Regel:

Falls die Maschine P_p an der aktuellen Arbeiterposition nicht von einem Job belegt ist, d.h. falls kein Job J_j mit $\rho^{\mathcal{T}}(j) = p$ existiert, so wähle die nächste ausführbare Task links, sonst wähle die nächste ausführbare Task rechts.

Abgesehen von den guten Ergebnissen, die dieser Algorithmus liefert, ist der Algorithmus Free-Left deswegen so interessant, weil er nur sehr wenig Information benötigt, welche darüber hinaus in der Praxis auch unmittelbar verfügbar ist. Der Arbeiter bearbeitet einfach solange wie möglich die Jobs an seiner Position. Anschließend bewegt er sich nach links, falls die Maschine an seiner Arbeitsposition danach frei bleibt, und nach rechts, falls sich dort ein Job staut. Man beachte, dass diese Regel auch für Nicht-Entscheidungskonfigurationen zutrifft, sofern nicht der letzte Job die Position des Arbeiters bereits passiert hat. In Abbildung 5.5 ist die Free-Left-Abarbeitung einer Taskmatrix Zustand für Zustand angegeben. In der Taskmatrix im oberen Teil der Abbildung sind die Bewegungen des Arbeiters in der gewohnten Weise durch eine Linie dargestellt. Die Free-Left-Abarbeitungen sind in diesen Darstellungen leicht daran zu erkennen, dass die Links-Bewegungen des Arbeiters immer auf horizontalen Abschnitten und die Rechts-Bewegungen des Arbeiters - sofern das Ende der Taskmatrix noch nicht erreicht ist - immer auf diagonalen Abschnitten erfolgen.

Experimentelle Auswertungen an zufälligen Matrizen zeigen, dass der Free-Left-Algorithmus bei linearen Distanzen fast immer bessere Ergebnisse liefert als andere einfache Bewegungsstrategien, wie etwa die im vorigen Abschnitt vorgestellten Algorithmen. Die



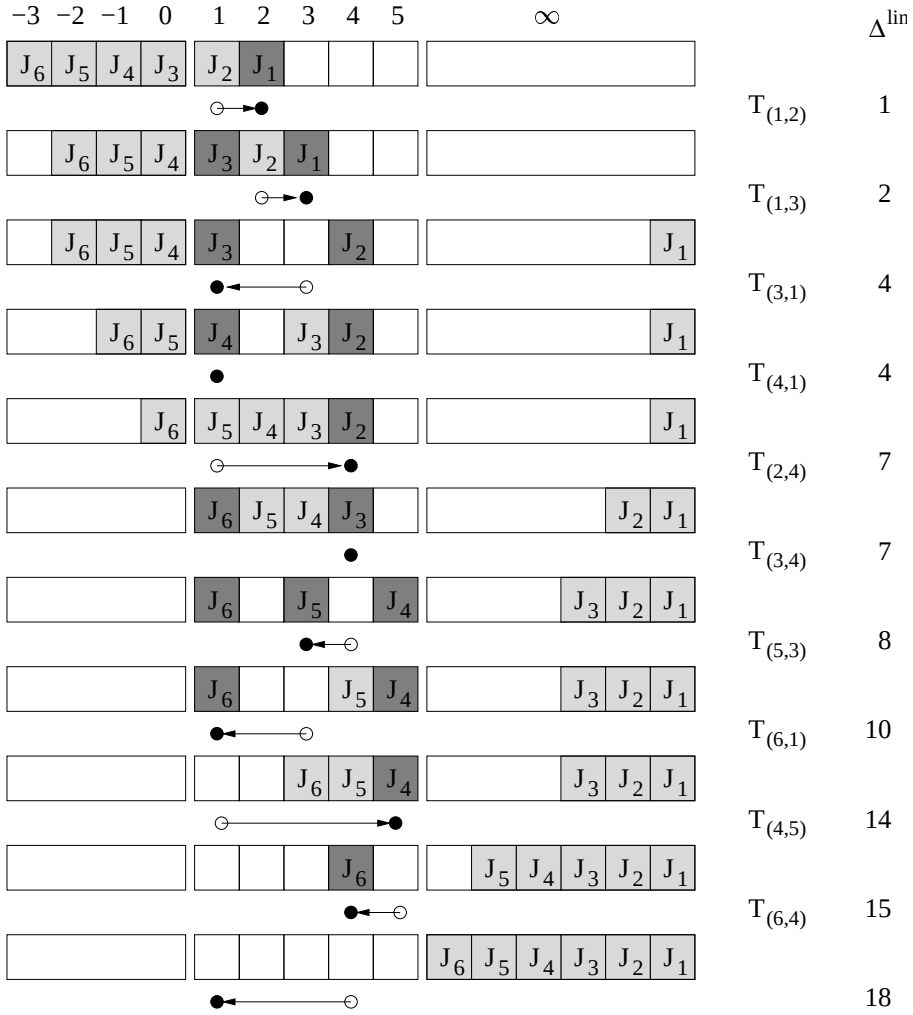


Abbildung 5.5: Free-Left-Abarbeitung der oben angegebenen Taskmatrix mit linearer Distanz 18. Die Positionen der Jobs in den einzelnen Zuständen sind graphisch dargestellt. Rechts ist die jeweils bearbeitete Task angegeben. Zwischen zwei Zuständen ist die Bewegung des Arbeiters mit einem Pfeil angedeutet. Die auf Bearbeitung wartenden Jobs sind dunkler gezeichnet. In der Matrix ist die Abarbeitung in der bekannten Weise durch die eingezeichnete Linie dargestellt.

lineare Distanz der Free-Left-Abarbeitungen liegt dabei fast nie um mehr als 30 Prozent über der unteren Schranke.

Tatsächlich aber hat der Free-Left-Algorithmus selbst für lineare Distanzen keinen konstanten kompetitiven Faktor. Die Beispiele, die das belegen, sind relativ schwierig zu konstruieren und werden erst in Abschnitt 5.3.3 besprochen. Im nächsten Unterabschnitt wollen wir zunächst zeigen, dass der Free-Left-Algorithmus in vielen Fällen optimale Entscheidungen trifft und für einige spezielle Taskmatrizen sogar minimale Abarbeitungen liefert.

5.3.1 Optimale Entscheidungen und minimale Abarbeitungen

Wir wollen zunächst spezielle Matrizen betrachten, für die der Free-Left-Algorithmus minimale Abarbeitungen liefert. Wir nennen die zum Job J_j gehörige Zeile $(\mu_{j,1}, \mu_{j,2}, \ldots, \mu_{j,m})$ der Taskmatrix das *Jobprofil von* J_j. In der Abbildung 5.4 hatten wir eine Taskmatrix angegeben, in der alle Jobs dasselbe Jobprofil haben. Die Abarbeitung im unteren Teil der Abbildung stellt eine Free-Left-Abarbeitung dar. Wir hatten behauptet, dass es sich dabei um eine minimale Abarbeitung handelt. Der folgende Satz beweist dies.

Satz 5.2 *Sei $\mathcal{M}$ eine Taskmatrix für m Maschinen und n Jobs, so dass alle Jobs dasselbe Jobprofil haben. Seien $P_{p_1}, \ldots, P_{p_r}$, $1 \leq p_1 < p_2 < \ldots < p_r \leq m$, genau die Maschinen, an denen alle Jobs bearbeitet werden müssen. Dann ist die Free-Left-Abarbeitung von $\mathcal{M}$ eine minimale Abarbeitung mit der Distanz*

$$2\left(\delta_{1,p_1} + \sum_{k=1}^{r-1} \left\lceil \frac{n}{p_{k+1} - p_k + 1} \right\rceil \delta_{p_k, p_{k+1}}\right).$$

Beweis Sei $1 \leq k < r$. Offensichtlich ist in jeder Abarbeitung die Anzahl der Rechts-P_s-Bewegungen für alle $s \in \{p_k, p_k + 1, \ldots, p_{k+1} - 1\}$ identisch. Nun führt der Arbeiter in der Free-Left-Abarbeitung eine Rechts-P_s-Bewegung für ein solches s nur aus, wenn die Maschine P_{p_k} von einem Job belegt ist, der dort bereits bearbeitet ist, oder wenn es überhaupt keine Jobs links von der Maschine P_{p_k} mehr gibt. Da der Arbeiter andererseits in der Free-Left-Abarbeitung eine Links-P_s-Bewegung für ein solches s nur ausführt, wenn die Maschine $P_{p_{k+1}}$ von keinem Job belegt ist, werden mit Ausnahme der letzten $(n \mod (p_{k+1} - p_k + 1))$ Jobs immer genau $(p_{k+1} - p_k + 1)$ Jobs an Maschine P_{p_k} bearbeitet, bevor die nächste Rechts-P_s-Bewegung für ein $s \in \{p_k, p_k+1, \ldots, p_{k+1}-1\}$ auftritt. Also ist die Anzahl der Rechts-P_s-Bewegungen für ein solches s in der Free-Left-Abarbeitung gleich $\left\lceil \frac{n}{p_{k+1}-p_k+1} \right\rceil$. Da offensichtlich die Anzahl der Rechts-P_s-Bewegungen für $s < p_1$ gleich 1 und für $s \geq p_r$ gleich 0 ist, folgt nach Gleichung (4.2), dass die Free-Left-Abarbeitung genau die im Satz angegebene Distanz hat.

Betrachten wir nun andererseits für $s = 1, \ldots, m-1$ die Anzahl $R_s^{\min}(\mathcal{M})$ der Tasks in der vollständigen P_s-Kette, die rechts von P_s liegen. Es ist offensichtlich, dass $R_s^{\min}(\mathcal{M}) = 1$ für $s < p_1$, $R_s^{\min}(\mathcal{M}) = 0$ für $s \geq p_k$ und

$$R_s^{\min}(\mathcal{M}) = \left\lceil \frac{n}{p_{k+1} - p_k + 1} \right\rceil$$

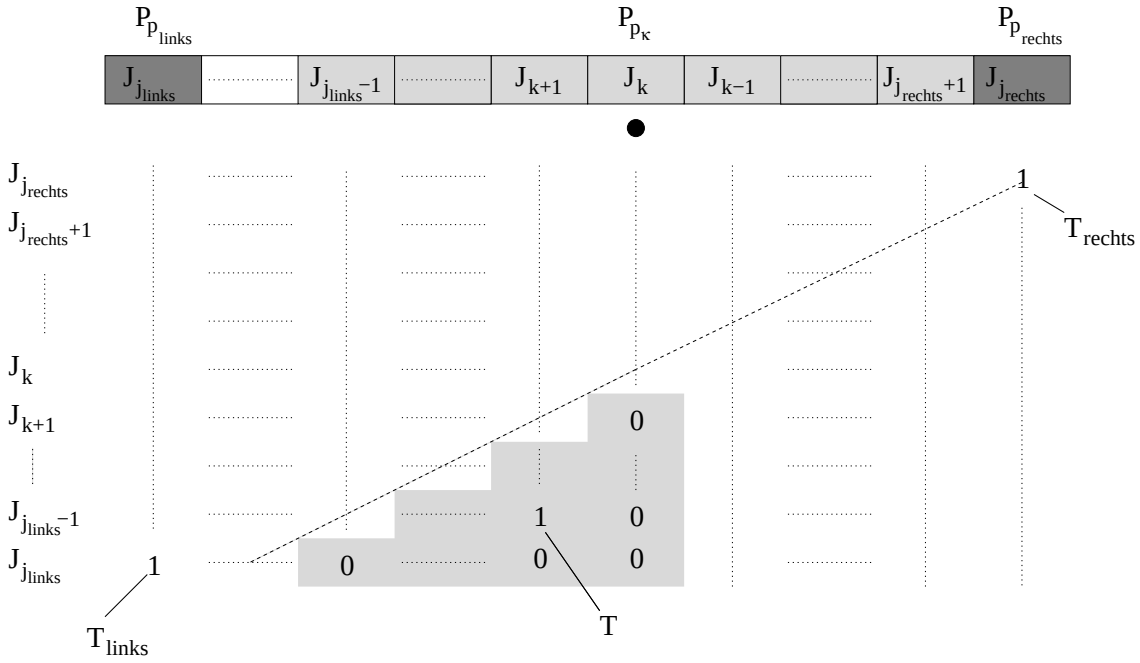


Abbildung 5.6: Konfiguration aus Satz 5.3. Die Jobs $J_{j_{\text{rechts}}}$ und $J_{j_{\text{links}}}$ warten auf Bearbeitung. Die dazwischenliegenden Jobs brauchen an ihren Positionen nicht mehr bearbeitet werden. Der Bereich auf dem Förderband zwischen $J_{j_{\text{links}}-1}$ und $J_{j_{\text{links}}}$ enthält keine Jobs. Die Arbeiterposition an der Maschine $P_{p\kappa}$ ist durch ein • gekennzeichnet. Falls der grau unterlegte Bereich der Taskmatrix eine Task T enthält, so ist die Free-Left-Entscheidung, als nächstes die Task T_{rechts} auszuführen, optimal.

für $s = p_k, p_k + 1, \ldots, p_{k+1} - 1$. Daher ist unsere untere Schranke aus Gleichung (4.3) genau gleich der Distanz der Free-Left-Abarbeitung. Also muss die Free-Left-Abarbeitung minimal sein. □

Die Tatsache, dass Free-Left-Abarbeitungen von Taskmatrizen, in denen alle Jobs dasselbe Jobprofil haben, minimale Abarbeitungen sind, folgt auch aus den beiden folgenden Sätzen. Diese zeigen die Minimalität von Free-Left-Abarbeitungen sogar für etwas allgemeinere Fälle. Darüber hinaus beweisen sie, dass der Free-Left-Algorithmus in vielen Entscheidungskonfigurationen eine optimale Entscheidung trifft.

Satz 5.3 *Sei κ eine Entscheidungskonfiguration, in der die Maschine an der Arbeiterposition p_κ von einem Job J_k belegt ist. Seien $T_{links} = T_{(j_{links}, p_{links})}$ und $T_{rechts} = T_{(j_{rechts}, p_{rechts})}$ die nächsten ausführbaren Tasks links bzw. rechts von P_{p_κ}.*

Falls es eine Task $T = T_{(j,p)} \in \mathcal{T}(\mathcal{M})$ mit $p \leq p_\kappa$, $j \leq j_{links}$ und $j + p > j_{rechts} + p_{rechts}$ gibt, so trifft der Algorithmus Free-Left in der Entscheidungskonfiguration κ bezüglich additiver Distanzen eine optimale Entscheidung, d.h. es gibt eine minimale Restabarbeitung für κ, in der die Task T_{rechts} zuerst bearbeitet wird.

Beweis Bei additiven Distanzen gibt es immer eine minimale Restabarbeitung $\Lambda = T_1, \ldots, T_r$ für κ, für die entweder $T_1 = T_{\text{links}}$ oder $T_1 = T_{\text{rechts}}$ gilt. Falls $T_1 = T_{\text{rechts}}$ ist, so ist nichts zu zeigen. Sei also $T_1 = T_{\text{links}}$ und sei $T = T_{(j,p)}$ eine Task mit $p \leq p_\kappa$, $j \leq j_{\text{links}}$ und $j + p > j_{\text{rechts}} + p_{\text{rechts}}$, siehe Abbildung 5.6. Man betrachte die RL_{p_κ}-Aufteilung

$$\Lambda = L_1, R_1, L_2, R_2, \ldots, L_t,$$

von Λ, d.h. die Aufteilung von Λ in maximale nicht-leere Teilfolgen L_i, R_i, so dass die L_i und R_i nur Tasks links bzw. rechts von P_{p_κ} enthalten. Da T eine Nachfolgertask von T_{rechts} ist, kann T nicht in L_1 enthalten sein, und somit existiert zumindest L_2. Tasks in L_1 müssen offensichtlich einen Jobindex $\geq j_{\text{links}}$ haben, da die noch nicht bearbeiteten Tasks von $J_k, J_{k+1}, \ldots, J_{j_{\text{links}}-1}$ Nachfolger von T_{rechts} sind. Andererseits müssen die Tasks in R_1 einen Jobindex $< j_{\text{links}}$ haben, da sie sonst Nachfolger der Task T wären, welche frühestens in L_2 bearbeitet wird. Also ist keine Task aus L_1 ein Vorgänger einer Task aus R_1. Somit stellt auch die Folge

$$\Lambda' := R_1, L_1, L_2, R_2, \ldots, L_t$$

eine gültige Restabarbeitung dar. Wir zeigen nun, dass für jedes $s \in \{1, \ldots, m-1\}$ die Anzahl der Rechts-P_s-Bewegungen in Λ' bezüglich κ höchstens so groß ist wie die in Λ. Nach Gleichung 5.1 folgt daraus, dass die Distanz von Λ' bezüglich κ höchstens so groß ist wie die von Λ. Wir betrachten dazu die Rechts-P_s-Bewegungen, die in beiden Restabarbeitungen unterschiedlich sind und unterscheiden die Fälle $s \geq p_\kappa$ und $s < p_\kappa$. Für $s \geq p_\kappa$ ist die einzige mögliche Rechts-P_s-Bewegung in Λ', die nicht in Λ vorkommt, die erste Bewegung von der Maschine P_{p_κ} zur ersten Task von R_1. Dafür ist aber die Bewegung von der letzten Task von L_1 zur ersten Task von R_1 aus Λ, die nicht in Λ' vorkommt, ebenfalls eine Rechts-P_s-Bewegung. Also ist für $s \geq p_\kappa$ die Anzahl der Rechts-P_s-Bewegungen in Λ' höchstens so groß wie in Λ. Für $s < p_\kappa$ ist die einzige mögliche Rechts-P_s-Bewegung in Λ', die nicht in Λ vorkommt, die Bewegung von der letzten Task von L_1 zur ersten Task von L_2. Dafür ist aber ebenfalls die Bewegung von der letzten Task von L_1 zur ersten Task von R_1 aus Λ eine Rechts-P_s-Bewegung. Also ist auch für $s < p_\kappa$ die Anzahl der Rechts-P_s-Bewegungen in Λ' höchstens so groß wie in Λ. Somit ist auch Λ' eine minimale Restabarbeitung für κ. Da in Λ' zuerst eine Task rechts von P_{p_κ} bearbeitet wird, gibt es auch eine minimale Restabarbeitung für κ, in der zunächst die Task T_{rechts} bearbeitet wird. □

Satz 5.3 schließt nicht aus, dass es in der gegebenen Konfiguration eine minimale Restabarbeitung gibt, die mit der Task T_{links} beginnt. Es gibt jedoch immer auch eine minimale Restabarbeitung, in der zuerst die Task T_{rechts} bearbeitet wird.

Satz 5.4 *Sei κ eine Entscheidungskonfiguration, in der die Maschine an der Arbeiterposition p_κ von keinem Job belegt ist. Seien $T_{links} = T_{(j_{links}, p_{links})}$ und $T_{rechts} = T_{(j_{rechts}, p_{rechts})}$ die nächsten ausführbaren Tasks links bzw. rechts von P_{p_κ}.*

Falls es eine Task $T = T_{(j,p)} \in \mathcal{T}(\mathcal{M})$ mit $p \geq p_\kappa$, $j \geq j_{links}$ und $j + p \leq j_{rechts} + p_{rechts}$ gibt, so trifft der Algorithmus Free-Left in der Entscheidungskonfiguration κ bezüglich additiver Distanzen eine optimale Entscheidung, d.h. es gibt eine minimale Restabarbeitung für κ, in der die Task T_{links} zuerst bearbeitet wird.

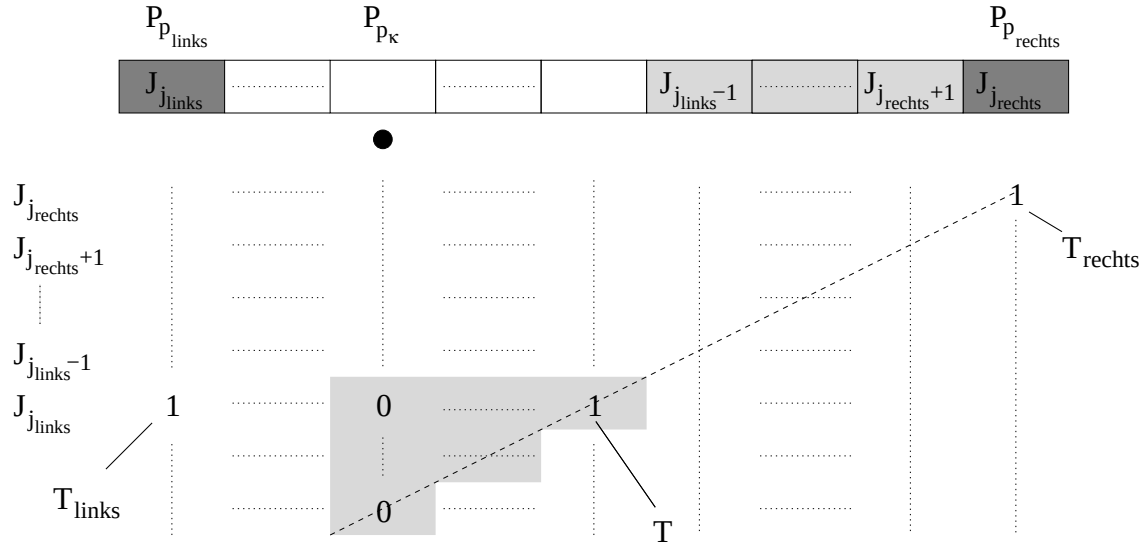


Abbildung 5.7: Konfiguration aus Satz 5.4. Die Jobs $J_{j_{\text{rechts}}}$ und $J_{j_{\text{links}}}$ warten auf Bearbeitung. Die dazwischenliegenden Jobs brauchen an ihren Positionen nicht mehr bearbeitet werden. Der Bereich auf dem Förderband zwischen $J_{j_{\text{links}}-1}$ und $J_{j_{\text{links}}}$ enthält keine Jobs. Die Arbeiterposition an der Maschine P_{p_κ} ist durch ein $\bullet$ gekennzeichnet. Falls der grau unterlegte Bereich der Taskmatrix eine Task T enthält, so ist die Free-Left-Entscheidung, als nächstes die Task T_{links} auszuführen, optimal.

Beweis Für additive Distanzen gibt es immer eine minimale Restabarbeitung $\Lambda = T_1, \ldots, T_r$ für κ, für die entweder $T_1 = T_{\text{links}}$ oder $T_1 = T_{\text{rechts}}$ gilt. Falls $T_1 = T_{\text{links}}$ ist, so ist nichts zu zeigen. Sei also $T_1 = T_{\text{rechts}}$ und sei $T = T_{(j,p)}$ eine Task mit $p \geq p_\kappa$, $j \geq j_{\text{links}}$ und $j + p \leq j_{\text{rechts}} + p_{\text{rechts}}$, siehe Abbildung 5.7. Man betrachte die $\text{RL}_{p_\kappa - 1}$-Aufteilung

$$\Lambda = R_1, L_1, R_2, L_2, \ldots, R_t, L_t,$$

von Λ, d.h. die Aufteilung von Λ in maximale nicht-leere Teilfolgen R_i, L_i, so dass die R_i und L_i nur Tasks rechts bzw. links von $P_{p_\kappa - 1}$ enthalten. Da T eine Nachfolgertask von T_{links} ist, kann T nicht in R_1 enthalten sein, und somit existiert zumindest R_2. Für Tasks $T' = T_{(j',p')}$ aus R_1 gilt $j' + p' \geq j_{\text{rechts}} + p_{\text{rechts}}$. Dies folgt aus Lemma 2.1, da sich der Job $J_{j_{\text{links}}-1}$ an Position $p_{\text{rechts}} - (j_{\text{links}} - 1 - j_{\text{rechts}})$ befindet und für die Tasks $T_{(j',p')}$ aus R_1 offensichtlich $j' < j_{\text{links}}$ gilt. Für die Tasks $T' = T_{(j',p')}$ in L_1 gilt andererseits $j' + p' \leq j + p \leq j_{\text{rechts}} + p_{\text{rechts}}$, da sie sonst Nachfolger der Task T wären, die frühestens in R_2 bearbeitet wird. Also ist keine Task aus R_1 ein Vorgänger einer Task aus L_1. Somit stellt auch die Folge

$$\Lambda' := L_1, R_1, R_2, L_2, \ldots, R_t, L_t$$

eine gültige Restabarbeitung dar. Wir vollenden den Beweis wiederum, indem wir zeigen, dass für jedes $s \in \{1, \ldots, m-1\}$ die Anzahl der Rechts-P_s-Bewegungen in Λ' bezüglich κ höchstens so groß ist wie die in Λ. Wir betrachten dazu wieder die Rechts-P_s-Bewegungen,

die in beiden Restabarbeitungen unterschiedlich sind und unterscheiden die Fälle $s < p_\kappa$ und $s \geq p_\kappa$. Für $s < p_\kappa$ ist die einzige mögliche Rechts-P_s-Bewegung in Λ', die nicht in Λ vorkommt, die Bewegung von der letzten Task von L_1 zur ersten Task von R_1. Dafür ist aber die Bewegung von der letzten Task von L_1 zur ersten Task von R_2 aus Λ, die nicht Λ' vorkommt, ebenfalls eine Rechts-P_s-Bewegung. Also ist für $s < p_\kappa$ die Anzahl der Rechts-P_s-Bewegungen in Λ' höchstens so groß wie in Λ.

Für $s \geq p_\kappa$ gibt es zwei mögliche Rechts-P_s-Bewegungen in Λ', die nicht in Λ vorkommen, nämlich die Bewegung von der letzten Task von L_1 zur ersten Task von R_1 und die Bewegung von der letzten Task von R_1 zur ersten Task von R_2. Für den ersten Fall ist auch die erste Bewegung in Λ von der Maschine P_{p_κ} zur ersten Task von R_1 eine Rechts-P_s-Bewegung, für den zweiten Fall ist die Bewegung in Λ von der letzten Task von L_1 zur ersten Task von R_2, die in Λ' nicht vorkommt, eine Rechts-P_s-Bewegung. Also ist auch für $s \geq p_\kappa$ die Anzahl der Rechts-P_s-Bewegungen in Λ' höchstens so groß wie in Λ. Somit ist Λ' ebenfalls eine minimale Restabarbeitung für κ. Da in Λ' zuerst eine Task links von P_{p_κ} bearbeitet wird, gibt es auch eine minimale Restabarbeitung für κ, in der zunächst die Task T_{links} bearbeitet wird. □

Satz 5.4 schließt wiederum nicht aus, dass es in der gegebenen Konfiguration eine minimale Restabarbeitung gibt, die mit der Task T_{rechts} beginnt.

Sei $\mathcal{M} = (\mu_{j,p})_{1 \leq j \leq n, 1 \leq p \leq m}$ eine Taskmatrix. Wir nennen $\mathcal{M}$ *spaltenmonoton*, wenn für $j = 1, 2, \ldots, n-1$ und $p = 1, \ldots, m$ gilt:

$$\mu_{j,p} = 1 \quad \Longrightarrow \quad \mu_{j+1,p} = 1.$$

Man beachte, dass $\mathcal{M}$ spaltenmonoton ist, falls alle Jobs dasselbe Jobprofil haben. Wir nennen $\mathcal{M}$ *zeilenmonoton*, wenn für $j = 1, 2, \ldots, n$ und $p = 1, \ldots, m-1$ gilt:

$$\mu_{j,p} = 1 \quad \Longrightarrow \quad \mu_{j,p+1} = 1.$$

Aus den beiden vorigen Sätzen erhalten wir folgendes Korollar:

Korollar 5.1 *Sei $\mathcal{M}$ eine zeilen- oder spaltenmonotone Taskmatrix. Dann ist die Free-Left-Abarbeitung von $\mathcal{M}$ eine minimale Abarbeitung bezüglich additiver Distanzen.*

Beweis Man betrachte eine Entscheidungskonfiguration κ in der Free-Left-Abarbeitung, wobei $T_{(j_{\text{links}}, p_{\text{links}})}$ die nächste ausführbare Task links von P_{p_κ} ist. Dann wurde zuletzt eine Task $T_{(j,p)}$ mit $p = p_\kappa$ und $j < j_{\text{links}}$ bearbeitet. Da $\mathcal{M}$ zeilen- oder spaltenmonoton ist, ist dann auch $T_{(j_{\text{links}}, p_\kappa)}$ eine Task. Diese erfüllt die Bedingungen aus Satz 5.3 bzw. 5.4. Also trifft Free-Left in der Entscheidungskonfiguration κ eine optimale Entscheidung. Da dies für alle Entscheidungskonfigurationen zutrifft, muss die resultierende Abarbeitung minimal sein. □

5.3.2 Eine $O(\log m)$-Schranke für den kompetitiven Faktor

In diesem Abschnitt werden wir beweisen, dass der kompetitive Faktor von Free-Left in $O(\log_2 m)$ liegt. Der Beweis beruht darauf, dass die Anzahl der Rechts-P_s-Bewegungen

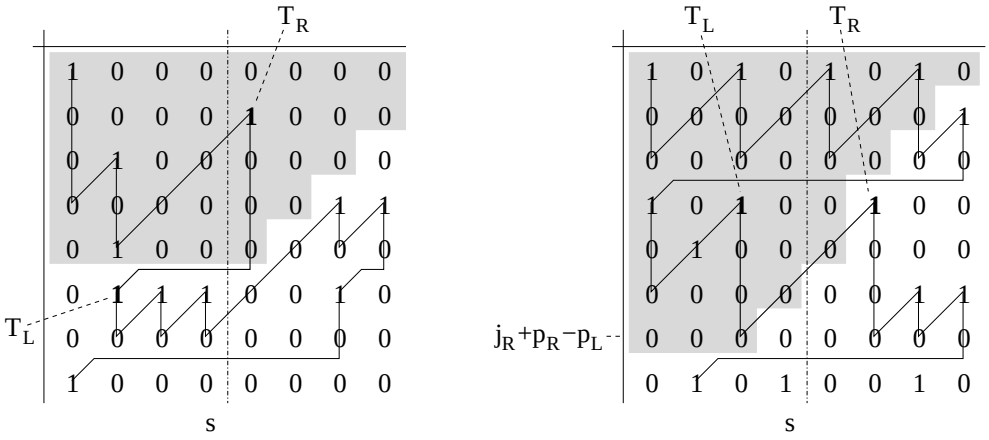


Abbildung 5.8: Zum Beweis von Lemma 5.1. Die Tasks in den grau unterlegten Bereiche sind in der Free-Left-Abarbeitung nach Bearbeitung der ersten Task T_R der Linksbewegung (T_R, T_L) (Bild links) bzw. nach der Bearbeitung der ersten Task T_L der Rechtsbewegung (T_L, T_R) (Bild rechts) bereits bearbeitet.

in der Free-Left-Abarbeitung höchstens um einen Faktor aus $O(\log_2 m)$ größer ist als die minimale Anzahl $R_s^{\min}(\mathcal{M})$ von Rechts-P_s-Bewegungen, die jede Abarbeitung enthalten muss. Indem wir gegebenenfalls m Jobs ohne Tasks am Ende der Taskmatrix hinzufügen, nehmen wir in diesem Abschnitt zur Vereinfachung der Beweise an, dass die letzten m Jobs keine Tasks besitzen. Dies ändert offensichtlich nichts an den Abarbeitungen. Wir erreichen dadurch jedoch, dass eine Rechts-Bewegung in einer Free-Left-Abarbeitung nur dann auftreten kann, wenn die Maschine an der Arbeiterposition von einem Job belegt ist.

Für eine Bewegung (T, T') nennen wir T die *Ausgangstask der Bewegung* und T' die *Zieltask der Bewegung*. Das folgende Lemma und das folgende Korollar zeigen einfache Eigenschaften einer Free-Left-Abarbeitung, die wir im Folgenden immer wieder benutzen werden. Am einfachsten verdeutlicht man sich die Aussagen anhand von Abbildung 5.8.

Lemma 5.1 *Sei Λ die Free-Left-Abarbeitung der Taskmatrix $\mathcal{M}$ und seien $T_L = T_{(j_L,p_L)}$ und $T_R = T_{(j_R,p_R)}$ zwei Tasks aus $\mathcal{T}(\mathcal{M})$.*

1. *Wenn (T_R, T_L) eine Linksbewegung in Λ ist, so sind nach der Bearbeitung der Ausgangstask T_R in Λ alle Tasks $T_{(j,p)}$ mit $j < j_L$ und $j + p < j_L + p_R$ bereits ausgeführt.*

2. *Wenn (T_L, T_R) eine Rechtsbewegung in Λ ist, so sind nach der Ausführung der Ausgangstask T_L in Λ alle Tasks $T = T_{(j,p)}$ mit $j \leq j_R + p_R - p_L$, für die $j + p < j_R + p_R$ oder $j + p = j_R + p_R$ und $p < p_R$ gilt, bereits ausgeführt.*

Beweis

1. Wenn (T_R, T_L) eine Links-Bewegung in Λ ist, so folgt aus der Definition einer Free-Left-Abarbeitung, dass nach der Ausführung von T_R die Maschinen $P_{p_L+1}, \ldots, P_{p_R}$

frei sind und der Job J_{j_L} an der Maschine P_{p_L} auf Bearbeitung wartet. Dann muss sich der Job J_{j_L-1} mindestens an der Position p_R+1 befinden. Nach Lemma 2.1 folgt daraus, dass alle Tasks $T_{(j,p)}$ mit $j < j_L$ und $j+p < j_L-1+p_R+1 = j_L+p_R$ bereits ausgeführt sind.

2. Wenn (T_L, T_R) eine Rechts-Bewegung in Λ ist, so folgt aus der Definition einer Free-Left-Abarbeitung (und unseren speziellen Voraussetzungen an die Taskmatrix), dass nach Bearbeitung der Task T_L die Maschine P_{p_L} belegt ist, und zwar von dem Job $J_{j_R+p_R-p_L}$. Nach Lemma 2.1 folgt daraus, dass alle Tasks $T_{(j,p)}$ mit $j \leq j_R+p_R-p_L$ und $j+p < j_R+p_R$ bereits ausgeführt sind. Eine noch nicht ausgeführte Task $T_{(j,p)}$ mit $j \leq j_R+p_R-p_L$, $p < p_R$ und $j+p = j_R+p_R$ kann nicht existieren, da diese dann ausführbar und somit T_R nicht die nächste ausführbare Task rechts von P_{p_L} wäre. □

Korollar 5.2 *Sei Λ die Free-Left-Abarbeitung der Taskmatrix $\mathcal{M}$ und sei s, $1 \leq s \leq m-1$, ein Maschinenindex.*

1. *Sei M ein Links-P_s-Bewegung in Λ mit Zieltask $T_{(j_L,p_L)}$ und sei $T = T_{(j,p)}$ eine Task, die nach Ausführung der Ausgangstask von M in Λ noch nicht ausgeführt ist. Dann gilt:*

   (i) $p \leq s \implies j \geq j_L$,

   (ii) $p > s \implies j+p \geq j_L+s+1$.

2. *Sei M ein Rechts-P_s-Bewegung in Λ mit Zieltask $T_{(j_R,p_R)}$ und sei $T = T_{(j,p)}$ eine Task, die nach Ausführung der Ausgangstask von M in Λ noch nicht ausgeführt ist. Dann gilt:*

   (i) $p \leq s \implies j \geq j_R+p_R-s+1$,

   (ii) $p > s \implies j+p \geq j_R+p_R$.

3. *Seien M_1, M_2 zwei Links-P_s-Bewegungen in Λ mit den Zieltasks $T_{(j_1,p_1)}$ bzw. $T_{(j_2,p_2)}$, so dass M_2 nach M_1 erfolgt. Dann gilt $j_2 > j_1+1$.*

4. *Seien M_1, M_2 zwei Rechts-P_s-Bewegungen in Λ mit den Zieltasks $T_{(j_1,p_1)}$ bzw. $T_{(j_2,p_2)}$, so dass M_2 nach M_1 erfolgt. Dann gilt $j_2+p_2 > j_1+p_1+1$.*

Beweis

1. Folgt unmittelbar aus Lemma 5.1 (1.).

2. Folgt unmittelbar aus Lemma 5.1 (2.).

3. Man betrachte die nächste auf M_1 folgende Rechts-P_s-Bewegung $M = (T_{(k,q)}, T_{(k',q')})$. Nach (1.) gilt $k'+q' \geq j_1+s+1$. Nach (2.) gilt andererseits $j_2 > k'+q'-s$. Insgesamt folgt also $j_2 > k'+q'-s \geq j_1+1$.

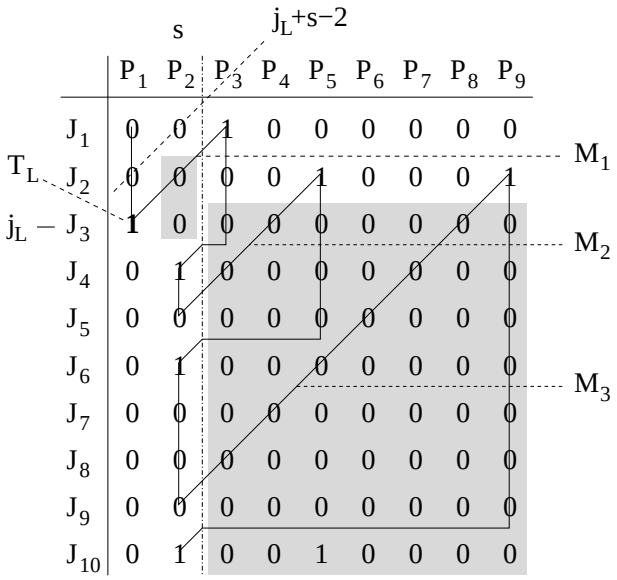


Abbildung 5.9: Zum Beweis von Lemma 5.2. Der grau unterlegte Bereich links der Trennlinie enthält aufgrund der Definition einer $L_s^\star$-Task keine Tasks. Auf der rechten Seite ist der Bereich der Nachfolger von T_L, die rechts von P_s liegen, grau unterlegt.

4. Man betrachte die nächste auf M_1 folgende Links-P_s-Bewegung $M = (T_{(k,q)}, T_{(k',q')})$. Nach (2.) gilt $k' \geq j_1 + p_1 - s + 1$. Nach (1.) gilt andererseits $j_2 + p_2 > k' + s$. Insgesamt folgt also $j_2 + p_2 > k' + s \geq j_1 + p_1 + 1$. □

Die Punkte (3.) und (4.) des obigen Korollars bedeuten, dass in der Darstellung der Free-Left-Abarbeitung in der Taskmatrix der Abstand zwischen zwei horizontalen bzw. zwischen zwei diagonalen Linien an einer Maschine mindestens 2 beträgt.

Zur Verdeutlichung des folgenden Lemmas siehe auch Abbildung 5.9.

Lemma 5.2 *Sei s, $1 \leq s \leq m-1$, ein Maschinenindex, $T_L = T_{(j_L,p_L)}$ eine $L_s^\star$-Task von $\mathcal{M}$ und Λ die Free-Left-Abarbeitung von $\mathcal{M}$. Für ein $r \geq 2$ sei M_r die r-te Rechts-P_s-Bewegung nach der Ausführung von T_L in Λ, so dass bis einschließlich der Ausführung der Zieltask $T_{(j_r,p_r)}$ von M_r in Λ kein Nachfolger von T_L rechts von P_s bearbeitet wurde. Dann gilt*

$$j_r + p_r \geq j_L + s - 2 + 2^r.$$

Beweis Wir zeigen das Lemma durch vollständige Induktion über r.

<u>$r = 2$:</u> Nach Ausführung der $L_s^\star$-Task T_L sind alle Tasks $T_{(j,p)}$ mit $j \leq j_L$, die links von P_s liegen, bereits bearbeitet, da alle diese Tasks, die von T_L verschieden sind, nach Definition einer $L_s^\star$-Task Vorgänger von T_L sind. Für den Jobindex k der Zieltask der ersten Links-P_s-Bewegung nach der Ausführung von T_L gilt daher $k \geq j_L + 1$. Nach Korollar 5.2 (1) folgt dann für die Zieltask $T_{(j_2,p_2)}$ der zweiten Rechts-P_s-Bewegung nach Ausführung von T_L

$$j_2 + p_2 \geq k + s + 1 \geq j_L + s + 2 = j_L + s - 2 + 2^2.$$

$\underline{r > 2:}$ Sei $T_{(j_{r-1},p_{r-1})}$ die Zieltask der vorhergehenden Rechts-P_s-Bewegung M_{r-1} und sei $(T_{(k,q)}, T_{(k',q')})$ die Links-P_s-Bewegung zwischen M_{r-1} und M_r. Nach Korollar 5.2 (2.) folgt

$$k + q \geq j_{r-1} + p_{r-1}, \tag{5.2}$$

$$k' \geq j_{r-1} + p_{r-1} - s + 1. \tag{5.3}$$

Nach Induktionsvoraussetzung folgt aus der letzten Ungleichung $k' > j_L$. Da $T_{(j_r,p_r)}$ nach Voraussetzung kein Nachfolger von T_L ist, ist andererseits $j_r < j_L$ und somit $j_r < k'$. Aus Lemma 5.1 (1.) folgt daher

$$j_r + p_r \geq k' + q. \tag{5.4}$$

Da auch $T_{(k,q)}$ nach Voraussetzung kein Nachfolger von T_L ist, gilt $k \leq j_L - 1$, und somit folgt aus (5.4)

$$j_r + p_r \geq k' + q \underset{(5.2)}{\geq} k' + j_{r-1} + p_{r-1} - k \underset{(5.3)}{\geq} 2(j_{r-1} + p_{r-1}) - s + 1 - k \geq 2(j_{r-1} + p_{r-1}) - s - j_L + 2.$$

Da nach Induktionsvoraussetzung

$$j_{r-1} + p_{r-1} \geq j_L + s - 2 + 2^{r-1}$$

gilt, folgt daraus die Induktionsbehauptung. □

Korollar 5.3 *Sei s, $1 \leq s \leq m-1$, ein Maschinenindex und $T_L = T_{(j_L,p_L)}$ eine $L_s^\star$-Task von $\mathcal{M}$. Für ein $r \geq 1$ sei M die r-te Rechts-P_s-Bewegung in der Free-Left-Abarbeitung Λ von $\mathcal{M}$ nach Ausführung von T_L, so dass nach Bearbeitung der Zieltask von M in Λ noch kein Nachfolger von T_L auf der rechten Seite von P_s ausgeführt wurde. Dann gilt*

$$r \leq \lfloor \log_2(m - s + 1) \rfloor$$

Beweis Für $r = 1$ ist die Aussage trivial. Andernfalls gilt nach Lemma 5.2 für die Zieltask $T_{(j,p)}$ von M

$$j + p \geq j_L + s - 2 + 2^r.$$

Da $T_{(j,p)}$ kein Nachfolger von T_L ist, folgt $j \leq j_L - 1$ und damit

$$p \geq j_L + s - 2 + 2^r - j \geq 2^r + s - 1.$$

Wegen $p \leq m$ folgt daraus das Korollar. □

Korollar 5.4 *Sei s, $1 \leq s \leq m-1$, ein Maschinenindex und (T_L, T_R) ein LR_s-Paar von $\mathcal{M}$. Dann gibt es in der Free-Left-Abarbeitung Λ von $\mathcal{M}$ höchstens*

$$1 + \lfloor \log_2(m - s + 1) \rfloor$$

Rechts-P_s-Bewegungen zwischen der Bearbeitung von T_L und der Bearbeitung von T_R.

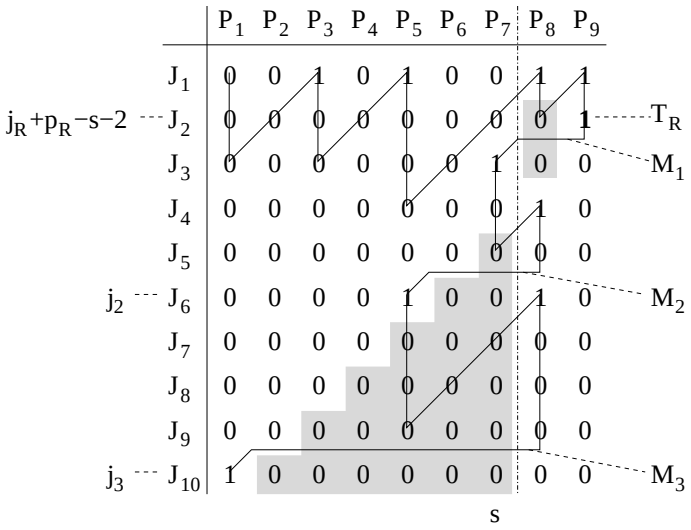


Abbildung 5.10: Zum Beweis von Lemma 5.3. Der grau unterlegte Bereich rechts der Trennlinie enthält aufgrund der Definition einer $R_s^\star$-Task keine Tasks. Auf der linken Seite ist der Bereich der Nachfolger von T_R, die links von P_s liegen, grau unterlegt.

Beweis Sei $T_R = T_{(j_R,p_R)}$ und $r = 1 + \lfloor \log_2(m-s+1) \rfloor$. Angenommen, es gibt in Λ mehr als r Rechts-P_s-Bewegungen zwischen der Ausführung von T_L und T_R. Dann sei M die r-te dieser Bewegungen. Nach Korollar 5.3 ist spätestens mit der Ausführung der Zieltask $T_{(j,p)}$ von M ein Nachfolger von T_L rechts von P_s bearbeitet. Da (T_L, T_R) ein LR_s-Paar ist, folgt aus Lemma 4.5 (1.), dass nach Ausführung der Ausgangstask von M alle Tasks $T_{(j',p')}$ mit $j'+p' < j_R+p_R$, die rechts von P_s liegen, bereits ausgeführt sind. Also gilt für die Zieltask $T_{(j,p)}$ von M, dass $j+p \geq j_R+p_R$. Nach Korollar 5.2 (4.) gilt für die Zieltask $T_{(j',p')}$ der nächsten Rechts-P_s-Bewegung M' nach M daher $j'+p' > j+p \geq j_R+p_R$. Nach Korollar 5.2 (2.) ist T_R dann bereits vor der Rechts-P_s-Bewegung M' ausgeführt. Das ist ein Widerspruch zu der Annahme, dass es mehr als r Rechts-P_s-Bewegungen zwischen der Bearbeitung von T_L und T_R gibt. □

Zur Verdeutlichung des folgenden Lemmas siehe auch Abbildung 5.10.

Lemma 5.3 *Sei s, $1 \leq s \leq m-1$, ein Maschinenindex, $T_R = T_{(j_R,p_R)}$ eine $R_s^\star$-Task von $\mathcal{M}$ und Λ die Free-Left-Abarbeitung von $\mathcal{M}$. Für ein $r \geq 2$ sei M_r die r-te Links-P_s-Bewegung nach der Ausführung von T_R in Λ, so dass bis einschließlich der Ausführung der Zieltask $T_{(j_r,p_r)}$ von M_r kein Nachfolger von T_R links von P_s bearbeitet wurde. Dann gilt*

$$j_r \geq j_R + p_R - s - 2 + 2^r.$$

Beweis Wir zeigen das Lemma durch vollständige Induktion über r.

<u>$r = 2$:</u> In der Free-Left-Abarbeitung kann eine Links-P_s-Bewegung nach Ausführung von T_R erst dann auftreten, wenn sich der Job J_{j_R} an einer Position $\geq p_R + 1$ befindet. Somit sind zu diesem Zeitpunkt alle Tasks $T_{(j,p)}$ mit $j \leq j_R$ und $j+p \leq j_R+p_R$ bereits

ausgeführt. Da T_R eine $R_s^\star$-Task ist, gibt es keine Tasks $T_{(j,p)}$ rechts von P_s mit $j > j_R$ und $j + p \leq j_R + p_R$. Also sind nach der ersten Links-P_s-Bewegung nach Ausführung der Task T_R alle Tasks $T_{(j,p)}$ rechts von P_s mit $j + p \leq j_R + p_R$ bereits bearbeitet. Daher gilt für die Zieltask $T_{(k,q)}$ der ersten Rechts-P_s-Bewegung nach der Ausführung von T_R, dass $k + q \geq j_R + p_R + 1$. Nach Korollar 5.2 (2.) folgt dann für die Zieltask $T_{(j_2,p_2)}$ der zweiten Links-P_s-Bewegung nach Ausführung von T_R

$$j_2 \geq k + q - s + 1 \geq j_R + p_R - s + 2 = j_R + p_R - s - 2 + 2^2.$$

$\underline{r > 2}$: Sei $T_{(j_{r-1},p_{r-1})}$ die Zieltask der vorhergehenden Links-P_s-Bewegung M_{r-1} und sei $M = (T_{(k,q)}, T_{(k',q')})$ die Rechts-P_s-Bewegung zwischen M_{r-1} und M_r. Nach Korollar 5.2 (1.) folgt

$$k \geq j_{r-1}, \tag{5.5}$$

$$k' + q' \geq j_{r-1} + s + 1. \tag{5.6}$$

Nach Induktionsvoraussetzung folgt aus der letzten Ungleichung $k' + q' > j_R + p_R$. Da die Zieltask $T_{(j_r,p_r)}$ der Links-P_s-Bewegung M_r nach Voraussetzung kein Nachfolger von T_R ist, gilt andererseits $j_r + p_r \leq j_R + p_R$, also $j_r + p_r < k' + q'$. Nach Lemma 5.1 (2.) folgt dann

$$j_r \geq k' + q' - q + 1. \tag{5.7}$$

Da auch die Ausgangstask $T_{(k,q)}$ der Rechts-P_s-Bewegung M nach Voraussetzung kein Nachfolger von T_R ist, folgt

$$k + q \leq j_R + p_R. \tag{5.8}$$

Somit folgt aus (5.7)

$$\begin{aligned} j_r &\geq k' + q' - q + 1 \\ &\underset{(5.6)}{\geq} j_{r-1} + s - q + 2 \\ &\underset{(5.8)}{\geq} j_{r-1} + s - j_R - p_R + k + 2 \\ &\underset{(5.5)}{\geq} 2j_{r-1} + s - j_R - p_R + 2. \end{aligned}$$

Da nach Induktionsvoraussetzung $j_{r-1} \geq j_R + p_R - s - 2 + 2^{r-1}$ gilt, folgt daraus die Induktionsbehauptung. □

Korollar 5.5 *Sei s, $1 \leq s \leq m-1$, ein Maschinenindex und $T_R = T_{(j_R,p_R)}$ eine $R_s^\star$-Task von $\mathcal{M}$. Für ein $r \geq 1$ sei M die r-te Links-P_s-Bewegung in der Free-Left-Abarbeitung Λ von $\mathcal{M}$ nach Ausführung von T_R, so dass nach Bearbeitung der Zieltask von M in Λ noch kein Nachfolger von T_R auf der linken Seite von P_s ausgeführt wurde. Dann gilt*

$$r \leq \lfloor \log_2(s+1) \rfloor.$$

Beweis Für $r = 1$ ist die Aussage trivial. Andernfalls gilt nach Lemma 5.3 für die Zieltask $T_{(j,p)}$ von M

$$j \geq j_R + p_R - s - 2 + 2^r.$$

Da $T_{(j,p)}$ kein Nachfolger von T_R ist, folgt $j + p \leq j_R + p_R$ und damit

$$p \leq j_R + p_R - j \leq s + 2 - 2^r.$$

Wegen $p \geq 1$ folgt daraus das Korollar. □

Korollar 5.6 *Sei s, $1 \leq s \leq m-1$, ein Maschinenindex und (T_R, T_L) ein RL_s-Paar von $\mathcal{M}$. Dann gibt es in der Free-Left-Abarbeitung Λ von $\mathcal{M}$ höchstens*

$$1 + \lfloor \log_2(s+1) \rfloor$$

Links-P_s-Bewegungen und somit höchstens

$$\lfloor \log_2(s+1) \rfloor$$

Rechts-P_s-Bewegungen zwischen der Bearbeitung von T_R und der Bearbeitung von T_L.

Beweis Sei $T_L = T_{(j_L,p_L)}$ und $r = 1 + \lfloor \log_2(s+1) \rfloor$. Angenommen, es gibt in Λ mehr als r Links-P_s-Bewegungen zwischen der Bearbeitung von T_R und T_L. Dann sei M die r-te dieser Bewegungen. Nach Korollar 5.5 ist spätestens mit der Ausführung der Zieltask $T_{(j,p)}$ von M ein Nachfolger von T_R links von P_s bearbeitet. Da (T_R, T_L) ein RL_s-Paar ist, folgt aus Lemma 4.5 (2.), dass nach Ausführung der Ausgangstask von M alle Tasks $T_{(j',p')}$ mit $j' < j_L$, die links von P_s liegen, bereits ausgeführt sind. Also gilt $j \geq j_L$. Nach Korollar 5.2 (3.) gilt für die Zieltask $T_{(j',p')}$ der nächsten Links-P_s-Bewegung M' nach M, dass $j' > j \geq j_L$. Nach Korollar 5.2 (1.) ist T_L dann bereits vor der Links-P_s-Bewegung M' ausgeführt. Das ist ein Widerspruch zu der Annahme, dass es mehr als r Links-P_s-Bewegungen zwischen der Bearbeitung von T_R und T_L gibt. □

Satz 5.5 *Der Algorithmus Free-Left ist $(2\log_2(m+2)-1)$-kompetitiv bezüglich additiver Distanzen.*

Beweis Wir beweisen den Satz, indem wir zeigen, dass die Anzahl der Rechts-P_s-Bewegungen für $s = 1, \ldots, m-1$ in der Free-Left-Abarbeitung Λ einer Taskmatrix $\mathcal{M}$ höchstens um den Faktor $(2\log_2(m+2)-1)$ höher ist als die minimal erforderliche Anzahl $R_s^{\min}(\mathcal{M})$.

Sei $s \in \{1, \ldots, m-1\}$ und sei $T_0, T_1, \ldots, T_z$ die vollständige s-Kette von $\mathcal{M}$. Dann ist $T_0 = T_{(0,0)}$, T_z ist eine $L_s^\star$-Task und $z = 2 \cdot R_s^{\min}(\mathcal{M})$. Offensichtlich kann es in Λ nur eine Rechts-P_s-Bewegung vor Bearbeitung der Task T_1 geben. Nach Korollar 5.6 und Korollar 5.4 gibt es für $i = 1, \ldots, z-1$ höchstens $\lfloor \log_2(s+1) \rfloor$ Rechts-P_s-Bewegungen zwischen der Bearbeitung von T_i und T_{i+1}, falls i ungerade ist, und höchstens $1 + \lfloor \log_2(m-s+1) \rfloor$ Rechts-P_s-Bewegungen, falls i gerade ist. Da T_z keinen Nachfolger rechts von P_s besitzt, gibt es

nach Korollar 5.3 höchstens $\lfloor \log_2(m-s+1) \rfloor$ Rechts-P_s-Bewegungen nach Ausführung von T_z. Also gibt es insgesamt höchstens

$$R_s^{\min}\left(1 + \lfloor \log_2(m-s+1) \rfloor + \lfloor \log_2(s+1) \rfloor\right)$$

Rechts-P_s-Bewegungen in Λ. Wegen

$$\begin{aligned} 1 + \lfloor \log_2(m-s+1) \rfloor + \lfloor \log_2(s+1) \rfloor &\leq 1 + \log_2(m-s+1) + \log_2(s+1) \\ &\leq 1 + \log_2\left((m-s+1)(s+1)\right) \\ &\leq 1 + \log_2\left(\left(\frac{m+2}{2}\right)^2\right) \\ &= 2\log_2(m+2) - 1 \end{aligned}$$

folgt der Satz. □

5.3.3 Eine untere Schranke für den kompetitiven Faktor

Die Verallgemeinerung des Beispiels aus Abbildung 5.11 zeigt, dass die Anzahl der Rechts-P_s-Bewegungen in einer Free-Left-Abarbeitung für ein einzelnes s tatsächlich um einen Faktor aus $\Omega(\log m)$ über der Anzahl der minimal benötigten Rechts-P_s-Bewegungen liegen kann. Für allgemeine additive Distanzen lässt sich aus diesen Beispielen ableiten, dass der kompetitive Faktor von Free-Left in $\Omega(\log m)$ liegt, indem man $\delta_{1,2} = 1$ und $\delta_{p,p+1} = 0$ für $p > 1$ wählt. Dies gilt jedoch nicht für lineare Distanzen. Die lineare Distanz der Free-Left-Abarbeitung bei diesen Beispielen liegt nämlich nur um einen Faktor ≤ 2 über der Distanz einer minimalen Abarbeitung. Die Verallgemeinerung des einfachen Beispiels aus Abbildung 5.12 auf eine beliebige Anzahl m von Maschinen zeigt immerhin, dass der kompetitive Faktor von Free-Left für lineare Distanzen mindestens 3 ist. Wir wollen jetzt sogar zeigen, dass er nicht konstant ist, sondern in $\Omega\left(\frac{\log_2 m}{\log_2 \log_2 m}\right)$ liegt.

Zu diesem Zweck werden wir für alle natürlichen Zahlen $c \geq 2$ und $r \geq 1$ Taskmatrizen $\mathcal{F}_{c,r}$ definieren. Für $m \geq 1$ sei $\mathcal{B}(m)$ die Taskmatrix für $m+1$ Jobs und m Maschinen, die als einzige Task die Task $T_{(1,m)}$ besitzt, d.h. $\mathcal{B}(m)$ hat das folgende Aussehen:

$$\mathcal{B}(m) := \overbrace{\left.\begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{pmatrix}\right\} m+1}^{m}$$

Für $m \geq 2$ sei $\mathcal{A}(m)$ die Taskmatrix für $m-1$ Jobs und m Maschinen, deren einzigen

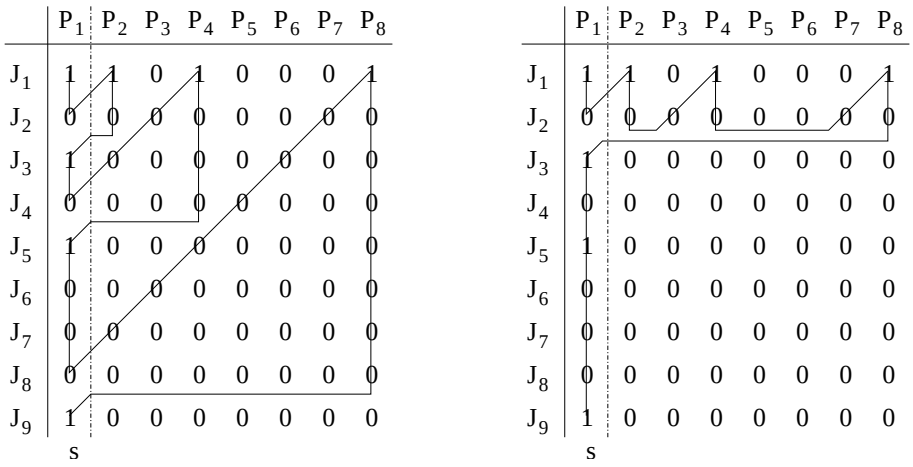


Abbildung 5.11: Free-Left-Abarbeitung (links) und minimale Abarbeitung (rechts) einer Taskmatrix $\mathcal{M}$. Verallgemeinert man das Beispiel auf $m = 2^k$ Maschinen, so ist die minimal erforderliche Anzahl $R_1^{\min}(\mathcal{M})$ von Rechts-P_1-Bewegungen gleich 1, während die Free-Left-Abarbeitung $k = \log_2 m$ Rechts-P_1-Bewegungen aufweist. Die lineare Distanz einer minimalen Abarbeitung beträgt $2(m-1)$, während die Free-Left-Abarbeitung die Distanz $2\sum_{i=1}^{k}(2^i-1) = 2(2^{k+1}-2-k) \leq 4(m-1)$ hat.

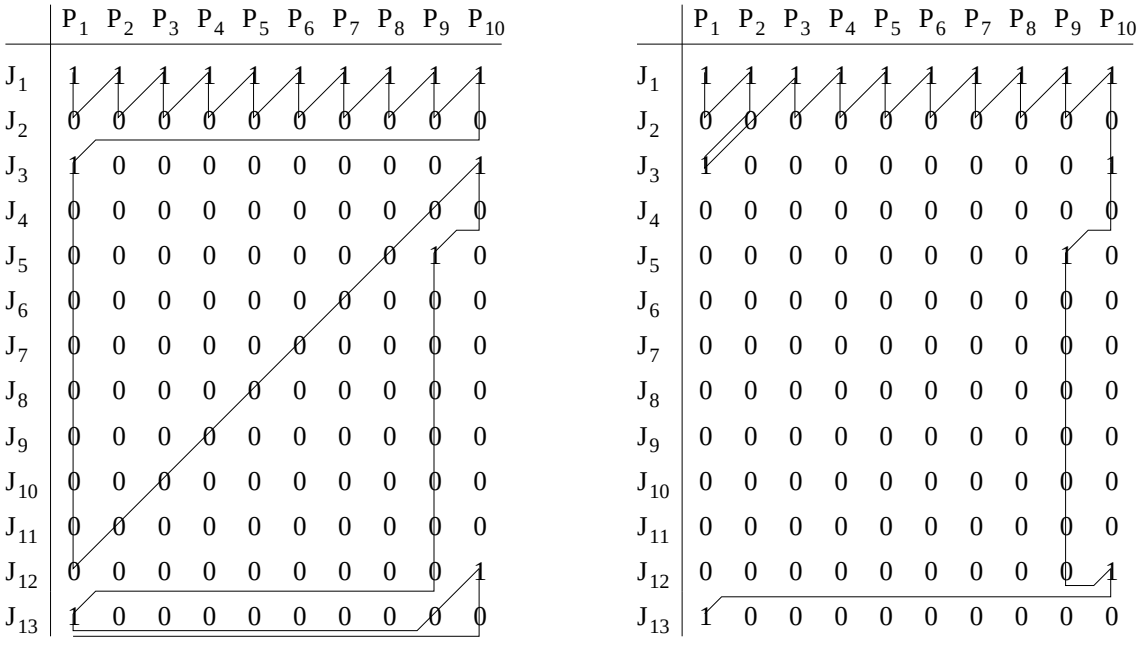


Abbildung 5.12: Links eine Free-Left-Abarbeitung mit linearer Distanz 54, rechts eine minimale Abarbeitung mit linearer Distanz 22. Verallgemeinert man das Beispiel auf m Maschinen, so benötigt Free-Left die lineare Distanz $6(m-1)$, während eine minimale Abarbeitung nur die lineare Distanz $2(m-1)+4$ hat.

beiden Tasks die Tasks $T_{(1,m)}$ und $T_{(m-2,1)}$ sind, d.h. $\mathcal{A}(m)$ hat das folgende Aussehen:

$$\mathcal{A}(m) := \overbrace{\left.\begin{pmatrix} 0 & 0 & 0 & \cdots & 0 & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 \end{pmatrix}\right\} m-1}^{m}$$

Außerdem sei für $n, m \geq 1$ die Matrix $\mathcal{Z}(n,m)$ die Matrix für n Jobs und m Maschinen, die nur Nullen enthält. Für jedes $c \geq 2$ sind die Taskmatrizen $\mathcal{F}_{c,r}$, $r = 1, 2, \ldots$ rekursiv wie folgt definiert:

1. $\mathcal{F}_{c,1}$ ist die 2×2 Taskmatrix mit den beiden Tasks $T_{(1,1)}$,$T_{(1,2)}$, siehe auch Abbildung 5.13.

2. Falls $r \geq 2$, so seien $n_{c,r-1}$ und $m_{c,r-1}$ die Anzahl der Jobs bzw. der Maschinen von $\mathcal{F}_{c,r-1}$. Dann ist die Anzahl $m_{c,r}$ der Maschinen der Matrix $\mathcal{F}_{c,r}$

   $$m_{c,r} := 2 + (c+1)m_{c,r-1}$$

   und die Matrix $\mathcal{F}_{c,r}$ ist definiert durch

$$\mathcal{F}_{c,r} := \begin{array}{|c|c|c|c|} \hline \mathcal{Z}(n_{c,r-1}, 2+m_{c,r-1}) & \overbrace{\mathcal{F}_{c,r-1}}^{c\text{-mal}} & \cdots & \mathcal{F}_{c,r-1} \\ \hline \mathcal{A}(2+m_{c,r-1}) & \mathcal{B}(m_{c,r-1}) & \cdots & \mathcal{B}(m_{c,r-1}) \\ \hline \end{array}$$

In der Abbildung 5.13 sind die Taskmatrizen $\mathcal{F}_{2,1}$, $\mathcal{F}_{2,2}$ und $\mathcal{F}_{2,3}$ jeweils mit der Free-Left-Abarbeitung dargestellt.

Wir zeigen zunächst, dass die Anzahl $m_{c,r}$ der Maschinen der Taskmatrix $\mathcal{F}_{c,r}$ für alle $c \geq 2$, $r \geq 1$ durch die Formel

$$m_{c,r} = \frac{2}{c}(c+1)^r - \frac{2}{c} \tag{5.9}$$

gegeben ist. Wir zeigen dies durch Induktion über r. Für $r = 1$ ergibt die Formel (5.9)

$$m_{c,1} = \frac{2}{c}(c+1) - \frac{2}{c} = \frac{2c+2-2}{c} = 2,$$

was mit der Definition übereinstimmt. Sei nun $r \geq 2$ und die Formel (5.9) gültig für $r-1$.

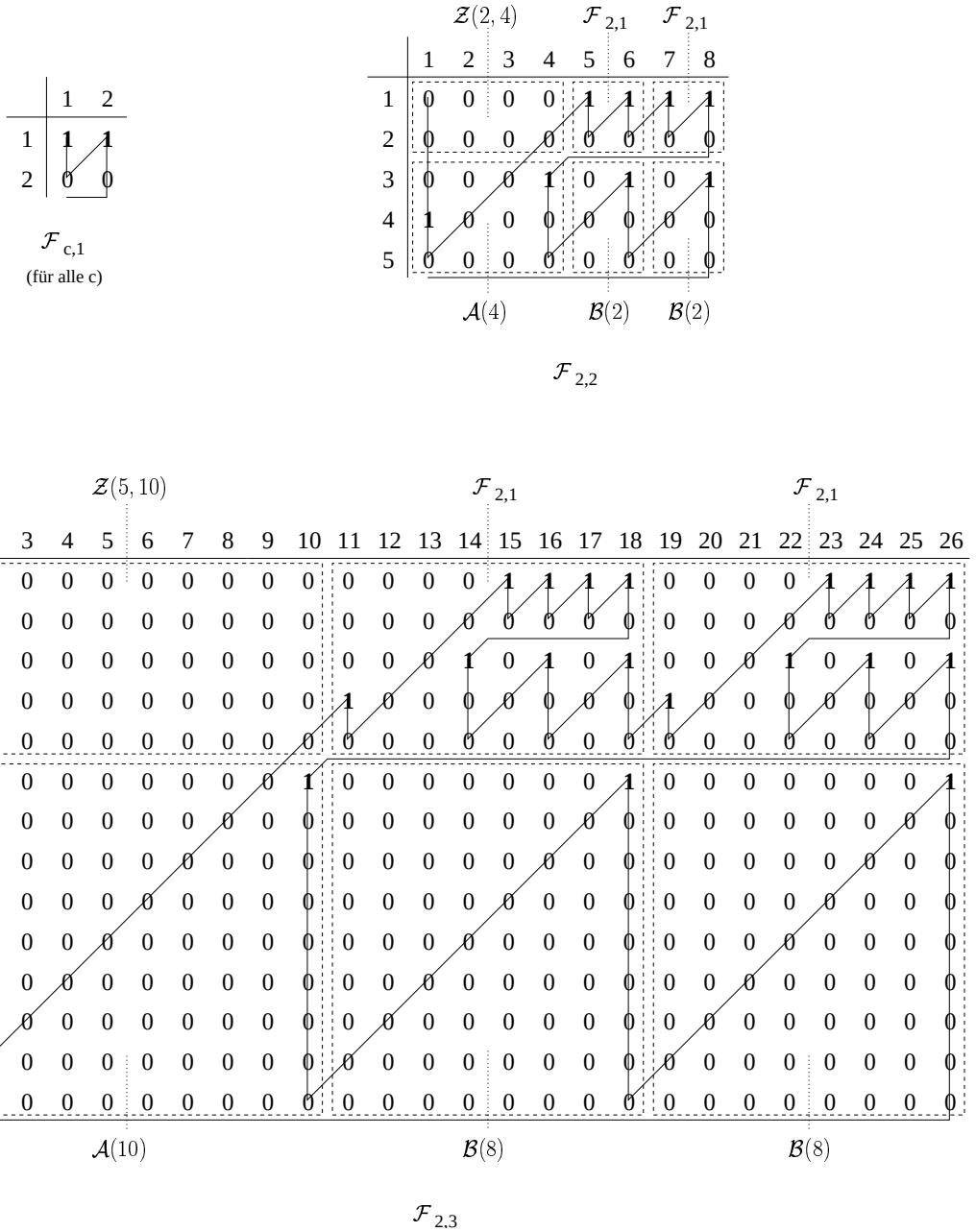


Abbildung 5.13: Free-Left-Abarbeitungen der Taskmatrizen $\mathcal{F}_{2,1}$, $\mathcal{F}_{2,2}$ und $\mathcal{F}_{2,3}$.

Nach Definition ist $m_{c,r} = 2 + (c+1)m_{c,r-1}$. Daraus folgt

$$\begin{aligned} m_{c,r} &= 2 + (c+1)m_{c,r-1} \\ &\underset{\text{I.V.}}{=} 2 + (c+1)\left(\frac{2}{c}(c+1)^{r-1} - \frac{2}{c}\right) \\ &= 2 + \frac{2}{c}(c+1)^r - \frac{2c+2}{c} \\ &= \frac{2}{c}(c+1)^r - \frac{2}{c} \end{aligned}$$

und somit die Gültigkeit von Formel (5.9).

Bei den folgenden Abarbeitungen lassen wir die spezielle Starttask $T_{(0,0)}$ und die spezielle Endtask $T_{(\infty,0)}$ immer weg. Wir wollen zunächst die lineare Distanz $\Delta^{\mathrm{FL}}(\mathcal{F}_{c,r})$ der Free-Left-Abarbeitung der Taskmatrix $\mathcal{F}_{c,r}$ bestimmen. Man betrachte den Fall $r > 1$. Wenn $\Lambda_1, \Lambda_2, \ldots, \Lambda_c$ die Free-Left-Abarbeitungen der Kopien der Matrix $\mathcal{F}_{c,r-1}$ in der Matrix $\mathcal{F}_{c,r}$ sind, so ist die Free-Left-Abarbeitung der Matrix $\mathcal{F}_{c,r}$ offenbar gegeben durch die Taskfolge

$$T_{(n_{c,r}-1,1)}, \Lambda_1, \Lambda_2, \ldots, \Lambda_c, T_{(n_{c,r-1}+1,2+m_{c,r-1})}, T_{(n_{c,r-1}+1,2+2m_{c,r-1})} \ldots, T_{(n_{c,r-1}+1,2+(c+1)m_{c,r-1})}$$

mit der linearen Distanz

$$\begin{aligned} \Delta^{\mathrm{FL}}(\mathcal{F}_{c,r}) &= m_{c,r-1} + 1 + c\left(1 + \Delta^{\mathrm{FL}}(\mathcal{F}_{c,r-1}) - (m_{c,r-1} - 1)\right) + 2c \cdot m_{c,r-1} + m_{c,r} - 1 \\ &= c \cdot \Delta^{\mathrm{FL}}(\mathcal{F}_{c,r-1}) + (c+1)m_{c,r-1} + m_{c,r} + 2c \\ &= c \cdot \Delta^{\mathrm{FL}}(\mathcal{F}_{c,r-1}) + (c+1)\left(\frac{2}{c}(c+1)^{r-1} - \frac{2}{c}\right) + \frac{2}{c}(c+1)^r - \frac{2}{c} + 2c \\ &= c \cdot \Delta^{\mathrm{FL}}(\mathcal{F}_{c,r-1}) + \frac{4}{c}(c+1)^r - \frac{4}{c} - 2 + 2c. \end{aligned}$$

Indem wir diese Rekursionsgleichung benutzen, zeigen wir nun durch Induktion, dass die lineare Distanz $\Delta^{\mathrm{FL}}(\mathcal{F}_{c,r})$ der Free-Left-Abarbeitung der Taskmatrix $\mathcal{F}_{c,r}$ für $c \geq 2$ und $r \geq 1$ durch die Formel

$$\Delta^{\mathrm{FL}}(\mathcal{F}_{c,r}) = \frac{4}{c}(c+1)^{r+1} - \frac{4}{c-1}c^{r+1} + \frac{4}{c(c-1)} - 2 \tag{5.10}$$

gegeben wird.

<u>$r = 1$:</u> Für $r = 1$ ergibt die obige Formel

$$\begin{aligned} \Delta^{\mathrm{FL}}(\mathcal{F}_{c,1}) &= \frac{4}{c}(c+1)^2 - \frac{4}{c-1}c^2 + \frac{4}{c(c-1)} - 2 \\ &= 4 \cdot \frac{(c+1)^2(c-1) - c^3 + 1}{c(c-1)} - 2 \\ &= 4 \cdot \frac{c^2 - c}{c(c-1)} - 2 = 2, \end{aligned}$$

was korrekt ist, da die Free-Left-Abarbeitung der Matrix $\mathcal{F}_{c,1}$ die lineare Distanz 2 hat, siehe Abbildung 5.13.

$r > 1$: Die Formel sei für $r' < r$ gültig. Dann folgt aus der Rekursionsgleichung und der Induktionsvoraussetzung:

$$\begin{aligned}
\Delta^{\mathrm{FL}}(\mathcal{F}_{c,r}) &= c \cdot \Delta^{\mathrm{FL}}(\mathcal{F}_{c,r-1}) + \frac{4}{c}(c+1)^r - \frac{4}{c} - 2 + 2c \\
&\underset{\text{I.V.}}{=} c \cdot \left(\frac{4}{c}(c+1)^r - \frac{4}{c-1}c^r + \frac{4}{c(c-1)} - 2 \right) + \frac{4}{c}(c+1)^r - \frac{4}{c} - 2 + 2c \\
&= (c+1)\frac{4}{c}(c+1)^r - \frac{4}{c-1}c^{r+1} + \frac{4}{c-1} - \frac{4}{c} - 2 \\
&= \frac{4}{c}(c+1)^{r+1} - \frac{4}{c-1}c^{r+1} + \frac{4}{c(c-1)} - 2.
\end{aligned}$$

Also gibt Formel (5.10) für alle $c \geq 2$ und alle $r \geq 1$ die lineare Distanz der Free-Left-Abarbeitung der Matrix $\mathcal{F}_{c,r}$ an.

Nun betrachten wir die Left-First-Abarbeitung der Taskmatrizen $\mathcal{F}_{c,r}$. Bei der Left-First-Abarbeitung wird in Entscheidungskonfigurationen immer zunächst die Task links von der Arbeiterposition bearbeitet. In der Abbildung 5.14 sind nochmals die Taskmatrizen $\mathcal{F}_{2,1}$, $\mathcal{F}_{2,2}$ und $\mathcal{F}_{2,3}$, dargestellt, nun jedoch mit der Left-First-Abarbeitung. Wir wollen jetzt die lineare Distanz $\Delta^{\mathrm{LF}}(\mathcal{F}_{c,r})$ der Left-First-Abarbeitung der Taskmatrix $\mathcal{F}_{c,r}$ für $c \geq 2$ und $r \geq 1$ bestimmen. Betrachten wir dazu wieder den Fall $r > 1$. Seien $\Lambda_1, \Lambda_2, \ldots, \Lambda_c$ die Left-First-Abarbeitungen der Kopien der Matrix $\mathcal{F}_{c,r-1}$ in der Matrix $\mathcal{F}_{c,r}$. Sei T_k die erste Task von Λ_k und sei Λ'_k die Folge Λ_k ohne die erste Task T_k. Die Left-First-Abarbeitung der Matrix $\mathcal{F}_{c,r}$ ist dann offenbar (siehe Abbildung 5.14) gegeben durch die Taskfolge

$$\begin{array}{c}
T_{(n_{c,r}-1,1)}, \\
T_1, T_{(n_{c,r-1}+1,2+m_{c,r-1})}, \Lambda'_1, \\
T_2, T_{(n_{c,r-1}+1,2+2m_{c,r-1})}, \Lambda'_2, \\
\vdots \\
T_c, T_{(n_{c,r-1}+1,2+cm_{c,r-1})}, \Lambda'_c, \\
T_{(n_{c,r-1}+1,2+(c+1)m_{c,r-1})}
\end{array}$$

mit der linearen Distanz

$$\begin{aligned}
\Delta^{\mathrm{LF}}(\mathcal{F}_{c,r}) &= m_{c,r-1} + 1 + c\left(3 + \Delta^{\mathrm{LF}}(\mathcal{F}_{c,r-1}) - (m_{c,r-1} - 1)\right) + m_{c,r} - 1 \\
&= c \cdot \Delta^{\mathrm{LF}}(\mathcal{F}_{c,r-1}) - (c-1)m_{c,r-1} + m_{c,r} + 4c \\
&= c \cdot \Delta^{\mathrm{LF}}(\mathcal{F}_{c,r-1}) - (c-1)\left(\frac{2}{c}(c+1)^{r-1} - \frac{2}{c}\right) + \frac{2}{c}(c+1)^r - \frac{2}{c} + 4c \\
&= c \cdot \Delta^{\mathrm{LF}}(\mathcal{F}_{c,r-1}) + \frac{2}{c}((c+1) - (c-1))(c+1)^{r-1} + \frac{2(c-1)}{c} - \frac{2}{c} + 4c \\
&= c \cdot \Delta^{\mathrm{LF}}(\mathcal{F}_{c,r-1}) + \frac{4}{c}(c+1)^{r-1} + 4c + 2 - \frac{4}{c}.
\end{aligned}$$

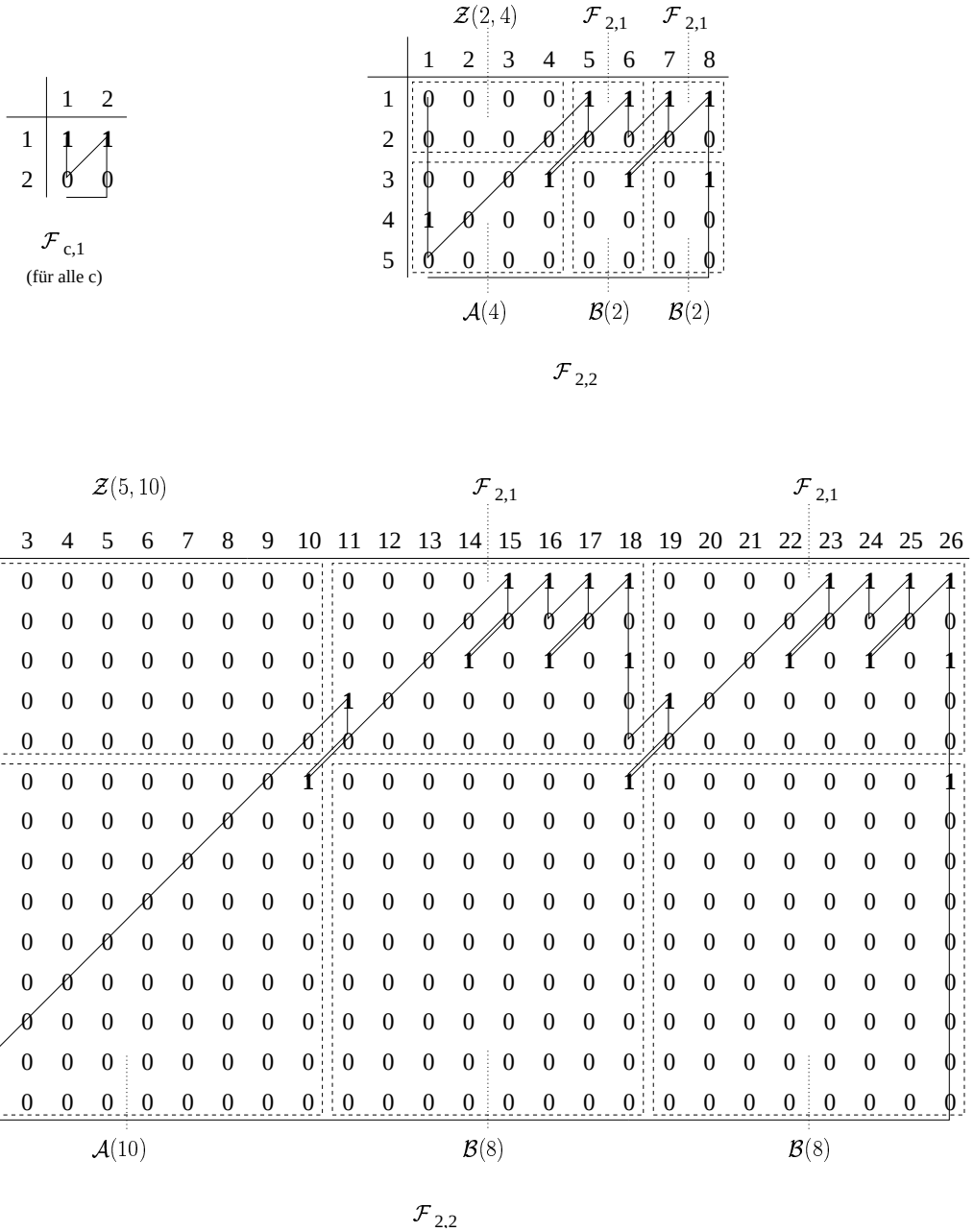


Abbildung 5.14: Left-First-Abarbeitungen der Taskmatrizen $\mathcal{F}_{2,1}$, $\mathcal{F}_{2,2}$ und $\mathcal{F}_{2,3}$.

Indem wir diese Rekursionsgleichung benutzen, zeigen wir nun durch Induktion, dass die lineare Distanz $\Delta^{\mathrm{LF}}(\mathcal{F}_{c,r})$ der Left-First-Abarbeitung der Taskmatrix $\mathcal{F}_{c,r}$ für $c \geq 2$ und $r \geq 1$ durch die Formel

$$\Delta^{\mathrm{LF}}(\mathcal{F}_{c,r}) = \frac{4}{c}(c+1)^r + \frac{2}{c-1}c^r - 4 - \frac{4}{c} - \frac{2}{c-1} \tag{5.11}$$

gegeben ist.

$\underline{r = 1:}$ Für $r = 1$ ergibt die Formel

$$\Delta^{\mathrm{LF}}(\mathcal{F}_{c,1}) = \frac{4}{c}(c+1) + \frac{2}{c-1}c - 4 - \frac{4}{c} - \frac{2}{c-1} = \frac{2c}{c-1} - \frac{2}{c-1} = 2,$$

was korrekt ist, da die Left-First-Abarbeitung der Matrix $\mathcal{F}_{c,1}$ die lineare Distanz 2 hat, siehe Abbildung 5.14.

$\underline{r > 1:}$ Die Formel sei für $r' < r$ gültig. Dann folgt aus der Rekursionsgleichung und der Induktionsvoraussetzung:

$$\begin{aligned}
\Delta^{\mathrm{LF}}(\mathcal{F}_{c,r}) &= c \cdot \Delta^{\mathrm{LF}}(\mathcal{F}_{c,r-1}) + \frac{4}{c}(c+1)^{r-1} + 4c + 2 - \frac{4}{c} \\
&\underset{\text{I.V.}}{=} c\left(\frac{4}{c}(c+1)^{r-1} + \frac{2}{c-1}c^{r-1} - 4 - \frac{4}{c} - \frac{2}{c-1}\right) + \frac{4}{c}(c+1)^{r-1} + 4c + 2 - \frac{4}{c} \\
&= \frac{4}{c}(c+1)^r + \frac{2}{c-1}c^r - 4 - \frac{4}{c} - \frac{2}{c-1}.
\end{aligned}$$

Also gibt Formel (5.11) für alle $c \geq 2$ und alle $r \geq 1$ die lineare Distanz der Left-First-Abarbeitung der Matrix $\mathcal{F}_{c,r}$ an.

Betrachten wir nun für $r \geq 2$ die Matrizen $\mathcal{F}_{r,r}$ so ergibt sich aus den Formeln (5.10) und (5.11)

$$\Delta^{\mathrm{FL}}(\mathcal{F}_{r,r}) = \frac{4}{r}(r+1)^{r+1} - \frac{4}{r-1}r^{r+1} + \frac{4}{r(r-1)} - 2,$$

$$\Delta^{\mathrm{LF}}(\mathcal{F}_{r,r}) = \frac{4}{r}(r+1)^r + \frac{2}{r-1}r^r - 4 - \frac{4}{r} - \frac{2}{r-1}.$$

Wegen $\lim_{r\to\infty}\left(\frac{r+1}{r}\right)^r = \lim_{r\to\infty}\left(1+\frac{1}{r}\right)^r = e \approx 2,718\ldots$ folgt hieraus

$$\begin{aligned}
\lim_{r\to\infty} \frac{\Delta^{\mathrm{FL}}(\mathcal{F}_{r,r})}{(r+1)\Delta^{\mathrm{LF}}(\mathcal{F}_{r,r})} &= \lim_{r\to\infty} \frac{\frac{4}{r}(r+1)^{r+1} - \frac{4}{r-1}r^{r+1} + \frac{4}{r(r-1)} - 2}{(r+1)\left(\frac{4}{r}(r+1)^r + \frac{2}{r-1}r^r - 4 - \frac{4}{r} - \frac{2}{r-1}\right)} \\
&= \lim_{r\to\infty} \frac{1 - \frac{r}{r-1}\left(\frac{r}{r+1}\right)^{r+1}}{1 + \frac{r}{2(r-1)}\left(\frac{r}{r+1}\right)^r} \\
&= \lim_{r\to\infty} \frac{1 - \frac{r^2}{r^2-1}\left(\frac{r}{r+1}\right)^r}{1 + \frac{r}{2(r-1)}\left(\frac{r}{r+1}\right)^r} \\
&= \frac{1 - \frac{1}{e}}{1 + \frac{1}{2e}} \approx 0,5339\ldots.
\end{aligned}$$

Somit gilt für das Verhältnis der linearen Distanz der Free-Left-Abarbeitung der Matrizen $\mathcal{F}_{r,r}$ zur linearen Distanz der Left-First-Abarbeitung der Matrizen $\mathcal{F}_{r,r}$

$$\frac{\Delta^{\mathrm{FL}}(\mathcal{F}_{r,r})}{\Delta^{\mathrm{LF}}(\mathcal{F}_{r,r})} \in \Omega(r).$$

Dies zeigt, dass der kompetitive Faktor von Free-Left nicht durch eine Konstante beschränkt ist. Nach Gleichung (5.9) gilt für die Maschinenanzahl $m_{r,r}$ der Matrizen $\mathcal{F}_{r,r}$ für $r \geq 2$

$$m_{r,r} = \frac{2}{r}(r+1)^r - \frac{2}{r}. \tag{5.12}$$

Hieraus folgt für $r \geq 2$ einerseits

$$m_{r,r} \geq 2(r+1)^{r-1} - 1 \geq (r+1)^{r-1}$$

und andererseits

$$m_{r,r} \leq (r+1)^r.$$

Daher gilt

$$\begin{aligned}
\frac{\log_2 m_{r,r}}{\log_2 \log_2 m_{r,r}} &\leq \frac{\log_2 (r+1)^r}{\log_2 \log_2 (r+1)^{r-1}} \\
&= r \cdot \frac{\log_2 (r+1)}{\log_2 (r-1) + \log_2 \log_2 (r+1)} \\
&\leq r \cdot \frac{\log_2 (r+1)}{\log_2 (r-1)} \\
&= r \cdot \left(1 + \frac{\log_2 \left(\frac{r+1}{r-1}\right)}{\log_2 (r-1)}\right) \in O(r).
\end{aligned}$$

Also ist $r \in \Omega\left(\frac{\log_2 m_{r,r}}{\log_2 \log_2 m_{r,r}}\right)$ und es ergibt sich der folgende Satz:

Satz 5.6 *Der kompetitive Faktor von Free-Left ist für lineare Distanzen nicht durch eine Konstante beschränkt, sondern liegt in* $\Omega\left(\frac{\log_2 m}{\log_2 \log_2 m}\right)$*, wobei* m *die Maschinenanzahl der Taskmatrix ist.*

Kapitel 6

Empirische Auswertungen

In diesem Kapitel sollen einige empirische Ergebnisse zum Förderband-Flow-Shop-Problem für Einheitsdistanzen und für lineare Distanzen dargestellt werden. Dabei vergleichen wir die Ergebnisse verschiedener Algorithmen miteinander sowie mit den unteren Schranken aus Kapitel 4 und mit optimalen Lösungen, sofern wir letztere berechnen konnten.

Bevor wir die Ergebnisse darstellen, gehen wir zunächst kurz auf die Berechnung von optimalen Lösungen ein. Anschließend stellen wir noch zwei weitere Algorithmen vor, die wir in die empirischen Untersuchungen mit einbeziehen. Zum einen handelt es sich dabei um einen Online-Algorithmus zur Minimierung der Einheitsdistanz, den wir hauptsächlich aufnehmen, um eine Vergleichsmöglichkeit mit dem 2-Approximationsalgorithmus aus Kapitel 4 zu haben. Zum Zweiten handelt es sich um einen Approximationsalgorithmus für lineare Distanzen, der auf einer Heuristik beruht, die intuitiv auch auf beliebige additive Distanzen anwendbar zu sein scheint. Dieser Algorithmus zeichnet sich bei den empirischen Untersuchungen durch gute Ergebnisse (bezüglich der untersuchten linearen Distanzen) aus, obwohl wir für ihn keinen konstanten Worst-Case-Fehler beweisen können. Allerdings kann dieser Algorithmus natürlich zu einem 3-Approximationsalgorithmus gemacht werden, indem man die berechnete Abarbeitung durch Anwendung des RL-Austausch-Algorithmus aus Kapitel 4 verbessert. Bei der Darstellung der empirischen Ergebnisse werden wir dies jedoch nicht tun, damit die Güte des eigentlichen Algorithmus erkennbar bleibt.

6.1 Berechnung optimaler Lösungen

Da das Förderband-Flow-Shop-Problem für Einheitsdistanzen NP-vollständig ist, kann es keinen polynomiellen Algorithmus für dieses Problem geben, sofern $P \neq NP$ gilt. Aber auch für lineare Distanzen ist uns kein polynomieller Algorithmus bekannt. Zur Berechnung optimaler Lösungen verwenden wir daher im Wesentlichen Algorithmus 1 aus Abschnitt 3.2. Dieser Algorithmus berechnet mit Hilfe dynamischer Programmierung für jede mögliche Konfiguration die minimale Distanz, die zum Erreichen der Konfiguration benötigt wird. Im Allgemeinen hat dieser Algorithmus natürlich eine Laufzeit, welche exponentiell in der Maschinenanzahl (oder Jobanzahl) ist. Der Algorithmus lässt sich jedoch häufig da-

durch beschleunigen, dass nicht immer alle Nachfolgekonfigurationen einer Konfiguration betrachtet werden müssen. Ist nämlich die nächste Task in einer minimalen Restabarbeitung bekannt, so reicht es aus, lediglich diejenige Nachfolgekonfiguration zu betrachten, die durch die Bearbeitung dieser Task entsteht. Ist etwa an der augenblicklichen Arbeiterposition eine Task ausführbar, so können wir uns auf die Nachfolgekonfiguration bezüglich der Bearbeitung dieser Task beschränken. Abhängig vom Distanzmaß lassen sich jedoch auch in weiteren Fällen gewisse Nachfolgekonfigurationen ausschließen.

Sehen wir uns zunächst die linearen Distanzen an. Hier brauchen wir für den Fall, dass an der aktuellen Arbeiterposition keine Task ausführbar ist, nur die beiden Nachfolgekonfigurationen bezüglich der nächsten ausführbaren Tasks links bzw. rechts von der augenblicklichen Arbeiterposition in Betracht zu ziehen. Darüber hinaus genügt es in den Entscheidungskonfigurationen, in denen wir aufgrund von Satz 5.3 bzw. Satz 5.4 eine optimale Entscheidung treffen können, nur eine dieser beiden Nachfolgekonfigurationen zu berücksichtigen.

Betrachten wir jetzt die Einheitsdistanzen. Falls an der aktuellen Arbeiterposition keine Task ausführbar ist, können wir uns nicht auf die beiden Nachfolgekonfigurationen bezüglich der nächsten ausführbaren Tasks rechts bzw. links von der aktuellen Position beschränken. Wir können jedoch testen, ob eine ausführbare Task $T_{(j,p)}$ existiert, so dass alle Tasks des maximal gültigen Taskblocks $\Phi_p^{[j,j']}$, $j' \geq j$, an der Maschine P_p nacheinander ausgeführt werden können. Ist dies der Fall, so gibt es offensichtlich eine mit der Task $T_{(j,p)}$ beginnende minimale Restabarbeitung, da an der Maschine P_p sowieso höchstens die Tasks des maximal gültigen Blocks $\Phi_p^{[j,j']}$ ohne einen Maschinenwechsel ausgeführt werden können. Wir können uns dann auf die Nachfolgekonfiguration bezüglich der Ausführung der Task $T_{(j,p)}$ beschränken.

Durch die gerade beschriebenen Modifikationen wird die Laufzeit des Algorithmus in den meisten Fällen erheblich verkürzt. Auf diese Weise können die optimalen Lösungen für Taskmatrizen mit bis zu 2000 Jobs und bis zu 10 Maschinen in der Regel innerhalb weniger Sekunden bestimmt werden. Für bis zu 20 Maschinen lassen sich die optimalen Lösungen meistens noch im Bereich einiger Minuten berechnen, falls der Anteil der Jobs, die an einer Maschine bearbeitet werden müssen, nicht zu hoch ist ($\leq 50\%$). Insbesondere für Einheitsdistanzen steigen die Berechnungszeiten aber stark an, wenn die Anzahl der Maschinen oder der prozentuale Anteil der Jobs, die an einer Maschine bearbeitet werden müssen, erhöht wird.

Es sei noch erwähnt, dass eine weitere Möglichkeit, eine optimale Lösung zu finden, in der Lösung eines ganzzahligen linearen Programms[1] besteht. Beispielsweise lässt sich das Förderband-Flow-Shop-Problem für Einheitsdistanzen leicht wie folgt durch ein solches Programm beschreiben. Wir definieren für jeden gültigen Block $\Phi_p^{[j,j']}$ eine 0/1-Variable $B_{p,j,j'}$ und führen folgende Restriktionen ein. Für je zwei unverträgliche Blöcke $\Phi_{p_1}^{[j_1,j_1']}$ und $\Phi_{p_2}^{[j_2,j_2']}$ muss die Ungleichung $B_{p_1,j_1,j_1'} + B_{p_2,j_2,j_2'} \leq 1$ gelten, und für jede Task $T_{(k,p)}$ muss die Summe der Variablen $B_{p,j,j'}$ mit $j \leq k \leq j'$ gleich 1 sein. Offensichtlich bewirken diese

[1] Für Informationen zu ganzzahliger linearer Programmierung siehe beispielsweise [16, 23]

Restriktionen, dass die Blöcke $\Phi_p^{[j,j']}$, deren zugehörige Variablen $B_{p,j,j'}$ in einer Lösung den Wert 1 haben, eine gültige Blocküberdeckung der Taskmatrix bilden. Eine Lösung des ganzzahligen linearen Programms, bei der die Summe über alle Variablen $B_{p,j,j'}$ minimal ist, stellt also eine minimale gültige Blocküberdeckung dar und entspricht somit nach den Ergebnissen aus Kapitel 3 einer Lösung des Förderband-Flow-Shop-Problems mit minimaler Einheitsdistanz. Für lineare Distanzen lassen sich ebenfalls ganzzahlige lineare Programme angeben, jedoch sind diese erheblich schwieriger zu formulieren. In beiden Fällen lassen sich diese Programme aber bereits für relativ kleine Taskmatrizen mit Standardprogrammen zur Lösung ganzzahliger linearer Programme nicht mehr in akzeptabler Zeit lösen.

6.2 Weitere Algorithmen

Wir wollen in diesem Abschnitt kurz zwei weitere Algorithmen skizzieren, die wir in die Auswertungen im nächsten Abschnitt mit einbeziehen. Dabei handelt es sich um einen Online-Algorithmus zur Minimierung der Einheitsdistanz und um einen Approximationsalgorithmus für additive Distanzen.

Der Online-Algorithmus *Min-Change* zur Minimierung der Einheitsdistanz, verhält sich wie folgt. Ist in einer Konfiguration an der Arbeiterposition eine Task ausführbar, so wird diese bearbeitet. Andernfalls überprüft der Algorithmus für jede ausführbare Task $T_{(j,p)}$, ob mit Hilfe der verfügbaren Informationen (d.h. der bereits bekannten Jobs) der maximale gültige Block $\Phi_p^{[j,j']}$, $j' \geq j$, bestimmt werden kann. Falls dies der Fall ist und falls alle Tasks des Blocks nacheinander ausgeführt werden können, so bearbeitet der Algorithmus als nächstes eine solche Task $T_{(j,p)}$ und anschließend die weiteren Tasks des Blocks, da diese an der neuen Arbeiterposition p der Reihe nach ausführbar werden. Offensichtlich ist dieses eine optimale Entscheidung des Algorithmus, da an der Maschine P_p sowieso höchstens die Tasks des maximal gültigen Blocks $\Phi_p^{[j,j']}$ ohne einen Maschinenwechsel ausgeführt werden können. Lässt sich keine solche optimale Entscheidung treffen, so führt der Algorithmus die ausführbare Task $T_{(j,p)}$ mit kleinstem Maschinenindex p aus.

Der Approximationsalgorithmus *LB-Decision* für lineare bzw. additive Distanzen wählt in Nicht-Entscheidungskonfigurationen immer die offensichtlich mögliche optimale Fortsetzung. Dies bedeutet, dass eine ausführbare Task an der aktuellen Arbeiterposition sofort bearbeitet wird und dass in einer Konfiguration, in der es nur auf einer Seite von der Arbeiterposition ausführbare Tasks gibt, die nächstgelegene dieser Tasks bearbeitet wird. In einer Entscheidungskonfiguration betrachtet der Algorithmus die beiden Nachfolgekonfigurationen, die sich nach Bearbeitung der nächstgelegenen ausführbaren Tasks T^L links bzw. T^R rechts von der Arbeiterposition ergeben. Für jede dieser Konfigurationen wird eine untere Schranke für eine Restabarbeitung bestimmt. Die untere Schranke wird dabei völlig analog zu der unteren Schranke für additive Distanzen aus Gleichung (4.3) berechnet, indem unter Berücksichtigung der Arbeiterposition für jeden Maschinenindex $s \in \{1, \ldots, m-1\}$ die minimale Anzahl der noch benötigten P_s-Bewegungen bestimmt wird. Der Algorithmus

wählt dann die Seite, für die sich inklusive der ersten Bewegung zur Task T^L bzw. T^R die niedrigere untere Schranke ergibt. Falls die beiden unteren Schranken gleich sind, wird eine Free-Left-Entscheidung getroffen. Für die Berechnung der minimalen Anzahl der noch benötigten P_s-Bewegungen kann durch eine Vorverarbeitung vermieden werden, dass die vollständigen s-Ketten jedes Mal neu berechnet werden müssen. Auf diese Weise lässt sich der Algorithmus sehr effizient implementieren. Da die Berechnung der unteren Schranken eventuelle unterschiedliche Distanzen zwischen benachbarten Maschinenpaaren berücksichtigt, scheint der Algorithmus LB-Decision auch für allgemeine additive Distanzen geeignet zu sein. Wir haben die empirischen Auswertungen jedoch nur für lineare Distanzen vorgenommen.

6.3 Auswertungen

Die empirischen Ergebnisse in diesem Abschnitt wurden von dem Programm *MoveMin* berechnet, welches im Rahmen dieser Arbeit erstellt wurde. Neben den einfachen Berechnungen der Abarbeitungen mit Hilfe der unterschiedlichen Algorithmen erlaubt das Programm auch die Visualisierung der berechneten Lösungen sowie der meisten anderen in dieser Arbeit benutzten Konzepte.

Die empirischen Ergebnisse sollen in erster Linie einen Eindruck von der Güte der Algorithmen vermitteln. Die Taskmatrizen wurden mit unterschiedlichen Parametern zufällig gewählt. Die Auswahl der Parameter war dabei recht willkürlich, so dass aus diesen empirischen Resultaten nicht unbedingt gesicherte Ergebnisse abgeleitet werden können. Allerdings zeigen alle unsere bisherigen Erfahrungen, dass diese Ergebnisse als typisch angesehen werden können. Die zufälligen Taskmatrizen wurden ebenfalls mit dem Programm MoveMin erzeugt. Dabei kann für jede Maschine angegeben werden, mit welcher Wahrscheinlichkeit ein Job an der Maschine bearbeitet werden soll. In Tabelle 6.1 sind 23 Beispielmatrizen mit je 2000 Jobs angegeben, für die wir die Auswertungen vorgenommen haben. Die Taskmatrizen A1 bis A7 haben jeweils 10 Maschinen, wobei die Jobs an den Maschinen mit unterschiedlicher Wahrscheinlichkeit bearbeitet werden müssen. Die Wahrscheinlichkeiten, mit denen eine Task an einer Maschine gewählt wurde, sind für jede Maschine in der Tabelle angegeben. So müssen z.B. bei der Taskmatrix A1 an der ersten Maschine etwa 5 Prozent der Jobs bearbeitet werden, an der zweiten etwa 10 Prozent, an der dritten etwa 15 Prozent, usw.. Die Taskmatrizen B1 bis B7 besitzen ebenfalls 10 Maschinen, wobei jedoch die Taskwahrscheinlichkeiten jeweils für jede Maschine gleich gewählt wurden. Die weiteren Matrizen haben 20, 100 bzw. 500 Maschinen mit jeweils identischen Taskwahrscheinlichkeiten an jeder Maschine.

Lineare Distanzen In der Tabelle 6.2 sind für die Taskmatrizen aus Tabelle 6.1 die linearen Distanzen der Abarbeitungen angegeben, welche von den Online-Algorithmen Free-Left, Left-First, Right-First, Keep-Direction, Change-Direction, Nearest-First(R) sowie den Approximationsalgorithmen RL-Exchange und LB-Decision berechnet werden. Das in Klammern gesetzte 'R' hinter dem Algorithmus Nearest-First deutet an, dass bei

Name	Jobs	Maschinen	Task-Wahrscheinlichkeiten (%)
A1	2000	10	5,10,15,20,25,30,35,40,45,50
A2	2000	10	25,30,35,40,45,50,55,60,65,70
A3	2000	10	50,45,40,35,30,25,20,15,10,5
A4	2000	10	70,65,60,55,50,45,40,35,30,25
A5	2000	10	10,20,30,40,50,50,40,30,20,10
A6	2000	10	50,40,30,20,10,10,20,30,40,50
A7	2000	10	20,40,20,40,20,40,20,40,20,40
B1	2000	10	10 % für jede Maschine
B2	2000	10	20 % für jede Maschine
B3	2000	10	30 % für jede Maschine
B4	2000	10	40 % für jede Maschine
B5	2000	10	50 % für jede Maschine
B6	2000	10	60 % für jede Maschine
B7	2000	10	70 % für jede Maschine
C1	2000	20	20 % für jede Maschine
C2	2000	20	40 % für jede Maschine
C3	2000	20	60 % für jede Maschine
D1	2000	100	20 % für jede Maschine
D2	2000	100	40 % für jede Maschine
D3	2000	100	60 % für jede Maschine
E1	2000	500	20 % für jede Maschine
E2	2000	500	40 % für jede Maschine
E3	2000	500	60 % für jede Maschine

Tabelle 6.1: Beispiel-Matrizen für die Auswertung der Algorithmen.

diesem Algorithmus in einer Entscheidungskonfiguration, in welcher der Abstand zu den nächsten ausführbaren Tasks links bzw. rechts identisch ist, die rechte Task gewählt wird. RL-Exchange ist der im Kapitel 4 vorgestellte Approximationsalgorithmus für additive Distanzen, der lineare Laufzeit und einen Worst-Case-Fehler von 3 hat.

Die Tabelle enthält für alle Taskmatrizen die untere Schranke aus Gleichung (4.3) und für jeden Algorithmus die prozentuale Abweichung von dieser unteren Schranke. Für die Instanzen, für die wir die optimale Lösung bestimmen konnten, ist außerdem das Optimum und die prozentuale Abweichung der unteren Schranke sowie der einzelnen Algorithmen von diesem Optimum angegeben. Für die Beispiele, für die wir das Optimum berechnen konnten, liegt die untere Schranke immer zwischen 5 und 9 Prozent unter dem Optimum. Der Algorithmus Free-Left erweist sich in allen Beispielen als der beste Online-Algorithmus, dessen Abweichung von der unteren Schranke bzw. vom Optimum immer kleiner als 22 bzw. 9 Prozent ist. Die Offline-Algorithmen RL-Exchange und LB-Decision schneiden bei allen Beispielen noch etwas besser ab und weichen um weniger als 5 Prozent vom Optimum bzw. weniger als 19 Prozent von der unteren Schranke ab. Dabei liefert der

Algorithmus LB-Decision durchgängig die etwas besseren Ergebnisse als der Algorithmus RL-Exchange. Dies ist allerdings teilweise durch die Free-Left-Entscheidungen bedingt, die der Algorithmus LB-Decision trifft, wenn in einer Entscheidungskonfiguration die beiden berechneten unteren Schranken identisch sind. Wird in solchen Fällen entweder immer die linke oder immer die rechte Task ausgeführt, ergeben sich schlechtere Abarbeitungen, deren Distanz teilweise sogar etwas größer als die der Free-Left-Abarbeitung ist. Die übrigen Online-Algorithmen schneiden deutlich schlechter ab als Free-Left, RL-Exchange und LB-Decision.

	Optimum	Untere Schranke	Free-Left	Nearest-First (R)	Right-First	Left-First	Keep-Direction	Change-Direction	LB-Decision	RL-Exchange
A1	7296	6870	7826	8764	10052	10010	8748	10928	7420	7494
Abw. v. u.S.			13.9	27.6	46.3	45.7	27.3	59.1	8.0	9.1
Abw. v. Opt.		-5.8	7.3	20.1	37.8	37.2	19.9	49.8	1.7	2.7
A2	10954	10214	11542	13928	15730	15880	13186	17734	11192	11244
Abw. v. u.S.			13.0	36.4	54.0	55.5	29.1	73.6	9.6	10.1
Abw. v. Opt.		-6.8	5.4	27.1	43.6	45.0	20.4	61.9	2.2	2.6
A3	7252	6780	7840	8442	10012	10022	8704	10650	7418	7444
Abw. v. u.S.			15.6	24.5	47.7	47.8	28.4	57.1	9.4	9.8
Abw. v. Opt.		-6.5	8.1	16.4	38.1	38.2	20.0	46.9	2.3	2.6
A4	11020	10244	11594	14360	16068	15806	13160	17894	11256	11338
Abw. v. u.S.			13.2	40.2	56.9	54.3	28.5	74.7	9.9	10.7
Abw. v. Opt.		-7.0	5.2	30.3	45.8	43.4	19.4	62.4	2.1	2.9
A5	7676	7282	8124	9154	10252	10360	8962	11412	7798	7852
Abw. v. u.S.			11.6	25.7	40.8	42.3	23.1	56.7	7.1	7.8
Abw. v. Opt.		-5.1	5.8	19.3	33.6	35.0	16.8	48.7	1.6	2.3
A6	8150	7474	8712	9506	12698	12908	10680	12984	8356	8528
Abw. v. u.S.			16.6	27.2	69.9	72.7	42.9	73.7	11.8	14.1
Abw. v. Opt.		-8.3	6.9	16.6	55.8	58.4	31.0	59.3	2.5	4.6
A7	7552	6996	8056	8842	10908	10932	9088	11622	7734	7828
Abw. v. u.S.			15.2	26.4	55.9	56.3	29.9	66.1	10.5	11.9
Abw. v. Opt.		-7.4	6.7	17.1	44.4	44.8	20.3	53.9	2.4	3.7
B1	8356	7788	8970	9964	12114	12016	9988	12920	8578	8634
Abw. v. u.S.			15.2	27.9	55.5	54.3	28.2	65.9	10.1	10.9
Abw. v. Opt.		-6.8	7.3	19.2	45.0	43.8	19.5	54.6	2.7	3.3
B2	6404	6030	6930	7460	8882	8780	7618	9576	6570	6618
Abw. v. u.S.			14.9	23.7	47.3	45.6	26.3	58.8	9.0	9.8
Abw. v. Opt.		-5.8	8.2	16.5	38.7	37.1	19.0	49.5	2.6	3.3
B3	8356	7788	8970	9964	12114	12016	9988	12920	8578	8634

Fortsetzung auf der nächsten Seite

	Optimum	Untere Schranke	Free-Left	Nearest-First (R)	Right-First	Left-First	Keep-Direction	Change-Direction	LB-Decision	RL-Exchange
Abw. v. u.S.			15.2	27.9	55.5	54.3	28.2	65.9	10.1	10.9
Abw. v. Opt.		-6.8	7.3	19.2	45.0	43.8	19.5	54.6	2.7	3.3
B4	10026	9262	10610	12494	14556	14872	11896	15968	10280	10370
Abw. v. u.S.			14.6	34.9	57.2	60.6	28.4	72.4	11.0	12.0
Abw. v. Opt.		-7.6	5.8	24.6	45.2	48.3	18.7	59.3	2.5	3.4
B5	11572	10706	12168	15032	16892	16956	13416	18770	11826	11900
Abw. v. u.S.			13.7	40.4	57.8	58.4	25.3	75.3	10.5	11.2
Abw. v. Opt.		-7.5	5.2	29.9	46.0	46.5	15.9	62.2	2.2	2.8
B6	12896	12004	13430	17236	18318	18504	14706	21026	13096	13268
Abw. v. u.S.			11.9	43.6	52.6	54.1	22.5	75.2	9.1	10.5
Abw. v. Opt.		-6.9	4.1	33.7	42.0	43.5	14.0	63.0	1.6	2.9
B7	14246	13398	14690	18542	19204	19258	15710	23248	14476	14566
Abw. v. u.S.			9.6	38.4	43.3	43.7	17.3	73.5	8.0	8.7
Abw. v. Opt.		-6.0	3.1	30.2	34.8	35.2	10.3	63.2	1.6	2.2
C1	14510	13236	15808	19156	26476	26528	18392	27024	15120	15224
Abw. v. u.S.			19.4	44.7	100.0	100.4	39.0	104.2	14.2	15.0
Abw. v. Opt.		-8.8	8.9	32.0	82.5	82.8	26.8	86.2	4.2	4.9
C2	21978	20194	23366	32216	38916	38742	26824	42552	22684	22820
Abw. v. u.S.			15.7	59.5	92.7	91.8	32.8	110.7	12.3	13.0
Abw. v. Opt.		-8.1	6.3	46.6	77.1	76.3	22.0	93.6	3.2	3.8
C3	27906	25850	29102	40730	42900	42838	31908	54814	28494	28712
Abw. v. u.S.			12.6	57.6	66.0	65.7	23.4	112.0	10.2	11.1
Abw. v. Opt.		-7.4	4.3	46.0	53.7	53.5	14.3	96.4	2.1	2.9
D1		71806	87224	157856	223544	226212	109080	193858	83602	84386
Abw. v. u.S.			21.5	119.8	211.3	215.0	51.9	170.0	16.4	17.5
D2		89980	108040	208096	242796	238972	131196	252334	104108	104886
Abw. v. u.S.			20.1	131.3	169.8	165.6	45.8	180.4	15.7	16.6
D3		105786	124882	231506	241904	245514	146530	313500	120964	121866
Abw. v. u.S.			18.1	118.8	128.7	132.1	38.5	196.4	14.3	15.2
E1		365924	445630	1119358	1296146	1279988	581514	1064294	427954	432332
Abw. v. u.S.			21.8	205.9	254.2	249.8	58.9	190.9	17.0	18.1
E2		534282	631086	1234286	1275866	1284132	773954	1756236	612620	616342
Abw. v. u.S.			18.1	131.0	138.8	140.3	44.9	228.7	14.7	15.4
E3		680296	773012	1192720	1222786	1248110	897692	4111320	759094	762552
Abw. v. u.S.			13.6	75.3	79.7	83.5	32.0	504.3	11.6	12.1

Tabelle 6.2: Lineare Distanzen für die Abarbeitungen der Taskmatrizen aus Tabelle 6.1.

Einheitsdistanzen In der Tabelle 6.3 sind für die Taskmatrizen aus Tabelle 6.1 die Ergebnisse zusammengefasst, die die Algorithmen Min-Change und Top-Down-Cover erzielen. Min-Change ist dabei der im vorigen Abschnitt skizzierte Online-Algorithmus, während Top-Down-Cover den Approximationsalgorithmus mit Worst-Case-Fehler 2 aus Kapitel 3 bezeichnet.

Neben der unteren Schranke aus Gleichung (4.1) ist für die Instanzen, für die wir die minimale Lösung bestimmen konnten, auch diese angegeben. Die untere Schranke liegt bei diesen Beispielen immer um weniger als 6 Prozent unter dem Optimum. Die Ergebnisse des Algorithmus Top-Down-Cover liegen für alle Beispiele um weniger als 10 Prozent über der unteren Schranke und für die Beispiele, für die wir das Optimum berechnen konnten, um weniger als 4 Prozent über dem optimalen Wert. Sie sind damit wesentlich besser als der Worst-Case-Faktor von 2. Diese Ergebnisse scheinen typisch zu sein. Der Online-Algorithmus Min-Change weicht dagegen um bis zu 68 Prozent von der unteren Schranke ab, wobei sich die größeren Abweichungen insbesondere bei großer Maschinenanzahl ergeben. Beim Algorithmus Top-Down-Cover ist ein Anstieg der Abweichungen von der unteren Schranke mit wachsender Maschinenanzahl nicht oder jedenfalls nicht in dem Maße festzustellen. Lediglich mit höheren Taskwahrscheinlichkeiten an den Maschinen scheinen die Abweichungen von der unteren Schranke anzusteigen. Allerdings werden dabei auch die Abweichungen der unteren Schranke vom Optimum größer.

	Optimum	Untere Schranke	Min-Change	Top-Down-Cover
A1	3667	3565	3776	3711
Abw. v. u.S.			5.9	4.1
Abw. v. Opt.		-2.8	3.0	1.2
A2	5925	5698	6467	6053
Abw. v. u.S.			13.5	6.2
Abw. v. Opt.		-3.8	9.1	2.2
A3	3634	3511	3774	3676
Abw. v. u.S.			7.5	4.7
Abw. v. Opt.		-3.4	3.9	1.2
A4	5955	5721	6386	6084
Abw. v. u.S.			11.6	6.3
Abw. v. Opt.		-3.9	7.2	2.2
A5	4078	3958	4190	4125
Abw. v. u.S.			5.9	4.2
Abw. v. Opt.		-2.9	2.7	1.2
A6	3796	3716	4000	3840
Abw. v. u.S.			7.6	3.3
Abw. v. Opt.		-2.1	5.4	1.2
A7	3413	3332	3532	3437
Abw. v. u.S.			6.0	3.2
Abw. v. Opt.		-2.4	3.5	0.7
B1	4052	3935	4214	4102
Abw. v. u.S.			7.1	4.2
Abw. v. Opt.		-2.9	4.0	1.2
B2	2840	2784	2913	2860
Abw. v. u.S.			4.6	2.7
Abw. v. Opt.		-2.0	2.6	0.7
B3	4052	3935	4214	4102
Abw. v. u.S.			7.1	4.2
Abw. v. Opt.		-2.9	4.0	1.2
B4	5174	5005	5453	5252
Abw. v. u.S.			9.0	4.9
Abw. v. Opt.		-3.3	5.4	1.5

	Optimum	Untere Schranke	Min-Change	Top-Down-Cover
B5	6252	6033	6767	6396
Abw. v. u.S.			12.2	6.0
Abw. v. Opt.		-3.5	8.2	2.3
B6	7232	6887	8185	7446
Abw. v. u.S.			18.8	8.1
Abw. v. Opt.		-4.8	13.2	3.0
B7	8233	7769	9781	8490
Abw. v. u.S.			25.9	9.3
Abw. v. Opt.		-5.6	18.8	3.1
C1	5744	5629	5929	5769
Abw. v. u.S.			5.3	2.5
Abw. v. Opt.		-2.0	3.2	0.4
C2	10540	10152	11587	10688
Abw. v. u.S.			14.1	5.3
Abw. v. Opt.		-3.7	9.9	1.4
C3	14802	14042	18590	15187
Abw. v. u.S.			32.4	8.2
Abw. v. Opt.		-5.1	25.6	2.6
D1		28369	32964	29140
Abw. v. u.S.			16.2	2.7
D2		40418	52641	42043
Abw. v. u.S.			30.2	4.0
D3		51173	72963	54061
Abw. v. u.S.			42.6	5.6
E1		143269	191277	147346
Abw. v. u.S.			33.5	2.8
E2		255966	391325	270290
Abw. v. u.S.			52.9	5.6
E3		353674	593145	383482
Abw. v. u.S.			67.7	8.4

Tabelle 6.3: Einheitsdistanzen für Abarbeitungen der Taskmatrizen aus Tabelle 6.1.

Kapitel 7

Zusammenfassung und Ausblick

Wir haben in dieser Arbeit das Problem der Bewegungsminimierung in der Förderband-Flow-Shop-Verarbeitung mit einem Arbeiter untersucht, das unseres Wissens bislang von niemandem untersucht wurde. Dabei betrachteten wir im Wesentlichen Einheitsdistanzen und additive bzw. lineare Distanzen, wobei letztere ein Spezialfall der additiven Distanzen sind. Die Untersuchungen erstreckten sich von Komplexitätsbetrachtungen über Approximations- und Online-Algorithmen bis hin zu empirischen Auswertungen.

Für das Förderband-Flow-Shop-Problem mit vorgegebener Jobreihenfolge konnten wir die NP-Vollständigkeit auch für den eingeschränkten Fall der Einheitsdistanzen nachweisen. Für lineare Distanzen bleibt die Komplexität dagegen offen. Falls auch die Jobreihenfolge gewählt werden darf, erweist sich das Problem sowohl für lineare Distanzen als auch für Einheitsdistanzen als NP-vollständig.

Die folgende Tabelle fasst die erzielten Ergebnisse der Komplexitätsuntersuchungen zusammen.

	allgemeine Distanzen	Einheits-distanzen	additive Distanzen	lineare Distanzen
Jobreihenfolge wählbar	NP-vollständig	NP-vollständig	NP-vollständig	NP-vollständig
Jobreihenfolge fest	NP-vollständig	NP-vollständig	?	?
Jobreihenfolge fest, Jobanzahl konstant	in Polynomzeit lösbar	in Polynomzeit lösbar	in Polynomzeit lösbar	in Polynomzeit lösbar
Jobreihenfolge fest, Maschinenanzahl konstant	in Linearzeit lösbar	in Linearzeit lösbar	in Linearzeit lösbar	in Linearzeit lösbar

Sowohl für Einheitsdistanzen als auch für additive Distanzen haben wir nicht-triviale untere Schranken $\mathrm{LB}(\mathcal{M})$ für die Distanzen der Abarbeitungen einer Taskmatrix $\mathcal{M}$ bestimmt. Ausgehend von diesen unteren Schranken entwickelten wir für beide Fälle Approximationsalgorithmen, deren Laufzeit linear in der Größe der gegebenen Taskmatrix

ist. Diese Algorithmen liefern Abarbeitungen, deren Distanz höchstens um einen Faktor 2 bzw. 3 von der jeweiligen unteren Schranke abweichen. Dies zeigt natürlich auch, dass die Distanz einer minimalen Lösung höchstens das Zweifache bzw. Dreifache unserer unteren Schranken beträgt. Im Falle der Einheitsdistanzen ist diese Abschätzung scharf, d.h. es gibt keine Konstante $c < 2$, so dass für alle Taskmatrizen die Distanz einer minimalen Abarbeitung kleiner oder gleich dem c-fachen der unteren Schranke ist. Definieren wir die Güte der unteren Schranke c_{LB} durch

$$c_{\mathrm{LB}} := \inf \{c \mid \Delta_{\min}(\mathcal{M}) \leq c \cdot \mathrm{LB}(\mathcal{M}) \text{ für alle Taskmatrizen } \mathcal{M}\},$$

so ist die Güte unserer unteren Schranke für Einheitsdistanzen gleich 2. Für lineare Distanzen oder additive Distanzen wissen wir, dass die Güte der unteren Schranke nicht besser als $\frac{5}{3}$ ist.

Im Bereich der Online-Algorithmen konnten wir nachweisen, dass die Abarbeitungen des Algorithmus Free-Left bezüglich beliebiger additiver Distanzmaße höchstens um einen Faktor aus $O(\log m)$ von der minimalen Abarbeitung abweichen, wobei m die Anzahl der Maschinen ist. Die empirischen Auswertungen für den besonders relevanten Spezialfall der linearen Distanzen belegen ferner, dass die Abweichungen von der minimalen Lösung bei zufällig gewählten Taskmatrizen in der Regel weniger als 20 Prozent betragen. Dies ist bereits bei kleiner Maschinenanzahl deutlich besser als die von uns bewiesene obere Schranke $2\log_2(m+2) - 1$ für den kompetitiven Faktor. Allerdings besitzt der Free-Left-Algorithmus selbst für lineare Distanzen keinen konstanten kompetitiven Faktor, da wir zeigen konnten, dass dieser in $\Omega\left(\frac{\log m}{\log\log m}\right)$ liegt. Die guten Resultate des Algorithmus sind auch darauf zurückzuführen, dass er in vielen Entscheidungssituationen eine beweisbar optimale Entscheidung trifft. Für zeilen- und spaltenmonotone Taskmatrizen berechnet der Algorithmus sogar minimale Abarbeitungen. Für die Praxis ist der Algorithmus Free-Left deswegen besonders interessant, weil er auf einer sehr einfachen Entscheidungsregel beruht, die nur sehr wenige, unmittelbar verfügbare Informationen verwendet. Die nachfolgende Tabelle fasst einige der Ergebnisse nochmals zusammen.

	Güte der unteren Schranke	Approximations-algorithmus mit Worst-Case-Fehler	Online-Algorithmus mit kompetitivem Faktor
Einheitsdistanzen	$c_{\mathrm{LB}} = 2$	2	-
lineare Distanzen	$\frac{5}{3} \leq c_{\mathrm{LB}} \leq 3$	3	$\in O(\log m) \cap \Omega\left(\frac{\log m}{\log\log m}\right)$
additive Distanzen	$\frac{5}{3} \leq c_{\mathrm{LB}} \leq 3$	3	$\in \Theta(\log m)$

Im Hinblick auf weitere Untersuchungen wäre es aus theoretischer Sicht besonders interessant, die Komplexität des Problems für lineare bzw. additive Distanzen bei vorgegebener

Jobreihenfolge zu bestimmen und die Frage zu klären, ob es für lineare Distanzen einen Online-Algorithmus mit einem konstanten kompetitiven Faktor gibt. Aus praktischer Sicht erscheint eine Verallgemeinerung des Problems auf mehrere Arbeiter interessant. Allerdings ergeben sich dabei einige grundlegende Probleme, beispielsweise bei der Festlegung der Optimierungsfunktion. Wählt man als Optimierungsfunktion z.B. die Summe der Distanzen, die die Arbeiter zurücklegen, so kann es vorkommen, dass sich in einer minimalen Abarbeitung einige Arbeiter gar nicht, andere dagegen sehr viel bewegen müssen. Dieses ist vermutlich nicht gewünscht. Außerdem kann es bei Optimierungsfunktionen, die nur die von den Arbeitern zurückgelegten Distanzen berücksichtigen, vorkommen, dass einige der Arbeiter in minimalen Abarbeitungen untätig an Maschinen warten, bis dort wieder Tasks auszuführen sind. Auch dieser Effekt ist natürlich nicht erwünscht, da diese Wartezeiten für die Bearbeitung der Tasks verlorengehen. Eine geeignete Optimierungsfunktion wird daher vermutlich die Bearbeitungszeiten der Tasks mit einbeziehen müssen, um diese Wartezeiten der Arbeiter berücksichtigen zu können. Hierdurch ergibt sich eine grundlegende Veränderung der Aufgabenstellung. Die unteren Schranken, die wir in dieser Arbeit entwickelt haben, lassen sich auf das veränderte Problem mit mehreren Arbeitern nicht anwenden. Für eine Analyse der Problematik mit mehreren Arbeitern scheinen daher völlig andersartige Methoden erforderlich zu sein.

Literaturverzeichnis

[1] N. Ascheuer, L. Escudero, M. Grötschel, and M. Stoer. A Cutting Plane Approach to the Sequential Ordering Problem (with Applications to Job Scheduling in Manufacturing). *SIAM Journal on Optimization*, 3:25–42, 1993.

[2] N. Ascheuer, M. Grötschel, and A. Abdel-Aziz Abdel-Hamid. Orderpicking in an Automatic Warehouse: Solving Online Asymetric TSPs. Technical Report 35/98, Konrad-Zuse-Zentrum für Informationstechnik, D-14195 Berlin, Germany, 1998.

[3] J. Błażewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Węglarz. *Scheduling Computer and Manufacturing Processes.* Springer, Berlin, Heidelberg, New York, 1996.

[4] Alan Borodin. *On-line Computation and Competitive Analysis.* Cambridge University Press, 1998.

[5] P. Brucker. *Scheduling Algorithms.* Springer-Verlag, 1998.

[6] R. de Koster. Performance approximation of pick-to-belt orderpicking systems. *European Journal of Operational Research*, 92:558–573, 1994.

[7] W. Espelage and E. Wanke. Movement minimization in conveyor flow shop processing. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 233–234, Philadelphia, 2000. ACM-SIAM.

[8] W. Espelage and E. Wanke. Movement optimization in flow shop processing with buffers. *Mathematical Methods of Operations Research*, 51(3):495–513, 2000.

[9] W. Espelage and E. Wanke. A 3-approximation algorithm for movement minimization in conveyor flow shop processing. In *Proceedings of Mathematical Foundations of Computer Science*, volume 2136 of *LNCS*, pages 363–374. Springer-Verlag, 2001.

[10] A. Fiat and G.J. Woeginger. Online Algorithms: The state of the art. volume 1442 of *LNCS*. Springer-Verlag, 1998.

[11] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, San Francisco, 1979.

[12] E. Gelenbe, G. Pujolle, and L. Kleinrock. *Introduction to Queueing Networks.* John Wiley & Sons, Chichester, 1987.

[13] N.G. Hall, H. Kamoun, and C. Sriskandarajah. Scheduling in robotic cells: complexity and steady state analysis. Working paper, College of Business, The Ohio Stat University, 1995.

[14] N.G. Hall and C. Sriskandarajah. A Survey of Machine Scheduling Problems with Blocking and No-Wait in Process. *Operations Research*, 44:510–525, 1996.

[15] M.S. Manasse, L.A. Mc Geoch, and D.D. Sleator. Competitive algorithms for on-line problems. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 322–333. ACM, 1988.

[16] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, New York, 1989.

[17] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Prentice Hall, Englewood Cliffs, N.J., 1995.

[18] H.D. Ratcliff and A.S. Rosenthal. Orderpicking in a Rectangular Warehouse: A Solvable Case of the Traveling Salesman Problem. *Operations Research*, 31:507–521, 1983.

[19] J. Rethmann and E. Wanke. An approximation algorithm for stacking up bins from a conveyer onto pallets. In *Proceedings of the Workshop on Algorithms and Data Structures*, volume 1272 of *LNCS*, pages 440–449. Springer-Verlag, 1997.

[20] J. Rethmann and E. Wanke. Competitive analysis of on-line stack-up algorithms. In *Proceedings of the Annual European Symposium on Algorithms*, volume 1284 of *LNCS*, pages 402–415. Springer-Verlag, 1997.

[21] J. Rethmann and E. Wanke. Storage Controlled Pile-Up Systems. *European Journal of Operational Research*, 103(3):515–530, 1997.

[22] J. Rethmann and E. Wanke. An optimal algorithm for on-line palletizing at delivery industry. In *Proceedings of the International Symposium on Algorithms and Computation*, volume 1533 of *LNCS*, pages 109–118. Springer-Verlag, 1998.

[23] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, New York, 1986.

[24] S.P. Sethi, C. Sriskandarajah, G. Sorger, J. Błażewicz, and W. Kubiak. Sequencing of parts and robot moves in a robotic cell. *International Journal of Flexible Manufacturing Systems*, 4:331–358, 1992.

[25] C. Sriskandarajah, N.G. Hall, H. Kamoun, and H. Wan. Scheduling large robotic cells. Working paper, University of Toronto, Toronto, 1995.

[26] V.S. Tanaev, V.S. Gordon, and Y.M. Shafransky. *Scheduling Theory. Single-Stage Systems.* Kluwer, Dordrecht, 1994.

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