Inhaltsverzeichnis

1 Einleitung

Bei synchronen Flow-Shop-Problemen handelt es sich um Produktionsplanungsprobleme, bei denen die Jobs sich immer nur synchron zur nächsten Maschine bewegen können. Im Gegensatz zum klassischen (asynchronen) Flow-Shop, bei dem die Jobs immer sofort nach Fertigstellung an einer Maschine zur nächsten gelangen können, müssen die Jobs beim synchronen Flow-Shop so lange warten, bis auch alle anderen Maschinen mit der Bearbeitung ihres jeweiligen Jobs fertig sind. Die Komplexität synchroner Flow-Shop-Probleme wurde zuerst von Karabati und Sayin [KS03] diskutiert. Soylu et al. [SKA07] stellten einen Branch-and-Bound-Algorithmus vor, der die gesamte Fertigstellungsdauer minimiert, und liefern einen (fehlerhaften) Beweis, dass dieses Problem mit drei Maschinen $\mathcal{NP}$-schwer ist. Waldherr und Knust korrigieren diesen Beweis in einem bis zum Zeitpunkt dieser Masterarbeit noch nicht veröffentlichten Paper [WK]. Darüber hinaus betrachten sie einige Spezialfälle des synchronen Flow-Shops mit einer geringen Anzahl an dominierenden Maschinen. Dominierende Maschinen zeichnen sich dadurch aus, dass ihre Prozesszeiten für alle Jobs mindestens so groß sind wie bei den übrigen Maschinen.

Das Konzept der dominierenden Maschinen aus der Arbeit von Waldherr und Knust basiert auf einer konkreten praktischen Anwendung von synchronen Flow-Shop-Problemen. Es handelt sich dabei um einen in Bad Essen ansässigen Küchenhersteller. In einer Fallstudie [WK14] haben Waldherr und Knust das bei diesem Küchenhersteller gegebene Optimierungsproblem genauer untersucht. Es handelt sich dabei um einen synchronen Flow-Shop, wobei die zu produzierenden Jobs Ressourcen benötigen und die Produktionsanlage beim Einlegen eines Jobs ggf. umgerüstet werden muss, was Rüstkosten verursacht. U.a. durch dieses Praxisbeispiel ist auch diese Masterarbeit motiviert.

1.1 Problembeschreibung

Die Produktionsanlage besteht aus m Stationen, in die die Jobs eingelegt werden. Die Anlage bewegt die Stationen und die in ihnen befindlichen Jobs der Reihenfolge nach zu den Maschinen $M_1, \ldots, M_m$, die außen, um die Anlage herum, aufgestellt sind und die einzelnen Produktionsschritte durchführen. Dabei handelt es sich bei Maschine M_1 um das Einlegen des Jobs in die Anlage und nach der letzten Maschine M_m wird das fertige Produkt entnommen. Die Stationen stehen derart miteinander in Verbindung, dass sie sich immer nur gemeinsam fortbewegen können. Die Anlage kann beispielsweise

kreisförmig aufgebaut sein, wie in Abbildung 1.1 dargestellt. Aus diesem Grund wird die Fortbewegung der Stationen mit den Jobs zur jeweils nächsten Maschine als *Rotation* bezeichnet, auch wenn die Anlage nicht zwangsweise zyklische Gestalt haben muss. Eine

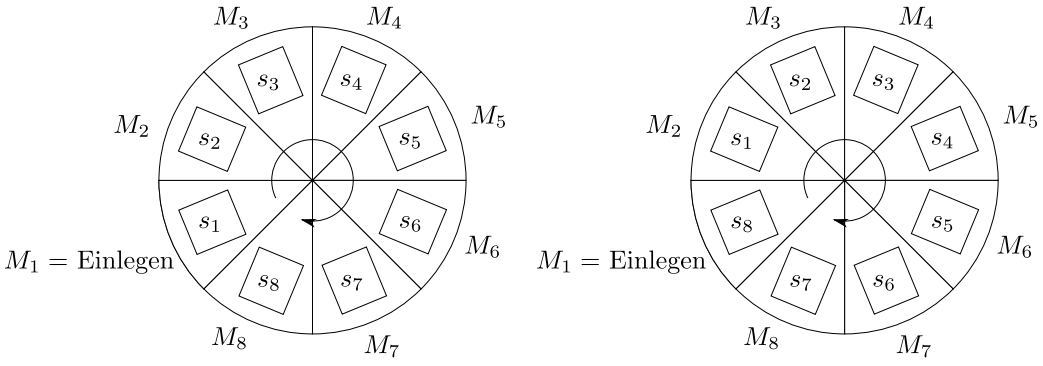

Abbildung 1.1: Kreisförmige Anlage mit $m = 8$ Maschinen und Stationen, rechts nach einer Rotation. Die Stationen, in die die Jobs eingelegt werden, drehen sich mit der Anlage zu den außen herum aufgestellten Maschinen.

Rotation darf immer nur dann stattfinden, wenn alle Maschinen ihren Produktionsschritt an ihrem aktuellen Job durchgeführt haben. Auf diese Weise können die Jobs, im Gegensatz zum klassischen (asynchronen) Flow-Shop, immer nur *synchron* zur nachfolgenden Maschine gelangen. Die Zeit, die zwischen zwei Rotationen vergeht, wird als *Zykluszeit* bezeichnet.

Die zu produzierenden Jobs sind gegeben durch die Menge $N = \{1, \ldots, n\}$ und die Prozesszeiten von Job j auf Maschine M_i sind durch p_{ij} gegeben. Ein Plan für eine Beispielinstanz mit $n = 5$ und $m = 3$ ist in Abbildung 1.2 zu sehen. Ziel ist es, eine

	1	2	3	4	5
M_1	4	1	2	5	4
M_2	6	5	5	2	5
M_3	5	6	4	4	4

Abbildung 1.2: Beispielinstanz mit 5 Jobs und 3 Maschinen. Die Werte in der Tabelle sind die Prozesszeiten p_{ij}.

Permutation π der Jobs zu erstellen, die die gesamte Produktionsdauer minimiert. Diese Zielfunktion wird mit $C_{\max}$ bezeichnet. Die Beispielinstanz, deren Plan in Abbildung 1.2 zu sehen ist, ist in Abbildung 1.3 als Gantt-Diagramm aufgetragen. Oben sind die Jobs in der initialen Reihenfolge (1-2-3-4-5) und unten in einer bezüglich $C_{\max}$ optimalen Reihenfolge (2-1-5-3-4). Die Zykluszeiten c_t mit $1 \leq t \leq n + m - 1$ berechnen sich wie

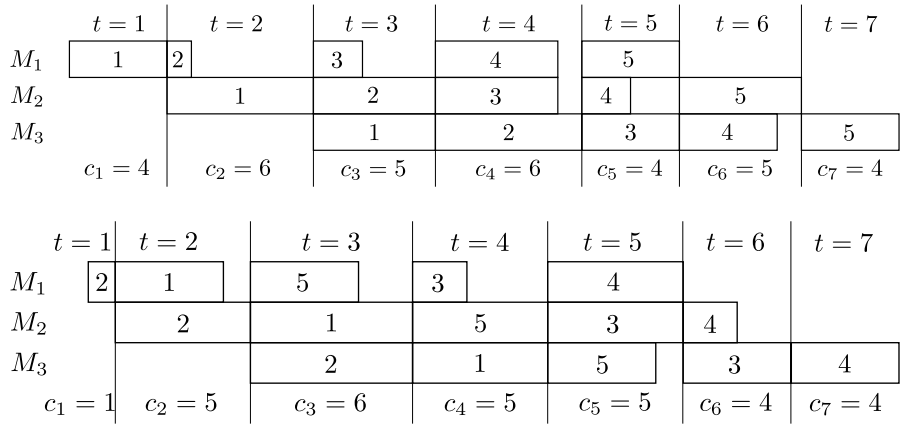

Abbildung 1.3: Gantt-Diagramme der initialen Reihenfolge der Beispielinstanz in Abb. 1.2 mit $C_{\max} = 34$ und einer optimalen Reihenfolge mit $C_{\max} = 30$.

folgt:

$$c_t = \max_{i=\max\{1,t-n+1\}}^{\min\{t,m\}} p_{i\pi_{t-i+1}}$$

Die Zielfunktion lässt sich also durch $C_{\max} = \sum_{t=1}^{n+m-1} c_t$ berechnen. Andere Zielfunktionen werden in dieser Masterarbeit nicht betrachtet.

Eine Teilmenge $D \subseteq \{1, \ldots, m\}$ der Maschinen heißt *dominierend*, wenn

$$p_{dj} \geq p_{ej} \quad \forall j \in N, d \in D, e \notin D$$

ist. Die Prozesszeiten aller Jobs auf dominierenden Maschinen sind also immer mindestens so groß wie die Prozesszeiten auf den restlichen Maschinen. Treten dominierende Maschinen auf, müssen für die Berechnung der Zykluszeiten die Prozesszeiten auf den übrigen Maschinen also nicht betrachtet werden. Dominierende Maschinen treten in der Praxis häufig auf. Bei dem Küchenhersteller aus Bad Essen sind z.B. zwei von acht Maschinen dominierend. Diese beiden Maschinen sind dort außerdem benachbart, weshalb dieser Fall in dieser Masterarbeit besonders intensiv behandelt wird.

Zusätzlich benötigen die Jobs unter Umständen Ressourcen (z.B. Paletten oder Klebeformen) aus einer Menge R, um in die Stationen eingelegt werden zu können. Diese Ressourcen können erst nach Fertigstellung eines Jobs, also nachdem er nach Maschine M_m aus der Anlage genommen wurde, wiederverwendet werden. Sie sind allerdings nur in begrenzter Zahl vorhanden und im Allgemeinen ist nicht jede Ressource für jeden Job geeignet. Für $j \in N$ sei $\rho_j \subseteq R$ die Menge der Ressourcen, die für j geeignet ist. Umgekehrt sei für $r \in R$ mit $\iota_r \subseteq N$ die Menge der Jobs bezeichnet, für die r geeignet ist. Wenn eine Station nach der Entnahme eines Jobs nach Maschine M_m wieder an Maschine M_1 angelangt, kann es daher notwendig sein, vor dem Einlegen des nächsten

Jobs die Ressource zu wechseln, wenn auf der Station zuvor Job $j \in N$ mit Ressource $r \in \rho_j$ fertiggestellt wurde und nun Job $j' \notin \iota_r$ eingelegt werden soll.

Für die Ressourcen können folgende Situationen auftreten:

1. Alle Ressourcen sind für alle Jobs geeignet, also $\rho_j = R$ für alle $j \in N$.

2. Die Jobs lassen sich in disjunkte Gruppen unterteilen, so dass für alle Jobs aus einer Gruppe dieselbe Ressourcenmenge geeignet ist. Wenn also $\rho_i \cap \rho_j \neq \emptyset$, dann folgt $\rho_i = \rho_j$.

3. Die Ressourcenmengen bilden Hierarchien. D.h., wenn $\iota_q \cap \iota_r \neq \emptyset$, dann folgt $\iota_q \subseteq \iota_r$ oder $\iota_r \subseteq \iota_q$.

4. Die ρ_j sind beliebige Teilmengen von R.

Der erste Fall ist trivial, da niemals eine Ressource gewechselt werden muss. Dies entspricht der Situation ohne Ressourcen und Rüstkosten. In dieser Masterarbeit liegt der Fokus auf dem zweiten Fall. Er ist dadurch motiviert, dass es in realen Anwendungsfällen häufig nicht der Fall ist, dass bei n zu produzierenden Jobs alle n Jobs unterschiedlich sind, d.h. unterschiedliche Prozesszeiten und Ressourcen haben. Stattdessen bietet ein Produzent eine gewisse Anzahl unterschiedlicher Güter an und ein Käufer gibt mehrere identische Güter auf einmal in Auftrag. Diese identischen Jobs können dann als *Jobgruppen* bezeichnet werden, für die es eine Menge an nur für sie geeigneten Ressourcen gibt. Der dritte Fall ist vom Prinzip her vergleichbar mit einem Generalschlüssel, der in mehrere Schlüssellöcher passt, wobei es für einzelne Schlüssellöcher auch speziell nur dafür passende Schlüssel gibt, die in die anderen Schlüssellöcher nicht passen. Ressourcen können in ähnlicher Weise durch geringe Unterschiede für viele Jobs auf einmal oder nur für einzelne geeignet sein. Der vierte Fall ist der allgemeinste und deckt alle anderen mit ab. Er wird in dieser Arbeit nur nur kurz betrachtet (vgl. Abschnitt 2.2.1).

Neben dem Wechsel von Ressourcen, der zusätzliche Zeit in Anspruch nimmt, können auch andere Formen von *Rüstkosten* auftreten. Z.B. kann es sein, dass an einer Station zunächst einige Umstellungen vorgenommen werden müssen, bevor der neue Job eingelegt werden kann. Die Jobs können in Familien $\mathcal{F}$ eingeteilt werden, so dass beim Übergang zwischen zwei Jobs aus den Familien f und g die Rüstkosten s_{fg} auftreten. Diese Rüstkosten können

- sowohl vom Vorgänger als auch vom Nachfolger abhängig sein (s_{fg}),
- nur vom Nachfolger abhängig sein ($s_{fg} = s_g$) oder
- konstant sein ($s_{fg} = s > 0$).

Dabei wird $s_{ff} = 0$ angenommen für alle $f \in \mathcal{F}$, es treten also keine Rüstkosten inner-

	1	2	3	4	5
1	0	8	5	9	4
2	6	0	6	8	8
3	4	7	0	5	4
4	5	7	8	0	9
5	4	5	8	9	0

Abbildung 1.4: Rüstkosten für die Beispielinstanz aus Tabelle 1.2. Beispielsweise treten Rüstkosten von 6 auf, wenn von Job 2 auf Job 1 gewechselt werden muss.

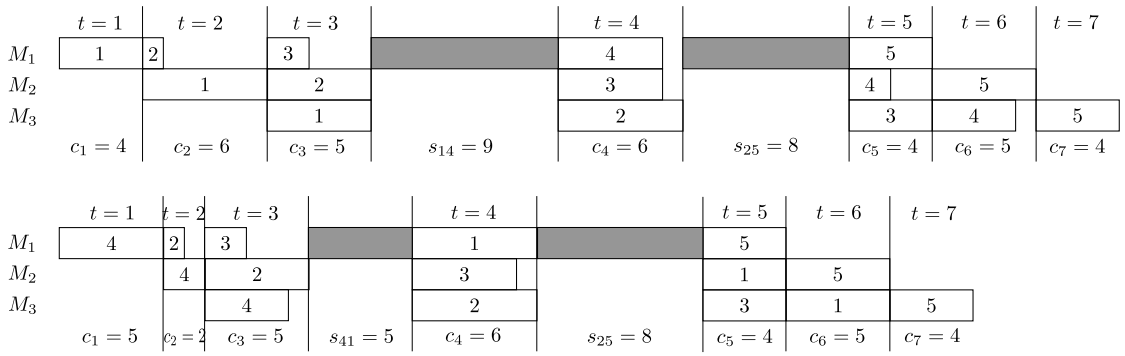

Abbildung 1.5: Initiale und eine optimale Lösung als Gantt-Diagramme für das Beispiel aus Abb. 1.2 erweitert um Rüstkosten aus Abb. 1.4.

halb einer Familie auf. Wenn Ressourcen notwendig sind, können die Familien über die Ressourcen definiert werden: Wenn $\rho_i = \rho_j$, dann gehören i und j zur gleichen Familie.

Wenn Rüstkosten auftreten, soll nicht mehr nur die Summe aller Zykluszeiten minimiert werden, sondern zusätzlich noch die Summe aller auftretenden Rüstkosten. Die Zielfunktion $C_{\max}$ lässt sich dann als

$$\min \left(\sum_{t=1}^{n+m-1} c_t + \sum_{i=1}^{n-m} s_{i,i+m} \right)$$

formulieren, wobei hier die Rüstkosten direkt mit den Jobs statt mit den Familien indiziert sind.

In Abbildung 1.4 ist eine mögliche Rüstkostentabelle für das Beispiel aus Abbildung 1.2 gegeben. Die initiale Reihenfolge (1-2-3-4-5) und eine optimale (4-2-3-1-5) sind in Abbildung 1.5 als Gantt-Diagramm dargestellt.

Insgesamt gilt es, neben der Reihenfolge π auch ein Mapping $f : N \to R$ zu finden, das jedem Job $j \in N$ eine Ressource $r \in \rho_j$ zuweist und folgenden Ansprüchen genügt:

- f muss zulässig sein in dem Sinne, dass beim Einlegen jedes Jobs $j \in N$ eine Ressource $r \in \rho_j$ verfügbar ist (d.h., dass r sich nicht gerade an anderer Stelle in der Anlage befindet).

- f und π zusammen sollen optimal sein in dem Sinne, dass die Summe aus den durch π und f definierten Zykluszeiten und Rüstkosten minimal ist.

Im Folgenden wird ein Mixed Integer Linear Program (MIP) vorgestellt, dass das Gesamtproblem löst, allerdings ohne die Verfügbarkeit von Ressourcen zu beachten. Dabei ist N die Indexmenge der Jobs, also $N = \{1, \ldots, n\}$ und $T = \{1, \ldots, n + m - 1\}$ ist die Indexmenge der Zykluszeiten. Die Binärvariablen x_{jk} geben für $j, k \in N$ an, an welcher Position k in π sich Job j befindet. Es gilt $x_{jk} = 1$ genau dann, wenn j an Position k ist ($\pi_k = j$). Die Binärvariablen y_{jh} ($j, h \in N$) haben genau dann den Wert 1, wenn Job j genau m Positionen vor Job h in π liegt ($\exists i : \pi_{i-m} = j \wedge \pi_i = h$). Damit können in der Zielfunktion die Rüstkosten s_{jh} aufsummiert werden, die beim Übergang von j zu h auftreten:

$$\min \quad \sum_{t=1}^{n+m-1} c_t + \sum_{j=1}^{n} \sum_{h=1}^{n} s_{jh} y_{jh} \tag{1.1}$$

$$\text{s.t.} \quad \sum_{k=1}^{n} x_{jk} = 1 \qquad j \in N \tag{1.2}$$

$$\sum_{j=1}^{n} x_{jk} = 1 \qquad k \in N \tag{1.3}$$

$$c_t \geq \sum_{j=1}^{n} p_{t-k+1,j} \cdot x_{jk} \quad t \in T, k = \max\{1, t-m+1\}, \ldots, \min\{n, t\} \tag{1.4}$$

$$y_{jh} + 1 \geq x_{j,k-m} + x_{hk} \qquad j, h \in N, k = m+1, \ldots, n \tag{1.5}$$

$$c_t \geq 0 \qquad t \in T \tag{1.6}$$

$$x_{jk} \in \{0, 1\} \qquad j, k \in N \tag{1.7}$$

$$y_{jh} \in \{0, 1\} \qquad j, h \in N \tag{1.8}$$

Die Nebenbedingungen (1.2) und (1.3) stellen sicher, dass sich jeder Job an genau einer Position in π befindet und dass sich an jeder Position genau ein Job befindet. In Nebenbedingung (1.4) werden die Zykluszeiten bestimmt. In (1.5) werden die y_{jh} auf 1 gesetzt, wenn $x_{j,k-m}$ und x_{hk} auf 1 gesetzt sind. In der Zielfunktion (1.1) wird die Summe aus allen Zykluszeiten gebildet und die Summe aller auftretenden Rüstkosten s_{jh}.

Obwohl in diesem MIP nicht die Verfügbarkeiten von Ressourcen beachtet werden, benötigt es schon bei $n = 20$ mehrere Stunden zum Finden der optimalen Lösung. Es ist also für Instanzen mit mehreren Tausend Jobs nicht geeignet. Aufgrund dieser Tatsache ist eine heuristische Herangehensweise an synchrone Flow-Shop-Probleme mit Ressourcen und Rüstkosten eine gute Alternative.

1.2 Motivation für Dekompositionsansätze

In [WK] wird gezeigt, dass synchrone Flow-Shop-Probleme ohne Ressourcen und Rüstkosten $\mathcal{NP}$-schwer sind für $m > 2$. Versuche, dieses Problem – oder auch einige Spezialfälle davon – mit ganzzahliger linearer Programmierung optimal zu lösen, waren nur für sehr kleine Instanzen mit $n < 30$ in hinnehmbarer Zeit erfolgreich, was weit hinter praktischen Anforderungen zurückliegt (vgl. Abschnitt 2.1.1). Aufgrund der Komplexität des Gesamtproblems (vgl. das MIP aus Abschnitt 1.1) sollen in dieser Arbeit zwei Dekompositionsansätze verfolgt werden, die $C_{\max}$ in zwei Stufen getrennt nach Zykluszeiten und Rüstkosten optimieren:

1. Zunächst wird eine Reihenfolge π aufgestellt, ohne Ressourcen und Rüstkosten zu betrachten, so dass die Summe der Zykluszeiten möglichst gering ist. Anschließend wird ein zulässiges Mapping f basierend auf π erstellt, möglichst ohne nachträgliche Änderung an π vorzunehmen, so dass die Summe der Rüstkosten minimiert wird.

2. Es wird zuerst ein Mapping f erstellt, so dass die Ressourcen zulässig zugewiesen sind und die durch die Ressourcenwechsel verursachten Rüstkosten minimal sind. Anschließend werden, ohne f zu verändern, die Jobs so in einer Reihenfolge π angeordnet, dass $C_{\max}$ möglichst minimal ist.

Beide Ansätze liefern natürlich im Allgemeinen keine optimalen Lösungen, da jeweils getrennt bezüglich π und f optimiert wird, obwohl die optimale Lösung von beiden zusammen abhängig ist. Außerdem sind selbst die aus den Ansätzen resultierenden Teilprobleme teilweise noch $\mathcal{NP}$-schwer. Beispielsweise ist bei Ansatz (1) das Berechnen einer $C_{\max}$-optimalen Reihenfolge π im Allgemeinen $\mathcal{NP}$-schwer. In dem Spezialfall, dass es nur eine dominierende Maschine gibt, ist es trivial, da $C_{\max}$ bei jeder Reihenfolge identisch ist, Wenn es genau zwei benachbarte dominierende Maschinen gibt, ist es mit dem Algorithmus von Gilmore und Gomory [GG64] in Polynomialzeit lösbar. Bei der anschließenden Zuweisung von Ressourcen ist noch unbekannt, ob ein polynomieller Algorithmus existiert.

Je nachdem, ob die Rüstkosten die Zykluszeiten dominieren oder umgekehrt, ist Ansatz (2) bzw. Ansatz (1) vielversprechender.

2 Der erste Dekompositionsansatz

Beim ersten Dekompositionsansatz wird zunächst eine (möglichst gute) Jobreihenfolge π bestimmt. Dabei werden Ressourcen und Rüstkosten außer Acht gelassen. Es wird also zunächst ausschließlich die Summe der Zykluszeiten optimiert. Anschließend wird ein Mapping f aufgestellt, so dass Ressourcen nur dann eingeplant werden, wenn sie auch zur Verfügung stehen, und darüber hinaus möglichst selten ausgetauscht werden müssen, so dass geringe Rüstkosten auftreten.

In Abschnitt 2.1 werden einige exakte und heuristische Verfahren für die Berechnung einer bezüglich der Zykluszeiten optimalen Reihenfolge π vorgestellt, was im Allgemeinen $\mathcal{NP}$-schwer ist. Anschließend werden in Abschnitt 2.2 Verfahren vorgestellt, die ein möglichst gutes Mapping f erzeugen. Dabei wird darauf eingegangen, ob mit der gegebenen Reihenfolge π und den gegebenen Ressourcen überhaupt eine zulässige Lösung möglich ist, und, wie dann ggf. die Zulässigkeit durch nachträgliche Änderungen an π erzeugt werden kann.

2.1 Berechnen einer Jobreihenfolge

Das Finden einer $C_{\max}$-optimalen Jobreihenfolge ist im Allgemeinen schon ohne Ressourcen und Rüstkosten $\mathcal{NP}$-schwer. Daher wird im ersten Unterabschnitt 2.1.1 zunächst ein MIP vorgestellt, das solch eine Reihenfolge findet. Anschließend wird in Unterabschnitt 2.1.2 auf einen Spezialfall eingegangen, der in polynomieller Zeit lösbar ist. Danach werden in 2.1.3 einige heuristische Verfahren vorgestellt.

2.1.1 Ganzzahlige Lineare Programmierung

Zum Finden einer bezüglich Zykluszeiten optimalen Lösung wurde folgendes MIP aufgestellt, das in dieser Form auch schon in [KS03] vorgestellt wurde. Es ist identisch mit dem MIP in Abschnitt 1.1 bis auf die Nichtberücksichtigung der Rüstkosten.

$$\min \quad \sum_{t=1}^{n+m-1} c_t \tag{2.1}$$

$$\text{s.t.} \quad \sum_{k=1}^{n} x_{jk} = 1 \qquad j \in N \quad (2.2)$$

$$\sum_{j=1}^{n} x_{jk} = 1 \qquad k \in N \quad (2.3)$$

$$c_t \geq \sum_{j=1}^{n} p_{t-k+1,j} \cdot x_{jk} \quad t \in T, k = \max\{1, t-m+1\}, \ldots, \min\{n, t\} \quad (2.4)$$

$$c_t \geq 0 \qquad t \in T \quad (2.5)$$

$$x_{jk} \in \{0, 1\} \qquad j, k \in N \quad (2.6)$$

Dieses MIP liefert für Instanzen mit $n \leq 30$ eine optimale Lösung in unter einer Stunde. Diese Laufzeiten sind zwar schon deutlich besser als die Variante mit Rüstkosten in Abschnitt 1.1, bei größeren Instanzen ist dieser Zeitaufwand allerdings immer noch nicht praktikabel.

Für Instanzen mit $n \leq 500$ ist die beste nach einer halben Stunde gefundene Lösung (obere Schranke) in ihrer Güte vergleichbar mit den Heuristiken, die in Unterabschnitt 2.1.3 vorgestellt werden (für einen Vergleich s. 4.1). In diesem Sinne kann dieses MIP daher ebenfalls als Heuristik betrachtet werden.

2.1.2 Der Algorithmus von Gilmore und Gomory

Der Algorithmus von Gilmore und Gomory [GG64] löst in $\mathcal{O}(n \log n)$ einen speziellen Fall des gerichteten Travelling-Salesman-Problems (TSP), bei dem alle Knoten i zwei Parameter a_i, b_i haben und die Kantenkosten c_{ij} zwischen je zwei Knoten i und j nur von a_i und b_j abhängig sind.

Diese Situation liegt beim synchronen Flow-Shop vor, wenn es nur zwei benachbarte dominierende Maschinen gibt. O.B.d.A seien dies M_1 und M_2. Jobs können durch Knoten repräsentiert werden und die beiden Prozesszeiten auf den dominierenden Maschinen liefern die Parameter a_i und b_i. Der Abstand zwischen zwei Knoten entspricht dann der Zykluszeit, die die entsprechenden Jobs verursachen, wenn sie nebeneinander liegen. Die Berechnung der Zykluszeiten vereinfacht sich hier zu $c_t = \max\{p_{2\pi_{t-1}}, p_{1\pi_t}\}$ für $2 \leq t \leq n$. Sie sind also für je zwei Jobs π_{t-1}, π_t nur noch von der Prozesszeit des vorderen Jobs auf der zweiten Maschine ($p_{2\pi_{t-1}}$) und der Prozesszeit des hinteren Jobs auf der ersten Maschine ($p_{1\pi_t}$) abhängig.

Auf die Funktionsweise des Algorithmus soll in dieser Arbeit nicht näher eingegangen werden. Eine Anwendung des Algorithmus von Gilmore und Gomory als Heuristik für ein Problem mit mehreren dominierenden Maschinen wird in 2.1.3 beschrieben.

2.1.3 Heuristische Verfahren

Aufgrund der $\mathcal{NP}$-Schwere der Optimierung von $C_{\max}$ im allgemeinen Fall und der schlechten Laufzeit des MIPs aus Abschnitt 2.1.1 bei Instanzen realer Größe werden hier einige heuristische Ansätze zur Berechnung von π vorgestellt.

Non-Full-Schedule-Heuristik

Diese Heuristik ist eine konstruktive Greedy-Heuristik, die Schritt für Schritt einen Job an π anhängt, beginnend mit einer leeren Reihenfolge. Sie arbeitet ähnlich wie die Nearest-Neighbor-Heuristik beim TSP. In jeder Iteration werden alle noch verbleibenden Jobs bewertet und der Job mit der besten Bewertung wird an π angehängt. Die Heuristik benötigt also genau n Iterationen. Die Bewertungsfunktion betrachtet die letzten $m-1$ Zykluszeiten der noch nicht fertigen Reihenfolge, wobei die Zykluszeiten am Anfang bei einer noch leeren Reihenfolge als 0 angenommen werden. Für jeden Job j, der noch nicht in π ist, werden diese $m-1$ Zykluszeiten mit den ersten $m-1$ Prozesszeiten von j verglichen. Die Idee ist, dass diese möglichst übereinstimmen sollten. Ist eine Prozesszeit sehr viel größer als die aktuelle Zykluszeit, zu der sie hinzugefügt werden würde, würde die Zykluszeit entsprechend um einen Wert c_+ ansteigen. Ist umgekehrt die Zykluszeit sehr viel größer als die zugehörige Prozesszeit von j, dann würde diese kurze Prozesszeit verschenkt werden. Die Differenz aus Zykluszeit und Prozesszeit wird mit c_- bezeichnet. Die Bewertungsfunktion berechnet für jeden Job die Summe aus den $m-1$ c_+-Werten. Diese Summe ist die Bewertung für einen Job j. Der Job mit der kleinsten Bewertung wird an π angehängt. Falls mehrere Jobs eine optimale Bewertung haben, wird für diese Jobs als zweites Kriterium die Summe der c_--Werte betrachtet. Der Job, bei dem diese Summe am kleinsten ist, verschenkt am wenigsten Zeit und wird ausgewählt. Das Vorgehen dieser Heuristik wird anhand der Beispielinstanz 1.2 in Abbildung 2.1 veranschaulicht.

Da in jeder Iteration alle verbleibenden Jobs betrachtet werden und für jeden dieser Jobs $m-1$ Zeiten verglichen werden, liegt die asymptotische Laufzeit dieser Heuristik in $\mathcal{O}(n^2 m)$.

Double Ended Non-Full-Schedule-Heuristik

Diese Heuristik basiert auf der Non-Full-Schedule-Heuristik. Der Unterschied besteht darin, dass π nicht nur von vorne, sondern gleichzeitig auch von hinten zur Mitte hin aufgebaut wird. Jeder noch nicht in π enthaltene Job wird pro Iteration mit beiden Enden der bisherigen Reihenfolge verglichen. Die Bewertungsfunktion für das hintere Ende arbeitet analog. Es wird der beste Job für das vordere Ende und der beste für

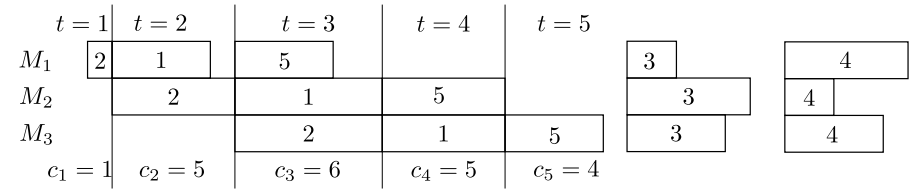

Abbildung 2.1: Illustration der Non-Full-Schedule-Heuristik anhand der Beispielinstanz in Tabelle 1.2. Bislang wurden die Jobs 2, 1 und 5 eingefügt. 3 und 4 müssen noch eingefügt werden. 2 wurde zuerst eingefügt, da dieser mit $p_{12}+p_{22} = 1+5$ die geringsten c_+-Werte auf den ersten $m-1 = 2$ Maschinen hat. 1 und 5 wurden eingefügt, weil ihre beiden ersten Prozesszeiten die beiden letzten Zykluszeiten nicht vergrößern. Als nächstes wird Job 4 eingefügt, da weder seine Prozesszeit auf M_1 ($p_{14} = 5$) die Zykluszeit $c_4 = 5$ vergrößern würde noch seine Prozesszeit auf M_2 ($p_{24} = 2$) die Zykluszeit $c_5 = 4$ vergrößern würde. Job 3 hingegen würde mit $p_{23} = 5$ die Zykluszeit $c_5 = 4$ um 1 vergrößern.

das hintere Ende gesucht und der mit der besseren Bewertung wird vorne bzw. hinten eingefügt. Die asymptotische Laufzeit liegt hier ebenfalls in $\mathcal{O}(n^2m)$.

Gilmore-Gomory-Heuristik

Diese Heuristik wendet den Algorithmus von Gilmore und Gomory auf beliebige Instanzen an. Es können zwei Fälle eintreten:

1. Unter den dominierenden Maschinen D gibt es zwei, die benachbart sind, also $\exists i \in \{1, \ldots, m-1\}$ mit $i, i+1 \in D$.

2. Es gibt keine benachbarten dominierenden Maschinen.

Wenn Fall (2) eintritt, werden alle m Maschinen als dominierend angesehen, da dies für den Algorithmus keine Einschränkung darstellt. Nun werden zwei benachbarte dominierende Maschinen gewählt. O.B.d.A seien dies die Maschinen M_1 und M_2. Seien

$$d_{\min} \coloneqq \min_{\substack{i \in \{1,2\} \\ j \in N}} p_{ij} \tag{2.7}$$

$$e_{\max} \coloneqq \max_{\substack{i \notin \{1,2\} \\ j \in N}} p_{ij} \tag{2.8}$$

$$K \coloneqq \frac{e_{\max}}{d_{\min}}, \tag{2.9}$$

wobei $d_{\min} = 0$ und daraus folgend $K = \infty$ erlaubt sind. Nun wird aus der gegebenen Instanz I eine neue Instanz I' erzeugt, die sich nur dadurch von I unterscheidet, dass die Prozesszeiten auf allen dominierenden Maschinen außer auf M_1 und M_2 mit $\frac{1}{K}$ skaliert werden, also

$$p'_{ij} \coloneqq \begin{cases} \frac{p_{ij}}{K} & \text{für } i \in D \setminus \{1,2\} \\ p_{ij} & \text{sonst.} \end{cases} \tag{2.10}$$

Auf diese Weise sind M_1 und M_2 in I' die einzigen dominierenden Maschinen und folglich kann I' mit dem Algorithmus von Gilmore und Gomory optimal gelöst werden. Die resultierende Reihenfolge π wird als heuristische Lösung für die ursprüngliche Instanz I verwendet.

Da die Prozesszeiten der in I' vernachlässigten dominierenden Maschinen sich um den Faktor K von denen in I unterscheiden, können sich die aus π ergebenen Zykluszeiten von I und I' auch maximal um den Faktor K unterscheiden:

$$c_t \leq K \cdot c'_t \quad \forall 1 \leq t \leq n + m - 1 \tag{2.11}$$

Da die optimale Lösung von I' eine untere Schranke für die optimale Lösung von I ist, also $C'_{\max} \leq C_{\max}$, folgt für die Lösung L_{GG} der Gilmore-Gomory-Heuristik:

$$L_{GG} \leq K \cdot C'_{\max} \leq K \cdot C_{\max} \tag{2.12}$$

Die Gilmore-Gomory-Heuristik hat also eine relative Gütegarantie von K. Bei der Wahl der beiden benachbarten dominierenden Maschinen sollte darauf geachtet werden, dass $d_{\min}$ möglichst groß und $e_{\max}$ möglichst klein ist. Dadurch ist dann auch der Gütefaktor K am kleinsten.

Das Finden der geeigneten dominierenden Maschinen kann in $\mathcal{O}(nm)$ durchgeführt werden. Das Anpassen der Prozesszeiten der übrigen dominierenden Maschinen ist nur in der Theorie von Interesse. Der Gilmore-Gomory-Algorithmus kann diese einfach ignorieren. Die Gesamtlaufzeit beträgt daher $\mathcal{O}(nm + n \log n)$.

Für die weitere Analyse dieser Heuristik soll der Begriff der *Semidominanz* eingeführt werden. Die Semidominanz ist ein Maß dafür, wie weit eine Teilmenge von Maschinen davon entfernt ist, dominierend zu sein. Eine Semidominanz von 0 bedeutet, die Maschinen sind dominierend. Für eine Teilmenge von Maschinen $D \subseteq \{1, \ldots, m\}$ sei

$$d_{\min} = \min_{\substack{i \in D \\ j \in \{1,\ldots,n\}}} p_{ij}$$

die kleinste ihrer Prozesszeiten, analog zur obigen Definition von $d_{\min}$. Die Semidominanz ist dann definiert als

$$\mathcal{D}_D \coloneqq \sum_{i \notin D} \sum_{j=1}^{n} \max\{0, p_{ij} - d_{\min}\}.$$

Unabhängig von der *relativen* Gütegarantie K liefert diese Heuristik auch eine *absolute* Gütegarantie von $\mathcal{D}_{\{i,i+1\}}$, wenn M_i und M_{i+1} als benachbarte Maschinen ausgewählt werden. Die Gilmore-Gomory-Heuristik liefert also besonders gute Näherungen, wenn zwei benachbarte Maschinen fast dominierend sind und nur wenige Prozesszeiten auf anderen Maschinen dies verhindern.

Nachbarschaftssuche

Mit einer Nachbarschaftssuche können bereits existierende (nicht optimale) Reihenfolgen verbessert werden. Mögliche Nachbarschaftsoperatoren sind

- xch_{jk}: vertauscht die Jobs an den Positionen j und k miteinander,
- $shift_{jk}$: „shiftet“ den Job an Position j nach Position k. Alle Jobs zwischen j und k rücken um eins nach links (falls $j < k$) bzw. nach rechts (falls $j > k$).

Auf Grundlage dieser Nachbarschaften lassen sich außerdem z.B. Iterative Improvement, eine Tabu-Suche oder Simulated Annealing implementieren.

Im Rahmen dieser Arbeit wurde ein Simulated Annealing Algorithmus auf Grundlage der xch_{jk}-Nachbarschaft implementiert. Es wird mit einer zufälligen initialen Reihenfolge π begonnen. Die Funktion, die $C_{\max}$ von π berechnet, sei mit c bezeichnet. Pro Iteration werden zufällige j und k bestimmt, mit denen eine benachbarte Lösung $\pi' = xch_{jk}(\pi)$ erzeugt wird. Ist π' besser als die aktuelle Lösung π, d.h. $c(\pi') < c(\pi)$, wird π' als neue Lösung übernommen. Ist π' schlechter als π, wird π' nur mit einer bestimmten Wahrscheinlichkeit übernommen. Diese Wahrscheinlichkeit beträgt $e^{\frac{c(\pi)-c(\pi')}{t_i}}$, wobei $(t_i)_i$ eine Nullfolge ist und i die Anzahl der bisher durchgeführten Iterationen angibt. Die Folge $(t_i)_i$ wird auch als „Temperatur“ bezeichnet, die im Laufe des Verfahrens abnimmt. Es wurde mit unterschiedlichen Temperaturen experimentiert. Es wurden

- $t_i = \alpha t_{i-1}$ für unterschiedliche $\alpha \in]0,1[$ und
- $t_i = \frac{k}{i}$,
- $t_i = \frac{k}{\log i}$ und
- $t_i = k\frac{|\cos \frac{i}{30}|}{i}$

jeweils für unterschiedliche $k \in \mathbb{R}_+$ getestet. Die besten Resultate lieferte $t_i = 50\frac{|\cos \frac{i}{30}|}{i}$. Durch den Cosinus schwankt die Temperatur, so dass abwechselnd verschlechternde Lösungen erlaubt bzw. nicht erlaubt werden. Das Verfahren terminiert, sobald in 1000 aufeinander folgenden Iterationen keine neue beste Lösung gefunden werden konnte.

Dieser Simulated Annealing Algorithmus lieferte in Tests meist etwas bessere Ergebnisse als die übrigen Heuristiken. Die Heuristiken werden in Kapitel 4 verglichen.

Heuristiken von Soylu et al.

Die oben beschriebenen Heuristiken werden in Kapitel 4 nicht nur untereinander verglichen, sondern auch mit drei Heuristiken von Soylu et al. [SKA07], die hier kurz beschrieben werden.

H_1 Diese Heuristik erstellt m unterschiedliche Jobreihenfolgen und wählt anschließend die beste aus. Die k-te dieser m Reihenfolgen wird erstellt, indem für die Zyklen $k, \ldots, n+k-1$ die Anzahl der Jobs betrachtet wird, die sich in diesem Zyklus in der Anlage befinden (diese Anzahl beträgt nur in den ersten und letzten $m-1$ Zyklen *nicht* m). Anschließend werden die Jobs den Zyklen so zugewiesen, dass der Job mit der i-t kleinsten Prozesszeit im Zyklus mit der i-t kleinsten Anzahl an Jobs liegt. Es werden also $m-1$ Jobs einer wohldefinierten Position in π zugewiesen (nämlich den Zyklen, in denen weniger als m Jobs in der Anlage sind) und die übrigen $n-m$ Jobs werden wahllos eingefügt, da alle übrigen Zyklen m Jobs enthalten und somit bezüglich dieser Metrik nicht unterscheidbar sind.

H_2 Hier wird in einer Iteration ein Job in die anfangs leere Reihenfolge eingefügt. Es wird unter allen noch nicht eingefügten Jobs jeweils derjenige ausgesucht, der die größte Prozesszeit hat. Dieser Job wird dann so in die Reihenfolge eingefügt, dass diese Prozesszeit ein den Zyklus fällt, dessen Zykluszeit bislang am größten war. Sollte das nicht möglich sein, wird stattdessen die zweitgrößte Zykluszeit gewählt usw. Auf diese Weise werden alle Jobs eingefügt nacheinander eingefügt. Der erste Job, der die größte aller Prozesszeiten hat, wird so eingefügt, dass diese Prozesszeit in Zyklus m liegt.

H_3 Diese Heuristik ähnelt der Non-Full-Schedule-Heuristik (vgl. Abschnitt 2.1.3). Der Unterschied besteht darin, dass zu der Summe der c_+-Werte noch die Prozesszeit auf der letzten Maschine des jeweiligen Jobs mit aufaddiert wird und diese Gesamtsumme dann als Einfügekriterium dient. Sollte es mehrere gleichwertige Jobs geben, wird kein zweites Kriterium zur Rate gezogen, sondern zufällig einer ausgewählt.

2.2 Zuweisung von Ressourcen

In diesem Abschnitt geht es darum, den Jobs in der Reihenfolge π, die in Abschnitt 2.1 aufgestellt wurde, Ressourcen zuzuweisen. An diese Zuweisung werden zwei Anforderungen gestellt:

1. Die Zuweisung muss zulässig sein. Das heißt, ist eine Ressource einem Job zugewiesen, darf sie den nachfolgenden $m-1$ Jobs nicht mehr zugewiesen werden. Das Problem der Zulässigkeit einer Zuweisung wird im ersten Unterabschnitt 3.1.2 betrachtet.

2. Die Zuweisung soll möglichst gut sein. Wenn zwei Jobs, die im Abstand von m in π liegen, die selbe Ressource benutzen können, werden Rüstkosten eingespart, da die Ressource nicht ausgetauscht werden muss. Dieses Problem wird in Unterabschnitt 2.2.2 diskutiert.

Es ist natürlich möglich, dass bei gegebenem π keine zulässige Zuweisung von Ressourcen möglich ist. In diesem Fall muss π nachträglich geändert werden, worauf in Unterabschnitt 4.2.1 eingegangen wird.

2.2.1 Zulässigkeit der Zuweisung

Abhängig davon, welche Ressourcen für welche Jobs geeignet sind, kann das Zulässigkeitsproblem unterschiedlich schwer zu lösen sein. Folgende Situationen werden hier betrachtet:

- Alle Ressourcen sind für alle Jobs geeignet (trivial, es müssen m Ressourcen vorhanden sein).

- Die Ressourcenmengen sind disjunkt ($\rho_i \cap \rho_j \neq \emptyset \;\Rightarrow\; \rho_i = \rho_j$).

- Die ρ_i sind beliebige Teilmengen von R.

Der Fall, dass die Ressourcen für hierarchische Jobgruppen geeignet sind ($\iota_q \cap \iota_r \neq \emptyset \;\Rightarrow\; \iota_q \subseteq \iota_r$ oder $\iota_r \subseteq \iota_q$), wird nicht gesondert betrachtet. Da dieser Fall aber natürlich auch ein Spezialfall der beliebigen Teilmengen ist, kann er mit demselben Verfahren gelöst werden.

Zulässigkeit bei disjunkten Ressourcenmengen

Bei disjunkten Ressourcenmengen können die Jobs in Gruppen unterteilt werden, so dass zu jeder Jobgruppe eine für sie exklusive Menge an zulässigen Ressourcen zur Verfügung

steht. Für die Zulässigkeit reicht es aus, statt der Ressourcenmengen ρ_i nur deren Größen $|\rho_i|$ zu betrachten. Mit einem Greedy-Algorithmus kann eine gegebene Reihenfolge π dann auf Zulässigkeit überprüft werden: Der Algorithmus durchläuft π von vorne nach hinten. Jede Jobgruppe erhält einen Counter, der mitzählt, wie viele Ressourcen momentan für sie zur Verfügung stehen. Diese Counter werden zu Beginn mit $|\rho_i|$ initialisiert. An jeder Position in π wird der Counter der zugehörigen Jobgruppe dekrementiert. Nach m Schritten wird dieser Counter wieder inkrementiert. Sollte ein Counter einmal negativ werden, gibt es keine zulässige Ressourcenzuteilung.

Zulässigkeit bei beliebigen Ressourcenteilmengen

Es ist nicht bekannt, ob das Problem, zu entscheiden, ob es bei gegebenem π und beliebigen Ressourcenmengen ein zulässiges Mapping f gibt, $\mathcal{NP}$-vollständig ist oder ob es einen polynomiellen Algorithmus gibt.

Es besteht allerdings die Vermutung, dass es bei konstantem m einen polynomiellen Algorithmus in n gibt. Ein Indiz für diese Vermutung liefert das nachfolgende MIP. O.B.d.A. sei hier $\pi_i = i$, d.h. Job i ist an Position i. Die Binärvariablen x_{ir} geben hier an, ob dem Job $i \in N$ die Ressource $r \in \rho_i$ zugeteilt wird. In diesem Fall ist $x_{ir} = 1$ und sonst $x_{ir} = 0$.

$$\max \quad 0 \tag{2.13}$$

$$\text{s.t.} \quad \sum_{r \in \rho_i} x_{ir} = 1 \qquad i \in N \tag{2.14}$$

$$x_{ir} + x_{jr} \leq 1 \qquad i \in N, j = i+1, \ldots, i+m-1\,, r \in \rho_i \cap \rho_j \tag{2.15}$$

$$x_{ir} \in \{0,1\} \qquad i \in N, r \in \rho_i \tag{2.16}$$

Die erste Nebenbedingung (2.14) bewirkt, dass jedem Job genau eine Ressource zugewiesen wird. Durch die zweite Nebenbedingung (2.15) kann jede Ressource jeweils nur einmal an m aufeinander folgende Jobs vergeben werden. Eine Zielfunktion ist nicht notwendig, da die Zulässigkeit nur durch die Nebenbedingungen entschieden wird. Mit diesem MIP können selbst sehr große Instanzen mit $n \approx 100000$ (allerdings mit $m \leq 8$) in weniger als einer Sekunde gelöst werden.

Nebenbedingung (2.15) verhindert immer nur für Paare von Jobs, deren Abstand kleiner als m ist, dass ihnen die selbe Ressource zugewiesen wird. Das ist für die Korrektheit des MIPs zwar ausreichend, durch zusätzliche Nebenbedingungen, die dasselbe für Tripel, Quadrupel, bis hin zu m-Tupeln verhindern, können aber einige fraktionale Lösungen der Relaxation „abgeschnitten“ werden. Für ein k-Tupel von Jobs, die alle in einem Abschnitt von π der Länge m stehen, sieht diese zusätzliche Nebenbedingung so

aus:

$$\sum_{l=1}^{k} x_{i_l r} \leq 1 \quad i_l \in N, i_1 < i_2 < \ldots < i_k < i_1 + m, r \in \bigcap_{l=1}^{k} \rho_{i_l} \tag{2.17}$$

Für $k = 2$ ist diese Nebenbedingung identisch mit (2.15). Die Anzahl dieser Nebenbedingungen lässt sich durch $\mathcal{O}(n(m-1)!|R|)$ abschätzen, was bei konstantem m polynomiell ist. Werden sie für $k = 3, \ldots, m$ zu obigem MIP hinzugefügt, so war bei allen bislang getesteten 100 Instanzen die Relaxation bereits ganzzahlig. Ein Beweis, dass dies tatsächlich bei allen Instanzen der Fall ist, ist noch nicht gelungen. Gelingt er, ist bewiesen, dass es in polynomieller Zeit (bei konstantem m) möglich ist, zu entscheiden, ob es für eine gegebene Jobreihenfolge π und gegebene (beliebige) Ressourcenteilmengen ρ_i möglich ist, allen Jobs eine Ressource zuzuweisen.

2.2.2 Optimierung der Ressourcenzuweisung

Da unbekannt ist, ob das Finden *irgendeiner* zulässigen Ressourcenzuweisung $\mathcal{NP}$-schwer ist, ist es natürlich ebenfalls unbekannt, ob das Finden einer *optimalen* Ressourcenzuweisung $\mathcal{NP}$-schwer ist.

Eine Ausnahme bildet der Fall von disjunkten Ressourcenteilmengen ρ_i. Immer, wenn zwei Jobs aus einer Familie im Abstand m aufeinander folgen, kann ihnen dieselbe Ressource zugewiesen werden. Wenn zwei Jobs im Abstand m nicht zur selben Familie gehören, gibt es auch keine Möglichkeit, Rüstkosten zu vermeiden oder zu verringern.

Für den allgemeinen Fall lässt sich das MIP aus Unterabschnitt 2.2.1 erweitern. Hinzu kommen für $i \in \{m+1, \ldots, n\}$ und $r \in \rho_i \cap \rho_{i-m}$ Binärvariablen y_{ir}, für die gilt: $y_{ir} = 1$ genau dann, wenn den Jobs an den Positionen i und $i - m$ in π die selbe Ressource r zugeteilt ist. Auch hier sei wieder o.B.d.A. $\pi_i = i$.

$$\max \quad \sum_{i=m}^{n} \sum_{r \in \rho_i \cap \rho_{i-m}} y_{ir} \tag{2.18}$$

$$\text{s.t.} \quad \sum_{r \in \rho_i} x_{ir} = 1 \qquad i \in N \tag{2.19}$$

$$x_{ir} + x_{jr} \leq 1 \qquad i \in N, j = i+1, \ldots, i+m-1, r \in \rho_i \cap \rho_j \tag{2.20}$$

$$y_{ir} \leq x_{ir} \qquad i = m, \ldots, n, r \in \rho_i \cap \rho_{i-m} \tag{2.21}$$

$$y_{ir} \leq x_{i-m,r} \qquad i = m, \ldots, n, r \in \rho_i \cap \rho_{i-m} \tag{2.22}$$

$$x_{ir} \in \{0, 1\} \qquad i \in N, r \in \rho_i \tag{2.23}$$

$$y_{ir} \in \{0, 1\} \qquad i = m, \ldots, n, r \in \rho_i \cap \rho_{i-m} \tag{2.24}$$

Durch die Nebenbedingungen (2.21) und (2.22) wird erzwungen, dass die y_{ir} die oben beschriebenen Werte annehmen. In der Zielfunktion (2.18) wird die Anzahl der y-Variablen,

die auf 1 gesetzt sind, maximiert, da so die wenigsten Rüstkosten auftreten. Dieses MIP liefert genau wie jenes aus 2.2.1 in weniger als einer Sekunde auch für sehr große Instanzen eine Lösung.

2.3 Unzulässige Reihenfolgen

Für den Fall, dass für eine gegebene Reihenfolge π kein zulässiges Mapping f erstellt werden kann, muss π nachträglich verändert werden.

Eine Nachbarschaftssuche ist eine Möglichkeit zur nachträglichen Änderung von π: Mit einer Vertauschung zweier Jobs kann π unter Umständen so abgeändert werden, dass doch ein zulässiges Mapping existiert. Es wird dazu eine Matrix erstellt, deren Einträge r_{ij} angeben, wie viele Ressourcen für Job j an Position i in π zur Verfügung stehen (d.h. sich nicht gerade in der Anlage befinden). Initial ist $r_{ij} = |\rho_j|$ für alle Positionen i. Indem für alle Positionen i die m zuletzt eingelegten Jobs j ermittelt werden (also die an den Positionen $i - m + 1, \ldots, i$), können jeweils die Einträge r_{ij} um 1 verringert werden. Sobald einer der Einträge negativ wird, bedeutet das, dass eine Ressource zugewiesen wurde, obwohl sie an dieser Position i nicht mehr verfügbar ist. Es kann dann ein Job j' an einer anderen Position in π gesucht werden, dessen Ressource an Position i noch verfügbar ist. Dafür muss $r_{ij'} > 0$ sein. Wenn solch ein Job existiert, wird er mit dem Job an Position i getauscht und die Matrix wird angepasst. Falls mehrere solcher Jobs existieren, wird derjenige ausgewählt, durch dessen Tausch sich die Summe der Zykluszeiten am wenigsten verschlechtert.[1] Sollte kein solcher Job gefunden werden, terminiert die Nachbarschaftssuche mit dem Ergebnis, dass kein zulässiges Mapping gefunden werden konnte. Dabei ist zu beachten, dass dieser Algorithmus nicht garantiert, ein zulässiges Mapping zu finden, falls eines existiert.

[1] Es ist nicht ausgeschlossen, dass sich die Summe der Zykluszeiten dadurch sogar verbessert.

3 Der zweite Dekompositionsansatz

In diesem Kapitel wird der zweite Dekompositionsansatz vorgestellt. Hierbei wird erst jedem Job eine Ressource zugewiesen, also ein Mapping f erzeugt, und anschließend werden die Jobs in eine Reihenfolge π gebracht. Neben der reinen Ressourcenzuteilung wird im ersten Schritt auch teilweise schon festgelegt, welche Jobs in der späteren Reihenfolge einen Abstand von m haben sollen. So wird sichergestellt, dass die durch das Mapping f vorgeschriebenen Ressourcenzuteilungen auch dazu führen, dass Rüstkosten eingespart werden.

In Abschnitt 3.1 wird erklärt, wie ein optimales Mapping f mit möglichst wenigen Rüstkosten mit einem Binpacking-Ansatz erstellt werden kann. Anschließend wird in Abschnitt 3.2 beschrieben, wie basierend auf dem gegebenen Mapping f eine Reihenfolge π erstellt wird, so dass die Summe der Zykluszeiten und Rüstkosten ($C_{\max}$) minimal ist.

In den meisten nachfolgend genannten Verfahren wird nur der Spezialfall mit konstanten Rüstkosten ($s_{fg} = s$), disjunkten Ressourcenmengen ρ_i und zwei dominierenden Maschinen betrachtet vor dem Hintergrund des gegebenen Praxisbeispiels des Küchenherstellers.

3.1 Ressourcenzuweisung

In diesem Abschnitt wird die Konstruktion des Mappings f erläutert. Dabei wird davon ausgegangen, dass Rüstkosten einzig durch einen Ressourcenwechsel verursacht werden. In Unterabschnitt 3.1.1 wird auf den Spezialfall $s_{fg} = s$, der mit einem Binpackingansatz gelöst werden kann, eingegangen. Allgemeinere Fälle der Ressourcenzuweisung beim zweiten Dekompositionsansatz werden in dieser Arbeit nicht gesondert betrachtet.

Wie in Abschnitt 1.1 erläutert, ist im Folgenden von *Jobgruppen* die Rede, wobei eine Jobgruppe g aus $n_g \in \mathbb{N}$ identischen Jobs besteht. n_g ist also die Größe einer Jobgruppe g. Die Anzahl der Jobgruppen wird mit $\tilde{n}$ bezeichnet. Für die Gesamtanzahl an Jobs gilt dann $n = \sum_{g=1}^{\tilde{n}} n_g$.

3.1.1 Ressourcenzuweisung mit Binpacking

Bei $\tilde{n}$ gegebenen Jobgruppen, wobei alle Jobs aus einer Gruppe die gleichen Ressourcen verwenden können, ist es naheliegend, die Jobs einer Gruppe immer im Abstand von m in die Anlage einzulegen. Auf diese Weise muss die Ressource in der entsprechenden Station so lange nicht ausgetauscht werden, bis alle Jobs aus einer Jobgruppe fertiggestellt worden sind. Erst danach, wenn Jobs aus einer anderen Gruppe eingelegt werden, muss einmalig die Ressource gewechselt werden, so dass nur dann Rüstkosten auftreten. Insgesamt muss dann nur $(\tilde{n}-m)$-mal ein Ressourcenwechsel stattfinden (beim Einlegen der ersten m Jobs muss keine Ressource gewechselt werden, evtl. treten aber initiale Rüstkosten auf, was in dieser Arbeit aber nicht beachtet wird). Dabei soll nur der Spezialfall mit disjunkten Ressourcenmengen und konstanten Rüstkosten ($s_{fg} = s$) betrachtet werden.

Zur Vereinfachung wird angenommen, dass n ein Vielfaches von m ist. Bei $\tilde{n}$ gegebenen Jobgruppen mit den Größen n_g, $g = 1, \ldots, \tilde{n}$ ist das Ziel, die Jobgruppen in m gleichgroße Teilmengen zu unterteilen, möglichst ohne eine der Jobgruppen zu teilen. Dieses Problem lässt sich als Binpacking-Instanz betrachten, wobei entschieden werden muss, ob $\tilde{n}$ Items mit den Größen $n_1, \ldots, n_{\tilde{n}}$ in m Bins der Größe $\frac{n}{m}$ untergebracht werden können. Jobs, die sich in einem Bin befinden, werden danach einer Station der Anlage zugewiesen. Da eine Station sich nach genau m Zyklen wieder an ihrer Ausgangsposition befindet, werden die Jobs eines Bins immer im Abstand von m in die Anlage eingelegt und wenn sie zur gleichen Jobgruppe gehören, treten keine Rüstkosten auf, da die Ressource in der Station bleiben kann.

Für den Fall, dass die Binpacking-Instanz eine Lösung hat, müssen noch die Bins den Stationen zugeordnet werden und die Reihenfolge der Jobgruppen in jedem Bin muss so bestimmt werden, dass die Zykluszeiten minimiert werden. Dies wird in Abschnitt 3.2 beschrieben. Falls die Binpacking-Instanz keine Lösung hat, bedeutet das, dass mindestens eine der Jobgruppen aufgeteilt werden muss auf zwei Bins, so dass ein weiteres Mal Rüstkosten auftreten.

Um herauszufinden, welche Jobgruppe geteilt werden muss und wie groß die beiden Hälften sein müssen, wird eine weitere Binpacking-Instanz[1] gelöst, die bis auf eine Änderung identisch zur vorherigen ist: Es gibt nicht mehr m Bins, sondern nur noch $m-1$ und einer dieser Bins hat die Größe $2\frac{n}{m}$ (alle anderen haben weiterhin die Größe $\frac{n}{m}$). Wenn hier eine Aufteilung möglich ist, kann der Bin mit doppelter Größe wieder halbiert werden, wobei dann eine der Jobgruppen in ihm auch geteilt werden muss wie in Abbildung 3.1 veranschaulicht. Sollte auch hier keine Lösung möglich sein, muss eine weitere Jobgruppe geteilt werden. Dazu wird wiederum eine neue Binpacking-Instanz erzeugt. Abbildung 3.2 zeigt, wie die Bingrößen in den weiteren Instanzen aussehen.

[1] Streng genommen handelt es sich hierbei nicht mehr um eine Binpacking-Instanz wegen der unterschiedlichen Bingrößen.

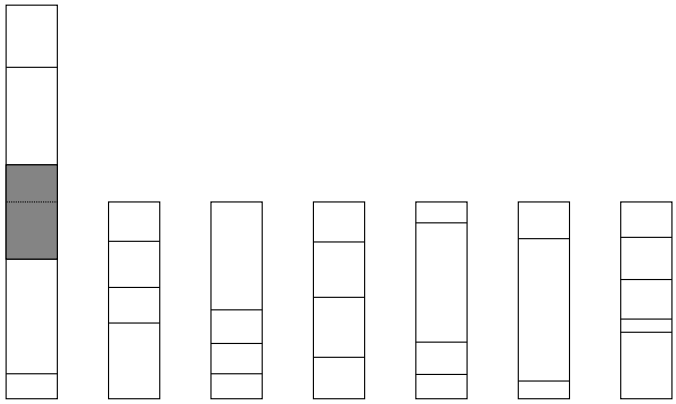

Abbildung 3.1: Jobgruppen, die $m = 8$ Bins zugeordnet werden. Da die ursprüngliche Binpacking-Instanz keine Lösung hatte, wurden zwei der Bins zu einem mit doppelter Größe verschmolzen. Auf diese Weise ist es möglich, die Jobgruppen passgenau zuzuordnen, wobei die grau markierte Jobgruppe getrennt werden muss und im Folgenden als zwei Jobgruppen angesehen wird.

Es handelt sich dabei um die *ungeordneten Zahlenpartitionen* von m. Spätestens bei der Binpacking-Instanz, die nur noch einen Bin enthält mit der Größe n, gibt es eine Lösung, bei der dann $m-1$ Jobgruppen aufgeteilt werden müssen, so dass $m-1$ weitere Ressourcenwechsel und damit Rüstkosten auftreten.

Da das Binpacking-Problem in der hier dargebotenen Form nur schwach $\mathcal{NP}$-vollständig in der Anzahl der Jobgruppen $\tilde{n}$ ist, lässt es sich trotzdem (theoretisch) polynomiell in der Eingabegröße n lösen. Im Rahmen dieser Arbeit wurde das nachfolgende MIP aufgestellt, um die Jobgruppen den Bins zuzuordnen. Für die Binärvariablen x_{gj} gilt: $x_{gj} = 1$ genau dann, wenn Jobgruppe g Bin j zugeordnet wird. Die erste Nebenbedingung fordert, dass jede Jobgruppe in genau einem Bin ist und die zweite Nebenbedingung verlangt, dass jeder Bin exakt gefüllt ist. Dabei sind B_j die Größen der Bins, die zusammen mit der Anzahl der Bins m' mit jedem Aufruf einer Zahlenpartition von m entsprechen:

$$\max \quad 0 \tag{3.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{m'} x_{gj} = 1 \qquad g = 1, \ldots, \tilde{n} \tag{3.2}$$

$$\sum_{g=1}^{\tilde{n}} n_g x_{gj} = B_j \qquad j = 1, \ldots, m' \tag{3.3}$$

$$x_{gj} \in \{0, 1\} \qquad g = 1, \ldots, \tilde{n},\ j = 1, \ldots, m' \tag{3.4}$$

Die Partitionen von m wurden der Reihenfolge nach (vgl. Abbildung 3.2) mit kleiner

1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	
2	2	1	1	1	1		
3	1	1	1	1	1		
2	2	2	1	1			
3	2	1	1	1			
4	1	1	1	1			
2	2	2	2				
3	2	2	1				
3	3	1	1				
4	2	1	1				
5	1	1	1				
3	3	2					
4	2	2					
4	3	1					
5	2	1					
6	1	1					
4	4						
5	3						
6	2						
7	1						
8							

Abbildung 3.2: Ungeordnete Zahlenpartitionen von 8 bezüglich m' gruppiert und sortiert. Jede Zeile stellt die Anzahl der Bins und deren Größe bei möglichen Binpacking-Instanzen dar. Die zweite Zeile beispielsweise stellt den Status der Instanz in Abbildung 3.1 dar.

Abbildung 3.3: Ein Bin der Größe $4\frac{n}{m}$ (ein 4-Bin) mit 6 Jobgruppen, von denen eine so groß ist, dass sie an drei Stellen geteilt werden muss. Alle mittleren Teilstücke haben dann die Größe $\frac{n}{m}$ und die Größen der beiden äußeren Teilstücke liegen im Intervall $[1, \frac{n}{m}]$.

werdendem m' durchlaufen, bis eine zulässige Aufteilung gefunden wurde.

3.1.2 Zulässigkeit der Zuweisung

In Unterabschnitt 3.1.1 wurde erläutert, wie Jobs bzw. Jobgruppen so angeordnet werden können, dass möglichst selten eine Ressource gewechselt werden muss, was Rüstkosten verursacht. Es wurde allerdings davon ausgegangen, dass immer genug Ressourcen vorhanden sind, d.h. es wurde nur optimiert, ohne zu beachten, ob diese Zuweisung von Ressourcen überhaupt zulässig ist. Für den Fall der disjunkten Ressourcenmengen wird im Folgenden beschrieben, unter welchen Umständen die Lösung des Binpackings zulässig ist bezüglich der vorhandenen Ressourcen. Dazu wird zunächst beschrieben unter welchen Voraussetzungen eine Unzulässigkeit überhaupt nur auftreten kann. Anschließend wird erläutert, wie selbst dann teilweise doch noch Zulässigkeit erreicht werden kann.

Voraussetzungen für Unzulässigkeit

O.B.d.A. kann davon ausgegangen werden, dass es für jede Jobgruppe mindestens eine zulässige Ressource gibt. Andernfalls könnten diese Jobs niemals produziert werden. Unzulässigkeit tritt genau dann auf, wenn weniger Ressourcen für eine Jobgruppe g vorhanden sind als sich Jobs aus g gleichzeitig in der Anlage befinden. Grundvoraussetzung dafür ist, dass g beim Binpacking auf mehrere Bins aufgeteilt wurde. Ist dies nicht der Fall, werden die Jobs aus g immer im Abstand von m in die Anlage eingelegt, so dass nur genau eine Ressource benötigt wird, um alle Jobs aus g zu produzieren. Eine Jobgruppe, die nicht geteilt wird, kann also keine Unzulässigkeit verursachen. Es müssen daher nur geteilte Jobgruppen betrachtet werden, von denen es maximal $m-1$ geben kann (wenn die letzte Binpacking-Instanz mit nur noch einem Bin der Größe n erreicht wurde).

Nun soll der Fall betrachtet werden, dass eine Jobgruppe mehrfach geteilt wurde (die Gesamtanzahl von Teilungen von allen Jobgruppen zusammen kann höchstens $m-1$ betragen). Eine Jobgruppe g kann nur dann mehr als einmal geteilt werden, wenn $n_g > \frac{n}{m}$ ist. Bei einer k-fachen Teilung von g gilt für die resultierenden Teilgruppen $n_{g_1}, n_{g_k} \leq \frac{n}{m}$ und $n_{g_i} = \frac{n}{m}$ für $i = 2, \ldots, k-1$ (vgl. Abbildung 3.3). Allen Teilgruppen mit $g_i = \frac{n}{m}$

Abbildung 3.4: Ein 2-Bin mit 6 Jobgruppen. Die Zahlen in den Jobgruppen geben an, wie viele Ressourcen für die Jobgruppe zur Verfügung stehen. Aktuell befindet sich eine Gruppe mit nur einer Ressource an der „Bruchstelle“. Durch Umordnung ist es evtl. möglich, dass eine der beiden Gruppen mit mehr als einer Ressource an diese Stelle gelangt.

kann eine Ressource fest zugeteilt werden, da sich Jobs dieser Teilgruppen durchgehend auf einer Station befinden. Sie können daher als eigenständige Jobgruppen mit genau einer zulässigen Ressource aufgefasst werden. Für g_1 und g_k stehen dann entsprechend weniger Ressourcen zur Verfügung. Daher kann o.B.d.A. davon ausgegangen werden, dass jede Jobgruppe höchstens einmal geteilt werden kann, d.h. $k \leq 1$.

Die Lösung des Binpackings kann also nur dann unzulässig sein, wenn es für eine Jobgruppe g, die geteilt wird, nur eine Ressource gibt. Denn wenn es für g mehr als eine Ressource gibt, ist es kein Problem, wenn zwei Jobs aus g gleichzeitig in der Anlage sind.

Anordnen der Jobgruppen vor der Teilung

Im vorigen Unterabschnitt wurde festgestellt, dass eine geteilte Jobgruppe nur dann zu Unzulässigkeit führen kann, wenn es für sie nur eine Ressource gibt. Wenn ein Bin der Größe $B_j = k\frac{n}{m}$ (im Folgenden ein *k-Bin* genannt) mit den ihm zugewiesenen Jobgruppen aufgeteilt wird in k Bins der Größe $\frac{n}{m}$, stellt sich daher die Frage, ob es möglich ist, die Jobgruppen des k-Bins vorher so anzuordnen, dass sich nur solche Jobgruppen an den „Bruchstellen“ befinden, für die es mehr als eine Ressource gibt (vgl. Abbildung 3.4), denn dann wäre die Lösung auf jeden Fall zulässig.

Satz 1. *Das Problem zu entscheiden, ob die Jobgruppen in einem k-Bin (mit $k > 1$) so angeordnet werden können, dass beim Zerlegen des k-Bins in 1-Bins nur Jobgruppen mit mindestens zwei Ressourcen geteilt werden, ist $\mathcal{NP}$-vollständig.*

Beweis. Dieses Problem liegt trivialer Weise in $\mathcal{NP}$. Es muss also nur noch ein $\mathcal{NP}$-vollständiges Problem darauf reduziert werden. Es folgt die Reduktion des *Partition*-Problems: Gegeben sei eine Instanz J des *Partition*-Problems mit Zahlen $a_1, \ldots, a_n$. Es sei $A \coloneqq \sum_{i=1}^{n} a_i$. Es wird eine Instanz J' des gegebenen Problems erstellt mit $n+1$ Jobgruppen und einem 2-Bin der Größe $A+2$. Es wird $g_i \coloneqq a_i$ für $1 \leq i \leq n$ gesetzt und $g_{n+1} \coloneqq 2$. Für die Jobgruppen 1 bis n gibt es eine Ressource und für die Jobgruppe $n+1$ gibt es zwei Ressourcen. Die Jobgruppen müssen also so angeordnet werden, dass die

Gruppe $n+1$ geteilt wird. Die resultierenden 1-Bins haben dann jeweils die Größe $\frac{A}{2}+1$ und enthalten jeweils eine Jobgruppe der Größe 1, die aus Gruppe $n+1$ hervorgehen. Wir zeigen: J hat genau dann eine Lösung, wenn auch J' eine hat.

„$\Rightarrow$“: J habe eine Lösung, nämlich eine Teilmenge $T \subseteq \{1,\ldots,n\}$ mit $\sum_{i\in T} a_i = \frac{A}{2}$. Dann hat auch J' eine Lösung: Alle $g \in T$ werden dem ersten 1-Bin zugewiesen und alle $g \in \{1,\ldots,n\} \setminus T$ dem zweiten. In beiden 1-Bins ist dann noch genau für eine Jobgruppe der Größe 1 Platz, sodass jeweils eine Hälfte der Gruppe $n+1$ diesen ausfüllen kann.

„$\Leftarrow$“: Habe nun umgekehrt J' eine Lösung. Da jeder 1-Bin eine Jobgruppe der Größe 1 enthält, die aus Gruppe $n+1$ hervorgegangen ist, müssen in beiden 1-Bins die restlichen Jobgruppen zusammen jeweils die Größe $\frac{A}{2}$ haben. Diese Jobgruppen sind dann auch eine Lösung für J. □

In der Praxis kommt es allerdings selten vor, dass es nur eine Jobgruppe mit mehr als einer Ressource gibt. Je mehr Jobgruppen mehr als eine Ressource haben, desto einfacher wird das Problem. Der Fall, dass es keine Jobgruppe mit nur einer Ressource gibt ist trivial, da alle Jobgruppen ohne Bedenken geteilt werden können.

Keine geeignete Anordnung möglich

Es bleibt noch der Fall zu klären, dass in einem k-Bin keine geeignete Reihenfolge der Jobgruppen gefunden werden kann. Es wird also mindestens eine Jobgruppe g geteilt, obwohl es nur eine Ressource für sie gibt.

Zunächst kann versucht werden, das Binpacking und darauf aufbauend das Anordnen der Jobgruppen in den k-Bins zu wiederholen. Evtl. besitzt eine Binpacking-Instanz mehrere Lösungen. Dort werden dann ggf. andere Jobgruppen geteilt, so dass es evtl. keine möglichen Unzulässigkeiten mehr gibt. Falls keine der Lösungen des Binpackings besser ist, kann die nächsten Zahlenpartition von m getestet werden. Je nachdem, ob in der nächsten Zahlenpartition m' verringert wird (vgl. Abbildung 3.2), wird dadurch allerdings eine weitere Jobgruppenteilung und damit ein weiteres Mal Rüstkosten in Kauf genommen.

Alternativ zum Anstieg der Rüstkosten können Bedingungen für die Nachfolgenden Verfahren, die $C_{\max}$ minimieren (s. Abschnitt 3.2), formuliert werden: Falls $n_g \leq \frac{n}{m}$ ist, ist es immer möglich, die beiden Teilgruppen, die aus g hervorgehen, so in den Bins anzuordnen, dass niemals zwei Jobs aus g zur gleichen Zeit in der Anlage sind, indem beispielsweise g_1 an den Anfang ihres Bins gelegt wird und g_2 ans Ende von ihrem Bin (vgl. Abbildung 3.5). Falls mehrere Jobgruppen mit $n_g \leq \frac{n}{m}$ geteilt werden (es können höchstens $m-1$ sein), können in jedem Bin höchstens zwei resultierende Teilgruppen vorkommen, so dass es immer möglich ist, eine von beiden an den Anfang und die andere

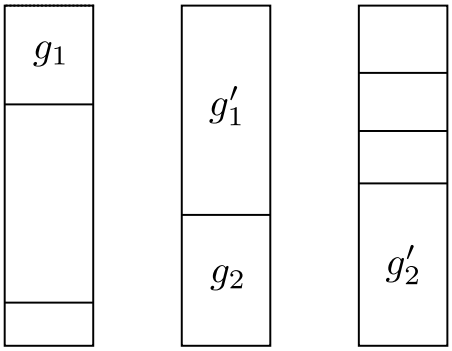

Abbildung 3.5: Eine Instanz mit $m = 3$ Bins, bei der die Jobgruppen g und g' geteilt wurden. Während es bei g_1 und g_2 immer möglich ist, sie so in ihren Bins anzuordnen, dass sie sich nicht überlappen (niemals gemeinsam in der Anlage sind), überlappen sich g_1' und g_2' zwangsweise immer.

an das Ende des Bins zu legen. Diese Bedingungen, dass die resultierenden Teilgruppen sich nicht *überlappen* (vgl. Abbildung 3.5) dürfen, müssen dann in den nachfolgenden Verfahren beachtet werden.

Wenn $n_g > \frac{n}{m}$ ist, gibt es keine Möglichkeit, g so aufzuteilen, dass sich in jedem Zyklus höchstens ein Job aus g in der Anlage befindet, da nur $\frac{n}{m}$ Zyklen gemacht werden (vgl. Abbildung 3.5). Da g zu groß ist für einen Bin, muss g aber zwangsweise irgendwo geteilt werden, so dass bei nur einer geeigneten Ressource definitiv keine zulässige Lösung existiert. Dieser Umstand kann aber bereits im Voraus geprüft werden, so dass dieser Fall hier o.B.d.A. nicht auftritt.

Zusammenfassung

Insgesamt kann also nur sehr selten keine zulässige Ressourcenzuweisung gefunden werden. Definitiv unzulässige Fälle sind häufig unabhängig von der Zuweisung der Jobgruppen an die Bins und können bereits im Voraus erkannt werden. Da im Zweifelsfall sämtliche Lösungen von allen Zahlenpartitionen beim Binpacking durchprobiert werden, ist es ausgeschlossen, dass eine Lösung, bei der nur solche Jobgruppen mit mehr als zwei Ressourcen geteilt werden, nicht gefunden wird, obwohl sie existiert. Falls tatsächlich keine solche Lösung existiert, gibt es noch die Möglichkeit, in den nachfolgenden Verfahren mit zusätzlichen Nebenbedingungen darauf zu achten, dass Teilgruppen sich nicht überlappen.

3.2 Anordnung der Jobgruppen

In diesem Abschnitt werden Verfahren zur Anordnung der Jobgruppen in den Bins einerseits und zum Zuordnen der Bins zu den Stationen andererseits vorgestellt. Aus den vorangegangenen Verfahren (vgl. Abschnitt 3.1) ist bekannt, zu welchen Bins die Jobgruppen gehören. Geteilte Jobgruppen werden hier als eigenständige Gruppen betrachtet. Auch eventuelle zusätzliche Restriktionen, dass die Teile einer geteilten Jobgruppe sich nicht überschneiden dürfen, sind bekannt. Ziel ist es, $C_{\max}$ zu minimieren, ohne die Jobgruppen zu teilen und somit die bereits minimierten Rüstkosten zu verändern.

In Unterabschnitt 3.2.1 und 3.2.2 werden dazu zwei MIPs vorgestellt, die allerdings der Einschränkung unterliegen, dass es nur zwei dominierende Maschinen geben darf. Die in Unterabschnitt 3.2.4 vorgestellte Nachbarschaftssuche hat diesbezüglich keine Einschränkungen.

3.2.1 MIP mit fixierten Bins

Hier wird zunächst ein MIP vorgestellt, das lediglich die Jobgruppen in den Bins anordnet. Die Bins sind dabei bereits fest den Stationen zugeordnet (der erste Bin zur ersten Station usw.). Dieses MIP ist allerdings nur anwendbar bei zwei dominierenden Maschinen, die nicht notwendiger Weise benachbart sein müssen. Ein MIP, das auch die Bins untereinander anordnet, d.h. den Stationen zuordnet, wird in Unterabschnitt 3.2.2 vorgestellt.

Die Idee des MIPs besteht darin, jeder Jobgruppe i in ihrem Bin b_i eine *Startposition* s_i zuzuweisen. Aus der Startposition und der Länge n_i einer Jobgruppe kann ihre *Endposition* bestimmt werden. Anhand dieser Start- und Endpositionen kann dann die Überlappung o_{ij} mit einer anderen Jobgruppen j aus einem anderen Bin b_j berechnet werden. Da die Bins den Stationen fest zugeordnet sind und es nur zwei dominierende Maschinen gibt, deren Abstand bekannt ist, müssen jeweils nur die Überlappungen von Jobs in Bins mit dem Abstand der dominierenden Maschinen berechnet werden. Dazu ein Beispiel: Es gebe $m = 8$ Maschinen, wobei M_3 und M_4 dominierend sind. Jobgruppe 1 habe die Größe $n_1 = 10$ und liege im ersten Bin. Jobgruppe 2 habe die Größe $n_2 = 7$ und liege im zweiten Bin. Da der Abstand der dominierenden Maschinen 1 beträgt, wird der zweite Bin als *Nachbarbin* von Bin 1 bezeichnet. Bin 3 ist der Nachbarbin von Bin 2, Bin 4 ist der Nachbarbin von Bin 3, ... und Bin 1 ist der Nachbarbin von Bin 8. Wenn der Abstand der dominierenden Maschinen hingegen z.B. 3 betragen würde, dann wäre Bin 4 der Nachbarbin von Bin 1 usw. Als erstes wird ein Job aus dem ersten Bin in die Anlage eingelegt, dann einer aus dem zweiten usw. Die Jobs im ersten Bin werden also immer genau in dem Moment von M_4 bearbeitet, wenn die Jobs im zweiten Bin von M_3 bearbeitet werden. Die Startposition von Gruppe 1 sei nun $s_1 = 0$ (als allererstes wird also ein Job aus Gruppe 1 in die Anlage eingelegt) und die Startposition von Gruppe 2

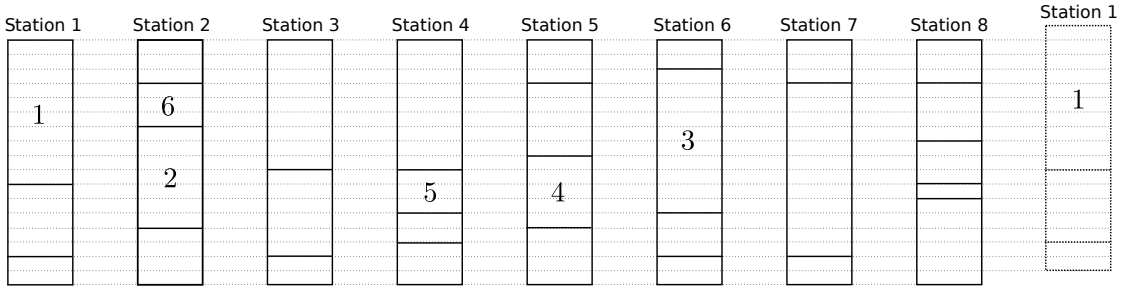

Abbildung 3.6: Visualisierung einer Instanz mit $m = 8$ Bins. Rechts ist der Übergang von Station 8 zu 1 dargestellt, der um 1 versetzt ist. Das liegt daran, dass nach dem i-ten Job im Bin auf Station 8 der $i + 1$-te Job im Bin auf Station 1 folgt.

sei $s_2 = 6$. Die Überlappung der Gruppen 1 und 2 beträgt also 4, wie auch in Abbildung 3.6 zu sehen ist. D.h. es kommt genau viermal vor, dass ein Job aus Gruppe 1 von Maschine M_4 bearbeitet wird, während ein Job aus Gruppe 2 von M_3 bearbeitet wird. Da diese beiden Maschinen dominierend sind, beträgt die Zykluszeit in diesen vier Zyklen $\max\{p_{41}, p_{32}\}$.

Ziel muss es also sein, den Jobgruppen Startpositionen in ihren Bins zuzuweisen und darauf basierend die Überlappungen mit Gruppen aus den Nachbarbins zu berechnen. Aus dem Maximum der Prozesszeiten der Jobgruppen und ihren Überlappungen kann dann $C_{\max}$ bestimmt werden.

Die Berechnung der Überlappungen o_{ij} bildet den Kern dieses MIPs. Es gibt fünf Möglichkeiten, wie sie berechnet werden müssen:

1. $o_{ij} = s_i + n_i - s_j$, falls $s_i \leq s_j < s_i + n_i \leq s_j + n_j$ (z.B. 1 und 2 in Abb. 3.6),
2. $o_{ij} = s_j + n_j - s_i$, falls $s_j \leq s_i < s_j + n_j \leq s_i + n_i$ (z.B. 3 und 4 in Abb. 3.6),
3. $o_{ij} = n_i$, falls $s_j \leq s_i < s_i + n_i \leq s_j + n_j$ (z.B. 5 und 4 in Abb. 3.6),
4. $o_{ij} = n_j$, falls $s_i \leq s_j < s_j + n_j \leq s_i + n_i$ (z.B. 1 und 6 in Abb. 3.6) und
5. $o_{ij} = 0$, falls $s_i + n_i < s_j$ oder $s_j + n_j < s_i$.

Dabei bezieht sich i immer auf den Job im „linken“ Bin und j auf den im „rechten“. Welche davon die passende ist, muss im MIP anhand einiger Hilfsvariablen und Nebenbedingung ermittelt werden. Außerdem gibt es auch noch den Sonderfall, dass zwischen zwei Nachbarbins der Übergang von Station m zu Station 1 liegt. Dieser Fall ist ebenfalls in Abbildung 3.6 rechts veranschaulicht. Wie dort zu sehen ist, muss dann die

Berechnung der Überlappung leicht abgeändert werden, da Bin 1 im Verhältnis zu Bin m um 1 nach oben verschoben ist. Die ersten beiden Möglichkeiten zur Berechnung der Überlappung ändern sich dann zu

1. $o_{ij} = s_i + n_i + 1 - s_j$ und

2. $o_{ij} = s_j + n_j - s_i - 1$.

Die Startpositionen von i werden also jeweils um 1 nach hinten verschoben, bzw. die von j um 1 nach vorn. Zur Vereinfachung wird im MIP ein binärer Parameter σ_{ij} benutzt, der genau dann den Wert 1 hat, wenn dieser Sonderfall eintritt, und sonst 0. Auf diese Weise kann σ_{ij} direkt in die Berechnung von o_{ij} integriert werden:

1. $o_{ij} = s_i + n_i + \sigma_{ij} - s_j$ und

2. $o_{ij} = s_j + n_j - s_i - \sigma_{ij}$.

Um festzustellen, welche der Berechnungen für o_{ij} benutzt werden muss, werden binäre Hilfsvariablen y_{ij} und z_{ij} verwendet. Es ist $y_{ij} = 1$ genau dann, wenn $s_j \geq s_i + \sigma_{ij}$ ist, d.h. j „startet“ gleichzeitig oder später als i. Für z_{ij} gilt in ähnlicher Weise: $z_{ij} = 1$ genau dann, wenn $s_i + n_i + \sigma_{ij} \geq s_j + n_j$, d.h. die Endposition von i liegt hinter der von j oder gleichauf. Aus den vier möglichen Kombinationen der Werte von y_{ij} und z_{ij} lässt sich bestimmen, welche der ersten vier Berechnungsmöglichkeiten für o_{ij} gewählt werden muss. Sollte diese Berechnung einen negativen Wert liefern, tritt der fünfte Fall ein. Für die Jobs 1 und 2 in Abbildung 3.6 wäre beispielsweise $y_{12} = 1$ und $z_{12} = 0$.

Im MIP wird mit $\tilde{N} = \{1, \ldots, \tilde{n}\}$ die Menge der Jobgruppen bezeichnet. $B = \{1, \ldots, m\}$ ist die Menge der Bins und für $b \in B$ ist $\tilde{N}_b \subset \tilde{N}$ die Teilmenge der Jobgruppen, die in Bin b liegen. Für $i \in \tilde{N}$ gibt $b_i \in B$ den Bin an, in dem sich i befindet. $\tilde{N}_b^* \subset \tilde{N}$ bezeichnet die Menge der Jobgruppen im Nachbarbin von b. p_{1i} und p_{2i} sind die Prozesszeiten von Jobgruppe i auf der ersten bzw. zweiten dominierenden Maschine. Für je zwei Jobgruppen $i, j \in N_b$, die im selben Bin b liegen, gibt es zwei Binärvariablen x_{ij}, x_{ji}. Es ist $x_{ij} = 1$ genau dann, wenn Jobgruppe i vor j in dem Bin liegt. Dabei muss i nicht notwendiger Weise direkt vor j liegen, sondern an beliebiger Stelle davor. Mithilfe der x_{ij} lassen sich die Startpositionen s_i bestimmen, mit denen wiederum die Überlappungen o_{ij} wie oben beschrieben berechnet werden:

$$\min \quad \sum_{b \in B} \sum_{\substack{i \in \tilde{N}_b \\ j \in \tilde{N}_b^*}} \max\{p_{2i}, p_{1j}\} o_{ij} \tag{3.5}$$

$$\text{s.t.} \quad x_{ij} + x_{ji} = 1 \qquad i \in \tilde{N}, j \in \tilde{N}_{b_i}, i < j \tag{3.6}$$

$$x_{ij} + x_{jk} \leq x_{ik} + 1 \qquad i \in \tilde{N},\ j, k \in \tilde{N}_{b_i} \tag{3.7}$$

$$s_i = \sum_{\substack{j \in \tilde{N}_{b_i} \\ i \neq j}} n_j x_{ji} \qquad i \in \tilde{N} \qquad (3.8)$$

$$My_{ij} \geq s_j - s_i - \sigma_{ij} + \epsilon \qquad i \in \tilde{N}, j \in \tilde{N}^*_{b_i} \qquad (3.9)$$

$$M(1 - y_{ij}) \geq s_i - s_j + \sigma_{ij} \qquad i \in \tilde{N}, j \in \tilde{N}^*_{b_i} \qquad (3.10)$$

$$Mz_{ij} \geq s_i + n_i - s_j - n_j + \sigma_{ij} + \epsilon \qquad i \in \tilde{N}, j \in \tilde{N}^*_{b_i} \qquad (3.11)$$

$$M(1 - z_{ij}) \geq s_j + n_j - s_i - n_i - \sigma_{ij} \qquad i \in \tilde{N}, j \in \tilde{N}^*_{b_i} \qquad (3.12)$$

$$o_{ij} \geq s_i + n_i - s_j + \sigma_{ij} - M(1 - y_{ij} + z_{ij}) \qquad i \in \tilde{N}, j \in \tilde{N}^*_{b_i} \qquad (3.13)$$

$$o_{ij} \geq s_j + n_j - s_i - \sigma_{ij} - M(1 - y_{ji} + z_{ji}) \qquad i \in \tilde{N}, j \in \tilde{N}^*_{b_i} \qquad (3.14)$$

$$o_{ij} \geq n_j - M(2 - y_{ij} - z_{ij}) \qquad i \in \tilde{N}, j \in \tilde{N}^*_{b_i} \qquad (3.15)$$

$$o_{ij} \geq n_i - M(2 - y_{ji} - z_{ji}) \qquad i \in \tilde{N}, j \in \tilde{N}^*_{b_i} \qquad (3.16)$$

$$o_{ij} \geq 0 \qquad i \in \tilde{N}, j \in \tilde{N}^*_{b_i} \qquad (3.17)$$

$$x_{ij} \in \{0, 1\} \qquad i \in \tilde{N}, j \in \tilde{N}_{b_i} \qquad (3.18)$$

$$y_{ij}, z_{ij} \in \{0, 1\} \qquad i \in \tilde{N}, j \in \tilde{N}^*_{b_i} \qquad (3.19)$$

Mit (3.6) wird erreicht, dass immer entweder i vor j liegt oder umgekehrt. (3.7) stellt die Transitivität dieser Beziehung her. Auf Grundlage der x_{ij} und der Größen der Jobgruppen n_i kann in (3.8) die Startposition jeder Jobgruppe berechnet werden. Die Variablen s_i dienen dabei nur der Übersicht und können in den übrigen Nebenbedingungen durch die Summe aus (3.8) ersetzt werden. Über (3.9) und (3.10) wird den Indikatorvariablen y_{ij} mit einem „Big-M“ die oben beschriebene Bedeutung gegeben: (3.9) setzt y_{ij} auf 1, wenn $s_j - s_i - \sigma_{ij} + \epsilon > 0$ ist, was äquivalent zu $s_j \geq s_i + \sigma_{ij}$ ist. ϵ ist ein hinreichend kleiner Parameter, falls alle Prozesszeiten ganzzahlig sind, ist jeder Wert $\epsilon < 1$ geeignet. (3.10) andererseits setzt y_{ij} auf 0, wenn $s_i - s_j + \sigma_{ij} > 0$ ist (oder äquivalent dazu: $s_j < s_i + \sigma_{ij}$). Analog wird mit (3.11) und (3.12) der Wert der z_{ij} entsprechend der obigen Bedeutung gesetzt. Je nachdem, welche der y_{ij} und z_{ij} auf 1 bzw. 0 gesetzt sind, werden drei der vier Nebenbedingungen (3.13) bis (3.16) durch die „Big-Ms“ faktisch bedeutungslos ($o_{ij} \geq$ „ein negativer Wert“) und nur eine bleibt bestehen, so dass genau die passende der vier Berechnungsmöglichkeiten für die o_{ij} gewählt wird. Für $y_{ij} = 1$ und $z_{ij} = 0$, wie bei den Jobs 1 und 2 in Abbildung 3.6 der Fall, würde beispielsweise das „Big-M“ aus (3.13) verschwinden, so dass die Bedingung $o_{ij} \geq s_i + n_i - s_j$ „übrigbleibt“. Da die Zielfunktion (3.5) die Überlappungen (mit dem Maximum der jeweiligen Prozesszeiten multipliziert) minimiert, wird daraus de facto $o_{ij} = s_i + n_i - s_j$ und die o_{ij} müssen auch nicht per Nebenbedingung nach oben beschränkt werden. Sie sind automatisch durch die Größe der Bins beschränkt. Damit sie nicht negativ werden, sind sie allerdings durch (3.17) nach unten durch 0 beschränkt. Dies bedeutet, dass die jeweiligen Jobgruppen sich nicht überschneiden.

Die Qualität der Lösung dieses MIPs ist stark von der gegebenen Zuordnung der Bins an die Stationen abhängig. Bei einigen Zuordnungen kann es sehr gute Lösungen geben, und bei anderen dagegen nur sehr schlechte, was sich nur durch eine Änderung

der Zuordnung (bzw. der Binreihenfolge) verbessern lässt, wozu dieses MIP allerdings nicht in der Lage ist. Im Folgenden wird daher ein weiteres MIP vorgestellt, das diese Fähigkeit besitzt, und heuristische Methoden, die die Bins den Stationen zuweisen, um darauf basierend dieses MIP anzuwenden.

3.2.2 MIP mit freien Bins

Das folgende MIP erweitert das MIP aus Unterabschnitt 3.2.1 um die Fähigkeit, die Bins untereinander zu vertauschen, was dem Zuweisen der Bins an die Stationen entspricht. Für die Berechnung der Überschneidungen zwischen den Jobgruppen sind daher weitere Variablen notwendig, die als Indikator dafür dienen, ob die Bins der Jobgruppen auf benachbarten Stationen liegen. Dazu dienen die Binärvariablen β_{bd}, die genau dann auf 1 gesetzt werden, wenn Bin b Station d zugewiesen ist. Weitere Binärvariablen ν_{bcd} werden auf 1 gesetzt, wenn Bin b Station d zugewiesen ist und Bin c der Nachbarstation von d (mit d^* bezeichnet). Auf diese Weise können die ν_{bcd} die σ_{ij} aus dem MIP in Unterabschnitt 3.2.1 ersetzen: Wenn die beiden dominierenden Maschinen benachbart sind (also den Abstand 1 haben), ist für zwei Jobgruppen i und j $\nu_{b_i b_j m} = 1$ genau dann, wenn der Bin von i der letzten Station m zugewiesen ist und der Bin von j der ersten Station (also genau dann, wenn im anderen MIP $\sigma_{ij} = 1$ ist). Außerdem gibt es weitere Binärvariablen w_{ij}, die als Indikator dafür dienen, ob die Bins der Jobgruppen i und j benachbarten Stationen zugewiesen sind. Denn nur dann muss zwischen diesen Jobgruppen die Überlappung o_{ij} berechnet werden.

$$\min \quad \sum_{\substack{i,j \in \tilde{N} \\ b_i \neq b_j}} \max\{p_{2i}, p_{1j}\} o_{ij} \tag{3.20}$$

$$\text{s.t.} \quad x_{ij} + x_{ji} = 1 \qquad i \in \tilde{N}, j \in \tilde{N}_{b_i} \tag{3.21}$$

$$x_{ij} + x_{jk} \leq x_{ik} + 1 \qquad i \in \tilde{N},\ j,k \in \tilde{N}_{b_i} \tag{3.22}$$

$$s_i = \sum_{\substack{j \in \tilde{N}_{b_i} \\ i \neq j}} g_j x_{ji} \qquad i \in \tilde{N} \tag{3.23}$$

$$\sum_{b \in B} \beta_{bd} = 1 \qquad d \in B \tag{3.24}$$

$$\sum_{d \in B} \beta_{bd} = 1 \qquad b \in B \tag{3.25}$$

$$\beta_{bd} + \beta_{cd^*} \leq \nu_{bcd} + 1 \qquad b,c,d \in B \tag{3.26}$$

$$\nu_{bcd} \leq \beta_{bd} \qquad b,c,d \in B \tag{3.27}$$

$$\nu_{bcd} \leq \beta_{bd^*} \qquad b,c,d \in B \tag{3.28}$$

$$M y_{ij} \geq s_j - s_i - \nu_{b_i b_j m} + \epsilon \qquad i,j \in \tilde{N} \tag{3.29}$$

$$M(1 - y_{ij}) \geq s_i + \nu_{b_i b_j m} - s_j \qquad i,j \in \tilde{N} \tag{3.30}$$

$$Mz_{ij} \geq s_i + n_i + \nu_{b_i b_j m} - s_j - n_j + \epsilon \qquad i,j \in \tilde{N} \qquad (3.31)$$

$$M(1 - z_{ij}) \geq s_j + n_j - s_i - n_i - \nu_{b_i b_j m} \qquad i,j \in \tilde{N} \qquad (3.32)$$

$$w_{ij} = \sum_{d \in B} \nu_{b_i b_j d} \qquad i,j \in \tilde{N} \qquad (3.33)$$

$$o_{ij} \geq s_i + n_i + \nu_{b_i b_j m} - s_j - M(2 - y_{ij} + z_{ij} - w_{ij}) \qquad i,j \in \tilde{N} \qquad (3.34)$$

$$o_{ij} \geq s_j + n_j - s_i - \nu_{b_i b_j m} - M(2 + y_{ij} - z_{ij} - w_{ij}) \qquad i,j \in \tilde{N} \qquad (3.35)$$

$$o_{ij} \geq n_j - M(3 - y_{ij} - z_{ij} - w_{ij}) \qquad i,j \in \tilde{N} \qquad (3.36)$$

$$o_{ij} \geq n_i - M(1 + y_{ij} + z_{ij} - w_{ij}) \qquad i,j \in \tilde{N} \qquad (3.37)$$

$$o_{ij} \geq 0 \qquad i,j \in \tilde{N} \qquad (3.38)$$

$$\beta_{bd} \in \{0,1\} \qquad b,d \in B \qquad (3.39)$$

$$\nu_{bcd} \in \{0,1\} \qquad b,c,d \in B, b \neq c \qquad (3.40)$$

$$x_{ij} \in \{0,1\} \qquad i \in \tilde{N}, j \in \tilde{N}_{b_i} \qquad (3.41)$$

$$y_{ij}, z_{ij}, w_{ij} \in \{0,1\} \qquad i,j \in \tilde{N} \qquad (3.42)$$

Die Nebenbedingungen (3.24) und (3.25) gewährleisten, dass jeder der m Bins genau einer der m Stationen zugewiesen wird. Über die Nebenbedingungen (3.26), (3.27) und (3.28) wird erzwungen, dass die Binärvariablen ν_{bcd} genau dann auf 1 gesetzt werden, wenn Bin b Station d zugewiesen ist und zugleich Bin c der Nachbarstation d^*. In (3.33) wird w_{ij} auf 1 gesetzt, falls die Bins von i und j benachbarten Stationen zugewiesen sind. Wenn $w_{ij} = 0$ ist, verlieren die Nebenbedingungen (3.34) bis (3.37) ihre Bedeutung. Davon abgesehen sind diese Nebenbedingungen aber identisch zu denen aus dem MIP im vorherigen Unterabschnitt. Ähnlich wie die s_i dienen die w_{ij} nur der Übersicht und können durch die Summe in (3.33) ersetzt werden.

In dieser Form verlangt das MIP, dass die beiden dominierenden Maschinen benachbart sein müssen. Denn dann gibt es nur zwei Stationen, zwischen denen die Berechnung der Überlappung um 1 versetzt stattfinden muss. Wenn die dominierenden Maschinen beispielsweise einen Abstand von 2 haben, tritt dieser Sonderfall der Berechnung sowohl zwischen den Station $m-1$ und 1 auf als auch zwischen m und 2, was durch die Variablen $\nu_{b_i,b_j,m-1}$ bzw. $\nu_{b_i b_j m}$ erkannt werden kann. Diese müssten dann auch beide in die Nebenbedingungen (3.29) bis (3.35) einfließen. Da immer höchstens eine von beiden gleichzeitig auf 1 gesetzt sein kann, bewirken dann diese beiden Variablen zusammen die um 1 versetzte Berechnung der Überlappung. Nebenbedingung (3.29) z.B. sähe dann so aus:

$$My_{ij} \geq s_j - s_i - \nu_{b_i b_j m} - \nu_{b_i,b_j,m-1} + \epsilon \quad i,j \in \tilde{N}$$

Das Lösen dieses MIPs mit CPLEX dauert sehr viel länger als beim MIP mit fixierten Bins. Allerdings ist die obere Schranke, also die aktuell beste gefundene Lösung, meist

schon nach wenigen Minuten besser als die beste Lösung mit fixierten Bins (abhängig davon, ob die gegebene Binreihenfolge dort zufällig „gut“ ist).

3.2.3 Mehrere fixierte Reihenfolgen

Das MIP in Unterabschnitt 3.2.2 liefert zwar theoretisch optimale Lösungen, diese zu berechnen benötigt aber zu viel Zeit. Schon bei kleinen Instanzen mit $\tilde{n} = 10$ Jobgruppen und $m = 8$ beträgt der LP-Gap nach einer Stunde Berechnung noch 20%. Man beachte, dass es bei 10 Jobgruppen und 8 Bins mindestens 6 Bins gibt mit genau einer Jobgruppe. Es müssen also im Wesentlichen nur noch die Bins den Stationen zugeordnet werden und trotzdem ist die Laufzeit so lang.

Das MIP mit fixierten Bins hingegen findet erst ab Instanzen mit ca. $\tilde{n} = 25$ nicht mehr die optimale Lösung innerhalb von einer Stunde. Je nach Größe der Jobgruppen kann dies ein beträchtlicher Unterschied zu $\tilde{n} = 10$ sein. Allerdings ist die Güte der Lösungen des MIPs mit fixierten Bins von der initialen Binreihenfolge abhängig. Dazu wird in diesem Unterabschnitt versucht, solch eine initiale Binreihenfolge zu erzeugen, mit der das MIP dann arbeiten kann.

Benutzung beider MIPs

Beobachtungen der Leistung der beiden MIPs haben ergeben, dass das MIP mit freien Bins oft schon nach relativ kurzer Zeit eine bessere obere Schranke (d.h. die beste bis zu diesem Zeitpunkt gefundene Lösung) liefert als das MIP mit fixierten Bins. Daher ist ein Ansatz, das MIP mit freien Bins für eine gewisse Zeit laufen zu lassen und dann die Zuteilung der Bins zu den Stationen auszulesen (diese ist in den β_{bd}-Variablen kodiert), um anschließend mit dieser Binreihenfolge das MIP mit fixierten Bins zu starten.

Optimum zwischen zwei Bins

Betrachtet man zwei beliebige Bins a und b auf benachbarten Stationen, ist es möglich, all jene Zyklen t zu betrachten, in denen ein Job aus a auf der zweiten dominierenden Maschine ist und ein Job aus b auf der ersten. Die zugehörigen Zykluszeiten c_t werden dann durch die Prozesszeiten dieser Jobs auf den beiden dominierenden Maschinen definiert.Die Summe dieser Zykluszeiten zwischen zwei benachbarten Bins a und b wird mit C_{ab} bezeichnet.

Satz 2. *Seien a und b zwei Bins auf benachbarten Stationen. Wenn die Jobgruppen in a absteigend nach ihren Prozesszeiten auf der zweiten dominierenden Maschine angeordnet werden und die Jobgruppen in b absteigend nach ihren Prozesszeiten auf der ersten*

dominierenden Maschine, dann ist C_{ab} minimal.

Beweis. Für den Beweis werden nicht die Jobgruppen, sondern die einzelnen Jobs betrachtet, was aber für die zu betrachtende Sortierung keinen Unterschied macht. Sei C^*_{ab} die Summe der Zykluszeiten zwischen a und b, wenn diese wie gefordert sortiert sind. Wir nehmen an, es gäbe eine andere Anordnung der Jobs mit $C_{ab} < C^*_{ab}$ und führen dies zu einem Widerspruch. Da die gegeben Sortierung in der anderen Anordnung nicht vorliegt, muss es dort in a oder b zwei benachbarte Jobs entgegen dieser Sortierung geben. Seien j^a_1 und j^a_2 diese Jobs und o.B.d.A. seien sie in Bin a. Die Jobs die an gleicher Position in Bin b sind, werden mit j^b_1 und j^b_2 bezeichnet. Falls auch j^b_1 und j^b_2 entgegen dieser Sortierung angeordnet sind, vertauschen wir sowohl j^a_1 mit j^a_2 als auch j^b_1 mit j^b_2 und ändern dadurch C_{ab} nicht. Insbesondere hat sich C_{ab} nicht verschlechtert, dadurch, dass wir uns der gewünschten Reihenfolge der Jobs in den Bins annähern.

Andernfalls, wenn j^b_1 und j^b_2 bereits in der gewünschten Reihenfolge sind, verwenden wir zur besseren Lesbarkeit die Abkürzungen $p_{1a} \coloneqq p_{2j^a_1}, p_{2a} \coloneqq p_{2j^a_2}, p_{1b} \coloneqq p_{1j^b_1}$ und $p_{2b} \coloneqq p_{1j^b_2}$. Es gilt also nach Annahme $p_{1a} < p_{2a}$ und $p_{1b} \geq p_{2b}$. Die entsprechenden beiden Zykluszeiten werden über $c_{t_1} = \max\{p_{1a}, p_{1b}\}$ und $c_{t_2} = \max\{p_{2a}, p_{2b}\}$ berechnet. Werden nun j^a_1 und j^a_2 vertauscht, ändert sich der Wert von C_{ab} zu

$$C'_{ab} = C_{ab} - \max\{p_{1a}, p_{1b}\} - \max\{p_{2a}, p_{2b}\} + \max\{p_{2a}, p_{1b}\} + \max\{p_{1a}, p_{2b}\}.$$

Wir betrachten nun zwei Fälle:

1. $p_{1a} \geq p_{1b}$: Es ist dann $p_{2b} \leq p_{1b} \leq p_{1a} < p_{2a}$ und somit

   $$C'_{ab} = C_{ab} - p_{1a} - p_{2a} + p_{2a} + p_{1a} = C_{ab}.$$

   C_{ab} verschlechtert sich in diesem Fall also nicht.

2. $p_{1a} < p_{1b}$: In diesem Fall unterscheiden wir wieder zwei Fälle:

   a) $p_{2a} \leq p_{2b}$: Es ist dann $p_{1a} < p_{2a} \leq p_{2b} \leq p_{1b}$ und somit

      $$C'_{ab} = C_{ab} - p_{1b} - p_{2b} + p_{1b} + p_{2b} = C_{ab}.$$

      C_{ab} verschlechtert sich in diesem Fall also ebenfalls nicht.

   b) $p_{2a} > p_{2b}$: Es ist dann $p_{1b}, p_{2a} \geq p_{1a}, p_{2b}$ und somit

      $$\begin{aligned} C'_{ab} &= C_{ab} - p_{1b} - p_{2a} + \max\{p_{2a}, p_{1b}\} + \max\{p_{1a}, p_{2b}\} \\ &= C_{ab} - \min\{p_{1b}, p_{2a}\} + \max\{p_{1a}, p_{2b}\} \\ &\leq C_{ab}. \end{aligned}$$

      Also verschlechtert sich C_{ab} auch hier nicht.

Insgesamt verschlechtert sich C_{ab} also niemals, während die Jobs in den Bins absteigend nach ihren Prozesszeiten sortiert werden. Das ist ein Widerspruch zu der Annahme, dass die ursprüngliche Anordnung besser war als die sortierte ($C_{ab} < C^*_{ab}$). □

Wie auch schon im Beweis geschehen, wird die Bezeichnung C^*_{ab} für ein minimales C^*_{ab} verwendet. Dieses Optimum zwischen zwei Bins ist auf die Anordnung aller m Bins übertragen aber nur ein lokales: Wird ein Bin bezüglich eines seiner Nachbarn in dieser Weise angeordnet, ist es im Allgemeinen nicht der Fall, dass diese Anordnung auch bezüglich des anderen Nachbarn optimal ist.

In den folgenden Absätzen werden einige Anwendungen dieses lokalen Optimierens vorgestellt.

Bestimmung einer unteren Schranke

Eine Anwendungsmöglichkeit der lokal optimalen Bins ist die Berechnung einer unteren Schranke für die Summe der Zykluszeiten im zweiten Dekompositionsansatz. Eine mögliche untere Schranke lässt sich wie folgt berechnen:

$$LB_{\sum c_t} = \frac{1}{2} \sum_{b=1}^{m} \left(\min_{a \neq b} C^*_{ab} + \min_{c \neq b} C^*_{bc} \right)$$

Es werden also für jeden Bin b sein optimaler linker Nachbar a und sein optimaler rechter Nachbar c gesucht. Da in der Summe all dieser C^*_{ab}-Werte jeder Bin zweimal (einmal als linker und einmal als rechter Nachbar) auftaucht, wird diese Summe noch halbiert. Zusammen mit den auftretenden Rüstkosten, die nach der Zuweisung der Jobgruppen an die Bins feststehen (vgl. Abschnitt 3.1), ist dies eine untere Schranke für die Zielfunktion $C_{\max}$.

Heuristik für initiale Binreihenfolge

Eine andere mögliche Anwendung der lokal optimalen Bins ist, eine initiale Binreihenfolge (bzw. eine initiale Zuweisung der Bins zu den Stationen) auf heuristischem Wege zu finden, mit der dann das MIP mit fixierten Bins arbeiten kann. Als erster Schritt wird unter allen möglichen C^*_{ab}-Werten (für alle Bins a, b) das Minimum gesucht. Die zugehörigen Bins a und b werden der ersten Station und der Nachbarstation der ersten Station zugewiesen. Anschließend wird unter allen noch nicht zugewiesenen Bins derjenige Bin c gesucht, der den besten C^*_{bc}-Wert aufweist. Da c ein „vielversprechender“ Nachbar für b ist, wird er der Nachbarstation von b zugewiesen. Dieses Vorgehen wird so lange wiederholt, bis jeder Bin einer Station zugewiesen ist.

Falls der Abstand der dominierenden Maschinen d_{dom}(=Abstand benachbarter Stationen) und die Anzahl der Maschinen m nicht teilerfremd sind, gibt es unter den Stationen disjunkte Gruppen, die untereinander nicht durch Nachbarschaft erreicht werden können. Beispielsweise bilden bei $m = 8$ und $d_{dom} = 2$ die Stationen $1, 3, 5, 7$ und $2, 4, 6, 8$ solche disjunkten Gruppen. In diesem Fall muss das obige Verfahren für jede dieser Gruppen separat durchgeführt werden.

Heuristik zum Anordnen der Jobgruppen in den Bins

Die lokal optimalen Bins können auch benutzt werden, um auf heuristischem Wege die Jobgruppen in den Bins bei gegebener Binreihenfolge anzuordnen. Dazu werden jeweils vier direkt benachbarte Bins a, b, c, d gesucht (a ist linker Nachbar von b, b ist linker Nachbar von c und c ist linker Nachbar von d). Seien C_{ab}, C_{bc} und C_{cd} die aktuellen Summen der Zykluszeiten zwischen diesen Bins. Nun werden die Jobgruppen in b und c so sortiert, dass $C_{bc} = C^*_{bc}$ wird. Dadurch ändern sich auch die Zykluszeiten zwischen a und b zu C'_{ab} und die zwischen c und d zu C'_{cd}. Wenn $C'_{ab}+C^*_{bc}+C'_{cd} < C_{ab}+C_{bc}+C_{cd}$ ist, wird die Änderung übernommen, andernfalls verworfen. Dieses Vorgehen wird so lange wiederholt, bis keine benachbarten Bins a, b, c, d mehr gefunden werden, die auf diese Weise optimiert werden können.

3.2.4 Nachbarschaftssuche

Als Alternative zu den MIPs mit freien oder fixierten Bins lassen sich Nachbarschaftsoperatoren beschreiben, die für diverse Nachbarschaftssuchverfahren benutzt werden können. Naheliegende Operatoren sind:

- bch_{bc}: vertauscht die Bins an den Stationen b und c miteinander,
- jch_{bjk}: vertauscht im Bin an Station b die Jobgruppen j und k miteinander.

Es wurde ein Simulated Annealing Algorithmus implementiert, der diese beiden Nachbarschaften benutzt. Dieser ähnelt dem Simulated Annealing aus Abschnitt 2.1.3 und unterscheidet sich bloß in einem Punkt von diesem: In jeder Iteration wird mit einer Wahrscheinlichkeit von $\frac{m}{\tilde{n}+m}$ eine neue Lösung aus der bch_{bc}-Nachbarschaft erzeugt (mit zufällig gewählten b und c) und mit einer Wahrscheinlichkeit von $\frac{\tilde{n}}{\tilde{n}+m}$ eine Lösung aus der jch_{bjk}-Nachbarschaft (mit ebenfalls zufälligen b, j, k). Auf diese Weise wird je nach Verhältnis zwischen der Anzahl der Jobgruppen ($\tilde{n}$) und der Bins (m) der entsprechende Operator häufiger benutzt.

Im Gegensatz zu den beiden MIPs sind Lösungsansätze, die auf diesen Nachbarschaftsoperatoren basieren, nicht darauf angewiesen, dass nur zwei Maschinen dominierend sind.

Ebenso müssen die Rüstkosten nicht konstant sein, wie von den MIPs verlangt, sondern können beliebig sein.

3.3 Algorithmenvarianten

Im ersten Abschnitt dieses Kapitels (3.1) wurde ein Verfahren vorgestellt, das die Jobgruppen m Bins zuordnet und im Fall der Teilung einer Jobgruppe auf Zulässigkeit achtet, um so die minimale Anzahl an Rüstkosten zu verursachen. Dieser Schritt ist essenziell für den zweiten Dekompositionsansatz, da sämtliche Verfahren, die im zweiten Abschnitt 3.2 vorgestellt wurden, auf dieser Zuordnung operieren und sie nachträglich nicht mehr ändern. Da die Verfahren in Abschnitt 3.2 aber sehr vielfältig sind, werden hier noch einmal alle Varianten der Zuordnung der Bins an die Stationen und der Anordnung der Jobgruppen in den Bins zusammengefasst:

1. Wende ausschließlich das MIP mit fixierten Bins an für eine Zeit t.

2. Wende das MIP mit freien Bins für eine Zeit t_1 an und lese die bis dahin berechnete Zuordnung der Bins an die Stationen aus. Wende das MIP mit fixierten Bins mit dieser Zuordnung für eine Zeit t_2 an.

3. Wende die Heuristik für eine initiale Binreihenfolge (3.2.3) an. Wende das MIP mit fixierten Bins mit dieser Zuordnung für eine Zeit t an.

4. Wende das MIP mit freien Bins für eine Zeit t an und lese die bis dahin berechnete Zuordnung der Bins an die Stationen aus. Wende die Heuristik zum Anordnen der Jobgruppen in den Bins (3.2.3) an auf diese Zuordnung an.

5. Wende die Heuristik für eine initiale Binreihenfolge (3.2.3) an. Wende die Heuristik zum Anordnen der Jobgruppen in den Bins (3.2.3) an auf diese Zuordnung an.

6. Wende den Simulated-Annealing-Ansatz (3.2.4) an.

Bis auf das Simulated-Annealing bestehen alle Ansätze aus zwei Stufen: In der ersten Stufe werden die Bins untereinander angeordnet und in der zweiten Stufe werden die Jobgruppen in den Bins angeordnet. Diese Ansätze werden in Kapitel 4 evaluiert.

4 Rechenergebnisse und Vergleiche

In diesem Kapitel werden Rechenergebnisse vorgestellt und miteinander verglichen. Es wurden sowohl generierte Instanzen getestet als auch reale. Getestet wurde auf einem PC mit Intel(R) Xeon(R) CPU E3-1230 v3, 8 Kerne @ 3.30GHz, und 16GB RAM unter Ubuntu 13.10. Die MIPs wurden mit ZIMPL [Koc04] und CPLEX v12.6 gelöst. Die übrigen Algorithmen wurden in C++ implementiert und mit dem GCC v4.8.1 mit O2-Optimierung kompiliert.

In Abschnitt 4.1 werden die in Abschnitt 2.1.3 vorgestellten Heuristiken evaluiert und mit drei Heuristiken verglichen, die Soylu et al. in ihrem Paper [SKA07] präsentieren.

In Abschnitt 4.2 werden die beiden Dekompositionsansätze miteinander verglichen.

4.1 Vergleich der Heuristiken

In diesem Abschnitt werden die Rechenergebnisse der Heuristiken aus Unterabschnitt 2.1.3 vorgestellt und miteinander verglichen. Es wurden vier eigene Testreihen mit zufälligen Instanzen erzeugt und auch eine Testreihe wie in [SKA07] beschrieben.

4.1.1 Eigene Testinstanzen

Die vier hier vorgestellten Testreihen wurden mit $m = 8$ Maschinen und $n = 50, 100, 150, \ldots, 5000$ Jobs erzeugt.

In der ersten Testreihe sind die Maschinen M_3, M_4, M_5 dominierend. Ihre Prozesszeiten wurden gleichverteilt aus dem Intervall $[10, 100]$ gewählt. Die Prozesszeiten der übrigen Maschinen wurden aus dem Intervall $[0, 9]$ gleichverteilt gewählt. In Abbildung 4.1 sind die Resultate aufgetragen. Oben sind die Summe der Zykluszeiten in Abhängigkeit von der Instanzgröße n aufgetragen. Da die Resultate sehr ähnlich sind, so dass die Kurven teilweise sehr dicht beieinander liegen, sind unten jeweils die relativen Differenzen $\frac{C_{\max,h} - C_{\max,nfs}}{C_{\max,nfs}}$ der übrigen Heuristiken (h) zur Non-Full-Schedule-Heuristik (nfs) aufgetragen. CPLEX LB und UB bezeichnen die untere und obere Schranke, die CPLEX nach 30 Minuten errechnet hat. Dabei ist die obere Schranke der Zielfunktionswert ei-

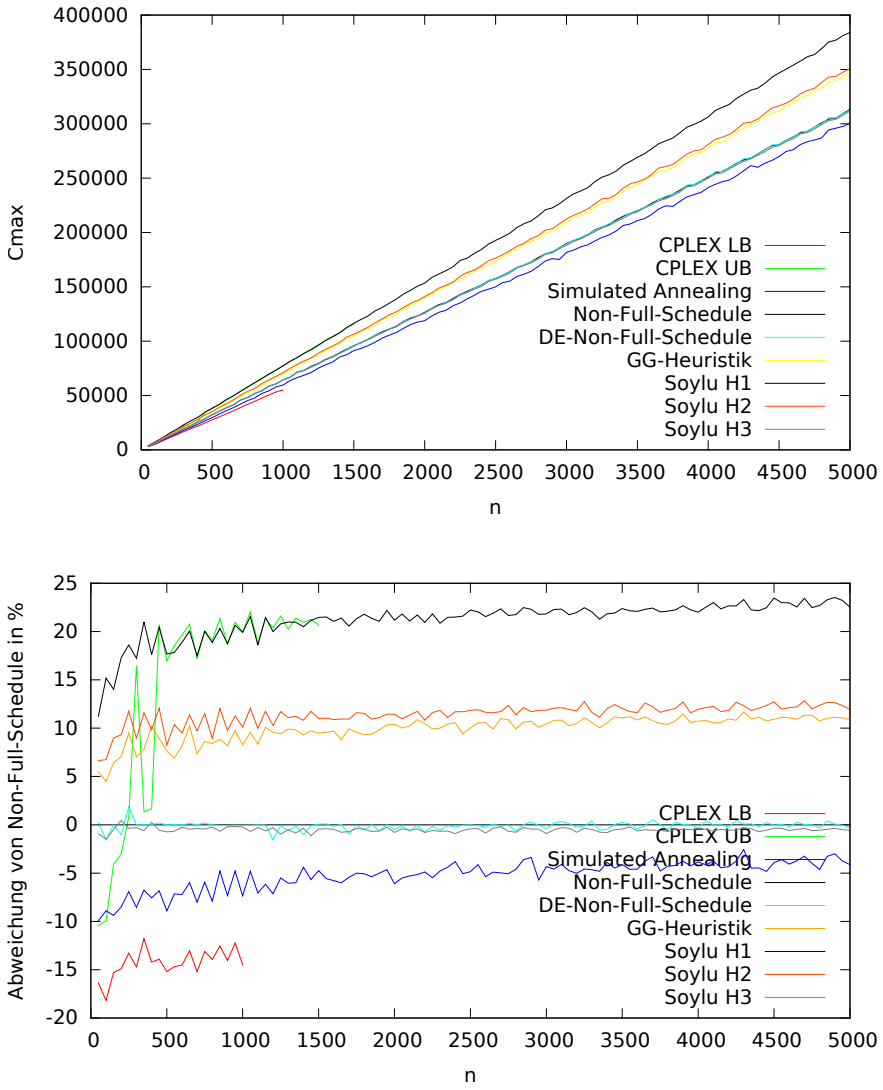

Abbildung 4.1: Resultate der Heuristiken für die Summe der Zykluszeiten (oben) und relative Unterschiede zwischen der Non-Full-Schedule-Heuristik und den übrigen Heuristiken (unten) bei drei dominierenden Maschinen (gleichverteilte Prozesszeiten).

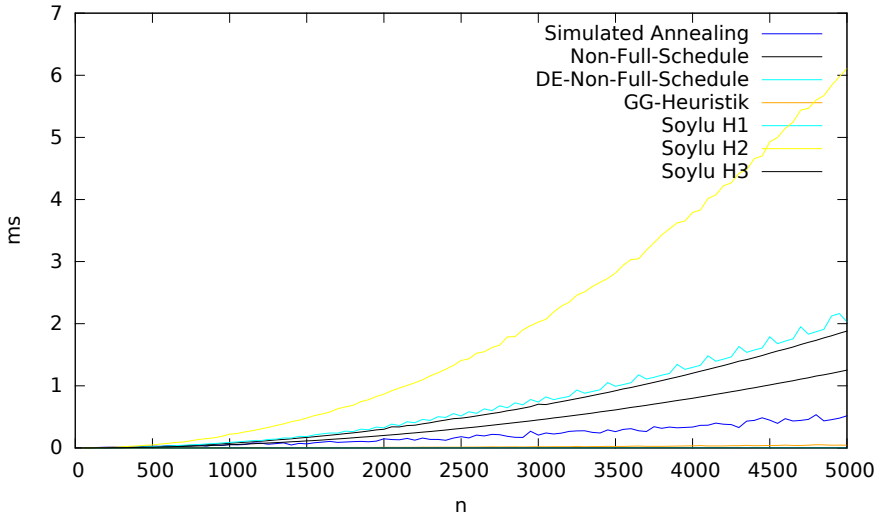

Abbildung 4.2: Laufzeiten der drei Heuristiken bei drei dominierenden Maschinen (gleichverteilte Prozesszeiten).

ner tatsächlich errechneten Lösung und kann daher als Heuristik verwendet werden. Zu sehen ist, dass die Lösungen des MIPs bei sehr kleinen Instanzen ($n < 300$) nach einer halben Stunde Rechenzeit vergleichbar mit den Lösungen der übrigen Heuristiken sind. Allerdings benutzt CPLEX alle 8 Prozessorkerne parallel, während die übrigen Heuristiken nicht parallel implementiert sind. Außerdem liegen die Rechenzeiten der übrigen Heuristiken weit unter einer halben Stunde (s. Abbildung 4.2). Aus diesen Gründen wurde das MIP auch nur bis zu Instanzgrößen von $n = 1500$ berechnet.

Außerdem ist zu erkennen, dass die Double Ended Non-Full-Schedule-Heuristik fast identische Resultate liefert im Vergleich zur Non-Full-Schedule-Heuristik. H_3 von Soylu ist meist etwas besser als als diese beiden. Die Gilmore-Gomory-Heuristik liefert hier relativ schlechte Resultate. Schlechter sind nur H_2 von Soylu und mit einigem Abstand H_3. Mit Simulated Annealing konnten hier offenbar die besten Lösungen erzeugt werden. Der Plot der relativen Abweichungen zeigt außerdem, dass diese Abweichungen offenbar nahezu konstant sind zwischen den Heuristiken (nicht beim MIP) und nicht von der Instanzgröße abhängen. Einzig die relative Abweichung des Simulated Annealings scheint mit zunehmender Instanzgröße leicht abzunehmen.

In Abbildung 4.2 sind die Zeiten aufgetragen, die die Heuristiken benötigten, um die in Abbildung 4.1 dargestellten Resultate zu berechnen. Deutlich zu erkennen ist der quadratische Anstieg der beiden Non-Full-Schedule-Heuristiken und Soylus H_3, wobei die Double Ended Variante etwas langsamer ist, da pro Iteration immer zwei Jobs statt nur einem gesucht werden, und H_3 etwas schneller, da kein zweites Vergleichskriterium angewandt wird. Außerdem treten in der Kurve der Double Ended Non-Full-Schedule-

Heuristik teilweise kleine Ausreißer nach oben auf. Eine plausible Begründung hierfür konnte nicht gefunden werden. Die Kurve des Simulated Annealings ist, wie zu erwarten war, sehr chaotisch, was durch die nicht deterministische Funktionsweise des Algorithmus erklärt werden kann: Die Laufzeit hängt stark vom Zufall ab und nur zweitrangig von der Instanzgröße. Am schnellsten sind die Gilmore-Gomory-Heuristik und H_1, die dafür aber beide vergleichsweise schlechte Resultate liefern. H_2 hingegen ist mit Abstand am langsamsten trotz der eher schlechten Resultate.

Die zweite Testreihe unterscheidet sich nur dadurch von der ersten, dass die Prozesszeiten der dominierenden Maschinen nicht gleichverteilt aus dem Intervall $[10, 100]$ ausgewählt wurden, sondern normalverteilt mit einem Erwartungswert von $\mu = 50$ und einer Standardabweichung von $\sigma = 20$. Die Resultate sind analog zur ersten Testreihe in Abbildung 4.3 dargestellt und die Laufzeiten dazu in Abbildung 4.4.

Die Laufzeiten der zweiten Testreihe unterscheiden sich nur marginal von denen der ersten, was zu erwarten war, da die Laufzeiten der Heuristiken nicht von den Prozesszeiten abhängig sind. Die Kurven in Abbildung 4.3 unterscheiden sich im Verlauf kaum von denen in Abbildung 4.1. Allerdings sind diese Kurven im Vergleich zur ersten Testreihe in etwa um den Faktor 2 gestaucht. Die Resultate der Heuristiken unterscheiden sich hier also weniger. Das muss daran liegen, dass die Prozesszeiten (da sie normalverteilt sind) weniger unterschiedlich sind. Die beiden Non-Full-Schedule-Heuristiken, H_2 und H_3 ziehen daraus einen Vorteil, da sie nach eben diesem Prinzip vorgehen, möglichst ähnliche Prozesszeiten zu finden. Auch die Gilmore-Gomory-Heuristik profitiert davon, da beim (theoretischen) Wiederhochskalieren der Prozesszeiten der dritten dominierenden Maschine nur vergleichsweise wenige Zykluszeiten mit vergrößert werden müssen. Auch CPLEX kommt bei diesen Daten offenbar ebenfalls schneller zu besseren Schranken. Selbst H_1 liefert bessere Ergebnisse, obwohl die Jobs hier im Wesentlichen (bis auf m Jobs) zufällig angeordnet sind. Durch die Normalverteilung ist es selbst bei zufälligen Reihenfolgen wahrscheinlich, dass in einem Zyklus ähnliche Prozesszeiten sind.

Die dritte Testreihe wurde mit einem identischen Verfahren wie die zweite Testreihe generiert. Allerdings wurde der Erwartungswert der Prozesszeiten auf Maschine M_3 auf $\mu = 30$ festgesetzt, also um 20 weniger gegenüber M_4 und M_5. Ziel sollte es sein, dass M_4 und M_5 nicht dominierend sind, sondern nur eine geringe Semidominanz haben, so dass die Vorzüge der Gilmore-Gomory-Heuristik zum Tragen kommen.

Und tatsächlich bestätigen die Rechenergebnisse diese Vermutung, wie in Abbildung 4.5 zu sehen ist. Fast alle Heuristiken liefern bessere Ergebnisse als sie es bei den ersten beiden Testreihen tun (in der gleichen Zeit, s. Abbildung 4.6). Simulated Annealing ist hier allerdings nicht mehr die beste Heuristik. Die besten Resultate liefern hier die Gilmore-Gomory-Heuristik, die die größte Verbesserung gegenüber den anderen Testreihen erfährt, und die Non-Full-Schedule-Heuristik. Die Resultate von beiden sind nahezu identisch. Die Double Ended Non-Full-Schedule-Heuristik ist hier bei allen Instanzen schlechter als die Non-Full-Schedule-Heuristik, wenn auch nur um weniger als 1% bzw.

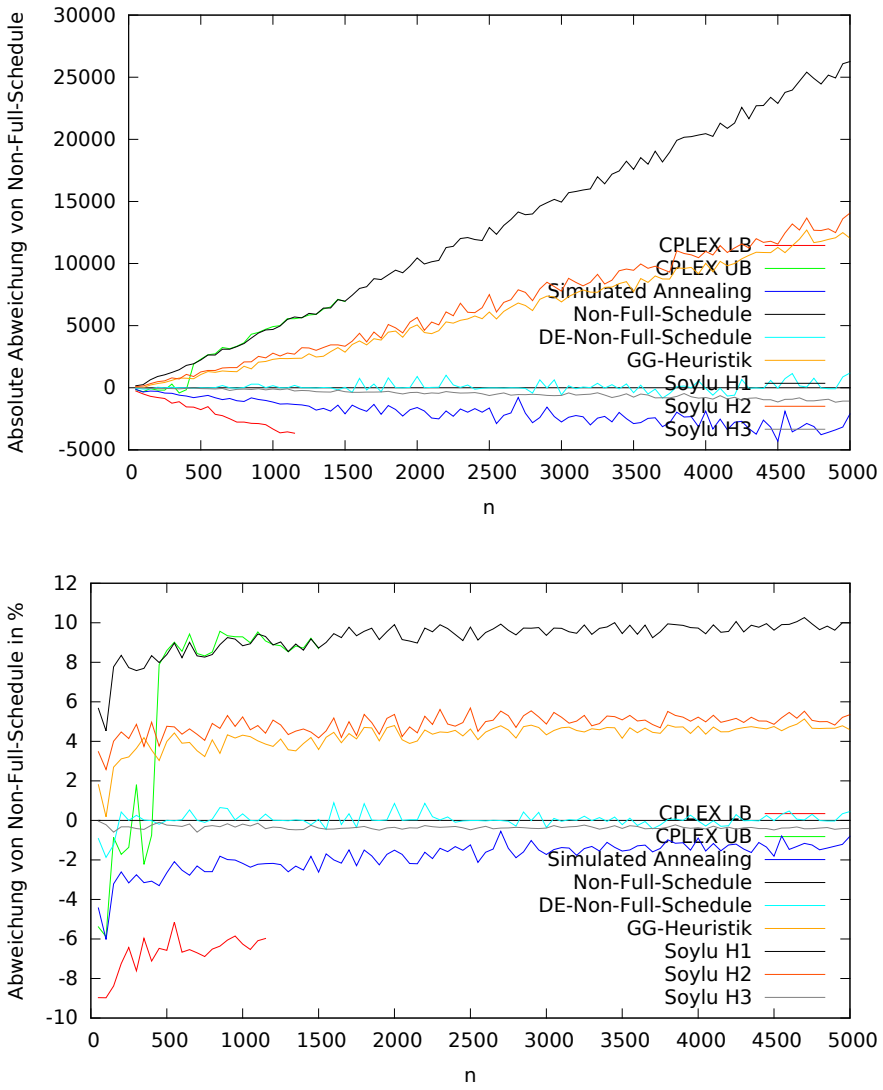

Abbildung 4.3: Resultate der Heuristiken für die Summe der Zykluszeiten (oben) und relative Unterschiede zwischen der Non-Full-Schedule-Heuristik und den übrigen Heuristiken (unten) bei drei dominierenden Maschinen (normalverteilte Prozesszeiten).

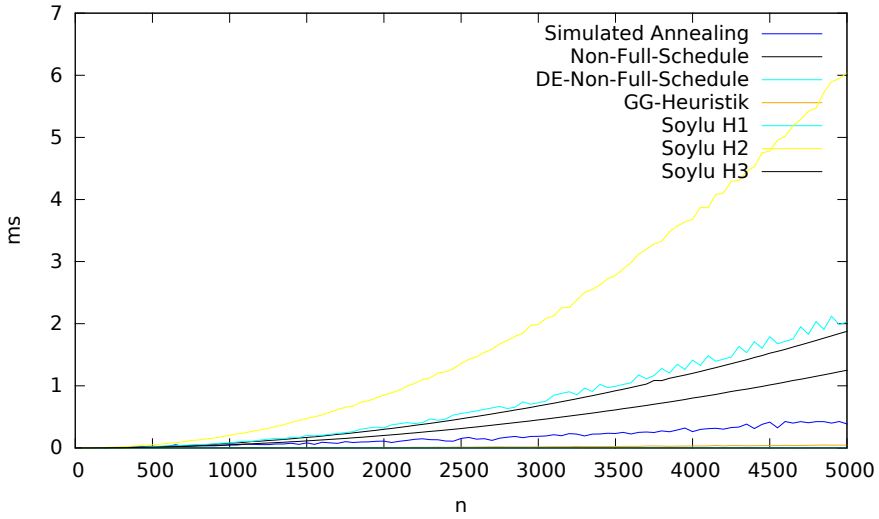

Abbildung 4.4: Laufzeiten der drei Heuristiken bei drei dominierenden Maschinen (normalverteilte Prozesszeiten).

bis zu 3% bei sehr kleinen Instanzen. Bemerkenswert ist außerdem, dass die untere Schranke, die von CPLEX für Instanzen mit $n \leq 1500$ berechnet wurde, oft bis an die Ergebnisse der Gilmore-Gomory- und der Non-Full-Schedule-Heuristik heranreicht. Das bedeutet, dass die Resultate dieser beiden Heuristiken nahezu optimal sind. Andererseits sind die oberen Schranken von CPLEX vergleichsweise weit von den Resultaten der Heuristiken entfernt.

Die vierte Testreihe wurde genau wie die erste generiert, allerdings mit nur zwei dominierenden Maschinen M_3 und M_4. Da in diesem Fall die Gilmore-Gomory-Heuristik ein optimales Ergebnis liefert, wurde das MIP nicht auf diese Instanzen angewandt. Ziel dieser Testreihe ist es, die Güte der Heuristiken anhand eines optimalen Vergleichswertes bestimmen zu können. In Abbildung 4.7 sind analog zu den anderen Testreihen die Ergebnisse aufgetragen. Wie zu sehen ist, liefert Simulated Annealing (abgesehen von Gilmore Gomory) die besten Ergebnisse mit nur etwa 2,5% Abweichung vom Optimum. Danach folgen fast gleich auf die beiden Non-Full-Schedule-Heuristiken und H_3. Bei vielen Instanzen liefern H_3 und die Non-Full-Schedule-Heuristik sogar identische Resultate, während die Double Ended Non-Full-Schedule-Heuristik minimal schlechter ist. Bis Instanzgrößen von $n = 500$ ist sie allerdings besser als die anderen beiden. Weit abgeschlagen sind H_1 und H_2. Da die einzelnen Heuristiken unterschiedlich viel Nutzen aus dem Umstand ziehen, dass es zwei benachbarte dominierende Maschinen gibt, sind die Resultate dieser Testreihe allerdings nur bedingt auf andere Instanzen mit mehr als zwei dominierenden Maschinen übertragbar.

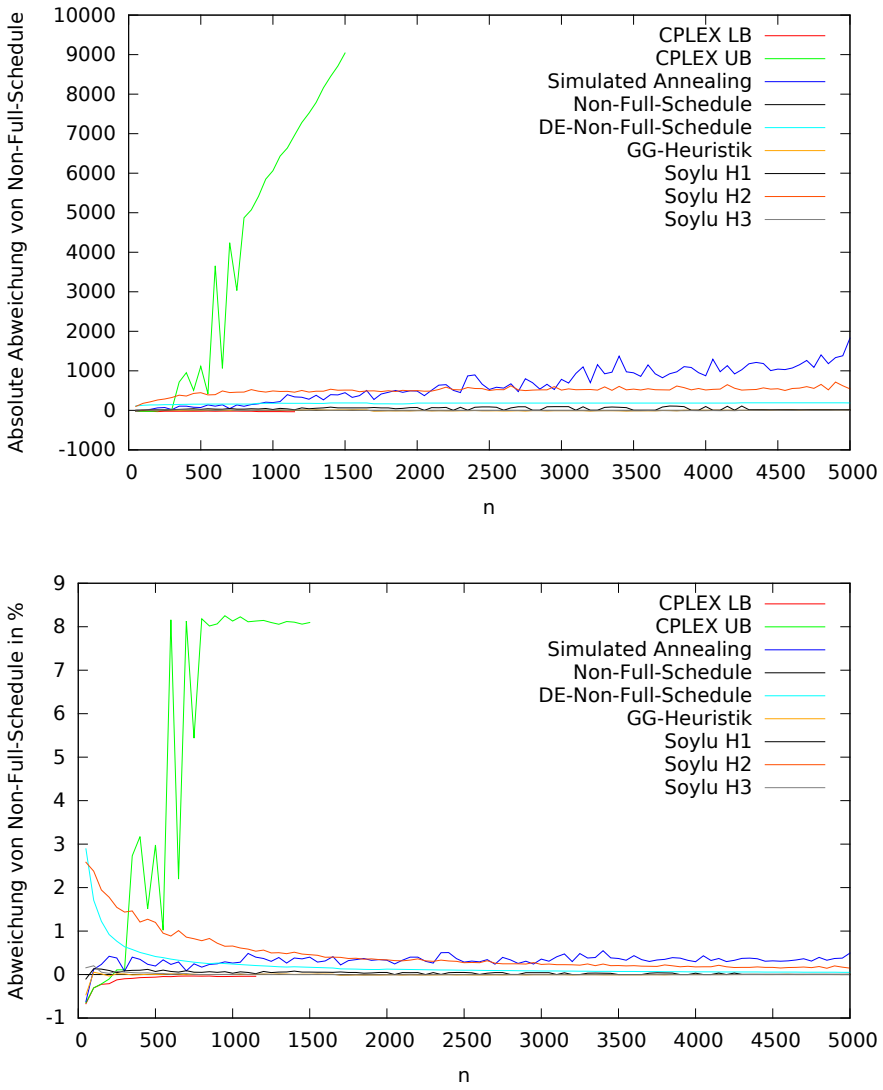

Abbildung 4.5: Resultate der Heuristiken für die Summe der Zykluszeiten (oben) und relative Unterschiede zwischen der Non-Full-Schedule-Heuristik und den übrigen Heuristiken (unten) bei zwei semidominierenden Maschinen (normalverteilte Prozesszeiten).

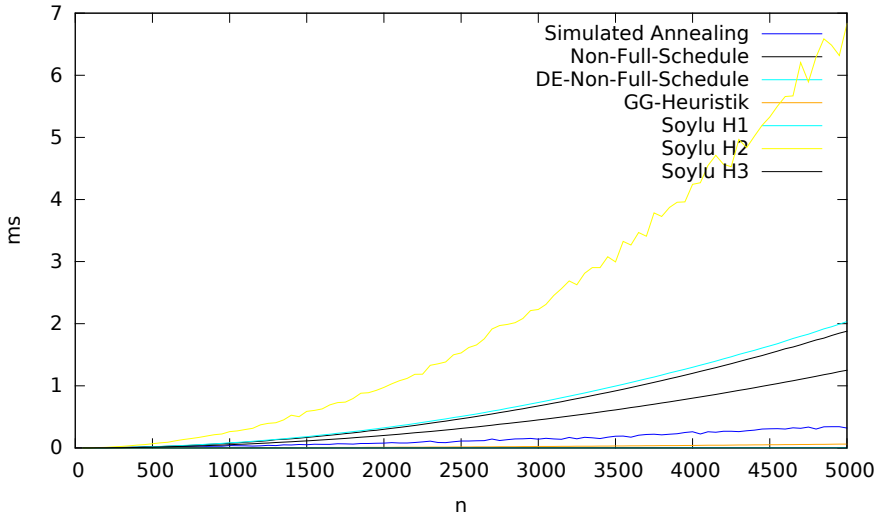

Abbildung 4.6: Laufzeiten der drei Heuristiken bei zwei semidominierenden Maschinen (normalverteilte Prozesszeiten).

4.1.2 Instanzen von Soylu et al.

In ihrem Paper [SKA07] generieren Soylu et al. ebenfalls einige Instanzen, mit denen sie ihre Heuristiken testen. Da diese mit $n \leq 50$ eine sehr viel kleinere Größe haben als die bislang hier getesteten Instanzen, ist es sinnvoll einige davon im Rahmen dieser Arbeit zu generieren und die Heuristiken auch mit anhand dieser zu evaluieren. Es wurden Instanzen mit $n = 20, 25, 30, 35$ Jobs generiert, die auf $\tilde{n} = 2, 3, \ldots, \frac{n}{5}$ Jobgruppen aufgeteilt sind. Dabei hat jeweils eine Jobgruppe die Größe $n_g = \frac{n}{2}$ und die übrigen Jobs werden gleichmäßig auf die anderen Jobgruppen aufgeteilt. Es gibt $m = 2, 3, 4$ Maschinen und die Prozesszeiten p_{ij} werden gleichverteilt aus dem Intervall $[1, 10]$ gewählt. In den Abbildungen 4.8, 4.9 und 4.10 sind die Resultate aller Heuristiken für diese Instanzen aufgetragen. An der Beschriftung der horizontalen Achse können die Eigenschaften der Instanz abgelesen werden. Dabei ist r Bezeichnung von Soylu et al. für die Anzahl der Jobgruppen ($\tilde{n}$).

Für $m = 2$ ist in Abbildung 4.8 zu sehen, dass Simulated Annealing immer die optimale Lösung gefunden hat, was daran zu erkennen ist, dass die Kurven für Simulated Annealing und Gilmore Gomory sich vollständig überlagern (Gilmore Gomory ist bei $m = 2$ optimal). Alle übrigen Heuristiken liefern stehts sehr ähnliche Ergebnisse, wobei häufig H_1 oder H_2 am schlechtesten ist. Interessant ist, dass jeweils bei Instanzen mit $r = \tilde{n} = 2$ alle Heuristiken eine optimale Lösung finden. Das kann dadurch erklärt werden, dass es bei zwei Maschinen und zwei Jobgruppen nicht viele sich wesentlich unterscheidende Möglichkeiten gibt, diese anzuordnen.

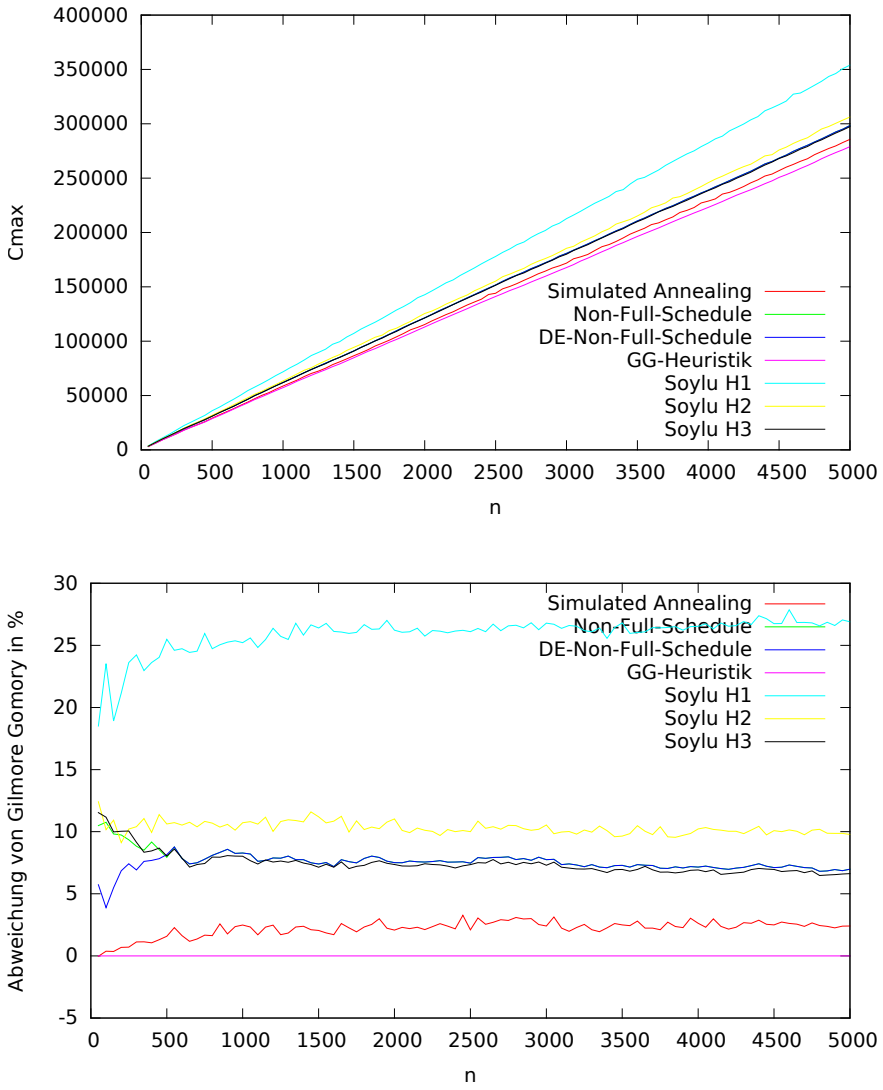

Abbildung 4.7: Resultate der Heuristiken für die Summe der Zykluszeiten (oben) und relative Unterschiede zwischen der Gilmore-Gomory-Heuristik und den übrigen Heuristiken (unten) bei zwei dominierenden Maschinen (gleichverteilte Prozesszeiten).

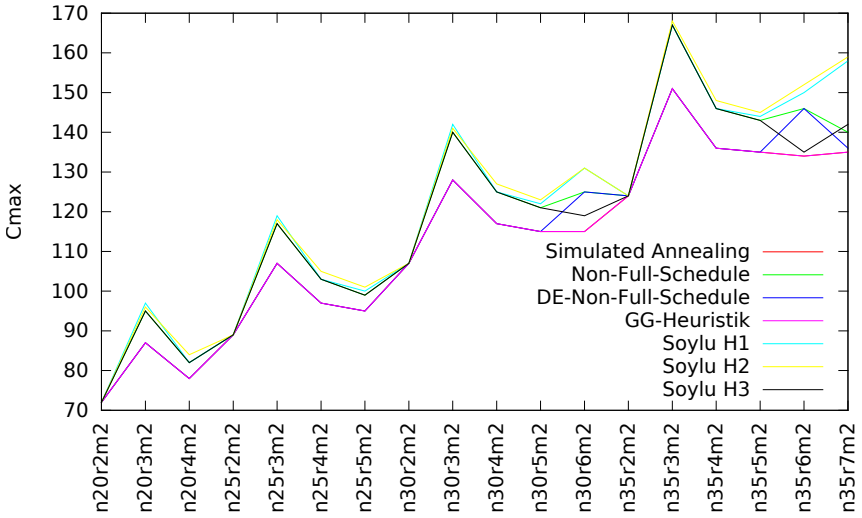

Abbildung 4.8: Resultate der Heuristiken auf den Instanzen von Soylu et al. mit $m = 2$.

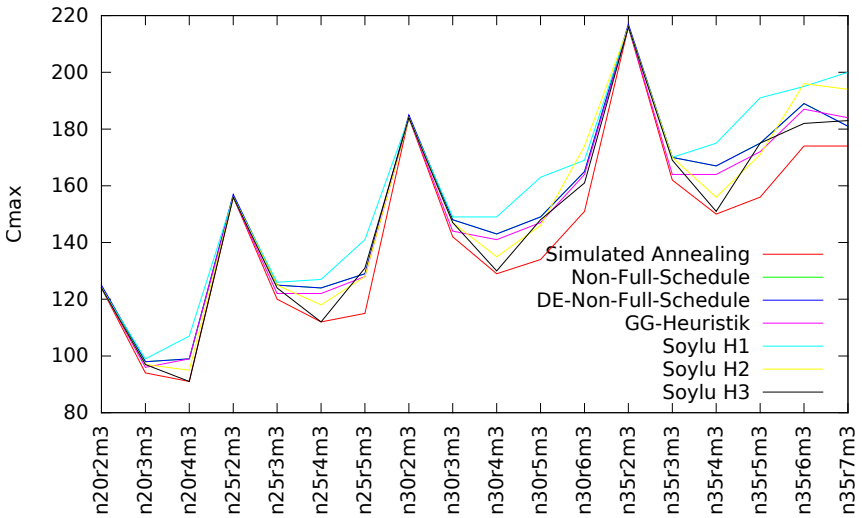

Abbildung 4.9: Resultate der Heuristiken auf den Instanzen von Soylu et al. mit $m = 3$.

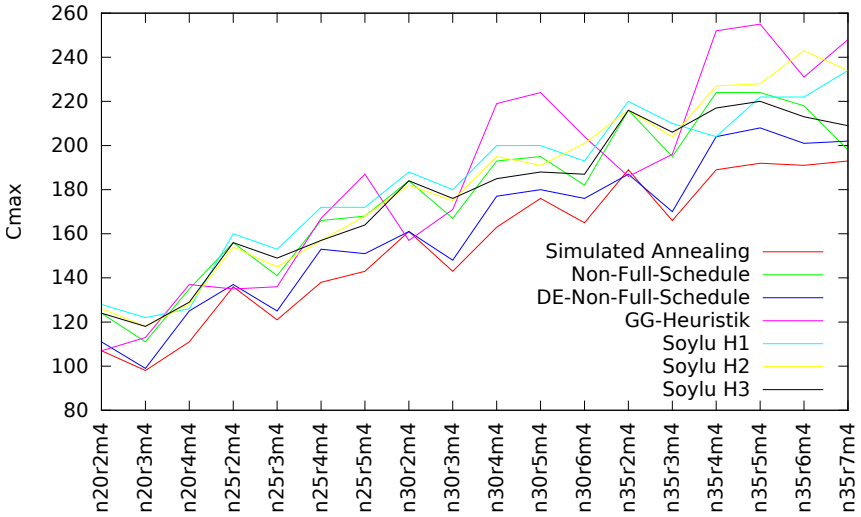

Abbildung 4.10: Resultate der Heuristiken auf den Instanzen von Soylu et al. mit $m = 4$.

Auch bei Instanzen mit $m = 3$ in Abbildung 4.9 liefern alle Heuristiken bei $r = \tilde{n} = 2$ die gleichen Resultate. Wie auch bei $m = 2$ liefert Simulated Annealing immer die besten Lösungen. Es kann allerdings nicht überprüft werden, ob diese optimal sind, da die Gilmore-Gomory-Heuristik hier keine optimalen Vergleichswerte liefern kann. Da aber diese Instanzen mit $n \leq 35$ genauso klein sind wie die in Abbildung 4.8, kann vermutet werden, dass die Ergebnisse des Simulated Annealings auch hier nahezu optimal sind. H_1 und H_2 schneiden auch hier häufig sehr schlecht ab. H_3 hingegen liefert bei Instanzen mit $r = \tilde{n} = 4$ die zweitbeste Lösung und ist dort besser als die beiden Non-Full-Schedule-Heuristiken (diese liefern fast immer das gleiche Ergebnis, weshalb sich die Graphen überlagern und kaum zu unterscheiden sind) obwohl diese nach einem ähnlichen Prinzip arbeiten.

Bei Instanzen mit $m = 4$ (s. Abbildung 4.10) liefern alle Heuristiken deutlich verschiedene Ergebnisse. Bis auf drei Ausnahmen bei $r = \tilde{n} = 2$ ist wieder Simulated Annealing am besten. Oft schneidet hier die Gilmore-Gomory-Heuristik besonders schlecht ab, da sie bei wenigen dominierenden Maschinen am besten ist. Besonders gute Resultate liefert hier neben Simulated Annealing die Double Ended Non-Full-Schedule-Heuristik und ist im Gegensatz zu den Instanzen mit $m = 2$ und $m = 3$ nicht fast identisch mit der Non-Full-Schedule-Heuristik und H_3.

4.1.3 Zusammenfassung

Unter allen Heuristiken ist Simulated Annealing oft am besten, während H_1 und H_2 von Soylu et al. oft sehr schlecht abschneiden. Das gilt sowohl für die selbst generierten Instanzen als auch für die von Soylu et al. Die Gilmore-Gomory-Heuristik liefert nur dann gute Ergebnisse, wenn zwei benachbarte Maschinen dominierend sind (dann sind die Ergebnisse optimal) oder semidominant. Sind diese Voraussetzungen nicht gegeben, sind auch keine guten Ergebnisse von der Gilmore-Gomory-Heuristik zu erwarten. Die Double Ended Non-Full-Schedule-Heuristik, die Non-Full-Schedule-Heuristik und H_3 von Soylu et al. gehen sehr ähnlich vor, was sich auch in ihren Resultaten widerspiegelt. Für jede der drei gibt es Instanzen, bei denen sie besser sind als die anderen beiden, so dass nur schwer vorausgesagt werden kann, welche wann am besten ist.

4.2 Vergleich der Dekompositionsansätze

In diesem Abschnitt werden die beiden Dekompositionsansätze aus den Kapiteln 2 und 3 miteinander verglichen. Ziel ist es, eine Aussage darüber zu treffen, in welchen Situationen welcher Ansatz die besseren Ergebnisse liefert. Da der zweite Dekompositionsansatz sich weitestgehend auf den Fall beschränkt, dass es zwei dominierende Maschinen gibt, Rüstkosten konstant sind ($s_{fg} = s$) und die Ressourcenmengen der Jobgruppen disjunkt sind, werden hier auch nur solche Instanzen betrachtet. Außerdem liegt diese Situation auch bei dem betrachteten Praxisfall des Küchenherstellers vor, der einige reale Instanzen zur Verfügung gestellt hat.

Im Fall von zwei benachbarten dominierenden Maschinen liefert der Algorithmus von Gilmore und Gomory eine optimale Lösung für den ersten Schritt des ersten Dekompositionsansatzes, das Erstellen einer Jobreihenfolge π ohne die Ressourcen und Rüstkosten zu berücksichtigen. Man könnte daher dem Irrtum auferliegen, dass die übrigen Heuristiken aus Abschnitt 2.1.3 oder auch die Heuristiken von Soylu et al. [SKA07] hier nicht betrachtet werden müssen. Allerdings folgt danach noch der zweite Schritt, in dem π auf Zulässigkeit bezüglich verfügbarer Ressourcen überprüft wird und ggf. nachträglich noch geändert wird.

In den folgenden Unterabschnitten werden zwei Testreihen betrachtet: In 4.2.1 werden Instanzen betrachtet, die von dem Küchenhersteller zur Verfügung gestellt wurden und daher direkten Praxisbezug haben. In Unterabschnitt 4.2.2 werden zufällig erzeugte Instanzen betrachtet.

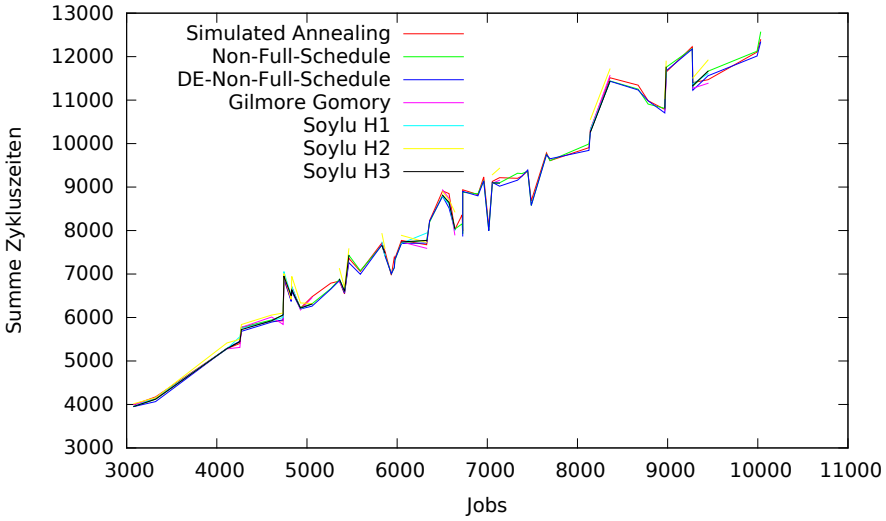

Abbildung 4.11: Summe der Zykluszeiten, nachdem π mit der jeweiligen Heuristik erzeugt und anschließend ggf. korrigiert wurde.

4.2.1 Reale Instanzen

Die hier betrachteten Instanzen stammen aus einer realen Anwendung. Die Instanzen haben eine Größe von 37-68 Jobgruppen, wobei jede Jobgruppe zwischen 1 und 1400 Jobs umfasst. Besonders große Jobgruppen sind dabei selten. Die meisten Jobgruppen bestehen aus weniger als 100 Jobs. Es gibt $m = 8$ Maschinen, von denen zwei benachbarte dominierend sind. Die Prozesszeiten auf den dominierenden Maschinen liegen zwischen 0,5 und 2 und für jede Jobgruppe gibt es zwischen 1 und 15 geeignete Ressourcen.

Resultate des ersten Dekompositionsansatzes

Zunächst soll der erste Dekompositionsansatz betrachtet werden. Dabei werden erst die unterschiedlichen Heuristiken angewandt und anschließend wird π wie in Abschnitt beschrieben korrigiert. In Abbildung 4.11 sind in Abhängigkeit von der Anzahl der Jobs n die Summen der Zykluszeiten aufgetragen, *nachdem* π korrigiert worden ist. Falls das Resultat einer Heuristik nicht so abgewandelt werden konnte, dass eine gültige Ressourcenzuweisung möglich ist, ist an der entsprechenden Stelle in der Kurve eine Lücke gelassen. Wie zu sehen ist, sind die Resultate der Heuristiken durch die Korrektur nahezu identisch, obwohl die Heuristiken teilweise von sehr unterschiedlicher Güte sind (vgl. Abschnitt 4.1). Allerdings treten einige Lücken in den Kurven auf, also nicht korrigierbare Ergebnisse (durch die Nähe der Kurven schwer zu erkennen).

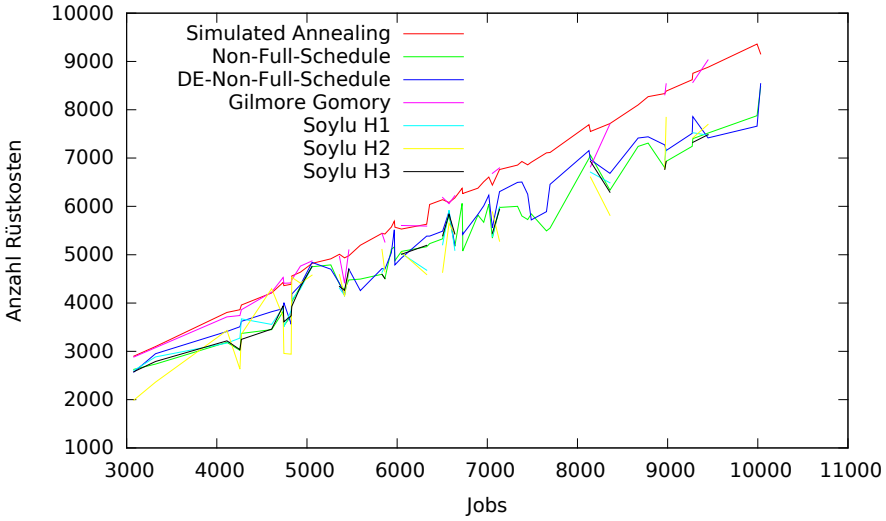

Abbildung 4.12: Anzahl der Rüstkosten, die auftreten, nachdem π mit der jeweiligen Heuristik erzeugt und anschließend ggf. korrigiert wurde.

In Abbildung 4.12 sind die zugehörigen Rüstkosten aufgetragen, die nach der Korrektur noch in π auftreten. Lücken bedeuten wieder, dass keine Korrektur möglich war. Es wird hier lediglich die *Anzahl* der Rüstkosten (d.h. Situationen, in denen eine Ressource gewechselt werden muss) betrachtet, nicht aber die tatsächliche Dauer in Zeiteinheiten. Diese kann aber leicht ermittelt werden aus der Größe der Rüstkosten ($s_{fg} = s$). Es ist zu sehen, dass gerade die Verfahren, die normalerweise sehr gute Ergebnisse liefern (z.B. Gilmore Gomory[1] oder Simulated Annealing) hier besonders schlecht abschneiden, während die zweite Heuristik von Soylu et al. hier (nach der Korrektur) vergleichsweise gute Resultate liefert. Besonders auffällig ist, dass offenbar (bei allen Heuristiken) in fast jedem Zyklus die Ressource gewechselt werden muss. Bei einer Instanzgröße von beispielsweise $n \approx 3000$ treten selbst bei dem dort besten Ergebnis (Soylu H2) ca. 2000 Ressoucenwechsel auf.

Es kann daher an dieser Stelle bereits vermutet werden, dass der erste Dekompositionsansatz (wenn überhaupt) nur bei sehr kleinen Rüstkosten s besser ist als der zweite.

Resultate des zweiten Dekompositionsansatzes

Beim zweiten Dekompositionsansatz muss zunächst die Zuordnung der Jobgruppen zu den Bins erfolgen. Unmittelbar danach steht fest, wie oft eine Ressource gewechselt

[1] Wegen der zwei benachbarten dominierenden Maschinen ist die Gilmore-Gomory-Heuristik hier ein exaktes Verfahren.

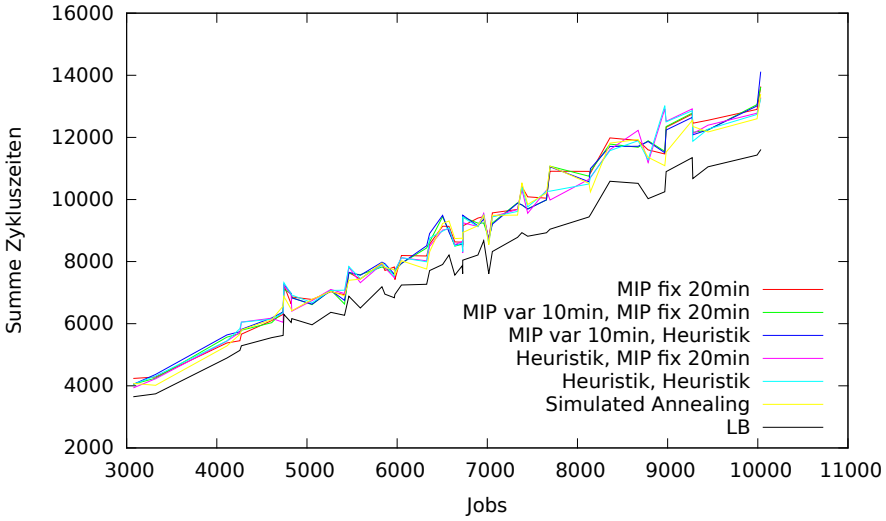

Abbildung 4.13: Summe der Zykluszeiten der einzelnen Algorithmenvarianten. Das MIP mit fixierten Bins (MIP fix) wurde in den jeweiligen Varianten 20 Minuten laufen gelassen und das mit freien Bins (MIP var) 10 Minuten.

werden muss. Für den Fall, dass keine Jobgruppe geteilt wird, sind es genau $\tilde{n} - m$ Ressourcenwechsel, wenn eine Jobgruppe geteilt wird, sind es $\tilde{n}-m+1$, usw. Da maximal $m - 1$ Jobgruppen geteilt werden, können höchstens $\tilde{n} - 1$ Rüstkosten auftreten, was sehr viel weniger ist als beim ersten Dekompositionsansatz (s. Unterabschnitt 4.2.1). Die Laufzeiten des Binpackings für diese Instanzen betrugen jeweils unter 100 Millisekunden.

In Abbildung 4.13 sind die Resultate (Summe der Zykluszeiten) der einzelnen Algorithmenvarianten (s. Abschnitt 3.3) aufgetragen bezogen auf die Anzahl der Jobs n. Außerdem ist mit LB eine untere Schranke angegeben, die wie in Unterabschnitt 3.2.3 beschrieben berechnet wurde. Die Varianten liefern offenbar sehr ähnliche Ergebnisse, wobei keine Regelmäßigkeit zu erkennen ist, in welchen Situationen welche Variante am besten ist. Es muss allerdings beachtet werden, dass einige Varianten dieses Ergebnis in sehr viel kürzerer Zeit erzielen als andere. Die Laufzeit einer Algorithmenvariante wird im Wesentlichen durch die involvierten MIPs bestimmt. Alle Verfahren, die nicht auf MIPs basieren terminierten jeweils in weniger als einer Sekunde.

In Abbildung 4.14 sind dieselben Daten aufgetragen, allerdings als Differenz zur unteren Schranke. Dort ist zu erkennen, dass die untere Schranke und die Resultate der Algorithmenvarianten sich je nach Instanzgröße um ca. 500 bis 1500 unterscheiden. Wie zu erwarten war, ist dieser Abstand nicht optimal, aber auch nicht außerordentlich groß. Es ist außerdem in Abbildung 4.14 zu erkennen (besser als in 4.13), dass in vielen Fällen das Simulated Annealing die besten Resultate liefert. Auch ist interessant, dass es einige wenige Fälle gibt, in denen das MIP mit fixierten Bins, also ohne vorherige Optimie-

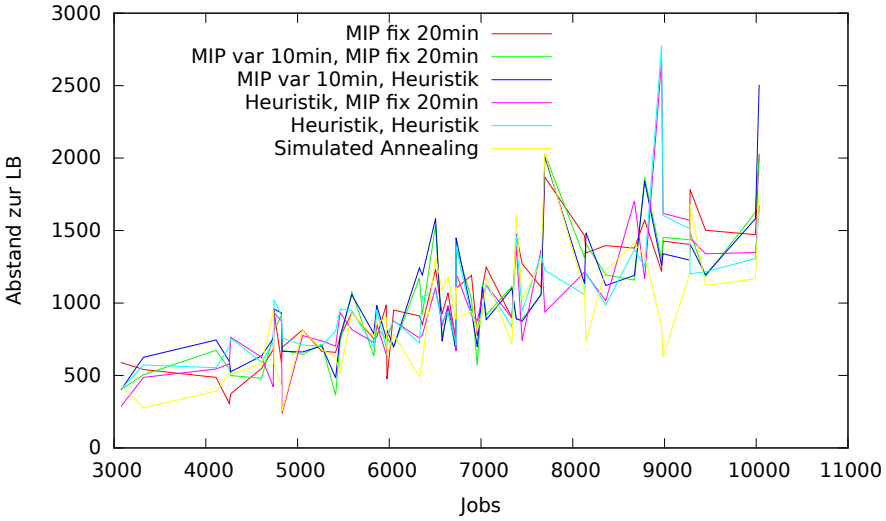

Abbildung 4.14: Differenzen zwischen den Algorithmenvarianten und der unteren Schranke (LB).

rung der Binreihenfolge, die beste Lösung liefert. Das kann dadurch erklärt werden, dass per Zufall die initiale Binreihenfolge sehr gut sein kann und sogar besser als durch die Heuristik oder das MIP mit freien Bins (das in diesem Kontext auch eine Heuristik ist) ermittelte Reihenfolgen.

Vergleich

Um die Resultate der beiden Dekompositionsansätze besser miteinander vergleichen zu können, wurde für die Kurven in den Abbildungen 4.11 und 4.13 jeweils eine Regressionsgerade berechnet, die in Abbildung 4.15 zu sehen sind. Außerdem ist dort die Differenz dieser beiden Geraden aufgetragen. Wie zu sehen ist, liegt diese Differenz ca. zwischen 0 und 1000. Das heißt je nach Instanzgröße ist der erste Dekompositionsansatz bezüglich der Summe der Zykluszeiten $\sum_{t=1}^{n-m+1} c_t$ um bis zu 1000 Zeiteinheiten besser. Andererseits wurde schon auf den immensen Unterschied der Rüstkosten hingewiesen: Beim ersten Dekompositionsansatz sind abhängig von der Instanzgröße bis zu ca. 8000 Ressourcenwechsel notwendig, während es beim zweiten Dekompositionsansatz höchstens $\tilde{n} - 1$ sind, was bei diesen Instanzen maximal 67 sind. Welcher Ansatz besser ist, hängt von der Höhe der Rüstkosten s ab. Bei $s \approx 0{,}12$ liefern beide Ansätze ähnliche Ergebnisse für $n \approx 10000$. Ist s größer, ist der zweite Dekompositionsansatz im Vorteil.

Wie groß ist s bei Kesseböhmer?

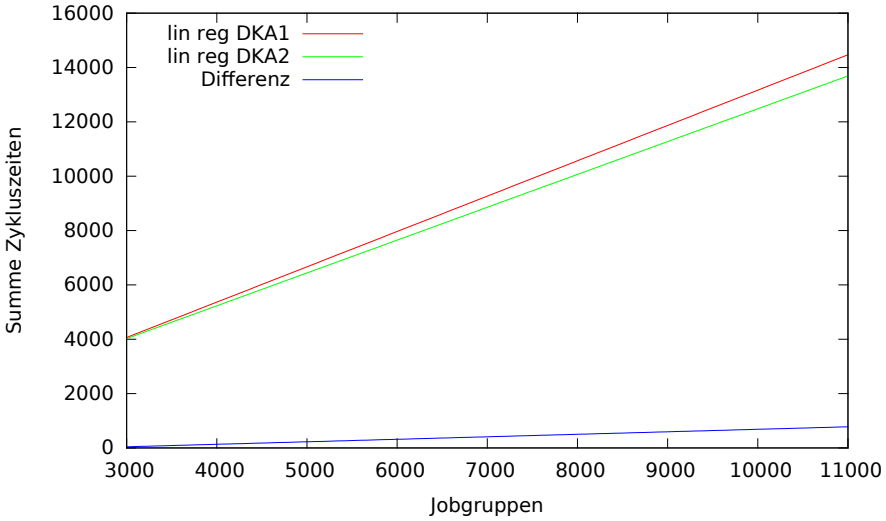

Abbildung 4.15: Regressionsgeraden für Kurven des ersten Dekompositionsansatzes (DKA1) aus Abbildung 4.11 und des zweiten Dekompositionsansatzes (DKA2) aus Abbildung 4.13 und die Differenz der beiden Geraden.

4.2.2 Generierte Instanzen

Zusätzlich zu den realen Datensätzen wurden zufällige Instanzen generiert, die im folgenden bezüglich der beiden Dekompositionsansätze evaluiert werden. Diese Testreihe besteht aus 30 Instanzen mit $\tilde{n} = 10, 11, \ldots, 40$ Jobgruppen. Die Größen der Jobgruppen wurden gleichverteilt aus dem Intervall $[1, 5\tilde{n}]$ gewählt. Die Prozesszeiten auf den beiden dominierenden Maschinen liegen gleichverteilt zwischen 0,5 und 2.

Leider noch keine Grafiken. Der Rest wird ähnlich aufgebaut wie der vorherige Unterabschnitt.

Literaturverzeichnis

[GG64] Gilmore, P. C. ; Gomory, R. E.: Sequencing a One State-Variable Machine: A Solvable Case of the Traveling Salesman Problem. In: *Operations Research* 12 (1964), Nr. 5, S. 655–679

[Koc04] Koch, Thorsten: *Rapid Mathematical Programming*, Technische Universität Berlin, Diss., 2004. `http://opus4.kobv.de/opus4-zib/frontdoor/index/index/docId/834`. – ZIB-Report 04-58

[KS03] Karabati, S. ; Sayin, S.: Assembly line balancing in a mixed-model sequencing environment with synchronous transfers. In: *European Journal of Operational Research* 149 (2003), Nr. 2, S. 417–429

[SKA07] Soylu, B. ; Kirca, Ö. ; Azizoğlu, M.: Flow shop-sequencing problem with synchronous transfers and makespan minimization. In: *International Journal of Production Research* 45 (2007), Nr. 15, S. 3311–3331

[WK] Waldherr, Stefan ; Knust, Sigrid: *Complexity results for flow shop problems with synchronous movement*

[WK14] Waldherr, Stefan ; Knust, Sigrid: Two-stage scheduling in shelf-board production: a case study. In: *International Journal of Production Research* 52 (2014), Nr. 13, S. 4078–4092