Bergische Universität Wuppertal

Fakultät für Wirtschaftswissenschaft

Schumpeter School of Business and Economics

Dissertation

Exakte Optimierungsverfahren für die Reihenfolgeplanung in der automobilen Zulieferkette

Eingereicht von:
Dipl.-Math. Paul Göpfert
Wuppertal

Gutachter:
Prof. Dr. Stefan Bock
Lehrstuhl für Wirtschaftsinformatik
und Operations Research

Prof. em. Dr. Gerhard Arminger
Lehrstuhl für Wirtschaftsstatistik

Datum der Disputation: 16. 12. 2016

Die Dissertation kann wie folgt zitiert werden:

urn:nbn:de:hbz:468-20170308-094231-8
[http://nbn-resolving.de/urn/resolver.pl?urn=urn%3Anbn%3Ade%3Ahbz%3A468-20170308-094231-8]

Danksagung

Diese Arbeit entstand während meiner Zeit als wissenschaftlicher Mitarbeiter am Lehrstuhl für Wirtschaftsinformatik und Operations Research von Professor Dr. Stefan Bock.

Die betrachtete Problemstellung ging aus dem vom Bundesland Nordrhein-Westfalen und der Europäischen Union im Rahmen des Ziel2-Programms geförderten Projekt „Produktion.NRW: Echtzeitfähige Losgrößen/Sequenzsteuerung für hochautomatisierte Serienfertigung innerhalb der (automotiven) Wertschöpfungskette“ hervor.

Vielen Dank an Professor Dr. Jörg Rambau, dass er mir schon zu Beginn meines Mathematikstudiums in Bayreuth den Bereich des Operations Research anschaulich und Interesse weckend vorgestellt hat. Die weiterführende und ausführliche Auseinandersetzung mit Themen des Operations Research wäre ohne die Tätigkeit als wissenschaftlicher Mitarbeiter nur schwer möglich gewesen, vielen Dank an Stefan und Iwona, dass ihr mir hier den Einstieg und die Weiterbeschäftigung ermöglicht habt. Für die schöne Zeit am Lehrstuhl mit vielen intensiven Diskussionen, einem guten Miteinander und verschiedenen außeruniversitären Veranstaltungen danke ich allen Kollegen.

Meiner Familie danke ich für die vielfältige Unterstützung und andauernde Motivation zur Fertigstellung dieser Arbeit.

Wuppertal im Februar 2017,
Paul Göpfert

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Kapitel 1

Einführung

Die Kombination der Wissenschaftsgebiete Wirtschaftswissenschaften, Informatik und Mathematik zeichnet die Forschungsrichtung des Operations Research aus. In dieser Arbeit wird ausgehend von einer betriebswirtschaftlichen Problemstellung ein Reihenfolgeplanungsproblem entwickelt, das mit Methoden der angewandten Mathematik exakt gelöst werden soll. Die Evaluation der entwickelten Verfahren wird dann empirisch auf Basis einer Implementation der Algorithmen am Computer durchgeführt. Die Interdisziplinarität des Operations Research wird somit auch mit dieser Arbeit veranschaulicht und es wird gezeigt, wie die Erkenntnisse der Mathematik zur Lösung von Problemen aus der betrieblichen Praxis beitragen können.

1.1 Motivation

> „Streit mit Zulieferer: VW-Produktion an sechs Standorten beeinträchtigt“ (FAZ.NET, 22. August 2016)

> „Zulieferer setzen sich gegen VW durch“ (sueddeutsche.de, 23. August 2016)

Im August 2016 waren viele solcher Schlagzeilen in deutschen Zeitungen zu lesen, als es bei Volkswagen zu einem teilweisen Produktionsstillstand kam. Eine Unternehmensgruppe von Zulieferfirmen weigerte sich, die Werke von Volkswagen weiter zu beliefern. Diese Krise machte auch der breiten Öffentlichkeit klar, dass die von den Kunden gekauften Fahrzeuge nicht mehr nur vom Automobilhersteller produziert werden, sondern ein großer Teil der Wertschöpfung bereits vor den Toren der Werke der Automobilhersteller stattfindet. Deshalb ist es sinnvoll und notwendig, auch diese vorgelagerten Prozesse möglichst effizient zu gestalten, und die in der Zulieferindustrie auftretenden Optimierungsprobleme mit Methoden des Operations Research anzugehen.

Die Fertigungstiefe bei den Automobilherstellern hat in den letzten Jahrzehnten immer weiter abgenommen (Corsten und Gabriel 2004, Seite 23). Dadurch bildet die Zulieferindustrie für die Fertigung von Automobilen einen Industriezweig, der kontinuierlich in den letzten Jahren an Bedeutung gewonnen

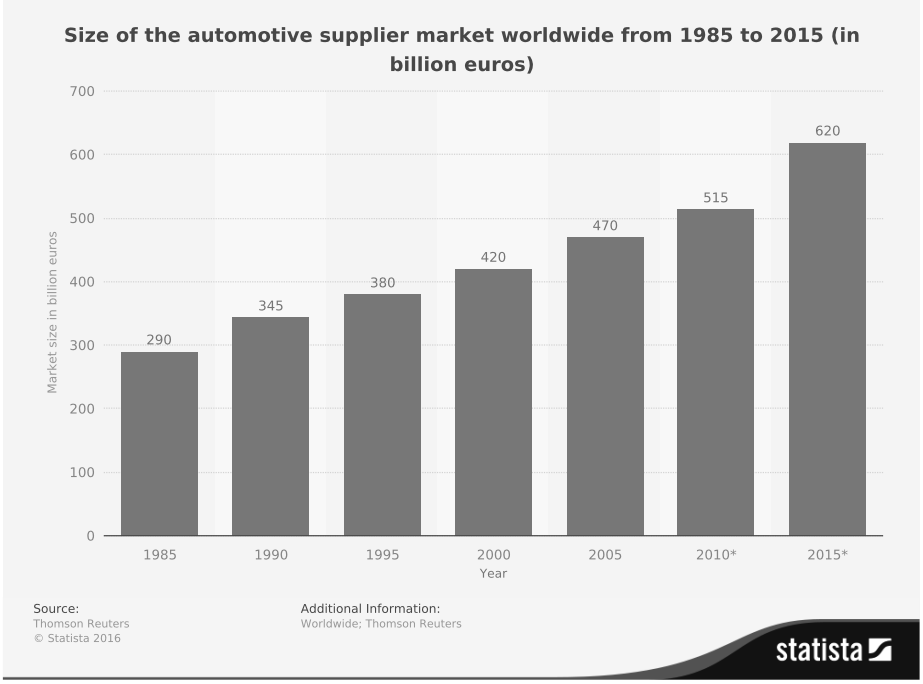


Abbildung 1.1: Größe des Automobilzuliefermarktes

hat (siehe Abbildung 1.1[1]) Aufgrund der hohen Variantenvielfalt durch die individuelle Befriedigung von Kundenwünschen in der Automobilfertigung steigt auch die Variabilität der von den Automobilzulieferern produzierten Komponenten (Corsten und Gabriel 2004, Seite 27). Die Umsetzung des Just-In-Time-Konzepts (Herlyn 2012, Seite 50), um unter anderem auch hohe Lagerbestände zu vermeiden, führt entlang der Zulieferkette zu kleinen Losgrößen und häufigen Lieferungen. Für ein Unternehmen in der automobilen Zulieferkette bedeutet das viele Kundenaufträge auf der Abnehmerseite und zahlreiche Materialbelieferungen, um die eigene Produktion am Laufen zu halten. Um nun unter diesen Bedingungen effizient produzieren zu können, ist ein Produktionsplan, der die Restriktionen des Produktionsprozesses beachtet und eine hohe Güte aufweist, unerlässlich. Welche einzelnen Aspekte der Produktionsplanung bei einem Automobilzulieferer nun genauer betrachtet werden sollen, und was unter der Güte eines Produktionsplans zu verstehen sein soll, ist Thema des nächsten Abschnitts.

1.2 Das Produktionsplanungsproblem

Für die vorliegende Arbeit wollen wir nun eine im Rahmen eines Praxisprojektes evaluierte Problemstellung in ein Ein-Maschinen-Schedulingproblem überfüh-

[1] Bildquelle: Focus Money. Size of the automotive supplier market worldwide from 1985 to 2015 (in billion euros). `https://www.statista.com/statistics/269618/size-of-the-automotive-supplier-market-worldwide-since-1985` (Zugriff am 26. September 2016)

ren. Da das betrachtete produzierende Unternehmen in die automobile Zulieferkette eingebettet ist, ergeben sich die folgenden Eigenschaften der Problemstellung:

1.2.1 Produktionsumgebung

Das betrachtete Unternehmen fertigt auf einer größtenteils automatisierten Fließlinie mit mehreren Umläufen verschiedene Produktvarianten. Die einzelnen Produkte werden in Losen von mehreren Einheiten gefertigt, eine gemischte Fertigung wie sie zum Beispiel die Variantenfließfertigung in der Automobilindustrie erlaubt, ist nicht möglich. In jedem Umlauf wird die zu montierende Produktkomponente auf einem Werkstückträger in mehreren Fertigungszellen (Stationen) montiert und getestet. Wir gehen davon aus, dass die in einem Umlauf fertiggestellte Teilkomponente ohne Zwischenpuffer direkt für den nächsten Umlauf zur Verfügung steht. Da auch manuelle Handgriffe zur Montage der Produkte innerhalb der Taktzeit c der Anlage stattfinden, können wir die gesamte Produktionsanlage als einzelne Maschine im Sinne des Scheduling betrachten. Ein Produktionsauftrag für die Fertigung einer bestimmten Anzahl von x Produkteinheiten wird dann umgewandelt in einen Auftrag j mit einer Prozesszeit $p_j = x \cdot c$.

1.2.2 Aufträge

Bei den Aufträgen gehen wir davon aus, dass zu liefernde Mengen und gewünschte Fertigstellungstermine festgelegt sind, und sich nicht ändern. Ein Auftrag ist immer vollständig zu fertigen, es ist aber erlaubt, mehrere Aufträge für das gleiche Produkt hintereinander zu fertigen, sofern nicht weitere Nebenbedingungen dem entgegenstehen. Eine weitergehende Planung im Sinne der Bestimmung einer Losgröße findet nicht statt[2]. Die gewünschten Fertigstellungszeitpunkte sind einzuhalten, da sonst dem Zulieferer zusätzlich Kosten entweder durch die Beauftragung einer Sonderfahrt des Logistikdienstleisters oder durch die Konventionalstrafe des Automobilherstellers entstehen. Werden die Produkte vor Auslieferung noch in einer weiteren Produktionsstufe des Zulieferers selbst bearbeitet, so sind die Fälligkeitstermine entsprechend vorzuterminieren. Jeder Auftrag besitzt ein Gewicht, das die Dringlichkeit eines Auftrags widerspiegelt. Zur Unterstützung der Qualitätssicherung ist es notwendig, dass alle Aufträge für eine Produktvariante in monoton aufsteigender Sortierung nach ihrem Fälligkeitstermin gefertigt werden (FIFO-Prinzip pro Produktvariante). Dadurch werden Vorrangbeziehungen in Form von Ketten impliziert.

1.2.3 Rüstaufwand

Wird von einem auf der Fließlinie zu fertigendem Produkt zu einem anderen gewechselt, so müssen alle Stationen der Anlage vom Anlagenführer über das neue Produktionsprogramm informiert werden. Auch muss am Ende der Produktionslinie ein anderes Verpackungsmaterial bereitgestellt werden. Zudem sind alle

[2]Siehe zum Beispiel das Lehrbuch von Tempelmeier (2006) für eine Vertiefung in die Losgrößenplanung

Magazinierer, deren im Magazin geladenes Material nicht dem gewünschten Material entspricht, entsprechend umzurüsten[3]. Dazu müssen die vorher benötigten Materialien aus dem Magazinierer entfernt werden, und die Trays mit den neu benötigten Materialien eingesetzt werden. Zur Berechnung der Rüstzeit gehen wir davon aus, dass für jeden Magazinierer entlang der Anlage eine unterschiedliche Rüstzeit anfällt. Diese Differenzen entstehen durch die unterschiedliche Zugänglichkeit des Magazinierers für die Belieferung mit Rohmaterialien oder technische Restriktionen. Muss ein Magazinierer mit anderen Materialien befüllt werden, ist dessen Rüstzeit unabhängig von den auszutauschenden Materialien, sondern nur vom Magazinierer abhängig.

Die anfallende Rüstzeit bestimmt sich nun zum einen aus einer Minimalzeit für den Wechsel des Produktes, die die notwendige Umstellung der Produktionsanlage und des Verpackungsmaterials widerspiegelt. Zum anderen vergrößert jeder notwendige Materialwechsel in einem der Magazinierer den Rüstaufwand. Wir gehen davon aus, dass die Materialwechsel in den Magazinierern der Reihe nach erfolgen, einerseits, weil nur eine Person für die Materialversorgung der Anlage zur Verfügung steht, andererseits, weil die Neubefüllung des Magazinierers erst dann möglich ist, wenn auch die letzte Einheit des zuvor gefertigten Produkts diesen Magazinierer passiert hat. Damit dann die neue Produktvariante gefertigt werden kann, muss nach der letzten Einheit des Vorgängerprodukts die Umbefüllung stattfinden.

Gleichteile sind, wie ihr Name schon sagt, für alle Produktvarianten identisch. Hier erfolgt die Materialzuführung über Wendel- und Stufenförderer, die beliebig nachgefüllt werden können. Da Gleichteile in großen Mengen bestellt werden, gehen wir davon aus, dass diese immer zur Verfügung stehen und in der Problemstellung keine Rolle spielen.

1.2.4 Materialverfügbarkeit

Um überhaupt produzieren zu können, sind wir selbst wieder auf die uns beliefernden Unternehmen im Zuliefernetzwerk angewiesen. Dabei gehen wir davon aus, dass die Materialdisposition mit den Zulieferern genaue Liefertermine vereinbart hat, und sowohl Art und Umfang der Lieferung sowie der tatsächliche Lieferzeitpunkt eingehalten werden. Zur Vereinfachung der Modellierung nehmen wir an, dass zum Lieferzeitpunkt sofort der volle Umfang der Lieferung an der Produktionslinie zur Verfügung steht, und somit für jeden Auftrag, der mit der aktuell vorhandenen Menge an Rohmaterialien gefertigt werden kann, ein sofortiger Produktionsstart möglich ist. Zum Startzeitpunkt der Fertigung muss bereits die gesamte Menge an erforderlichen Materialien zur Verfügung stehen, ein Nachfüllen des zuzuführenden Materials während der Produktion ist nicht möglich. Damit entsteht ein geschlossenes Produktionssystem. So gilt auch die Annahme, dass in der Wartezeit, sofern ein Rüstvorgang erforderlich war, dieser auch schon entsprechend durchgeführt worden ist. Die Magazinierer sind also im Falle einer Wartezeit zum Materialverfügbarkeitszeitpunkt schon von den Materialien für das vorherige Produkt geleert und mit dem gerade gelieferten Material bestückt.

[3]Eine Beschreibung der Funktionsweise sowie der Konstruktion von Magazinierern findet sich in Kapitel 5.3 in Wiendahl (2004)

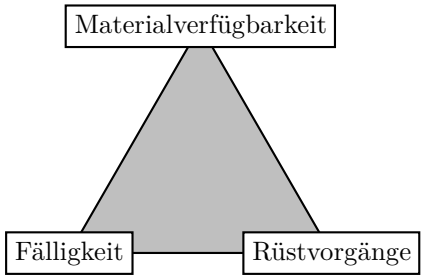


Abbildung 1.2: Spannungsfeld

1.2.5 Personal

Für die Produktion an der betrachteten Anlage ist die Anwesenheit von Personal unerlässlich. Der Betrieb der Anlage muss überwacht werden, und nicht alle Arbeitsgänge sind voll automatisiert. In der Schichtplanung werden die Früh-, Spät- und Nachtschichten, die für einen festgelegten Zeitraum (in der Regel ein bis zwei Wochen) an der Produktionslinie stattfinden sollen, festgelegt. Damit ist dann auch festgelegt, wie viel Produktionskapazität zur Verfügung steht. Die Modellierung vereinfachen wir dahingehend, das weder die Produktionsgeschwindigkeit der Anlage noch die Palette an Produktvarianten, deren Fertigung möglich ist, durch das in einer bestimmten Schicht zur Verfügung stehende Personal beeinflusst wird. Je nachdem, welche Schichten ausgewählt werden, kann nun ein Fälligkeitszeitpunkt für jeden Auftrag in Abhängigkeit der zur Verfügung stehenden Produktionskapazität festgelegt werden. Die Zeitrechnung im aus der Problemstellung abgeleiteten Optimierungsproblem berücksichtigt nur die tatsächlich zur Produktion zur Verfügung stehende Zeit. Alle von der Fertigstellungszeit eines oder mehrerer Aufträge abhängigen Zielfunktionen sind deshalb in Hinblick auf diesen Umstand zu interpretieren.

Da mit der Schichtplanung der entscheidende Kostenfaktor für die Produktion, die Personalkosten, festgelegt sind, müssen wir nun einen Produktionsplan finden, der die damit eingekaufte Produktionskapazität möglichst effizient nutzt. Als Maß dafür verwenden wir die Summe der gewichteten Fertigstellungszeiten aller Aufträge.

1.2.6 Konflikte zwischen Nebenbedingungen und Zielsetzung

Bevor im nächsten Kapitel eine streng formale Definition der Komplexität eines Problems einführt wird, werden hier schon einmal kurz die Schwierigkeiten beschrieben, die sich aus der Kombination der verschiedenen Eigenschaften des Problems ergeben. Das Konfliktpotential der verschiedenen im Problem betrachteten Nebenbedingungen verdeutlicht Abbildung 1.2. Konzentrieren wir uns in der Optimierung auf die Erzeugung eines Produktionsplans mit möglichst wenig Rüstaufwand, kommt das zunächst einmal unserer Zielfunktion zu Gute, da wir wenig Zeit mit nicht produktiven Tätigkeiten verlieren. Doch um mög-

lichst wenig zu rüsten, sind wir darauf angewiesen, möglichst ähnliche Produkte hintereinander zu fertigen, die dann natürlich auch ähnliche Anforderungen an die Verfügbarkeit von Rohmaterialien stellen, was zu unerwünschten Wartezeiten auf Material führen kann. Bei allen Anstrengungen, lange Rüstvorgänge zu vermeiden, kann es passieren, dass wir Aufträge außer Acht lassen, die uns einen hohen Rüstaufwand bescheren, aber dennoch aufgrund ihrer Fälligkeit eingeplant werden müssen. Auch bei der Suche nach einem Produktionsplan mit möglichst wenig Materialwartezeiten kann uns die Fälligkeit eines Auftrages in die Quere kommen, nämlich dann, wenn wir unbedingt einen Auftrag einplanen müssen, der durch seinen Materialverbrauch für die nachfolgenden Aufträge zu hohen Wartezeiten auf Materialverfügbarkeit führt.

1.3 Ergebnisse und Gliederung der Arbeit

Die soeben beschriebene Problemstellung wird in der vorliegenden Arbeit als durch die Kombination der Nebenbedingungen neuartiges Ein-Maschinen-Scheduling-Problem betrachtet. Kernthema der Arbeit ist die Vorstellung von verschiedenen Optimierungsalgorithmen, die geeignet sind, die hinsichtlich der Zielfunktion optimale Produktionsreihenfolge unter Beachtung der gegebenen Nebenbedingungen zu ermitteln.

Um dieses Ziel zu erreichen, werden einerseits sehr bekannte Methoden angewendet, andererseits aber auch fortgeschrittene Techniken des Operations Research in auf die Problemstellung angepasster Weise kombiniert.

Kapitel 2 führt das aus der beschriebenen Problemstellung abgeleitete und in dieser Arbeit erstmalig betrachtete Schedulingproblem ein, und wir beschäftigen uns mit dessen Eigenschaften und seiner Komplexität aus der Perspektive des Scheduling.

In dieser Arbeit wollen wir die im Problem auftretenden Vorrangbeziehungen einerseits zur Zerlegung des Optimierungsproblems in mehrere Teilprobleme verwenden, andererseits bildet die betrachtete Zielfunktion in Kombination mit Vorrangbeziehungen bereits seit Jahrzehnten einen interessanten Forschungsgegenstand. Wie die Ergebnisse dieser Forschungstradition für das vorliegende Optimierungsproblem nutzbar gemacht werden können, ist das Thema von **Kapitel 3**.

Kapitel 4 thematisiert die Eigenschaften zulässiger Lösungen und präsentiert eine Tabusuche zur Generierung guter Lösungen, die als obere Schranken in den folgenden Verfahren eingesetzt werden können.

Ein genereller Lösungsansatz für Optimierungsprobleme ist die Ganzzahlige Programmierung. Nach einer kurzen Einführung in die Methodik stellen wir in **Kapitel 5** insbesondere ein Modell der Ganzzahligen Programmierung für das vorliegende Problem vor und gehen auf alternative Modellierungsansätze ein.

Die Entwicklung eines Branch-and-Bound-Algorithmus mit verschiedenen unteren Schranken wird in **Kapitel 6** diskutiert. Auch wenn die einzelnen Schranken jeweils nur einen kleinen Teil der Nebenbedingungen aufgreifen, lassen sich die anderen im Problem reflektierten Aspekte durch Modifikationen der in der Schranke betrachteten Instanz teilweise erhalten.

Die Ganzzahlige Programmierung und Branch-and-Bound-Algorithmen lassen sich in einem Branch-and-Price-and-Cut-Ansatz kombinieren, der in **Kapitel 7** vorgestellt wird. Dazu ist zunächst ein Zerlegung eines Schedules in

mehrere zu definierende Objekte, die die Spalten beziehungsweise Variablen des Optimierungsmodells bilden, notwendig. Mit großen Rechenaufwand verbunden ist die Suche nach Spalten mit negativen reduzierten Kosten, die deshalb besonders effizient zu gestalten ist. Für diese Problemstellung werden deshalb in dieser Arbeit mehrere Branch-and-Bound-Algorithmen vorgestellt.

Die Evaluation der Optimierungsverfahren wird in **Kapitel 8** erläutert. Neben einem Schema für die Erstellung von Testinstanzen werden auch Testergebnisse für die verschiedenen Ansätze bereitgestellt. Dieses Kapitel beschließt die Arbeit mit einem Resümee und Vorschlägen für die zukünftige Forschung.

Kapitel 2

Das Schedulingproblem

Das in dieser Arbeit ausführlich behandelte Schedulingproblem lässt sich mit der $\alpha|\beta|\gamma$-Klassifikation von Graham u. a. (1979) als

$$1|setups, rm, chains, \delta_j| \sum w_j C_j \tag{2.1}$$

bezeichnen. Hierbei beschreibt der α-Teil die im Problem betrachtete Produktionsumgebung. Wie in der Einführung erwähnt, wollen wir die Produktionsanlage als eine einzelne Maschine auffassen. Der β-Teil beschreibt die Nebenbedingungen des Produktionsprozesses. Wir haben Rüstzeiten (*setups*), Rohmaterialanforderungen (rm), Vorrangbeziehungen in Form von Ketten für die einzelnen Aufträge pro Produkt (*chains*) und Fälligkeiten für jeden Auftrag (δ_j) vorliegen. Die Aufträge sind dabei ohne Unterbrechung (Preemption) zu fertigen. Als Zielfunktion betrachten wir die Summe der gewichteten Fertigstellungszeiten aller Aufträge. In diesem Kapitel wollen wir nun mit den Eigenschaften des Problems vertraut werden und einen Überblick über verwandte Problemstellungen in der Literatur gewinnen.

2.1 Definition der Rüstvorgänge

Die Rohmaterialien $m \in M$ die während des Produktionsprozesses verwendet werden, sind verschiedene Materialgruppen G aus der Menge der Materialgruppen $\mathcal{G}$ zugeordnet. Wir bezeichnen mit $G(m)$ die Materialgruppe von Material $m \in M$. Auch die Menge von Produkten P selbst bildet eine Materialgruppe. Die Stückliste für ein Produkt $p \in P$ beinhaltet genau einen Repräsentanten $m \in G$ für jede rüstrelevante Gruppe $G \in \mathcal{G}$. Sei $Bom : P \times \mathcal{G} \mapsto M$ eine Funktion, die für ein Produkt $p \in P$ und eine Materialgruppe $G \in \mathcal{G}$ dasjenige Material angibt, das als einziges aus G in p verbaut wird. Für jede Gruppe G in der Menge der Materialgruppen $\mathcal{G}$ kommt es zu einer Rüstzeit $st(G)$, wenn von einem Material $m \in G$ zu einem anderen Material $m' \in G, m' \neq m$ aus der selben Gruppe gerüstet werden muss. Wenn auf der Produktionsanlage von einem Produkt auf ein anderes umgerüstet wird, führt das zu einer Rüstzeit, die von den beiden beteiligten Produkten abhängig ist. Wir betrachten die Indika-

torfunktion $I^S : P \times P \times \mathcal{G} \mapsto \mathbb{N}_0$ mit

$$I^S(i,j,G) = \begin{cases} 0 & \text{für } Bom(i,G) = Bom(j,G) \\ st(G) & \text{sonst.} \end{cases} \tag{2.2}$$

Die Rüstzeit $St : P \times P \mapsto \mathbb{N}_0$ zwischen zwei Produkten $i \in P$ und $j \in P$ ist dann definiert durch

$$st(i,j) := \sum_{G \in \mathcal{G}} I^S(i,j,G). \tag{2.3}$$

Die Rüstzeit zwischen zwei Aufträgen $i \in J$ und $j \in J$ entspreche im Folgenden dabei immer der Rüstzeit zwischen den jeweiligen Produkten $p(i)$ und $p(j)$ der Aufträge. Ein spezieller Fall ist der initiale Rüstzustand, der im künstlichen Auftrag 0 betrachtet wird. Wird in einer Instanz kein initialer Rüstzustand durch Definition des zu Beginn der Fertigung gerüsteten Produktes angegeben, beträgt die Rüstzeit zum ersten Auftrag eines Schedules 0.

Lemma 2.1
Die auf obige Weise konstruierten Rüstzeiten erfüllen die Dreiecksungleichung, das heißt für jedes Tupel von $(i,j,k) \in P \times P \times P$ gilt:

$$st(i,k) + st(k,j) \geq st(i,j) \tag{2.4}$$

Beweis. Betrachte die in (2.2) eingeführte Indikatorfunktion für eine beliebige Gruppe $G \in \mathcal{G}$. Dann gilt für jede Gruppe $G \in \mathcal{G}$ die Ungleichung

$$I^S(i,k,G) + I^S(k,j,G) \geq I^S(i,j,G). \tag{2.5}$$

Sobald es nämlich auf der rechten Seite zu einer Umrüstung kommt, es gilt also $Bom(i,G) \neq Bom(j,G)$, muss auf der linken Seite mindestens eine der beiden Gleichungen $Bom(i,G) = Bom(k,G)$ oder $Bom(k,G) = Bom(j,G)$ verletzt sein. Summation der Ungleichung (2.5) über alle Gruppen G aus $\mathcal{G}$ liefert dann mit Gleichung (2.3) die Behauptung (2.4). □

2.2 Materialverfügbarkeit

Für die Materialverfügbarkeit lehnen wir uns an die Notation aus Grigoriev, Holthuijsen und Klundert (2005) an. Für jedes Rohmaterial $m \in M$ aus der Menge der Materialien M sei $R_{m,\tau}$ die bis zum Zeitpunkt τ insgesamt gelieferte Menge des Materials m. Da wir nur eine begrenzte Menge an Materiallieferungen vorliegen haben, und diese nicht kontinuierlich erfolgen, bilden die relevanten Lieferzeitpunkte T eine diskrete Menge. Im Folgenden betrachten wir also nur die Zeitpunkte zur Überprüfung der Materialverfügbarkeit, zu denen sich an den Rohmaterialbeständen eine positive Änderung durch eine Lieferung eines oder mehrerer Materialien auftritt. Nur für diese Zeitpunkte $\tau \in T$ brauchen wir auch eine absolute bis zu diesem Zeitpunkte gelieferte Menge $R_{m,\tau}$ zu speichern und zu verwenden. Der Materialverbrauch eines Auftrags $j \in J$ an Rohmaterial m wird mit $a_{j,m}$ bezeichnet. Um den frühesten Startzeitpunkt S_j für die Produktion eines Auftrags $j \in J$ unter Beachtung der Materialverfügbarkeitsrestriktionen zu bestimmen, wenn zusammen mit j alle Aufträge $J' \subseteq J$ gefertigt

worden sind (es gilt also auch $j \in J'$), können wir folgende Minimumsbetrachtung anwenden:

$$S_j(J') \geq Inv(J') := \min\{\tau \in T : R_{m,\tau} \geq \sum_{i \in J'} a_{i,m} \forall m \in M\}. \quad (2.6)$$

Die Bestimmung des frühesten Zeitpunktes, zu dem alle in einem Bedarfsvektor über M Materialien angegebenen Rohmaterialbedarfe befriedigt werden können, ist in einer asymptotischen Laufzeit von $\mathcal{O}(|M| + |T|)$ möglich. Soll ein ganzer Schedule hinsichtlich der Materialverfügbarkeit überprüft werden, kann immer beim letzten Belieferungszeitpunkt für den Vorgängerauftrag mit der Prüfung der Materialverfügbarkeit begonnen werden. Hierbei ist dann für einen Auftrag auch nur für jede Materialgruppe genau ein Materialbestand interessant, die Auswertung hat dann insgesamt eine Laufzeit von $\mathcal{O}(|M| + |J| \cdot |\mathcal{G}| + |T|)$ über den gesamten Schedule mit $|J|$ Aufträgen. Zu Beginn muss das Array mit den Materialbedarfen noch mit 0 initialisiert werden, letztendlich werden insgesamt T Lieferzeitpunkte untersucht.

2.3 Die Zielfunktion

Als Zielfunktion im Schedulingproblem betrachten wir die Summe der gewichteten Fertigstellungszeiten aller Aufträge. Jede Zeiteinheit, die vor Beendigung des letzten Auftrags des Schedules zur korrekten Ausführung des Schedules notwendig ist, wirkt sich dabei auf den Zielfunktionswert mit dem Gewicht der noch fertigzustellenden Aufträge aus. Aufgrund der individuellen Gewichtung der Aufträge muss deshalb immer bekannt sein, welche Aufträge genau von dieser Verzögerung noch betroffen sind. Soll für ein Ein-Maschinen-Schedulingproblem ohne weitere Nebenbedingungen diese Zielfunktion minimiert werden, so lässt sich mit der Regel von Smith (1956) eine optimale Lösung erzeugen:

Lemma 2.2 (Regel von Smith (1956))
Für eine beliebige Instanz des Problems $1||\sum w_j C_j$ *ist eine optimale Lösung durch eine monoton fallende Sortierung der Aufträge nach* w_j/p_j *gegeben. Das Problem ist deshalb in* $\mathcal{O}(n \log n)$ *lösbar.*

Beweis. Angenommen, es gibt in einer Sequenz S zwei Aufträge i und j, die direkt aneinander angrenzen, und es gelte $w_i/p_i < w_j/p_j \Leftrightarrow w_i \cdot p_j < w_j \cdot p_i$. Nun tauschen wir die beiden Aufträge aus und erhalten eine Sequenz S'. Die Differenz im Zielfunktionswert von S' und S ergibt sich allein aus den Beiträgen von i und j zum Zielfunktionswert:

$$c(S) - c(S') = (w_i \cdot p_i + w_j \cdot (p_i + p_j)) - (w_j \cdot p_j + w_i \cdot (p_i + p_j)) = w_j \cdot p_i - w_i \cdot pj > 0 \quad (2.7)$$

Also hat die neue Sequenz einen niedrigeren Zielfunktionswert. □

Die in Abbildung 2.1 gewählte Darstellung wird in Goemans und Williamson (2000) als zweidimensionales Gantt-Chart bezeichnet, erste Skizzen dieser Art sind in Eastman, Even und Isaacs (1964) zu finden. Wird wie in Abbildung 2.1 die Prozesszeit gegen das verbleibende Gewicht aller Aufträge abgetragen, so sehen wir, dass die Aufträge selbst immer einen konstanten Anteil am Zielfunktionswert haben. Insbesondere ist auch der Anteil über der gepunkteten Linie,

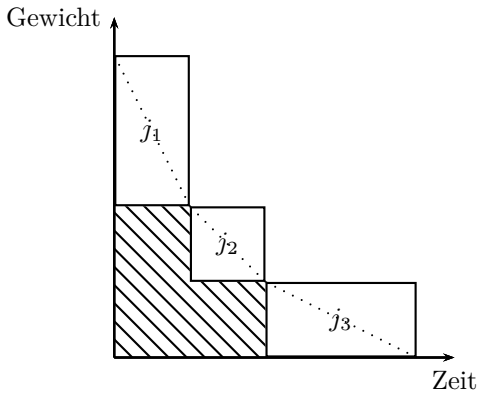


Abbildung 2.1: Visualisierung der Regel von Smith (1956)

die die Prioritäten der Aufträge verdeutlicht, immer konstant. Durch die Sortierung der Aufträge nach absteigenden w_j/p_j wird somit die schraffierte Fläche unterhalb der Aufträge minimiert.

Den Bruch w_j/p_j bezeichnen wir im Folgenden auch als Priorität eines Auftrags.

2.4 Evaluation von Schedules

Zunächst wollen wir die in der Schedulingliteratur weit verbreitete Definition (siehe zum Beispiel Pinedo 2008; Conway, Maxwell und L. W. Miller 2003) von drei Schedule-Klassen betrachten und auf unser Optimierungsproblem anwenden:

Definition 2.3 (Schedule-Klassen)
*Ein Schedule ist **verzögerungsfrei**, wenn immer dann, wenn die Maschine einen weiteren Auftrag ausführen könnte, auch ein Auftrag eingeplant wird. Ein Schedule ist **aktiv**, wenn kein Auftrag früher fertiggestellt werden kann, ohne die Fertigstellung eines anderen Auftrags zu verzögern. Wir nennen einen Schedule **semi-aktiv**, wenn kein Auftrag früher fertiggestellt werden kann, ohne die Reihenfolge auf der Maschine zu ändern.*

Da wir mit der Summe der gewichteten Fertigstellungszeiten eine reguläre Zielfunktion vorliegen haben, der Zielfunktionswert bei der späteren Fertigstellung eines Auftrags bei Beibehaltung aller anderen Fertigstellungszeiten also nicht besser wird, sind für uns auf der Suche nach der optimalen Lösung nur Schedules interessant, die aktiv sind. Ansonsten könnten wir durch die Vertauschung von Aufträgen noch eine frühere Fertigstellung eines Auftrags erreichen, und damit den Zielfunktionswert weiter verbessern.

Die ausschließliche Betrachtung von verzögerungsfreien Schedules kann dagegen zu einer Lösung führen, die nicht optimal ist. In unserem Fall könnte es sein, dass für einen sehr wichtigen Auftrag noch eine kurze Wartezeit auf Material in Kauf genommen werden muss, während ein unwichtiger Auftrag sofort eingeplant werden könnte. Eine Schedulingpolitik für verzögerungsfreie Schedules würde jetzt die Produktion des unwichtigen Auftrags vor dem wichtigen

Auftrag forcieren, obwohl die andere Reihenfolge einen besseren Zielfunktionswert liefert.

Jede Permutation der Aufträge können wir als semi-aktiven Schedule auswerten, indem wir immer dann mit der Produktion eines Auftrags starten, sobald der Rüstvorgang abgeschlossen ist und das notwendige Material vorhanden ist.

Ein solcher semi-aktiver Schedule kann nur dann nicht aktiv sein, wenn eine Wartezeit auf Materialverfügbarkeit vorliegt. Liegt nämlich keine Wartezeit vor, so ergibt sich aus der Dreiecksungleichung für die Rüstzeiten (siehe Lemma 2.1), dass bei Vorziehen eines Auftrags vor einen anderen dieser dann später fertiggestellt werden würde. Auch wenn die Wartezeit auf Materialien groß genug ist, um einen weiteren Auftrag, dessen Materialien vorhanden sind, einzuplanen, ist dies immer noch kein hinreichendes Kriterium dafür, dass der semi-aktive Schedule nicht aktiv war, da durch das Vorziehen des Auftrags Materialmangel bei anderen Aufträgen entstehen kann. Deren Fertigstellungszeit wird dann verzögert werden.

Als Fazit für diesen Abschnitt können wir festhalten, dass wir immer an semiaktiven Schedules interessiert sind, uns es jedoch schwerfällt, semi-aktive von aktiven Schedules zu trennen, ohne die Folgen eines Tausches bis zum letzten Auftrag eines Schedules durchrechnen zu müssen. Einfacher wird es nur dann, wenn wir wissen, dass der vorgezogene Auftrag keinen Einfluss mehr auf die Materialverfügbarkeit aller nachfolgenden Aufträge hat.

2.5 Komplexitätstheorie

Für die Grundversion der Minimierung der Summe der Fertigstellungszeiten konnten wir eine einfache Prioritätsregel zur Erzeugung einer optimalen Lösung angeben, das Optimierungsproblem ist somit leicht zu lösen. In den folgenden Abschnitten werden wir sehen, dass die Einführung von zusätzlichen Nebenbedingungen die Existenz von solchen Regeln zur optimalen Lösung des betrachteten Problems höchstwahrscheinlich ausschließt und wir somit ein schwer zu lösendes Optimierungsproblem vorliegen haben. Um nun eine Klassifikation von Optimierungsproblemen in leicht zu lösende auf der einen Seite und schwer zu lösende auf der anderen Seite vorzunehmen, wurde Anfang der siebziger Jahre des zwanzigsten Jahrhunderts die Komplexitätstheorie entwickelt. Anstelle von Optimierungsproblemen wird hier mit Entscheidungsproblemen gearbeitet, die mit „ja“ beziehungsweise „nein“ zu beantworten sind. Jedes Optimierungsproblem kann in ein Entscheidungsproblem überführt werden, indem nach einer Lösung für das Optimierungsproblem gefragt wird, die einen bestimmten Zielfunktionswert nicht überschreitet (im Falle der Minimierung). Auch die Frage nach der Existenz einer zulässigen Lösung eines Optimierungsproblems ist ein Entscheidungsproblem. Die nun vorgestellte Klassifikation weist dann für ein vorliegendes Entscheidungsproblem nach, dass dieses mindestens genauso schwer zu lösen ist wie andere, bereits als schwer bekannte Probleme.

Zur Vertiefung der Theorie (eine ausführliche Einführung und ein Standardwerk bildet das Buch von Garey und D. S. Johnson (1979)) wollen wir uns zunächst aber die einfachen Probleme ansehen:

Definition 2.4 (Problemklasse $\mathcal{P}$)
Die Menge aller Entscheidungsprobleme, für die in Polynomialzeit eine Lösung berechnet werden kann, bezeichnen wir mit $\mathcal{P}$.

Eine weitere Problemklasse, in der alle Probleme aus $\mathcal{P}$ automatisch eingeschlossen sind, ist die folgende Problemklasse:

Definition 2.5 (Problemklasse $\mathcal{NP}$)
Die Menge aller Entscheidungsprobleme, für die ein Zertifikat in Polynomialzeit überprüft werden kann, bezeichnen wir mit $\mathcal{NP}$.

Hier wird für ein Problem nur gefordert, dass ein Zertifikat für die Lösbarkeit in Polynomialzeit ausgewertet werden können muss, wir müssen also in der Lage sein, schnell zu erkennen, ob ein Objekt, dass eine Lösung kodiert, tatsächlich auch eine Lösung unseres Entscheidungsproblems darstellt. Bei den Problemen aus $\mathcal{P}$ bekommen wir sogar das Zertifikat in Polynomialzeit geliefert, es gilt also $\mathcal{P} \subseteq \mathcal{NP}$. Die noch offene und bis zum heutigen Tage ungelöste Frage ist, ob auch $\mathcal{NP} \subseteq \mathcal{P}$ gilt.

Solange wir diese Frage nicht beantworten können, wollen wir nun ein Problem als schwer bezeichnen, wenn das Problem mindestens genauso schwer wie alle anderen Probleme in $\mathcal{NP}$ ist. Was unter dem Vergleich „genau so schwer, wie“ zu verstehen ist, zeigt die folgende Definition:

Definition 2.6 (Reduktion)
Ein Entscheidungsproblem P lässt sich auf ein anderes Entscheidungsproblem Q reduzieren, sobald eine Funktion $f : P \mapsto Q$ existiert, die zu jeder Instanz p von P in Polynomialzeit eine Instanz q von Q erzeugt, so dass es für q nur dann ein positives Zertifikat gibt, wenn auch p ein positives Zertifikat besitzt.

Und so sind wir jetzt in der Lage, ein Problem als „höchstwahrscheinlich schwer zu lösen“ zu klassifizieren:

Definition 2.7 ($\mathcal{NP}$-schwer, $\mathcal{NP}$-vollständig)
Ein Problem Q heißt $\mathcal{NP}$-schwer, wenn sich jedes Problem aus $\mathcal{NP}$ in Polynomialzeit auf Q reduzieren lässt. Ein Problem Q heißt $\mathcal{NP}$-vollständig, wenn es $\mathcal{NP}$-schwer ist und selbst in $\mathcal{NP}$ ist, also $Q \in \mathcal{NP}$ erfüllt ist.

Das erste Problem, dessen Komplexität als $\mathcal{NP}$-schwer klassifiziert wurde, ist das Erfüllbarkeitsproblem (engl. Satisfiability (SAT) Cook 1971). Ein erster Grundstock von 21 $\mathcal{NP}$-schweren Problemen wurde dann mit dem Artikel Karp (1972) gelegt.

Um nun ein Problem als $\mathcal{NP}$-schwer zu charakterisieren, müssen wir nicht alle Probleme in $\mathcal{NP}$ betrachten, und dann eine Reduktion konstruieren, sondern es reicht aus, wenn wir dies ausgehend von einem $\mathcal{NP}$-vollständiges Problem Q schaffen, denn wenn alle Probleme aus $\mathcal{NP}$ sich auf Q reduzieren lassen, so lassen sich diese dann auch über Q in Polynomialzeit auf unser Problem P reduzieren, wenn beide Reduktionen hintereinander ausgeführt werden.

Ein Entscheidungsproblem wird als $\mathcal{NP}$-schwer im strengen Sinne bezeichnet, wenn die Komplexität unabhängig von der verwendeten Kodierung der Instanz ist, die Größe der vorkommenden Zahlen also keine Rolle spielt. Ein Problem das diese Voraussetzung nicht erfüllt, ist zum Beispiel das Rucksackproblem, das nur dann zu einem $\mathcal{NP}$-schweren Problem wird, wenn die Kapazität des Rucksacks exponentiell mit der Problemgröße ansteigt.

Nach dieser Einführung in die Theorie sind wir nun in der Lage, auch den Komplexitätsstatus des in der Arbeit betrachtete Schedulingproblems zu bestimmen:

Lemma 2.8
Das Schedulingproblem

$$1|setups, rm, chains, \delta_j| \sum w_j C_j \tag{2.8}$$

ist $\mathcal{NP}$-vollständig im strengen Sinne.

Beweis. Zu zeigen ist, dass das Problem in $\mathcal{NP}$ enthalten ist (Teil 1), und sich alle Probleme aus $\mathcal{NP}$ auf dieses Problem reduzieren lassen (Teil 2).

Teil 1: Für jede beliebige Permutation von Aufträgen können wir in Polynomialzeit die folgenden Aufgaben durchführen:

- ✓ Die Ermittlung der Zulässigkeit der Permutation hinsichtlich der Vorrangbeziehungen für jedes Produkt.
- ✓ Die Berechnung der Fertigstellungszeiten der Aufträge, wenn wir die Permutation als semi-aktiven Schedule auswerten und dabei Rüstzeiten und Wartezeiten auf Materialverfügbarkeit berücksichtigen.
- ✓ Die Ermittlung der Zulässigkeit der Fertigstellungszeiten hinsichtlich der Fälligkeiten der Aufträge.
- ✓ Die Ermittlung des Zielfunktionswertes auf Basis der Fertigstellungszeiten.

Damit kann jede Permutation von $|J|$ Elementen als ein Zertifikat von polynomieller Größe für das Entscheidungsproblem angesehen werden.

Teil 2: Die Komplexität ergibt sich aus der Komplexität der im Problem enthaltenen Unterprobleme, so ist für die gegebene Zielfunktion der Summe der gewichteten Fertigstellungszeiten allein die Betrachtung von Materialverfügbarkeiten oder von Fälligkeiten bereits $\mathcal{NP}$-schwer im strengen Sinne. Siehe hierzu die Beweise in den folgenden Abschnitten 2.7 und 2.8. □

In den nun folgenden Abschnitten wollen wir nun die einzelnen Aspekte des Problems einer komplexitätstheoretischen Betrachtung unterziehen sowie Hinweise zur jeweils vertiefenden Literatur geben.

2.6 Scheduling mit Vorrangbeziehungen

Die Minimierung der Summe der gewichteten Fertigstellungszeiten bei Vorliegen von beliebigen Vorrangbeziehungen, also das Problem $1|prec|\sum w_j C_j$ ist $\mathcal{NP}$-schwer, siehe dazu (Lenstra und Kan 1978) und (Lawler 1978). Analysieren wir nun die sich für die Aufträge ergebenden Zeitfenster und leiten weitere Vorrangbeziehungen ab, so haben wir anstelle von Ketten in der Regel einen beliebigen Vorranggraphen vorliegen. Allerdings lassen sich für viele Spezialfälle von Vorranggraphen Polynomialzeitalgorithmen angegeben, darunter auch für das Problem $1|chains|\sum w_j C_j$ (Sidney 1975). Den polynomiell lösbaren Fällen ist im Kapitel 3 ein eigener Abschnitt gewidmet.

2.7 Scheduling mit Zeitfenstern

Die Minimierung der gewichteten Fertigstellungszeit bei Vorliegen von Freigabezeitpunkten oder Deadlines, also die Probleme $1|r_j|\sum w_jC_j$ und $1|\delta_j|\sum w_jC_j$ sind $\mathcal{NP}$-schwer, siehe dazu (Lenstra, Kan und Brucker 1977). Da die Autoren in (Lenstra, Kan und Brucker 1977) nur eine Reduktion vom Rucksackproblem auf das Problem $1|\delta_j|\sum w_jC_j$ durchführen, greifen wir den Vorschlag der Autoren auf und erzeugen eine Reduktion von 3-Partition auf $1|\delta_j|\sum w_jC_j$.

Definition 2.9 (3-Partitionierung)
Gegeben sei eine Menge von Zahlen A mit $3m$ Elementen, eine Schranke $B \in \mathbb{Z}^+$, und eine Größe $s(a) \in \mathbb{Z}^+$ für jedes Element $a \in A$, so dass $B/4 < s(a) < B/2$ und $\sum_{a\in A} s(a) = mB$. Gesucht ist eine Zerlegung der Menge A in m verschiedene Teilmengen $A_1, \ldots, A_m$ so dass für alle Mengen $A_i, i = 1, \ldots m$ gilt: $\sum_{a\in A_i} s(a) = B$.

Dieses Problem ist $\mathcal{NP}$-schwer im strengen Sinne (Garey und D. S. Johnson 1979). Die Größenbeschränkung für jedes Element der Menge führt dazu, dass jede Menge A_i genau drei Elemente enthalten muss.

Zur Reduktion des Problems erzeugen wir aus einer Instanz der 3-Partitionierung nun auf folgende Weise eine Instanz von $1|\delta_j|\sum w_jC_j$: Die Gesamtzahl der Aufträge sei $4m-1$, aus jeder Zahl a wird ein Auftrag mit Gewicht $w_a = s(a)$ und Prozesszeit $p_a = s(a)$. Zusätzlich führen wir $m-1$ weitere Aufträge ein, deren Gewicht $w_i = 0$ gesetzt wird, mit einer Prozesszeit von $p_i = 1$ für $i = 1, \ldots, m-1$. Diese weiteren Aufträge erhalten eine Fälligkeit von $\delta_i = (B+1)\cdot i$ für $i = 1, \ldots, m-1$. Die Fälligkeit der aus den Zahlen $a \in A$ abgeleiteten Aufträge wird dagegen auf die Gesamtprozesszeit aller Aufträge der Instanz gesetzt, es gilt also $\delta_a = B\cdot m + m - 1$ Der angestrebte Zielfunktionswert ist

$$y \leq \sum_{a\in A} \sum_{a'\in A, a\leq a'} s(a)\cdot s(a') + \frac{1}{2}\cdot m\cdot(m-1)\cdot B. \tag{2.9}$$

Während der erste Summand des Zielfunktionswertes den reinen Beitrag der aus der Menge A abgeleiteten Aufträge zum Zielfunktionswert darstellt, bildet der zweite Summand die Verzögerung dieser Aufträge durch die Zusatzaufträge ab. Hierbei und in den folgenden Beweisen gehen wir davon aus, dass die Elemente in A alle indiziert sind, um die Vergleichsoperation $a \leq a'$ zu ermöglichen.

Es existiere nun eine 3-Partitionierung von A. Dann lässt sich eine Lösung mit Zielfunktionswert y des Schedulingproblems erzeugen, indem abwechselnd immer drei Aufträge mit $\sum_{a\in A_i} w_a = B$ und $\sum_{a\in A_i} p_a = B$ für ein $i = 1, \ldots, m$ eingeplant werden, und dann der als nächstes fällige Zusatzauftrag mit Prozesszeit $p_j = 1$, der dann genau zu seiner Deadline fertiggestellt wird, siehe dazu Abbildung 2.2. Dabei ist die Reihenfolge der Aufträge innerhalb der Blöcke, die von je drei Aufträgen mit Gesamtgewicht und Gesamtprozesszeit B gebildet werden, sowie die Reihenfolge dieser Blöcke selbst beliebig, da alle Aufträge die aus der Menge A generiert werden, die gleiche Priorität haben. Nun ist noch zu zeigen, dass ein solcher Schedule der einzig mögliche ist, um einen Zielfunktionswert $\leq y$ zu erreichen, und damit eine 3-Partitionierung der $a \in A$ möglich sein muss. Da wir im zur 3-Partitionierung korrespondierenden Schedule keinen der zusätzlichen Aufträge später ausführen können, müssen wir überprüfen, ob wir nicht einen dieser Aufträge früher ausführen könnten, um damit die in Abbildung 2.2 vorliegende 3-Partitionierung aufzubrechen. Sobald wir aber einen

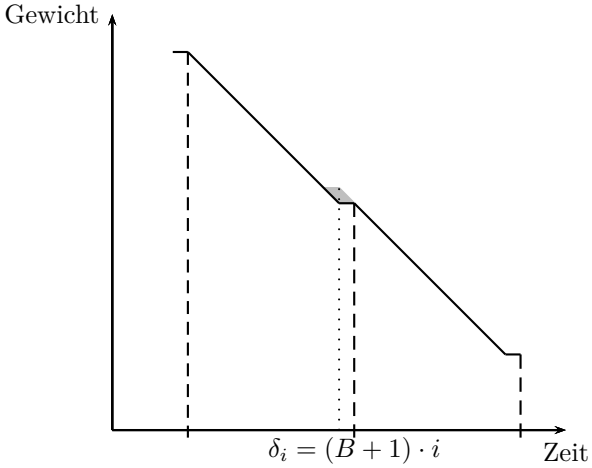


Abbildung 2.2: Reduktion von 3-Partitionierung auf $1|\delta_j|\sum w_j C_j$

dieser zusätzlichen Aufträge vor seiner Fälligkeit fertigstellen wollen, steigt der Zielfunktionswert an, wie die graue Fläche in Abbildung 2.2 verdeutlicht.

Sind bei Vorliegen von Fälligkeiten δ_j für jeden Auftrag $j \in J$ jedoch die Gewichte alle identisch, es gilt also $w_j = 1$ für alle $j \in J$, so ist das Problem noch in Polynomialzeit lösbar (Smith 1956).

Nun wollen wir statt der Fälligkeiten für jeden Auftrag Freigabezeitpunkte r_j betrachten, die Produktion von Auftrag j darf also frühestens zum Zeitpunkt r_j starten. War soeben der Fall mit identischen Gewichten noch einfach lösbar, so ist nun bereits das Problem $1|r_j|\sum C_j$ $\mathcal{NP}$-schwer im strengen Sinne. In Lenstra, Kan und Brucker (1977) wird allerdings lediglich auf eine Reduktion des Rucksackproblems auf $1|r_j|\sum C_j$ aus (Kan 1976) oder die $\mathcal{NP}$-Schwere im strengen Sinne von $F2||\sum C_j$ (Garey, D. S. Johnson und Sethi 1976) verwiesen und ein Beweis der Reduzierbarkeit der 3-Partitionierung auf $1|r_j|\sum C_j$ dem Leser überlassen. Jedoch wird in Kan (1976) (Theorem 4.26) auch für das Problem mit individuellen Gewichten, $1|r_j|\sum w_j C_j$, die $\mathcal{NP}$-Vollständigkeit durch eine Reduktion des Rucksackproblems vorgestellt. Diese kann in eine Reduktion der 3-Partitionierung umgewandelt werden, um die $\mathcal{NP}$-Schwere im strengen Sinne nachzuweisen.

Setze hierfür die Gesamtzahl der Aufträge wieder auf $4 \cdot m - 1$, für die Aufträge, die sich aus den Zahlen der Instanz der 3-Partitionierung ergeben, setze auch wiederum $w_a = p_a = s(a)$. Zudem sei für diese Aufträge der Freigabezeitpunkt $r_a = 0$ vorgegeben. Für die $m - 1$ zusätzlichen Aufträge sei dann der Freigabezeitpunkt bei $r_i = B \cdot i + i - 1$ für $i = 1, \ldots, m - 1$. Das Gewicht der zusätzlichen Aufträge wird auf $w_i = 2$ gesetzt, während für die Prozesszeit $p_i = 1$ gilt. Für den angestrebten Zielfunktionswert gilt jetzt:

$$y \leq \sum_{a \in A} \sum_{a' \in A, a \leq a'} s(a) \cdot s(a') + \frac{3}{2} \cdot m \cdot (m-1) \cdot B + m \cdot (m-1) \qquad (2.10)$$

Um diesen Zielfunktionswert zu erreichen, müssen wir jeden Zusatzauftrag zu seinem Freigabezeitpunkt starten, zwischen den Zusatzaufträge stehen dann

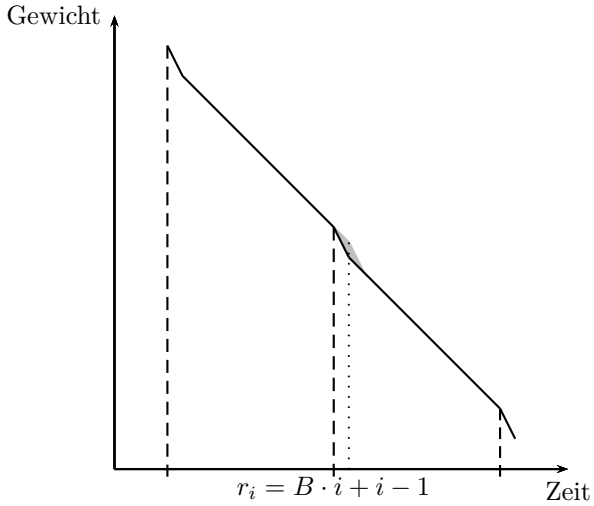


Abbildung 2.3: Reduktion von 3-Partitionierung auf $1|r_j|\sum w_j C_j$

immer genau B Zeiteinheiten für weitere Aufträge zur Verfügung, siehe dazu Abbildung 2.3. Eine 3-Partitionierung der Menge A erreicht damit diesen Zielfunktionswert. Im Gegensatz zur Betrachtung der Fälligkeiten tragen hier nun auch die Zusatzaufträge aktiv zum Zielfunktionswert bei. Dies geschieht zusätzlich zur Verzögerung der aus der Menge A abgeleiteten Aufträge. Während im Problem $1|\delta_j|\sum w_j C_j$ die frühere Fertigstellung der zusätzlichen Aufträge die Lösung teurer gemacht hat, ist es in diesem Fall die spätere Fertigstellung und damit auch der spätere Start als zum Freigabezeitpunkt der Zusatzaufträge, die den angestrebten Zielfunktionswert unmöglich macht. Auch hier verdeutlicht die graue Fläche in Abbildung 2.3 den Anstieg im Zielfunktionswert, wenn diesmal um eine Zeiteinheit später gestartet wird.

Das Problem mit Freigabezeitpunkten ist für uns auch aus einem ganz bestimmten Grund interessant: Für den Fall, dass die Produkte sich in ihrem Materialverbrauch vollständig unterscheiden, werden alle Probleme, die die Materialverfügbarkeit als Nebenbedingung betrachten, zu Problemen mit Freigabezeitpunkten, da keinerlei Konkurrenz zwischen den Produkten um die Rohmaterialien herrscht, und jeder Auftrag auf sein individuellen Belieferungszeitpunkt warten muss (siehe auch Grigoriev, Holthuijsen und Klundert 2005). Für das Scheduling mit Rohmaterialbeständen wollen wir uns darum im folgenden Abschnitt das andere Extrem anschauen, alle Aufträge konkurrieren dann um genau ein Rohmaterial und dessen Lieferungen.

2.8 Scheduling mit Rohmaterialbeständen

Die grundlegenden Probleme $1|rm|C_{\max}$ und $1|rm|L_{\max}$ für das Scheduling mit Rohmaterialbeständen werden in Grigoriev, Holthuijsen und Klundert (2005) behandelt. Der Komplexitätsbeweis der Autoren weist bereits für ein Rohmaterial und beliebige Prozesszeiten die $\mathcal{NP}$-Schwere im strengen Sinne des ersten Problems $1|rm|C_{\max}$ nach. Das Problem $1|rm|L_{\max}$ ist für den Fall von $rm = 2$ (die Aufträge konkurrieren also um zwei verschiedene Rohmaterialien

en) und einheitliche Prozesszeiten $p = 1$ $\mathcal{NP}$-schwer im strengen Sinne. Da der Komplexitätsbeweis auf einen Zielfunktionswert $L_{\max} \leq 0$ prüft, ist auch die Frage nach einem zulässigen Schedule für ein Problem mit Rohmaterialanforderungen und Fälligkeiten ein $\mathcal{NP}$-schweres Problem. Für den Fall der Summe der gewichteten Fertigstellungszeiten wollen wir nun mit Hilfe der Konstruktionen aus dem vorherigen Abschnitt den Nachweis der $\mathcal{NP}$-Schwere im strengen Sinne erbringen:

Lemma 2.10
Das Problem der 3-Partitionierung lässt sich auf

$$1|rm = 1|\sum w_j C_j \tag{2.11}$$

reduzieren.

Beweis. Sei dazu die Anzahl der Aufträge $3 \cdot m$, und es gelte für die Aufträge $w_a = p_a = s(a)$ für $a \in A$. Zudem werden zu den Zeitpunkten $i \cdot B, i = 0, \ldots, m-1$ jeweils B Einheiten des Rohmaterials geliefert. Andersherum lässt sich sagen, dass bis zum Zeitpunkt $i \cdot B$ noch mindestens $(m-i) \cdot B$ Einheiten des Rohmaterials geliefert werden müssen. Diese ausstehenden Materiallieferungen sind in Abbildung 2.4 durch die schraffierte Fläche gekennzeichnet. Der Rohmaterialverbrauch m_a eines Auftrages entspreche jeweils der Prozesszeit p_a des Auftrags, es gilt also $m_a = p_a = w_a = s(a)$. Dann lässt sich ein Zielfunktionswert

$$y \leq \sum_{a \in A} \sum_{a' \in A, a \leq a'} s(a) \cdot s(a') \tag{2.12}$$

nur erreichen, wenn auch eine 3-Partitionierung möglich ist: Sei eine erfüllbare Instanz zur 3-Partitionierung gegeben, dann erreicht der Schedule, der die Teilmengen der Partitionierung, sowie die Aufträge innerhalb der Teilmengen in beliebiger Reihenfolge anordnet, den angegebenen Zielfunktionswert. Wie auch Abbildung 2.4 verdeutlicht, kommen wir bei Vorliegen der 3-Partitionierung nie in Materialnöte, und können alle Aufträge ohne Wartezeit fertigen, so dass sich der Zielfunktionswert ergibt.

In der anderen Richtung ist zu zeigen, dass nur das Vorliegen einer 3-Partitionierung diesen Zielfunktionswert ermöglicht. Sei dazu eine beliebige Reihenfolge aller $3 \cdot m$ Aufträge gegeben, und die Unterteilung der Sequenz in Blöcke zu je drei Aufträgen bilde keine 3-Partitionierung hinsichtlich der Prozesszeiten beziehungsweise Gewichte und Materialbedarfe der Aufträge. Dann erreicht diese Reihenfolge nur dann einen Zielfunktionswert von y, wenn es keinen Auftrag gibt, der auf die Materialverfügbarkeit warten muss. Deshalb gehen wir nun davon aus, dass alle Aufträge direkt hintereinander ohne Wartezeit gefertigt werden. Da keine 3-Partitionierung vorliegt, gibt es jetzt aber einen Auftrag $j \in J$ und ein m' mit $1 \leq m' < m$, so dass für die Startzeit S_j des Auftrags j gilt $S_j < m' \cdot B$, für die Fertigstellungszeit jedoch $C_j > m' \cdot B$. Da jedoch aufgrund der Gleichheit $p_j = m_j$ die Fertigstellungszeit auch den Materialbedarf zur Ausführung von j angibt, und $C_j > m' \cdot B$ gilt, wissen wir, dass wir aufgrund der maximalen Materialverfügbarkeit von $m' \cdot B$ Einheiten des Materials zum Zeitpunkt $S_j < m' \cdot B$ den Auftrag j nicht zum Zeitpunkt S_j hätten starten dürfen, sondern frühestens zum Zeitpunkt $m' \cdot B$. In dieser Reihenfolge tritt somit eine Wartezeit auf Material auf, die den angestrebten Zielfunktionswert

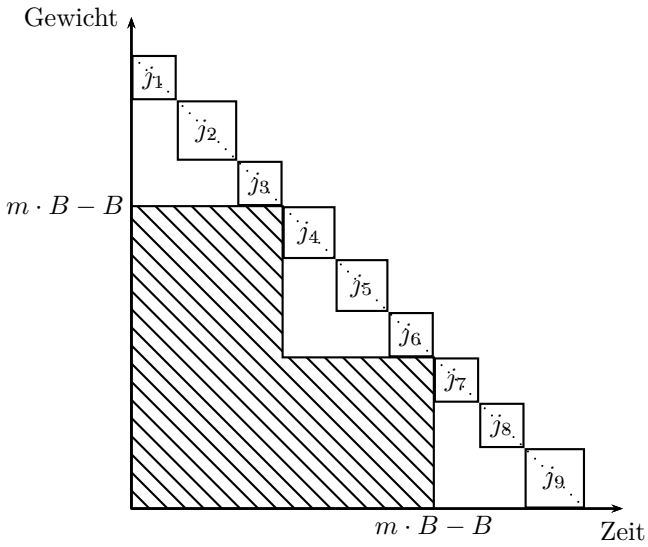


Abbildung 2.4: Reduktion von 3-Partitionierung auf $1|rm|\sum w_j C_j$

nicht mehr realisierbar macht und damit ist gezeigt, dass nur das Vorliegen einer 3-Partitionierung den Zielfunktionswert y möglich macht. □

Sind in einem Schedulingproblem auch die Lieferungen des Materials frei planbar, so ist nur in sehr speziellen Fällen eine Lösung des Problems in Polynomialzeit möglich, sollte für die Lieferungen selbst dann eine Prozesszeit > 0 eingeplant werden müssen (Briskorn u. a. 2010).

2.9 Scheduling mit Rüstzeiten

Neben der in diesem Schedulingproblem zutage tretenden Notwendigkeit, Rüstzeiten explizit zu betrachten, werden auch im Artikel von Allahverdi und Soroush (2008) verschiedene weitere Beispiele von Problemstellungen vorgestellt, in denen Rüstüberlegungen nicht zu vernachlässigen sind. Einen ausführlichen Überblick über die Schedulingliteratur zu Rüstzeiten und -kosten bildet die Reihe von Übersichtsartikeln Allahverdi, Gupta und Aldowaisan (1999), Allahverdi, Ng u. a. (2008) und Allahverdi (2015).

2.9.1 Reihenfolgeunabhängige Rüstzeiten

Ist die Rüstzeit $st(j)$ auf einen Auftrag $j \in J$ unabhängig vom davor gefertigten Auftrag, so kann in vielen Fällen die Rüstzeit in die Prozesszeit p_j des Auftrags mit aufgenommen werden, die Prozesszeit kann also ersetzt werden durch $p'_j = st(j) + p_j$. Dies ist allerdings nur dann möglich, wenn nicht zusätzliche Restriktionen diesem Vorhaben im Wege stehen. So kann bei Vorliegen von Freigabezeitpunkten oder Materialrestriktionen der Rüstvorgang auf den Auftrag schon dann vorgenommen werden, wenn noch auf den Freigabezeitpunkt

beziehungsweise die Materialverfügbarkeit gewartet wird, und kein weiterer Auftrag ausgeführt wird. In diesen Fall würde die Rüstzeit dann doppelt gezählt werden. Für unser Problem werden wir uns gleich noch einmal in einem anderen Zusammenhang mit reihenfolgeunabhängigen Rüstzeiten auseinandersetzen.

2.9.2 Reihenfolgeabhängige Rüstzeiten

Eine reihenfolgeabhängige Rüstzeit $st(i, j)$ tritt dann auf, wenn sowohl der gleich auszuführende Auftrag als auch sein direkter Vorgänger im Schedule bekannt sein müssen, um den Rüstaufwand zu bestimmen. Das Problem $1|st(i, j)|\sum C_j$ im allgemeinen Fall ist $\mathcal{NP}$-schwer im strengen Sinne. In Kan (1976) wird dazu eine Reduktion vom Problem des gerichteten Hamiltonpfades durchgeführt. Sei dazu $G = (V, E)$ der Graph, in dem ein gerichteter Hamiltonpfad gesucht wird. Für jeden Knoten $i \in V$ erzeuge einen Auftrag mit Prozesszeit $p_i = 1$. Setze dann für jede Kante $(i, j) \in E$ die reihenfolgeabhängige Rüstzeit $st(i, j)$ auf 0. Für jedes Paar von Knoten $(i, j) \notin E$ setze die reihenfolgeabhängige Rüstzeit $st(i, j) = 1$. Als angestrebten Zielfunktionswert für den Schedule wählen wir $y \leq \frac{1}{2} \cdot n \cdot (n + 1)$. Dieser Zielfunktionswert kann nur dann erreicht werden, wenn es einen Schedule gibt, bei dem wir keine Zeit für Rüstvorgänge aufwenden müssen. Dafür ist die Existenz eines gerichteten Hamiltonpfades in G notwendig und hinreichend.

Ein Literaturüberblick, der ausschließlich dem Scheduling und der Losgrößenplanung bei Vorliegen von reihenfolgeabhängigen Rüstzeiten gewidmet ist, findet sich in Zhu und Wilhelm (2006).

2.9.3 Auftragsfamilien

In unserem Problem lassen sich die Aufträge in mehrere Familien (in unserer Terminologie Produkte) aufteilen, und es treten nur bei einem Wechsel zwischen den Familien der Aufträge Rüstzeiten auf. Im Fall von reihenfolgeabhängigen Rüstzeiten zwischen den Familien ist das Problem immer noch $\mathcal{NP}$-schwer (für eine beliebige Anzahl von Familien), da alle Familien nur aus einem Auftrag bestehen könnten, und somit wieder der Komplexitätsbeweis aus Abschnitt 2.9.2 greift. Ist die Reihenfolge innerhalb einer Familie frei wählbar, sollte auf jeden Fall für die Aufträge die Sortierung nach nicht ansteigendem Verhältnis w_j/p_j eingehalten werden (Bruno und Sethi 1977). Wird dagegen die Anzahl der Familien fixiert, werden die in Monma und Potts (1989) und Ghosh (1994) vorgeschlagenen Ansätze der Dynamischen Programmierung zu Verfahren mit polynomieller Laufzeit. Bemerkenswert bei der Literaturdurchsicht ist der Umstand, dass der Komplexitätsstatus bei reihenfolge**un**abhängigen Rüstzeiten und einer beliebigen Anzahl von Familien, $1|s_f|\sum(w_j)C_j$ immer noch nicht geklärt ist. Für dieses Problem erläutern Webster und Baker (1995), unter welchen Umständen der Schedule, der alle Aufträge einer Familie direkt hintereinander fertigt, den optimalen Schedule einer Instanz des Problems $1|s_f|\sum(w_j)C_j$ darstellt. Ein weitere Beobachtung für optimale Schedules wird in Mason und Anderson (1991) angegeben, hier wird das Verhältnis von Rüstzeit zu Prozesszeit je Klasse in einem optimalen Schedule betrachtet. Ein 2-Approximationsalgorithmus für dieses Problem findet sich in Divakaran und Saks (2008), hier wird weiterhin auf den unklaren Komplexitätsstatus verwiesen. Bei zusätzlicher Betrachtung von Fälligkeiten wird ein genetischer Algorithmus in Herrmann und

Lee (1995) vorgestellt.

Bruno und Downey (1978) betrachten das Problem der maximalen Verspätung $1|s_f|L_{\max}$ bei Vorliegen von Auftragsfamilien. Hier ist das Problem bereits für zwei Familien und identische Rüstzeiten $\mathcal{NP}$-schwer. Der Fall von identischen Rüstzeiten ist auch für das in dieser Arbeit behandelte Problem interessant, muss nämlich neben der Gruppe der Produkte keine weitere Materialgruppe für die Rüstzeitberechnung betrachtet werden, erhalten wir auch für unser Problem Auftragsfamilien mit identischen Rüstzeiten. Würde also das Problem $1|s_f|\sum(w_j)C_j$ mit identischen Rüstzeiten als $\mathcal{NP}$-schwer eingeordnet werden können, wäre auch der Komplexitätsstatus für unsere Konstruktion von Rüstzeiten geklärt. Der Artikel von Bruno und Downey (1978) liefert auch einen Beweis, dass die Entscheidung über die Existenz eines zulässigen Schedules für unser Problem ein $\mathcal{NP}$-schweres Problem darstellt, da im Beweis die Frage nach der Existenz einer Lösung von $1|s_f|L_{\max}$ mit einem Zielfunktionswert $L_{\max} \leq 0$ gestellt wird.

Die Literatur zum Scheduling bei Vorliegen von Auftragsfamilien auch für weitere Produktionsumgebungen ist recht umfangreich, wie die Übersichtsartikel Potts und Wassenhove (1992), Liaee und Emmons (1997) und Potts und Kovalyov (2000) belegen.

2.10 Scheduling mit Verspätungen

Müssen die Fälligkeiten δ_j nicht mehr auf jeden Fall eingehalten werden und werden durch sogenannte Duedates d_j ersetzt, so sind, in Abhängigkeit der verwendeten Zielfunktion, meist schon die Grundprobleme ohne zusätzliche Erweiterungen schwer zu lösen. Sei dazu $T_j = \max\{C_j - d_j, 0\}$ die Verspätung des Auftrags $j \in J$ in Abhängigkeit seiner Fertigstellungszeit C_j. Dann ist das Problem der Minimierung der Gesamtverspätung $1||\sum T_j$ $\mathcal{NP}$-schwer (Du und Leung 1990), es lässt sich jedoch mit Hilfe eines pseudopolynomiellen Ansatzes der dynamischen Programmierung in $\mathcal{O}(n^4 \cdot \sum p_j)$ lösen (Lawler 1977).

Wird die Verspätung eines Auftrags jedoch individuell mit einem Gewicht w_j versehen, so ist das Problem $1||\sum w_j T_j$ schon $\mathcal{NP}$-schwer im strengen Sinne (Lenstra, Kan und Brucker 1977). Der Komplexitätsbeweis für diesen Fall ist ähnlich zum Beweis von $1|\delta_j|\sum w_j C_j$ (siehe Abschnitt 2.7). Nur die zusätzlichen Aufträge weisen ein Duedate > 0 auf, die Aufträge aus der 3-Partitionierung erhalten ein Duedate von 0, so dass für diese Aufträge eigentlich die gewichtete Fertigstellungszeit gemessen wird.

Würden wir für unser Problem die Fälligkeiten durch Duedates ersetzen, und eine der regulären Zielfunktionen zur Verspätungsmessung betrachten, so würde jeder mit den Vorrangbeziehungen pro Produkt in Einklang stehende Permutation der Aufträge einen zulässigen Schedule darstellen.

In diesem Kapitel haben wir einen weiteren Einblick in das betrachtete Schedulingproblem gewonnen, und festgestellt, dass in den meisten Fällen allein eine einzige Klasse von Nebenbedingungen im Zusammenspiel mit der Zielfunktion ein $\mathcal{NP}$-schweres Problem ergibt. Neben der im vorherigen Kapitel eher gefühlten Schwierigkeit bei der Suche nach dem optimalem Schedule für unser Problem haben wir jetzt mit den Komplexitätsresultaten aus diesem Kapitel nun ein weit schlagkräftigeres Argument für die Entwicklung von auf dieses Schedulingproblem spezialisierten Algorithmen.

Kapitel 3

Vorranggraphen und Halbordnungen

Die Vorschrift, dass ein Auftrag in jedem zulässigen Schedule vor einem anderen Auftrag eingeplant werden muss, trägt einerseits dazu bei, dass wir auf der Suche nach dem optimalen Schedule weniger weitere Schedules untersuchen müssen. Andererseits führen solche Vorrangbeziehungen dazu, dass das Optimierungsproblem komplexer wird, da einfache Lösungsverfahren eventuell nicht darauf ausgelegt sind, diese Vorrangbeziehungen zu beachten. Dieses Kapitel beschäftigt sich ausführlich mit verschiedenen Eigenschaften von Vorrangbeziehungen, Vorranggraphen und Halbordnungen sowie der Zerlegung der Auftragsmenge anhand der gegebenen Vorrangbeziehungen. Lassen sich die Vorranggraphen einer speziellen Klasse von Halbordnungen zuordnen, so kann es auch im Fall der $\mathcal{NP}$-Vollständigkeit eines Optimierungsproblems für allgemeine Vorranggraphen einen Polynomialzeitalgorithmus geben, der einen Vorranggraphen dieser speziellen Klasse vorraussetzt. Wichtige Aspekte sind hier zum einen das Erkennen der Zugehörigkeit eines Vorranggraphen zu einer bestimmten Klasse und die Reduktion eines Vorranggraphen auf einen dieser Klasse zugehörigen Vorranggraphen. Diese Reduktion besteht in unserem Fall aus der Entfernung von Vorrangbeziehungen aus dem Vorranggraphen, bis der Graph dieser Klasse angehört. Hier ist es wünschenswert, einerseits möglichst viele Vorrangbeziehungen zu erhalten, und andererseits zu wissen, welche Vorrangbeziehungen der Reduktion zum Opfer gefallen sind.

3.1 Vorrangbeziehungen

Soll in jeder betrachteten Lösung gelten, dass ein Auftrag i unbedingt vor einem Auftrag j ausgeführt werden muss, machen wir das durch die Notation $i \prec j$ kenntlich. Um nun kombinatorische Optimierungsprobleme auf Graphen betrachten zu können, die aus den Vorrangbeziehungen zwischen den Aufträgen hervorgehen, sind wir auf die Definition des Vorranggraphen angewiesen:

Definition 3.1 (Vorranggraph)
Sei $G(J, E)$ der gerichtete Graph, dessen Knotenmenge geben ist durch die Aufträge aus J. Die Kantenmenge sei gegeben durch die Vorrangbeziehungen, das

heißt $E := \{(i,j) \in J \times J : i \prec j\}$.

Lemma 3.2
$G(J,E)$ ist ein gerichteter azyklischer Graph.

Beweis. Angenommen es existiert ein Kreis $j_1, \ldots j_k, j_{k+1} = j_1$ in G. Dann bedeutet dieser Kreis, dass die Aufträge $j_2, \ldots, j_k$ sowohl nach j_1 als auch vor j_{k+1} ausgeführt werden müssen. Da nun aber j_{k+1} identisch mit j_1 ist, führt dies zu einer unmöglich lösbaren Aufgabe, eine Instanz mit so einem Vorranggraphen ist nicht zulässig lösbar. □

Die Vorrangbeziehungen können wir auch als Halbordnung $P(J, \prec)$ auf der Auftragsmenge J mit Relation $\prec$ betrachten.

Definition 3.3 (Topologische Sortierung, Lineare Erweiterung)
Jede Reihenfolge $\sigma = (\sigma_1, \ldots, \sigma_{|J|})$ der Aufträge aus J, die alle Vorrangbeziehungen beachtet, wird als topologische Sortierung der Aufträge aus J oder Lineare Erweiterung der Halbordnung $P(J, \prec)$ bezeichnet.

Lemma 3.4
Jede zulässige Lösung des betrachteten Schedulingproblems entspricht einer topologischen Sortierung der Aufträge hinsichtlich des Vorranggraphen.

Beweis. Da die Vorrangbeziehungen allein aus den Anforderungen an zulässige Lösungen hervorgehen, muss jeder zulässige Schedule alle Aufträge aus J enthalten und all diese Vorrangbeziehungen erfüllen. □

Das Vorliegen einer topologischen Sortierung der Aufträge ist jedoch nur eine notwendige, nicht hinreichende Bedingung für einen zulässigen Schedule für das von uns betrachtete Optimierungsproblem, da durch Rüstzeiten und Wartezeiten auf Materialverfügbarkeit dennoch einige Fälligkeiten von Aufträgen verletzt werden können.

3.2 Abschluss und Reduktion

Ist eine Menge von Vorrangbeziehungen und ein Vorranggraph G gegeben, stellt sich häufig die Frage, ob eigentlich nicht noch mehr Vorrangbeziehungen durch diesen Graphen aufgrund von transitiven Abhängigkeiten ausgedrückt werden. Die folgende Definition befasst sich nun mit dieser Fragestellung:

Definition 3.5 (Transitiver Abschluss)
Zu einem Vorranggraphen G bezeichne der Vorranggraph G^C den transitiven Abschluss von G, wenn für jede Kante $(i,j) \in E(G^C)$ ein (i,j)-Pfad in G existiert und umgekehrt.

Andererseits ist es auch häufig von Interesse, nur eine minimale Menge von Vorrangbeziehungen als Kanten in einem Graphen zu speichern, ohne Information über alle vorhandenen Vorrangbeziehungen zu verlieren.

Definition 3.6 (Transitive Reduktion)
Zu einem Vorranggraphen G bezeichne der Vorranggraph G^R die transitive Reduktion von G, wenn jede Kante $(i,j) \in E(G)$ nur dann in G^R enthalten ist, wenn keine weiterer Auftrag k existiert mit $i \prec k \wedge k \prec j$.

Betrachten wir dann alle Pfade in G^R können wir wieder den transitiven Abschluss G^C herstellen, andersherum entspricht auch die transitive Reduktion des transitiven Abschlusses der transitiven Reduktion des Graphen: Das folgende Lemma fasst diese Erkenntnis zusammen:

Lemma 3.7
Es gilt $(G^R)^C = G^C$ *und* $(G^C)^R = G^R$ *für jeden Vorranggraphen* G.

3.2.1 Theoretische Ergebnisse

Laut Aho, Garey und Ullman (1972) ist die Laufzeitkomplexität zur Berechnung des Transitiven Abschlusses identisch zur Laufzeitkomplexität zur Berechnung der transitiven Reduktion. Zudem ist diese identisch zur Komplexität der Matrixmultiplikation, deren naive Implementierung in $\mathcal{O}(n^3)$ abläuft.

3.2.2 Berechnung

Die Berechnung des transitiven Abschluss und der transitiven Reduktion ist eine in den Lehrbüchern zu Graphentheorie und Algorithmen häufig aufgegriffene Problemstellung, siehe zum Beispiel (Skiena 2008) und (Mehlhorn und Sanders 2008, Seite 177, Übung 9.7).

Für unsere Zwecke greifen wir die Übungsaufgabe 3.9.7. von Seite 90ff aus Jungnickel (2008) auf, um mit Hilfe des Algorithmus von Floyd und Warshall (Floyd 1962; Warshall 1962) nicht kürzeste, sondern längste Wege zwischen den Aufträgen zu bestimmen. Damit können wir gleichzeitig sowohl den transitiven Abschluss als auch die transitive Reduktion bestimmen, wie Algorithmus 3.1 zeigt. Für jede bekannte Vorrangbeziehung setzen wir zunächst die Distanz $\pi_{i,j}$ zwischen den Aufträgen i und j auf 1, besteht noch keine Vorrangbeziehung, so setzen wir $\pi_{i,j} = 0$. Durch die Hauptiteration des Floyd-Warshall-Algorithmus, in der immer ein neuer Zwischenauftrag k untersucht wird, werden sowohl immer neue Vorrangbeziehungen generiert und dem transitiven Abschluss hinzugefügt, andererseits aber auch kurze Pfade durch längere Pfade ersetzt, und so Vorrangbeziehungen aus der transitiven Reduktion entfernt.

Aufgrund der vielen Anwendungsmöglichkeiten des transitiven Abschlusses beziehungsweise der transitiven Reduktion gibt es zu diesem Problem und seinen Variationen vielfältige Literatur. So betrachtet zum Beispiel Nuutila (1995) in seiner Dissertation die Berechnung des Abschlusses in großen gerichteten Graphen. Ist nicht die asymptotische Laufzeit, sondern die erwartete Laufzeit von Interesse, lässt sich der Abschluss in linearer Zeit bezüglich der Anzahl der Kanten bestimmen (Schnorr 1978).

3.3 Darstellung von Halbordnungen

Fassen wir die Vorrangbeziehungen als eine Halbordnung $P(J, \prec)$ auf der Menge der Aufträge J verbunden durch die Relation $\prec$ auf, so gibt es hier weitere etablierte Darstellungsformen.

Algorithm 3.1 Algorithmus zur simultanen Berechnung von transitiver Reduktion und transitiven Abschluss.

Input: Vorranggraph G

for $i = 1, \ldots |J|$ **do**
  for $j = 1, \ldots |J|$ **do**
    if $(i, j) \in E(G)$ **then**
      $\pi_{i,j} \leftarrow 1$ ▷ Längster bekannter Pfad hat eine Kante
    else
      $\pi_{i,j} \leftarrow 0$ ▷ Es existiert (noch) kein Pfad
for $k = 1, \ldots |J|$ **do** ▷ Hauptiteration des Floyd-Warshall Algorithmus
  for $i = 1, \ldots |J|$ **do**
    for $j = 1, \ldots |J|$ **do**
      if $i \neq j \wedge j \neq k \wedge i \neq j \wedge \pi_{i,k} > 0 \wedge \pi_{k,j} > 0 \wedge \pi_{i,k} + \pi_{k,j} > \pi_{i,j}$ **then**
        $\pi_{i,j} \leftarrow \pi_{i,k} + \pi_{k,j}$ ▷ Neuer oder längerer (i, j)-Pfad über k
$E(G^C) \leftarrow \emptyset$
$E(G^R) \leftarrow \emptyset$
for $i = 1, \ldots |J|$ **do**
  for $j = 1, \ldots |J|$ **do**
    if $\pi_{i,j} > 0$ **then** ▷ Pfad zwischen i und j
      $E(G^C) \leftarrow E(G^C) \cup (i, j)$
    if $\pi_{i,j} = 1$ **then** ▷ Direkte Verbindung zwischen i und j
      $E(G^R) \leftarrow E(G^R) \cup (i, j)$

Output: G^C als transitiver Abschluss von G, G^R als transitive Reduktion von G

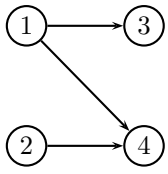


Abbildung 3.1: N-Graph

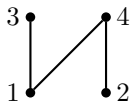


Abbildung 3.2: Hasse-Diagramm des N-Graphen

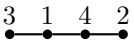


Abbildung 3.3: Vergleichbarkeitsgraph des N-Graphen

3.3.1 Hasse-Diagramm

Die transitive Reduktion G^R eines Vorranggraphen G kann auch durch ein Hasse-Diagramm dargestellt werden. Hierbei handelt es sich um einen ungerichteten Graphen, der für jede direkte Vorrangbeziehung eine Kante enthält. Um die Priorisierung der Knoten untereinander zu verdeutlichen, werden bezüglich der Ordnung $\prec$ kleinere Elemente weiter unten im Graphen angeordnet. Das Hasse-Diagramm des N-Graphen aus Abbildung 3.1 ist in Abbildung 3.2 dargestellt.

3.3.2 Vergleichbarkeitsgraph

Der Vergleichbarkeitsgraph (engl. Comparability-Graph) ist ein ungerichteter Graph, in dem zwei Knoten i und j durch eine Kante verbunden sind, sobald $i \prec j$ oder $j \prec i$ gilt, es wird also für je zwei vergleichbare Knoten eine Verbindung hergestellt. Somit geht der Vergleichbarkeitsgraph aus dem transitiven Abschluss G^C von G hervor. Der Vergleichbarkeitsgraph für den N-Graphen aus Abbildung 3.1 ist in Abbildung 3.3 dargestellt, und entspricht dem Pfad P_4 mit vier Knoten und drei Kanten.

3.4 Der Satz von Dilworth und die Zerlegung in Ketten

In der Theorie der Halbordnungen spielt der Begriff der *Kette* eine wichtige Rolle:

Definition 3.8 (Kette und Antikette)
*Sei J eine Menge mit Halbordnung $\prec$. Eine Teilmenge $S \subset J$ bezeichnen wir als **Kette**, wenn für **alle** Paare $(i,j) \in S \times S$ gilt: $i \prec j \vee j \prec i$. Eine **Antikette** dagegen ist eine Teilmenge $S \subset J$, so dass für **kein** Paar $(i,j) \in S \times S$ gilt: $i \prec j \vee j \prec i$.*

Bezogen auf unser Optimierungsproblem bilden die Aufträge die zum selben Produkt gehören jeweils eine Kette, und die Menge aller Aufträge lässt sich damit in produktbezogene Ketten zerlegen. Da gemäß Definition auch jede einelementige Menge eine Kette bildet, ist eine Zerlegung in Ketten einer mit einer

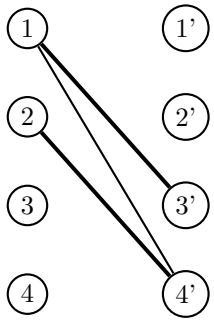


Abbildung 3.4: Transformation zur Berechnung der Partitionierung in Ketten

Halbordnung versehenen Menge immer möglich. Darum ergibt sich auch fast zwangsläufig die Frage nach einer minimalen Zerlegung, die Anzahl der Ketten soll also möglichst klein sein. Diese Frage beantwortet der Satz von Dilworth:

Satz 3.9 (Dilworth (1950))
Die minimale Kardinalität einer Zerlegung in Ketten entspricht der Kardinalität der maximalen Antikette.

Um nun eine Zerlegung in möglichst wenige Ketten zu erhalten, können wir uns auf die Berechnung eines Matchings von maximaler Kardinalität in einem ungerichteten bipartiten Graphen stützen. Dieses Vorgehen wurde bereits von Fulkerson (1956) als konstruktive Anwendung des Satzes von Dilworth beschrieben, und von Ford und Fulkerson (1962, Seite 117ff) als Anwendung von Netzwerkflüssen angeführt. Eine weitere Einordnung des Satzes von Dilworth in die Theorie der Matchings wird in Lovász und Plummer (2009) vorgenommen.

3.4.1 Transformation in ein Matching-Problem

Um die Menge J in Ketten aufzuteilen, erstellen wir einen bipartiten Graphen mit $2 \cdot |J|$ Knoten, da wir für jeden Auftrag $j \in J$ eine Kopie $j' \in J'$ anlegen. Eine Kante in diesem Graphen wird für jede Vorrangbeziehung $i \prec j$ aus dem transitiven Abschluss G^C erstellt, in dem wir Auftrag i mit der Kopie j' von j verbinden. Das Matching mit maximaler Kardnalität kann nun mit dem Algorithmus von Hopcroft und Karp (Hopcroft und Karp 1973) in einer Laufzeit von $\mathcal{O}(|J|^{2.5})$ berechnet werden. Der Algorithmus selbst basiert auf der Berechnung alternierender augmentierender Pfade, in jeder Iteration wird also versucht, das Matching zu vergrößern. Aus im maximalen Matching ausgewählten Kanten können wir nun die Ketten bestimmen. Ist die Kante (i, j') ausgewählt, wissen wir dass i und j Teil der gleichen Kette sind, und i der direkte Vorgänger von j in der Kette ist.

In Abbildung 3.4 wird diese Transformation für den N-Graphen aus Abbildung 3.1 dargestellt. Die zwei sich ergebenden Ketten $C_1 = (1, 3)$ und $C_2 = (2, 4)$ bilden dann die Zerlegung in Ketten $\mathcal{C}$ dieses Vorranggraphen.

3.4.2 Maximale Antikette

Um die maximale Antikette zu bestimmen, also die größte Menge von Aufträgen, die paarweise unvergleichbar sind, können wir auch wieder auf das Mat-

ching von Maximaler Kardinalität zurückgreifen. In Pijls und Pothars (2013) wird jedoch ein anderes Verfahren vorgestellt, das nicht auf dem transitiven Abschluss, sondern auf der transitiven Reduktion basiert. Hierzu wird ein spezielles **Min**-Flow-Netzwerk betrachtet, das in ähnlicher Weise in Möhring (1985) zur Bestimmung einer unabhängigen Menge mit maximalem Gewicht in einem azyklischen gerichteten Graphen verwendet wird.

3.4.3 Anwendungsbeispiel der Zerlegung

In K. G. Murty (2010) wird eine Anwendung der Partitionierung in eine minimale Anzahl von Ketten zur heuristischen Lösung eines Problems im Personennahverkehr vorgestellt. Gegeben sind mehrere Fahrten von Bussen, die Aufgrund ihrer Ausführungszeit partiell geordnet werden können. Nun sollen mit dem Einsatz von möglichst wenigen Fahrzeugen alle Fahrten abgearbeitet werden. Eine aus den Vorrangbeziehungen abgeleitete Kette repräsentiert dann die Zusammenstellung der Fahrten für einen Bus.

3.4.4 Der Algorithmus von Sidney (1975)

Liegen Vorrangbeziehungen in Form von Ketten vor, so kann das Problem $1|chains|\sum w_jC_j$ in Polynomialzeit mit einer Laufzeit von $\mathcal{O}(n^2)$ mit dem Algorithmus für Ketten von Sidney (1975) gelöst werden. Um die Vorrangbeziehungen zu beachten, werden immer die ersten k Aufträge einer Kette $C = (j_1, \ldots, j_{|C|})$ zu einer Teilsequenz zusammengefasst, solange das Verhältnis der Summe $\sum_{k'=1}^{k} w_{j(k')}$ zur Summe $\sum_{k'=1}^{k} p_{j(k')}$ ansteigend ist. Dieser Teilsequenzen werden dann auf Basis dieses Verhältnisses gemäß der Regel von Smith (1956) aneinandergereiht.

Wird der Algorithmus im Laufe eines Branch-and-Bound-Verfahrens wiederholt bei unveränderten Prozesszeiten der Aufträge und von vorne verkürzten Ketten eingesetzt, lässt sich eine Verbesserung der asymptotischen und tatsächlichen Laufzeit dadurch erreichen, dass die Berechnung des besten Summenverhältnisses bei unterschiedlichem Startauftrag der Kette bereits im Vorfeld geschieht und die Teilsequenzen bestimmt werden. Muss dann das Problem in einem Knoten gelöst werden, ist nur noch eine Sortierung dieser Teilsequenzen notwendig. Auch hier schaltet erst die Einplanung einer Teilsequenz der Kette die nächste Teilsequenz derselben Kette frei.

3.5 Anforderungen an polynomiell lösbare Klassen von Halbordnungen

In diesem Abschnitt wollen wir uns dem Problem $1|prec|\sum w_jC_j$ für verallgemeinerte Vorrangbeziehungen widmen. Das Problem ist $\mathcal{NP}$-schwer im Strengen Sinne Lenstra und Kan 1978; Lawler 1978, und stellt ein Teilproblem des in dieser Arbeit betrachteten Optimierungsproblems dar. Optimale Lösungen für $1|prec|\sum w_jC_j$ beziehungsweise für Teilprobleme und Relaxierungen von $1|prec|\sum w_jC_j$ können deshalb als untere Schranken in einem Branch-and-Bound-Ansatz zur Anwendung kommen. Nehmen die Vorrangbeziehungen bestimmte Formen an und können einer bestimmten Klasse von Vorrangbeziehungen zugeordnet werden, so können wir dennoch auf Polynomialzeitalgorithmen

zurückgreifen. Ziel dieses Abschnitts ist es, diese Klassen besonders in Hinblick auf die Anwendung in einem Branch-and-Bound-Verfahren zu untersuchen. Dazu definieren wir zunächst einige Anforderungen an solche Klassen:

Zuordnung zur Klasse Wollen wir einen Polynomialzeitalgorithmus einsetzen, der uns das Optimierungsproblem $1|prec_K|\sum w_jC_j$ für eine Klasse K von Halbordnungen optimal löst, so können in einem ersten Schritt den Algorithmus immer dann einsetzen, wenn die aktuell vorliegende Instanz des Problems $1|prec|\sum w_jC_j$ tatsächlich auch eine Instanz des Problems $1|prec_K|\sum w_jC_j$ darstellt. Die aktuell vorliegenden Vorrangbeziehungen sind also der Klasse K zuzuordnen. Dazu müssen wir aber in der Lage sein, zu erkennen, ob die Vorrangbeziehungen tatsächlich dieser Klasse angehören. Je nachdem, ob wir diese Erkennung nur zu Beginn eines Branch-and-Bound-Verfahrens, oder gar in jedem Knoten des Verfahrens durchführen wollen, können für das Erkennen der Klasse unterschiedliche asymptotische Laufzeiten von Interesse und sinnvoll sein.

Reduktion auf die Klasse Anstelle die polynomielle Lösbarkeit für eine Klasse K nur für bestimmte Instanzen oder bestimmte Knoten im Branch-Bound-Verfahren auszunutzen, können wir auch versuchen, durch die Entfernung von Vorrangbeziehungen einen Vorranggraphen dieser Klasse zu erstellen. Zusätzlich zu der Anforderung, dass wir in der Lage sein müssen, nach Entfernung bestimmter Kanten festzustellen, ob die Vorrangbeziehungen der gewünschten Klasse angehören, ergeben sich hier noch weitere Aspekte, die zu beachten sind. So kann es durchaus zu unterschiedlichen Teilmengen von Vorrangbeziehungen führen, wenn wir entweder am Erhalt möglichst vieler Vorrangbeziehungen interessiert sind oder ob wir die erhaltenen Vorrangbeziehungen nach Qualität der unteren Schranke bestimmen.

Implementierbarkeit des Optimierungsverfahrens Sollte ein Optimierungsproblem der Klasse $\mathcal{P}$ zugeordnet werden können, muss das nicht dazu führen, dass der Lösungsalgorithmus mit polynomieller Laufzeit auch tatsächlich sinnvoll anwendbar ist. Negativbeispiele für die Implementierbarkeit wären entweder ein Beweis für die polynomielle Lösbarkeit, der nur auf der linearen Programmierung beruht, ohne dass ein kombinatorischer Algorithmus bekannt ist, oder ein zwar von der asymptotischen Laufzeit her tatsächlich polynomieller Algorithmus, dessen praktische Laufzeit jedoch nicht vernünftig ist. Vor allem wenn der Polynomialitätsbeweis auf mehreren verschachtelten polynomiellen Reduktionen basiert, kann es so zu in der Praxis nicht sinnvoll umsetzbaren Algorithmen kommen.

Anwendbarkeit im Branch-and-Bound-Verfahren Wollen wir nur zu Beginn eines Branch-and-Bound-Verfahrens die geeignete Halbordnung durch Auswahl von Vorrangbeziehungen festlegen, so müssen die in den entstehenden Instanzen in den einzelnen Knoten immer noch zu dieser Klasse gehören. Zudem ist es sinnvoll, wenn das eingesetzte Polynomialzeitverfahren gut auf sich veränderte Probleminstanzen eingestellt ist.

Ein Beispiel für eine Klasse von Halbordnungen, die all diesen Voraussetzungen entspricht, sind die Vorrangbeziehungen in Form von Ketten. Sie sind einfach zu erkennen, mit dem konstruktiven Beweis des Satzes von Dilworth

(1950) durch Fulkerson (1956) haben wir ein entsprechendes Werkzeug, um aus einem gegebenen Vorranggraphen eine Zerlegung in Ketten zu erzeugen. Der Algorithmus von Sidney (1975) liefert uns dann das gewünschte Polynomialzeitverfahren, das auch im Branch-and-Bound-Verfahren gut einzusetzen ist.

3.6 Zweidimensionale Halbordnungen

In diesem Abschnitt wollen wir die bis jetzt größte bekannte Klasse von Vorrangbeziehungen näher betrachten, für die das Problem $1|prec|\sum w_jC_j$ noch polynomiell lösbar ist, die zweidimensionalen Halbordnungen (Ambühl und Mastrolilli 2008). Zunächst sind wir deshalb auf die Definition der Dimension einer Halbordnung angewiesen:

Definition 3.10 (Dimension (Dushnik und E. W. Miller 1941))
Die Dimension einer Halbordnung ist gegeben durch die minimale Anzahl von linearen Erweiterungen, deren Schnitt die Halbordnung ergibt.

Den Schnitt von linearen Erweiterungen bilden dabei all die Vorrangbeziehungen, die in allen betrachteten linearen Erweiterungen erfüllt sind. Die Bestimmung der Dimension einer Halbordnung ist allgemein ein schwieriges Problem. So ist es zwar linear zur Anzahl der Elemente und der Vorrangbeziehungen möglich, zu überprüfen, ob eine Halbordnung Dimension 1 (eine totale Ordnung) oder 2 (McConnell und Spinrad 1997) hat, die Überprüfung auf eine Dimension drei oder größer ist jedoch bereits $\mathcal{NP}$-vollständig (Yannakakis 1982). Dies gilt auch dann, wenn die längste Kette der Halbordnung die Länge zwei hat (Felsner, Mustata und Pergel 2016).

Die Überprüfung auf Dimension zwei lässt sich dagegen über die transitive Orientierbarkeit des Vergleichbarkeitsgraphen mit Hilfe der Modularen Zerlegung (engl. Modular Decomposition, Substitution Decomposition) sogar in linearer Zeit bezüglich der Knoten und Kanten des Vergleichbarkeitsgraphen bewerkstelligen (McConnell und Spinrad 1997). Somit ist es für diesen Typ von Vorrangbeziehungen in einer guten asymptotischen Laufzeit möglich, die Zugehörigkeit zu einer bestimmten Klasse nachzuvollziehen. Ein Beispiel für eine zweidimensionale Halbordnung bildet der N-Graph aus Abbildung 3.1. Die zwei linearen Erweiterungen sind in diesem Fall die Reihenfolgen $(1, 2, 3, 4)$ und die Reihenfolge $(2, 4, 1, 3)$. Nur die Vorrangbeziehungen $1 \prec 3$, $2 \prec 3$ und $2 \prec 4$ kommen in diesen beiden linearen Erweiterungen der Halbordnung vor. Betrachtet man den zugehörigen Vergleichbarkeitsgraphen, so entspricht dieser einem Pfad mit vier Knoten und drei Kanten, dem P_4. Dieser ist transitiv orientierbar (alle Kanten werden mit einem Pfeil in die gleiche Richtung versehen), und somit ist auch so die Zweidimensionalität gezeigt.

Die polynomielle Lösbarkeit des Optimierungsproblems $1|prec_{2-dim}|\sum w_jC_j$ wird im Aufsatz von Ambühl und Mastrolilli (2008) bewiesen. Die wesentliche Erkenntnis basiert auf der auch schon in anderen Artikeln beobachtete Verwandtschaft des Problems $1|prec|\sum w_jC_j$ mit dem Problem der Knotenüberdeckung von minimalem Gewicht (engl. minimum weighted vertex cover) in einem ungerichteten Graphen. Bevor wir jedoch auf das Problem der Knotenüberdeckung zurückkommen, geben wir hier zunächst ein ganzzahliges Optimierungsmodell aus Potts (1980) für das Schedulingproblem wieder, das als Ausgangs-

punkt für den Polynomialitätsbeweis dient:

$$\min \sum_{j \in J} p_j \cdot w_j + \sum_{(i,j) \in J \times J} p_i \cdot w_j \cdot \pi_{i,j} \tag{3.1}$$

$$\pi_{i,j} + \pi_{j,i} = 1 \forall i < j, i, j \in J \tag{3.2}$$

$$\pi_{i,j} = 1 \forall i \prec j \tag{3.3}$$

$$\pi_{i,j} + \pi_{j,k} + \pi_{k,i} \leq 2 \forall (i,j,k) \in J \times J \times J \tag{3.4}$$

$$\pi_{i,j} \in \{0,1\} \forall (i,j) \in J \times J \tag{3.5}$$

Relaxieren wir die Nebenbedingungen (3.4) teilweise, erhalten wir das Optimierungsmodell, das in Chudak und Hochbaum (1999) zur Ableitung einer 2-Approximation für $1|prec|\sum w_j C_j$ verwendet wurde. Hier wurde die Nebenbedingung (3.4) ersetzt durch

$$\pi_{j,k} + \pi_{k,i} \leq 1 \forall (i,j,k) \in J \times J \times J, i \prec j. \tag{3.6}$$

Anstelle alle möglichen Tripel von Aufträgen zu betrachten, wird in (3.6) das Vorhandensein einer Vorrangbeziehung innerhalb des Tripels eingefordert. Die Leistung des Artikels von Ambühl und Mastrolilli (2008) ist der Beweis, dass jede zulässige Lösung des Modells von Chudak und Hochbaum (1999) **ohne** Veränderung des Zielfunktionswertes in eine zulässige Lösung des Modells (3.1) bis (3.5) überführt werden kann, und das in einer Laufzeit von $\mathcal{O}(n^3)$. Sobald wir das Modell von Chudak und Hochbaum (1999) (CH-IP) optimal gelöst haben, haben wir auch das Problem $1|prec|\sum w_j C_j$ optimal gelöst. Aber auch das CH-IP wollen wir nicht direkt lösen, sondern betrachten eine weitere Relaxation von (3.1) bis (3.5), die im Artikel (Correa und Schulz 2005) betrachtet wird. Dieses Optimierungsmodell sieht wie folgt aus:

$$\min \sum_{j \in J} p_j \cdot w_j + \sum_{i \prec j} p_j \cdot w_j + \sum_{i \| j} p_i \cdot w_j \cdot \pi_{i,j} \tag{3.7}$$

$$\pi_{i,j} + \pi_{j,i} \geq 1 \forall i \parallel j \tag{3.8}$$

$$\pi_{i,k} + \pi_{k,j} \geq 1 \forall i \prec j, j \parallel k, k \parallel i \tag{3.9}$$

$$\pi_{i,l} + \pi_{k,j} \geq 1 \forall i \prec j, j \parallel k, k \prec l, l \parallel i \tag{3.10}$$

$$\pi_{i,j} \in \{0,1\} \forall i < j, i, j \in J, i \parallel j \tag{3.11}$$

Die ersten beiden Terme der Zielfunktion sind konstant, und es wird der Einfluss der gegebenen Vorrangbeziehungen auf die Zielfunktion bereits im Vorfeld berücksichtigt. Für jedes Paar von Aufträgen (i,j), die nicht in einer Vorrangbeziehung zueinander stehen (damit nicht vergleichbar sind), wird eine Variable angelegt. Jede Nebenbedingung gibt ein Paar von Variablen an, von denen mindestens eine auf den Wert eins gesetzt werden muss. Da jede Variable in der Zielfunktion mit einem Gewicht versehen ist, stellt dieses Optimierungsproblem auch die Bestimmung einer Knotenüberdeckung von minimalen Gewicht in einem ungerichteten Graphen dar, wenn wir die Variablen als Knoten und die Nebenbedingungen (3.8) bis (3.10) als Kanten dieses ungerichteten Graphen

auffassen. In Correa und Schulz (2005) wird gezeigt, dass jede zulässige Lösung auch in eine zulässige Lösung für das Modell von Chudak und Hochbaum (CH-IP) umgewandelt werden kann, ohne den Zielfunktionswert zu beeinflussen. Laut (Ambühl und Mastrolilli 2008) ist dies in einer asymptotischen Laufzeit von $\mathcal{O}(n^2)$ möglich. Wir kommen von einer zulässigen Lösung für unser Knotenüberdeckungsproblem (3.7) bis (3.11) in Polynomialzeit zu einer zulässigen Lösung unseres Optimierungsproblem $1|prec|\sum w_j \cdot C_j$. Nun müssen wir nur noch dieses Knotenüberdeckungsproblem näher betrachten. Das Knotenüberdeckungsproblem hat nur dann nur ganzzahlige Basislösungen, wenn der zugrunde liegende Graph bipartit ist (G. L. Nemhauser und L. E. Trotter 1974)[1] Der hier betrachtete Graph ist aber auch nur dann bipartit, wenn die zugrunde liegenden Vorrangbeziehungen zweidimensional sind (Correa und Schulz 2005). In Ambühl, Mastrolilli und Svensson (2006) wird erläutert, dass der hier betrachtete Graph dem Graph der unvergleichbaren Paare aus der Theorie der Halbordnungen entspricht. Dass dieser Graph nur für den Fall zweidimensionaler Halbordnungen bipartit ist wurde unter anderem in Felsner und W. T. Trotter (2000) bewiesen. Wollen wir eine Knotenüberdeckung von minimalem Gewicht in einem bipartiten Graphen bestimmen, so können wir auf die Berechnung eines minimalen Schnittes in einem auf den bipartiten Graphen angepassten Netzwerk zurückgreifen. Dazu führen wir zu den beiden Knotenmengen der Partition A und B noch eine Quelle s und eine Senke t ein. Die ursprünglichen Kanten des ungerichteten Graphen werden zu gerichteten Kanten zwischen den Mengen A und B und mit unendlicher Kapazität versehen, die Quelle s wird mit jedem Knoten $a \in A$ über eine Kante (s, a) mit Kapazität w_a verbunden, und jeder Knoten $b \in B$ wird mit der Senke t über eine Kante $b, t)$ mit Kapazität w_b verbunden. Eine ähnliche Transformation findet auch statt, wenn das Hitchcock Transportproblem mit dem primal-dualen Simplexalgorithmus gelöst wird (Papadimitriou und Steiglitz 1998). Haben wir in diesem Graphen den minimalen s-t-Schnitt $(C, \bar{C})$ bestimmt, so bildet die Knotenmenge $(A \backslash C) \cup (B \cap C)$ die Knotenüberdeckung von minimalem Gewicht[2]. Im allgemeinen Fall ist das Problem der Bestimmung einer Knotenüberdeckung von minimalem Gewicht jedoch $\mathcal{NP}$-schwer (Karp 1972), so dass wir auf diesem Weg nur für zweidimensionale Halbordnungen einen Algorithmus mit polynomieller Laufzeit gefunden haben. Haben wir einen allgemeinen Vorranggraphen gegeben, und wollen die zweidimensionalen Halbordnungen zur Berechnung einer Schranke heranziehen, so müssen wir den Graphen der unvergleichbaren Paare von Aufträgen zu einem bipartiten Graphen machen. Relaxieren wir Vorrangbeziehungen in der Hoffnung, eine zweidimensionale Halbordnung zu bekommen, so werden hier keine Knoten oder Kanten entfernt, sondern es werden weitere Knoten (zwei für jedes weitere nicht vergleichbare Paar von Aufträgen) hinzugefügt. Auch die Kantenmenge wird modifiziert, so entsteht eine weitere Kante aus der Nebenbedingung (3.8), und da die Nebenbedingungen (3.9) und (3.10) sowohl auf vergleichbaren wie auf unvergleichbaren Aufträgen beruhen, werden hier Kanten aus dem Netzwerk hinzugefügt und entfernt. Auf den ersten Blick scheint es deshalb auf diese Wei-

[1] Der Artikel selbst behandelt unabhängige Mengen von maximalem Gewicht, das Komplement einer unabhängigen Menge bildet aber immer eine Knotenüberdeckung aller Kanten. Da das Gesamtgewicht aller Knoten konstant ist, kann nun statt einer minimalen Knotenüberdeckung mit minimalem Gewicht auch eine unabhängige Menge mit maximalem Gewicht gesucht werden.

[2] In Baïou und Barahona 2016 wird diese Konstruktion als „allgemein bekannt“ bezeichnet.

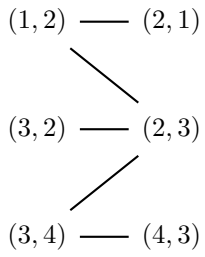


Abbildung 3.5: Knotenüberdeckungsproblem des N-Graphen

se nicht einfach zu sein, eine Reduktion auf eine zweidimensionale Halbordnung zu erstellen.

Die Berechnung einer unteren Schranke bei gegebener zweidimensionaler Halbordnung auch für den Fall, dass die betrachtete Menge von Aufträgen variiert, ist dagegen durch die Anwendung der Berechnung eines maximalen Flusses und der daraus resultierenden Bestimmung eines minimalen Schnittes gut zu implementieren.

Anhand der Betrachtung des N-Graphen aus Abbildung 3.1 wollen wir hier noch einmal den Lösungsansatz verdeutlichen. Die Knoten im Knotenüberdeckungsproblem setzen sich zusammen aus den unvergleichbaren Paaren $(3, 4)$ und $(4, 3)$, $(1, 2)$ und $(2, 1)$ sowie $(3, 2)$ und $(2, 3)$. Der resultierende bipartite Graph, in dem dann ein von den Auftragsprozesszeiten und Gewichten abhängige minimale Knotenüberdeckung zu ermitteln ist, ist in Abbildung 3.5 dargestellt. Die waagerechten Verbindungslinien resultieren dabei aus der Nebenbedingung (3.8), die diagonalen Verbindungen aus der Nebenbedingung (3.9) aus dem Modell von Correa und Schulz (2005). Um eine Knotenüberdeckung zu erhalten, müssen wir dann immer drei Knoten auswählen, wobei die Auswahl von $(3, 2)$ sofort die Auswahl der Knoten $(3, 4)$ und $(1, 2)$ und damit den Schedule $(1, 3, 2, 4)$ festschreibt. Wählen wir dagegen $(2, 3)$ als Teil der minimalen Knotenüberdeckung, so erhalten wir vier weitere Möglichkeiten, um auf alle fünf für den N-Graphen möglichen linearen Erweiterungen zu kommen.

3.7 VSP-Graphen

Ein Spezialfall der zweidimensionalen Halbordnungen bilden die durch VSP-Graphen gegebenen Halbordnungen, die wie folgt auf rekursive Weise definiert werden:

Definition 3.11 (MVSP-Graph (Valdes, Tarjan und Lawler 1982))
Ein gerichteter azyklischer Graph G ist Minimal-Vertex-Series-Parallel (MVSP), wenn er

1. *Aus einem Knoten und keiner Kante besteht.*
2. *Sich aus zwei MVSP-Graphen G_1 und G_2 über die folgenden Operationen konstruieren lässt:*

   Parallele Komposition: *Für die Knotenmenge von G gilt*

   $$V(G) = V(G_1) \cup V(G_2)$$

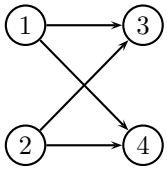


Abbildung 3.6: VSP-Graph

und für die Kantenmenge von G gilt

$$E(G) = E(G_1) \cup E(G_2).$$

Serielle Komposition: *Für die Knotenmenge von G gilt*

$$V(G) = V(G_1) \cup V(G_2)$$

und für die Kantenmenge von G gilt

$$E(G) = E(G_1) \cup E(G_2) \cup T(G_1) \times S(G_2),$$

wobei $T(G)$ die Menge der Senken[3] und $S(G)$ die Menge der Quellen[4] eines gerichteten Graphen G bezeichne.

Definition 3.12 (VSP-Graph (Valdes, Tarjan und Lawler 1982))
Ein azyklischer gerichteter Graph G ist genau dann vertex-series-parallel (VSP), wenn seine transitive Reduktion G^R die MVSP-Eigenschaft besitzt.

Lemma 3.13
Jede durch einen VSP-Graphen induzierte Halbordnung ist zweidimensional.

Beweisskizze. Erstelle eine lineare Erweiterung, die den VSP-Vorrangbeziehungen genügt. Tausche die Inhalte der zueinander parallelen Komponenten in der linearen Erweiterung aus, um eine zweite lineare Erweiterung zu erhalten. □

Die Erkennung von Vorranggraphen dieser Klasse in linearer Laufzeit wurde bereits in der Dissertation von Valdes (1978) behandelt. Zudem ist der Einsatz dieser Problemklasse in Branch-and-Bound-Verfahren mit Vorrangbeziehungen in den Artikeln McMahon und Lim (1993) und Davari u. a. (2015) dokumentiert.

Aufgrund ihrer Definition können VSP-Graphen durch ihre Zerlegung repräsentiert werden, dazu wird ein Baum verwendet, dessen Blätter die einzelnen Aufträge sind, die inneren Knoten stellen entweder eine serielle oder eine parallele Kombination der Kindknoten dar. Der kleinste Graph, der nicht zerlegt werden kann, ist der sogenannte N-Graph aus Abbildung 3.1, der selbst sein transitiver Abschluss ist.

3.7.1 Polynomieller Lösungsalgorithmus

Im Artikel von Lawler (1978) wird gezeigt, dass eine Lösung in einer asymptotischen Laufzeit von $\mathcal{O}(|J| \cdot \log |J|)$ für das Problem $1|prec_{VSP}| \sum w_j C_j$ möglich

[3] Knoten ohne Nachfolger
[4] Knoten ohne Vorgänger

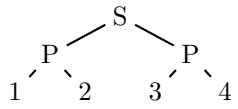


Abbildung 3.7: Zerlegung in serielle und parallele Komponenten

ist. Dazu werden für Serielle und die Parallele Kombination von Auftragsmengen zwei Operationen definiert, die aus diesen Mengen eine neue Auftragsmenge der Kombination erzeugen, die bei Sortierung nach der Regel von Smith (1956) dann automatisch die optimale Reihenfolge beinhaltet und alle gegebenen Vorrangbeziehungen beachtet.

Algorithm 3.2 Lawlers Algorithmus für $1|prec_{VSP}|\sum w_jC_j$

Input: Knoten ν aus dem Zerlegungsbaum
Output: $S(\nu)$ als optimale Sequenz für Knoten ν

```
if ν ist Blatt then
    ▷ Sequenz besteht nur aus dem durch den Knoten repräsentierten Auftrag j
    S(ν) ← (j)
else if ν ist P-Knoten then
    Geordnete Verschmelzung der Sequenzen nach Regel von Smith (1956)
    S(ν) ← S(left(ν)) ∪ S(right(ν))
else if ν ist S-Knoten then
    L ← S(left(ν))                    ▷ Linke Seite der seriellen Kombination
    R ← S(right(ν))                   ▷ Rechte Seite der seriellen Kombination
    if min L > max R then
        S(ν) ← L ∪ R                  ▷ Regel von Smith (1956) liefert zulässige Lösung
    else
        s ← (min L, max R)            ▷ Erzeugung eines kombinierten Auftrags s
        L ← L − min L
        R ← R − max R
        while min L ≤ s ∨ max R ≥ s do
            if min L ≤ s then
                s ← (min L, s)
                L ← L − min L
            if max R ≥ s then
                s ← (s, max R)
                R ← R − max R
        S(x) ← L ∪ R ∪ s
```

Die parallele Kombination ist dabei die einfachere Operation, da beide Auftragsmengen einfach vereinigt werden, zwischen den betrachteten Mengen bestehen ja auch keine Vorrangbeziehungen und die Sortierung nach der Regel von Smith (1956) führt zur optimalen Sequenz. In der Seriellen Kombination werden nun eventuell Aufträge zu einem zusammengesetzten Auftrag s verschmolzen, wie die Wiedergabe des Pseudocodes aus Lawler (1992) in Algorithmus 3.2 verdeutlicht[5]. Durch diese Verschmelzung zu einem zusammengesetzten Auftrag

[5]Die Minima und Maxima einer Menge im Algorithmus sind gegeben durch die Aufträge

wird sichergestellt, dass der Widerspruch zwischen der gegebenen Vorrangbeziehung und der durch die Regel von Smith (1956) vorgegebenen Sortierung aufgelöst wird. Bei Anwendung des Algorithmus ist zu beachten, dass bereits die Kindknoten des betrachteten Knoten durch Algorithmus 3.2 evaluiert worden sein müssen, der Zerlegungsbaum muss also in post-order durchlaufen werden. Werden die Knoten des Zerlegungsbaums in einer einfach verketteten Liste gespeichert, sollte der Vaterknoten deshalb immer nach seinen Kindknoten in die Liste eingefügt werden, damit dann diese Liste zur Bestimmung der optimalen Sequenz einfach in Vorwärtsrichtung durchlaufen werden kann.

Für die Korrektheit des Verfahrens wurden im Laufe der Zeit noch weitere Beweise entwickelt. Ein eher graphisch orientierter Ansatz findet sich in Goemans und Williamson (2000), ein primal dualer Beweis wird im unveröffentlichen Aufsatz von Paes Leme und Shmoys (o.D.) vorgestellt.

Zur Implementation des Verfahrens in einer asymptotischen Laufzeit von $\mathcal{O}(n \cdot \log n)$ mit $n = |J|$ wird eine Datenstruktur benötigt, die ein schnelles Auffinden des minimalen und des maximalen Elementes in einer Menge sowie das Verschmelzen von zwei Mengen unterstützt. Eine Möglichkeit, all diese Operationen in $\mathcal{O}(\log n)$ zu implementieren, besteht in der Verwendung der Binomial-Heaps aus Vuillemin (1978). Da unter anderem auch Vorrangbeziehungen in Form von Ketten eine VSP-Halbordnung darstellen, hat dieser Algorithmus einen asymptotischen Vorteil gegenüber dem Algorithmus von Sidney (1975) aus Abschnitt 3.4.4.

Vorranggraphen mit der VSP-Eigenschaft erlauben nicht nur für die Minimierung der gewichteten Fertigstellungszeit auf einer Maschine, sondern auch für andere Probleme eine Lösung in Polynomialzeit. So wird für das zweistufige Flowshop-Scheduling Problem zur Minimierung des Makespans, das in der nicht weiter beschränkten Form durch den Algorithmus von S. M. Johnson (1954) lösbar ist, in den Artikeln (Monma 1979), (Sidney 1979) und (Monma und Sidney 1979) auch für den Fall eines VSP-Vorranggraphen ein $\mathcal{O}(n \log n)$-Lösungsverfahren vorgestellt. In Anlehnung an die in Monma und Sidney (1979) entwickelte Theorie werden in Gordon u. a. (2008) Schedulingprobleme betrachtet, in der die Bearbeitungszeit eines Auftrags entweder mit der Position eines Auftrags oder abhängig von der Startzeit eines Auftrags im Schedule ansteigt. Auch hier ist dann eine Lösung in Polynomialzeit möglich, wenn der Vorranggraph die VSP-Eigenschaft aufweist. Der Algorithmus von Lawler (1978) lässt sich auch auf N-erweiterbare Halbordnungen ausbauen (Peter und Wambach 2000), ohne die Laufzeitkomplexität von $\mathcal{O}(n \log n)$ zu gefährden. Diese Klasse von Halbordnungen erfüllt zwar nicht die VSP-Eigenschaft, die in der modularen Zerlegung eines solchen Vorranggraphen vorkommenden Primelemente sind dennoch transitiv orientierbar und damit immer noch ein Spezialfall der zweidimensionalen Halbordnungen.

3.7.2 Erkennung von VSP-Graphen

Satz 3.14 (Valdes (1978))
Ein Vorranggraph G erfüllt genau dann die VSP-Eigenschaft, wenn keine aus vier Elementen bestehende Knotenmenge existiert, die im transitiven Abschluss G^C einen N-Graphen als Untergraphen induziert.

mit niedrigster beziehungsweise höchster Priorität w_j/p_j.

Dieser Satz legt das Vorhandensein eines einfachen Algorithmus zur Erkennung eines VSP-Graphen nahe, indem wir in $\mathcal{O}(n^4)$ alle Mengen von vier Elementen auf den N-Graphen testen.

Das von Valdes (1978) entwickelte Verfahren, das im Artikel Valdes, Tarjan und Lawler (1982) beschrieben ist, überprüft für einen beliebigen Graphen, ob dieser VSP ist und liefert im positiven Fall gleich einen Zerlegungsbaum des Graphen mit. Das Verfahren von Valdes läuft im Gegensatz zur oben genannten naiven Implementation in linearer Zeit $\mathcal{O}(n+m)$ für einen beliebigen Vorranggraphen G und nutzt in den vier Schritten jeweils spezifische Eigenschaften von VSP-Graphen aus.

Schritt 1: Bestimmung einer Pseudoreduktion Für den gegebenen Vorranggraphen G wird ein Kandidat $G^{R'}$ für dessen transitive Reduktion G^R in linearer Zeit bezüglich der Knoten und Kanten von G bestimmt. Da die eigentliche transitive Reduktion G^R nicht in linearer Zeit zu bestimmen ist (siehe hierzu auch Abschnitt 3.2.1), kann diese Reduktion inkorrekt sein, also $G^R \neq G^{R'}$ gelten, außer es handelt sich um einen VSP-Graphen. In diesem Fall ist dann die Korrektheit von $G^{R'}$ und damit auch $G^R = G^{R'}$ garantiert.

Schritt 2: Zerlegung in Bicliquen Die transitive Reduktion wird auf vollständige bipartite Untergraphen, sogenannte Bicliquen untersucht. Jede dieser Bicliquen bildet dann einen Knoten in einem weiteren Graphen $L^{-1}(G^{R'})$, die ursprünglichen Knoten in $G^{R'}$ werden dann zu Kanten, die die den Bicliquen zugeordneten Knoten verbinden. Es kann durchaus vorkommen, dass in $L^{-1}(G^{R'})$ mehrere Kanten mit gleichem Start- und Endknoten vorliegen, es handelt sich hierbei um einen sogenannten Multidigraphen. Kann nicht jede Vorrangbeziehung beziehungsweise Kante in $G^{R'}$ einer Biclique zugeordnet werden, so erfüllen $G^{R'}$ und G nicht die VSP-Eigenschaft. Der Schritt überprüft somit die Reduktion $G^{R'}$ auf direkt vorliegende N-Graphen, die nicht erst durch den Abschluss induziert sind.

Schritt 3: Reduktion von $L^{-1}(G^{R'})$ Nun wird der Versuch unternommen, den Graphen $L^{-1}(G^{R'})$ schrittweise durch parallele und serielle Reduktionen auf eine einzelne Kante zu reduzieren. Sobald mehr als eine Kante zwei identische Knoten verknüpft, sind diese durch eine einzelne zu ersetzen, diese Operation wird parallele Reduktion genannt, zwischen den durch die Kante repräsentierten Auftragsmengen liegt keine Vorrangbeziehung vor. Eine serielle Reduktion ist dann möglich, wenn es einen Knoten gibt, der sowohl nur eine eingehende Kante als auch nur eine ausgehende Kante hat. Die beiden mit den Kanten verknüpften Aufträge beziehungsweise Auftragsmengen sind dann hintereinander auszuführen, die beiden Kanten werden durch eine Kante ersetzt, die den Startknoten der eingehenden Kante mit dem Zielknoten der ausgehenden Kante verknüpft. Gelingt die Reduktion von $L^{-1}(G^{R'})$ mittels dieser Operationen auf eine einzelne Kante, so war der Ausgangsgraph $L^{-1}(G^{R'})$ Edge-Series-Parallel (ESP). Bleibt ein größerer Graph nach den Operationen zurück, so erfüllen $G^{R'}$ und G nicht die VSP-Eigenschaft, weil ein durch den Abschluss induzierter N-Graph vorliegt. Während der Durchführung dieser Reduktionen wird gleichzeitig auch der Zerlegungsbaum in serielle und parallele Komponenten von $G^{R'}$ erstellt.

Schritt 4: Verifizierung der Pseudoreduktion Mit Hilfe des Zerlegungsbaums aus Schritt 3 lässt sich die Pseudoreduktion $G^{R'}$ mit linearem Aufwand als tatsächliche transitive Reduktion G^R verifizieren. Dieser Schritt wird notwendig, da für einen Graph ohne die VSP-Eigenschaft die verwendete transitive Reduktion inkorrekt sein kann. Lässt sich die Identität von $G^{R'}$ mit G^R nicht nachweisen, erfüllt der Graph G trotz der Erfolge in den Schritten 2 und 3 nicht die VSP-Eigenschaft.

Haben wir bereits die korrekte transitive Reduktion G^R des Vorranggraphen G vorliegen, werden die Schritte 1 und 4 überflüssig, und nur die Schritte 2 und 3 sind dann für einen Test auf die VSP-Eigenschaft zu implementieren.

Im folgenden wollen wir uns anhand verschiedener Beispiele die Funktionsweise des Algorithmus veranschaulichen, und feststellen, wann die Schritte 2 und 3 des Verfahrens von Valdes, Tarjan und Lawler die Verletzung der VSP-Eigenschaft feststellen. Abbildung 3.8 zeigt den Multidigraphen, der aus Analyse der bipartiten Komponenten des Graphen aus Abbildung 3.6 hervorgeht. In diesem Beispiel gehören alle Kanten des Ursprungsgraphen der bipartiten Komponente B_1 an, zusätzlich werden immer zwei Komponente B_α und B_ω angelegt, die die Terminalknoten des Multidigraphen bilden, und für Aufträge ohne Vorgänger beziehungsweise Nachfolger den Beginn oder Endknoten der Kante im Multidigraphen bilden. Betrachten wir dagegen den N-Graphen aus Abbildung 3.1, so fehlt eine Kante, um einen vollständigen bipartiten Graphen beziehungsweise eine Biclique zu erzeugen. In diesem Fall würden wir also schon in Schritt 2 eine Verletzung der VSP-Eigenschaft feststellen, und nicht mehr zu Schritt 3 weitergehen. Als weiteres Beispiel wollen wir den sogenannten Z-Graphen aus Abbildung 3.9 untersuchen. Hier haben wir auch bereits die transitive Reduktion eines Vorranggraphen vorliegen. Wählen wir die Knoten 1,2,3 und 4, so stellen wir fest, dass diese im transitiven Abschluss einen N-Graphen induzieren, ein direkter N-Graph wie im Fall vorher ist jedoch nicht zu sehen. Erzeugen wir durch die Analyse der durch die Kanten des Z-Graphen induzierten bipartiten Komponenten einen Multidigraphen, erhalten wir den Graphen aus Abbildung 3.10. Während sich der Graph aus Abbildung 3.8 durch Anwendung von parallelen und seriellen Reduktionsoperationen auf eine einzelne Kante reduzieren lässt, stellen wir im Fall des Graphen aus Abbildung 3.10 fest, dass hier weder eine serielle noch eine parallele Reduktion möglich ist. Dadurch stellen wir im Fall des Z-Graphen erst in Schritt 3 des Verfahrens von Valdes, Tarjan und Lawler (1982) fest, dass die VSP-Eigenschaft nicht erfüllt ist. Der Graph aus Abbildung 3.10 stellt somit auch den kleinsten Multidigraphen dar, der nicht auf eine einzelne Kante reduzierbar ist. Da in der Messtechnik häufig eine Schaltung eingesetzt wird, die für die auf den Kanten 1, 2, 4 und 5 ohmsche Widerstände und auf der Kante 3 ein Spannungsmessgerät platziert, um bei drei bekannten Widerstandswerten den Wert des vierten Widerstands zu ermitteln, und diese Schaltung nach Sir Charles Wheatstone benannt ist, wird auch dieser Graph Wheatstone-Brücke genannt.

Die Bezeichnung $L^{-1}(G^{R'})$ für den in Schritt 2 abgeleiteten und in Schritt 3 reduzierten Multidigraphen lässt sich graphentheoretisch begründen, wenn wir die folgende Definition (Bang-Jensen und Gutin 2009) betrachten:

Definition 3.15 (Line-Digraph)
Für einen gerichteten Graphen G heißt der Graph, der für jede Kante aus G einen Knoten erzeugt, und diese neuen Knoten mit einer Kante verbindet, sobald

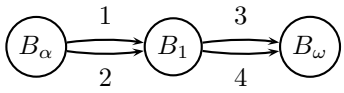


Abbildung 3.8: ESP-Graph aus der Anwendung des Algorithmus von Valdes, Tarjan und Lawler (1982) auf das Beispiel aus Abbildung 3.6

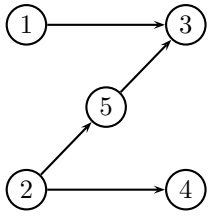


Abbildung 3.9: Z-Graph

in G das Ende einer Kante mit dem Anfang einer Kante übereinstimmt, der Line-Digraph $L(G)$ *zu G.*

Bei der Untersuchung der Bicliques in Schritt 2 des Verfahrens von Valdes, Tarjan und Lawler (1982) erzeugen wir das Inverse $L^{-1}(G^{R'})$ des Line-Digraphen $G^{R'}$, also einen Graphen, dessen Line-Digraph wieder $G^{R'}$ ist, es gilt also

$$L(L^{-1}(G^{R'})) = G^{R'}. \tag{3.12}$$

Dieses Inverse muss nicht eindeutig sein, damit aber ein Inverses existiert, müssen die Kanten jeweils eine Biclique induzieren (Harary und Norman 1960, siehe auch Abschnitt 3.7.3.3).

3.7.3 Maximale VSP-Untergraphen

In diesem Abschnitt wollen wir Verfahren betrachten und entwickeln, die uns für einen beliebigen Vorranggraphen G einen VSP-Untergraphen G_{VSP} erzeugen, für den jede Vorrangbeziehung auch im Abschluss G^C von G enthalten ist. Diese Problemstellung lässt sich auch mit der folgenden Übungsaufgabe (3-24) aus Parker (1995) umschreiben:

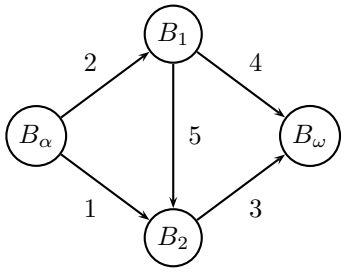


Abbildung 3.10: Wheatstone-Brücke

> „Jeder gerichtete Graph kann zu einem VSP-Graphen gemacht werden, indem Kanten weggelassen werden. Nun soll eine minimale Anzahl an Kanten entfernt werden. Geben Sie entweder einen schnellen Algorithmus für dieses Problem an, oder zeigen Sie, dass das Problem $\mathcal{NP}$-schwer ist!“

Beim Blick auf die aktuelle Literatur lässt sich feststellen, dass diese Übungsaufgabe bis zum heutigen Tag nicht vollständig gelöst ist. Auch in dieser Arbeit werden wir uns einer Lösung des Problems nur annähern, dabei aber weitere Methoden entwickeln, um zu einem VSP-Graphen zu gelangen.

3.7.3.1 Das Verfahren von McMahon und Lim (1993)

In McMahon und Lim (1993) wird ein heuristisches Verfahren für die Erstellung eines VSP-Untergraphen aus der gegebenen transitiven Reduktion G^R eines Vorranggraphen G vorgestellt: Hier wird die Reduktion um eine Quelle s und eine Senke t erweitert. Die Quelle s wird mit allen Aufträgen verbunden, die keinen Vorgänger in der transitiven Reduktion besitzen, die Senke dagegen mit allen Aufträgen, die keinen Nachfolger besitzen. Eine Tiefensuche beginnt im Knoten s. Die Knotenmarkierungen geben entweder den durch den Knoten repräsentierten Auftrag oder die VSP-Zerlegung des durch den Knoten repräsentierten Teilgraphen an. Am Ende des Verfahrens enthält der Knoten t die gesamte Zerlegung als Markierung. Die Kantenmarkierungen listen alle Verzweigungspunkte im Graphen auf dem durchlaufenen Weg zu dieser Kante auf, dabei ist der zuletzt durchlaufene Verzweigungspunkt der entscheidende. Ein Knoten selbst wird erst zum Ausgangspunkt der Tiefensuche, wenn alle eingehenden Kanten markiert worden sind. Führt die Markierung einer eingehenden Kante eines Knotens zur vollständigen Markierung aller eingehenden Kanten, werden parallele und serielle Reduktion durchgeführt, beziehungsweise eingehende Kanten verworfen, bis der Knoten nur noch maximal eine eingehende Kante hat. Hat ein Knoten v einen ausgehenden Knotengrad > 1, wird v als Verzweigungspunkt in die Markierung aller ausgehenden Kanten aufgenommen. Hat ein Knoten nur einen direkten Nachfolger, übernimmt die Kante die Markierung der eventuell letzten verbliebenen eingehenden Kante des Knotens. Laut Autoren hat dieses Verfahren eine Laufzeit von $\mathcal{O}(n + m)$, wobei hier von der Anzahl der Knoten und Kanten in der transitiven Reduktion des Vorranggraphen gesprochen wird.

Zu beachten ist, dass der Anspruch der Autoren, auch bei Vorliegen eines VSP-Graphen diesen als solchen zu erkennen und entsprechend zu zerlegen, nicht erfüllt ist. So wird der Graph aus Abbildung 3.6 mit der verwendeten Markierungsmethode nicht als VSP-Graph erkannt. Da in Abbildung 3.11 im Knoten 3 alle eingehenden Kanten markiert sind, wird an dieser Stelle des Algorithmus versucht, eine parallele Struktur anhand gleicher eingehender Markierungen an 3 zu erkennen. Dies schlägt fehl, und es wird davon ausgegangen, dass ein verbotener N-Graph vorliegt. Dies führt zu einer Entfernung einer Kante, wo keine Entfernung notwendig ist und die Parallelität der Knoten 1 und 2 wird unterschlagen. Zulässige Sequenzen für den Originalgraphen sind $(1, 2, 3, 4)$, $(2, 1, 3, 4)$, $(1, 2, 4, 3)$ und $(2, 1, 4, 3)$. Wird der Algorithmus von McMahon und Lim (1993) angewandt, werden zwei Kanten entfernt, und je nach Wahl der Kanten entstehen entweder zwei Ketten zu je zwei Elementen (bei Entfernung jeweils einer ausgehenden Kante von 1 und 2), oder es wird eine

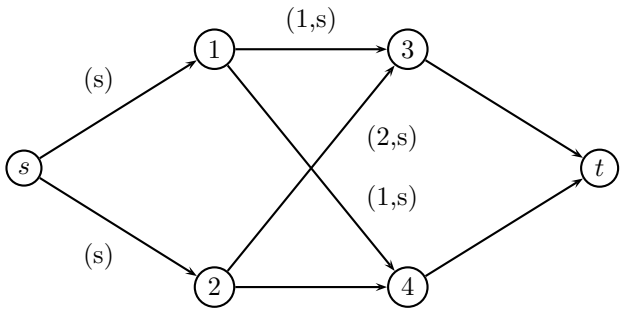


Abbildung 3.11: Markierung der Kanten nach McMahon und Lim (1993) für den Graphen aus Abbildung 3.6

Parallelität zwischen 3 und 4 erkannt und seriell mit 1 oder 2 verknüpft, und dann insgesamt mit 2 oder 1 parallel kombiniert. In beiden Fällen ist die Anzahl der noch zulässigen Schedules dann größer als im Originalproblem, wir erhalten eventuell also auch für Graphen, die die VSP-Eigenschaft erfüllen, nur eine gültige Relaxation der gegebenen Vorrangbeziehungen.

Betrachten wir im Gegensatz dazu das Verfahren von Valdes, Tarjan und Lawler (1982) zum Test auf die VSP-Eigenschaft eines Vorranggraphen, so scheint der Fehler bezüglich der korrekten Erfassung von VSP-Graphen im Verfahren von McMahon und Lim (1993) in der unzureichenden Betrachtung der in der transitiven Reduktion vorliegenden Bicliquen zu liegen.

3.7.3.2 Adaption des Erkennungsverfahrens

Für die Entwicklung weiterer Methoden wollen wir uns deshalb an den Schritten des Verfahrens von Valdes, Tarjan und Lawler (1982) orientieren. Da wir aber nicht nur an der Erkennung eines VSP-Graphen (und dem Negativtestat in Form eines N-Graphen) interessiert sind, müssen wir nun die Schritte aus dem Verfahren von Valdes, Tarjan und Lawler (1982) etwas genauer betrachten. Weil nun grundsätzlich die echte transitive Reduktion G^R des Graphen zum Einsatz kommen soll, sind für uns ausschließlich die Schritte 2, die Erstellung von $L^{-1}(G^R)$ und Schritt 3, der Reduktion von $L^{-1}(G^R)$ des Verfahrens interessant.

3.7.3.3 Biclique Tests von Valdes

Damit ein Graph G ein Line-Digraph ist, und damit auch ein entsprechendes Inverses besitzen kann, müssen alle Kanten in G auf bestimmte Weise Teil von vollständigen bipartiten (Bicliquen) sein. In Abschnitt 3.4 haben wir bereits einmal den transitiven Abschluss G^C eines Vorranggraphen in einen ungerichteten bipartiten Graphen überführt, indem alle Startpunkte von Vorrangbeziehungen zu Knoten auf der linken Seite des bipartiten Graphen, und alle Zielpunkte von Vorrangbeziehungen zu Knoten auf der rechten Seite dieses bipartiten Graphen gemacht haben. Die gleiche Konstruktion wenden wir nun auf die transitive Reduktion G^R an, und erzeugen den bipartiten Graphen $B(G^R)$ mit einer linken Seite L und einer rechten Seite R. Dabei bildet jede Teilmenge $H \subseteq L$ von Knoten auf der linken Seite L einen potentiellen sogenannten Head einer Biclique,

jede Teilmenge von Knoten $T \subseteq R$ der rechten Seite R bildet einen potentiellen Tail. Eine Kante (u, v) induziert einen Head $H_{u,v} \subseteq L$ durch die Betrachtung aller Knoten $u' \in L$ mit der Eigenschaft, dass die Kante (u', v) eine Kante in $B(G^R)$ darstellt. Eine Kante (u, v) induziert dann auf gleiche Weise einen Tail $T_{u,v} \subseteq R$ durch die Betrachtung aller Knoten $v' \in R$, mit der Eigenschaft, dass die Kante (u, v') Teil von $B(G^R)$ ist. Damit eine Kante (u, v) dann auch eine Biclique $B_{u,v}$ induziert, muss nun jeder Knoten aus $H_{u,v}$ auch tatsächlich mit jedem Knoten aus $T_{u,v}$ verbunden sein, und es darf keine weiteren Kanten geben, die die Knoten des Heads $H_{u,v}$ mit weiteren Knoten aus R und die Knoten des Tails $T_{u,v}$ mit weiteren Knoten aus L verbinden. Der Algorithmus von Valdes, Tarjan und Lawler überprüft nun jede Kante (u, v) darauf, ob sie tatsächlich eine Biclique induziert. Diese Bicliques bilden dann die Knoten in dem in Schritt 3 betrachteten Multidigraphen.

Die drei Tests aus der Dissertation (Valdes 1978, Seite 134ff) auf die Existenz eines N-Graphen bei Betrachtung der durch die Kante (u, v) induzierten Head- und Tailmengen sehen wie folgt aus:

Test 1 Prüfe, ob alle direkten Nachfolger der Headmitglieder im Tail $T_{u,v}$ enthalten sind. Wenn das nicht der Fall ist, bildet die induzierende Kante (u, v) mit der zur Konstruktion des Biclique-Kandidaten verwendeten Kante (u', v) und der aus dem Head $H_{u,v}$ des Kandidaten herausführenden Kante (u', w) einen N-Graphen.

Test 2 Prüfe, ob alle direkten Vorgänger der Tailmitglieder im Head $H_{u,v}$ enthalten sind. Wenn das nicht der Fall ist, bildet die induzierende Kante (u, v) mit der zur Konstruktion des Biclique-Kandidaten verwendeten Kante (u, v') und der aus dem Tail $T_{u,v}$ des Kandidaten herausführenden Kante (w, v') einen N-Graphen.

Test 3 Prüfe, ob die Anzahl der Kanten in dem Biclique-Kandidaten vollständig ist, also $|E(B_{u,v})| = |H_{u,v}| \cdot |T_{u,v}|$ gilt. In der Implementation kann die Anzahl der tatsächlich vorhandenen Kanten in Test 1 oder Test 2 gezählt werden. Auch hier ist die induzierende Kante (u, v) Teil eines N-Graphen, seien dazu $u' \in H$ und $v' \in T$ diejenigen Knoten aus Head $H_{u,v}$ und Tail $T_{u,v}$, die nicht miteinander verbunden sind. Dann bilden (u, v) und die in der Induktion verwendeten Kanten (u, v') und (u', v) einen N-Graphen.

Ein Beispiel dafür, dass alle drei Tests unabhängig voneinander nötig sind, bildet der N-Graph aus Abbildung 3.1 selbst. Je nachdem, welche der drei Kanten zur Ableitung der Menge für Head und Tail der potentiellen Biclique ausgewählt werden, greift entweder Test 1 oder Test 2 oder Test 3, aber nie mehr als einer dieser Tests.

Auf Basis dieser Tests können wir nun eine einfache Heuristik zur Konstruktion eines Teilgraphen $G^{R'}$ von G^R entwickeln, so dass die Kanten von $G^{R'}$ alle eine Biclique in $B(G^{R'})$ induzieren. Wir betrachten dazu die Kanten in G^R in beliebiger Reihenfolge. Jedes mal, wenn die aktuelle Kante (u, v) und die von ihr induzierten Mengen $H_{u,v}$ und $T_{u,v}$ die Biclique-Tests bestehen, haben wir eine Biclique gefunden, die dann in Schritt 3 des Verfahrens von Valdes, Tarjan und Lawler einen Knoten des Multidigraphen bildet. Alle Kanten, die zu

dieser Biclique gehören, können dann aus der Menge der Kanten von G^R entfernt werden, zusätzlich entfernen wir nun auch alle Kanten, die eventuell noch mit Teilen des Heads oder des Tails dieser Komponente verbunden sind. Wenn einer der Tests jedoch fehlschlägt, entfernen wir diese Kante, und fahren mit der nächsten Kante fort. Das Ergebnis dieses Vorgehens ist offensichtlich sehr stark von der Reihenfolge der Kantenbetrachtungen abhängig, und die Menge der zu entfernenden Kanten kann stark variieren. Bei diesem Vorgehen steigt auch der asymptotische Aufwand auf $\mathcal{O}(|J| \cdot |E(G^R)|)$ im Gegensatz zur linearen Laufzeit $\mathcal{O}(|J| + |E(G^R)|)$ im Verfahren von Valdes, Tarjan und Lawler, da nun die Knoten mehrfach Teil eines Bicliquekandidaten sein können, und nicht mehr bei der ersten Verfehlung abgebrochen wird.

3.7.3.4 Biclique-Packing

Um möglichst wenige Kanten bei der Erzeugung der Bicliques aus $B(G^R)$ zu verlieren, können wir diese Problematik auch als eigenständiges Optimierungsproblem betrachten. Gesucht wird nun eine maximale Auswahl von Kanten, so dass jeder Auftrag als Knoten $u \in L$ maximal einer Biclique als Teil des Heads und als Knoten $v \in R$ maximal einer Biclique als Teil des Tails zugeordnet ist.

Für die Bestimmung von Bicliques hinsichtlich verschiedener Zielfunktionen und auch das Packing bzw. die Partitionierung der Knotenmenge eines Graphen in eine Menge von Bicliques finden sich in der Literatur Komplexitätsresultate, Algorithmen und Anwendungen, die im folgenden kurz dargestellt werden sollen.

Nachteilig für unsere Anwendung ist das Komplexitätsresultat aus Peeters (2003), dass die Bestimmung einer Biclique mit einer maximalen Anzahl von Kanten in einem bipartiten Graphen als $\mathcal{NP}$-schweres Problem klassifiziert. Dagegen ist das Auffinden einer Biclique mit einer maximalen Anzahl an Knoten im Gegensatz dazu in bipartiten Graphen in Polynomialzeit lösbar. Zu Berechnen ist in diesem Fall eine maximale unabhängige Menge im Komplement[6] des bipartiten Graphen. Dieses Problem kann über den Satz von Kőnig in die Bestimmung eines Matchings von maximaler Kardinalität in einem bipartiten Graphen überführt werden. Für uns ist dieser polynomiell lösbare Fall allerdings weniger interessant, da die Anzahl erhaltener Vorrangbeziehungen bei großem Kardinalitätsunterschied von Head und Tail der Biclique doch eher gering ausfällt. Die Anwendung maximaler Bicliques spielt auch eine wichtige Rolle in der Bioinformatik, als Beispiel soll hier der Artikel Zhang u. a. (2014) zur Auswertung von Gensequenzen dienen.

Geht es nun darum, mit mehreren Bicliques im bipartiten Graphen bestimmte Ziele zu erreichen, wurden in der Literatur bereits die folgenden Probleme untersucht: Soll jeder Knoten des bipartiten Graphen Teil genau einer Biclique sein, und die Anzahl der Bicliques minimiert werden, entsteht das Biclique Partition Problem. Die $\mathcal{NP}$-Vollständigkeit dieses Problems wird in Heydary u. a. (2007) nachgewiesen. Die Existenz von Approximationsalgorithmen für das Problem ist Thema in D. Bein u. a. (2008). Der Artikel von Fleischner u. a. (2009) beschäftigt sich mit Komplexitätsresultaten bei Fixierung von Parametern des Problems. Hier werden auch Probleme untersucht, in denen (im Gegensatz zu den Knoten) jede Kante genau einmal in eine Biclique aufzunehmen ist.

[6] Im Komplement eines bipartiten Graphen tauschen vorhandene und nicht vorhandene Kanten zwischen den beiden Knotenmengen ihre Rolle.

In (Heydary u. a. 2007) wird auch eine Konstruktionsheuristik zur Erstellung einer Partitionierung in Bicliques vorgestellt, die sich auf unsere Problemstellung adaptieren lässt. Die Grundidee besteht darin, zunächst zwei Knoten aus einer Seite (zum Beispiel L) zu wählen, die möglichst viele Knoten auf der anderen Seite (also im Beispiel R) als gemeinsame Partner haben. Die beiden Knoten bilden als Head dann mit diesen gemeinsamen Partnern (dem Tail) eine Biclique. Nun versuchen wir in den weiteren Schritten, den Head zu erweitern, und müssen dafür eventuell die Verkleinerung des Tails in Kauf nehmen. Solange die Vergrößerung des Heads bei Verkleinerung des Tails dennoch zu einem Wachstum der verwendeten Anzahl an Kanten führt, wird die Vergrößerung des Heads durchgeführt, ansonsten wird die erhaltene Biclique fixiert, und ein neues Knotenpaar als weiterer Keim für eine Biclique gesucht, bis alle Knoten in Bicliques untergebracht sind.

Verschiedene Problemstellungen mit Bicliques werden im Artikel Acuña u. a. (2014) betrachtet, eine davon kommt dabei unserem Problem sehr nahe. Die Autoren suchen nach einem Biclique vertex packing in einem bipartiten Graphen, so dass die Anzahl der verwendeten Kanten maximiert wird. Dabei darf die Anzahl der dazu konstruierten Bicliques einen Wert k nicht überschreiten. Die Komplexität des Problems wird über die Setzung von $k = 1$ und damit über die Reduktion des Problems, eine Biclique mit einer maximalen Anzahl an Kanten zu finden, als $\mathcal{NP}$-schwer bestimmt. Da wir in unserer Anwendung die Anzahl der Bicliquen nicht beschränken wollen, ist dies für unser Problem kein ausreichender Komplexitätsnachweis. Die Packing-Restriktion wird von den Autoren entweder nur auf die Knoten der linken Seite L oder rechten Seite R oder aber auf alle Knoten des bipartiten Graphen angewandt. Als Lösungsansatz stellen die Autoren einen Branch-and-Price-Ansatz[7] für das Problem vor, wobei im Pricing-Problem der Trade-Off zwischen einer großen Anzahl an Kanten und dem Gewicht der Knoten aus den Packing-Restriktionen zu betrachten ist. Das Pricing-Problem wird mittels verschiedener IP-Modelle gelöst, die vor allem auf eine Anwendung mit $|L| \ll |R|$ und Packing-Restriktionen nur auf der kleineren Seite L abzielen, die Knoten aus R dürfen also mehr als einer Biclique enthalten sein. Die Übertragung dieses Ansatzes auf eine unbeschränkte Anzahl von Bicliquen bei Packing-Restriktionen auf beiden Seiten wäre somit die ultimative Lösung des Problems, wenn der aus den Bicliquen erzeugte Multidigraph keine künstlichen Kanten enthalten soll, wie es im folgenden Abschnitt der Fall ist.

3.7.3.5 Projektnetzpläne

In den beiden vorherigen Abschnitten haben wir zur Vermeidung von N-Graphen den Multidigraphen für Schritt 3 des Verfahrens von Valdes, Tarjan und Lawler durch Auswahl von Kanten erzeugt. In diesem Abschnitt wollen wir nun die transitive Reduktion G^R als ganzes in einen Multidigraphen für Schritt 3 transformieren. Bei dieser Aufgabe hilft uns die Literatur zum Projektmanagement weiter. So können wir die transitive Reduktion G^R als ein Activity-on-Node-(AoN)-Netzwerk auffassen, und dieses dann in ein Activity-on-Arc-(AoA)-Netzwerk transformieren. Während im ersten Fall die Aktivitäten im Projekt durch die Knoten im Graphen repräsentiert werden, und die Vorrangbeziehungen durch Kanten dargestellt werden, werden im AoA-Netzwerk die Aktivitäten

[7] Für eine ausführliche Einführung in diese Thematik siehe Kapitel 7 dieser Arbeit.

durch Kanten dargestellt. Die Knoten in diesem Netzwerk können dann als Ereignisse (Events) betrachtet werden, die unter anderem den Abschluss der mit den eingehenden Kanten assoziierten Aktivitäten repräsentieren. Die Vorzüge einer AoA-Repräsentation werden in Elmaghraby (1995) zusammengefasst. Um im Rahmen der Transformation in einer AoA-Netzwerk die Beibehaltung aller Vorrangbeziehungen zu garantieren, müssen in der Regel Dummy-Kanten in das AoA-Netzwerk eingefügt werden. Diese Transformation ist im allgemeinen Fall nicht eindeutig, so kann manchmal durch eine höhere Anzahl von Knoten im AoA-Netzwerk die Anzahl der notwendigen Dummykanten verringert werden. Eine zusätzliche Schwierigkeit besteht darin, dass je nach Anforderung an das AoA-Netzwerk auch die Transformation ein nicht einfaches Optimierungsproblem darstellt. So ist die Minimierung der Anzahl der verwendeten Dummykanten bei Verwendung eines knotenminimalen AoA-Netzwerks $\mathcal{NP}$-schwer (Krishnamoorthy und Deo 1979) Dagegen ist die Minimierung des Komplexitätsindex CI[8] des resultierenden AoA-Netzwerks in Polynomialzeit möglich, wie der Artikel von Kamburowski, Michael und Stallmann (Kamburowski, Michael und Stallmann 2000) zeigt, wenn das AoA-Netzwerk zugleich die minimal mögliche Anzahl von Knoten aufweisen soll. In manchen Fällen kann der Komplexitätsindex CI weiter verringert werden, wenn nicht in das knotenminimale AoA-Netzwerk transformiert wird (Kamburowski, Michael und Stallmann 2000, Abbildung 4). Für den Fall, dass nicht unbedingt das AoA-Netzwerk mit minimaler Knotenanzahl erzeugt werden muss, ist die Komplexität der Transformation in ein AoA-Netzwerk mit minimalem CI dagegen noch offen.

Im Verfahren von Kamburowski, Michael und Stallmann wird zunächst ein Netzwerk aufgebaut, dessen Kanten nur die tatsächlichen Aufträge sind. Die Knoten entsprechen den verschiedenen Vorgängermengen und den Schnitten von Vorgängermengen. Jede Kante, die einem Auftrag entspricht, startet nun in demjenigen Knoten, der die Vorgängermenge des Auftrages repräsentiert, und endet in demjenigen Knoten, der die Schnittmenge aller Vorgängermengen von Nachfolgern des Auftrags repräsentiert. Dabei wird der Schnitt über keine Menge als die Gesamtmenge aller Aufträge aufgefasst, die Kante für einen Auftrag ohne Nachfolger erhält somit den Knoten B_ω, der die gesamte Auftragsmenge repräsentiert, als Endpunkt. Im Gegensatz dazu erhält ein Auftrag ohne Vorgänger die leere Menge als Startpunkt B_α für die ihn repräsentierende Kante. Im Anschluss daran werden nun alle Vorrangbeziehungen des Ausgangsgraphen untersucht. Eine Vorrangbeziehung $i \prec j$ wird dann durch das Netzwerk nicht repräsentiert, wenn der Endknoten t_i von i nicht mit dem Startknoten s_j von j zusammenfällt. Dadurch wird nun mit einer Kante von t_i nach s_j ein Dummypfad eingeführt. Die Kanten der transitive Reduktion aller Dummypfade werden dann als Dummykanten in das AoA-Netzwerk mit eingefügt.

Die Transformation N-Graphen aus Abbildung 3.1 in ein AoA-Netzwerk würde wie folgt erfolgen: Aufgrund der Vorgänger und Nachfolger erhalten wir die Knoten $B_\alpha \cong \emptyset$, $B_\omega \cong \{1,2,3,4\}$ und $B_1 \cong \{1\}$, sowie $B_2 \cong \{1,2\}$, da in möglichen nichtleeren Mengen von Vorgängern nur 1 und 2 enthalten sein können. Auftrag 1 hat keinen Vorgänger, und ist der einzige gemeinsame Vorgänger seiner Nachfolger, also wird Auftrag 1 durch eine Kante (B_α, B_1) repräsentiert. Für Auftrag 2 repräsentiert B_2 die Schnittmenge der Vorgängermengen der

[8] Auf Definition und Bedeutung des Komplexitätsindex CI werden wir im nächsten Abschnitt genauer eingehen.

Nachfolger (hier nur Auftrag 4), und Auftrag 2 wird durch eine Kante (B_α, B_2) repräsentiert. Auftrag 3 und 4 haben keine Nachfolger, die Kanten enden demnach in B_ω und starten in B_1 beziehungsweise B_2. Da in diesem AoA-Netzwerk die Vorrangbeziehung $1 \prec 4$ noch nicht berücksichtigt ist, da die Kante von Auftrag 1 in einem anderen Knoten endet als die Kante von Auftrag 4 startet, müssen wir hier nun B_1 und B_4 durch eine Dummykante beziehungsweise Dummyauftrag verbinden. Dadurch erhalten wir eine Wheatstone-Brücke, und können auch hier sehen, dass der N-Graph die VSP-Eigenschaft nicht erfüllt.

Führen wir diese Transformation jedoch für einen VSP-Graphen aus, sind wir niemals auf Dummykanten angewiesen, da bei der Notwendigkeit von Dummykanten auch der im folgenden Abschnitt definierte Konfliktgraph ein Kante für jeden notwendigen Dummyauftrag besitzt (Kamburowski, Michael und Stallmann 2000, Lemma A.1).

3.7.3.6 Reduktion von Multidigraphen

In den Abschnitten 3.7.3.3, 3.7.3.4 und 3.7.3.5 wurden verschiedene Methoden vorgestellt, einen Multidigraphen mit zwei Terminalknoten B_α und B_ω zu erzeugen. Auf diesen Multidigraphen können wir den Schritt 3 des Verfahrens von Valdes, Tarjan und Lawler anwenden, und parallele und serielle Reduktionen durchführen, um einen Zerlegungsbaum zu erhalten und das Netzwerk hoffentlich auf eine Kante zu reduzieren. Doch was tun, wenn keine weitere serielle oder parallele Reduktion mehr möglich ist, und das reduzierte Netzwerk dennoch mehr als eine Kante besitzt?

Zunächst können wir versuchen, die im Netzwerk enthaltene Wheatstone-Brücke automatisch zu erkennen. Im Algorithmus von Valdes werden dazu in einer Tiefensuche Pfade berechnet, um eine enthaltene Wheatstone-Brücke zu erkennen, und diese in einen N-Graphen als Zertifikat für die nicht vorhandene VSP-Eigenschaft des gegebenen Vorranggraphen umzuwandeln. Doch damit fehlt uns immer noch eine Möglichkeit, den N-Graphen möglichst „minimalinvasiv“, also ohne großen Verlust von Vorrangbeziehungen, aus dem Netzwerk zu entfernen.

Im Artikel (W. W. Bein, Kamburowski und Stallmann 1992) stellen W. W. Bein, Kamburowski und Stallmann deshalb die Knotenreduktion als eine Operation vor, die geeignet ist, diese Wheatstone-Brücken elegant aufzulösen. Die Knotenreduktion ist anwendbar, sobald ein Knoten entweder nur noch einen Vorgänger (und mehrere Nachfolger) oder einen Nachfolger (und mehrere Vorgänger) hat, und ist damit eine Verallgemeinerung der seriellen Reduktion. Sei (u, v) die einzige Kante mit Zielknoten v, und es gebe eine Menge von Knoten $\{w_1, \ldots, w_k\}$, die Ziele der ausgehenden Kanten von v. Dann wird die Kante (u, v) entfernt und die Kanten $(v, w_i), i = 1, \ldots, k$ ersetzt durch die Kanten $(u, w_i), i = 1, \ldots, k$. Der umgekehrte Fall wird analog definiert.

Im Beispiel der Wheatstone-Brücke aus Abbildung 3.10 würde entweder Knoten B_1 mit einer eingehenden Kante oder Knoten B_2 mit einer ausgehenden Kante einer Knotenreduktion unterzogen. Für eine Knotenreduktion in Knoten B_1 machen wir alle von B_1 ausgehenden Kanten zu ausgehenden Kanten von B_α, die Kante (B_α, B_1) und damit der Auftrag 4 geht somit mit dem Knoten B_1 verloren. Im anderen Fall, also für eine Reduktion des Knoten B_2, bekommt dessen Nachfolger B_ω alle eingehenden Kanten von B_2 als Vorgänger zugeteilt, auch hier wird Knoten B_2 und die ausgehende Kante (B_2, B_ω) mit Auftrag 5

zunichte gemacht.

Um die Knotenreduktion und das Verfahren von W. W. Bein, Kamburowski und Stallmann (1992) für unsere Problemstellung nutzbar zu machen müssen wir den Verlust von Kanten und damit Aufträgen heilen. Dazu ist die folgende Überlegung hilfreich: Im betrachteten Graphen spiegelt ein Knoten die Menge der Vorrangbeziehungen zwischen den Aufträgen und Auftragsmengen der eingehenden und den ausgehenden Kanten wider. Führen wir die Reduktion eines Knotens durch, der nur eine eingehende Kante besitzt, wissen wir, dass der Auftrag beziehungsweise die Auftragsmenge auf der ersten Kante vor den Aufträgen der ausgehenden Kanten eingeplant werden muss. Durch die Knotenreduktion können wir diese Vorrangbeziehungen soweit relaxieren, dass wir die eingehende Auftragsmenge nur noch mit der Auftragsmenge einer ausgehenden Kante seriell verknüpfen, der Vorrang dieser Menge zu allen anderen ausgehenden Kanten geht dann verloren. Im Fall der Reduktion von Knoten B_1 heißt das, dass wir ursprünglich die Vorrangbeziehungen $4 \prec 2$ und $4 \prec 3$ gegeben haben. Nun müssen wir uns entscheiden, welche der Vorrangbeziehungen wir beibehalten, in dem wir diese als serielle Verknüpfung in einer der ausgehenden Kanten mit aufnehmen. Eine weitere Möglichkeit ist es nun, sich zunächst gar nicht darauf festzulegen, welcher ausgehenden Kante die Auftragsmenge der eingehenden Kante durch die serielle Verknüpfung zugedacht wird, sondern sich alle diese Möglichkeiten offen zu halten.

Mit der Knotenreduktion wird in (W. W. Bein, Kamburowski und Stallmann 1992) somit ein Werkzeug vorgestellt, mit dem sich jeder Graph mit zwei Terminalknoten unter weiterer Zuhilfenahme von serieller und paralleler Reduktion zu einem Graphen mit nur einer Kante machen lässt. Doch dieses Werkzeug richtet unweigerlich an unserem Vorranggraphen Schaden an, der dann möglichst gering ausfallen soll.

Die Knotenreduktion als Zusatzoperation zur seriellen und parallelen Reduktion erhält auch in anderen Anwendungsfällen eine besondere Bedeutung. In W. W. Bein, Kamburowski und Stallmann (1992) untersuchen die Autoren die Verlässlichkeit eines $s-t$-Netzwerkes, wenn jede Kante mit einer gewissen Wahrscheinlichkeit ausfällt. Zu bestimmen ist dann die Wahrscheinlichkeit mit der trotz dieser Ausfallwahrscheinlichkeiten dennoch eine $s-t$-Verbindung in dem Netzwerk vorhanden ist. Wird eine Knotenreduktion durchgeführt, ist dann die Verlässlichkeit des Netzwerks in zwei Fällen zu ermitteln. Im einen Fall geht das verbliebene Netzwerk vom Ausfall der reduzierte Kante aus, während im anderen Fall die Kante als zuverlässig eingestuft wird.

Wird die Knotenreduktion zur Analyse eines PERT-Netzwerkes[9] angewandt, so entspricht jede Reduktion eines Knotens der Fixierung der eigentlich zufälligen Aktivitätslänge derjenigen Aktivität, die durch die einzige ausgehende beziehungsweise eingehende Kante des von der Reduktion betroffenen Knotens repräsentiert wird (Kamburowski, Michael und Stallmann 2000).

Der Artikel Elmaghraby (1993) beschäftigt sich dagegen mit der Frage der Ressourcenzuweisung in einem Projektnetzwerk. Hier muss der von der Reduktion des Knotens betroffenen Aktivität dann bereits ein fester Anteil an der zu verteilenden Ressource zugewiesen werden, während dieser Anteil für alle anderen Aufträge erst nach der vollständigen Reduktion des Multidigraphen

[9] Die Dauer der einzelnen Aktivitäten des Projektes ist nicht deterministisch, sondern stochastisch gegeben.

ermittelt werden muss.

W. W. Bein, Kamburowski und Stallmann (1992) entwickeln einen Polynomialzeitalgorithmus, um die Reduktion auf eine Kante eins Graphen G mit einer minimalen Anzahl von Knotenreduktionen durchzuführen. Dazu gelingt es den Autoren, zunächst die möglichen Fälle, die zu einer Wheatstone-Brücke im betrachteten Graphen führen, zu klassifizieren und daraus mit Rückgriff auf bekannte Probleme aus der Graphentheorie ein effizientes Verfahren abzuleiten. Grundlage für die Reduktion des Graphen G ist die Konstruktion eines Konfliktgraphen $C(G)$. Dessen Knoten sind immer auch Knoten in G, die eventuell einer Knotenreduktion zu unterziehen sind. Jede Kante in $C(G)$ gibt dann an, dass mindestens einer der beiden angrenzenden Knoten auch tatsächlich einer Knotenreduktion unterzogen werden muss. Zur Feststellung von auftretenden Konflikten sind wir auf die nachfolgende Definition angewiesen:

Definition 3.16 (Dominanz)
Ein Knoten v dominiert einen Knoten w, wenn in allen Pfaden von s nach w auch der Knoten v vorkommt. v wird als Dominator von w bezeichnet. Wenn jeder Pfad von v nach t auch den Knoten w beinhaltet, dann dominiert w den Knoten v rückwärts (reverse), und w ist ein Rückwärtsdominator von v. Für die strikte Dominanz gilt zusätzlich $v \neq w$ in beiden Fällen.

Dominanz ist also dann gegeben, wenn es Knoten gibt, die unbedingt auf den Pfaden zum Ziel- beziehungsweise Startknoten liegen müssen.

Definition 3.17 (Komplexitätsgraph $C(G)$ W. W. Bein, Kamburowski und Stallmann (1992))
Eine Kante (v, w) im Komplexitätsgraphen $C(G)$ von G ist genau dann gegeben, wenn es einen (v, w)-Pfad $P(v, w)$ in G gibt, so dass jedes $u \in P(v, w)$ weder w strikt dominiert, noch v strikt rückwärts dominiert.

Es gibt also sowohl für den Startknoten v des (v, w)-Pfades einen alternativen Weg zur Senke t und auch für den Zielknoten w einen alternativen Weg von der Quelle s zu diesem Knoten. Jede Kante im Komplexitätsgraphen $C(G)$ deutet an, dass mindestens einer der Endpunkte einer Knotenreduktion unterzogen werden muss. Im Fall der Wheatstone-Brücke aus Abbildung 3.10 können wir gut den Konflikt zwischen den Knoten B_1 und B_2 erkennen. Zwischen B_1 und B_2 besteht offensichtlich eine Pfad, aber dennoch dominiert weder B_1 den Knoten B_2, da wir direkt von B_α zu B_2 kommen, noch dominiert B_2 den Knoten B_1 in Rückwärtsrichtung, da wir auch ohne an B_2 vorbeizukommen direkt von B_1 zu B_ω gelangen können.

Um alle Konflikte zwischen Knoten aus G und damit $C(G)$ zu erstellen, werden zwei Hilfsgraphen $C_1(G)$ und $C_2(G)$ auf Basis des transitiven Abschlusses von G erstellt. Dazu wird der Baum von Dominatoren $T(G)$ benötigt, dessen Berechnung ein bekanntes graphentheoretisches Problem ist. Die Dominanzrelationen zwischen den einzelnen Knoten können deswegen als Baum dargestellt werden, da einerseits die Quelle s alle weiteren Knoten dominiert, und andererseits jeder Knoten v einen einzigen unmittelbaren Dominator w hat, das heißt jeder weitere Dominator von v ist auch ein Dominator von w (Aho und Ullman 1972). Eine bekannte Anwendung ist zum Beispiel der Compilerbau, die ersten Algorithmen für dieses Problem sind insbesondere aus dieser Anwendung motiviert. So wird in (Aho und Ullman 1972) ein Algorithmus mit Laufzeit $\mathcal{O}(n \cdot m)$

vorgestellt. Ein bewährter Algorithmus für die Bestimmung von Dominatoren hinsichtlich des Trade-Offs zwischen asymptotischer Laufzeit und Aufwand für die Implementation ist der Algorithmus von Lengauer und Tarjan (1979) mit einer asymptotischen Laufzeit von $\mathcal{O}(m \log n)$ in der einfachen Variante, den wir in unserer Implementation verwenden. Beispiele weiterer Entwicklungen mit verbesserter asymptotischer Laufzeit gibt es in (Alstrup u. a. 1999),(Buchsbaum u. a. 2008) und (Fraczak u. a. 2013) mit Erratum in (Fraczak u. a. 2014). Um C_1 zu bestimmen, also den Graphen, dessen Kanten dadurch gegeben sind, dass der erste Teil der Definition 3.17 (es gibt kein u das w strikt dominiert) erfüllt ist, gehen wir alle Knoten v des Graphen G durch, um die Menge aller Nachfolger w mit in $(v, w) \in C_1(G)$ zu bestimmen. Wir markieren für einen Knoten v alle Nachfolger im transitiven Abschluss von G, für diese gibt es also einen Pfad von v nach w. Mit einer Tiefensuche im Baum der Dominatoren $T(G)$ suchen wir alle markierten Knoten, die keinen markierten Ahnen in $T(G)$ haben [10]. Jeder dieser Knoten w ist ein Nachfolger von v, und da keiner seiner Ahnen im Baum ein Nachfolger von v ist, wissen wir, dass er die gewünschte Eigenschaft hat, in $C_1(G)$ enthalten zu sein. Die gleiche Prozedur wird mit den Rückwärtsdominatoren durchgeführt, um den Graphen $C_2(G)$ zu erhalten, der alle Konflikte des zweiten Teils von Definition 3.17 behandelt. Kanten, die sowohl in $C_1(G)$ als auch in $C_2(G)$ enthalten sind, sind dann die Kanten des Konfliktgraphen $C(G)$

Sind alle Konflikte bekannt, ergibt sich dann die minimale Anzahl an Knotenreduktionen durch die Kardinalität einer minimalen Knotenüberdeckung aller Kanten in $C(G)$. Die Knoten, die in dieser Knotenüberdeckung vorkommen, müssen dann alle einer Knotenreduktion unterzogen werden.

Da $C(G)$ ein gerichteter azyklischer transitiver Graph ist, können wir den Graphen $C(G)$ wie bei der Zerlegung einer Halbordnung in Ketten (siehe Abschnitt 3.4) wieder in einen bipartiten Graphen überführen und mit dem Algorithmus von Hopcroft und Karp (Hopcroft und Karp 1973) ein Matching von maximaler Kardinalität bestimmen. Aus diesem Matching lässt sich dann über den Satz von Kőnig eine minimale Knotenüberdeckung mit gleicher Kardinalität ableiten (Bondy und U. S. R. Murty 1976).

W. W. Bein, Kamburowski und Stallmann (1992) zeigen, dass immer dann, wenn keine seriellen oder parallelen Reduktionen[11] mehr möglich sind, mindestens eine der Knotenreduktionen für einen Knoten aus der minimalen Knotenüberdeckung von $C(G)$ ausführbar ist. Es gibt also immer einer Knoten aus dieser Überdeckung, der nur noch einen Vorgänger oder Nachfolger hat, und auch nicht Endpunkt einer zusammenziehbaren Kante ist. Eine Kante (i, j) heißt hier zusammenziehbar, sobald sie die einzige ausgehende Kante von i und gleichzeitig die einzig in j eingehende Kante darstellt. In diesem Fall würde eine Knotenreduktion keine Konflikte im Konfliktgraphen $C(G)$ auflösen, sondern diesen nur insofern verändern, als dass der neue Konfliktgraph die Knoten i und j als einen einzigen Knoten auffasst. Weiterhin weisen W. W. Bein, Kamburowski und Stallmann (1992) nach, dass nach Durchführung aller ausgewählten Knotenreduktionen der Graph tatsächlich auf eine einzige Kante zusammengefallen ist.

Einen guten Überblick über das Vorgehen zur Bestimmung der notwendi-

[10] Bei Antreffen eines markierten Knotens müssen wir dann nicht weiter in die Tiefe gehen, weil alle Knoten unter diesem Knoten einen markierten Ahnen besitzen.

[11] Siehe Schritt 3 des Algorithmus von Valdes, Tarjan und Lawler (1982) auf Seite 37 zur Definition der Operationen

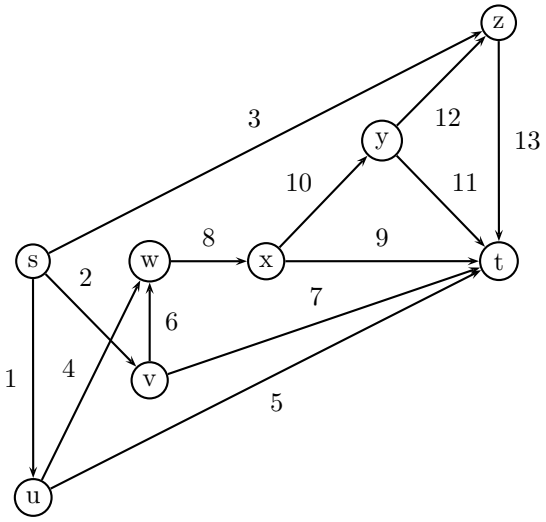


Abbildung 3.12: Beispielgraph für die Anwendung des Algorithmus von W. W. Bein, Kamburowski und Stallmann

gen Knotenreduktionen bildet die Abbildung 2 aus Elmaghraby (1993), die das Zusammenspiel der verschiedenen Algorithmen aus der kombinatorischen Optimierung verdeutlicht.

Um vor jeder Knotenreduktion (und vor der Konstruktion des Komplexitätsgraphen $C(G)$) sicherzustellen, dass keine seriellen und parallelen Reduktionen mehr möglich sind, führen wir den Reduktionsalgorithmus aus Schritt 3 des Verfahrens von Valdes, Tarjan und Lawler aus. Dazu werden alle Knoten in eine Liste von aktiven Knoten abgelegt. Der Reihe nach werden die Knoten aus der Liste entnommen und auf die Möglichkeit serieller und paralleler Reduktionen überprüft. Sind serielle oder parallele Reduktionen möglich, werden die beteiligten Knoten gegebenenfalls wieder in die Liste der aktiven Knoten aufgenommen.

Als Beispiel für eine Anwendung des Reduktionsalgorithmus von W. W. Bein, Kamburowski und Stallmann betrachten wir den Graphen aus Abbildung 3.12. Die Vorwärts- und Rückwärts-Dominatoren sind in den Abbildungen 3.13 und 3.14 angegeben. Aus dem transitiven Abschluss und den Dominatoren lassen sich dann die Kanten des Konfliktgraphen $C(G)$ aus Abbildung 3.15 erstellen. Die beiden Knoten w und z bilden die einzige minimale Knotenüberdeckung und müssen folglich reduziert werden. Da w der Startpunkt der zusammenziehbaren Kante (w, x) ist, wird eine Knotenreduktion von w zum jetzigen Zeitpunkt noch zu keiner Reduktion von Konflikten führen. Daher müssen wir zunächst den Knoten z reduzieren. Das Ergebnis der Reduktion ist in Abbildung 3.16 zu sehen[12]. Nun sind zunächst wieder einige serielle und parallele Reduktionen möglich, die den Graphen aus Abbildung 3.18 ergeben. Die mit A beschriftete Kante steht hierbei für eine ganze Auftragsmenge, deren Zerlegung in serielle und parallele Komponenten in Abbildung 3.17 dargestellt ist. Nach Reduktion-

[12] Zur besseren Darstellung wird in dieser und den folgenden Abbildungen der Graph teilweise neu angeordnet

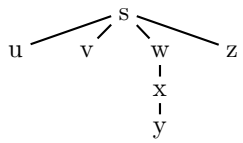


Abbildung 3.13: Vorwärtsdominatoren für den Graph aus Abbildung 3.12

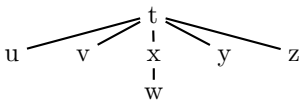


Abbildung 3.14: Rückwärtsdominatoren für den Graph aus Abbildung 3.12

on von von w ergibt sich dann der Graph aus Abbildung 3.19, der sich dann durch serielle und parallele Reduktionen vollständig auf eine $s-t$-Kante reduzieren lässt. Der sich ergebende Zerlegungsbaum ist dann in Abbildung 3.20 dargestellt. Durch die zwei Knotenreduktionen gibt es nun auch zwei Aufträge beziehungsweise Auftragsmengen, die an mehreren Stellen des Auftragsbaumes vorkommen, zum einen ist das die Auftragsmenge A, zum anderen auch der Auftrag 13, der einmal als Teil von A und einmal außerhalb von A anzutreffen ist.

Die Definition der Knotenreduktion und die Möglichkeit, die minimale Anzahl an Knotenreduktionen für einen gegebenen Graphen mit zwei Terminalknoten in Polynomialzeit zu bestimmen, haben zu einem großen Einfluss des Artikels W. W. Bein, Kamburowski und Stallmann (1992) in der Literatur zum Projektmanagement geführt (siehe hierzu Demeulemeester, Herroelen und Elmaghraby 1996; Reyck und Herroelen 1996; Brucker, A. Drexl u. a. 1999; Tavares 2002). So wurde in der Folge die minimale Anzahl an Knotenreduktionen zur Reduktion des Netzwerks auf eine einzige Kante als Komplexitätsindex (engl. complexity index) CI benannt. Dieser Komplexitätsindex ist dann auch zu einem wichtigen Parameter zur Charakterisierung von Projektmanagement-Problemen geworden, und so wurden auch Instanzgeneratoren entwickelt, die Netzwerke mit einem gegebenen Komplexitätsindex erzeugen (Agrawal, Elmaghraby und Herroelen 1996; Akkan, A. Drexl und Kimms 2005).

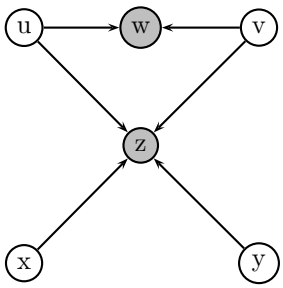


Abbildung 3.15: Konfliktgraph $C(G)$ für den Graph aus Abbildung 3.12

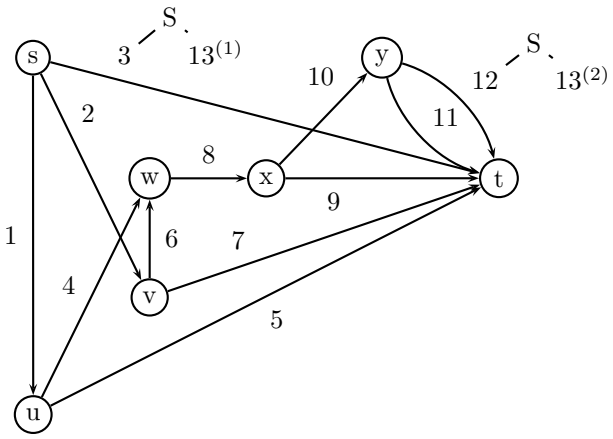


Abbildung 3.16: Beispielgraph nach Knotenreduktion von z

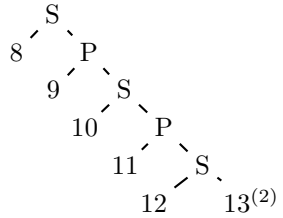


Abbildung 3.17: Zerlegungsbaum des Teilgraphen A

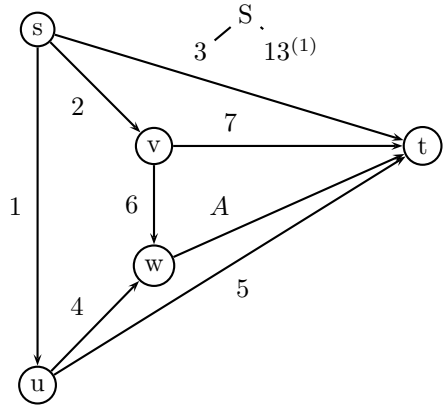


Abbildung 3.18: Beispielgraph nach weiteren parallelen und seriellen Reduktionen

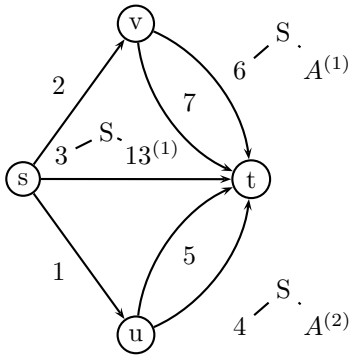


Abbildung 3.19: Beispielgraph nach Reduktion von w

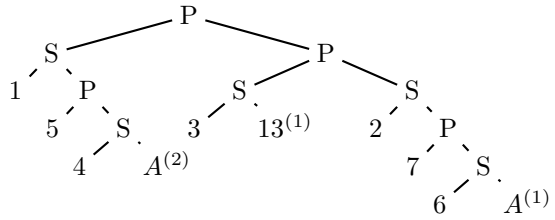


Abbildung 3.20: Zerlegungsbaum des Graphen aus Abbildung 3.12

3.7.3.7 Nachoptimierung der VSP-Zerlegung

Die Operation der Knotenreduktion ermöglicht es uns, nicht nur einen, sondern mehrere Zerlegungsbäume zu erzeugen, da wir bei der Wahl des Partners für die einzige eingehende (ausgehende) Kante für die serielle Verknüpfung mehrere Möglichkeiten haben. Im Beispiel der Zerlegung des Graphen aus Abbildung 3.12 haben wir letztendlich vier Möglichkeiten, eine VSP-Halbordnung zu erzeugen, da wir uns auf je ein Vorkommen des Auftrags 13 und ein Vorkommen der durch die Kante A repräsentierte Vorgängermenge einigen müssen. Bezeichnen wir mit $c(G)$ den Komplexitätsindex des Graphen G, so gibt es damit $\mathcal{O}(n^c)$ Möglichkeiten, eine VSP-Halbordnung aus dem Zerlegungsbaum des Graphen zu ermitteln. Die Qualität einer Zerlegung können wir dann entweder anhand der Maximalanzahl von erhaltenen Vorrangbeziehungen oder anhand des maximal erreichten optimalen Zielfunktionswertes für das Problem $1|prec_{VSP}|\sum w_j C_j$ messen. In unserer Implementation wird das zweite Maß verwendet und in der Reihenfolge der Knotenreduktionen aus dem Algorithmus von W. W. Bein, Kamburowski und Stallmann der Reihe nach jede Möglichkeit für die serielle Verknüpfung getestet. Wir testen somit nur eine Teilmenge aller Möglichkeiten, im Beispiel aus Abbildung 3.12 testen wir so nur drei der vier verschiedenen VSP-Halbordnungen. Für jede Knotenreduktion wird bei fixierter Auswahl der Partner für alle anderen Knotenreduktionen dann der Partner ausgewählt, der den besten Wert erreicht.

3.7.4 Bestimmung relaxierter Vorrangbeziehungen

Wenn wir nun bei Vorliegen eine VSP-Zerlegung daran interessiert sind, die Menge aller relaxierten Vorrangbeziehungen zu bestimmen, sind wir auf den transitiven Abschluss aller erhaltenen Vorrangbeziehungen angewiesen. Aufgrund der Zweidimensionalität einer Series-Parallelen Halbordnung kann im vierten Schritt des Algorithmus von Valdes, Tarjan und Lawler eine Einbettung der entstandenen Halbordnung in die kartesische Ebene berechnet werden. Gilt nun für zwei Knoten v und w, dass die Koordinate von v sowohl in x- als auch in y-Richtung kleiner ist als die Koordinaten von w, wissen wir, dass die Vorrangbeziehungen $v \prec w$ in der Zerlegung erhalten geblieben ist. Wir haben damit ein einfaches Mittel mit einer Laufzeit von $\mathcal{O}(|J|^2)$, um die relaxierten Vorrangbeziehungen zuverlässig zu identifizieren.

3.7.5 VSP-Vorranggraphen und Branch-and-Bound

Benötigen wir nun zur Berechnung von Schranken in einem Branch-and-Bound-Verfahren den Zerlegungsbaum eines VSP-Graphen, so können wir diesen auf einfache Weise für alle im Branch-and-Bound-Verfahren erzeugte Kindknoten ν anpassen. Da in der Schrankenberechnung nur noch offene Aufträge betrachtet werden, können wir in der Berechnung mit dem Algorithmus von Lawler die Blattknoten ignorieren, die einen schon eingeplanten Auftrag repräsentieren. Kommen wir in Folge zu einem S- oder P-Knoten mit einer leeren Seite, so übernehmen wir die Aufträge der nicht-leeren Seite einfach unverändert.

3.8 Weitere untere Schranken für $1|prec|w_jC_j$

Bis jetzt haben wir uns auf spezielle Klassen von Vorrangbeziehungen für das Schedulingproblem $1|prec|w_jC_j$ zurückgezogen, und untersucht, inwieweit wir aus den allgemein gegebenen nun eine spezielle Klasse von Vorrangbeziehungen ableiten können. Natürlich kann man sich nun auch weitere Vorgehensweisen überlegen, für das Gesamtproblem $1|prec|w_jC_j$ untere Schranken zu ermitteln. In der Literatur ist hier vor allem die Lagrange-Relaxation untersucht worden (zum Beispiel Potts 1985), an dieser Stelle wollen wir uns nun auf Techniken konzentrieren, die bereits in diesem Kapitel vorgekommen sind.

3.8.1 Verwendung der minimal gewichteten Knotenüberdeckung

Bei der Betrachtung der zweidimensionalen Halbordnungen haben wir die Problemformulierung von Correa und Schulz (2005) verwendet, um das Optimierungsproblem $1|prec|\sum w_jC_j$ als Problem der minimial gewichteten Knotenüberdeckung darzustellen. Die LP-Relaxation des Knotenüberdeckungsproblem lässt sich mit der Transformation in ein Fluss-Netzwerk aus G. L. Nemhauser und L. E. Trotter (1975) lösen, um eine untere Schranke zu erhalten. Diese Transformation verdoppelt die Anzahl der Knoten und Kanten. In dem resultierenden Netzwerk muss dann ein maximaler Fluss beziehungsweise minimaler Schnitt bestimmt werden. Deshalb ist eine Anwendung in Branch-and-Bound-Verfahren aufgrund der zu erwartenden hohen Laufzeit der Schrankenberechnung eine Herausforderung.

3.8.2 Modulare Zerlegung

Die Modulare Zerlegung eines Vergleichbarkeitsgraphen haben wir bisher nur beim Test auf eine Dimension von zwei für eine Halbordnung erwähnt. Haben wir einen Vergleichbarkeitsgraphen für einen beliebigen Vorranggraphen vorliegen, können wir allerdings auch auf Basis seiner Modularen Dekomposition ein Lösungsverfahren und untere Schranken für das Problem $1|prec|\sum w_j C_j$ entwickeln. In der Zerlegung wird ein Vorranggraph in serielle und parallele Komponenten zerlegt, zusätzlich zu den Singletons die einzelne Knoten oder Aufträge repräsentieren, gibt es dann im Fall eines Vorranggraphen ohne die VSP-Eigesnschaft noch kompliziertere Teilgraphen, die sich nicht zerlegen lassen, und deshalb als *prim* bezeichnet werden. Die Verfahren zur Bestimmung einer Modularen Zerlegung wurden ähnlich wie die Verfahren für die Bestimmung von Dominatoren in einem Netzwerk über die Jahrzehnte immer weiter verfeinert. Einen frühen Überblick über damals schon bekannte Verfahren liefert der Artikel Möhring (1985), die Weiterentwicklung bis zur heutigen Zeit lässt sich unter anderem in den Artikeln Dahlhaus, Gustedt und McConnell (2001) und die Kombination zwei Techniken in Habib, Montgolfier und Paul (2004) und Bergeron u. a. (2008) verfolgen.

Haben wir nun eine Modulare Zerlegung gegeben, so sind es eigentlich nur die Primknoten der Zerlegung, die Schwierigkeiten bereiten, für die Behandlung serieller und paralleler Verknüpfungen haben wir ein Patentrezept auf der Hand. Sind die Lösungen für die Primknoten bekannt, so können wir daraus auch die Lösungen für das Ursprungsproblem ableiten, wie die Artikel Sidney und Steiner (1986) (mit Fokus auf der Zerlegung) und Monma und Sidney (1987) (mit Fokus auf der Zielfunktion $\sum w_j C_j$) zeigen. Auch wenn der Vorrangraph insgesamt vielleicht nicht die VSP-Eigenschaft erfüllt, kann er dennoch in weiten Teilen diese Eigenschaft erfüllen, und es gibt nur wenige Primknoten, denen jeweils eine Teilmenge von Aufträgen zugeteilt ist. Die Lösung für diese Primknoten kann dann mit Hilfe der Dynamischen Programmierung in einer Laufzeit von $\mathcal{O}(n^\omega)$ bestimmt werden, wobei ω die minimale Anzahl von Ketten in der durch den Knoten repräsentierten Auftragsmenge darstellt(Möhring 1989; Steiner 1990). Ist die minimale Anzahl von Ketten ω in einem Primknoten fixiert, so bildet auch dieser Ansatz der dynamischen Programmierung einen Polynomialzeitalgorithmus. Erscheint uns der Wert für ω zu hoch, um direkt den Ansatz der Dynamischen Programmierung einzusetzen, können wir mit dem Verfahren von Steiner (1992) eine maximale Teilordnung mit fester Breite $\omega' \leq \omega$ bestimmen. In diesem Verfahren wird dazu ein spezielles Min-Cost-Flow-Problem erstellt, das dann mit entsprechenden Verfahren gelöst werden kann (Ahuja, Magnanti und J. B. Orlin 1993). Wir erhalten dann eine untere Schranke mit durch Wahl von ω' konfigurierbarer asymptotischer Laufzeit.

Kapitel 4

Vorbehandlung der Probleminstanzen

Für das hier betrachtete Schedulingproblem gilt, dass nicht jede Permutation der Aufträge, die sich an die vorgegebene Reihenfolge für Aufträge eines Produktes hält, auch eine zulässige Lösung des Problems darstellt. Da wir aufgrund der Komplexität des Problems alle Lösungen des Problems zumindest implizit betrachten müssen, ist es vorteilhaft, unzulässige Lösungen frühzeitig zu erkennen. Dazu fokussieren wir uns auf Eigenschaften, die von allen zulässigen Lösungen erfüllt werden müssen, oder eben auch auf Eigenschaften, die nur von unzulässigen Lösungen erfüllt werden. Durch die Vorbehandlung erhalten wir dann eine zur ursprünglichen Instanz des Problems äquivalente Instanz, die aufgrund der zusätzlichen Information aber hoffentlich leichter zu lösen ist.

Ein weiterer Aspekt, den wir in diesem Kapitel betrachten wollen, ist die Erzeugung von zulässigen Lösungen mittels einer Prioritätsregel und einer Heuristik. Wenn wir die optimale Lösung finden wollen, und dann auch zeigen wollen, dass es keine bessere Lösung gibt, ist es bereits hilfreich, überhaupt eine Lösung des Problems mit relativ gutem Zielfunktionswert an der Hand zu haben. Damit können wir dann Lösungen, die einen schlechteren Zielfunktionswert erzielen, sofort von unserer Suche nach der optimalen Lösung ausschließen.

4.1 Zeitfenster

In diesem Abschnitt wollen wir Zeitfenster für die einzelnen Aufträge auf Basis der Daten der Instanz ableiten, um dann anhand dieser weitere Vorrangbeziehungen generieren zu können.

4.1.1 Einführung von Bereitstellungszeitpunkten

Obwohl in unserer Problemdefinition Bereitstellungszeitpunkte (engl. release dates r_j) in der klassischen Variante nur dann zum Tragen kommen, wenn die Materialverfügbarkeit für jedes Produkt individuell bestimmt ist, können wir dennoch einen frühestmöglichen Bereitstellungszeitpunkt für jeden Auftrag angeben. Dieser ist einerseits bestimmt durch die minimale Zeit, um alle Vorgänger abzuarbeiten, andererseits durch den Materialverbrauch des Auftrags und seiner

Vorgänger. Dadurch wird ein frühester Zeitpunkt r_j für den Produktionsstart des Auftrags j vorgegeben.

4.1.2 Verschärfung der Fälligkeiten

Die Fälligkeiten δ_j der einzelnen Aufträge $j \in J$ lassen sich verschärfen, indem die Nachfolger betrachtet werden. Versuchen wir alle Nachfolger eines Auftrags anhand ihrer Prozesszeiten so einzuplanen, dass keiner dieser Aufträge verspätet fertiggestellt wird, entspricht diese Problemstellung auf umgekehrte Weise betrachtet dem Problem $1|r_j, prec|C_{\max}$. Dieses ist polynomiell lösbar (Lenstra, Kan und Brucker 1977), da es in Polynomialzeit auf das ebenfalls in Polynomialzeit lösbare Problem $1|prec|L_{\max}$ reduziert werden kann, es gilt also

$$1|r_j, prec|C_{\max} \propto 1|prec|L_{\max}. \tag{4.1}$$

Die Duedates d_j für das zweite Problem lassen sich aus den Freigabezeitpunkten r_j über die Verwendung einer Konstante K mit $d_j = K - r_j$ und $K > \max\{r_j\}$ ermitteln[1].

4.1.3 Frühester Produktionsstart bei Einplanung weiterer Aufträge

Definition 4.1 (Frühestmöglicher Produktionsstart (EAT))
Der früheste Produktionsstart $eat(i, j)$ eines Auftrags j in Abhängigkeit von einem anderen Auftrag $i \neq j$ ist der frühestmögliche Zeitpunkt zu dem der Auftrag j gestartet werden kann, wenn Auftrag i vorher abgearbeitet wird.

Die Berechnung von $eat(i, j)$ ermöglicht es uns, weitere Vorrangbeziehungen abzuleiten. Sobald $eat(i, j) + p_j > \delta_j$ gilt, wissen wir, dass wir nach vorheriger Produktion von i nicht mehr rechtzeitig vor der Fälligkeit δ_j den Auftrag j abschließen können (vergleiche Dumas u. a. 1995). Es entsteht damit eine unzulässige Verspätung, und wir können folgern, dass $j \prec i$ gelten muss. Die Betrachtung des frühestmöglichen Produktionsstarts für jede Kombination von Aufträgen ist damit eine Möglichkeit, die durch die Ketten gegebenen Vorrangbeziehungen um weitere Vorrangbeziehungen zu erweitern.

4.2 Abschätzung von Rüstzeiten

Der Aufbau und die Berechnung der Rüstzeiten erlauben es uns, die mindestens notwendige Rüstzeit vor der Produktion von Aufträgen abzuschätzen. Dabei stützen wir uns auf die vorhandenen Vorrangbeziehungen zwischen den Aufträgen. Ausgangspunkt einer jeden Berechnung ist dabei immer ein Auftrag i^*, der als *Rüstbasis* bezeichnet wird. Unser Interesse bei der Rüstzeitabschätzung gilt nur Aufträgen, die noch nach der Rüstbasis eingeplant werden können. Die natürliche Wahl der Rüstbasis ist damit der künstliche Startauftrag 0, sollten bereits weitere Planungsentscheidungen getroffen worden sein, kann auch ein anderer Auftrag die Rolle der Rüstbasis übernehmen.

[1] Die Originalquelle für diese Konstruktion konnte leider nicht ermittelt werden.

4.2.1 Minimale Rüstzeiten

Für die Berechnung der minimal auftretenden Rüstzeit vor einem Auftrag versehen wir die aktuelle transitive Reduktion G^R des Vorranggraphen mit für jedes rüstrelevante Material $m \in M$ individuellen Gewichten. Sei dazu $w^S : J \cup 0 \times J \times M \mapsto \mathbb{N}$ eine Gewichtsfunktion für alle möglichen Paare (i, j) von Aufträgen, definiert durch

$$w^S(i,j,m) = \begin{cases} st_{G(m)} & \text{wenn } Bom(j, G(m)) = m \wedge Bom(i, G(m)) \neq m \\ 0 & \text{sonst.} \end{cases} \tag{4.2}$$

Wir gewichten also die Kante von Auftrag i zu Auftrag j mit der Rüstzeit von Gruppe G, sofern das Material m für den Auftrag j gerüstet sein muss, aber nicht in der Stückliste von i enthalten ist. Für jeden Auftrag $j \in J$ gibt die Länge des längsten Pfades von i^* nach j in der transitiven Reduktion des Vorranggraphen in Hinblick auf ein Material $m \in M$ eine untere Schranke $st(j, m)$ für die Rüstzeit für Material m an, bevor Auftrag j gefertigt werden kann. Berechnen wir für jedes Material $m \in M$ diesen Pfad, können wir eine untere Schranke für die gesamte auftretende Rüstzeit vor der Fertigung von $j \in S$ ableiten:

$$st(j) = \sum_{m \in M} st(j, m) \tag{4.3}$$

Die Berechnung der Pfade selbst kann dabei in $\mathcal{O}(|E(G^R)| \cdot |M|)$ durchgeführt werden, da sie in einem azyklischen gerichteten Graphen berechnet werden (Cormen u. a. 2009).

4.2.2 Zusätzliche Rüstzeiten

Sobald wir nicht nur die durch den Vorranggraphen festgelegten Vorgänger eines Auftrags j vor diesem einplanen wollen, sondern auch noch einen weiteren Auftrag i, für den aktuell noch $i \not\prec j$ gilt, können wir relativ einfach den Anstieg der minimalen Rüstzeit berechnen. Die minimal auftretende Rüstzeit $st(i, j, m)$ für alle Vorkommen von Material m vor der Produktion von Auftrag j, bei vorheriger Fertigung von i, ergibt sich als:

$$st(i,j,m) = \max\{st(j,m), st(i,m) + w^S(i,j,m)\}. \tag{4.4}$$

Entweder muss bereits für die Produktion von j schon vorher oft auf Material m gerüstet werden, oder durch die Einplanung von i vor j muss bereits oft für die Produktion von i auf m gerüstet werden und/oder es kommt zu einer weiteren erforderlichen Umrüstung vor der Produktion von j. Für jede beliebige Menge von Aufträgen $P_j \subset J, j \in P_j$, die vor einem Auftrag j eingeplant werden können, ergibt sich somit die minimale Rüstzeit für ein Material $m \in M$ zwischen der Rüstbasis und dem Auftrag j durch:

$$st(P_j, j, m) = \max\{st(i, j, m) : i \in P_j\} \tag{4.5}$$

Die Abschätzung für die gesamte Rüstzeit für die Produktion der Menge $P_j \subseteq J$ mit letztem Auftrag $j \in P_j$ ist dann gegeben durch

$$st(P_j, j) = \sum_{m \in M} st(P_j, j, m). \tag{4.6}$$

Zusätzlich können wir die Rüstbasis auch auf einen beliebigen Auftrag $i \in J$ setzen, und von diesem aus alle Aufträge aus einer Menge $\tilde{P}_{i,j} \subset J$ mit $i \notin \tilde{P}_{i,j}$ mit letztem Auftrag j betrachten. Hier beginnen dann alle Pfade durch $\tilde{P}_{i,j}$ anhand des Vorranggraphen zunächst in i, das mit allen Aufträgen ohne weiteren Vorgänger in $\tilde{P}_{i,j}$ verbunden ist. Alle Aufträge ohne weiteren Nachfolger in $\tilde{P}_{i,j}$ werden dann mit dem Auftrag j verbunden. Auf diese Weise erhalten wir durch die durchzuführende Berechnung der längsten Pfade in diesem Netzwerk eine Abschätzung der minimalen Rüstzeit zwischen den Aufträgen i und j, wenn die Menge $\tilde{P}_{i,j}$ erst nach Fertigstellung von i zu bearbeiten ist. Diese Abschätzung einer minimalen Rüstzeit bezeichnen wir mit $st(i, \tilde{P}_{i,j}, j)$.

4.2.3 Alternative Abschätzungen

Eine weitere untere Schranke für die Rüstzeit innerhalb einer Auftragsmenge ergibt sich aus der Bestimmung eines kürzesten Hamilton-Pfades, in dem wir die Kanten zwischen zwei Aufträgen mit der Rüstzeit aus der Rüstmatrix gewichten. Dieses Problem ist im Allgemeinen jedoch auch ohne Beachtung von Vorrangbeziehungen $\mathcal{NP}$-schwer (Garey und D. S. Johnson 1979).

4.3 Positionsüberlegungen

Jede Jobsequenz für die Auftragsmenge J umfasst $|J|$ Positionen, wenn die Aufträge auf einer einzigen Maschine ausgeführt werden müssen. Auf jede dieser Positionen muss einer der Aufträge aus J platziert werden, und andererseits muss auch für jeden Job eine Position gefunden werden. Damit haben wir hier ein klassisches Matching-Problem in einem bipartiten Graphen vorliegen und jede zulässige Jobsequenz bildet ein perfektes Matching in diesem bipartiten Graphen. Unsere Nebenbedingungen führen dazu, dass nicht alle möglichen Kombinationen von Jobs und Positionen zulässig sein können, und wir wollen nun möglichst viele unzulässige Zuordnungen herausfiltern.

4.3.1 Maximal mögliche Positionen für einen Auftrag

Die Fälligkeit δ_j eines Auftrags $j \in J$ limitiert auf natürliche Weise die Fertigstellungszeit C_j des Auftrags, die wiederum auch davon abhängig ist, wie viele andere Aufträge vor diesem Auftrag eingeplant werden. Mit einem Branch-and-Bound-Ansatz versuchen wir eine Auftragsmenge $P_j \subseteq J$ mit $j \in P_j$ von maximaler Größe zu finden. Dabei muss die Abschätzung für die Fertigstellungszeit $C_j^{LB}(P_j)$ in Abhängigkeit der Auftragsmenge P_j die Restriktion $C_j^{LB}(P_j) \leq \delta_j$ einhalten. Die Größe der maximalen Menge P_j^* ist damit eine obere Schranke für die maximal mögliche Position des Auftrags j in jedem zulässigen Schedule und es gilt

$$p^{\max}(j) \leq |P_j^*|. \tag{4.7}$$

4.3.2 Auftragsmengen für Positionen

Für jede Position $1 \leq p \leq |J|$ in einer zulässigen Lösung gibt es eine Menge von Aufträgen $J_p \subseteq J$, die an dieser Position eingeplant werden können. Durch das Vorhandensein von Fälligkeitszeitpunkten und Vorrangbeziehungen

ist nicht jeder Auftrag für jede Position geeignet. Insbesondere für den hinteren Teil eines Schedules müssen wir davon ausgehen, dass grundsätzlich nur wenige Aufträge für diese Positionen in Frage kommen (siehe auch die Abschätzung der maximalen Positionen in Abschnitt 4.3.1). Werden im Verlauf des Optimierungsalgorithmus Sequenzierungsentscheidungen getroffen, verringern sich die Größen dieser Mengen, da potentielle Kandidaten für eine Position p aus der Menge J_p an anderer Position eingeplant werden. Um zu prüfen, ob auch für die letzten Positionen eines Schedules noch genug Aufträge zur Verfügung stehen, überprüfen wir einen Teil der Voraussetzung des Satzes von Hall (1935), der auch unter dem Namen Heiratssatz bekannt ist. Allgemein muss für jede Teilmenge von Positionen $\Psi \subseteq \{1, \ldots, |J|\}$ die Ungleichung

$$|\bigcup_{p \in \Psi} J_p| \geq |\Psi| \tag{4.8}$$

als notwendige und hinreichende Bedingung für die Existenz eines perfekten Matchings erfüllt sein.

In der Spezialisierung betrachten wir nicht jede beliebige Teilmenge von Positionen, sondern fordern für alle Positionen p, dass $|\bigcup_{p' \geq p} J_{p'}| \geq |J| - p + 1$ erfüllt ist. Wenn wir zusätzlich voraussetzen, dass ein Auftrag bereits dann Teil von J_p ist, wenn $p^{\max}(j) \geq p$ erfüllt ist, können wir in einer asymptotischen Laufzeit von $\mathcal{O}(|J|)$ überprüfen, ob bereits eine Verletzung der Voraussetzung des Satzes von Hall (1935) vorliegt. Zunächst zählen wir für jede Position p die Aufträge mit maximaler Position $p_j^{\max} = p$, und bestimmen dann in absteigender Reihenfolge der Position p die Mengen J_p beziehungsweise die Kardinalität der Vereinigung der Mengen $|\bigcup_{p' \geq p} J_{p'}|$ und vergleichen diese mit der Kardinalität der Menge der Positionen $\Psi = \{p, \ldots, |J|\}$.

In Abschnitt 6.6.6.4 wird ein weiterer Zulässigkeitstest auf Basis von Positionsbetrachtungen vorgestellt, der neben maximalen Positionen auch minimale Positionen und Vorrangbeziehungen berücksichtigt.

Gilt für eine Position p die Gleichung

$$|\bigcup_{p' \geq p} J_{p'}| = |J| - p + 1, \tag{4.9}$$

die Voraussetzung des Satzes von Hall ist also mit Gleichheit erfüllt, können wir Vorrangbeziehungen $i \prec j$ mit $i \in J \backslash (\bigcup_{p' \geq p} J_{p'})$ und $j \in \bigcup_{p' \geq p} J_{p'}$ ableiten.

4.3.3 Positionen und Prozesszeiten

In den in den weiteren Kapiteln vorgestellten Optimierungsansätzen werden wir darauf angewiesen sein, für jede Position die minimale kumulative Prozesszeit zu bestimmen, also auf optimale Lösungen der folgenden Klasse von Optimierungsproblemen für jede Position $1 \leq p \leq |J|$ zugreifen müssen:

$$\min \sum_{j \in J'} p_j \text{ s.t. } J' \subseteq J \wedge |J'| = p \tag{4.10}$$

Liegen keine Vorrangbeziehungen vor, ergibt sich die Lösung für eine Position p aus der Summe der Prozesszeiten der ersten p Jobs, wenn diese nach monoton steigender Prozesszeit p_j sortiert sind. Sollen jedoch Vorrangbeziehungen in

Form von Ketten beachtet werden, können wir folgenden Ansatz der Dynamischen Programmierung mit einer Laufzeit von $\mathcal{O}(|J|^2)$ einsetzen: Sei $\mathcal{C}$ eine geordnete Partitionierung von J in Ketten. Ein Zustand $T(p,c)$ gibt die minimale kumulative Prozesszeit für Position p an, sobald die ersten c Ketten aus $\mathcal{C}$ evaluiert worden sind. Die initialen Zustände sind $T(0,0) = 0$ und $T(p,0) = \infty \forall p > 1$. Der letztendlich optimale Zielfunktionswert für Position p ist $T(p,|\mathcal{C}|)$. Die Rekurrenzgleichung wird für $c \geq 1$ angewendet, und $C = (j_1, \ldots, j_{|C|})$ sei die c-te Kette aus $\mathcal{C}$:

$$T(p,c) = \min\{T(p,c-1), \min\{T(p-k,c-1) + \sum_{k'=1}^{k} p_{j_{k'}} : k = 1, \ldots, |C|\}\} \quad (4.11)$$

Die Laufzeit des durch (4.11) beschriebenen Ansatzes lässt sich asymptotisch durch $\mathcal{O}(|J|^2)$ abschätzen und hat einen Speicherplatzbedarf von $\mathcal{O}(|J|)$. Für jede Kette C werden die $|C|$-Jobs an maximal $|J|$ Positionen um eine Verbesserung des Zielfunktionswertes überprüft. Mit $\sum_{C \in \mathcal{C}} |C| = |J|$ ergibt sich damit die Laufzeit. Wenn die Rekurrenzgleichung mit absteigenden Werten für p' und die gesamte Kette c ab p' ausgewertet wird, kann der Wert $T(p, c-1)$ durch den neuen Wert $T(p,c)$ überschrieben werden, da er dann nicht mehr benötigt wird. Damit wird zusätzlich zu den optimalen Zielfunktionswerten kein weiterer Speicherplatz beansprucht, sofern wir auf die Möglichkeit der Rekonstruktion von optimalen Lösungen verzichten.

4.3.4 Positionen und Materialverfügbarkeit

Neben der Prozesszeit hat auch der Materialverbrauch beziehungsweise die Materialverfügbarkeit einen Einfluss auf die minimale Fertigstellungszeit von Aufträgen, wenn wir diese an bestimmten Positionen einplanen wollen. Mit den folgenden zwei Ansätzen wird eine untere Schranke für den früheste Zeitpunkt der Materialverfügbarkeit pro Position in einem zulässigen Schedule ermittelt.

4.3.4.1 Dynamische Programmierung

Der soeben in Abschnitt 4.3.3 vorgestellte Ansatz der Dynamischen Programmierung kann auch zur Berechnung weiterer Minima pro Position bei Vorliegen von Jobketten angepasst werden. So kann auch die mindestens benötigte Menge jedes Rohmaterials $m \in M$, um p verschiedene Jobs zu fertigen dazu dienen, einen frühesten Zeitpunkt der Materialverfügbarkeit für jede Position anzugeben. Da hierzu der Algorithmus allerdings nur auf jedes Material einzeln angewendet werden kann, bilden die minimalen Materialaufwendungen dann einen Idealen Punkt hinsichtlich der Multikriteriellen Optimierung (Ehrgott 2005). Damit werden wir also tunlichst die Einplanung von Aufträgen, die das betrachtete Material benötigen, soweit irgend möglich auf spätere Positionen verlagern. Um diesen Effekt abzumildern, können wir von einer Betrachtung der einzelnen Materialien abweichen, und uns ganzen Materialgruppen zuwenden. Unabhängig davon, welches Material aus der Gruppe von dem Auftrag benötigt wird, zählt dann ein Auftrag als Verbraucher in der Materialgruppe. Dann kann jede Lieferung eines Materials aus der Gruppe auch zur Abdeckung des Bedarfes eines anderen Materials aus derselben Gruppe verwendet werden.

4.3.4.2 Branch-and-Bound

Für jeden Auftrag j kann die früheste Zeit der Materialverfügbarkeit, sobald ein Auftrag an Position p eines Schedules eingeplant wird, auch durch eine Adaption des Branch-and-Bound-Verfahrens aus Abschnitt 4.3.1 bestimmt werden. Dies hat den Vorteil, dass auch alle Vorrangbeziehungen beachtet werden können. Die Auswertung wird nicht nur auf den Materialgruppen basierend, sondern auf den Rohmaterialien selbst durchgeführt. Die berechneten Werte können dann verwendet werden, um die mit dem Ansatz der Dynamischen Programmierung zuvor (siehe Abschnitt 4.3.4.1) berechneten Werte weiter zu verschärfen. Dieses Branch-and-Bound-Verfahren muss für jeden Auftrag einzeln durchgeführt werden, allerdings kann dann gleichzeitig für alle noch möglichen Positionen des Auftrags die optimale Lösung im Sinn einer frühesten Materialverfügbarkeitszeit für den Auftrag an einer Position bestimmt werden. Da das Verfahren für größere Instanzen trotzdem sehr aufwendig ist, wird es nur einmal nach der allgemeinen Vorbehandlung einer Probleminstanz durchgeführt.

4.4 Filterung von Kanten

Nun können wir uns auch Gedanken darüber machen, wann denn ein Auftrag direkt vor einem anderen Auftrag eingeplant werden darf, beziehungsweise wann mit Sicherheit gesagt werden kann, dass es keine zulässige Lösung gibt, in der Auftrag i direkt vor Auftrag j gefertigt wird.

Das folgende Lemma listet hinreichende Gründe für die Entfernung einer Kante aus dem Graphen auf:

Lemma 4.2
Ist eine der folgenden Bedingungen erfüllt, so kann die Kante $(i,j) \in (J \cup \{0\}) \times J$ *nicht Teil eines zulässigen Schedules sein:*

1. $j \prec i$
2. $i \prec j \wedge \exists k \neq i,j : i \prec k, k \prec j$
3. $p^{\max}(i) + 1 < p^{\min}(j)$.

Beweis.
1. Ist j ein Vorgänger von i, so kann i niemals vor j eingeplant werden, und j kann damit auch nie direkt auf i folgen.
2. Die direkte Folge von j auf i ist nicht möglich, da k zwischen i und j einzuplanen ist.
3. Die maximale Position $p^{\max}(i)$ ist zu klein, um ohne Einplanung eines weiteren Auftrags den Auftrag j direkt folgen zu lassen. □

Für alle Paare $(i,j) \in (J \cup \{0\}) \times J$ können wir die Bedingungen von Lemma 4.2 überprüfen, um die Menge der verbotenen Kanten zu definieren:

Definition 4.3 (Verbotene Kanten)
Die Menge der verbotenen Kanten $E^{\not\rightarrow} \subseteq (J \cup \{0\}) \times J$ *ist gegeben durch*

$$E^{\not\rightarrow} := \{(i,j) \in (J \cup \{0\}) \times J : \textit{Lemma 4.2 trifft auf } (i,j) \textit{ zu.}\}. \tag{4.12}$$

Definition 4.4 (Kantengraph)
Sei $G^{\rightarrow}(J \cup \{0\}, E^{\rightarrow})$ *der gerichtete Graph mit einer Knotenmenge aus den Aufträgen inklusive des künstlichen Startauftrags* 0 *und der Kantenmenge*

$$E^{\rightarrow} = (J \cup \{0\}) \times J) \backslash E^{\nrightarrow}. \tag{4.13}$$

Lemma 4.5
Jede zulässige Lösung des betrachteten Schedulingproblems ist ein Hamilton-Pfad in $G^{\rightarrow}$*, der im Knoten* 0 *beginnt.*

Das Vorliegen eines Hamilton-Pfades in diesem Graphen ist nur eine notwendige, jedoch nicht hinreichende Bedingung für eine zulässige Lösung.

4.5 Heuristiken zur Lösungsgenerierung

Wollen wir einen möglichst guten oder sogar den besten Produktionsplan finden, können wir zunächst versuchen, mit Heuristiken eine Lösung unseres Problems zu bestimmen. Die Frage nach der Existenz einer mit den Fälligkeitszeitpunkten δ_j für jeden Auftrag j in Einklang stehenden Lösung ist bereits $\mathcal{NP}$-vollständig (siehe Abschnitte 2.8 und 2.9.3), und so müssen wir in Kauf nehmen, dass wir mit einer unzulässigen Lösung starten müssen. Da die Unzulässigkeit durch die Überschreitung der Fälligkeiten einzelner Aufträge gegeben ist, wählen wir als Maß für die Größe der Unzulässigkeit die Summe der Verspätungen aller Aufträge.

4.5.1 Konstruktion einer ersten Lösung

Eine erste nicht notwendigerweise zulässige Lösung erhalten wir, in dem wir die Aufträge nach monoton ansteigender Fälligkeit sortieren. Da die Ketten für jedes Produkt durch die monoton ansteigenden Fälligkeitszeitpunkte innerhalb des Produktes gegeben waren, erfüllt jede dieser Lösungen die Vorrangbeziehungen in Form von Ketten, die Unzulässigkeit der Lösung ist nur auf etwaige Fälligkeitsüberschreitungen beschränkt.

4.5.2 Tabusuche

Zur weiteren Verbesserung dieser ersten Lösung wenden wir eine Tabusuche (Glover 1989; Glover 1990) an. Dabei handelt es sich um eine Metaheuristik[2], für die Konstruktion einer auf unser Problem angepassten Tabusuche sind also die Nachbarschaftsoperationen und die Tabuliste entsprechend zu definieren. Ein Reihe von aufeinander folgenden Aufträgen in der aktuellen Lösung wird als Block bezeichnet, wenn sie hinsichtlich eines Kriteriums übereinstimmen. Das kann einerseits das Produkt sein, oder die Produkte der beteiligten Aufträge stimmen in der Stückliste bezüglich eines Materials aus einer bestimmten Materialgruppe überein. Zusätzlich kommt auch ein einzelner Auftrag als Block infrage. In der Nachbarschaftsoperation werden dann alle Blöcke der Lösung einzeln an weitere zulässige Positionen verschoben, um neue Lösungen zu erzeugen. Die zulässigen Verschiebungen sind dabei diejenigen Verschiebungen, die

[2]Eine Vertiefung in diese Thematik bietet zum Beispiel der Sammelband von Gendreau und Potvin (2010)

den gesamten Block um einen oder mehrere Aufträge nach vorne oder hinten schieben. Dabei ist es zulässig, andere Blöcke, sofern sie aus mehreren Aufträgen zusammengesetzt sind, auseinander zu brechen. Die maximale Verschiebung in eine der beiden Richtungen wird entweder durch Anfang und Ende der Sequenz bestimmt, oder durch einen gleichartigen Block von Aufträgen. Damit dürfen sich Aufträge, die im blockbildenden Kriterium übereinstimmen, nicht überholen. Um auch im Fall von Blöcken aus Einzelaufträgen die Produktketten beizubehalten, darf ein Einzelauftrag auch keinen Auftrag desselben Produkts überholen.

In der Tabuliste speichern wir Charakteristika der übernommenen Lösungen, um nicht wieder zu diesen Lösungen zurückzukehren (vergleiche Taillard u. a. 1997; Ferrucci, Bock und Gendreau 2013). Um den Tabustatus einer Lösung zu ermitteln, berechnen wir für jede eventuell als nächste zu übernehmende Lösung eine CRC32-Checksumme (Peterson und Brown 1961), um sie mit den bereits in der Tabuliste vorhandenen Lösungen abzugleichen. Während der Tabusuche wird keine Lösung aus der Tabuliste entfernt. Um die Ermittlung des Tabu-Status zu beschleunigen, wird zuerst anhand des Zielfunktionswertes z der Rest der ganzzahligen Division $z \mod p$, wobei p eine Primzahl darstellt, als Vorfilterung verwendet. Erst wenn zu diesem Rest auch Lösungen in der Tabuliste[3] gespeichert worden sind, ist dann die Checksumme zu berechnen.

Die Tabusuche wird in der Implementation immer in der gesamten Nachbarschaft über alle Blocktypen durchgeführt. Für jeden Blocktyp wird ein einzelner Thread benutzt, um jede Iteration der Tabusuche zu beschleunigen, so dass die Nachbarschaftsevaluation maximal um den Faktor $|\mathcal{G}| + 1$ beschleunigt werden kann, wenn $\mathcal{G}$ die Menge der Materialgruppen inklusive der Produkte repräsentiert. Die Tabusuche endet, nachdem eine festgelegte Anzahl an Iterationen erfolgt ist, oder wenn sich über eine festgelegte Anzahl von Iterationen keine globale Verbesserung des Zielfunktionswertes ergeben hat. Bei Bewertung der Lösungsqualität sind Lösungen mit niedrigerer beziehungsweise keiner Gesamtverspätung immer vorzuziehen (lexikographische Sortierung), ansonsten entscheidet die gewichtete Fertigstellungszeit über die Priorität der Lösungen.

[3] Implementiert ist die Tabuliste deswegen auch nicht als Liste, sondern als Array von Suchbäumen, als Schlüssel zu einem Eintrag des Suchbaums bezüglich eines fixen Restwerts wird dann die CRC32-Checksumme verwendet.

Kapitel 5

Ganzzahlige Optimierungsmodelle

In dieser Arbeit sollen vor allem exakte Verfahren zur Lösung des vorliegenden Optimierungsproblems im Vordergrund stehen. Nicht fehlen darf dabei die Methode der Ganzzahligen Programmierung, die aufgrund ihrer recht universellen Anwendbarkeit ein Standardwerkzeug für die Optimierung verschiedenster Problemstellungen darstellt. Grundlage für die Anwendung der Ganzzahligen Programmierung ist immer das Vorliegen eines (gemischt) ganzzahligen Optimierungsmodells, dass dann mit Hilfe geeigneter Softwarepakete gelöst wird. Im vorliegenden Kapitel wird nun die Methodik der Modellierung und des Lösungsverfahrens kurz vorgestellt. Anschließend werden verschiedene Möglichkeiten der Modellierung des vorliegenden Schedulingproblems diskutiert.

5.1 Methodik

Die Ganzzahlige Programmierung (engl. Integer Programming IP) stellt in der Anwendung auf beliebige Optimierungsmodelle ein $\mathcal{NP}$-schweres Optimierungsproblem (Karp 1972) dar, da eine direkte Reduktion des Erfüllbarkeitsproblems möglich ist. Sie bildet eine Erweiterung der Linearen Programmierung. Während in der Linearen Programmierung ein lineares Programm immer als lineares Gleichungssystem $Ax = b$ mit einer durch ein Skalarprodukt gegebenen Zielfunktion $\min / \max c^\top x$ und einem Wertebereich $x \geq 0$ für die Variablen dargestellt werden kann, wird nun in der (gemischt) Ganzzahligen Programmierung eine Einschränkung des Wertebereichs aller (oder eines Teils) der Variablen auf ganzzahlige Werte eingeführt. Wird diese letzte Einschränkung in einem (gemischt) Ganzzahligen Programm aufgehoben, so bezeichnet man das resultierende Lineare Programm deshalb als LP-Relaxation des Ganzzahligen Programms.

Bei der Erstellung eines gemischt ganzzahligen Modells für ein gegebenes Optimierungsproblem sind deshalb immer Variablen festzulegen, die mögliche Entscheidungen und mit ihren Werten damit Lösungen des Problems kodieren. Damit diese Wertebelegungen auch tatsächlich einer Lösung des Problems entsprechen, sind die Nebenbedingungen entsprechend der Anforderungen an ein lineares Gleichungssystem zu formulieren. Auch die Kosten einer Lösung müssen sich vollständig aus den Koeffizienten einzelner Variablen in der Zielfunktion er-

geben. Dieser Transformationsprozess des gegebenen Optimierungsproblems in ein Modell ist nicht eindeutig, und es kann durchaus verschiedene Ansätze zur Modellierung ein und desselben Optimierungsproblems geben, die dann allerdings unterschiedlich gut zu lösen sind. Deshalb kann hier auch von der Kunst der Modellierung gesprochen werden.

Für eine gegebene Instanz des Optimierungsproblems werden dann entsprechend der Dimensionen der Instanz die einzelnen Variablen angelegt und die Koeffizienten von Gleichungssystem und Zielfunktion mit den aus den Parametern der Instanz berechneten Werten belegt, um ein ganzzahliges Optimierungsproblem genau für die gegebene Instanz zu erhalten. Dieses ist dann zu lösen.

Da die Werte der Variablen ganzzahlige Werte annehmen sollen, kann das Modell in einer Vorbehandlung modifiziert werden (siehe dazu Savelsbergh 1994), indem zum Beispiel Koeffizienten und die rechte Seite der Zielfunktion angepasst werden, und redundante Spalten und Zeilen des Modells entfernt werden. Ziel dieser Modifikationen ist eine Verkleinerung des Modells, sowie eine Verschärfung der LP-Relaxation, deren Lösung als Schranke und zur Ableitung des nächsten Verzweigungsschrittes in dem darauf folgenden Branch-and-Bound-Verfahren (Dakin 1965) eingesetzt wird.

Die LP-Relaxation kann durch die Einführung von Schnittebenen, die Teile des Polyeders ohne ganzzahlige Lösungen abschneiden, weiter verschärft werden. Diese Schnittebenen können entweder Schnitte sein, die generell für Ganzzahlige Programme geeignet sind, da sie LP-Relaxation im Allgemeinen betrachten, oder Schnitte, die speziell auf das modellierte Optimierungsproblem angepasst sind. Während die erstere Gruppe Teil eines jeden vernünftigen Softwarepaketes zur Lösung von ganzzahligen Optimierungsproblemen sein sollten, ist für die letzte Gruppe von Schnittebenen eine spezielle Kenntnis über das zugrunde liegende Optimierungsproblem notwendig. Der in Kapitel 7 vorgestellte Optimierungsansatz beinhaltet Beispiele für den zweiten Fall.

Aufgrund der Anforderungen an ein ganzzahliges Optimierungsmodell können auch recht kleine Instanzen eines betrachteten Optimierungsproblems in großen Modellen resultieren, so dass dann dieses Universalwerkzeug aufgrund von Ressourcenengpässen hinsichtlich Speicherplatz und Prozesszeit nicht mehr geeignet ist. Eindrücklich beschrieben anhand eines Beispiels aus der Frühzeit der Ganzzahligen Programmierung wird dieses Problem in Gomory (2002).

5.2 Grundmodell

In diesem Abschnitt führen wir ein erstes ganzzahliges Optimierungsmodell für das betrachtete Schedulingproblem ein, dass dann in Kapitel 8 auch experimentell evaluiert werden soll. Dieses basiert auf der Formulierung von Manne (1960), definiert also im wesentlichen für jeden Auftrag die Menge der Aufträge, die vor einem Auftrag ausgeführt werden sollen. In Abhängigkeit davon lässt sich dann die Fertigstellungszeit der Aufträge und damit die Zielfunktion bestimmen.

Die Fertigstellungszeit eines Auftrags $j \in J$ wird durch die kontinuierliche, aber implizit ganzzahlige Variable $C_j \geq 0$ abgebildet. Dann lässt sich die Zielfunktion unseres Schedulingproblems $1|chains, rm, setup, \delta_j| \sum w_j C_j$ einfach in

das Modell übernehmen:

$$\min \sum_{j \in J} w_j \cdot C_j. \tag{5.1}$$

Die Fertigstellungszeit eines Auftrags darf die Fälligkeit des Auftrags nicht übersteigen, es gilt also

$$C_j \leq \delta_j \quad \forall j \in J. \tag{5.2}$$

Andererseits muss die Fertigstellungszeit so gewählt werden, dass nicht vor dem Zeitpunkt 0 mit der Produktion gestartet wird, es gilt also

$$C_j \geq p_j \quad \forall j \in J. \tag{5.3}$$

Anstelle eine Reihenfolge der Aufträge direkt abzubilden, verwenden wir binäre Variablen $\pi_{i,j} \in \{0, 1\}$, um anzuzeigen, ob ein Auftrag $i \in J$ einem Auftrag $j \in J$ im von der Lösung repräsentierten Schedule vorangeht oder nicht. Zur Vereinfachung der Notation sei jeder Auftrag sein eigener Vorgänger, es gelte also $\pi_{j,j} = 1$ für alle $j \in J$. Da wir einen Schedule für eine einzelne Maschine erzeugen, alle Aufträge also der Reihe nach auf dieser Maschine zu fertigen sind, gilt für jedes Paar $(i, j) \in J \times J, i \neq j$ von Aufträgen, das entweder i vor j oder j vor i gefertigt wird. Deshalb muss auch die Nebenbedingung

$$\pi_{i,j} + \pi_{j,i} = 1 \quad \forall i, j \in J : i < j \tag{5.4}$$

erfüllt sein. Alle vorbestimmten Vorrangbeziehungen $i \prec j$ für zwei Aufträge i und j können durch Setzung von $\pi_{i,j} = 1$ erzwungen werden. Dies trifft insbesondere auf die Bedingung zu, die Aufträge eines Produkts in aufsteigender Reihenfolge nach Fälligkeit zu fertigen. Wird ein Auftrag i zum Vorgänger eines Auftrags j in der aktuellen Lösung bestimmt, darf die Fertigstellung C_j von j erst mit einem gewissen Mindestabstand zur Fertigstellungszeit C_i von i eintreten, der durch die Rüstzeit $st(i, j)$ und die Fertigungszeit p_j definiert wird:

$$\pi_{i,j} = 1 \Rightarrow C_i + st(i, j) + p_j \leq C_j \tag{5.5}$$

Ist die Vorrangbeziehung zwischen den beiden Aufträgen $i, j \in J$ nicht im Vorhinein gegeben, so muss die Implikation in Formel (5.5) mit Hilfe der BigM-Technik abgebildet werden, um ein lineares Modell zu erhalten (Roshanaei, Azab und ElMaraghy 2013).

Die Überprüfung der Materialverfügbarkeit wird nun in drei Klassen von Nebenbedingungen ausgedrückt:

$$\sum_{\tau \in T} \rho_{j,\tau} = 1 \quad \forall j \in J \tag{5.6}$$

$$\sum_{i \in J} a_{i,m} \cdot \pi_{i,j} \leq \sum_{\tau \in T^M} R_{m,\tau} \cdot \rho_{j,\tau} \quad \forall m \in M, j \in J \tag{5.7}$$

$$\sum_{\tau \in T} \tau \cdot \rho_{j,\tau} + p_j \leq C_j \quad \forall j \in J \tag{5.8}$$

Die Zeitpunkte, zu denen neue Rohmaterialien eintreffen (Lieferzeitpunkte), sind die für die Materialüberprüfung entscheidenden Zeitpunkte T. Für jeden Zeitpunkt $\tau \in T$, sowie für jeden Auftrag $j \in J$ bezeichnet die Variable

$\rho_{j,\tau} \in \{0,1\}$, ob $\tau \in T$ als Zeitpunkt ausgewählt wird, zu dem alle Materialien für die Produktion von j und seiner ausgewählten Vorgänger zur Verfügung stehen sollen. Dabei ist für jeden Auftrag genau ein solcher Zeitpunkt auszuwählen, siehe Gleichung (5.6). Die für die Vorgänger und den Auftrag selbst benötigten Materialien haben dann einen direkten Einfluss auf diese Auswahl, wie die Ungleichungen (5.7) verdeutlichen. Es dürfen nur solche Lieferzeitpunkte ausgewählt werden, die einen ausreichenden Materialvorrat beinhalten. Der Auswirkung des Zeitpunktes der Materialverfügbarkeit auf die Fertigstellungszeit eines Auftrages wird dann in Ungleichung (5.8) Rechnung getragen.

Lemma 5.1
Das Modell mit den Variablen $C_j \geq 0, \pi_{i,j} \in \{0,1\}, \rho_{j,\tau} \in \{0,1\}$, den Nebenbedingungen (5.2) bis (5.8) und der Zielfunktion (5.1) bildet das betrachtete Schedulingproblem vollständig ab.

5.3 Weitere Nebenbedingungen

Die Ergebnisse der Vorbehandlung können durch die Fixierung von Variablen $\pi_{i,j}$ auf den Wert 1 oder 0 für gegebene beziehungsweise verbotene Vorrangbeziehungen in das Modell mit aufgenommen werden. Auch die Variablen C_j können durch die in der Vorbehandlung bestimmten Zeitfenster eingeschränkt werden. Um die Formulierung des Modells weiter zu verschärfen, können wir neben den Variablenfixierungen auch zusätzliche Nebenbedingungen einführen, die wiederum auf die Ergebnisse der Vorbehandlung zurückgreifen, oder die Techniken aus der Vorbehandlung auch in unserem Modell anwenden.

5.3.1 Maximale Positionen

$$\sum_{i \in J} \pi_{i,j} \leq p^{\max}(j) \forall j \in J. \tag{5.9}$$

Die Anzahl der Aufträge, die vor einem Auftrag j eingeplant werden darf nicht den Wert $p^{\max}(j)$ übersteigen. Dabei bezeichnet $p^{\max}(j)$ die maximale mögliche Position für Auftrag j in einem zulässigen Schedule, wie sie in der Vorbehandlung ermittelt wurde (siehe Abschnitt 4.3.1).

5.3.2 Bearbeitungszeit für die Vorgänger

Die Entscheidung, einen Auftrags $i \in J$ vor einem Auftrag $j \in J$ einzuplanen, kann die Mindestanzahl an notwendigen Rüstaktivitäten auf bestimmte Materialien vor der Produktion von j erhöhen. Dazu führen wir die Menge von kontinuierlichen Variablen $\sigma_{j,m} \geq 0$ mit $j \in J, m \in M$ ein, die die Anzahl der Rüstaktivitäten auf ein Material m vor der Fertigstellung von Auftrag j mit der Gruppenrüstzeit der Materialgruppe $g(m)$ gewichten. Sei $st(i,j,m)$ die Mindestanzahl von notwendigen Rüstaktivitäten auf Material m, gewichtet mit der Gruppenrüstzeit von $G(m)$, für den Fall dass Auftrag i vor Auftrag j fertiggestellt wird. Dieser Wert bildet einen Parameter, der vor der Optimierung bestimmt werden kann. Dann ergibt sich für die minimale durch Material m

induzierte Rüstzeit vor der Fertigstellung von Auftrag j folgende Nebenbedingung:

$$\sigma_{j,m} \geq st(i,j,m) \cdot \pi_{i,j} \tag{5.10}$$

Durch die Einbeziehung der Rüstvariablen $\sigma_{j,m}$ können wir dann eine weitere untere Schranke für die Fertigstellungszeit C_j von Auftrag $j \in J$ in Bezug auf die gewählten Vorgänger von j ableiten:

$$C_j \geq \sum_{i \in J} p_i \cdot \pi_{i,j} + \sum_{m \in M} st(G(m)) \cdot \sigma_{j,m} - \sum_{G \in \mathcal{G}} st(G) \tag{5.11}$$

Die Summe $\sum_{G \in \mathcal{G}} st(G)$ stellt dabei einen Korrekturterm dar, der der Beliebigkeit des initialen Rüstzustands geschuldet ist.

5.3.3 Beachtung der Vorrangbeziehungen

Obschon alle Vorrangbeziehungen im Grundmodell Beachtung finden, so können wir diese auch für die Definition weiterer gültiger Ungleichungen heranziehen. Wenn ein Auftrag i vor einem Auftrag j eingeplant werden soll, dann müssen auch alle Vorgänger i' von i mit $i' \prec i$ vor dem Auftrag j eingeplant werden:

$$\pi_{i,j} \leq \pi_{i',j} \tag{5.12}$$

5.3.4 Verschachtelte Mengen

Andererseits hat auch die Entscheidung, den Auftrag i vor einem anderen Auftrag j einzuplanen, Auswirkung auf die Nachfolger j' von j mit $j \prec j'$. Denn dann muss i eben auch vor j' eingeplant werden:

$$\pi_{i,j} \leq \pi_{i,j'} \tag{5.13}$$

Die Menge von eingeplanten Vorgängern von j ist somit immer eingebettet in die Menge der Vorgänger von j'.

5.4 Zeitindizierte Formulierung

Eine der ersten zeitindizierten Formulierungen eines Job-Shop-Problems ist in Bowman (1959) gegeben. In dieser Formulierung geben die Variablen für jede Operation eines Auftrags für jeden Zeitpunkt an, ob diese durchgeführt wird oder nicht. Die ununterbrochene Ausführung von Operationen wird dabei durch zusätzliche Nebenbedingungen sichergestellt.

Für das Ein-Maschinen-Scheduling wird in Sousa und Wolsey (1992) dagegen ein einfacheres Modell angegeben, dass die Modellierung einiger Zielfunktionen und die Einbettung von Zeitfenstern sowie Ausfallzeiten von Maschinen auf einfache Art ermöglicht. Hierbei sei $x_{j,t} \in \{0,1\}$ eine binäre Variable, die angibt, ob Auftrag $j \in J$ zum Zeitpunkt t gestartet wird oder nicht. Dabei werden alle Zeitpunkte bis zum Ende des Zeithorizontes T betrachtet. Die Variablendefinition lässt direkt eine Modellierung der Zielfunktion der gewichteten Fertigstellungszeit $\sum_j w_j \cdot C_j$ zu, indem für jede Variable $x_{j,t}$ der Zielfunktionskoeffizient

$c_{j,t} := w_j \cdot (t + p_j)$ definiert wird. Damit erhalten wir die Formulierung der Zielfunktion als

$$\min \sum_{j \in J} \sum_{t=0}^{T} c_{j,t} \cdot x_{j,t}. \tag{5.14}$$

Der erst Satz von Nebenbedingungen stellt sicher, dass jeder Auftrag genau einmal gestartet wird:

$$\sum_{t=1}^{T} x_{j,t} = 1 \forall j \in J \tag{5.15}$$

Der zweite Satz von Nebenbedingungen stellt die Ausführung maximal eines Auftrages pro Zeiteinheit sicher:

$$\sum_{j \in J} \sum_{\tau=t}^{t+p_j} x_{j,\tau} \leq 1 \forall t = 0, \ldots, T \tag{5.16}$$

Die Einbettung von Freigabe- und Fälligkeitszeitpunkten (Deadlines) ist durch die Fixierung von Variablen $x_{j,t}$ auf den Wert 0 problemlos möglich, sollte ein Start zum Zeitpunkt t vor dem Freigabezeitpunkt oder eine Fertigstellung zum Zeitpunkt $t + p_j > \delta_j$ verspätet sein. Die von Sousa und Wolsey (1992) gewählte Herleitung des Modells aus dem RCPSP motiviert dazu, auch die bis jetzt noch nicht abgebildeten Aspekte unseres Problems aus der Literatur zum RCPSP und seinen Erweiterungen (siehe Hartmann und Briskorn 2010, für einen Überblick) zu übernehmen. Um die Vorrangbeziehungen zwischen Paaren von Aufträgen $(i, j) \in J \times J$ mit $i \prec j$ abzubilden, muss die Startzeit des Nachfolgers j nach der Startzeit des Vorgängers i liegen, und Fertigungs- und Rüstzeit sind zu berücksichtigen (siehe hierzu auch das Modell für das RCPSP in Brucker und Knust 2006, S. 52, Nebenbedingung (2.39), dort in Abhängigkeit der Fertigstellungszeiten formuliert):

$$\sum_{t=0}^{T} t \cdot x_{j,t} - \sum_{t=0}^{T} (t + st(i, j) + p_i) \cdot x_{i,t} \geq 0. \tag{5.17}$$

Die Überprüfung der Materialverfügbarkeit lässt sich dadurch abbilden, dass der Materialverbrauch aller bis zu einem Zeitpunkt t gestarteten Aufträge nicht die bis zum Zeitpunkt t angelieferte Materialmenge überschreiten darf, vergleiche hierzu das Modell in Shirzadeh Chaleshtarti, Shadrokh und Fathi (2014).

$$\sum_{\tau=1}^{t} \sum_{j \in J} a_{j,m} \cdot x_{j,\tau} \leq R_{m,t} \forall m \in M, t = 0, \ldots, T \tag{5.18}$$

Um reihenfolgeabhängige Rüstzeiten korrekt abzubilden, müssen wir für jedes Paar von Aufträgen und jeden Zeitpunkt sicherstellen, dass der Abstand der Startpunkte der beiden Aufträge groß genug ist (Nogueira, Carvalho und Ravetti 2014).

$$x_{i,t} + \sum_{\tau=\max\{r_j, t-p_j-st(j,i)\}}^{\min\{t+p_i+st(i,j), \delta_i - p_i + 1\}} x_{j,\tau} \leq 1 \forall i, j \in J, i \neq j, t \in \{r_j, \ldots, \delta_j - p_j + 1\} \tag{5.19}$$

Dadurch, dass die Anzahl der Aufträge zum Quadrat und multipliziert mit allen Zeitpunkten eingeht, führt vor allem diese Klasse von Nebenbedingungen zu einem kaum zu bewältigenden Optimierungsmodell. Eine Anwendungsmöglichkeit von zeitindizierten Formulierungen werden wir in Kapitel 7 ansprechen.

5.5 Weitere Modellierungsmöglichkeiten

Ein weiterer schon in der Frühzeit der ganzzahligen Optimierung vorgestellter Modellierungsansatz für Schedulingprobleme ist Gegenstand des Artikels Wagner (1959). Hier wird in den wesentlichen Variablen über die Position eines Auftrags im Schedule entschieden.

Für Schedulingprobleme auf einer Maschine mit reihenfolgeabhängigen Rüstzeiten und Freigabezeiten vergleicht der Artikel von Nogueira, Carvalho und Ravetti (2014) die verschiedenen Modellierungstechniken in einem Literaturüberblick. Eine für die in Nogueira, Carvalho und Ravetti (2014) betrachtete Problemstellung anwendbare alternative Formulierung zur Vermeidung von Teilzyklen aus Eijl (1995) und Maffioli und Sciomachen (1997) vermeidet die Anwendung der BigM-Technik, die in den Nebenbedingungen (5.5) verwendet wird. Diese Modellierungsansätze haben allerdings den Nachteil, dass die Überprüfung der Materialverfügbarkeit nicht mehr einfach darzustellen ist, da sich die Bedeutung der $\pi_{i,j}$-Variablen ändert.

Ein weiteres Modell für $1|seq-dep|\sum w_jC_j$ lässt sich aus dem Modell von Claus (1984) für das Problem des Handelsreisenden (TSP) ableiten. Hierbei handelt es sich um eine Formulierung auf Basis disaggregierter Flüsse (Gouveia und Pires 2001). Neben den für TSP-Modelle klassischen Binärvariablen $x_{i,j}$, die eine direkte Folge des Auftrags j auf den Auftrag i ausdrücken, werden hier zur Vermeidung von Subtouren weitere Binärvariablen $y_{i,j,k}$ eingeführt, die die direkte Folge des Auftrags j auf den Auftrag i auf dem Weg zu Auftrag k ausdrücken. Dabei darf eine Variable $y_{i,j,k}$ nur dann ausgewählt werden, wenn die Variable $x_{i,j}$ auf den Wert 1 gesetzt ist. Zusätzlich zu den Gradungleichungen für die $x_{i,j}$-Variablen muss über die $y_{i,j,k}$-Variablen ein $0-k$-Fluss mit Wert 1 für jeden Knoten k bei künstlichem Startknoten 0 erzeugt werden. In Rocha, Soares und Fern (2005) werden die $y_{i,j,k}$-Variablen verwendet, um die Zielfunktion für $1|seq-dep|\sum w_jC_j$ zu berechnen. Der Kostenkoeffizient $c_{i,j,k}$ dieser Variablen in der Zielfunktion des Modells von Claus (1984) ist dann über $c_{i,j,k} = w_k \cdot (st(i,j) + p_j)$ gegeben. Diese Modellformulierung impliziert eine Zerlegung des Problems in die Bestimmung von Flüssen für jeden einzelnen Auftrag $k \in J$ auf der einen Seite und die Berechnung eines Assignments auf der anderen Seite, und motivierte den Autor der vorliegenden Arbeit zu ersten Ideen für den im Kapitel 7 vorgestellten Optimierungsansatz.

Kapitel 6

Ein Branch-and-Bound-Ansatz

Die Komplexität des betrachteten Schedulingsproblems schließt die Existenz eines Polynomialzeitalgorithmus für das Problem höchstwahrscheinlich aus. Anstelle nun mit einem Standardverfahren wie der Ganzzahligen Programmierung das Problem zu lösen, können wir auch versuchen, das Problem mit speziell zugeschnittenen Algorithmen in den Griff zu bekommen. Um das Problem für eine gegebene Instanz optimal zu lösen, müssen wir einerseits die beste Lösung finden, und andererseits zeigen, dass es keine bessere Lösung geben kann. Wenn wir wirklich alle Lösungen der Reihe nach erzeugen, und die Zielfunktionswerte der Lösungen betrachten, können wir unser Ziel erreichen.

Durch die Existenz der Vorrangbeziehungen in Form von Ketten für die Produkte entspricht die maximale Anzahl der zu erzeugenden Schedules dem Multinomialkoeffizienten

$$\frac{|J|!}{\prod_{C\in\mathcal{C}}|C|!}. \tag{6.1}$$

Auch für kleine Instanzen liefert diese Abschätzung noch große Werte, und so zielen wir nun darauf ab, den Enumerationsprozess aller Lösungen abzukürzen.

Unser hier gewähltes Mittel ist die Entwicklung eines Branch-and-Bound-Verfahrens. Obwohl der Begriff durch den Artikel von Little u. a. (1963) geprägt wurde, ist die Idee für das Verfahren schon älter (siehe Land und Doig 1960). Wie wir in Kapitel 5 gesehen haben, lässt sich das Verfahren auch auf ganzzahlige Optimierungsmodelle anwenden (Dakin 1965).

Die Grundidee des Verfahrens besteht darin, durch einen Enumerationsbaum eine Partitionierung des Lösungsraums vorzunehmen. Jeder Knoten des Baumes repräsentiert eine Teilmenge aller Lösungen der vorliegenden Instanz. Wird ein Knoten verzweigt (der erste Teil von *Branch*-and-Bound), wird der durch diesen Knoten repräsentierte Lösungsraum so auf die Kindknoten aufgeteilt, dass die Vereinigung der durch die Kindknoten repräsentierten Mengen von Lösungen wieder der Menge der durch den Vaterknoten repräsentierten Lösungen entspricht. So ist sichergestellt, dass kein Teil des Lösungsraumes verloren geht. Jeder Kindknoten stellt an die repräsentierten Lösungen mindestens eine weitere Anforderung zusätzlich zu den Anforderungen, die durch den Vaterknoten und die weiteren Ahnen gegeben sind. Alle Blattknoten des Baumes zusammen erge-

ben immer den gesamten Lösungsraum. Um nun das Verfahren gegenüber einer vollständigen Enumeration des Lösungsraumes durch schrittweise Konstruktion aller Schedules zu beschleunigen, können die folgenden Techniken eingesetzt werden:

Zulässigkeitsüberprüfung: Die durch einen Kindknoten repräsentierte Lösungsmenge kann leer sein, wenn die Anforderungen, die an die Lösungen durch die Verzweigungsentscheidungen gestellt werden, unerfüllbar sind. In unserem Fall kann es vorkommen, dass wir durch die Einplanung von Aufträgen die Fälligkeit eines weiteren Auftrages verletzen. Erkennen wir frühzeitig eine derartige Unzulässigkeit, müssen wir den Knoten nicht weiter verzweigen.

Dominanzkriterien: Die von einen Knoten repräsentierte Menge von Lösungen kann hinsichtlich des erreichbaren optimalen Zielfunktionswertes nicht besser sein als die von einem anderen Knoten repräsentierte Lösungsmenge. Dadurch müssen die Lösungen des ersten Knotens gar nicht mehr berücksichtigt werden.

Schranken an das Optimum: Jede zulässige Lösung und damit jeder zulässige Schedule liefert uns eine obere Schranke für den optimalen Zielfunktionswert der Instanz, da der Zielfunktionswert der optimalen Lösung nicht schlechter sein kann als der Zielfunktionswert einer beliebigen zulässigen Lösung. Andersherum können wir auch versuchen, eine Annäherung von unten an den optimalen Zielfunktionswert aller durch einen Knoten repräsentierten Lösungen zu berechnen, eine sogenannte untere Schranke. Stellen wir fest, dass alle durch einen Knoten repräsentierten Lösungen mindestens genauso teuer sind, wie eine bereits bekannte Lösung, so müssen wir uns keine Mühe mehr machen, den Knoten weiter zu verzweigen, da wir sowieso keine bessere Lösung erhalten können. Somit haben wir aufgrund der Schranke (dem zweiten Teil von Branch-and-*Bound*) einen Teil des Lösungsraumes von der weiteren Betrachtung ausschließen. Durch den Informationsgewinn durch eine Verzweigung rückt auch die untere Schranke immer näher an den tatsächlich erzielbaren Zielfunktionswert[1].

An dieser Stelle wird nun auch deutlich, wieso wir uns in Kapitel 4 mit Eigenschaften zulässiger Lösungen und der Ermittlung guter zulässiger Lösungen auseinandergesetzt haben. Während eine gute Kenntnis der Eigenschaften zulässiger Lösungen die Zulässigkeitsüberprüfung effizienter macht, können wir mit guten zulässigen Lösungen und damit guten oberen Schranken in Zusammenspiel mit den unteren Schranken den Enumerationsbaum verkleinern.

Um nun einen Branch-and-Bound-Ansatz für das vorliegende Problem zu erzeugen, müssen wir uns nun Gedanken darüber machen, wie wir die einzelnen Aspekte implementieren. Neben den eben genannten Zulässigkeitstests, Dominanzkriterien und der Schrankenbestimmung müssen wir uns auch klar darüber werden, wie wir den Lösungsraum überhaupt aufteilen wollen. Da bei der Aufteilung eines Lösungsraumes zwangsweise immer mehrere Kindknoten entstehen,

[1] Eine Ausnahme können hier Schranken bilden, die nicht aus der optimalen Lösung einer Relaxation des Problems resultieren, so zum Beispiel bei Anwendung von Subgradientenverfahren oder Spaltengenierungsansätzen, die in der Regel vor Erreichen des tatsächlichen Optimums abgebrochen werden. In diesem Fall kann aber die Schranke des Vaterknotens übernommen werden

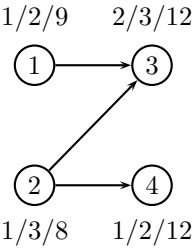


Abbildung 6.1: Branch-and-Bound-Instanz

wir die Kindknoten aber nur sequentiell weiter verzweigen können, müssen wir uns dann auch Gedanken darüber machen, in welcher Reihenfolge die aktuellen Blattknoten des Baumes verzweigt werden sollen. Somit ist die Agenda für das vorliegende Kapitel vorgegeben.

Zur Veranschaulichung des für das vorliegende Schedulingproblem entwickelten Branch-and-Bound-Ansatzes wollen wir nun die in Abbildung 6.1 dargestellte Instanz mit vier Aufträgen betrachten. Die Beschriftung der einzelnen Knoten und Aufträge gibt die Werte für das Gewicht, die Prozesszeit und die Fälligkeit der einzelnen Aufträge an, so gilt für den ersten Auftrag $w_1 = 1$, $p_1 = 2$ und $\delta_1 = 9$, die Pfeile verdeutlichen die Vorrangbeziehungen. Die Aufträge 1 und 3 sind dem ersten Produkt zugeordnet, die Aufträge 2 und 4 sind dem zweiten Produkt zugeordnet, beim Wechsel zwischen den Produkten tritt eine Rüstzeit von einer Zeiteinheit auf. Der entstehende Enumerationsbaum ist in Abbildung 6.2 dargestellt, die Blattknoten des Baumes stellen dann die fünf möglichen Schedules für diesen Vorranggraphen dar. In jedem Knoten ist der durch den Knoten ν repräsentierte Teilschedule mit seinem Makespan $t(\nu)$ und seinem Zielfunktionswert $c(\nu)$ in der Form von $t(\nu)/c(\nu)$, sowie die untere Schranke notiert. Allerdings sind die Schedules $(1, 2, 3, 4)$ und $(2, 1, 4, 3)$ unzulässig, da insgesamt zu viel Zeit auf die Rüstvorgänge aufgewandt wird, und somit die zuletzt eingeplanten Aufträge erst nach ihrer Fälligkeit fertiggestellt werden. Die verwendete untere Schranke wurde durch Anwendung der Regel von Smith (1956) ohne Beachtung von Rüstzeiten, Fälligkeiten und Vorrangbeziehungen erzielt.

6.1 Literatur zu Branch-and-Bound-Verfahren

Die Entwicklung von Branch-and-Bound-Verfahren für Ein-Maschinen-Scheduling-Probleme ist in der Literatur weit verbreitet, solange die Aussicht darauf besteht, mit Hilfe dieser Technik noch optimale Lösungen erzielen zu können. In diesem Abschnitt sollen nun zwei Ansätze diskutiert werden, die wesentlich die Entwicklung des hier vorliegenden Ansatzes beeinflusst haben.

Chou, Wang und Chang (2009) widmen sich in ihrem Artikel der Minimierung der Summe der gewichteten Fertigstellungszeiten bei Vorliegen von reihenfolgeabhängigen Rüstzeiten und Freigabezeitpunkten, also dem Problem $1|s_{i,j}, r_j|\sum w_j C_j$. Zur optimalen Lösung des Problems wird ein Branch-and-Bound Verfahren mit sehr einfach gehaltenen Schranken vorgeschlagen. Die erste Schranke vernachlässigt die Freigabezeitpunkte und größtenteils die Rüstzeiten, und mit der Regel von Smith (1956) eine untere Schranke. Für eine Schranke mit Beachtung der Freigabezeiten werden alle Aufträge gleichzeitig zu ihrem Freigabezeitpunkt gestartet (die Maschinenkapazität bleibt also unberücksichtigt),

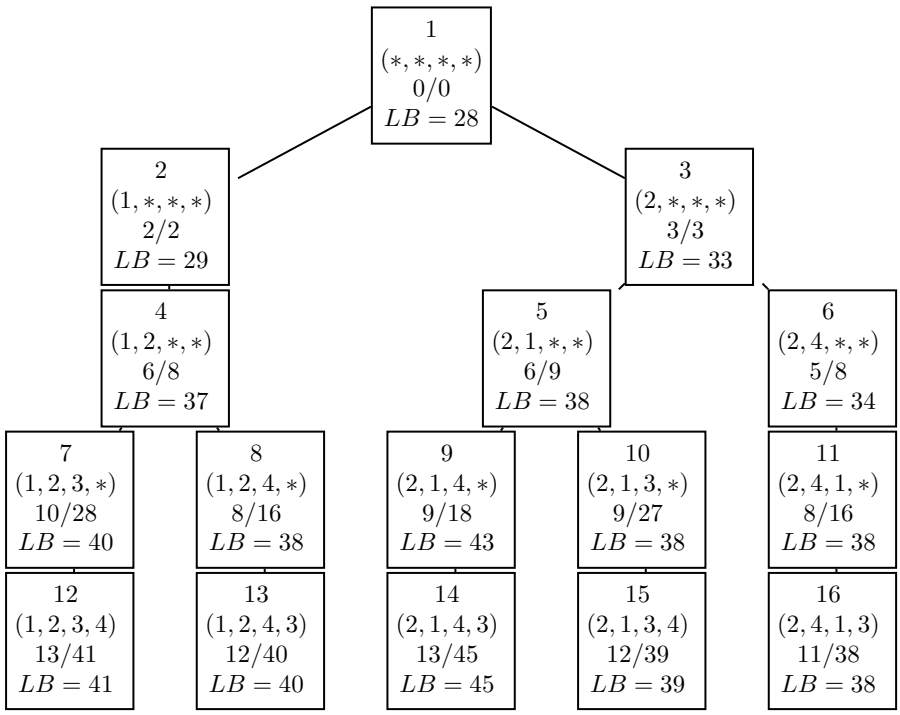


Abbildung 6.2: Branch-and-Bound Beispiel

und die daraus folgende gewichtete Fertigstellungszeit berechnet. Die Idee, die Rüstzeit zumindest teilweise zu erfassen, wurde diesem Ansatz entlehnt, wiewohl wir im hier vorgestellten Ansatz noch mehr die Struktur der Rüstzeiten ausnutzen werden. Die Verwendung von nur sehr einfachen Schranken im Ansatz von Chou, Wang und Chang (2009) ist jedoch nicht nachvollziebar.

Die Minimierung der Summe der gewichteten Auftragsverspätungen bei Vorliegen von auf jeden Fall einzuhaltenden Zeitfenstern und zusätzlich vorliegenden Vorrangbeziehungen wird im Artikel von Davari u. a. (2015) behandelt. Aufgrund der verwendeten Lagrangerelaxation wird in den unteren Schranken des vorgestellten Branch-and-Bound-Verfahrens auch mit Problemen mit der Zielfunktion der Minimierung der Summe der gewichteten Fertigstellungszeiten gearbeitet. Anders als im vorher erwähnten Ansatz wird hier auch auf Schranken eingegangen, die Freigabezeitpunkte und andere Aspekte des Optimierungsproblems berücksichtigen. Diese werden wir auch in diesem Ansatz verwenden, und werden darum im Abschnitt 6.6 näher betrachten. Zur Berücksichtigung von Vorrangbeziehungen wird die Reduktion auf einen VSP-Vorranggraphen mittels des Verfahrens von McMahon und Lim (1993) verwendet. Da in diesem relativ neuen Ansatz auf diese alte Methode zurückgegriffen wurde, sind im Rahmen dieser Arbeit weitere Möglichkeiten zur Reduktion der Vorrangbeziehungen auf einen VSP-Vorranggraphen untersucht worden.

Bei der Literaturrecherche zu Branch-and-Bound-Ansätzen fällt auf, dass die Betrachtung der Materialverfügbarkeit für die gegebene Zielfunktion bisher nicht in den Fokus der Forschung geraten ist.

6.2 Knotenauswahlstrategien

Die Auswahl des nächsten aktiven Knotens, der weiter verzweigt werden soll, hat einen großen Einfluß auf die Performanz eines Branch-and-Bound-Verfahrens (Ibaraki 1976). Im Folgenden werden wir drei Strategien näher betrachten. Alle Knoten, die aufgrund ihrer zu niedrigen unteren Schranke noch verzweigt werden müssen, um den Optimalitätsbeweis zu erbringen, bezeichnen wir hier als *aktive* Knoten.

6.2.1 Tiefensuche

Wird die Tiefensuche verwendet, wird immer einer der aktiven Knoten verzweigt, der in der tiefsten Ebene liegt. Der Vorteil liegt in der schnellen Erzeugung von zulässigen Lösungen, allerdings kann sich das Verfahren auch in ineffizienten Teilen des Lösungsraums stecken bleiben. In dem Fall, dass nicht alle Lösungen zulässig sind, und die Unzulässigkeit der bisherigen Entscheidungen erst spät erkannt wird, kann sich somit auch der eben genannte Vorteil des Verfahrens ins Gegenteil verkehren. Die Anzahl der gleichzeitig aktiven Knoten ist hier beschränkt durch $\mathcal{O}(|J| \cdot |\mathcal{C}|)$, also dem Produkt aus maximaler Tiefe des Baumes und dem maximalen Verzweigungsgrad. Dies ermöglicht es auch, den Speicherbedarf eines Branch-and-Bound-Verfahrens konstant zu halten, sofern die vollständig erforschten Knoten konsequent gelöscht werden.

6.2.2 Bestensuche

Bei diesem Auswahlschema wird immer derjenige Knoten verzweigt, der die niedrigste untere Schranke aller aktiven Knoten aufweist. Dadurch wird der Fokus auf den Anstieg der globalen unteren Schranke gelegt, um den Optimalitätsbeweis schnellstmöglich durchzuführen. Sobald der vom Auswahlschema zurückgegebene Knoten eine untere Schranke aufweist, die nicht kleiner ist als die Kosten des besten bekannten Schedules, ist die Optimalität des besten bekannten Schedules bewiesen.

6.2.3 Zyklische Bestensuche (Cylic Best First Search)

Diese Auswahlstrategie wird in der Dissertation von Kao (2008) eingeführt und ist eine Kombination von Tiefen- und Bestensuche. Anstelle immer den Knoten mit der global kleinsten unteren Schranke auszuwählen, wählen wir immer den Knoten mit der kleinsten unteren Schranke aller aktiven Knoten in einer bestimmten Ebene des Baumes. Diese Auswahlebene wird nach jeder Entnahme eines Knoten erhöht. Befindet sich in einer Ebene des Baumes kein aktiver Knoten, wird die Ebene übersprungen, und solange zur nächsten Ebene weitergegangen, bis letztendlich ein Knoten entnommen werden konnte. Wird die maximale Ebene des Baumes überschritten, wird wieder in der kleinsten Ebene gestartet. Dies führt dazu, dass Knoten mit geringen unteren Schranken auch über mehrere Ebenen weiterentwickelt werden. Andererseits kann, wenn die Kinder eines Knotens eine deutlich schlechtere Schranke aufweisen, auf bessere Knoten aus der Nachfolgeebene ausgewichen werden.

Angewendet wurde diese Auswahlstrategie unter anderem auch zur Minimierung der Gesamtverspätung bei Vorliegen von reihenfolgeabhängigen Rüstzeiten

(Sewell, Sauppe u. a. 2012) oder bei Vorliegen von Freigabezeitpunkten (Kao, Sewell und Jacobson 2009). Mit Hilfe dieser Auswahlstrategie lässt sich auch für das Fließbandabstimmungsproblem SALBP-1 eine Effizienzsteigerung (Sewell und Jacobson 2012) gegenüber SALOME (Scholl und Klein 1997) realisieren.

6.2.4 Implementierung

Um alle drei soeben vorgestellten Auswahlstrategien im Branch-and-Bond-Verfahren je nach aktueller Situation dynamisch anwenden zu können, wird die Verwaltung der aktiven Knoten folgendermaßen organisiert: Für jede Ebene, also mögliche Position p in einem Schedule wird ein Heap H_p für die Knoten angelegt. Zur Implementierung der Bestensuche bilden die einzelnen Heaps $H_p, p = 0, \ldots, |J|$ einen weiteren Heap H_A, der immer das Minimum über alle Heaps H_p bereithält. Für die Tiefensuche wird die aktuell tiefste Ebene bei jeder Einfüge- und Löschoperation aktualisiert. Im Rahmen der vorliegenden Implmentierung werden auch in der Tiefensuche alle Kindknoten eines ausgewählten Knotens sofort erzeugt, so dass bei ausschließlicher Anwendung der Tiefensuche immer der Kindknoten mit der niedrigsten unteren Schranke als erstes verzweigt wird. Für die CBFS-Strategie wird die nächste Entnahmeposition nach jeder Entnahme eines Knotens inkrementiert und der nächste nicht leere Heap H_p gesucht.

Im Beispiel aus Abbildung 6.2 werden in der Tiefensuche zunächst die Knoten 1, 2, 4, 8 verzweigt, bis Knoten 13 die die obere Schranke von $UB = 40$ liefert. Knoten 7 muss dann nicht mehr verzweigt werden, so dass der Auswahlprozess Knoten 3, 6 und 11 auswählt.

Wird das Beispiel mit der Bestensuche durchgeführt, werden die Knoten in der Reihenfolge 1, 2, 3, 6, 4 verzweigt, ehe einer der Knoten mit der unteren Schranke von 38 verzweigt wird. Wird hier Knoten 11 gewählt, findet das Verfahren die optimale Lösung und kann sofort enden, ansonsten müssen noch weitere Verweigungen durchgeführt werden. Dieses Beispiel verdeutlicht die Notwendigkeit des Vorhandenseins einer guten oberen Schranke, die hier sofort zur Löschung alle Knoten mit einer unteren Schranke von 38 führen würde.

Die zyklische Bestensuche würde zunächst die Knoten 1, 2, 4 und 8 verzweigen, um dann die Knoten 3, 6 und 11 zu verzweigen. Im Beispiel ist die Verwandtschaft mit der Tiefensuche deutlich sichtbar, obwohl der Neustart in Knoten 3 hier aufgrund der Auswahlregel erfolgt, und nicht durch die Eigenschaften der Instanz bedingt ist.

6.3 Verzweigungsregeln

Die Konstruktion des Branch-and-Bound-Baums in diesem Verfahren basiert auf der Erzeugung von Auftragsreihenfolgen, die Schritt für Schritt um je einen zusätzlichen Auftrag erweitert werden. Aufgrund der notwendigen Materialverfügbarkeitsüberprüfung und zur besseren Integration der Fälligkeiten wird der Schedule streng von vorne nach hinten aufgebaut. In der Literatur gibt es Beispiele, in denen der Schedule vom Ende aus beginnend, beziehungsweise von Anfang und Ende aus aufgebaut wird (Davari u. a. 2015). In den Verzweigungen vom Ende muss dann aber viel mit Abschätzungen gearbeitet werden (siehe zum Beispiel Sewell, Sauppe u. a. 2012), während in Vorwärtsrichtung die tatsäch-

liche Fertigstellungszeit und damit der Beitrag zum Zielfunktionswert bekannt ist.

Der Wurzelknoten repräsentiert im vorliegenden Ansatz mit σ_0 den initialen Rüstzustand, ein Knoten ν auf der k-ten Ebene des Baumes einen partiellen Schedule $\sigma(\nu) = (\sigma_0, \ldots, \sigma_k)$ mit k Aufträgen. Zugleich werden von einem Knoten ν damit auch alle vollständigen Schedules repräsentiert, die mit dieser partiellen Reihenfolge $\sigma(\nu)$ beginnen. Der im Knoten ν angehängte Auftrag $j \in J$ wird über $j(\nu)$ identifiziert, $t(\nu)$ entspricht der Fertigstellungszeit von $j(\nu)$ in der Reihenfolge $\sigma(\nu)$, und $c(\nu)$ seien die Kosten von $\sigma(\nu)$. In einem beliebigen Knoten sei $J(\nu)$ die Menge der Aufträge, die bereits eingeplant sind, und $J'(\nu)$ sind die Aufträge, die gültige Nachfolger von $j(\nu)$ darstellen, und somit an den Schedule $\sigma(\nu)$ angehängt werden können. Wird nun ein Knoten ν verzweigt, wird jedem Kindknoten $\nu_1, \ldots, \nu_{|J'(\nu)|}$ ein der Aufträge aus $J'(\nu)$ zugeordnet, und der partielle Schedule wird entsprechend erweitert.

6.3.1 Wiederherstellung der Teillösung

Immer wenn wir einen neuen Knoten ν verzweigen wollen, müssen wir zunächst die durch den Knoten repräsentierte Teillösung $\sigma(\nu)$ wiederherstellen und unter anderem den aktuellen Materialbedarf ermitteln. Für diese Wiederherstellung betrachten wir alle Ahnen inklusive des zu verzweigenden Knotens ν. Sofern ein dominierter Ahne vorliegt, kann die Verzweigung des Knotens unterblieben.

6.3.2 Ermittlung der gültigen Verzweigungen

Die Menge der Aufträge $J'(\nu)$, die an den Auftrag $j(\nu)$ im Schedule $\sigma(\nu)$ angehängt werden können, wird über die folgenden Kriterien bestimmt:

Topologische Sortierung: Damit für einen Auftrag $j \in J$ auch $j \in J'(\nu)$ gelten kann, müssen bereits alle Vorgänger von j in $J(\nu)$ enthalten sein, es gilt also $i \in J(\nu) \forall i \prec j$.

Minimale Position: Weiterhin muss die in der Vorbehandlung ermittelte minimale Position $p_j^{\min}$, also die minimale Position von j in einem zulässigen Schedule erreicht worden sein. Das ist dann der Fall, wenn $|J(\nu)|+1 \geq p_j^{\min}$ gilt.

Direkte Folge: Auch die direkte Folge von Auftrag j auf Auftrag $j(\nu)$ darf durch die Vorbehandlung nicht ausgeschlossen worden sein.

6.4 Dominanzkriterien

Um weitere Knoten aus dem Baum schneiden zu können, kommen in kombinatorischen Branch-and-Bound-Verfahren häufig sogenannte Dominanzkriterien zum Einsatz:

Definition 6.1 (Dominanz (Kohler und Steiglitz 1974))
Ein Knoten ν dominiert *einen Knoten ν', wenn jeder vollständige kostenminimale Schedule, der aus dem Teilschedule $\sigma(\nu)$ hervorgeht, nicht schlechter ist als jeder kostenminimale Schedule, der sich aus dem Teilschedule $\sigma(\nu')$ von ν konstruieren lässt.*

Ibaraki (1977) untersucht den Einfluss der Dominanzkriterien in Branch-and-Bound-Verfahren auf die Effizienz des Algorithmus bei Verwendung unterschiedlicher Knotenauswahlstrategien, und kommt zu dem Schluss, dass in der Bestensuche immer das strengstmögliche Dominanzkriterium verwendet werden sollte. Die Anwendung von Dominanzkriterien in Branch-and-Bound-Verfahren für die Ganzzahlige Programmierung ist erst in jüngster Zeit in den Fokus der Forschung geraten, siehe hierzu den Artikel von Fischetti und Salvagnin (2010). Im hier betrachteten Branch-and-Bound-Ansatz kommen nun zwei Dominanzkriterien zum Einsatz, die in den folgenden Abschnitten vorgestellt werden.

6.4.1 Pareto Dominanz

Das erste Dominanzkriterium ergibt sich aus folgendem Lemma:

Lemma 6.2
Die Menge der bereits eingeplanten Aufträge $J(\nu)$ des Knotens ν entspreche der Menge der Aufträge $J(\nu')$ von ν', und es gelte $j(\nu) = j(\nu')$, also auch die zuletzt eingeplanten Aufträge stimmen überein. Dann ist ν' von ν dominiert wenn für die Kosten der Knoten beziehungsweise Teilschedules $c(\nu') \geq c(\nu)$ und gleichzeitig für die Endzeiten $t(\nu') \geq t(\nu)$ gilt.

Beweis. Da uns durch frühere Fertigstellung eines Auftrags kein Nachteil entsteht, kann die von Knoten ν repräsentierte verbleibende Instanz des Problems keinen schlechtere optimale Lösung besitzen als die durch den Knoten ν' repräsentierte Instanz. Durch die Übereinstimmung des letzten Auftrages gehen wir auch beide Male vom gleichen initialen Rüstzustand aus. Da die von ν repräsentierte Teillösung einen besseren (oder gleichen) Zielfunktionswert aufweist wie ν', können wir sicher sein, dass auch die von ν repräsentierten Lösungen des Originalproblems nicht schlechter sind als die Lösungen, die durch den Knoten ν' repräsentiert werden. □

Sollten zwei Knoten, die die Voraussetzung des Lemmas erfüllen, sowohl in Kosten als auch Zeiten übereinstimmen, so ist darauf zu achten, dass nur einer der Knoten aus dem Branch-and-Bound-Baum entfernt wird. In dem hier vorgestellten Verfahren dominiert dann immer der zuerst erzeugte Knoten den später erzeugten, um die Korrektheit des Verfahrens zu gewährleisten. Das Dominanzkriterium aus Lemma 6.2 ist in der Literatur weit verbreitet und findet zum Beispiel auch Anwendung in einem Ansatz der Dynamischen Programmierung für das TSP mit Zeitfenstern (Dumas u. a. 1995) und für das TSP mit kumulativer Zielfunktion (auch als TRP bekannt) in Bianco, Mingozzi und Ricciardelli (1993). Auch ein früher Ansatz der Dynamischen Programmierung (Held und Karp 1962) für Schedulingprobleme basiert auf der Betrachtung der bereits eingeplanten Mengen von Aufträgen. In Sewell, Sauppe u. a. (2012) wird diese Verwandtschaft mit der Dynamischen Programmierung dadurch deutlich gemacht, dass der dort vorgestellte Branch-and-Bound-Ansatz als Branch-and-Bound-and-Remember-Verfahren bezeichnet wird.

In Abbildung 6.3 wird das Dominanzkriterium für vier verschiedene Knoten, die in der eingeplanten Menge von Aufträgen sowie den durch die Knoten repräsentierten Aufträgen übereinstimmen, verdeutlicht. Nur Knoten, die bezüglich der Kosten und Zeit pareto-effiziente Teilschedules im Sinne der multikriteriellen Optimierung repräsentieren, sind nicht dominiert.

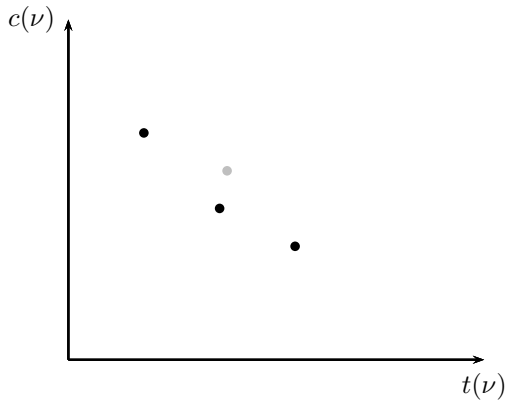


Abbildung 6.3: Pareto Dominanz

6.4.2 Dominanz der oberen Schranke

Lemma 6.3
Wenn die Menge der Aufträge s in einem Knoten ν mit $LB(\nu) \geq UB$, mit der Menge der Aufträge in einem anderen Knoten ν' übereinstimmt, und es gilt $j(\nu') = j(\nu)$, dann ist ν' von ν dominiert, sobald $t(\nu) \leq t(\nu')$ und $c(\nu') + LB(\nu) - c(\nu) \geq UB$ gilt.

Wir können also davon ausgehen, dass auch im von Knoten ν' repräsentierten Teil des Lösungsraums keine bessere Lösung als die aktuelle obere Schranke UB enthalten ist.

Beweis. Da die Zielfunktion regulär ist, die spätere Fertigstellung eines Auftrags also keinen Vorteil bietet, liefert uns eine Instanz des Schedulingproblems für die verbleibenden Aufträge des Knotens ν', die bis auf einen späteren Startzeitpunkt mit der Instanz des Knotens ν identisch ist, auf jeden Fall eine höhere gewichtete Gesamtfertigstellungszeit für die Aufträge aus $J \backslash J(\nu)$. Wäre das nicht der Fall, könnte auch die Instanz des Knoten ν mit einem besseren Zielfunktionswert gelöst werden. □

Diese Dominanzregel ermöglicht es uns, auf die Schrankenberechnung zu verzichten, sobald wir schon einmal eine Instanz dieses Problems mit gleichem initialen Rüstzustand (garantiert durch die Bedingung $j(\nu') = j(\nu)$) und einem nicht späteren Anfangszeitpunkt (garantiert durch $t(\nu) \leq t(\nu')$) gelöst haben.

In Abbildung 6.4 haben wir für den linken Teilschedule bereits die obere Schranke berechnet, und festgestellt, dass die aktuelle obere Schranke überschritten wird. Für den mittleren Teilschedule müssen wir nun die Schranke nicht mehr neu berechnen, da uns die Schranke des linken Teilschedule nun schon eine kommende Überschreitung der oberen Schranke anzeigt. Der rechte Teilschedule ist allerdings zu günstig, um mit der Schranke des linken Teilschedules eliminiert zu werden, deswegen muss für diesen nun doch noch eine untere Schranke berechnet und damit geprüft werden, ob der Kostenvorteil den späteren Produktionsstart für die weiteren Aufträge aufwiegen kann.

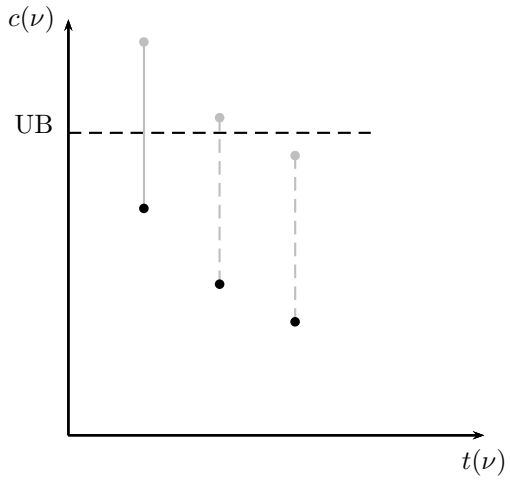


Abbildung 6.4: Dominanz der oberen Schranke

6.4.3 Implementierung

Um auf die Existenz von dominierenden Knoten zu prüfen, verwenden wir eine sogenannte Dominanztabelle[2], in der alle bis jetzt erzeugten Teilmengen $J(\nu) \subset J$ für alle bereits evaluierten Knoten ν gespeichert sind. Da die Anzahl aller Teilmengen einer Menge der Kardinalität n sich nur durch $\mathcal{O}(2^n)$ (bei Vorliegen von k verschiedenen Ketten lässt sich diese Abschätzung nur auf $\mathcal{O}(n^k)$ reduzieren) abschätzen lässt, ist es ab einer gewissen Größe der Instanz unmöglich, genug Speicherplatz für alle möglichen Teilmengen bereitzustellen. Deshalb wird in der Implementation der Dominanztabelle nur Speicherplatz für eine feste Anzahl von Teilmengen mit Endknoten reserviert, der dann nach dem First-Come-First-Serve-Prinzip an die tatsächlich auftretenden Mengen $J(\nu)$ vergeben wird, solange dies möglich ist. Tritt Speichermangel auf, kann das Dominanzkriterium nicht angewendet werden, die Korrektheit des Verfahrens wird aber nicht gefährdet. Um nun zu einem Knoten ν mit enthaltener Auftragsmenge $J(\nu)$ einen anderen Knoten ν' mit $J(\nu') = J(\nu)$ zu finden, wird ein Array von AVL-Bäumen (siehe zum Beipiel Cormen u. a. 2009) verwendet. Jeder Baum im Array beinhaltet Auftragsmengen von gleicher Kardinalität. Die Einträge in den Bäumen werden durch eine lexikographische Sortierung der Kardinalitäten der Schnittmengen $J(\nu) \cap C_k, k = 1, \ldots, |\mathcal{C}|$ geordnet. Zusätzlich muss für die Anwendbarkeit der Dominanzkriterien auch der letzte Auftrag übereinstimmen, deswegen wird zu jeder Menge $J(\nu)$ ein Array von einfach verketteten Listen gepflegt. Jede Liste im Array hinterlegt die Zeiten $t(\nu)$ und Kosten $c(\nu)$ der Branch-and-Bound-Knoten, die im zuletzt betrachteten Auftrag der Menge übereinstimmen.

Im Beispiel aus Abbildung 6.2 und in allen Instanzen können die Dominanzkriterien erst mit Teilschedules ab einer Länge von drei Aufträgen eingesetzt werden, da es bei kürzeren Teilschedules keine zwei verschiedenen Knoten geben kann, die in der betrachteten Menge der Aufträge und im letzten Auftrag übereinstimmen. Für das soeben erwähnte Beispiel besteht hier auf der

[2]Der Begriff ‚Tabelle' deutet schon auf den Wunsch hin, dass die Datenstruktur möglichst effizient sein soll.

dritten Ebene des Baumes eine Dominanz des Knotens 8 über den Knoten 9, da hier für übereinstimmende Mengen von Aufträgen mit gleichem Endauftrag $t(8) = 8 \leq t(9) = 9$ und $c(8) = 16 \leq c(9) = 18$ erfüllt ist, sowie des Knotens 10 über Knoten 7, da hier $t(10) = 9 \leq t(7) = 10$ und $c(10) = 27 \leq c(7) = 28$ erfüllt ist. Das hier ausgerechnet die unzulässigen Schedules dominiert werden, ist Zufall. Anhand des Beispiels wird auch deutlich, dass die in Dominanzrelation zueinander stehenden Knoten sich in gänzlich unterschiedlichen Teilbäumen des Verfahrens befinden können, so dass die Überprüfung des Dominanzkriteriums über die soeben beschriebene Dominanztabelle erfolgen muss.

6.5 Zulässigkeitstests

Aufgrund der Fälligkeiten δ_j für jeden Auftrag $j \in J$ kann es passieren, dass wir für den aktuellen durch einen Knoten ν repräsentierten Teilschedule schon zu viel Zeit verschwendet haben, um noch zu einer zulässigen Lösung gelangen zu können, die diesen Teilschedule $\sigma(\nu)$ mit einschließt. Deshalb ist es in unserem Interesse, schon für möglichst kurze Teilschedules festzustellen, ob noch eine zulässige Lösung möglich ist. Im Folgenden werden hierfür zwei erste Kriterien vorgestellt, weitere Zulässigkeitsvoraussetzungen können bei der Berechnung von unteren Schranken überprüft werden.

6.5.1 Überprüfung der Fälligkeiten

Wir wollen überprüfen, ob auch alle weiteren Aufträge pünktlich eingeplant werden können. Ein notwendiges Kriterium ist hierfür, dass die optimale Lösung des Optimierungsproblems $1|prec|L_{\max}$ angewendet auf die verbliebenen Jobs und mit initialer Zeit $t(\nu)$ einen Zielfunktionswert von $L_{\max} \leq 0$ hat. Dabei ist $L_{\max} = \min\{L_j, j \in J\}$ und $L_j = C_j(\sigma) - \delta_j$, wobei die Fertigstellungszeit vom betrachteten Schedule σ abhängig ist. Die Lösung dieses Problems ist mit Hilfe des rückwärtsgerichteten Dynamischen Programmierungsansatz von Lawler (Lawler 1973) in $\mathcal{O}(|J|^2)$ möglich. Der Algorithmus kann abgebrochen werden, sobald der Job, der laut Prioritätsregel eingeplant werden soll, eine Verspätung aufweist. Da es uns aber nicht um die Minimierung der maximalen Verspätung geht, sondern nur um die Frage, ob alle Aufträge unter Beachtung der Vorrangbeziehungen eingeplant werden können, können wir uns auf ein früheres Resultat von Lawler und Moore für das Zulässigkeitsproblem stützen:

Satz 6.4 (Lawler und Moore (1969), Abschnitt 2)
Für jede Vorrangbeziehung zwischen zwei Aufträgen $i \prec j$ sei die Bedingung $\delta_i < \delta_j$ für deren Fälligkeitszeitpunkte erfüllt. Dann gilt: Jeder Auftrag kann unter Beachtung der Vorrangbeziehungen genau dann pünktlich gefertigt werden, wenn er auch in der Sequenzierung nach aufsteigendem Fälligkeitszeitpunkt (EDD) pünktlich fertiggestellt werden kann. [3]

Beweis: Wenn die Sortierung nach aufsteigender Fälligkeit zulässig ist, haben wir offensichtlich eine zulässige Reihenfolge gefunden. Für die andere Richtung ist nun zu zeigen, dass das Vorhandensein einer beliebigen zulässigen

[3] Der originale Satz verwendet statt der ursprünglichen Fälligkeitszeitpunkte δ_j modifizierte Zeitpunkte $\bar{d}_j$, mit $i \prec j \Rightarrow \bar{d}_i < \bar{d}_j$. Diese werden zunächst aus den originalen Fälligkeitszeitpunkten d_j errechnet.

Reihenfolge auch der Zulässigkeit der Sortierung nach Fälligkeitszeitpunkten entspricht. Sei nun eine weitere zulässige Reihenfolge gegeben, und es gebe zwei direkt aufeinander folgende Aufträge i und j mit $d_i \geq d_j$, die EDD-Regel ist also verletzt. Dann gilt $i \not\prec j$ gemäß Voraussetzung und die beiden Aufträge können vertauscht werden. Diese Vertauschung hat keine negative Konsequenz für die Zulässigkeit der Reihenfolge, wenn i und j weiterhin pünktlich fertiggestellt werden, die Fertigstellungszeiten der übrigen Aufträge sind unbeeinflusst von dieser Vertauschung. Wenn wir j nun direkt vor i setzen, wird j auf keinen Fall später fertiggestellt, es gilt also für die Fertigstellungszeit von j nach der Vertauschung $\tilde{C}_j \leq C_j \leq \delta_j$. Vor der Vertauschung gilt $C_j \leq \delta_j \leq \delta_i$ und die Fertigstellungszeit von i nach der Vertauschung $\tilde{C}_i$ entspricht der Fertigstellungszeit C_j vor der Vertauschung. Somit gilt $\tilde{C}_i = C_j \leq \delta_j < \delta_i$, also wird auch i in der neuen Sequenz weiterhin pünktlich fertiggestellt. Jede zulässige Reihenfolge lässt sich durch eine endliche Anzahl solcher Vertauschungen auf die Sortierung nach Fälligkeit zurückführen, die damit eine zulässige Lösung des Problems darstellt. □

Dieser Satz kann für unseren Test angewandt werden, da die Vorbehandlung unserer Instanz immer dafür sorgt, dass für $i \prec j$ auch $\delta_i < \delta_j$ erfüllt ist. Zur Effizienzsteigerung können wir schon vor Start des Branch-and-Bound-Algorithmus die Aufträge nach Fälligkeit aufsteigend sortieren. Da wir in $\mathcal{O}(1)$ überprüfen können, ob ein Auftrag bereits eingeplant wurde, ist der Zulässigkeitstest für die übrigen Aufträge dann in $\mathcal{O}(|J \backslash J(\nu)|)$ realisierbar. Die Sortierung ist offensichtlich unabhängig von der tatsächlichen Prozesszeit der Aufträge, die deshalb dynamisch um eine gültige Unterschätzung der Rüstzeit erweitert werden kann.

6.5.2 Heiratssatz von Hall (1935)

Auch mit der in Abschnitt 4.3.2 vorgestellten Regel können wir die zulässige Erweiterbarkeit der Teilschedules in linearer Zeit bezüglich der verbliebenen Aufträge überprüfen.

6.6 Berechnung von unteren Schranken

In einem Knoten ν des Branch-and-Bound-Baumes ist der Beitrag $c(\nu)$ der bereits eingeplanten Aufträge $J(\nu)$ zum Zielfunktionswert bekannt. Um jedoch diesen Knoten ν aus dem Baum herausschneiden zu können, weil der Zielfunktionswert aller durch den Knoten repräsentierten Lösungen nicht besser ist als die beste bis jetzt bekannte Lösung, müssen wir den optimalen Zielfunktionswert aller Lösungen, die sich ausgehend vom Teilschedule $\sigma(\nu)$ erzeugen lassen, möglichst gut abschätzen.

Zum Erreichen dieses Zieles untersuchen wir nun mehrere Möglichkeiten. Zunächst einmal können wir Nebenbedingungen relaxieren, also komplett oder teilweise weglassen und uns nur auf Teilaspekte des Problems konzentrieren. So können wir durch den Zielfunktionswert der Lösung des Problems $1||\sum w_j C_j$ mit der Regel von Smith (1956) immer eine gültige untere Schranke berechnen, indem wir den β-Teil der $\alpha|\beta|\gamma$-Notation von Graham u. a. (1979) zunächst gänzlich vernachlässigen. In den hier betrachteten unteren Schranken werden wir dann aber versuchen, den β-Teil auszubauen, und relaxierte Aspekte in anderer Form in die Schranken mit aufzunehmen. Ein weiteres Konzept ist die

Ableitung eines vom Charakter her zunächst anderen Optimierungsproblems, das wir aus unserem Optimierungsproblem erstellen. Der optimale Zielfunktionswert dieses Problems muss dann eine gültige Unterschätzung des optimalen Zielfunktionswertes des Ausgangsproblems liefern. Ein Beispiel für dieses Vorgehen ist in dieser Arbeit die auf dem linearen Zuordnungsproblem basierende untere Schranke aus Abschnitt 6.6.6.

6.6.1 Betrachtung von Vorrangbeziehungen

Für die Berechnung dieser Schranke ziehen wir uns auf die Vorrangbeziehungen zurück, die wir einerseits durch die Bedingung, Aufträge eines Produktes in festgelegter Reihenfolge zu fertigen und andererseits aus den Zeitfenstern ableiten können. Das entstehende Optimierungsproblem $1|prec|\sum w_jC_j$ ist für beliebige Vorrangbeziehungen $\mathcal{NP}$-schwer, wie in Lawler (1978) und Lenstra und Kan (1978) gezeigt wurde.

In Kapitel 3 haben wir uns dieser Thematik ausführlich gewidmet, als untere Schranken setzen wir auf Klassen von Vorrangbeziehungen, für die das Problem polynomiell lösbar ist. Im hier vorgestellten Branch-and-Bound-Ansatz setzen wir den Algorithmus von Sidney (1975) auf die Partitionierung der Auftragsmenge in Ketten und den Algorithmus von Lawler (1978) auf einen VSP-Graphen, der aus den Vorrangbeziehungen ermittelt wurde, an. Beide Vorranggraphen der betrachteten Klasse erstellen wir zu Beginn des Branch-and-Bound-Verfahrens, für einzelne Knoten ν des Baumes ist dann das jeweilige Verfahren bei Nichtbeachtung der bereits erledigten Aufträge $J(\nu)$ durchzuführen. Für den VSP-Vorranggraphen setzen wir hierzu auf die Transformation in einer Activity-on-Arc-Netzwerk nach Kamburowski, Michael und Stallmann (2000) und wenden dann den Reduktionsalgorithmus von W. W. Bein, Kamburowski und Stallmann (1992) zur Erstellung des VSP-Zerlegungsbaumes an.

6.6.2 Betrachtung von Fälligkeiten

Wir betrachten das $\mathcal{NP}$-schwere (Lenstra, Kan und Brucker 1977) Problem $1|\delta_j|\sum w_jC_j$. In den Artikeln Potts und Wassenhove (1983), Posner (1985) und Pan (2003) werden jeweils verschiedene Branch-and-Bound-Verfahren für dieses Problem entwickelt. Während in Potts und Wassenhove (1983) die untere Schranke über die Lagrange-Relaxation der Fälligkeiten bestimmt wird, baut Posner (1985) die von Smith (1956) vorgeschlagene Heuristik für das Problem zu einer unteren Schranke um, indem Aufträge auch aufgeteilt werden können. Diese Schranke dominiert die Schranke aus Potts und Wassenhove (1983). In Pan (2003) wird keine weitere Schranke entwickelt, sondern mit einer verstärkten Anwendung von Dominanzkriterien eine Verbesserung des Verfahrens von Posner (1985) erzielt.

Da die Prozesszeiten bekannt sind und keine Freigabezeitpunkte vorliegen, beginnt die Schrankenberechnung zum Zeitpunkt $P := t(\nu)+\sum_{j\in J\setminus J(\nu)} p_j$. Zum Zeitpunkt P können alle Aufträge aus der Menge $E = \{i \in J\setminus J(\nu) : \delta_i \geq P\}$ zulässig beendet werden. Aus der Menge E wird dann immer der Auftrag j mit der niedrigsten Priorität w_j/p_j ausgewählt, um als letztes mit Startzeitpunkt $P - p_j$ eingeplant zu werden. Bei der Einplanung des Auftrags j überprüfen wir nun die Menge $T' := \{i \in J\setminus J(\nu) : P - p_j < \delta_i < P \wedge w_i/p_i < w_j/p_j\}$. Das sind alle Aufträge aus T, die im Intervall $]P - p_j, P[$ fällig sind und eine

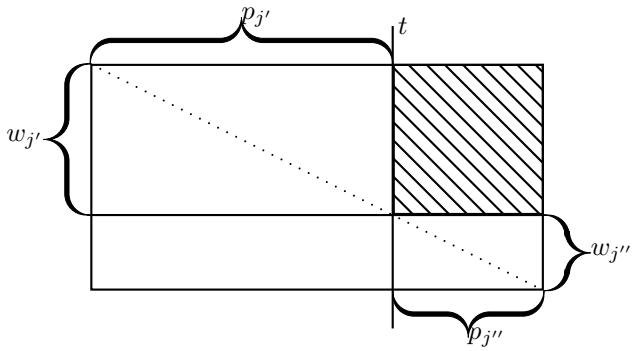


Abbildung 6.5: Aufteilung eines Auftrags

niedrigere Priorität als j besitzen. Ist die Menge T' nicht leer, wissen wir, dass wir denjenigen Auftrag $r \in T'$, der am spätesten fällig ist, aufgrund seiner niedrigeren Priorität zu seiner Fälligkeit δ_r fertigstellen sollten, und den Auftrag j deshalb nicht vollständig einplanen können. Der Auftrag j wird deshalb zunächst in zwei Teile j' und j'' geteilt, deren Priorität gleich der Priorität von j ist, für die Prozesszeit des hinteren Teils j'', der sofort eingeplant wird, gilt $p_{j''} = P - \delta_r$, für den noch offenen Rest j' gilt $p_{j'} = p_j - P + \delta_r$. Dieser Rest j' wird erst im weiteren Verlauf des Verfahrens eingeplant, und bildet jetzt einen eigenständigen Auftrag mit Gewicht $w_{j'} = w_j \cdot p_{j'}/p_j$. Wird ein Auftrag oder ein Teil eingeplant, wird die Endzeit P des verbleibenden Schedules angepasst, und die Menge E auf Basis der neuen Endzeit P entsprechend erweitert.

Die untere Schranke wird nun nicht nur durch die gewichtete Fertigstellungszeit aller (Teil-)Aufträge gebildet, sondern bei jeder Aufteilung eines Auftrags j in zwei Teilaufträge j' und j'' lässt sich die Schranke um den Wert $w_{j'} \cdot p_{j''}$ verschärfen. Diese „Kosten der Aufteilung“ (Posner 1985) lassen sich grafisch mit Hilfe der zweidimensionalen Gantt-Charts aus Goemans und Williamson (2000) sehr schön veranschaulichen. In Abbildung 6.5 finden sich die Kosten der Aufteilung im oberen rechten Quadranten des ursprünglichen konstanten Beitrags des Auftrags j zum Zielfunktionswert wieder. Wird der Auftrag aufgeteilt, so geht diese Fläche in jedem für das neue Problem zulässigen Schedule, der j' vor j'' einplant, unwiederbringlich verloren und wird deshalb als Konstante auf die untere Schranke addiert.

6.6.3 Betrachtung von Freigabezeitpunkten

Wir betrachten das $\mathcal{NP}$-schwere (Lenstra, Kan und Brucker 1977) Problem $1|r_j|\sum w_j C_j$. Branch-and-Bound-Verfahren für dieses Problem werden unter anderem beschrieben in Hariri und Potts (1983), Belouadah, Posner und Potts (1992) und Nessah und Kacem (2012). Während in Hariri und Potts (1983) die Schranke auf einer Lagrange-Relaxation basiert, wird in Belouadah, Posner und Potts (1992) das Job-Splitting-Konzept von Posner (1985) aus Abschnitt 6.6.2 verallgemeinert. Ein verallgemeinertes Splitting hebt die Bedingung auf, dass die Priorität der Teilaufträge gleich der Priorität des eigentlichen Auftrags ist. Da im Falle von Preemption nur der letzte Teil des Auftrags mit

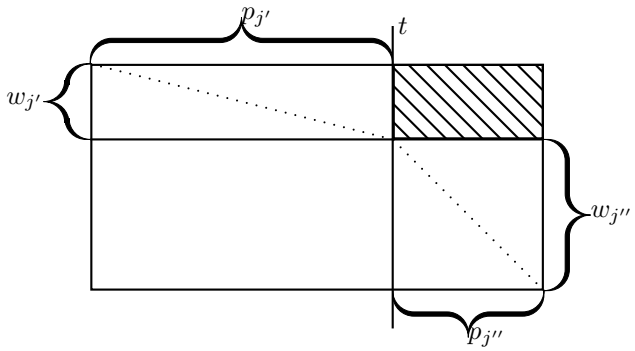


Abbildung 6.6: Verallgemeinerte Aufteilung eines Auftrags

dem Gesamtgewicht des Auftrags zum Zielfunktionswert beiträgt, kann diese Technik auch als Verallgemeinerung der Schranke durch Preemption betrachtet werden. Weiterhin gilt, dass das preemptive Problem auch die bestmögliche Job-Splitting-Schranke erzielt. Das preemptive Problem $1|r_j, prmp|\sum w_j C_j$ ist in diesem Fall aber auch $\mathcal{NP}$-schwer (Labetoulle u. a. 1984), und damit bietet sich die Verwendung von Job-Splitting-Schranken an, die zwar nicht die Qualität der preemptiven Schranke erreichen können, aber in Polynomialzeit berechenbar sind. Das Vorgehen von Belouadah, Posner und Potts (1992) verfährt nun genau wie das Verfahren von Posner (1985), nur in vorwärts gerichteter Reihenfolge. Von den aktuell verfügbaren Aufträgen wird immer der Auftrag $j \in J$ mit dem größten Verhältnis w_j/p_j eingeplant. Sollte während der Ausführung des Auftrags ein weiterer Auftrag $k \in J$ verfügbar werden, der eine höhere Priorität hat $(w_k/p_k > w_j/p_j)$, so wird Auftrag $j \in J$ in zwei Teile j' und j'' gesplittet. Der erste Teil wird bis zum Zeitpunkt r_k ausgeführt, damit dann der Auftrag k gestartet werden kann, der zweite Teil wird zur späteren Ausführung zurückgestellt. Beide Teilaufträge werden auch hier mit einem eigenen Gewicht versehen.

Unterschieden wird hier zwischen einer einfachen Aufteilung eines Auftrags (Simple Split) und einer verallgemeinerten Aufteilung (General Split) eines Auftrags. Während der Simple Split für alle Teile die Priorität des Ausgangsauftrags beibehalten wird, wird im General Split versucht, ein möglichst großes Gewicht auf den hinteren Teil des Auftrags zu legen (siehe Abbildung 6.6). Dabei ist dann zu beachten, dass das Gewicht für den ersten Teil so gewählt wird, dass die Priorität des ersten Teils immer noch groß genug ist, um auch an dieser Stelle eingeplant zu werden. Für den zweiten Teil jedoch darf das Gewicht nicht so groß werden, dass der zweite Teil schon früher hätte eingeplant werden müssen. Um diese Bedingungen einzuhalten, werden nun die Mengen der Konkurrenten für diese zwei Teilstücke untersucht, und das Gewicht entsprechend bestimmt (Belouadah, Posner und Potts 1992).

In Nessah und Kacem (2012) werden etwas aufwändigere Schranken verwendet, die aber, ausgehend von den experimentellen Ergebnissen des Artikels, keinen entscheidenden Vorteil gegenüber den Schranken aus Belouadah, Posner und Potts (1992) besitzen.

6.6.4 Kombinierte Betrachtung von Freigabezeitpunkten und Fälligkeiten

Für das Problem $1|r_j, \delta_j|\sum w_j C_j$ entwickeln Pan und Shi (2005) ein Branch-and-Bound-Verfahren. Die in Pan und Shi (2005) verwendete Schranke basiert auf der Multiplier-Adjustment-Methode von Wassenhove (1979) und verwendet einige Resultate aus der Arbeit von Hariri und Potts (1983). Dabei werden die Multiplikatoren λ einer Lagrange-Relaxation so berechnet, dass die mit Hilfe einer Heuristik konstruierte Lösung σ eine optimale Lösung der Lagrange-Relaxation $L(\lambda)$ ist. Diese Anpassungstechnik wurde auch auf weitere Schedulingprobleme angewendet, siehe Potts und Wassenhove (1983) für $1|\delta_j|\sum w_j C_j$, Potts und Wassenhove (1985) für $1||\sum w_j T_j$ und Hariri und Potts (1983) für $1|r_j|\sum w_j C_j$. Für die hier verwendete Schranke wird ein verzögerungsfreier Schedule σ mit Hilfe einer kombinierten Heuristik erzeugt, die zunächst die Freigabezeiten beachtet, und dann an zwei Stellen auf das Vorliegen von Fälligkeiten Rücksicht nimmt. Einerseits werden Aufträge mit hoher Dringlichkeit in Hinblick auf die Fälligkeit bevorzugt eingeplant, andererseits wird bei Verfügbarkeit aller verbleibender Aufträge die rückwärtsgerichtete Schedulingpolitik von Smith (1956) für das Problem $1|\delta_j|\sum w_j C_j$ angewendet. Der erzeugte Schedule σ muss hinsichtlich der Fälligkeiten nicht immer zulässig sein. Dieser Umstand gefährdet allerdings nicht die Korrektheit der sich daraus ergebenden unteren Schranke. Für die Bestimmung der Multiplikatoren a_j (aus der Lagrange-Relaxation der Freigabezeiten) und b_j (aus der Lagrange-Relaxation der Fälligkeiten) für jeden Auftrag ist zunächst eine Zerlegung des Schedules in Blöcke notwendig.

Definition 6.5 (Zerlegung in Blöcke)
Für einen Schedule σ ist ein Auftrag v_k der letzte Auftrag eines Blocks B_k, wenn für alle folgenden Aufträge i im Schedule gilt $r_i \geq C^{\sigma}_{v_k}$. Die Menge $\{\sigma(u_k), \ldots, \sigma(v_k)\}$ von direkt aufeinander folgenden Aufträgen im Schedule σ bildet einen Block, wenn $\sigma(u_k - 1)$ der letzte Auftrag eines Blockes ist oder $u_k = 1$ gilt, und keiner der Aufträge $u_k, \ldots, v_k - 1$ die Bedingung erfüllt, der letzte Auftrag eines Blocks zu sein.

Die Aufteilung einer Sequenz in Blöcke ist in $\mathcal{O}(n)$ möglich, wenn die Sequenz rückwärts durchlaufen wird und wir uns immer den minimalen Freigabezeitpunkt für den Rest der Sequenz merken.

Lemma 6.6 (Hariri und Potts (1983))
Ein Schedule σ ist optimal für $1|r_j|\sum w_j C_j$, wenn die Aufträge in jedem Block nach monoton absteigender Priorität w_j/p_j sortiert sind.

Das Lemma 6.6 und die Definition der Blöcke erlauben es uns nun, die Multiplikatorenbestimmung und Schrankenberechnung separat für jeden Block $B_k(\sigma)$ der Referenzsequenz σ durchzuführen. Da wir nun einen Schedule und ein Optimalitätskriterium für diesen Schedule vorliegen haben, passen wir die Multiplikatoren $a_j \geq 0$ und $b_j \geq 0$ für die Aufträge so an, dass die modifizierten Gewichte $w'_j = w_j - a_j + b_j$ und damit die Prioritäten aller Aufträge das Optimalitätskriterium aus Lemma 6.6 für diesen Schedule σ erfüllen. Um nun eine zulässige Schranke zu erzeugen, die dann möglichst nah am Optimum liegen sollte, haben wir nun ein Optimierungsproblem im Raum der Multiplikatoren vorliegen. Dazu leiten Pan und Shi (2005) zunächst eine weitere Bedingung für

die Multiplikatoren ab, nämlich kann zusätzlich $a_j \cdot b_j = 0$ gefordert werden, ohne die Qualität der Schranke zu beeinträchtigen. Dies ermöglicht es, die zwei Multiplikatoren a_j und b_j für jeden Auftrag durch eine einzige Variable x_j zu ersetzen, wobei $a_j = \max\{w_j - p_j \cdot x_j, 0\}$ und $b_j = \max\{p_j \cdot x_j - w_j, 0\}$ gesetzt wird. Der Wert der Unteren Schranke beläuft sich dabei auf

$$LBPS = -\bar{z} + \sum_{j=1}^{n} w_j C_j(\sigma) \tag{6.2}$$

mit $C_j(\sigma)$ als Fertigstellungszeit des Auftrags j in der Sequenz σ, die durch geeignete Wahl der Multiplikatoren zu einer optimalen Lösung gemacht werden soll. Damit ist der zweite Term als konstant anzusehen. Für den ersten Teil gilt

$$\bar{z} = \min \sum_{j=1}^{n} f_j(x_j) \text{ s. t. } x_{\sigma_1} \geq, \ldots, x_{\sigma_n} \tag{6.3}$$

mit

$$\begin{aligned} f_j(x_j) = p_j \cdot (&\max\{(C_j(\sigma) - r_j - p_j)(w_j/p_j - x_j), 0\} \\ &+ \max\{(\delta_j - C_j(\sigma))(x_j - w_j/p_j), 0\}). \end{aligned} \tag{6.4}$$

Diese Funktion $f_j(x_j)$ ist stückweise linear und konvex mit einziger Knickstelle w_j/p_j, und ihr Wert $f_j(x_j)$ muss von der gewichteten Gesamtfertigstellungszeit $w_j C_j(\sigma)$ eines Auftrags in der Referenzsequenz σ abgezogen werden. Das Optimierungsproblem über eine Summe solcher Funktionen bei Vorliegen der Bedingungen aus (6.3) wird als verallgemeinertes isotonisches Regressionsproblem bezeichnet (Ahuja und James B. Orlin 2001). Pan und Shi (2005) entwickeln nun einen Algorithmus, der die hier vorliegende Variante des Problems mit einer Laufzeit von $\mathcal{O}(n \cdot \log n)$ lösen kann, um damit dann auch die Schranke aus (6.2) in $\mathcal{O}(n \cdot \log n)$ zu bestimmen.

Die Schranke von Pan und Shi (2005) wird mit einer Technik aus Hariri und Potts (1983) noch weiter verschärft. Die ermittelten Multiplikatoren a_j für die Freigabezeiten werden Gegenstand einer geschickten Umformung. Betrachtet wird hierfür die Sortierung π der Aufträge eines Blockes nach monoton aufsteigenden Multiplikatoren a_j, es gilt also $a_{\pi(k)} \geq a_{\pi(k-1)}$ für $k > 1$. Dann sei $\tilde{a}_{\pi(1)} = a_{\pi(1)}$ und $\tilde{a}_{\pi(k)} = a_{\pi(k)} - a_{\pi(k-1)}$ für $k > 1$. Aufgrund der aufsteigenden Sortierung sind auch die Werte $\tilde{a}_j$ nicht negativ, und können selbst als Multiplikatoren einer weiteren Lagrangerelaxation aufgefasst werden. Während die ursprüngliche Lagrangerelaxation Ungleichungen, die nur auf einen Auftrag bezogen sind, betrachtet, geht es in der neuen Relaxation immer um einen Auftrag und alle seine Folgeaufträge in der Sortierung π. Mit den neuen Lagrangemultiplikatoren wird nun für den k-ten und alle folgenden Aufträge der Sortierung π die Summe $\sum_{k'=k}^{n}(r_{\pi(k')} + p_{\pi(k')}) \leq \sum_j C_j$ als untere Schranke für die Gesamtfertigstellungszeit bei Vorliegen von Freigabezeitpunkten als Lagrange-relaxierte Nebenbedingung betrachtet. Diese Summe ist somit eine untere Schranke für das Problem $1|r_j|\sum_j C_j$. Die untere Schranke selbst können wir dann verbessern, indem wir auf die optimale Lösung des Problems $1|r_j, prmp|\sum_j C_j$ zurückgreifen. Diese Problem kann mit Hilfe der SRPT-Regel (Vorgestellt in Schrage und L. W. Miller (1966) und Conway, Maxwell und L. W. Miller (2003), Beweis

in Schrage (1968)) in Polynomialzeit optimal gelöst werden. Hier soll bei Freigabe eines weiteren Auftrags immer derjenige Auftrag eingeplant werden, der noch die kürzeste verbleibende Prozesszeit aufweist. Die in diesem preemptiven Schedule ermittelte Fertigstellungszeit für einen Auftrag j sei mit C_j^{prmp} bezeichnet. Führen wir nun eine Lagrangerelaxation der verschärften Ungleichung mit gleichen Multiplikatoren $\tilde{a}_j$ durch, so stellen wir fest, dass die ursprüngliche Schranke um die Differenz

$$\sum_{k=1}^{n} \tilde{a}_{\pi(k)} \cdot \sum_{k'=k}^{n} (C_{\pi(k')}^{prmp} - r_{\pi(k')} - p_{\pi(k')}) \tag{6.5}$$

auf zulässige Weise verschärft wird (Hariri und Potts 1983).

Die Schranke von Pan und Shi (2005) lässt sich somit durch Anwendung von verschiedenen Prioritätsregeln auf verschiedenen Auftragsmengen bestimmen. Zunächst wird die heuristische Lösung bestimmt, in Blöcke zerlegt und dann das verallgemeinerte Regressionsproblem pro Block gelöst. Zur Verbesserung der Schranke wird dann insgesamt maximal $|J|$-mal auf die SRPT-Reihenfolge zurückgegriffen, nachdem die Aufträge pro Block gemäß der Multiplikatoren a_j sortiert worden sind. Die Anwendungen der SRPT-Regel kann unterbleiben, wenn $\tilde{a}_j = 0$ erfüllt ist, da dann keine Verbesserung für Auftrag j erzielt werden kann, wie Formel (6.5) zeigt.

6.6.5 Anpassung von Problemparametern

In den in obigen Abschnitten haben wir jeweils die Rüstzeiten, die Überprüfung der Materialverfügbarkeit und die Vorrangbeziehungen zumindest teilweise außer Acht gelassen. Die folgenden Überlegungen erlauben es uns dennoch, auch diese Aspekte in die den obigen Schranken zugrunde liegenden Probleme einfließen zu lassen.

6.6.5.1 Erweiterung von Prozesszeiten

Die Prozesszeit einer Teilmenge der verbliebenen Aufträge kann etwas vergrößert werden, wenn wir eine minimal benötigte Rüstzeit, die zur Fertigung eines einzelnen Auftrags notwendig ist, berücksichtigen.

Produktbasierte Rüstzeitabschätzung Auf jedes Produkt, zu dem es noch mindesten einen offenen Auftrag gibt, müssen wir auch mindestens einmal rüsten. Der Rüstvorgang erfolgt entweder vom Produkt des zuletzt eingeplanten Auftrags, oder von einem weiteren verbliebenen Produkt auf dieses Produkt. Wenn wir uns also bereits für einen neuen Vaterknoten ν entschieden haben, aber noch keinen weiteren Auftrag an den partiellen Schedule angehängt haben, sind alle Produkte von Aufträgen aus der noch offenen Menge $J \backslash J(\nu)$ nun Kandidaten für einen Rüstvorgang auf ein verbliebenes Produkt. Dabei gehen wir davon aus, dass zumindest auf den ersten Auftrag eines Produktes gerüstet werden muss, sofern dieses Produkt nicht das Produkt des zuletzt eingeplanten Auftrags ist. Für eine gültige Rüstzeitabschätzung müssen wir dann annehmen, dass die Rüstung auf dieses Produkt von dem Produkt aus der Menge der verbliebenen Produkte erfolgt, das die minimale Rüstzeit aufweist. Eine derartige Abschätzung ohne Auftragsfamilien beziehungsweise Produkte findet sich auch

im Branch-and-Bound-Ansatz von Chou, Wang und Chang (2009) für das Problem $1|r_j, s_{i,j}|w_jC_j$.

Materialbasierte Rüstzeitabschätzung Eine zweite Möglichkeit zur Einbindung von Rüstzeiten besteht darin, die Struktur der Rüstvorgänge zu Hilfe zu nehmen. Damit ein Auftrag produziert werden kann, muss die Maschine auf die entsprechenden Materialien des betroffenen Produktes gerüstet sein. Die Rüstzeit für ein Material wird nur dann fällig, wenn dieses Material nicht schon vorher gerüstet war. Auch hier betrachten wir die offene Menge von Aufträgen $J\backslash J(\nu)$, sobald ein neuer Vaterknoten ν ausgewählt wurde. Nun können wir für jeden verbleibenden Auftrag $j \in J\backslash J(\nu)$ und jedes Material $m \in M$ eine Bedingung ableiten, die uns garantiert, dass auf jeden Fall ein Rüstvorgang auf Material m direkt vor der Produktion von Auftrag j ausgeführt werden muss. Wenn es keinen weiteren verbleibenden Auftrag $i \in J\backslash J(\nu)$ gibt, der auch auf dieses Material angewiesen ist und direkt vor j platziert werden könnte, ist eine Rüstung auf m zur Produktion von j auf jeden Fall notwendig. Dann ist es für eine gültige untere Schranke zulässig, die Rüstzeit auf Material m auf die Prozesszeit p_j von j aufzuschlagen. Zur Implementierung dieses Kriteriums legen wir für jeden Auftrag und jedes im Produkt des Auftrags enthaltene rüstrelevante Material eine Liste von Aufträgen an, die einen Rüstvorgang auf diese Material **nicht** notwendig machen könnten, da sie auf das gleiche Material angewiesen sind und in einem Schedule direkt vor dem betrachteten Auftrag platziert werden könnten. Ist keiner dieser Aufträge mehr offen, erweitern wir die Prozesszeit dieses Auftrages entsprechend. Um die tatsächliche und asymptotische Laufzeit zu verringern, kann die Menge dieser Aufträge noch reduziert werden. Ein Auftrag wird nur dann in die Menge der zu überprüfenden Aufträge aufgenommen, wenn nicht auch noch ein Nachfolger dieses Auftrags in der Menge ist. Ansonsten wäre dieser Nachfolger in jedem Fall auch noch ein offener Auftrag, der den Aufschlag auf die Rüstzeit verhindern würde. Die asymptotische Laufzeit zur Bestimmung der erweiterten Prozesszeiten liegt damit bei $\mathcal{O}(|J\backslash J(\nu)| \cdot |\mathcal{C}| \cdot |\mathcal{G}|)$ für jeden verzweigten Vaterknoten ν.

Da die Rüstzeitabschätzung auf Basis der zu rüstenden Materialien für jeden Auftrag mindestens die Werte der Rüstzeitabschätzung auf Basis der Produkte erreicht, werden die tatsächlich auftretenden Rüstzeiten in der zweiten Variante besser abgeschätzt.

6.6.5.2 Verschärfung von Freigabezeitpunkten

In jedem Kindknoten können die Freigabezeitpunkte der einzelnen Aufträge angehoben werden, indem der aktuelle Endzeitpunkt des Teilschedules und der aktuelle Materialverbrauch berücksichtigt werden, um wie in Abschnitt 4.1.1 einen frühestmöglichen Freigabezeitpunkt für die Aufträge auf Basis von notwendigen Rüstvorgängen und Prozesszeiten sowie Materialverbrauch der verbliebenen Vorgänger zu berechnen. Auch die minimalen Positionen der noch offenen Aufträge in zulässigen Schedules lassen sich in diesem Zusammenhang anpassen. Bei Verwendung der auf diese Weise berechneten Freigabezeitpunkte darf die Prozesszeitanpassung aus Abschnitt 6.6.5.1 nicht angewandt werden, da sonst Rüstzeiten doppelt gezählt werden könnten. Gilt für den neuen Freigabezeitpunkt $r_j(\nu)$ eines Auftrages $j \in J\backslash J(\nu))$, dass die Ungleichung $r_j(\nu) + p_j > \delta_j$

erfüllt ist, wissen wir, dass von Knoten ν keine zulässige Lösung repräsentiert wird. Dadurch erhalten wir einen weiteren Zulässigkeitstest.

6.6.5.3 Lagrangerelaxation der Vorrangbeziehungen

Mit Hilfe der Lagrangerelaxation (Geoffrion 2010) können wir Vorrangbeziehungen, die in den geschilderten Schranken vernachlässigt worden sind, doch noch zu Berücksichtigung verhelfen. Für jede direkte Vorrangbeziehung $i \prec j$ wird ein Lagrange-Multiplikator $\mu_{i,j} \geq 0$ eingeführt, der die Verletzung der Nebenbedingung $C_j \geq C_i + p_j$ bestraft. Aus der Zielfunktion $\min \sum w_j \cdot C_j$ des betrachteten Optimierungsproblems wird durch diese Lagrangerelaxation

$$\min \sum_{j \in J} w_j \cdot C_j + \sum_{i \prec j} \mu_{i,j} \cdot (C_i + p_j - C_j). \tag{6.6}$$

Nun können wir die Zielfunktion derartig umformen, dass die Lagrange-Multiplikatoren das Gewicht der einzelnen Aufträge modifizieren und wir eine additive Konstante zum Zielfunktionswert erhalten. Für die modifizierten Auftragsgewichte $w_j(\mu)$ gilt dann

$$w_j(\mu) = w_j + \sum_{j \prec i} \mu_{j,i} - \sum_{i \prec j} \mu_{i,j} \tag{6.7}$$

und für die Konstante

$$c(\mu) = \sum_{i \prec j} \mu_{i,j} \cdot p_j, \tag{6.8}$$

so dass die neue Zielfunktion sich auch formulieren lässt als

$$\min c(\mu) + \sum_{j \in J} w_j(\mu) \cdot C_j = c(\mu) + \min \sum_{j \in J} w_j(\mu) \cdot C_j \tag{6.9}$$

Eine erste Anwendung dieser Lagrangerelaxation als untere Schranke für das Problem $1|prec|\sum w_j C_j$ wird in Potts (1985) beschrieben. Durch die Einführung der Lagrangerelaxation entsteht ein neues Optimierungsproblem, die Bestimmung von möglichst guten Multiplikatoren, die die Schranke möglichst nah an das tatsächliche Optimum heranführen. Eine Annäherung an gute Multiplikatoren, die direkt im Raum der Multiplikatoren durchgeführt wird, stellt Quick-Ascent-Direction-Verfahren von Velde (1995) dar. Ausgehend von den aktuellen Multiplikatoren werden der Reihe nach für jede in der Lagrangerelaxation betrachtete Vorrangbeziehung $i \prec j$ die relativen Gewichte $w'_i(\mu) = w_i(\mu)/p_i$ und $w'_j(\mu) = w_j(\mu)/p_j$ betrachtet. Gilt $w'_i(\mu) < w'_j(\mu)$, so würde die Lösung der Lagrangerelaxation durch die Regel von Smith (1956) die Vorrangbeziehung verletzen, und der Multiplikator $\mu_{i,j}$ wird erhöht. Gilt dagegen $w'_i(\mu) > w'_j(\mu)$, so kann der Multiplikator $\mu_{i,j}$ verkleinert werden, sofern weiterhin $\mu_{i,j} \geq 0$ erfüllt ist und auch mit den neuen Multiplikatoren dann $w'_i(\mu) \geq w'_j(\mu)$ gilt. Die Schrittweite für die Änderung der Multiplikatoren lässt sich jeweils in konstanter Zeit bestimmen. Ein wichtiges theoretisches Resultat in Velde (1995) ist die strikte Verbesserung der Lagrangerelaxation nach jeder Iteration des Verfahrens. Das Verfahren wird entweder nach einer bestimmten Anzahl an Iterationen I, oder wenn keine Veränderung eines Multiplikators mehr auftritt, beendet und liefert lediglich eine Näherung für die optimalen Multiplikatoren.

Die Schranke durch die Lagrangerelaxation kann weiter verbessert werden (Hoogeveen und Velde 1995), indem die Ungleichungen der Vorrangbeziehungen mit Hilfe von Schlupfvariablen zu Gleichungen transformiert werden, die Ungleichung $C_j \geq C_i + p_j$ also durch die Gleichung $C_j = C_i + p_j + y_{i,j}$ ersetzt wird. Die Schlupfvariable für die Vorrangbeziehung $i \prec j$ lässt sich dabei als Wartezeit des Auftrags j nach dem Freigabezeitpunkt C_i deuten. Hat ein Auftrag mehr als zwei direkte Vorgänger oder mehr als zwei direkte Nachfolger, tritt somit zwangsweise eine Wartezeit auf, und mindestens eine der involvierten Schlupfvariablen hat einen Wert > 0. Führen wir zunächst eine Lagrangerelaxation der neu entstandenen Gleichungen durch, wird die Zielfunktion zu

$$\begin{aligned} &\min \sum_{j \in J} w_j \cdot C_j + \sum_{i \prec j} \mu_{i,j} \cdot (C_i + p_j + y_{i,j} - C_j) \\ &= \sum_{j \in J} w_j(\mu) \cdot C_j + \sum_{i \prec j} \mu_{i,j} \cdot p_j + \sum_{i \prec j} \mu_{i,j} \cdot y_{i,j}. \end{aligned} \tag{6.10}$$

Die Zielfunktion setzt sich aus mehreren Teilen zusammen, die getrennt betrachtet werden können, um eine untere Schranke für das Problem abzuleiten. Der erste Teil repräsentiert das soeben behandelte Lagrangeproblem $\min \sum_{j \in J} w_j(\mu) \cdot C_j$, der mittlere Teil ist für ein gegebenes μ konstant und der letzte Teil bildet das sogenannte Schlupfvariablenproblem

$$\min\{\sum_{i \prec j} \mu_{i,j} \cdot y_{i,j} | y \in Y\}, \tag{6.11}$$

wobei Y die Menge aller zulässigen Schlupfvariablenbelegungen darstellen soll. Für das Schlupfvariablenproblem soll nun eine untere Schranke bestimmt werden. Erreicht diese untere Schranke einen Wert > 0, wird durch diese Vorgehensweise die Lagrangeschranke (6.10), die selbst den Schlupfvariablen zunächst nur den Wert 0 zuweisen würde, erhöht. Für einen Auftrag i mit einer Menge von Nachfolgern $\mathcal{S}_i$ mit $|\mathcal{S}_i| > 1$ betrachten wir das Schlupfvariablenproblem

$$F'(\mu, i) = \min\{\sum_{j \in \mathcal{S}_i} \mu_{i,j} \cdot y_{i,j} | y \in Y\}. \tag{6.12}$$

Dann können die Schlupfvariablen $y_{i,j}$ als die Startzeiten der Aufträge $j \in \mathcal{S}_i$ eines zulässigen Schedules für die Auftragsmenge $\mathcal{S}_i$ interpretiert werden. Damit kann das Problem auch wieder als Schedulingproblem $1||\sum_j w_j \cdot C_j$ aufgefasst werden, wobei die Multiplikatoren die Gewichte darstellen, und statt der Fertigstellungszeiten die Startzeiten der Aufträge Berücksichtigung finden. Dieses Schedulingproblem kann deshalb auch wieder mit der Prioritätsregel von Smith (1956) gelöst werden. Analog zum Problem für die Menge der Nachfolger (6.12) lässt sich auch für die Menge der Vorgänger $\mathcal{P}_j$ eines jeden Auftrags j das Schlupfvariablenproblem

$$F''(\mu, j) = \min\{\sum_{i \in \mathcal{P}_j} \mu_{i,j} \cdot y_{i,j} | y \in Y\}. \tag{6.13}$$

betrachten, und ebenso mit der Regel von Smith (1956) lösen. Hoogeveen und Velde (1995) zeigen, dass die Summe der optimalen Zielfunktionswerte der Optimierungsprobleme (6.12) und (6.13) für jeden Auftrag mit mehr als zwei Nachfolgern oder Vorgängern einer unteren Schranke für das ursprüngliche Schlupfvariablenproblem (6.11) entspricht.

6.6.6 Das lineare Zuordnungsproblem und Vorrangbeziehungen

In Kapitel 4 haben wir schon festgestellt, dass das Schedulingproblem auch als bipartites Matching von Positionen zu Aufträgen aufgefasst werden kann. Ordnen wir jedem Paar (j, p) aus Auftrag $j \in J$ und Position $p > p'$ ein Gewicht zu, das eine untere Schranke für die gewichtete Fertigstellungszeit $w_j \cdot C_{j,p}^{LB}$ von Auftrag j an Position p entspricht, erhalten wir das folgende lineare Zuordnungsproblem (Linear Assignment Problem LAP):

$$\min \sum_j \sum_p w_j C_{j,p}^{LB} x_{j,p}. \tag{6.14}$$

Der erste Satz von Nebenbedingungen fordert die Zuordnung einer Position zu jedem Auftrag:

$$\sum_p x_{j,p} = 1 \forall \quad j \in J \tag{6.15}$$

Der zweite Satz fordert die Zuordnung eines Auftrags zu jeder Position:

$$\sum_j x_{j,p} = 1 \forall \quad p \in \mathcal{P} \tag{6.16}$$

$$x_{j,p} \in \{0, 1\} \tag{6.17}$$

Jede optimale Lösung des Problems entspricht einer unteren Schranke für unser Schedulingproblem. Auch jede zulässige Lösung des zum linearen Zuordnungsproblem dualen Problems bildet aufgrund des schwachen Dualitätssatzes eine untere Schranke. Die Idee, diese Schranke zur Lösung von Sequenzierungsproblemen einzusetzen, kommt bereits in einem Ansatz von Kan, Lageweg und Lenstra (1975) für generelle Ein-Maschinen-Schedulingprobleme mit kumulativer regulärer Zielfunktion zur Anwendung.

6.6.6.1 Die Ungarische Methode

Zur Lösung dieses gewichteten bipartiten Matchingproblems wird die ungarische Methode angewendet. Dabei handelt es sich um ein Primal-Duales Verfahren (siehe auch Papadimitriou und Steiglitz 1998), das eine asymptotische Laufzeit von $\mathcal{O}(n^3)$ für n zuzuordnende Elemente hat (Kuhn 1955; Munkres 1957; Burkard, Dell'Amico und Martello 2009). Ausgehend von einer dual zulässigen Lösung wird schrittweise die Kardinalität des Matchings erhöht, bis auch tatsächlich jedem Element der einen Seite ein Element der anderen Seite zugeordnet ist und umgekehrt.

6.6.6.2 Berechnung der Kosten

Die Koeffizienten $w_j \cdot C_{j,p}^{LB}$ der Kostenmatrix müssen vor Anwendung der ungarischen Methode in einem eigenen Verfahren bestimmt werden. Dazu verwenden wir die Ansätze der Dynamischen Programmierung aus den Abschnitten 4.3.3 und 4.3.4.1. Auch die Verschärfung der Freigabezeitpunkte aus Abschnitt 6.6.5.2 liefert uns eine untere Schranke für die Fertigstellungszeit.

6.6.6.3 Erweiterung um Vorrangbeziehungen

Die mit der ungarischen Methode ermittelten Lösungen des Zuordnungsproblems müssen zunächst nicht die Vorrangbeziehungen der Jobs untereinander einhalten, es kann also vorkommen, dass für zwei Aufträge i und j mit $i \prec j$ für die den Aufträgen zugeordneten Positionen $p(i) < p(j)$ zutrifft. Um die Verletzung von Vorrangbeziehungen zu verhindern, wollen wir zunächst einmal das Modell des Zuordnungsproblems selbst um entsprechende Ungleichungen erweitern.

Ein erster, klassischer Satz von Nebenbedingungen um die Verletzung der Vorrangbeziehungen zu verhindern wird in Ungleichung (6.18) vorgestellt, hierbei ist für jede Vorrangbeziehung aus der transitiven Reduktion des Vorranggraphen nur eine Nebenbedingung nötig:

$$\sum_p p \cdot x_{i,p} \leq \sum_p p \cdot x_{j,p} \forall i \prec j \tag{6.18}$$

Diese Formulierung findet sich zum Beispiel auch in den Optimierungsmodellen für das Fließbandabstimmungsproblem (SALBP, siehe Domschke, Scholl und Voß 1997, Seite 196).

Eine Alternative stellt die folgende Formulierung dar, hierbei wird für jede Vorrangbeziehung $i \prec j$ und je zwei unterschiedliche, sich widersprechende Positionen p' und p für i und j eine eigene Nebenbedingung angelegt:

$$x_{i,p'} + x_{j,p} \leq 1 \quad \forall i \prec j, p' > p \tag{6.19}$$

Die Nebenbedingungen (6.19) drücken aus, dass wir Paare von möglichen Zuordnungen von Aufträgen zu Positionen vorliegen haben, die nicht miteinander kompatibel sind.

Die optimalen Lösungen des linearen Zuordnungsproblems ohne Erweiterung bilden perfekte bipartite Matchings von minimalem Kantengewicht. Im Artikel von Rusu (2008) wird das Optimierungsproblem der Suche nach einem Matching mit maximalem Kantengewicht betrachtet, unter der Annahme, dass es Paare von Kanten gibt, die nicht gleichzeitig im Matching vorkommen dürfen und damit inkompatibel sind. Dieses Optimierungsproblem wird dort als ungerades Matching mit maximalem Kantengewicht bezeichnet, und stimmt zunächst einmal mit dem von uns betrachteten Problem (bis auf den Sinn der Optimierungsrichtung) überein. Der Komplexitätsbeweis von Rusu (2008) basiert auf einer Reduktion einer Variante von 3-SAT auf die Bestimmung eines ungeraden Matchings von maximalem Gewicht, wobei auch die Struktur der Inkompatibilitäten eine Rolle spielt. Diese Reduktion hat zur Folge, dass die $\mathcal{NP}$-schwere des Problems für bipartite Graphen mit einem maximalem Knotengrad von drei gilt. Für unseren Fall hieße das aber, dass ein Auftrag immer maximal an drei Positionen eingeplant werden könnte, und eine Position für maximal Aufträge in Frage kommt. Zudem ist das Problem nur genau dann $\mathcal{NP}$-schwer, wenn ein aus den Inkompatibilitäten gebildeter Graph ein unbeschränktes induziertes Matching ermöglicht. Ein Matching ist induziert, sofern sich keine zwei am Matching beteiligten Knoten über eine weitere Kante des Graphen außerhalb des Matchings verbinden lassen (Stockmeyer und Vazirani 1982; Cameron 1989). Unbeschränkt ist ein induziertes Matching dabei dann, wenn es für jede ganze Zahl $h > 0$ eine Instanz des ungeraden Matchings gibt, so dass der aus den Inkompatibilitäten gebildete Graph ein induziertes Matching der Kardinalität von

mindestens h aufweist. Das induzierte Matching muss also mit der Instanzgröße des ungeraden Matchingproblems mitwachsen, ansonsten kann ein Polynomialzeitalgorithmus angegeben werden (Rusu 2008). An dieser Stelle wäre nun zu klären, ob die Ungleichungen (6.19) für bestimmte Formen von Vorrangbeziehungen (zum Beispiel für Ketten) zu einem Inkompatibilitätsgraphen führen, der diese Eigenschaften aufweist, um den Komplexitätsstatus des um Vorrangbeziehungen erweiterten linearen Zuordnungsproblems festzulegen.

Da in unserem Fall aber die sehr künstliche Beschränkung auf drei mögliche Zuordnungen pro Zeile/Spalte der Kostenmatrix nicht greift, jedoch andere Einschränkungen für die Instanzen vorliegen (zum Beispiel stehen für jeden Auftrag $j \in J$ immer alle Positionen p mit $p^{\min}(j) \leq p \leq p^{\max}(j)$ zur Verfügung), ist der Komplexitätsstatus dieses Problems zum jetzigen Zeitpunkt noch offen.

6.6.6.4 Zulässigkeitstest

Um lediglich die Frage nach der Existenz einer zulässigen Lösung für das lineare Zuordnungsproblem mit beliebigen Vorrangbeziehungen zu beantworten, können wir jedoch das Ein-Maschinen-Schedulingproblem $1|r_j, prec, p_j = 1|L_{max}$ auf einen optimalen Zielfunktionswert $L_{max} \leq 0$ überprüfen. Die Freigabezeiten der einzelnen Aufträge für dieses Problem ergeben sich dabei aus den minimal möglichen Positionen der Auftrags, die Fälligkeiten aus den maximal möglichen Positionen der Aufträge und die Prozesszeit von einer Einheit eines Auftrags entspricht einer Position, die für die Ausführung des Auftrags bereitzuhalten ist. Brucker (2007) zeigt, dass die Prioritätsregel von Horn (1974) für $1|r_j, prmp|L_{max}$ auf das vorliegende Problem angewendet werden kann, sofern Freigabezeitpunkte und Fälligkeiten mit den Vorrangbeziehungen konsistent sind. Für zwei Aufträge i und j mit $i \prec j$ muss also in unserer Anwendung von Problem und Verfahren sowohl $p^{\min}(i) < p^{\min}(j)$ und $p^{\max}(i) < p^{\max}(j)$ gelten. Die Erfüllung dieser Voraussetzung stellen wir bereits bei Erstellung der Kostenmatrix für das Zuordnungsproblem mit Hilfe einer topologischen Sortierung der Aufträge sicher. Bei Anwendung der Prioritätsregel von Horn (1974) wird jeweils der Auftrag mit der kleinsten maximalen Position aus der Menge der gemäß Vorrangbeziehungen aktuell zur Verfügung stehenden Aufträge zuerst eingeplant. Sollten wir dabei nie die maximale Position eines Auftrags überschreiten, haben wir dann eine Zuordnung gefunden, die alle Vorrangbeziehungen und die Positionsfenster beachtet.

Der Zielfunktionswert der mit dieser Regel erstellten Zuordnung bildet eine obere Schranke für das lineare Zuordnungsproblem und kann somit auch für einen Test darauf verwendet werden, ob die Lösung des linearen Zuordnungsproblems überhaupt zu einer Steigerung der unteren Schranke des aktuell betrachteten Knotens führen würde.

6.6.6.5 Lagrangerelaxation des LAP mit Vorrangketten

Trotz des ungeklärten Komplexitätsstatus können wir versuchen, ein Lösungsverfahren für das lineare Zuordnungsproblem mit zusätzlichen Vorrangrestriktionen in Form von Ketten entwickeln. Relaxieren wir die Nebenbedingungen (6.16), fordern also nicht mehr für jede Position die Zuordnung genau eines Auftrags, kann das verbliebene Problem in einer Laufzeit von $\mathcal{O}(|J|^2)$ gelöst werden, in dem ein später vorgestellter Ansatz der Dynamischen Programmierung aus

Abschnitt 7.5.2.4 der Reihe nach auf alle Ketten von Aufträgen angewendet wird. Wenn wir die Nebenbedingungen nicht vollständig außer Acht lassen, sondern mit Hilfe der Lagrange-Relaxation mit in die Zielfunktion aufnehmen, so kann das Lagrange-relaxierte Problem weiterhin mit der selben asymptotischen Laufzeit gelöst werden, da die Lagrange-Multiplikatoren nur die Kosten für die einzelnen Positionen eines jeden Auftrags modifizieren. Die Zielfunktion bei Anwendung der Lagrange-Relaxation auf die Nebenbedingungen (6.16) lässt sich nämlich in folgender Weise darstellen:

$$\begin{aligned} \min & \sum_{j \in J} \sum_{p=1}^{|J|} c_{j,p} \cdot x_{j,p} + \sum_{p} \pi_p (1 - \sum_{j} x_{j,p}) \\ & = \sum_{j} \sum_{p} (c_{j,p} - \pi_p) \cdot x_{j,p} + \sum_{p} \pi_p \end{aligned} \tag{6.20}$$

Das durch die Lagrange-Relaxation neu aufgeworfene Optimierungsproblem, gute Multiplikatoren zu finden, die den Zielfunktionswert von (6.20) möglichst stark ansteigen lassen, kann nun mit Hilfe eines Subgradientenverfahrens (Held, Wolfe und Crowder 1974; Fisher 2004) in Angriff genommen werden. Zur Erzeugung einer zulässigen oberen Schranke für das Problem greifen wir auf den Zulässigkeitstest aus Abschnitt 6.6.6.4 zurück, die initialen Multiplikatoren π_p für jede Position p setzen wir auf 0. Die Schrittweite und damit die neuen Multiplikatoren werden dabei auf die klassische Weise (Held, Wolfe und Crowder 1974; Fisher 2004) in Abhängigkeit des Abstandes von oberer zu untere Schranke und des Grades der aktuellen Verletzung der relaxierten Nebenbedingungen bestimmt. Das Verfahren wird dann nach einer vorgegebenen Anzahl von Iterationen abgebrochen.

Wird zusätzlich die untere Schranke für das lineare Zuordnungsproblem ohne weitere Nebenbedingungen verwendet, und entspricht dessen optimale Lösung auch den gegebenen Vorrangbeziehungen, so ist Anwendung der auf dieser Lagrange-Relaxation basierenden Schranke nicht mehr nötig.

Die hier vorgestellte Schranke auf Basis des Subgradientenverfahrens bildet nun eine Brücke zwischen der Schranke auf Basis des linearen Zuordnungsproblems und des im nächsten Kapitel 7 beschriebenen Spaltengenerierungsansatzes. Während wir soeben noch mit festen Kosten für die Zuordnung eines Auftrags auf eine Position gearbeitet haben, werden wir dann auch die Kosten für die Platzierung eines Auftrags von mehren Faktoren abhängig machen. Darunter ist dann auch die Platzierung des Vorgängerauftrags im Schedule. Anstelle der Lagrange-Multiplikatoren werden dann die aus dem Masterproblem abgeleiteten dualen Preise für einzelne Positionen den Zielfunktionswert des Unterproblems beeinflussen.

6.7 Ablauf des Branch-and-Bound-Verfahrens

Nachdem nun alle notwendigen Komponenten für das Branch-and-Bound-Verfahren vorgestellt sind, soll nun der gesamte Ablauf des Verfahrens noch einmal zusammenfassend dargestellt und wesentliche Charakteristika des Ansatzes kurz erläutert werden.

Initialisiert wird das Verfahren mit dem Einfügen des Wurzelknotens in den Branch-and-Bound-Baum. Dieser bildet dann auch den ersten aktiven Knoten,

der mit Hilfe einer der in Abschnitt 6.2 vorgestellten Techniken als zu verzweigender Knoten ausgewählt werden kann. Für jeden zu verzweigenden Knoten werden dann alle Kindknoten der Reihe nach evaluiert. Nach Berechnung der Fertigstellungszeit des neu angefügten Auftrages können dann die Kosten des Teilschedules ermittelt werden. Handelt es sich um einen vollständigen Schedule, wird dieser auf die globale Verbesserung der oberen Schranke überprüft. Ansonsten können nun auch die Dominanzkriterien aus Lemma 6.2 und Lemma 6.3 angewandt werden. Anschließend werden die ersten in Abschnitt 6.5 vorgestellten Zulässigkeitsüberprüfungen durchgeführt. Führt auch das nicht zu einem Herausschneiden des Knoten aus dem Branch-and-Bound-Baum, werden nun für den neuen Knoten die in der aktuellen Konfiguration des Verfahrens aktivierten unteren Schranken berechnet, die größte dieser Schranken bildet dann die untere Schranke für den Knoten. Sollte die untere Schranke die bis zum jetzigen Zeitpunkt bekannte beste obere Schranke nicht erreichen, wird der neue Knoten auch in die Menge der aktiven Knoten übernommen. Sobald bei Verzweigung eines Knotens keine neuen Knoten entstehen, wird der verzweigte Knoten und seine dann eventuell auch kinderlosen Ahnen gelöscht.

Das Verfahren endet mit dem Beweis der Optimalität der besten gefundenen oberen Schranke, sobald entweder die Menge der aktiven Knoten leer ist oder die untere Schranke aller aktiven Knoten mindestens den Zielfunktionswert der besten oberen Schranke erreicht. Sollte jedoch der vorhandene Speicherplatz nicht für die benötigte Anzahl an Knoten ausreichen, muss das Verfahren abgebrochen werden, die kleinste untere Schranke aller aktiven Knoten liefert nach vollständiger Verzweigung des zuletzt ausgewählten Knotens eine globale untere Schranke für die vorliegende Instanz des Schedulingproblems. Alternativ kann bei in Kürze vorhersehbaren Speicherplatzmangel auf die Tiefensuche als Knotenauswahlverfahren ausgewichen werden, da dann bei vollständiger Löschung aller bereits vollständig abgearbeiteten Bereiche des Branch-and-Bound-Baumes die Knotenanzahl eine feste Grenze nicht mehr übersteigt. Ein Optimalitätsbeweis in vertretbarer Rechenzeit ist in der Regel jedoch dann nur möglich, wenn auch mit der Bestensuche und einer modereaten Anzahl an zusätzlichen Knoten relativ bald ein Optimalitätsbeweis hätte erreicht werden können.

In dem hier vorgestellten Branch-and-Bound-Verfahren ist das Verzweigungsschema fest vorgegeben, sobald wir wissen welcher Knoten aufgrund der Knotenauswahlregeln verzweigt wird, wissen wir auch, wie er zu verzweigen ist. Im nächsten Kapitel werden wir dann Enumerationsschemata betrachten, in denen die anzuwendende Aufteilung des Lösungsraums, ähnlich wie im Branch-and-Bound-Verfahren für die Ganzzahlige Programmierung, nun aus der unteren Schranke des zukünftigen Vaterknotens bestimmt wird.

Kapitel 7

Ein Branch-and-Price-and-Cut-Ansatz

In den beiden vorherigen Kapiteln 5 und 6 haben wir uns entweder auf Optimierungsmodelle und die Ganzzahlige Programmierung gestützt, oder die Lösungen kombinatorisch durchprobiert wie im Branch-and-Bound-Ansatz. In diesem Kapitel werden wir nun die Methoden der Linearen und Ganzzahligen Programmierung mit Branch-and-Bound-Algorithmen verknüpfen. Dazu zerlegen wir das vorliegende Optimierungsproblem in ein Masterproblem, das mit Hilfe der Spaltengenerierung gelöst wird, und in mehrere Teilprobleme, die auch mit Hilfe von Branch-and-Bound-Verfahren gelöst werden können. Während das Verzweigungsschema im Branch-and-Bound-Ansatz aus Kapitel 6 fest vorgegebenen war, werden wir in diesem Kapitel die Verzweigungsentscheidungen in einem Knoten insbesondere auch aus der Struktur der unteren Schranke ableiten. Zur Bestimmung unterer und obere Schranken kommen in den Pricing-Algorithmen Ansätze der Dynamischen Programmierung zur Anwendung, so dass das hier vorgestellte Verfahren einen Einblick in das Zusammenspiel vielfältiger Methoden des Operations Research liefert.

7.1 Spaltengenerierungsverfahren

Wird der Simplexalgorithmus zur Lösung eines Linearen Programms verwendet, wird zunächst einmal davon ausgegangen, dass bereits alle für die Optimierung notwendige Information im Simplextableau vorhanden ist. Betrachten wir jedoch den revidierten Simplexalgorithmus (siehe zum Beispiel Papadimitriou und Steiglitz 1998), so stellen wir fest, dass dieser nur dann auf die Nicht-Basis-Variablen angewiesen ist, wenn das Optimalitätskriterium überprüft werden muss. Nur wenn es eine Nicht-Basis-Variable mit negativen reduzierten Kosten gibt, muss diese in die Basis eingetauscht werden. Anstelle alle Nicht-Basis-Variablen und deren Koeffizienten von Beginn an bereitzuhalten, können wir die Spalten der Nicht-Basisvariablen dann auch erst bei Beantwortung der Frage nach einer weiteren Spalte mit negativen reduzierten Kosten gegebenenfalls erzeugen. Die Suche nach einer Spalte mit negativen reduzierten Kosten lässt sich dabei in einer Optimierungsproblem umdeuten, indem wir eine Spalte mit minimalen reduzierten Kosten suchen. Dieses Optimierungsproblem soll im Folgenden als

Pricing-Problem bezeichnet werden. Eine optimale Lösung mit negativen Zielfunktionswert des Pricing-Problems entspricht dann einer Nicht-Basis-Variable, die für den Fall, dass keine primale Entartung vorliegt, bei Eintausch in die Basis auch eine Verbesserung des Zielfunktionswertes des zu lösenden Linearen Programms liefert. Die Auslagerung in ein Optimierungsproblem bietet den Vorteil, das nicht alle Spalten bekannt sein müssen, sondern erst im Verlauf des Verfahrens bei Bedarf generiert werden. Deswegen spricht man an dieser Stelle auch von einem Spaltengenerierungsverfahren.

Bereits in Ford und Fulkerson (1958) wird diese Technik zur Lösung eines Multi-Commodity-Flow-Problems verwendet. Eine sehr bekannte frühe Anwendung eines Spaltengenerierungansatzes auf eine praktische Problemstellung bildet der Artikel von Gilmore und Gomory (1961) zur Lösung des Cutting-Stock-Problems. Die Lösungen des Pricing-Problems repräsentieren hierbei Schnittmuster, die als Spalten in ein Masterproblem eingefügt werden. Das Masterproblem hat dann die Aufgabe, durch eine geeignete Auswahl von Schnittmustern die gegebenen Bedarfe zu decken. Durch diese Aufteilung ist es auch möglich, spezielle Anforderungen an Schnittmuster zu stellen (Gilmore und Gomory 1963).

Kennington (2004) bezeichnet in seiner Retrospektive den Artikel von Ford und Fulkerson (1958) als Ausgangspunkt für die Entwicklung der Dantzig-Wolfe-Zerlegung (Dantzig und Wolfe 1960). Im Rahmen dieser Zerlegung wird ein großes Lineares Programm, dessen Koeffizientenmatrix sich auf bestimmte Weise in mehrere Blöcke zerlegen lässt, in ein Masterproblem und verschiedene Unterprobleme unterteilt und mittels der Spaltengenerierung gelöst. Das Masterproblem enthält dabei nur noch die Nebenbedingungen, die Teil von mindestens zwei Unterproblemen sind, während die Unterprobleme in sich abgeschlossen sind, und nur Nebenbedingungen enthalten, die sich ausschließlich auf die dem jeweiligen Unterproblem zugeordneten Variablen beziehen. Damit können die Unterprobleme auch als autarke Lineare Programme aufgefasst werden, während das Masterproblem versucht, eine für das gesamte lineare Programm zulässige und optimale Lösung zu ermitteln. Jede Lösung des Masterproblems stellt dabei eine Reihe von Konvexkombination der Eckpunkte eines jeden Unterproblems dar, die auch die im Masterproblem verbliebenen Nebenbedingungen zusätzlich erfüllen. Jede Spalte des Masterproblems ist damit ein Eckpunkt beziehungsweise eine Basislösung genau eines Unterproblems, durch die Definition der Basislösung erhalten wir damit ein exponentielle Anzahl von Variablen. Auch hier wird wieder versucht, die Spalten nur bei Bedarf mit Hilfe der Unterprobleme zu erzeugen. Da es in den Unterproblemen aber nicht mehr darum geht, nur eine Optimale Basislösung unter Beachtung der ursprünglichen Zielfunktion zu finden, sondern eine Spalte mit minimalen reduzierten Kosten für das Masterproblem, muss das Unterproblem abgeändert werden. Diese Modifikation betrifft allerdings nur die Zielfunktion des Unterproblems, die nun zusätzliche Information aus der dualen Lösung des Masterproblems enthält. Hier teilt das Masterproblem mit, wie es bestimmte Eigenschaften der Basislösungen des Unterproblems bewertet, in der Hoffnung, vom Unterproblem eine Basislösung zurückzuerhalten, die es erlaubt, auch den Zielfunktionswert des Masterproblems unter Berücksichtigung aller Nebenbedingungen zu verbessern. Da aber erst nach mehreren Iterationen dann tatsächlich auch die optimale Lösung des Masterproblems erreicht wird, gleicht dieses Vorgehen einem Verhandlungsprozess, in dem auf der einen Seite das Masterproblem und auf der anderen Seite

die Unterprobleme (die jetzt zu Pricing-Problemen geworden sind) durch den Austausch von Informationen (Duale Preise gegen Basislösungen) gemeinsam um die beste Lösung ringen. Dabei ist es dem Masterproblem egal, wie diese Basislösungen ermittelt werden, und auch die Unterprobleme müssen nur Informationen darüber haben, wie mit der dualen Information des Masterproblems umzugehen ist.

Spaltengenierungsverfahren sind zunächst einmal Erweiterungen des revidierten Simplexalgorithmus und damit zur Lösung von Linearen Programmen geeignet. Wollen wir nun auch (gemischt) Ganzzahlige Optimierungsprobleme mit Hilfe von Spaltengenerierungsverfahren lösen, müssen wir wieder durch die Anwendung eins Branch-and-Bound-Verfahrens auf Basis der LP-Relaxation des Masterproblems (LPM) versuchen, ganzzahlige Lösungen in den entstehenden Teilbereichen zu erzeugen. Diese Branch-and-Price genannte Technik wird in den Artikeln Vanderbeck und Wolsey (1996) und Barnhart u. a. (1998) sowie im Lehrbuch von Wolsey (1998) näher beschrieben. Auf die durch Anwendung dieser Technik aufkommenden Herausforderungen werden wir dann in Abschnitt 7.7 noch näher eingehen.

7.2 Spaltengenerierungansätze im Scheduling

Die Methode der Spaltengenerierung findet auch Anwendung im Bereich des Scheduling. Einen Überblick über die Verwendung von Spaltengenerierungsansätzen im Scheduling bildet das Kapitel von Akker, Hoogeveen und Velde (2005) im Sammelband von Desaulniers, Desrosiers und Solomon (2005). In Chen und Powell (2003) wird eine Produktionsumgebung mit parallel identischen Maschinen betrachtet. Die Spalten bilden hierbei dann Schedules für einzelne Maschinen. Während diese Möglichkeit der Zerlegung eines Schedulingproblems mit mehreren Maschinen recht offensichtlich ist, gibt es auch für Ein-Maschinen-Scheduling-Probleme bereits Literatur, die Vorschläge für deren Zerlegung enthält. In Brucker und Knust (2002) wird ein Ein-Maschinen-Schedulingproblem beschrieben, dass sich aus der Betrachtung eines Transportroboters in einer Job-Shop-Umgebung ergibt. Dabei wird ein Schedule für den Job-Shop vorgegeben, und der Transportroboter muss die Materialtransporte bei minimaler Durchlaufzeit ausführen. Dadurch, dass die Operationen der Aufträge auf jeder Maschine des Job-Shops schon eingeplant sind, ergeben sich auch hier aus den Operationen der Aufträge des Job-Shops Ketten von Transportaufträgen für den Roboter, die die Grundlage für eine Zerlegung des Problems bilden.

Die Zerlegung der zeitindizierten Formulierung für das Ein-Maschinen-Scheduling von Sousa und Wolsey (1992) (siehe hierzu Abschnitt 5.4) mittels einer Dantzig-Wolfe-Zerlegung wird in Akker, Hurkens und Savelsbergh (2000) diskutiert. Hier wird im Masterproblem die Forderung beibehalten, dass jeder Auftrag genau einmal gestartet werden muss, während im Unterproblem die Kapazitätsbeschränkungen der Maschine auf einen ausgeführten Auftrag pro Zeiteinheit betrachtet werden. Die Lösungen des Unterproblems werden als *Pseudoschedules* bezeichnet, da die Bedingung des Masterproblems im Unterproblem verletzt werden kann, und dann ein Auftrag auch gar nicht oder mehrfach gestartet werden kann. Um das Unterproblem zu lösen, muss ein kürzester Weg in einem gerichteten, azyklischem Netzwerk gefunden werden. Dieses kann über einen Ansatz der dynamischen Programmierung in einer Laufzeit von $(|J| \cdot T)$ gesche-

hen.

7.3 Zerlegung eines Schedules

Allen Spaltengenerierungsansätzen, ob im Scheduling oder für weitere Probleme ist die Zusammenfassung von Einzelentscheidungen in den Unterproblemen gemein. Im Vehicle-Routing-Problem wird dann nicht mehr danach gefragt, ob ein Fahrzeug v von Kunde i zu Kunde j fahren soll, sondern es wird vom Pricing-Algorithmus eine komplette Tour für das Fahrzeug vorgeschlagen. Im Fall der soeben vorgestellten zeitindizierten Formulierung wird nicht mehr der Start eines einzelnen Auftrags bestimmt, sondern allen Aufträgen gleichzeitig eine Startzeit zugewiesen.

Auch für den in dieser Arbeit vorgestellten Spaltengenerierungsansatz müssen wir uns überlegen, welche Entscheidungen von dem oder den Pricing-Verfahren zu einer Variable des Masterproblems zusammengefasst werden sollen. Dazu betrachten wir einen Schedule, und überlegen uns, welche Entscheidungen getroffen werden, und wie diese sinnvoll gruppiert werden könnten.

Definition 7.1 (Eingeplante Menge von Vorgängern)
*Sei $\sigma \in \mathfrak{S}^{|J|}$ mit $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_{|J|})$ ein zulässiger Schedule, wobei $\sigma_0 = 0$ den künstlichen Startauftrag darstellt. Für jeden Auftrag $j \in J$ mit $j = \sigma_k$, bezeichnen wir die Auftragsmenge $P_j(\sigma) := \{i \in J : \exists 1 \leq k' \leq k, \sigma_{k'} = i\}$ als die **Menge von eingeplanten Vorgängern** für Auftrag j im Schedule σ.*

Lemma 7.2
Jede eingeplante Menge von Vorgängern $P_j(\sigma) \subseteq J$ für einen Auftrag $j \in J$ und eine zulässigen Schedule $\sigma \in \mathfrak{S}^{|J|}$ hat die folgenden Eigenschaften:

1. $P_j^{\min} := \{i \in J : i \prec j\} \cup \{j\} \subseteq P_j(\sigma) \subseteq P_j^{\max} := J \backslash \{i \in J : j \prec i\}$
2. $i \in P_j(\sigma) \Rightarrow \{l \in J, l \prec i\} \subseteq P_j(\sigma)$.

Lemma 7.3
Für jedes Paar von Aufträgen $(i, j) \in J \times J$ mit $i \prec j$ und einen zulässigen Schedule $\sigma \in \mathfrak{S}^{|J|}$ gilt dass $P_i(\sigma) \subset P_j(\sigma)$.

Beweis. Trivial. □

Definition 7.4 (Zulässige Menge von Vorgängern)
*Jede Menge $P_j \subseteq J$, für die eine topologische Sortierung der Aufträge $\sigma \in \mathfrak{S}^{|J|}$ existiert mit $P_j = P_j(\sigma)$, wird als **zulässige Menge von Vorgängern** für $j \in J$ bezeichnet.*

Definition 7.5 (Eingeplanter direkter Vorgänger)
*Sei $\sigma \in \mathfrak{S}^{|J|}$ mit $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_{|J|})$ ein zulässiger Schedule, wobei $\sigma_0 = 0$ den initialen Rüstzustand bezeichne. Dann bezeichnen wir für jeden Auftrag $j \in J$ mit $\sigma_k = j$ und $k > 0$ den Auftrag $i \in J \cup \{0\}$ mit $\sigma_{k-1} = i$ als **eingeplanten direkten Vorgänger** $x_j(\sigma)$ von j im Schedule σ.*

Lemma 7.6
Jeder eingeplante direkte Vorgänger $i = x_j(\sigma)$ von j hat die folgenden Eigenschaften:

1. *$P_i(\sigma) = P_j(\sigma)\backslash\{j\}$ ist eine zulässige Menge von Vorgängern für i.*
2. *Für $j' \prec j, j' \neq i$ gilt dass $i \in P_j(\sigma)\backslash P_{j'}(\sigma)$*
3. *Wenn $j' \prec j, j' = i$, dann ist $P_j(\sigma)\backslash P_{j'}(\sigma) = \{j\}$*
4. *$i = 0 \Leftrightarrow P_j(\sigma) = \{j\}$.*

Beweis. Trivial. □

Definition 7.7 (Zulässiger direkter Vorgänger)
*Jeder Auftrag $i \in J$, der für zwei Aufträge $j' \in J$ und $j \in J$ mit $j' \prec j$ und dazugehörigen zulässigen Mengen von Vorgängern $P_{j'}$ und P_j die Eigenschaften von Lemma 7.6 erfüllt, wird als **zulässiger direkter Vorgänger** x_j für $j \in J$ bezeichnet.*

Die Zulässigkeit eines direkten Vorgängers lässt sich hier also nur dann ableiten, wenn auch entsprechende Mengen von Vorgängern für den betrachteten Auftrag j und seinen Vorgänger j' gegeben sind.

Anstelle einen Schedule Schritt für Schritt zu erzeugen, wie das in dem vorher besprochenen Branch-and-Bound-Ansatz (siehe hierzu das Kapitel 6) der Fall ist, konzentrieren wir uns auf zulässige Mengen von Vorgängern und zulässige direkte Vorgänger. Diese Begriffe erlauben es uns, die Spalten unseres Spaltengenerierungsverfahrens zu definieren:

Definition 7.8 (Inklusionsmuster)
Ein Inklusionsmuster ι für Kette $C(\iota) = (j_1 \prec \ldots \prec j_{|C(\iota)|}) \subseteq J$ ist definiert durch eine Folge von zulässigen Mengen von Vorgängern

$$\mathcal{P} = P_{j_1}(\iota) := P_{j_1}(\sigma)) \subset \ldots \subset P_{j_{|C(\iota)|}}(\iota) := P_{j_1}(\sigma)) \tag{7.1}$$

für mindestens eine zulässige topologische Sortierung der Aufträge $\sigma \in \mathfrak{S}^{|J|}$. Zusätzlich bestimmt das Inklusionsmuster ι für jeden Auftrag $j_k \in C(\iota)$ in der Kette einen zulässigen direkten Vorgänger $x_{j_k}(\iota)$ in Abhängigkeit von $P_{j_k}(\iota)$ und $P_{j_{k-1}}(\iota)$. Die Kosten $z(\iota)$ eines Inklusionsmusters ι ergeben sich aus einer unteren Schranke der summierten gewichteten Fertigstellungszeit aller Aufträge $j \in C(\iota)$:

$$z(\iota) = \sum_{j \in C} w_j C_j^{LB}(\iota), \tag{7.2}$$

wobei $C_j^{LB}(\iota)$ eine Abschätzung für die minimale Fertigstellungszeit des Auftrags $j \in C(\iota)$ hinsichtlich des Inklusionsmusters ι bezeichne.

In den Inklusionsmustern fassen wir für die Aufträge einer Kette $C \in \mathcal{C}$ die Entscheidungen, die auch von einem Schedule für das eigentlich betrachtete Optimierungsproblem getroffen werden, zusammen. Anhand dieser Entscheidungen versuchen wir dann, den Beitrag zum Zielfunktionswert der Aufträge aus der Kette abzuschätzen, und erhalten dann eine untere Schranke für den tatsächlichen Zielfunktionswert des Schedules. Die zulässigen Vorgängermengen $P_j(\iota)$ für $j \in C(\iota)$ eines Inklusionsmusters ι entscheiden über den Wert der Vorrangvariablen $\pi_{i,j}$ aus der Formulierung von Manne (1960), während die Wahl der direkten Vorgänger Entscheidungen des klassischen TSP-Modells (Dantzig, Fulkerson und Selmer M. Johnson 1954) widerspiegelt. Hier legen wir für jeden

Auftrag aus der Kette bereits eine Seite der Zuordnungsbedingungen fest (jeder Auftrag hat genau einen Vorgänger), während wir die Frage nach dem direkten Nachfolger noch offen lassen. Weiterhin bestimmt die Kardinalität von $P_j(\iota)$ auch eine für den Auftrag $j \in C(\iota)$ angestrebte Position im Schedule.

Definition 7.9
*Sei $\mathcal{C}$ eine Partitionierung der Auftragsmenge J in Ketten und $\sigma \in \mathfrak{S}^{|J|}$ ein zulässiger Schedule. Dann bezeichnen wir die Menge von $|\mathcal{C}|$ Inklusionsmustern $I(\sigma) = \{\iota_1, \ldots, \iota_{|\mathcal{C}|}\}$, die sich aus den eingeplanten Mengen von Vorgängern und den eingeplanten direkten Vorgängern ergeben als die **Zerlegung in Inklusionsmuster** von σ.*

7.3.1 Eigenschaften der Zerlegung

Jede zulässige Schedule σ erfüllt bezüglich der eingeplanten Mengen von Vorgängern $P_j(\sigma)$ für jeden Auftrag $j \in J$ und der eingeplanten direkten Vorgänger verschiedene Eigenschaften, die dann auch auf die Eigenschaften der Zerlegung in Inklusionsmuster $I(\sigma)$ zutreffen. Diese sollen im Folgenden kurz aufgezählt werden, der Beweis ist jeweils offensichtlich:

Lemma 7.10
Für einen gegebenen zulässigen Schedule σ existiert für jede Position p mit $1 \leq p \leq |J|$ genau ein Auftrag $j \in J$ für dessen eingeplante Menge von Vorgängern gilt: $|P_j(\sigma)| = p$.

Lemma 7.11
Für zwei verschiedene Aufträge $i, j \in J$ gilt bezüglich der eingeplanten Mengen von Vorgängern von i und j entweder $i \in P_j(\sigma)$ oder $j \in P_i(\sigma)$.

Lemma 7.12
Für einen gegebenen zulässigen Schedule σ sei $G(J, J \times J)$ ein vollständiger gerichteter Graph. Zusätzlich sei die Kantenmenge $E = \{(i, j) \in J \times J : x_j(\sigma) = i\}$ gegeben. Dann bilden die Kanten aus E einen Hamilton-Pfad in $G(J, J \times J)$.

Diese Eigenschaften wollen wir später als Richtschnur für die Erstellung der Nebenbedingungen des Masterproblems benutzen.

7.3.2 Zeitabschätzung

Um die Kosten eines Inklusionsmusters ι zu berechnen, sind wir auf eine Abschätzung der Fertigstellungszeit $C_j^{LB}(\iota)$ eines jeden Auftrags $j \in C(\iota)$ aus der Kette angewiesen. Wenn wir eine gültige untere Schranke der Fertigstellungszeit $C_{j_k}^{LB}(\iota)$ für den k-ten Auftrag der Kette berechnen wollen, können wir uns nur auf die Mengen von Vorgängern $P_{j_1}(\iota) \subset \ldots \subset P_{j_k}(\iota)$ und die Wahl der direkten Vorgänger für die Aufträge der Kette stützen. Für die Durchführung dieser Abschätzung gibt es mehrere Möglichkeiten, von denen wir zwei näher betrachten wollen.

7.3.2.1 Unabhängige Zeitabschätzung

Diese Zeitabschätzung für einen Auftrag aus der Kette $j \in C(\iota)$ basiert allein auf der zulässigen Menge von Vorgängern $P_j(\iota)$ und der zulässigen Wahl des

direkten Vorgängers $x_j(\iota) = i$ für j. Zunächst schätzen wir die Fertigstellungszeit des direkten Vorgängers $x_j(\iota) = i$ ab. Das Ergebnis sei mit $\tilde{C}_i^{LB}$ bezeichnet:

$$\tilde{C}_i^{LB}(\iota) := \max\{\sum_{l \in P_j(\iota)\backslash\{j\}} p_l + st(P_j(\iota)\backslash\{j\}, i), Inv(P_j(\iota)\backslash\{j\}) + p_i\} \quad (7.3)$$

Dieses Vorgehen entspricht der Zeitabschätzung für die Berechnung einer Menge von Vorgängern mit maximaler Kardinalität in Abschnitt 4.3.1, die Berechnung des frühesten Zeitpunktes der Materialverfügbarkeit $Inv(J')$ für eine Menge $J' \subseteq J$ ist in Abschnitt 2.2 diskutiert worden. Ist die Zeit für den direkten Vorgänger bestimmt, kann aus dessen Zeit (beziehungsweise der Zeit der frühesten Materialverfügbarkeit für die Bearbeitung der ganzen Menge von Vorgängern $P_j(\iota)$) auch die Fertigstellungszeit von Aufrag $j \in C$ abgeschätzt werden:

$$C_j^{LB}(\iota) := \max\{\tilde{C}_i^{LB}(\iota) + st(i, j) + p_j, Inv(P_j(\iota)) + p_j\} \quad (7.4)$$

Hier simulieren wir das Anfügen von j an einen partiellen Schedule, der in i endet.

7.3.2.2 Abhängige Zeitabschätzung

Für eine Kette $C(\iota) = (j_1, \ldots, j_k, \ldots j_{|C|})$ können wir für $k > 1$ auch die Abschätzung $C_{j_{k-1}}^{LB}(\iota)$ als Ausgangspunkt für die Berechnung von $C_{j_k}^{LB}(\iota)$ hinzuziehen, die Berechnung ist also abhängig von den für die Vorgänger in der Kette berechneten Zeiten. Zunächst ist das Vorgehen für $k = 1$ identisch mit der Berechnung in (7.3) und (7.4). Für $k > 1$ wird die untere Schranke für die Fertigstellungszeit $\tilde{C}_i^{LB}(\iota)$ des direkten Vorgängers i von j_k berechnet durch

$$\begin{aligned} \tilde{C}_i^{LB}(\iota) := \max\{&C_{j_{k-1}}^{LB}(\iota) \\ &+ st(j_{k-1}, P_{j_k}(\iota)\backslash(P_{j_{k-1}}(\iota) \cup \{j_k\}), i) \\ &+ \sum_{l \in P_{j_k}(\iota)\backslash(P_{j_{k-1}}(\iota) \cup \{j_k\})} p_l, \\ &Inv(P_{j_k}(\iota)\backslash\{j_k\}) + p_i\}. \end{aligned} \quad (7.5)$$

An dieser Stelle wird die in Abschnitt 4.2.2 vorgestellte Technik zur Abschätzung der Rüstzeit bei beliebiger Wahl der Rüstbasis angewandt. Die Rüstbasis ist in diesem Fall immer der Vorgängerauftrag aus der Kette j_{k-1}. Die untere Schranke für die Fertigstellungszeit $C_{j_k}^{LB}(\iota)$ des Auftrags aus der Kette ist dann analog zu 7.4 wieder gegeben durch

$$C_{j_k}^{LB}(\iota) := \max\{\tilde{C}_i^{LB}(\iota) + st(i, j_k) + p_j, Inv(P_{j_k}(\iota)) + p_{j_k}\} \quad (7.6)$$

7.3.2.3 Zulässigkeitstest

Sobald $\tilde{C}_i^{LB}(\iota) > \delta_i$ oder $C_j^{LB}(\iota) > \delta_j$ erfüllt ist, bezeichnen wir das zu dieser Zeitabschätzung führende Inklusionsmuster ι als zeitlich unzulässig. Es existiert kein zulässiger Schedule σ für unser Ausgangsproblem, der in seiner Zerlegung in Inklusionsmuster $I(\sigma)$ zu einem zeitlich unzulässigem Inklusionsmuster führt.

7.3.2.4 Qualität der Abschätzung

Werden die Kosten eines Inklusionsmusters mit der unabhängigen Zeitabschätzung berechnet, sind die Kosten kleiner oder gleich den Kosten durch die abhängige Zeitabschätzung. Die Qualitätssteigerung der abhängigen Zeitabschätzung ist einerseits durch die bessere Integration von weiteren Rüstzeiten, die sich durch die wiederholten Rüstvorgänge von und zu Produkten der Kette ergeben, begründet. Andererseits werden auch Wartezeiten auf Materialverfügbarkeit, die bei Vorgängern in der Kette auftreten, mit in die Zeitberechnung der direkten Vorgänger einbezogen. Der Vorteil der unabhängigen Zeitabschätzung liegt jedoch in einer insgesamt einfacheren Evaluation, die auch die Formulierung eines ganzzahligen Optimierungsmodells zur Spaltengenerierung ohne größere Schwierigkeiten erlaubt (siehe dazu Abschnitt 7.5.1).

Die Abschätzungen in 7.3.2.1 und 7.3.2.2 lassen sich in Polynomialzeit berechnen. Der größte Aufwand liegt dabei in der Berechnung der Rüstzeit. Eine weitere Einbeziehung der Materialverfügbarkeit über eine Überprüfung für direkte Vorgänger und Aufträge aus der Kette hinaus ist schwierig zu erreichen, da das Problem $1|rm|C_{\max}$ bereits bei Konkurrenz der Aufträge um ein Rohmaterial $\mathcal{NP}$-schwer ist (Grigoriev, Holthuijsen und Klundert 2005).

Zerlegen wir einen Schedule σ in die Menge seiner Inklusionsmuster $I(\sigma)$, so bildet die Summe der Kosten der Inklusionsmuster damit nur eine untere Schranke für den Zielfunktionswert des Schedules σ. Mit der Zerlegung in Inklusionsmuster geben wir also Informationen auf, die notwendig sind, um die Kosten des gesamten Schedules zu bestimmen. Würde dagegen der Zielfunktionswert eines Schedules nur von den in den Inklusionsmustern hinterlegten Informationen abhängen, ließe sich auf diese Weise eine exakte Zerlegung erreichen. Dies ist insbesondere dann der Fall, wenn wir keinerlei Verzögerungen der Produktion durch Rüstzeiten oder Wartezeit auf Material betrachten würden.

7.4 Das Masterproblem

Das Masterproblem in unserem Spaltengenerierungsansatz soll eine kostenminimale Auswahl aus allen ihm bekannten Inklusionsmustern treffen. Die Auswahl eines Inklusionsmusters wird dabei über den Wert einer binären Variablen $\lambda_t \in \{0,1\}$ mit $t \in \mathcal{T}$ angezeigt. Die Menge $\mathcal{T}$ stellt dabei die Indexmenge aller aktuell im Masterproblem vertretenen Spalten beziehungsweise Inklusionsmuster dar. Für die t-te Spalte bezeichne $\iota(t)$ das mit dieser Spalte assoziierte Inklusionsmuster ι. Um die Kosten der Auswahl zu minimieren, lässt sich die Zielfunktion formulieren als

$$\min \sum_{t \in \mathcal{T}} z(\iota(t)) \cdot \lambda_t \tag{7.7}$$

Damit die getroffene Auswahl an Inklusionsmustern möglichst einer Zerlegung eines Schedules in Inklusionsmuster gemäß Definition 7.9 möglichst nahe kommt, sind neben der Zielfunktion noch Nebenbedingungen hinzuzufügen.

7.4.1 Konvexitätsbedingungen

Die Konvexitätsbedingungen, wie sie auch in jeder Dantzig-Wolfe-Zerlegung (Dantzig und Wolfe 1960) vorkommen, erzwingen die Auswahl genau eines In-

klusionsmusters für jede Kette $C \in \mathcal{C}$. Für deren Formulierung und weitere Anwendungen führen wir die Indikatorfunktion $I^c : \mathcal{C} \times \mathcal{T} \mapsto \{0;1\}$ mit

$$I^c(C,t) = \begin{cases} 1 & \Leftrightarrow C(\iota(t)) = C \\ 0 & \text{sonst} \end{cases} \tag{7.8}$$

ein, die anzeigt, ob die t-te Spalte ein Inklusionsmuster für Kette $C \in \mathcal{C}$ darstellt. Die Konvexitätsbedingungen selbst sind dann gegeben durch

$$\sum_{t \in \mathcal{T}} I^c(C,t) \cdot \lambda_t = 1 \quad \forall\, C \in \mathcal{C} \tag{7.9}$$

Dadurch ist auch jede Variable λ_t mit $t \in \mathcal{T}$ automatisch nach oben auf den Wert 1 beschränkt. Jeder Klasse von Nebenbedingungen entspricht im dualen des RLPM eine Klasse von dualen Variablen, deren Vektor wir für die Konvexitätsbedingungen (7.9) im Folgenden mit $(\gamma_1, \ldots, \gamma_c, \ldots, \gamma_{|\mathcal{C}|})^\top$ bezeichnen wollen.

7.4.2 Elementare Entscheidungen

Jede Spalte in einem Spaltengenerierungsverfahren, und damit auch jedes Inklusionsmuster fasst verschiedene elementare Entscheidungen zusammen. Für Inklusionsmuster wollen wir drei Typen von elementaren Entscheidungen aus deren Definition ableiten und näher betrachten. Diese elementaren Entscheidungen lassen sich mit den originalen Variablen eines IP-Modells vergleichen, auf das eine Dantzig-Wolfe-Zerlegung angewandt wird. Sei im Folgenden λ eine zulässige primale Lösung für das RLPM und (γ, μ) bezeichne eine zulässige duale Lösung für das RLPM. Diese ist zusammengesetzt aus dem Vektor $\gamma \in \mathbb{R}^{|\mathcal{C}|}$ der dualen Variablen für die Konvexitätsbedingungen (7.9) und dem Vektor $\mu \in \mathbb{R}^{|\mathcal{R}|}$ für die dualen Variablen aller weiteren Nebenbedingungen mit Indexmenge $\mathcal{R}$ des RLPM.

7.4.2.1 Entscheidungen über Vorgänger

Die primale Indikatorfunktion für die Entscheidung über die Aufnahme eines Auftrags $i \in J$ in die Vorgängermenge von $j \in J$, $I^P_{\pi_{i,j}} : \mathcal{T} \mapsto \{0;1\}$ wird definiert durch

$$I^P_{\pi_{i,j}}(t) = \begin{cases} 1 & \text{für } I^c(C(j),t) = 1 \text{ und } i \in P_j(\iota(t)) \\ 0 & \text{sonst} \end{cases} \tag{7.10}$$

Wir bezeichnen mit

$$\pi^P_{i,j}(\lambda) := \sum_{t \in \mathcal{T}} \lambda_t I^P_{\pi_{i,j}}(t) \in [0,1] \tag{7.11}$$

den Wert für die Entscheidung, den Auftrag $i \in J$ in die Menge von Vorgängern P_j von $j \in J$ in der aktuellen Lösung des RLPM λ aufzunehmen. Die duale Indikatorfunktion $I^D_{\pi_{i,j}} : \mathcal{R} \mapsto \mathbb{Z}$ für eine Vorrangbeziehung $i \in P_j$ liefert uns den Koeffizienten für die Wahl von i als Vorgänger von j, $\pi(i,j)$, in Nebenbedingung $r \in \mathcal{R}$ des RLPM. Der duale Preis für die Aufnahme von i in die Vorgängermenge von j ist dann bezüglich der dual zulässigen Lösung μ des RLPM gegeben durch

$$\pi^D_{i,j}(\mu) := \sum_{r \in \mathcal{R}} \mu_r \cdot I^D_{\pi_{i,j}}(r). \tag{7.12}$$

Jede Menge von Vorgängern P_j eines Auftrags j fasst damit $|P_j|$ Entscheidungen über Vorgänger zusammen, die Kardinalität $|P_j|$ der Menge von Vorgängern eines Auftrags ist dann Gegenstand der nächsten Klasse von elementaren Entscheidungen:

7.4.2.2 Entscheidungen über Positionen

Für jeden Auftrag $j \in C(\iota)$ wird eine Position $p = |P_j(\iota(t))|$ in einem Inklusionsmuster ι bestimmt. Die primale Indikatorfunktion für die Entscheidung über Positionen $I^P_{\psi_{j,p}} : \mathcal{T} \mapsto \{0;1\}$ wird definiert durch

$$I^P_{\psi_{j,p}}(t) = \begin{cases} 1 & \text{für } I^c(C(j),t) = 1 \text{ und } |P_j(\iota(t))| = p \\ 0 & \text{sonst} \end{cases} \tag{7.13}$$

Wir bezeichnen mit

$$\psi^P_{j,p}(\lambda) := \sum_{t \in \mathcal{T}} \lambda_t I^P_{\psi_{jp}}(t) \in [0,1] \tag{7.14}$$

den Wert der Entscheidung für eine Platzierung von Auftrag j auf Position p in der aktuellen Lösung λ des RLPM. Die duale Indikatorfunktion $I^D_{\psi_{j,p}} : \mathcal{R} \mapsto \mathbb{Z}$ für eine Platzierung von j auf Position p liefert uns den Koeffizienten für die Elementare Entscheidung $\psi_{j,p}$ in Nebenbedingung $r \in \mathcal{R}$ des RLPM. Der duale Preis für die Entscheidung, den Auftrag j auf Position p einzuplanen, ist dann für die dual zulässige Lösung μ des RLPM gegeben durch

$$\psi^D_{j,p}(\mu) := \sum_{r \in R} \mu_r I^D_{\psi_{j,p}}(r). \tag{7.15}$$

7.4.2.3 Entscheidungen über Kanten

Kanten, die einen Auftrag $i \in J$ mit einem Auftrag $j \in C$ verbinden, werden durch die Wahl von direkten Vorgängern in einem Inklusionsmuster definiert. Die primale Indikatorfunktion für die Auswahl von i als direkten Vorgänger für Auftrag j ist gegeben durch $I^P_{x_{i,j}} : \mathcal{T} \mapsto \{0;1\}$ mit

$$I^P_{x_{i,j}}(t) = \begin{cases} 1 & \text{für } I^c(C(j),t) = 1 \text{ und } x_j(\iota(t)) = i \\ 0 & \text{sonst} \end{cases} \tag{7.16}$$

Wir bezeichnen dann mit

$$x^P_{i,j}(\lambda) := \sum_{t \in \mathcal{T}} \lambda_t I^P_{x_{i,j}}(t) \in [0,1] \tag{7.17}$$

den aktuellen Wert der Auswahl von i als direkten Vorgänger für j in der Lösung des RLPM λ. Die duale Indikatorfunktion $I^D_{x_{i,j}} : \mathcal{R} \mapsto \mathbb{Z}$ für die Wahl von i als direkten Vorgänger für j liefert uns den Koeffizienten für die elementare Entscheidung $x_{i,j}$ in Nebenbedingung $r \in \mathcal{R}$ des RLPM. Der duale Preis für die Entscheidung, den Auftrag i als direkten Vorgänger von j zu wählen, lässt sich dann durch

$$x^D_{i,j}(\mu) := \sum_{r \in R} \mu_r I^D_{x_{i,j}}(r). \tag{7.18}$$

berechnen, wobei μ eine zulässige duale Lösung des RLPM darstellt.

Die Auffächerung eines Inklusionsmusters in elementare Entscheidungen erlaubt es uns, die weiteren Nebenbedingungen des Masterproblems auf einfache Weise zu formulieren. Zudem lässt sich durch die elementaren Entscheidungen die Berechnung der reduzierten Kosten und der Spaltenkoeffizienten eines Inklusionsmusters deutlich vereinfachen.

7.4.2.4 Reduzierte Kosten

Die reduzierten Kosten $\bar{z}_{t,j}$ eines Auftrages $j \in J$ in der t-ten Spalte mit $I^c(C(j), t) = 1$ lassen sich folgendermaßen berechnen:

$$\begin{aligned} \bar{z}_{t,j} :=& w_j \cdot C_j^{LB} \\ &- \sum_{i \in J} I^P_{\pi_{i,j}}(t) \pi^D_{i,j}(\mu) \\ &- \sum_{p=1}^{|J|} I^P_{\psi_{j,p}}(t) \psi^D_{j,p}(\mu) \\ &- \sum_{i \in J} I^P_{x_{i,j}}(t) x^D_{i,j}(\mu) \end{aligned} \tag{7.19}$$

Die reduzierten Kosten $\bar{z}_t$ einer Spalte $t \in \mathcal{T}$ beziehungsweise des Inklusionsmusters $\iota(t)$ ergeben sich dann durch:

$$\bar{z}_t = -\gamma_C + \sum_{j \in C} \bar{z}_{t,j} \tag{7.20}$$

7.4.2.5 Spaltenkoeffizienten

Sobald wir eine neue Spalte in das Masterproblem einfügen, müssen wir die Koeffizienten aller Nebenbedingungen für diese Spalte richtig setzen. Für die Konvexitätsbedingungen (7.9) ist das relativ einfach, da wir nur den Wert der Indikatorfunktion $I^c(C, t)$ für jede Kette C abfragen müssen.

Der Koeffizient $a_{r,t}$ für eine weitere Nebenbedingung $r \in \mathcal{R}$ von Spalte $t \in \mathcal{T}$ kann folgendermaßen berechnet werden:

$$\begin{aligned} a_{r,t} =& \sum_{i \in J} \sum_{j \in J} I^P_{\pi_{i,j}}(t) I^D_{\pi_{i,j}}(r) \\ &+ \sum_{j \in J} \sum_{p=1}^{|J|} I^P_{\psi_{j,p}}(t) I^D_{\psi_{j,p}}(r) \\ &+ \sum_{i \in J} \sum_{j \in J} I^P_{x_{i,j}}(t) I^D_{x_{i,j}}(r) \end{aligned} \tag{7.21}$$

7.4.3 Set-Cover-Bedingungen

Die Konvexitätsbedingungen (7.9) reichen nicht aus, damit die Auswahl von Inklusionsmustern des Masterproblems sich der Form eines zulässigen Schedules annähert. Bis jetzt haben wir nur sichergestellt, dass jedem Auftrag eine Menge von Vorgängern, und ein direkter Vorgänger zur Verfügung gestellt wird. Das

Masterproblem wird deswegen für jede Kette das günstigste Muster wählen, also eines, in der in jeder Menge von Vorgängern nur die vordefinierten Vorgänger enthalten sind und somit $P_j = P_j^{\min}$ erfüllt ist. Wenn wir uns an das lineare Zuordnungsproblem als untere Schranke erinnern (siehe Abschnitt 6.6.6), haben wir jetzt jedem Auftrag eine Position zugeordnet, aber noch nicht für jede Position mindestens einen Auftrag gefunden. Um zu erzwingen, dass auch für jede Position $p = 1, \ldots, |J|$ ein Auftrag gefunden wird, können wir die neben der Klasse der Konvexitätsbedingungen zweite wesentliche Klasse von Nebenbedingungen für das Masterproblem folgendermaßen definieren:

$$\sum_{j \in J} \psi_{j,p}^P(\lambda) \geq 1 \quad \forall\, p = 1, \ldots, |J| \tag{7.22}$$

Die eigentlich zu erwartende Formulierung $\sum_{j \in J} \psi_{j,p}^P(\lambda) = 1$ als Gleichung wurde hier durch eine Set-Cover-Bedingung ersetzt. Dies ist ohne weiteres möglich, da durch die Gleichheit in den Konvexitätsbedingungen (7.9) auch die Ungleichungen (7.22) immer mit Gleichheit erfüllt sind. Es stehen dann immer noch nur $|J|$ verschiedene Aufträge für $|J|$ verschiedene Positionen zur Verfügung, kein Auftrag kann mehr als eine Position einnehmen. Die Formulierung als Set-Cover-Bedingung erlaubt es uns aber, die zu den Nebenbedingungen (7.22) korrespondierenden dualen Variablen auf positive Werte zu beschränken.

Allgemein sind Set-Cover- oder Set-Partitioning-Formulierungen in Spaltengenerierungsansätzen weit verbreitet, so zum Beispiel auch für das Vehicle-Routing-Problem (VRP, Laporte 1992b).

7.4.4 Schnittebenen

Die Elementarentscheidungen aus Abschnitt 7.4.2 können wir für die Formulierung weiterer Klassen von gültigen Ungleichungen verwenden:

7.4.4.1 Auftragspaare

Die Nebenbedingungen (5.4) aus dem IP-Modell, die für jedes Paar von Aufträgen $(i, j) \in J \times J, i < j$ fordern, dass entweder i als Vorgänger von j eingeplant wird, oder eben j als Vorgänger von i, können wir mit Hilfe der Masterentscheidungen auch in unser Masterproblem übertragen. Für das Masterproblem lautet die Ungleichung dann:

$$\pi_{i,j}^P(\lambda) + \pi_{j,i}^P(\lambda) = 1 \tag{7.23}$$

Diese Gleichungen sind nicht in jeder zulässigen Lösung des RLPM erfüllt, es gilt im allgemeinen Fall

$$0 \leq \pi_{i,j}^P(\lambda) + \pi_{j,i}^P(\lambda) \leq 2. \tag{7.24}$$

Für einen Spaltengenerierungsansatz hat die Formulierung (7.23) jedoch mehrere Nachteile. Einerseits wirkt sich eine einzelne Gleichung dieses Typs nur auf ein Paar von Aufträgen und damit auch nur auf zwei Unterprobleme aus. Damit verbunden ist dann auch eine hohe Anzahl an notwendigen Nebenbedingungen dieses Typs, die den Hang zur primalen Degeneration des Problems verschärfen. Andererseits ist durch die Gleichheit der zulässige Bereich der mit der Ungleichung verbundenen dualen Variablen unbeschränkt und durch eine hohe Anzahl

an Ungleichungen wird die Dimension des dualen Lösungsraumes sehr stark erhöht. Um beiden Problemen entgegenzuwirken, ersetzen wir die Formulierung (7.23) auf scheinbar widersinnige Weise durch eine viel Größere Anzahl an Ungleichungen, von denen wir jedoch dann nur wenige tatsächlich in das Masterproblem übernehmen wollen. Sei dazu $E \subseteq \{(i,j) \in J \times J : i < j\}$ eine beliebige Menge von Auftragspaaren, dann muss die Anzahl aller Vorrangbeziehungen innerhalb dieser Menge mit der Kardinalität von E übereinstimmen. Deshalb ist die Ungleichung

$$\sum_{(i,j)\in E} (\pi^P_{i,j}(\lambda) + \pi^P_{j,i}(\lambda)) \geq |E| \tag{7.25}$$

eine gültige Ungleichung für das Masterproblem. In der Formulierung (7.25) haben wir dadurch, dass die Ungleichung für jede Teilmenge E aller Auftragspaare erfüllt sein muss, die mögliche Anzahl von Ungleichungen diesen Typs ins Exponentielle gesteigert. Durch die Formulierung als Ungleichung ist der Wertebereich für die zugehörigen Variablen auf den positiven Bereich eingeschränkt, es gilt also für eine duale Variable μ_r mit $r \in \mathcal{R}^{SetPrec}$, dass $\mu_r \geq 0$ erfüllt sein muss. Die Richtung der Beschränkung auf Werte $\geq |E|$ forciert für jedes Paar $(i,j) \in E$ die Aufnahme des jeweils anderen Auftrages in die Vorgängermenge, sollte das bis jetzt nicht der Fall gewesen sein. Dabei können ein oder mehrere Paare aus E dazu verwendet werden, die rechte Seite von (7.23) zu überschreiten, damit andere Paare aus der Menge E die rechte Seite von (7.23) unterschreiten dürfen. Wollen wir eine verletzte Ungleichung dieses Typs für eine aktuelle Lösung λ des RLPM generieren, sortieren wir die Kanten $(i,j) \in J \times J, i < j$ nach monoton ansteigenden Werten $(\pi^P_{i,j}(\lambda) + \pi^P_{j,i}(\lambda))$. Die Menge E ist dann gegeben durch die ersten maximal k Paare, die alle die Ungleichung $(\pi^P_{i,j}(\lambda) + \pi^P_{j,i}(\lambda)) \geq 1$ verletzen. Der Wert k stellt in diesem Fall einen Konfigurationsparameter dar, um die Anzahl der mit Nicht-Null-Koeffizienten beteiligten Spalten an der separierten Ungleichung zu beschränken.

7.4.4.2 Vorrangbeziehungen und Positionen

Die Auswertungen der Elementarentscheidungen $\psi^P_{j,p}(\lambda)$ für die Wahl von Position p für Auftrag $j \in J$ kann dazu verwendet werden, auch die Beachtung von Vorrangbeziehungen im Masterproblem zu überprüfen. Für jede Vorrangbeziehung $i \prec j$ muss in jeder Lösung des Masterproblems λ folgende Ungleichung erfüllt sein:

$$\sum_{p=1}^{|J|} p \cdot \psi^P_{j,p}(\lambda) - \sum_{p=1}^{|J|} p \cdot \psi^P_{i,p}(\lambda) \geq 0 \quad \forall i \prec j \tag{7.26}$$

Für die Vorrangbeziehungen innerhalb einer Kette gilt:

Lemma 7.13
Die Ungleichungen (7.26) sind erfüllt für jede zulässige Lösung $\lambda \geq 0$ der LP-Relaxation des Masterproblems bezüglich der Nebenbedingungen (7.9) und (7.22), sofern i und j aus derselben Kette $C \in \mathcal{C}$ der Partitionierung in Ketten $\mathcal{C}$ stammen.

Beweis. Jedes Inklusionsmuster für eine Kette ist dazu gezwungen, für einen späteren Auftrag der Kette auch eine höhere Position zu vergeben. Auch bei einer Mischung der Inklusionsmuster über die Konvexitätsbedingungen (7.9) ist dann eine Verletzung der Ungleichung (7.26) weiterhin unmöglich. □

Die Ungleichungen (7.26) können jedoch von einer zulässigen Lösung $\lambda \geq 0$ bezüglich der Nebenbedingungen (7.9) und (7.22) verletzt werden, sofern i und j nicht aus derselben Kette stammen, aber dennoch eine Vorrangbeziehung zwischen beiden Aufträgen vorliegt. Nehmen wir die Sichtweise aus der Statistik an, so handelt es sich bei den Werten $\psi^P_{j,p}(\lambda)$ für einen festen Auftrag j um eine diskrete Wahrscheinlichkeitsfunktion über die Werte $p = 1, \ldots |J|$ mit Mittelwert $\sum_{p=1}^{|J|} p \cdot \psi^P_{j,p}(\lambda)$. Die Ungleichungen (7.26) fordern dann, dass der Mittelwert dieser Verteilung für Auftrag i kleiner ist als der Mittelwert der Verteilung für Auftrag j. Betrachten wir ein Paar von Aufträgen (i, j) mit $i \prec j$ und eine fest gewählte Position p. Dann lässt sich ein weiterer Satz von gültigen Nebenbedingungen in folgender Weise definieren:

$$\sum_{p'=1}^{p} \psi^P_{j,p'}(\lambda) + \sum_{p'=p}^{|J|} \psi^P_{i,p'}(\lambda) \leq 1 \tag{7.27}$$

Die Ungleichungen (7.27) müssen nicht erfüllt sein, wenn die Ungleichungen (7.26) erfüllt sind. Für eine Vorrangbeziehung zwischen Aufträgen aus der gleichen Kette gilt aber auch hier:

Lemma 7.14
Die Ungleichungen (7.27) sind erfüllt für jede zulässige Lösung $\lambda \geq 0$ der LP-Relaxation des Masterproblems bezüglich der Nebenbedingung (7.9), sofern i und j aus derselben Kette $C \in \mathcal{C}$ der Partitionierung in Ketten $\mathcal{C}$ stammen.

Beweis. Hier kann die gleiche Argumentation wie in Lemma 7.13 verwendet werden. □

7.4.4.3 Gradungleichungen

Die Gradungleichungen (Laporte 1992a) aus dem klassischen Modell von Dantzig, Fulkerson und Selmer M. Johnson (1954) für das Problem des Handelsreisenden (TSP) lassen sich mit Hilfe der Elementarentscheidungen $x^P_{i,j}(\lambda)$, die durch die Wahl der direkten Vorgänger gegeben sind, auch auf unser Masterproblem übertragen. Wie im klassischen TSP gilt dann, dass jeder Auftrag einen direkten Vorgänger braucht:

$$\sum_{i \in J \cup \{0\}} x^P_{i,j}(\lambda) = 1 \quad \forall j \in J \tag{7.28}$$

Lemma 7.15
Die Ungleichungen (7.28) sind erfüllt für jede zulässige Lösung $\lambda \geq 0$ der LP-Relaxation des Masterproblems LPM bezüglich der Nebenbedingung (7.9).

Beweisskizze. Die Konvexitätsbedingungen (7.9) für jede Kette und die Auswahl genau eines direkten Vorgängers für jeden Auftrag einer Kette in einem Inklusionsmuster liefern die Behauptung. □

Da wir nur einen Hamiltonpfad und keine Rundtour suchen, und so nicht jeder Auftrag zwangsweise einen direkten Nachfolger hat (beziehungsweise als direkter Vorgänger für einen Auftrag gewählt wird), können wir hier nur eine

obere Schranke von 1 für die Anzahl der direkten Nachfolger eines Auftrags $i \in J \cup \{0\}$ angeben:

$$\sum_j x^P_{i,j}(\lambda) \leq 1 \forall\, i \in J \cup \{0\} \tag{7.29}$$

Diese Nebenbedingung bildet eine gültige Ungleichung, die sich nicht aus den vorherigen Nebenbedingungen ableiten lässt.

7.4.4.4 Vermeidung von Unterzyklen

Neben den Gradungleichungen können viele weitere für das TSP gültige Ungleichungen auf unser Problem angewendet werden. Die Nebenbedingung zur Vermeidung von Unterzyklen lassen sich hier formulieren als

$$\sum_{(i,j) \in S \times J \setminus S} x^P_{i,j}(\lambda) \geq 1 \forall S \subset J \cup \{0\}, 0 \in S. \tag{7.30}$$

Der minimale Fluss von der Menge S zur Menge $J \setminus S$ muss also mindestens den Wert 1 haben, und das für jede gültige Wahl von S.

Zur Separation dieser Ungleichungen betrachten wir das Netzwerk, das dadurch entsteht, dass wir jedes Paar von Aufträgen $(i, j) \in J \cup \{0\} \times J$, als Kante mit Kapazität $x^P_{i,j}(\lambda)$ betrachten. Gesucht ist dann für jeden Knoten $j \in J$ ein maximaler $0 - j$-Fluss f_j, wobei Knoten 0 als die Quelle des Netzwerks und der Auftrag j als die Senke des Netzwerks betrachtet wird. Ist $f_{j'} < 1$ für einen bestimmten Auftrag $j' \in J$, so haben wir eine verletzte Ungleichung (7.30) gefunden. Dazu Betrachten wir den aus dem Fluss $f_{j'}$ resultierenden $0 - j'$-Schnitt. Alle Aufträge, die im Residualnetzwerk von 0 aus erreichbar sind, bilden dann die Menge S. Zur Lösung der Flussprobleme wird dabei der Highest-Label-Preflow-Push-Algorithmus verwendet (siehe hierzu Ahuja, Magnanti und J. B. Orlin 1993).

Ein weiteres einfacheres Separationsverfahren wird in M. Drexl (2013) beschrieben. Hier werden nur in einem gewissen Sinne verbundene Komponenten betrachtet, so dass es dazu kommen kann, das eigentlich verletzte Subtoureliminationsbedingungen nicht erkannt werden.

7.4.4.5 Unzulässige Pfade

Aus den in Ascheuer, Fischetti und Grötschel (2000) vorgestellten Ungleichungen für das Problem des Handelsreisenden mit harten Zeitfenstern (TSPTW) können wir die Tournament-Version für unzulässige Pfade adaptieren. Sei $\sigma = (\sigma_0 = 0, \sigma_1, \ldots, \sigma_k)$ ein unzulässiger Teilschedule mit k Aufträgen. Um diesen Teilschedule zu verbieten, erstellen wir die Nebenbedingungen

$$\sum_{k'=1}^{k} \sum_{k''=1}^{k'-1} x^P_{\sigma_{k''},\sigma_{k'}}(\lambda) \leq k - 1. \tag{7.31}$$

Hierbei betrachten wir also nicht nur die Kanten, die direkt auf dem unzulässigen Pfad liegen, sondern auch alle Kanten, die im transitiven Abschluss des Pfades liegen. Dann besteht die einzige zulässige Möglichkeit (im Sinne der Konstruktion eines Pfades) zur Auswahl von k oder mehr Kanten immer noch nur in der Auswahl des unzulässigen Teilschedules σ.

Um Ungleichungen dieses Typs zu separieren, werden alle Pfade im durch die Werte $x^P_{i,j}(\lambda)$ induzierten gewichteten Netzwerk enumeriert. Hierzu wird die Tiefensuche verwendet, von einer weiteren Verzweigung eines Knotens kann abgesehen werden, wenn für den aktuellen Teilschedule die Ungleichung (7.31) bereits erfüllt ist. Denn dann bleibt die Ungleichung auch weiterhin erfüllt, wenn der Schedule um einen oder mehrere Knoten erweitert wird. Auf der rechten Seite der Ungleichung wird nämlich für jeden Knoten immer ein Wert von 1 addiert, während auf der linken Seite nur ein Wert von maximal 1 dazukommt. Sobald die Gradungleichungen (7.28) und (7.29) erfüllt sind, ist in dem Netzwerk nur eine polynomielle Anzahl von Pfaden zu untersuchen. Als Quelle für die letzte Aussage wird in Ascheuer, Fischetti und Grötschel (2001) ein persönliches Gespräch mit Savelsbergh angeführt.

7.4.4.6 Generalisierte Clique Ungleichungen

In den vorherigen Abschnitten zu verschiedenen Schnittebenen haben wir gesehen, dass es durchaus einige Elementarentscheidungen gibt, die nicht miteinander vereinbar sind. Betrachten wir zum Beispiel die Gradungleichungen aus Abschnitt 7.4.4.3, so kann aus einer Menge von Elementarentscheidungen (hier die $x^P_{i,j}$ mit festgelegtem i und variabel gewähltem $j \in J$) nur genau eine Elementarentscheidung getroffen werden und damit den Wert 1 annehmen, die anderen Entscheidungen dürfen dann nicht mehr getroffen werden. Erzeugen wir für jede Elementarentscheidung einen Knoten in einem Graphen, und für jedes Paar von unvereinbaren Elementarentscheidungen eine Kante zwischen Knoten des Graphen, so bilden die in einer Gradungleichung aufgeführten Elementarentscheidungen eine Clique in diesem Graphen, da alle diese Entscheidungen paarweise unvereinbar sind. Für die Gradungleichungen mussten wir einerseits deutlich machen, welche Elementarentscheidungen für diese Klasse von Ungleichungen zu berücksichtigen sind, und andererseits einen Separationsalgorithmus angegeben. Nun kann es aber auch eine Clique von Knoten beziehungsweise Elementarentscheidungen geben, für die wir nicht explizit eine Klasse von Ungleichungen angelegt haben, sondern haben nur eine Menge von Paaren von Elementarentscheidungen bestimmt, in der jedes Paar zwei unvereinbare Elementarentscheidungen darstellt. Um automatisch eine eventuell verletzte Ungleichung zu erzeugen, müssen wir nur eine Clique von Knoten angeben, die in der Summe der Knotengewichte den Schwellwert 1 überschreitet. Die soeben genannten Knotengewichte ergeben sich aus den aktuellen Werten $\pi^P_{j,i}(\lambda)$, $\psi^P_{j,p}(\lambda)$ und $x^P_{i,j}(\lambda)$ für die einzelnen Elementarentscheidungen über die Wahl von Vorgängern, Positionen und direkten Vorgängern. Eine maximal verletzte Ungleichung ergibt sich dann aus der Konstruktion einer Clique mit maximalem Knotengewicht in diesem Graphen. Das Separationsproblem ist damit ein $\mathcal{NP}$-schweres Optimierungsproblem. Diese Technik zur Erzeugung von Ungleichungen wird auch in Softwarepaketen zur Ganzzahligen Programmierung angewendet, dabei muss dann auch vorher automatisch festgestellt werden, welche Binärvariablen des betrachteten Modells miteinander unvereinbar sind (siehe dazu auch Guignard und Spielberg 1981; E. L. Johnson und Padberg 1982; Escudero, Garín und Pérez 1996). In Atamtürk, George L. Nemhauser und Savelsbergh (2000) werden Aspekte der Implementierung von generalisierten Cliqueungleichungen betrachtet.

7.5 Generierung von Spalten

Um einen Spaltengenierungsansatz anwenden zu können, müssen wir in der Lage sein, Spalten mit negativen reduzierten Kosten zu finden, oder zu zeigen, dass es solche nicht geben kann und die LP-Relaxation des Masterproblems damit bereits optimal gelöst ist. In unserem Fall müssen wir also unter allen Inklusionsmustern die mit den geringsten negativen Kosten herausfiltern. Das dieses Vorhaben nicht sonderlich einfach ist, zeigt das folgende Lemma:

Lemma 7.16
Für eine Kette $C \in \mathcal{C}$ ist das zugrunde liegende Pricing-Problem $\mathcal{NP}$-schwer im strengen Sinne.

Beweisskizze. Die Deadlines für die Aufträge der Kette, die Prozesszeiten der Aufträge, die Vorrangbeziehungen sowie die dualen Preise $\pi^D_{i,j}(\mu)$ definieren ein um Vorrangbeziehungen erweitertes Rucksackproblem, das $\mathcal{NP}$-schwer im strengen Sinne ist (Garey und D. S. Johnson 1979, Problem MP12). Ein zweite Reduktion ist möglich über die 3-Partitionierung, erzeuge hierzu für jede Zahl $a \in A$ einen fremden Auftrag mit Prozesszeit $p_a = s(a)$, und erzeuge für die Kette m verschiedene Aufträge mit $p_j = 1$ und $w_j = 1$ und fixiere deren Freigabe- und Fälligkeitszeitpunkt derart, dass keine andere Fertigstellungszeit als die Fälligkeit möglich ist. Erzeuge dann für jede vierte Position einen dualen Preis, der die Fertigstellungszeit des dazugehörigen Auftrags aus der Kette wieder aufhebt. □

Aus diesem Grund sind wir auch für dieses Problem auf spezialisierte exakte Optimierungsalgorithmen angewiesen, die in den folgenden Abschnitten vorgestellt werden.

7.5.1 Optimierungsmodell für das Pricing-Problem

Für die einfache Zeitabschätzung (siehe dazu Abschnitt 7.3.2.1), können wir das folgende gemischt-ganzzahlige Optimierungsmodell formulieren:

- Sei $\psi_{j,p} \in \{0,1\}$ eine Binärvariable für die Auswahl von Position p für Auftrag $j \in C$
- Sei $\pi_{i,j} \in \{0,1\}$ eine Binärvariable für die Auswahl von Auftrag $i \in J$ als Vorgänger von $j \in J$.
- Sei $x_{i,j} \in \{0,1\}$ eine Binärvariable für die Auswahl von $i \in J$ als direkten Vorgänger für $j \in C$
- Sei $r_j + p_j \leq C_j \leq \delta_j$ die untere Schranke für die Fertigstellungszeit von Auftrag $j \in C$.
- Sei $r_i + p_i \leq \tilde{C}_i \leq \delta_i$ die untere Schranke für die Fertigstellungszeit von Auftrag $i \in J$, wenn i als direkter Vorgänger für einen beliebigen Auftrag $j \in C$ ausgewählt wird

Zusätzliche Hilfsvariablen geben für jeden Auftrag $j \in J$ die anfallenden Rüstzeiten pro rüstrelevantem Material und den Zeitpunkt der Materialverfügbarkeit

zur Produktion des Auftrags an.

$$\begin{aligned} \min -\gamma_C + & \sum_{j \in C} w_j \cdot C_j \\ & - \sum_{j \in C} \sum_{p=1}^{|J|} \psi_{i,j}^D(\mu) \cdot \psi_{j,p} \\ & - \sum_{j \in C} \sum_{i \in J} \pi_{i,j}^D(\mu) \cdot \pi_{i,j} \\ & - \sum_{j \in C} \sum_{i \in J} x_{i,j}^D(\mu) \cdot x_{i,j} \end{aligned} \tag{7.32}$$

Die Zielfunktion berechnet die reduzierten Kosten mit Hilfe der dualen Lösung (γ, μ) des RLPM unter Verwendung der drei Variablengruppen $\psi_{j,p}$, $\pi_{i,j}$ und $x_{i,j}$, die die elementaren Entscheidungen eines Inklusionsmusters widerspiegeln. Die mit den Konvexitätsbedingungen (7.9) assoziierten dualen Variablen γ bilden dabei die Konstante der Zielfunktion.

Die Nebenbedingungen des Modells:

$$\sum_{p} \psi_{j,p} = 1 \forall j \in C \tag{7.33}$$

Für jeden Auftrag $j \in C$ aus der Kette muss eine Position festgelegt werden.

$$\sum_{p} p \cdot \psi_{j,p} = \sum_{i \in J} \pi_{i,j} \forall j \in C \tag{7.34}$$

Die Position des Auftrags ist dabei gegeben durch die Kardinalität der Menge von Vorgängern.

$$\sum_{i \in J} x_{i,j} = 1 \forall j \in C \tag{7.35}$$

Für jeden Auftrag $j \in C$ in der Kette muss ein zulässiger direkter Vorgänger bestimmt werden.

$$x_{i,j} \leq \pi_{i,j} \forall i \in J, j \in C \tag{7.36}$$

Ein direkter Vorgänger $i \neq 0$ für j muss auch in der Menge von Vorgängern von j enthalten sein.

$$x_{i,j} + \pi_{l,i} - \pi_{l,j} \leq 1 \quad \forall i \in J, j \in C, l \in J, l \neq j \tag{7.37}$$

$$x_{i,j} - \pi_{l,i} + \pi_{l,j} \leq 1 \quad \forall i \in J, j \in C, l \in J, l \neq j \tag{7.38}$$

Die Nebenbedingungen (7.37) und (7.38) erzwingen eine Übereinstimmung der Menge von Vorgängern für einen direkten Vorgänger $i \in C$ für $j \in C$ und der Menge von Vorgängern von j bis auf j (vergleiche Lemma 7.6).

Soll der initiale Rüstzustand 0 als direkter Vorgänger für einen Auftrag der Kette ausgewählt werden, müssen wir sicherstellen, dass nur der Auftrag j in der Menge von Vorgängern von j (vergleiche Lemma 7.6) enthalten ist:

$$x_{0,j} + \pi_{i,j} \leq 1 \forall j \in C, i \in J, i \neq j \tag{7.39}$$

Die Auswahl von $i \notin C$ als direkten Vorgänger für j_{k+1} ist nur dann möglich, wenn i nicht bereits in einer Menge von Vorgängern für ein früheren Auftrag $j_k, k > 1$ der Kette enthalten ist:

$$x_{i,j_{k+1}} + \pi_{i,j_k} \leq 1 \forall i \notin C \tag{7.40}$$

$$\pi_{i,j} \leq \pi_{i',j} \forall i, i' \in J, i' \prec i, j \in C \tag{7.41}$$

Wird ein Auftrag i in die Menge der Vorgänger von $j \in C$ aufgenommen, so müssen auch alle Vorgänger von i in diese Menge aufgenommen werden, vergleiche hierzu die gültige Ungleichung (5.12) aus Kapitel 5 und Lemma 7.2.

$$\pi_{i,j} \leq \pi_{i,j'} \forall i \in J, j, j' \in C, j \prec j' \tag{7.42}$$

Die Mengen von Vorgängern für zwei Aufträge aus der Kette müssen verschachtelt sein, siehe dazu die gültige Ungleichung (5.13). Die Nebenbedingungen

$$\sum_{i \in J} a_{i,m} \cdot \pi_{i,j} \leq \sum_{\tau \in T} R_{m,\tau} \cdot \rho_{j,\tau} \quad \forall m \in M, j \in J \tag{7.43}$$

und

$$\sum_{\tau \in T} \rho_{i,\tau} = 1 \quad \forall i \in J \tag{7.44}$$

stellen sicher, dass für jeden Auftrag ein passendes Materialverfügbarkeitslevel ausgewählt wird.

Die Fertigstellungszeit für jeden Auftrag $j \in C$ der Kette ist dann nach unten beschränkt durch die Zeit der Materialverfügbarkeit:

$$\sum_{\tau \in T} \tau \cdot \rho_{j,\tau} + p_j \leq C_j \tag{7.45}$$

Dies gilt auch für die minimale Fertigstellungszeit als möglicher direkter Vorgänger:

$$\sum_{\tau \in T} \tau \cdot \rho_{i,\tau} + p_i \leq \tilde{C}_i \forall i \in J \tag{7.46}$$

Die minimale Rüstzeit wird anhand der ausgewählten Vorgänger bestimmt:

$$\sigma_{j,m} \geq st(i,j,m) \cdot \pi_{i,j} \forall j \in J \tag{7.47}$$

Die Fertigstellungszeit eines möglichen direkten Vorgängers j für einen Auftrag aus der Kette ist dann durch die Wahl aller Vorgänger für j und den damit verbunden minimalen Rüstaufwand bestimmt, vergleiche hierzu auch die gültige Ungleichung (5.11) aus Kapitel 5:

$$\tilde{C}_j \geq \sum_{i \in J} p_i \cdot \pi_{i,j} + \sum_{m \in M} \sigma_{j,m} - \sum_{G \in \mathcal{G}} st(G) \forall j \in J \tag{7.48}$$

Die Fertigstellungszeit des gewählten direkten Vorgängers i für einen Auftrag $j \in C$ aus der Kette beeinflusst dann auch die Fertigstellungszeit des Auftrags j, vergleiche hierzu die Nebenbedingung (5.5) aus der Manne-Formulierung (Manne 1960):

$$x_{i,j} = 1 \Rightarrow C_j \geq \tilde{C}_i + st(i,j) + p_j \forall j \in C \tag{7.49}$$

Die Nebenbedingung (5.11) beziehungsweise (7.48) kann auch für alle Aufträge $j \in C$ aus der Kette als zwar nicht notwendige, aber dennoch gültige Ungleichung verwendet werden.

$$C_j \geq \sum_{i \in J} p_i \cdot \pi_{i,j} + \sum_{m \in M} st(G(m)) \cdot \sigma_{j,m} - \sum_{G \in \mathcal{G}} st(G) \forall j \in C \tag{7.50}$$

Wie auch in der Formulierung des gemischt-ganzzahligen Optimierungsmodells aus Kapitel 5 gilt auch hier, dass die Variablen der Fertigstellungszeiten immer einen ganzzahligen Wert in der optimalen Lösung annehmen.

7.5.2 Spaltenerzeugung mit Branch-and-Bound

Um Spalten für das Masterproblem zu erzeugen, ist der eben gerade vorgestellte Ansatz der gemischt ganzzahligen Programmierung wenig geeignet, da von einem IP-Solver nur wenige Spalten erzeugt werden, und der Lösungprozess aufgrund der Komplexität des Optimierungsproblems zu lange dauert. Im Folgenden werden darum drei verschiedene kombinatorische Branch-and-Bound-Verfahren vorgestellt. Diese basieren in großen Teilen auf den gleichen Prinzipien, die Enumeration des Lösungsraumes wird jedoch auf unterschiedliche Weise durchgeführt. In den verschiedenen Ansätzen wollen wir systematisch alle Inklusionsmuster untersuchen, müssen also insbesondere in der Lage sein, jede Folge von zulässigen Mengen von Vorgängern zu erzeugen. Das folgende Lemma zeigt, dass es auf jeden Fall immer zwei (nicht notwendigerweise verschiedene) Reihen von zulässigen Mengen von Vorgängern gibt:

Lemma 7.17
Für jede Kette von Aufträgen $C = (j_1, \ldots, j_{|C|})$ gilt:

$$P_{j_1}^{\min} \subset \ldots \subset P_{j_k}^{\min} \subset \ldots \subset P_{j_{|C|}}^{\min} \tag{7.51}$$

ist eine Folge von zulässigen Mengen von Vorgängern und

$$P_{j_1}^{\max} \subset \ldots \subset P_{j_k}^{\max} \subset \ldots \subset P_{j_{|C|}}^{\max} \tag{7.52}$$

ist eine Folge von zulässigen Mengen von Vorgängern.

Beweis. Seien j und j' zwei Aufträge aus Kette C mit $j \prec j'$. Die Teilmengenbeziehung ist strikt, da in jedem Fall $j' \in P_{j'}^{\min\,/\,\max}$, aber $j' \notin P_j^{\min\,/\,\max}$ gilt. Für jeden Auftrag $i \in J$ mit $i \in P_j^{\min}$ gilt auch $i \in P_{j'}^{\min}$, und für jeden Auftrag $i \in J$ mit $i \notin P_{j'}^{\max}$ gilt auch $i \notin P_j^{\max}$. □

Die Aussage des Lemmas 7.17 wollen wir in allen Knoten ν unserer Branch-and-Bound-Verfahren erfüllen. Sei dazu $P_j^{\min}(\nu)$ die minimale und $P_j^{\max}(\nu)$ die maximale Menge von Vorgängern von $j \in C$ im Knoten ν eines Branch-and-Bound-Verfahrens.

Definition 7.18 (Invarianten der Branch-and-Bound-Verfahren)
Für jeden Knoten ν im Branch-and-Bound-Baum seien für die betrachtete Kette $C = (j_1, \ldots, j_k, \ldots, j_{|C|})$ die Folge

$$P_{j_1}^{\min}(\nu) \subset \ldots \subset P_{j_k}^{\min}(\nu) \subset \ldots \subset P_{j_{|C|}}^{\min}(\nu) \tag{7.53}$$

das kanonische Inklusionsmuster $\iota(\nu)$ im Knoten ν, und bildet eine Folge von zulässigen Mengen von Vorgängern für die Aufträge aus C. Auch die Folge

$$P_{j_1}^{\max}(\nu) \subset \ldots \subset P_{j_k}^{\max}(\nu) \subset \ldots \subset P_{j_{|C|}}^{\max}(\nu) \tag{7.54}$$

sei immer eine Folge von zulässigen Mengen von Vorgängern. Zusätzlich gelte für jeden Knoten ν und jeden beliebigen Ahnen ν' von ν für jeden Auftrag $j \in C$ die folgenden Beziehungen:

$$P_j^{\min}(\nu) \supseteq P_j^{\min}(\nu') \tag{7.55}$$

und

$$P_j^{\max}(\nu) \subseteq P_j^{\max}(\nu') \tag{7.56}$$

In den Branch-and-Bound-Bäumen wollen wir somit mindestens immer eine Reihe von zulässigen Mengen von Vorgängern vorliegen haben, und die Verzweigungsentscheidungen fügen entweder Knoten zur minimalen Menge von Vorgängern $P_j^{\min}(\nu')$ eines Auftrags j hinzu (7.55), oder entfernen Knoten aus der maximalen Menge von Vorgängern $P_j^{\max}(\nu')$ eines Auftrags j im Vaterknoten ν' (7.56). Natürlich kann es auch vorkommen, dass einige oder alle Mengen von einer Verzweigungsentscheidung unbeeinflusst bleiben. In einem Knoten ν des Branch-and-Bound-Baumes bezeichnen wir eine Menge von Vorgängern für Auftrag $j \in C$ als **fixiert**, sobald $P_j^{\min}(\nu) = P_j^{\max}(\nu) =: P_j(\nu)$ gilt. Des Weiteren bezeichnen wir die Menge $P_j(\nu)$ als **versiegelt**, sobald sie fixiert ist und zusätzlich ein zulässiger direkter Vorgänger $x_j(\nu) \in P_j(\nu)$ für den Auftrag $j \in C$ bestimmt wurde. Die Bestimmung eines zulässigen direkten Vorgängers $x_j(\nu)$ für einen Auftrag j_k, mit $k > 1$ ist nur möglich, wenn auch die Menge $P_{j_{k-1}}(\nu)$ bereits fixiert und versiegelt ist. Darum ist die grundsätzliche Strategie der Branch-and-Bound-Verfahren, der Reihe nach die Mengen von Vorgängern für die Aufträge zunächst zu fixieren, und dann im nächstmöglichen Verzweigungsschritt zu versiegeln. Die Festlegung eines direkten Vorgängers $x_{j_k}(\nu)$ in einem Knoten ν für einen Auftrag $j_k \in C$ aus der Kette hat in der abhängigen Zeitabschätzung (siehe Abschnitt 7.3.2.2) auch einen Einfluss auf die Zeitabschätzung aller Nachfolger $j'_k \in C$ mit $k' > k$, und damit auf die Kosten eines Inklusionsmusters.

7.5.2.1 Verzweigungsentscheidungen

Um verschiedene Wege der Enumeration aller Inklusionsmuster für eine Kette C zu verdeutlichen, führen wir neben der Versiegelung als Wahl eines direkten Vorgängers noch zwei weitere Arten von Verzweigungsentscheidungen ein. Dabei sei ν' der Vaterknoten des aktuell erzeugten Knotens ν.

Fixierung fremder Ketten Sei $C' \in \mathcal{C}$ mit $C' \neq C$ eine weitere Kette von Aufträgen und betrachte die Schnitte $(C' \cap P_j^{\min}(\nu')) \subseteq (C' \cap P_j^{\max}(\nu'))$ für einen bestimmten Auftrag $j \in C$. Ist diese Teilmengenbeziehung streng, somit $|C' \cap P_j^{\min}(\nu')| < |C' \cap P_j^{\max}(\nu')|$, können wir in den neu erzeugten Knoten $\nu_0, \ldots \nu_m$ mit $m = |C' \cap P_j^{\max}(\nu')| - |C' \cap P_j^{\min}(\nu')|$ die Schnittmengen $C' \cap P_j^{\max}(\nu_l)$ und $C' \cap P_j^{\max}(\nu_l)$ mit $l = 0, \ldots, m$ auf die ersten $|C' \cap P_j^{\min}(\nu')| + l$ Aufträge von Kette C' fixieren. Für den Extremfall $C' \cap P_j^{\min}(\nu') = \emptyset$ und $C' \cap P_j^{\max}(\nu') = C'$ erhalten wir mit dieser Verzweigungsentscheidung $|C'| + 1$

Kindknoten $\nu_0, \ldots, \nu_{|C'|}$. Um die Invariante aus Definition 7.18 zu erfüllen, wird bei jeder Ausführung der Entscheidung, die Anzahl fremder Aufträge aus Kette C' auf k' festzulegen, auf einen Knoten ν und alle sein Nachfahren folgendes Vorgehen angewendet: Für $k' > 0$ und alle Aufträge j_l mit $l = k, \ldots, |C|$ setze $P^{\min}_{j_l}(\nu) = P^{\min}_{j_l}(\nu) \cup \{i \in J : i \prec i_{k'}\}$ und für $k' < |C'|$ und alle Aufträge j_l mit $l = 1, \ldots, k$ setze $P^{\max}_{j_l}(\nu) = P^{\max}_{j_l}(\nu) \backslash \{i \in J, i_{k'+1} \prec i\}$.

Fixierung des ersten Kettennachfolgers Betrachten wir eine Folge von zulässigen Mengen von Vorgängern $P_{j_1} \subset \ldots \subset P_{j_{|C|}}$, ist ein fremder Auftrag $i \notin C$ entweder in $P_{j_{|C|}}$ enthalten oder nicht. Gilt $i \in P_{j_{|C|}}$, so ist der Auftrag j_k mit $k = \min\{1 \leq k' \leq |C| : i \in P_{j_{k'}}\}$ der erste Auftrag aus der Kette C, vor dem Auftrag i gefertigt werden soll, und somit der erste Nachfolger von i aus Kette C. Die hier vorgestellte Branchingentscheidung wählt für einen Auftrag $i \notin C$ den Kettennachfolger $j \in C$ von i aus, oder legt fest, dass i gar nicht Teil des Inklusionsmusters sein soll. Wir erhalten darum maximal $|C|+1$ Kindknoten, die jeweils einen anderen (oder keinen) Auftrag der Kette als ersten Nachfolger für i festlegen. Um die Invariante aus Definition 7.18 zu erfüllen, werden bei jeder Entscheidung dieser Art die minimalen und maximalen Mengen von Vorgängern im Knoten ν und allen Nachfahren wie folgt angepasst. Für $l = k, \ldots, |C|$ wird die Menge $P^{\min}_{j_l}(\nu)$ auf $P^{\min}_{j_l}(\nu) \cup \{i' \in J : i' \prec i\}$ gesetzt, der Auftrag i und seine Vorgänger werden also integriert, und für $l = 1, \ldots, k-1$ die Menge $P^{\max}_{j_l}(\nu)$ auf $P^{\max}_{j_l}(\nu) \backslash \{i' \in J : i \prec i'\}$ gesetzt, also Auftrag i und seine Nachfolger ausgeschlossen.

Versiegelung Ist ein Auftrag $j_k \in C$ der erste Auftrag der Kette, also $k = 1$ oder bereits die Menge von Vorgängern von j_{k-1} versiegelt und P_{j_k} fixiert, kann ein direkter zulässiger Vorgänger gewählt werden. Hierzu gibt es maximal $|\mathcal{C}| - 1$ Möglichkeiten. Die Versiegelung einer Menge von Vorgängern für einen Auftrag $j \in C$ in einem Knoten ν durch Wahl des zulässigen direkten Vorgängers i bezeichnen wir mit $\Phi^{seal}(j, i, \nu)$. Jeder Versiegelungsknoten weist zwei zusätzliche Eigenschaften auf, nämlich Kosten $c^{seal}(\nu) = \sum_{l=1}^{k} \bar{c}(P_{j_k})$ und Zeit $t^{seal}(\nu) = C^{LB}_{j_k}(P_{j_k}(\nu))$ der Versiegelung. Eine Versiegelungsentscheidung hat keine Auswirkung auf die Mengen $P^{\min}_j(\nu)$ und $P^{\max}_j(\nu)$ für beliebige $j \in C$.

7.5.2.2 Anpassung des Vorranggraphen

Durch ein Inklusionsmuster ι für eine Kette $C \in \mathcal{C}$ werden immer auch Vorrangbeziehungen zwischen Aufträgen aus der Kette und den anderen Aufträgen ausgedrückt. So gilt für einen Auftrag $j \in C$ und für jeden Auftrag $i \in P_j(\iota) \backslash j$ auch $i \prec_\iota j$, sowie für jeden Auftrag $i \in J \backslash P_j(\iota)$ auch $j \prec_\iota i$. Zusammen mit den Vorrangbeziehungen, die durch den Vorranggraphen G beziehungsweise seine transitive Reduktion G^R gegeben sind, entsteht dann ein neuer Vorranggraph $G(\iota)$ beziehungsweise $G^R(\iota)$. Anhand dieses Vorranggraphen können dann mögliche direkte Vorgänger sowie eine Abschätzung für die mindestens anfallende Rüstzeit vor der Produktion eines Auftrages aus der Kette bestimmt werden. Die transitive Reduktion $G^R(\iota)$ für ein Inklusionsmuster ι lässt sich durch eine Einschränkung der Knotenmenge von G^R auf die Mengen $P_{j_k} \backslash (P_{j_{k-1}} \cup \{j\})$ für $k = 1, \ldots, |C|$ bestimmen. Diese Einschränkung von G^R auf ein Teilmenge der Aufträge $S \subseteq J$ soll mit $G^R|_S$ bezeichnet werden. $G^R(\iota)$ enthält für

$k = 1, \ldots, |C|$ jeweils den Graphen $G^R|_{P_{j_k} \setminus (P_{j_{k-1}} \cup \{j\})} =: G^R_{j_k}(\iota)$. Zusätzlich werden alle Senken in $G^R_{j_k}(\iota)$ mit dem Auftrag $j_k \in C$ verbunden, und alle Quellen bekommen eine eingehende Kante ausgehend vom Auftrag $j_{k-1} \in C \cup \{0\}$. Ist $P_{j_k} \setminus (P_{j_{k-1}} \cup \{j\})$ leer, wird j_{k-1} direkt mit j_k verbunden. Durch diese Konstruktionsweise wird auch der Umstand verdeutlicht, dass ein Inklusionsmuster für jeden Auftrag $i \notin C$ eine Einordnung zwischen den Aufträgen der Kette C vornimmt.

Für einen beliebigen Knoten ν des Branch-and-Bound-Verfahrens zur Bestimmung von Inklusionsmustern mit reduzierten Kosten ist für die einzelnen Aufträge aus $i \notin C$ jedoch diese Einordnung noch nicht vollständig vorgegeben. Anstelle der Menge der Vorgänger $P_j(\iota)$ für einen Auftrag $j \in C$ ist jetzt nur ein grober Rahmen zwischen $P_j^{\min}(\nu)$ und $P_j^{\max}(\nu)$ für die Menge der Vorgänger von $j \in C$ bekannt. Dennoch sind wir auch hier auf die transitive Reduktion des Vorranggraphen $G^R(\nu)$ im Knoten ν angewiesen, um die aktuelle Abschätzung der anfallenden Rüstzeiten zu verbessern, sowie die möglichen direkten Vorgänger innerhalb des kanonischen Inklusionsmusters $\iota(\nu)$ zu bestimmen.

Zur Bestimmung der transitiven Reduktion $G^R(\nu)$ wollen wir zunächst Kanten aus G^R auf ihr weiteres Vorliegen in $G^R(\nu)$ überprüfen, und zwar für die beiden Fälle, in denen für $(i,j) \in E(G^R)$ entweder $i \notin C \wedge j \notin C$ (Fall 1) oder $i \in C \wedge j \in C$ (Fall 2) gilt.

Lemma 7.19
Es gelte $(i,j) \in E(G^R)$.

Fall 1: *Es gelte zusätzlich: $i \notin C \wedge j \notin C$. Dann ist $(i,j) \in E(G^R(\nu))$, wenn kein $l \in C$ existiert mit $j \notin P_l^{\max}(\nu) \wedge i \in P_l^{\min}(\nu)$.*

Fall 2: *Es gelte zusätzlich: $i \in C \wedge j \in C$. Dann ist $(i,j) \in E(G^R(\nu))$ wenn kein $l \notin C$ existiert mit $l \in P_j^{\min}(\nu) \setminus P_i^{\max}(\nu)$.*

Beweis. Durch Widerspruch. Wenn in beiden Fällen jeweils die Existenz des Auftrags $l \in J$ mit den genannten Eigenschaften angenommen wird, lässt sich der Pfad $i \prec_\nu l \prec_\nu j$ erzeugen. Damit kann (i,j) nicht mehr Kante in der transitiven Reduktion $G^R(\nu)$ sein. □

Die Kanten zwischen zwei fremden beziehungsweise eigenen Aufträgen aus der originalen transitiven Reduktion G^R bleiben also nur dann erhalten, wenn wir keinen Auftrag aus der Kette C (Fall 1) oder keinen fremden Auftrag (Fall 2) dazwischen geschoben haben. Für die teilweise neu definierten Vorrangbeziehungen zwischen Aufträgen aus der Kette C und den übrigen Aufträgen führen wir zur Ermittlung der Kanten in $G^R(\nu)$ für $(i,j) \in E(G^R(\nu))$ mit $i \in C \wedge j \notin C$ (Fall 3) oder $i \notin C \wedge j \in C$ (Fall 4) die folgende Notation ein:

Definition 7.20 (Erster Vorgänger und Nachfolger)
Sei $i \notin C$ und es gelte $i \notin P_j^{\max}(\nu)$ für mindestens einen Auftrag $j \in C$. Dann ist $FP_i(\nu)$ derjenige Auftrag $j \in C$, für den $i \notin P_j^{\max}(\nu)$ erfüllt ist, und für alle weiteren Aufträge $j' \in C$ mit $i \notin P_{j'}^{\max}(\nu)$ gilt $j' \prec j$. j ist damit der erste Vorgänger *aus C von i im Knoten ν.*

Sei $i \notin C$ und es gelte $i \in P_j^{\min}(\nu)$ für mindestens einen Auftrag $j \in C$. Dann ist $FS_i(\nu)$ derjenige Auftrag $j \in C$, für den $i \in P_j^{\min}(\nu)$ erfüllt ist, und für alle weiteren Aufträge $j' \in C$ mit $i \in P_{j'}^{\min}(\nu)$ gilt $j \prec j'$. j ist damit der erste Nachfolger *aus C von i im Knoten ν.*

Lemma 7.21
Es gelte $i \notin C$.

Fall 3: *Es gebe zusätzlich ein $j \in C$ mit $i \notin P_j^{\max}(\nu)$. Dann ist die Kante $(FP_i(\nu), i) \in E(G^R(\nu))$, wenn kein Auftrag $i' \notin C$ existiert mit $i' \prec i$ und $FP_{i'}(\nu) = FP_i(\nu)$.*

Fall 4: *Es gebe zusätzlich ein $j \in C$ mit $i \in P_j^{\min}(\nu)$. Dann ist die Kante $(i, FS_i(\nu)) \in E(G^R(\nu))$, wenn kein Auftrag $i' \notin C$ existiert mit $i \prec i'$ und $FS_{i'}(\nu) = FS_i(\nu)$.*

Beweis. Durch Widerspruch. Wenn in beiden Fällen die Existenz von i' angenommen wird, lässt sich mit $FP_i(\nu) = FP_{i'}(\nu) \prec_\nu i' \prec i$ (Fall 3) beziehungsweise $i \prec i' \prec_\nu FS_{i'}(\nu) = FS_i(\nu)$ ein Pfad erzeugen. Damit können die Kanten $(FP_i(\nu), i)$ beziehungsweise $(i, FS_i(\nu))$ nicht mehr Teil der transitiven Reduktion sein. □

Die Bestimmung dieser ersten Vorgänger $FP_i(\nu)$ und Nachfolger $FS_i(\nu)$ aus C für Aufträge $i \notin C$ erlaubt es somit, die Bestimmung der transitiven Reduktion des Vorranggraphen $G^R(\nu)$ auf effiziente Weise abzuschließen.

7.5.2.3 Dominanzkriterium

Diejenigen Knoten im Branch-and-Bound-Baum, die eine Versiegelungsentscheidung repräsentieren, erlauben die Anwendung eines Dominanzkriteriums.

Lemma 7.22
Seien ν_r und ν_s zwei Knoten, die eine Versiegelungsentscheidung für den gleichen Auftrag $j_k \in C$ treffen. Dann dominiert ν_r den Knoten ν_s wenn folgende fünf Bedingungen erfüllt sind:

1. $P_{j_k}(\nu_r) = P_{j_k}(\nu_s)$
2. $t^{seal}(\nu_r) \leq t^{seal}(\nu_s)$
3. $c^{seal}(\nu_r) \leq c^{seal}(\nu_s)$
4. $P_{j_{k'}}^{\min}(\nu_r) \subseteq P_{j_{k'}}^{\min}(\nu_s) \forall k' = k+1, \ldots, |C|$
5. $P_{j_{k'}}^{\max}(\nu_s) \subseteq P_{j_{k'}}^{\max}(\nu_r) \forall k' = k+1, \ldots, |C|$

Beweis. Ein Knoten ν_r kann einen Knoten ν_s nur dann dominieren, wenn alle Lösungen, die von ν_r repräsentiert werden, nicht schlechter sind als alle Lösungen, die von ν_s repräsentiert werden. Nun erfüllen aber durch die Voraussetzungen 4 und 5 alle möglichen Mengen von Vorgängern $P_{j_{k'}}(\nu_s)$ für Knoten ν_s mit $k' > k$ und $P_{j_{k'}}^{\min}(\nu_s) \subseteq P_{j_{k'}}(\nu_s) \subseteq P_{j_{k'}}^{\max}(\nu_s)$ auch $P_{j_{k'}}^{\min}(\nu_r) \subseteq P_{j_{k'}}(\nu_s) \subseteq P_{j_{k'}}^{\max}(\nu_r)$. Es sind damit ab j_{k+1} mindestens die gleichen Erweiterungen zu einem vollständigen Inklusionsmuster möglich. Jede zeitlich zulässige Erweiterung im von ν_s erzeugten Teilbaum ist durch Bedingung 2 auch zeitlich zulässig im Teilbaum von ν_r. Durch Bedingung 3 sind die Kosten jeder Erweiterung auf ein vollständiges Inklusionsmuster im Teilbaum von ν_r auch nicht schlechter als im Teilbaum von ν_s. □

Wird die Versiegelung einer fixierten Menge von Vorgängern vorgenommen, erfüllen die entstehenden Kindknoten untereinander automatisch die Bedingungen 1, 4 und 5 des Dominanzkriteriums. Dies erlaubt eine lokale Anwendung des Dominanzkriteriums, indem diese Kindknoten hinsichtlich ihrer Versiegelungszeiten und -kosten miteinander verglichen werden.

7.5.2.4 Berechnung von unteren Schranken

Um eine untere Schranke für einen Knoten ν zu erhalten, müssen wir die reduzierten Kosten der noch nicht versiegelten Mengen von Vorgängern P_j mit $P_j^{\min} \subseteq P_j \subseteq P_j^{\max}$ auf gültige Weise unterschätzen. Die reduzierten Kosten einer Menge von Vorgängern für einen Auftrag $j \in C$ ergeben sich laut Abschnitt 7.4.2.4 aus der minimalen Fertigstellungszeit des Auftrags, der Position des Auftrags, den Aufträgen in der Menge von Vorgängern sowie der Wahl des direkten Vorgängers.

Abschätzung der besten Vorgängermenge Um eine höhere, und damit eventuell lukrativere Position im Schedule für einen Auftrag $j \in J$ zu erreichen, müssen wir einen Anstieg in der Fertigstellungszeit des Auftrags in Kauf nehmen. Bestünde nur ein Trade-off bezüglich dieser beiden Größen, kann der Ansatz der Dynamischen Programmierung aus Abschnitt 4.3.3 verwendet werden, um die minimale Fertigstellungszeit pro noch erreichbarer Position p mit dem dualen Preis $-\psi_{j,p}^D(\mu)$ für diese Position zu verrechnen. Hierbei wird für die Aufnahme weiterer Aufträge zusätzlich zu denen in $P_j^{\min}(\nu)$ nur auf die Vorrangbeziehungen, die durch die vorliegende Partitionierung in Ketten $\mathcal{C}$ abgedeckt sind, Rücksicht genommen. Die dualen Preise $-\pi_{i,j}^D(\mu)$ für die Aufnahme eines Vorgängers i in die Menge von Vorgängern sowie der duale Preis $-x_{i,j}^D(\mu)$ für die Wahl von direkten Vorgängern erschweren jetzt jedoch die Schrankenberechnung. Damit die Schranke nicht zu schwach wird, müssen wir versuchen, die dualen Preise $-\pi_{i,j}^D(\mu)$ gegen den für die Aufnahme des Auftrages i in die Menge von Vorgängern von j anfallenden Anstieg der Fertigstellungszeit von j abzuschätzen. Dazu erweitern wir den Ansatz der Dynamischen Programmierung aus Abschnitt 4.3.3. Hierfür wird ein Zustand $\Pi(p,t,c)$ jetzt definiert als der minimale duale Preis für die Einbettung von p zusätzlichen Aufträgen mit minimaler Fertigstellungszeit t, sobald die ersten c Ketten aus der Partitionierung $\mathcal{C}$ im Algorithmus Berücksichtigung gefunden haben. In diesem Ansatz ist ein Zustand $\Pi(p,t,c)$ erst dann von einem Zustand $\Pi'(p,t',c)$ dominiert, wenn neben $t' \leq t$ auch $\Pi'(p,t',c) \leq \Pi(p,t,c)$ erfüllt ist. Da jetzt eben nicht mehr nur die Zeit eine Rolle spielt, steigt auch die Komplexität von $\mathcal{O}(|J|^2)$ auf $\mathcal{O}(|J|^2 \cdot \delta_j)$ an und führt damit zu einem pseudopolynomiellen Algorithmus. Die maximale Anzahl an gleichzeitig vorhandenen Zuständen ist beschränkt durch $\mathcal{O}(|J| \cdot \delta_j)$. Die minimalen Zeiten pro Position können verschärft werden, wenn im Ansatz der Dynamischen Programmierung gleichzeitig auch ein minimaler Materialverbrauch pro Materialgruppe und Position p berechnet wird (siehe auch 4.3.4), und dann die früheste Zeit der Materialverfügbarkeit t_p^{inv} für diese Position ermittelt wird. Die Ketten lassen sich vor Anwendung des Ansatzes der Dynamischen Programmierung verkürzen, sobald der zusätzliche Materialverbrauch durch einen weiteren Auftrag $i \in C'$ aus der fremden Kette C' die Zeit der frühesten Materialverfügbarkeit soweit verzögert, dass die Fälligkeit δ_j von $j \in C$ überschritten wird und das Einbinden dieses Auftrages (und aller weiteren in

der Kette) damit keine zeitlich zulässige Menge von Vorgängern mehr darstellen würde.

Abschätzung des dualen Preises für direkte Vorgänger Für die Berücksichtigung der dualen Preise $-x^D_{i,j}(\mu)$ für die Wahl eines direkten Vorgängers $i \in J$ für einen Auftrag $j \in C$ aus der Kette konstruieren wir eine positionsabhängige Abschätzung $x^{LB}_{j,p}$, den minimalen dualen Preis für einen direkten Vorgänger für Auftrag $j \in C$, falls Auftrag j eine Menge von Vorgängern mit Kardinalität p erreichen soll, also an p-ter Stelle im Schedule platziert werden soll. Für $p = |P^{\min}_j(\nu)|$ ist das gegeben durch alle Aufträge, die in $P^{\min}_j(\nu)$ einen zulässigen direkten Vorgänger darstellen. Für $p = |P^{\min}_j(\nu)| + 1, \ldots, |P^{\max}_j(\nu)|$ prüfen wir alle Aufträge $i \in P^{\max}_j(\nu)$, deren maximale Position in einem zulässigen Schedule $p^{\max}(i)$ eine Wahl als direkten Vorgänger bei der Platzierung von j an Position p erlaubt, es muss also die Bedingung $p^{\max}(i) + 1 \geq p$ erfüllt sein.

$$x^{LB}_{j,p} := \min\{-x^D_{i,j}(\mu) : i \in J, (i,j) \in E^{\rightarrow}, i \in P^{\max}_j, p^{\max}(i) + 1 \geq p\} \quad (7.57)$$

Sollte es keinen Auftrag $i \in J$ geben, der alle Eigenschaften für eine Position p erfüllt, so kann es auch kein zulässiges Inklusionsmuster mit einer Vorgängermenge der Größe p für Auftrag j mehr geben, die Wahl der Position p für j ist damit unzulässig.

Abschätzung der reduzierten Kosten der Vorgängermenge Die untere Schranke für die mindestens anfallenden reduzierten Kosten für die Platzierung von Auftrag $j \in C$ auf Position p lässt sich wie folgt ermitteln:

$$\bar{z}^{LB}_{j,p} = \min_{t \geq t^{inv}_p + p_j} \{w_j \cdot t + \Pi(p, t, |C|)\} - \psi^D_{j,p}(\mu) + x^{LB}_{j,p}. \quad (7.58)$$

Abschätzung für die gesamte Kette Nachdem wir für jeden noch nicht versiegelten Auftrag die minimalen reduzierten Kosten $\bar{z}^{LB}_{j,p}$ für jede noch mögliche Position $p = |P^{\min}_j(\nu)|, \ldots, |P^{\max}_j(\nu)|$ bestimmt haben, müssen wir aus diesen Werten die minimalen reduzierten Kosten eines vollständigen Inklusionsmusters abschätzen. In jedem Inklusionsmuster für eine Kette $C = (j_1, \ldots, j_{|C|})$ gilt laut Definition $P_{j_1} \subset \ldots \subset P_{j_{|C|}}$. Diese Bedingung relaxieren wir zu $|P_{j_1}| < \ldots < |P_{j_{|C|}}|$, wir suchen also für die Aufträge aus unserer Kette eine Reihe von streng monoton ansteigenden Positionen $p_{j_1} < \ldots < p_{j_{|C|}}$, die zu einer Minimierung der Gesamtkosten

$$\sum_{k=1}^{|C|} \bar{z}^{LB}_{j_k, p_k} \quad (7.59)$$

führen. Dieses Problem ist in der Literatur als Maximum-Weight-Non-Crossing-Matching bekannt (Khoo und Cong 1992), wobei in dem zitierten Artikel eine etwas veränderte Problemstellung betrachtet wird. Zur Lösung des Problems lässt sich ein Ansatz der Dynamischen Programmierung mit einer asymptotischen Laufzeit von $\mathcal{O}(|C| \cdot |J|)$ definieren. Ein Zustand $\Psi(p, k)$ repräsentiert dabei die minimalen Kosten der Zuordnung der k ersten Aufträge aus der Kette

C auf Positionen im Intervall $[1, p]$. Die initialen Zustände sind dann gegeben durch

$$\Psi(p, 0) = 0 \quad \forall p = 1, \ldots, |J| \tag{7.60}$$

und

$$\Psi(0, k) = \infty \quad \forall k = 1, \ldots, |C|. \tag{7.61}$$

Die Rekurrenzgleichung des Ansatzes für $k > 0, p > 0$ lautet dann

$$\Psi(p, k) = \min\{\Psi(p-1, k), \Psi(p-1, k-1) + \bar{z}^{LB}_{j_k,p}\}, \tag{7.62}$$

Die letztendlich erzeugte untere Schranke $LB(\nu)$ für den Knoten ν ist dann gegeben durch den Zustand $\Psi(|J|, |C|)$ und den dualen Preis für die Bereitstellung eines Inklusionsmusters für Kette C:

$$LB(\nu) := -\gamma_C + \Psi(|J|, |C|). \tag{7.63}$$

Sollten Vorgängermengen bereits versiegelt sein, werden die dort ermittelten reduzierten Kosten für die Vorgängermenge der versiegelten Aufträge zugrunde gelegt.

7.5.2.5 Konstruktionsheuristik zur Spaltengenerierung

Bei Vorliegen von Mengen von Vorgängern unterschiedlicher Größe p für jeden Auftrag $j \in C$ kann der im Verfahren für die Bestimmung unterer Schranken eingesetzte Ansatz der Dynamischen Programmierung so in eine Heuristik umgewandelt werden, dass auch neue zulässige Spalten konstruiert werden. Sei dazu $P_{j,p}$ eine Menge von Vorgängern von $j \in C$ mit $|P_{j,p}| = p$ und $\hat{z}^{LB}(P_{j,p})$ sei eine gültige Unterschätzung der reduzierten Kosten von $P_{j,p}$ für den Einsatz von $P_{j,p}$ in einem Inklusionsmuster. Die Heuristik geht in drei Schritten vor:

Phase I Zunächst prüfen wir, ob sich aus den Mengen von Vorgängern überhaupt Inklusionsmuster erstellen lassen und berechnen eine untere Schranke $\Psi^{LB}_{j,p}$ für die reduzierten Kosten, die auf jeden Fall anfallen, wenn um die Kandidatenmenge für Auftrag j auf Position p erweitert wird. Die Schranke und die zulässige Erweiterbarkeit wird in Rückwärtsrichtung, beginnend bei dem letzten Auftrag der Kette ermittelt. Existiert für einen Auftrag j an Position p keine Kandidatenmenge, so gilt $\Psi^{LB}_{j,p} = \infty$. Für den letzten Auftrag der Kette $j_{|C|}$ gilt bei vorliegen einer Kandidatenmenge für diese Schranke $\Psi^{LB}_{j_{|C|},p} = \hat{z}^{LB}(P_{j_{|C|},p})$. Für den Fall, dass es für einen Auftrag j_k mit $k < |C|$ und eine Menge p keine zulässige Erweiterung in der Form $P_{j_k,p} \subset P_{j_{k+1},p'}$ mit $p' > p$ geben sollte, wird $\Psi^{LB}_{j_k,p} = \infty$ gesetzt und die Menge entfernt. In allen weiteren Fällen ergibt sich die untere Schranke als

$$\Psi^{LB}_{j_k,p} = \hat{z}^{LB}(P_{j_k,p}) + \min\{\Psi^{LB}_{j_{k+1},p'} : p' > p, P_{j_k,p} \subset P_{j_{k+1},p'}\}. \tag{7.64}$$

Phase II Ein Zustand $\Psi(j_k, p, t)$ bezeichne die minimalen Kosten für ein partielles Inklusionsmuster ι' mit Mengen $P_{j_1}(\iota') \subset \ldots \subset P_{j_k}(\iota') = P_{j_k,p}$ und einer unteren Schranke $C^{LB}_{j_k}(\iota')$ für die Fertigstellungszeit von j_k von mindestens t. Dann dominiert ein Zustand $\Psi(j_k, p, t)$ einen anderen Zustand $\Psi(j_k, p, t')$, falls $t \leq t'$ und $\Psi(j_k, p, t) \leq \Psi(j_k, p, t')$ erfüllt ist, wir müssen also nur die

pareto-effizienten Zustände speichern. Die Erweiterung der vorliegenden Zustände $\Psi(j_k, p, t)$ für einen Auftrag $j_k \in C$ mit $k < |C|$ mit gegebener Position p und beliebigen Zeiten $t \leq \delta_{j_k}$ um eine Vorgängermenge $P_{j_{k+1},p'}$ für den Nachfolgeauftrag j_{k+1} mit Position $p' > p$ wird nun auf folgende Weise realisiert:

In Phase I wurde bereits geprüft, ob $P_{j_k,p} \subset P_{j_{k+1},p'}$ erfüllt ist. Ist diese Voraussetzung gegeben, prüfen wir, ob die Erweiterung noch zu einem Inklusionsmuster mit negativen reduzierten Kosten von Interesse $\hat{z}^{UB}$ führen kann, indem wir die in Phase I berechnete Schranke zu Hilfe nehmen. Dazu suchen wir nun den zeitlich ersten Zustand, für dessen Zeit t' die folgende Ungleichung erfüllt ist:

$$\Psi(j_k, p, t') + \Psi^{LB}_{j_{k+1},p'} < \hat{z}^{UB} \tag{7.65}$$

Gibt es keinen solchen Zustand, hat uns die untere Schranke $\Psi^{LB}_{j_{k+1},p'}$ vor zusätzlichem Rechenaufwand bewahrt. Ist jedoch ein solcher Zustand vorhanden, erweitern wir diesen und alle weiteren Zustände $\Psi(j_k, p, t'')$ mit $t'' > t'$ (und damit auch $\Psi(j_k, p, t'') < \Psi(j_k, p, t')$) um die Menge $P_{j_{k+1},p'}$. Das Erweitern eines Zustandes ist mit einem Versiegelungsbranching gleichzusetzen, aus einem Zustand können somit für verschiedene Wahlen von direkten Vorgängern für den Auftrag j_{k+1} mehrere Zustände werden. In der Implementierung wird die Erweiterung zunächst nur für den Zustand $\Psi(j_k, p, t')$ durchgeführt, es findet hier unter anderem die Bestimmung der möglichen direkten Vorgänger und des minimalen Rüstaufwands sowie die Materialüberprüfung statt. Dabei werden die Wartezeiten auf die Materialverfügbarkeit für jeden direkten Vorgänger und für den Auftrag j_{k+1} selbst gespeichert. Die weiteren Zustände $\Psi(j_k, p, t'')$ mit $t'' > t'$ können dann in konstanter Zeit erweitert werden, indem diese Wartezeiten mit der Differenz $t'' - t' > 0$ abgeglichen werden, um die unteren Schranken für die Fertigstellungszeit von j_{k+1} und des jeweiligen direkten Vorgängers zu bestimmen und damit auch Zeit und Kosten der neuen Zustände zu berechnen.

Phase III Die zulässigen Zustände $\Psi(j_{|C|}, p, t) \forall p = |C|, \ldots, |J|$ für den letzten Auftrag der Kette C repräsentieren dann zulässige Inklusionsmuster und diese können durch ein Backtracking der Zustände rekonstruiert werden, sofern die Kosten des Zustands und damit die reduzierten Kosten des damit verbundenen Inklusionsmusters die Qualitätsanforderung $\hat{z}^{UB}$ erfüllen.

Die Gesamtanzahl der rechnerisch aufwändigen Erweiterungen ist asymptotisch beschränkt durch $\mathcal{O}(|C| \cdot |J|^2)$ und die maximale Anzahl an Zuständen ist beschränkt durch $\mathcal{O}(|C| \cdot |J| \cdot \delta_{\max}(C))$ in der vom Vorgängerauftrag abhängigen Zeitabschätzung aus Abschnitt 7.3.2.2. Hierbei bezeichne $\delta_{\max}(C)$ die maximale Fälligkeit eines Auftrags $j \in C$. Wird die vom Vorgänger in der Kette unabhängige Zeitabschätzung gewählt, wird die Heuristik einfacher, da dann in einer Erweiterung nur auf die Teilmengenbeziehung und die zulässige Wahl eines direkten Vorgängers geachtet werden muss. Die Heuristik lässt sich dann in eine vorwärtsgerichtete Phase zur Ermittlung der Zustände und eine Backtrackingphase gliedern.

7.5.3 Branch-and-Bound-Pricer Set-Jump

Das erste Enumerationsschema zur (potentiellen) Erzeugung aller möglichen Inklusionsmuster für eine Kette C beruht auf der Beobachtung, dass es für jeden fremden Auftrag $i \notin C$ in einem Inklusionsmuster entweder einen Auftrag

$j \in C$ gibt, in dessen Menge von Vorgängern P_j der Auftrag i zum erstenmal enthalten ist, oder es tritt der Fall auf, dass der Auftrag i in keiner Menge von Vorgängern P_j für $j \in C$ enthalten ist. Deshalb wird im Set-Jump-Pricer für jeden Auftrag $i \notin C$ dieser erste Auftrag $j \in C$ ausgewählt, oder entschieden, dass der Auftrag gar nicht Teil des Inklusionsmusters sein soll. Der Name des Enumerationsschemas wird aus der Beobachtung abgeleitet, dass der Auftrag $i \notin C$, sollte er Gegenstand des Branchings sein, entlang der Kette C von Auftrag zu Auftrag weiterspringt bis er letztendlich im Extremfall das Inklusionsmuster ganz verlässt.

7.5.3.1 Verzweigungsentscheidungen

Sobald die Menge von Vorgängern für den ersten nicht versiegelten Auftrag der Kette fixiert ist, wird eine Versiegelungsentscheidung getroffen. Wenn dies nicht der Fall ist, also keine Versiegelung möglich ist, sei $j \in C$ der erste Auftrag der Kette mit nicht versiegelter und damit dann auch nicht fixierter Menge von Vorgängern. Dann gilt $P_j^{\min} \subset P_j^{\max}$, und damit $P_j^{\max} \backslash P_j^{\min} \neq \emptyset$. Es gibt also mindestens einen Auftrag $i \in P_j^{\max} \backslash P_j^{\min}$, von dem nicht klar ist, ob er denn zur Menge von Vorgängern von j gehören soll, und damit P_j die erste Menge von Vorgängern wäre, in der dieser Auftrag i aufgenommen wird. Der Auftrag i^*, der die Mengen von Vorgängern von j und seiner Nachfolger in der Kette durchlaufen soll, wird bestimmt über

$$i^* := \underset{i \in P_j^{\max} \backslash P_j^{\min}}{\arg\max} |\{i' \in J : i \prec i'\}|. \tag{7.66}$$

Es wird also der Auftrag mit den meisten Nachfolgern gesucht. Dies stellt einerseits sicher, dass bereits alle Vorgänger von i^* in $P_j^{\min}$ enthalten sind. Für die Struktur des Enumerationsbaumes ergeben sich jedoch weitere Vorteile. Einerseits werden bei Nichtaufnahme von i^* in die Menge $P_j^{\min}$ sowohl der Auftrag i^* als auch alle seine Nachfolger aus der Menge $P_j^{\max}$ entfernt, dies gilt in den weiteren Zweigen dann auch für alle Vorgänger in der Kette, die maximal möglichen Mengen werden damit sehr stark reduziert. Andererseits ist bei der Wahl eines Auftrags mit vielen Nachfolgern die Wahrscheinlichkeit höher, dass auch ein Auftrag $j' \in C$ mit $j \prec j'$ ein Nachfolger von i^* ist, und damit der Auftrag i^* nicht durch die ganze Kette springen muss. Dies reduziert dann die Anzahl der neu erzeugten Zweige.

7.5.3.2 Dominanzkriterien

In diesem Branch-and-Bound-Ansatz wird nur in den Versiegelungsknoten das Dominanzkriterium 7.22 lokal angewendet. Eine globale Anwendung des Kriteriums auf den gesamten Baum ist schwierig, da das Vorliegen der Bedingungen 4 und 5 hier eine tatsächliche Überprüfung aller Mengen $P_j^{\min}(\nu)$ und $P_j^{\max}(\nu)$ für nicht versiegelte $j \in C$ voraussetzt.

7.5.4 Branch-and-Bound-Pricer Set-Fill

Das zweite Enumerationsschema zur impliziten Erzeugung aller Inklusionsmuster fixiert für die Aufträge der Kette $C = (j_1, \ldots, j_k, \ldots, j_{|C|})$ der Reihe nach alle Mengen von Vorgängern $P_{j_1}, \ldots, P_{j_{|C|}}$.

7.5.4.1 Verzweigungsentscheidungen

Auch hier wird zunächst bevorzugt eine Versiegelungsentscheidung getroffen. Sollte die Menge von Vorgängern des ersten nicht versiegelten Auftrags $j \in C$ noch nicht fixiert sein, gibt es dann immer eine fremde Kette $C' \in \mathcal{C}\backslash\{C\}$, für die noch unklar ist, wie viele Aufträge aus C' denn nun in P_j enthalten sein sollen. Aufgrund der Vorrangbeziehungen innerhalb der fremden Kette C' dürfen wir hier immer von der Anzahl der Aufträge aus C' sprechen, da sofort klar ist, welche Aufträge aus C' gemeint sind. Der Verzweigungsschritt hat dann die Aufgabe, alle noch offenen Möglichkeiten für die Schnittmenge $P_j \cap C'$ zu evaluieren. Die fremde Kette $C^* \in \mathcal{C}\backslash\{C\}$, über die verzweigt werden soll, wird dabei bestimmt durch die Funktion

$$\Delta^{Fill}(j, C') := |(P_j^{\max}\backslash P_j^{\min}) \cap C'| \tag{7.67}$$

Für nicht fixierte Mengen von Vorgängern gibt es mindestens eine Kette C', für diese Funktion einen Wert > 0 annimmt. Die Kette C^* ist dann gegeben durch

$$C^* := \underset{j=j^{noseal}, C' \in \mathcal{C}\backslash\{C\}}{\arg\max} \Delta^{Fill}(j, C'). \tag{7.68}$$

7.5.4.2 Dominanzkriterien

Das Dominanzkriterium 7.22 wird in jedem Versiegelungsschritt angewendet. Dabei wird neben der lokalen Dominanz auch eine Dominanztabelle zur Hilfe genommen, in der für $k > 1$ nach Knoten gesucht wird, die bereits eine Versiegelungsentscheidung für dieselbe Menge von Vorgängern repräsentieren. Dann können wir immer davon ausgehen, dass die Bedingungen 4 und 5 des Dominanzkriteriums erfüllt sind, da alle Branchingentscheidungen bezüglich der minimalen und maximalen Mengen von Vorgängern nur Aufträge $j_{k'}$ in der Kette mit $k' \leq k$ direkt betroffen haben. Durch die Fixierung von P_{j_k} wurden zwar auch die nachfolgenden minimalen Mengen von Vorgängern manipuliert (es muss gelten $P_{j_k}(\nu) \subset P_{j'_k}^{\min}(\nu)$ für alle $k' > k$), aber dies ist auch für den Knoten ν' in der Dominanztabelle der Fall. Die Dominanztablle ist hier so organisiert, dass in einem zweidimensionalen Array zunächst eine Vorfilterung stattfindet bezüglich des versiegelten Knotens und der Größe der Menge von Vorgängern. Für jeden Matrixeintrag gibt es dann einen AVL-Baum mit den Mengen von Vorgängern, die hinsichtlich dieser beiden Kriterien identisch sind. Für jede Menge von Vorgängern wird dann wie im Dominanzkriterium für den herkömmlichen Branch-and-Bound-Ansatz eine einfach verkettete Liste mit monoton ansteigenden Versiegelungszeiten und monoton absteigenden Versiegelungskosten für diese Menge von Vorgängern angelegt, um die Kriterien 2 und 3 des Dominanzkriteriums 7.22 zu überprüfen.

7.5.5 Branch-and-Bound-Pricer Set-Synchronization

In diesem Branch-and-Bound-Ansatz ist die Berechnung unterer Schranken für die minimalen reduzierten Kosten eines Inklusionsmusters etwas aufwändiger gestaltet, lässt dafür aber auch eine Ableitung einer Verzweigungsentscheidung aufgrund der unteren Schranke zu.

7.5.5.1 Bevorzugte Mengen von Vorgängern

Hierzu berechnen wir in jedem Knoten ν des Baumes für jeden noch nicht versiegelten Auftrag $j \in C$ die im Knoten ν bevorzugte Menge von Vorgängern $P_j^*(\nu)$ mit direktem Vorgänger $x_j^*(\nu)$, so dass die Kostenabschätzung minimal ist. Dabei wird die Kostenabschätzung auf Basis von $t_{j'}^{seal}(\nu)$ des letzten versiegelten $j' \in C$ Knotens berechnet, und auch die aktuellen minimalen und maximalen Mengen von Vorgängern $P_j^{\min}(\nu)$ und $P_j^{\max}[\nu)$ für alle weiteren Aufräge $j \in C$ werden zur Kostenabschätzung herangezogen.

7.5.5.2 Verzweigungsentscheidungen

Auch hier wird sobald als möglich ein Versiegelungsbranching durchgeführt. Sollte dies nicht möglich sein, versuchen wir eine Verzweigungs aufgrund der Ergebnisse der Schrankenberechnung für Knoten ν abzuleiten. Hierfür betrachten wir für die Kette $C = (j_1, \ldots, j_k, \ldots, j_{k'}, \ldots j_{|C|})$ die für jeden Auftrag ermittelten bevorzugten Mengen von Vorgängern $P_{j_k}^*(\nu)$ für $k = 1, \ldots, |C|$. Sobald $P_{j_k}^* \nsubseteq P_{j_{k'}}^*$ für ein Paar von Aufträgen (j_k, j_k') aus der Kette mit $k' > k$ gilt, wissen wir, dass diese Mengen von Vorgängern nicht zusammen in einem Inklusionsmuster vorkommen können, da zur Bildung eines Inklusionsmusters die Mengen ja perfekt verschachtelt sein müssen. In diesem Fall versuchen wir, einen Auftrag $j_{k^*} \in C$ aus der Kette und eine fremde Kette $C^* \neq C$ zu finden, so dass ein im Anschluss definiertes Maß der Verletzung der Teilmengenrelation maximiert wird. Wie im Bucket Fill Pricer (7.5.4) fixieren wir dann die Anzahl der Aufträge aus Kette C^* die in jeder Menge von Vorgängern $P_{j_{k^*}}(\nu')$ eines Kindknotens ν' von ν enthalten sein müssen. Die Verletzung der Teilmengenbeziehungen für eine fremde Kette C' und einen Auftrag $j_k \in C$ ist dabei gegeben durch

$$\begin{aligned}\Delta^{Sync}(j_k, C') = &\sum_{k'=1}^{k-1} |\{P_{j_{k'}}^*(\nu) \cap C'\} \backslash \{P_{j_k}^*(\nu) \cap C'\}| \\ &+ \sum_{k''=k+1}^{|C|} |\{P_{j_k}^*(\nu) \cap C'\} \backslash \{P_{j_{k''}}^*(\nu) \cap C'\}|.\end{aligned} \tag{7.69}$$

Der erste Term der Verletzungsfunktion untersucht für jeden Vorgänger $j_{k'}$ von j_k in der Kette die Anzahl an Aufträgen aus Kette C', die zwar in $P_{j_{k'}}^*(\nu)$, aber nicht in $P_{j_k}^*(\nu)$ enthalten sind. Der zweite Term ist dagegen den Nachfolgern $j_{k''}$ gewidmet, hier prüfen wir, inwieweit in $P_{j_k}^*(\nu)$ Aufträge aus C' enthalten sind, aber nicht in der Menge von Nachfolgern $P_{j_{k''}}^*(\nu)$. Das bei Verzweigung des Knotens ν zu wählende Paar (j_{k^*}, C^*) aus eigenem Auftrag und fremder Kette ist dann gegeben durch

$$(j_{k^*}, C^*) = \underset{j_k \in C, C' \neq C, C' \in \mathcal{C}}{\arg\max} \Delta^{Sync}(j_k, C'). \tag{7.70}$$

Sollten dagegen die gewünschten Mengen von Vorgängern perfekt verschachtelt sein, das Maximum von $\Delta^{Sync}(j_k, C')$ also den Wert 0 besitzen, aber dennoch eine Lücke zwischen aktueller unterer und oberer Schranke bestehen, wird wie im Bucket-Fill-Pricer verfahren. Die Mengen von Vorgängern werden der Reihe nach fixiert, um weitere Versiegelungen zu ermöglichen. Damit wird die

Abschätzung von Fertigstellungszeiten und damit der reduzierten Kosten in den gewünschten Mengen von Vorgängern $P_j^*(\nu)$ für $j \in C$ genauer, und die Optimalitätslücke weiter geschlossen.

7.5.5.3 Berechnung von Mengen von Vorgängern

Die Mengen $P_j^*(\nu)$ selbst werden wieder über ein Branch-and-Bound-Verfahren bestimmt. Wird ein Knoten verzweigt, wird auch hier darüber entschieden, wieviele Aufträge einer fremden Kette $C' \in \mathcal{C} \backslash \{C\}$ in die Menge von Vorgängern von j aufgenommen werden sollen. Die Auswertung von möglichen bevorzugten Mengen von Vorgängern wird inkrementell über die Geschwisterknoten durchgeführt. Um die Berechnung der unteren Schranken in diesem Verfahren zu beschleunigen, wird der Ansatz der Dynamischen Programmierung zur Berechnung ohne Berücksichtigung von C' durchgeführt, und die Zustände dann für alle Geschwisterknoten evaluiert.

Sollten wir nicht in der Lage sein, dass Problem vollständig bis zum Optimum zu lösen (zum Beispiel bei Überschreitung einer festgelegten Anzahl von Knoten), so liefert das Verfahren uns dennoch eine untere Schranke $\bar{z}_j^{LB(B\&B)}$ für die minimalen reduzierten Kosten für Auftrag $j \in C$.

7.5.5.4 Wiederverwendung von Mengen von Vorgängern

Um die aufwändige Berechnung der bestmöglichen Menge von Vorgängern $P_j^*(\nu)$ für einen Auftrag $j \in C$ mittels des Branch-and-Bound-Verfahrens aus Abschnitt 7.5.5.3 eventuell gänzlich zu vermeiden, wird in jedem Knoten ν für jeden nicht versiegelten Auftrag die ermittelte bevorzugte Menge $P_j^*(\nu)$ sowie eine untere $\bar{z}_j^{LB(B\&B)}(\nu)$ und eine obere Schranke $\bar{z}_j^{UB(B\&B)}(\nu)$ für die reduzierten Kosten der bevorzugten Menge gespeichert. Wenn die alte Menge $P_j^*(\nu')$ für den Vaterknoten ν' nicht mehr zulässig ist, es gilt also $P_j^*(\nu') \nsubseteq P_j^{\max}(\nu)$ oder $P_j^{\min}(\nu) \nsubseteq P_j^*(\nu')$, beziehungsweise es gibt für die alte Menge $P_j^*(\nu)$ keine Möglichkeit mehr, eine zeitlich und auch sonst zulässige Wahl eines direkten Vorgängers zu erreichen, wird eine neue beste Menge $P_j^*(\nu)$ für den aktuellen Kindknoten ν gesucht. Ansonsten kann die alte Menge $P_j^*(\nu')$ auch als bevorzugte Menge für den Kindknoten ν' übernommen werden, es gilt dann $P_j^*(\nu') = P_j^*(\nu)$. Hierbei ist allerdings zu beachten, dass durch die im Knoten ν' getroffene Verzweigungsentscheidung die Abschätzung der reduzierten Kosten der Menge $P_j^*(\nu)$ einen höheren Wert ergeben kann, als das im Vaterknoten der Fall war. Somit muss die Menge $P_j^*(\nu)$ nicht die optimale Lösung im Kindknoten ν' sein. Um die Korrektheit des Verfahrens weiterhin zu gewährleisten, wird deshalb die Abschätzung der minimalen reduzierten Kosten einer bevorzugten Menge des Vaterknotens ν' auch für den Kindknoten ν übernommen, es gilt also $\bar{z}_j^{LB(B\&B)}(\nu) = \bar{z}_j^{LB(B\&B)}(\nu')$.

7.5.5.5 Zweistufige Schrankenberechnung

Die Speicherung der bevorzugten Mengen sowie ihrer voraussichtlichen reduzierten Kosten macht auch einen zweistufigen Prozess zur Berechnung unterer Schranken in einem Knoten ν des Pricingverfahrens möglich. Zunächst berechnen wir die auf der Dynamischen Programmierung basierende Schranke zur Abschätzung der minimalen reduzierten Kosten $\bar{z}_{j,p}^{LB}$ für jeden Auftrag j der Kette

C. Alle zu günstigen Schranken $\bar{z}_{j,p}^{LB}(\nu) < \bar{z}_j^{LB(B\&B)}(\nu')$ für einen Auftrag $j \in C$ werden dann durch die Kostenabschätzung $\bar{z}_j^{LB(B\&B)}(\nu')$ aus der Berechnung für den Vaterknoten ersetzt. Mit diesen Werten sind wir in der Lage eine erste untere Schranke für die gesamte Kette und damit ein vollständiges Inklusionsmuster abzuleiten (siehe Abschnitt 7.5.2.4). Führt diese untere Schranke nicht zu einem Abschneiden des Knotens ν, berechnen wir für jeden Auftrag $j \in C$ mit dem Branch-and-Bound-Verfahren aus Abschnitt 7.5.5.3 eine neue Schranke $\bar{z}_j^{LB(B\&B)}(\nu)$, sofern wir nicht auf eine erneute Berechnung verzichten wollen. Auch hier ersetzen wir zu günstige Schranken $\bar{z}_{j,p}^{LB}(\nu)$ durch den neu berechneten Wert $\bar{z}_j^{LB(B\&B)}(\nu)$.

7.5.5.6 Dominanzkriterien

Auch in diesem Verfahren wird in den Versiegelungsentscheidungen das Dominanzkriterium 7.22 angewendet. Die lokale Verwendung für die in der Versiegelungsentscheidung entstehenden Kindknoten ist uneingeschränkt möglich. Um auch andere Bereiche des Baumes für die Anwendung des Dominanzkriteriums zugänglich zu machen, wird auf die Dominanztabelle aus Abschnitt 7.5.4.2 zurückgegriffen. Allerdings müssen wir hier etwas vorsichtiger agieren, was die Eintragung eines Knotens in die Dominanztabelle betrifft, um die Einhaltung der Kriterien 4 und 5 von 7.22 zu garantieren. Wird ein Knoten versiegelt ν, prüfen wir in jedem Fall, ob es einen dominierenden Knoten ν' in der Dominanztabelle gibt. Ein aktuell versiegelter Knoten darf allerdings nur dann in die Dominanztabelle eingetragen werden, und kann damit dann auch Knoten aus der Dominanztabelle oder später erzeugte Knoten dominieren, wenn hinter dem zuletzt versiegelten Auftrag j_k aus der Kette keine direkte Manipulation von minimalen und maximalen Mengen von Vorgängern für Aufträge $j_{k'}$ mit $k' > k$ durch eine Reparaturverzweigung stattgefunden hat. Somit sind die Kriterien 4 und 5 von 7.22 auf jeden Fall erfüllt, die Einschränkung führt aber dazu, dass das Dominanzkriterium nur sehr selten greifen kann.

7.5.5.7 Verwendung der einfachen Zeitabschätzung

Dieses Branch-and-Bound-Verfahren zur Erzeugung von Inklusionsmustern kann auch auf die vom Vorgänger in der Kette unabhängige Zeitabschätzung (siehe Abschnitt 7.3.2.1) angepasst werden. Hierfür sind dann keine Versiegelungsentscheidungen notwendig, die Heuristik zur Erzeugung von oberen Schranken lässt sich deutlich vereinfachen. Die reduzierten Kosten einer bevorzugten Menge von Vorgängern bleiben dann für den gesamten Verlauf des Branch-and-Bound-Verfahrens konstant.

7.6 Verzweigungregeln im Masterproblem

Die Variablen $\lambda_t \in \{0,1\}, t \in \mathcal{T}$ des Masterproblems MP kennzeichnen die Auswahl eines Inklusionsmusters. In der LP-Relaxation des Masterproblems LPM und dessen auf eine Auswahl an Variablen beschränkter Variante RLPM können einige dieser Variablen in einer optimalen Lösung der Probleme auch fraktionelle Werte annehmen, es gibt dann also einen Index $t' \in \mathcal{T}$, für dessen Variable

$\lambda_{t'}$ gilt: $0 < \lambda_{t'} < 1$. Um nun eine ganzzahlige Lösung des Problems zu erzwingen, würde in einer traditionellen Implementierung eines LP-basierten Branch-and-Bound-Verfahrens zur ganzzahligen Programmierung über eine dieser Variablen verzweigt werden. Erzwingen wir in einem Zweig die Aufrundung von $\lambda_{t'}$ auf $\lceil \lambda_{t'} \rceil = 1$, so muss in allen Knoten dieses Teilbaumes das mit der Variablen assoziierte Inklusionsmuster ausgewählt werden, damit verschwindet auch das Unterproblem für die Kette dieses Inklusionsmusters. Alle anderen Unterprobleme können dann dahingehend vereinfacht werden, dass alle Entscheidungen, die mit diesem Inklusionsmuster verbunden sind, auch deren Lösungsraum beschränken. Andererseits bringt die Abrundung von $\lambda_{t'}$ auf $\lfloor \lambda_{t'} \rfloor = 0$ zum Ausdruck, dass wir in allen Lösungen des von dieser Entscheidung abhängigen Teilbaumes die Auswahl dieses Inklusionsmusters verbieten. Hier wird nur ein bestimmtes Inklusionsmuster als Lösung eines Unterproblems verboten. Da diese Lösung hinsichtlich ihrer reduzierten Kosten schon einmal optimal war, ist die Wahrscheinlichkeit hoch, dass dieses Inklusionsmuster noch einmal als profitable Spalte vorgeschlagen wird. In unseren Pricingalgorithmen müssten wir dann in der Lage sein, die beste nicht verbotene Lösung zu finden. Im Rahmen von Branch-and-Bound-Verfahren für die Spaltengenerierung ließe sich eine solche Überprüfung realisieren, würde aber zu Effizienzeinbußen führen. Ein weitaus gravierender Nachteil liegt jedoch darin, dass der im Branch-and-Price-Verfahren entstehende Baum sehr unausgewogen ist. Die wiederholte Entscheidung für ein Inklusionsmuster führt in wenigen Verzweigungsschritten zu einer vollständigen Lösung, wohingegen die Entscheidung gegen ein Inklusionsmuster jeweils nur eine von sehr vielen Alternativen aussschließt. Diese Problematik tritt in jeder Implementierung eines Branch-and-Price-Algorithmus für ein Optimierungsproblem auf (Barnhart u. a. 1998). Wird ein Branch-and-Price-Verfahren auf Basis einer Dantzig-Wolfe-Zerlegung konstruiert, so ist es eine weit verbreitete Methode, die Lösung λ des RLPM wieder zurückzuübersetzen in Werte für die Variablen des Originalproblems. In unserem Fall können wir hierzu zurückgreifen auf die in Abschnitt 7.4.2 definierten Elementarentscheidungen für die Auswahl von Positionen $\psi^P_{j,p}(\lambda)$, Vorrangbeziehungen $\pi^P_{i,j}(\lambda)$ und Kanten $x^P_{i,j}(\lambda)$.

7.6.1 Verzweigung über Kanten

Wir betrachten das gerichtete und gewichtete Netzwerk, dass aus den Werten $x^P_{i,j}(\lambda)$ der Elementarentscheidungen für die Wahl eines direkten Vorgängers i für jeden Auftrag j hervorgeht. Im Spaltengenerierungsansatz von Chen und Powell (1999), der für verschiedene Probleme mit identischen parallelen Maschinen ausgelegt ist, und dessen Spalten Schedules für einzelne Maschinen darstellen, wird immer über die fraktionellste Kante, gegeben durch

$$\arg\min_{(i,j)\in E} |x^P_{i,j}(\lambda) - 1/2|, \tag{7.71}$$

verzweigt, und entweder $x^P_{i,j}(\lambda) = 0$ oder $x^P_{i,j}(\lambda) = 1$ in den beiden Kindknoten erzwungen. Diese Art der Verzweigung ist damit sehr stark an das LP-basierte Branch-and-Bound für (gemischt) ganzzahlige Optimierungsmodelle angelehnt. Da wir hier aber ein Ein-Maschinen-Scheduling-Problem vorliegen haben, und das gewichtete Netzwerk in der Zerlegung eines Schedules in Inklusionsmuster (siehe Definition 7.9) einen Hamiltonpfad durch alle Aufträge darstellen soll,

können wir zur Ableitung einer Verzweigungsentscheidung auch Bezug auf die Verzweigungsentscheidungen in für das TSP entwickelten Branch-and-Bound-Verfahren nehmen. Einen guten Überblick über die Verzweigungsmöglichkeiten bietet hier Balas und Toth (1985).

Speziell betrachten wollen wir die Analyse der bekannten untere Schranke für das symmetrische TSP auf der Basis von 1-Bäumen von Held und Karp (1970), die in Held und Karp (1971) und Volgenant und Jonker (1982) durchgeführt wird, um eine Verzweigungsentscheidung abzuleiten. Volgenant und Jonker (1982) analysieren den einzigen Zyklus im nach dem Subgradientenverfahren entstandenen 1-Baum, und suchen nach einem Knoten mit drei angrenzenden Kanten im 1-Baum. Dieser Knoten und die im 1-Baum vorhandenen Kanten werden dann zu einem ternären beziehungsweise binären Branching herangezogen, wobei jeweils eine der angrenzenden Kanten verboten wird. Da in Branch-and-Bound-Verfahren eine Partitionierung des Lösungsraumes angestrebt wird, kann das Verbieten von Kanten auch mit dem Erzwingen von weiteren Kanten in zwei der drei entstehenden Teilbäume gleichgesetzt werden. In unserem Fall wollen wir das - nun zusätzlich auf den Kanten gewichtete - Netzwerk auf ähnliche Weise wie den 1-Baum im Verfahren von Volgenant und Jonker (1982) untersuchen:

Für einen Auftrag $i \in J$, sei $X_i(\lambda) := \{j \in J | x^P_{i,j}(\lambda) > 0\}$ die Menge aller Aufträge, für die i als direkter Vorgänger (zumindest teilweise) ausgewählt wird. Nun können wir die Aufträge in $X_i(\lambda)$ nach monoton absteigenden Werten $x^P_{i,j}(\lambda)$ sortieren, und $X^k_i(\lambda) = (j_1, \ldots, j_k)$ sei damit dann die sortierte Teilmenge der ersten k Aufträge aus $X_i(\lambda)$ bezüglich dieser Sortierung. Um nun eine Branchingentscheidung abzuleiten, suchen wir einen Auftrag i^* und einen Wert $k^* < k^{\max}$, wobei $k^{\max}$ den konfigurierbaren maximalen Verzweigungsgrad bei Verwendung der Kantenverzweigungsregel darstellt. Diese beiden Werte sind gegeben über

$$(i^*, k^*) := \underset{i \in J, |X_i(\lambda)|>1, 2 \leq k \leq \min\{k^{\max}-1, |X_i(\lambda)|\}}{\arg\min} \frac{x^P_{i,j_1}(\lambda) - x^P_{i,j_k}(\lambda)}{\sum_{j \in X^k_i(\lambda)} x^P_{i,j}(\lambda)}. \tag{7.72}$$

Der Nenner der Funktion in (7.72) kumuliert das Gewicht der ersten k Aufträge aus $X^k_i(\lambda)$ als direkte Nachfolger von i. Fällt dieses sehr groß aus, tendiert die Lösung λ dazu, einerseits für den Auftrag i überhaupt einen direkten Nachfolger auszuwählen, und andererseits soll dieser dann aus der Menge $X^k_i(\lambda)$ stammen. Der Zähler gibt an, inwieweit der erste Auftrag aus $j_1 \in X^k_i(\lambda)$ gegenüber dem Auftrag $j_k \in X^k_i(\lambda)$ als direkter Nachfolger bevorzugt wird. Fällt diese Differenz sehr klein aus, ist die Entscheidung für einen Nachfolger als uneindeutig zu bewerten. Haben wir einen Auftrag i^* und einen Verzweigungsgrad k^* gefunden, so können wir die Entscheidungen der Kindknoten festlegen. In den ersten Kindknoten $\nu_1, \ldots, \nu_{k'}, \nu_{k^*}$ bestimmen wir jeweils den Auftrag i zum direkten Vorgänger von Auftrag $j_{k'} \in X^{k^*}_i(\lambda)$, untersuchen in den Kindknoten damit jeweils alle Schedules, in denen auf den Auftrag i der jeweilige Auftrag $j_{k'}$ direkt folgt. Im Kindknoten ν_{k^*+1} verbieten wir die Auswahl von Knoten i als direkten Vorgänger für alle Aufträge aus $X^{k^*}_i(\lambda)$ gleichzeitig, und wir untersuchen in diesem Kindknoten alle Schedules, in denen kein Auftrag aus $X^{k^*}_i(\lambda)$ auf den Auftrag i folgt.

Die Fixierung einer Kante $(i, j) \in E^{\rightarrow}$ auf den Wert 1 und die damit verbundene Folge eines Auftrags direkt auf einen anderen Auftrag führt zu Implika-

tionen, die auch aus anderen Branch-and-Bound-Verfahren wie dem Verfahren von Little u. a. (1963) bekannt sind. So werden alle anderen Kanten gelöscht, die den Knoten i verlassen oder in den Knoten j eingehen. Außerdem sind alle Vorgänger von i nun auch Vorgänger von j, aber auch alle Vorgänger von j Vorgänger von i. Dasselbe gilt entsprechend für die Nachfolger.

7.6.2 Verzweigung über Vorrangbeziehungen

Anstelle die Auswahl von direkten Vorgängern zu betrachten, können wir auch die Auswahl von Vorgängern betrachten. Im Gegensatz zu den Schnittebenen (7.25), die einen Mangel an Vorrang zwischen je zwei Aufträgen zu beheben versucht haben, prüfen wir hier für jedes Paar $(i,j), i \neq j$ von Aufträgen, ob nicht die linke Seite der Gleichung (7.23) deutlich den Wert 1 der rechten Seite übersteigt. Wenn dies der Fall ist, möchte sowohl i den Auftrag j als auch j den Auftrag i in der jeweiligen Menge von Vorgängern vertreten sehen, aber nur eine von beiden Möglichkeiten kann tatsächlich realisiert werden. Um diesen Konflikt zu lösen, kann in einem Teilbaum die Vorrangbeziehung $i \prec j$, und im anderen Teilbaum die Vorrangbeziehung $j \prec i$ erzwungen werden, was zu einem binären Verzweigungsschritt führt. Um zu bestimmen, zwischen welchen Aufträgen die Vorrangbeziehung fixiert werden soll, sei $\Pi(\varepsilon, \lambda) := \{(i,j) \in J \times J, i > j | \pi^P_{i,j}(\lambda) + \pi^P_{j,i}(\lambda) \geq 1 + \varepsilon\}$ für $\varepsilon > 0$ die Menge aller Paare von Aufträgen, für die die Summe der Vorrangbeziehungen den Wert 1 um mindestens ε übersteigt. Nun sei $\varepsilon^{\max} > 0$ so gewählt, dass $\Pi(\varepsilon^{\max}, \lambda) \neq \emptyset$ und $\nexists \varepsilon > \varepsilon^{\max} : \Pi(\varepsilon, \lambda) \neq \emptyset$. Damit ist $\varepsilon^{\max}$ also der höchste auftretende Wert für eine Übererfüllung der Gleichung (7.23). Sofern $\varepsilon^{\max} \geq \varepsilon^{\min}$ erfüllt ist, wobei $\varepsilon^{\min} > 0$ als Konfigurationsparameter den minimalen Schwellwert zur Ermittlung einer Verzweigung auf der Basis von Vorrangbeziehungen darstellen soll, können wir ein beliebiges Paar (i^*, j^*) von Aufträgen mit $(i^*, j^*) \in \Pi(\varepsilon^{\max}, \lambda)$ für eine Verzweigung auf der Basis von Vorrangbeziehungen heranziehen.

7.6.3 Einfaches Verzweigen

Sollte die aktuelle Lösung des RLPM keine Ableitung der Verzweigungsregeln erlauben, so zum Beispiel dann, wenn die aktuelle Lösung λ einer Zerlegung eines Schedules in Inklusionsmuster entspricht, oder das RLPM nicht zulässig lösbar ist, greifen wir auf die Verzweigungsregel des kombinatorischen Branch-and-Bound-Verfahrens aus Kapitel 6 zurück. In den erzeugten Knoten wird der ab dem Depot bereits fixierte Teilschedule also jeweils um einen der nächstmöglichen Aufträge erweitert, wir erhalten also im schlimmsten Fall ein $|\mathcal{C}(\nu)|$-äres Branching.

7.6.4 Weitere Verzweigungsentscheidungen

Während die in den vorherigen Abschnitten 7.6.1, 7.6.2, 7.6.3 diskutierten Verzweigungsregeln in der Implementierung des Branch-and-Price-and-Cut-Ansatzes auch tatsächlich Verwendung finden, wollen wir hier weitere Überlegungen aus der Literatur zur Verzweigung in Branch-and-Price-Verfahren betrachten.

7.6.4.1 Ryan Foster Branching

Die von Ryan und Foster (1981) entwickelte Regel wird häufig zur Verzweigung in Set-Partitioning Problemen angewendet. Liegt eine fraktionelle Lösung des RLPM vor, so lassen sich in der aufzuteilenden Menge zwei Elemente finden, die beide nur teilweise von einer Spalte abgedeckt werden. Die Verzweigungsregel besagt, dass die Spalten im ersten Zweig entweder keines der beiden Elemente oder beide gleichzeitig enthalten müssen und im zweiten Zweig genau eines oder keines der beiden Elemente. Da bei uns die aufzuteilende beziehungsweise abzudeckende Menge die Menge aller Positionen eines Schedules ist, müssen wir während des Pricingverfahrens überprüfen, ob Positionen in Konflikt miteinander stehen oder nicht. Die Position eines Auftrags ergibt sich immer aus der Größe der Menge der Vorgänger eines Auftrags. Erst wenn wir also eine Auftragsmenge in einem Inklusionsmuster fixiert haben, sind wir in der Lage zu entscheiden, ob auch eine bestimmte weitere Position von dem Inklusionsmuster anzustreben oder zu vermeiden ist.

7.6.4.2 Einschränkung des Ausführungsintervalls

In den Abschnitten 7.6.1 und 7.6.2 haben wir eine Verzweigungsentscheidung auf Basis des Verhältnisses von Aufträgen untereinander in der aktuellen Lösung λ des RLPM abgeleitet. Dieser und der folgende Abschnitt untersuchen Eigenschaften eines einzelnen Auftrags in der aktuellen Lösung λ des RLPM, um zu einer Verzweigungsentscheidung zu gelangen. In Akker, Hoogeveen und Velde (1999) wird in Anlehnung an den in Carlier (1987) beschriebenen Branch-and-Bound-Ansatz das Ausführungsintervall eines Auftrags betrachtet. Generell kann ein Auftrag jederzeit innerhalb seines Zeitfensters gestartet werden, und es wird dann untersucht, inwieweit dieses Zeitfenster tatsächlich ausgeschöpft wird, indem die früheste Fertigstellungszeit eines Auftrags in Abhängigkeit der aktuellen Lösung λ des RLPM bestimmt und mit seiner durchschnittlichen Fertigstellungszeit in der Lösung verglichen wird. In unserer Notation ist die minimale Fertigstellungszeit gegeben durch $C_j^{\min}(\lambda) = \min\{C_j^{LB}(\iota(t))|\lambda_t > 0, I^c(C(j),t) = 1\}$. Der durchschnittliche Fertigstellungszeitpunkt wird berechnet mit Hilfe der Summe $\sum_{t\in\mathcal{T}, I^c(C(j),t)=1} C_j^{LB}(\iota(t))\cdot\lambda_t$. Wenn der durchschnittliche Fertigstellungszeitpunkt vom minimalen Fertigstellungszeitpunkt für einen Auftrag j abweicht, bezeichnen Akker, Hoogeveen und Velde (1999) den Auftrag j als *fraktionellen* Auftrag. Für die Verzweigung wird dann in einem Zweig die Fälligkeit verschärft, in dem $\delta_j = C_j^{\min}(\lambda)$ gesetzt wird, im anderen Fall wird die Freigabezeit verschärft, in dem $r_j = C_j^{\min}(\lambda) + 1 - p_j$ verlangt wird. Diese neuen Zeitfenster für den Auftrag j können dann auch dazu verwendet werden, die Zeitfenster der Vorgänger (für den Fall der neuen Fälligkeit) beziehungsweise Nachfolger (für den Fall der neuen Freigabezeit) weiter zu verschärfen.

7.6.4.3 Verzweigung über Positionen

Alternativ zum vorherigen Abschnitt 7.6.4.2 können wir statt der Fertigstellungszeiten auch die Positionierung eines Auftrags im Schedule betrachten. Wir benötigen dann die durchnittliche Position $\sum_{p=1}^{|J|} p \cdot \psi_{j,p}^P(\lambda)$ und die minimale Position eines Auftrags j in einem aktiven Inklusionsmuster, gegeben durch $\min\{p : \psi_{j,p}^P(\lambda) > 0\}$. Anstelle von Freigabe- und Fälligkeitszeitpunkten werden

hier in Folge minimale und maximale Positionen eines Auftrags sowie seiner Vorgänger und Nachfolger beeinflusst.

7.7 Branch-and-Price-and-Cut

Die Kombination der in den obigen Abschnitten einzeln vorgestellten Komponenten erlaubt es uns, ein exaktes Branch-and-Bound-Verfahren zu erstellen. Da die Schrankenberechnung mittels der Spaltengenerierung erfolgt, und zudem dynamisch Schnitte in jedem Knoten des Branch-and-Bound-Baumes erstellt werden, spricht wird hier von einem **Branch-and-Price-and-Cut**-Verfahren. Die hier implementierte Knotenauswahlstrategie ist die Bestensuche, es wird also immer der Knoten mit der niedrigsten unteren Schranke zuerst verzweigt (siehe hierzu Abschnitt 6.2.2). In jedem Knoten des Branch-and-Price-Baumes sind die folgende Schritte durchzuführen:

7.7.1 Adaptierung der Verzweigungsentscheidungen

Die Entscheidungen in einem Knoten und seinen Ahnen sind entweder die Erzwingung von weiteren Vorrangbeziehungen (siehe Abschnitt 7.6.2) oder die Verwendung oder das Verbot von bestimmten Kanten (siehe die Abschnitte 7.6.1 und 7.6.3). Diese Entscheidungen werden in die ursprüngliche Instanz des Problems integriert und diese neue Instanz als Teilproblem des ursprünglichen Problems einer Vorbehandlung unterzogen (siehe Abschnitt 4). Sollte die Vorbehandlung des Teilproblems die Unzulässigkeit nachweisen, kann bereits hier die Bearbeitung des Knotens abgebrochen werden.

7.7.2 Zerlegung in Unterprobleme

Für den Branch-and-Price-Knoten ν wird dann eine Partitionierung in Ketten $\mathcal{C}(\nu)$ des Vorranggraphen $G(\nu)$ erstellt (siehe Abschnitt 3.4). Dabei gilt $|\mathcal{C}(\nu)| \leq |\mathcal{C}(\nu')|$, wenn ν' der Elternknoten von ν ist, da die Verzweigungsentscheidungen den Vorranggraphen immer weiter verdichten. Damit wird es dann möglich, die Auftragsmenge J in weniger Ketten zu zerlegen.

7.7.3 Erzeugung initialer Spalten

Die Menge aller bekannten Schedules wird in Inklusionsmuster zerlegt. Jedes Inklusionsmuster, das zulässig ist in Hinblick auf die im aktuellen Knoten vorhandenen Vorrangbeziehungen, Kanten und Zeitfenster, wird als Spalte in das Masterproblem eingefügt. Dabei kann auch ein mit den aktuellen Branching-Entscheidungen eigentlich unvereinbarer Schedule dennoch einige zulässige Inklusionsmuster zur Verfügung stellen. Um weitere Spalten für das RLPM zu erzeugen, adaptieren wir das Branch-and-Bound-Verfahren aus Kapitel 6. Als Knotenauswahlverfahren wird hier immer die zyklische Bestensuche CBFS (siehe auch Abschnitt 6.2.3) verwendet. Es kommen die einfachen Zulässigkeitstests und die Dominanztabelle zum Einsatz, als Schranke wird nur die Lösung des Problems $1|chains|\sum_j w_j C_j$ mit nicht modifizierten Prozesszeiten und Gewichten verwendet. Jedesmal, wenn ein partieller Schedule mit dem letzten Auftrag einer Kette C endet, kann daraus ein zulässiges Inklusionsmuster für diese Kette

erzeugt werden, bis eine Obergrenze an die Anzahl erzeugter Spalten erreicht ist. Das Branch-and-Bound-Verfahren wird erst dann vorzeitig beendet, wenn die Größe des Baumes eine vorgegebene Anzahl von Knoten übersteigt, oder bereits in diesem Verfahren eine nachgewiesen optimale Lösung gefunden wurde beziehungsweise die Unzulässigkeit des betrachteten Unterproblems nachgewiesen wurde. Die globale untere Schranke dieses Branch-and-Bound-Verfahrens liefert eine erste initiale untere Schranke für diesen Knoten, und kann dehalb auch zum Abschneiden des Knotens vor der eigentlichen Ausführung des Spaltengenerierungsverfahrens führen.

7.7.4 Spaltengenerierung

Das RLPM wird mit den initialen Spalten gelöst. Damit auf jeden Fall eine Lösung existiert, wird für jede Konvexitätsbedingung 7.9 und jede Set-Cover-Bedingung 7.22 eine künstliche Variable mit hohem Kostenkoeffizienten eingeführt. Diese Übernahme der Idee des Zwei-Phasen-Simplex stellt die Lösbarkeit des RLPM sicher. Die Suche nach neuen Spalten wird dann mit dem ausgewählten Pricing-Verfahren durchgeführt.

7.7.4.1 Untere Schranke

Nach jeder Iteration des Pricing-Verfahrens kann aus den unteren Schranken für die einzelnen Unterprobleme auch eine untere Schranke für das Gesamtproblem abgeleitet werden. Sei dazu z_C^{LB} eine untere Schranke für das für Kette c zu lösende Unterproblem. Dann lässt sich die untere Schranke auf Basis des Spaltengenerierungsverfahrens LB_{CG} ableiten über

$$LB_{CG} = z^*_{RLPM} + \sum_{C \in \mathcal{C}} z_C^{LB}, \tag{7.73}$$

wobei z^*_{RLPM} den optimalen Zielfunktionswert des RLPM darstelle. Die Korrektheit der Schranke ergibt sich aus der Beobachtung, dass jede Kette genau ein Inklusionsmuster in der Lösung des Masterproblems zur Verfügung stellen muss (7.9) (Lasdon 1974). Gemäß der Theorie der linearen Programmierung geben die unteren Schranken für die jeweiligen reduzierten Kosten z_C^{LB} einer Kette $C \in \mathcal{C}$ dann an, um welchen Wert der aktuell optimale Zielfunktionswert z^*_{RLPM} unter Verwendung eines weiteren Inklusionsmusters für Kette C maximal verbessert werden kann. Ist $z_C^{LB} = 0$ für alle $C \in \mathcal{C}$ so ist die LP-Relaxation des Masterproblems (LPM) optimal gelöst. Sobald auch diese untere Schranke in einem Knoten den Wert der besten bekannten zulässigen Lösung übersteigt, können wir auch hier den Knoten aus dem Branch-and-Price-Baum entfernen.

Eine weitere in der Literatur zu findende untere Schranke ist die Schranke von Farley (1990), die in Spaltengenierungsansätzen zur Kostenminimierung mit nicht-negativen Kostenkoeffizienten Anwendung finden kann. Anstelle der minimalen reduzierten Kosten wird hier das relative Verbesserungspotential aller Spalten mit negativen reduzierten Kosten untersucht.

7.7.4.2 Parallele Lösung der Unterprobleme

Da wir die Unterprobleme (Pricing-Probleme) nicht unbedingt bis zur Optimalität lösen müssen, sondern uns auch schon gute untere Schranken zur Konstruktion der unteren Schranke LB_{CG} aus Abschnitt 7.7.4.1 und gute obere Schranken

als Inklusionsmuster mit negativen reduzierten Kosten im Lösungsprozess weiterhelfen, werden die Unterprobleme nicht alle sequentiell vollständig bis zum Erreichen des Optimialitätsnachweises oder eines Abbruchkriteriums (zum Beispiel Anzahl verarbeiteter Knoten), sondern parallel in einem koordinierten Prozess gelöst. Dabei ist die Anzahl gleichzeitig aktiver Threads einerseits durch die Anzahl der Kerne im Prozessor und die Kardinalität der Partitionierung $\mathcal{C}$ der Auftragsmenge J in Ketten beschränkt. Sobald ein Pricing-Problem Rechenkapazität zur Verfügung gestellt bekommt, führt es eine festgelegte Anzahl von Verzweigungschritten durch. Sollte innerhalb dieser Zeit eine bestimmte Anzahl von neuen Spalten mit negativen reduzierten Kosten gefunden worden sein oder das Problem gar schon optimal gelöst worden sein, wird schon früher der Berechnungsfortschritt in Form von aktueller unterer und oberer Schranke und den neuen Spalten an die das Pricing koordinierende Komponente gemeldet. Ist das Pricing-Problem noch nicht optimal gelöst, bittet das Pricing-Verfahren für dieses Unterproblem um weitere Rechenkapazität, es sei denn, die maximale Anzahl von Knoten für das Branch-and-Bound-Verfahren des Unterproblems ist erreicht. Dabei werden diejenigen Unterprobleme bevorzugt, deren Branch-and-Bound-Baum die kleinste Größe hat. Die koordinierende Komponente verwirft den Antrag auf weitere Rechenkapzität, sobald insgesamt eine ausreichende Anzahl an neuen Spalten gefunden wurde, oder im aktuellen Pricing-Schritt keine bessere untere Schranke als bekannt generiert werden kann. Weiterhin wird bei jedem Folgeantrag auf Rechenzeit das Überschreiten des Zeitlimits für das Pricing-Verfahren überprüft. Auch die Anzahl der erlaubten Anträge auf Rechenzeit kann im Ansatz beschränkt werden.

7.7.4.3 Spaltenpool

Die während des Pricings erzeugten neuen Spalten werden dem RLPM hinzugefügt, für die Bestimmung der Koeffizienten wird dabei auf die Formel 7.4.2.5 zurückgegriffen. Sollte die Obergrenze für die Anzahl der Spalten im RLPM überschritten werden, werden die Spalten mit den für die aktuelle optimale Lösung des RLPM höchsten reduzierten Kosten durch die neuen Spalten ersetzt. Um auch im Fall des Einfügens von Schnittebenen eine zulässige Lösung des RLPMs bereithalten zu können, werden niemals Spalten ersetzt, die sich aus einer Zerlegung eines für diesen Knoten zulässigen Schedules ergeben. Sobald die neuen Spalten eingefügt sind, muss das RLPM neu gelöst werden.

7.7.4.4 Erzeugung von Schnittebenen

Um die Ganzzahligkeitslücke zwischen dem LPM und der aktuell besten zulässigen Lösung zu schließen, können wir die in Abschnitt 7.4.4 vorgestellten Schnittebenen einfügen. Da die Bestimmmung einer optimalen Lösung für das LPM sehr viele Iterationen und damit auch viel Zeit in Anspruch nehmen kann, und das Einfügen von Schnittebenen sowieso wieder dazu führt, dass in mehreren Iterationen eine gute Annäherung an die optimalen dualen Variablen der alten und neuen Nebenbedingungen des Masterproblems zu finden ist (Fading-In), fügen wir auch dann Schnittebenen ein, wenn das LPM noch nicht optimal gelöst wurde. Hierzu betrachten wir die Lücke zwischen dem aktuell optimalen Zielfunktionswert für das RLPM und der aktuellen unteren Schranke LB_{CG} (LP-Lücke) und vergleichen diese mit der Lücke zwischen der besten bekannten

oberen Schranke UB und dem Zielfunktionswert des RLPM (Ganzzahligkeitslücke). Falls die Ganzzahligkeitslücke im Verhältnis zur LP-Lücke sehr groß wird, versuchen wir neue Schnittebenen zu erzeugen. Sobald wir in der aktuellen Lösung des LPM eine verletzte Ungleichung gefunden haben, wird diese in das RLPM eingefügt. Dazu wird für jedes im RLPM eingepflegte Inklusionsmuster dessen Koeffizient in der verletzten Ungleichung bestimmt. Auch nach dem Einfügen von Schnittebenen muss das RLPM neu gelöst werden.

7.7.5 Verzweigungsvorschlag

Sobald die Obergrenze für die Anzahl der Iterationen des Spaltengenierungsverfahrens oder ein Zeitlimit erreicht ist, wird die aktuelle optimale Lösung des RLPM analysiert um die Verzweigungsentscheidungen für die Kinder des aktuellen Branch-and-Price-Knotens zu bestimmen. Die Ableitung des Vorschlags selbst wurde bereits in Abschnitt 7.6 abgehandelt. Dieser Vorschlag wird im Knoten gespeichert und der Knoten in die Liste[1] der noch zu verzweigenden Knoten aufgenommen. Sobald der Knoten dann verzweigt wird, wird der gespeicherte Verzweigungsvorschlag dann auch tatsächlich umgesetzt.

7.8 Weitere Aspekte der Spaltengenerierung

In der vorliegenden Arbeit verwenden wir eine recht einfache Form des Spaltengenerierungsverfahrens in Bezug auf das Masterproblem. In den letzten Jahren wurden hier aber auch weitere Techniken entwickelt, um Spaltengenerierungsansätze zu beschleunigen, zwei davon wollen wir nun kurz ansprechen.

7.8.1 Stabilisierung

Der Anstieg der unteren Schranke (siehe Abschnitt 7.7.4.1) ist immer abhängig von der aktuellen dualen Lösung des RLPM. Nun kann man versuchen, die für die Qualität der Schranke richtige duale Lösung zu finden, wenn das RLPM primal entartet ist, da es in diesem Fall dann mehrere optimale duale Lösungen gibt. Diese Problemstellung ähnelt damit sehr stark dem Problem der Wahl der richtigen Multiplikatoren bei der Anwendung der Lagrangerelaxation. In der von Merle u. a. (1999) beschriebenen Stabilisierungstechnik werden deshalb die zu den Nebenbedingungen des RLPM korrespondierenden Variablen durch die Einführung zusätzlicher Variablen in Boxen gesteckt. Verlässt eine duale Variable die Box, werden zusätzliche Strafkosten fällig. Diese Strafkosten werden im RLPM dann durch zusätzliche Nebenbedingungen abgebildet. Einen weiteren Überblick über Stabilisierungstechniken und ihre Vorteile bietet der Artikel von Amor, Desrosiers und Frangioni (2009).

7.8.2 Umgang mit Primaler Entartung

Ist in einem linearen Programm die aktuelle Basislösung primal entartet, so gibt es in dieser Lösung eine oder mehrere Basisvariablen, die auf den Wert Null gesetzt sind. Wenn wir versuchen, einen Basiswechsel mit einer weiteren

[1] Aufgrund der Verwendung der Bestensuche wird in der tatsächlichen Implementierung ein Heap verwendet.

Spalte mit negativen reduzierten Kosten durchzuführen, kann es sein, dass wir aus den negativen reduzierten Kosten keinen Vorteil für unseren Zielfunktionswert ziehen können, da wir durch die Koeffizienten der Matrix $\bar{A}$ auch diese neue Variable auf den Wert null setzen müssen. Im Artikel von Desrosiers, Gauthier und Lübbecke (2014) wird die Idee verfolgt, aus der primalen Entartung einen Vorteil zu ziehen, indem die Suche auf diejenigen Spalten mit negativen reduzierten Kosten eingegrenzt wird, für die sich auch tatsächlich eine Verbesserung des Zielfunktionswertes realisieren lässt. Bei geeigneten Problemen wird so die Anzahl benötigter Iterationen des Spaltengenerierungsverfahrens zum Erreichen einer guten Lösung veringert, und der Lösungsraum der Unterprobleme verkleinert. Allerdings muss das Lösungsverfahren für die Unterprobleme für diese zusätzlichen Einschränkungen geeignet sein.

Kapitel 8

Evaluation der Optimierungsverfahren

Die in der Arbeit vorgestellten Algorithmen sollen in diesem Kapitel nun experimentell untersucht werden. Grundlage hierfür ist neben der Implementation der Verfahren auch das Vorliegen von Probleminstanzen, anhand derer die Experimente durchgeführt werden.

8.1 Erzeugung von Probleminstanzen

Da das vorliegende Problem in dieser Form in der Literatur noch nicht betrachtet wurde, werden zur Evaluation der Algorithmen Instanzen künstlich unter Zuhilfenahme eines Zufallsgenerators[1] erzeugt. Das Verfahren zur Erzeugung ist dabei an den Planungsprozess aus der Praxis angelehnt, um die Existenz zumindest einer zulässigen Lösung der Instanzen zu gewährleisten. Diese Referenzlösung wird den Lösungsverfahren allerdings nicht mitgeteilt, so dass Verfahren auch ohne das Auffinden einer zulässigen Lösung abbrechen können (siehe zum Beispiel Abschnitt 8.2.4).

8.1.1 Parameter

Für die Erzeugung der Instanzen definieren wir die folgenden Parameter:

$|J|$ Anzahl der Aufträge

$|P|$ Anzahl der Produkte, diese bildet eine obere Schranke für die maximale Antikette innerhalb des Vorranggraphen der Instanz

$|\mathcal{G}|^M$ Anzahl der weiteren Materialgruppen

$S^{\min}$ Minimale Setupzeit in einer Materialgruppe

$S^{\max}$ Maximale Setupzeit in einer Gruppe

$M^{\min}$ Minimale Anzahl von Materialien in einer Gruppe prozentual zu $|P|$

[1] Als Zufallsgenerator wurde der Mersenne-Twister aus Matsumoto und Nishimura (1998) verwendet.

$M^{\max}$ Maximale Anzahl von Materialien in einer Gruppe prozentual zu $|P|$

$p^{\min}$ Minimale Prozesszeit eines Auftrags, hier wird davon ausgegangen, dass eine Einheit des Produkts in genau einer Zeiteinheit gefertigt werden kann

$p^{\max}$ Maximale Prozesszeit eines Auftrags, hier wird davon ausgegangen, dass eine Einheit des Produkts in genau einer Zeiteinheit gefertigt werden kann

$W^{\max}$ Maximales Gewicht eines Auftrags

$\Delta\delta^{\min}$ Minimale Relaxierung der Fälligkeit prozentual zur Gesamtprozesszeit $\sum p_j$

$\Delta\delta^{\max}$ Minimale Relaxierung der Fälligkeit prozentual zur Gesamtprozesszeit $\sum p_j$

$\Delta r^{\min}$ Minimale Relaxierung des Freigabezeitpunkts prozentual zur Gesamtprozesszeit $\sum p_j$

$\Delta r^{\max}$ Minimale Relaxierung des Freigabezeitpunkts prozentual zur Gesamtprozesszeit $\sum p_j$

$InvAgg$ Anzahl der Materiallieferungen eines Materials, die zu einer Lieferung zusammengefasst werden.

Um eine Instanz zu erzeugen, müssen die einzelnen Komponenten der Reihe nach erstellt werden, das Vorgehen wird nun in den nächsten Abschnitten erläutert.

8.1.2 Produkte und Materialien

Es werden $|\mathcal{G}^M|$ verschiedene Materialgruppen angelegt. Für jede Gruppe $G \in \mathcal{G}^M$ werden verschiedene Materialien angelegt, die Anzahl der Materialien in der Gruppe wird dabei anhand der Parameter $M^{\min}$ und $M^{\max}$ sowie $|P|$ bestimmt. Die Produkte bilden eine weitere Materialgruppe mit $|P|$ verschiedenen Materialien. Die Rüstzeit dieser Gruppe verschiedener Produkte charakterisiert die minimale Rüstzeit zwischen Aufträgen, so zum Beispiel die Bereitstellung von anderem Verpackungsmaterial. Diese wird, wie die anderen Gruppenrüstzeiten auch, aus dem Intervall $[S^{\min}, S^{\max}]$ zufällig gezogen. Um die Stücklisten zu erstellen, wird nun für jedes Material ein Produkt ausgewählt, in dem dieses Material vorhanden ist. Für die Produkte, die nach dieser Prozedur noch kein Material aus einer bestimmten Gruppe zugewiesen bekommen haben, wird zufällig ein Material aus der Gruppe ausgewählt. Somit ist jedes Material in mindestens einem Produkt vertreten, und jedes Produkt beinhaltet für alle weiteren Materialgruppen aus $\mathcal{G}^M$ jeweils einen Repräsentanten. Durch diese Zuordnung wird für jedes Produkt die Stückliste festgelegt, und die Rüstzeitmatrix kann nun bestimmt werden.

8.1.3 Aufträge

Zunächst werden $|J|$ leere Aufträge in einer Sequenz $\sigma = (1, \ldots, |J|)$ angelegt. Jedem Auftrag wird eine Menge an Einheiten $\sim U[1, W^{\max}]$ zur Bestimmung der Prozesszeit p_j und des Materialbedarfs $a_{j,m}$ für alle Materialien sowie ein

Gewicht $w_j \sim U[1, W^{\max}]$ zufällig zugewiesen. Dabei ist der Materialbedarf proportional zur Prozesszeit der Aufträge. Für jedes Produkt wird ein Auftrag aus der Menge der Aufträge bestimmt, so dass auch tatsächlich für jedes Produkt ein Auftrag vorliegt. Die übrigen Aufträge werden dann zufällig auf die Produkte verteilt, es wird also jedem Auftrag ein Produkt zufällig zugewiesen. Mit der Rüstmatrix lässt sich nun der Schedule σ auswerten. Wir gehen davon aus, dass immer alle benötigten Materialien verfügbar sind. Die Fertigstellungszeitpunkte der Aufträge $C_j(\sigma)$, sowie die Gesamtprozesszeit werden nun verwendet, um Fälligkeiten und Freigabezeitpunkte zu bestimmen. Dabei werden die aktuellen Fertigstellungszeiten $C_j(\sigma)$ als vorläufige Fälligkeit festgelegt, und dann um einen Wert $\Delta\delta_j \sim U[\Delta\delta^{\min}, \Delta\delta^{\max}]$ relaxiert. Die tatsächliche Fälligkeit δ_j von Auftrag j ergibt sich dann durch $\delta_j = C_j(\sigma) + \Delta\delta_j \cdot \sum_{j' \in J} p_{j'}/100$. Im Rahmen der Relaxierung der Fälligkeiten kann es dazu kommen, dass die Fälligkeiten der Aufträge eines Produkts nicht mehr die *EDD*-Reihenfolge einhalten. Zur Korrektur werden die Fälligkeitszeitpunkte der Aufträge eines Produkts in aufsteigender Reihenfolge diesen Aufträgen zugewiesen, und somit die Vorrangbeziehungen in Form von Ketten festgeschrieben. Ohne Korrektur wird auf gleiche Weise aus den Startzeiten $S_j(\sigma)$ der Aufträge in σ vorläufige Freigabezeitpunkte der Aufträge bestimmt, der Wert der Relaxierung Δr_j des Freigabezeitpunkts für Auftrag $j \in J$ wird bestimmt als $\Delta r_j \sim U[\Delta r^{\min}, \Delta r^{\max}]$, der vorläufige Freigabezeitpunkt r_j wird bestimmt durch $r_j = S_j(\sigma) - \Delta r_j \cdot \sum_{j' \in J} p_{j'}/100$.

Dieses Vorgehen stellt sicher, dass die Sequenz σ auf jeden Fall zulässig ist, und somit die erzeugte Instanz auch zulässig lösbar ist.

Ein ähnliches Vorgehen bezüglich der Zeitfenster findet sich in Dumas u. a. (1995), die zugrunde liegende Reihenfolge ist hier durch eine einfache Heuristik für das Problem des Handelsreisenden (TSP) erstellt worden. Die Abhängigkeit der Relaxierung der Zeitfenster von der Gesamtprozesszeit $\sum p_j$ wird auch in ähnlicher Weise in Davari u. a. (2015) angewendet.

8.1.4 Materiallieferungen

Zunächst werden für jeden Auftrag separate Materiallieferungen zum im vorigen Abschnitt bestimmten vorläufigen Freigabezeitpunkt angelegt. Da in der Praxis die Materiallieferungen meist aggregiert werden oder einer Bestellpolitik unterliegen, werden bis zu *InvAgg* Lieferungen eines Materials zusammengefasst und die Lieferung auf den frühesten Termin der betrachteten Lieferungen festgelegt. Dadurch kann ein Auftrag noch früher gestartet werden, und die Konkurrenz der Aufträge untereinander um die Lieferungen nimmt zu. Der vorläufige Freigabezeitpunkt wird dann nicht weiter betrachtet und bildet keinen Teil der Instanz.

8.1.5 Die Testsets

Die in der Auswertung der Verfahren betrachteten 16 Testsets enthalten jeweils 20 Instanzen, die mit den gleichen Parametereinstellungen erzeugt worden sind, die Parametereinstellungen selbst sind in Tabelle 8.1 angegeben.

Die Unterschiede in den Testsets betreffen zuallererst die Anzahl der Aufträge und die Anzahl der Produkte. Unabhängig davon lässt sich die Menge von 320 Instanzen dann noch in zwei weiteren Dimensionen in je zwei Gruppen teilen. Die eine Dimension betrifft die Rüstzeit, die zwischen den Wechseln eines

Instanzmenge	$\|J\|$	$\|P\|$	G	M^{min}	M^{max}	S^{min}	S^{max}	p^{min}	p^{max}	W^{max}	$\Delta\delta^{min}$	$\Delta\delta^{max}$	Δr^{min}	Δr^{max}	$InvAgg$
50_10_BigSetup	50	10	5	50	100	5	10	20	70	10	20	30	20	30	3
50_10_BigSetup_SmallDeadline	50	10	5	50	100	5	10	20	70	10	5	10	20	30	3
50_10_SmallSetup	50	10	5	10	50	2	5	50	100	10	20	30	20	30	3
50_10_SmallSetup_SmallDeadline	50	10	5	10	50	2	5	50	100	10	5	10	20	30	3
50_15_BigSetup	50	15	5	50	100	5	10	20	70	10	20	30	20	30	3
50_15_BigSetup_SmallDeadline	50	15	5	50	100	5	10	20	70	10	5	10	20	30	3
50_15_SmallSetup	50	15	5	10	50	2	5	50	100	10	20	30	20	30	3
50_15_SmallSetup_SmallDeadline	50	15	5	10	50	2	5	50	100	10	5	10	20	30	3
75_10_BigSetup	75	10	5	50	100	5	10	20	70	10	20	30	20	30	3
75_10_BigSetup_SmallDeadline	75	10	5	50	100	5	10	20	70	10	5	10	20	30	3
75_10_SmallSetup	75	10	5	10	50	2	5	50	100	10	20	30	20	30	3
75_10_SmallSetup_SmallDeadline	75	10	5	10	50	2	5	50	100	10	5	10	20	30	3
75_15_BigSetup	75	15	5	50	100	5	10	20	70	10	20	30	20	30	3
75_15_BigSetup_SmallDeadline	75	15	5	50	100	5	10	20	70	10	5	10	20	30	3
75_15_SmallSetup	75	15	5	10	50	2	5	50	100	10	20	30	20	30	3
75_15_SmallSetup_SmallDeadline	75	15	5	10	50	2	5	50	100	10	5	10	20	30	3

Tabelle 8.1: Testinstanzen

Produktes anfällt, im Vergleich zur Prozesszeit der Aufträge, hier gibt es einmal Instanzen mit relativ hoher Rüstzeit, und andererseits Instanzen mit relativ niedriger Rüstzeit. In der anderen Dimension wird die Fälligkeit der Aufträge näher betrachtet, einmal werden relativ weite Zeitfenster ermöglicht, im anderen Fall wird die Fälligkeit eingeschränkt. Zu bemerken ist hierbei, dass bei Wahl der Instanzen mit relativ hoher Rüstzeit auch die Fälligkeit der Aufträge mit erniedrigt wird, da die Fälligkeit in der Instanzerzeugung von der Gesamtprozesszeit der Aufträge abhängt, die in diesen Instanzen kleiner ausfällt. In der folgenden Diskussion der Ergebnisse werden wir dann auch auf den Einfluss dieser Dimensionen eingehen.

8.2 Testergebnisse

In den folgenden Abschnitten werden wir gemäß der Reihenfolge der Kapitel nun die einzelnen entwickelten Verfahren gegen die oben vorgestellten Instanzen testen und auf einige Aspekte und Auffälligkeiten der Ergebnisse eingehen.

8.2.1 Implementierung der Verfahren

Die in den vorherigen Kapiteln vorgestellten Verfahren wurden in C++ implementiert. Um den Auswertungsprozess zu unterstützen, werden die Instanzen, die Konfigurationen und die wichtigsten Kennzahlen, die während der Lösung der Probleme ermittelt werden, in einer MySQL-Datenbank abgelegt, deren relationales Schema in Abbildung 8.1 wiedergegeben ist. Eine zentrale Rolle für die Durchführung der Experimente nimmt die Relation Run ein, die einen Lauf eines bestimmten Lösungsalgorithmus auf einer bestimmten Instanz dokumentiert. Das C++-Programm kann dann mit Hilfe der in der Relation hinterlegten Information hinsichtlich der zu lösenden Instanz und Konfiguration genau diese angeforderte Berechnung durchführen. Die Ergebnisse werden als obere (Relation Solution) und untere Schranken (Relation LowerBound) sowie Statistische Werte (Relation StatisticValue) für einzelne statistische Parameter gespeichert. Dadurch können auch die verschiedenen Verfahrensklassen gut hinsichtlich erreichter oberer und unterer Schranken miteinander verglichen werden.

Die Berechnungen wurden auf Rechnern mit einem Intel Core i7 Prozessor mit $4 \times 4,0$ Gigahertz (für die Evaluation in Abschnitt 8.2.6.2 $4 \times 3,4$ Gigahertz) und mit 32 Gigabyte Arbeitsspeicher durchgeführt. Durch die Hyperthreading-Technologie konnten in den parallelisierten Berechnungen bis zu 8 Threads gleichzeitig ausgeführt werden.

8.2.2 Ergebnisse der Vorbehandlung

Die Ergebnisse der Vorbehandlung, die die Eigenschaften zulässiger Lösungen eingrenzen soll, sind in den Tabellen 8.3 und 8.2 zusammengefasst. Vor allem in den Instanzen mit engeren Zeitfenstern, sowie relativ großen Rüstzeiten ist hier eine deutliche Zunahme der Vorrangbeziehungen und eine Verkleinerung der Zeit- und Positionsfenster festzustellen. Als ursprüngliche Vorrangbeziehungen wurden hier alle Vorrangbeziehungen aus dem transitiven Abschluss der Produktketten gezählt, und diese werden dann mit allen Vorrangbeziehungen aus dem transitiven Abschluss nach Durchführung der Vorbehandlung verglichen.

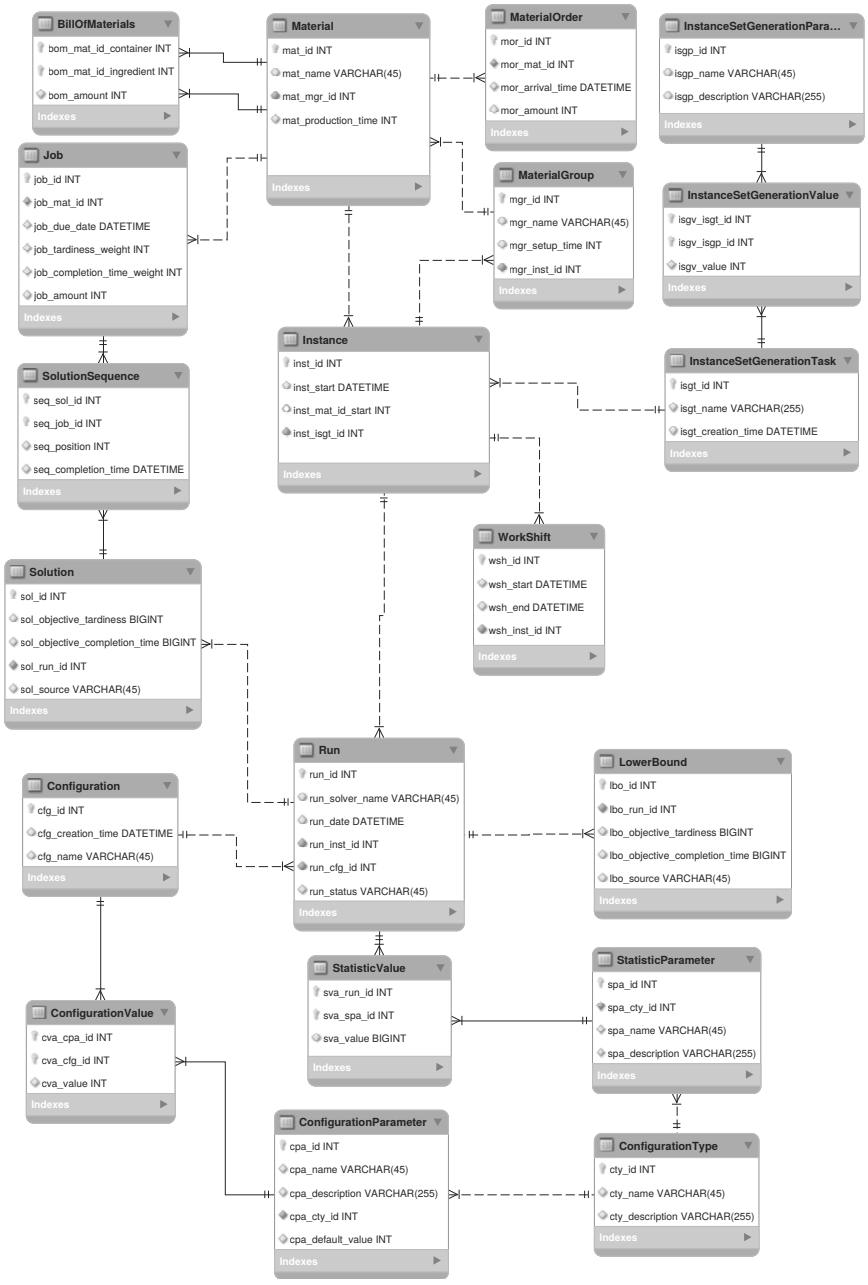


Abbildung 8.1: Datenbankmodell

Instanzmenge	Total	$\varnothing\|\mathcal{C}\|$	$\varnothing\|E(G^C)\|$ vorher	$\varnothing\|E(G^C)\|$ nachher	Zuwachs in %
50_10_BigSetup	20	9,65	117,9	617,2	423,49
50_10_BigSetup_SmallDeadline	20	8,95	116,75	812,7	596,1
50_10_SmallSetup	20	10	118,15	211,6	79,09
50_10_SmallSetup_SmallDeadline	20	10	117,6	439,5	273,72
50_15_BigSetup	20	12,8	73,7	585,85	694,9
50_15_BigSetup_SmallDeadline	20	10,85	74,55	794,05	965,12
50_15_SmallSetup	20	14,75	73,65	208,2	182,68
50_15_SmallSetup_SmallDeadline	20	13,85	73,85	455,95	517,4
75_10_BigSetup	20	10	273	1.412,95	417,56
75_10_BigSetup_SmallDeadline	20	9,75	272,55	1.825,05	569,62
75_10_SmallSetup	20	10	277,4	427,2	54
75_10_SmallSetup_SmallDeadline	20	9,95	272,1	919,8	238,04
75_15_BigSetup	20	14,15	176,9	1.322,55	647,62
75_15_BigSetup_SmallDeadline	20	12,85	179,65	1.781	891,36
75_15_SmallSetup	20	15	175,5	458,55	161,28
75_15_SmallSetup_SmallDeadline	20	14,9	178,7	951,9	432,68

Tabelle 8.2: Ergebnisse der Vorbehandlung - Vorrangbeziehungen

Instanzmenge	*Total*	*∅ Zeit(s)*	*∅ Zeit(s) Material*	*∅ Zeitfenster vorher*	*∅ Zeitfenster nachher*	*Abnahme in %*	*∅ Pos. vorher*	*∅ Pos. nachher*	*Abnahme in %*
50_10_BigSetup	20	$1{,}4 \cdot 10^{-2}$	$3{,}55 \cdot 10^{-2}$	75.694,75	67.590	10,71	2.264,2	1.193,55	47,29
50_10_BigSetup_SmallDeadline	20	$1{,}02 \cdot 10^{-2}$	$8{,}17 \cdot 10^{-3}$	56.739,05	45.057,95	20,59	2.266,5	794,6	64,94
50_10_SmallSetup	20	$3{,}72 \cdot 10^{-2}$	0,19	$1{,}3 \cdot 10^{5}$	$1{,}26 \cdot 10^{5}$	3,4	2.263,7	1.672,15	26,13
50_10_SmallSetup_SmallDeadline	20	$2 \cdot 10^{-2}$	$2{,}27 \cdot 10^{-2}$	97.441,55	82.109,3	15,74	2.264,8	1.121,15	50,5
50_15_BigSetup	20	$2{,}46 \cdot 10^{-2}$	$9{,}17 \cdot 10^{-2}$	78.311,8	70.085,55	10,5	2.352,6	1.250,8	46,83
50_15_BigSetup_SmallDeadline	20	$1{,}27 \cdot 10^{-2}$	$1{,}22 \cdot 10^{-2}$	55.911,15	45.783,15	18,11	2.350,9	822,75	65
50_15_SmallSetup	20	0,14	0,54	$1{,}32 \cdot 10^{5}$	$1{,}25 \cdot 10^{5}$	4,71	2.352,7	1.689,95	28,17
50_15_SmallSetup_SmallDeadline	20	$3{,}29 \cdot 10^{-2}$	$4{,}63 \cdot 10^{-2}$	97.366,45	82.651,5	15,11	2.352,3	1.134,55	51,77
75_10_BigSetup	20	$5{,}34 \cdot 10^{-2}$	0,36	$1{,}63 \cdot 10^{5}$	$1{,}51 \cdot 10^{5}$	7,38	5.079	2.649,3	47,84
75_10_BigSetup_SmallDeadline	20	$3{,}07 \cdot 10^{-2}$	$5{,}8 \cdot 10^{-2}$	$1{,}18 \cdot 10^{5}$	99.616,1	15,77	5.079,9	1.806,35	64,44
75_10_SmallSetup	20	0,24	2,58	$3 \cdot 10^{5}$	$2{,}96 \cdot 10^{5}$	1,59	5.070,2	3.833,7	24,39
75_10_SmallSetup_SmallDeadline	20	$8{,}86 \cdot 10^{-2}$	0,24	$2{,}2 \cdot 10^{5}$	$1{,}91 \cdot 10^{5}$	13,05	5.080,8	2.620,95	48,42
75_15_BigSetup	20	0,17	1,26	$1{,}71 \cdot 10^{5}$	$1{,}56 \cdot 10^{5}$	8,81	5.271,2	2.806	46,77
75_15_BigSetup_SmallDeadline	20	$5{,}2 \cdot 10^{-2}$	0,13	$1{,}27 \cdot 10^{5}$	$1{,}02 \cdot 10^{5}$	19,77	5.265,7	1.872,15	64,45
75_15_SmallSetup	20	1,61	11,85	$2{,}99 \cdot 10^{5}$	$2{,}87 \cdot 10^{5}$	4,14	5.274	3.834	27,3
75_15_SmallSetup_SmallDeadline	20	0,26	0,7	$2{,}22 \cdot 10^{5}$	$1{,}91 \cdot 10^{5}$	13,8	5.267,6	2.660,15	49,5

Tabelle 8.3: Ergebnisse der Vorbehandlung - Zeiten und Zeitfenster

Auch bei der Größe der Zeit- und Positionsfenster wurde in der ursprünglichen Variante nur die Vorrangbeziehungen in Form von Produktketten berücksichtigt, um eine frühestmögliche Position, bzw. Zeit für einen Auftrag festzulegen. Manche Instanzgruppen weisen dann auch eine Reduktion der Dilworth-Partitionierung in Ketten auf, die dann zum Beispiel auch die Menge der zu berücksichtigten Pricing-Probleme im Branch-and-Price-and-Cut-Ansatz verringert. Die Zeiten für die eigentliche Vorbehandlung sind sehr überschaubar, die Zeit, die notwendig ist, um die minimalen Materiallieferzeitpunkte für alle Aufträge auf allen Positionen zu bestimmen, ist aber deutlich größer (Spalte Zeit(s) Material, siehe auch Abschnitt 4.3.4.2). Insgesamt erhalten wir durch die Anwendung der Vorbehandlung für einen geringen zeitlichen Aufwand einen besseren Einblick in die Struktur der zulässigen Lösungen einer Instanz, wobei diese Verbesserung sehr stark von den Fälligkeiten und der Größe der Rüstzeiten abzuhängen scheint.

8.2.3 Die Tabusuche

In Tabelle 8.4 sind die Ergebnisse der Tabusuche dargestellt. Die Spalte # Beste UB zählt dabei alle Instanzen, für die die beste bekannte obere Schranke auch durch die Tabusuche erzielt wurde, die Spalte ∅ Abstand beste UB gibt den prozentualen durchschnittlichen Abstand der in der Tabusuche erzielten besten Lösung von der besten bekannten oberen Schranke an. Die weiteren Spalten neben der Laufzeit des Verfahrens geben an, wie oft sich eine globale beziehungsweise lokale Verbesserung des Zielfunktionswertes im Lauf des Verfahrens ergeben hat. Hierbei wurden in jeder Instanz 10000 Iterationen durchgeführt, auch bei längerfristigem Nicht-Erzielen einer global besseren Lösung wurde das Verfahren nicht abgebrochen. Insgesamt hat das Verfahren nur in seltenen Fällen die beste bekannte Lösung für die verschiedenen Instanzen gefunden. Eine Ausnahme bildet dabei die letzte Gruppe von Instanzen, auf der allerdings nicht alle weiteren exakten Verfahren getestet wurden. Die angegebene Laufzeit wurde ohne Einsatz von Techniken zur inkrementellen Nachbarschaftsevaluation erzielt, so dass bei Zwischenspeicherung von sinnvollen Informationen noch eine Verbesserung der Laufzeit möglich sein sollte. Es wurde immer die komplette Nachbarschaft durchsucht, also jeder Block an alle möglichen Positionen verschoben. Die Laufzeitdifferenzen lassen sich zum einen durch die Instanzgröße erklären, die die Evaluationszeit einer Lösung beeinflusst, zum anderen aber auch durch die Struktur der Nachbarschaft. Die Instanzen mit weniger Produkten verringern die Anzahl an Produktblöcken, die Instanzen mit hohen Rüstzeiten haben eine größere Materialvariabilität, und Lösungen enthalten dadurch mehr Blöcke pro Material. Liegen mehr Blöcke vor, sind auch mehr Verschiebungen möglich, und die Nachbarschaft vergrößert sich. Die Zeit für die Evaluation der Nachbarschaft steigt damit an.

8.2.4 Das IP-Model

Für die Untersuchung der Effizienz der ganzzahligen Programmierung zur Lösung des in dieser Arbeit behandelten Schedulingproblems wurde das Modell auf Basis der Formulierung von Manne (1960) mit IBM ILOG CPLEX 12.6 getestet. Die Laufzeit des Verfahrens wurde auf eine Stunde begrenzt, die maximale Größe des unkomprimierten Branch-and-Cut-Baumes auf 32 Gigabyte

Instanzmenge	Total	# Beste UB	∅ Abstand beste UB	∅ Laufzeit(s)	∅ Globale Verb.	∅ Lokale Verb.
50_10_BigSetup	20	5	1,05%	10,0	35,3	4.386,2
50_10_BigSetup_SmallDeadline	20	7	0,72%	10,6	30,4	4.391,05
50_10_SmallSetup	20	1	1,24%	9,0	38	4.374,15
50_10_SmallSetup_SmallDeadline	20	1	0,71%	9,1	26,55	4.361,15
50_15_BigSetup	20	1	1,14%	13,5	38,05	4.437,7
50_15_BigSetup_SmallDeadline	20	6	0,74%	13,8	31,85	4.456,55
50_15_SmallSetup	20	1	0,93%	11,7	41,95	4.381,65
50_15_SmallSetup_SmallDeadline	20	0	1,07%	13,0	27,9	4.356,15
75_10_BigSetup	20	0	1,52%	25,0	51,45	4.495,95
75_10_BigSetup_SmallDeadline	20	0	0,83%	25,7	45,25	4.563,55
75_10_SmallSetup	20	0	1,11%	20,2	57,7	4.331,95
75_10_SmallSetup_SmallDeadline	20	0	1,17%	21,5	42,25	4.406,5
75_15_BigSetup	20	0	1,93%	34,5	57,55	4.536,45
75_15_BigSetup_SmallDeadline	20	1	1,29%	35,0	50,95	4.540,55
75_15_SmallSetup	20	0	1,70%	30,3	58,7	4.269,7
75_15_SmallSetup_SmallDeadline	20	12	0,59%	32,0	47,1	4.370,8

Tabelle 8.4: Ergebnisse der Tabusuche (10000 Iterationen)

beschränkt. Die in der Vorbehandlung ermittelten Vorrangbeziehungen wurden hier zur Fixierung der $\pi_{i,j}$ Variablen verwendet, auch das Zeitfenster für die Fertigstellungszeit wurde dem Solver mitgeteilt. Die Abschätzung der Rüstzeiten durch Ungleichung (5.11) und die Bedingungen an die verschachtelten Vorgängermengen bei Vorliegen von Vorrangbeziehungen waren aktiviert.

Die Ergebnisse sind in Tabelle 8.5 zusammengefasst. Die Spalte ∅ Lösungen listet die durchschnittliche Anzahl an gefundenen zulässigen Lösungen pro Instanz auf. Wie oft die beste bekannte untere Schranke über alle Verfahren vom IP-Solver erreicht wurde, ist Spalte # Beste LB zu entnehmen, wohingegen der durchschnittliche prozentuale Abstand der unteren Schranke des IP-Solvers gegenüber der insgesamt besten unteren Schranke in Spalte ∅ Abstand beste LB verzeichnet ist. Dabei fällt auf, dass bei den kleineren Instanzen eher die maximale Baumgröße erreicht wurde, als das Zeitlimit. Keine der Instanzen wurde unter den vorgegebenen Rahmenbedingungen optimal gelöst. In den großen Instanzen mit kleinen Fälligkeiten lässt sich feststellen, dass sich der Solver offenbar sehr schwer damit getan hat, überhaupt zulässige Lösungen zu finden.

8.2.5 Das Branch-and-Bound-Verfahren

Das Branch-and-Bound-Verfahren bietet durch die Vielzahl an unteren Schranken und die Dominanztabelle viele Möglichkeiten zur individuellen Konfiguration. Deswegen werden die im Rahmen der vorgenommenen Evaluation getesteten Konfigurationen in Tabelle 8.6 vorgestellt.

Die Konfigurationen unterscheiden sich dabei im Wesentlichen durch die

Instanzmenge	# Beste LB	∅ Abstand beste LB	∅ Laufzeit(s)	∅ Knoten	∅ Lösungen
50_10_BigSetup	0	2,74%	3.221,7	$3{,}82 \cdot 10^5$	131,45
50_10_BigSetup_SmallDeadline	0	3,68%	2.973,4	$9{,}2 \cdot 10^5$	151,25
50_10_SmallSetup	0	3,72%	3.362,5	$2{,}47 \cdot 10^5$	235,95
50_10_SmallSetup_SmallDeadline	0	5,92%	3.531,7	$4{,}86 \cdot 10^5$	74,5
50_15_BigSetup	0	4,11%	3.211,4	$3{,}29 \cdot 10^5$	191,15
50_15_BigSetup_SmallDeadline	0	5,78%	3.048,9	$7{,}14 \cdot 10^5$	149,45
50_15_SmallSetup	0	4,78%	3.488,7	$1{,}73 \cdot 10^5$	334
50_15_SmallSetup_SmallDeadline	0	8,11%	3.588,7	$3{,}11 \cdot 10^5$	51,2
75_10_BigSetup	0	2,66%	3.607,3	60.182,75	175,25
75_10_BigSetup_SmallDeadline	0	3,13%	3.309,1	$1{,}71 \cdot 10^5$	332,65
75_10_SmallSetup	0	2,08%	3.608,3	30.687,3	8,2
75_10_SmallSetup_SmallDeadline	0	5,48%	3.610,1	48.822,15	0
75_15_BigSetup	2	3,00%	3.611,9	53.565,35	171,85
75_15_BigSetup_SmallDeadline	0	4,13%	3.463,1	$1{,}54 \cdot 10^5$	216,5
75_15_SmallSetup	13	0,22%	3.602,9	14.882,2	5,45
75_15_SmallSetup_SmallDeadline	0	6,84%	3.606,0	30.202,45	0

Tabelle 8.5: Ergebnisse des IP-Models

Konfig.	Nodes	LevelSize	Encodings	Labels	LB Chain	LB VSP	LB LAP	LB Belouadah	LB PanShi	LB Posner	LB LAP + LR
BnBSimpleBounds	$1 \cdot 10^8$	30	$2 \cdot 10^7$	$1 \cdot 10^8$	✓	✓	–	✓	✓	✓	0
BnBChain	$1 \cdot 10^8$	30	$2 \cdot 10^7$	$1 \cdot 10^8$	✓	–	–	–	–	–	0
BnBLAP	$1 \cdot 10^8$	30	$2 \cdot 10^7$	$1 \cdot 10^8$	–	–	✓	–	–	–	0
BnBVSP	$1 \cdot 10^8$	30	$2 \cdot 10^7$	$1 \cdot 10^8$	–	✓	–	–	–	–	0
BnBBelouadah	$1 \cdot 10^8$	30	$2 \cdot 10^7$	$1 \cdot 10^8$	–	–	–	✓	–	–	0
BnBLAPLagr20	$1 \cdot 10^8$	30	$2 \cdot 10^7$	$1 \cdot 10^8$	–	–	–	–	–	–	20
BnBPosner	$1 \cdot 10^8$	30	$2 \cdot 10^7$	$1 \cdot 10^8$	–	–	–	–	–	✓	0
BnBPanShi	$1 \cdot 10^8$	30	$2 \cdot 10^7$	$1 \cdot 10^8$	–	–	–	–	✓	–	0
BnBChainNoDom	$1 \cdot 10^8$	30	0	0	✓	–	–	–	–	–	0
BnBChainNoDom3xNodes	$3 \cdot 10^8$	10	0	0	✓	–	–	–	–	–	0
BnBSimpleBoundsAndLAP	$1 \cdot 10^8$	30	$2 \cdot 10^7$	$1 \cdot 10^8$	✓	✓	✓	✓	✓	✓	0

Tabelle 8.6: Konfigurationen des Branch-and-Bound-Verfahrens

Auswahl der verwendeten Schranken. Während der Evaluation eines neuen Knoten im Branch-and-Bound-Verfahren nimmt die Berechnung der unteren Schranke und damit die Abschätzung der optimalen Kosten eines Schedules, der mit dem Teilschedule des Knotens beginnt, die meiste Rechenzeit in Anspruch. Die gewählte untere Schranke wirkt sich einerseits auf die Anzahl der aufgrund der unteren Schranke aus dem Baum eliminierten Knoten aus, andererseits wird die Prioritätsreihenfolge in denjenigen Knotenauswahlverfahren, die auch auf der unteren Schranke basieren, beeinflusst. Die Testläufe des Branch-and-Bound-Verfahrens unterliegen hier grundsätzlich keinem Zeitlimit, aber der Speicherplatz wird durch eine Anzahl von maximal gleichzeitig verfügbaren Knoten im Branch-and-Bound-Baum begrenzt. Dadurch, dass Knoten erst in den Branch-and-Bound-Baum übernommen werden, wenn sie einen Selektionsprozess durch Zulässigkeitstests, Dominanzregeln und untere Schranken überstanden haben, können wesentlich mehr Knoten untersucht werden, als durch diesen Konfigurationsparameter vorgegeben werden. Zudem werden Knoten, die nicht mehr für den Optimalitätsbeweis benötigt werden und keine Kindknoten (mehr) besitzen, aus dem Baum gelöscht, sofern sie bei der Suche zufällig angetroffen werden. Das kann entweder bei Überprüfung der Dominanzkriterien passieren oder wenn der letzte Kindknoten eines bereits vollständig verzweigten Knotens gelöscht wird, und erhöht zusätzlich die Anzahl von insgesamt evaluierbaren Knoten.

Während des Suchprozesses wurde durchgehend die Bestensuche angewendet, also immer der aktive Knoten mit der niedrigsten unteren Schranke verzweigt.

In Tabelle 8.7 sind die Ergebnisse bei Verwendung von fünf auf Relaxationen des betrachteten Schedulingproblems basierenden Schranken aufgelistet, während Tabelle 8.8 die Ergebnisse bei ausschließlicher Verwendung der Schranke auf Basis von Ketten angibt.

Die Spalten in diesen Tabellen und allen weiteren Tabellen für die Evaluation des Branch-and-Bound-Verfahrens geben neben der Laufzeit die Anzahl an durchgeführten Verzweigungen, sowie die Anzahl der insgesamt evaluierten Knoten an. Ein Knoten gilt bereits ab dem Zeitpunkt als evaluiert, wenn er eine gültige Erweiterung des Teilschedules seines Vaterknotens darstellt. Darum geben dann die folgenden Spalten an, wie oft ein Knoten aufgrund von Dominanzkriterien, vorhergesagter Unzulässigkeit der vollständigen Lösung oder zu hoher unterer Schranke erst gar nicht in den Baum aufgenommen worden ist. Natürlich konnte der Fall der Unzulässigkeit oder der zu hohen unteren Schranke dann nicht mehr auftreten, wenn bereits eines der vorherigen Kriterien gegriffen hat.

Die angegebenen Laufzeiten in den soeben erwähnten beiden Tabellen machen deutlich, dass die Einbindung von mehreren Schranken die Laufzeit des Verfahrens massiv ansteigen lässt, während die Anzahl der Knoten, die für den erfolgreichen Optimalitätsbeweis ausgewertet werden müssen, abnimmt.

Instanzmenge	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
50_10_BigSetup	20	20	16,5	$3{,}6 \cdot 10^5$	$2{,}41 \cdot 10^6$	$1{,}8 \cdot 10^6$	$1{,}25 \cdot 10^5$	29.674,75
50_10_BigSetup_SmallDeadline	20	20	1,4	36.744	$2{,}16 \cdot 10^5$	$1{,}44 \cdot 10^5$	24.138,8	2.432,1
50_10_SmallSetup	20	20	540,7	$7{,}58 \cdot 10^6$	$7{,}03 \cdot 10^7$	$5{,}24 \cdot 10^7$	$4{,}96 \cdot 10^6$	$8{,}11 \cdot 10^5$
50_10_SmallSetup_SmallDeadline	20	20	11,2	$2{,}25 \cdot 10^5$	$1{,}97 \cdot 10^6$	$1{,}26 \cdot 10^6$	$3{,}83 \cdot 10^5$	16.021,35
50_15_BigSetup	20	20	33,3	$6{,}33 \cdot 10^5$	$5{,}04 \cdot 10^6$	$3{,}76 \cdot 10^6$	$3{,}71 \cdot 10^5$	59.816,8
50_15_BigSetup_SmallDeadline	20	20	2,0	45.471	$2{,}89 \cdot 10^5$	$1{,}82 \cdot 10^5$	36.788,6	6.829,75
50_15_SmallSetup	20	14	3.801,8	$3{,}94 \cdot 10^7$	$4{,}92 \cdot 10^8$	$3{,}63 \cdot 10^8$	$4{,}24 \cdot 10^7$	$7{,}65 \cdot 10^6$
50_15_SmallSetup_SmallDeadline	20	20	37,8	$6{,}74 \cdot 10^5$	$7{,}61 \cdot 10^6$	$5{,}09 \cdot 10^6$	$1{,}52 \cdot 10^6$	34.791,45
75_10_BigSetup	20	20	721,4	$8{,}75 \cdot 10^6$	$6{,}66 \cdot 10^7$	$5{,}31 \cdot 10^7$	$2{,}27 \cdot 10^6$	$1{,}55 \cdot 10^5$
75_10_BigSetup_SmallDeadline	20	20	42,9	$5{,}83 \cdot 10^5$	$3{,}89 \cdot 10^6$	$2{,}87 \cdot 10^6$	$2{,}84 \cdot 10^5$	7.049,55
75_10_SmallSetup	20	1	11.329,8	$6{,}44 \cdot 10^7$	$6{,}2 \cdot 10^8$	$4{,}31 \cdot 10^8$	$3{,}26 \cdot 10^7$	$1{,}89 \cdot 10^6$
75_10_SmallSetup_SmallDeadline	20	20	884,3	$1{,}03 \cdot 10^7$	$9{,}62 \cdot 10^7$	$6{,}92 \cdot 10^7$	$1{,}2 \cdot 10^7$	72.765,9
75_15_BigSetup	20	18	4.942,7	$4{,}43 \cdot 10^7$	$4{,}07 \cdot 10^8$	$3{,}15 \cdot 10^8$	$2{,}36 \cdot 10^7$	$1{,}13 \cdot 10^6$
75_15_BigSetup_SmallDeadline	20	20	162,2	$2{,}12 \cdot 10^6$	$1{,}69 \cdot 10^7$	$1{,}27 \cdot 10^7$	$1{,}39 \cdot 10^6$	27.923,4
75_15_SmallSetup	20	0	11.809,5	$4{,}01 \cdot 10^7$	$5{,}34 \cdot 10^8$	$3{,}66 \cdot 10^8$	$2{,}65 \cdot 10^7$	$7{,}19 \cdot 10^5$
75_15_SmallSetup_SmallDeadline	20	8	7.208,3	$5{,}38 \cdot 10^7$	$6{,}88 \cdot 10^8$	$4{,}38 \cdot 10^8$	$1{,}24 \cdot 10^8$	$1{,}49 \cdot 10^6$

Tabelle 8.7: Ergebnisse für die Konfiguration BnBSimpleBounds

Instanzmenge	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
50_10_BigSetup	20	20	2,4	$4{,}79 \cdot 10^5$	$3{,}15 \cdot 10^6$	$2{,}29 \cdot 10^6$	$1{,}97 \cdot 10^5$	26.680,55
50_10_BigSetup_SmallDeadline	20	20	$1{,}8 \cdot 10^{-1}$	48.820,9	$2{,}84 \cdot 10^5$	$1{,}84 \cdot 10^5$	34.709	2.512,35
50_10_SmallSetup	20	20	72,5	$9{,}36 \cdot 10^6$	$8{,}61 \cdot 10^7$	$6{,}42 \cdot 10^7$	$6{,}59 \cdot 10^6$	$8{,}5 \cdot 10^5$
50_10_SmallSetup_SmallDeadline	20	20	1,5	$3{,}26 \cdot 10^5$	$2{,}82 \cdot 10^6$	$1{,}85 \cdot 10^6$	$5{,}44 \cdot 10^5$	13.043,3
50_15_BigSetup	20	20	6,1	$9{,}82 \cdot 10^5$	$7{,}74 \cdot 10^6$	$5{,}6 \cdot 10^6$	$7{,}2 \cdot 10^5$	73.920,5
50_15_BigSetup_SmallDeadline	20	20	$2{,}5 \cdot 10^{-1}$	60.165,35	$3{,}77 \cdot 10^5$	$2{,}32 \cdot 10^5$	51.623,4	7.851,75
50_15_SmallSetup	20	11	615,5	$5{,}69 \cdot 10^7$	$6{,}95 \cdot 10^8$	$4{,}85 \cdot 10^8$	$7{,}92 \cdot 10^7$	$1{,}87 \cdot 10^7$
50_15_SmallSetup_SmallDeadline	20	20	7,1	$1{,}06 \cdot 10^6$	$1{,}19 \cdot 10^7$	$8{,}14 \cdot 10^6$	$2{,}34 \cdot 10^6$	33.178,1
75_10_BigSetup	20	20	86,6	$1{,}04 \cdot 10^7$	$7{,}89 \cdot 10^7$	$6{,}11 \cdot 10^7$	$3{,}44 \cdot 10^6$	$1{,}78 \cdot 10^5$
75_10_BigSetup_SmallDeadline	20	20	4,1	$7{,}06 \cdot 10^5$	$4{,}67 \cdot 10^6$	$3{,}3 \cdot 10^6$	$4{,}23 \cdot 10^5$	8.022,55
75_10_SmallSetup	20	1	766,0	$6{,}54 \cdot 10^7$	$6{,}29 \cdot 10^8$	$4{,}36 \cdot 10^8$	$3{,}49 \cdot 10^7$	$1{,}79 \cdot 10^6$
75_10_SmallSetup_SmallDeadline	20	20	117,6	$1{,}29 \cdot 10^7$	$1{,}19 \cdot 10^8$	$8{,}64 \cdot 10^7$	$1{,}52 \cdot 10^7$	49.090,85
75_15_BigSetup	20	15	646,3	$5{,}86 \cdot 10^7$	$5{,}38 \cdot 10^8$	$3{,}97 \cdot 10^8$	$4{,}09 \cdot 10^7$	$2{,}88 \cdot 10^6$
75_15_BigSetup_SmallDeadline	20	20	21,5	$2{,}7 \cdot 10^6$	$2{,}13 \cdot 10^7$	$1{,}53 \cdot 10^7$	$2{,}15 \cdot 10^6$	31.464,6
75_15_SmallSetup	20	0	681,2	$4{,}26 \cdot 10^7$	$5{,}62 \cdot 10^8$	$3{,}88 \cdot 10^8$	$2{,}91 \cdot 10^7$	$5{,}89 \cdot 10^5$
75_15_SmallSetup_SmallDeadline	20	3	791,5	$6{,}07 \cdot 10^7$	$7{,}75 \cdot 10^8$	$4{,}76 \cdot 10^8$	$1{,}61 \cdot 10^8$	$3{,}12 \cdot 10^5$

Tabelle 8.8: Ergebnisse für die Konfiguration BnBChain

Instanzmenge	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
50_10_BigSetup	20	20	26,4	$3{,}51 \cdot 10^5$	$2{,}36 \cdot 10^6$	$1{,}79 \cdot 10^6$	$1{,}14 \cdot 10^5$	28.712,05
50_10_BigSetup_SmallDeadline	20	20	2,0	36.126,05	$2{,}14 \cdot 10^5$	$1{,}44 \cdot 10^5$	23.260,3	2.560,2
50_10_SmallSetup	20	20	3.173,9	$2{,}96 \cdot 10^7$	$2{,}7 \cdot 10^8$	$2{,}09 \cdot 10^8$	$1{,}61 \cdot 10^7$	$3{,}17 \cdot 10^6$
50_10_SmallSetup_SmallDeadline	20	20	20,9	$3{,}18 \cdot 10^5$	$2{,}75 \cdot 10^6$	$1{,}83 \cdot 10^6$	$4{,}97 \cdot 10^5$	17.473,8
50_15_BigSetup	20	20	63,4	$7{,}48 \cdot 10^5$	$6 \cdot 10^6$	$4{,}55 \cdot 10^6$	$4{,}02 \cdot 10^5$	69.706,35
50_15_BigSetup_SmallDeadline	20	20	4,0	76.509,5	$5{,}34 \cdot 10^5$	$3{,}53 \cdot 10^5$	61.073,85	11.213,25
50_15_SmallSetup	20	7	11.155,8	$5{,}76 \cdot 10^7$	$7{,}16 \cdot 10^8$	$5{,}1 \cdot 10^8$	$6{,}39 \cdot 10^7$	$1{,}48 \cdot 10^7$
50_15_SmallSetup_SmallDeadline	20	20	69,1	$1 \cdot 10^6$	$1{,}12 \cdot 10^7$	$7{,}77 \cdot 10^6$	$2{,}03 \cdot 10^6$	36.632,15
75_10_BigSetup	20	20	1.710,9	$8{,}22 \cdot 10^6$	$6{,}25 \cdot 10^7$	$5{,}02 \cdot 10^7$	$2{,}04 \cdot 10^6$	$1{,}5 \cdot 10^5$
75_10_BigSetup_SmallDeadline	20	20	84,7	$5{,}74 \cdot 10^5$	$3{,}83 \cdot 10^6$	$2{,}87 \cdot 10^6$	$2{,}69 \cdot 10^5$	7.360,65
75_10_SmallSetup	20	0	42.974,5	$6{,}6 \cdot 10^7$	$6{,}42 \cdot 10^8$	$4{,}68 \cdot 10^8$	$1{,}47 \cdot 10^7$	$4{,}4 \cdot 10^5$
75_10_SmallSetup_SmallDeadline	20	20	2.373,9	$1{,}58 \cdot 10^7$	$1{,}45 \cdot 10^8$	$1{,}07 \cdot 10^8$	$1{,}64 \cdot 10^7$	50.652,95
75_15_BigSetup	20	18	11.539,8	$4{,}35 \cdot 10^7$	$3{,}99 \cdot 10^8$	$3{,}12 \cdot 10^8$	$2{,}15 \cdot 10^7$	$1{,}34 \cdot 10^6$
75_15_BigSetup_SmallDeadline	20	20	288,9	$2{,}08 \cdot 10^6$	$1{,}65 \cdot 10^7$	$1{,}27 \cdot 10^7$	$1{,}27 \cdot 10^6$	19.261,1
75_15_SmallSetup	20	0	45.872,2	$4{,}12 \cdot 10^7$	$5{,}58 \cdot 10^8$	$4{,}08 \cdot 10^8$	$8{,}7 \cdot 10^6$	$2{,}75 \cdot 10^5$
75_15_SmallSetup_SmallDeadline	20	2	17.152,4	$5{,}95 \cdot 10^7$	$7{,}65 \cdot 10^8$	$4{,}74 \cdot 10^8$	$1{,}48 \cdot 10^8$	$5{,}34 \cdot 10^5$

Tabelle 8.9: Ergebnisse für die Konfiguration BnBLAP

Instanzmenge	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
50_10_BigSetup	20	0	186,4	$2{,}13 \cdot 10^7$	$1{,}35 \cdot 10^8$	0	$3{,}19 \cdot 10^7$	$2{,}06 \cdot 10^6$
50_10_BigSetup_SmallDeadline	20	0	218,3	$3{,}42 \cdot 10^7$	$1{,}93 \cdot 10^8$	0	$7{,}83 \cdot 10^7$	$1{,}07 \cdot 10^7$
50_10_SmallSetup	20	0	214,5	$2{,}17 \cdot 10^7$	$2{,}02 \cdot 10^8$	0	$8{,}84 \cdot 10^7$	$1{,}15 \cdot 10^7$
50_10_SmallSetup_SmallDeadline	20	0	238,1	$3{,}53 \cdot 10^7$	$3{,}14 \cdot 10^8$	0	$2{,}09 \cdot 10^8$	8.655,65
50_15_BigSetup	20	0	209,6	$1{,}9 \cdot 10^7$	$1{,}36 \cdot 10^8$	0	$3{,}31 \cdot 10^7$	$2{,}06 \cdot 10^6$
50_15_BigSetup_SmallDeadline	20	3	215,4	$3{,}15 \cdot 10^7$	$1{,}96 \cdot 10^8$	0	$9{,}17 \cdot 10^7$	$8{,}55 \cdot 10^6$
50_15_SmallSetup	20	0	226,2	$1{,}35 \cdot 10^7$	$1{,}55 \cdot 10^8$	0	$5{,}21 \cdot 10^7$	$2{,}52 \cdot 10^6$
50_15_SmallSetup_SmallDeadline	20	0	311,6	$3{,}88 \cdot 10^7$	$4{,}55 \cdot 10^8$	0	$3{,}46 \cdot 10^8$	14.898,1

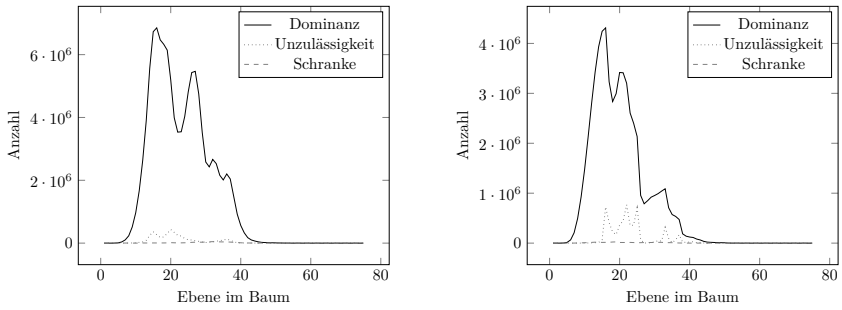
Tabelle 8.10: Ergebnisse für die Konfiguration BnBNoDominance

Instanzmenge	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
50_10_BigSetup	20	0	564,2	$6{,}52 \cdot 10^7$	$4{,}15 \cdot 10^8$	0	$1{,}03 \cdot 10^8$	$9{,}47 \cdot 10^6$
50_10_BigSetup_SmallDeadline	20	1	612,0	$9{,}5 \cdot 10^7$	$5{,}33 \cdot 10^8$	0	$2{,}09 \cdot 10^8$	$2{,}31 \cdot 10^7$
50_10_SmallSetup	20	1	734,7	$9{,}27 \cdot 10^7$	$8{,}66 \cdot 10^8$	0	$5{,}04 \cdot 10^8$	$5{,}53 \cdot 10^7$
50_10_SmallSetup_SmallDeadline	20	0	757,7	$1{,}21 \cdot 10^8$	$1{,}07 \cdot 10^9$	0	$7{,}5 \cdot 10^8$	$1{,}29 \cdot 10^5$
50_15_BigSetup	20	0	648,9	$6{,}21 \cdot 10^7$	$4{,}45 \cdot 10^8$	0	$1{,}29 \cdot 10^8$	$1{,}35 \cdot 10^7$
50_15_BigSetup_SmallDeadline	20	5	615,5	$9{,}28 \cdot 10^7$	$5{,}58 \cdot 10^8$	0	$2{,}57 \cdot 10^8$	$3{,}91 \cdot 10^7$
50_15_SmallSetup	20	0	721,6	$5{,}84 \cdot 10^7$	$6{,}74 \cdot 10^8$	0	$3{,}61 \cdot 10^8$	$6{,}65 \cdot 10^6$
50_15_SmallSetup_SmallDeadline	20	0	990,9	$1{,}34 \cdot 10^8$	$1{,}54 \cdot 10^9$	0	$1{,}21 \cdot 10^9$	25.434,95

Tabelle 8.11: Ergebnisse für die Konfiguration BnBNoDominance3xNodes

Instanzmenge	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound	∅ LAP ↓	∅ LAP ↑?	∅ LAP ↑!
50_10_BigSetup	20	20	35,8	$3{,}38 \cdot 10^5$	$2{,}26 \cdot 10^6$	$1{,}72 \cdot 10^6$	$1{,}11 \cdot 10^5$	29.204,2	291,55	$4{,}37 \cdot 10^5$	$3{,}43 \cdot 10^5$
50_10_BigSetup_SmallDeadline	20	20	2,9	34.973,1	$2{,}07 \cdot 10^5$	$1{,}39 \cdot 10^5$	22.810,85	2.600,55	98,75	44.964,9	35.658,3
50_10_SmallSetup	20	20	1.497,0	$7{,}55 \cdot 10^6$	$7{,}01 \cdot 10^7$	$5{,}23 \cdot 10^7$	$4{,}93 \cdot 10^6$	$8{,}09 \cdot 10^5$	27,6	$1{,}29 \cdot 10^7$	$2{,}9 \cdot 10^5$
50_10_SmallSetup_SmallDeadline	20	20	26,6	$2{,}23 \cdot 10^5$	$1{,}95 \cdot 10^6$	$1{,}26 \cdot 10^6$	$3{,}8 \cdot 10^5$	15.945,1	804,15	$3{,}17 \cdot 10^5$	22.736,75
50_15_BigSetup	20	20	74,4	$6{,}15 \cdot 10^5$	$4{,}89 \cdot 10^6$	$3{,}68 \cdot 10^6$	$3{,}48 \cdot 10^5$	58.486,1	4.784,7	$8{,}57 \cdot 10^5$	$4{,}43 \cdot 10^5$
50_15_BigSetup_SmallDeadline	20	20	3,9	44.618,5	$2{,}84 \cdot 10^5$	$1{,}8 \cdot 10^5$	35.857,05	6.962,7	5.969,9	62.325,75	19.686,95
50_15_SmallSetup	20	14	10.226,9	$3{,}94 \cdot 10^7$	$4{,}92 \cdot 10^8$	$3{,}62 \cdot 10^8$	$4{,}24 \cdot 10^7$	$7{,}69 \cdot 10^6$	4.765,1	$8{,}71 \cdot 10^7$	$2{,}75 \cdot 10^6$
50_15_SmallSetup_SmallDeadline	20	20	84,3	$6{,}73 \cdot 10^5$	$7{,}59 \cdot 10^6$	$5{,}08 \cdot 10^6$	$1{,}51 \cdot 10^6$	34.744,65	18.044,8	$9{,}8 \cdot 10^5$	6.252,2
75_10_BigSetup	20	20	2.080,5	$8{,}18 \cdot 10^6$	$6{,}23 \cdot 10^7$	$5 \cdot 10^7$	$2{,}03 \cdot 10^6$	$1{,}52 \cdot 10^5$	1.603,85	$1{,}02 \cdot 10^7$	$9{,}8 \cdot 10^6$
75_10_BigSetup_SmallDeadline	20	20	108,3	$5{,}71 \cdot 10^5$	$3{,}81 \cdot 10^6$	$2{,}85 \cdot 10^6$	$2{,}69 \cdot 10^5$	7.613,3	378,45	$6{,}92 \cdot 10^5$	$5{,}98 \cdot 10^5$
75_10_SmallSetup	20	1	47.337,4	$6{,}44 \cdot 10^7$	$6{,}2 \cdot 10^8$	$4{,}31 \cdot 10^8$	$3{,}26 \cdot 10^7$	$1{,}89 \cdot 10^6$	0,3	$1{,}57 \cdot 10^8$	1.587,35
75_10_SmallSetup_SmallDeadline	20	20	2.590,8	$1{,}03 \cdot 10^7$	$9{,}61 \cdot 10^7$	$6{,}91 \cdot 10^7$	$1{,}2 \cdot 10^7$	74.890,6	2.043	$1{,}5 \cdot 10^7$	15.335,05
75_15_BigSetup	20	18	13.732,3	$4{,}23 \cdot 10^7$	$3{,}88 \cdot 10^8$	$3{,}04 \cdot 10^8$	$2{,}11 \cdot 10^7$	$9{,}67 \cdot 10^5$	25.516,45	$6{,}28 \cdot 10^7$	$5{,}19 \cdot 10^7$
75_15_BigSetup_SmallDeadline	20	20	373,2	$2{,}04 \cdot 10^6$	$1{,}63 \cdot 10^7$	$1{,}24 \cdot 10^7$	$1{,}29 \cdot 10^6$	23.471,45	963,8	$2{,}56 \cdot 10^6$	$1{,}73 \cdot 10^6$
75_15_SmallSetup	20	0	50.687,5	$4{,}01 \cdot 10^7$	$5{,}34 \cdot 10^8$	$3{,}66 \cdot 10^8$	$2{,}65 \cdot 10^7$	$7{,}19 \cdot 10^5$	0	$1{,}42 \cdot 10^8$	0,15
75_15_SmallSetup_SmallDeadline	20	8	20.336,6	$5{,}37 \cdot 10^7$	$6{,}87 \cdot 10^8$	$4{,}38 \cdot 10^8$	$1{,}24 \cdot 10^8$	$1{,}49 \cdot 10^6$	2.476,75	$1{,}26 \cdot 10^8$	$8{,}88 \cdot 10^5$

Tabelle 8.12: Ergebnisse für die Konfiguration BnBSimpleBoundsAndLAP



(a) 75_10_BigSetup Instanz (b) 75_10_SmallSetup Instanz

Abbildung 8.2: Gründe für die Elimination eines Knotens

8.2.5.1 Einfluss der Dominanzregeln und Zulässigkeitstests

Die Dominanzregel aus Abschnitt 6.4.1 hat einen großen Einfluss auf die Menge der lösbaren Instanzen, wie die Ergebnisse in Tabelle 8.10 zeigen. Da die Dominanzregel vollständig ausgeschaltet wurde, konnten nur 3 der 160 Instanzen mit je fünfzig Aufträgen optimal gelöst werden, wobei diese Fälle alle im „einfachsten“ Testset aufgetreten sind.

Nutzen wir den durch die Entfernung der Dominanztabelle frei gewordenen Speicherplatz, indem wir die Anzahl der im Branch-and-Bound-Baum gleichzeitig verfügbaren Knoten verdreifachen, so kommen wir zu den Ergebnissen aus Tabelle 8.11. Hier sind nun mehr Instanzen optimal gelöst worden, die Unterschiede beider Testläufe sind in Tabelle 8.14 ausgearbeitet. Die Spalten $-/=/+$ Opt einer solchen Vergleichstabelle geben an, wie viele Instanzen von der zweitgenannten Konfiguration weniger, ebenso oder mehr pro Testmenge optimal gelöst worden sind. Ist eine Instanz von beiden Konfigurationen nicht optimal gelöst worden, wird hier der Faktor der Optimaliätslücke in Spalte Gapfaktor bestimmt. Ein Wert < 1 gibt an, dass die Optimalitätslücke durch die zweite Konfiguration im Durchschnitt verkleinert wurde, ein Wert > 1 steht für einen Anstieg der Optimalitätslücke gegenüber der ersten Konfiguration. Über wie viele Werte der Durchschnitt gebildet wurde, wird in Spalte Gapvergleiche angegeben. Auf ähnliche Weise sind die Werte in der Spalte Laufzeitfaktor zu interpretieren, der die verschiedenen Laufzeiten der Konfigurationen vergleicht.

Der Einfluss der Dominanztabelle tritt besonders im Vergleich der Konfigurationen BnBNoDominance und BnBChain zu Tage, die sich nur in der Anwendung der Dominanzkriterien unterscheiden, siehe hierzu Tabelle 8.13.

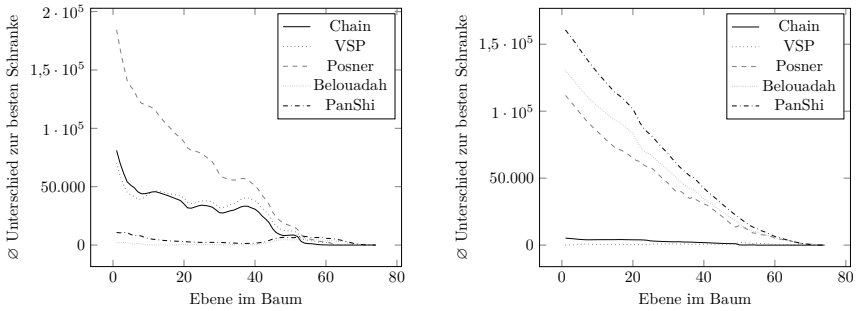
In Abbildung 8.2 wird für zwei Instanzen der Einfluss der Dominanzregeln und der Zulässigkeitstests im Vergleich mit den unteren Schranken dargestellt. Der Einfluss ist dabei über die Ebenen des Branch-and-Bound-Baumes aufgeschlüsselt. Die Zulässigkeitstests schlagen hier vor allem in den mittleren Ebenen an. Während die untere Schranke äußerst selten zu einer Entfernung eines Knotens aus dem Baum beiträgt, ist das Dominanzkriterium entscheidend für den Erfolg des Verfahrens. Damit wird der Branch-and-Bound-Ansatz nahezu zu einem Ansatz der Dynamischen Programmierung, dessen Enumerationsreihenfolge durch untere Schranken gesteuert wird.

Instanzmenge	− Opt	// Opt	+ Opt	∅ Gapfaktor	Gapvergleiche	∅ Laufzeitfaktor
50_10_SmallSetup	0	0	20	0	0	0,38
50_10_BigSetup	0	0	20	0	0	$1{,}35 \cdot 10^{-2}$
50_15_SmallSetup	0	0	11	0,28	9	2,79
50_15_BigSetup	0	0	20	0	0	$2{,}92 \cdot 10^{-2}$
50_10_BigSetup_SmallDeadline	0	0	20	0	0	$8{,}85 \cdot 10^{-4}$
50_15_SmallSetup_SmallDeadline	0	0	20	0	0	$2{,}32 \cdot 10^{-2}$
50_10_SmallSetup_SmallDeadline	0	0	20	0	0	$6{,}6 \cdot 10^{-3}$
50_15_BigSetup_SmallDeadline	0	3	17	0	0	$1{,}18 \cdot 10^{-3}$

Tabelle 8.13: Vergleich von BnBNoDominance und BnBChain

Instanzmenge	− Opt	// Opt	+ Opt	∅ Gapfaktor	Gapvergleiche	∅ Laufzeitfaktor
50_10_SmallSetup	0	0	1	0,94	19	3,49
50_10_BigSetup	0	0	0	0,92	20	3,03
50_15_SmallSetup	0	0	0	0,97	20	3,2
50_15_BigSetup	0	0	0	0,94	20	3,09
50_10_BigSetup_SmallDeadline	0	0	1	0,78	19	2,92
50_15_SmallSetup_SmallDeadline	0	0	0	0,94	20	3,22
50_10_SmallSetup_SmallDeadline	0	0	0	0,92	20	3,22
50_15_BigSetup_SmallDeadline	0	3	2	0,82	15	2,82

Tabelle 8.14: Vergleich von BnBNoDominance und BnBNoDominance3xNodes



(a) 75_10_BigSetup Instanz (b) 75_10_SmallSetup Instanz

Abbildung 8.3: Vergleich der Schrankenqualität

8.2.5.2 Vergleich der Schranken

In Abbildung 8.3 sind für zwei exemplarische Instanzen die durchschnittlichen Abweichungen der verschiedenen Schranken von der besten Schranke für jeden Knoten des Branch-and-Bound-Baumes berechnet worden. Eine Schranke, die relativ niedrige Werte in der Abweichung erzielt, ist also nicht so weit entfernt von der besten Schranke wie eine Schranke, die höhere Werte in der durchschnittlichen Abweichung erzielt, und damit eine bessere Schranke für diese Ebene des Branch-and-Bound-Baumes. Insgesamt nimmt der Unterschied zwischen den Schranken mit ansteigender Tiefe des Baumes ab. Je tiefer die Ebene im Baum ist, desto länger wird der partielle Schedule und damit die allen Schranken gemeinsam zur Verfügung stehende Information größer. Dabei wird deutlich, dass sich je nach Ebene die Bedeutung einer Schranke für das Verfahren ändern kann. So ist in Abbildung 8.3a die VSP-Schranke in den mittleren Ebenen des Baumes besser als die Schranke auf Basis von Ketten, bevor Sie in den tiefsten Ebenen letztendlich die beste Schranke stellt. Dass je nach Instanz auch verschiedene Schranken gut geeignet sind, zeigt dann der Vergleich zwischen den Abbildungen 8.3a und 8.3b. Während im ersten Fall die Schranken unter Einbeziehung der Freigabezeitpunkte (Belouadah und PanShi) über weite Teile die besten Schranken liefern, sind es im zweiten Fall die Schranken auf Basis der Vorrangbeziehungen.

Den Einfluss der Schranke auf Basis des Linearen Zuordnungsproblems wollen wir nun durch Betrachtung von drei Konfigurationen des Branch-and-Bound-Verfahrens verdeutlichen. In Tabelle 8.8 sind nur die einfachen Schranken (ohne das LAP) aktiv, während in Tabelle 8.9 nur die Schranke auf Basis des linearen Zuordnugnsproblems (ohne die einfachen Schranken) aktiv war. In Tabelle 8.12 sind nun beide Schrankentypen gleichzeitig aktiviert. Die Veränderungen im Vergleich zu den Konfigurationen mit weniger Schranken sind nun in den beiden Vergleichstabellen 8.15 und 8.16 dokumentiert. Hier ist bemerkenswert, dass sich der hohe Aufwand für das lineare Zuordnungsproblem im Sinne einer Verbesserung der Optimalitätslücke anscheinend nicht auszahlt. In Tabelle 8.12 sind dazu in den letzten drei Spalten auch noch die durchschnittlichen Anzahlen von Schrankenberechnungen in den Knoten des Branch-and-Bound-Verfahrens dokumentiert:

LAP ↓ Die Lösung des LAP in den Knoten war nicht notwendig, da diejenige

Lösung des LAP, die alle Vorrangbeziehungen einhält (siehe Abschnitt 6.6.6.4), eine niedrigere Schranke impliziert als die anderen Schranken. Wenn diese zulässige Lösung des LAP schon keine bessere Schranke liefert, kann auch die optimale Lösung des LAP keine bessere Schranke liefern.

LAP ↑? Die zulässige Lösung des LAP mit Vorrangbeziehungen hat einen Zielfunktionswert, der höher ist als die aktuelle Schranke. Damit könnte es zu einer Verbesserung der Schranke bei Lösung des LAP kommen. Der Wert in dieser Spalte gibt damit dann auch die durchschnittliche Anzahl der durchgeführten LAP-Berechnungen mit Hilfe der ungarischen Methode an.

LAP ↑! Die optimale Lösung des LAP liefert tatsächlich eine bessere untere Schranke

Auffällig ist hier, dass selten eine Verschlechterung der LAP-Schranke gegenüber den anderen Schranken vorausgesagt wird, es aber dann gewaltige Unterschiede gibt, ob es tatsächlich zu einer Verbesserung kommt. In den Instanzen mit großen Rüstzeiten ist der Anteil an Verbesserungen der Schranke sehr hoch, während bei den Instanzen mit niedrigen Rüstzeiten die LAP-Schranke fast nie zu einer Verbesserung beiträgt, den extremen Fall bilden hier die Instanzen aus 75_15_BigSetup.

8.2.6 Der Branch-and-Price-and-Cut-Ansatz

Der in Kapitel 7 beschriebene Branch-and-Price-and-Cut-Ansatz weist aufgrund seiner Komplexität nun eine Vielzahl von zu untersuchenden Aspekten auf. Hierbei wollen wir uns nun zunächst der Definition der Inklusionsmuster widmen.

8.2.6.1 Die Zerlegung in Inklusionsmuster

Die Zerlegung eines Schedules in Inklusionsmuster liefert für jede Kette der Partitionierung des Vorranggraphen ein Inklusionsmuster. Die Summe der Kosten der entstehenden Inklusionsmuster bildet eine untere Schranke für den Zielfunktionswert des zerlegten Schedules. Hier stellt sich zunächst die Frage, wie gut die erreichte Abschätzung ist, beziehungsweise wie viel durch die Zerlegung vom ursprünglichen Zielfunktionswert verloren geht.

Ein zweiter Aspekt ist die Vielseitigkeit der Inklusionsmuster. Ein Inklusionsmuster ist nicht nur Teil eines Schedules, sondern kann auch mit weiteren Inklusionsmustern, die nicht aus dem aktuellen Schedule hervorgehen, weitere Schedules darstellen.

Im folgenden Experiment zerlegen wir nun 5000 Schedules einer Instanz, die unter anderem während einer Tabusuche mit 10000 Iterationen als aktuelle Lösung übernommen worden sind. Diese sind nach aufsteigendem Zielfunktionswert sortiert. Da die Instanz eine Dilworth-Partitionierung in Ketten der Kardinalität 15 besitzt, können wir maximal 75000 Spalten erhalten. In Abbildung 8.4 wurden nun die Lösungen der Reihe nach in Inklusionsmuster zerlegt, die farbige Kodierung gibt an, wie viele der erzeugten 15 Inklusionsmuster pro Schedule tatsächlich noch unbekannt waren. Auf der y-Achse ist dann der prozentuale Verlust der in den Inklusionsmustern abgeschätzten Kosten gegenüber dem

Instanzmenge	− Opt	= Opt	+ Opt	∅ Gapfaktor	Gapvergleiche	∅ Laufzeitfaktor
50_10_SmallSetup	0	20	0	0	0	2,84
50_10_BigSetup	0	20	0	0	0	2,24
75_15_BigSetup	0	18	0	0,86	2	2,72
50_15_SmallSetup	0	14	0	0,99	6	2,68
50_15_BigSetup	0	20	0	0	0	2,27
75_10_BigSetup	0	20	0	0	0	2,86
75_10_SmallSetup	0	1	0	1	19	4,16
75_15_SmallSetup	0	0	0	1	20	4,31
50_10_BigSetup_SmallDeadline	0	20	0	0	0	2
50_15_SmallSetup_SmallDeadline	0	20	0	0	0	2,23
50_10_SmallSetup_SmallDeadline	0	20	0	0	0	2,38
50_15_BigSetup_SmallDeadline	0	20	0	0	0	1,97
75_10_BigSetup_SmallDeadline	0	20	0	0	0	2,52
75_10_SmallSetup_SmallDeadline	0	20	0	0	0	2,98
75_15_BigSetup_SmallDeadline	0	20	0	0	0	2,33
75_15_SmallSetup_SmallDeadline	0	8	0	1	12	2,85

Tabelle 8.15: Vergleich von BnBSimpleBounds und BnBSimpleBoundsAndLAP

Instanzmenge	− Opt	= Opt	+ Opt	∅ Gapfaktor	Gapvergleiche	∅ Laufzeitfaktor
50_10_SmallSetup	0	20	0	0	0	0,5
50_10_BigSetup	0	20	0	0	0	1,34
75_15_BigSetup	0	18	0	1	2	1,19
50_15_SmallSetup	0	7	7	0,35	6	0,88
50_15_BigSetup	0	20	0	0	0	1,18
75_10_BigSetup	0	20	0	0	0	1,22
75_10_SmallSetup	0	0	1	0,22	19	1,09
75_15_SmallSetup	0	0	0	0,42	20	1,11
50_10_BigSetup_SmallDeadline	0	20	0	0	0	1,4
50_15_SmallSetup_SmallDeadline	0	20	0	0	0	1,22
50_10_SmallSetup_SmallDeadline	0	20	0	0	0	1,29
50_15_BigSetup_SmallDeadline	0	20	0	0	0	1,18
75_10_BigSetup_SmallDeadline	0	20	0	0	0	1,28
75_10_SmallSetup_SmallDeadline	0	20	0	0	0	1,1
75_15_BigSetup_SmallDeadline	0	20	0	0	0	1,27
75_15_SmallSetup_SmallDeadline	0	2	6	0,59	12	1,16

Tabelle 8.16: Vergleich von BnBLAP und BnBSimpleBoundsAndLAP

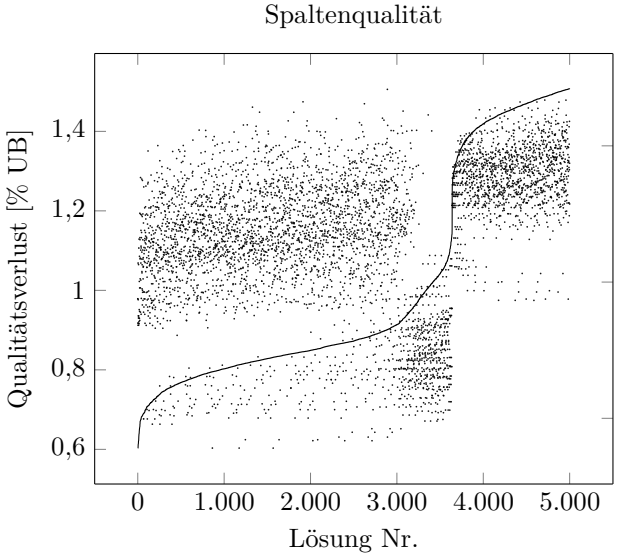


Abbildung 8.4: Qualitätsuntersuchung der Zerlegung in Inklusionsmuster

ursprünglichen Zielfunktionswert abgetragen. Tatsächlich ergeben sich dann insgesamt nur 1370 verschiedene Spalten, 4076 Schedules lassen sich komplett aus bei deren Zerlegung bekannten Inklusionsmustern zusammensetzen, und steuern deswegen keine neuen Spalten bei. Ein Inklusionsmuster kann somit insbesondere in großen Instanzen in vielen verschiedenen Schedules enthalten sein. Die in Abbildung 8.4 zusätzlich eingezeichnete Kurve gibt den absoluten Zielfunktionswert der zerlegten Schedules an, auffällig ist hier der starke Anstieg in der zweiten Hälfte der Lösungsmenge. Vor allem die vom Zielfunktionswert schlechteren Lösungen scheinen hier einen stärkeren Verlust in der Abschätzungsqualität bei der Zerlegung in Inklusionsmuster zu unterliegen.

8.2.6.2 Pricing-Verfahren

Für das Pricing-Problem werden in dieser Arbeit drei verschiedene Branch-and-Bound-Algorithmen vorgestellt, die hier nun anhand exemplarischer Szenarien evaluiert werden sollen. Um diese Algorithmen unter sonst gleichen Bedingungen zu testen, wurde eine Instanz des Schedulingproblems aus der Menge der hier untersuchten schwierigsten Instanzen ausgewählt. Im Wurzelknoten des Branch-and-Price-and-Cut-Ansatzes wurden dann in jeder Iteration des Spaltengenerierungsverfahrens alle drei Ansätze nacheinander auf die somit gleichen Pricing-Probleme angewendet, die Anzahl der Iterationen war auf 100 beschränkt. Für die Durchführung eines der drei Pricing-Verfahrens war maximal eine Stunde vorgesehen, für alle 100 Iterationen mit allen drei Verfahren waren maximal 100 Stunden erlaubt. Ein Pricing-Verfahren konnte vorzeitig abbrechen, wenn mindestens 100 neue Spalten gefunden worden sind. Da in den Pricing-Verfahren das Erreichen des Zeitlimits nur dann überprüft wird, wenn von der koordinierenden Komponente des Pricing-Verfahrens weitere Rechenkapazität angefordert wird, kann es hier zu Überschreitungen des Zeitlimits kommen, wenn die vergebene Kapazität relativ hoch ist. Im Rahmen dieses Tests durfte ein Pricer für

eine Kette jeweils 10000 Branchings durchführen, bevor es zu einer erneuten Überprüfung des Zeitlimits kam.

Da auch die Qualität und Struktur der gefundenen Spalten das weitere Spaltengenerierungsverfahren erheblich beeinflusst, wurden hier vier Szenarien hinsichtlich der Übernahme von gefundenen Spalten überprüft.

Szenario 1 Nur die Spalten, die der Set-Jump-Pricer gefunden hat, werden als neue Spalten in das Masterproblem übernommen.

Szenario 2 Nur die Spalten, die der Set-Fill-Pricer gefunden hat, werden als neue Spalten in das Masterproblem übernommen.

Szenario 3 Nur die Spalten, die der Set-Synchronization-Pricer gefunden hat, werden als neue Spalten in das Masterproblem übernommen.

Szenario 4 Alle Spalten, die von mindestens einem der drei Verfahren gefunden worden sind, werden als neue Spalten in das Masterproblem übernommen.

Die jeweils erzielte untere Schranke in jeder Iteration für die betrachtete Instanz über die Laufzeit des Verfahrens wird in Abbildung 8.5 dargestellt. Diese untere Schranke wird dabei über die maximale untere Schranke der drei Pricing-Verfahren bestimmt. Abbildung 8.6 zeigt die Anzahl der neu in das Masterproblem übernommenen Spalten. Werden die neuen Spalten von allen Verfahren übernommen, gibt es hier pro Iteration mehr neue Spalten als wenn nur eines der drei Pricing-Verfahren die neuen Spalten beisteuern darf.

Die Schranken für die Summe der reduzierten Kosten über alle Teilprobleme pro Kette werden in Abbildung 8.7 für die verschiedenen Pricing-Verfahren angezeigt. Für jedes Verfahren sind zwei Kurven angegeben, die obere Kurve gibt die Summe der reduzierten Kosten über die besten gefundenen Spalten für jedes Teilproblem an, die untere Kurve die Summe der mindestens anfallenden reduzierten Kosten, über die dann auch die untere Schranke LB_{CG} mit Hilfe des optimalen Zielfunktionswertes des RLPM bestimmt werden kann (siehe Abschnitt 7.7.4.1). Hier ist auffällig, dass der Set-Synchronization-Pricer in allen Szenarien diese Werte am besten abschätzt, somit sowohl gute Spalten als auch gute untere Schranken für die Teilprobleme findet. Somit sind die unteren Schranken in Abbildung 8.5 auch alle durch den Set-Synchronization-Pricer erzeugt worden.

Vergleichen wir den Set-Jump- und den Set-Fill-Pricer, so ist anhand der Abbildungen zu sehen, dass sie relativ ähnlich sind, was die Qualität der gefunden Spalten und die Qualität der unteren Schranken betrifft. Auffällig ist auch die Komplementarität der Verfahren, werden die Spalten in Szenario 1 vom Set-Jump-Pricer übernommen, so findet der Set-Jump-Pricer dann irgendwann nur noch wenige Spalten, während das Set-Fill-Verfahren weitere Spalten finden würde (siehe Abbildung 8.8a). Andersherum verhält es sich in Szenario 2, in dem der Set-Jump-Pricer weitere Spalten zu den vom Set-Fill-Pricer übernommenen Spalten vorschlagen könnte (Abbildung 8.8b). In beiden Fällen liefert allerdings der Set-Synchronization-Pricer häufig viele Spalten, das Verfahren bricht dann relativ früh ab, erzielt aber dennoch gute untere Schranken. Für den Set-Jump- und Set-Fill-Pricer ist die Anzahl von verfügbaren Knoten (in allen Szenarien 5 Millionen pro Kette) für das Pricing-Verfahren das entscheidende Kriterium, das zum Abbruch führt, das Zeitlimit von einer Stunde wird von diesen Verfahren nie erreicht. Dagegen wird das Set-Synchronization-

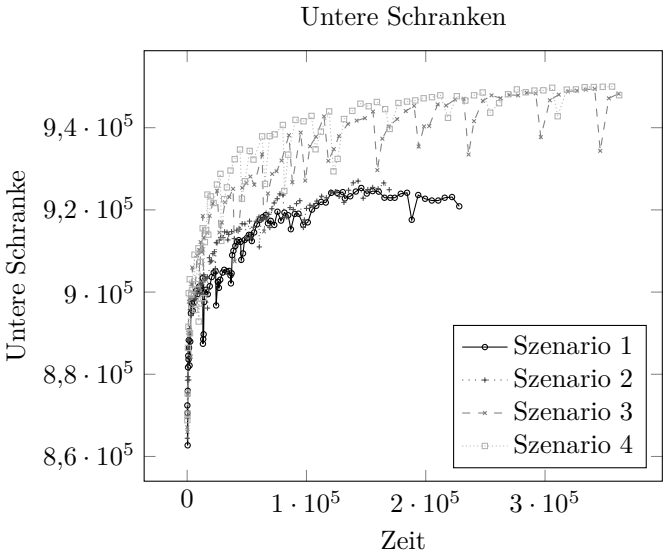


Abbildung 8.5: Vergleich der Szenarien - Untere Schranken

Verfahren in den Szenarien 3 und 4 häufig wegen Erreichens beziehungsweise Überschreitens des Zeitlimits abgebrochen (siehe Abbildungen 8.8c und 8.8d). Je mehr Iterationen abgelaufen sind, desto schwieriger zu bewältigen wird auch das Pricing-Problem für die Verfahren, obwohl sich nicht der zulässige Bereich, sondern nur die Zielfunktionskoeffizienten des Pricingproblems ändern.

In allen Szenarien ist deutlich der Einfluss des Einfügens von Vorrangschnitten in das Masterproblem auf die Qualität der unteren Schranke zu erkennen. Wird ein Schnitt eingefügt, so erlaubt die aktuelle duale Lösung des Masterproblems zunächst Spalten mit sehr niedrigen reduzierten Kosten, und es braucht einige Iterationen, bis der Schnitt vernünftig eingepreist ist, was dann allerdings zum weiteren Anstieg der globalen unteren Schranke führt. Je mehr Iterationen erreicht und Schnitte eingefügt werden, desto länger brauchen die Pricing-Verfahren.

Dass der Set-Synchronization-Pricer viel Zeit pro Knoten beansprucht, verdeutlichen die Punktmengen aus Abbildung 8.9. Hier ist auch zu erkennen, dass die zeitliche Bandbreite zum Erreichen des Knotenlimits im Set-Jump- und Set-Fill-Verfahren recht hoch ausfällt.

Zwischen Szenario 3 und Szenario 4 bestehen hinsichtlich der ermittelten Daten nur wenige Unterschiede. Aufgrund der höheren Spaltenvielfalt durch die verschiedenen Pricing-Verfahren scheint der Verlauf der Schranken in Szenario 4 etwas stabiler zu sein. Das erreichte Level bei der unteren Schranke ist aber vergleichbar, somit können wir hier das Set-Synchronization-Verfahren vorbehaltlich einer Bestätigung in weiteren Evaluationen als überlegenes Pricing-Verfahren bezeichnen. Ein gravierender Nachteil ist jedoch der hohe zeitliche Aufwand, der mit der Berechnung der favorisierten Mengen von Vorgängern verbunden ist, der sich aber in Qualität von oberen und unteren Schranken sowie der zielgerichteten Verzweigung auch auszahlt.

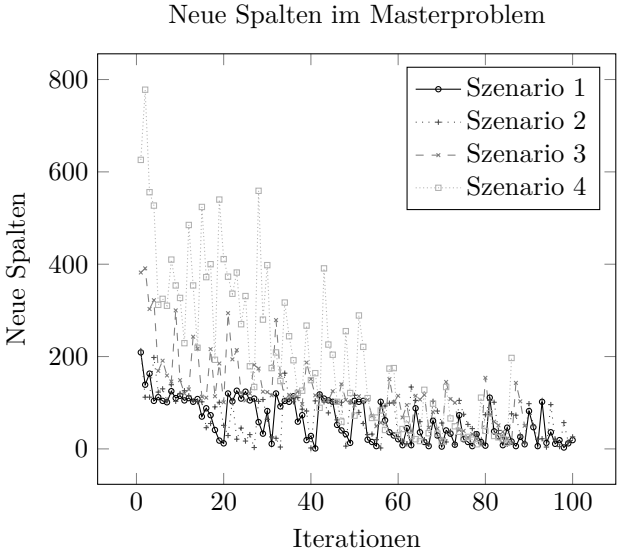
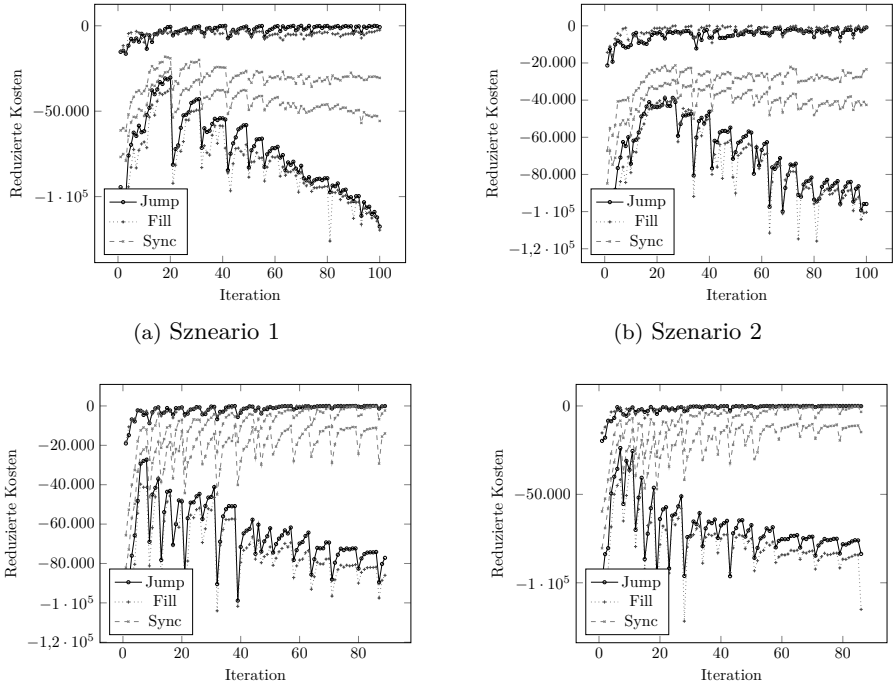


Abbildung 8.6: Vergleich der Szenarien - Neue Spalten



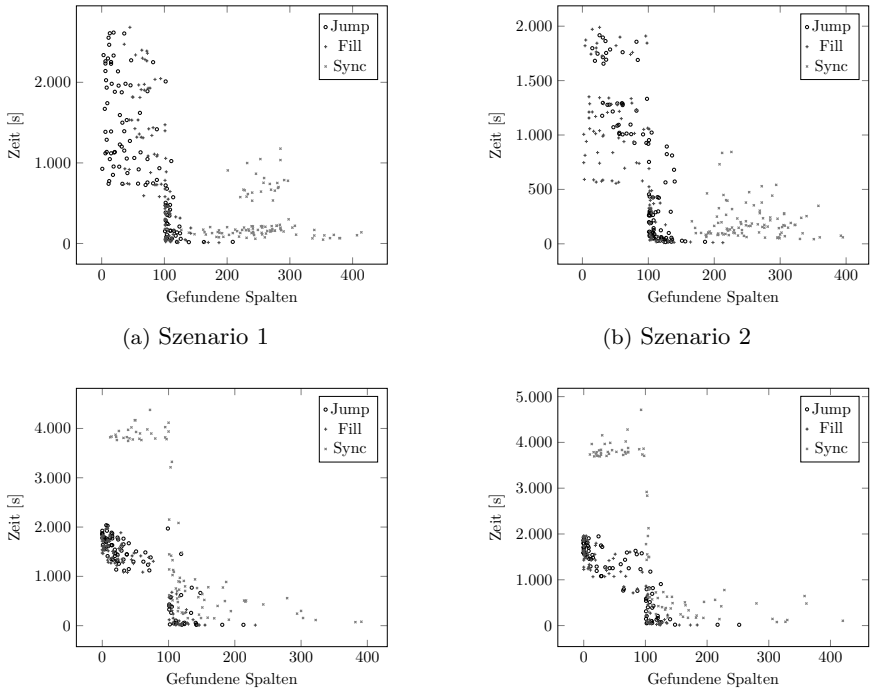
(a) Szneario 1

(b) Szenario 2

(c) Szenario 3

(d) Szenario 4

Abbildung 8.7: Szenarien - Schranken



(a) Szenario 1

(b) Szenario 2

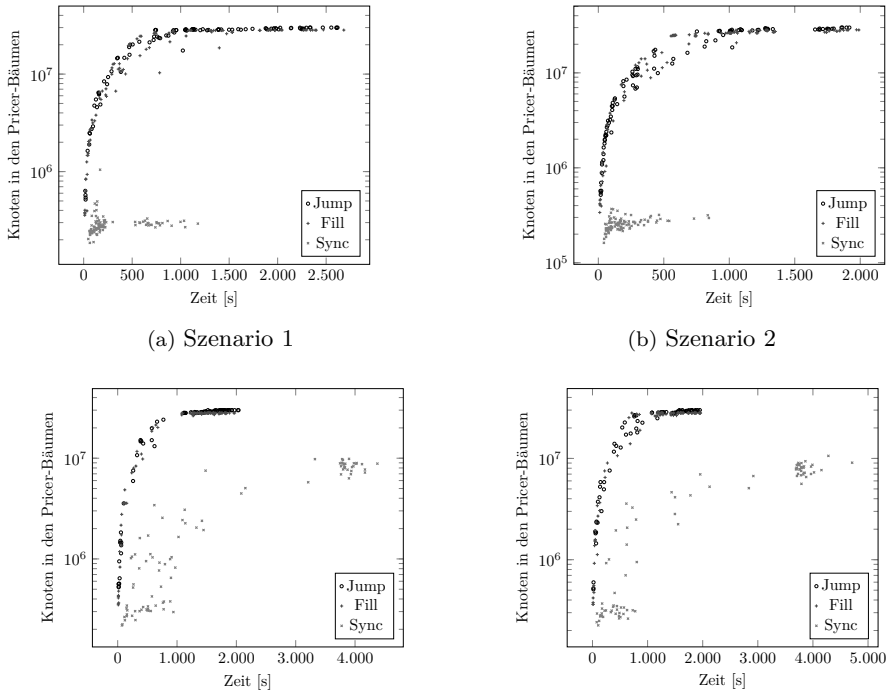
(c) Szenario 3

(d) Szenario 4

Abbildung 8.8: Szenarien - Zeit und Spalten

8.2.6.3 Evaluation des Gesamtverfahrens

Für die Testsets ohne verschärfte Fälligkeiten (damit 160 Instanzen) wurde eine ausgewählte Konfiguration des Branch-and-Price-and-Cut-Verfahrens getestet. Dabei durften maximal 100 Knoten im Branch-and-Price-Baum evaluiert werden. Die Lösungszeit für einen Knoten des Branch-and-Price-Baums war auf 10 Minuten beschränkt. Für eine Iteration des Pricing-Verfahrens waren maximal 3 Minuten erlaubt, die Anzahl der Iterationen war auf 20 beschränkt. Als Pricing-Verfahren wurde der Set-Synchronization-Pricer verwendet. Die Ergebnisse sind in Tabelle 8.17 zu finden, die neben der Anzahl der insgesamt betrachteten Instanzen pro Testmenge auch die Anzahl der optimal gelösten Instanzen ausgibt. Zusätzliche Informationen sind neben der Laufzeit auch die tatsächlich verwendeten Knoten im Branch-and-Price-Baum, sowie die Anzahl der durchschnittlich im Verlauf des gesamten Verfahrens erzeugten Vorrangschnitte. Auffällig ist hier, dass nur in den auch für das Branch-and-Bound-Verfahren einfachen Instanzen eine optimale Lösung erzielt werden konnte. In je einer Instanz aus 75_10_SmallSetup und 75_15_SmallSetup hat sich auch durch die Verzweigungen keine bessere Schranke als die schon im Wurzelknoten des Branch-and-Price-and-Cut-Baumes berechnete Schranke ergeben. In sechs Instanzen dagegen konnte nach einigen Verzweigungen die Adaption des Branch-and-Bound-Verfahrens zur Generierung der initialen Spalten doch noch den Optimalitätsbeweis erbringen. Verzweigt wurde hierbei in fast allen Fällen auf Basis der durch die Inklusionsmuster der Lösung des RLPM induzierten uneindeutigen Vorrangbeziehungen (siehe Abschnitt 7.6.2).



(a) Szenario 1

(b) Szenario 2

(c) Szenario 3

(d) Szenario 4

Abbildung 8.9: Szenarien - Zeit und Knoten

Instanzmenge	*Total*	*Optimal*	*∅ Laufzeit(s)*	*∅ Knoten*	*∅ Vorrangschnitte*
50_10_BigSetup	20	20	4,0	1	0
50_10_SmallSetup	20	5	7.084,9	76,9	90,25
50_15_BigSetup	20	20	40,8	1,3	0,5
50_15_SmallSetup	20	0	30.280,6	99	339,15
75_10_BigSetup	20	5	9.338,3	77,7	0
75_10_SmallSetup	20	0	46.274,8	99	1,7
75_15_BigSetup	20	0	31.049,4	99	23,15
75_15_SmallSetup	20	0	71.170,9	99	0,45

Tabelle 8.17: Ergebnisse für den Branch-and-Price-and-Cut-Ansatz

Instanzmenge	− Opt	= Opt	+ Opt	∅ Gapfaktor	Gapvergleiche	∅ Laufzeitfaktor
50_10_SmallSetup	15	5	0	0	0	12,5
50_10_BigSetup	0	20	0	0	0	0,33
75_15_BigSetup	18	0	0	1,66	2	9,43
50_15_SmallSetup	14	0	0	20,93	6	10,8
50_15_BigSetup	0	20	0	0	0	0,73
75_10_BigSetup	15	5	0	0	0	13,98
75_10_SmallSetup	1	0	0	3,9	19	4,67
75_15_SmallSetup	0	0	0	1,93	20	6,07

Tabelle 8.18: Vergleich von Branch-and-Bound mit Branch-and-Price

Um die erzielten Zeiten und Schranken besser einordnen zu können, vergleichen wir hier die Werte mit den Werten aus Tabelle 8.7 für den Branch-and-Bound-Ansatz mit Verwendung der einfachen Schranken (Konfiguration BnBSimpleBounds), und erhalten die Werte aus Tabelle 8.18. Auch hier wird deutlich, dass der Branch-and-Bound-Ansatz deutlich mehr Instanzen optimal lösen kann, und auch bessere Schranken bietet. Nur bei den relativ einfach zu lösenden Instanzen hat hier der Branch-and-Price-Ansatz einen scheinbaren Zeitvorteil, weil er auf die einfache Implementation des Branch-and-Bound-Verfahrens zur Erzeugung zurückgreift, das hier sehr schnell eine Lösung finden konnte.

8.2.7 Erzielte Schranken

Die Tabelle 8.19 zeigt nun für jede Testmenge von Instanzen und für einige der getesteten Konfigurationen auf, in wie vielen Fällen die beste (und damit höchste) bekannte untere Schranke beziehungsweise die beste (und damit niedrigste) bekannte obere Schranke für eine Instanz über alle Konfigurationen erreicht wurde. Hier sind zwei Resultate auffällig, die wir nun näher betrachten wollen: Für zwei Instanzen aus 75_15_BigSetup und dreizehn Instanzen aus 75_15_SmallSetup liefert das mit CPLEX gelöste Modell der gemischt ganzzahligen Programmierung die beste untere Schranke, obwohl Rechenzeit und Baumgröße limitiert waren. Anscheinend kann hier die Schranke aus der LP-Relaxation mehr Information über den optimalen Zielfunktionswert ableiten als das Branch-and-Bound-Verfahren, das hier anscheinend nicht gut genug die Folgen der Entscheidungen in einem partiellen Schedule für die noch offenen Aufträge abschätzt. Die zweite Beobachtung betrifft das Branch-and-Price-and-Cut-Verfahren, hier wurde für alle mit der Konfiguration BranchPrice100 getesteten Instanzen die beste bekannte obere Schranke durch das Verfahren generiert. Auch hier leistet der einfache Branch-and-Bound-Ansatz zur Generierung des initialen Spaltensets gute Arbeit, die auch durch die Verwendung des Knotenauswahlverfahrens der zyklischen Bestensuche (siehe Abschnitt 6.2.3) ermöglicht wird.

Instanzmenge	Model1h	BnBSimpleBounds	BnBLAP	BnBSimpleBoundsAndLAP	BnBChain	BnBVSP	BnBPosner	BnBBelouadah	BnBPanShi	BranchPrice100
50_10_BigSetup	0/4	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20
50_10_BigSetup_SmallDeadline	0/2	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	0/0
50_10_SmallSetup	0/0	20/20	20/20	20/20	20/20	20/20	20/20	19/19	17/17	5/20
50_10_SmallSetup_SmallDeadline	0/0	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	0/0
50_15_BigSetup	0/0	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20
50_15_BigSetup_SmallDeadline	0/0	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	0/0
50_15_SmallSetup	0/0	15/14	7/7	20/14	11/11	11/11	11/11	6/6	4/4	0/20
50_15_SmallSetup_SmallDeadline	0/0	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	0/0
75_10_BigSetup	0/0	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	5/20
75_10_BigSetup_SmallDeadline	0/0	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	0/0
75_10_SmallSetup	0/0	9/1	0/0	19/1	1/1	1/1	0/0	0/0	0/0	0/20
75_10_SmallSetup_SmallDeadline	0/0	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	0/0
75_15_BigSetup	2/0	18/18	18/18	18/18	15/15	15/15	14/14	18/18	17/17	0/20
75_15_BigSetup_SmallDeadline	0/0	20/20	20/20	20/20	20/20	20/20	20/20	20/20	20/20	0/0
75_15_SmallSetup	13/0	6/0	0/0	7/0	0/0	0/0	0/0	0/0	0/0	0/20
75_15_SmallSetup_SmallDeadline	0/0	8/20	2/14	20/20	3/15	3/15	4/16	2/14	1/13	0/0

Tabelle 8.19: Algorithmenvergleich: Beste obere/untere Schranken

8.3 Folgerungen und Forschungsideen

In der vorliegenden Arbeit wurde das aus der Praxis abgeleiteten Schedulingproblem

$$1|setups, rm, chains, \delta_j| \sum w_j C_j$$

betrachtet sowie verschiedene exakte Optimierungsansätze erarbeitet. In diesem Abschnitt wollen wir nun einige Folgerungen aus den in dieser Arbeit und bei der Evaluation der Verfahren gewonnenen Erkenntnissen ziehen.

8.3.1 Komplexität

Im Verlauf der Arbeit sind wir immer wieder auf Probleme gestoßen, deren Komplexitätsstatus noch offen ist. Das prominenteste Beispiel dafür ist auch die Frage, ob die Struktur der Rüstzeiten, wie sie hier betrachtet wurde, allein im Zusammenspiel mit den Zielfunktionen $C_{\max}, \sum(w_j)C_j$ ein $\mathcal{NP}$-schweres Problem erzeugt. Damit verwandt ist auch die Frage nach dem Komplexitätsstatus von $1|s_f| \sum(w_j)C_j$. Auch die Materialverfügbarkeit kann weiteren Komplexitätsbetrachtungen unterzogen werden.

Bei der Suche nach guten unteren Schranken für einen Branch-and-Bound-Ansatz sind wir auf die Erweiterung des linearen Zuordnungsproblems um Vorrangbeziehungen gestoßen, und haben festgestellt, dass für diesen Anwendungsfall die theoretische Untersuchung der Komplexität ebenfalls noch notwendig ist.

8.3.2 Halbordnungen

In Kapitel 3 haben wir uns unter anderem ausführlich mit der Reduktion von Vorranggraphen auf eine bestimmte Klasse von Vorrangbeziehungen befasst, in unserem Fall insbesondere die VSP-Halbordnungen. Ein zukünftiger Forschungsaspekt ist hier sicherlich auch die Entwicklung eines effizienten Reduktionsalgorithmus auf zweidimensionale Halbordnungen, sollte sich die Verwendung von zweidimensionalen Halbordnungen in unteren Schranken als vorteilhaft erweisen.

8.3.3 Der Branch-and-Bound-Ansatz

Die Rechenergebnisse zeigen, dass mehrere Schranken zusammen zu einer besseren Lösbarkeit von Problemen führen, wenn dabei der zusätzlich beanspruchten Rechenzeit keine Bedeutung beigemessen wird. Hier lohnt es sich, nun zu überprüfen, inwiefern eine dynamische Auswahl der passenden Schranken während der Durchführung eines Branch-and-Bound-Ansatzes automatisch getroffen werden kann. Eine etwaige Verschärfung der bestehenden Schranken kann durch ein weiteres Einbinden der Lagrange-Relaxation erreicht werden. Um die Knotenauswahlstrategie der Bestensuche zu unterstützen, sind gute obere Schranken notwendig. Hier sollte wie im Pricing-Verfahren eine Heuristik während des Branch-and-Bound-Ansatzes eingesetzt werden, was aber wiederum zu einem neuen Trade-Off zwischen zusätzlich notwendiger Rechenzeit und Verkleinerung des Branch-and-Bound-Baumes führt. Auch für die initiale obere

Schranke können noch weitere Verbesserungen des verwendeten heuristischen Ansatzes entwickelt werden. In der aktuellen Implementierung des Branch-and-Bound-Ansatzes wird die Materialverfügbarkeit nur in der Berechnung der Kostenkoeffizienten für das lineare Zuordnungsproblem und bei der Bestimmung von abgeleiteten Freigabezeitpunkten eingesetzt. An dieser Stelle könnten nun weitere untere Schranken auf Basis der Materiallieferungen berechnet werden. Auch die Ideen aus Kapitel 3 für die Entwicklung von Schranken für Scheduling-Probleme mit Vorrangbeziehungen verdienen eine weitere Betrachtung.

8.3.4 Der Branch-and-Price-and-Cut-Ansatz

Die Zerlegung eines Schedules in Inklusionsmuster liefert eine neue Möglichkeit, Ein-Maschinen-Schedulingprobleme mit Hilfe der Spaltengenerierung zu lösen. In unserem Fall ergaben sich mit dieser Definition einige Nachteile. Zunächst ist hier festzustellen, dass die Zerlegung in Inklusionsmuster für die gegebene Problemstellung nur eine untere Schranke für den realen Zielfunktionswert des Schedules liefert. Dabei erreicht der verloren gegangene Anteil durchaus auch den Wert der Optimalitätslücke bei Anwendung des Branch-and-Bound-Ansatzes aus Kapitel 6. Eine Perspektive für die Forschung ist hier die Suche nach Problemstellungen, für die die im Inklusionsmuster hinterlegten Informationen ausreichend sind, um den Beitrag der vom Inklusionsmuster vertretenen Aufträge zum Zielfunktionswert exakt zu ermitteln. Auch das Pricing-Problem, dessen Zielfunktionsauswertung durch die verwendete Abschätzungsmethode sehr zeitaufwendig ist, sollte einfacher zu evaluieren sein.

Positiv an dieser Stelle zu erwähnen ist die Schrankenqualität des Set-Synchronization-Pricers, was sowohl obere als auch untere Schranken betrifft. Hier wirkt sich die Ableitung einer Verzweigungsentscheidung aus der Analyse der bevorzugten Mengen gut auf die Leistungsfähigkeit des Algorithmus aus. Die Branch-and-Bound-Verfahren auf Basis eines festen Enumerationsschemas haben sich dagegen mit den ihnen anvertrauten Aufgaben, einerseits gute Spalten zu finden und andererseits die minimalen reduzierten Kosten abzuschätzen, schwer getan.

Sollte ein für die Zerlegung in Inklusionsmuster geeignetes Optimierungsproblem untersucht werden, sollte auch die Wiederverwendungsquote der berechneten bevorzugten Mengen im Set-Synchronization-Pricer ansteigen, und somit die Leistungsfähigkeit dieses Algorithmus erhöht werden. Eine zeitliche Unzulässigkeit dieser Mengen kann dann im Verlauf des Pricing-Verfahrens gar nicht mehr auftreten.

Somit bleibt es hier eine spannende Frage, ob auch mit Hilfe der Spaltengenerierung konkurrenzfähige Algorithmen für Ein-Maschinen-Scheduling-Probleme entwickelt werden können.

Anhang A

Weitere Tabellen

Die folgenden Tabellen zeigen weitere Ergebnisse der Evaluation des Branch-and-Bound-Verfahrens. Hier sind zunächst für jede Instanzmenge die Ergebnisse über alle Konfigurationen aufgeführt, bevor der Anhang mit Tabellen für die weiteren Konfigurationen des Branch-and-Bound-Verfahrens schließt.

Konfig.	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
BnBSimpleBounds	20	20	540,7	$7{,}58 \cdot 10^6$	$7{,}03 \cdot 10^7$	$5{,}24 \cdot 10^7$	$4{,}96 \cdot 10^6$	$8{,}11 \cdot 10^5$
BnBChain	20	20	72,5	$9{,}36 \cdot 10^6$	$8{,}61 \cdot 10^7$	$6{,}42 \cdot 10^7$	$6{,}59 \cdot 10^6$	$8{,}5 \cdot 10^5$
BnBLAP	20	20	3.173,9	$2{,}96 \cdot 10^7$	$2{,}7 \cdot 10^8$	$2{,}09 \cdot 10^8$	$1{,}61 \cdot 10^7$	$3{,}17 \cdot 10^6$
BnBVSP	20	20	85,5	$9{,}43 \cdot 10^6$	$8{,}72 \cdot 10^7$	$6{,}5 \cdot 10^7$	$6{,}6 \cdot 10^6$	$9{,}06 \cdot 10^5$
BnBBelouadah	20	19	805,5	$3{,}91 \cdot 10^7$	$3{,}53 \cdot 10^8$	$2{,}63 \cdot 10^8$	$2{,}61 \cdot 10^7$	$7{,}65 \cdot 10^6$
BnBLAPLagr20	20	19	8.833,6	$4{,}5 \cdot 10^7$	$4{,}12 \cdot 10^8$	$3{,}09 \cdot 10^8$	$2{,}67 \cdot 10^7$	$5{,}01 \cdot 10^6$
BnBPosner	20	20	234,8	$2{,}46 \cdot 10^7$	$2{,}21 \cdot 10^8$	$1{,}68 \cdot 10^8$	$1{,}39 \cdot 10^7$	$3{,}38 \cdot 10^6$
BnBPanShi	20	17	1.986,6	$4{,}76 \cdot 10^7$	$4{,}31 \cdot 10^8$	$3{,}2 \cdot 10^8$	$3{,}25 \cdot 10^7$	$5{,}45 \cdot 10^6$
BnBChainNoDom	20	0	214,5	$2{,}17 \cdot 10^7$	$2{,}02 \cdot 10^8$	0	$8{,}84 \cdot 10^7$	$1{,}15 \cdot 10^7$
BnBChainNoDom3xNodes	20	1	734,7	$9{,}27 \cdot 10^7$	$8{,}66 \cdot 10^8$	0	$5{,}04 \cdot 10^8$	$5{,}53 \cdot 10^7$
BnBSimpleBoundsAndLAP	20	20	1.497,0	$7{,}55 \cdot 10^6$	$7{,}01 \cdot 10^7$	$5{,}23 \cdot 10^7$	$4{,}93 \cdot 10^6$	$8{,}09 \cdot 10^5$

Tabelle A.1: Branch-and-Bound: 50_10_SmallSetup

Konfig.	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
BnBSimpleBounds	20	20	16,5	$3{,}6 \cdot 10^5$	$2{,}41 \cdot 10^6$	$1{,}8 \cdot 10^6$	$1{,}25 \cdot 10^5$	29.674,75
BnBChain	20	20	2,4	$4{,}79 \cdot 10^5$	$3{,}15 \cdot 10^6$	$2{,}29 \cdot 10^6$	$1{,}97 \cdot 10^5$	26.680,55
BnBLAP	20	20	26,4	$3{,}51 \cdot 10^5$	$2{,}36 \cdot 10^6$	$1{,}79 \cdot 10^6$	$1{,}14 \cdot 10^5$	28.712,05
BnBVSP	20	20	3,0	$4{,}76 \cdot 10^5$	$3{,}12 \cdot 10^6$	$2{,}28 \cdot 10^6$	$1{,}93 \cdot 10^5$	27.100,75
BnBBelouadah	20	20	8,1	$3{,}77 \cdot 10^5$	$2{,}53 \cdot 10^6$	$1{,}9 \cdot 10^6$	$1{,}29 \cdot 10^5$	28.008,25
BnBLAPLagr20	20	20	63,6	$4{,}33 \cdot 10^5$	$2{,}9 \cdot 10^6$	$2{,}2 \cdot 10^6$	$1{,}47 \cdot 10^5$	28.400,7
BnBPosner	20	20	3,1	$5{,}29 \cdot 10^5$	$3{,}48 \cdot 10^6$	$2{,}57 \cdot 10^6$	$2{,}11 \cdot 10^5$	25.830,6
BnBPanShi	20	20	11,7	$3{,}9 \cdot 10^5$	$2{,}61 \cdot 10^6$	$1{,}97 \cdot 10^6$	$1{,}33 \cdot 10^5$	26.721,35
BnBChainNoDom	20	0	186,4	$2{,}13 \cdot 10^7$	$1{,}35 \cdot 10^8$	0	$3{,}19 \cdot 10^7$	$2{,}06 \cdot 10^6$
BnBChainNoDom3xNodes	20	0	564,2	$6{,}52 \cdot 10^7$	$4{,}15 \cdot 10^8$	0	$1{,}03 \cdot 10^8$	$9{,}47 \cdot 10^6$
BnBSimpleBoundsAndLAP	20	20	35,8	$3{,}38 \cdot 10^5$	$2{,}26 \cdot 10^6$	$1{,}72 \cdot 10^6$	$1{,}11 \cdot 10^5$	29.204,2

Tabelle A.2: Branch-and-Bound: 50_10_BigSetup

Konfig.	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
BnBSimpleBounds	20	18	4.942,7	$4{,}43 \cdot 10^7$	$4{,}07 \cdot 10^8$	$3{,}15 \cdot 10^8$	$2{,}36 \cdot 10^7$	$1{,}13 \cdot 10^6$
BnBChain	20	15	646,3	$5{,}86 \cdot 10^7$	$5{,}38 \cdot 10^8$	$3{,}97 \cdot 10^8$	$4{,}09 \cdot 10^7$	$2{,}88 \cdot 10^6$
BnBLAP	20	18	11.539,8	$4{,}35 \cdot 10^7$	$3{,}99 \cdot 10^8$	$3{,}12 \cdot 10^8$	$2{,}15 \cdot 10^7$	$1{,}34 \cdot 10^6$
BnBVSP	20	15	806,2	$5{,}92 \cdot 10^7$	$5{,}43 \cdot 10^8$	$4{,}01 \cdot 10^8$	$4{,}14 \cdot 10^7$	$3{,}09 \cdot 10^6$
BnBBelouadah	20	18	2.722,7	$4{,}49 \cdot 10^7$	$4{,}13 \cdot 10^8$	$3{,}19 \cdot 10^8$	$2{,}39 \cdot 10^7$	$1{,}05 \cdot 10^6$
BnBPosner	20	14	838,7	$6{,}25 \cdot 10^7$	$5{,}73 \cdot 10^8$	$4{,}17 \cdot 10^8$	$4{,}28 \cdot 10^7$	$4{,}89 \cdot 10^6$
BnBPanShi	20	17	3.725,6	$4{,}66 \cdot 10^7$	$4{,}29 \cdot 10^8$	$3{,}3 \cdot 10^8$	$2{,}55 \cdot 10^7$	$1{,}57 \cdot 10^6$
BnBSimpleBoundsAndLAP	20	18	13.732,3	$4{,}23 \cdot 10^7$	$3{,}88 \cdot 10^8$	$3{,}04 \cdot 10^8$	$2{,}11 \cdot 10^7$	$9{,}67 \cdot 10^5$

Tabelle A.3: Branch-and-Bound: 75_15_BigSetup

Konfig.	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
BnBSimpleBounds	20	14	3.801,8	$3{,}94 \cdot 10^7$	$4{,}92 \cdot 10^8$	$3{,}63 \cdot 10^8$	$4{,}24 \cdot 10^7$	$7{,}65 \cdot 10^6$
BnBChain	20	11	615,5	$5{,}69 \cdot 10^7$	$6{,}95 \cdot 10^8$	$4{,}85 \cdot 10^8$	$7{,}92 \cdot 10^7$	$1{,}87 \cdot 10^7$
BnBLAP	20	7	11.155,8	$5{,}76 \cdot 10^7$	$7{,}16 \cdot 10^8$	$5{,}1 \cdot 10^8$	$6{,}39 \cdot 10^7$	$1{,}48 \cdot 10^7$
BnBVSP	20	11	614,6	$5{,}1 \cdot 10^7$	$6{,}3 \cdot 10^8$	$4{,}46 \cdot 10^8$	$6{,}78 \cdot 10^7$	$1{,}08 \cdot 10^7$
BnBBelouadah	20	6	2.239,0	$6{,}35 \cdot 10^7$	$7{,}74 \cdot 10^8$	$5{,}27 \cdot 10^8$	$8{,}47 \cdot 10^7$	$2{,}5 \cdot 10^7$
BnBPosner	20	11	628,8	$5{,}17 \cdot 10^7$	$6{,}36 \cdot 10^8$	$4{,}51 \cdot 10^8$	$5{,}63 \cdot 10^7$	$2{,}13 \cdot 10^7$
BnBPanShi	20	4	4.179,6	$6{,}06 \cdot 10^7$	$7{,}43 \cdot 10^8$	$5{,}26 \cdot 10^8$	$7{,}34 \cdot 10^7$	$2{,}59 \cdot 10^6$
BnBChainNoDom	20	0	226,2	$1{,}35 \cdot 10^7$	$1{,}55 \cdot 10^8$	0	$5{,}21 \cdot 10^7$	$2{,}52 \cdot 10^6$
BnBChainNoDom3xNodes	20	0	721,6	$5{,}84 \cdot 10^7$	$6{,}74 \cdot 10^8$	0	$3{,}61 \cdot 10^8$	$6{,}65 \cdot 10^6$
BnBSimpleBoundsAndLAP	20	14	10.226,9	$3{,}94 \cdot 10^7$	$4{,}92 \cdot 10^8$	$3{,}62 \cdot 10^8$	$4{,}24 \cdot 10^7$	$7{,}69 \cdot 10^6$

Tabelle A.4: Branch-and-Bound: 50_15_SmallSetup

Konfig.	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
BnBSimpleBounds	20	20	33,3	$6{,}33 \cdot 10^5$	$5{,}04 \cdot 10^6$	$3{,}76 \cdot 10^6$	$3{,}71 \cdot 10^5$	59.816,8
BnBChain	20	20	6,1	$9{,}82 \cdot 10^5$	$7{,}74 \cdot 10^6$	$5{,}6 \cdot 10^6$	$7{,}2 \cdot 10^5$	73.920,5
BnBLAP	20	20	63,4	$7{,}48 \cdot 10^5$	$6 \cdot 10^6$	$4{,}55 \cdot 10^6$	$4{,}02 \cdot 10^5$	69.706,35
BnBVSP	20	20	6,9	$9{,}54 \cdot 10^5$	$7{,}52 \cdot 10^6$	$5{,}45 \cdot 10^6$	$6{,}94 \cdot 10^5$	71.902,3
BnBBelouadah	20	20	19,6	$7{,}49 \cdot 10^5$	$6{,}01 \cdot 10^6$	$4{,}47 \cdot 10^6$	$4{,}42 \cdot 10^5$	72.431,95
BnBPosner	20	20	7,7	$1{,}14 \cdot 10^6$	$9{,}04 \cdot 10^6$	$6{,}6 \cdot 10^6$	$7{,}89 \cdot 10^5$	94.094,65
BnBPanShi	20	20	28,1	$7{,}92 \cdot 10^5$	$6{,}38 \cdot 10^6$	$4{,}76 \cdot 10^6$	$4{,}66 \cdot 10^5$	73.414,6
BnBChainNoDom	20	0	209,6	$1{,}9 \cdot 10^7$	$1{,}36 \cdot 10^8$	0	$3{,}31 \cdot 10^7$	$2{,}06 \cdot 10^6$
BnBChainNoDom3xNodes	20	0	648,9	$6{,}21 \cdot 10^7$	$4{,}45 \cdot 10^8$	0	$1{,}29 \cdot 10^8$	$1{,}35 \cdot 10^7$
BnBSimpleBoundsAndLAP	20	20	74,4	$6{,}15 \cdot 10^5$	$4{,}89 \cdot 10^6$	$3{,}68 \cdot 10^6$	$3{,}48 \cdot 10^5$	58.486,1

Tabelle A.5: Branch-and-Bound: 50_15_BigSetup

Konfig.	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
BnBSimpleBounds	20	20	721,4	$8{,}75 \cdot 10^6$	$6{,}66 \cdot 10^7$	$5{,}31 \cdot 10^7$	$2{,}27 \cdot 10^6$	$1{,}55 \cdot 10^5$
BnBChain	20	20	86,6	$1{,}04 \cdot 10^7$	$7{,}89 \cdot 10^7$	$6{,}11 \cdot 10^7$	$3{,}44 \cdot 10^6$	$1{,}78 \cdot 10^5$
BnBLAP	20	20	1.710,9	$8{,}22 \cdot 10^6$	$6{,}25 \cdot 10^7$	$5{,}02 \cdot 10^7$	$2{,}04 \cdot 10^6$	$1{,}5 \cdot 10^5$
BnBVSP	20	20	112,7	$1{,}05 \cdot 10^7$	$7{,}96 \cdot 10^7$	$6{,}19 \cdot 10^7$	$3{,}44 \cdot 10^6$	$1{,}75 \cdot 10^5$
BnBBelouadah	20	20	377,3	$8{,}78 \cdot 10^6$	$6{,}68 \cdot 10^7$	$5{,}33 \cdot 10^7$	$2{,}28 \cdot 10^6$	$1{,}46 \cdot 10^5$
BnBPosner	20	20	116,9	$1{,}13 \cdot 10^7$	$8{,}55 \cdot 10^7$	$6{,}66 \cdot 10^7$	$3{,}73 \cdot 10^6$	$1{,}78 \cdot 10^5$
BnBPanShi	20	20	508,4	$8{,}95 \cdot 10^6$	$6{,}82 \cdot 10^7$	$5{,}44 \cdot 10^7$	$2{,}33 \cdot 10^6$	$1{,}43 \cdot 10^5$
BnBSimpleBoundsAndLAP	20	20	2.080,5	$8{,}18 \cdot 10^6$	$6{,}23 \cdot 10^7$	$5 \cdot 10^7$	$2{,}03 \cdot 10^6$	$1{,}52 \cdot 10^5$

Tabelle A.6: Branch-and-Bound: 75_10_BigSetup

Konfig.	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
BnBSimpleBounds	20	1	11.329,8	$6{,}44 \cdot 10^7$	$6{,}2 \cdot 10^8$	$4{,}31 \cdot 10^8$	$3{,}26 \cdot 10^7$	$1{,}89 \cdot 10^6$
BnBChain	20	1	766,0	$6{,}54 \cdot 10^7$	$6{,}29 \cdot 10^8$	$4{,}36 \cdot 10^8$	$3{,}49 \cdot 10^7$	$1{,}79 \cdot 10^6$
BnBLAP	20	0	42.974,5	$6{,}6 \cdot 10^7$	$6{,}42 \cdot 10^8$	$4{,}68 \cdot 10^8$	$1{,}47 \cdot 10^7$	$4{,}4 \cdot 10^5$
BnBVSP	20	1	979,6	$6{,}4 \cdot 10^7$	$6{,}15 \cdot 10^8$	$4{,}26 \cdot 10^8$	$3{,}34 \cdot 10^7$	$1{,}47 \cdot 10^6$
BnBBelouadah	20	0	3.623,4	$6{,}8 \cdot 10^7$	$6{,}58 \cdot 10^8$	$4{,}71 \cdot 10^8$	$2{,}23 \cdot 10^7$	$2{,}28 \cdot 10^5$
BnBPosner	20	0	1.162,1	$6{,}59 \cdot 10^7$	$6{,}39 \cdot 10^8$	$4{,}6 \cdot 10^8$	$1{,}49 \cdot 10^7$	$6{,}05 \cdot 10^5$
BnBPanShi	20	0	8.770,7	$6{,}71 \cdot 10^7$	$6{,}51 \cdot 10^8$	$4{,}65 \cdot 10^8$	$1{,}98 \cdot 10^7$	$1{,}17 \cdot 10^5$
BnBSimpleBoundsAndLAP	20	1	47.337,4	$6{,}44 \cdot 10^7$	$6{,}2 \cdot 10^8$	$4{,}31 \cdot 10^8$	$3{,}26 \cdot 10^7$	$1{,}89 \cdot 10^6$

Tabelle A.7: Branch-and-Bound: 75_10_SmallSetup

Konfig.	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
BnBSimpleBounds	20	0	11.809,5	$4{,}01 \cdot 10^7$	$5{,}34 \cdot 10^8$	$3{,}66 \cdot 10^8$	$2{,}65 \cdot 10^7$	$7{,}19 \cdot 10^5$
BnBChain	20	0	681,2	$4{,}26 \cdot 10^7$	$5{,}62 \cdot 10^8$	$3{,}88 \cdot 10^8$	$2{,}91 \cdot 10^7$	$5{,}89 \cdot 10^5$
BnBLAP	20	0	45.872,2	$4{,}12 \cdot 10^7$	$5{,}58 \cdot 10^8$	$4{,}08 \cdot 10^8$	$8{,}7 \cdot 10^6$	$2{,}75 \cdot 10^5$
BnBVSP	20	0	844,7	$3{,}91 \cdot 10^7$	$5{,}2 \cdot 10^8$	$3{,}53 \cdot 10^8$	$2{,}55 \cdot 10^7$	$6{,}87 \cdot 10^5$
BnBBelouadah	20	0	4.162,9	$4{,}27 \cdot 10^7$	$5{,}71 \cdot 10^8$	$4{,}06 \cdot 10^8$	$1{,}56 \cdot 10^7$	$1{,}1 \cdot 10^5$
BnBPosner	20	0	969,9	$4{,}07 \cdot 10^7$	$5{,}48 \cdot 10^8$	$3{,}93 \cdot 10^8$	$8{,}87 \cdot 10^6$	$4{,}98 \cdot 10^5$
BnBPanShi	20	0	8.884,3	$4{,}18 \cdot 10^7$	$5{,}61 \cdot 10^8$	$4 \cdot 10^8$	$1{,}19 \cdot 10^7$	47.929,2
BnBSimpleBoundsAndLAP	20	0	50.687,5	$4{,}01 \cdot 10^7$	$5{,}34 \cdot 10^8$	$3{,}66 \cdot 10^8$	$2{,}65 \cdot 10^7$	$7{,}19 \cdot 10^5$

Tabelle A.8: Branch-and-Bound: 75_15_SmallSetup

Konfig.	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
BnBSimpleBounds	20	20	1,4	36.744	$2{,}16 \cdot 10^5$	$1{,}44 \cdot 10^5$	24.138,8	2.432,1
BnBChain	20	20	$1{,}8 \cdot 10^{-1}$	48.820,9	$2{,}84 \cdot 10^5$	$1{,}84 \cdot 10^5$	34.709	2.512,35
BnBLAP	20	20	2,0	36.126,05	$2{,}14 \cdot 10^5$	$1{,}44 \cdot 10^5$	23.260,3	2.560,2
BnBVSP	20	20	$2{,}2 \cdot 10^{-1}$	47.198	$2{,}75 \cdot 10^5$	$1{,}77 \cdot 10^5$	33.611,6	2.458,75
BnBBelouadah	20	20	$7{,}4 \cdot 10^{-1}$	37.997,75	$2{,}24 \cdot 10^5$	$1{,}5 \cdot 10^5$	24.694,75	2.571,85
BnBPosner	20	20	$2{,}2 \cdot 10^{-1}$	52.598,2	$3{,}06 \cdot 10^5$	$2{,}01 \cdot 10^5$	35.932,05	1.965,7
BnBPanShi	20	20	1,0	38.899,05	$2{,}3 \cdot 10^5$	$1{,}54 \cdot 10^5$	25.039,8	2.564,85
BnBChainNoDom	20	0	218,3	$3{,}42 \cdot 10^7$	$1{,}93 \cdot 10^8$	0	$7{,}83 \cdot 10^7$	$1{,}07 \cdot 10^7$
BnBChainNoDom3xNodes	20	1	612,0	$9{,}5 \cdot 10^7$	$5{,}33 \cdot 10^8$	0	$2{,}09 \cdot 10^8$	$2{,}31 \cdot 10^7$
BnBSimpleBoundsAndLAP	20	20	2,9	34.973,1	$2{,}07 \cdot 10^5$	$1{,}39 \cdot 10^5$	22.810,85	2.600,55

Tabelle A.9: Branch-and-Bound: 50_10_BigSetup_SmallDeadline

Konfig.	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
BnBSimpleBounds	20	20	37,8	$6{,}74 \cdot 10^5$	$7{,}61 \cdot 10^6$	$5{,}09 \cdot 10^6$	$1{,}52 \cdot 10^6$	34.791,45
BnBChain	20	20	7,1	$1{,}06 \cdot 10^6$	$1{,}19 \cdot 10^7$	$8{,}14 \cdot 10^6$	$2{,}34 \cdot 10^6$	33.178,1
BnBLAP	20	20	69,1	$1 \cdot 10^6$	$1{,}12 \cdot 10^7$	$7{,}77 \cdot 10^6$	$2{,}03 \cdot 10^6$	36.632,15
BnBVSP	20	20	7,7	$1{,}04 \cdot 10^6$	$1{,}16 \cdot 10^7$	$7{,}94 \cdot 10^6$	$2{,}29 \cdot 10^6$	32.392,3
BnBBelouadah	20	20	21,1	$1{,}16 \cdot 10^6$	$1{,}28 \cdot 10^7$	$8{,}84 \cdot 10^6$	$2{,}38 \cdot 10^6$	34.549,45
BnBPosner	20	20	5,7	$8{,}07 \cdot 10^5$	$9 \cdot 10^6$	$5{,}98 \cdot 10^6$	$1{,}81 \cdot 10^6$	45.303,9
BnBPanShi	20	20	39,6	$1{,}31 \cdot 10^6$	$1{,}43 \cdot 10^7$	$1 \cdot 10^7$	$2{,}58 \cdot 10^6$	35.349,65
BnBChainNoDom	20	0	311,6	$3{,}88 \cdot 10^7$	$4{,}55 \cdot 10^8$	0	$3{,}46 \cdot 10^8$	14.898,1
BnBChainNoDom3xNodes	20	0	990,9	$1{,}34 \cdot 10^8$	$1{,}54 \cdot 10^9$	0	$1{,}21 \cdot 10^9$	25.434,95
BnBSimpleBoundsAndLAP	20	20	84,3	$6{,}73 \cdot 10^5$	$7{,}59 \cdot 10^6$	$5{,}08 \cdot 10^6$	$1{,}51 \cdot 10^6$	34.744,65

Tabelle A.10: Branch-and-Bound: 50_15_SmallSetup_SmallDeadline

Konfig.	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
BnBSimpleBounds	20	20	11,2	$2{,}25 \cdot 10^5$	$1{,}97 \cdot 10^6$	$1{,}26 \cdot 10^6$	$3{,}83 \cdot 10^5$	16.021,35
BnBChain	20	20	1,5	$3{,}26 \cdot 10^5$	$2{,}82 \cdot 10^6$	$1{,}85 \cdot 10^6$	$5{,}44 \cdot 10^5$	13.043,3
BnBLAP	20	20	20,9	$3{,}18 \cdot 10^5$	$2{,}75 \cdot 10^6$	$1{,}83 \cdot 10^6$	$4{,}97 \cdot 10^5$	17.473,8
BnBVSP	20	20	1,9	$3{,}22 \cdot 10^5$	$2{,}79 \cdot 10^6$	$1{,}82 \cdot 10^6$	$5{,}38 \cdot 10^5$	12.982,15
BnBBelouadah	20	20	5,8	$4{,}22 \cdot 10^5$	$3{,}59 \cdot 10^6$	$2{,}41 \cdot 10^6$	$6{,}34 \cdot 10^5$	13.205,1
BnBPosner	20	20	1,4	$2{,}65 \cdot 10^5$	$2{,}3 \cdot 10^6$	$1{,}48 \cdot 10^6$	$4{,}4 \cdot 10^5$	19.750,65
BnBPanShi	20	20	12,0	$4{,}66 \cdot 10^5$	$3{,}94 \cdot 10^6$	$2{,}67 \cdot 10^6$	$6{,}82 \cdot 10^5$	12.799,85
BnBChainNoDom	20	0	238,1	$3{,}53 \cdot 10^7$	$3{,}14 \cdot 10^8$	0	$2{,}09 \cdot 10^8$	8.655,65
BnBChainNoDom3xNodes	20	0	757,7	$1{,}21 \cdot 10^8$	$1{,}07 \cdot 10^9$	0	$7{,}5 \cdot 10^8$	$1{,}29 \cdot 10^5$
BnBSimpleBoundsAndLAP	20	20	26,6	$2{,}23 \cdot 10^5$	$1{,}95 \cdot 10^6$	$1{,}26 \cdot 10^6$	$3{,}8 \cdot 10^5$	15.945,1

Tabelle A.11: Branch-and-Bound: 50_10_SmallSetup_SmallDeadline

Konfig.	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
BnBSimpleBounds	20	20	2,0	45.471	$2{,}89 \cdot 10^5$	$1{,}82 \cdot 10^5$	36.788,6	6.829,75
BnBChain	20	20	$2{,}5 \cdot 10^{-1}$	60.165,35	$3{,}77 \cdot 10^5$	$2{,}32 \cdot 10^5$	51.623,4	7.851,75
BnBLAP	20	20	4,0	76.509,5	$5{,}34 \cdot 10^5$	$3{,}53 \cdot 10^5$	61.073,85	11.213,25
BnBVSP	20	20	$3 \cdot 10^{-1}$	59.239,35	$3{,}72 \cdot 10^5$	$2{,}3 \cdot 10^5$	50.591,75	8.263,15
BnBBelouadah	20	20	1,6	78.384,85	$5{,}46 \cdot 10^5$	$3{,}48 \cdot 10^5$	69.140,05	12.922,15
BnBPosner	20	20	$3{,}2 \cdot 10^{-1}$	68.966,15	$4{,}43 \cdot 10^5$	$2{,}8 \cdot 10^5$	55.077,85	10.536,7
BnBPanShi	20	20	2,3	86.903,75	$6{,}11 \cdot 10^5$	$3{,}96 \cdot 10^5$	75.820,8	13.676,45
BnBChainNoDom	20	3	215,4	$3{,}15 \cdot 10^7$	$1{,}96 \cdot 10^8$	0	$9{,}17 \cdot 10^7$	$8{,}55 \cdot 10^6$
BnBChainNoDom3xNodes	20	5	615,5	$9{,}28 \cdot 10^7$	$5{,}58 \cdot 10^8$	0	$2{,}57 \cdot 10^8$	$3{,}91 \cdot 10^7$
BnBSimpleBoundsAndLAP	20	20	3,9	44.618,5	$2{,}84 \cdot 10^5$	$1{,}8 \cdot 10^5$	35.857,05	6.962,7

Tabelle A.12: Branch-and-Bound: 50_15_BigSetup_SmallDeadline

Konfig.	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
BnBSimpleBounds	20	20	42,9	$5{,}83 \cdot 10^5$	$3{,}89 \cdot 10^6$	$2{,}87 \cdot 10^6$	$2{,}84 \cdot 10^5$	7.049,55
BnBChain	20	20	4,1	$7{,}06 \cdot 10^5$	$4{,}67 \cdot 10^6$	$3{,}3 \cdot 10^6$	$4{,}23 \cdot 10^5$	8.022,55
BnBLAP	20	20	84,7	$5{,}74 \cdot 10^5$	$3{,}83 \cdot 10^6$	$2{,}87 \cdot 10^6$	$2{,}69 \cdot 10^5$	7.360,65
BnBVSP	20	20	5,6	$7{,}05 \cdot 10^5$	$4{,}66 \cdot 10^6$	$3{,}3 \cdot 10^6$	$4{,}2 \cdot 10^5$	8.443,2
BnBBelouadah	20	20	23,3	$5{,}86 \cdot 10^5$	$3{,}91 \cdot 10^6$	$2{,}89 \cdot 10^6$	$2{,}85 \cdot 10^5$	7.034,55
BnBPosner	20	20	5,6	$7{,}3 \cdot 10^5$	$4{,}83 \cdot 10^6$	$3{,}43 \cdot 10^6$	$4{,}33 \cdot 10^5$	8.423
BnBPanShi	20	20	30,2	$5{,}93 \cdot 10^5$	$3{,}95 \cdot 10^6$	$2{,}93 \cdot 10^6$	$2{,}86 \cdot 10^5$	6.943,15
BnBSimpleBoundsAndLAP	20	20	108,3	$5{,}71 \cdot 10^5$	$3{,}81 \cdot 10^6$	$2{,}85 \cdot 10^6$	$2{,}69 \cdot 10^5$	7.613,3

Tabelle A.13: Branch-and-Bound: 75_10_BigSetup_SmallDeadline

Konfig.	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
BnBSimpleBounds	20	20	884,3	$1{,}03 \cdot 10^7$	$9{,}62 \cdot 10^7$	$6{,}92 \cdot 10^7$	$1{,}2 \cdot 10^7$	72.765,9
BnBChain	20	20	117,6	$1{,}29 \cdot 10^7$	$1{,}19 \cdot 10^8$	$8{,}64 \cdot 10^7$	$1{,}52 \cdot 10^7$	49.090,85
BnBLAP	20	20	2.373,9	$1{,}58 \cdot 10^7$	$1{,}45 \cdot 10^8$	$1{,}07 \cdot 10^8$	$1{,}64 \cdot 10^7$	50.652,95
BnBVSP	20	20	164,8	$1{,}42 \cdot 10^7$	$1{,}31 \cdot 10^8$	$9{,}53 \cdot 10^7$	$1{,}64 \cdot 10^7$	37.505,35
BnBBelouadah	20	20	472,0	$1{,}89 \cdot 10^7$	$1{,}72 \cdot 10^8$	$1{,}28 \cdot 10^8$	$1{,}94 \cdot 10^7$	33.302,4
BnBPosner	20	20	137,5	$1{,}25 \cdot 10^7$	$1{,}15 \cdot 10^8$	$8{,}27 \cdot 10^7$	$1{,}42 \cdot 10^7$	96.754,95
BnBPanShi	20	20	966,4	$2{,}13 \cdot 10^7$	$1{,}93 \cdot 10^8$	$1{,}44 \cdot 10^8$	$2{,}11 \cdot 10^7$	26.541,9
BnBSimpleBoundsAndLAP	20	20	2.590,8	$1{,}03 \cdot 10^7$	$9{,}61 \cdot 10^7$	$6{,}91 \cdot 10^7$	$1{,}2 \cdot 10^7$	74.890,6

Tabelle A.14: Branch-and-Bound: 75_10_SmallSetup_SmallDeadline

Konfig.	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
BnBSimpleBounds	20	20	162,2	$2{,}12 \cdot 10^6$	$1{,}69 \cdot 10^7$	$1{,}27 \cdot 10^7$	$1{,}39 \cdot 10^6$	27.923,4
BnBChain	20	20	21,5	$2{,}7 \cdot 10^6$	$2{,}13 \cdot 10^7$	$1{,}53 \cdot 10^7$	$2{,}15 \cdot 10^6$	31.464,6
BnBLAP	20	20	288,9	$2{,}08 \cdot 10^6$	$1{,}65 \cdot 10^7$	$1{,}27 \cdot 10^7$	$1{,}27 \cdot 10^6$	19.261,1
BnBVSP	20	20	26,2	$2{,}65 \cdot 10^6$	$2{,}09 \cdot 10^7$	$1{,}51 \cdot 10^7$	$2{,}1 \cdot 10^6$	31.828,45
BnBBelouadah	20	20	93,9	$2{,}15 \cdot 10^6$	$1{,}72 \cdot 10^7$	$1{,}29 \cdot 10^7$	$1{,}41 \cdot 10^6$	25.981,85
BnBPosner	20	20	25,9	$2{,}82 \cdot 10^6$	$2{,}23 \cdot 10^7$	$1{,}63 \cdot 10^7$	$2{,}18 \cdot 10^6$	33.413,15
BnBPanShi	20	20	118,3	$2{,}2 \cdot 10^6$	$1{,}75 \cdot 10^7$	$1{,}33 \cdot 10^7$	$1{,}42 \cdot 10^6$	25.378,2
BnBSimpleBoundsAndLAP	20	20	373,2	$2{,}04 \cdot 10^6$	$1{,}63 \cdot 10^7$	$1{,}24 \cdot 10^7$	$1{,}29 \cdot 10^6$	23.471,45

Tabelle A.15: Branch-and-Bound: 75_15_BigSetup_SmallDeadline

Konfig.	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
BnBSimpleBounds	20	8	7.208,3	$5{,}38 \cdot 10^7$	$6{,}88 \cdot 10^8$	$4{,}38 \cdot 10^8$	$1{,}24 \cdot 10^8$	$1{,}49 \cdot 10^6$
BnBChain	20	3	791,5	$6{,}07 \cdot 10^7$	$7{,}75 \cdot 10^8$	$4{,}76 \cdot 10^8$	$1{,}61 \cdot 10^8$	$3{,}12 \cdot 10^5$
BnBLAP	20	2	17.152,4	$5{,}95 \cdot 10^7$	$7{,}65 \cdot 10^8$	$4{,}74 \cdot 10^8$	$1{,}48 \cdot 10^8$	$5{,}34 \cdot 10^5$
BnBVSP	20	3	993,3	$6 \cdot 10^7$	$7{,}67 \cdot 10^8$	$4{,}67 \cdot 10^8$	$1{,}62 \cdot 10^8$	$2{,}96 \cdot 10^5$
BnBBelouadah	20	2	3.136,0	$6{,}12 \cdot 10^7$	$7{,}86 \cdot 10^8$	$4{,}7 \cdot 10^8$	$1{,}69 \cdot 10^8$	$2{,}14 \cdot 10^5$
BnBPosner	20	4	860,4	$5{,}65 \cdot 10^7$	$7{,}21 \cdot 10^8$	$4{,}5 \cdot 10^8$	$1{,}31 \cdot 10^8$	$1{,}36 \cdot 10^6$
BnBPanShi	20	1	5.873,3	$6{,}12 \cdot 10^7$	$7{,}86 \cdot 10^8$	$4{,}64 \cdot 10^8$	$1{,}75 \cdot 10^8$	$1{,}28 \cdot 10^5$
BnBSimpleBoundsAndLAP	20	8	20.336,6	$5{,}37 \cdot 10^7$	$6{,}87 \cdot 10^8$	$4{,}38 \cdot 10^8$	$1{,}24 \cdot 10^8$	$1{,}49 \cdot 10^6$

Tabelle A.16: Branch-and-Bound: 75_15_SmallSetup_SmallDeadline

Instanzmenge	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
50_10_BigSetup	20	20	3,0	$4{,}76 \cdot 10^5$	$3{,}12 \cdot 10^6$	$2{,}28 \cdot 10^6$	$1{,}93 \cdot 10^5$	27.100,75
50_10_BigSetup_SmallDeadline	20	20	$2{,}2 \cdot 10^{-1}$	47.198	$2{,}75 \cdot 10^5$	$1{,}77 \cdot 10^5$	33.611,6	2.458,75
50_10_SmallSetup	20	20	85,5	$9{,}43 \cdot 10^6$	$8{,}72 \cdot 10^7$	$6{,}5 \cdot 10^7$	$6{,}6 \cdot 10^6$	$9{,}06 \cdot 10^5$
50_10_SmallSetup_SmallDeadline	20	20	1,9	$3{,}22 \cdot 10^5$	$2{,}79 \cdot 10^6$	$1{,}82 \cdot 10^6$	$5{,}38 \cdot 10^5$	12.982,15
50_15_BigSetup	20	20	6,9	$9{,}54 \cdot 10^5$	$7{,}52 \cdot 10^6$	$5{,}45 \cdot 10^6$	$6{,}94 \cdot 10^5$	71.902,3
50_15_BigSetup_SmallDeadline	20	20	$3 \cdot 10^{-1}$	59.239,35	$3{,}72 \cdot 10^5$	$2{,}3 \cdot 10^5$	50.591,75	8.263,15
50_15_SmallSetup	20	11	614,6	$5{,}1 \cdot 10^7$	$6{,}3 \cdot 10^8$	$4{,}46 \cdot 10^8$	$6{,}78 \cdot 10^7$	$1{,}08 \cdot 10^7$
50_15_SmallSetup_SmallDeadline	20	20	7,7	$1{,}04 \cdot 10^6$	$1{,}16 \cdot 10^7$	$7{,}94 \cdot 10^6$	$2{,}29 \cdot 10^6$	32.392,3
75_10_BigSetup	20	20	112,7	$1{,}05 \cdot 10^7$	$7{,}96 \cdot 10^7$	$6{,}19 \cdot 10^7$	$3{,}44 \cdot 10^6$	$1{,}75 \cdot 10^5$
75_10_BigSetup_SmallDeadline	20	20	5,6	$7{,}05 \cdot 10^5$	$4{,}66 \cdot 10^6$	$3{,}3 \cdot 10^6$	$4{,}2 \cdot 10^5$	8.443,2
75_10_SmallSetup	20	1	979,6	$6{,}4 \cdot 10^7$	$6{,}15 \cdot 10^8$	$4{,}26 \cdot 10^8$	$3{,}34 \cdot 10^7$	$1{,}47 \cdot 10^6$
75_10_SmallSetup_SmallDeadline	20	20	164,8	$1{,}42 \cdot 10^7$	$1{,}31 \cdot 10^8$	$9{,}53 \cdot 10^7$	$1{,}64 \cdot 10^7$	37.505,35
75_15_BigSetup	20	15	806,2	$5{,}92 \cdot 10^7$	$5{,}43 \cdot 10^8$	$4{,}01 \cdot 10^8$	$4{,}14 \cdot 10^7$	$3{,}09 \cdot 10^6$
75_15_BigSetup_SmallDeadline	20	20	26,2	$2{,}65 \cdot 10^6$	$2{,}09 \cdot 10^7$	$1{,}51 \cdot 10^7$	$2{,}1 \cdot 10^6$	31.828,45
75_15_SmallSetup	20	0	844,7	$3{,}91 \cdot 10^7$	$5{,}2 \cdot 10^8$	$3{,}53 \cdot 10^8$	$2{,}55 \cdot 10^7$	$6{,}87 \cdot 10^5$
75_15_SmallSetup_SmallDeadline	20	3	993,3	$6 \cdot 10^7$	$7{,}67 \cdot 10^8$	$4{,}67 \cdot 10^8$	$1{,}62 \cdot 10^8$	$2{,}96 \cdot 10^5$

Tabelle A.17: Ergebnisse für die Konfiguration BnBVSP

Instanzmenge	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
50_10_BigSetup	20	20	8,1	$3{,}77 \cdot 10^5$	$2{,}53 \cdot 10^6$	$1{,}9 \cdot 10^6$	$1{,}29 \cdot 10^5$	28.008,25
50_10_BigSetup_SmallDeadline	20	20	$7{,}4 \cdot 10^{-1}$	37.997,75	$2{,}24 \cdot 10^5$	$1{,}5 \cdot 10^5$	24.694,75	2.571,85
50_10_SmallSetup	20	19	805,5	$3{,}91 \cdot 10^7$	$3{,}53 \cdot 10^8$	$2{,}63 \cdot 10^8$	$2{,}61 \cdot 10^7$	$7{,}65 \cdot 10^6$
50_10_SmallSetup_SmallDeadline	20	20	5,8	$4{,}22 \cdot 10^5$	$3{,}59 \cdot 10^6$	$2{,}41 \cdot 10^6$	$6{,}34 \cdot 10^5$	13.205,1
50_15_BigSetup	20	20	19,6	$7{,}49 \cdot 10^5$	$6{,}01 \cdot 10^6$	$4{,}47 \cdot 10^6$	$4{,}42 \cdot 10^5$	72.431,95
50_15_BigSetup_SmallDeadline	20	20	1,6	78.384,85	$5{,}46 \cdot 10^5$	$3{,}48 \cdot 10^5$	69.140,05	12.922,15
50_15_SmallSetup	20	6	2.239,0	$6{,}35 \cdot 10^7$	$7{,}74 \cdot 10^8$	$5{,}27 \cdot 10^8$	$8{,}47 \cdot 10^7$	$2{,}5 \cdot 10^7$
50_15_SmallSetup_SmallDeadline	20	20	21,1	$1{,}16 \cdot 10^6$	$1{,}28 \cdot 10^7$	$8{,}84 \cdot 10^6$	$2{,}38 \cdot 10^6$	34.549,45
75_10_BigSetup	20	20	377,3	$8{,}78 \cdot 10^6$	$6{,}68 \cdot 10^7$	$5{,}33 \cdot 10^7$	$2{,}28 \cdot 10^6$	$1{,}46 \cdot 10^5$
75_10_BigSetup_SmallDeadline	20	20	23,3	$5{,}86 \cdot 10^5$	$3{,}91 \cdot 10^6$	$2{,}89 \cdot 10^6$	$2{,}85 \cdot 10^5$	7.034,55
75_10_SmallSetup	20	0	3.623,4	$6{,}8 \cdot 10^7$	$6{,}58 \cdot 10^8$	$4{,}71 \cdot 10^8$	$2{,}23 \cdot 10^7$	$2{,}28 \cdot 10^5$
75_10_SmallSetup_SmallDeadline	20	20	472,0	$1{,}89 \cdot 10^7$	$1{,}72 \cdot 10^8$	$1{,}28 \cdot 10^8$	$1{,}94 \cdot 10^7$	33.302,4
75_15_BigSetup	20	18	2.722,7	$4{,}49 \cdot 10^7$	$4{,}13 \cdot 10^8$	$3{,}19 \cdot 10^8$	$2{,}39 \cdot 10^7$	$1{,}05 \cdot 10^6$
75_15_BigSetup_SmallDeadline	20	20	93,9	$2{,}15 \cdot 10^6$	$1{,}72 \cdot 10^7$	$1{,}29 \cdot 10^7$	$1{,}41 \cdot 10^6$	25.981,85
75_15_SmallSetup	20	0	4.162,9	$4{,}27 \cdot 10^7$	$5{,}71 \cdot 10^8$	$4{,}06 \cdot 10^8$	$1{,}56 \cdot 10^7$	$1{,}1 \cdot 10^5$
75_15_SmallSetup_SmallDeadline	20	2	3.136,0	$6{,}12 \cdot 10^7$	$7{,}86 \cdot 10^8$	$4{,}7 \cdot 10^8$	$1{,}69 \cdot 10^8$	$2{,}14 \cdot 10^5$

Tabelle A.18: Ergebnisse für die Konfiguration BnBBelouadah

Instanzmenge	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
50_10_BigSetup	20	20	3,1	$5{,}29 \cdot 10^5$	$3{,}48 \cdot 10^6$	$2{,}57 \cdot 10^6$	$2{,}11 \cdot 10^5$	25.830,6
50_10_BigSetup_SmallDeadline	20	20	$2{,}2 \cdot 10^{-1}$	52.598,2	$3{,}06 \cdot 10^5$	$2{,}01 \cdot 10^5$	35.932,05	1.965,7
50_10_SmallSetup	20	20	234,8	$2{,}46 \cdot 10^7$	$2{,}21 \cdot 10^8$	$1{,}68 \cdot 10^8$	$1{,}39 \cdot 10^7$	$3{,}38 \cdot 10^6$
50_10_SmallSetup_SmallDeadline	20	20	1,4	$2{,}65 \cdot 10^5$	$2{,}3 \cdot 10^6$	$1{,}48 \cdot 10^6$	$4{,}4 \cdot 10^5$	19.750,65
50_15_BigSetup	20	20	7,7	$1{,}14 \cdot 10^6$	$9{,}04 \cdot 10^6$	$6{,}6 \cdot 10^6$	$7{,}89 \cdot 10^5$	94.094,65
50_15_BigSetup_SmallDeadline	20	20	$3{,}2 \cdot 10^{-1}$	68.966,15	$4{,}43 \cdot 10^5$	$2{,}8 \cdot 10^5$	55.077,85	10.536,7
50_15_SmallSetup	20	11	628,8	$5{,}17 \cdot 10^7$	$6{,}36 \cdot 10^8$	$4{,}51 \cdot 10^8$	$5{,}63 \cdot 10^7$	$2{,}13 \cdot 10^7$
50_15_SmallSetup_SmallDeadline	20	20	5,7	$8{,}07 \cdot 10^5$	$9 \cdot 10^6$	$5{,}98 \cdot 10^6$	$1{,}81 \cdot 10^6$	45.303,9
75_10_BigSetup	20	20	116,9	$1{,}13 \cdot 10^7$	$8{,}55 \cdot 10^7$	$6{,}66 \cdot 10^7$	$3{,}73 \cdot 10^6$	$1{,}78 \cdot 10^5$
75_10_BigSetup_SmallDeadline	20	20	5,6	$7{,}3 \cdot 10^5$	$4{,}83 \cdot 10^6$	$3{,}43 \cdot 10^6$	$4{,}33 \cdot 10^5$	8.423
75_10_SmallSetup	20	0	1.162,1	$6{,}59 \cdot 10^7$	$6{,}39 \cdot 10^8$	$4{,}6 \cdot 10^8$	$1{,}49 \cdot 10^7$	$6{,}05 \cdot 10^5$
75_10_SmallSetup_SmallDeadline	20	20	137,5	$1{,}25 \cdot 10^7$	$1{,}15 \cdot 10^8$	$8{,}27 \cdot 10^7$	$1{,}42 \cdot 10^7$	96.754,95
75_15_BigSetup	20	14	838,7	$6{,}25 \cdot 10^7$	$5{,}73 \cdot 10^8$	$4{,}17 \cdot 10^8$	$4{,}28 \cdot 10^7$	$4{,}89 \cdot 10^6$
75_15_BigSetup_SmallDeadline	20	20	25,9	$2{,}82 \cdot 10^6$	$2{,}23 \cdot 10^7$	$1{,}63 \cdot 10^7$	$2{,}18 \cdot 10^6$	33.413,15
75_15_SmallSetup	20	0	969,9	$4{,}07 \cdot 10^7$	$5{,}48 \cdot 10^8$	$3{,}93 \cdot 10^8$	$8{,}87 \cdot 10^6$	$4{,}98 \cdot 10^5$
75_15_SmallSetup_SmallDeadline	20	4	860,4	$5{,}65 \cdot 10^7$	$7{,}21 \cdot 10^8$	$4{,}5 \cdot 10^8$	$1{,}31 \cdot 10^8$	$1{,}36 \cdot 10^6$

Tabelle A.19: Ergebnisse für die Konfiguration BnBPosner

Instanzmenge	Total	Optimal	∅ Laufzeit(s)	∅ Branchings	∅ Knoten	∅ Dominance	∅ Infeasibility	∅ Bound
50_10_BigSetup	20	20	11,7	$3{,}9 \cdot 10^5$	$2{,}61 \cdot 10^6$	$1{,}97 \cdot 10^6$	$1{,}33 \cdot 10^5$	26.721,35
50_10_BigSetup_SmallDeadline	20	20	1,0	38.899,05	$2{,}3 \cdot 10^5$	$1{,}54 \cdot 10^5$	25.039,8	2.564,85
50_10_SmallSetup	20	17	1.986,6	$4{,}76 \cdot 10^7$	$4{,}31 \cdot 10^8$	$3{,}2 \cdot 10^8$	$3{,}25 \cdot 10^7$	$5{,}45 \cdot 10^6$
50_10_SmallSetup_SmallDeadline	20	20	12,0	$4{,}66 \cdot 10^5$	$3{,}94 \cdot 10^6$	$2{,}67 \cdot 10^6$	$6{,}82 \cdot 10^5$	12.799,85
50_15_BigSetup	20	20	28,1	$7{,}92 \cdot 10^5$	$6{,}38 \cdot 10^6$	$4{,}76 \cdot 10^6$	$4{,}66 \cdot 10^5$	73.414,6
50_15_BigSetup_SmallDeadline	20	20	2,3	86.903,75	$6{,}11 \cdot 10^5$	$3{,}96 \cdot 10^5$	75.820,8	13.676,45
50_15_SmallSetup	20	4	4.179,6	$6{,}06 \cdot 10^7$	$7{,}43 \cdot 10^8$	$5{,}26 \cdot 10^8$	$7{,}34 \cdot 10^7$	$2{,}59 \cdot 10^6$
50_15_SmallSetup_SmallDeadline	20	20	39,6	$1{,}31 \cdot 10^6$	$1{,}43 \cdot 10^7$	$1 \cdot 10^7$	$2{,}58 \cdot 10^6$	35.349,65
75_10_BigSetup	20	20	508,4	$8{,}95 \cdot 10^6$	$6{,}82 \cdot 10^7$	$5{,}44 \cdot 10^7$	$2{,}33 \cdot 10^6$	$1{,}43 \cdot 10^5$
75_10_BigSetup_SmallDeadline	20	20	30,2	$5{,}93 \cdot 10^5$	$3{,}95 \cdot 10^6$	$2{,}93 \cdot 10^6$	$2{,}86 \cdot 10^5$	6.943,15
75_10_SmallSetup	20	0	8.770,7	$6{,}71 \cdot 10^7$	$6{,}51 \cdot 10^8$	$4{,}65 \cdot 10^8$	$1{,}98 \cdot 10^7$	$1{,}17 \cdot 10^5$
75_10_SmallSetup_SmallDeadline	20	20	966,4	$2{,}13 \cdot 10^7$	$1{,}93 \cdot 10^8$	$1{,}44 \cdot 10^8$	$2{,}11 \cdot 10^7$	26.541,9
75_15_BigSetup	20	17	3.725,6	$4{,}66 \cdot 10^7$	$4{,}29 \cdot 10^8$	$3{,}3 \cdot 10^8$	$2{,}55 \cdot 10^7$	$1{,}57 \cdot 10^6$
75_15_BigSetup_SmallDeadline	20	20	118,3	$2{,}2 \cdot 10^6$	$1{,}75 \cdot 10^7$	$1{,}33 \cdot 10^7$	$1{,}42 \cdot 10^6$	25.378,2
75_15_SmallSetup	20	0	8.884,3	$4{,}18 \cdot 10^7$	$5{,}61 \cdot 10^8$	$4 \cdot 10^8$	$1{,}19 \cdot 10^7$	47.929,2
75_15_SmallSetup_SmallDeadline	18	1	5.781,2	$6{,}11 \cdot 10^7$	$7{,}82 \cdot 10^8$	$4{,}65 \cdot 10^8$	$1{,}7 \cdot 10^8$	$1{,}42 \cdot 10^5$

Tabelle A.20: Ergebnisse für die Konfiguration BnBPanShi

Literatur

Acuña, V. u. a. (2014). „Solving the maximum edge biclique packing problem on unbalanced bipartite graphs“. In: *Discrete Applied Mathematics* 164, Part 1. Combinatorial Optimization, S. 2–12.

Agrawal, M. K., Salah E. Elmaghraby und Willy S. Herroelen (1996). „dagen: A generator of testsets for project activity nets“. In: *European Journal of Operational Research* 90.2, S. 376–382.

Aho, Alfred V., M. R. Garey und Jeffrey D. Ullman (1972). „The Transitive Reduction of a Directed Graph.“ In: *SIAM J. Comput.* 1.2, S. 131–137.

Aho, Alfred V. und Jeffrey D. Ullman (1972). *The Theory of Parsing, Translation, and Compiling*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Ahuja, Ravindra K., T.L. Magnanti und J. B. Orlin (1993). *Network Flows: Theory, Algorithms and Applications*. New Jersey: Prentice Hall.

Ahuja, Ravindra K. und James B. Orlin (2001). „A Fast Scaling Algorithm for Minimizing Separable Convex Functions Subject to Chain Constraints“. In: *Operations Research* 49.5, S. 784–789.

Akkan, Can, Andreas Drexl und Alf Kimms (2005). „Generating two-terminal directed acyclic graphs with a given complexity index by constraint logic programming“. In: *The Journal of Logic and Algebraic Programming* 62.1, S. 1–39.

Akker, J. M. van den, J. A. Hoogeveen und Steef L. van de Velde (1999). „Parallel Machine Scheduling by Column Generation“. In: *Operations Research* 47.6, S. 862–872.

— (2005). „Applying Column Generation to Machine Scheduling“. In: *Column Generation*. Hrsg. von Guy Desaulniers, Jacques Desrosiers und Marius M. Solomon. Springer. Kap. 11, S. 303–330.

Akker, J. M. van den, C.A.J. Hurkens und Martin W. P. Savelsbergh (2000). „Time-Indexed Formulations for Machine Scheduling Problems: Column Generation“. In: *INFORMS Journal on Computing* 12.2, S. 111–124.

Allahverdi, Ali (2015). „The third comprehensive survey on scheduling problems with setup times/costs“. In: *European Journal of Operational Research* 246.2, S. 345–378.

Allahverdi, Ali, Jatinder N.D Gupta und Tariq Aldowaisan (1999). „A review of scheduling research involving setup considerations“. In: *Omega* 27.2, S. 219–239.

Allahverdi, Ali, C Ng u. a. (2008). „A survey of scheduling problems with setup times or costs“. In: *European Journal of Operational Research* 187.3, S. 985–1032.

Allahverdi, Ali und H. M. Soroush (2008). „The significance of reducing setup times/setup costs“. In: *European Journal of Operational Research* 187.3, S. 978–984.

Alstrup, Stephen u. a. (1999). „Dominators in Linear Time“. In: *SIAM Journal on Computing* 28.6, S. 2117–2132.

Ambühl, Christoph und Monaldo Mastrolilli (2008). „Single Machine Precedence Constrained Scheduling Is a Vertex Cover Problem“. In: *Algorithmica* 53.4, S. 488–503.

Ambühl, Christoph, Monaldo Mastrolilli und Ola Svensson (2006). „Approximating Precedence-Constrained Single Machine Scheduling by Coloring“. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques: 9th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2006 and 10th International Workshop on Randomization and Computation, RANDOM 2006, Barcelona, Spain, August 28-30 2006. Proceedings*. Hrsg. von Josep Díaz u. a. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 15–26.

Amor, Hatem M.T. Ben, Jacques Desrosiers und Antonio Frangioni (2009). „On the choice of explicit stabilizing terms in column generation“. In: *Discrete Applied Mathematics* 157.6. Reformulation Techniques and Mathematical Programming, S. 1167–1184.

Ascheuer, Norbert, Matteo Fischetti und Martin Grötschel (2000). „A polyhedral study of the asymmetric traveling salesman problem with time windows.“ In: *Networks* 36.2, S. 69–79.

— (2001). „Solving the Asymmetric Travelling Salesman Problem with time windows by branch-and-cut“. In: *Mathematical Programming Series A* 90.3, S. 475–506.

Atamtürk, Alper, George L. Nemhauser und Martin W. P. Savelsbergh (2000). „Conflict graphs in solving integer programming problems.“ In: *European Journal of Operational Research* 121.1, S. 40–55.

Baïou, Mourad und Francisco Barahona (2016). „Stackelberg Bipartite Vertex Cover and the Preflow Algorithm“. In: *Algorithmica* 74.3, S. 1174–1183.

Balas, E. und Paolo Toth (1985). „Branch and Bound methods“. In: *The traveling salesman problem*. Hrsg. von Eugene L. Lawler u. a. John Wiley & Sons, Inc. Kap. 10.

Bang-Jensen, J. und G. Gutin (2009). *Digraphs: Theory, Algorithms and Applications (Second Edition)*. Springer.

Barnhart, Cynthia u. a. (1998). „Branch-and-Price: Column Generation for Solving Huge Integer Programs“. In: *Operations Research* 46.3, S. 316–329.

Bein, D. u. a. (2008). „Clustering and the Biclique Partition Problem“. In: *Proceedings of the 41st Hawaii International Conference on System Sciences*.

Bein, Wolfgang W., Jerzy Kamburowski und Matthias F. M. Stallmann (1992). „Optimal Reduction of Two-Terminal Directed Acyclic Graphs“. In: *SIAM Journal on Computing* 21.6, S. 1112–1129.

Belouadah, H., Marc E. Posner und Chris N. Potts (1992). „Scheduling with release dates on a single machine to minimize total weighted completion time“. In: *Discrete Applied Mathematics* 36.3, S. 213–231.

Bergeron, Anne u. a. (2008). „Computing Common Intervals of K Permutations, with Applications to Modular Decomposition of Graphs“. In: *SIAM Journal on Discrete Mathematics* 22.3, S. 1022–1039.

Bianco, Lucio, Aristide Mingozzi und Salvatore Ricciardelli (1993). „The traveling salesman problem with cumulative costs“. In: *Networks* 23.2, S. 81–91.

Bondy, J. A. und U. S. R. Murty (1976). *Graph Theory with Applications*. The Macmillan Press Ltd.

Bowman, Edward H. (1959). „The Schedule-Sequencing Problem“. In: *Operations Research* 7.5, S. 621–624.

Briskorn, Dirk u. a. (2010). „Complexity of single machine scheduling subject to nonnegative inventory constraints“. In: *European Journal of Operational Research* 207.2, S. 605–619.

Brucker, Peter (2007). *Scheduling algorithms.* 5. Aufl. Springer, S. I–XII, 1–367.

Brucker, Peter, Andreas Drexl u. a. (1999). „Resource-constrained project scheduling: Notation, classification, models, and methods“. In: *European Journal of Operational Research* 112.1, S. 3–41.

Brucker, Peter und Sigrid Knust (2002). „Lower Bounds for Scheduling a Single Robot in a Job-Shop Environment.“ In: *Annals OR* 115.1-4, S. 147–172.

— (2006). *Complex Scheduling*. Springer.

Bruno, John und Peter Downey (1978). „Complexity of Task Sequencing with Deadlines, Set-Up Times and Changeover Costs“. In: *SIAM Journal on Computing* 7.4, S. 393–404.

Bruno, John und Ravi Sethi (1977). „Task Sequencing in a Batch Environment with Setup Times“. In: *Proceedings of the International Workshop Organized by the Commision of the European Communities on Modelling and Performance Evaluation of Computer Systems*. Amsterdam, The Netherlands, The Netherlands: North-Holland Publishing Co., S. 81–88.

Buchsbaum, Adam L. u. a. (2008). „Linear-Time Algorithms for Dominators and Other Path-Evaluation Problems“. In: *SIAM Journal on Computing* 38.4, S. 1533–1573.

Burkard, Rainer, Mauro Dell'Amico und Silvano Martello (2009). *Assignment Problems*. Philadelphia: Society for Industrial und Applied Mathematics.

Cameron, Kathie (1989). „Induced matchings“. In: *Discrete Applied Mathematics* 24.1–3, S. 97–102.

Carlier, Jacques (1987). „Scheduling jobs with release dates and tails on identical machines to minimize the makespan“. In: *European Journal of Operational Research* 29.3, S. 298–306.

Chen, Zhi-Long und Warren B. Powell (1999). „Solving Parallel Machine Scheduling Problems by Column Generation“. In: *INFORMS Journal on Computing* 11.1, S. 78–94.

— (2003). „Exact algorithms for scheduling multiple families of jobs on parallel machines“. In: *Naval Research Logistics (NRL)* 50.7, S. 823–840.

Chou, Fuh-Der, Hui-Mei Wang und Tzu-Yun Chang (2009). „Algorithms for the single machine total weighted completion time scheduling problem with release times and sequence-dependent setups“. In: *The International Journal of Advanced Manufacturing Technology* 43.7-8, S. 810–821.

Chudak, Fabián A. und Dorit S. Hochbaum (1999). „A half-integral linear programming relaxation for scheduling precedence-constrained jobs on a single machine“. In: *Operations Research Letters* 25.5, S. 199–204.

Claus, A. (1984). „A new Formulation for the Travelling Salesman Problem“. In: *SIAM Journal on Algebraic and Discrete Methods* 5.1, S. 21–25.

Conway, R.W., W.L. Maxwell und Louis W. Miller (2003). *Theory of Scheduling*. Dover Books on Computer Science Series. Dover.

Cook, Stephen A. (1971). „The Complexity of Theorem-proving Procedures“. In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. STOC '71. Shaker Heights, Ohio, USA: ACM, S. 151–158.

Cormen, Thomas H. u. a. (2009). *Introduction to Algorithms (3. ed.)*. MIT Press, S. I–XIX, 1–1292.

Correa, José R. und Andreas S. Schulz (2005). „Single-Machine Scheduling with Precedence Constraints“. In: *Mathematics of Operations Research* 30.4, S. 1005–1021.

Corsten, Daniel und Christoph Gabriel (2004). Springer Berlin Heidelberg.

Dahlhaus, Elias, Jens Gustedt und Ross M. McConnell (2001). „Efficient and Practical Algorithms for Sequential Modular Decomposition“. In: *J. Algorithms* 41.2, S. 360–378.

Dakin, R.J. (1965). „A Tree Search Algorithm for Mixed Integer Programming Problems“. In: *The Computer Journal* 8, S. 250–255.

Dantzig, George B., Delbert R. Fulkerson und Selmer M. Johnson (1954). „Solution of a large-scale Traveling-Salesman Problem“. In: *Operations Research* 2, S. 393–410.

Dantzig, George B. und Philip Wolfe (1960). „Decomposition Principle for Linear Programs“. In: *Operations Research* 8.1, pp. 101–111.

Davari, Morteza u. a. (2015). „Exact algorithms for single-machine scheduling with time windows and precedence constraints“. In: *Journal of Scheduling*, S. 1–26.

Demeulemeester, Erik L., Willy S. Herroelen und Salah E. Elmaghraby (1996). „Optimal procedures for the discrete time/cost trade-off problem in project networks“. In: *European Journal of Operational Research* 88.1, S. 50–68.

Desaulniers, Guy, Jacques Desrosiers und Marius M. Solomon, Hrsg. (2005). *Column Generation*. Springer.

Desrosiers, Jacques, Jean Bertrand Gauthier und Marco E. Lübbecke (2014). „Row-reduced column generation for degenerate master problems.“ In: *European Journal of Operational Research* 236.2, S. 453–460.

Dilworth, Robert P. (1950). „A Decomposition Theorem for Partially Ordered Sets“. In: *Annals Of Mathematics*. Second Series 51.1, S. 161–166.

Divakaran, Srikrishnan und Michael Saks (2008). „Approximation algorithms for problems in scheduling with set-ups“. In: *Discrete Applied Mathematics* 156.5, S. 719–729.

Domschke, Wolfgang, Armin Scholl und Stefan Voß (1997). *Produktionsplanung: ablauforganisatorische Aspekte*. 2., überarb. und erw. Aufl. Berlin u.a.: Springer.

Drexl, Michael (2013). „A note on the separation of subtour elimination constraints in elementary shortest path problems“. In: *European Journal of Operational Research* 229.3, S. 595–598.

Du, Jianzhong und Joseph Y.-T. Leung (1990). „Minimizing Total Tardiness on One Machine Is NP-Hard“. In: *Mathematics of Operations Research* 15.3, pp. 483–495.

Dumas, Yvan u. a. (1995). „An Optimal Algorithm for the Traveling Salesman Problem with Time Windows“. In: *Operations Research* 43.2, S. 367–371.

Dushnik, Ben und E. W. Miller (1941). „Partially Ordered Sets“. In: *American Journal of Mathematics* 63.3, S. 600–610.

Eastman, W. L., S. Even und I. M. Isaacs (1964). „Bounds for the Optimal Scheduling of n Jobs on m Processors“. In: *Management Science* 11.2, S. 268–279.

Ehrgott, Matthias (2005). *Multicriteria Optimization (2. ed.)*. Springer, S. I–XIII, 1–323.

Eijl, C.A. van (1995). *A Polyhedral Approach to the Delivery Man Problem*.

Elmaghraby, Salah E. (1993). „Project Management anf Scheduling Resource allocation via dynamic programming in activity networks“. In: *European Journal of Operational Research* 64.2, S. 199–215.

— (1995). „Activity nets: A guided tour through some recent developments“. In: *European Journal of Operational Research* 82.3, S. 383–408.

Escudero, L. F., A. Garín und G. Pérez (1996). „Some properties of cliques in 0–1 mixed integer programs“. In: *Top* 4.2, S. 215–223.

Farley, A. A. (1990). „A Note on Bounding a Class of Linear Programming Problems, including Cutting Stock Problems“. In: *Operations Research* 38.5, S. 922–923.

Felsner, Stefan, Irina Mustata und Martin Pergel (2016). „The Complexity of the Partial Order Dimension Problem - Closing the Gap“. In:

Felsner, Stefan und William T. Trotter (2000). „Dimension, Graph and Hypergraph Coloring“. In: *Order* 17.2, S. 167–177.

Ferrucci, Francesco, Stefan Bock und Michel Gendreau (2013). „A pro-active real-time control approach for dynamic vehicle routing problems dealing with the delivery of urgent goods“. In: *European Journal of Operational Research* 225.1, S. 130–141.

Fischetti, Matteo und Domenico Salvagnin (2010). „Pruning Moves.“ In: *INFORMS Journal on Computing* 22.1, S. 108–119.

Fisher, Marshall L. (2004). „The Lagrangian Relaxation Method for Solving Integer Programming Problems“. In: *Management Science* 50.12_supplement, S. 1861–1871.

Fleischner, Herbert u. a. (2009). „Covering graphs with few complete bipartite subgraphs“. In: *Theoretical Computer Science* 410.21–23, S. 2045–2053.

Floyd, Robert W. (1962). „Algorithm 97: Shortest path“. In: *Communications of the ACM* 5.6, S. 345.

Ford, L. R. und Delbert R. Fulkerson (1958). „A Suggested Computation for Maximal Multi-Commodity Network Flows“. In: *Management Science* 5.1, S. 97–101.

— (1962). *Flows in Networks*. Princeton University Press.

Fraczak, Wojciech u. a. (2013). „Finding dominators via disjoint set union“. In: *Journal of Discrete Algorithms* 23. 23rd International Workshop on Combinatorial Algorithms (IWOCA 2012), S. 2–20.

— (2014). „Corrections to “Finding dominators via disjoint set union” [J. Discrete Algorithms 23 (2013) 2–20]“. In: *Journal of Discrete Algorithms* 26, S. 106–110.

Fulkerson, Delbert R. (1956). „Note on Dilworth’s Decomposition Theorem for Partially Ordered Sets“. In: *Proceedings of the American Mathematical Society* 7.4, S. 701–702.

Garey, M. R. und D. S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. First Edition. W. H. Freeman.

Garey, M. R., D. S. Johnson und Ravi Sethi (1976). „The Complexity of Flowshop and Jobshop Scheduling“. In: *Mathematics of Operations Research* 1.2, S. 117–129.

Gendreau, Michel und Jean-Yves Potvin, Hrsg. (2010). *Handbook of Metaheuristics*. International Series in Operations Research & Management Science. Springer US.

Geoffrion, Arthur M. (2010). „Lagrangian Relaxation for Integer Programming.“ In: *50 Years of Integer Programming*. Hrsg. von Michael Jünger u. a. Springer, S. 243–281.

Ghosh, Jay B. (1994). „Batch scheduling to minimize total completion time.“ In: *Oper. Res. Lett.* 16.5, S. 271–275.

Gilmore, P. C. und Ralph E. Gomory (1961). „A Linear Programming Approach to the Cutting-Stock Problem“. In: *Operations Research* 9.6, S. 849–859.

— (1963). „A Linear Programming Approach to the Cutting Stock Problem-Part II“. In: *Operation Research* 11, S. 863–859.

Glover, Fred (1989). „Tabu Search—Part I“. In: *ORSA Journal on Computing* 1.3, S. 190–206.

— (1990). „Tabu Search—Part II“. In: *ORSA Journal on Computing* 2.1, S. 4–32.

Goemans, Michel X. und David P. Williamson (2000). „Two-Dimensional Gantt Charts and a Scheduling Algorithm of Lawler“. In: *SIAM Journal on Discrete Mathematics* 13.3, S. 281–294.

Gomory, Ralph E. (2002). „Early Integer Programming“. In: *Operations Research* 50.1, S. 78–81.

Gordon, V. S. u. a. (2008). „Single machine scheduling models with deterioration and learning: handling precedence constraints via priority generation“. In: *Journal of Scheduling* 11.5, S. 357–370.

Gouveia, Luis und Jose Manuel Pires (2001). „The asymmetric travelling salesman problem: on generalizations of disaggregated Miller–Tucker–Zemlin constraints“. In: *Discrete Applied Mathematics* 112.1–3. Combinatorial Optimization Symposium, Selected Papers, S. 129–145.

Graham, R. L. u. a. (1979). „Optimization and approximation in deterministic sequencing and scheduling: a survey“. In: *Annals of discrete mathematics* 5.2, S. 287–326.

Grigoriev, Alexander, Martijn Holthuijsen und Joris van de Klundert (2005). „Basic scheduling problems with raw material constraints“. In: *Naval Research Logistics (NRL)* 52.6, S. 527–535.

Guignard, Monique und Kurt Spielberg (1981). „Logical Reduction Methods in Zero-One Programming—Minimal Preferred Variables“. In: *Operations Research* 29.1, S. 49–74.

Habib, Michel, Fabien de Montgolfier und Christophe Paul (2004). „A Simple Linear-Time Modular Decomposition Algorithm for Graphs, Using Order Extension“. In: *Algorithm Theory - SWAT 2004*. Hrsg. von Torben Hagerup und Jyrki Katajainen. Bd. 3111. Lecture Notes in Computer Science. Springer Berlin Heidelberg, S. 187–198.

Hall, Philip (1935). „On representatives of subsets“. In: *J. London Math. Soc* 10.1, S. 26–30.

Harary, Frank und Robert Z. Norman (1960). „Some properties of line digraphs“. In: *Rendiconti del Circolo Matematico di Palermo* 9.2, S. 161–168.

Hariri, A.M.A. und Chris N. Potts (1983). „An algorithm for single machine sequencing with release dates to minimize total weighted completion time“. In: *Discrete Applied Mathematics* 5.1, S. 99–109.

Hartmann, Sönke und Dirk Briskorn (2010). „A survey of variants and extensions of the resource-constrained project scheduling problem“. In: *European Journal of Operational Research* 207.1, S. 1–14.

Held, Michael und Richard M. Karp (1962). „A dynamic programming approach to sequencing problems“. In: *Journal of the Society for Industrial and Applied Mathematics*, S. 196–210.

— (1970). „The Traveling-Salesman Problem and Minimum Spanning Trees“. In: *Operations Research* 18.6, S. 1138–1162.

— (1971). „The traveling-salesman problem and minimum spanning trees: Part II“. In: *Mathematical Programming* 1.1, S. 6–25.

Held, Michael, Philip Wolfe und HarlanP. Crowder (1974). „Validation of subgradient optimization“. In: *Mathematical Programming* 6.1, S. 62–88.

Herlyn, Wilmjakob (2012). *PPS im Automobilbau: Produktionsprogrammplanung und -steuerung von Fahrzeugen und Aggregaten*. München: Hanser.

Herrmann, Jeffrey W. und Chung-Yee Lee (1995). „Solving a Class Scheduling Problem with a Genetic Algorithm“. In: *ORSA Journal on Computing* 7.4, S. 443–452.

Heydary, M. H. u. a. (2007). „Computing Cross Associations for Attack Graphs and other Applications“. In: *Proceedings of the 40th Hawaii International Conference on System Sciences*.

Hoogeveen, J. A. und Steef L. van de Velde (1995). „Stronger Lagrangian bounds by use of slack variables: Applications to machine scheduling problems“. In: *Mathematical Programming* 70.1-3, S. 173–190.

Hopcroft, John E. und Richard M. Karp (1973). „An n5/2 Algorithm for Maximum Matchings in Bipartite Graphs.“ In: *SIAM J. Comput.* 2.4, S. 225–231.

Horn, W. A. (1974). „Some simple scheduling algorithms“. In: *Naval Research Logistics Quarterly* 21.1, S. 177–185.

Ibaraki, Toshihide (1976). „Theoretical comparisons of search strategies in branch-and-bound algorithms“. In: *International Journal of Computer & Information Sciences* 5.4, S. 315–344.

— (1977). „The Power of Dominance Relations in Branch-and-Bound Algorithms“. In: *J. ACM* 24.2, S. 264–279.

Johnson, Ellis L. und Manfred W. Padberg (1982). „Degree-two Inequalities, Clique Facets, and Biperfect Graphs“. In: *Bonn Workshop on Combinatorial OptimizationBased on lectures presented at the IV. Bonn Workshop on Combinatorial Optimization, organised by the Institute of Operations Research and sponsored by the Deutsche Forschungsgemeinschaft through the Sonderforschungsbereich 21*. Hrsg. von Achim Bachem, Martin Grötschel und Bemhard Korte. Bd. 66. North-Holland Mathematics Studies. North-Holland, S. 169–187.

Johnson, S. M. (1954). „Optimal two- and three-stage production schedules with setup times included“. In: *Naval Research Logistics Quarterly* 1.1, S. 61–68.

Jungnickel, Dieter (2008). *Graphs, Networks and Algorithms*. Springer.

Kamburowski, Jerzy, David J. Michael und Matthias F. M. Stallmann (2000). „Minimizing the complexity of an activity network“. In: *Networks* 36.1, S. 47–52.

Kan, Alexander H. G. Rinnooy (1976). *Machine scheduling problems. classification, complexity and computations*. The Hague: Nijhoff. IX, 180.

Kan, Alexander H. G. Rinnooy, B. J. Lageweg und J. K. Lenstra (1975). „Minimizing Total Costs in One-Machine Scheduling“. In: *Operations Research* 23.5, S. 908–927.

Kao, Gio K. (2008). „Two Combinatorial Optimization Problems at the Interface of Computer Science and Operations Research“. Diss. University of Illinois.

Kao, Gio K., Edward C. Sewell und Sheldon H. Jacobson (2009). „A branch, bound, and remember algorithm for the $1|r_i|\sum t_i$ scheduling problem“. In: *Journal of Scheduling* 12.2, S. 163–175.

Karp, Richard M. (1972). „Reducibility among Combinatorial Problems“. In: *Complexity of Computer Computations*. Hrsg. von Raymond E. Miller, James W. Thatcher und Jean D. Bohlinger. The IBM Research Symposia Series. Springer US, S. 85–103.

Kennington, Jeff (2004). „Comments on “A Suggested Computation for Maximal Multi-Commodity Network Flows”“. In: *Management Science* 50.12_supplement, S. 1781–1781.

Khoo, Kei-Yong und Jason Cong (1992). „A Fast Multilayer General Area Router for MCM Designs“. In: *Proceedings of the Conference on European Design Automation*. EURO-DAC '92. Congress Centrum Hamburg, Hamburg, Germany: IEEE Computer Society Press, S. 292–297.

Kohler, Walter H. und Kenneth Steiglitz (1974). „Characterization and Theoretical Comparison of Branch-and-Bound Algorithms for Permutation Problems“. In: *J. ACM* 21.1, S. 140–156.

Krishnamoorthy, M. S. und N. Deo (1979). „Complexity of the minimum-dummy-activities problem in a pert network“. In: *Networks* 9.3, S. 189–194.

Kuhn, Harold W. (1955). „The Hungarian Method for the Assignment Problem“. In: *Naval Research Logistics Quarterly* 2.1–2, S. 83–97.

Labetoulle, J. u. a. (1984). „Preemptive Scheduling of Uniform Machines Subject to Release Dates“. In: *Progress in Combinatorial Optimization*. Hrsg. von William R. Pulleyblank. Academic Press, S. 245–261.

Land, A. H. und A. G. Doig (1960). „An Automatic Method of Solving Discrete Programming Problems“. In: *Econometrica* 28.3, S. 497–520.

Laporte, Gilbert (1992a). „The traveling salesman problem: An overview of exact and approximate algorithms“. In: *European Journal of Operational Research* 59.2, S. 231–247.

— (1992b). „The vehicle routing problem: An overview of exact and approximate algorithms“. In: *European Journal of Operational Research* 59.3, S. 345–358.

Lasdon, Leon S. (1974). *Optimization theory for large systems*. Nachdr. Macmillan series in operations research. New York, NY: Macmillan u.a.

Lawler, Eugene L. (1973). „Optimal Sequencing of a Single Machine Subject to Precedence Constraints“. In: *Management Science* 19.5, S. 544–546.

— (1977). „A “Pseudopolynomial” Algorithm for Sequencing Jobs to Minimize Total Tardiness*“. In: *Studies in Integer Programming*. Hrsg. von P. L. Hammer u. a. Bd. 1. Annals of Discrete Mathematics. Elsevier, S. 331–342.

— (1978). „Sequencing Jobs to Minimize Total Weighted Completion Time Subject to Precedence Constraints“. In: *Algorithmic Aspects of Combinatorics*.

Hrsg. von P. Hell B. Alspach und D. J. Miller. Bd. 2. Annals of Discrete Mathematics. Elsevier, S. 75–90.

Lawler, Eugene L. (1992). „On Preference Orders for Sequencing Problems Or, What Hath Smith Wrought?“ In: *Combinatorial Optimization*. Hrsg. von Mustafa Akgül, HorstW. Hamacher und Süleyman Tüfekçi. Bd. 82. NATO ASI Series. Springer Berlin Heidelberg, S. 133–159.

Lawler, Eugene L. und J. M. Moore (1969). „A Functional Equation and Its Application to Resource Allocation and Sequencing Problems“. In: *Management Science* 16.1, S. 77–84.

Lengauer, Thomas und Robert Endre Tarjan (1979). „A Fast Algorithm for Finding Dominators in a Flowgraph“. In: *ACM Trans. Program. Lang. Syst.* 1.1, S. 121–141.

Lenstra, J. K. und Alexander H. G. Rinnooy Kan (1978). „Complexity of Scheduling under Precedence Constraints“. In: *Operations Research* 26.1, S. 22–35.

Lenstra, J. K., Alexander H. G. Rinnooy Kan und Peter Brucker (1977). „Complexity of Machine Scheduling Problems“. In: *Studies in Integer Programming*. Hrsg. von E.L. Johnson B.H. Korte P.L. Hammer und G.L. Nemhauser. Bd. 1. Annals of Discrete Mathematics. Elsevier, S. 343–362.

Liaee, M. M. und H. Emmons (1997). „Scheduling families of jobs with setup times“. In: *Int. J. Production Economics* 51, S. 165–176.

Little, John DC u. a. (1963). „An algorithm for the traveling salesman problem“. In: *Operations research* 11.6, S. 972–989.

Lovász, L. und M.D. Plummer (2009). *Matching Theory*. AMS Chelsea Publishing Series. AMS Chelsea Pub.

Maffioli, Francesco und Anna Sciomachen (1997). „A mixed-integer model for solving ordering problems with side constraints“. In: *Annals of Operations Research* 69, S. 277–297.

Manne, Alan S. (1960). „On the Job-Shop Scheduling Problem“. In: *Operations Research* 8.2, S. 219–223.

Mason, A. J. und E. J. Anderson (1991). „Minimizing flow time on a single machine with job classes and setup times“. In: *Naval Research Logistics (NRL)* 38.3, S. 333–350.

Matsumoto, Makoto und Takuji Nishimura (1998). „Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator“. In: *ACM Trans. Model. Comput. Simul.* 8.1, S. 3–30.

McConnell, Ross M. und Jeremy P. Spinrad (1997). „Linear-time Transitive Orientation“. In: *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '97. New Orleans, Louisiana, USA: Society for Industrial und Applied Mathematics, S. 19–25.

McMahon, Graham B. und Chong-John Lim (1993). „The two-machine flow shop problem with arbitrary precedence relations“. In: *European Journal of Operational Research* 64.2. Project Management anf Scheduling, S. 249–257.

Mehlhorn, Kurt und Peter Sanders (2008). *Algorithms and Data Structures: The Basic Toolbox*. Berlin: Springer.

Merle, Olivier du u. a. (1999). „Stabilized column generation.“ In: *Discrete Mathematics* 194.1-3, S. 229–237.

Möhring, Rolf H. (1985). „Algorithmic Aspects of Comparability Graphs and Interval Graphs“. In: *Graphs and Order*. Hrsg. von Ivan Rival. Bd. 147. NATO ASI Series. Springer Netherlands, S. 41–101.

Möhring, Rolf H. (1989). „Computationally Tractable Classes of Ordered Sets“. In: *Algorithms and Order*. Hrsg. von Ivan Rival. Bd. 255. NATO ASI Series. Springer Netherlands, S. 105–193.

Monma, Clyde L. (1979). „The Two-Machine Maximum Flow Time Problem with Series-Parallel Precedence Constraints: An Algorithm and Extensions“. In: *Operations Research* 27.4, S. 792–798.

Monma, Clyde L. und Chris N. Potts (1989). „On the Complexity of Scheduling with Batch Setup Times“. In: *Operations Research* 37.5, S. 798–804.

Monma, Clyde L. und Jeffrey B. Sidney (1979). „Sequencing with Series-Parallel Precedence Constraints“. In: *Mathematics of Operations Research* 4.3, S. 215–224.

— (1987). „Optimal Sequencing via Modular Decomposition: Characterization of Sequencing Functions“. In: *Mathematics of Operations Research* 12.1, S. 22–31.

Munkres, James R. (1957). „Algorithms for the Assignment and Transportation Problems“. In: *Journal of the Society for Industrial and Applied Mathematics* 5.1, S. 32–38.

Murty, Katta G (2010). Bd. 137. International Series in Operations Research & Management Science. Springer US.

Nemhauser, G. L. und L. E. Trotter (1974). „Properties of vertex packing and independence system polyhedra“. In: *Mathematical Programming* 6.1, S. 48–61.

— (1975). „Vertex packings: Structural properties and algorithms“. In: *Mathematical Programming* 8.1, S. 232–248.

Nessah, Rabia und Imed Kacem (2012). „Branch-and-bound method for minimizing the weighted completion time scheduling problem on a single machine with release dates“. In: *Computers & Operations Research* 39.3, S. 471–478.

Nogueira, Thiago Henrique, CRV de Carvalho und Martın Gómez Ravetti (2014). „Analysis of mixed integer programming formulations for single machine scheduling problems with sequence dependent setup times and release dates“. In:

Nuutila, Esko (1995). „Efficient transitive closure computation in large digraphs“. Diss. PhD thesis, Helsinki University of Technology, 1995. Acta Polytechnica Scandinavica, Mathematics und Computing in Engineering Series.

Paes Leme, Renato und David B. Shmoys. „ A simpler Primal-Dual Proof of Lawler's Algorithm“.

Pan, Yunpeng (2003). „An improved branch and bound algorithm for single machine scheduling with deadlines to minimize total weighted completion time“. In: *Operations Research Letters* 31.6, S. 492–496.

Pan, Yunpeng und L. Shi (2005). „Dual constrained single machine sequencing to minimize total weighted completion time“. In: *Automation Science and Engineering, IEEE Transactions on* 2.4, S. 344–357.

Papadimitriou, Christos H. und Kenneth Steiglitz (1998). *Combinatorial Optimization : Algorithms and Complexity*. Dover Publications.

Parker, R. Garey (1995). *Deterministic Scheduling Theory*. Chapman & Hall.

Peeters, René (2003). „The maximum edge biclique problem is NP-complete“. In: *Discrete Applied Mathematics* 131.3, S. 651–654.

Peter, Markus und Georg Wambach (2000). „N-extendible posets, and how to minimize total weighted completion time“. In: *Discrete Applied Mathematics* 99.1–3, S. 157–167.

Peterson, W. W. und D. T. Brown (1961). „Cyclic Codes for Error Detection“. In: *Proceedings of the IRE* 49.1, S. 228–235.

Pijls, Wim und Rob Pothars (2013). „Another Note on Dilworth's Decomposition Theorem“. In: *Journal of Discrete Mathematics*.

Pinedo, M. (2008). *Scheduling : theory, algorithms, and systems*. 3. Aufl. Prentice-Hall, Englewood Cliffs.

Posner, Marc E. (1985). „Minimizing Weighted Completion Times with Deadlines“. In: *Operations Research* 33.3, S. 562–574.

Potts, Chris N. (1980). „An algorithm for the single machine sequencing problem with precedence constraints“. In: *Combinatorial Optimization II*. Hrsg. von V. J. Rayward-Smith. Berlin, Heidelberg: Springer Berlin Heidelberg, S. 78–87.

— (1985). „A Lagrangean Based Branch and Bound Algorithm for Single Machine Sequencing with Precedence Constraints to Minimize Total Weighted Completion Time“. In: *Management Science* 31.10, S. 1300–1311.

Potts, Chris N. und Mikhail Y. Kovalyov (2000). „Scheduling with batching: A review.“ In: *European Journal of Operational Research* 120.2, S. 228–249.

Potts, Chris N. und Luk N. Van Wassenhove (1983). „An algorithm for single machine sequencing with deadlines to minimize total weighted completion time“. In: *European Journal of Operational Research* 12.4, S. 379–387.

— (1985). „A Branch and Bound Algorithm for the Total Weighted Tardiness Problem“. In: *Operations Research* 33.2, S. 363–377.

— (1992). „Integrating Scheduling with Batching and Lot-Sizing: A Review of Algorithms and Complexity“. In: *The Journal of the Operational Research Society* 43.5, pp. 395–406.

Reyck, Bert De und Willy S. Herroelen (1996). „On the use of the complexity index as a measure of complexity in activity networks“. In: *European Journal of Operational Research* 91.2, S. 347–366.

Rocha, Ana Maria, João Soares und Edite M. G. P. Fern (2005). „Solving the Traveling Repairman Problem with differentiated waiting times through lagrangian relaxation“. In:

Roshanaei, V., Ahmed Azab und H. ElMaraghy (2013). „Mathematical modelling and a meta-heuristic for flexible job shop scheduling.“ In: *International Journal of Production Research* 51.20, S. 6247–6274.

Rusu, Irena (2008). „Maximum weight edge-constrained matchings“. In: *Discrete Applied Mathematics* 156.5, S. 662–672.

Ryan, David M und Brian A Foster (1981). „An integer programming approach to scheduling“. In: *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling : Papers Based on Presentations at the International Workshop Held at the University of Leeds, 16-18 July, 1980*. Hrsg. von A. Wren. Computer scheduling of public transport. North-Holland, Amsterdam, S. 269–280.

Savelsbergh, Martin W. P. (1994). „Preprocessing and Probing Techniques for Mixed Integer Programming Problems“. In: *ORSA Journal on Computing* 6.4, S. 445–454.

Schnorr, C. P. (1978). „An Algorithm for Transitive Closure with Linear Expected Time“. In: *SIAM Journal on Computing* 7.2, S. 127–133.

Scholl, Armin und Robert Klein (1997). „SALOME: A Bidirectional Branch-and-Bound Procedure for Assembly Line Balancing.“ In: *INFORMS Journal on Computing* 9.4, S. 319–334.

Schrage, Linus E. (1968). „A Proof of the Optimality of the Shortest Remaining Processing Time Discipline“. In: *Operations Research* 16.3, S. 687–690.

Schrage, Linus E. und Louis W. Miller (1966). „The Queue M/G/1 with the Shortest Remaining Processing Time Discipline“. In: *Operations Research* 14.4, S. 670–684.

Sewell, Edward C. und Sheldon H. Jacobson (2012). „A Branch, Bound, and Remember Algorithm for the Simple Assembly Line Balancing Problem“. In: *INFORMS Journal on Computing* 24.3, S. 433–442.

Sewell, Edward C., Jason J. Sauppe u. a. (2012). „A BB&R algorithm for minimizing total tardiness on a single machine with sequence dependent setup times“. In: *Journal of Global Optimization* 54.4, S. 791–812.

Shirzadeh Chaleshtarti, A., S. Shadrokh und Y. Fathi (2014). „Branch and Bound Algorithms for Resource Constrained Project Scheduling Problem Subject to Nonrenewable Resources with Prescheduled Procurement“. In: *Mathematical Problems in Engineering*.

Sidney, Jeffrey B. (1975). „Decomposition Algorithms for Single-Machine Sequencing with Precedence Relations and Deferral Costs“. In: *Operations Research* 23.2, S. 283–298.

— (1979). „The Two-Machine Maximum Flow Time Problem with Series Parallel Precedence Relations“. In: *Operations Research* 27.4, S. 782–791.

Sidney, Jeffrey B. und George Steiner (1986). „Optimal Sequencing by Modular Decomposition: Polynomial Algorithms“. In: *Operations Research* 34.4, S. 606–612.

Skiena, Steven (2008). *The Algorithm Design Manual (2. ed.)*. Springer, S. I–XVI, 1–730.

Smith, Wayne E. (1956). „Various optimizers for single-stage production“. In: *Naval Research Logistics Quarterly* 3.1-2, S. 59–66.

Sousa, Jorge P. und Laurence A. Wolsey (1992). „A time indexed formulation of non-preemptive single machine scheduling problems“. In: *Mathematical Programming* 54.1, S. 353–367.

Steiner, George (1990). „On the complexity of dynamic programming for sequencing problems with precedence constraints“. In: *Annals of Operations Research* 26.1, S. 103–123.

— (1992). „Finding the largest suborder of fixed width“. In: *Order* 9.4, S. 357–360.

Stockmeyer, Larry J. und Vijay V. Vazirani (1982). „NP-completeness of some generalizations of the maximum matching problem“. In: *Information Processing Letters* 15.1, S. 14–19.

Taillard, Éric u. a. (1997). „A Tabu Search Heuristic for the Vehicle Routing Problem with Soft time Windows.“ In: *Transportation Science* 31.2, S. 170.

Tavares, L. V. (2002). „A review of the contribution of Operational Research to Project Management“. In: *European Journal of Operational Research* 136.1, S. 1–18.

Tempelmeier, Horst (2006). *Material-Logistik: Modelle und Algorithmen für die Produktionsplanung und -steuerung in Advanced-Planning-Systemen*. 6., neubearb. Aufl. Berlin u.a.: Springer.

Valdes, Jacobo (1978). *Parsing flowcharts and series-parallel graphs*. Techn. Ber. Stanford: Standford University.

Valdes, Jacobo, Robert Endre Tarjan und Eugene L. Lawler (1982). „The Recognition of Series Parallel Digraphs“. In: *SIAM Journal on Computing* 11.2, S. 298–313.

Vanderbeck, François und Laurence A. Wolsey (1996). „An exact algorithm for IP column generation“. In: *Operations Research Letters* 19.4, S. 151–159.

Velde, Steef L. van de (1995). „Dual decomposition of a single-machine scheduling problem“. In: *Mathematical Programming* 69.1, S. 413–428.

Volgenant, Ton und Roy Jonker (1982). „A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation“. In: *European Journal of Operational Research* 9.1, S. 83–89.

Vuillemin, Jean (1978). „A Data Structure for Manipulating Priority Queues“. In: *Communications of the ACM* 21.4, S. 309–315.

Wagner, Harvey M. (1959). „An integer linear-programming model for machine scheduling“. In: *Naval Research Logistics Quarterly* 6.2, S. 131–140.

Warshall, Stephen (1962). „A Theorem on Boolean Matrices.“ In: *J. ACM* 9.1, S. 11–12.

Wassenhove, Luk N. Van (1979). „Special-purpose algorithms for one-mchine sequencing problems with single and composite objecitves“. Diss. Katholieke Universiteit Leuven.

Webster, Scott und Kenneth R. Baker (1995). „Scheduling Groups of Jobs on a Single Machine“. In: *Operations Research* 43.4, pp. 692–703.

Wiendahl, Hans-Peter (2004). *Variantenbeherrschung in der Montage: Konzept und Praxis der flexiblen Produktionsendstufe*. Engineering online library; VDI. Berlin u.a.: Springer.

Wolsey, Laurence A. (1998). *Integer Programming*. John Wiley & Sons, New York.

Yannakakis, Mihalis (1982). „The Complexity of the Partial Order Dimension Problem“. In: *SIAM Journal on Algebraic Discrete Methods* 3.3, S. 351–358.

Zhang, Yun u. a. (2014). „On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types“. In: *BMC Bioinformatics* 15.1, S. 1–18.

Zhu, Xiaoyan und Wilbert E. Wilhelm (2006). „Scheduling and lot sizing with sequence-dependent setup: A literature review“. In: *IIE Transactions* 38.11, S. 987–1007.

Abkürzungen und Symbole

C	Kette von Aufträgen
$\mathcal{C}$	Partitionierung der Menge J in Ketten
$C_j(*)$	Fertigstellungszeit des Auftrags $j \in J$ in Abhängigkeit von $*$
DP	Dynamische Programmierung
G	Materialgruppe
$\mathcal{G}$	Menge der Materialgruppen
ι	Inklusionsmuster
$I(\sigma)$	Zerlegung des Schedules σ in eine Menge von Inklusionsmustern
IP	Ganzzahlige Programmierung
J	Menge der Aufträge
LB	Untere Schranke
LP	Lineare Programmierung
LPM	LP-Relaxation des Masterproblems
M	Menge der Materialien
MP	Masterproblem
ν	Knoten im Branch-and-Bound-Baum
$p^{\max}(j)$	Maximale Position des Auftrags $j \in J$ in einem Schedule
$\pi_{i,j}$	Binärvariable für die Einplanung von Auftrag $i \in J$ vor Auftrag $j \in J$
p_j	Fertigungszeit für Auftrag $j \in J$
P	Menge der Produkte
RLPM	Auf eine Teilmenge aller Spalten restringierte Version des Masterproblems
$\rho_{j,\tau}$	Binärvariable für die Auswahl des Zeitpunkts $\tau \in T$ als Zeitpunkt der Materialverfügbarkeit für Auftrag $j \in J$
$R_{m,\tau}$	Gesamte gelieferte Menge von Material $m \in M$ bis zum Zeitpunkt $\tau \in T$
$a_{j,m}$	Verbrauch von Material $m \in M$ durch Auftrag $j \in J$
$\mathcal{R}$	Indexmenge der Zeilen im Masterproblem
σ	Schedule
$st(*)$	Rüstzeit in Abhängigkeit von $*$
$\mathcal{T}$	Indexmenge der Spalten im Masterproblem
T	Indexmenge der Materialanlieferungszeitpunkte
UB	Obere Schranke
$\prec$	Vorrangrelation