Christian Lennerz

Die Algorithmen von Bartal und Karger zur Lösung des Graham-Problems

Seminararbeit im Vertiefungsfach
Informations- und Technologiemanagement
an der Wirtschaftswissenschaftlichen Fakultät
der Universität des Saarlandes

Saarbrücken 1999

ANFORDERUNGSDEFINITION

In dieser Arbeit sollen die Arbeiten von BARTAL ET AL. [BFKV92] und von KARGER ET AL. [KPT96] untersucht werden. Neben der Darstellung der Funktionsweise der Algorithmen, sollen auch Anwendungen aus dem betriebswirtschaftlichen Umfeld gefunden werden.

BARTAL ET AL. stellen in ihrem Paper einen Algorithmus vor, der $(2 - 1/70) = 1,986$-kompetitiv ist, also besser als der von GRAHAM für alle $m > 70$. In dem Paper von KARGER ET AL. wird dieser Algorithmus verallgemeinert und ein kompetitiver Faktor von $1,945$ bewiesen.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Grundlagen der Betrachtung von Schedulingproblemen und -algorithmen

1.1. Scheduling – Begriffsdefinition und Geschichte des Forschungsgebietes

Scheduling oder *Ablaufplanung* ist ein Forschungsgebiet, das sich mit der *zeitlichen Zuteilung knapper oder kostbarer Ressourcen* zu einer Menge *durchzuführender Aufgaben* beschäftigt. Es handelt sich um einen Entscheidungsprozeß, dessen Ziel die *Optimierung* von mindestens einer vorgegebenen Zielgröße ist [Pin95]. Statt Ressourcen werden wir häufig die Begriffe *Prozessoren* und *Maschinen* verwenden, denen Aufgaben oder *Aufträge* zugewiesen werden müssen. Ressourcen und Aufträge können in realen Schedulingproblemen verschiedenste Ausprägungen annehmen. Es kann sich dabei um Fertigungsaufträge in der betrieblichen Produktion handeln, die von Maschinen abzuarbeiten sind, oder redundant konfigurierte Festplattensysteme, auf denen Leseoperationen durchgeführt werden müssen. Ähnlich vielfältig wie diese Zuordnungsbeziehungen können auch die dabei verfolgten *Zielsetzungen* sein. Während man in der Produktion beispielsweise die Verspätungen der Fertigungsaufträge gegenüber den vorgegebenen Deadlines minimieren möchte, besteht bei Festplattensystemen die Herausforderung, die Leselast möglichst gleichmäßig auf die Massenmedien zu verteilen.

Da die Ablaufplanung eine Form der Entscheidungsfindung ist, welche sowohl in der Industrie als auch im Dienstleistungssektor eine entscheidende Rolle spielt, entstanden auch hier die frühesten Arbeiten auf dem Gebiet. Pioniere wie HENRY GANTT waren die ersten, die sich zu Beginn dieses Jahrhunderts mit Schedulingproblemen in der Fertigung beschäftigten. Es dauerte jedoch beinahe ein halbes Jahrhundert, bis in den frühen 50er Jahren die ersten Veröffentlichungen zum Themenkreis Ablaufplanung in der Operations-Research-Literatur erschienen. In den 60er Jahren lag der Forschungsschwerpunkt auf der Beschreibung von Ablaufplanungsproblemen mittels Ganzzahliger oder Dynamischer Programmierung.

Die Ergebnisse der Komplexitätstheorie Ende der 60er und in den 70er Jahren lieferten ein tiefgreifenderes Verständnis der Probleme. Die damit verbundene Ein-

sicht, daß für viele dieser Probleme vermutlich keine deterministischen und exakten Lösungsverfahren mit akzetablen Laufzeiten existieren, lenkte die Schedulingtheorie in eine neue Richtung. Für solche „schweren" Probleme versuchte man, Algorithmen zu finden, die zwar keine optimale Lösung, jedoch eine gewisse Mindestgüte garantieren können.

In den 80er Jahren war es dann die Wahrscheinlicheitstheorie, die großen Einfluß auf das Forschungsgebiet nahm, indem sie die Formulierung stochastischer Schedulingprobleme und deren Analyse erlaubte.

In den 90er Jahren gewannen immer mehr praktisch motivierte Fragestellungen an Bedeutung. Zum einen interessierte man sich zunehmend für *Online*-Probleme, deren Parameter nicht a priori vollständig bekannt sind und zum anderen waren es konkrete Anwendungsgebiete, wie beispielsweise Schedulingprobleme in Netzwerken, die das Forschungsinteresse weckten.

1.2. Klassifikation von Schedulingproblemen

Wir wollen uns in dieser Arbeit mit Lösungsverfahren für ein konkretes, deterministisches Schedulingproblem beschäftigen. Wie wir bereits gesehen haben, war es das Gebiet der deterministischen Ablaufplanung, dem über Jahrzehnte hinweg das Forschungsinteresse galt. Es verwundert daher nicht, daß dabei eine Vielfalt von Schedulingproblemen untersucht wurden, so daß ein Klassifikationsschema zur systematischen Einordnung der einzelnen Problemstellungen erforderlich war.

1.2.1. Die $\alpha|\beta|\gamma$-Notation

Wir wollen im folgenden das Schema von GRAHAM ET AL. [GLRK79] und BLAZEWICZ ET AL. [BLRK83] vorstellen, welches als $\alpha|\beta|\gamma$-Notation bekannt ist.

α: Das erste Feld $\alpha = \alpha_1\alpha_2\alpha_3$ charakterisiert die Prozessoren.

1. $\alpha_1 \in \{\emptyset, P, Q, R, F, J, O\}$ beschreibt den Prozessortyp. P, Q und R stehen für parallele Prozessoren mit identischen (P), proportionalen (Q) bzw. beliebigen Geschwindigkeiten.
   F, J und O beschreiben dagegen spezialisierte Prozessoren, die als Flow Shop (F), Jop Shop (J) oder Open Shop (O) organisiert sind.
2. $\alpha_2 \in \{\emptyset, k\}$ gibt die Zahl k der verfügbaren Prozessoren an, wobei $\emptyset$ für eine beliebige Anzahl steht.
3. $\alpha_3 \in \{\emptyset, NC\}$ trifft eine Aussage darüber, ob die Prozessoren kontinuierlich verfügbar sind ($\emptyset$) oder nicht (NC).

β: Das Feld $\beta_1\beta_2\cdots\beta_6$ charakterisiert die Jobs, welche den Prozessoren zuzuordnen sind.

1. $\beta_1 \in \{\emptyset, \mathtt{pmtn}\}$ gibt an, ob die Jobs nach Beginn ihrer Bearbeitung beliebig unterbrochen werden dürfen ($\mathtt{pmtn}$) oder nicht ($\emptyset$).
2. $\beta_2 \in \{\emptyset, \mathtt{res}\}$ trifft eine Aussage über die Notwendigkeit zusätzlicher Ressourcen.
3. $\beta_3 \in \{\emptyset, \mathtt{prec}, \mathtt{tree}, \mathtt{chain}\}$ beschreibt die Vorrangsbeziehungen, welche zwischen den Jobs gelten. Sie können als Kette bzw. Baum repräsentiert oder beliebig gestaltet sein ($\mathtt{prec}$).
4. $\beta_4 \in \{\emptyset, r_j\}$ gibt an, ob die Bearbeitung der einzelnen Jobs zu einem beliebigen Zeitpunkt beginnen kann ($\emptyset$) oder ob dabei feste Termine eingehalten werden müssen (r_j).
5. $\beta_5 \in \{\emptyset, p_j = p, \underline{p} \leq p_j \leq \bar{p}\}$ charakterisiert die Jobbearbeitungszeiten. Sie können beliebig ($\emptyset$), konstant ($p_j = p$) oder nach oben und unten beschränkt ($\underline{p} \leq p_j \leq \bar{p}$) sein.
6. $\beta_6 \in \{\emptyset, \tilde{d}\}$ gibt an, ob Deadlines beachtet werden müssen $\left(\tilde{d}\right)$ oder nicht ($\emptyset$).

γ: Das letzte Feld $\gamma \in \{-, C_{max}, \sum C_j, \sum w_j C_j, L_{max}, \sum U_j, \sum CO\}$ beschreibt die verfolgte Zielgröße. Im Fall $\gamma = -$ sucht man nur einen zulässigen Plan, während in den übrigen Fällen ein Optimierungsziel vorliegt. Entweder ist die Planlänge (C_{max}), die ungewichtete bzw. gewichtete Summe der Fertigstellungstermine ($\sum C_j, \sum w_j C_j$), die maximale auftretende Terminabweichung (L_{max}), oder die Anzahl der verspäteten Aufträge ($\sum U_j$) bzw. Umrüstvorgänge ($\sum CO$) zu minimieren. Diese Aufzählung ist bereits um äquivalente Zielsetzungen bereinigt. In [BEP+96] und [Sch97] findet man jedoch weitere denkbare Optimierungsziele.

Das folgende Beispiel soll die Notation verdeutlichen.

Beispiel 1.1. Die Signatur $F|p_j = p| \sum w_j C_j$ beschreibt eine Flow Shop Umgebung mit einer beliebigen Zahl an Prozessoren, auf denen alle Jobs eine konstante Bearbeitungszeit p besitzen. Die Aufgabe besteht darin, eine Reihenfolge zu finden, in der die Jobs das System so durchlaufen, daß die Summe der gewichteten Fertigstellungstermine minimiert wird. In der Realität entspricht das Szenario einer Fließbandfertigung, in der alle Jobs die gleiche Bearbeitungszeit benötigen, bestimmte Aufträge jedoch höhere Priorität genießen als andere.

Neben dieser grundsätzlichen Problemcharakterisierung differenziert man weiter nach dem Grad der Informationsverfügbarkeit über die Probleminstanz und spricht von der sogenanten *Online*- bzw. *Offline*-Variante des Problems.

1.2.2. Online- und Offline-Problemstellungen

In der Realität kann man nur selten davon ausgehen, daß alle Informationen über die zu lösende Probleminstanz a priori bekannt sind. Stattdessen kommt es in der Praxis

nicht selten vor, daß die Problemdaten erst nach und nach im Laufe der Zeit eintreffen und Entscheidungen auf der Basis des bisher verfügbaren Wissens getroffen werden müssen. Gerade im Bereich der Ablaufplanung ist dieses Szenario sehr real. So weiß ein Workflowmanagementsystem beispielsweise nicht, wann ein Kundenauftrag eintreffen wird, den es zur Bearbeitung den Vertriebsmitarbeitern zuteilen muß. So können Situationen entstehen, in denen ein Mitarbeiter mit einem Auftrag ausgelastet wird, obwohl kurze Zeit später ein bedeutender Kundenauftrag eintrifft, welcher nur von ihm als Key-Account-Manager bearbeitet werden darf.

Ein solches Szenario, in dem die Probleminstanz erst im Laufe der Zeit vollständig bekannt wird, bezeichnet man als *Online*-Problem und grenzt es somit gegen die klassische Problemstellung der Algorithmentheorie ab, in der alle Problemparameter von vornherein als bekannt vorausgesetzt sind (*offline*). Online-Algorithmen stehen somit vor der Herausforderung, eine Eingabe, die sie lediglich stückweise lernen, ohne Kenntnis der Zukunft in eine Ausgabe zu transformieren, die für jeden zukünftigen Zeitpunkt eine möglichst optimale Problemlösung darstellt.

Es ist offensichtlich, daß die Lösung der Online-Problemstellung „schwieriger" ist, als die ihres Offline-Pendants. Schließlich kann man nicht erhoffen mit weniger Information bessere Ergebnisse zu erhalten. Daher können Komplexitätsresultate, wie sie schon seit langem für die meisten Offline-Probleme bekannt sind, dahingehend übertragen werden, daß die Online-Variante auf der gleichen Jobsequenz keine besseren asymptotischen Laufzeiten erzielen kann.

Der Schwierigkeitsgrad von Online-Problemen hängt selbstverständlich davon ab, welche Informationen a priori bekannt sind und wie viele Daten erst im Laufe der Zeit eintreffen. In den von uns betrachteten Schedulingproblemen sind es die Jobs, die nach und nach eintreffen und online verarbeitet werden müssen. Die Ankunftszeiten und/oder Bearbeitungszeiten späterer Aufträge können bei der Einplanung eines Jobs nicht berücksichtigt werden, weil sie frühestens beim Eintreffen der zukünftigen Jobs bekannt werden.

1.3. Analyse von Schedulingproblemen und -algorithmen

Deterministische Schedulingprobleme sind nichts anderes als kombinatorische Optimierungsprobleme. Daher können die Werkzeuge aus der Komplexitäts- und Optimierungstheorie, welche für diese große Klasse von Problemen entwickelt wurden, auch hier sinnvoll angewendet werden. Betrachtet man jedoch Ablaufplanungsprobleme vor einem realen Hintergrund, so muß man sich bewußt sein, daß die Zeit zur Lösung der Probleme meist sehr begrenzt ist. Aussagen über asymptotische Laufzeiten, wie sie beispielsweise durch die O-Notation gemacht werden, müssen durch empirische Laufzeitmessungen oder Aussagen über erwartete Laufzeiten bei gegebenen Wahrscheinlichkeitsverteilungen gestützt bzw. relativiert werden.

In [BEP+96] wird der folgende Ansatz zur Analyse von Schedulingproblemen so-

wie zur Entwicklung und Bewertung geeigneter Algorithmen empfohlen. Die Komplexitätsanalyse ist ein guter Ausgangspunkt für einen Einblick in den Schwierigkeitsgrad des vorliegenden Problems. Falls die Zeit zur Lösung des Problems durch ein Polynom nach oben beschränkt werden kann, dann bestehen gute Aussichten, einen praktisch verwendbaren Polynomialzeitalgorithmus konstruieren zu können, der das Problem exakt löst.

Unglücklicherweise haben viele – oft sehr einfach zu formuliende – Schedulingprobleme diese Eigenschaft nicht. Man kann für die meisten dieser Probleme nachweisen, daß sie NP-hart sind [GJ79], so daß die Entwicklung eines deterministischen Polynomialzeitalgorithmus' vermutlich hoffnungslos ist. Es erfordert daher weitere Anstrengungen, um ein praktisch verwendbares Verfahren zur Lösung des Problems zu finden.

Zunächst können wir das Problem *relaxieren*. Dies bedeutet, daß wir bestimmte Nebenbedingungen, die das Problem erschweren, in der Erwartung ignorieren, daß die so gefundene Lösung eine gute Annäherung an die eigentliche Lösung darstellt. Wir werden im nächsten Kapitel 2 ein Beispiel kennenlernen, in der die Problemrelaxation den Einsatz von Polynomialzeitalgorithmen erlaubt.

Eine andere Möglichkeit, das Komplexitätsproblem praktisch anzugehen, besteht in dem Einsatz von *exakten enumerativen Verfahren*, welche den gesamten Lösungsraum geschickt aufzählen. Ihre Laufzeit ist im worst-case exponentiell in der Eingabegröße, doch in der Praxis erlauben sie meist, realistische Problemgrößen zu lösen.

Im Schedulingbereich ist schließlich die folgende Vorgehensweise weit verbreitet. Dabei ist man im Gegensatz zur Relaxation bemüht, das Problem optimal zu lösen, kann jedoch nicht garantieren, daß in allen Fällen eine optimale Lösung auch gefunden wird. Durch diese Einschränkung erkaufen sich die *Approximationsalgorithmen* eine polynomielle Laufzeit in der Eingabegröße. Da die Optimalität der Lösung nicht gesichert ist, versucht man selbstverständlich, eine Aussage darüber treffen zu können, wie „weit" die ermittelte Lösung von dem optimalen Zielfunktionswert abweicht. Da dieser relative Fehler im allgemeinen von Instanz zu Instanz schwanken wird, bietet es sich an, den average- oder worst-case zu betrachten [BEP$^+$96]. Average-case-Analysen sind sehr selten, da man realistische Verteilungsfunktionen benötigt, die die Analyse erschweren. Man konzentriert sich daher im allgemeinen auf den worst-case. Doch leider müssen auch diese Bemühungen nicht von Erfolg gekrönt sein. Für einige Probleme kann man zeigen, daß das Finden eines Approximationsalgorithmus' selbst ein NP-hartes Problem darstellt. Oft läßt sich jedoch ein solches Verfahren finden und seine Leistungsgüte durch den sogenannten *Approximationsfaktor* angeben. Dieser beschreibt das Verhältnis der Approximationslösung zum optimalen Zielfunktionswert. Ein Algorithmus heißt c-*approximativ*, wenn seine Lösung höchstens c mal schlechter als der optimale Wert ist.

Übertragen wir dieses Konzept auf die Online-Fragestellung, dann müssen wir die Lösung eines Algorithmus', dem die Probleminstanz erst im Laufe der Zeit bekannt wird, mit einem Wert vergleichen, den man nur ermitteln kann, wenn man über vollkommene Information verfügt. Einen solchen „unfairen" Gegner findet man in dem optimalen Algorithmus, der die gesamte Zukunft kennt. Dieser erzeugt auf der

Basis der ihm zur Verfügung stehenden Daten stets eine optimale Lösung. In Analogie können wir unseren Online-Algorithmus c-*kompetitiv* nennen, wenn er einen *kompetitiven Faktor* von c erzielt, d.h. wenn die von ihm generierte Lösung höchstens c mal schlechter als die des optimalen Algorithmus' ist.

2. Das Graham-Problem

Im Jahre 1966 warf GRAHAM [Gra66] ein sehr einfaches und gut zu verstehendes Schedulingproblem auf, das in dem von uns vorgestellten Klassifikationsschema als $P||C_{max}$-Problem charakterisiert werden kann. Gegeben sind m identisch qualifizierte Prozessoren oder Maschinen sowie n Jobs, welche aus jeweils einer Verrichtung mit gegebener Bearbeitungszeit bestehen. Die Jobs sind nicht unterbrechbar und völlig unabhängig voneinander, es gibt also keine Vorrangbeziehungen. Das Ziel besteht darin, die Planlänge, d.h. den Endtermin des letzten Jobs zu minimieren. Das Problem kann als Lastbalancierungsproblem aufgefaßt werden, da die Zielsetzung äquivalent zur Lastminimierung der am stärksten ausgelasteten Maschine ist.

2.1. Komplexität des Problems

Auch wenn unser Problem in seiner Formulierung sehr einfach erscheint, wird unsere Komplexitätsbetrachtung zeigen, daß die Offline- und somit auch die Online-Variante von $P||C_{max}$ bereits für alle Instanzen mit $m \geq 2$ ein NP-hartes Problem darstellt:

Satz 2.1. $P2||C_{max}$ *ist NP-hart.*

Beweis. Betrachten wir das als NP-vollständig bekannte Problem PARTITION [GJ79]. Eine Instanz dieses Problems ist gegeben als eine endliche Menge A, wobei jedes $a_i \in A$ durch seine Größe $s(a_i)$ gekennzeichnet ist.

Die Frage, die nun zu beantworten ist, lautet: Gibt es eine Teilmenge $A' \subseteq A$, so daß A', bezogen auf die Gesamtgröße aller in ihr enthaltenen Elemente, eine „faire" Partition induziert:

$$\sum_{a_i \in A'} s(a_i) = \sum_{a_i \in A \setminus A'} s(a_i) \tag{2.1}$$

Nehmen wir an, es gäbe ein Verfahren, mit dem man das $P2||C_{max}$-Problem in polynomieller Zeit lösen könnte. Dann wären wir in der Lage auch das PARTITION-Problem in dieser Zeitklasse zu lösen. Zunächst transformieren wir die Instanz in eine Ausprägung des $P2||C_{max}$-Problems. Die Anzahl der Jobs entspricht dabei der Kardinalität von A und die Größe der Elemente von A den Bearbeitungszeiten der Verrichtungen.

Es ist somit offensichtlich, daß genau dann eine Partition von A existiert, die unseren Forderungen genügt, wenn wir einen perfekt balancierten Plan erzeugen

können, d.h. wenn für die von unserem Algorithmus erzeugte Planlänge C^*_{max} gilt: $C^*_{max} \leq \frac{1}{2} \sum_{a_i \in A} s(a_i)$ gilt.

Unser Verfahren für $P2||C_{max}$ beantwortet uns diese Frage und wir können die Lösung in unser PARTITION-Problem zurückübersetzen. Alle Problemtransformationen können in polynomieller Zeit ausgeführt werden, was bedeuten würde, daß PARTITION kein NP-vollständiges Problem wäre. Da wir dies jedoch besser wissen, kann es kein deterministisches Polynomialzeitvefahren zur Lösung des $P2||C_{max}$-Problems geben. □

Relxieren wir das Problem, indem wir die Unterbrechbarkeit der Jobbearbeitung erlauben, dann kann das $P||C_{max}$-Problem in polynomieller Zeit effizient gelöst werden [McN59]. Wir wollen uns jedoch im folgenden mit der NP-harten Problemvariante beschäfigen, denn trotz dieses überaus ernüchternden Resultats, oder vielleicht gerade deswegen, hat sich die Wissenschaft sehr intensiv mit dem Problem beschäftigt.

2.2. Die Algorithmen zur Lösung der Offline-Variante

Für die Offline-Variante des $P||C_{max}$ -Problems, d.h. für den Fall vollständiger a priori-Information über die Probleminstanz, präsentierte GRAHAM 1969 den Approximationsalgorithmus LPT (*L*ongest *P*rocessing *T*ime) [Gra69], welcher einen Approximationsfaktor von $\frac{4}{3} - \frac{1}{3m}$ erzielt. Dazu werden die Jobs entsprechend nichtsteigender Bearbeitungszeit in einer Liste angeordnet und in dieser Reihenfolge auf die Prozessoren verteilt, und zwar erhält der erste Prozessor, der frei wird, den größten noch einzuplanenden Job. Auch wenn der erzeugte Plan im worst-case 33% schlechter als der optimale Plan ist, zeigen empirische Studien, wie die von KEDIA [Ked70], daß das Verfahren im allgemeinen nahezu optimale Pläne liefert. Theoretische Resultate bestätigen diese Beobachtung. So konnte in [Cof76] gezeigt werden, daß der Approximationsfaktor $r_{LPT} = 1 + \frac{1}{k} - \frac{1}{km}$ beträgt, wobei k die Anzahl der Jobs ist, die der planlängenbestimmenden Maschine zugewiesen wurden. Mit steigender Maschinenzahl liefert die LPT-Regel also Pläne, die immer näher an der optimalen Lösung liegen. Nimmt man eine exponentielle oder Gleichverteilung an, dann kann man weiter zeigen, daß LPT asymptotisch optimal ist [FRK86], [FRK87]. Die LPT-Regel erzielt also auch für große Jobsequenzen bessere Approximationsfaktoren.

HOCHBAUM und SHMOYS [HS87] verfolgten einen anderen Ansatz, um der Komplexität des Problems Herr zu werden. Sie entwickelten ein in polynomieller Zeit arbeitendes Approximationsschema, welches einen Approximationsfaktor von $1 + \varepsilon$, mit $\varepsilon > 0$ liefert.

Neben diesen Approximationsverfahren mit hoher Lösungsgüte war es möglich einen pseudopolynomiellen Algorithmus [Rot66] mit Laufzeit $O(nC^m)$ zu präsentieren, wobei C eine obere Schranke für die optimale Planlänge darstellt. Für fixes m

kann das Problem somit in pseudopolynomieller Zeit (bezogen auf n) gelöst werden. Folglich ist der Algorithmus praktisch anwendbar, falls C und m kleine Werte annehmen.

2.3. Die Algorithmen zur Lösung der Online-Variante

Das Modell, welches der Offline-Variante zugrunde liegt, konnte der Praxis jedoch in vielen Fällen nicht gerecht werden. Insbesondere die Annahmen, daß alle Jobs zum Zeitpunkt 0 bereits eingeplant werden können und daß ihre Bearbeitungszeiten a prioiri bekannt sind, disqualifizieren das Modell und somit die darauf basierenden Verfahren in vielen Anwendungsgebieten. Daher beschäftigte man sich bereits sehr früh mit der Online-Variante des $P||C_{max}$-Problems. In diesem Modell treffen die Jobs nacheinander ein und müssen unverzüglich und unwiderruflich, ohne Wissen über zukünftige Jobs eingeplant werden. Die Bearbeitunsgzeit eines Jobs ist erst zum Zeitpunkt seines Eintreffens bekannt.

GRAHAM konnte bereits in seiner Arbeit von 1966 [Gra66] eine einfache Heuristik mit Namen LIST präsentieren, die einen kompetitiven Faktor von $2 - \frac{1}{m}$ erzielt. Das Ziel einer möglichst ausgeglichenen Lastverteilung soll dabei durch folgende Einplanungsregel erreicht werden: Weise den gerade eingetroffenen Job der Maschine mit der aktuell niedrigsten Last zu. Die Bedeutung dieses Algorithmus' für das Forschungsgebiet läßt sich ermessen, wenn man sich vor Augen hält, daß über 25 Jahre hinweg keine Verbesserung des kompetitiven Faktors erzielt werden konnte.

Langsam war man davon überzeugt, daß es keinen deterministischen Algorithmus geben kann, der einen besseren asymptotischen Faktor als 2 erreichen kann. Im Rahmen zahlreicher Beweisversuche, gelang es FAIGLE, KERN und TURAN [FKT89] nachzuweisen, daß der kompetitive Faktor von LIST für $m = 2, 3$ optimal ist und daß für $m \geq 4$ kein besserer Faktor als $1 + 1/\sqrt{2}$ möglich ist. BARTAL, KARLOFF und RABANI [BKR94] verbesserten die untere Schranke für $m \geq 4$ auf $1 + 1/\sqrt{2} + \varepsilon_m$, was für große Maschinenzahlen einem Wert von ungefähr 1,837 entspricht. Auch Arbeiten von 1991 [GW], [NC91], in denen $(2 - 1/m - \varepsilon_m)$-kompetitive Algorithmen vorgestellt wurden, konnten die Vermutung nicht widerlegen, da ε_m für große Maschinenzahlen verschwindet.

Das Forschungsgebiet erhielt 1992 neuen Auftrieb, als BARTAL ET AL. [BFKV92] zum ersten Mal einen konstanten kompetitiven Faktor kleiner als 2, nämlich $(2 - 1/70) \approx 1,986$, päsentieren konnten. KARGER ET AL. [KPT96] verbesserten mit dem $CHASM_\alpha$-Algorithmus die Marke auf $1,945$. Die Analyse dieses Algorithmus' zeigt jedoch, welche Anstrengungen notwendig sind, um die obere Schranke weiter nach unten zu drücken. Die Autoren waren gezwungen, Programme zu schreiben, um unter Einsatz des Computers wesentliche Aussagen in ihren Beweisen verifizieren zu können.

Der beste derzeit bekannte Algorithmus für das von uns beschriebene Online-Modell, ist das Verfahren von ALBERS [Alb97]. Es erzielt einen kompetitiven Faktor von $1,923$ für alle $m \geq 2$.

Eine weitere Frage, mit der man sich beschäftigt hat, ist, ob man mittels Randomisierung die Resultate der deterministischen Verfahren verbessern kann. BARTAL ET AL. [BFKV92] machten 1992 Hoffnung, als sie einen $\frac{4}{3}$-kompetitiven Algorithmus für $m = 2$ vorstellten. SHMOYS ET AL. [SWW95] zeigten jedoch 1995, daß Zufall keine große Hilfe im Fall des Online-$P||C_{max}$-Problems darstellt, da jeder randomisierte Algorithmus in dem Modell einen kompetitiven Faktor von mindestens $2 - O(1/\sqrt{m})$ erzielt.

Mit der von uns vorgestellten Online-Variante des $P||C_{max}$-Problems können bereits viele real existierende Probleme in Angriff genommen werden. In Kapitel 3 werden wir einige Anwendungen vorstellen, denen unser Modell im Kern gerecht wird. Aber auch die Grenzen des Modells sind in der Praxis schnell erreicht. Die Annahme, daß die Bearbeitungszeit eines Jobs zum Zeitpunkt seines Eintreffens bekannt ist, ist in vielen Anwendungen des Lastbalancierungsproblems nicht gegeben. Als Beispiel möge man sich die Verteilung von Prozessen auf CPUs in Mehrprozessorsystemen vor Augen halten. Das Betriebssystem hat im allgemeinen keine Chance die Dauer eines Benutzerprozesses abzuschätzen, geschweige denn exakt zu bestimmen.

Wenn wir die gerade beschriebenen Annahmen fallen lassen, erhalten wir eine neue Variante des Online-Problems. Da sie aber nicht Gegenstand dieser Arbeit ist, wollen wir hier lediglich auf die wesentlichen Resultate in diesem Bereich verweisen. Zunächst läßt sich beobachten, daß der LIST-Algorithmus auch diesem Modell gerecht wird, da er bei der Einplanung keine Informationen über Bearbeitungszeiten von Jobs verwendet. AZAR, BRODER und KARLIN bestätigten in [ABK92], daß LIST weiterhin $2 - \frac{1}{m}$-kompetitiv ist. BARTAL ET AL. [KPT96] konnten 1994 zeigen, daß in diesem Modell kein Online-Algorithmus einen besseren kompetitiven Faktor als $2 - \frac{1}{m}$ erzielen kann. Somit folgt, daß LIST ein optimaler Algorithmus für diese Variante des $P||C_{max}$-Problems ist.
Graphik 2.1 systematisiert unsere Darstellung des GRAHAM-Problems.

2.4. Notation

Da wir im folgenden konkrete Scheduling-Algorithmen betrachten und auch analysieren werden, wollen wir an dieser Stelle folgende Notation einführen. Wie bereits erwähnt, bezeichnet m die Anzahl der Maschinen, die unserem Problem zugrunde liegen. Für die abzuarbeitende Jobfolge der Länge n führen wir das Symbol σ_n ein. Jeder Job J_i der Sequenz ist gekennzeichnet durch seine Länge, Größe oder besser Bearbeitungszeit $p_i \geq 0$. Die Zeit wird diskret über die einzuplanenden Jobs gezählt. Der Zeitpunkt t ist erreicht, nachdem die ersten t Jobs eingeplant wurden. Unter der Last einer Maschine verstehen wir die Summe der Bearbeitungzeiten aller Jobs, die ihr zugewiesen sind. Wir gehen davon aus, daß die Maschinen nach ihrer Last geordnet sind, so daß wir mit M_j^t die Maschine mit der j-kleinsten Last l_j^t bezeichnen können. Aussagen der Form „kleinste" oder „größte" Maschine beziehen sich im folgenden auf diese Ordnung. Desweiteren wollen wir die Variable A_j^t einführen, welche die mittlere Last der j kleinsten Maschinen angibt: $A_j^t = \frac{1}{j} \sum_{i=1}^{j} l_i^t$. Da alle

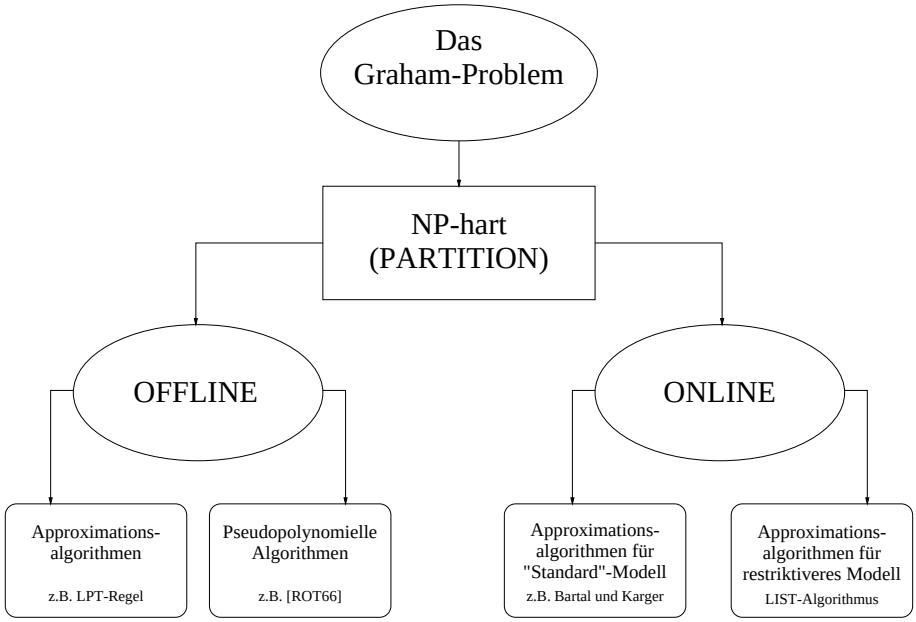


Abbildung 2.1.: **Das** $P||C_{max}$**-Problem:** Varianten und Lösungsverfahren

Bearbeitungszeiten nicht-negativ sind, folgt aus der Definition, daß die Funktion A_j^t monoton im Zeitverlauf sowie der Maschinenzahl wächst. Unsere Algorithmen erzeugen einen Plan, für dessen Länge wir uns interessieren. Daher bezeichnen wir mit $ALG(\sigma_n)$ die von einem beliebigen Algorithmus ALG auf der Jobfolge σ_n erzielte Planlänge. Die Variable $OPT(\sigma_n)$ gibt das Minimum aller für unsere Online-$P||C_{max}$-Instanz konstruierbaren Planlängen an. Somit kann der kompetitive Faktor folgendermaßen definiert werden:

$$c_{P||C_{max}} := \sup_{\sigma} \frac{ALG(\sigma)}{OPT(\sigma)}$$

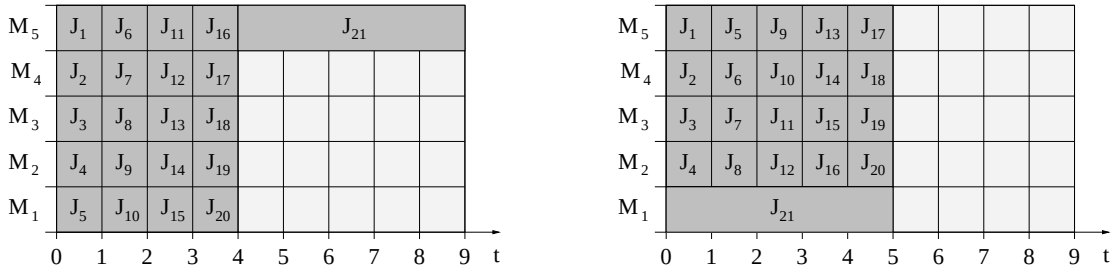
2.5. Die worst-case Analyse von LIST als Ausgangspunkt kompetitiverer Algorithmen

Dazu betrachten wir einfach eine Situation, in der die Last perfekt über alle Maschinen verteilt ist, es aber mindestens eine Maschine gibt, der sehr viele kleine Jobs zugewiesen wurden. Wenn jetzt ein extrem großer Job eintrifft, dann ist leicht einzusehen, daß es besser gewesen wäre, diese Maschine für den großen Job zu reservieren und die kleinen Jobs möglichst gleichmäßig auf die übrigen Maschinen zu verteilen. Eine Jobsequenz, die diese Beobachtung optimal ausnutzt, ist die folgende worst-case-Sequenz:

(i) Zunächst treffen $m(m-1)$ Jobs der Größe 1 ein, welche perfekt auf die m Prozessoren verteilt werden können.

(ii) Nachdem alle Maschinen eine Auslastung von $m-1$ erreicht haben, trifft ein Job der Größe m ein, der unabhängig von der Maschine, auf der er eingeplant wird, einen Plan der Länge $2m-1$ erzeugt.

Der optimale Algorithmus würde natürlich die erste Teilsequenz auf lediglich $m-1$ Maschinen lastbalanciert verteilen. Den großen Job kann er anschließend auf der noch freien Maschine einplanen, was zu einer Planlänge von m führt. Gaphik 2.2 zeigt die von beiden Algorithmen erzeugten Pläne. Somit erhalten wir eine untere



LIST-Algorithmus: $\mathrm{LIST}(\sigma_{21}) = 9$ Optimaler Algorithmus: $\mathrm{OPT}(\sigma_{21}) = 5$

Abbildung 2.2.: **Worst-case der Listanalyse:** $m = 5$

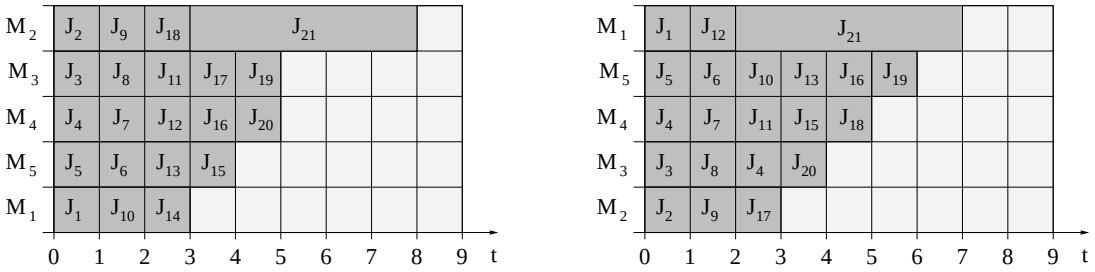
Schranke für den kompetitiven Faktor. Dieser kann natürlich nicht besser sein als in unserer Beispielsequenz: $c_{\mathrm{LIST}} \geq \frac{2m-1}{m} = 2 - \frac{1}{m}$.

Jeder Algorithmus, der den kompetitiven Faktor von LIST verbessern will, muß daher bemüht sein, Pläne zu vermeiden, in denen die Last möglichst gleichmäßig über alle Prozessoren verteilt ist. Trifft nämlich, wie wir gerade gesehen haben, in einer solchen Situation ein extrem großer Job ein, dann wird die Last der Maschine, auf der er eingeplant wird, sehr viel höher als die der übrigen Maschinen sein. Die Planlänge kann suboptimal hoch werden, falls zuvor viele kleine Jobs einer Maschine zugewiesen wurden. Wäre ein Teil dieser kleinen Jobs auf die übrigen Maschinen verteilt worden, dann hätte der große Job das Lastgleichgewicht nicht derart empfindlich stören können.

Diese Überlegungen liegen allen Algorithmen zugrunde, die einen besseren kompetitiven Faktor als den von LIST erzielen konnten. Obwohl die Idee durch eine bewußt ungleichmäßige Lastverteilung den Auswirkungen von Joblängenschwankungen begegnen und somit den worst-case der LIST-Analyse vermeiden zu können, sehr einleuchtend ist, hat es mehr als 25 Jahre gedauert, bis der erste Online-Algorithmus präsentiert wurde, dessen Analyse einen besseren kompetitiven Faktor als den von LIST aufzeigen konnte.

Die Umsetzung obiger Ideen erfordert nämlich sehr viel Fingerspitzengefühl, da ein zu starkes Lastungleichgewicht das kompetitive Verhalten des Algorithmus beeinträchtigt. Im Falle einer gleichmäßigen Verteilung der Joblängen erzielt ein solches Verfahren Pläne mit suboptimal hoher Länge. Diesen durch Lastbalancierung bzw. -debalancierung hervorgerufenen Trade-Off gilt es zu optimieren, will man den kompetitiven Faktor von LIST schlagen.

Wir werden im folgenden zwei Verfahren vorstellen, die das Konzept der Lastdebalancierung auf zwei unterschiedliche Weisen nutzen. Der Algorithmus von BARTAL ET AL.[BFKV92] hält dabei etwas weniger als die Hälfte der Maschinen unter „niedriger" Last und den anderen Teil unter „hoher Last". Die Analyse wird zeigen, daß das Verfahren für $m \geq 70$ einen besseren kompetitiven Faktor als LIST, nämlich $(2 - 1/70) \approx 1,986$, erzielt. Der Algorithmus von KARGER ET AL. [KPT96] verallgemeinert das Verfahren von BARTAL ET AL. und kann als dessen stetige Variante interpretiert werden. Statt sich auf zwei Gruppen von Lastniveaus zu beschränken, wird hier eine Art Treppenmuster der Lastverteilung aufgebaut. Der erzielte kompetitive Faktor von $1,945$ ist für $m \geq 6$ besser als der von LIST. Graphik 2.3 veranschaulichen das Verhalten der Algorithmen, angewendet auf die worst-case-Sequenz der LIST-Analyse.



BARTAL ET AL.: $BFKV(\sigma_{21}) = 8$ KARGER ET AL.: $CHASM_\alpha(\sigma_{21}) = 7$

Abbildung 2.3.: **Debalancierung der Lastverteilung:** verhindert den worst-case der LIST-Analyse ($m = 5$).

2.6. Der Algorithmus von Bartal, Fiat, Karloff und Vohra

Wie wir bereits erwähnt haben, ist dieses Verfahren bestrebt etwa $44,5\%$ der Maschinen in einem Zustand „niedriger" Last zu belassen. Kleine Jobs werden zunächst auf den Maschinen mit „hoher" Last eingeplant, allerdings nur solange, wie das Ungleichgewicht der Lastverteilung nicht so groß wird, daß der kompetitive Faktor seinen avisierten Wert überschreitet. Tritt dieser Fall ein, dann werden auch kleinere Jobs den niedrig belasteten Maschinen zugewiesen. Ähnlich verhält sich der Algorithmus, wenn ein großer Job eintrifft. Auch dieser wird auf einer Maschine mit geringer Auslastung eingeplant, um eine unerwünschte Vergrößerung der Planlänge zu vermeiden. Das Kriterium, das sicherstellen soll, daß der angestrebte kompetitive Faktor eingehalten wird, basiert auf folgender Beobachtung. Wir nehmen an, daß die hoch belastete Maschine, auf der der Job eingeplant werden soll, nach der Einplanung die Planlänge definiert. Diesen Wert vergleichen wir mit der mittleren Last aller niedrig ausgelasteten Maschinen, was aufgrund von A.3 sowie den Monotonie-

eigenschaften von A_j^t eine untere Schranke für den optimalen Algorithmus darstellt. Das Verhältnis dieser Werte liefert uns eine Schätzung des kompetitiven Faktors.

2.6.1. Die Einplanungsregel

Ein eingetroffener Job J_t wird auf der kleinsten Maschine mit hoher Auslastung, nämlich M_{d+1}^{t-1}, eingeplant, wenn das Verhältnis ihrer Last zum Zeitpunkt $t-1$ sowie der Länge des einzuplanenden Jobs zu der mittleren Last aller niedrig belasteten Maschinen höchstens $2-\varepsilon$ mit $\varepsilon = 1/70$ beträgt, d.h. falls gilt:

$$l_{d+1}^{t-1} + p_t \leq (2-\varepsilon)A_d^{t-1} \tag{2.2}$$

Falls diese Bedingung nicht erfüllt ist, dann weisen wir J_t der kleinsten Maschine, d.h. M_1^{t-1} zu.

2.6.2. Der BFKV-Algorithmus

Den Namen der Autoren entsprechend werden wir im folgenden vom sogenannten BFKV-Algorithmus sprechen. Gleichzeitig wollen wir mit $\mathrm{BFKV}(\sigma_t)$ die Planlänge bezeichnen, welche von diesem Algorithmus für die Jobsequenz σ_t erzeugt wird. $\langle x \rangle$ steht für den Funktionswert der *RoundToNearest*-Funktion, die für eine reelle Zahl x den nahesten Wert auf der ganzzahligen Zahlenachse liefert. Die oben formulierte Einplanungsregel erlaubt, einen einfachen Algorithmus zur Lösung des Online-$P||C_{max}$-Problems zu formulieren.

Algorithm 1 Verfahren von Bartal et al. für $P||C_{max}$

```
d := ⟨0,445m⟩
ε := 1/70
for j := 1 to m do
   A_j := 0 (* Initialisierung: t = 0 *)
end for
for all jobs J_t in job sequence do
   if l_{d+1} + p_t ≤ (2 − ε)A_d then
      Assign job J_t to machine M_{d+1}
      l_{d+1} := l_{d+1} + p_t
   else
      Assign job J_t to machine M_1
      l_1 := l_1 + p_t
   end if(* l_j, A_j, M_j, 1 ≤ j ≤ m zu t − 1 *)
   Order M according to non-decreasing machine loads
   A_d := (1/d) Σ_{i=1}^{d} l_i (* l_j, A_j und M_j, 1 ≤ j ≤ m zu t *)
end for(* Laufzeit: O(n log m) *)
```

2.6.3. Analyse der Lösungsgüte

Satz 2.2 (Kompetitiver Faktor). *Sei* $m \geq m_0 \geq 70$ *und* $\varepsilon := \frac{1}{m_0}$, *dann ist der* BFKV-*Algorithmus* $(2-\varepsilon)$*-kompetitiv.*

Beweis. Wir wollen uns im folgenden auf eine Beweisskizze beschränken. Der vollständige Beweis kann in Anhang B nachgelesen werden. Um den Satz zu verifizieren, führt man einen Widerspruchsbeweis durch. Dazu nehmen wir an, es gäbe eine minimale Jobfolge $\sigma_t = J_1, \ldots, J_t$, so daß

$$\mathrm{BFKV}(\sigma_t) > (2-\varepsilon)\mathrm{OPT}(\sigma_t)$$

gilt.

Diese Bedingung impliziert, daß der letzte Job der Sequenz auf der kleinsten Maschine M_1^{t-1} eingeplant, die anschließend die Planlänge definiert, d.h.

$$\mathrm{BFKV}(\sigma_t) = l_1^t = l_1^{t-1} + p_t$$

Wenn wir nun zeigen können, daß es mindestens $m+1$ sogenannte *fette* Jobs mit Mindestlänge $\frac{1}{2(1-\varepsilon)} l_1^{t-1}$ in der Sequenz gibt, dann folgt, daß auch der $(m+1)$größte Job in σ_n diese Länge erreicht. Aus A.2 und A.4 erhalten wir somit:

(i) $p_t \leq \mathrm{OPT}(\sigma_t)$

(ii) $\frac{1}{1-\varepsilon} l_1^{t-1} \leq \mathrm{OPT}(\sigma_t) \iff l_1^{t-1} \leq (1-\varepsilon)\mathrm{OPT}(\sigma_t)$

Es folgt der gesuchte Widerspruch zur Beweisannahme:

$$\begin{aligned} \mathrm{BFKV}(\sigma_t) &= l_1^{t-1} + p_t \\ &\leq (1-\varepsilon)\mathrm{OPT}(\sigma_t) + \mathrm{OPT}(\sigma_t) \\ &= (2-\varepsilon)\mathrm{OPT}(\sigma_t) \end{aligned}$$

Man identifiziert nun $m+1$ fette Jobs, indem man zeigt, daß der letzte Job fett ist und daß jeder der m Maschinen im Laufe der Einplanung mindestens ein fetter Job zugewiesen wird. Die zweite Aussage kann nicht unmittelbar für alle Maschinen verifiziert werden, so daß eine geeignete Partition der Maschinenmenge gefunden werden muß. Für die dabei entstehenden Maschinengruppen läßt sich die Behauptung jedoch leicht nachweisen. □

2.6.4. Beispiel

Wir werden nun an einer Beispielsequenz die Arbeitsweise des Algorithmus nachvollziehen. In den Tabellen geben wir die Werte der Variablen an, die für unsere Entscheidungsregel von Bedeutung sind. Der Schedulingplan gibt uns Auskunft über die Lastsituation nach der Einplanung.
Wir betrachten die folgende Jobsequenz:

$$\sigma_{15} = (J_1, J_2, \ldots, J_{15})$$

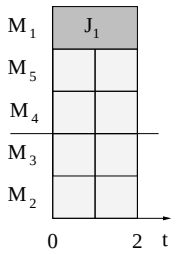
mit den Bearbeitungszeiten

$$\{2,4,3,5,7,3,4,9,5,2,2,3,4,3,10\},$$

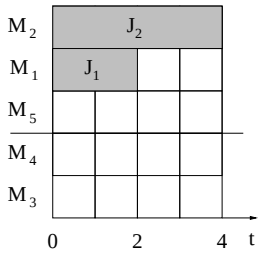
welche auf 5 identisch qualifizierten Maschinen einzuplanen ist.

Im Zeitintervall $[0,4]$ werden die Jobs zunächst auf verschiedenen Maschinen eingeplant, bis alle über eine positive Last verfügen. Anschließend wird bis zum Zeitpunkt $t = 7$ ein bewußtes Lastungleichgewicht aufgebaut, so daß der große Job J_8 auf einer kleinen Maschine eingeplant werden kann. Im Zeitintervall $[8,13]$ wird die avisierte Differenzierung in hoch und niedrig belastete Maschinen realisiert. Dabei ist man stets bemüht, die kleinen Jobs $J_{10}, J_{11}, J_{12}, J_{13}$ und J_{14} auf den großen Maschinen einzuplanen ($t = 9, t = 10, t = 12, t = 13, t = 14$), muß jedoch die kleinen Maschinen zum Zeitpunkt $t = 11$ „nachziehen", um den angestrebten kompetitiven Faktor nicht zu gefährden. Zum Zeitpunkt $t = 13$ und $t = 14$ sieht man sehr schön, daß es zwei Gruppen von Machinen gibt, nämlich die unter geringer (M_3, M_4) und die unter großer Last (M_1, M_2, M_5). Wir haben die Länge des letzten Jobs J_{15} bewußt sehr groß gewählt, um noch einmal deutlich zu machen, wie der Algorithmus die Schwäche von LIST in der worst-case-Analyse umgeht.

$t = 0$	$l^0_{d+1} = 0$ $p_1 = 2$ $A^0_d = 0$	$l^0_{d+1} + p_1$ $>$ $1,986A^0_d$
J_1 auf M_1 einplanen		



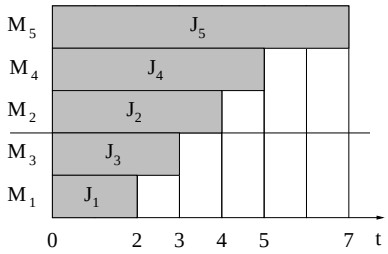
$t = 1$	$l^1_{d+1} = 0$ $p_2 = 4$ $A^1_d = 0$	$l^1_{d+1} + p_2$ $>$ $1,986A^1_d$
J_2 auf M_2 einplanen		



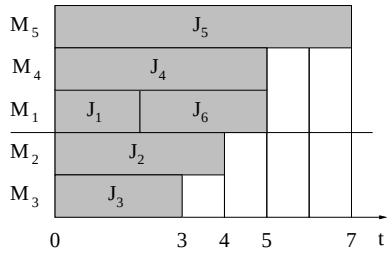
$t = 2$	$l^2_{d+1} = 0$ $p_3 = 3$ $A^2_d = 0$	$l^2_{d+1} + p_3$ $>$ $1,986A^2_d$
J_3 auf M_3 einplanen		

$t = 3$	$l^3_{d+1} = 1$ $p_4 = 5$ $A^3_d = 0$	$l^3_{d+1} + p_4$ $>$ $1,986A^3_d$
J_4 auf M_4 einplanen		

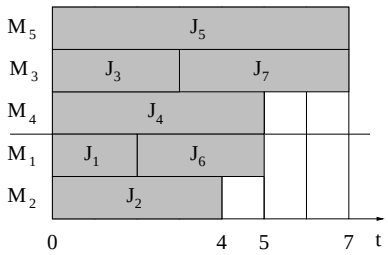
$t = 4$	$l^4_{d+1} = 3$ $p_5 = 7$ $A^4_d = 1$	$l^4_{d+1} + p_5$ $>$ $1,986A^4_d$
J_5 auf M_5 einplanen		



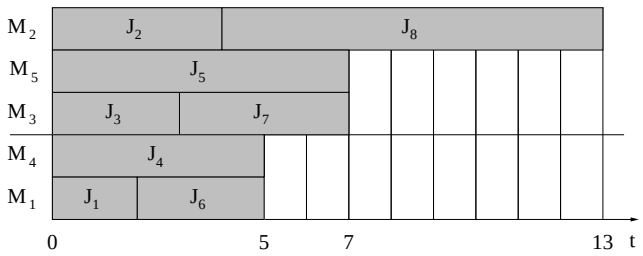
$t = 5$	$l^5_{d+1} = 4$ $p_6 = 3$ $A^5_d = 2,5$	$l^5_{d+1} + p_6$ $>$ $1,986A^5_d$
J_6 auf M_1 einplanen		



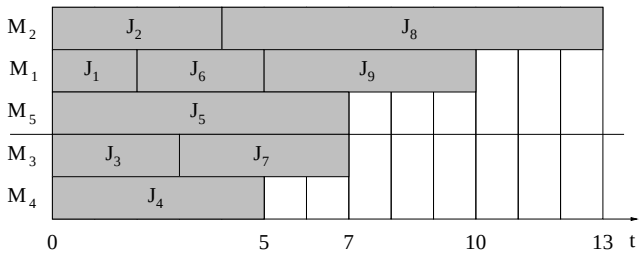
$t = 6$	$l^6_{d+1} = 5$ $p_7 = 4$ $A^6_d = 3,5$	$l^6_{d+1} + p_7$ $>$ $1,986A^6_d$
J_7 auf M_3 einplanen		



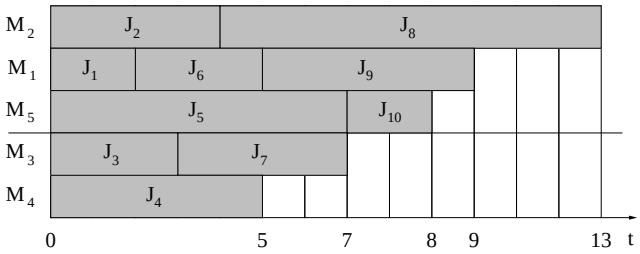
$t = 7$	$l^7_{d+1} = 5$ $p_8 = 9$ $A^7_d = 4.5$	$l^7_{d+1} + p_8$ $>$ $1,986A^7_d$
J_8 auf M_2 einplanen		



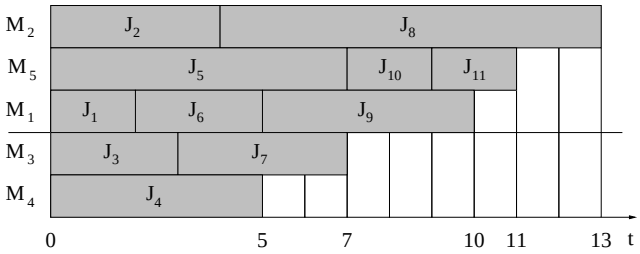
$t = 8$	$l^8_{d+1} = 7$ $p_9 = 5$ $A^8_d = 5$	$l^8_{d+1} + p_9$ $>$ $1,986A^8_d$
J_9 auf M_1 einplanen		



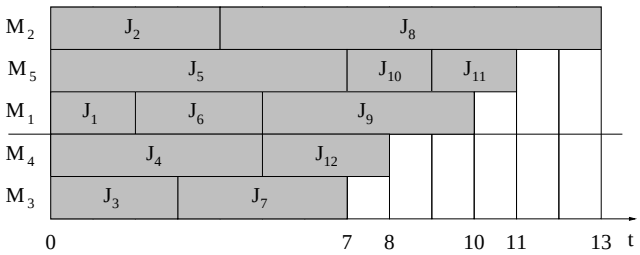
$t = 9$	$l^9_{d+1} = 7$ $p_{10} = 2$ $A^9_d = 6$	$l^9_{d+1} + p_{10}$ $\leq$ $1,986A^9_d$
J_{10} auf M_5 einplanen		



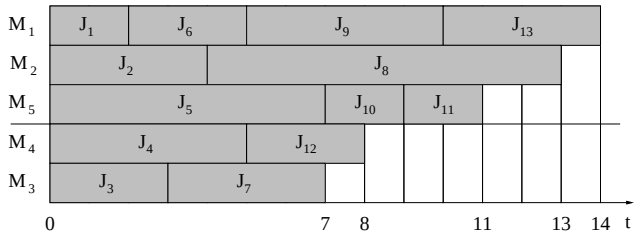
$t = 10$	$l^{10}_{d+1} = 9$ $p_{11} = 2$ $A^{10}_d = 6$	$l^{10}_{d+1} + p_{11}$ $\leq$ $1,986A^{10}_d$
J_{11} auf M_5 einplanen		



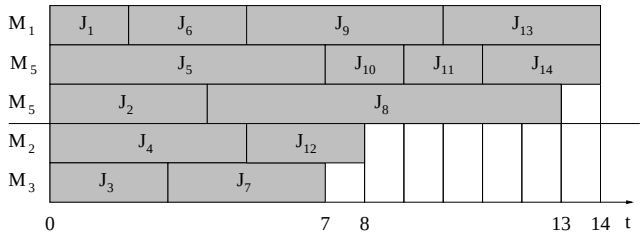
$t = 11$	$l^{11}_{d+1} = 10$ $p_{12} = 3$ $A^{11}_d = 6$	$l^{11}_{d+1} + p_{12}$ $>$ $1,986A^{11}_d$
J_{12} auf M_4 einplanen		



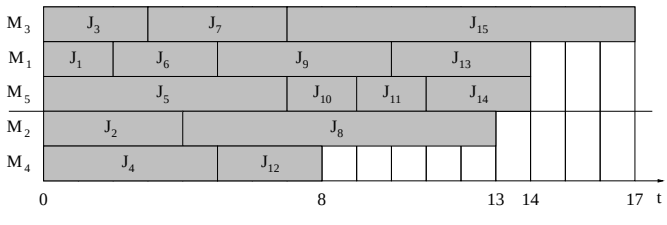
$t = 12$	$l^{12}_{d+1} = 10$ $p_{13} = 4$ $A^{12}_d = 7,5$	$l^{12}_{d+1} + p_{13}$ $\leq$ $1,986A^{12}_d$
J_{13} auf M_1 einplanen		



$t = 13$	$l^{13}_{d+1} = 11$ $p_{14} = 3$ $A^{13}_d = 7,5$	$l^{13}_{d+1} + p_{14}$ $\leq$ $1,986A^{13}_d$
J_{14} auf M_5 einplanen		



$t = 14$	$l^{14}_{d+1} = 13$ $p_{15} = 10$ $A^{14}_d = 7,5$	$l^{14}_{d+1} + p_{15}$ $>$ $1,986A^{14}_d$
J_{15} auf M_3 einplanen		



2.7. Der Algorithmus von Karger, Phillips und Torng

Wie bereits oben erwähnt, kann der Algorithmus von KARGER ET AL. als eine stetige Variante des BFKV-Algorithmus aufgefaßt werden. Das BFKV-Verfahren unterscheidet im wesentlichen zwischen „akzetablen" und „zu großen" Jobs, die zwar auf zwei unterschiedlichen Lastniveaus, jedoch stets auf der kleinsten Maschine des jeweiligen Niveaus, eingeplant werden. Innerhalb eines Lastniveaus wird daher im Prinzip der LIST-Algorithmus angewandt, der den Job der aktuell am niedrigsten belasteten Maschine zuweist.

Der Algorithmus, den wir im folgenden betrachten werden, geht noch einen Schritt weiter, indem ein im besten Fall m-stufiges Treppenmuster der Lastverteilung aufrechterhält. Ein Job wird abhängig von seiner Größe auf der größten Maschine eingeplant, die jedoch die Voraussetzung erfüllen muß, daß der avisierte kompetitive Faktor eingehalten werden kann. Der Algorithmus versucht also niemals einen Teil der Maschinen optimal ausbalanciert zu halten, sondern bemüht sich im Gegenteil um eine möglichst fein abgestufte Unausgewogenheit der Lastverteilung. Der Parameter α bestimmt dabei den Grad der Lastdebalancierung.

2.7.1. Die Einplanungsregel

Ein eingetroffener Job J_t wird auf der größten Maschine M_k^{t-1} eingeplant, für die gilt, daß das Verhältnis ihrer Last zum Zeitpunkt $t-1$ sowie der Länge des einzuplanenden Jobs zu der mittleren Last aller kleineren Maschinen höchstens $\alpha = 1,945$ beträgt. Wir suchen also das maximale k mit der folgenden Eigenschaft:

$$l_k^{t-1} + p_t \leq \alpha A_{k-1}^{t-1} \tag{2.3}$$

Falls keine solche Maschine M_k^{t-1} existiert, dann weisen wir J_t der kleinsten Maschine, d.h. M_1^{t-1} zu.

2.7.2. Der CHASM$_\alpha$-Algorithmus

Die oben formulierte Einplanungsregel erlaubt, einen einfachen Algorithmus zur Lösung des $P||C_{max}$ Problems zu formulieren.

2.7.3. Analyse der Lösungsgüte

Satz 2.3 (Kompetitiver Faktor). *Der* CHASM$_\alpha$*-Algorithmus ist* α*-kompetitiv, mit* $\alpha = 1,945$.

Beweis. Wir wollen uns auch hier mit einer Beweisskizze begnügen und verweisen für eine detaillierte Darstellung auf Anhang C. Die Beweisidee ist ähnlich der, die wir für den Algorithmus von BARTAL ET AL. 2.6 gesehen haben. Die Verifikation

Algorithm 2 Verfahren von Karger, Phillips und Torng für $P||C_{max}$

$\alpha := 1,945$
$A_0 := \infty$
for $j := 1$ to m **do**
  $A_j := 0$ (* Initialisierung: $t = 0$ *)
end for
for all jobs J_t in job sequence **do**
  $k := \max\{j \mid l_j + p_t \leq \alpha A_{j-1}\}$
  Assign job J_t to machine M_k (* l_j, A_j, M_j, $1 \leq j \leq m$ zu $t-1$ *)
  $l_k := l_k + p_t$
  Order M according to non-decreasing machine loads
  for $j := 1$ to m **do**
    $A_j := \frac{1}{j} \sum_{i=1}^{j} l_i$
  end for(* l_j, A_j und M_j, $1 \leq j \leq m$ zu t *)
end for(* Laufzeit: $O(mn)$ *)

der Behauptung erfolgt jedoch mittels Induktion über die Länge der Jobsequenz. Im Induktionsschritt ist daher zu zeigen:

$$CHASM_\alpha(\sigma_t) \leq \alpha OPT(\sigma_t)$$

Nachdem man einige Fälle ausgeschlossen hat, in denen die Behauptung offensichtlich ist, läßt sich nachweisen, daß der letzte Job J_t auf der kleinsten Maschine M_1^{t-1} eingeplant wird, welche anschließend die Planlänge definiert. Mit dieser Erkenntnis ist es möglich, die Planlänge des $CHASM_\alpha$-Algorithmus' abzuschätzen: Sie beträgt maximal $\alpha 2b$, d.h.

$$CHASM_\alpha(\sigma_t) \leq \alpha 2b$$

Auch bei diesem Verfahren kann man zeigen, daß es mindestens $m+1$ *fette* Jobs einer gewissen Mindestlänge b geben muß, so daß für die Länge des $(m+1)$größten Jobs $p_{(m+1)} \geq b$ gilt, was nach A.4 eine untere Schranke für den optimalen Algorithmus darstellt. Somit erhält man die gesuchte Aussage:

$$CHASM_\alpha(\sigma_t) \leq \alpha 2b \leq \alpha OPT(\sigma_t).$$

Es ist wieder relativ einfach zu sehen, daß der letzte Job fett ist. Wollen wir jedoch verifizieren, daß jeder der m Maschinen ein solcher Job zugewiesen wird, so sind wir gezwungen, den Einplanungsprozeß in zwei Phasen zu zerlegen. Man definiert sich dazu sogenannte *kritische* Jobs, welche *niedrig belastete* Maschinen in einen Zustand *hoher Belastung* überführen. Für die erste Phase läßt sich auf analytische Art und Weise zeigen, daß alle kritischen Jobs dieser Phase fett sind. Um diese Behauptung auch für die zweite Phase verifizieren zu können, müssen Lineare Programme gelöst werden, welche untere Schranken für die Längen der kritischen Jobs berechnen. Man müßte nun für alle Maschinenzahlen diese Programme lösen und dabei α so wählen,

daß die kritischen Jobs von Phase 2 fett sind. Die technischen Ressourcen der Autoren ließen jedoch nur folgende Erkenntnis zu:

Lemma 2.4. *Für* $6 \leq m \leq 13000$ *sind die kritischen Jobs der zweiten Phase fett und der* CHASM_α*-Algorithmus somit* $\alpha = 1,943$*-kompetitiv.*

Diese Aussage gilt auch für $\alpha \geq 1,943$. Will man die LPs für größere Maschinenzahlen lösen, dann muß man die Maschinen gruppieren und die Nebenbedingungen für diese Maschinenblöcke formulieren. Auf diese Weise ist es möglich m als Parameter durch eine andere Größe x zu ersetzen, so daß die für x gefundenen Aussagen auch für alle m ab einer von x abhängigen Mindestgröße gelten. Diese Konstruktion erlaubt den Beweis mit folgendem Lemma abzuschließen.

Lemma 2.5. *Für* $m > 13000$ *und* $\alpha = 1,945$ *sind die kritischen Jobs in Phase 2 fett und der* CHASM_α*-Algorithmus* α*-kompetitiv.*

□

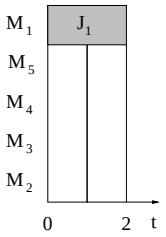
Abschließend wollen wir noch zwei wichtige Bemerkungen machen. Die Autoren vermuten, daß ihr Algorithmus sogar α-kompetitiv für $\alpha = 1,943$ ist. Sie konnten diese Behauptung jedoch nur für $m \leq 13000$ sowie für $m > 20000^2$ verifizieren. Die zweite Aussage betrifft die Schärfe der Analyse. KARGER ET AL.konnten in [KPT96] zeigen, daß CHASM_α für beliebiges α bestenfalls $1,9378$-kompetitiv ist. Dies zeigt, daß die Analyse sehr präzise ist und daß im average-case keine wesentlich besseren kompetitiven Faktoren zu erwarten sind.

2.7.4. Beispiel

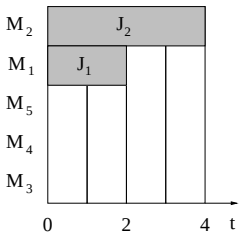
Um die Arbeitsweisen der beiden vorgestellten Algorithmen besser vergleichen zu können, wollen wir an dieser Stelle auf die Beispielsequenz für den BFKV-Algorithmus 2.6.4 zurückgreifen. Der Einplanungsprozeß sieht somit folgendermaßen aus.

Im Zeitintervall $[0,4]$ wird zunächst analog zur Vorgehensweise des Algorithmus von BARTAL ET AL. jeder Maschine ein Job zugewiesen. Im Zeitpunkt $t = 7$ ist bereits eine relativ gleichmäßige Lastdebalancierung über alle Maschinen erreicht, so daß der große Job J_8 einer kleinen Maschine zugeteilt werden kann. Die Lastdebalancierung ist nun zu extrem, was durch die Einplanung von J_9 auf der kleinsten Maschine wieder ausgeglichen wird. Anschließend dehnt der Algorithmus die Lastdebalancierung zu einem Treppenmuster aus, indem er kleine Jobs $J_{10}, J_{11}, J_{12}, J_{13}$ und J_{14} abwechselnd auf die Maschinen mit hoher ($t = 9, t = 10, t = 13$) mittlerer ($t = 11$) und niedriger Last ($t = 12$) veteilt. Zum Zeitpunkt $t = 11$ und $t = 12$ sieht man sehr schön, wie gleichmäßig abgestuft die extreme Lastdebalancierung zwischen der kleinsten und größten Maschine ausgestaltet ist. Der letzte Job J_{15} veranschaulicht die positiven Eigenschaften der Lastdebalancierung im Fall von starken Joblängenschwankungen.

M_j^{t-1}	l_j^{t-1}	A_{j-1}^{t-1}	$l_j^{t-1} + p_t$	αA_{j-1}^{t-1}
5	0	0	2	0
4	0	0	2	0
3	0	0	2	0
2	0	0	2	0
1	0	∞	2	∞
$t = 0$:	J_1 auf M_1 einplanen			



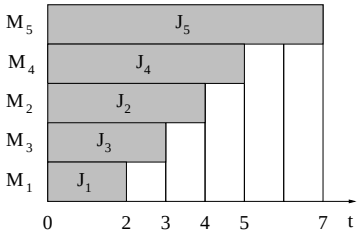
M_j^{t-1}	l_j^{t-1}	A_{j-1}^{t-1}	$l_j^{t-1} + p_t$	αA_{j-1}^{t-1}
1	2	0	6	0
5	0	0	4	0
4	0	0	4	0
3	0	0	4	0
2	0	∞	4	∞
$t = 1$:	J_2 auf M_2 einplanen			



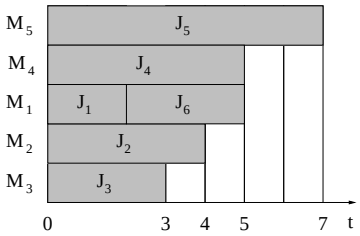
M_j^{t-1}	l_j^{t-1}	A_{j-1}^{t-1}	$l_j^{t-1} + p_t$	αA_{j-1}^{t-1}
2	4	0,5	7	< 1
1	2	0	5	0
5	0	0	3	0
4	0	0	3	0
3	0	∞	3	∞
$t = 2$:	J_3 auf M_3 einplanen			

M_j^{t-1}	l_j^{t-1}	A_{j-1}^{t-1}	$l_j^{t-1} + p_t$	αA_{j-1}^{t-1}
2	4	1,25	9	$< 2,5$
3	3	$0,\bar{6}$	8	$< 1,\bar{3}$
1	2	0	7	0
5	0	0	5	0
4	0	∞	5	∞
$t = 3$:	J_4 auf M_4 einplanen			

M_j^{t-1}	l_j^{t-1}	A_{j-1}^{t-1}	$l_j^{t-1} + p_t$	αA_{j-1}^{t-1}
4	5	2,25	12	$< 4,5$
2	4	$1,\bar{6}$	11	$< 3,\bar{3}$
3	3	1	10	< 2
1	2	0	9	0
5	0	∞	7	∞
$t = 4$:	J_5 auf M_5 einplanen			



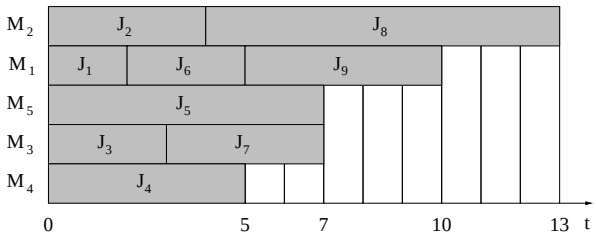
M_j^{t-1}	l_j^{t-1}	A_{j-1}^{t-1}	$l_j^{t-1} + p_t$	αA_{j-1}^{t-1}
5	7	3,5	10	< 7
4	5	3	8	< 6
2	4	2,5	7	< 5
3	3	2	6	< 4
1	2	∞	5	∞
$t = 5$:	J_6 auf M_1 einplanen			



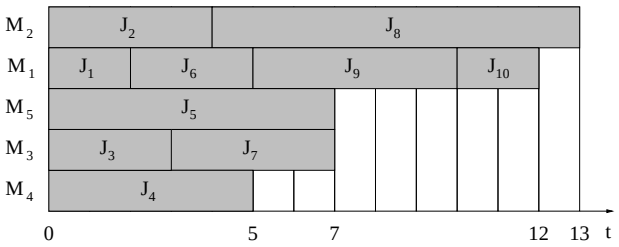
M_j^{t-1}	l_j^{t-1}	A_{j-1}^{t-1}	$l_j^{t-1} + p_t$	αA_{j-1}^{t-1}
5	7	4,25	11	$< 8{,}5$
4	5	4	9	< 8
1	5	3,5	9	< 7
2	4	3	8	< 6
3	3	∞	7	∞
$t = 6$:	J_7 auf M_3 einplanen			

M_j^{t-1}	l_j^{t-1}	A_{j-1}^{t-1}	$l_j^{t-1} + p_t$	αA_{j-1}^{t-1}
5	7	5,25	16	$< 10{,}5$
3	7	$4, \bar{6}$	16	$< 9, \bar{3}$
4	5	4,5	14	< 9
1	5	4	14	< 8
2	4	∞	13	∞
$t = 7$:	J_8 auf M_2 einplanen			

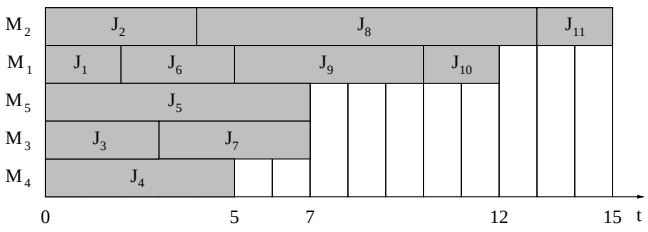
M_j^{t-1}	l_j^{t-1}	A_{j-1}^{t-1}	$l_j^{t-1} + p_t$	αA_{j-1}^{t-1}
2	13	6	18	< 12
5	7	$5, \bar{6}$	12	$< 11, \bar{3}$
3	7	5	12	< 10
4	5	5	10	< 10
1	5	∞	10	∞
$t = 8$:	J_9 auf M_1 einplanen			



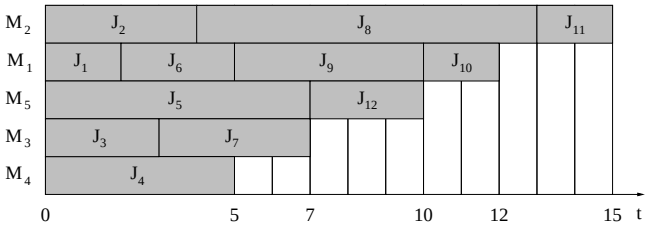
M_j^{t-1}	l_j^{t-1}	A_{j-1}^{t-1}	$l_j^{t-1} + p_t$	αA_{j-1}^{t-1}
2	13	7,25	15	$< 14{,}5$
1	10	$6, \bar{3}$	12	$\approx 12, \bar{6}$
5	7	6	9	≈ 12
3	7	5	9	≈ 10
4	5	∞	7	∞
$t = 9$:	J_{10} auf M_1 einplanen			



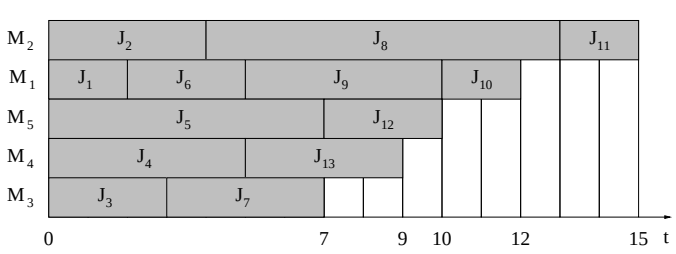
M_j^{t-1}	l_j^{t-1}	A_{j-1}^{t-1}	$l_j^{t-1} + p_t$	αA_{j-1}^{t-1}
2	13	7,75	15	$< 15{,}5$
1	12	$6, \bar{3}$	14	$\approx 12, \bar{6}$
5	7	6	9	≈ 12
3	7	5	9	≈ 10
4	5	∞	7	∞
$t = 10$:	J_{11} auf M_2 einplanen			



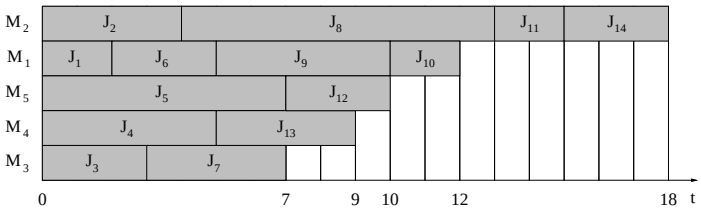
M_j^{t-1}	l_j^{t-1}	A_{j-1}^{t-1}	$l_j^{t-1} + p_t$	αA_{j-1}^{t-1}
2	15	7,75	18	$< 15{,}5$
1	12	$6, \bar{3}$	15	$< 12, \bar{6}$
5	7	6	10	≈ 12
3	7	5	10	< 10
4	5	∞	8	∞
$t = 11$:	J_{12} auf M_5 einplanen			



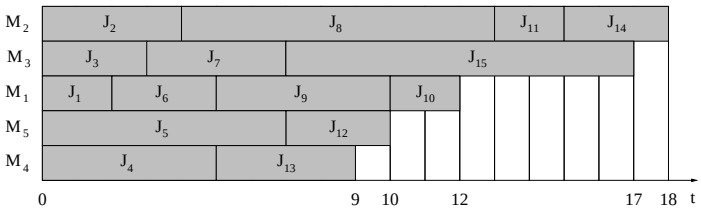
M_j^{t-1}	l_j^{t-1}	A_{j-1}^{t-1}	$l_j^{t-1} + p_t$	αA_{j-1}^{t-1}
2	15	8,5	19	< 17
1	12	$7,\bar{3}$	16	$< 14,\bar{6}$
5	10	6	14	< 12
3	7	5	11	< 10
4	5	∞	9	∞
$t = 12$:	J_{13} auf M_4 einplanen			



M_j^{t-1}	l_j^{t-1}	A_{j-1}^{t-1}	$l_j^{t-1} + p_t$	αA_{j-1}^{t-1}
2	15	9,5	18	≈ 19
1	12	$8,\bar{6}$	15	$\approx 17,\bar{3}$
5	10	8	13	≈ 16
4	9	7	12	≈ 14
3	7	∞	10	∞
$t = 13$:	J_{14} auf M_2 einplanen			



M_j^{t-1}	l_j^{t-1}	A_{j-1}^{t-1}	$l_j^{t-1} + p_t$	αA_{j-1}^{t-1}
2	18	9,5	28	< 19
1	12	$8,\bar{6}$	22	$< 17,\bar{3}$
5	10	8	20	< 16
4	9	7	19	< 14
3	7	∞	17	∞
$t = 14$:	J_{15} auf M_3 einplanen			



3. Das Graham-Problem im betriebswirtschaftlichen Umfeld

Schedulingprobleme begegnen uns im alltäglichen Leben an vielen Stellen. Die meisten dieser Fragestellungen sind uns so selbstverständlich geworden, daß wir nicht mehr weiter über sie nachdenken. Ob wir nun eine Menge von Terminen und Verabredungen auf die uns zur Verfügung stehende Arbeits- bzw. Freizeit verteilen oder beim Einkauf die Kasse mit der kürzesten Warteschlange wählen, all diese $P||C_{max}$-Probleme lösen wir mit einfachsten Entscheidungsregeln, völlig intuitiv und zu unserer vollsten Zufriedenheit. Die Gründe, warum wir über diese Probleme so wenig nachdenken müssen, sind offensichtlich. Die Problemistanzen, mit denen wir in der Relität konfrontiert sind so übersichtlich, daß man mit „gesundem Menschenverstand" nahezu optimale Pläne entwickeln kann. Desweiteren reduzieren wir die reale Komplexität derart, daß simple Strategien, wie die des LIST-Verfahrens, auf die meisten Probleme angewendet werden können.

Im betriebswirtschaftlichen Umfeld sind diese Voraussetzungen nicht gegeben. Die Probleminstanzen, die man optimal lösen möchte, übersteigen bei weitem die Leistungsfähigkeit des menschlichen Gehirns. Ohne Computerunterstützung ist eine Lösung nicht denkbar. Desweiteren existieren meist Restriktionen, die man nicht ignorieren darf oder möchte. Suboptimale Ergebnisse stellen in Unternehmen niemanden zufrieden, es sei denn, die Problemkomplexität erlaubt keine bessere Lösungsgüte.

Somit gestaltet es sich schwierig, reale betriebswirtschaftliche Anwendungsbeispiele für solch einfach zu formulierende Probleme wie $P||C_{max}$ zu finden. Wie gut betriebswirtschaftliche Sachverhalte von unserem Modell erfaßt werden, hängt im wesentlichen vom Detaillierungsgrad ab, mit dem wir die Problemstellungen analysieren. Je weitergehender man den Sachverhalt spezifiert, umso scheller wird man feststellen, daß zusätzliche Nebenbedingungen berücksichtigt werden müssen, die ein komplexeres Modell fordern.

Dennoch macht es in vielen Fällen Sinn gewisse Idealisierungen zu akzeptieren. Es ist nicht garantiert, daß der Einsatz komplexerer Modelle zu Resultaten führt, die den gestiegenen Zeitbedarf zur Lösung der Probleme rechtfertigen. Außerdem können Verfahren, denen einfache Modelle zugrunde liegen, auch große Instanzen mit vielen Unsicherheitsfaktoren, wie sie beispielsweise in der Auftragsplanung auf der dispositven Ebene auftreten, lösen. Dennoch wollen wir auf die Darstellung der klassischen Anwendungsbeispiele aus der Fertigungssteuerung [Sch97] verzichten und

uns aktuelleren Fragestellungen zuwenden.

3.1. Lastbalancierung und E-Commerce

Ausfallsicherheit, Redundanz oder Failover im Zusammenhang mit Computersystemen sind keine völlig neuen Themen, denen sich Rechenzentren von Unternehmen, Universitäten oder sonstigen Organisationen stellen müssen. Gerade für Unternehmen hat sich jedoch die Problemstellung in diesem Bereich grundlegend verändert [Tör99].

Auch wenn bisher eine hohe Verfügbarkeit der EDV-Systeme ein bedeutendes Ziel der Systemtechniker und Administratoren war, so stellten Ausfälle höchstens im finanzwirtschaftlichen Sektor, insbesondere in der Bankenbranche, potentiell erfolgs- oder gar überlebenskritische Ereignisse dar. Es entand jedoch selbst in diesen Bereichen kein echter Bedarf an automatisierten Lösungen, da die Last, denen die Systeme ausgesetzt waren, jederzeit absehbar war. Nahm der Transaktionsverkehr oder die Zahl der vernetzten Mitarbeiter zu, so wurde die Rechenleistung entsprechend skaliert. Rechenzentren anderer Branchen und Organisationen entwickelten manuelle Lösungen, die Ausfallzeiten bewußt in Kauf nahmen.

Aus dem Nischengeschäft für Failoverlösungen ist jedoch in den letzten beiden Jahren ein potentieller Wachstumsmarkt geworden. Der Grund liegt in der zunehmenden Bedeutung des Internets als Handelsplatz. Der Wunsch, 24 Stunden am Tag, 365 Tage im Jahr Electronic Commerce betreiben zu können, hat den erforderlichen Kundendruck zur Entwicklung hochverfügbarer Systeme aufkeimen lassen. Die oben geschilderten Vorgehensweisen der Rechenzentren werden den neuen Anforderungen nicht mehr gerecht, da der Kundeneinzugsbereich globales Ausmaß erreicht hat und die Last, die die Webserver erwartet, nicht mehr absehbar ist. Doch gerade im E-Commerce-Umfeld bedeutet Verfügbarkeit Serviceverfügbarkeit. Ausfallzeiten sind somit nicht mehr nur Produktivitätsverluste, die Kosten verursachen, sondern gleichzeitig Umsatzeinbußen. Die Abhängigkeit – selbst der kleinen Unternehmen – von der Einsatzbereitschaft ihrer Webserver ist somit enorm gestiegen. Dabei steht natürlich nicht die Einsatzbereitschaft eines speziellen Rechners im Vordergrund, sondern die Verfügbarkeit der Dienste und Ressourcen, welche Anwendern, insbesondere Kunden, angeboten werden.

Die entscheidende Erkenntnis in diesem Zusammenhang ist nun, daß die Verfügbarkeit von Webservern und erfolgreicher Lastausgleich auf diesen eng miteinander verknüpfte Problemstellungen sind. Es wurden daher sogenannte Lastbalancierer für Internet-Lösungen entwickelt. Dabei handelt es sich um ein System, das vor die Webserver geschaltet wird und die Anfragen aus dem Netz an die noch zur Verfügung stehenden Server verteilt. Graphik 3.1 verdeutlicht das Konzept. Das Unternehmen tritt somit mit einer einzigen IP-Adresse nach außen auf („one face to the customer"), kann jedoch die ankommende Last auf mehrere beliebig entfernte Rechner verteilen. Insbesondere können sich die Server im Falle eines Ausfalls gegenseitig ersetzen. Man verfolgt also zwei Strategien zur Erreichung der Verfügbarkeit:

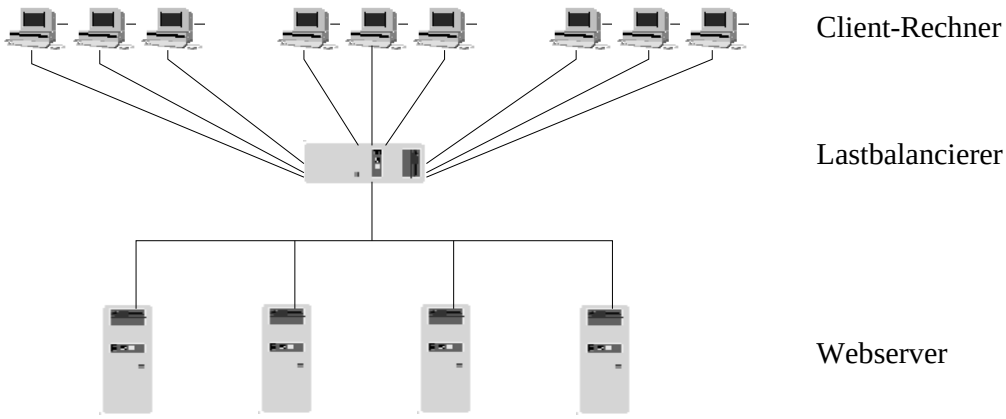


Abbildung 3.1.: **Der Lastbalcierer** verteilt die Anfragelast gleichmäßig auf Backend-Server.

1. *Redundanz*: Die Rechnersysteme springen füreinander ein und „mounten" die Platten des ausgefallenen Systems.

2. *Lastbalancierung*: Es findet eine Verteilung der Last auf die verfügbaren Systeme statt.

Nebenbei führt der Einsatz von Load-Balancern zu höherer Serverleistung, welche durch den einfachen Aufbau IP-basierter Cluster weiter gesteigert werden kann.

Eine Studie von *Collaborative Research*[Tör99] prognostiziert diesem Markt eine vielversprechende Entwicklung. Ihrer Einschätzung nach wird das Marktvolumen von heute 259,5 Mio USD auf 800 Mio USD im Jahre 2002 anwachsen. Viele Großunternehmen mit hoher Netzlast, wie beispielsweise Microsoft, Chase Manhattan Bank oder Reuters setzen die Technik bereits ein.

Nachdem wir das betriebswirtschaftliche Potential dieser Systeme dargelegt haben, wollen wir uns mit der algorithmischen Seite der Lastbalancierung auf Webservern beschäftigen. Das Schedulingmodell hierfür ist sehr einfach. Wenn wir von der nicht sehr unrealistischen Annahme ausgehen, daß ein Unternehmen seine E-Commerce-Lösung inklusive der gesamten Hardware einkauft und die Redundanz der Server auf die einfachste Art und Weise sicherstellen möchte, dann wird es sich bei den Rechnern um identische Systeme handeln. Somit stehen uns m parallele, identisch qualifizierte Prozessoren zur Verfügung. Die Anzahl der Server, m, kann natürlich aufgrund von Ausfällen schwanken. Die Jobankunftszeiten sind a priori unbekannt, da der Zugriff auf Internetseiten nicht an bestimmte Zeiten gebunden ist. Wir haben es also mit einer Online-Fragestellung zu tun. Die Joblänge, d.h. die Dauer der Bearbeitung einer Anfrage durch den Server, ist zunächst ebenfalls unbekannt. Sie hängt ganz entscheidend von der Art der Anfrage ab. Wird ein reines Textdokument an den Klienten gesendet, dann ist die Jobgröße sehr gering, muß jedoch ein kompliziertes Skript oder ein Download abgewickelt werden, dann ist die Bearbeitungszeit des Jobs entsprechend hoch. Diese Beobachtung macht deutlich, warum

nicht die Anzahl der Verbindungen, die ein Server bearbeitet, entscheidend ist, sondern seine Gesamtauslastung.

Die Joblängen können zum Zeitpunkt des Eintreffens entweder über die Art der Anfrage abgeschätzt oder über Serverstatistiken relativ präzise ermittelt werden. Es handelt sich offensichtlich um eine Instanz des $P||C_{max}$-Problems, welches dem LIST-Algorithmus Schwierigkeiten bereiten kann, da dessen worst-case in dem gerade geschilderten Umfeld nicht unrealistisch ist. Man stelle sich dazu lediglich vor, es träfen sehr viele kleine Anfragen wie das Lesen der Angebote ein. Anschließend müßte jedoch ein extrem großer Job wie die Durchführung eines Bestellskripts bearbeitet werden.

Dementsprechend können wir unsere Beispielsequenz aus 2.6.4 als Zugriffsmuster oder fiktive Last auffassen, der die Webserver im Laufe eines bestimmten Zeitintervalls ausgestzt sind. Das Beispiel repräsentiert daher ein Lastbalancierungssystem mit fünf redundanten Webservern. Setzt man die Algorithmen von BARTAL ET AL. und KARGER ET AL. ein, um die Anfragen auf die Rechner zu verteilen, so könnte das Zugriffsmuster bzw. die fiktive Last unserer Beispielsequenz in 17 Zeiteinheiten durch den BFKV-ALgorithmus bzw. 18 Zeiteinheiten durch den zA-Algorithmus abgearbeitet werden.

3.2. Prozeßsteuerung und Workflow-Engines

Workflowsysteme gehören der allgemeinen Gruppe der Büroautomatisierungssysteme an. Im Gegensatz zu dokumentorientierten Systemen, handelt es sich bei Workflowmanagementsystemen um Systeme, die verschiedene, gut strukturierte, arbeitsteilige und zielgerichtete Abläufe unter eigener Kontrolle durchführen, koordinieren und überwachen. Ihr Wert für das Unternehmen liegt in der hohen Flexibilität insbesondere gegenüber Änderungen der Abläufe, der Organisation, herrschender Regeln und durchzuführender Aufgaben [VB96]. Wir wollen im folgenden sogenannte kontroll-und transportorientierte Systeme betrachten. Sie steuern die Aufgabenausführung, so daß der Anwender den genauen Ablauf des Prozesses nicht kennen muß. Desweiteren lenken sie den zur Bearbeitung erforderlichen Daten- und Dokumentenstrom. Seit der 2. Generation der Systeme benutzt man Vorgangsbeschreibungssprachen, um explizite Vorgangsmodelle zu erstellen, welche die Basis der Workflowsteuerung darstellen. Ablaufstrukturen werden mit Konstrukten für die Sequenz, Auswahl, Verzweigung, Vereinigung und Parallelität von Vorgangsaktivitäten beschrieben. Ergänzt man diese Konstrukte um Beschreibungselemente für die benötigten Ressourcen, so lassen sich aus der Ablaufstruktur effektive Workflows generieren. Die Vorgangsmodelle erlauben zum einen eine weitgehende Analyse und Effizienzoptimierung des Prozesses (Vorgangsadministration und Revision). Zum anderen werden, wie bereits erwähnt, konkrete Instanzen eines beschriebenen Prozeßtyps durch sogenannte Workflow-Engines erzeugt, gestartet, abgearbeitet und beendet. Betrachtet man Prozeßtypen, die besonders häufig instanziiert werden, beispielsweise die Auftragsbearbeitung, so werden sehr viele Aufgaben erzeugt, die alle

von dem gleichen Mitarbeiterpool bearbeitet werden. Diese Beobachtung liegt in der Arbeitsweise des Systems begründet. Das Workflowmanagementsystem greift mittels einer Datenbankanbindung auf die Mitarbeiterdaten zu und vergleicht die Anforderungsprofile der Aufgaben mit den Qualifikationsprofilen der Mitarbeiter. Es ergibt sich somit ein Pool an Mitarbeitern mit identischer Qualifikation, die immer wieder für den Prozeß der Auftragsbearbeitung herangezogen werden. In unserem Schedulingmodell reden wir daher von einem Pool identisch qualifizierter Prozessoren.

Aus einem Kundenauftrag werden entsprechend dem Vorgangsmodell Aufgaben generiert und vergeben. Erst nachdem eine Tätigkeit beendet ist, kann die nächste Aufgabe innerhalb des Prozeßmodells einem Mitarbeiter zugeteilt werden. Da gleichzeitig mehrere Prozesse aktiv sind, kann man nicht sagen, welche Aktivität aus welcher Prozeßinstanz als nächstes freigegeben wird. Man hat es also mit einer Online-Fragestellung zu tun, was durch die Tatsache untermauert wird, daß selbst die Ankunftszeiten der prozeß- und somit aufgabenauslösenden Kundenaufträge unbekannt sind. Die Jobbearbeitungsdauern können jedoch anhand von Simulationsergebnissen (Vorgangsanalyse) abgeschätzt werden, so daß entsprechende Daten verfügbar sind, sobald eine konkrete Aufgabe vom System freigegeben wurde.

Es ist offensichtlich, daß der Durchsatz des Systems maximiert bzw. die Durchlaufzeit der Prozesse minimiert wird, indem die Aufgabenlast möglichst gleichmäßig auf die Mitarbeiter der Auftragsbearbeitung verteilt wird. Wartezeiten und Transportzeiten sollten durch eine Analyse und Optimierung des Vorgangsmodells bereits im Vorfeld minimiert worden sein.

3.3. Optimierung von Logistikprozessen

Unter Logistik versteht man alle Prozesse, die der Bewegung und Lagerung von Gütern dienen. Sie umfassen die planerische und dispositive Begleitung der Güterströme in der Unternehmung [Sch95]. Im produzierenden Gewerbe können wir neben den übergreifenden Informations- und Kommunikationsprozessen zwischen Beschaffungs-, Produktions- und Vertriebslogistik unterscheiden, während diese Begriffe im Dienstleistungsbereich eher unpassend klingen. Entsprechend der Wertschöpfungskette nach PORTER [Por85] kann man hier zwischen Eingangs-, Operationen- und Ausgangslogistik unterscheiden.

Wir werden im folgenden ein Anwendungsbeispiel des GRAHAM-Problems aus dem Bereich Ausgangslogistik vorstellen. Eine effektive und effiziente Gestaltung solcher Prozesse ist beispielsweise für Handelsunternehmen außerordentlich wichtig, da hier ein wesentlicher Teil ihrer Wertschöpfung generiert wird.

Der hier vorgestellte Logistikprozeß ist Gegenstand eines Projekts der Firma *Aditor* [1], die ihren Sitz in Schwalbach/Saar hat und insbesondere Hard- und Softwareplanung sowie Regel- und Schaltanlagenbau für Logistikprozesse betreibt.

[1] `http://www.aditor.de`

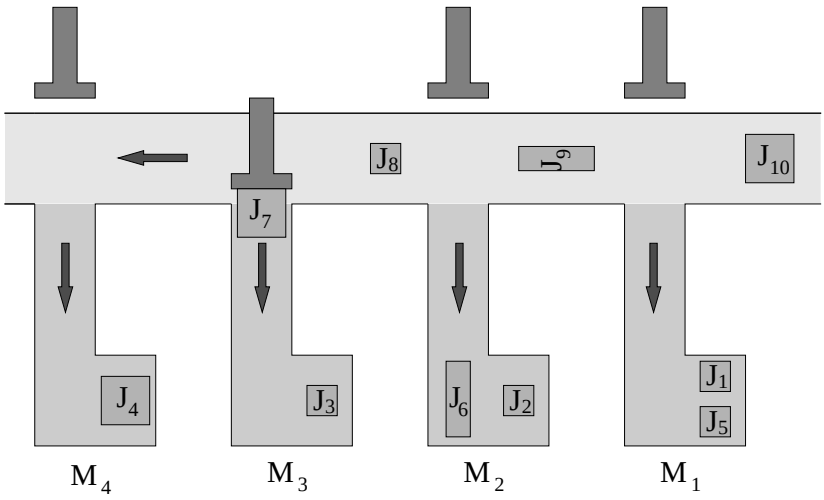


Abbildung 3.2.: **Logistikprozesse:** Verteilung von Paketen auf Verpackungsstationen

Das Problem, das wir nun betrachten werden, findet sich in der Bestellabwicklung vieler großer Versandunternehmen wieder. Für jeden Bestellauftrag wird ein Paketkarton auf einem Transportband, welches entlang eines Artikeldepots läuft, bewegt. Aus den Regalen dieses Depots entnehmen Mitarbeiter die gewünschten Waren und legen sie in das entsprechende Paket. Vor dem Versand müssen die Kartons noch gepackt werden, da die Waren ungeordnet in dem Paket verteilt liegen. Desweiteren ist es notwendig, Füllmaterial hinzuzufügen, um eine Beschädigung der Waren beim Transport zu vermeiden. Der Verpackungsprozeß kann nicht manuell ausgeführt werden, weswegen die Pakete auf Bearbeitungsstationen verteilt werden müssen. Die Kartons laufen daher über ein sehr schnelles Transportband, von dem zahlreiche Zulieferbänder zu den Verpackungsstationen abzweigen. Am zentralen Transportband befindet sich für jedes Zulieferband ein Schieber, der – durch eine Steuerlogik betätigt – ein vorbeikommendes Paket auf das zugehörige Transportband schieben kann. Die Verpackungszeit, welche im Minutenbereich liegt, ist abhängig von der Anzahl der Artikel in dem Karton, während die Verteilzeit als konstant angesehen werden kann, da sie für alle Zulieferbänder in der Größenordnung von Sekunden liegt. Um den Paketdurchsatz zu maximieren ist es notwendig die „Last" aller Verpackungsstationen möglichst gleich zu halten. Der Verteilungsprozeß ist in Graphik 3.2 veranschaulicht.

An dieser Stelle eröffnet sich uns ein interessantes Schedulingproblem. Wir betrachten die Bearbeitungsstationen bzw. Zulieferbänder als Prozessoren, denen sequentiell eintreffende Jobs, d.h. Kartons zugeteilt werden müssen. Da die Leistung aller Verpacker nur geringfügig variiert, handelt es sich um ein online-Schedulingproblem mit identisch qualifizierten Prozessoren. Die Zahl m bezeichnet dabei die Anzahl der besetzten Bearbeitungsstationen, die je nach Auftragslage schwanken kann. Betrachtet man die Bearbeitungszeit als die Zeit vom Eintreffen des Pakets auf dem Transportband bis zum Abschluß der Verpackungstätigkeit, so müßten wir von pro-

portionalen Prozessoren reden, da die Joblänge abhängig von der zugeteilten Bearbeitungsstation ist. Die Verteilungszeit ist offensichtlich bei einer Zuteilung zur ersten Station geringer als bei einer Einplanung auf der letzten Verpackungsstation, da das Paket in letzterem Fall weiter transportiert werden muß, bevor es mit dem Schieber dem Zulieferband zugeteilt werden kann. Auch wenn die Joblängen, d.h. die Verpackungsdauern bei Eingang des Kundenauftrags bekannt sind, handelt es sich dennoch um eine Online-Fragestellung. Man könnte zwar argumentieren, daß Kundenaufträge gesammelt und mit Hilfe eines Offline-Algorithmus auf die Paketstationen verteilt werden könnten, doch SHMOYS ET AL. haben in [SWW95] gezeigt, daß man mit einer solchen Vorgehensweise bestenfalls einen kompetitiven Faktor von $2 + \varepsilon$, mit $\varepsilon > 0$ erzielen kann. Wir können somit eine bessere Leistungsgarantie abgeben, wenn wir einen unserer Online-Algorithmen verwenden. Die Jobbearbeitungszeiten müssen somit erst bei Ankunft des Pakets auf dem Transportband, d.h. unmittelbar vor der Verteilung auf die Stationen, anhand der Begleitpapiere elektronisch ermittelt werden. Diese Beobachtungen erlauben also, das Problem mit einem Online-Algorithmus für $P||C_{max}$ zu lösen.

3.4. Personaleinsatz durch Personalleitstände

Ein weiteres betriebswirtschaftliches Konzept, bei dem wir auf die potentielle Anwendbarkeit von Online-$P||C_{max}$-Algorithmen gestoßen sind, ist das sogenannte Personalleitstandskonzept. Die Idee entstammt der Fertigung, in der elektronische Leitstände im Rahmen der dezentralen Fertigungssteuerung eingesetzt werden [Sch95]. Diese übernehmen Auftragsdaten als Eckwerte, ordnen autonom Arbeitsgänge den Ressourcen zu und terminieren diese. Mit Hilfe graphischer Oberflächen und GANTT-Diagrammen unterstützen sie einen menschlichen Disponenten, indem sie durch Schedulingalgorithmen eigenständig Pläne erzeugen können.

SCHOLZ [Sch94] beschreibt den Personalleitstand noch als Vision und erklärt, daß es bei zunehmender Verfügbarkeit relevanter Personaldaten – ähnlich wie beim Fertigungsleitstand – möglich sein wird, leitstandsartig die notwendigen Personaldaten aus dem Personalbereich abzurufen und in dispositiven Tätigkeien verarbeiten zu können.

Die *SAP AG* hat in ihrem *R/3-System* eine solche Funktionalität geschaffen. Sie benutzt selbst ein solches Konzept im Unternehmenssitz Walldorf, mit dem Ziel die Mitarbeiter der Rechnerbetriebsgruppe zu koordinieren. Auch hier wird durch Anforderungs-Qualifikationsabgleich des Systems ein Pool identisch qualifizierter Mitarbeiter ausgewählt. Die anfallenden Reparatur- und Instandhaltungsaufgaben können nun mit Hilfe von Online-$P||C_{max}$-Algorithmen auf die Mitarbeiter verteilt werden, so daß die Arbeitslast möglichst gleichmäßig über diese verteilt wird.

Auch an dieser Stelle müssen wir anmerken, daß in der Praxis häufig zusätzliche Prioritätsbeziehungen zu beachten sind, welche von unserem Modell nicht erfaßt werden.

A. Untere Schranken für die Planlänge des optimalen Online-Algorithmus'

Satz A.1 (Untere Schranken für $\mathrm{OPT}(\sigma_n)$). *Die folgenden Zahlenwerte stellen untere Schranken für die optimale Planlänge $\mathrm{OPT}(\sigma_n)$ dar:*

(i) Die Länge irgendeines Jobs J_t:

$$\mathrm{OPT}(\sigma_n) \geq p_t \tag{A.1}$$

(ii) Die Länge des größten Jobs $J_{(1)}$:

$$\mathrm{OPT}(\sigma_n) \geq p^{(1)} \tag{A.2}$$

(iii) Die optimale Planlänge bei Unterbrechbarkeit A_m^n:

$$\mathrm{OPT}(\sigma_n) \geq \frac{1}{m} \sum_{i=1}^{n} p_i \tag{A.3}$$

(iv) Zweimal die Länge des (m+1)größten Jobs $J^{(m+1)}$:

$$\mathrm{OPT}(\sigma_n) \geq 2p_{(m+1)} \tag{A.4}$$

Beweis. Die Beobachtungen (i) und (ii) sind offensichtlich, da kein Job unterbrochen werden darf und somit jede Verrichtung vollständig auf einer Maschine abgearbeitet werden muß. Die Planlänge entspricht somit mindestens der Länge des größten Jobs, d.h. aller Jobs in der Sequenz.

(iii) entspräche der optimalen Planlänge, falls die Jobs beliebig unterbrechbar wären. In diesem Fall ist eine perfekte Verteilung der Gesamtlast $\sum_{i=1}^{n} p_i$ auf die m Maschinen möglich, indem die mittlere Gesamtlast A_m^n jeder Maschine zugewiesen wird. Da die Jobs in unserem Modell jedoch nicht unterbrechbar sind, muß die erzielte Planlänge mindestens so groß sein, wie im Fall der Relaxierung.

Wir nehmen an, daß mindestens $m + 1$ Jobs einzuplanen sind. Gilt $m \leq n$, dann können wir die optimale Planlänge exakt bestimmen. Sie entspricht gerade der Länge des größten Jobs, da jeder Maschine genau ein Job zugewiesen wird. Betrachten wir im folgenden ausschließlich die $m + 1$ größten Jobs der Sequenz. Da diese auf m Maschinen verteilt werden, muß zumindest einer Maschine zwei Jobs zugeteilt werden. Somit ist die Last dieser Maschine mindestens so hoch wie die Summe der Bearbeitungszeiten der beiden kleinsten Jobs, nämlich $p_{(m)} + p_{(m+1)} \geq 2p_{(m+1)}$, was eine untere Schranke für die optimale Planlänge darstellt. □

B. Die Analyse des BFKV-Algorithmus'

Zunächst wollen wir einige Bezeichnungen und Parameter einführen, die uns in der Analyse hilfreich sein werden. Mit L_t bezeichnen wir die Menge der $\langle 0,445m\rangle$ niedrig belasteten Maschinen nach der Einplanung des Jobs J_t. Unter diesen ist S_t, die Menge der $\langle 0,14\langle 0,445m\rangle\rangle$ kleinsten Maschinen, besonders ausgezeichnet. Die hochbelasteten Maschinen aus H_t vervollständigen die Partition der Maschinenmenge.

Der Parameter $\delta m = \langle 0,445m\rangle$ gibt das Verhältnis von niedrig zu hoch belasteten Maschinen an, während τ den Anteil der Maschinen aus S_t an der Menge der niedrig belasteten Maschinen beschreibt. Aus $\delta m = \langle 0,445m\rangle$ folgt, daß δ im Intervall $[0,445 - 1/(2m), 0,445 + 1/(2m)]$ liegt. Diese Beobachtung ergibt sich unmittelbar aus einer RoundToNearest-Abschätzung. Analog kann τ in das Rundungsintervall $[0,14 - 1/(2\delta m), 0,14 + 1/(2\delta m)]$ eingeschlossen werden, da $\tau\delta m = \langle 0,14\langle 0,445m\rangle\rangle$ gilt. Da wir den kompetitiven Faktor von LIST, $c_{\text{LIST}} = 2 - \frac{1}{m}$, schlagen wollen und wir dabei einen kompetitiven Faktor von $2-\varepsilon$ anstreben, bietet es sich an ε als eine Konstante $\frac{1}{m_0}$ zu wählen, wobei m_0 im Laufe der Analyse festgelegt wird.

Satz B.1 (Kompetitiver Faktor). *Sei* $m \geq m_0 \geq 70$ *und* $\varepsilon := \frac{1}{m_0}$, *dann ist der* BFKV-*Algorithmus* $(2-\varepsilon)$*-kompetitiv.*

Um den Satz zu verifizieren, führen wir einen Widerspruchsbeweis durch. Wir nehmen also an, es gäbe eine minimale Jobfolge $\sigma_t = J_1, \ldots, J_t$, so daß

$$\text{BFKV}(\sigma_t) > (2-\varepsilon)\text{OPT}(\sigma_t) \tag{B.1}$$

gilt.

Lemma B.2. *Der letzte Job der Sequenz wird auf der kleinsten Maschine* M_1^{t-1} *eingeplant, welche anschließend die Planlänge definiert, d.h.*

$$\text{BFKV}(\sigma_t) = l_1^t = l_1^{t-1} + p_t$$

Beweis. Nehmen wir an J_t erhöht die Planlänge nicht. Dann gibt es eine echt kleinere Jobfolge, die ebenfalls Bedingung B.1 erfüllt, was unserer Wahl von σ_t widerspricht. Würde J_t nicht auf der kleinsten Maschine M_1^{t-1} eingeplant, dann müßte er der kleinsten Maschine aus H_{t-1}, nämlich M_{d+1}^{t-1} zugewiesen werden. Gemäß unserer Einplanungsregel gilt in diesem Fall $l_{d+1}^{t-1} + p_t \leq (2-\varepsilon)A_d^{t-1}$. Die Planlänge kann nun mit Hilfe einer unteren Schranke für den optimalen Algorithmus A.3 abgeschätzt werden.

$$\text{BFKV}(\sigma_t) = l_{d+1}^{t-1} + p_t \leq (2-\varepsilon)A_d^{t-1} \leq (2-\varepsilon)A_m^t \leq (2-\varepsilon)\text{OPT}(\sigma_t)$$

Hierbei haben wir verwendet, daß A_j monoton in der Zeit und Maschinenzahl wächst. Das Ergebnis stellt natürlich einen Widerspruch zu Annahme B.1 dar, wodurch unser Lemma bewiesen wäre. □

Wir benötigen zunächst weitere Definitionen.

Definition B.1 (Fette Jobs). Ein Job J heißt *fett*, wenn seine Länge mindestens $\frac{1}{2(1-\varepsilon)} l_1^{t-1}$ beträgt. □

In A.4 haben wir gesehen, daß zweimal die Länge des $(m+1)$größten Jobs eine untere Schranke für den optimalen Algorithmus darstellt. Wenn wir nun die Existenz von mindestens $m+1$ fetten Jobs in der Folge nachweisen können, so folgt unmittelbar, daß auch der $(m+1)$größte Job fett sein muß. Berücksichtigen wir weiter, daß der letzte Job auf der kleinsten Maschine eingeplant wird und diese anschließend die Planlänge definiert B.2, dann erhalten wir: $OPT(\sigma_t) \geq \max\left\{p_t, \frac{1}{1-\varepsilon} l_1^{t-1}\right\}$ Somit folgt der gesuchte Widerspruch zu Annahme B.1

$$\begin{aligned} BFKV(\sigma_t) &= l_1^{t-1} + p_t \\ &\leq (1-\varepsilon)OPT(\sigma_t) + OPT(\sigma_t) \\ &= (2-\varepsilon)OPT(\sigma_t) \end{aligned}$$

Um unseren Satz zu beweisen, bleibt also nachzuweisen, daß mindestens $m+1$ fette Jobs eingeplant werden. Wir werden im folgenden nachweisen, daß jeder Maschine im Laufe vor der Einplanung des letzten Jobs mindestens ein fetter Job zugewiesen wird und daß der letzte Job ebenfalls fett ist.

Definition B.2 (Hohe und niedrige Belastung). Maschinen, deren Last zum Zeitpunkt i den Wert $(1-\varepsilon)A_m^i$ noch nicht überschritten hat, gehören der Gruppe der *niedrig belasteten* Maschinen an. Alle anderen Maschinen heißen *hoch belastet*. □

Definition B.3 (Flache, halbflache und annähernd flache Pläne). Ein Plan ist zu einem gegebenen Zeitpunkt i *flach*, falls bereits die kleinste Maschine hoch belastet ist, d.h. $l_1^i > (1-\varepsilon)A_m^i$ gilt. Zum Zeitpunkt r, zu dem lediglich die Maschinen aus L_r eine Auslastung von $(1-\varepsilon)A_m^r$ noch nicht überschritten haben, bezeichnen wir den Plan als *halbflach*. Zum Zeitpunkt s mit $r \leq s < i$ sind nur noch die Maschinen aus S_s niedrig belastet, so daß wir bereits von einem *annähernd flachen* Plan sprechen können. □

Es ist aus der Einplanungsstrategie offensichtlich, daß es einen Zeitpunkt r geben muß, in dem das Ungleichgewicht der Lastverteilung zu einem halbflachen Plan führt. Das folgende Lemma zeigt, daß vor der Einplanung des letzten Jobs ein flacher Plan vorliegt.

Lemma B.3. *Zum Zeitpunkt* $t-1$*, d.h. bevor der letzte Job eingeplant wird, ist der vom* BFKV*-Algorithmus erzeugte Plan flach.*

Beweis. Nehmen wir an, daß $l_1^{t-1} \leq (1-\varepsilon)A_m^{t-1}$ gilt, d.h. daß der erzeugte Plan nicht flach ist. In Lemma B.2 haben wir gesehen, daß M_1^{t-1} die Planlänge bestimmt, nachdem der lezte Job auf ihr eingeplant wurde. Außerdem wissen wir nach A.1, daß die Länge des letzten Jobs eine untere Schranke für $OPT(\sigma_t)$ darstellt. Es gilt also:

$$\begin{aligned} BFKV(\sigma_t) &= l_1^t = l_1^{t-1} + p_t \\ &\leq (1-\varepsilon)A_m^{t-1} + p_t \\ &\leq (1-\varepsilon)OPT(\sigma_t) + OPT(\sigma_t) \\ &= (2-\varepsilon)OPT(\sigma_t), \end{aligned}$$

was Annahme B.1 widerspricht. □

Korollar B.4. Zum Zeitpunkt $t-1$ sind alle Maschinen hoch belastet.

Da zum Zeitpunkt r noch keine Maschine aus L_r eine Auslastung von über $(1-\varepsilon)A_m^{t-1}$ erreicht, zum Zeitpunkt $t-1$ jedoch bereits alle, muß es einen Zeitpunkt s mit $r \leq s < t-1$ geben, zu dem der erzeugte Plan annähernd flach ist.

Durch diese Beobachtung können wir drei Gruppen von Maschinen unterscheiden, die eine Partition der gesamten Maschinenmenge darstellen. Graphik B.1 aus [Sch98] zeigt die gewünschte Partitionierung.

(i) Die niedrig belasteten Maschinen zum Zeitpunkt s: S_s.

(ii) Die niedrig belasteten Maschinen zum Zeitpunkt r, die aber zum Zeitpunkt s bereits hoch belastet sind: $L_r \setminus S_s$.

(iii) Die hoch belasteten Maschinen zum Zeitpunkt r: H_r.

Für jede Gruppe werden wir im folgenden zeigen. daß alle ihr angehörenden Maschinen im Laufe der Einplanung einen fetten Job erhalten. Da die Vereinigung dieser disjunkten Gruppen die gesamte Maschinenmenge ergibt, ist die Aussage äquivalent zu der Behauptung, daß jeder Maschine ein fetter Job zugewiesen wird.

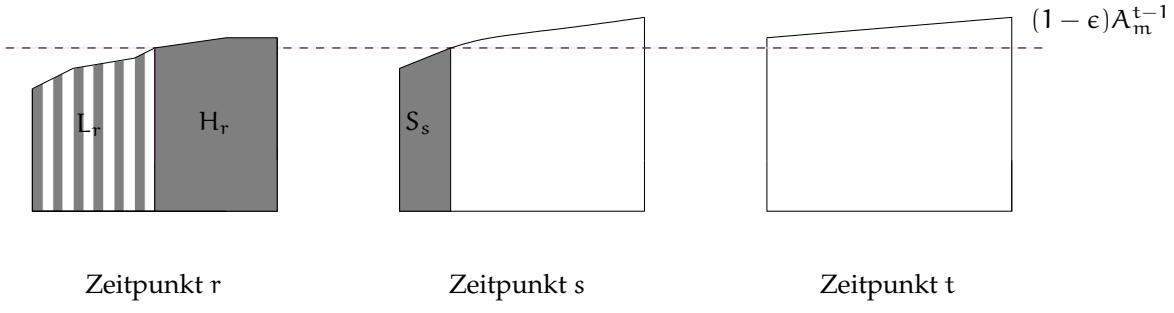


Abbildung B.1.: Der Plan zum Zeitpunkt r, s und t

Die Analyse von Gruppe (i)

Lemma B.5. *Im Zeitintervall (s,t-1) wird jeder Maschine in S_s ein Job mit Mindestlänge $\beta := \beta_1 A_m^{t-1} + \beta_2 A_S^s$ zugewiesen, wobei A_S^s die mittlere Last der Maschinen aus S_s bezeichnet und $\beta_1 := \left(2 - \varepsilon + \frac{\delta}{1-\delta}\right)(1-\tau)(1-\varepsilon) - \frac{1}{1-\delta}$ sowie $\beta_2 := \left(2 - \varepsilon + \frac{\delta}{1-\delta}\right)\tau$ eindeutig durch ε, δ und τ bestimmt sind.*

Beweis. Sei M_k, $1 \leq k \leq \tau\delta m$ eine beliebige Maschine aus S_s und J_{l+1} der erste Job nach s, der auf M_k eingeplant wird. Einen solchen Job muß es geben, da M_k zum Zeitpunkt $t-1$ hoch belastet ist. Eine niedrig belastete Maschine kann gemäß unserer Einplanungsregel zum Zeitpunkt l jedoch nur dann einen Job erhalten, wenn sie die kleinste Maschine zu diesem Zeitpunkt ist und die folgende Bedingung erfüllt ist:

$$l_{d+1}^l + p_{l+1} > (2-\varepsilon)A_d^l \iff p_{l+1} > (2-\varepsilon)A_d^l - l_{d+1}^l \tag{B.2}$$

Der Parameter δ kann als Gewicht der niedrig belasteten Maschinen bei der Bildung der mittleren Last verstanden werden, so daß wir aufgrund der Monotonieeigenschaften von A_m die folgende Abschätzung erhalten:

$$A_m^{t-1} \geq A_{d+1}^l \geq \delta A_d^l + (1-\delta) l_{d+1}^l$$

Lösen wir die Ungleichung nach l_{d+1}^l auf und setzen sie in B.2 ein, so ergibt sich:

$$\begin{aligned} p_{l+1} &> (2-\varepsilon)A_d^l - \left(\frac{A_m^{t-1} - \delta A_d^l}{1-\delta}\right) \\ &= \left(2 - \varepsilon + \frac{\delta}{1-\delta}\right) A_d^l - \frac{1}{1-\delta} A_m^{t-1} \end{aligned} \tag{B.3}$$

Um A_d^l, d.h. die mittlere Last der niedrig ausgelasteten Maschinen zum Zeitpunkt l abzuschätzen, können wir verwenden, daß die $\delta m - \tau\delta m$ Maschinen aus L_r, die nicht in S_s sind, bereits zum Zeitpunkt s hochbelastet sind. Die Last der einzelnen Maschinen dieser Gruppe beträgt daher über $(1-\varepsilon)A_m^{t-1}$. Die Gesamtlast von S_s kann dagegen durch $\tau\delta m A_S^s$ abgeschätzt werden. Wir erhalten somit:

$$\begin{aligned} A_d^l &\geq \frac{\tau\delta A_S^s + (\delta m - \tau\delta m)(1-\varepsilon)A_m^{t-1}}{\delta m} \\ &= \tau A_S^s + (1-\tau)(1-\varepsilon)A_m^{t-1} \end{aligned}$$

Kombinieren wir dieses Ergebnis mit B.3, so ergibt sich:

$$\begin{aligned} p_{l+1} &> \left(2 - \varepsilon + \frac{\delta}{1-\delta}\right)\left(\tau A_S^s + (1-\tau)(1-\varepsilon)A_m^{t-1}\right) - \frac{1}{1-\delta} A_m^{t-1} \\ &= \left[\left(2 - \varepsilon + \frac{\delta}{1-\delta}\right)(1-\tau)(1-\varepsilon) - \frac{1}{1-\delta}\right] A_m^{t-1} + \left(2 - \varepsilon + \frac{\delta}{1-\delta}\right)\tau A_S^s \\ &= \beta_1 A_m^{t-1} + \beta_2 A_S^s \end{aligned}$$

□

Korollar B.6. Im Zeitintervall $(s, t-1)$ wird jeder Maschine in S_s ein fetter Job zugewiesen.

Beweis. Man kann leicht nachrechnen, daß für $m_0 = 70$ und $m \geq m_0$ sowie für δ und τ innerhalb des Rundungsintervalls

$$\beta_1 \geq \frac{1}{2(1-\varepsilon)} \qquad \text{und} \qquad \beta_2 \geq 0 \tag{B.4}$$

gilt.
Da M_1^{t-1} die kleinste Maschine ist, folgt aus $l_1^{t-1} \leq A_m^{t-1}$ die gesuchte Abschätzung: $p_{l+1} > \frac{1}{2(1-\varepsilon)} A_m^{t-1} \geq \frac{1}{2(1-\varepsilon)} l_1^{t-1}$ □

Die Analyse von Gruppe (ii)

Zunächst wollen wir die mittlere Last der Maschinen aus L_r bzw. S_s abschätzen.

Lemma B.7. $A_S^s \leq \alpha$, *mit* $\alpha := \left(\frac{1+\beta_1}{1+\beta_2} + \varepsilon \frac{1-\tau\delta}{\tau\delta(1+\beta_2)}\right)$

Beweis. In Lemma B.5 haben wir gesehen, daß im Zeitintervall $(s, t-1)$ jeder Maschine aus S_s ein Job der Mindestlänge β zugewiesen wird. Im Zeitpunkt $t-1$ sind alle Maschinen hoch belastet B.4, so daß gilt:

$$\begin{aligned} A_m^{t-1} &\geq (1-\tau\delta)(1-\varepsilon)A_m^{t-1} + \tau\delta(A_S^s + \beta_1 A_m^{t-1} + \beta_2 A_S^s) \\ \iff A_S^s &\leq \left(\frac{1+\beta_1}{1+\beta_2} + \varepsilon\frac{1-\tau\delta}{\tau\delta(1+\beta_2)}\right) A_m^{t-1} \\ &= \alpha A_m^{t-1} \end{aligned}$$

□

Lemma B.8. *Für eine beliebige, zum Zeitpunkt* r *niedrig ausgelastete Maschine* M_k^r*, die zum Zeitpunkt* $j \in [r, s]$ *eine Last* $> \alpha A_m^{t-1}$ *erreicht, gilt, daß ihre Last im Zeitfenster* $[j, s]$ *unverändert belibt, d.h.*

$$\alpha A_m^{t-1} < l_{M_k^r}^i \leq (1-\varepsilon)\alpha A_m^{t-1} \quad i \in [j, s]$$

Beweis. Betrachten wir eine beliebige Maschine M_k^r aus L_r, die zum Zeitpunkt $j \in [r, s]$ die Last $l_{M_k^r}^j > \alpha A_m^{t-1}$ erreicht. Da die Last der kleinsten Maschine zum Zeitpunkt s nach B.7 höchstens $A_S^s \leq \alpha A_m^{t-1}$ beträgt, kann M_k^r frühestens zum Zeitpunkt $s+1$ die kleinste Maschine sein. Dies ist aber nach unserer Einplanungsregel die Voraussetzung, um als niedrig ausgelastete Maschine einen Job zu empfangen. Sie behält somit ihre Last im Zeitfenster $[j, s]$ bei. □

Korollar B.9. Jede Maschine aus L_r, die im Zeitfenster $[r, s]$ eine Last aus dem Intervall $(\alpha A_m^{t-1}, (1-\varepsilon)A_m^{t-1}]$ erreicht, ist in S_s enthalten.

Diese Beobachtung erlaubt uns, die Maschinen in solche aus S_s und solche, deren Last höchstens αA_m^{t-1} beträgt, zu zerlegen. Berücksichtigen wir die zeitliche Monotonie von A_j sowie Lemma B.7, so liefert diese Zerlegung:

$$\begin{aligned} A_L^r &\leq \tau A_S^r + (1-\tau)\alpha A_m^{t-1} \\ &\leq \tau A_S^s + (1-\tau)\alpha A_m^{t-1} \\ &\leq \tau\alpha A_m^{t-1} + (1-\tau)\alpha A_m^{t-1} = \alpha A_m^{t-1} \end{aligned} \tag{B.5}$$

Lemma B.10. *Jeder Maschine in* $L_r \setminus S_s$ *wird im Zeitfenster* $[r, s]$ *ein fetter Job zugewiesen.*

Beweis. In Korollar B.9 haben wir gesehen, daß alle Maschinen aus L_r, die im Zeitfenster $[r, s]$ eine Last aus dem Intervall $(\alpha A_m^{t-1}, (1-\varepsilon)A_m^{t-1}]$ erreichen, in S_s enthalten sind. Aus der Gruppendefinition folgt daher, daß keine Maschine aus Gruppe (ii) im Zeitintervall $[r, s]$ eine Last innerhalb des Lastintervalls annehmen darf. Da diese Maschinen zum Zeitpunkt s jedoch hoch belastet sind, muß auf jeder von ihnen ein Job eingeplant worden sein, durch die Maschine das Lastintervall überbrückt hat. Die Länge dieses Jobs beträgt daher:

$$\begin{aligned} p &\geq (1-\varepsilon)A_m^{t-1} - \alpha A_m^{t-1} \\ &= (1-\varepsilon-\alpha)A_m^{t-1} \\ &\geq \frac{1}{2(1-\varepsilon)} A_m^{t-1} \\ &\geq \frac{1}{2(1-\varepsilon)} l_1^{t-1} \end{aligned}$$

In der vorletzten Ungleichung haben wir verwendet, daß die folgende Bedingung für unsere Parameterwahl erfüllt ist:

$$\alpha \leq \left(1 - \varepsilon - \frac{1}{2(1-\varepsilon)}\right) \tag{B.6}$$

□

Die Analyse von Gruppe (iii)

Lemma B.11. *Auf jeder Maschine in* H_r *wird im Zeitintervall* $[r, s]$ *ein fetter Job eingeplant.*

Beweis. Wir betrachten den zuletzt auf der Maschine M_k^r, $d+1 \leq k \leq m$ eingeplanten Job J_{l+1}. Da die Maschinen in H_r hoch belastet sind, gilt aufgrund von B.5 $l_k^r > (1-\varepsilon)A_m^{t-1} \geq (1-\varepsilon)\frac{A_L^r}{\alpha}$. Wir können weiter verwenden, daß aufgrund unserer Wahl der Parameter $\alpha \leq 1 - \varepsilon - \frac{1}{2(1-\varepsilon)} \leq \frac{1-\varepsilon}{2-\varepsilon}$ erfüllt ist, so daß

$$l_k^r > (2-\varepsilon)A_L^r \tag{B.7}$$

gilt.
Entsprechend unserer Einplanungsregel gibt es zwei Möglichkeiten, wie J_{l+1} auf M_k^r

eingeplant werden konnte. Nehmen wir zunächst an, M_k^r sei die kleinste Maschine in H_l gewesen als J_{l+1} eintraf, d.h. $M_k^r = M_{d+1}^l$. Dann folgt aus der Einplanungsregel und der zeitlichen Monotonie von A_L:

$$l_{d+1}^{l+1} = l_{d+1}^l + p_{l+1} \leq (2-\varepsilon)A_d^l = (2-\varepsilon)A_L^l \leq (2-\varepsilon)A_L^r$$

Da J_{l+1} der zuletzt eingeplante Job auf M_k^r ist, gilt $l_k^r = l_{d+1}^{l+1} \leq (2-\varepsilon)A_L^r$, was einen Widerspruch zu B.7 darstellt. Wir können also davon ausgehen, daß M_k^r die kleinste Maschine war, als J_{l+1} eintraf. Für ihre Last gilt daher $l_k^r = l_1^l + p_{l+1}$. Wir können nun die Länge von J_{l+1} abschätzen, wobei wir verwenden, daß die Last der kleinsten Maschine höchsten so hoch ist wie die mittlere Last aller niedrig belasteten Maschinen aus L_l, d.h. $l_1^l \leq A_L^l \leq A_L^r$:

$$\begin{aligned} p_{l+1} &= l_k^r - l_1^l \\ &> (1-\varepsilon)A_m^{t-1} - A_L^r \\ &\geq (1-\varepsilon)A_m^{t-1} - \alpha A_m^{t-1} = (1-\varepsilon-\alpha)A_m^{t-1} \\ &\geq \frac{1}{2(1-\varepsilon)}A_m^{t-1} \geq \frac{1}{2(1-\varepsilon)}l_m^{t-1} \end{aligned}$$

Die letzte Abschätzung folgt aus B.6 und liefert unsere Behauptung. □

Wir haben somit nachgewiesen, daß bevor der letzte Job eintrifft, auf jeder Maschine mindestens ein fetter Job eingeplant wird. Das folgende Lemma vervollständigt unseren Beweis, indem es zeigt, daß auch der letzte Job fett ist. Wir haben also insgesamt $m+1$ fette Jobs identifiziert.

Lemma B.12 (Länge des letzten Jobs). *Der letzte Job der Jobsequenz J_t ist fett.*

Beweis. Wir betrachten zunächst den Fall $p_t \leq (1-\varepsilon)l_1^{t-1}$. Da J_t nach Lemma B.2 auf der kleinsten Maschine eingeplant wird, welche anschließend die Planlänge definiert und da $A_m^{t-1} \leq A_m^t$ eine untere Schranke für $OPT(\sigma_t)$ darstellt, können wir einen Widerspruch zur ursprünglichen Annahme B.1 herleiten:

$$\begin{aligned} BFKV(\sigma_t) &= l_1^{t-1} + p_t \\ &\leq l_1^{t-1} + (1-\varepsilon)l_1^{t-1} = (2-\varepsilon)l_1^{t-1} \\ &\leq (2-\varepsilon)A_m^t \\ &\leq (2-\varepsilon)OPT(\sigma_t) \end{aligned}$$

Wir dürfen daher annehmen:

$$p_t > (1-\varepsilon)l_1^{t-1} \geq \left(\alpha + \frac{1}{2(1-\varepsilon)}\right)l_1^{t-1} \geq \frac{1}{2(1-\varepsilon)}l_1 t - 1$$

Die letzten beiden Ungleichungen ergeben sich aufgrund von Bedingung B.6, die wir für α gefordert haben. □

C. Die Analyse des $CHASM_\alpha$-Algorithmus'

Satz C.1 (Kompetitiver Faktor). *Der* $CHASM_\alpha$*-Algorithmus ist* $\alpha = 1,945$*-kompetitiv.*

Wir wollen diesen Satz mittels Induktion über die Länge der Jobfolge, welche von dem Algorithmus verarbeitet wird, beweisen. Der Induktionsanfang ist trivial, da im Falle eines einzelnen Jobs jede Einplanungsregel, welche den Job einer Maschine zuweist, die optimale Planlänge liefert. Sei $\sigma_n = J_1, J_2, ..., J_n$ eine beliebige Jobfolge. Entsprechend der Induktionshypothese nehmen wir an, daß

$$CHASM_\alpha(\sigma_{t-1}) \leq \alpha OPT(\sigma_{t-1})$$

und müssen nun nachweisen, daß

$$CHASM_\alpha(\sigma_t) \leq \alpha OPT(\sigma_t)$$

gilt. Um die Notation etwas zu vereinfachen, können wir o.B.d.A. die Jobs in der Folge σ_t normalisieren, so daß die Summen ihrer Längen gerade m ergibt. Dies hat zur Folge, daß die mittlere Last A_m^t über alle Maschinen am Ende der Einplanung eins ist.

Zunächst wollen wir einige Fälle ausschließen, in denen unsere Behauptung offensichtlich ist. Aufgrund der Normalisierung erhalten wir nach A.3 die folgende untere Schranke für den optimalen Algorithmus: $OPT(\sigma_t) \geq \frac{1}{m}\Sigma_{i=1}^{t} p_t = 1$. Falls $CHASM_\alpha(\sigma_t) \leq \alpha$ gilt, ergibt sich somit $CHASM_\alpha(\sigma_t) \leq \alpha OPT(\sigma_t)$.

Verwenden wir in Fall $CHASM_\alpha(\sigma_t) \leq \alpha p_t$, daß $OPT(\sigma_t) \geq \max p_i \geq p_t$, so folgt unsere Behauptung unmittelbar.

Für Fall $CHASM_\alpha(\sigma_t) \neq l_1^{t-1} + p_t$ ist der Induktionsschritt etwas komplizierter. Nehmen wir zunächst an, der letzte Job J_t würde nicht auf der kleinsten Maschine, sondern M_k^{t-1}, mit $k < 1$ eingeplant werden. Dann können die folgenden Fälle unterschieden werden.

1. Nach Einplanung von J_t definiert Maschine M_k^{t-1} die Planlänge nicht.
   Es gilt:

$$CHASM_\alpha(\sigma_t) = CHASM_\alpha(\sigma_{t-1}) \leq \alpha OPT(\sigma_{t-1}) \leq \alpha OPT(\sigma_t)$$

2. Nach Einplanung von J_t definiert Maschine M_k^{t-1} die Planlänge.
   In diesem Fall läßt sich die Planlänge mittels der Einplanungsregel sowie der unteren Schranke für OPT A.3 abschätzen:

$$\mathrm{CHASM}_\alpha(\sigma_t) = l_k^{t-1} + p_t \leq \alpha A_{k-1}^{t-1} \leq \alpha A_m^t \leq \alpha \mathrm{OPT}(\sigma_t)$$

Planen wir dagegen J_t auf der kleinsten Maschine M_1^{t-1} ein, so können wir annehmen, daß diese die Planlänge bestimmt. Wäre dem nicht so, könnten wir den Induktionsschritt wie in Fall 1 für $k > 1$ durchführen und hätten unsere Behauptung bewiesen.

Für den weiteren Beweis können wir also von den folgenden Annahmen über die Planlänge des CHASM_α-Algorithmus ausgehen:

$$\mathrm{CHASM}_\alpha(\sigma_t) \geq \alpha \tag{C.1}$$
$$\mathrm{CHASM}_\alpha(\sigma_t) \geq \alpha p_t \tag{C.2}$$
$$\mathrm{CHASM}_\alpha(\sigma_t) = l_1^{t-1} + p_t \tag{C.3}$$

Wir werden nun zeigen, daß es mindestens $m + 1$ Jobs einer gewissen Mindestlänge b geben muß, so daß für die Länge des $(m+1)$größten Jobs $J_{(m+1)}$ gilt: $p_{(m+1)} \geq b$. In A.4 haben wir gesehen, daß der optimale Algorithmus zwei der $(m+1)$größten Jobs auf eine Maschine packen muß, so daß sich als untere Schranke $\mathrm{OPT}(\sigma_t) \geq 2p_{(m+1)} \geq 2b$ ergibt. Können wir nun nachweisen, daß unser Algorithmus einen Plan von Maximallänge $\alpha 2b$ erzeugt, so erhalten wir

$$\mathrm{CHASM}_\alpha(\sigma_t) \leq \alpha 2b \leq \alpha \mathrm{OPT}(\sigma_t)$$

und unsere Behauptung ist bewiesen.

Zunächst benötigen wir einige Definitionen, die uns die Analyse etwas vereinfachen werden.

Definition C.1. Sei $\varepsilon := 2 - \alpha$. Wir definieren β über die Eigenschaft $1 - \beta = l_1^{t-1}$. Mit b bezeichnen wir den Wert $\frac{1}{2}\frac{1-\beta}{1-\varepsilon}$ und mit a den Wert $1 - \beta - b$. □

Definition C.2 (Fette Jobs). Ein Job J heißt *fett*, wenn seine Länge mindestens b beträgt. □

Im ersten Schritt werden wir also eine obere Schranke für die maximale Planlänge des CHASM_α-Algorithmus in Abhängigkeit von α und b bestimmen.

Lemma C.2 (Maximale Planlänge des CHASM_α-Algorithmus). *Die Planlänge unseres Algorithmus beträgt maximal $\alpha 2b$, d.h.*

$$\mathrm{CHASM}_\alpha(\sigma_t) \leq \alpha 2b \tag{C.4}$$

Beweis. Mit Hilfe von C.2 und C.3 können wir die Länge des letzten Jobs abschätzen:

$$\alpha p_t \leq \mathrm{CHASM}_\alpha(\sigma_t) = l_1^{t-1} + p_t \qquad \Longleftrightarrow \qquad p_t \leq \frac{l_1^{t-1}}{\alpha - 1} = \frac{l_1^{t-1}}{1 - \varepsilon} \tag{C.5}$$

Aufgrund von C.3 und C.5 gilt somit:

$$\begin{aligned} \mathrm{CHASM}_\alpha(\sigma_t) &= l_1^{t-1} + p_t \\ &\leq l_1^{t-1} + \frac{l_1^{t-1}}{1-\varepsilon} = l_1^{t-1}\left(\frac{2-\varepsilon}{1-\varepsilon}\right) \\ &= (1-\beta)\frac{\alpha}{1-\varepsilon} = \alpha 2b \end{aligned}$$

□

Um den Beweis des Satzes zu vervollständigen, müssen wir nun zeigen, daß selbst der $(m+1)$größte Job $J_{(m+1)}$ noch eine Länge von mindestens b besitzt.

Definition C.3 (Hohe und niedrige Belastung). Eine Maschine heißt *hoch belastet*, wenn ihre Last mindestens $l_1^{t-1} = 1-\beta$ beträgt, sonst sprechen wir von *niedriger Belastung*. Einen Job nennen wir *kritisch*, wenn er eine Maschine mit niedriger Belastung in den Zustand hoher Belastung überführt. Mit s bezeichnen wir die aktuelle Anzahl an Maschinen mit niedriger Belastung. □

Da M_1^{t-1} die am niedrigsten belastete Maschine zum Zeitpunkt $t-1$ ist, sind alle Maschinen per Definition zu diesem Zeitpunkt hochbelastet. Somit muß es m kritische Jobs in der Jobfolge σ_t geben. Können wir nun zeigen daß J_t, d.h. der letzte Job, ein fetter Job ist und daß alle kritischen Jobs ebenfalls diese Eigenschaft erfüllen, dann haben wir $m+1$ Jobs mit Mindestlänge b gefunden. Daraus folgt natürlich unmittelbar, daß der $(m+1)$größte Job fett ist. Betrachten wir also zunächst den letzten Job J_t.

Lemma C.3 (Länge des letzten Jobs). *Der letzte Job J_t ist fett.*

Beweis. J_t wird der aktuell kleinsten Maschine zugewiesen, wobei deren Last anschließend die Planlänge definiert. Aufgrund von C.1 wissen wir, daß die Einplanung von J_t die Last dieser Maschine von $l_1^{t-1} = 1-\beta$ auf $l_1^t = \mathrm{CHASM}_\alpha(\sigma_t) > \alpha$ erhöht. Somit können wir die Joblänge von J_t abschätzen:

$$p_t = l_1^t - l_1^{t-1} \geq \alpha - (1-\beta) = 1-\varepsilon+\beta \tag{C.6}$$

Da M_1^{t-1} die Maschine mit geringster Last ist und die mittlere Last über alle Maschinen in der Zeit t monoton wächst, gilt:

$$1-\beta = l_1^{t-1} \leq A_m^{t-1} \leq A_m^t = 1,$$

woraus $\beta \geq 0$ folgt.
Dies erlaubt uns nun, p_t und b abzuschätzen:

$$p_t \geq 1-\varepsilon+\beta \geq 1-\varepsilon \qquad\qquad b = \frac{1}{2}\frac{1-\beta}{1-\varepsilon} \leq \frac{1}{2}\frac{1}{1-\varepsilon}$$

Somit ist J_t also fett, falls $1-\varepsilon \geq \frac{1}{2}\frac{1}{1-\varepsilon}$ gilt. Diese Bedingung ist erfüllt falls

$$\varepsilon_1 \leq 1-\frac{1}{\sqrt{2}} \iff \alpha_1 \geq 1+\frac{1}{\sqrt{2}} \qquad \text{bzw.} \qquad \varepsilon_2 \leq 1+\frac{1}{\sqrt{2}} \iff \alpha_2 \leq 1-\frac{1}{\sqrt{2}}$$

gilt. Da aufgrund existierender unterer Schranken für den kompetitiven Faktor $\alpha > 1$ sein muß, kommt ausschließlich die erste Bedingung in Frage. □

Wir können also festhalten, daß der Job J_t fett ist, falls $\alpha \geq 1+\frac{1}{\sqrt{2}}$ erfüllt ist.

Nun verbleibt zu zeigen, daß alle m kritischen Jobs fett sind. Für die Analyse des Algorithmus wird es sich als vorteilhaft herausstellen, das betrachtete Zeitintervall $(0, t-1]$ in zwei Phasen zu zerlegen. In Phase 1 ist zu jedem Zeitpunkt $\bar{t}$ die mittlere Last über alle niedrig belasteten Maschinen $A_s^{\bar{t}}$ kleiner als a, während in der zweiten Phase $A_s^{\bar{t}} \geq a$ gilt. Wir wollen zunächst zeigen, daß eine solche Zerlegung möglich ist.

Lemma C.4 (Phasentrennung). *Wenn die mittlere Last über alle niedrig belasteten Maschinen zu einem bestimmten Zeitpunkt $\bar{t}$ kleiner als a ist, dann ist diese Eigenschaft auch für jeden früheren Zeitpunkt erfüllt, d.h. falls $A_s^{\bar{t}} < a$, dann gilt auch $A_s^{\bar{t}-1} < a$*

Beweis. Wir wollen zunächst unterscheiden, ob $J_{\bar{t}}$ ein kritischer Job ist oder nicht.

1. $J_{\bar{t}}$ ist kein kritischer Job.
   Die Anzahl der niedrig belasteten Maschinen ändert sich nicht. Da jedoch ein weiterer Job eingeplant wurde, gilt: $A_s^{\bar{t}-1} \leq A_s^{\bar{t}} < a$

2. $J_{\bar{t}}$ ist ein kritischer Job.
   Wir wollen zunächst zeigen, daß $J_{\bar{t}}$ der kleinsten Maschine zugewiesen werden muß.
   Nehmen wir an, $J_{\bar{t}}$ wäre auf einer anderen Maschine $M_k^{\bar{t}-1}$ eingeplant worden. Da $J_{\bar{t}}$ kritisch ist und die Einplanungsregel erfüllt sein muß, gilt die folgende Beziehung: $1-\beta \leq l_k^{\bar{t}-1} + p_{\bar{t}} \leq \alpha A_{k-1}^{\bar{t}-1}$, woraus sich

   $$A_{k-1}^{\bar{t}-1} \geq \frac{1-\beta}{\alpha} \tag{C.7}$$

   ableiten läßt.

   Die $k-1$ kleinsten Maschinen sind von der Einplanung nicht betroffen, so daß $A_{k-1}^{\bar{t}-1} = A_{k-1}^{\bar{t}} < a$ gilt. Dieser Ausdruck kann weiter abgeschätzt werden:

   $$A_{k-1}^{\bar{t}-1} < a = 1-\beta-b = 1-\beta-\frac{1}{2}\frac{1-\beta}{\alpha-1} \leq \frac{1-\beta}{\alpha},$$

   falls $\alpha - \frac{\alpha}{2(\alpha-1)} \leq 1 \iff \left(\alpha \geq \frac{1}{2} \text{ und } \alpha \leq 2\right)$
   Wählen wir α so, daß $1+\frac{1}{\sqrt{2}} \leq \alpha \leq 2$ erfüllt ist, dann haben wir einen Widerspruch zu Aussage C.7 hergeleitet. Unsere Annahme war also falsch, was bedeutet, daß $J_{\bar{t}}$ auf der kleinsten Maschine $M_1^{\bar{t}-1}$ eingeplant wird. Da es sich

um ein kritischen Job handelt, verläßt sie die Menge der niedrig ausgelasteten Maschinen, so daß die mittlere Last der verbleibenden Maschinen dieser Kategorie zunehmen muß. Somit gilt: $A_s^{\bar{t}-1} \leq A_s^{\bar{t}}$.

□

Wir werden im folgenden die beiden Phasen getrennt betrachten und nachweisen, daß alle kritischen Jobs, die in ihnen eingeplant werden, fett sind.

Die Analyse von Phase 1

Lemma C.5. *Alle in Phase 1 eingeplanten, kritischen Jobs sind fett.*

Beweis. Betrachten wir einen kritischen Job $J_{\bar{t}}$, der in Phase 1 einzuplanen ist. In C.3 haben wir gesehen, daß ein solcher Job der kleinsten Maschine $M_1^{\bar{t}-1}$ zugewiesen wird. Um die Länge des Jobs abschätzen zu können, vergleichen wir die Last der Maschine vor der Einplanung, $l_1^{\bar{t}-1} \leq A_s^{\bar{t}-1} < a$, mit der nach der Einplanung: $l_1^{\bar{t}} \geq 1 - \beta$. Somit erhalten wir für die Länge von $J_{\bar{t}}$:

$$p_{\bar{t}} = l_1^{\bar{t}} - l_1^{\bar{t}-1} \geq 1 - \beta - a = b,$$

qas zeigt, daß $J_{\bar{t}}$ fett ist. □

Die Analyse von Phase 2

Seien $T_s, \ldots, T_1$ die kritischen Jobs dieser Phase in der Reihenfolge ihres Eintreffens, d.h. T_s ist der erste und T_1 der letzte Job mit dieser Eigenschaft. Mit s bezeichnen wir die Anzahl niedrig belasteter Maschinen am Ende der ersten Phase. Um zu zeigen, daß die Jobs fett sind, berechnen wir mit Hilfe eines Linearen Programms für jeden Job T_i eine untere Schranken für seine Länge q_i, $1 \leq i \leq s$. Um das Programm für T_i lösen zu können, müssen wir bereits über untere Schranken für $q_{i-1}, \ldots, q_1$ verfügen. Natürlich kennen wir s, die Anzahl der kritischen Jobs dieser Phase, nicht im vorhinein. Dieses Problem läßt sich jedoch umgehen, indem wir sukzessive die Linearen Programme für die einzelnen kritischen Jobs lösen, beginnend mit dem für T_1, bis wir ein Programm bearbeiten, nämlich das für $T_{\tilde{s}}$, welches keine zulässige Lösung besitzt. Wir werden sehen, daß man aus dieser Tatsache schließen kann, daß der Anfang von Phase 2 überschritten worden ist, d.h. daß es maximal $\tilde{s} \geq s$ kritische Jobs in der zweiten Phase geben kann.

Wir wollen nun das Lineare Programm für den Job T_i mit Länge q_i vorstellen. Da alle auftretenden Größen sich auf den Zeitpunkt vor Einplanung von T_i beziehen, können wir der Übersicht wegen auf die entsprechenden Superskripte verzichten.

Zielfunktion

Da wir eine untere Schranke für q_i ermitteln wollen und wir bereits über untere

Schranken für die Längen der späteren Jobs $T_{i-1}, \ldots, T_1$ verfügen, genügt es q_i zu minimieren.

$$\text{MIN} \quad q_i \tag{C.8}$$

Nebenbedingungen

(i) Die Gesamtlast über alle Maschinen am Ende der Jobfolge ist m, da wir die Jobs gerade so skaliert haben. Wenn T_i eintrifft, beträgt die Gesamtlast aller Maschinen mA_m. Um die in Zukunft zu verteilende Last nach unten abzuschätzen, berücksichtigen wir lediglich die Längen der kritischen Jobs, sowie die des letzten Jobs C.6. Somit ergibt sich:

$$m \geq mA_m + \sum_{j=1}^{i} q_j + 1 - \varepsilon + \beta. \tag{C.9}$$

(ii) Für alle hoch belasteten Maschinen muß die folgende Lastbedingung erfüllt sein. Die aktuelle Last l_j, $s \leq j \leq m$ muß definitionsgemäß mindestens $1 - \beta$ betragen. Da es sich bei T_i um einen kritischen Job handelt, darf er nicht einer hoch belasteten Maschine zugewiesen weden. Bezüglich der Einplanungsregel bedeutet dies, daß $l_j + q_i > \alpha A_{j-1}$, $i \leq j \leq m$ gelten muß. Beide Forderungen lassen sich durch folgende Nebenbedingung formulieren:

$$l_j \geq \max\{\alpha A_{j-1} - q_i,\, 1 - \beta\} \quad i \leq j \leq m \tag{C.10}$$

(iii) Gegenstand unserer Betrachtung ist die zweite Analysephase, die folgendermaßen charakterisiert werden kann:

$$A_i \geq a \tag{C.11}$$

(iv) Nun müssen wir noch sagen, wie die mittlere Last berechnet wird:

$$A_j = \frac{1}{j} \sum_{k=1}^{j} l_k\,, \quad 1 \leq j \leq m \tag{C.12}$$

Haben wir die Programme für $T_1, \ldots, T_{\tilde{s}+1}$ gelöst, wobei die unteren Schranken für $q_1, \ldots, q_{\tilde{s}}$ nachgewiesen haben, daß die entsprechenden Jobs fett sind, dann bedeutet die Tatsache, daß das letzte Lineare Programm keine zulässige Lösung hat, daß die Summe der Joblängenschranken zu einer Verletzung der Phasenbedingung C.11 geführt hat. Wir sind also wieder in Phase 1 angelangt, was bedeutet, daß es maximal $\tilde{s}$ kritische Jobs in Phase 2 gibt.

Die Variablen unseres Linearen Programms sind q_i, h_j und A_j, $1 \leq j \leq m$, wobei A_j abhängig ist. Da wir α erst am Ende unserer Analyse minimal wählen wollen, müssen wir zum jetzigen Zeitpunkt β fixieren. Dazu werden wir β in einem Intervall einschließen und mittels einer worst-case-Analyse einen festen Wert zuweisen. In C.3

haben wir gesehen, daß $\beta \geq 0$ gilt, so daß wir lediglich eine obere Grenze für β ermitteln müssen. Nach C.1 und C.2 gilt $CHASM_\alpha = l_1^{t_1} + p_t \geq \alpha$. In C.5 konnten wir p_t durch $p_t \leq \frac{l_1^{t-1}}{1-\varepsilon}$ abschätzen. Zusammenfassend gilt also:

$$l_1^{t_1} + \frac{l_1^{t-1}}{1-\varepsilon} > \alpha \iff l_1^{t_1} \geq 1-\varepsilon \iff \beta = 1 - l_1^{t_1} \leq \varepsilon$$

Somit liegt β im Intervall $[0, \varepsilon]$.

Bezüglich der worst-case-Analyse für β verweisen wir auf [KPT96]. Das folgende Lemma faßt jedoch ihr Ergebnis zusammen.

Lemma C.6 (Worst-Case-Analyse für β). *Die unteren Schranken für e_i, $1 \leq i \leq \tilde{s}$, die mittels* LP1 *für ein festes m und α sowie für $\beta = \varepsilon$ berechnet wurden, gelten für jedes $\beta \in [0, \varepsilon]$, falls $\varepsilon \leq 0,2$, d.h. $\alpha \geq 1,8$ gilt.*

Die bisherige Analyse erlaubt jedoch lediglich, ein Teilresultat zu formulieren.

Lemma C.7. *Für $6 \leq m \leq 13000$ sind alle kritischen Jobs in Phase 2 fett.*

Beweis. Die Autoren haben ein Programm geschrieben, welches das Lineare Programm LP1 für $6 \leq m \leq 13000$ löst. Innerhalb der für α gefundenen Schranken wurde mittels binärer Suche das kleinste α gesucht, so daß die unteren Schranken für die Joblängen mindestens b betrugen. Für jedes der vogegebenen m konnte ein solcher Wert gefunden werden, der zudem die Bedingung $\alpha \leq 1,943$ erfüllt. □

Die Aussage gilt natürlich auch für $\alpha = 1,945$, da die Nebenbedingung des Linearen Programms schärfer und somit die untere Schranke größer wird.

Bisher sind wir nicht in der Lage, den kompetitiven Faktor für $m > 13000$ zu bestätigen. Da m ein Parameter unserer Linearen Programme ist, wären wir gezwungen, diese für jedes noch so große m zu lösen. Hier stoßen wir jedoch schnell an die Grenzen der technischen Ressourcen. Wollen wir die LPs für größere Maschinenzahlen lösen, dann müssen wir den Parameter durch eine andere Größe x ersetzen, so daß die für x gefundenen Aussagen auch für alle m ab einer von x abhängigen Mindestgröße gelten.

Dies werden wir durch Diskretisierung des LPs erreichen. Wir gruppieren die m Maschinen in x Blöcke, von denen $x-1$ jeweils $q = \lceil \frac{m}{x} \rceil$ Maschinen enthalten und ein Block die verbleibenden $m-(x-1)q$ Maschinen. Diese Aufteilung ist möglich, wenn x so gewählt ist, daß $m > x(x-2)$. In diesem Fall können $x-1$ Blocks der Mindestgröße $x-1$ gebildet werden. Die Aufteilung stellt weiterhin sicher, daß Block x maximal q Maschinen umfaßt, da $m-(x-1)q = m - x\lceil \frac{m}{x} \rceil + q \leq q$ gilt. Für den weiterern Beweis wollen wir folgende Notation vereinbaren:
Ist ν eine beliebige ganzzahlige Variable, dann bezeichnen wir mit ν' den Wert $\lfloor \frac{\nu}{q} \rfloor$.
Somit stellt s' die Anzahl an Blöcken dar, die ausschließlich niedrig belastete Maschinen umfassen, während Block $s'+1$ solche Maschinen höchstens noch zu einem Teil enthalten kann. Graphik C.1 aus [Sch98] veranschaulicht die Situation.

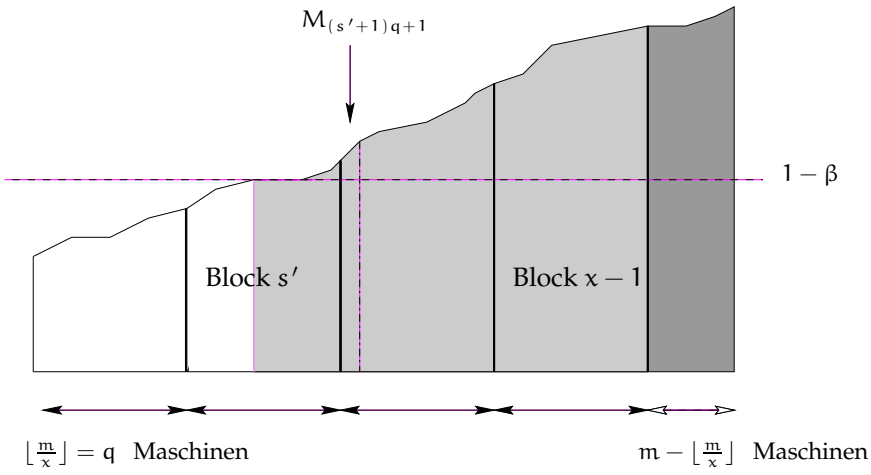


Abbildung C.1.: **Diskretisierung: Ein Beispielplan für** $x = 5$

Wir werden unser LP nun so verändern, daß m als Parameter durch x substituiert wird. Sind wir in der Lage, durch Lösung der LPs unsere Behauptung für ein festes x mit $x(x-2) < 13000$ zu zeigen, dann gilt die Aussage für alle $m > x(x-2)$. Diese Erkenntnis würde unseren Beweis abschließen.

(i) Betrachten wir zunächst den Einfluß der Diskretisierung auf die Lastbeschränkung C.10. Für alle Maschinen in Blöcken, welche ausschließlich hoch belastete Maschinen enthalten, und mit Ausnahme des letzten Blocks, verwenden wir die ursprüngliche Lastbeschränkung C.10. Wir ersetzen jedoch A_{j-1} durch $A_{(j-1)'q} = A_{\lfloor \frac{j-1}{q} \rfloor q} \leq A_{j-1}$, so daß wir erhalten:

$$l_j \geq \max\left\{\alpha A_{(j-1)'q} - e_s,\, 1-\beta\right\}, \quad (s'+1)q < j \leq (x-1)q$$

Wir können nun die folgende Beobachtung machen. Für alle l_i, l_j innerhalb eines Maschinenblocks gilt dieselbe Lastbedingung, da $(i-1)' = (j-1)'$. Da es keine Nebenbedingungen gibt, in denen die Lasten der Maschinen innerhalb eines Blocks differenziert beschränkt werden, können wir alle Bedingungen für einen Block zu einer Klasse zusammenfasen, welche durch eine einzige Restriktion für $\bar{l}_j$ repräsentiert wird. Wenn wir dedoch nicht mehr zwischen Maschinen eines Blocks bzgl. ihrer Last unterscheiden, dann macht es Sinn, eine neue Variable $\bar{A}_j$ einzuführen, welche die mittlere Last der Maschinen in den ersten j Blöcken beschreibt. Somit erhalten wir:

$$\bar{l}_j \geq \max\left\{\alpha \bar{A}_{j-1} - e_s,\, 1-\beta\right\}, \quad s'+1 < j \leq x-1 \tag{C.13}$$

Für die Maschinen im letzten Block schwächen wir die Beschränkung ab, indem wir die Maschinenlast auf

$$\bar{l}_x = 1-\beta \tag{C.14}$$

festsetzen.

(ii) Wenden wir uns nun Nebenbedingung C.11 des ursprünglichen LPs zu. Da Block $s'+1$ sowohl hoch als auch niedrig belastete Maschinen enthält, müssen wir im Rahmen einer Blockbetrachtung die Beschränkung aufweichen:

$$\bar{A}_{s'+1} = A_{(s'+1)q} \geq A_s \geq a \tag{C.15}$$

(iii) Auch die Definition der A_j-Variablen muß, wie oben bereits erwähnt, diskretisiert werden. Zudem haben wir in C.13 und C.15 gesehen, daß diese Variablen nur noch für vollständige Blöcke aus dem Intervall $[s'+1, x-1]$ benötigt werden. Entsprechend unserer Notation können wir uns daher auf folgende Nebenbedingungen beschränken:

$$\begin{aligned} \bar{A}_j &= A_{jq} = \frac{1}{jq}\sum_{k=1}^{jq} l_k = \frac{1}{jq}\sum_{k=1}^{j} q\bar{l}_k \\ &= \frac{1}{j}\sum_{k=1}^{j} \bar{l}_k, \quad s'+1 \leq j \leq x-1 \end{aligned} \tag{C.16}$$

(iv) Die erste Nebenbedingung C.9 kann folgendermaßen diskretisiert werden. Betrachten wir zunächst die Last der ersten $s'+1$ Blöcke. Sie beträgt mindestens $(s'+1)qA_s \geq (s'+1)qa$, da die Phasentrennung erst im $(s'+1)$ten Block stattfindet. Die Last in den verbleibenden Blöcken lautet unter Betrachtung von Lastklassen somit:

$$\sum_{j=(s'+1)q+1}^{m} l_j = \sum_{j=(s'+1)q+1}^{m'q} l_j + \sum_{j=m'q+1}^{m} l_j = \sum_{j=s'+2}^{m'} q\bar{l}_j + (m-m'q)\bar{l}_x$$

Die zusätzliche Last durch die in Zukunft eintreffenden kritischen Jobs kann abgeschätzt werden, indem die Jobs mit Zielmaschinen im $(s'+1)$ten Block vernachlässigt werden, d.h. $\sum_{j=1}^{s} e_j \geq \sum_{j=1}^{s'q} e_j$. Vernachlässigen wir gegenüber C.9 den letzten Job, so ergibt sich:

$$m \geq (s'+1)qa + \sum_{j=s'+2}^{m'} q\bar{l}_j + (m-m'q)\bar{l}_x + \sum_{j=1}^{s'q} e_j$$

Dividieren wir linke und rechte Seite der Nebenbedingung mit q, dann erhalten wir:

$$\frac{m}{q} \geq (s'+1)a + \sum_{j=s'+2}^{m'} \bar{l}_j + \frac{1}{q}(m-m'q)\bar{l}_x + \frac{1}{q}\sum_{j=1}^{s'q} e_j \tag{C.17}$$

Ist m ein Vielfaches von q, dann gilt $m = m'q$ und $m' = x$, und weiter

$$\sum_{j=s'+2}^{m'} \bar{l}_j + \frac{1}{q}(m-m'q)\bar{l}_x = \sum_{j=s'+2}^{x} \bar{l}_j \,.$$

Andernfalls sehen wir aufgrund von $x < \frac{m}{q}$ und $l_x \geq 0$:

$$\frac{1}{q}(q-(m-m'q))l_x = (m'+1-\frac{m}{q})l_x = (x-\frac{m}{q})l_x \leq x - \frac{m}{q}$$

Dies erlaubt uns, $x - m/q$ zur linken Seite sowie $(1/q)(q-(m-m'q))l_x$ zur rechten Seite von C.17 zu addieren. In beiden Fällen vereinfacht sich die Nebenbedingung zu:

$$x \geq (s'+1)a + \sum_{j=s'+2}^{x} \bar{l}_j + \frac{1}{q}\sum_{j=1}^{s'q} e_j \tag{C.18}$$

Es ist noch eine weitere Vereinfachung möglich, die auf folgender Beobachtung beruht. Falls für zwei kritische Jobs e_i und e_j gilt, daß $i' = j'$, dann sind die entsprechenden LPs identisch. Man muß das LP also nur einmal lösen, da die unteren Schranken für p_i und p_j sich nicht unterscheiden werden. Wir können daher o.B.d.A. annehmen, daß s ein Vielfaches von q ist und fassen folglich alle kritischen Jobs e_i, mit $\lfloor i \rfloor = j$ zu einer Klasse zusammen, welche durch die Variable $\bar{e}_j$ repräsentiert wird.

Klasse $\bar{e}_0$:	$e_1, \ldots, e_{q-1}$	$\Rightarrow$	$q-1$ Jobs
Klasse $\bar{e}_j$:	$e_{jq}, \ldots, e_{2jq-1}$	$\Rightarrow$	q Jobs
Klasse $\bar{e}_{s'}$:	e_s	$\Rightarrow$	1 Job

Somit läßt sich der letzte Term von C.18 vereinfachen :

$$\frac{1}{q}\sum_{j=1}^{s'q} e_j = \frac{1}{q}\left[(q-1)\bar{q}_0 + \sum_{j=1}^{s'+1} q\bar{e}_j + \bar{e}_{s'}\right] \geq \sum_{j=1}^{s'-1} \bar{e}_j$$

Da wir die LPs nur noch für die Repräsentanten der Jobklassen lösen müssen, können wir in Nebenbedingung C.13 sowie in der Zielfunktion C.8 e_s durch $\bar{e}_s$ ersetzen.

Wir können unseren Beweis nun mit folgendem Lemma abschließen.

Lemma C.8. *Für* $m > 13000$ *sind die kritischen Jobs in Phase 2 fett.*

Beweis. Die Autoren haben die diskretisierten LPs für $\alpha = 1,945$ und $x = 115$ gelöst. Es hat sich dabei herausgestellt, daß alle unteren Schranken für die Joblängen der kritischen Jobs mindestens b betragen.
Die Aussage gilt, wie wir gezeigt haben, für $m > x(x-2) = 12995$. □

Literaturverzeichnis

[ABK92] Y. Azar, A. Broder, and A. Karlin. On-line load balancing. pages 218–225, 1992.

[Alb97] S. Albers. Better bounds for online scheduling. In *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC97)*, pages 130–139, 1997.

[BEP+96] J. Blażewicz, K. H. Enger, E. Pesch, G. Schmidt, and J. Węglarz. *Scheduling Computer and Manufacturing Processes*. Springer, 1996.

[BFKV92] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In N. Alon, editor, *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing*, pages 51–58, Victoria, B.C., Canada, May 1992. ACM Press.

[BKR94] Y. Bartal, H. Karloff, and Y. Rabani. A better lower bound for on-line scheduling. *Information Procesing Letters*, 50:113–116, 1994.

[BLRK83] J. Blażewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathmatics*, 5:11–24, 1983.

[Cof76] E. G. Coffman. A generalized bound on lpt sequencing. *RAIRO-Informatique*, 10:17–25, 1976.

[FKT89] U. Faigle, W. Kern, and G. Turan. On the performance of on-line algorithms for particular problems. *Acta Cybernetica*, 9:107–119, 1989.

[FRK86] J. B. G. Frenck and A. H. G. Rinnooy Kan. The rate of covergence to optimality of the lpt-rule. *Discrete Applied Mathmatics*, 14:187–197, 1986.

[FRK87] J. B. G. Frenck and A. H. G. Rinnooy Kan. The asymptotic optimality of the lpt-rule. *Math. Oper. Res.*, 12:241–254, 1987.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. A Series of Books in the Mathematical Sciences. Freemann And Company, San Francisco, 1979.

[GLRK79] R.L Graham, E.L. Lawler, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling theory: a survey. volume 5, pages 287–326, 1979.

[Gra66] R. L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[Gra69] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Computing*, 17:263–269, 1969.

[GW] G. Galambos and G. Woeginger. An on-line scheduling heuristic with better worst case ratio than graham's list scheduling. manuscript.

[HS87] D. Hochbaum and D Shmoys. Using dual approximation algorithms for scheduling problems. *Journal of the ACM*, 34:144–162, 1987.

[Ked70] S. K. Kedia. A job scheduling problem with parallel processors. Technical report, Dept. of Ind. Eng., University of Michigan, Ann Arbor, 1970.

[KPT96] D. R. Karger, S. J. Philips, and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20:400–430, 1996.

[McN59] R. McNaughton. A better algorithm for an ancient scheduling problem. *Management Science*, 6:1–12, 1959.

[NC91] P. R. Narayanan and R. Chandrasekaran. Optimal on-line algorithms for scheduling. manuscript, 1991.

[Pin95] M. Pinedo. *Scheduling : theory, algorithms, and systems*. Prentice Hall international series in industrial and systems engineering. Prentice-Hall, Englewood Cliffs, NJ, 1st ed. edition, 1995.

[Por85] M.E. Porter. *Competitive Advantage*. Collier Macmillan, 1985.

[Rot66] M. H. Rothkopf. Scheduling independent tasks on parallel processors. *Management Science*, 1966.

[Sch94] C. Scholz. *Personalmanagement*. Vahlens Handbücher der Wirtschfts- und Sozialwissenschaften. Vahlen, 4 edition, 1994.

[Sch95] A.-W. Scheer. *Wirtschaftsinformatik – Referenzmodelle für industrielle Geschäftsprosesse*. Springer, 1995.

[Sch97] G. Schmidt. *Prozeßmanagement – Modelle und Methoden*. Springer, 1997.

[Sch98] B. Schröder. Upper and lower bounds for basic scheduling problems. Master's thesis, University of Saarland, Saarbrücken, 1998.

[SWW95] D. Shmoys, J. Wein, and D. Williamson. Scheduling parallel machines online. *SIAM Journal on Computing*, 24(6):1313–1331, December 1995.

[Tör99] E. Török. Spiegelkabibnett. *ct – Magazin für Computertechnik*, (6):302–308, 1999.

[VB96] G. Vossen and J. Becker, editors. *Geschäftsprozeßmodellierung und Workflow-Management – Modelle, Methoden, Werkzeuge*. International Thomson Publishing, 1996.