Ein Approximationsalgorithmus für die zeitliche Einteilung unabhängiger parallel laufender Maschinen

Bachelor-Thesis von Stefan Schwarzkopf aus München
November 2010

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Mathematik
AG Optimierung

Ein Approximationsalgorithmus für die zeitliche Einteilung unabhängiger parallel laufender Maschinen

Vorgelegte Bachelor-Thesis von Stefan Schwarzkopf aus München

1. Gutachten: Dr. habil. Marko Lübbecke
2. Gutachten: Prof. Dr. Stefan Ulbrich

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 7. November 2010

(Stefan Schwarzkopf)

Inhaltsverzeichnis

1 Einführung 3

2 Historisches 5
2.1 Problembeschreibung . . . 5
2.2 Das Einteilungsproblem ohne Vereinfachungen . . . 5
2.3 Identische Maschinen . . . 5
2.4 Verallgemeinerung durch Maschinengeschwindigkeiten . . . 5
2.5 Feste Anzahl von Maschinen . . . 5
2.6 Der Spezialfall $m = 2$ . . . 6

3 Ein Rundungstheorem 7

4 Der Approximationsalgorithmus 10
4.1 Der Approximationsalgorithmus für den allgemeinen Fall . . . 10
4.2 Ein PAS für das Laufzeiten Problem mit einer festen Anzahl von Maschinen . . . 12

5 Grenzen der Approximation 14

6 Feste Anzahl an Laufzeiten 16
6.1 Die Fälle $p_{ij} \in \{1, 2\}$ und $p_{ij} \in \{1, \infty\}$ . . . 16
6.2 Der allgemeine Fall zweier Laufzeiten . . . 17

7 Fazit 18

1 Einführung

In dieser Arbeit werden wir uns mit dem folgenden Problem beschäftigen: Man habe m Maschinen, die n voneinander unabhängige Jobs erledigen sollen. Es sei dabei $p_{ij} \in \mathbb{N}$ die Zeit, die Maschine i für Job j benötigt. Unabhänig bedeutet in diesem Zusammenhang, dass die Produktionszeiten p_{ij} nicht voneinander abhängen. Es existieren also keine „schnellen“ oder „langsame“ Maschinen, die im Vergleich zu den restlichen Maschinen alle Jobs schneller bzw. langsamer erledigen. Ziel ist es, in möglichst kurzer Zeit alle Jobs zu erledigen. Es handelt sich hierbei um ein ganzzahliges, lineares Optimierungsproblem, denn Aufgabe ist es, eine Einteilung für die Jobs mit minimaler Produktionsdauer zu finden. Nebenbedingungen des Problems werden sein, dass jeder Job nur einer Maschine zugeordnet werden kann, eine Deadline auf jeder Maschine eingehalten werden muss und dass eine Maschine einen Job nur ganz oder gar nicht erledigen kann. In Kapitel 3 werden wir dies auch mathematisch modellieren. Später werden wir auf Basis dieses Modells das Problem mit Hilfe von Algorithmen näherungsweise lösen.
Es wird sich u.a. um einen polynomiellen Algorithmus handeln, der uns eine Lösung liefert, deren Wert maximal um den Faktor 2 schlechter ist als das Optimum. Wir werden zudem einsehen, dass man bei einer festen Anzahl von Maschinen, also wenn m gegeben ist, das Problem vereinfacht wird und die Lösung bis auf eine maximale Abweichung um den Faktor $(1 + \varepsilon)$ in der Zielfunktion approximiert werden kann.
Später werden wir noch beweisen, dass kein Algorithmus existiert, dessen Lösung im Worst-Case weniger als um den Faktor 1,5 in der Zielfunktion von der Optimallösung abweicht, wenn nicht $P = NP$ gilt. Diese Aussage gilt aber nur, wenn die Anzahl der Maschinen m in die Problemstellung integriert ist und steht somit nicht im Widerspruch zur vorherigen Aussage.
Zum Ende dieser Arbeit werden wir das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit für eine feste Anzahl an verschiedenen Maschinenlaufzeiten untersuchen. Wir werden aber feststellen, dass diese Einschränkung das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit nur in wenigen Fällen vereinfacht.
Wir wenden uns nun mit einem der zentralen Begriffe dieser Arbeit zu, dem des Approximationsalgorithmus. Wir betrachten die Definition nur für Minimierungsprobleme, da es sich bei dem Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit um ein Minimierungsproblem handelt:

Definition 1.1. *Sei p ein ganzzahliges Optimierungsproblem und c_p der Zielfunktionswert des Optimums x_p.*
Ein Algorithmus A heißt ***Approximationsalgorithmus*** *für ein Problem p, wenn sein ausgegebener Wert c_a der Lösung maximal um einen Faktor ε größer als das Optimum c_p ist, also*

$$c_a \leq \varepsilon c_p$$

gilt. Ein Approximationsalgorithmus, dessen Lösung um den Faktor ε größer als das Optimum c_p ist, heißt auch ε- ***Approximationsalgorithmus****. Man nennt ε die* ***Gütegarantie*** *eines ε- Approximationsalgorithmus.*

[1]

Dabei ist zu beachten, dass diese Algorithmen keine polynomielle Laufzeit und nicht als Lösung das Optimum x_p haben müssen, sie garantieren nur eine maximale Abweichung vom Optimum c_p in der Zielfunktion. Sie werden häufig bei komplizierten Problemen angewandt, bei denen es keine exakten Lösungsalgorithmen gibt oder jene eine extrem lange Laufzeit haben. Das Verhalten solcher Algorithmen wird schon lange untersucht, es ist aber recht wenig über ihre Struktur bekannt, aufgrund derer wir kleinere Abweichungen vom Optimum c_p garantieren könnten.
Besonders oft wurde ihr Verhalten bezüglich der Problemklasse des Erstellen eines Plans wie im Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit und der Behälterprobleme analysiert. Ein Behälterproblem sieht wie folgt aus:

Definition 1.2. *Es sei eine feste Anzahl $k \in \mathbb{N}$ von Behältern der Größe b vorgegeben. Dazu gebe es $n \in \mathbb{N}$ Gewichte mit Masse a_i mit $a_i \leq b$ für $i = 1, ..., n$. Das* ***Behälterproblem*** *ist nun diese Gewichte so auf die Behälter zu verteilen, dass keiner „$\ddot{u}berl\ddot{a}uft$“, also dass eine Abbildung*

$$f : \{1, ..., n\} \to \{1, ..., k\}$$

existiert, so dass:

$$\sum_{f(i)=j} a_i \leq b \qquad \forall j \in \{1, ..., k\}$$

Auf diese Art von Problemen wollen wir an dieser Stelle nicht näher eingehen.
Die Frage ist nun, wie wir für das ganzzahlige Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit eine approximative Lösung erhalten bzw. wie wir es zu einem Problem vereinfachen können, dessen Lösung wir finden können. Eine intuitiver Ansatz ist, die Ganzzahligkeitsbedingungen zunächst wegzulassen und die Lösung des linearen Problems auf ein ganzzahliges Ergebnis zu runden. Darauf wird auch später unser Algorithmus basieren. Hierbei kann es aber folgende Schwierigkeiten geben: Die so gefundene Lösung kann stark vom Optimum abweichen oder es kann sogar unmöglich sein, eine zulässige Lösung des ganzzahligen Problems zu finden. Es existieren aber viele Problemfelder z.B. in der Graphentheorie, in der diese Methode zu zulässigen Lösungen führt. Auch zum Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit beweisen wir ein Rundungstheorem, das besagt, dass sich auf diese Art und Weise eine Lösung finden lässt.

2 Historisches

2.1 Problembeschreibung

Doch bevor wir uns näher mit diesem Theorem beschäftigen, wollen wir aufzeigen, was bisher zum Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit herausgefunden wurde. Dazu stellen wir das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit noch einmal kurz dar:
Wir haben n unabhängige Jobs, die von m Maschinen in möglichst wenig Zeit erledigt werden sollen. Die Produktionszeit einer Maschine ist die Summe über alle ihr zugewiesenen p_{ij} und die Gesamtproduktionszeit ist das Maximum der einzelnen Maschinenproduktionszeiten. Wir suchen einen Plan, der uns diese minimiert. Das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit wurde der Problemklasse der Einteilungsprobleme $R||C_{max}$ zugeteilt[2]. Diese Notation bedeutet, dass keinerlei Anforderungen an die Jobs gestellt werden und die Produktionszeiten unabhängig voneinander sind.

2.2 Das Einteilungsproblem ohne Vereinfachungen

Nun zu dem, was bisher zu dem Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit herausgefunden wurde:
Es ist ein Algorithmus bekannt, dessen Ergebnis nicht größer als $2\sqrt{m}$ mal das Optimum c_p ist[3]. Wie schon erwähnt, finden wir einen Approximationsalgorithmus mit maximaler Abweichung um den Faktor 2. Dies ist natürlich keine besonders gute Approximation, deshalb hat man sich in der Vergangenheit verschiedene Vereinfachungen für das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit überlegt:

2.3 Identische Maschinen

So wurde zum Beispiel gefordert, dass die Produktionszeit für alle Jobs auf jeder Maschine gleich ist, also

$$p_{1j} = p_{2j} = ... = p_{mj} \qquad \text{für alle } j \in \{1, ..., n\}\,.$$

Hier wurde ein Approximationsalgorithmus gefunden, dessen Lösung maximal um den Faktor $2 - 1/m$ vom Optimum c_p abwich[4].
Für die nächsten Lösungsmethoden benötigen wir folgende Definition:

Definition 2.1. *Sei p ein ganzzahliges Optimierungsproblem und c_p der Zielfunktionswert des Optimums x_p.*
a)Ein Algorithmus A zur Lösung von p heißt ***polynomielles Approximationsschema(PAS)****, falls A für jedes feste $\varepsilon > 0$ in polynomieller Laufzeit eine Gütegarantie von $1 + \varepsilon$ hat.*
b)Ein PAS A heißt ***vollpolynomielles Approximationsschema(FPAS)****, falls die Laufzeit von A polynomiell von der Inputlänge und $1/\varepsilon$ abhängt.*

[1]

Wenn wir nun in dem Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit eine feste Anzahl gleicher Maschinen betrachten, dann finden wir ein solches FPAS[5]. Wenn allerdings die Anzahl der Maschinen zur Problemstellung gehört, ist es, auch wenn die Maschinen gleich sind, unmöglich ein FPAS zu finden, wenn nicht P=NP[6][7]. Für diesen Fall wurde allerdings ein PAS gefunden[8].

2.4 Verallgemeinerung durch Maschinengeschwindigkeiten

Das Problem mit gleichen Maschinen lässt sich verallgemeinern, wenn wir jeder Maschine eine Produktionsgeschwindigkeit s_i zuweisen und p_{ij} sich durch p_j/s_i ergibt. Hierzu existiert ein 2-Approximationsalgorithmus[9]. Wenn die Anzahl der Maschinen in diesem Fall festgehalten wird, findet man ein FPAS [10] und wenn sie in das Problem integriert ist, ein PAS[11].

2.5 Feste Anzahl von Maschinen

Für eine feste Zahl unabhängiger Maschinen findet man ebenfalls ein FPAS[10]. Hier werden wir ein PAS vorstellen, dessen Laufzeit abhängig von einem Polynom der Inputlänge, m und $ln(1/\varepsilon)$ ist. Wir werden auch darauf eingehen, was die Verbesserung an diesem PAS zu dem bereits gefundenen FPAS ist.

Allerdings existiert kein PAS für $\varepsilon < 1/2$, wenn die Anzahl der Maschinen zum Problem gehört. Wir zeigen später, dass hieraus $P = NP$ folgen würde.

Ebenfalls erwähnenswert ist ein von Potts präsentierter 2-Approximationsalgorithmus, dessen Laufzeit polynomiell von der Inputlänge und m^{m-1} abhängt. Für ein festes m erhalten wir also ein FPAS[12]. Die Frage ist natürlich, was an diesem Algorithmus besser ist als an dem zuvor gefundenen, der ebenfalls ein FPAS für eine feste Anzahl von Maschinen liefert. Die Verbesserung ist, dass der alte Algorithmus eine Laufzeit in $O(nm(nm/\varepsilon)^{m-1})$ hat, hingegen hängt die Laufzeit des Algorithmus von Potts nur von m^{m-1} und einem Polynom der Inputlänge ab. Auch der Speicherplatz von Potts Algorithmus hängt von einem Polynom der Inputlänge ab, während der des alten Algorithmus in $O((nm/\varepsilon)^{m-1})$ liegt. Der Algorithmus bringt also auch erhebliche Verbesserungen für die Laufzeit und den Speicherplatz, für den Fall, dass m zur Probleminstanz gehört.
Dieser Algorithmus basiert auf dem Ansatz der Linearisierung des ganzzahligen Problems. Wir werden diesen Ansatz im nächsten Kapitel durch ein Rundungstheorem vertiefen, und auf dessen Basis einen Algorithmus präsentieren, dessen Laufzeit nicht mehr exponentiell von m abhängt.

2.6 Der Spezialfall $m = 2$

Nun kurz zu dem Spezialfall, wenn nur 2 Maschinen vorhanden sind. Hier wurde ein 3/2-Approximationsalgorithmus gefunden[12], und dieser wurde zu einem PAS eweitert[13]. Wir werden im nächsten Kapitel das Rundungstheorem vorstellen, mit dem man aus der Lösung des linearen Problems durch Runden in polynomieller Zeit eine ganzzahlige Lösung erhalten kann.

Wir kommen nun zu dem ersten zentralen Theorem dieser Arbeit. Auf ihm wird später der Algorithmus basieren. Es besagt, dass wir Ecken eines Polyeders, das durch die Bedingungen eines linearen Problems bestimmt wird, zu einer Lösung eines sehr ähnlichen ganzzahligen Problems runden können.
Dazu führen wir ein paar Variablen ein: Seien $J_i(t)$ die Menge von Jobs, für die Maschine i nicht länger als t Zeiteinheiten benötigt. Analog sei $M_j(t)$ die Menge von Maschinen, die Job j in weniger als t Zeiteinheiten bearbeitet. Zudem habe jede Maschine i ihre Deadline d_i, zu der jeder Job vollendet sein muss. Wir betrachten das folgende Problem:

Theorem 3.1. *(Das Rundungstheorem)*
Sei $P = (p_{ij}) \in \mathbb{N}^{m\times n}, \vec{d} = (d_1, ..., d_m) \in \mathbb{N}^m$ und $t \in \mathbb{N}$. Wenn das lineare Problem LP(P,$\vec{d}$,t) gegeben durch

$$\sum_{i\in M_j(t)} x_{ij} = 1 \quad \text{für } j = 1, ..., n$$

$$\sum_{j\in J_i(t)} p_{ij}x_{ij} \leq d_i \quad \text{für } i = 1, ..., m$$

$$x_{ij} \geq 0 \quad \text{für } j \in J_i(t), i = 1, ..., m$$

eine zulässige Lösung besitzt, dann kann jede Ecke $\tilde{x}$ dieses Polyeders zu einer zulässigen Lösung $\overline{x}$ des ganzzahligen Problems GP(P,$\vec{d}$,t), gegeben durch

$$\sum_{i\in M_j(t)} x_{ij} = 1 \quad \text{für } j = 1, ..., n$$

$$\sum_{j\in J_i(t)} p_{ij}x_{ij} \leq d_i + t \quad \text{für } i = 1, .., n$$

$$x_{ij} \in \{0, 1\} \quad \text{für } j \in J_i(t) \text{ und alle } i \in \{1, ..., m\},$$

gerundet werden und dieses Runden geschieht in polynomieller Zeit.

Bevor wir dieses Theorem beweisen werden, möchten wir kurz den Zusammenhang zwischen dem Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit und den Bedingungen des ganzzahligen Problems klären.
Die erste Bedingung bedeutet, dass jeder Job von genau einer Maschine erledigt wird.
Die zweite Nebenbedingung gibt eine obere Schranke für die Maschinenlaufzeiten vor. Wir werden später Entscheidungsprobleme betrachten, ob eine Einteilung mit Laufzeit kleiner gleich $k \in \mathbb{N}$ existiert und dazu ist diese Bedingung sehr hilfreich.
Die dritte Bedingung bedeutet einfach nur, dass ein Job von einer Maschine erledigt werden kann oder nicht und eine Maschine keine halben oder dreiviertel Jobs vollbringt.
Dieses ganzzahlige Problem ist die Modellierung des Problems der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit sowie gegebener Deadline. Doch nun kommen wir zum Beweis des Rundungstheorems:

Beweis. **Schritt 1: Eckenbeschreibung im LP**

Sei ν die Anzahl der Variablen im LP(P,$\vec{d}$,t). Dann ist das Polyeder durch m (Bedingung 1)+n (Bedingung 2)+ν (Variablen größer gleich 0) Bedingungen beschrieben und liegt im Einheitswürfel(Bedingung 1). Ecken $\tilde{x}$ dieses Polyeders sind u.a. die Einheitsvektoren, für die $p_{ij} \leq d_i$ gilt.
Jede Ecke $\tilde{x}$ wird von ν unabhängigen Zeilen der Nebenbedingungsmatrix beschrieben, die mit Gleichheit erfüllt sein müssen. Es haben also für jede Ecke $\tilde{x}$ höchstens $m + n$ Variablen einen Wert ungleich 0.
Um dies einzusehen, initialisieren wir den Simplex-Algorithmus. Wir müssen also zunächst die m Ungleichungen der zweiten Nebenbedingung auf Gleichheit bringen und führen dazu m Schlupfvariablen y_i ein. Wir haben jetzt als Nebenbedingungsmatrix eine $(\nu+m)\times(n+m)$ Matrix. Eine Ecke wird von einer quadratischen Matrix beschrieben, wir setzen also $\nu+m-(n+m)$ Variablen auf 0, also sind höchstens $n+m$ Variablen ungleich 0. Angenommen, alle auf 0 gesetzten Variablen wären keine Schlupfvariablen, dann haben wir durch die Vorzeichenbeschränkung von x_{ij} und Nebenbedingung 1 $\nu+m-(n+m)+n = \nu$ Gleichungen, die unsere Ecke beschreiben. Wenn wir nun noch Schlupfvariablen y_i auf 0 setzen müssen, haben wir eine nicht 0-Variable x_{ij} mehr, also eine Gleichung in der Vorzeichenbeschränkung weniger, allerdings

wäre eine Nebenbedingung 2 mit Gleichheit erfüllt, da der Schlupf y_i 0 ist. Somit haben wir immer ν Gleichungen, die unsere Ecke beschreiben.

Schritt 2: Beschreibung des Problems durch einen Graphen

Wir schreiben in die Matrix P zeilenweise die Zeit, die Maschine i für die Jobs j benötigt, also

$$P = (p_{ij}) \in \mathbb{N}^{m \times n}.$$

Für jede zulässige Lösung x des LP(P,$\vec{d}$,t) stellen wir das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit als bipartiten Graph

$$G(x) = (M, J, E)$$

dar.

$$M := \{1, ..., m\}$$

sind die Maschinenknoten,

$$J := \{1, ..n\}$$

die Jobknoten, und die Kanten seien definiert durch

$$E := \left\{(i,j)|x_{ij} > 0\right\}.$$

$G(\tilde{x})$, der Graph zu einer Ecke $\tilde{x}$, hat nicht mehr Kanten als Knoten, da höchstens $m + n$ Variablen nicht 0 waren. Um fortzufahren, benötigen wir folgende Definition:

Definition 3.1. *Sei G=(V,E) ein Graph. G heißt* ***Pseudowald****, wenn jede Zusammenhangskomponente von G höchstens einen Kreis enthält.*

[14]

Wir zeigen nun, dass $G(\tilde{x})$ ein Pseudowald ist. Sei dazu C eine Zusammenhangskomponente von $G(\tilde{x})$. Seien M_C und J_C die Maschinen und Jobs in C und

$$\tilde{x}_C := \left\{\tilde{x}_{ij}|i \in M_C, j \in J_C\right\}.$$

Die restlichen Komponenten von $\tilde{x}$ nennen wir $\tilde{x}_{\bar{C}}$. Der Einfachheit halber sei $\tilde{x} = (\tilde{x}_C, \tilde{x}_{\bar{C}})$. Sei P_C die Matrix, in der die Produktionszeiten der Maschinen in M_C für die Jobs aus J_C stehen. $\vec{d}_C$ seien die Deadlines für die Maschinen aus M_C.
Wir zeigen zunächst, dass $\tilde{x}_C$ eine Ecke des Polyeders, das durch das $LP(P_C, \vec{d}_C, t)$ beschrieben wird, ist. Angenommen, dies wäre nicht der Fall. Dann existieren $y_1, y_2 \in LP(P_C, \vec{d}_C, t)$, so dass

$$\tilde{x}_C = 1/2(y_1 + y_2).$$

Dann gelte aber

$$\tilde{x} = 1/2((y_1, \tilde{x}_{\bar{C}}) + (y_2, \tilde{x}_{\bar{C}})),$$

was allerdings ein Widerspruch dazu ist, dass $\tilde{x}$ eine Ecke ist.
Da nun $\tilde{x}_C$ eine Ecke ist, hat $G(\tilde{x}_C)$ nicht mehr Kanten als Knoten und da C zusammenhängend ist, haben wir einen Baum oder einen Baum mit zusätzlicher Kante. Denn als einzige Möglichkeit haben wir entweder eine Kante weniger als Knoten oder genauso viele Knoten wie Kanten, da C zusammenhängend ist. Also enthält C höchstens einen Kreis und $G(\tilde{x})$ ist somit ein Pseudowald.

Schritt 3: Das Runden durch Paaren im Graphen

Wir benutzen dies jetzt, um die Ecke $\tilde{x}$ zu runden. Wir untersuchen nun jede Kante, für die gilt:

$$\tilde{x}_{ij} = 1.$$

Für diese Kanten setzen wir einfach $\overline{x}_{ij} = 1$. Diese Kanten gehören aufgrund der Bedingung 1 zu den Jobknoten vom Grad 1. Wenn wir diese Kanten und Knoten löschen, erhalten wir also einen Pseudowald $G'(\tilde{x})$, dessen Jobknoten mindestens Grad 2 haben.

Der Beweis motiviert zu folgender Definition:

Definition 3.2. *Eine Menge M unabhängiger Kanten in einem Graphen G=(V,E) nennt man* ***Paarung****. Unabhängig bedeutet in diesem Zusammenhang, dass die Kanten keine gemeinsamen inzidenten Knoten haben.*

[15]

Wir beweisen, dass die Jobknoten in $G'(\tilde{x})$ von einer Paarung überdeckt werden. (D.h. jeder Jobknoten ist zu einer Kante aus der Paarung inzident). Für jeden Baum in $G'(\tilde{x})$, betrachte jeden Knoten im Baum als Wurzel, und paare jeden Knoten mit einem seiner Kinder. Jeder Jobknoten hat mindestens ein Kind und jeder Knoten hat höchstens einen Vaterknoten, da $G'(\tilde{x})$ ein Baum ist, also ist keine Maschine mit mehr als einem Job verbunden.
Für jede Komponente mit einem Kreis, nehmen wir abwechselnd die Kanten aus dem Kreis in die Paarung. Dies ist möglich, weil der Kreis gerade Länge hat, da $G'(\tilde{x})$ bipartit ist. Wenn wir nun die Kanten des Kreises löschen, erhalten wir verschiedene Bäume mit den Knoten des Kreises als Wurzeln. Jeden Jobknoten, der noch nicht in der Paarung ist, paaren wir mit einem seiner Kinder. Also haben wir eine Paarung gefunden.
Wenn jetzt (i,j) in der Paarung ist, setzen wir

$$\overline{x}_{ij} = 1.$$

Die restlichen $\overline{x}_{ij}$ setzen wir auf 0.

Schritt 4: Überprüfen, ob $\overline{x}$ wirklich Lösung des GP ist

Dies ist aus folgenden Gründen eine Lösung des $GP(P, \vec{d}, t)$: Jeder Job ist genau einer Maschine zugeordnet, also gilt:

$$\sum_{i \in M_j(t)} \overline{x}_{ij} = 1 \qquad f\ddot{u}r \;\; j = 1, ..., n$$

Für jede Maschine i gibt es höchstens ein Job, so dass $\tilde{x}_{ij} < \overline{x}_{ij} = 1$ und da $p_{ij} \leq t$, erhalten wir:

$$\sum_{j \in J_i(t)} p_{ij}\overline{x}_{ij} \leq \sum_{j \in J_i(t)} p_{ij}\tilde{x}_{ij} + t \leq d_i + t \;\; \text{für } i = 1, .., m$$

Die erste Ungleichung folgt aus:

$$t - \sum_{j \in J_i(t)} p_{ij}(\overline{x}_{ij} - \tilde{x}_{ij}) \geq t - t \sum_{j \in J_i(t)} \overline{x}_{ij} - \tilde{x}_{ij} = t(1 - \sum_{j \in J_i(t)} \overline{x}_{ij} - \tilde{x}_{ij}) \geq 0,$$

dies gilt, da $t \geq 0$ ist und

$$\underbrace{\sum_{j \in J_i(t)} \overline{x}_{ij}}_{\leq 1} - \underbrace{\sum_{j \in J_i(t)} \tilde{x}_{ij}}_{\geq 0} \leq 1.$$

Somit ist $\overline{x}$ eine zulässige Lösung des GP(P,$\vec{d}$,t). Das Suchen von Ecken sowie das Paaren des Graphen geschieht in polynomieller Zeit, also ist die Behauptung bewiesen. □

Aus diesem Theorem erhalten wir verschiedene Ergebnisse, auf die wir im nächsten Kapitel eingehen werden.

4 Der Approximationsalgorithmus

4.1 Der Approximationsalgorithmus für den allgemeinen Fall

Wir finden mit dem Rundungstheorem einen 2- Approximationsalgorithmus. Wir werden dazu in diesem Kapitel den Begriff des p - relaxierten Entscheidungsproblems einführen. Später werden wir herausfinden, dass ein p- Approximationsalgorithmus für das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit existiert, wenn sich das Problem in ein solches p- relaxiertes Entscheidungsproblem umschreiben lässt. Dies vereinfacht uns das Finden eines 2- Approximationsalgorithmus, denn wir müssen nur die Frage, ob eine Lösung für das ganzzahlige Problem aus dem vorherigen Kapitel existiert, in ein 2- relaxiertes Entscheidungsproblem umschreiben.
Beginnen wir also mit folgender Definition:

Definition 4.1. *Seien die Matrix $P \in \mathbb{N}^{m\times n}$ und der Vektor $d \in \mathbb{N}^m$ gegeben.*
a) Die Ausgabe eines ***Entscheidungsproblems*** *ist „ja“ oder „nein“, z. B. zu der Frage,e ob ein Vektor $x \in \mathbb{R}^n$ existiert, so dass*

$$Px \leq d$$

b)Ein ***p- relaxiertes Entscheidungsproblem*** *hat als Ausgabe „fast“ oder „nein“, genauer ist die Ausgabe entweder „nein“, dann existiert keine Einteilung x, so dass $Px \leq d$ oder man erhält eine Einteilung x, so dass*

$$Px \leq pd \quad \textit{für ein } p > 1$$

Wir kommen nun zu folgender Aussage:

Lemma 4.1. *Wenn ein p- elaxiertes Entscheidungsproblem zum Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit existiert, dann existiert ein polynomieller p- Approximationsalgorithmus für dieses Problem.*

Beweis. **Schritt 1: Startschranken für eine Binärsuche nach der Optimallösung**

Für ein gegebenes P konstruieren wir folgenden Plan: Wir ordnen jeden Job der Maschine zu, auf der er am schnellsten erledigt wird. Wenn die Laufzeit für diesen Plan o beträgt, dann ist o eine obere Schranke für das Optimum, da es sich um ein Minimierungsproblem handelt und jede zulässige Lösung eine obere Schranke für das Optimum ist. o/m ist eine untere Schranke u.
Dies erhalten wir mit folgender Argumentation: Im schlechtesten Fall werden alle Jobs von einer Maschine erledigt. Wenn wir nun dies auf alle Maschinen gleichmäßig aufteilen, beträgt die Produktionszeit o/m. Wir erhalten somit eine durchschnittliche Produktionszeit auf jeder Maschine, die dann genauso schnell arbeiten müssten, wie die schnellste und haben somit eine untere Schranke für das Optimum.

Schritt 2: Die Binärsuche

Mit diesen Schranken starten wir jetzt eine binäre Suche und setzen $\vec{d} = (d, ..., d)^T \in \mathbb{N}^m$ anhand der jeweiligen oberen Schranke o und unteren Schranke u auf

$$d := \lfloor 1/2(o + u) \rfloor .$$

Auf dieses $\vec{d} = (d, ..., d)^T$ wenden wir das p-relaxierte Entscheidungsproblem an. Wenn wir „fast“ als Antwort erhalten, setzen wir o auf d, denn das Problem $Px \leq p\vec{d}$ besitzt eine zulässige Lösung, also ist d größer als das Optimum und somit eine obere Schranke. Sonst setzen wir u auf $d + 1$, was aufgrund der Ganzzahligkeit der Produktionszeiten eine untere Schranke für das Optimum ist. Irgendwann stimmen obere und untere Grenze überein, denn die Binärsuche terminiert, und somit ist die Optimallösung gefunden.
Der Algorithmus läuft klarerweise in polynomieller Zeit, denn nach k Iterationen haben wir eine Abweichung von 2^{-k} vom Optimum. □

Mit dem Rundungstheorem und dem vorhergehenden Lemma kommen wir zu folgender Aussage:

Theorem 4.1. *Es existiert ein polynomieller 2- Approximationsalgorithmus für das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit.*

Beweis. Nach dem obigen Lemma genügt es zu zeigen, dass ein 2-relaxiertes Entscheidungsproblem existiert. Seien also P und d gegeben. Wir wenden nun das Rundungstheorem aus dem vorherigen Kapitel mit

$$d_1 = d_2 = ... = d_m = t = d$$

an. Wenn das LP(P,$\vec{d}$,t) lösbar ist, erhalten wir eine zulässige Lösung. Die Lösung des Problems ist also nicht leer, wir finden also in polynomieller Zeit eine Ecke $\tilde{x}$[16][17]. Die gerundete Lösung $\overline{x}$ des GP(P,$\vec{d}$,t) kann als Maschineneinteilung für das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit gesehen werden. Wir weisen einfach einer Maschine einen Job zu, wenn $\overline{x}_{ij} = 1$ und sonst nicht.
Wenn wir keine Ecke finden, ist das LP nicht lösbar, wir haben also eine „Nein"-Instanz. Wir haben also ein 2-relaxiertes Entscheidungsproblem konstruiert. □

Wir haben also einen 2- Approximationsalgorithmus für das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit gefunden. Wir werden nun skizzieren, wie dieser Algorithmus arbeitet und dazu noch einen Pseudocode für den Algorithmus aufstellen. Danach werden anhand eines Beispielswir feststellen, dass die Abweichung in der Zielfunktion im Worst-Case nicht kleiner als der Faktor 2 ist.
Es seien (P, d) gegeben. Um festzustellen, ob d wirklich die Optimallösung für das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit ist, führen wir die in Lemma 1 beschriebene Binärsuche durch und setzen dann d gleich der Optimallösung.
Wir betrachten dann das lineare Problem $LP(P, \vec{d}, t)$, wobei wir $t = d_1 = d_2 = ... = d_m = d$ setzen. Zu diesem linearen Problem finden wir in polynomieller Zeit eine Ecke $\tilde{x}$, wenn das Problem zulässige Lösungen besitzt[16][17]. Diese Ecke $\tilde{x}$ können wir in polynomieller Zeit zu einer Lösung $\overline{x}$ des ganzzahligen Problems $GP(P, \vec{d}, t)$ runden. Da $t + d_i = 2d$ für alle $i = 1, ..., m$ gilt, finden wir eine Maschineneinteilung mit Laufzeit von höchstens $2d$ Zeiteinheiten. Die Einteilung erhält man dadurch, in dem man für $\overline{x}_{ij} = 1$ den Job j der Maschine i zuordnet und für $\overline{x}_{ij} = 0$ nicht.
In einem Pseudocode sieht der Algorithmus wie folgt aus:

1: **Input:**;
2: n Anzahl der Jobs;
3: $P \in^{m \times n}$ Produktionszeiten;
4: d Deadline;
5: Gehe zu Zeile 8;
6: {Schritt 1:};
7: Starte Binärsuche mit oberer Schranke o= Dauer des Plans, wo jeder Job der Maschine zugewiesen wird, die ihn am schnellsten erledigt und untere Schranke $u = o/m$;
8: **repeat**
9: **if** x existiert, so dass gilt: $Px \leq d$ **then**
10: Setze $o = d$;
11: Setze $d = 1/2 \lfloor o + u \rfloor$;
12: **else**
13: Setze $u = d + 1$;
14: Setze $d = 1/2 \lfloor o + u \rfloor$;
15: **end if**
16: **until** $o = u = d$
17: Gehe zu Zeile 20;
18: {Schritt 2:};
19: Betrachte $LP(P, d, t)$ mit $d = d_1 = d_2 = ... = d_m = t$;
20: Suche z.B. mit Hilfe Simplex-Algorithmus Ecke $\tilde{x}$ von $LP(P, d, t)$;
21: Gehe zu Zeile 24;
22: {Schritt 3:};
23: $\tilde{x}$ runden zu Lösung $\overline{x}$ von $GP(P, d, t)$ mit Theorem 3.1;
24: mit Einträgen $\overline{x}_{ij} = 0$, d.h. Maschine i erledigt Job j nicht, $\overline{x}_{ij} = 1$, d.h. Maschine i erledigt Job j;
25: **Output:**;
26: Einteilung, die höchstens $t + d = 2d$ Zeiteinheiten benötigt;

Wir werden im folgenden Beispiel feststellen, dass die Wahl der Ecke $\tilde{x}$ entscheidend für die Abweichung vom Optimum ist.

Beispiel 4.1. *a)Es gebe m^2-m+1 Jobs und m gleiche Maschinen. Der erste Job benötige m Zeiteinheiten, die restlichen eine Zeiteinheit.*
Wir überlegen uns folgende Einteilung: Der erste Job wird von einer Maschine erledigt, die restlichen Jobs werden gleichmäßig auf die anderen Maschinen verteilt. Die Produktionszeit beträgt

$$\frac{m^2-m}{m-1}=m.$$

Dies ist die Optimallösung für dieses Problem, da jede Maschine durchgängig produziert.
Wenn wir allerdings unseren oben gefundenen Algorithmus darauf anwenden, finden wir für kein $d \leq m$ eine Lösung des LP(P,$\vec{d}$,t). Wenn $d=m$ und $t=m-1$, gehören zu $J_i(t)$ auf jeder Maschine alle Jobs außer dem ersten und $M_j(t)$ ist für den ersten Job leer, für die anderen Jobs gehören alle Maschinen zu dieser Menge.
Wir überlegen uns folgenden Plan, der eine zulässige Lösung des LP(P,$\vec{d}$,t) ist: Zu jeder Maschine seien $m-1$ Jobs, die eine Zeiteinheit benötigen, zugewiesen. Der Job der m Zeiteinheiten benötigt, wird auf alle Maschinen aufgeteilt, es gilt also für diesen Job $x_{ij}=1/m$ für alle i. Die restlichen Jobs verteilen wir gleichmäßig auf die anderen Maschinen, erhalten also für sie eine Laufzeit von

$$\frac{m^2-m+1-m}{m-1}=m-1$$

Zeiteinheiten.
Diese Einteilung beschreibt eine Ecke des Polyeders, denn sie wird durch

$$n(\text{Bedingung 1})+\nu-n(\text{Variablen=0})=\nu$$

Gleichungen beschrieben. Sie kann also in polynomieller Zeit zu einer Lösung des ganzzahligen Problems gerundet werden(Theorem 3.1). Die gerundete Lösung gibt uns dann eine Maschineneinteilung, in dem wir den Job j der Maschine i zuordnen, wenn $x_{ij}=1$ und sonst nicht. Die Laufzeit hiervon ist allerdings $2m-1$, denn eine Maschine erledigt den Job, der m Zeiteinheiten benötigt und noch $m-1$ Jobs, die eine Zeiteinheit benötigen. Die Abweichung von der optimalen Laufzeit $m-1$ resultiert hier nicht aus dem Runden der Ecke, sondern aus der Wahl der Ecke.

b)Sogar wenn wir $m=2$ und somit $n=3$ setzen, ist keine bessere Worstcase-Schranke zu erreichen. Wir setzen

$$P=\begin{pmatrix}1 & q & \frac{3q+2}{2q+1}\\ q & q+1 & q+1\end{pmatrix}.$$

Die Optimallösung hat Laufzeit $q+1$, die ersten beiden Jobs ordnen wir Maschine 1 zu, den letzten Maschine 2. Wenn die Laufzeit kleiner $q+1$ wäre, könnten Job 2 und 3 nicht von Maschine 2 erledigt werden, die Summe der beiden Laufzeiten ist aber bei Maschine 1 größer als $q+1$.
Wenn wir nun in unserem LP(P, $\vec{d}$, t) $d:=q+1$ setzen, könnte die Ecke $\tilde{x}$ z.B. so aussehen:

$$\tilde{x}=\begin{pmatrix}0 & \frac{2q+1}{2q+2} & \frac{2q+1}{2q+2}\\ 1 & \frac{1}{2q+2}\frac{1}{2q+2} & \end{pmatrix}$$

Job 1 wird also Maschine 2 zugeordnet, die Jobs 2 und 3 dürfen beiden zugeordnet werden, allerdings dürfen wegen Bedingung 1 des LP nicht beide Jobs zu einer Maschine gehören. Da beide Jobs in Maschine 2 eine Laufzeit von $q+1$ haben, erhalten wir eine Gesamtlaufzeit von $2q+1$, wenn wir einen von beiden Maschine 2 zuweisen.
Wenn wir beide Maschine 1 zuweisen erhalten wir als Gesamtlaufzeit $q+\frac{3q+2}{2q+1}$ und sind fast optimal, diese Zuteilung ist aber aus oben genannten Gründen keine zulässige Lösung des LP(P, $\vec{d}$, t).

4.2 Ein PAS für das Laufzeiten Problem mit einer festen Anzahl von Maschinen

Wir können, wie bereits in der Einleitung erwähnt, das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit vereinfachen, indem wir nur eine feste Anzahl von Maschinen betrachten. Genauer erhalten wir folgendes Theorem:

Theorem 4.2. *Sei $m \in \mathbb{N}$. Dann existiert eine Familie A_ε von Algorithmen, derart, dass für jedes $\varepsilon>0$ A_ε ein $(1+\varepsilon)$-Approximationsalgorithmus für das Problem der Einteilung m unabhängiger parallel laufender Maschinen mit minimaler Laufzeit ist, dessen obere Schranke der Laufzeit ein Polynom der Inputlänge ist und dessen Speicherplatz durch ein Polynom abhängig von m, $ln(1/\varepsilon)$ und der Inputlänge beschränkt ist.*

Beweis. **Schritt 1: Klassifizierung der Jobs**

Wir wenden wiederum das vorherige Lemma an und müssen nun ein $(1+\varepsilon)$-relaxiertes Entscheidungsproblem finden. Für gegebene P und d klassifizieren wir die Jobs. Wir nennen sie „kurz“ bzw. „lang“, je nachdem, ob ihre Produktionsdauer auf der jeweiligen Maschine größer oder kleiner εd ist. Keine Maschine kann also mehr als $1/\varepsilon$ „lange“ Jobs vor Ablauf von d erledigen. Es gibt also maximal $(n+1)^{m/\varepsilon}$ verschiedene Einteilungen, denn jeder Maschine können 0 bis n Jobs zugewiesen werden, man hat also für jede Maschine $n+1$ Möglichkeiten und es gibt m/ε Plätze, denn jeder Maschine können maximal $1/\varepsilon$ „lange“ Jobs zugewiesen werden und es gibt m Maschinen.

Schritt 2: Anwenden des Rundungstheorems

Wenn das lineare Problem aus Theorem 3.1 eine zulässige Lösung besitzt, gibt es auch eine Lösung für die „langen“ Jobs (deren Menge auch leer sein kann). Wenn eine Maschine i für diese Jobs die Zeit t_i benötigt, dann hat sie für die restlichen Jobs $d_i = d - t_i$ Zeit. Wir setzen nun $t = \varepsilon d$, das LP(P,$\vec{d}$,t) muss immer noch eine Lösung besitzen, und wenden das Rundungstheorem an. Dies liefert uns eine Lösung für die kurzen Jobs, mit maximaler Dauer $d - t_i + \varepsilon d$. Insgesamt erhalten wir also

$$d - t_i + \varepsilon d + t_i = (1+\varepsilon)d.$$

Also rechnen wir dies für alle Kombinationen von „langen“ Jobs auf jeder Maschine durch, von denen es maximal $(n+1)^{m/\varepsilon}$ gibt, die Laufzeit ist also hierdurch beschränkt, wenden das Rundungstheorem an, die Laufzeit des Rundens war polynomiell, und erhalten so die Lösung.
Wir haben also entweder eine „nein“-Instanz oder eine Laufzeit von maximal $(1-\varepsilon)d$. Also haben wir eine $(1+\varepsilon)$-relaxierte Entscheidung gefunden, und somit ist die Behauptung bewiesen. □

Wir finden also ein PAS für eine feste Anzahl von Maschinen, dessen Laufzeit allerdings von $(n+1)^{m/\varepsilon}$ abhängt und das deswegen kein FPAS ist. Die Frage ist natürlich, was die Verbesserung zu dem bereits gefundenen FPAS ist[10]: Der Speicherplatz, den dieser Algorithmus benötigt, ist beschränkt durch ein Polynom abhänigig von m, $ln(1/\varepsilon)$ und der Inputlänge. Der Platz für das alte FPAS ist beschränkt durch $(nm/\varepsilon)^m$.
Wir werden nun anhand eines Beispiels die Arbeitsweise des PAS veranschaulichen.

Beispiel 4.2. *Wir betrachten wieder die Probleminstanz aus dem oberen Beispiel mit $q = 3$, also $m = 2$, $n = 3$ und*

$$P = \begin{pmatrix} 1 & 3 & \frac{11}{7} \\ 3 & 4 & 4 \end{pmatrix}.$$

Wir setzen $d = 4$, was auch gleichzeitig die Optimallösung ist und $\varepsilon = \frac{3}{4}$. Zu Maschine 1 existiert also ein „langer“ Job, auf Maschine 2 sind alle Jobs „lang“. Wir setzen nun $t = 3 = \varepsilon d$ und runden mit Theorem 3.1 die Ecke

$$\tilde{x} = \begin{pmatrix} 0 & \frac{7}{8} & \frac{7}{8} \\ 1 & \frac{1}{8} & \frac{1}{8} \end{pmatrix}.$$

Den auf „Maschine“ 1 kurzen Job 1 weisen wir Maschine 1 nicht zu, somit wird er Maschine 2 zugeordnet. Maschine 1 weisen wir den bzgl. dieser Maschine „kurzen“ Job 3 zu. Da Job 1 nicht Maschine 1 zugewiesen wird, weisen wir ihn automatisch Maschine 2 zu. Bislang hat Maschine 1 eine Laufzeit $\frac{11}{7}$ und Maschine 2 die Laufzeit 3. Als letztes testen wir, welcher Maschine Job 2 zugewiesen wird, damit die Produktion möglichst schnell zu Ende ist. Hierbei stellt sich heraus, diesen Job Maschine 1 zuzuweisen. Die Gesamtproduktionszeit beträgt also $\frac{11}{7} + 3 = 4,57$, was auf jeden Fall kleiner als $(1+\varepsilon)d = 7$ ist.

In diesem Kapitel haben wir gesehen, wie gut wir die Lösung des GP(P,$\vec{d}$,t), was dem Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit entspricht, approximieren können, je nachdem, ob die Maschinenanzahl zur Problemstellung gehört oder nicht. Im nächsten Kapitel werden wir einsehen, dass die Abweichung im Worst-Case nicht mehr viel geringer werden kann.

5 Grenzen der Approximation

Im vorherigen Kapitel haben wir einen 2- Approximationsalgorithmus für das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit gefunden. In diesem Kapitel werden wir zeigen, dass es nicht viel bessere Algorithmen geben kann, genauer ist der beste mögliche Approximationsalgorithmus ein 3/2- Approximationsalgorithmus. Gebe es einen besseren Approximationsalgorithmus, würde daraus P=NP folgen.
Wir werden dies anhand zweier Theoreme zeigen, die zur Aussage haben, dass das Entscheidungsproblem, ob die Laufzeit im Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit kleiner gleich 3 bzw. 2 ist, in NP-vollständig liegt. Dazu müssen wir das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit auf das 3-dimensionale Paarungsproblem reduzieren, das wie folgt aussieht:

Definition 5.1. *Das **3-dimensionale Paarungsproblem** ist ein Entscheidungsproblem mit folgendem Input:*
Es seien disjunkte Mengen $A := \{a_1, ..., a_n\}$, $B := \{b_1, ..., b_n\}$ und $C := \{c_1, ..., c_n\}$ gegeben. Zudem existiere eine Familie $F := \{T_1, ...T_m\}$ von Tripeln mit

$$|T_i \cap A| = |T_i \cap B| = |T_i \cap C| = 1 \quad \text{für } i = 1, ..., m$$

Die Frage ist, ob F eine Teilfamile F' mit $|F'| = n$ besitzt, so dass gilt:

$$\bigcup_{T_i \in F'} T_i = A \cup B \cup C$$

Wir müssen also aus einer Überdeckung eine Teilüberdeckung finden, so dass kein Element aus $A \cup B \cup C$ in 2 Tripeln enhalten ist. Von diesem Problem wurde bereits gezeigt, dass es in NP-vollständig liegt, woraus wir später die NP-Vollständigkeit unseres Einteilungsproblems schließen können.
Ein kleines Beispiel hierzu wäre:
$A = \{1, 2, 3\}$; $B = \{4, 5, 6\}$; $C = \{7, 8, 9\}$ und F=$\{1, 4, 7\}$; $\{1, 6, 9\}$; $\{2, 5, 8\}$; $\{2, 4, 7\}$; $\{3, 6, 9\}$; $\{3, 5, 8\}$. F' wäre in diesem Fall z.B.:$\{1, 4, 7\}$; $\{2, 5, 8\}$; $\{3, 6, 9\}$.
Im nachfolgenden Theorem werden wir den Bezug zwischen diesem Problem und dem Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit und gegebener Deadline herstellen.

Theorem 5.1. *Für das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit ist die Frage, ob ein Plan mit Laufzeit kleiner gleich 3 existiert, in NP-vollständig.*

Beweis. **Schritt 1: Der Bezug zwischen unserem Problem und dem 3-dimensionalen Paarungsproblem**

Sei eine Instanz für das obige Problem gegeben, wir konstruieren daraus eine Instanz für das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit mit m Maschinen und $2n + m$ Jobs. Maschine i gehöre zum Tripel T_i. Dann gibt es $3n$ Jobs, die der Menge $A \cup B \cup C$ zugeordnet sind. Es gibt also noch $m - n$ übrige Jobs. (Wenn m<n, konstruieren wir eine einfache Nein-Instanz für das Einteilungsproblem). Maschine i des Tripels $T_i = (a_j, b_k, c_l)$ kann jeden Job zu a_j, b_k und c_l in einer Zeiteinheit erledigen, jeden anderen in 3 Zeiteinheiten. Die übrigen $m - n$ Jobs benötigen also 3 Zeiteinheiten auf jeder Maschine.

Schritt 2: Paarung existiert $\Leftrightarrow$ Einteilung zu unserem Problem existiert

Wir zeigen also jetzt, dass ein Plan mit Laufzeit 3 genau dann existiert, wenn eine 3-dimensionale Paarung existiert.
" $\Leftarrow$ ": Es existiere eine Paarung. Für jedes Tripel $T_i = (a_j, b_k, c_l)$ haben wir also 3 Jobs a_j, b_k und c_l, die in unserem Plan zur Maschine i eingeteilt werden. Also werden $3n$ Jobs von den zugehörigen n Maschinen produziert. Die $m - n$ übrigen Jobs gehören zu Triples außerhalb der Paarung und werden von den $m - n$ übrigen Maschinen erledigt.
" $\Rightarrow$ ": Es existiere eine Einteilung. Jeder von den $m - n$ übrigen Jobs benötigt drei Zeiteinheiten auf jeder Maschine, also produzieren $m - n$ Maschinen jeweils einen der übrigen Jobs. Die restlichen n Maschinen produzieren $3n$ Jobs, für die sie jeweils eine Zeiteinheit benötigen. Jeder der drei Jobs auf einer Maschine gehört also zu einem Element im Maschinentripel, darin kommt jedes Element einmal vor, also ergibt sich aus den n Maschinen, die keinen der übrigen Jobs produzieren, eine Paarung. □

Mit diesem Theorem erhalten wir die erste Worstcase-Schranke für einen Approximationsalgorithmus:

Korollar 5.1. *Für jedes $p < \frac{4}{3}$ existiert kein p-Approximationsalgorithmus für das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit, wenn nicht P=NP.*

Begründung: Wenn ein p-Approximationsalgorithmus mit $p < \frac{4}{3}$ existieren würde, könnten wir in polynomieller Zeit entscheiden, ob für $d = 3$ und $p = 1{,}1$ z.B. eine zulässige Lösung $\tilde{x}$ des LP(P,,t)

$$Px \leq pd = 3,3$$

existiert, was aber aufgrund der Ganzzahligkeit der Produktionszeiten das gleiche Problem wie $Px \leq 3$ ist. Somit hätten wir ein Problem in NP-vollständig in polynomieller Zeit gelöst, woraus P=NP folgen würden.

Also haben wir gezeigt, dass es keinen besseren Approximationsalgorithmus als einen 4/3-Approximationsalgorithmus gibt. Wir können dieses Resultat mit Hilfe des folgenden Theorems noch verstärken:

Theorem 5.2. *Für das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit ist die Frage, ob ein Plan mit Laufzeit kleiner gleich 2 existiert, in NP-vollständig.*

Beweis. **Schritt 1: Der Bezug zwischen unserem Problem und dem 3-dimensionalen Paarungsproblem**

Wir beweisen dies wieder mit dem 3-dimensionalen Paarungsproblem. Wir nennen Triples, die a_j enthalten, **Tripel vom Typ j**. Sei t_j die Anzahl der Triples vom Typ j für $j = 1, ..., n$. Wie im vorherigen Beweis gehört Maschine i zum Tripel T_i für $i = 1, ..., m$. Es gibt nun $2n$ Jobs, die den Elementen aus $B \cup C$ zugewiesen sind. Es gibt $t_j - 1$ übrige Jobs für $j = 1, ..n$, der zu a_j gehörige Job wird einer der t_j Maschinen, die zu dem Tripel des Typs j gehören, zugewiesen und es bleiben somit $t_j - 1$ Jobs übrig. (Es gibt also insgesamt wieder $m - n$ übrige Jobs, denn es gibt m Maschinentripel, die auf n Jobtripel aufgeteilt sind, also bleiben $m - n$ Jobs übrig.) Maschine i gehöre zu einem Tripel vom Typ j, z.B. $T_i = (a_j, b_k, c_l)$. Maschine i benötige für b_k und c_l eine Zeiteinheit, für a_j zwei und für die restlichen 3 Zeiteinheiten.

Schritt 2: Paarung existiert $\Leftrightarrow$ Einteilung zu unserem Problem existiert

Wir zeigen nun analog zum vorherigen Beweis, dass eine Paarung existiert, genau dann wenn ein Zeitplan mit Laufzeit kleiner gleich 2 existiert.
„$\Rightarrow$“: Es existiere eine Paarung. Für jedes $T_i = (a_j, b_k, c_l)$ in der Paarung, teilen wir die Jobs b_k und c_l zur Maschine i ein. Für jedes j existieren $t_j - 1$ leerlaufende Maschinen, denen wir $m - n$ übrig gebliebene Jobs zuweisen. Also haben wir einen Plan mit Laufzeit 2.
„$\Leftarrow$“: Es existiere eine Einteilung. Jeder der übrigen Jobs ist einer Maschine zugeteilt, die zu einem Tripel vom Typ j gehört. Also existiert für jedes $j = 1, .., n$ eine Maschine, die keinen der übrigen Jobs erledigt. Diesen Maschinen teilen wir 2 Jobs zu, für die sie jeweils eine Zeiteinheit benötigen. □

Die Aussage des ersten Korollars kann also noch verstärkt werden:

Korollar 5.2. *Für alle $p < 3/2$, existiert kein p-Approximationsalgorithmus für das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit, wenn nicht P=NP.*

Der Beweis läuft analog zum Beweis von Korollar 5.1 nur unter Verwendung des Theorems 5.1 statt Theorem 5.1.

Wir haben also mit dem 2- Approximationsalgorithmus bzgl. der Worst-Case-Schranke in der Abweichung fast den bestmöglichen Fall erreicht. Der beste Approximationsalgorithmus, der zu diesem Problem existieren kann, ist ein 1,5-Approximationsalgorithmus. Es wird also auch in Zukunft unmöglich bleiben, für jedes P eine Einteilung mit mininmaler Laufzeit zu finden.

6 Feste Anzahl an Laufzeiten

Nun werden wir das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit für den Fall untersuchen, dass nur eine feste Anzahl fester Laufzeiten vorliegt. Dies ist in bestimmten Fällen eine wesentliche Vereinfachung, wenn z.B. alle Laufzeiten gleich sind, lässt sich das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit in polynomieller Zeit lösen.

6.1 Die Fälle $p_{ij} \in \{1, 2\}$ und $p_{ij} \in \{1, \infty\}$

Wir zeigen nun, dass wenn $p_{ij} \in \{1, 2\}$, das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit und gegebener Deadline immer noch in polynomieller Zeit lösbar ist.

Theorem 6.1. *Wir finden in polynomieller Zeit zum Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit eine Einteilung mit Laufzeit kleiner gleich d, wenn alle* $p_{ij} \in \{1, 2\}$.

Beweis. **Schritt 1: Das Teilgraphenproblem**

Wie im vorherigen Kapitel werden wir die Äquivalenz zu einem anderen Problem zeigen, von dem bekannt ist, dass es in polynomieller Zeit gelöst werden kann. Das Problem sieht wie folgt aus:
Sei G=(S,T,E) ein bipartiter Graph. Die Aufgabe ist es einen Teilgraphen mit maximaler Anzahl an Kanten zu finden, in dem jeder Knoten in S Grad 0 oder 2 hat und jeder Knoten in T Grad 1. Dies geschieht in polynomieller Zeit[18].

Schritt 2: Der Bezug des Teilgraphenproblems zu unserem Problem, für $d = 2k$

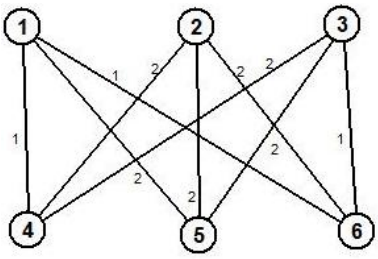
Für die Deadline $d = 2k$, konstruieren wir G wie folgt: In S gebe es k Knoten für jede Maschine, diese kann man als time slots der Länge 2 in den Maschinen interpretieren, in T gebe es für jeden Job einen Knoten, die Kanten gehören zu den Produktionszeiten der Länge 1. Nun lösen wir das Teilgraphenproblem in G. Daraus bekommen wir eine Teileinteilung mit den Jobs, die eine Zeiteinheit benötigen, aus dem Teilgraphen und wir versuchen dies zu einer Gesamteinteilung zu erweitern, in dem wir die Jobs mit Länge 2 den verbliebenen Maschinen zuweisen und dabei die Deadline einhalten. Wenn eine Einteilung mit Laufzeit $2k$ existiert, finden wir sie so.

Schritt 3: Vorgehen bei ungeradem d

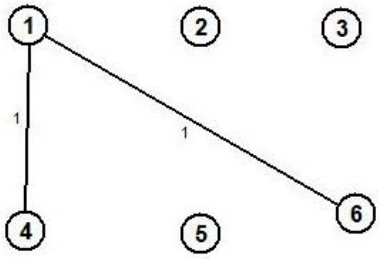
Wenn nun allerdings $d = 2k - 1$, dann fügen wir zu dem obigen Plan einen Job zu jeder Maschine hinzu, der an der jeweiligen Maschine eine Zeiteinheit benötigt und an den restlichen 2. Dieses Problem hat Laufzeit $2k$ genau dann, wenn das ursprüngliche Problem Laufzeit $2k - 1$ hatte. □

Dieser Fall ist natürlich äquivalent zu $p_{ij} \in \{l, 2l\}$ für l beliebig aus $\mathbb{N}$, denn die skalare Multiplikation ändert das Problem nicht.
Anhand eines Beispiels machen wir uns klar, was hier passiert:

Beispiel 6.1. *Sei* $P = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 2 & 2 \\ 2 & 2 & 1 \end{pmatrix}$ *und* $d = 2$. *Der Graph hierzu sieht wie folgt aus:*



Daraus kann man in polynomieller Zeit folgenden Teilgraph finden, in dem jeder Jobknoten Grad 0 oder 2 hat:



Es Job 1 und Job 3 werden also Maschine 1 zugewiesen. Job 2 können wir entweder Maschine 2 oder 3 zuweisen, die Deadline wird in beiden Fällen nicht überschritten.

Wenn $p_{ij} \in \{1, \infty\}$ gilt, kann man durch ähnliches Vorgehen wie oben eine Einteilung finden. Bei einer Deadline $d = k$ gebe es zu jeder Maschine k Knoten, die wiederum time slots diesmal der Länge 1 repräsentieren, und zu jedem Job einen Knoten. Die Kanten seien die (i, j), für die $p_{ij} = 1$ gilt. Nun teilen wir die Jobs so auf, dass jedem der Jobknoten ein Maschinenknoten zugewiesen wird. Finden wir keine solche Einteilung, existiert auch keine Einteilung mit Laufzeit maximal $d = k$. Für den Fall $p_{ij} \in \{l, \infty\} : l \in \mathbb{N}$ gehen wir analog vor.

6.2 Der allgemeine Fall zweier Laufzeiten

Wir werden aber gleich sehen, dass das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit allerdings für jede andere feste Anzahl an Laufzeiten nicht gelöst werden kann, wenn nicht $P = NP$.

Theorem 6.2. *Das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit sowie gegebener Deadline d ist in NP-schwer, wenn alle* $p_{ij} \in \{p, q\}$ *mit* $p<q$ *und* $2p \neq q$

Beweis. OBdA seien p und q teilerfremd. Ansonsten teilen wir p, q und d durch den größten gemeinsamen Teiler von p und q.
Für $p_{ij} \in \{1, 3\}$ haben wir die Behauptung schon in Theorem 5.1 durch das reduzieren auf die 3-dimensionale Paarung gezeigt, denn NP-vollständig liegt in NP-schwer.
Wir schreiben das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit in eine q-dimensionale Paarung um. Es sei eine Paarungsinstanz mit m q-Tupeln über einer Grundmenge mit qn Elementen gegeben. Daraus bilden wir eine Instanz mit qn Jobs, $p(m - n)$ übrigen Jobs und m Maschinen. Ein Job kann in p Zeiteinheiten von jeder Maschine, die einem Tupel zugewiesen ist, das das Jobelement enthält, erledigt werden. Alle anderen Herstellungszeiten seien q. Es existiert also für jede Paarung eine Einteilung mit Laufzeit pq. Daraus folgt, dass jede Maschine entweder q ihr zugewiesene Jobs erledigt oder p der übrigen Jobs. □

7 Fazit

Wir wollen nun ein Fazit dieser Arbeit ziehen.
Zunächst haben wir ein Rundungstheorem bewiesen, das grundlegend für den Approximationsalgorithmus war. Dort haben wir im Prinzip die Ganzzahligkeitsbedingungen des Problems der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit vernachlässigt, um dann die Ecken des erhaltenen linearen Problems zu einer Lösung des ganzzahligen Problems zu runden. Dieses Runden geschah in polynomieller Zeit, was später wichtig für die Laufzeit des Algorithmus war.
Der Algorithmus selbst, der im Mittelpunkt dieser Arbeit stand, war ein 2- Approximationsalgorithmus, der nur polynomielle Zeit benötigte. Hierbei haben wir keinerlei Einschränkungen bzgl. Laufzeiten oder Maschinenanzahl gemacht, wie es bei früher gefundenen Algorithmen mit ähnlicher Gütegarantie der Fall war. Der beste vor unserer Arbeit gefundene Approximationsalgorithmus hatte eine Gütegarantie von $2\sqrt{m}$, hing also von der Maschinenanzahl m ab, wir erreichten also eine erhebliche Verbesserung in der Gütegarantie.
Im selben Kapitel haben wir für eine feste Anzahl m an Maschinen ein PAS gefunden. Dieses setzte sich im Vergleich zu dem bereits gefundenen FPAS durch einen erheblich geringeren Speicherplatz ab. Leider ist es bis zum jetzigen Zeitpunkt nicht möglich, ein FPAS mit geringerem Speicherplatz zu finden, dies wird wohl ein Problem sein, mit dem man sich auseinandersetzen kann, wenn man sich in Zukunft mit dem Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit beschäftigt.
Im nachfolgenden Kapitel haben wir eingesehen, dass es unmöglich ist, einen polynomiellen Approximationsalgorithmus mit einer besseren Gütegarantie als 1,5 zu finden, wenn nicht P=NP. Es wird vermutet, dass die Aussage P=NP falsch ist, allerdings ist es bisher noch nicht gelungen, dies zu beweisen. Wir können also höchstwahrscheinlich davon ausgehen, dass kein Approximationsalgorithmus mit einer Gütegarantie kleiner gleich 1,5 existiert. Da wir einen Approximationsalgorithmus mit Gütegarantie 2 gefunden haben, sind wir relativ nah am Optimum. Vielleicht werden in Zukunft auf Basis unserer Arbeit polynomielle Approximationsalgorithmen gefunden, deren Gütegarantie näher am Optimum 1,5 sind.
Zum Schluss veränderten wir das Problem der Einteilung unabhängiger parallel laufender Maschinen mit minimaler Laufzeit, in dem wir die Anzahl an Produktionszeiten festhielten. Hierbei stellten wir aber fest, dass diese Einschränkung nur in den Fällen $p_{ij} \in \{1, 2\}$ oder $p_{ij} \in \{1, \infty\}$ das Problem vereinfacht und das Entscheidungsproblem, ob eine Einteilung mit Laufzeit kleiner gleich d existiert, in polynomieller Zeit lösbar ist. Für den Fall $p_{ij} \in \{p, q\}$ mit $2p \neq q$ ist das gleiche Entscheidungsproblem in NP-schwer.

Literatur

[1] A. Martin and S. O. Krumke. *Diskrete Optimierung*. Springer, 2009. noch nicht erschienen.

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

[3] Davis and Jaffe. 1981.

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

[5] S. Sahni. Algorithms for scheduling independent tasks. *Journal of the Association for Computing Machinery*, 23:116–127, 1976.

[6] M.R. Garey and D.S. Johnson. Strong NP-Completeness results: motivation, examples and implications. *Journal of the Association for Computing Machinery*, 25:499–508, 1978.

[7] M.R. Garey and D.S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4:397–411, 1975.

[8] D.S. Hochbaum and D.B. Shmoys. Using dual approximation algorithms for scheduling problems: practical and theoretical results. *Journal of the Association for Computing Machinery*, 34:144–162, 1987.

[9] O.H. Ibarra T. Gonzales and S. Sahni. Bounds for LPT schedules on uniform processors. *SIAM Journal on Computing*, 6:155–166, 1977.

[10] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling nonidentical processors. *Journal of the Association for Computing Machinery*, 23:317–327, 1976.

[11] D.S. Hochbaum and D.B. Shmoys. A polynomial approximation scheme for machine scheduling on uniform processors: using the dual approximation approach. *SIAM Journal on Computing*, 17:539–551, 1988.

[12] C.N. Potts. Analysis of a linear programming heuristic for scheduling unrelated parallel machines. *Discrete Applied Mathematics*, 10:155–164, 1985.

[13] K. Numata. Approximate and exact algorithms for scheduling independent tasks on unrelated processors. *Discrete Applied Mathematics*, 31:61–81, 1988.

[14] J.K. Lenstra. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

[15] R. Diestel. *Graphentheorie*. Springer, 2006.

[16] L.G. Khachian. A polynomial time algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.

[17] L. Lovász M. Grötschel and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.

[18] A. Schrijver. Min-max results in combinatorial optimization. *Mathematical Programming: The State of the Art*, pages 439–500, 1983.