Analyse einer Heuristik zur Reihenfolgeplanung paralleler Maschinen

Analysis of a heuristic for scheduling unrelated parallel machines

Bachelor-Thesis von Lars Bauer

August 2010

TECHNISCHE UNIVERSITÄT DARMSTADT

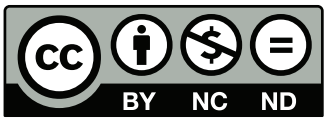
Fachbereich Mathematik
Arbeitsgruppe Optimierung

Analyse einer Heuristik zur Reihenfolgeplanung paralleler Maschinen
Analysis of a heuristic for scheduling unrelated parallel machines

Vorgelegte Bachelor-Thesis von Lars Bauer

1. Gutachten: Dr. habil. Marco 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 6. August 2010

(L. Bauer)

Inhaltsverzeichnis

Algorithmenverzeichnis

Kapitelübergreifende Notation

n	Anzahl der gegebenen Aufträge
m	Anzahl der gegebenen Maschinen
P	Matrix der gegebenen Bearbeitungsdauern mit Einträgen p_{ij}
p_{ij}	Bearbeitungsdauer des Auftrags j, ausgeführt durch Maschine i
X	Verteilungsmatrix mit Einträgen $x_{ij} \in \{0, 1\}$
x_{ij}	trägt Wert 1 falls Auftrag j durch Maschine i ausgeführt wird
C^*_{max}	Minimum aller durch mögliche Lösungsalgorithmen bestimmten Gesamt-Fertigstellungszeiten
C^X_{max}	Gesamt-Fertigstellungszeit, berechnet durch einen Algorithmus bzw. eine Heuristik X
ρ	Worst-Case Quotient

1 Problemstellung

Diese Arbeit befasst sich mit der Problemstellung, eine gegebene Anzahl an Aufträgen auf eine ebenfalls gegebene Anzahl vorhander Maschinen zu verteilen. Dabei sollen die Aufträge so verteilt werden, dass die benötigte Zeit zur Fertigstellung aller Aufträge minimal wird.

Anschauliche Beispiele eines solchen Problems sind gegeben durch:

- Verteilung von Aufgaben auf verschiedene Personen mit unterschiedlicher Arbeitsgeschwindigkeit je nach Themengebiet
- Bearbeitung von unabhängigen Prozessen durch parallel arbeitende Computerprogramme verschiedener Prozessoren

Allen Problemen ist gemein, dass die Maschinen selbst parallel zueinander arbeiten, d.h. ihre Aufträge unabhängig voneinander zeitgleich ausführen. Somit ergibt sich als Fertigstellungsdauer gerade die Zeit, die der Dauer der am längsten arbeitenden Maschine entspricht.
(„makespan“)

Bei der Erstellung eines Ablaufplans ist also zu gewährleisten, dass sich die Fertigstellungszeit jeder einzelnen Maschine der durchschnittlich benötigten Zeit aller Maschinen annähern. Hinzu kommt, dass bestimmte Aufträge durch gewisse Maschinen schneller bearbeitet werden können, wodurch das Problem zusätzlich erschwert wird.

Eine triviale Lösung der Problematik ist durch einen Algorithmus gegeben, der alle potentiellen Lösungsmöglichkeiten für einen Ablaufplan testet und am Ende eine optimale Lösung ausgibt. Diese Methode nennt man „Brute-Force-Suche“. Bei Problemstellungen, für die keine besseren Lösungsalgorithmen bekannt sind, steht dem Anwender nur ein solcher Brute-Force-Algorithmus zur Verfügung. Jedoch besitzt der Algorithmus keine polynomielle Laufzeit, sodass eine praktische Anwendung in den meisten Fällen nicht möglich ist.

Im Folgenden wird ein mathematisches Modell vorgestellt, um das obige Problem zu modellieren. Später wird unter anderem Heuristiken vorgestellt, mit deren Hilfe eine hinreichend genaue Lösung des Problems in polynomieller Zeit gefunden werden kann.

2 Modellbildung

Es seien m unabhängige, parallel arbeitende Maschinen gegeben, auf die n Aufträge verteilt werden sollen. Jede Maschine kann ohne Unterbrechung zu jeder Zeit genau einen Auftrag ausführen. Wir bezeichnen die (stets positive) Bearbeitungsdauer eines Auftrags j ($j = 1, ..., n$), ausgeführt durch eine Maschine i ($i = 1, ..., m$), als p_{ij}.

Gilt $p_{ij} = p_j$ ($i = 1, ..., m; j = 1, ..., n$), so handelt es sich, bezogen auf den Auftrag j, um identische Maschinen. Gilt andererseits $p_{ij} = p_j/q_i$ für eine Laufzeit p_j des Auftrags j und für eine konstante Geschwindigkeit q_i der Maschine i, so arbeiten die Maschinen homogen. In beiden Fällen arbeiten die Maschinen selbst unabhängig voneinander.

Zusammengefasst ergibt sich somit im allgemeinen Fall eine Laufzeitmatrix P mit (beliebigen) positiven Einträgen,

Approximationen

Der Mathematiker *Karp* fand heraus, dass das Problem bereits im Falle von zwei identischen Maschinen zu den NP-schweren Problemen gehört. Demzufolge hat der Großteil der Forscher davon abgesehen, eine exakte Lösung bzw. einen exakten polynomiellen Lösungsalgorithmus finden zu wollen. Stattdessen versuchte man, geeignete Heuristiken zu finden, deren approximative Lösung der exakten möglichst nahe kommen. (vgl. [1, Seite 155])

Bei gegebenem beliebigem Datensatz definiert man C^*_{max} als minimale Gesamt-Fertigstellungszeit, berechnet durch mögliche Lösungsalgorihmen (oder Lösungsheuristiken). Bezeichnet man demgegenüber die durch Anwendung einer Heuristik H bestimmte Fertigstellungszeit als C^H_{max}, so gilt es, den Quotienten $C^H_{max}/C^*_{max} \leq \rho$ zu minimieren. Den Wert ρ nennt man hierbei „worst case performance ratio“. Die Qualität einer Heuristik H lässt sich somit anhand des Wertes ρ analysieren. Je kleiner ρ, desto besser arbeitet die gewählte Heuristik.

Verschiedene Forscher bzw. Forschungsgruppen haben versucht, Heuristiken zur Lösungsfindung zu verwenden. Folgende allgemeine Ergebnisse konnten dabei erziehlt werden:

- *Horowitz* und *Sani*: Algorithmus A_ε mit Laufzeit $O(n(n^2/e)^{m-1})$
  $\Rightarrow C^{A_\varepsilon}_{max}/C^*_{max} \leq 1 + \varepsilon$
- *Ibarra* und *Kim*: ECT Heuristik mit Laufzeit $O(mn^2)$
  $\Rightarrow C^{ECT}_{max}/C^*_{max} \leq m$
- *Davis* und *Jaffe*: verschiedene Heuristiken
  $\Rightarrow C_{max}/C^*_{max} \leq 2 + 3\sqrt{m}/2 + 1/(2\sqrt{m})$
- *Potts*: Heuristik LPE
  $\Rightarrow C^{LPE}_{max}/C^*_{max} \leq 2$

Hierzu sei erwähnt, dass die Forschungsergebnisse keineswegs unabhängig voneinander erarbeitet wurden. *Horowitz* und *Sani* veröffentlichen ihre Ergebnisse im Jahre 1976 (vgl. [2]), *Ibarra* und *Kim* ihre 1977 (vgl. [3]). Demgegenüber wurden die Ergebnisse von *Davis* und *Jaffe* im Jahre 1981 (vgl. [4]), die LPE-Heuristik von *Potts* sogar erst im Jahre 1985 (vgl. [1]) veröffentlicht. Im Laufe der Entwicklung eigener Heuristiken konnten folglich bekannte Erkenntnisse früherer Forschungsgruppen verwendet und verbessert werden.

3 Grundlagen

Innerhalb der im Hauptteil dieser Arbeit betrachteten LPE-Heuristik werden verschiedene Algorithmen der klassischen linearen Optimierung verwendet. So wird es unter anderem nötig sein, ein lineares Programm zu lösen. Ein solches lässt sich etwa mit Hilfe der Ellipsoidmethode in polynomieller Zeit lösen, während sich für die Praxis das Simplex-Verfahren anbietet. Beide Algorithmen lassen sich innerhalb der LPE-Heuristik verwenden bzw. implementieren. Darüber hinaus lässt sich, sofern man von einer kompletten Berechnung aller möglichen Verteilungpläne in einem späteren Schritt der Heuristik absieht, ein Branch-and-Bound Algorithmus realisiseren.

Im Spezialfall von nur zwei gegebenen Maschinen, auf die Aufträge verteilt werden, wird außerdem ein Median-Algorithmus zur Bestimmung des k-größten Elements einer Datenmenge verwendet. Im Folgenden werden die einzelnen Algorithmen kurz vorgestellt:

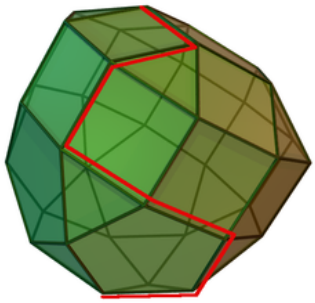
3.1 Simplex-Verfahren

Das Simplex-Verfahren ist derzeit das wichtigste Lösungsverfahren der linearen Optimierung. Es löst ein lineares Programm nach endlich vielen Schritten bzw. stellt dessen Unlösbarkeit oder Unbeschränktheit fest. Die Grundidee stammt von *George Dantzig* aus dem Jahre 1947, durch zahlreiche Verbesserungen wurde es seitdem ständig weiterentwickelt.

Die Grundidee des Verfahrens geht von einem linearen Programm folgender Form aus:
$max \left\{ c^T x | Ax \leq b, x \geq 0 \right\}$ mit $A \in \mathbb{R}^{m \times n}, c \in \mathbb{R}^n, b \in \mathbb{R}^m$

Jedes lineare Programm kann durch einfache Umformungen in obige Form gebracht werden. Ziel ist es, den Zielfunktionswert $c^T x$ unter den gegebenen Nebenbedingungen zu maximieren.

Die Menge der zulässigen Lösungen lassen sich geometrisch als konvexes Polyeder interpretieren. Man kann zeigen, dass das Optimum an einer der Ecken dieses Polyeders angenommen wird. (vgl. [5, Seite 46])



Das Simplex-Verfahren besteht darin, sich schrittweise von Ecke zu Ecke zu bewegen. Da die Zahl der Ecken eines Polyeders exponentiell in der Anzahl der Variablen und Ungleichungen sein kann (z.B. lässt sich der n-dimensionale Einheitswürfel durch $2n$ lineare Ungleichungen beschreiben), ergibt sich im ungünstigsten Fall eine exponentielle Laufzeit.

Der Simplex-Algorithmus selbst arbeitet in zwei Schritten. Der erste Schritt des Algorithmus besteht darin, eine zulässige Startlösung zu finden. (vgl. [5, Seite 95]). Im darauffolgenden Optimierungsschritt wird ausgehend von der gefundnen primal zulässigen Basis B (d.h. $A^{-1}._B b \geq 0$) eine Optimallösung für das lineare Programm gesucht. (vgl. [5, Seite 84])

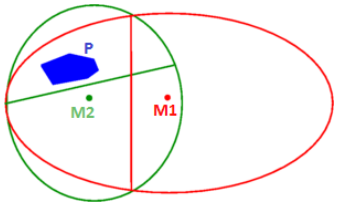
3.2 Ellipsoidmethode

Die Ellipsoidmethode wurde im Jahre 1976 entwickelt und im Jahre 1979 durch den russischen Mathematiker *Khachian* zu einem Lösungsalgorithmus für lineare Programme erweitert. Der Vorteil zum Simplex-Verfahren liegt darin, dass die Methode in jedem Fall in polynomieller Zeit zu einer Lösung führt.

Ausgangspunkt ist ein Polyeder folgender Form:
$P = \{x \in \mathbb{R}^n | Ax \leq b\}$ mit $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$

Die Methode prüft, ob ein gegebenes Polyeder P leer ist oder nicht. Man kann zeigen, dass diese Entscheidung äquivalent zum Finden der Optimallösung eines linearen Programms ist. (vgl. [5, Seite 127])

Der Algorithmus konstruiert ein Ellipsoid, das P enthält und betrachtet anschließend den Mittelpunkt M dieses Ellipsoids. Liegt M in P, so terminiert der Algorithmus. Liegt M nicht in P, so wird eine neues, kleineres Ellipsoid geeignet konstruiert und der Vorgang wiederholt sich. (vgl. [5, Seite 140])



3.3 Branch-and-Bound Algorithmen

Branch-and-Bound (vgl. [6, Seite 253/254]) ist eine Methode, deren Ziel darin besteht, für ein gegebenes ganzzahliges Optimierungsproblem schrittweise eine Lösung zu finden. In jedem Schritt werden mögliche zulässige Lösungen des Optimierungsproblems ermittelt und mit vorhergehenden verglichen.

Bei vielen Optimierungsproblemen ist es nicht in polynomieller Zeit möglich, alle zulässigen Lösungen eines Problems zu testen. Das Branch-and-Bound Verfahren führt dementsprechend auf einen Entscheidungsbaum für mögliche Lösungen. Dabei wird zunächst der zulässige Bereich in mehrere Teilmengen aufgespaltet (Branch). Durch geeignete Festlegung von unteren und oberen Schranken (Bound) werden dann bereits frühzeitig suboptimale Zweige des Entscheidungsbaums erkannt und verworfen.

Initialisierung:
Ausgangspunkt der Methode (Wurzel des Entscheidungsbaums) ist ein ganzzahliges Optimierungsproblem. Das Problem lässt sich durch Relaxation gegebenenfalls erweitern. Desweiteren ist bei Minimierungsproblemen eine aktuelle obere Schranke für den Zielfunktionswert, bei Maximierungsproblemen eine untere Schranke für den Zielfunktionswert anzugeben. Lässt sich keine Schranke z.B. anhand einer bekannten zulässigen Lösung ableiten, so wird $+\infty$ bzw. $-\infty$ gewählt.

Branch-Schritt:
Der Branch-Schritt dient dazu, das Problem in zwei oder mehr Teilprobleme aufzuspalten. Durch rekursives Ausführen wird eine Baumstruktur erzeugt. Man unterscheided zwischen zwei Verfahren, die hier Anwendung finden:

Tiefensuche: Von den noch nicht bearbeiteten Teilproblemen wird das gewählt, welches zuletzt eingefügt wurde. (Last In - First Out)
Beitensuche: Es wird das Teilproblem ausgewählt, welches als erstes in den Baum eingefügt wurde. (First In - First Out)

Beide Verfahren sind kombinierbar. Von einer durch Tiefensuche bestimmten lokale Lösung lässt sich beispielsweise durch eine anschließende Breitensuche bereits eine globale obere bzw. untere Schranke für die zulässigen Lösungen des Optimierungsproblems ermitteln.

Bound-Schritt:
Der Bound-Schritt hat die Aufgabe, suboptimale Zweige des Entscheidungsbaums „abzuschneiden“. Durch den Verzicht der Betrachtung dieser Zweige wird der Rechenaufwand reduziert.

Ein Knoten des Baumes wird nicht weiter betrachtet, wenn etweder nachgewiesen wird, dass der nach vorherigem Branch-Schritt erzeugte Lösungsraum leer ist, oder aber wenn z.B. im Falle eines Maximierungsproblems eine obere Schranke angegeben werden kann, die den Wert der aktuellen unteren Schranke nicht übersteigt.

Die Hauptaufgabe innerhalb eines Branch-and-Bound Algorithmus besteht in der Festlegung der Schranken. Diese werden gewöhnlich (evtl. durch Anwendung separater Heuristiken) vom bekannten Zielfunktionswert abgeleitet. Anzahl und Schärfe der festzulegenden Schranken pro Schritt sind hierbei beliebig und abhängig vom gewünschten Rechenaufwand.

3.4 Median-Algorithmus

Generell ist es möglich, sowohl Minimum, als auch Maximum einer Datenmenge in linearer Zeit zu ermitteln. Eine komplette Sortierung der Daten, bzw. das Auffinden des k-größten Elements, ist mit Hilfe bekannter Algorithmen (z.B. heap sort) in $O(n \log n)$ durchführbar. (vgl. [7, Seite 218ff.])

Der Median-Algorithmus (vgl. [8, Seite 45/46]) ist demgegenüber ein Algorithmus, der auch das Auffinden des Elements vom Rang k in linearer Zeit ermöglicht. Er arbeitet nach dem Divide-and-Conquer Prinzip. Ausgehend von einer Folge $a_0, ..., a_{n-1}$ der Länge $n \in \mathbb{N}$ lässt sich die Zahl $k \in \{0, ..., n-1\}$ und somit das Element mit Rang k, auf folgende Weise ermitteln:

(1) Gilt $n = 1$, dann gilt: $k = a_0$

(2) Wähle ein beliebiges Element x

(3) Partitioniere die Folge in drei Teilstücke:
$a_0, ..., a_{q-1}$,
$a_q, ..., a_{g-1}$ und
$a_g, ..., a_{n-1}$

Alle Elemente im ersten Stück sollen kleiner als x, die Elemente im zweiten Stück gleich x und die Elemente im dritten Stück größer als x sein.

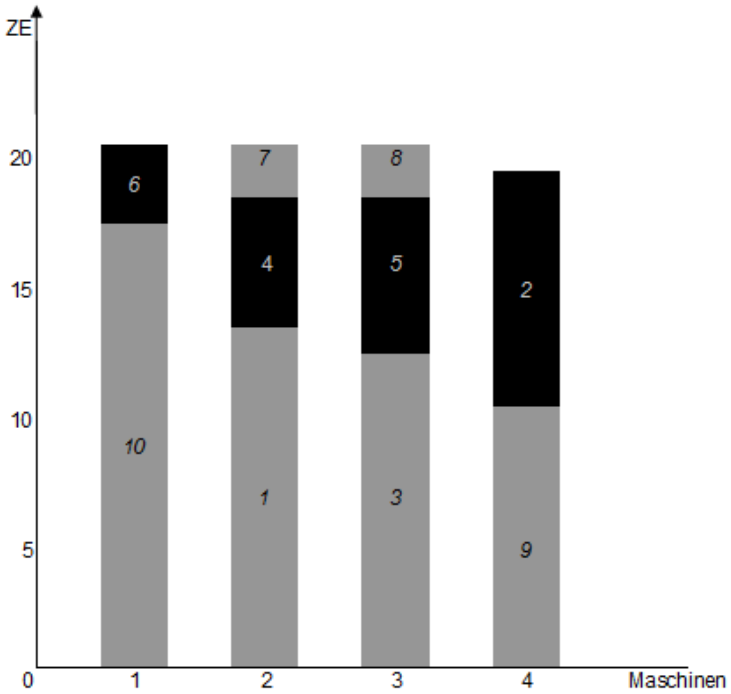
(4) Gilt $k < q$, dann: Wiederhole ab Schritt (1) für erstes Teilstück $a_0, ..., a_{q-1}$
Gilt $k < g$, dann gilt: $k = x$
Gilt $k \geq g$, dann: Wiederhole ab Schritt (1) für drittes Teilstück $a_g, ..., a_{n-1}$

4 ECT-Heuristik von Ibarra und Kim

Ein besonderes Augenmerk wird zunächst auf die bereits 1977 erarbeitete ECT-Heuristik von *Ibarra* und *Kim* geworfen. Die Heuristik verfolgt das LPTF-Prinzip („Longest Processing Time First"), d.h. Aufträge mit größerer Bearbeitungsdauer werden zuerst abgearbeitet.

In ihrer einfachsten Form sortiert eine nach dem LPTF-Prinzip arbeitende Heuristik zunächst die gegebenen Aufträge absteigend nach der jeweiligen zur Bearbeitung benötigten Zeit. Handelt es sich nicht um identische Maschinen, so wird zuvor eine durchschnittliche Bearbeitungsdauer aller Aufträge berechnet, bevor die Sortierung vorgenommen wird. Anschließend werden die m Aufträge, welche die längste (durchschnittliche) Bearbeitungsdauer besitzen, auf die m gegebenen Maschinen verteilt. Wird ein Auftrag beendet und eine Maschine frei, so wird der Auftrag mit der nächstgrößten (durchschnittlichen) Bearbeitungsdauer dieser Maschine zugeordnet. Da auf diese Weise die Aufträge mit kleinerer (durchschnittlich) benötigter Fertigstellungzeit zunächst nicht beachtet werden, kann am Ende für eine „Ausbalancierung" der Maschinen gesorgt werden. Es wird gewährleistet, dass keine der Maschinen erheblich früher als eine andere terminiert.

Beispiel:
Verteilung von 10 Aufträgen auf 4 identische Maschinen:



Aufträge:	benötige Zeit:
1	13 ZE
2	9 ZE
3	12 ZE
4	5 ZE
5	6 ZE
6	3 ZE
7	2 ZE
8	2 ZE
9	10 ZE
10	17 ZE

Die erste auf diesem Prinzip basierende Heuristik von *Ibarra* und *Kim* wurde im Jahre 1976 vorgestellt. Sie erzielte die im vorherigen Abschnitt erwähnten Ergebnisse.

Die Heuristik wurde im Laufe der Zeit jedoch modifiziert und weiter verbessert. Es sind insgesamt vier weitere auf der ECT-Heuristik basierende Heuristiken erarbeitet worden. Es findet sich unter anderem eine Variante der Heuristik, die der Abschätzung $C_{max}^{ECT}/C_{max}^{*} \leq 1 + \log_2 m$ genügt. Im Spezialfall $m = 2$ wurde sogar eine Variante mit Komplexität $O(n \log n)$ vorgestellt, für die $C_{max}^{ECT}/C_{max}^{*} \leq (1 + \sqrt{5})/2$ gilt.

Demgegenüber wird sich diese Arbeit in den folgenden Kapiteln genauer mit der LPE-Heuristik und deren Modifikationen befassen. Die Laufzeiten der Algorithmen werden insbesondere mit den verschiedenen Laufzeiten der ECT-Heuristiken von *Ibarra* und *Kim* verglichen. Der Hauptvorteil der LPE-Heuristik liegt darin begründet, dass das vorgestellte Problem in ein lineares Programm transferiert wird, welches in linearer Zeit mit Hilfe von Algorithmen der linearen Optimierung gelöst werden kann.

5 Die LPE-Heuristik von C.N. Potts

5.1 Allgemeine LPE-Heuristik

Zunächst sei erwähnt, dass die Fertigstellungszeit unverändert bleibt, sollte man die Reihenfolge der Aufträge, durchgeführt durch ein und dieselbe Maschine, ändern. Das Problem reduziert sich also auf die Zuordnung der Aufträge zu den jeweiligen Maschinen.

Sei hierzu definiert: $(i = 1, ..., m; j = 1, ..., n)$

$$x_{ij} = \begin{cases} 1 & \text{wenn Maschine } i \text{ Auftrag } j \text{ durchführt} \\ 0 & \text{sonst} \end{cases}$$

Die Gesamt-Fertigstellungszeit C_{max}^{LPE} ändert sich je nach Wahl der x_{ij}. Mit Hilfe der Zuordnungsvariablen x_{ij} und C_{max}^{LPE} als zu bestimmende Bearbeitungsdauer lässt sich das Problem wie folgt als lineares Programm darstellen:

$$\begin{array}{lll} \min & C_{max}^{LPE} & \\ \text{s.t.} & \sum_{j=1}^{n} p_{ij} x_{ij} \leq C_{max}^{LPE} \quad (i = 1, ..., m) & (1) \\ & \sum_{i=1}^{m} x_{ij} = 1 \quad (j = 1, ..., n) & (2) \\ & \mathrm{x}_{ij} \in \{0, 1\} \quad (i = 1, ..., m; j = 1, ..., n) & (3) \end{array}$$

(C_{max}^{LPE} wird also minimiert. Nebenbedinung (1) entspricht hierbei der Berechnung der Fertigstellungsdauer, die Nebenbedingungen (2) und (3) gewährleisten, dass jeder Auftrag auf genau einer Maschine ausgeführt wird.)

Die Heuristik von *Potts* (vgl. [1, Seite 156/157]) vereinfacht zunächst obiges Programm, indem in einer neuen Nebenbedingung, die (3) ersetzt, alle $x_{ij} \geq 0$ zugelassen werden. Nebenbedingung (2) gewährleistet demgegenüber $x_{ij} \leq 1$. Diese LP-Relaxation $0 \leq x_{ij} \leq 1$ bewirkt, dass Aufträge auf mehrere Maschienen aufgeteilt und die einzelnen Teile simultan bearbeitet werden können. Davon ausgehend arbeitet der Algorithmus in drei Schritten:

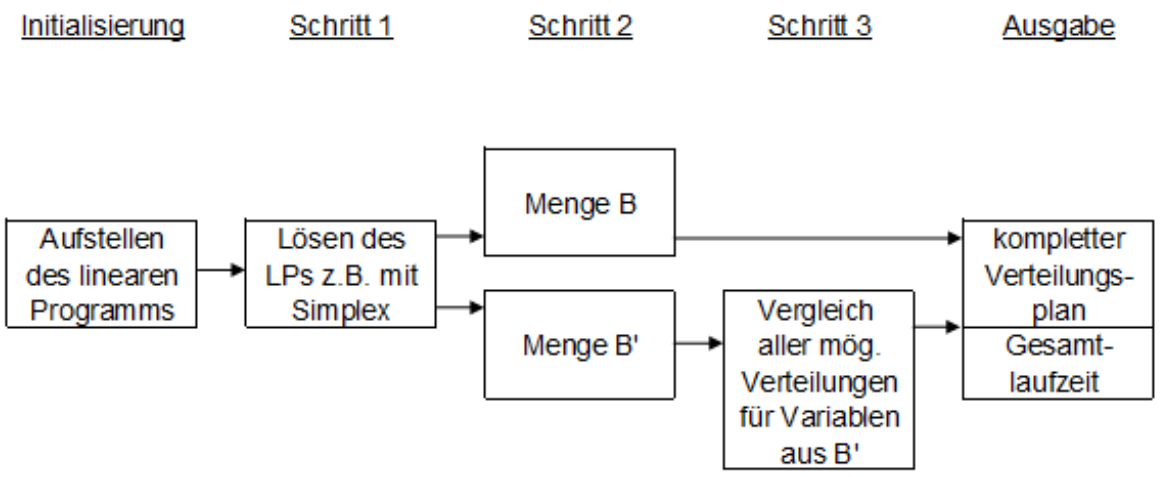
Schritt 1:
Im ersten Schritt wird ein Plan für die Verteilung eines Teils der gegebenen Aufträge aufgestellt. Dieser Teilplan enthält alle Aufträge, die nur an einer Maschine ausgeführt werden, also für die $x_{ij} = 1$ gilt. Die zu den nicht ganzzahligen x_{ij} gehörigen Aufträge j werden zunächst als unbedeutende Aufträge angesehen. Im günstigsten Fall enthält der Teilplan bereits alle Aufträge und stimmt somit mit einer Optimallösung des ursprünglichen Problems überein.

Das vereinfachte lineare Programm besitzt $m + n$ Nebenbedingungen und die Nichtnegativitätsbedingung aller x_{ij}. Ein solches Programm kann z.B. mit Hilfe des Simplex-Algorithmus (vgl. Kapitel 3.1) gelöst werden. Man erhält eine Basislösung des LPs mit $m + n$ Basisvariablen, während den restlichen Variablen der Wert 0 zugeordnet wird. Da $C_{max}^{LPE} > 0$ gilt, besteht die Basislösung aus eben dieser Variablen C_{max}^{LPE}, sowie $m + n - 1$ Variablen x_{ij}.

Schritt 2:
Es wird nun eine Menge B aus n Basisvariablen so definiert, dass jede der n Nebenbedingungen von (2) genau eine Variable aus der Menge B enthält. Die übrigen $m-1$ Basisvariablen werden in einer Menge B' zusammengefasst.

Für $n \geq m-1$ enthalten maximal $m-1$ Nebenbedingungen von (2) zwei oder mehr Variablen vom Wert größer 0. Im Umkehrschluss enthalten mindestens $n-m+1$ Nebenbedingungen nur eine Variable vom Wert größer 0. Es existieren also mindestens $n-m+1$ Variablen x_{ij} vom Wert 1, die im Teilplan auftauchen, maximal $m-1$ Variablen bleiben nach Durchführung dieses ersten Schritts unsortiert. (Für $n \leq m-1$ können wir keine solche Aussage treffen. In diesem Fall kann für alle Variablen $x_{ij} < 1$ gelten, sodass der im ersten Schritt generierte Teilplan leer wäre.)

Schritt 3:
Im dritten Schritt des Algorithmus werden schließlich die verbleibenden unbedeutenden Aufträge an den vorhandenen Teilplan angehängt. Im Ungünstigsten Fall enthält die Menge B' $m-1$ Variablen, sodass m^{m-1} mögliche Verteilungspläne generiert und verglichen werden. Die ursprüngliche LPE-Heuristik sieht hierfür keinen separaten Methode wie z.B. einen Branch-and-Bound Algorithmus (vgl. Kapitel 3.3) vor, sondern testet alle möglichen Kombinationen. Da für die meisten Anwendungen die Anzahl n der Aufträge sehr groß, die Anzahl m der Machinen jedoch eher klein ist, bleibt der Aufwand, die verbliebenen $m-1$ Aufträge zu verteilen, in den meisten Fällen hinreichend klein.



5.1.1 Komplexität

Zur Bestimmung der Komplexität der Heuristik (vgl. [1, Seite 158]) werden die beiden Teile des Algorithmus separat betrachtet:

Im ersten Schritt wird ein lineares Programm gelöst. Dieses umfasst $mn+1$ Variablen und $m+n$ Nebenbedingungen. Während in der Praxis der Simplex-Algorithmus Anwendung findet (welcher allgemein nicht in polynomieller Zeit auf eine Lösung führt), lässt sich das Programm durchaus in polynomieller Zeit lösen. Ein möglicher Algorithmus zur Lösung linearer Programme in polynomieller Zeit ist die Ellipsoidmethode (vgl. Kapitel 6.3.2).

Nach Verteilung der Aufträge in die Mengen B und B' (ebenfalls möglich in linearer Zeit), existieren im dritten Schritt des Algorithmus maximal $m-1$ weitere, noch keiner Maschine zugeordnete Aufträge. Wie bereits erwähnt werden im ungünstigsten Fall m^{m-1} mögliche Verteilungspläne generiert und verglichen. Für eine festgelegte Anzahl an Maschinen m arbeitet also auch der dritte Schritt der Heuristik in polynomieller Zeit.

Allgmein besitzt die LPE-Heuristik somit eine Komplexität polynomiell in n. Die Heuristik arbeitet zwar allgemein exponentiell in m, jedoch ist in der Praxis nicht davon auszugehen, dass die Anzahl der Maschinen unbekannt ist, sodass dieser Fall außen vor gelassen werden kann.

Im Gegensatz zu den ECT-Heuristiken von *Ibarra* und *Kim* besitzt die LPE-Heuristik polynomielle Laufzeit, was als einer der Hauptvorteile der Heuristik gewertet werden kann.

5.1.2 Worst-Case Abschätzung

Im Folgenden wird der Wert $C_{max}^{LPE}/C_{max}^{*}$ abgeschätzt. (vgl. [1, Seite 158/159])

Behauptung: $C_{max}^{LPE}/C_{max}^{*} \leq 2$

Beweis:
Sei C_{max}^{LP} die Gesamt-Fertigstellungszeit der Aufträge, die durch die Lösung des linearen Problems festgelegt werden.
Und sei C_{max}^{E} die benötigte Zeit zur Fertigstellung der Restaufträge nach optimaler Verteilung im dritten Schritt des Algorithmus.

Zunächst gilt für beide Zeiträume:
$C_{max}^{*} \geq C_{max}^{LP}$ und
$C_{max}^{*} \geq C_{max}^{E}$.

Die durch den Algorithmus bestimmte Gesamt-Fertigstellungszeit aller Aufträge setzt sich aus den beiden definierten Teilen C_{max}^{LP} und C_{max}^{E} zusammen. Bei Verzicht eines kompletten Vergleichs aller Möglichkeiten im dritten Schritt der Heuristik weicht der zweite Summand jedoch vom oben definierten Optimum ab. Es kann gefolgert werden:
$C_{max}^{LPE} \geq C_{max}^{LP} + C_{max}^{E}$

Einsetzen liefert schließlich:

$C_{max}^{LPE} \geq C_{max}^{*} + C_{max}^{*} = 2C_{max}^{*} \Leftrightarrow C_{max}^{LPE}/C_{max}^{*} \leq 2$ □

Der Beweis orientiert sich lediglich an den Definitionen einzelner Teile der Gesamt-Fertigstellungszeit und vergleicht diese. Dennoch ist diese Abschätzung für mehr als zwei Maschinen bereits die best mögliche, was im Folgenden durch ein Beispiel veranschaulicht wird:

Beispiel:
Es seien m Aufträge und m Maschinen gegeben mit Bearbeitungsdauern
$p_{ij} = p_j/q_i$ $(i, j = 1, ..., m)$. Es soll gelten:

$q_1 = m - 1$,
$q_i = 1$ (für $i = 2, ..., m$)
$p_1 = (m - 1)p$,
$p_j = p$ (für $j = 2, ..., m$, $p > 0$ beliebig)

Offensichtlich ist eine optimale Verteilung der Aufträge gegeben, wenn Auftrag j durch Maschine j ausgeführt wird. Insbesondere gilt also: $C_{max}^{*} = p$.

Andererseits ist eine zulässige Lösung des linearen Problems gegeben durch:
$x_{11} = 0$,
$x_{1j} = 1$ (für $j = 2, ..., m$),
$x_{i1} = 1/(m - 1)$ (für $i = 2, ..., m$),
$x_{ij} = 0$ (für $i, j = 2, ..., m$)

Die Aufträge 2 bis m werden alle von Maschine 1 ausgeführt. In diesem Fall nimmt C_{max}^{LPE} maximalen Wert an. Es gilt: $C_{max}^{LPE} = 2p$

Schließlich folgt: $C_{max}^{LPE}/C_{max}^{*} = 2$

Aufgrund des obigen Beispiels entspricht die durch die allgemeine LPE-Heuristik berechnete Gesamt-Fertigstellungszeit einem mindestens doppelt so langen Zeitraum wie das definierte Minimum C_{max}^{*}.

5.2 LPE-Heuristik für $m = 2$

Das Problem, gegebene Aufträge auf nur zwei unabhängige Maschinen zu verteilen, wird separat betrachtet, da es in diesem Fall nicht nötig ist, einen Lösungsalgorithmus wie z.B. das Simplexverfahren zur Lösung des linearen Programms in Anspruch zu nehmen. Stattdessen lässt sich die nachstehende Methode von *Gonzalez* (vgl. [1, Seite 160] oder [9, Seite 219-224]) verwenden:

Schritt 1:
Sei S die Menge der Aufträge und $p_i(S)$ die benötigte Bearbeitungszeit dieser Aufträge, ausgeführt auf Maschine i. Im ersten Schritt der Methode werden die Bearbeitungsdauern aller Aufträge ins Verhältnis gesetzt. Es wird also p_{1j}/p_{2j} für alle $j = (1, ..., n)$ berechnet, wobei bei Gleichheit der Auftrag mit kleinerem Index bei einer späteren Sortierung als kleiner angesehen wird.

Anhand der Verhältnisse werden die Aufträge auf zwei Mengen aufgeteilt:
$S_{1k} = \left\{ j | p_{1j}/p_{2j} < p_{1k}/p_{2k} \right\}$ und
$S_{2k} = \left\{ j | p_{1j}/p_{2j} > p_{1k}/p_{2k} \right\}$

Es gilt, einen Auftrag k zu finden, sodass sich die von den Maschinen beanspruchten Fertigstellungszeiten der jeweiligen Aufträge annähern. Dabei soll die Differenz beider Zeiträume die Bearbeitungszeit des zuletzt verbliebenen Auftrags k, unabhängig davon welcher Maschine dieser zugeordnet wird, nicht übersteigen:
$p_1(S_{1k}) + p_{1k} \geq p_2(S_{2k})$ bzw. $p_1(S_{1k}) < p_2(S_{1k}) + p_{2k}$.

Zur Bestimmung des Auftrags k werden nach Berechnung aller p_{1j}/p_{2j} zwei Mengen S_{1l} und S_{2l} mit $|S_{1l}| = |S_{2l}|$ bzw. $|S_{1l}| = |S_{2l}| - 1$ durch Anwendung eines Median-Algorithmus (vgl. Kapitel 6.3.4) bestimmt.

Gilt $p_1(S_{1l}) + p_{1l} = p_2(S_{2l})$ oder $|S_{1l}| = 1$, so entspricht l dem Auftrag k.
Gilt $p_1(S_{1l}) + p_{1l} > p_2(S_{2l})$, so wird dieser Teil (beginnend mit der Bestimmung zweier Mengen durch einen Median-Algorithmus) der Heuristik für die Menge $S_{1l} \cup \{l\}$ erneut aufgerufen.
Gilt demgegenüber $p_1(S_{1l}) + p_{1l} < p_2(S_{2l})$, so erfolgt ein erneuter Aufruf für die Menge S_{2l}.
Der Vorgang terminiert nach spätestens $\log_n$ Iterationen.

Der Auftrag k wird dann als „trennender Auftrag“ bezeichnet. *Gonzalez* zeigt, dass eine Lösung des linearen Problems folgendermaßen gegeben ist:
$x_{1j} = 1$
$x_{2j} = 0 \ (j \in S_{1k})$
$x_{1j} = 0$
$x_{2j} = 1 \ (j \in S_{2k})$
$x_{1k} = (p_2(S_{2k}) + p_{2k} - p_1(S_{1k}))/(p_{1k} + p_{2k}) \in (0, 1)$
$x_{2k} = (p_1(S_{1k}) + p_{1k} - p_2(S_{2k}))/(p_{1k} + p_{2k}) \in (0, 1)$

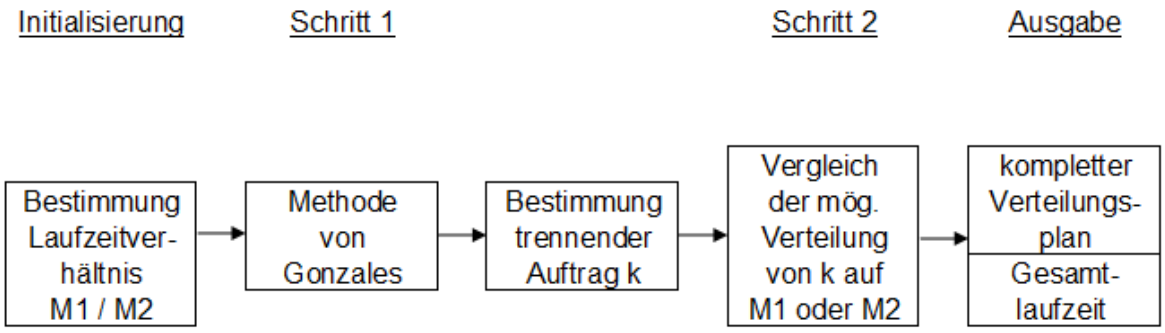
$$\Rightarrow C_{max} = p_1(S_{1k}) + p_2(S_{2k}) + x_{1k}p_{1k} + x_{2k}p_{2k}$$
$$= (p_{1k}p_2(S_{2k}) + p_{2k}p_1(S_{1k}) + p_{1k}p_{2k})/(p_{1k} + p_{2k})$$

Schritt 2:
Zur Entscheidung, welcher der beiden Maschinen der Auftag k zugeornet werden soll, lassen sich zunächst zwei mögliche Fäll beobachten:

Gilt $p_1(S_{1k})+p_{1k} = p_2(S_{2k})$, so existiert kein unbedeutender Auftrag. In diesem Fall wird $S_{1k} \cup \{k\}$ Maschine 1 und S_{2k} Maschine 2 zugeordnet.
Gilt andererseits $p_1(S_{1k}) + p_{1k} > p_2(S_{2k})$, so werden S_{1k} Maschine 1 und S_{2k} Maschine 2 zugeordnet. Der Auftrag k ist dann ein unbedeutender Auftrag. Die k ausführende Maschine wird ausgewählt, indem man beide Fälle betrachtet und die Laufzeiten vergleicht.

Als Gesamt-Fertigstellungszeit ergibt sich schließlich:

$$C_{max}^{LPE} = min\{p_1(S_{1k}) + p_{1k}, p_2(S_{2k}) + p_{2k}\}$$



5.2.1 Komplexität

Im Folgenden wird gezeigt, dass der Algorithmus lineare Laufzeit besitzt:

- Die Werte p_{1j}/p_{2j} können in linearer Zeit O(n) bestimmt werden.
- Laufzeiten $p_1(S_{1k})$ und $p_2(S_{2k})$ können in linearer Zeit bestimmt werden.
- Mengen S_{1l} und S_{2l} mit $|S_{1l}| = |S_{2l}|$ oder $|S_{1l}| = |S_{2l}| - 1$, sowie der Auftrag l, können in linearer Zeit gefunden werden. (Vorteil des Median-Algorithmus)
- Im Falle $p_1(S_{1l}) + p_{1l} = p_2(S_{2l})$ terminiert der Algorithmus.
  Im Falle $p_1(S_{1l}) + p_{1l} > p_2(S_{2l})$ bzw. $p_1(S_{1l}) + p_{1l} < p_2(S_{2l})$ erfolgt ein erneute Durchführung der letzten beiden Schritte, wobei sich die Mächtigkeit der betrachteten Menge halbiert ($n, n/2, n/4$ usw.). Nach O($logn$) Aufrufen ergibt sich schließlich der trennende Auftrag.
  Der trennende Auftrag k wird somit in linearer Zeit O(n) gefunden.
- Die Zuordnung des Auftrags k erfolgt durch Vergleiche möglicher Gesamt-Fertigstellungszeiten unabhängig von der Gesamtanzahl n der Aufträge.

Da alle Teilschritte des Algorithmus lineare Laufzeit besitzen, ergibt sich auch insgesamt eine solche. Im Folgenden werden Worst-Case Abschätzungen für diese Methode betrachtet.

5.2.2 Worst-Case Abschätzung

Die Abschätzung $C^{LPE}_{max}/C^{*}_{max} \leq 2$ (vgl. Kapitel 5.1.2) gilt für eine beliebige Anzahl Maschinen m. Im Falle $m = 2$ sind im Gegensatz zum allgemeinen Fall genauere, bessere Abschätzungen möglich. (vgl. [1, S.7])

Behauptung: $C^{LPE}_{max}/C^{*}_{max} \leq (1 + \sqrt{5})/2$ (für $m = 2$)

Beweis:
Sei C^{LP}_{max} die benötigte Zeit zur Fertigstellung der durch die Lösung des linearen Problems (unter Verwendung des oben beschriebenen Algorithmus für $m = 2$) verteilten Aufträge. Für die Abwicklung der Aufträge, eingeteilt in die beiden Teilmengen S_{1k} und S_{2k}, ergibt sich:
$p_1(S_{1k}) = C^{LP}_{max} - p_{1k}x_{1k}$ und
$p_2(S_{2k}) = C^{LP}_{max} - p_{2k}x_{2k}$.

Der trennende Auftrag k ist der einzige, der auf beide Maschinen aufgeteilt wird. Es gilt $x_{1k} + x_{2k} = 1$. Für die Gesamt-Fertigstellungszeit folgt somit in diesem Fall:

$$\begin{aligned} C^{LPE}_{max} &= min\left\{p_1(S_{1k}) + p_{1k}, p_2(S_{2k}) + p_{2k}\right\} \\ &= min\left\{C^{LP}_{max} - p_{1k}x_{1k} + p_{1k}, C^{LP}_{max} - p_{2k}x_{2k} + p_{2k}\right\} \\ &= C^{LP}_{max} + min\left\{p_{1k}(1 - x_{1k}), p_{2k}x_{1k}\right\} \end{aligned}$$

Außerdem gilt:
1.) Da $p_1(S_{1k}), p_2(S_{2k}) \geq 0$ gilt, kann die Bearbeitungszeit von Auftrag k auf Maschine 2 klarer-

weise C_{max}^{LP} nicht übersteigen:
$p_{2k}(1 - x_{1k}) \leq C_{max}^{LP}$

2.) Wir nehmen ObdA $p_{1k} \leq p_{2k}$ an (ansonsten Umnummerierung) und können so zwei weitere Aussagen in Bezug auf den Wert C_{max}^{*} treffen:
$p_{1k} \leq C_{max}^{*}$ und
$C_{max}^{LP} \leq C_{max}^{*}$ (siehe auch Kapitel 5.1.2)

Setzt man 1.) in obige Gleichung ein, so erhält man:
$C_{max}^{LPE} = C_{max}^{LP} + min\{p_{1k}(1 - x_{1k}), p_{2k}x_{1k}\}$
$\leq C_{max}^{LP} + min\left\{p_{1k}(1 - x_{1k}), x_{1k}C_{max}^{LP}/(1 - x_{1k})\right\}$

Einsetzen von 2.) liefert dann:
$C_{max}^{LPE}/C_{max}^{*} \leq 1 + min\{1 - x_{1k}, x_{1k}/(1 - x_{1k})\}$

Für $0 \leq x_{1k} \leq 1$ ist die Funktion $1 - x_{1k}$ monoton fallend, die Funktion $x_{1k}/(1 - x_{1k})$ demgegenüber monoton steigend. Wir folgern, dass der zu minimierende Ausdruck genau dann maximal wird, wenn gilt:
$1 - x_{1k} = x_{1k}/(1 - x_{1k}) \Leftrightarrow x_{1k}^2 - 3x + 1 = 0$

Nach Anwendung der PQ-Formel ergibt sich unter der Bedingung $x_{1k} \in (0, 1)$ als Lösung: $x_{1k} = (3 - \sqrt{5})/2$.
Somit gilt schließlich:

$C_{max}^{LPE}/C_{max}^{*} \leq 1 + min\{1 - x_{1k}, x_{1k}/(1 - x_{1k})\} \leq (1 + \sqrt{5})/2$ □

Auch diese Abschätzung ist im Falle $m = 2$ bereits die best mögliche, was erneut durch ein Beispiel verifiziert werden kann:

Beispiel:
Es seien 2 Aufträge mit nachstehenden Bearbeitungsdauern gegeben:
$p_{11} = (-1 + \sqrt{5})p/2$,
$p_{12} = p$,
$p_{21} = p$,
$p_{22} = (1 + \sqrt{5})p/2$
(p ≥ 0 beliebig)

Die Optimallösung ist gegeben, wenn Maschine 1 Auftrag 2 und Maschine 2 Auftrag 1 ausführt. Es gilt $C_{max}^{*} = p$.
Demgegenüber wählt die Heuristik $k = 2$ und somit $S_{1k} = \{1\}$ und $S_{2k} = \emptyset$.

Daraus folgt:
$C_{max}^{LPE} = (1 + \sqrt{5})p/2 \Leftrightarrow C_{max}^{LPE}/C_{max}^{*} = (1 + \sqrt{5})/2$.

6 Modifikationen der LPE-Heuristik

6.1 Die Heuristik LPH

Allgemein ist eine Alternative zur LPE-Heuristik gegeben, indem man im dritten Schritt von einem kompletten Vergleich aller Möglichkeiten absieht. Ziel einer hier angewendeten Heuristik H ist es stattdessen, durch insgesamt weniger Vergleiche trotzdem eine hinreichend gute Lösung zu gewährleisten. Analog zum normalen Fall würde gelten:
$C^{LPH}_{max}/C^{*}_{max} \leq 1 + C^{H}_{max}/C^{*}_{max}$ (vgl. Kapitel 5.1.2)

Trotz Einbußen in Bezug auf die berechnete Gesamt-Fertigstellungszeit arbeitet die gewählte Heuristik H im besten Fall polynomiell in n und m und genügt einer besseren Worst-Case Abschätzung.

Unglücklicherweise ergeben sich durch Anwendung diverser Heuristiken von *Ibarra* und *Kim* oder auch *Davis* und *Jaffe* als Heuristik H nicht die gewünschten großen Vorteile in Bezug auf obige Worst-Case Abschätzungen. Eine geeignete Heuristik H wird folglich weiterhin gesucht.

6.2 Die Heuristik LPE'

Die alternative Heuristik LPE' (vgl. [1, Seite 162/163]) bezieht sich auf den Fall $m = 2$. Ziel ist eine verbesserte Worst-Case Abschätzung (ohne große Laufzeiteinbußen).

Im ersten Schritt arbeitet die Heuristik analog zur normalen LPE-Heuristik. Erst nach einem kompletten LPE-Durchlauf wird ein modifiziertes Problem betrachtet, in dem der im ersten Durchlauf gefundenen Auftrag k durch eine zusätzliche Nebenbedingung derjenigen Maschine zugeteilt wird, auf der k eine kleinere Laufzeit besitzt:

Sei $p_{1k} \leq p_{2k}$ (ansonsten Umnummerierung). Wir setzen $p_{2k} = M$ mit M genügend groß, damit der Wert p_{1k}/p_{2k} kleiner ist als alle übrigen Werte $p_{1j} \leq p_{2j}$ für Aufträge j. Der Auftrag k wird somit Maschine 1 zugeordnet.

Nun wird die LPE-Heuristik mit neuer Zeit p_{2k} ein zweites Mal aufgerufen. Ein neuer trennender Auftrag k' mit zughörigen Mengen $S_{1k'}$ und $S_{2k'}$ wird gefunden. Schließlich wird die bessere der beiden Lösungen ausgewählt.

6.2.1 Komplexität

Obwohl zwei Aufrufe der LPE-Heuristik nötig sind und sich dadurch die Anzahl der Iterationen im Wesentlichen verdoppelt, ändert sich die Komplexität des Algorithmus nicht. Somit wird eine Lösung dieser modifizierten LPE'-Heuristik ebenfalls in linearer Zeit O(n) gefunden!

6.2.2 Worst-Case Abschätzung

Im Folgenden soll verifiziert werden, dass diese modifizierte Version in der Tat einer verbesserten Worst-Case Abschätzung genügt:

Behauptung: $C_{max}^{LPE'}/C_{max}^{*} \leq 3/2$

Beweis:
i) Es wird zunächst angenommen, dass der Auftrag k Maschine 2 zugeordnet wird.
Da die Lösung der Heuristik LPE' mindestens so gut ist wie die von der Heuristik LPE gefundene Lösung, gilt auch hier:
$C_{max}^{LPE'} \leq C_{max}^{LP} + min\{p_{1k}(1-x_{1k}), p_{2k}x_{1k}\}$ (vgl. Kapitel 5.2.2)

Nach Vorraussetzung wird Auftrag k Maschine 2 zugeordnet. Da desweiteren (nach eventueller Umnummerierung) $p_{1k} \leq p_{2k}$ gilt, folgt:
$C_{max}^{*} \geq p_{2k} \geq p_{1k}$

Mit $C_{max}^{*} \geq C_{max}^{LP}$ gilt schließlich:
$C_{max}^{LPE'}/C_{max}^{*} \leq C_{max}^{LP}/C_{max}^{*} + min\{1-x_{1k}, x_{1k}\} \leq 1 + min\{1-x_{1k}, x_{1k}\}$

Der zu minimierende Term nimmt sein Maximum in $x_{1k} = 1/2$ an. Im Falle einer Zuordnung von Auftrag k zu Maschine 2 gilt somit die Behauptung:

$C_{max}^{LPE'}/C_{max}^{*} \leq 3/2$

ii) Nun wird demgegenüber angenommen, dass Auftrag k bei optimaler Verteilung aller Aufträge Maschine 1 zugeornet wird.
Seien k' der trennende Auftrag und $C_{max}^{LP'}$ die Fertigungsstellungszeit, die während des zweiten LPE-Durchlaufs berechnet werden. Gilt $k' = k$, so wird beim zweiten LPE-Durchlauf Auftrag k bzw. k' Maschine 1 zugeordnet, während alle anderen Aufträge Maschine 2 zugeordnet werden. Da $p_{1k} \leq p_{2k}$ vorrausgesetzt wird, folgt in diesem Spezialfall, dass die Lösung der Heuristik mit Gesamt-Fertigstellungszeit p_{1k} optimal ist. Es gilt: $C_{max}^{LPE'}/C_{max}^{*} = 1$.
Die Behauptung ist in diesem Fall erfüllt.

Im Folgenden wird $k \neq k'$ angenommen. Anhand der Beschreibung der Heuristik LPE' folgt dann zunächst:
$C_{max}^{LPE'} \leq min\left\{C_{max}^{LP} + p_{1k}(1-x_{1k}), C_{max}^{LP} + p_{2k}x_{1k}, C_{max}^{LP'} + p_{1k'}(1-x_{1k'}), C_{max}^{LP'} + p_{2k'}x_{1k'}\right\}$

Desweiteren gilt bekanntlich $C^*_{max} \geq C^{LP}_{max}$. Da die Zuordnung von Auftrag k auf Maschine 1 die Fertigstellungszeit nicht beeinflusst, gilt außerdem $C^*_{max} \geq C^{LP'}_{max}$.

An dieser Stelle wird erneut eine Fallunterscheidung benötigt.
Werden beide Aufträge k und k' Maschine 1 zugeordnet, so gilt:
$C^*_{max} \geq p_{1k} + p_{1k'}$

Durch Einsetzen folgt:
$C^{LPE'}_{max} \leq min\left\{C^*_{max} + p_{1k}(1 - x_{1k}), C^*_{max} + p_{1k'}(1 - x_{1k'})\right\}$

Der zu minimierende rechte Term wird für $x_{1k} = x_{1k'} = 0$ maximal.
$\Rightarrow C^{LPE'}_{max} \leq min\left\{C^*_{max} + p_{1k}, 2C^*_{max} - p_{1k}\right\}$

Für $p_{1k} = (1/2)C^*_{max}$ wird schließlich dieser zu minimierende Term maximal. Es folgt die Behauptung:

$C^{LPE'}_{max} / C^*_{max} \leq 3/2$

Wird Auftrag k Maschine 1, Auftrag k' jedoch Maschine 2 zugeordnet, so ist Auftrag $k' \in S_{1k}$ aus der Menge S_{1k} gewählt worden. Da der zweite trennende Auftrag k' in jedem Fall aus der Menge $S_{1k} \cup \{k\}$ gewählt wird, folgt an dieser Stelle $p_{1k'}/p_{2k'} \leq p_{1k}/p_{2k}$ bzw. $p_{1k'} \leq p_{2k'}$.

Einsetzen liefert hier:
$C^{LPE'}_{max} \leq C^{LP'}_{max} + min\left\{p_{1k'}(1 - x_{1k'}), p_{2k'}x_{1k'}\right\}$

Mit $C^*_{max} \geq p_{2k'} \geq p_{1k'}$ und $C^*_{max} \geq C^{LP'}_{max}$ folgt:
$C^{LPE'}_{max} / C^*_{max} \leq 1 + min\left\{1 - x_{1k'}, x_{1k'}\right\}$

Der zu minimierende Term nimmt für $x_{1k'} = 1/2$ sein Maximum an, sodass auch in diesem Fall die Behauptung folgt.

Dass diese Abschätzung für die Heuristik LPE' ebenfalls die bereits best mögliche ist, soll erneut durch ein Beispielt verifiziert werden:

Beispiel:
Es werden 3 Aufträge betrachtet. Für die Bearbeitungsdauern soll gelten:
$p_{11} = p/2$,
$p_{12} = p$,
$p_{13} = p/2$,
$p_{21} = p$,
$p_{22} = p$,
$p_{23} = p/2$,
($p > 0$ beliebig)

Eine Optimallösung ist gegeben, wenn Maschine 1 die Aufträge 1 und 3 ausführt, während Maschine 2 Auftrag 2 ausführt. Es gilt: $C^*_{max} = p$

Die Heuristik LPE' setzt im ersten LPE-Durchlauf
$k = 2$, $S_{1k} = \{1\}$ und $S_{2k} = \{3\}$ mit $C^{LPE}_{max} = 3p/2$.

Nun wird $p_{22} \leq p/4$ gesetzt, sodass Auftrag 2 Maschine 1 zugeordnet wird Der zweite LPE-Durchlauf liefert dann:
$k' = 1$, $S_{1k'} = \{2\}$ und $S_{2k'} = \{3\}$.

Für beide Verteilungen von Auftrag 1 folgt an dieser Stelle ebenfalls eine Gesamtlaufzeit von $3p/2$.

Es gilt somit: $C^{LPE'}_{max}/C^*_{max} = 3/2$.

Eine bessere Abschätzung ist also in der Tat nicht möglich.

7 Algorithmische Darstellung

In diesem Kapitel werden mögliche Implementierungen der LPE-Heuristik für die Fälle $m \geq 3$ und $m = 2$ vorgestellt. Die Algorithmen lassen sich unabhängig von der zugrundeliegenden Programmiersprache anhand des beschriebenen Pseudocodes implementieren.

7.1 LPE-Heuristik für $m \geq 3$

Im Fall $m \geq 3$ verwendet die LPE-Heuristik einen Hilfsalgorithmus zur Lösung des linearen Programms. MATLAB beispielsweise verfügt über eine integrierte Version des Simplex-Verfahrens (inklusive Phase I), auf die zugegriffen werden kann. Die gegebenen Daten (also die Laufzeitmatrix) werden den benötigten Input-Variablen des Simplex-Verfahrens anpasst und die durch das Verfahren erhaltene Lösung anschließend in den nächsten Schritten des Algorithmus weiterverwendet.

Insbesondere zur Bearbeitung kleinerer Datenmengen ist es desweiteren nicht nötig, z.B. den Branch-and-Bound Algorithmus innerhalb des dritten Schrittes der LPE-Heuristik zu initialisieren.

Die nachfolgende Darstellung des Algorithmus vergleicht dementsprechend alle möglichen Verteilungspläne für die unbedeutenden Aufträge. Hierzu werden im Teil (3) Zählschleifen entsprechend der Mächtigkeit $|B'|$ der Menge B' initialisiert. Im Maximum ergeben sich $m - 1$ Zählschleifen mit jeweils m Schleifendurchläufen. Dieser Fall entspricht den maximalen m^{m-1} Vergleichsoperationen.

Algorithm 1 LPE-Heuristik für $m \geq 3$

Input: p_{ij} für alle $i = (1, ..., m), j = (1, ..., n)$
Output: Verteilungsmatrix X mit Einträgen x_{ij}, Laufzeit C_{max}^{LPE}

(1) **Lösung des LPs:**
(1a) Bestimmung von $A \in \mathbb{R}^{m \times n}, c \in \mathbb{R}^n$ und $b \in \mathbb{R}^m$
für benötigte Form: $max \left\{ c^T x | Ax \leq b, x \geq 0 \right\}$
(1b) Initialisierung Phase I Simplex-Verfahren
(1c) Initialisierung Grundform Simplex-Verfahren
$\Rightarrow$ Optimallösung / Basislösung x^*

(2) **Zwischenschritte:**
(2a) Vereinige n Basisvariablen x_{ij} zur Menge B:
Wiederhole für $j = 1$ bis n:
Setze $B = \emptyset$,
$i = 1$
Gilt $x_{ij} = 1$, dann: $B = B \cup \left\{ x_{ij} \right\}$
Gilt $x_{ij} \neq 1$, dann setze $m = m + 1$
(2b) Vereinige $m - 1$ Restvariablen x_{ij} zur Menge B':
Setze $B' = \emptyset$
Wiederhole für $i = 1$ bis m:
Wiederhole für $j = 1$ bis n:
Gilt $0 < x_{ij} < 1$ und $x_{ij} \notin B$, dann setze $B' = B' \cup \left\{ x_{ij} \right\}$

(3) **Verteilung der Aufträge** $x_{ij} \in B'$**:**
(3a) Neusortierung der Aufträge in Matrix K mit Einträgen x_{ik}, $k = (1, ..., |B'|)$
Setze $x_{1k} = 1$ für alle $k = (1, ..., |B'|)$
$x_{ik} = 0$ für alle $i = (2, ..., m), k = (1, ..., |B'|)$
$C_{max}^{LPE} = \infty$
$k = 0$
(3b) Rekursiver Aufruf von $|B'|$ Zählschleifen:
Wiederhole für $i_1 = 1$ bis m:
$x_{i_1 k_1} = 1$
Wiederhole für $i_2 = 1$ bis m:
$x_{i_2 k_2} = 1$
...
Wiederhole für $i_k = 1$ bis m:
Setze $x_{i_k k_{|B'|}} = 1$
Rückübertragung der x_{ik} in x_{ij}
Berechne Gesamtlaufzeit $C = max \left\{ \sum_{j=1}^{n} p_{ij} x_{ij} | i = (1, ..., m) \right\}$
Gilt $C < C_{max}^{LPE}$, dann: $C_{max}^{LPE} = C$,
Verteilungsmatrix $X = X_{ij}$ (akt. Konfiguration)
$x_{i_k k_{|B'|}} = 0$
...
$x_{i_2 k_2} = 0$
$x_{i_1 k_2} = 0$

7.2 LPE-Heuristik für $m = 2$

Wie bereits erwähnt benötigt die LPE-Heuristik im Fall $m = 2$ keine Hilfsalgorithmen zur Lösung des linearen Programms. Demgegenüber wird ein Median-Algorithmus zur Bestimmung des Elements mittlerer Größe verwendet.

Eine mögliche Implementierung der Methode von Gonzales und der anschließenden Zuordnung des trennenden Auftrags k wird im Folgenden dargestellt.

Algorithm 2 LPE-Heuristik für $m = 2$

Input: p_{1j} und p_{2j} für alle $j = (1, ..., n)$
Output: Menge S_{1k} S_{2k}, Laufzeit C_{max}^{LPE}

(1) **Bestimmung des Auftrags k:**
- (1a) Berechne p_{1j}/p_{2j} für alle $j = (1, ..., n)$
- (1b) Bestimmung von Mengen S_{1l} und S_{2l} durch Median-Algorithmus
- (1c) Berechne für l zwei neue Mengen nS_{1l} und nS_{2l} für alle Aufträge
  (nur interessant nach erneutem Aufruf von Schritt (1b))
- (1d) Berechne $p_1(nS_{1l})$
  $p_2(nS_{2l})$
  Gilt $p_1(nS_{1l}) + p_{1l} = p_2(nS_{2l})$ oder gilt $|S_{1l}| = 1$,
  dann: $l = k$
  Gilt $p_1(nS_{1l}) + p_{1l} > p_2(nS_{2l})$,
  dann: Wiederhole ab Schritt (1b) für Menge $S_{1l} \cup \{l\}$
  Gilt $p_1(nS_{1l}) + p_{1l} < p_2(nS_{2l})$,
  dann: Wiederhole ab Schritt (1b) für Menge S_{2l}
- (1e) Bestimme für k Mengen S_{1k} und S_{2k}
  Berechne $p_1(S_{1k})$
  $p_2(S_{2k})$

(2) **Zuordnung des Auftrags k:**
- (2a) Gilt $p_1(S_{1k}) + p_{1k} = p_2(S_{2k})$,
  dann: $S_{1k} = S_{1k} \cup \{k\}$
- (2b) Gilt $p_1(S_{1k}) + p_{1k} > p_2(S_{2k})$ und $p_1(S_{1k}) + p_{1k} < p_2(S_{2k}) + p_{2k}$,
  dann: $S_{1k} = S_{1k} \cup \{k\}$
- (2c) Gilt $p_1(S_{1k}) + p_{1k} > p_2(S_{2k})$ und $p_1(S_{1k}) + p_{1k} > p_2(S_{2k}) + p_{2k}$,
  dann: $S_{2k} = S_{2k} \cup \{k\}$

Laufzeit:
$C_{max}^{LPE} = min\{p_1(S_{1k}), p_2(S_{2k})\}$

8 Implementierung in MATLAB

MATLAB ist eine kommerzielle, plattformunabhängige Software des Unternehmens TheMathWorks, die insbesondere zu numerischen Berechnungen mit Hilfe von Matrizen herangezogen wird.

Im weiteren Verlauf werden die Besonderheiten der Programmierung der LPE-Heuristiken im Falle $m \geq 3$ und $m = 2$ unter MATLAB angesprochen, sowie beispielhaft einige Ergebnisse beider Varianten präsentiert.

8.1 LPE-Heuristik für $m \geq 3$

Im Falle $m \geq 3$ wird der in MATLAB integrierte Simplex-Algorithmus zur Lösung des linearen Programms verwendet. Eine mögliche Programmierung unter Aufrufung des Befehls „linprog“ zur Lösung des linearen Problems ist denkbar. Um den ersten Schritt der Heuristik implementieren zu können, müssen dementsprechend die für die Ausführung von „linprog“ benötigten Parameter definiert werden.

Die Funktion „min $f(x) = g^T x$“ wird optimiert. Deshalb muss zunächst die Verteilungsmatrix X als Spaltenvektor mit mn Einträgen definiert und die gesuchte Gesamtlaufzeit als Eintrag x_{mn+1} angehängt werden. Desweiteren müssen zur Übertragung des linearen Problems sowohl der Zielfunktionsvektor $g^T = (0, ..., 0, 1) \in \mathbb{R}^{mn+1}$ gesetzt, als auch die Nebenbedingungen geeignet eingelesen und in Matrixform neu definiert werden.

Die übrigen Schritte der Heuristik lassen sich anhand des beschriebenen Codes übertragen. Es ist jedoch zu beachten, dass MATLAB keine Mengenoperationen durchführen kann, sodass die Zwischenschritte in Vektor- bzw. Matrizenform durchgeführt und abgespeichert werden müssen.

Im Folgenden werden jeweils die Ergebnisse zweier Beispieldurchläufe dargestellt.

8.1.1 Beispiel 1

Input:
Verteilungsmatrix P (Einträge $\triangleq$ benötigte Zeiteinheiten):

$$\begin{pmatrix} 3 & 2 & 4 & 7 & 4 & 2 & 9 \\ 5 & 3 & 4 & 2 & 3 & 11 & 5 \\ 1 & 9 & 7 & 2 & 7 & 10 & 4 \end{pmatrix}$$

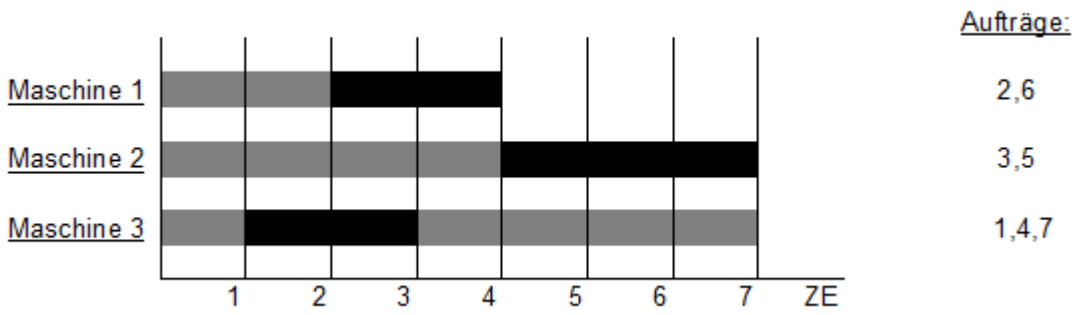
Output:
Lösungsmatrix nach Schritt 1:

$$\begin{pmatrix} 0 & 1 & 0.5 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0.5 & 0.5 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0.5 & 0 & 0 & 1 \end{pmatrix}$$

Verteilungsmatrix X nach Schritt 3:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

graphische Darstellung:



Laufzeit:
7 Zeiteinheiten

8.1.2 Beispiel 2

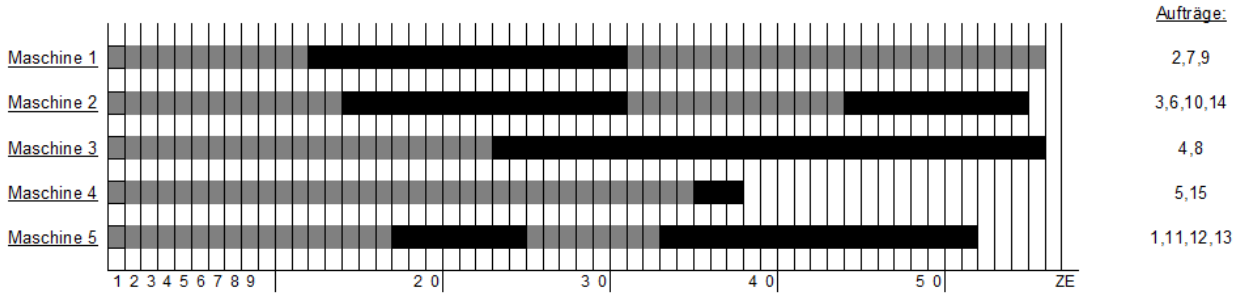
Input:
Verteilungsmatrix P:

$$\begin{pmatrix} 53 & 12 & 24 & 37 & 43 & 26 & 19 & 33 & 25 & 69 & 15 & 93 & 95 & 24 & 89 \\ 58 & 32 & 14 & 28 & 63 & 18 & 75 & 25 & 57 & 12 & 80 & 57 & 83 & 11 & 34 \\ 45 & 25 & 74 & 23 & 17 & 40 & 44 & 33 & 23 & 93 & 11 & 50 & 83 & 79 & 22 \\ 64 & 35 & 41 & 75 & 35 & 65 & 90 & 96 & 63 & 27 & 45 & 40 & 77 & 28 & 3 \\ 17 & 74 & 51 & 64 & 55 & 65 & 68 & 91 & 66 & 15 & 8 & 8 & 19 & 49 & 64 \end{pmatrix}$$

Output:
Verteilungsmatrix X nach Schritt 3:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$$

graphische Darstellung:



Laufzeit:
56 Zeiteinheiten

8.2 LPE-Heuristik für $m = 2$

Im Falle $m = 2$ lässt sich der im vorherigen Abschnitt beschriebene Algorithmus nahezu komplett übertragen. Lediglich die Implementierung der vier Schritte des Median-Algorithmus (idealerweise in einer separaten Prozedur), sowie die Darstellung der Mengen als Vektoren bzw. Matrizen, müssen auch in diesem Fall realisiert werden.

Im Folgenden werden die Ergebnisse eines beispielhaften Durchlaufs präsentiert.

8.2.1 Beispiel 3

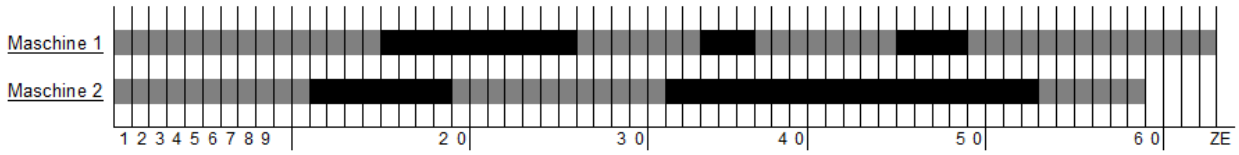
Input:
Verteilungsmatrix P:

$$\begin{pmatrix} 23 & 15 & 19 & 27 & 11 & 7 & 3 & 30 & 9 & 4 & 13 & 14 & 23 \\ 11 & 16 & 8 & 12 & 15 & 20 & 7 & 22 & 18 & 25 & 6 & 16 & 10 \end{pmatrix}$$

Output:
Verteilungsmengen:

$S_{1k} = \{2, 5, 6, 7, 9, 10, 12\}$
$S_{2k} = \{1, 3, 4, 8, 11, 13\}$

graphische Darstellung:



Laufzeit:
63 Zeiteinheiten

9 Zusammenfassung

Die verschiedene Ergebnisse der vorangegangenen Analyse werden im Folgenden erneut kurz zusammengestellt:

Die LPE-Heuristik für $m = 3$ besitzt eine Laufzeit polynomiell in n (und exponentiell in m). Für den Wort-Case Quotienten gilt in diesem Fall $\rho = 2$. Diese Erkenntnis stellt ein deutlich besseres Ergebnis dar als z.B. die Heuristiken von *Ibarra* und *Kim*, oder *Davis* und *Jaffe*. Während beide anderen Heuristiken einen Worst-Case Quotienten besitzen, der sich als Funktion von m ausdrückt, besitzt die LPE-Heuristik eine Güte unabhängig von n und m.

Im Fall $m = 2$ ergibt sich für die LPE-Heuristik mit $\rho = 2$ der gleiche Worst-Case Quotient wie für die Heuristik von *Ibarra* und *Kim*. Der Vorteil ist an dieser Stelle aufgrund der niedrigeren Laufzeit gegeben. Die Heuristik von *Ibarra* und *Kim* benötigt nach den Werten p_{1j}/p_{2j} geordnete Aufträge im ersten Schritt. Dies entspricht einer kompletten Sortierung des Datensatzes, was zur Folge hat, dass die Komplexität im Bereich $O(n \log n)$ liegt. Die LPE-Heuristik unter Anwendung des Median-Algorithmus ist mit Laufzeit $O(n)$ folglich auch im Falle $m = 2$ attraktiver für eventuelle Anwendungen!

Die LPE-Heuristik erzielt also für das gegebene Problem sehr gute Ergebnisse. Demgegenüber sei aber erwähnt, dass sich in der vorgestellten Form keine weiteren Einschränkungen an die gegebenen Aufträge (wie z.B. Fristen) integrieren lassen. Nach Erweiterung des linearen Programms durch neue Nebenbedingungen beispielsweise würde eine gefundene Basislösung nicht mehr die benötigte Form besitzen, sodass nicht genügend Aufträge dem im ersten Schritt generierten Teilplan zugeordnet werden könnten.

Varianten der LPE-Heuristik, die unter anderem Fristen für die gegebenen Aufträge berücksichtigen, werden von *C.N. Potts* seperat betrachtet. (vgl. z.B. [10]) Auf eine Beschreibung dieser Heuristiken wird im Rahmen dieser Arbeit verzichtet.

Literaturverzeichnis

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

[2] E. Horowitz; S. Sahni. Exact and approximation algorithm for non-identical processors. *Journal of Association for Computing Machinery*, 23:317–327, 1976.

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

[4] E. David; J.M. Jaffe. Algorithms for scheduling tasks on unrelated processors. *Journal of the Association for Computing Machinery*, 28:721–736, 1981.

[5] A. Martin; S. Ulbricht; M. Dür. Skript zur Vorlesung WS2009/2010.

[6] H.J. Zimmermann. *Operations Reseach: Methoden und Modelle*. Vieweg, 2008.

[7] R.H. Güting; S. Dieker. *Datenstrukturen und Algorithmen*. Teubner, 2004.

[8] H.W. Lang. *Algorithmen in Java*. Oldenbourg Wissenschaftsverlag, 2002.

[9] T. Gonzales; E.L. Lawler; S. Sahni. Optimal preemptive scheduling of two unrelated parallel processors. *Orsa journal on computing*, 2:219–224, 1990.

[10] C.N. Potts. Analysis of a heuristic for one maschine sequencing with release dates and delivery times. *Oper. Res.*, 28:1436–1441, 1980.