Online Scheduling-Spiele

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Matthias Kretschmer

aus

Bonn

Bonn, 2014

Anfertigung mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn.

1. Gutachter: Prof. Dr. Norbert Blum
2. Gutachter: Prof. Dr. Heiko Röglin

Tag der Promotion: 21.10.2014
Erscheinungsjahr: 2014

Zusammenfassung

Ziel dieser Arbeit ist die Modellierung von online Scheduling-Spielen. Es gibt viele verschiedene Arten von Scheduling-Problemen. Wir betrachten Probleme, bei denen eine Menge von Jobs Maschinen zugewiesen werden müssen. Jeder Job wird dann von der Maschine bearbeitet, der er zugeteilt wurde. In der Praxis treten Scheduling- und Lastbalancierungsprobleme häufig in der Form auf, dass die Jobs nicht zentral auf die Maschinen oder auch Ressourcen verteilt werden, sondern, dass jeder Job einem sogenannten Spieler gehört, der eine Maschine selbstständig für seinen Job auswählt. Dies ist insbesondere bei öffentlichen Netzwerken, wie zum Beispiel dem Internet, der Fall. Bei solchen Netzwerken werden einzelne Dienste teilweise von mehreren verschiedenen Servern angeboten. Ein Benutzer kann sich dann frei einen dieser Server zur Bearbeitung aussuchen. Damit findet keine zentrale Verteilung der Jobs mehr statt.

In dieser Arbeit werden wir solche Systeme in klassische spieltheoretische Modelle einbetten und analysieren. Im Gegensatz zu den offline Scheduling-Spielen, die von Koutsoupias und Papadimitriou [KP99] eingeführt wurden, werden wir ein allgemeineres Modell einführen, das dazu verwendet werden kann, offline und online Modelle zu beschreiben. Bei dieser Modellierung können die Spieler nicht die Reihenfolge, mit der die Jobs auf einer Maschine bearbeitet werden, beeinflussen. Diese ist durch das für das konkrete Spiel fest vorgegebene Maschinenmodell definiert. Die Wahl des Maschinenmodells beeinflusst bei solchen Spielen nicht nur die Abarbeitungsreihenfolge auf den Maschinen, sondern indirekt auch die Strategien, die ein rationaler Spieler auswählen würde. Somit werden die Lösungen, die von den Strategien rationaler Spieler erzeugt werden, durch das gewählte Maschinenmodell beeinflusst. Wir werden zeigen, dass unsere Modellierung es erlaubt, die von Koutsoupias und Papadimitriou eingeführten Spiele mittels eines speziellen Maschinenmodells zu beschreiben. Zudem werden wir online Scheduling-Spiele beschreiben und verschiedene Maschinenmodelle für diese online Spiele analysieren.

Inhaltsverzeichnis

Kapitel 1

Einleitung

In dieser Arbeit werden wir uns mit Scheduling-Problemen beschäftigen. Es gibt viele verschiedene Arten von Scheduling-Problemen. Wir betrachten Probleme, bei denen eine Menge von Jobs Maschinen zugewiesen werden müssen. Jeder Job wird dann von der Maschine bearbeitet, der er zugewiesen wurde.

1.1 Motivation

In der Praxis treten Scheduling- und Lastbalancierungsprobleme häufig in der Form auf, dass die Jobs nicht zentral auf die Maschinen oder auch Ressourcen verteilt werden, sondern, dass jeder Job einem sogenannten Spieler gehört, der eine Maschine selbstständig für seinen Job auswählt.

Dies ist beispielsweise bei Dateiservern, die über das Internet den Nutzern Dateien und Informationen anbieten, der Fall. Bei frei erhältlicher Software ist es üblich, dass diese Software auf mehreren sogenannten Spiegelservern zur Verfügung steht. Der Benutzer kann frei entscheiden, welchen der verfügbaren Spiegelserver er verwenden möchte. Hierbei hängt die Zeit, die benötigt wird, um die Daten auf den eigenen Computer zu laden, nicht nur von der Anbindung und Hardware des ausgewählten Spiegelservers ab, sondern auch davon, wie viele andere Nutzer zur gleichen Zeit Daten von diesem Server laden. Die Nutzer sind in diesem Fall die Spieler dieses spieltheoretischen Problems. Das Ziel des Spielers ist es, einen Server in der Weise auszuwählen, dass er seine Daten in möglichst kurzer Zeit erhält.

Übertragen auf allgemeine Scheduling-Spiele bedeutet dies, dass ein Spieler eine Maschine so auszuwählen versucht, dass sein eigener Job so schnell wie

möglich bearbeitet wird. Für den einzelnen Spieler ist hierbei uninteressant, wie die eigene Wahl die Antwortzeiten der anderen Spieler beeinflusst. Als Antwortzeit bezeichnet man die Zeit, die benötigt wird, bis der Spieler das Resultat seines Jobs erhält. Dadurch kann es bei Scheduling-Spielen passieren, dass der Wunsch der Spieler im Gegensatz zu einem globalen Ziel steht. So könnte das allgemeine Interesse darin bestehen, die maximale Antwortzeit so gering wie möglich zu halten. Das heißt, dass rationale Spieler nicht in jedem Fall einen Zustand erreichen, bei dem eine solche Zielfunktion wie die maximale Antwortzeit aller Jobs optimiert wird.

In Koutsoupias und Papadimitriou [KP99] werden Lastbalancierungsspiele eingeführt. Dazu werden diese Spiele als sogenannte strategische Spiele modelliert, bei denen die Spieler simultan, ohne die Kenntnis über die Strategie der anderen Spieler oder die Möglichkeit der Kommunikation mit diesen, eine Maschine auswählen. Den Spielern sind alle Jobs aller Spieler bekannt und sie können auf der Basis dieser Informationen ihre eigene Strategie planen. Das heißt, dass durch dieses Modell eine Art von offline Lastbalancierungsspielen abgebildet wird.

Die Annahme, dass jeder Spieler jeden anderen Job kennt, ist in vielen Fällen unrealistisch. In dem Beispiel der Spiegelserver kann man davon ausgehen, dass kein Nutzer weiß, für welche Dateien die anderen Nutzer – und dies sind alle anderen Menschen auf dieser Welt mit einem Computer und Internetzugang – sich interessieren und möglicherweise von einem der Spiegelserver herunterladen werden. Das heißt, wir können davon ausgehen, dass die Spieler keine Informationen über die anderen Nutzer besitzen und eventuell nur eine vage Vorstellung haben, wie viele andere Nutzer an welchen anderen Dateien interessiert sind.

In der Praxis ist es durchaus nicht unüblich, dass die Spieler zwar nicht alle anderen Jobs kennen, aber dennoch weitere Informationen besitzen. In dem Beispiel der Spiegelserver ist es technisch möglich, den Nutzern eine Statistik bereitzustellen, so dass diese erfahren, wie viel Bandbreite derzeit von allen Nutzern dieses Servers beansprucht wird. Ebenso besteht die Möglichkeit die Anzahl der Nutzer, die aktuell Daten des Servers beziehen, als Information bereitzustellen. Der Server `http://ftp.halifax.rwth-aachen.de` der RWTH Aachen dient als Spiegelserver für eine Vielzahl von frei erhältlichen Softwareprojekten. Zum Beispiel werden verschiedene Linux-Distributionen über diesen Server angeboten. Auf dessen Hauptseite wird die aktuelle Auslastung des Servers den Nutzern präsentiert (siehe Abbildung 1.1 und `http://ftp.halifax.rwth-aachen.de/index.html`). Damit ist dieser Dienst ein gutes Beispiel dafür, dass den Nutzern, bzw. Spielern, mehr Informationen zur Verfügung stehen, als das KP-Modell zulässt. Sie besitzen

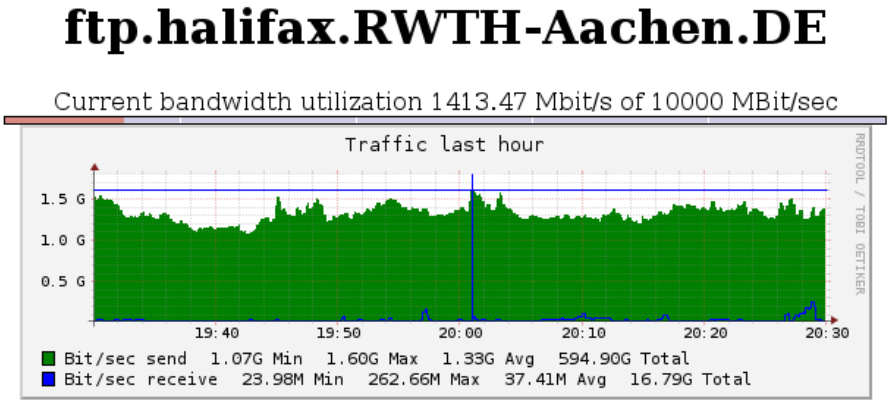

Abbildung 1.1: Beispiel für zusätzliche Informationen: Auslastung des Spiegelservers der RWTH Aachen (Stand: 15. Juli 2013)

somit ausgewählte Informationen über die Aktionen anderer Spieler.

Bei dem vorstehenden Problem kommen neue Jobs zum Spiel hinzu, während die Aufträge der anderen Spieler bearbeitet werden. Bei einer anderen aus der Literatur bekannten Variante von online Scheduling-Problemen werden zuerst die Jobs auf die Maschinen verteilt, bevor die Bearbeitung dieser Jobs beginnt. Die Spieler machen nacheinander ihren Zug und wählen eine Maschine zur Bearbeitung aus. Dabei kann ein Spieler später seine getroffene Entscheidung nicht mehr rückgängig machen. Die Maschinen haben die Möglichkeit die Reihenfolge der ihnen zugewiesenen Jobs zur Bearbeitung selber zu bestimmen. Eine Übersicht über bekannte Online-Algorithmen zu dieser Art von online Scheduling-Problemen findet sich in der Übersichtsarbeit von Pruhs, Sgall und Torng [PST03, Kapitel 3] und [LKA04, Kapitel 15]. Motiviert ist diese Art von Problemen als online Problem durch Systeme, bei denen zuerst die Jobs von den Maschinen gesammelt werden, um sie dann zu bearbeiten.

Zum Beispiel könnten die Maschinen Dienstleistungsunternehmen sein. Am Tag bevor die Aufträge abgearbeitet werden, werden sie von dem Dienstleister entgegen genommen und gesammelt. Sobald sich ein Kunde meldet, kennt der Dienstleister schon einen Teil der Aufträge, die er am nächsten Tag bearbeiten wird und kann eine Einschätzung abgeben, wie schnell er am nächsten Tag den Auftrag bearbeiten kann.

Ziel dieser Arbeit ist die Modellierung von online Scheduling-Spielen. Die hier vorgestellte Form der Modellierung erlaubt es, wie es auch bei klassischen Scheduling-Problemen der Fall ist, den zweiten Typ von Scheduling-

Spielen, bei dem die Jobs der Reihe nach verteilt werden, als Spezialfall der Scheduling-Spiele des ersten Typs aufzufassen. Bei der Modellierung von Koustoupias und Papadimitriou sind die Jobgrößen fest vorgegeben. Im Fall von online Scheduling-Spielen darf dies natürlich nicht der Fall sein, um die Ungewissheit über zukünftige Ereignisse zu modellieren. Dies wird dadurch erreicht, dass nicht Spiele in Normalform als Grundlage verwendet werden, sondern Spiele in Extensivform.

Diese Modellierung hat einen weiteren Vorteil. Sie ist eine Verallgemeinerung der Modellierung von Koutsoupias und Papadimitriou, so dass sowohl die Spiele des KP-Modells abgebildet als auch verschiedene Varianten von online Scheduling-Problemen dargestellt werden können.

Die grundlegende Idee der Modellierung besteht darin, die sogenannten Spiele in Extensivform mit imperfekten Informationen, die aus der klassischen Spieltheorie bekannt sind, zu verwenden. Mit Hilfe dieser Modellierung kann man die oben angesprochenen Probleme des KP-Modells lösen, indem zum einen die Eingabe, das heißt die Jobs der Spieler, nicht von vornherein vollständig bekannt ist. Dies wird erreicht, indem ein zusätzlicher Spieler, den wir Natur nennen werden, nach einer festgelegten Wahrscheinlichkeitsverteilung eine Eingabe auswählt. Zum anderen führen die Spieler ihre Aktionen nacheinander durch. Um nun zu verhindern, dass jeder Spieler exakte Kenntnis über die schon getätigten Züge und die Eingabe bekommt, wird die Menge der möglichen Spielzustände partitioniert. Ein Spielzustand besteht aus der von der Natur gewählten Eingabe und den Aktionen der Spieler, die vorher einen Zug durchgeführt haben. Dem Spieler, der als nächstes an der Reihe ist, wird nun nicht der aktuelle Spielzustand mitgeteilt, sondern nur die Partition, zu der dieser Spielzustand gehört. Durch eine geschickte Wahl der Partitionierung können auf diese Art und Weise die Informationen, die einem Spieler zur Verfügung stehen, beschränkt werden.

Nach unserem Wissen ist dies die erste Arbeit, die online Scheduling-Spiele aus Sicht der Informatik beschreibt und modelliert. Wir werden aufzeigen, wie man solche Spiele darstellen kann. Des Weiteren präsentieren wir Maschinenmodelle, bei denen rationale Spieler vernünftige Verteilungen erzeugen. Hierbei heißt vernünftig, dass sie nicht zu schlecht bezüglich der maximalen und durchschnittlichen Antwortzeit aller Jobs sind.

1.2 Aufbau der Arbeit

In Kapitel 1.3 werden zunächst verwandte Arbeiten vorgestellt. Wir werden erläutern, in welchem Zusammenhang diese Arbeiten zu unseren Ergebnissen

und unserer gewählten Modellierung stehen.

Die grundlegenden Definitionen und die benötigten spieltheoretischen Grundlagen werden in Kapitel 2 erläutert. Wir werden keine allgemeine Einführung in die Spieltheorie und Spiele in Extensivform bieten. Stattdessen werden wir direkt Scheduling-Spiele beschreiben und definieren. Für eine allgemeine Einführung sei auf entsprechende Lehrbücher verwiesen (zum Beispiel [OR94, Ras89]).

In Kapitel 3 zeigen wir, dass die Modellierung von Koutsoupias und Papadimitriou ein Spezialfall von den hier vorgestellten Scheduling-Spielen ist. In Kapitel 4 werden wir sehen, dass man ein Spiel so definieren kann, dass rationale Spieler sich immer an einen gegebenen deterministischen Online-Algorithmus halten werden. Allerdings wird dies dadurch erreicht, dass wir ein sehr unnatürliches Maschinenmodell verwenden, das viele Leerlaufzeiten erzeugt und jeden Spieler, der sich nicht an den Online-Algorithmus hält, bestraft. Ziel ist es demnach, ein natürliches Maschinenmodell zu finden, so dass rationale Spieler eine für die Allgemeinheit vernünftige Lösung erzeugen. Ein solches natürliches Modell wird in Kapitel 5 vorgestellt. Es wird gezeigt, dass dies äquivalent zu List Scheduling ist und auch dessen Nachteile als Online-Algorithmus bezüglich der durchschnittlichen Antwortzeit erbt.

In Kapitel 6 führen wir ein verbessertes Modell für Spiele, bei denen zuerst die Aufträge gesammelt und danach die Jobs von den Maschinen bearbeitet werden, ein. Bei diesem Modell teilen wir die Jobs in Klassen ein und lassen die Maschinen die Jobs priorisiert nach den Klassen abarbeiten, um das Problem mit der durchschnittlichen Antwortzeit des einfachen Modells zu umgehen. Wir werden zeigen, dass bei identischen Maschinen der Preis der Anarchie, das heißt, das Verhältnis zwischen einem schlechtesten stabilen Zustand, der von rationalen Spielern erreicht werden kann, und optimaler Strategie bezüglich einer Zielfunktion, im Fall der durchschnittlichen Antwortzeit durch eine Konstante beschränkt ist. Das Resultat erhalten wir, indem wir zeigen werden, dass wir ein vereinfachtes Modell verwenden können, in dem die Spieler weniger Informationen haben, als durch das online Modell vorgegeben ist. Dies erlaubt es uns, zu zeigen, dass die Spieler innerhalb einer Klasse sich wie List Scheduling verhalten. Daraus erhalten wir dann die nötigen Rückschlüsse, um den konstanten Preis der Anarchie zu beweisen. Im Anschluss an die Analyse dieses Modells mit beschränkten Informationen werden wir beweisen, dass im Fall von identischen Maschinen die Resultate auf das unbeschränkte Modell übertragen werden können. Am Ende des Kapitels werden wir zudem darauf eingehen, warum sich dieses Modell nicht direkt auf allgemeinere Scheduling-Spiele, bei denen die Jobs erst im Verlauf der Zeit bekannt werden, übertragen lässt.

1.3 Verwandte Arbeiten

Es gibt viele verschiedene Varianten von offline und online Scheduling-Problemen. Die hier vorgestellte Modellierung beschreibt Scheduling-Probleme mit mehreren Maschinen. Jeder Job wird durch eine feste Größe x beschrieben und die Bearbeitungszeit dieses Jobs auf einer Maschine der Geschwindigkeit s beträgt x/s Zeiteinheiten. Für einen Job darf einmalig eine Maschine ausgewählt werden. Diese Entscheidung kann später nicht mehr geändert werden. Damit betrachten wir online Scheduling mit *immediate dispatch*. Zudem werden die Jobs den Maschinen zugewiesen, bevor die Bearbeitung der Jobs beginnt. Es werden somit Probleme modelliert, bei denen der Ablauf in zwei Phasen unterteilt werden kann: Verteilung und Bearbeitung. Es existieren viele verschiedene Arbeiten über Algorithmen und untere Schranken für diese Art von Scheduling-Problemen.

Für diese Arbeit ist unter anderem der List Scheduling Algorithmus von Graham [Gra66] von besonderem Interesse. Wir werden zeigen, dass in der einfachsten Form der online Scheduling-Spiele, die Spieler bei stabilen Strategien sich genau so verhalten, wie dieser Algorithmus. Diese Verwandtschaft ist nicht zufällig. List Scheduling ist ein einfacher Greedy-Algorithmus, bei dem der nächste Job auf genau die Maschine verteilt wird, die diesen Job am schnellsten bearbeiten würde. Wenn die Maschinen die Jobs in der Reihenfolge, in der sie in der Eingabe vorkommen, abarbeiten, wird man erwarten, dass die Spieler sich wie dieser Greedy-Algorithmus verhalten, da jeder Spieler versucht seine Antwortzeit zu minimieren.

Für den hier verwendeten Typ von online Scheduling-Problemen haben Fiat und Woeginger [FW99] gezeigt, dass es im Fall von einer Maschine keinen online Algorithmus gibt, der ein kompetitives Verhältnis von $O(\log n)$ hat, wobei n die Anzahl der Jobs ist. Bei dem von Fiat und Woeginger beschriebenen Modell wählt der online Algorithmus das Zeitintervall aus, in dem der Job bearbeitet wird. Das heißt, bei diesem Problem entscheidet nicht die Maschine über die Bearbeitungsreihenfolge.

Nicht alle Online-Algorithmen lassen sich übertragen, ohne unnatürliche Abarbeitungsreihenfolgen auf den Maschinen festzulegen. Dies liegt daran, dass effiziente Online-Algorithmen teilweise eine schlechte Wahl für einen einzelnen Job treffen, um insgesamt sicherstellen zu können, dass die so generierte Lösung möglichst nah am Offline-Optimum liegt. Spieler achten allerdings nicht auf eine globale Optimierung, sondern versuchen nur ihre eigene Antwortzeit zu minimieren. Damit würde ein Spieler einen solchen Zug nicht durchführen wollen. Die Übersichtsarbeiten von Sgall [Sga98] und Pruhs,

Torng und Sgall [PST03] und [LKA04, Kapitel 15] geben einen Überblick über verschiedene untere Schranken und Algorithmen zu verschiedenen Varianten von online Scheduling-Problemen.

Im Fall von Lastbalancierungsspielen geben Feldmann, Gairing, Lücking, Monien und Rode [FGL+03b] einen Überblick über verschiedene Ergebnisse, die das Modell von Koutsoupias und Papadimitriou [KP99] verwenden. Immorlica, Li, Mirrokni und Schulz [ILMS09] und Kretschmer [Kre06] führen eine Variante dieses Modells ein, bei dem die Jobs in monoton aufsteigender Sortierung bearbeitet werden. Dies ist insofern interessant, als dass dann ein Nash-Gleichgewicht, das einen stabilen Zustand des Spiels darstellt, mittels der SPT-Regel (Shortest Processing Time first) durch List Scheduling generiert wird. Die SPT-Regel besagt, dass die Jobs nach ihren Größen monoton aufsteigend sortiert und dann in dieser Reihenfolge durch List Scheduling auf die Maschinen verteilt werden. Dies ist von Bedeutung, da dies bei identischen Maschinen ein optimaler Algorithmus für die Minimierung der durchschnittlichen Antwortzeit ist. Man kann weiterhin zeigen, dass bei der Gewichtung der Antwortzeit von Job j um $1/x_j$, wobei x_j die Größe des Jobs ist, List Scheduling mit SPT-Regel ein optimaler Algorithmus für die gewichtete durchschnittliche Antwortzeit darstellt [Kre06]. Wir werden versuchen, die Eigenschaften dieser SPT-Regel auszunutzen, um ein natürliches Spielmodell zu erhalten, das gute Ergebnisse bei der durchschnittlichen Antwortzeit garantiert.

Kapitel 2

Grundlagen

Bevor wir die Definition von (online) Scheduling-Spielen präsentieren, werden wir die zugrunde liegenden spieltheoretischen Konzepte einführen. Da es sich um Scheduling-Spiele handelt, werden wir zuerst Scheduling-Probleme bzw. online Scheduling-Probleme beschreiben und dann die spieltheoretischen Grundlagen aufführen, mit denen online Varianten dieser Probleme als Spiel modelliert werden können.

2.1 Scheduling-Probleme

Wir verstehen im Folgenden unter Scheduling-Problemen Probleme, bei denen Jobs auf eine feste Menge von Maschinen verteilt werden. Jeder Job wird dann von derjenigen Maschine verarbeitet, auf die dieser verteilt wurde. Sei $M = \{1, 2, \ldots, m\}$ die Menge aller Maschinen. Jeder Maschine $i \in M$ ist eine Geschwindigkeit $s_i \in \mathbb{Q}_+$ $(s_i > 0)$ zugeordnet. Das bedeutet, dass eine Maschine i mit Geschwindigkeit s_i einen Job der Größe $x \in \mathbb{N}$ in Zeit x/s_i bearbeitet.

Eine Eingabe der Länge n ist nun eine Folge $((x_1, r_1), (x_2, r_2), \ldots, (x_n, r_n)) \in (\mathbb{N} \times \mathbb{R}_{\geq 0})^n$ von Jobs, die durch ihre Jobgrößen und die sogenannten *Einlastzeiten* (Englisch: *release dates*) repräsentiert werden. Die Einlastzeit r_j eines Jobs j der Größe x_j ist der Zeitpunkt, ab dem dieser Job bearbeitet werden kann. Wir verlangen, dass die Jobs in einer Eingabe nach ihren Einlastzeiten sortiert sind. Für alle $j < j'$ ist die Einlastzeit r_j von Job j kleiner oder gleich der Einlastzeit $r_{j'}$ von Job j'. Die Menge $\overline{\mathcal{E}}(n)$ ist die Menge aller Eingaben

der Länge $\leq n$. Das heißt,

$$\overline{\mathcal{E}}(n) := \{((x_1, r_1), (x_2, r_2), \ldots, (x_k, r_k)) \in (\mathbb{N} \times \mathbb{R}_{\geq 0})^k \mid 1 \leq k \leq n, \forall 1 \leq j < k : r_j \leq r_{j+1}\}.$$

Mit $\overline{\mathcal{E}}$ bezeichnen wir die Menge aller Eingaben

$$\overline{\mathcal{E}} = \bigcup_{n \in \mathbb{N}} \overline{\mathcal{E}}(n).$$

Ein Schedule für eine gegebene Eingabe $((x_1, r_1), (x_2, r_2), \ldots, (x_n, r_n)) \in \overline{\mathcal{E}}$ ist eine Verteilung der Jobs $N = \{1, 2, \ldots, n\}$ auf die Maschinen M, so dass jeder Job j nur zu einem Zeitpunkt $t \in \mathbb{R}_{\geq 0}$ auf einer Maschine i bearbeitet wird, falls Maschine i zum Zeitpunkt t keinen anderen Job bearbeitet und der Job schon zur Verfügung steht ($r_j \leq t$).

Es gibt verschiedene Varianten von Scheduling-Problemen. Zu diesen Varianten zählen unter anderem die Folgenden:

- Ein Job darf unterbrochen werden. Das heißt, dass der Job zu einem beliebigen Zeitpunkt von der Maschine genommen werden darf, auch wenn er noch nicht fertig bearbeitet wurde. Falls er noch nicht fertig bearbeitet wurde, muss mit der Bearbeitung von vorne begonnen werden.
- Präemptives Scheduling: Die Jobs dürfen unterbrochen werden und können an der Stelle, an der sie unterbrochen wurden, fortgesetzt werden.
- Jobs dürfen später die Maschine wechseln.

Die Abbildung 2.1 skizziert einen präemptiven Schedule für eine Eingabe mit drei Jobs und zwei Maschinen, bei der die Jobs die Maschine auch nach einer Unterbrechung wechseln dürfen. Die Einlastzeiten für alle drei Jobs sind $r_1 = r_2 = r_3 = 0$.

Im Folgenden werden wir davon ausgehen, dass ein Job nicht die Maschine wechseln darf. Das heißt, dass ein Schedule eindeutig bestimmt ist, wenn zu jedem Job eine Maschine, die diesen Job bearbeitet, festgelegt wird und zusätzlich die Zeiten, an denen die Maschine den Job bearbeitet. Das heißt, ein Schedule $S = (S_1, S_2, \ldots, S_n)$ für die Eingabe $x = ((x_1, r_1), (x_2, r_2), \ldots, (x_n, r_n))$ kann folgendermaßen repräsentiert werden: für alle $j \in N$ ist $S_j = (m_j, T_j)$ mit

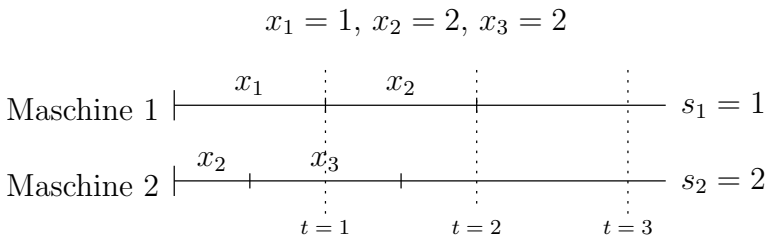

Abbildung 2.1: Präemptiver Schedule

- $m_j \in M$ und
- $T_j \subset \mathbb{R}_{\geq 0}$.

Hierbei muss gelten, dass für alle $j, j' \in N$ mit $m_j = m_{j'}$

$$T_j \cap T_{j'} = \emptyset.$$

Das bedeutet, dass eine Maschine nur einen Job gleichzeitig bearbeiten darf.

Weiterhin darf ein Job erst bearbeitet werden, wenn er zur Verfügung steht. Das bedeutet, dass

$$\forall j \in N : \forall t \in T_j : r_j \leq t$$

gelten muss.

Je nach Variante des Scheduling-Problems müssen folgende zusätzliche Eigenschaften gelten:

- nicht unterbrechbar und nicht präemptiv:

  $$\forall j \in N : \exists a \in \mathbb{R}_{\geq 0} : T_j = \left[a, a + \frac{x_j}{s_{m_j}}\right).$$

  Das heißt, dass jeder Job an einem Stück und vollständig bearbeitet wird.

- unterbrechbar und nicht präemptiv:

  $$\forall j \in N : \exists a \in \mathbb{R}_{\geq 0} : \left[a, a + \frac{x_j}{s_{m_j}}\right) \subset T_j \wedge \forall t \geq a + \frac{x_j}{s_{m_j}} : t \notin T_j.$$

  Das heißt, dass für jeden Job der letzte zusammenhängende Bearbeitungsabschnitt auf der Maschine den Job vollständig an einem Stück bearbeitet.

- unterbrechbar und präemptiv: In diesem Modell dürfen die Jobs beliebig unterbrochen und fortgesetzt werden. Das heißt, dass ein Job bearbeitet ist, falls insgesamt die Zeit x_j/s_{m_j} von Maschine m_j auf diesen Job aufgewendet wurde. T_j ist eine Menge von l Intervallen $T_{j,1}, T_{j,2}, \ldots, T_{j,l}$. $|T_{j,\lambda}|$ ist die Größe des λ-ten Intervalls. Dann gilt formal in diesem Modell

$$\forall j \in N : \sum_{\lambda=1}^{l} |T_{j,\lambda}| = \frac{x_j}{s_{m_j}}.$$

Nach dieser Definition können wir nun auch den Fertigstellungszeitpunkt $C_j(S)$ von Job j bei Schedule S für Eingabe x definieren.

$$C_j(S) := \sup_{t \in T_j} t.$$

Unter der *Flusszeit* $F_j(x, S)$ eines Jobs j bei Schedule S für Eingabe x versteht man die Zeit, die von dem Einlastzeitpunkt bis zum Fertigstellungszeitpunkt vergeht. Das heißt,

$$F_j(x, S) := C_j(S) - r_j.$$

Insbesondere gilt, dass $F_j(x, S) = C_j(S)$, falls $r_j = 0$.

Ein Scheduling-Problem ist nun, wie folgt, definiert. Sei dazu eine Zielfunktion f gegeben, die jedem Schedule S für eine Eingabe $x \in \overline{\mathcal{E}}$ einen reellen Wert $f(x, S)$ zuordnet. Dann ist das Scheduling-Problem definiert durch:

Scheduling-Problem:

- Eingabe: $x \in \overline{\mathcal{E}}$.
- Ausgabe: Schedule S für x, so dass $f(x, S)$ minimal ist.

Gängige Zielfunktionen sind die maximale Antwortzeit $f_{[C_{\max}]}$ und die durchschnittliche Antwortzeit $f_{[\sum C_j]}$, die folgendermaßen definiert sind:

$$f_{[C_{\max}]}(x, S) := \max_{j \in N} C_j(S) \quad \text{und}$$
$$f_{[\sum C_j]}(x, S) := \sum_{j \in N} C_j(S).$$

Weitere häufig verwendete Zielfunktionen sind die maximale Flusszeit $f_{[F_{\max}]}$ und die durchschnittliche Flusszeit $f_{[\sum F_j]}$, die folgendermaßen definiert sind:

$$f_{[F_{\max}]}(x, S) := \max_{j \in N} F_j(x, S) \quad \text{und}$$
$$f_{[\sum F_j]}(x, S) := \sum_{j \in N} F_j(x, S).$$

Wir bezeichnen mit $f_{[\sum C_j]}$ die durchschnittliche Antwortzeit, obwohl das formal nicht korrekt ist. Die Zielfunktion wird nur in einem Zusammenhang verwendet, bei dem die Anzahl der Jobs konstant ist. Zudem betrachten wir die Zielfunktionen fast ausschließlich in einem Verhältnis zwischen einem optimalen Schedule und einem Schedule mit speziellen Eigenschaften. Dabei kann die Anzahl der Jobs gekürzt werden. Deswegen wird im Folgenden der Dividend direkt weggelassen. Entsprechendes gilt für die durchschnittliche Flusszeit $f_{[\sum F_j]}$.

2.2 Online Scheduling-Probleme

Bei online Scheduling-Problemen müssen die Jobs in der Reihenfolge, in der sie in der Eingabe auftreten, auf die Maschinen verteilt werden. Dabei sind dem Algorithmus bei der Verteilung von Job j nur die schon verteilten Jobs $1, 2, \ldots, j-1$ bekannt und auf welche Maschinen diese Jobs gelegt wurden. Der Algorithmus legt nur die Maschine fest, die einen Job ausführt. Die Reihenfolge der Jobs auf einer Maschine ist eindeutig durch die Jobs, die auf diese Maschine gesetzt wurden, definiert. Die Bestimmung dieser Reihenfolge wird durch die Problembeschreibung vorgegeben. Die Verteilung des Jobs j bei Eingabe $x = ((x_1, r_1), (x_2, r_2), \ldots, (x_n, r_n)) \in \overline{\mathcal{E}}(n)$ findet zum Zeitpunkt r_j statt. Das heißt, der Algorithmus kennt die schon den Maschinen zugewiesenen Jobs und den Zustand der Maschinen zum Zeitpunkt r_j.

Aus der Problemstellung kann folgen, dass nur bestimmte Eingaben vorkommen können. So wäre es zum Beispiel möglich, dass nur Eingaben ausgewählt werden, bei denen die Jobgrößen monoton wachsen. Oder es könnte zu Beginn bekannt sein, dass man alle Jobs innerhalb eines bestehenden Zeitintervalls auf den Maschinen bearbeiten kann, wie dies beim Bin-Stretching-Problem der Fall ist (siehe zum Beispiel [AR98]). Um diese Fälle darzustellen, wird die Menge der möglichen Eingaben eingeschränkt. Sei $\overline{\mathcal{E}}' \subseteq \overline{\mathcal{E}}$ diese Menge der eingeschränkten Eingaben.

Ein Online-Algorithmus kann nicht in jedem Fall einen optimalen Schedule erzeugen, da der Algorithmus die noch folgenden Jobs nicht kennt. Wir interessieren uns nun hierbei für einen Algorithmus, bei dem bei jeder Eingabe das Verhältnis zwischen generiertem Schedule und einem optimalen Schedule möglichst klein ist. Dieses Verhältnis nennt man *kompetitives Verhältnis* und ist, wie folgt, definiert. Sei f eine Zielfunktion, $\mathcal{A}$ ein Online-Algorithmus und S_x^{OPT} der optimale Schedule für eine Eingabe $x \in \overline{\mathcal{E}}'$ bezüglich der Zielfunktion f. Das kompetitive Verhältnis von $\mathcal{A}$ ist durch

$$\sup_{x \in \overline{\mathcal{E}}'} \frac{f(x, \mathcal{A}(x))}{f(x, S_x^{\mathrm{OPT}})}$$

definiert.

Wenn nun die Einlastzeiten alle Null sind, beschreiben wir genau die Form von online Scheduling-Problemen, bei denen wir den Ablauf in zwei Teile unterteilen können: Verteilung der Jobs auf die Maschinen und Abarbeitung der Jobs auf den Maschinen. Des Weiteren sind jeder Maschine, sobald diese anfängt einen Job zu bearbeiten, alle Jobs bekannt, die auf die Maschinen verteilt wurden.

Ein Beispiel für einen Online-Algorithmus für Scheduling-Probleme mit diesen speziellen Einlastzeiten ist der List Scheduling Algorithmus von Graham [Gra66]. Der Algorithmus ist ein einfacher Greedy-Algorithmus. Gehen wir im Folgenden davon aus, dass die Jobs auf den Maschinen in der Reihenfolge, in der sie in der Eingabe vorkommen, bearbeitet werden. Das heißt, bei Eingabe $((x_1, 0), (x_2, 0), \ldots, (x_n, 0))$ ist der Zeitabschnitt, in dem Job j bearbeitet wird,

$$T_j = \left[\sum_{\substack{1 \le j' < j \\ m_{j'} = m_j}} \frac{x_{j'}}{s_{m_j}}, \sum_{\substack{1 \le j' \le j \\ m_{j'} = m_j}} \frac{x_{j'}}{s_{m_j}} \right)$$

und damit

$$C_j(S) = \sum_{\substack{1 \le j' \le j \\ m_{j'} = m_j}} \frac{x_{j'}}{s_{m_j}}.$$

Das heißt auch, dass die Jobs unterbrechungsfrei auf den Maschinen ausgeführt werden. Der List Scheduling Algorithmus ist in Abbildung 2.2 auf der nächsten Seite definiert.

Graham [Gra66] hat gezeigt, dass für identische Maschinen, das heißt $(s_i) = 1$, das kompetitive Verhältnis bei der Zielfunktion $f_{[C_{\max}]}$

$$2 - \frac{1}{m}$$

Eingabe: Folge von Jobs $x = (x_1, x_2, \ldots, x_n)$.

Ausgabe: Verteilung der Jobs auf die Maschinen $\mathcal{LS}(x) = (m_1, m_2, \ldots, m_n)$.

Verfahren:

1. $\forall i \in M : L_i := 0$ (L_i entspricht aktueller Last auf Maschine i)
2. **for** $j = 1$ **to** n **do**
   (a) Wähle Maschine $i \in M$, so dass
   $$L_i + \frac{x_j}{s_i}$$
   minimiert wird.
   (b) Setze Job j auf Maschine i: $m_j := i$.
   (c) Aktualisiere die Last auf der Maschine i: $L_i := L_i + x_j/s_i$.

Abbildung 2.2: List Scheduling

ist. Diese Schranke ist scharf.

2.3 Scheduling-Spiele

Ziel ist es, online Scheduling-Spiele zu modellieren. Das heißt, dass die Spieler ihre Aktionen nacheinander durchführen und dass die Spieler keine Informationen über zukünftige Ereignisse haben. Dazu bedienen wir uns der Spiele in Extensivform mit imperfekten Informationen.

Wir werden im Folgenden Spiele mit beliebiger, aber begrenzter Anzahl an Spielern definieren. Wir beschreiben direkt Scheduling-Spiele und geben keine allgemeine Definition von Spielen in Extensivform mit imperfekten Informationen.

Zur Modellierung von Scheduling-Spielen verwenden wir verschiedene Konstruktionen. Wir benutzen einen speziellen Spieler, der die Eingabe bestimmt. Er wird auch häufig als *Natur* bezeichnet. Die Strategie dieses Spielers ist fest vorgegeben. Wir verwenden die Natur zur Festlegung der Eingabe, um zu modellieren, dass die Spieler keine vollständige Kenntnis über alle Jobs haben. Die Spieler kennen nur die Wahrscheinlichkeitsverteilung über den

möglichen Eingaben, mit der die Natur eine Eingabe auswählt. Der Zug der Natur ist immer der erste Zug in einem Spiel.

Da die Spieler nacheinander ihre Züge durchführen, müssen wir den aktuellen Zustand eines Spieles beschreiben, den wir als *Historie* bezeichnen. Eine Historie ist eine Sequenz der durchgeführten Züge der Spieler. Durch die Historie ist eindeutig bestimmt, ob das Spiel schon zu Ende ist (alle Spieler haben einen Zug gemacht) und, falls das Spiel noch nicht zu Ende ist, welcher Spieler den nächsten Zug macht.

Wir benötigen noch ein weiteres Mittel, um online Scheduling-Spiele darzustellen. Wir müssen noch festlegen, welche Informationen einer Historie dem Spieler bekannt sind, wenn er eine Aktion auswählt. Es kann nicht die ganze Historie sein, da wir gesagt haben, dass die Historie alle schon getätigten Züge enthält und damit auch die von der Natur ausgewählte Eingabe. Für die Modellierung von online Scheduling-Spielen darf einem Spieler aber nicht die ganze Eingabe bekannt sein.

Anstelle einem Spieler nur einen Teil der Historie mitzuteilen, wird hier eine andere Modellierung gewählt. Den Spielern wird, wenn sie am Zug sind, anstelle einer einzelnen Historie eine Menge von Historien mitgeteilt, zu der die aktuelle Historie gehört. Fasst man nun alle Historien, bei denen die vorangegangenen Jobs übereinstimmen, zu dieser Menge zusammen, so hat man die Informationen von online Scheduling-Spielen modelliert.

Konkret wird dies dadurch erreicht, dass wir die Menge der Historien, an denen ein Spieler einen Zug durchführt, partitionieren und der Spieler für jede Historie einer Partition eine einheitliche Strategie auswählen muss.

Bevor wir an Beispielen verdeutlichen, wie diese Modellierung konkret durchgeführt wird, werden Scheduling-Spiele formal definiert.

Ein *(online) Scheduling-Spiel* Γ ist ein Tupel $\langle N, M, (s_i), H, Z, \rho_{\mathfrak{n}}, (\mathcal{I}_j), (C_j)\rangle$. Dabei gilt:

- $N = \{1, 2, \ldots, n\}$, $n \in \mathbb{N}$, ist die Menge der Spieler. Die Anzahl der Spieler ist immer positiv. Der Spieler Natur, auch mit $\mathfrak{n}$ bezeichnet, ist nicht in der Menge N enthalten. Wir behandeln diesen Spieler gesondert, da er eine feste Strategie spielt.

  Jeder Job gehört zu genau einem Spieler, aber nicht jeder Spieler hat einen Job. Jeder Spieler besitzt maximal einen Job. Wir erlauben es, dass ein Spieler keinen Job hat, um Unsicherheit über die Anzahl der noch folgenden Jobs erzeugen zu können, auch wenn, wie bei Online-Problemen, alle vorher getätigten Aktionen bekannt sind. Wenn einem

Spieler, alle vorangegangenen Aktionen bekannt sind, so ist ihm auch die Anzahl der Spieler, die schon einen Zug durchgeführt haben, bekannt. Wenn jetzt jeder Spieler immer genau einen Job hätte, wäre bekannt, wie viele Jobs noch folgen. Das heißt, jede Eingabe definiert $\leq n$ Jobs bzw. $\leq n$ Spieler, die an dem Spiel teilnehmen.

Diese Anzahl ist nicht identisch mit der Länge einer Eingabe bei Scheduling-Spielen. Zur allgemeinen Modellierung betrachten wir beliebige Eingaben mit $k \leq n$ Jobs. n beschränkt nur die maximale Anzahl an Spielern, die an dem Spiel teilnehmen. Die Menge der Eingaben wird mit $\mathcal{E}(n)$ bezeichnet und besteht nun nicht nur aus den Größen $x_1, x_2, \ldots, x_k \in \mathbb{N}$ und Einlastzeiten $r_1, r_2, \ldots, r_k \in \mathbb{R}_{\geq 0}$ der Jobs, sondern auch aus den Spielern $p_1, p_2, \ldots, p_k \in N$, denen diese Jobs gehören. Das heißt,

$$\begin{aligned}\mathcal{E}(n) := \{((x_1, r_1, p_1), (x_2, r_2, p_2), \ldots, (x_k, r_k, p_k)) \mid \\ 1 \leq k \leq n, (x_1, x_2, \ldots, x_k) \in \mathbb{N}^k, (r_1, r_2, \ldots, r_k) \in \mathbb{R}^k_{\geq 0}, \\ \forall 1 \leq j < k : r_j \leq r_{j+1}, |\{p_1, p_2, \ldots, p_k\}| = k, \\ \{p_1, p_2, \ldots, p_k\} \subseteq N\}.\end{aligned}$$

Da jeder Spieler höchstens einen Job besitzt, müssen alle p_j paarweise verschieden sein. Sei $h_0 = ((x_1, r_1, p_1), (x_2, r_2, p_2), \ldots, (x_k, r_k, p_k))$, dann bezeichnen wir mit $x_j(h_0) = x_j$, $r_j(h_0) = r_j$, $p_j(h_0) = p_j$ und $|h_0| = k$. Wir bezeichnen mit j auch den *Index von Spieler* $p_j(h_0)$ *in Eingabe* h_0.

Da jeder Spieler maximal einen Job besitzt, wird im Folgenden mit Job $j \in N$ sowohl der Job, als auch der Spieler, dem der Job gehört, bezeichnet.

Falls wir nur Eingaben betrachten, bei denen die Einlastzeiten der Jobs Null sind, lassen wir in der Notation die Einlastzeiten weg und schreiben $((x_1, p_1), (x_2, p_2), \ldots, (x_k, p_k)) \in \mathcal{E}(n)$ für eine Eingabe $((x_1, 0, p_1), (x_2, 0, p_2), \ldots, (x_k, 0, p_k)) \in \mathcal{E}(n)$.

- $M = \{1, 2, \ldots, m\}$, $m \in \mathbb{N}$ mit $m > 1$, ist die Menge der Maschinen.
- $(s_1, s_2, \ldots, s_m) \in \mathbb{Q}^m_+$ sind die Geschwindigkeiten der Maschinen.
- H ist die Menge der Historien. Wie oben beschrieben, modelliert eine Historie den Zustand eines Spiels. Das heißt, dass eine Historie aus allen schon getätigten Zügen besteht. Der erste Zug ist immer die Auswahl der Eingabe durch die Natur. Die folgenden Züge sind die Verteilung der Jobs auf die Maschinen. Wie bei online Scheduling-Problemen werden

die Maschinen die Jobs nach einem eindeutigen Muster abarbeiten, so dass wir den Schedule alleine durch die Maschine, auf der der Job bearbeitet werden soll, repräsentieren können. Das heißt, die Menge der Historien ist folgendermaßen definiert:

$$H = \{(h_0, h_1, h_2, \ldots, h_l) \mid \\ h_0 = ((x_1, r_1, p_1), (x_2, r_2, p_2), \ldots, (x_k, r_k, p_k)) \in \mathcal{E}(n), \\ 0 \leq l \leq k, \forall 1 \leq j \leq l : h_j \in M\} \cup \{\emptyset\}.$$

Dabei bezeichnet h_0 die ausgewählte Eingabe und h_j für $j > 0$ die Maschine, auf die der Job von Spieler $p_j(h_0)$ gelegt wurde. Die Historie $\emptyset$ bezeichnet den Spielstart, an dem noch kein Spieler inklusive der Natur einen Zug durchgeführt hat.

- $Z \subseteq H$ ist die Menge der terminalen Historien. Eine terminale Historie kennzeichnet das Ende des Spiels. Das heißt, dass alle Spieler, die einen Job besitzen, eine Maschine ausgewählt haben. Oder anders ausgedrückt, Z besteht aus all den Historien $(h_0, h_1, \ldots, h_k) \in H$ mit $h_0 = ((x_1, r_1, p_1), (x_2, r_2, p_2), \ldots, (x_k, r_k, p_k))$.

- $\rho_{\mathfrak{n}} : \mathcal{E}(n) \to \mathbb{R}_{\geq 0}$ ist eine Wahrscheinlichkeitsverteilung auf den Eingaben. Die Natur wählt genau nach dieser Verteilung zufällig eine Eingabe aus $\mathcal{E}(n)$ aus.

- Wie oben beschrieben, partitionieren wir die Menge der Historien an denen ein Spieler an der Reihe ist, um Informationen zu verschleiern. Sei dazu H_j die Menge der Historien, an denen Spieler $j \in N$ den nächsten Zug durchführt.

  $$H_j := \{(h_0, h_1, \ldots, h_l) \in H \setminus Z \mid |h_0| > l, p_{l+1}(h_0) = j\}.$$

  Die Menge $\mathcal{I}_j$ ist eine Partition der Menge H_j. Wenn ein Spieler einen Zug plant, dann kennt er nicht die Historie $h \in H_j$, in der das Spiel ist, sondern nur die Partition $I \in \mathcal{I}_j$, zu der diese Historie gehört ($h \in I$).

  Diese Partitionen modellieren somit, dass ein Spieler keine genauen Informationen über den aktuellen Spielstand hat. Wählt man die Informationspartitionen $\mathcal{I}_j$ so, dass alle Historien mit einem gleichen Präfix (ein Präfix besteht aus den Zügen der vorangegangen Jobs, deren Größe und die Größe des eigenen Jobs) in einer Informationsmenge liegen, so modelliert man die Informationen, die einem Online-Algorithmus zur Verfügung stehen würden. Das bedeutet, in diesem Fall wäre $\mathcal{I}_j$ durch

  $$\mathcal{I}_j := \{\{h \in H_j \mid \mathfrak{A}(h) = \mathfrak{a}, \mathfrak{X}(h) = \mathfrak{x}\} \mid \mathfrak{a} \in M^{|h|}, \mathfrak{x} \in \mathbb{N}^{|h|+1}\}$$

definiert, wobei

$$\mathfrak{A}(h) = \big(h_1, h_2, \ldots, h_{|h|}\big)$$

das Tupel der schon gespielten Aktionen ist und

$$\mathfrak{X}(h) = \big(x_1(h), x_2(h), \ldots, x_{|h|+1}(h)\big)$$

das Tupel der Jobgrößen der Spieler, die den ersten bis $(|h| + 1)$-ten Zug durchführen, darstellt. Spiele mit diesen Informationspartitionen bezeichnen wir auch als *online Scheduling-Spiele*.

- $C_j : Z \to \mathbb{R}$ ist die Antwortzeitfunktion von Spieler j. Sei eine terminale Historie $h \in Z$ und damit auch ein Schedule für die dazugehörige Eingabe gegeben, dann gibt $C_j(h)$ den Zeitpunkt an, zu dem Spieler j das Ergebnis seines Jobs erhält. Das ist die Zeit, die der Spieler als Fertigstellungszeitpunkt wahrnimmt.

  Über C_j werden indirekt die Maschinenmodelle, das heißt, in welcher Reihenfolge die einer Maschine zugewiesenen Jobs bearbeitet werden, definiert.

  Falls ein Spieler $j \in N$ nicht in Eingabe $h_0 \in \mathcal{E}(n)$ einer Historie $h \in Z$ vorkommt, so gilt $C_j(h) = 0$.

Mit dieser Modellierung ist es nun möglich, verschiedene Situationen abzubilden. Die folgenden Beispiele sollen verdeutlichen, dass ein breites Spektrum an Spieltypen dargestellt werden kann. Auch wenn das Ziel in der Beschreibung von online Scheduling-Spielen besteht, so bietet dieses Modell die Möglichkeit auch weitere Typen von Scheduling-Spielen zu beschreiben. Die Modellierung beschränkt sich nicht alleine auf online Spiele. Wir werden bei den folgenden Beispielen auf eine Definition des Maschinenmodells bzw. die Definition von (C_j) nicht eingehen. Hier sollen nur verschiedene Spieltypen bezüglich der bekannten Informationen beschrieben werden.

Beispiel 2.1. Bei klassischen Online-Algorithmen sind dem Algorithmus nur die schon getätigten Züge und der nächste Job der Eingabe bekannt. Über die möglichen folgenden Jobs erhalten die Spieler keine weitere Informationen. Insbesondere ist aus der Sicht der Spieler jede Eingabe mit dem schon bekannten Präfix gleich wahrscheinlich. Das heißt, dass in dieser Situation davon ausgegangen wird, dass jede mögliche Eingabe mit gleicher Wahrscheinlichkeit von der Natur ausgewählt wird.

Das bedeutet, zur Modellierung von Online-Spielen definieren wir $\rho_{\mathfrak{n}}$ durch eine Gleichverteilung auf $\mathcal{E}(n)$. Die Definition von $(\mathcal{I}_j)$ entspricht der oben

für online Scheduling-Spiele beschriebenen

$$\mathcal{I}_j := \{\{h \in H_j \mid \mathfrak{A}(h) = \mathfrak{a}, \mathfrak{X}(h) = \mathfrak{x}\} \mid \mathfrak{a} \in M^{|h|}, \mathfrak{x} \in \mathbb{N}^{|h|+1}\}.$$

Semi-Online-Modelle, wie sie von Ebenlendr und Sgall [ES11] beschrieben werden, können auch dargestellt werden. Dies wird durch eine geeignete Wahl von $\rho_{\mathfrak{n}}$ erreicht. Dazu wird $\rho_{\mathfrak{n}}$, wie in der Arbeit von Ebenlendr und Sgall beschrieben, gewählt. △

Beispiel 2.2. In dem Modell von Koutsoupias und Papadimitriou sind den Spielern die Jobs der anderen Spieler bekannt. Unbekannt sind aber die durchgeführten Aktionen der anderen Spieler. Dies kann auch durch die oben vorgestellte Modellierung von Scheduling-Spielen abgebildet werden. Dazu sei $\rho_{\mathfrak{n}}$ beliebig gewählt und die Informationspartitionen wie folgt definiert

$$\mathcal{I}_j := \{\{h \in H_j \mid h_0 = a_0\} \mid a_0 \in \mathcal{E}(n)\}.$$

Das heißt, jede Partition aus $\mathcal{I}_j$ besteht aus allen Historien, die in der Eingabe a_0 übereinstimmen.

Formal ist bei der Modellierung von Koutsoupias und Papadimitriou die Eingabe in den Spielregeln festgeschrieben und wird nicht von der Natur ausgewählt. Dies kann auch durch unsere Modellierung ausgedrückt werden, indem man für die gewünschte Eingabe $a_0 \in \mathcal{E}(n)$ die Verteilung $\rho_{\mathfrak{n}}$ so wählt, dass $\rho_{\mathfrak{n}}(a_0) = 1$. Das heißt, die Natur wählt immer die gleiche Eingabe aus. In diesem Fall könnte man die Informationspartitionen vereinfacht als $\mathcal{I}_j := \{H_j\}$ definieren, da nur eine Eingabe ausgewählt wird. △

Beispiel 2.3. Es ist auch möglich, Spiele zu modellieren, bei denen die Spieler weniger Informationen besitzen als bei klassischen Online-Modellen. Die Extremform davon ist, dass die Informationspartitionen, wie folgt, definiert sind

$$\mathcal{I}_j := \{H_j\}.$$

Das heißt, dass die Spieler blind sind und keine Informationen über die Aktionen der anderen Spieler haben. Sie haben auch keine weiteren Informationen über die von der Natur ausgewählte Eingabe, außer dass sie $\rho_{\mathfrak{n}}$ kennen.

Weniger extrem ist die Situation, die durch folgende Informationspartitionen ausgedrückt wird.

$$\mathcal{I}_j := \{\{h \in H_j \mid x_{|h|+1}(h) = x\} \mid x \in \mathbb{N}\}$$

Im Gegensatz zu der vorherigen Definition der Informationspartitionen sind den Spielern zumindest die Jobgrößen der eigenen Jobs bekannt. △

Wir führen folgende Definitionen ein, um die Notation zu vereinfachen. Sei dazu $h = a_0 a_1 \dots a_l$ eine Historie und $a_0 = ((x_1, r_1, p_1), (x_2, r_2, p_2), \dots, (x_k, r_k, p_k))$ eine Eingabe.

- $|h|$ ist die Länge der Historie. Die Länge der Historie ist die Anzahl der Züge, die die Spieler (ohne Spieler $\mathfrak{n}$) durchgeführt haben. Das heißt, $|h| = l$ und insbesondere $|\emptyset| = -1$.
- h_j mit $0 \leq j \leq |h|$ ist der Zug des j-ten Spielers, beziehungsweise der Zug des Spielers $\mathfrak{n}$ im Fall $j = 0$. Damit gilt $h_j = a_j$.
- $x_j(h)$ ist die Größe des Jobs von Spieler $p_j(a_0)$. Falls $j > |a_0|$, dann definieren wir die Größe auf Null.

$$x_j(h) := \begin{cases} x_j, & \text{falls } 1 \leq j \leq |a_0| \\ 0, & \text{sonst} \end{cases}$$

- $r_j(h)$ ist der Einlastzeitpunkt des Jobs von Spieler $p_j(a_0)$. Falls $j > |a_0|$, dann definieren wir den Einlastzeitpunkt auf Null.

$$r_j(h) := \begin{cases} r_j, & \text{falls } 1 \leq j \leq |a_0| \\ 0, & \text{sonst} \end{cases}$$

- h^j ist der j-te Präfix des Historie h. Das heißt, $h^j := h_0 h_1 \dots h_j$. Insbesondere gilt $|h^j| = j$ und $h^{-1} = \emptyset$.

Wir erweitern die Definition auf Informationspartitionen. Das heißt, sofern der entsprechende Wert für alle Historien $h \in I$ aus einer Informationsmenge $I \in \mathcal{I}_p$ eines Spielers $p \in N$ identisch ist, verwenden wir $|I|$ für die Länge der Historien aus I, I_j für den j-ten Zug, $x_j(I)$ für die Größe des j-ten Jobs und $r_j(h)$ für die Einlastzeit des j-ten Jobs.

2.3.1 Strategien und Profile

Sei im Folgenden ein beliebiges, aber festes Spiel Γ gegeben. Jeder Spieler wählt eine Strategie aus. Eine *Strategie* von Spieler $j \in N$ ist eine Abbildung, die jeder Partition $I \in \mathcal{I}_j$ eine Wahrscheinlichkeitsverteilung $\phi_j(I, \cdot) : M \to \mathbb{R}$ über die Maschinen zuweist. Wenn eine Historie $h \in I$, $I \in \mathcal{I}_j$, erreicht wird, wählt Spieler $j \in N$ die Maschine $i \in M$ mit Wahrscheinlichkeit $\phi_j(I, i)$ aus. Da in der Partition I indirekt der Spieler, der den Zug macht, codiert wird, schreiben wir auch $\phi(I, \cdot)$ anstelle von $\phi_j(I, \cdot)$.

Ein *Profil* $\phi = (\phi_1, \phi_2, \ldots, \phi_n)$ ist eine Kombination von jeweils einer Strategie für jeden Spieler.

Daraus folgt, dass eine terminale Historie $h = (h_0, h_1, \ldots, h_k) \in Z$ mit der Wahrscheinlichkeit

$$\Pr_\phi[h] = \rho_{\mathfrak{n}}(h_0) \cdot \Pi_{j=1}^{k} \phi(I((h_0, h_1, \ldots, h_{j-1})), h_j)$$

erreicht wird, wobei $I(h')$ die Informationspartition bezeichnet, die h' enthält. $I(h')$ ist eindeutig für alle $h' \in H \setminus Z$ definiert.

Jeder Spieler versucht eine Strategie ϕ_j zu wählen, so dass die erwartete Antwortzeit $\mathrm{E}_\phi[C_j]$ minimiert wird. Wir schreiben auch $C_j(\phi) = \mathrm{E}_\phi[C_j]$. Aus der Definition der Strategien folgt

$$C_j(\phi) = \sum_{h \in Z} C_j(h) \cdot \Pr_\phi[h].$$

Von besonderem Interesse sind Profile, die einen stabilen Zustand darstellen. Solche Profile bezeichnet man als Gleichgewicht. Wir werden diese verwenden, um die Güte eines Spiels zu bewerten. Dazu vergleichen wir die Lösungen, die durch ein Gleichgewicht generiert werden, mit Lösungen von einer optimalen Strategie oder eines optimalen Schedules bezüglich einer Zielfunktion. Bevor wir zu den genauen Definitionen kommen, werden in dem folgenden Abschnitt Gleichgewichte eingeführt.

2.3.2 Gleichgewichte

Wie schon oben erwähnt, stellen Gleichgewichte stabile Zustände des Spiels dar. Die bekannteste Form eines Gleichgewichts dürfte das von Nash [Nas50] eingeführte sogenannte Nash-Gleichgewicht sein.

Ein Profil $\phi = (\phi_1, \phi_2, \ldots, \phi_n)$ ist ein *Nash-Gleichgewicht* genau dann, wenn kein Spieler seine erwartete Antwortzeit durch die Wahl einer anderen Strategie verbessern kann, falls die übrigen Spieler ihre Strategien beibehalten.

Bevor wir Nash-Gleichgewichte formal definieren, führen wir zuerst eine Hilfsdefinition ein. Sei $\phi = (\phi_1, \phi_2, \ldots, \phi_n)$ ein Profil. Dann ist die Strategie ϕ'_j von Spieler j eine *bestmögliche Antwort auf* ϕ, falls

$$\forall \phi''_j : C_j(\phi_{-j}, \phi'_j) \leq C_j(\phi_{-j}, \phi''_j)$$

gilt, wobei ϕ_{-j}, ψ_j das Profil $(\phi_1, \phi_2, \ldots, \phi_{j-1}, \psi_j, \phi_{j+1}, \ldots, \phi_n)$ bezeichnet.

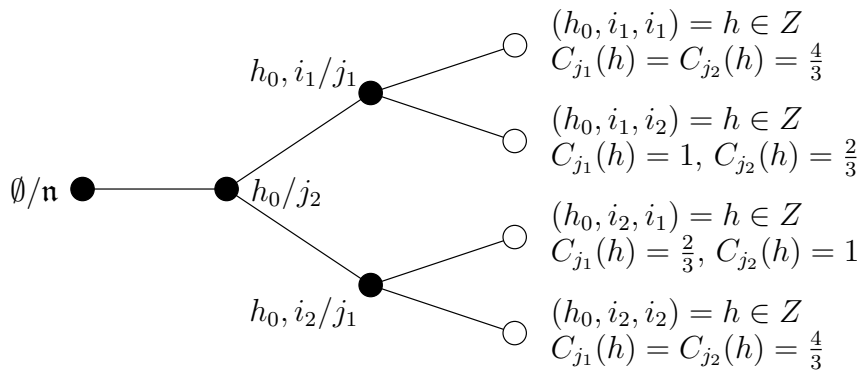

Abbildung 2.3: Beispiel für ein vereinfachtes Schedulingspiel

Ein Profil $\phi = (\phi_1, \phi_2, \ldots, \phi_n)$ heißt *Nash-Gleichgewicht* genau dann, wenn für alle Spieler $j \in N$ gilt, dass ϕ_j eine bestmögliche Antwort auf ϕ ist.

Falls ϕ ein Nash-Gleichgewicht ist, folgt aus der Definition, dass für alle Spieler $j \in N$ und alle Strategien ϕ'_j für j die Ungleichung $C_j(\phi) \leq C_j(\phi_{-j}, \phi'_j)$ gilt. Dies beschreibt genau die umgangssprachliche Definition von oben.

Beispiel 2.4. Die Abbildung 2.3 beschreibt den Spielbaum eines sehr einfachen Scheduling-Problems. Die Knoten stellen die Historien dar. Die Knoten, die die nicht-terminalen Historien repräsentieren, sind markiert mit der Historie und dem Spieler, der den nächsten Zug macht. Die Knoten der terminalen Historien sind mit der Historie und den Antwortzeiten für die Spieler markiert.

An diesem Spiel nehmen die zwei Spieler $j_1 = 1$ und $j_2 = 2$ teil. Es gibt nur eine Eingabe $h_0 = ((2, j_2), (2, j_1))$, die von der Natur ausgewählt wird. Das heißt, $\rho_{\mathfrak{n}}(h_0) = 1$. Bei dieser Eingabe ist zuerst Spieler j_2 am Zug und danach Spieler j_1. Die Einlastzeiten sind bei beiden Spielern Null. Beide haben einen Job der Größe 2. Die Informationspartitionen sind wie bei normalen online Spielen gewählt. Das heißt, dass Spieler j_1 den Zug von Spieler j_2 kennt. Da nur eine Eingabe mit positiver Wahrscheinlichkeit ausgewählt wird, ist zudem jedem Spieler von Beginn an bekannt, welche Jobs in dem Spiel vorkommen. Es gibt zwei Maschinen $i_1 = 1$ und $i_2 = 2$ mit den Geschwindigkeiten $s_1 = 3$ und $s_2 = 2$. Die Antwortzeit eines Spielers $j \in \{j_1, j_2\}$ ist die gesamte Berechnungszeit auf den Maschinen. Das heißt,

$$C_j(h_0, h_1, h_2) = \sum_{\substack{k \in \{1,2\} \\ h_k = h_j}} \frac{2}{s_{h_k}}.$$

Die folgenden zwei Profile sind Nash-Gleichgewichte dieses Spiels.

1. Erste Nash-Gleichgewicht ϕ: Spieler j_2 wählt Maschine i_1 und Spieler j_1 wählt in diesem Fall Maschine i_2. Für den Spieler j_1 muss für eine vollständige Strategie noch angegeben werden, welchen Zug er durchführt, falls Spieler j_2 die Maschine i_2 wählt. In diesem Fall wählt der Spieler die Maschine i_1. Das heißt,

$$\begin{aligned} &\phi_{j_2}(\{h_0\}, i_1) = 1, && \phi_{j_2}(\{h_0\}, i_2) = 0, \\ &\phi_{j_1}(\{(h_0, i_1)\}, i_1) = 0, && \phi_{j_1}(\{(h_0, i_1)\}, i_2) = 1, \\ &\phi_{j_1}(\{(h_0, i_2)\}, i_1) = 1 \quad \text{und} && \phi_{j_1}(\{(h_0, i_2)\}, i_2) = 0. \end{aligned}$$

2. Zweite Nash-Gleichgewicht ψ: Der Spieler j_2 wählt die Maschine i_2. Der Spieler j_1 wählt in jedem Fall die Maschine i_1 unabhängig von der Wahl des Spielers j_2. Das heißt,

$$\begin{aligned} &\psi_{j_2}(\{h_0\}, i_1) = 0, && \psi_{j_2}(\{h_0\}, i_2) = 1, \\ &\psi_{j_1}(\{(h_0, i_1)\}, i_1) = 1, && \psi_{j_1}(\{(h_0, i_1)\}, i_2) = 0, \\ &\psi_{j_1}(\{(h_0, i_2)\}, i_1) = 1 \quad \text{und} && \psi_{j_1}(\{(h_0, i_2)\}, i_2) = 0. \end{aligned}$$

Während das erste Nash-Gleichgewicht noch Sinn zu machen scheint, widerspricht das zweite der Struktur eines Spiels in Extensivform. Der Spieler j_2 kann, sofern Spieler j_1 seine Strategie nicht ändert, seine Antwortzeit nicht verbessern, wenn er die schnellere Maschine i_1 auswählt. Da aber Spieler j_2 den ersten Zug durchführt, kann er den Spieler j_1 dazu bringen, die langsamere Maschine zu wählen, da dies die beste Möglichkeit ist, falls Spieler j_2 die schnellere Maschine gewählt hat. Das zweite Nash-Gleichgewicht ψ widerspricht damit der Struktur von Spielen in Extensivform. △

Da wir nicht an solchen unnatürlichen Gleichgewichten interessiert sind, bedienen wir uns einem anderen Gleichgewichtsbegriff, der von Selten [Sel75] eingeführt wurde.

Das Problem von dem unnatürlichen Nash-Gleichgewicht ψ aus Beispiel 2.4 besteht darin, dass die Informationsmenge $\{(h_0, i_1)\}$ nicht erreicht wird und Spieler j_1 für diesen Fall mit einer Aktion dem zweiten Spieler drohen kann, die ihm auch selber Schaden zufügt. Es ist für Spieler j_1 in Informationsmenge $\{(h_0, i_1)\}$ günstiger die Maschine i_2 auszuwählen, während er in Strategie ψ_2 die Maschine i_1 auswählen würde. Man kann dieses Problem nun dadurch lösen, dass man dafür sorgt, dass jede Historie $h \in H$, bei der die Natur die Eingabe h_0 mit positiver Wahrscheinlichkeit auswählt, auch mit positiver Wahrscheinlichkeit erreicht wird.

Die von Selten beschriebenen *trembling hand perfect equilibria* – hier *perfekte Gleichgewichte* genannt – haben zudem den Vorteil, dass die dadurch beschriebenen Profile auch robust gegenüber leichten Variationen in den Strategien der Spieler sind. Zur Definition dieser führen wir erst einmal weitere Hilfsdefinitionen ein.

Sei eine Folge von Funktionen $\eta^1, \eta^2, \ldots$ gegeben, bei der jede Funktion η^k jeder Informationsmenge $I \in \mathcal{I}_j$ und Aktion $i \in M$ für jeden Spieler $j \in N$ einen Wert $\eta_j^k(I,i)$ zuweist. Dabei muss Folgendes für η^k gelten:

- $\forall j \in N : \forall I \in \mathcal{I}_j : \forall i \in M : 0 < \eta_j^k(I,i) < 1$ und
- $\forall j \in N : \forall I \in \mathcal{I}_j : \sum_{i \in M} \eta_j^k(I,i) \leq 1$.

Eine Strategie ϕ_j ist gültig für das Spiel (Γ, η^k), falls

$$\forall I \in \mathcal{I}_j : \forall i \in M : \eta_j^k(I,i) \leq \phi_j(I,i)$$

gilt. Das bedeutet, dass η^k minimale Wahrscheinlichkeiten für die möglichen Aktionen der Spieler definiert. Die Menge aller gültigen Strategien von Spieler $j \in N$ für (Γ, η^k) bezeichnen wir mit $\Phi_j^{\eta^k}$. Entsprechend ist Φ^{η^k} die Menge aller gültigen Profile für das Spiel (Γ, η^k). Sei $\Phi_j(I)$ die Menge aller Strategien in Zustand $I \in \mathcal{I}_j$. Man kann diese Menge als m-Simplex auffassen, denn

$$\Phi_j(I) := \left\{ x \in \mathbb{R}^m \,\middle|\, \sum_{i \in M} x_i = 1 \wedge \forall i \in M : x_i \geq 0 \right\}.$$

Die Einschränkung auf das Spiel (Γ, η^k) hat den Effekt, dass nur noch Strategien aus dem Inneren des Simplex zugelassen werden.

$$\Phi_j^{\eta^k}(I) := \left\{ x \in \mathbb{R}^m \,\middle|\, \sum_{i \in M} x_i = 1 \wedge \forall i \in M : x_i \geq \eta^k(I,i) \right\}.$$

Abbildung 2.4 veranschaulicht die Situation.

Durch die Definition von η^k ist sichergestellt, dass jede Aktion mit positiver Wahrscheinlichkeit gewählt wird. Wir werden solche Spiele nun verwenden, um Gleichgewichte einzuführen, die robust gegenüber solchen leichten Variationen in den Strategien der Spieler sind. Dazu betrachten wir Nash-Gleichgewichte von Spielen (Γ, η^k) und Folgen von Spielen $(\Gamma, \eta^1), (\Gamma, \eta^2), \ldots$.

Sei $\phi = (\phi_1, \phi_2, \ldots, \phi_n) \in \Phi^{\eta^k}$ ein Profil für (Γ, η^k). Dann heißt $\phi_j' \in \Phi_j^{\eta^k}$ *bestmögliche Antwort* auf ϕ, falls für alle $\phi_j'' \in \Phi_j^{\eta^k}$

$$C_j(\phi_{-j}, \phi_j') \leq C_j(\phi_{-j}, \phi_j'')$$

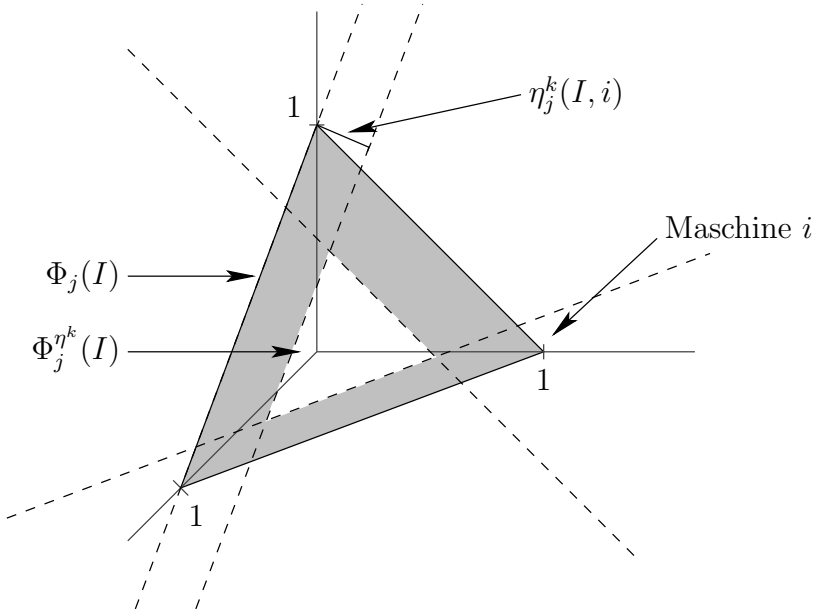

Abbildung 2.4: Menge möglicher Strategien $\Phi_j(I)$ bei Spiel Γ und $\Phi_j^{\eta^k}(I)$ bei Spiel (Γ, η^k) in Informationsmenge $I \in \mathcal{I}_j$

---

gilt. Entsprechend ist ϕ ein *Nash-Gleichgewicht* für (Γ, η^k), falls für alle Spieler $j \in N$ die Strategie ϕ_j eine bestmögliche Antwort auf ϕ ist.

Die Folge der Spiele $(\Gamma, \eta^1), (\Gamma, \eta^2), \ldots$ heißt *Testfolge für* Γ, falls

$$\lim_{k \to \infty} \eta^k = 0,$$

das heißt,

$$\forall j \in N, I \in \mathcal{I}_j, i \in M : \lim_{k \to \infty} \eta_j^k(I,i) = 0.$$

Seien nun eine Testfolge $(\Gamma, \eta^1), (\Gamma, \eta^2), \ldots$ und ein Profil ϕ von Γ gegeben. Dann heißt das Profil ϕ *Grenzwertgleichgewicht von* Γ *für die Testfolge* $(\Gamma, \eta^1), (\Gamma, \eta^2), \ldots$, falls es eine Folge von Profilen $\phi^1, \phi^2, \ldots$ gibt, so dass

- für alle $k \in \mathbb{N}$ das Profil ϕ^k ein Nash-Gleichgewicht von (Γ, η^k) ist und
- $\lim_{k \to \infty} \phi^k = \phi$.

Nun können wir perfekte Gleichgewichte definieren. Ein Profil ϕ eines Spiels Γ heißt *perfektes Gleichgewicht*, falls es eine Testfolge $(\Gamma, \eta^1), (\Gamma, \eta^2), \ldots$ für Γ gibt, so dass ϕ ein Grenzwertgleichgewicht dieser Testfolge ist.

Beispiel 2.5. Betrachten wir erneut das Spiel aus Beispiel 2.4 auf Seite 23 und die zwei Nash-Gleichgewichte

$$\begin{aligned} &\phi_{j_2}(\{h_0\}, i_1) = 1, && \phi_{j_2}(\{h_0\}, i_2) = 0, \\ &\phi_{j_1}(\{(h_0, i_1)\}, i_1) = 0, && \phi_{j_1}(\{(h_0, i_1)\}, i_2) = 1, \\ &\phi_{j_1}(\{(h_0, i_2)\}, i_1) = 1 \quad \text{und} && \phi_{j_1}(\{(h_0, i_2)\}, i_2) = 0 \end{aligned}$$

und

$$\begin{aligned} &\psi_{j_2}(\{h_0\}, i_1) = 0, && \psi_{j_2}(\{h_0\}, i_2) = 1, \\ &\psi_{j_1}(\{(h_0, i_1)\}, i_1) = 1, && \psi_{j_1}(\{(h_0, i_1)\}, i_2) = 0, \\ &\psi_{j_1}(\{(h_0, i_2)\}, i_1) = 1 \quad \text{und} && \psi_{j_1}(\{(h_0, i_2)\}, i_2) = 0. \end{aligned}$$

Beginnen wir mit dem zweiten Nash-Gleichgewicht, für das wir uns überlegt hatten, dass es keinen natürlichen Ausgang des Spiels darstellt.

Sei $(\Gamma, \eta^1), (\Gamma, \eta^2), \ldots$ eine beliebige Testfolge für Γ. Betrachten wir dazu ein beliebiges $k \in \mathbb{N}$ und das dazugehörige Spiel (Γ, η^k). Sei ϕ^k ein Nash-Gleichgewicht von (Γ, η^k). Zur einfacheren Schreibweise sei $\alpha := \phi^k_{j_2}(\{h_0\}, i_1)$, $\beta := \phi^k_{j_1}(\{(h_0, i_1)\}, i_1)$ und $\gamma := \phi^k_{j_1}(\{(h_0, i_2)\}, i_1)$. Die erwartete Antwortzeit für Spieler j_1 ist nach Definition

$$C_{j_1}(\phi^k) = \alpha \cdot \beta \cdot \frac{4}{3} + \alpha \cdot (1 - \beta) \cdot 1 + (1 - \alpha) \cdot \gamma \cdot \frac{2}{3} + (1 - \alpha) \cdot (1 - \gamma) \cdot \frac{4}{3}.$$

Für ein fest gewähltes $\alpha > 0$ wird die erwartete Antwortzeit für den ersten Spieler maximiert, falls β minimal und γ maximal gewählt wird. Das heißt, $\beta = \eta^k_{j_1}(\{(h_0, i_1)\}, i_1)$ und $\gamma = 1 - \eta^k_{j_1}(\{h_0, i_2)\}, i_2)$. Daraus folgt, dass für jede Folge von Nash-Gleichgewichten $\phi^1, \phi^2, \ldots$ der Spiele $(\Gamma, \eta^1), (\Gamma, \eta^2), \ldots$

$$\lim_{k \to \infty} \phi^k \neq \psi$$

gilt, da

$$\lim_{k \to \infty} \phi^k_{j_1}(\{h_0, i_1\}, i_1) = 0.$$

Da dies für beliebige Testfolgen und beliebige Nash-Gleichgewichte für die Spieler dieser Testfolgen gilt, folgt, dass ψ kein perfektes Gleichgewicht für Γ ist.

Das erste Nash-Gleichgewicht ϕ ist ein perfektes Gleichgewicht. Um dies zu zeigen, reicht es, eine Testfolge $(\Gamma, \eta^1), (\Gamma, \eta^2), \ldots$ zu finden, so dass ϕ ein Grenzwertgleichgewicht dieser Testfolge ist. Sei dazu die Folge $\eta^1, \eta^2, \ldots$ beliebig definiert. Wählen wir ein k beliebig, aber fest.

Betrachten wir zunächst Spieler j_1 und die Historie (h_0, i_1). In diesem Fall kann Spieler j_1 seine Antwortzeit minimieren, indem er Maschine i_2 mit der größtmöglichen Wahrscheinlichkeit auswählt. Diese Wahl ist unabhängig von der Strategie von Spieler j_2, da wir in Historie (h_0, i_1) schon die gespielte Aktion von Spieler j_2 kennen. Das heißt,

$$\phi^k_{j_1}(\{(h_0, i_1)\}, i_1) = \eta^k(\{(h_0, i_1)\}, i_1) \quad \text{und}$$
$$\phi^k_{j_1}(\{(h_0, i_1)\}, i_2) = 1 - \eta^k(\{(h_0, i_1)\}, i_1).$$

In der Historie (h_0, i_2) kann Spieler j_1 seine Antwortzeit minimieren, indem er Maschine i_1 mit der größtmöglichen Wahrscheinlichkeit auswählt. Damit erhalten wir

$$\phi^k_{j_1}(\{(h_0, i_2)\}, i_1) = 1 - \eta^k(\{(h_0, i_2)\}, i_2) \quad \text{und}$$
$$\phi^k_{j_1}(\{(h_0, i_2)\}, i_2) = \eta^k(\{(h_0, i_2)\}, i_2).$$

Nach der Definition von $\phi^k_{j_1}$ folgt $\lim_{k\to\infty} \phi^k_{j_1} = \phi_{j-1}$. Da beide Historien (h_0, i_1) und (h_0, i_2) mit positiver Wahrscheinlichkeit in einem Spiel (Γ, η^k) erreicht werden, muss Spieler j_1 in einem Nash-Gleichgewicht die Strategie $\phi^k_{j_1}$ spielen, da er sich ansonsten verbessern könnte.

Nun müssen wir nur noch die Strategie von Spieler j_2 betrachten. Dieser Spieler macht nur in Historie h_0 einen Zug. Seien $\alpha^k := \phi^k_{j_1}(\{(h_0, i_1)\}, i_1)$, $\beta^k := \phi^k_{j_1}(\{(h_0, i_2)\}, i_1)$ und $\gamma^k := \phi^k_{j_2}(\{h_0\}, i_1)$. Die Antwortzeit von Spieler j_2 ist

$$\begin{aligned} C_{j_2}(\phi^k) &= \gamma^k\alpha^k\frac{4}{3} + \gamma^k(1-\alpha^k)\frac{2}{3} + (1-\gamma^k)\beta^k + (1-\gamma^k)(1-\beta^k)\frac{4}{3} \\ &= \gamma^k\left(\alpha^k\frac{4}{3} + (1-\alpha^k)\frac{2}{3}\right) + (1-\gamma^k)\left(\beta^k + (1-\beta^k)\frac{4}{3}\right). \end{aligned}$$

Da $\lim_{k\to\infty} \phi^k_{j_1} = \phi_{j_1}$ und damit $\alpha^k \to 0$, gibt es ein $K \in \mathbb{N}$, so dass für alle $k' \geq K$

$$\alpha^{k'}\frac{4}{3} + (1-\alpha^{k'})\frac{2}{3} < \beta^{k'} + (1-\beta^{k'})\frac{4}{3}$$

gilt. Damit folgt, dass es ein $K \in \mathbb{N}$ gibt, so dass für alle $k' \geq K$ das Profil $\phi^{k'}$ ein Nash-Gleichgewicht für das Spiel $(\Gamma, \eta^{k'})$ darstellt, falls $\phi^{k'}_{j_1}$ wie oben definiert ist und Spieler j_2 mit größtmöglicher Wahrscheinlichkeit Maschine i_1 auswählt. Das heißt,

$$\phi^{k'}_{j_2}(\{h_0\}, i_1) = 1 - \eta^{k'}(\{h_0\}, i_2) \quad \text{und}$$
$$\phi^{k'}_{j_2}(\{h_0\}, i_2) = \eta^{k'}(\{h_0\}, i_2).$$

Das heißt, dass ϕ ein Grenzwertgleichgewicht für die Testfolge (Γ, η^K), $(\Gamma, \eta^{K+1}), \ldots$ ist und damit ein perfektes Gleichgewicht von Γ darstellt. $\triangle$

Das Beispiel zeigt, dass wir mit dem Begriff eines perfekten Gleichgewichts das ungewünschte Nash-Gleichgewicht aussortiert haben. Zudem weisen perfekte Gleichgewichte eine gewissen Robustheit gegenüber leichten Variationen in den Strategien der Spieler auf. Dies wird durch die Existenz einer Testfolge mit den gewünschten Eigenschaften sichergestellt.

Für endliche Spiele in Extensivform mit imperfekten Informationen und perfekter Erinnerung ist bekannt, dass es immer ein perfektes Gleichgewicht gibt [Sel75]. Perfekte Erinnerung bedeutet, dass ein Spieler vollständiges Wissen über seine eigenen schon getätigten Züge hat. Da in den Scheduling-Spielen jeder Spieler höchstens einen Zug durchführt, hat jedes Scheduling-Spiel diese Eigenschaft. Das heißt, diese Aussage gilt für endliche Scheduling-Spiele und wir erhalten folgendes Korollar.

Korollar 2.1. *Sei* $\Gamma = \langle N, M, (s_i), H, Z, \rho_{\mathfrak{n}}, (\mathcal{I}_j), (C_j)\rangle$ *ein Scheduling-Spiel mit*

$$\exists x_{\max} \in \mathbb{N} : \forall 1 \leq k \leq n,$$
$$a_0 = ((x_1, r_1, p_1), (x_2, r_2, p_2), \ldots, (x_k, r_k, p_k)) \in \mathcal{E}(k, n) :$$
$$\forall 1 \leq l \leq k : \rho_{\mathfrak{n}}(a_0) > 0 \Rightarrow x_l \leq x_{\max}.$$

Dann hat Γ *mindestens ein perfektes Gleichgewicht.*

Die Bedingung an die Strategie $\rho_{\mathfrak{n}}$ aus dem vorstehenden Korollar bedeutet, dass die Größe der Jobs von vorkommenden Eingaben von oben beschränkt ist und damit nur endlich viele verschiedene Eingaben mit positiver Wahrscheinlichkeit von der Natur ausgewählt werden.

Weiterhin ist bekannt, dass jedes perfekte Gleichgewicht auch ein Nash-Gleichgewicht ist. Dies kann folgendermaßen eingesehen werden. Sei Γ ein Scheduling-Spiel und ϕ ein perfektes Gleichgewicht von Γ. Nehmen wir an, ϕ sei kein Nash-Gleichgewicht. Dann existiert ein Spieler $j \in N$ und eine Strategie ψ_j von Spieler j, so dass

$$\mathrm{E}_{\phi}[C_j] > \mathrm{E}_{\phi_{-j},\psi_j}[C_j].$$

Da ϕ ein perfektes Gleichgewicht ist, gibt es die Folgen (η^k) und (ϕ^k), so dass

- $\lim_{k\to\infty} \eta^k = 0$,
- $\lim_{k\to\infty} \phi^k = \phi$ und
- ϕ^k für jedes $k \in \mathbb{N}$ ein Nash-Gleichgewicht von (Γ, η^k) ist.

Aus der Stetigkeit des Erwartungswertes folgt, dass

$$\lim_{k\to\infty} \mathrm{E}_{\phi^k}[C_j] = \mathrm{E}_{\phi}[C_j].$$

Wir können nun eine Folge ψ_j^k definieren mit

- $\lim_{k\to\infty} \psi_j^k = \psi_j$ und
- ψ_j^k ist für alle $k \in \mathbb{N}$ eine gültige Strategie von Spieler j im Spiel (Γ, η^k).

Daraus folgt, dass

$$\lim_{k\to\infty} \mathrm{E}_{\phi^k_{-j},\psi_j^k}[C_j] = \mathrm{E}_{\phi_{-j},\psi_j}[C_j].$$

Aus der Definition des Grenzwertes folgt, dass es ein $k \in \mathbb{N}$ gibt, so dass ϕ^k kein Nash-Gleichgewicht von (Γ, η^k) ist. Diese Konstruktion gilt für beliebige Testfolgen. Damit ist ϕ kein perfektes Gleichgewicht. Das ist ein Widerspruch zur Annahme und damit ist ϕ ein Nash-Gleichgewicht. Dies impliziert:

Fakt 2.1. *Sei $\Gamma = \langle N, M, (s_i), H, Z, \rho_{\mathfrak{n}}, (\mathcal{I}_j), (C_j)\rangle$ ein Scheduling-Spiel und ϕ ein perfektes Gleichgewicht von Γ. Dann ist ϕ ein Nash-Gleichgewicht von Γ.*

Wie wir an Beispiel 2.4 gesehen haben, ist nicht jedes Nash-Gleichgewicht ein perfektes Gleichgewicht. Allerdings ist jedes vollständig gemischte Nash-Gleichgewicht ein perfektes Gleichgewicht. Ein Profil ϕ heißt vollständig gemischt, wenn jeder Spieler $j \in N$ in jeder Informationsmenge $I \in \mathcal{I}_j$ jede Maschine i mit positiver Wahrscheinlichkeit $\phi_j(I, i) > 0$ auswählt. In diesem Fall gibt es eine Folge $\eta^1, \eta^2, \ldots$ mit $\lim_{k\to\infty} \eta^k = 0$, so dass ϕ ein gültiges Profil von (Γ, η^k) für alle $k \in \mathbb{N}$ ist. Damit ist die Folge $\phi^1 = \phi, \phi^2 = \phi, \ldots$ eine Folge von Nash-Gleichgewichten für die Testfolge $(\Gamma, \eta^1), (\Gamma, \eta^2), \ldots$. Offensichtlich ist ϕ das Grenzwertgleichgewicht dieser Testfolge und damit ein perfektes Gleichgewicht.

Fakt 2.2. *Sei $\Gamma = \langle N, M, (s_i), H, Z, \rho_{\mathfrak{n}}, (\mathcal{I}_j), (C_j)\rangle$ ein Scheduling-Spiel und ϕ ein vollständig gemischtes Nash-Gleichgewicht von Γ. Dann ist ϕ ein perfektes Gleichgewicht von Γ.*

2.3.3 Modelle

So wie wir Scheduling-Spiele definiert haben, ist die maximale Anzahl an Spielern für jedes Spiel fest vorgegeben. Dies hat den Vorteil, dass die Spiele

endlich sind, sofern wir die Größen der Jobs begrenzen, und es damit unabhängig von dem Maschinenmodell und der Informationspartitionen immer ein perfektes Gleichgewicht gibt. Modelle, oder auch Spielmodelle, führen wir ein, um diese Einschränkung in der Analyse von Scheduling-Spielen zu umgehen.

Wir werden im Folgenden Modelle verwenden, um Scheduling-Spiele zu beschreiben. Dies erlaubt es uns, von der maximalen Anzahl an Spielern, die an einem Spiel teilnehmen, zu abstrahieren.

Um die Einschränkung der maximalen Anzahl der Spieler in einem Scheduling-Spiel zu umgehen, werden wir Familien von Scheduling-Spielen betrachten, die sich ausschließlich durch die maximale Anzahl der Spieler unterscheiden. Als ein Modell bezeichnen wir nicht die Familie selber, sondern eine Abbildung von der maximalen Anzahl an Spielern auf ein Scheduling-Spiel dieser Familie mit der entsprechenden maximalen Anzahl an Spielern.

Sei dazu $\mathfrak{G}$ die Menge aller Scheduling-Spiele. Formal ist ein Modell μ von Scheduling-Spielen eine Funktion $\mu : \mathbb{N} \to \mathfrak{G}$ mit folgenden Eigenschaften: Für alle $n, n' \in \mathbb{N}$, $\mu(n) = \Gamma = \langle N, M, (s_i), H, Z, \rho_\mathfrak{n}, (\mathcal{I}_j), (C_j)\rangle$ und $\mu(n') = \Gamma' = \langle N', M', (s'_i), H', Z', \rho_\mathfrak{n}', (\mathcal{I}'_j), (C'_j)\rangle$ gilt

- $N = \{1, 2, \ldots, n\}$ und $N' = \{1, 2, \ldots, n'\}$.
- Die Anzahl und Geschwindigkeit der Maschinen ist unabhängig von der Anzahl der Spieler: $M = M'$ und $(s_i) = (s'_i)$.
- Die Menge der (terminalen) Historien von Γ ist eine Teilmenge der (terminalen) Historien von Γ': $H \subset H'$ und $Z \subset Z'$.
- Die Antwortzeitfunktionen (C'_j) sind eine Erweiterung der Antwortzeitfunktionen (C_j), so dass $C_j(h) = C'_j(h)$ für alle $h \in Z$.
- $\rho_\mathfrak{n}'$ ist eine Erweiterung von $\rho_\mathfrak{n}$ auf die Eingaben $\mathcal{E}(n')$. Das heißt, für alle $a_0 \in \mathcal{E}(n)$ gilt:

$$\rho_\mathfrak{n}(a_0) = \frac{\rho_\mathfrak{n}'(a_0)}{\sum_{a'_0 \in \mathcal{E}(n)} \rho_\mathfrak{n}'(a_0)}.$$

- Die Informationspartitionen $(\mathcal{I}'_j)$ sind eine Erweiterung von $(\mathcal{I}_j)$. Das heißt, für alle $j \in N$ und $I \in \mathcal{I}_j$ existiert ein $I' \in \mathcal{I}'_j$ mit $I = I' \cap H$.

2.3.4 Preis der Anarchie und Koordinationsverhältnis

Wie wir oben erläutert haben, sind wir an stabilen Spielzuständen, den perfekten Gleichgewichten, interessiert. Den Preis der Anarchie und das Koordinationsverhältnis führen wir als Maß für die Güte von Spielen ein. Es ist eine Variation des kompetitiven Verhältnisses.

Wir betrachten Scheduling-Probleme als Minimierungsprobleme. Das heißt, wir haben eine Zielfunktion, die es zu minimieren gilt. Ein Beispiel für eine Zielfunktion ist die durchschnittliche Antwortzeit aller Jobs. Wir werden immer den Wert der Zielfunktion für eine feste Eingabe eines perfekten Gleichgewichtes mit einer optimalen Lösung für diese Eingabe vergleichen.

Hierbei gibt es zwei Möglichkeiten eine optimale Lösung zu definieren. Zum einen können wir als eine optimale Lösung ein Profil definieren, das bei der gegebenen Eingabe die Zielfunktion minimiert. Das heißt, dass diese Lösung, genau wie die Spieler, durch das Maschinenmodell des Spiels eingeschränkt ist. Die zweite Möglichkeit besteht darin, diese Einschränkung aufzuheben. In diesem Fall ist die optimale Lösung ein beliebiger Schedule für die gegebene Eingabe.

Als Preis der Anarchie bezeichnen wir das Verhältnis zwischen dem Wert der Zielfunktion für ein perfektes Gleichgewicht, das unter allen perfekten Gleichgewichten die Zielfunktion maximiert, und eines Profils, das diese Zielfunktion minimiert. Im Fall des Koordinationsverhältnisses verwenden wir anstelle eines beliebigen Profils einen beliebigen Schedule, der die Zielfunktion minimiert.

Formal ist *der erwartete Wert* $\mathrm{E}f(\Gamma, a_0, \phi)$ *der Zielfunktion* f *bei Eingabe* a_0 *und Profil* ϕ durch

$$\mathrm{E}f(\Gamma, a_0, \phi) = \sum_{\substack{h \in Z \\ h_0 = a_0}} \Pr_\phi[h|a_0] f(a_0, C(h)),$$

definiert, wobei $C(h) = (C_1(h), C_2(h), \ldots, C_n(h))$ das Tupel der Antwortzeiten der Spieler ist.

Der Wert der Zielfunktion für ein perfektes Gleichgewicht, das unter allen perfekten Gleichgewichten die Zielfunktion für die gegebene Eingabe a_0 maximiert, ist demnach

$$\sup\{\, \mathrm{E}f(\Gamma, a_0, \phi) \mid \phi \text{ ist ein perfektes Gleichgewicht}\}.$$

Es ist das für die Allgemeinheit ungünstigste Ergebnis bei Eingabe a_0, falls die Spieler sich wie in perfekten Gleichgewichten verhalten. Da perfekte

Gleichgewichte einen stabilen Zustand des Spiels repräsentieren, ist es das für die Allgemeinheit ungünstigste Ergebnis in einem stabilen Zustand.

Betrachten wir nun den Preis der Anarchie. Dazu definieren wir zuerst den Preis der Anarchie für eine feste Eingabe a_0 eines festes Spiels Γ. Danach erweitern wir die Definition auf Spielmodelle und beliebige Eingaben. Der *Preis der Anarchie für Spiel* Γ *bei Eingabe* a_0 *für Zielfunktion* f ist

$$\mathrm{PA}(\Gamma, a_0, f) := \frac{\sup\{\,\mathrm{E}f(\Gamma, a_0, \phi) \mid \phi \text{ ist ein perfektes Gleichgewicht}\}}{\inf\{\,\mathrm{E}f(\Gamma, a_0, \phi) \mid \phi \text{ ist ein beliebiges Profil}\}}. \tag{2.1}$$

Damit ist der *Preis der Anarchie für das Modell* μ *und Zielfunktion* f

$$\mathrm{PA}(\mu, f) := \limsup_{n\to\infty} \sup_{\substack{a_0 \in \mathcal{E}(n) \\ \rho_{\mathfrak{n}}(a_0) > 0}} \mathrm{PA}(\mu(n), a_0, f). \tag{2.2}$$

Wie wir hier sehen, ist der Preis der Anarchie das Verhältnis zwischen dem ungünstigsten Wert der Zielfunktion für stabile Spielzustände und einem optimalen Schedule, der sich an die Spielregeln halten muss.

Im Gegensatz zum Preis der Anarchie vergleichen wir beim Koordinationsverhältnis nicht die stabilen Zustände mit einem die Zielfunktion minimierenden Profil, sondern mit einem beliebigen Schedule, der die Zielfunktion minimiert. Das *Koordinationsverhältnis für Spiel* Γ *bei Eingabe* a_0 *für die Zielfunktion* f und das *Koordinationsverhältnis für Modell* μ *und Zielfunktion* f sind wie folgt definiert:

$$\mathrm{CR}(\Gamma, a_0, f) := \frac{\sup\{\,\mathrm{E}f(\Gamma, a_0, \phi) \mid \phi \text{ ist ein perfektes Gleichgewicht}\}}{\inf\{f(C(S)) \mid S \text{ ist ein Schedule}\}}$$

und

$$\mathrm{CR}(\mu, f) := \limsup_{n\to\infty} \sup_{\substack{a_0 \in \mathcal{E}(n) \\ \rho_{\mathfrak{n}}(a_0) > 0}} \mathrm{CR}(\mu(n), a_0, f).$$

Hierbei ist $C(S)$ das Tupel der Antwortzeiten, die durch den Schedule S definiert werden.

Wir werden in Kapitel 4 erläutern, dass ein großer Unterschied zwischen dem Preis der Anarchie und dem Koordinationsverhältnis darauf hindeutet, dass das Spielmodell die Spieler zu bestimmten Aktionen zwingt. Dies kann so weit gehen, dass die Spieler keine richtige Wahl mehr haben, da sie ansonsten in jedem Fall ihre Antwortzeit verschlechtern.

Wir werden uns im Folgenden insbesondere mit zwei Zielfunktionen beschäftigen. Dies sind die *maximale Antwortzeit* $f_{[C_{\max}]}$ und die *gesamte*

oder auch *durchschnittliche Antwortzeit* $f_{[\sum C_j]}$.

$$f_{[C_{\max}]}(a_0, (C_1, C_2, \ldots, C_n)) := \max_{1 \leq j \leq n} C_j.$$
$$f_{[\sum C_j]}(a_0, (C_1, C_2, \ldots, C_n)) := \sum_{1 \leq j \leq n} C_j.$$

$f_{[\sum C_j]}$ wird üblicherweise auch als durchschnittliche Antwortzeit bezeichnet, obwohl sie als gesamte Antwortzeit definiert ist. Da die Zielfunktionen nur im Zusammenhang mit dem Preis der Anarchie, dem Koordinationsverhältnis, dem kompetitiven Verhältnis oder der Approximationsgüte vorkommen und dort die Anzahl der Spieler fix ist, wird der Einfachheit halber der Divisor von vornherein weggelassen.

Falls alle Einlastzeiten Null sind, macht es Sinn die maximale Antwortzeit $f_{[C_{\max}]}$ und durchschnittliche Antwortzeit $f_{[\sum C_j]}$ zu betrachten. Andernfalls allerdings werden sich die Spieler mehr für die Flusszeit interessieren. Das heißt, für die Zeit die zwischen dem Einlastzeitpunkt und dem Zeitpunkt, an dem sie die Antwort auf ihren Job erhalten, vergeht. Die Flusszeit $F_j(a_0, C)$ ist bei Eingabe $a_0 \in \mathcal{E}(n)$ und Antwortzeiten $C = (C_1, C_2, \ldots, C_n)$ formal durch

$$F_j(a_0, C) := \begin{cases} C_j - r_l(a_0), & \text{falls es ein } 1 \leq l \leq |a_0| \text{ mit } p_l = j \text{ gibt} \\ 0, & \text{sonst} \end{cases}$$

definiert. Hier sorgen wir dafür, dass die Flusszeit für alle Spieler, die in Eingabe a_0 keinen Job besitzen, Null ist. Da jeder Spieler nur einen Job besitzt, gibt es keine zwei Züge $1 \leq l < k \leq |a_0|$ mit $p_l = j$ und $p_k = j$. Das heißt, falls Spieler j einen Zug bei Eingabe a_0 durchführt, so ist die Nummer des Zuges l eindeutig durch diese Eingabe a_0 definiert.

Mit dieser Definition der Flusszeiten können wir nun die Zielfunktionen von oben abwandeln, so dass wir die Flusszeiten anstelle der Antwortzeiten betrachten.

$$f_{[F_{\max}]}(a_0, C) := \max_{1 \leq j \leq n} F_j(a_0, C).$$
$$f_{[\sum F_j]}(a_0, C) := \sum_{1 \leq j \leq n} F_j(a_0, C).$$

Die Flusszeit ist identisch zur Antwortzeit, falls die Einlastzeit des Jobs Null ist. Das heißt, falls wir nur Modelle μ betrachten, bei denen die Einlastzeiten

Null sind, gilt $\mathrm{PA}(\mu, f_{[C_{\max}]}) = \mathrm{PA}(\mu, f_{[F_{\max}]})$, $\mathrm{CR}(\mu, f_{[C_{\max}]}) = \mathrm{CR}(\mu, f_{[F_{\max}]})$, $\mathrm{PA}(\mu, f_{[\sum C_j]}) = \mathrm{PA}(\mu, f_{[\sum F_j]})$ und $\mathrm{CR}(\mu, f_{[\sum C_j]}) = \mathrm{CR}(\mu, f_{[\sum F_j]})$.

Im Gegensatz zum Preis der Anarchie vergleichen wir bei der Bildung des Koordinationsverhältnisses die Ergebnisse bezüglich einer Zielfunktion nicht mit einer Zielfunktion-minimierenden Strategie, sondern mit einem optimalen Schedule. Da der Schedule sich nicht an das vom Spiel festgelegte Maschinenmodell halten muss, wird man erwarten, dass der Preis der Anarchie immer kleiner oder gleich dem Koordinationsverhältnis ist.

Wir werden dies formal nicht nur für die Zielfunktionen $f_{[C_{\max}]}$, $f_{[\sum C_j]}$, $f_{[F_{\max}]}$ und $f_{[\sum F_j]}$ zeigen, sondern für eine Klasse von Zielfunktionen, zu denen diese beiden gehören.

Sei $\mathfrak{F}$ die Menge der Zielfunktionen, die folgende Monotonieeigenschaft erfüllen:

$$\forall f \in \mathfrak{F} : \forall n \in \mathbb{N} : \forall a_0 \in \mathcal{E}(n) : \forall C, C' \in \mathbb{Q}_+^n : \\ (\forall 1 \leq j \leq n : C_j \leq C'_j) \Rightarrow f(a_0, C) \leq f(a_0, C').$$

Das bedeutet, falls sich die Antwortzeiten verschlechtern, dann soll sich auch der Wert der Zielfunktion verschlechtern.

Nach Definition erfüllen die Zielfunktionen $f_{[C_{\max}]}$, $f_{[F_{\max}]}$, $f_{[\sum C_j]}$ und $f_{[\sum F_j]}$ diese Eigenschaft. Die Monotonie ist eine natürliche Bedingung für Zielfunktionen. Eine Zielfunktion, deren Bewertung steigt, falls die Antwortzeiten sich verschlechtern, macht keinen Sinn.

Aus der Definition von $\mathrm{PA}(\mu, f)$ und $\mathrm{CR}(\mu, f)$ und der Tatsache, dass für monotone Zielfunktionen f, Spiele Γ und eine Eingabe a_0

$$\inf\{f(a_0, C(S)) \mid S \text{ ist Schedule}\} \leq \inf\{\mathrm{E} f(\Gamma, a_0, \phi) \mid \phi \text{ ist ein Profil}\}$$

gilt, folgt folgendes Lemma.

Lemma 2.1. *Seien ein Modell μ und eine Zielfunktion $f \in \mathfrak{F}$ gegeben. Dann gilt*

$$\mathrm{PA}(\mu, f) \leq \mathrm{CR}(\mu, f).$$

Kapitel 3

Das KP-Modell

Die Form von Scheduling-Spielen, die von Koutsoupias und Papadimitriou [KP99] eingeführt wurden, ist, wie schon beschrieben, ein Spezialfall der in dieser Arbeit vorgestellten Scheduling-Spiele.

In dem sogenannten KP-Modell von Koutsoupias und Papadimitriou sind jedem Spieler die Jobs der anderen Spieler bekannt. Die Spieler wählen ohne Wissen über die Strategien der anderen Spieler ihre eigene Strategie aus. Formal ist ein *KP-Modell-Scheduling-Spiel* ein Tupel $\Gamma^{\mathrm{KP}} = \langle N, M, (x_j), (s_i) \rangle$ mit

- der Menge der Spieler $N = \{1, 2, \ldots, n\}$,
- der Menge der Maschinen $M = \{1, 2, \ldots, m\}$,
- den Größen der Jobs $(x_1, x_2, \ldots, x_n) \in \mathbb{N}^n$ und
- den Geschwindigkeiten der Maschinen $(s_1, s_2, \ldots, s_m) \in \mathbb{Q}_+^m$.

Bei diesem Modell sind die Einlastzeiten aller Jobs Null.

Eine Strategie ϕ_j eines Spielers $j \in N$ für das Spiel Γ^{KP} ist eine Wahrscheinlichkeitsverteilung auf M. Das heißt, ein Spieler wählt nach dieser Wahrscheinlichkeitsverteilung eine Maschine $i \in M$ aus, auf der der eigene Job bearbeitet werden soll. Eine Strategie ϕ_j heißt *rein*, falls es eine Maschine $i \in M$ gibt, so dass $\phi_j(i) = 1$. Seien $\phi_1, \phi_2, \ldots, \phi_n$ Strategien aller Spieler für das Spiel Γ^{KP}. Dann bezeichnen wir $\phi = (\phi_1, \phi_2, \ldots, \phi_n)$ als Profil von Γ^{KP}.

Die erwartete Antwortzeit $C_j^{\mathrm{KP}}(\phi)$ eines Spielers $j \in N$, falls das Profil ϕ von den Spielern gespielt wird, ist im KP-Modell, wie folgt, definiert:

$$C_j^{\mathrm{KP}}(\phi) = \sum_{i \in M} \phi_j(i) \frac{1}{s_i} \left(x_j + \sum_{k \in N \setminus \{j\}} \phi_k(i) x_k \right).$$

Im Fall von reinen Strategien $\phi_1, \phi_2, \ldots, \phi_n$ bedeutet dies, dass die Antwortzeit der Bearbeitungszeit aller Jobs auf der ausgewählten Maschine entspricht.

Wir werden nun folgenden Satz zeigen, der besagt, dass zu jedem KP-Modell-Scheduling-Spiel ein äquivalentes Scheduling-Spiel existiert. Dazu definieren wir für ein KP-Modell-Scheduling-Spiel $\Gamma^{\mathrm{KP}} = \langle N, M, (x_j), (s_i) \rangle$ ein Scheduling-Spiel $\Gamma(\Gamma^{\mathrm{KP}}) = \langle N, M, (s_i), H, Z, \rho_{\mathfrak{n}}, (\mathcal{I}_j), (C_j) \rangle$. Wir übernehmen dazu die Mengen der Spieler und Maschinen und die Geschwindigkeit der Maschinen. Die Menge der Historien und terminalen Historien ist eindeutig durch die Anzahl der Spieler und Maschinen bestimmt. Des Weiteren definieren wir:

- Sei $a_0 = ((x_1, 0, 1), (x_2, 0, 2), \ldots, (x_n, 0, n)) \in \mathcal{E}(n)$ die Eingabe, die die feste Eingabe des KP-Modell-Scheduling-Spiels repräsentiert. Wie schon beschrieben sind alle Einlastzeiten Null. Des Weiteren definieren wir, dass der Spieler $j \in N$ den j-ten Zug durchführt. Wir könnten hier eine beliebige Reihenfolge für die Spieler wählen, nehmen allerdings diese spezielle, da wir so die Notation später vereinfachen.

- Die Natur wählt immer die Eingabe a_0 aus. Das heißt, $\rho_{\mathfrak{n}}(a_0) = 1$. Da alle Spieler die Wahrscheinlichkeitsverteilung $\rho_{\mathfrak{n}}$ kennen, ist allen Spielern die komplette Eingabe bekannt.

- Die Spieler haben keine Informationen über schon getätigte Züge. Da durch die Wahl von $\rho_{\mathfrak{n}}$ schon die komplette Eingabe bekannt ist, können wir $\mathcal{I}_j$ wie folgt wählen: $\mathcal{I}_j := \{H_j\}$. Damit muss ein Spieler unabhängig von jedem anderen Zug eine Maschine auswählen. Dies ist genau die Situation bei KP-Modell-Scheduling-Spielen.

- Die Antwortzeitfunktionen sind durch

$$C_j(h) = \sum_{\substack{k \in N \\ h_j = h_k}} \frac{x_k}{s_{h_j}}$$

  definiert.

Satz 3.1. *Seien* $\Gamma^{\mathrm{KP}} = \langle N, M, (x_j), (s_i) \rangle$ *ein KP-Modell-Scheduling-Spiel und* $\Gamma = \Gamma(\Gamma^{\mathrm{KP}})$ *das dazugehörige Scheduling-Spiel.*

Dann gilt

1. *für alle Profile* ϕ^{KP} *von* Γ^{KP} *existiert ein Profil* ϕ *von* Γ *mit*

$$\forall j \in N : C_j^{\mathrm{KP}}(\phi^{\mathrm{KP}}) = C_j(\phi) \quad \textit{und}$$

2. *für alle Profile* ϕ *von* Γ *existiert ein Profil* ϕ^{KP} *von* Γ *mit*

$$\forall j \in N : C_j(\phi) = C_j^{\mathrm{KP}}(\phi^{\mathrm{KP}}).$$

Beweis. Zum Beweis müssen nur die entsprechenden Profile definiert und dann nachgerechnet werden. Gegeben sei ein Profil ϕ^{KP} und definieren wir ϕ durch

$$\phi_j(H_j, i) := \phi_j^{\mathrm{KP}}(i).$$

Setzen wir dieses Profil nun in die Definition der erwarteten Antwortzeit für Scheduling-Spiele ein, dann erhalten wir

$$\begin{aligned} C_j(\phi) &= \sum_{h \in Z} \left(\rho_{\mathfrak{n}}(h_0) \prod_{k=1}^{n} \phi_k(H_k, h_k) \right) C_j(h) \\ &= \sum_{\substack{h \in Z \\ \rho_{\mathfrak{n}}(h_0)=1}} \left(\prod_{k=1}^{n} \phi_k(H_k, h_k) \right) C_j(h) \\ &= \sum_{\substack{h \in Z \\ \rho_{\mathfrak{n}}(h_0)=1}} \left(\prod_{k=1}^{n} \phi_k(H_k, h_k) \right) \sum_{\substack{k \in N \\ h_j = h_k}} \frac{x_k}{s_{h_j}}. \end{aligned}$$

Wir ziehen nun den Zug von Spieler j aus der rechten Seite heraus. Dazu summieren wir über alle möglichen Aktionen des Spielers. Das heißt, wir summieren über die Maschinen. Sei zur Vereinfachung der Notation $Z_{j,i}$ die Menge der terminalen Historien, bei denen Spieler $j \in N$ die Maschine i auswählt und die Eingabe von der Natur mit positiver Wahrscheinlichkeit gewählt wird. Das heißt,

$$Z_{j,i} := \{h \in Z \mid \exists l : p_l(h_0) = j, h_l = i, \rho_{\mathfrak{n}}(h_0) = 1\}.$$

Damit erhalten wir

$$\begin{aligned} C_j(\phi) &= \sum_{\substack{h\in Z\\ \rho_{\mathrm{n}}(h_0)=1}} \left(\prod_{k=1}^{n} \phi_k(H_k,h_k)\right) \sum_{\substack{k\in N\\ h_j=h_k}} \frac{x_k}{s_{h_j}} \\ &= \sum_{\substack{h\in Z\\ \rho_{\mathrm{n}}(h_0)=1}} \left(\phi_j(H_j,h_j)\cdot\prod_{k\neq j} \phi_k(H_k,h_k)\right) \sum_{\substack{k\in N\\ h_j=h_k}} \frac{x_k}{s_{h_j}} \\ &= \sum_{i\in M}\sum_{h\in Z_{j,i}} \phi_j(H_j,i)\left(\prod_{k\neq j} \phi_k(H_k,h_k)\right) \sum_{\substack{k\in N\\ h_k=i}} \frac{x_k}{s_i}. \end{aligned}$$

Da nun $\phi_j(H_j,i)$ unabhängig von h ist, können wir dies aus der Summe herausziehen. Das Gleiche gilt auch für den Faktor $\frac{1}{s_i}$ und somit erhalten wir

$$\begin{aligned} C_j(\phi) &= \sum_{i\in M}\sum_{h\in Z_{j,i}} \phi_j(H_j,i)\left(\prod_{k\neq j} \phi_k(H_k,h_k)\right) \sum_{\substack{k\in N\\ h_k=i}} \frac{x_k}{s_i} \\ &= \sum_{i\in M} \phi_j(H_j,i)\frac{1}{s_i}\sum_{h\in Z_{i,j}} \left(\prod_{k\neq j} \phi_k(H_k,h_k)\right) \sum_{\substack{k\in N\\ h_k=i}} x_k. \end{aligned}$$

Nun ziehen wir noch die Last, die von dem Job des Spielers j verursacht wird, aus der Summe heraus. Sie wird bei jeder Historie aus $Z_{j,i}$ mit aufsummiert. Dies ergibt

$$\begin{aligned} C_j(\phi) &= \sum_{i\in M} \phi_j(H_j,i)\frac{1}{s_i}\sum_{h\in Z_{i,j}} \left(\prod_{k\neq j} \phi_k(H_k,h_k)\right) \sum_{\substack{k\in N\\ h_k=i}} x_k \\ &= \sum_{i\in M} \phi_j(H_j,i)\frac{1}{s_i}\left(x_j + \sum_{h\in Z_{i,j}} \left(\prod_{k\neq j} \phi_k(H_k,h_k)\right) \sum_{\substack{k\neq j\\ h_k=i}} x_k\right). \end{aligned}$$

Da die Wahl der Maschine von jedem Spieler unabhängig von den anderen Spielern ist, können wir die erwartete zusätzliche Last der Spieler $\neq j$ auf der Maschine i, wie folgt zusammenfassen.

$$\sum_{h\in Z_{j,i}} \left(\prod_{k\neq j} \phi_k(H_k,h_k)\right) \sum_{\substack{k\neq j\\ h_k=i}} x_k = \sum_{k\neq j} \phi_k(H_k,i)x_k.$$

Setzen wir dies ein und beachten, dass für jeden Spieler $k \in N$ gilt, dass $\phi_k(H_k, i) = \phi_k^{\mathrm{KP}}(i)$, so erhalten wir direkt das gewünschte Ergebnis.

$$\begin{aligned} C_j(\phi) &= \sum_{i \in M} \phi_j(H_j, i) \frac{1}{s_i} \left(x_j + \sum_{h \in Z_{j,i}} \left(\prod_{k \neq j} \phi_k(H_k, h_k) \right) \sum_{\substack{k \neq j \\ h_k = i}} x_k \right) \\ &= \sum_{i \in M} \phi_j(H_j, i) \frac{1}{s_i} \left(x_j + \sum_{k \neq j} \phi_k(H_k, i) x_k \right) \\ &= \sum_{i \in M} \phi_j^{\mathrm{KP}}(i) \frac{1}{s_i} \left(x_j + \sum_{k \neq j} \phi_k^{\mathrm{KP}}(i) x_k \right) \\ &= C_j^{\mathrm{KP}}(\phi^{\mathrm{KP}}). \end{aligned}$$

Entsprechend definieren wir ϕ^{KP} bei einem gegebenen Profil ϕ durch

$$\phi_j^{\mathrm{KP}}(i) := \phi_j(H_j, i).$$

Durch analoges Nachrechnen erhalten wir die gewünschte Aussage. ■

Insbesondere gilt die Aussage dieses Satzes für Profile, die ein Nash-Gleichgewicht oder ein perfektes Gleichgewicht darstellen. Da die Übertragung der Profile unter Erhalt der Antwortzeiten in beide Richtungen möglich ist, folgt folgendes Korollar.

Korollar 3.1. *Seien $\Gamma^{\mathrm{KP}} = \langle N, M, (x_j), (s_i) \rangle$ ein KP-Modell-Scheduling-Spiel und $\Gamma = \Gamma(\Gamma^{\mathrm{KP}})$ das dazugehörige Scheduling-Spiel.*

Dann gilt

1. *für alle Nash-Gleichgewichte ϕ^{KP} von Γ^{KP} existiert ein Nash-Gleichgewicht ϕ von Γ mit*

$$\forall j \in N : C_j^{\mathrm{KP}}(\phi^{\mathrm{KP}}) = C_j(\phi),$$

2. *für alle perfekten Gleichgewichte ϕ^{KP} von Γ^{KP} existiert ein perfektes Gleichgewicht ϕ von Γ mit*

$$\forall j \in N : C_j^{\mathrm{KP}}(\phi^{\mathrm{KP}}) = C_j(\phi),$$

3. *für alle Nash-Gleichgewichte ϕ von Γ existiert ein Nash-Gleichgewicht ϕ^{KP} von Γ mit*

$$\forall j \in N : C_j(\phi) = C_j^{\mathrm{KP}}(\phi^{\mathrm{KP}}) \quad \textit{und}$$

4. *für alle perfekten Gleichgewichte ϕ von Γ existiert ein perfektes Gleichgewicht ϕ^{KP} von Γ mit*

$$\forall j \in N : C_j(\phi) = C_j^{\mathrm{KP}}(\phi^{\mathrm{KP}}).$$

So übertragen sich die Ergebnisse für den Preis der Anarchie eines KP-Modell-Scheduling-Spiels Γ^{KP} direkt auf das dazugehörige Scheduling-Spiel $\Gamma(\Gamma^{\mathrm{KP}})$. Deshalb folgt direkt aus Satz 3.1 und Korollar 3.1 das Korollar 3.2.

Definieren wir hierzu den Preis der Anarchie $\mathrm{PA}(\Gamma, f)$ des Spiels Γ für die Zielfunktion f als das Verhältnis zwischen schlechtestem perfekten Gleichgewicht von Γ und einer optimalen Strategie für dieses Spiel. Das heißt,

$$\mathrm{PA}(\Gamma, f) := \frac{\sup\{f(a_0, (C_1(\phi), \ldots, C_n(\phi))) \mid \phi \text{ ist perfektes Gleichgewicht}\}}{\inf\{f(a_0, (C_1(\phi), \ldots, C_n(\phi))) \mid \phi \text{ ist beliebiges Profil}\}}.$$

Im Unterschied zu der für Scheduling-Spiele eingeführten Definition, betrachten wir hier keine Modelle sondern einzelne Spiele. Weiterhin vereinfacht sich die Definition, da nur eine einzige Eingabe von den Natur ausgewählt wird.

Korollar 3.2. *Sei Γ ein Scheduling-Spiel, das wie in Satz 3.1 definiert wurde. Sei Γ^{KP} das dazugehörige KP-Modell-Spiel. Dann gilt:*

- *Bei identischen Maschinen $(s_i) = 1$:*
  - $\mathrm{PA}(\Gamma, f_{[C_{\max}]}) \leq 2 - \frac{2}{m+1}$ *[GLM+03].*
- *Bei beliebigen Maschinen:*
  - $\mathrm{PA}(\Gamma, f_{[C_{\max}]}) \leq O\left(\log m/(\log\log\log m)\right)$ *[CV02].*
  - $\mathrm{PA}(\Gamma, f_{[C_{\max}]}) \leq \Gamma^{-1}(m)$*, wobei Γ^{-1} das Inverse der Gamma-Funktion ist [FGL+03a].*
  - $\mathrm{PA}(\Gamma, f_{[\sum C_j]}) \leq 4\frac{x_{\max}}{x_{\min}}$*, wobei $x_{\max}$ die Größe des größten und $x_{\min}$ die Größe des kleinsten Jobs sind [BGGM04].*

Auch das von Immorlica et al. [ILMS09] und Kretschmer [Kre06] beschriebene Prioritätsmodell, das sich nur durch das Maschinenmodell von dem KP-Modell unterscheidet, kann analog zu der Konstruktion aus Satz 3.1 nachgebildet werden. Bei diesem Prioritätsmodell werden die Jobs auf den Maschinen in fester Reihenfolge abgearbeitet und das Ergebnis direkt den Spielern mitgeteilt. Die Reihenfolge ist so gewählt, dass die Jobs in nicht-absteigender Reihenfolge bezüglich ihrer Jobgrößen bearbeitet werden. Die Aussage von Satz 3.1 überträgt sich entsprechend. Das heißt, Nash-Gleichgewichte bleiben erhalten und damit übertragen sich die Ergebnisse des Preises der Anarchie.

Kapitel 4

Nachbildung von Online-Algorithmen

Wir betrachten zunächst Modelle, bei denen die Spieler die gleichen Informationen haben, wie ein Online-Algorithmus hätte. Das heißt, den Spielern sind die Aktionen der vorangegangenen Spieler, die Größe der Jobs dieser Spieler und die Größe des eigenen Jobs bekannt. Damit sind die Informationspartitionen, wie folgt, definiert.

$$\mathcal{I}_j := \{\{h \in H \mid \mathfrak{A}(h) = \mathfrak{a}, \mathfrak{X}(h) = \mathfrak{x}\} \mid \mathfrak{a} \in M^{|h|}, \mathfrak{x} \in \mathbb{N}^{|h|+1}\}, \tag{4.1}$$

wobei

$$\mathfrak{A}(h) = \big(h_1, h_2, \ldots, h_{|h|}\big)$$

das Tupel der schon gespielten Aktionen ist und

$$\mathfrak{X}(h) = \big(x_1(h), x_2(h), \ldots, x_{|h|+1}(h)\big)$$

das Tupel der Jobgrößen der Spieler, die den ersten bis $(|h| + 1)$-ten Zug durchführen, darstellt.

Haben wir nun einen c-kompetitiven Online-Algorithmus für eine Zielfunktion f gegeben, so können wir aus diesem ein Maschinenmodell entwickeln, welches garantiert, dass das Koordinationsverhältnis identisch zu c ist. Um dies zu erreichen, wird jeder Spieler bestraft, der sich nicht an die Strategie des Online-Algorithmus hält. Ein Spieler kann nur so bestraft werden, indem seine Antwortzeit künstlich erhöht wird, da die Spieler versuchen ihre Antwortzeit zu minimieren. Die Jobs von den Spielern, die nach der Strategie des Online-Algorithmus spielen, werden zu genau dem Zeitpunkt bearbeitet, an dem der Algorithmus auch den Job abgearbeitet hätte. Alle anderen

Jobs werden herausgezögert, so dass die Spieler das Ergebnis des Jobs erst später erhalten, als es der Fall gewesen wäre, wenn sie sich an die Strategie des Online-Algorithmus gehalten hätten. Dies impliziert, dass nur dasjenige Profil in einem Gleichgewicht ist, bei dem die Spieler genau wie der Online-Algorithmus arbeiten. Daraus folgt dann direkt das Koordinationsverhältnis.

Eine solche Benachteiligung von Jobs, die sich nicht an die Strategie des Online-Algorithmus halten, ist nicht immer sinnvoll. Anstelle die Spieler zu benachteiligen bzw. zu belohnen, kann man auch direkt die Verteilung der Jobs zentral durchführen. Deswegen ist dieses Kapitel 4 inklusive dem Lemma 4.1 nur der Vollständigkeit halber aufgeführt.

Wir beschreiben zunächst das Modell μ. Für alle $n \in \mathbb{N}$ sind die Informationspartitionen $\mathcal{I}_j$ für das Scheduling-Spiel $\Gamma = \langle N, M, (s_i), H, Z, \rho_{\mathfrak{n}}, (\mathcal{I}_j), (C_j)\rangle = \mu(n)$ wie in (4.1) definiert. Das heißt, Γ ist ein online Scheduling-Spiel. Seien zudem eine Zielfunktion f und ein deterministischer c-kompetitiver Online-Algorithmus $\mathcal{A}$ für die Zielfunktion f gegeben.

Bezeichne $S_{\mathcal{A}}(h_0)$ den von $\mathcal{A}$ generierten Schedule bei Eingabe $h_0 \in \mathcal{E}(n)$ und $S_{\mathcal{A}}(h_0)(j)$ die Maschine, auf dem der Job von Spieler j bearbeitet wird.

Seien nun die Maschinen nach ihrer Geschwindigkeit sortiert. Das heißt, $s_1 \leq s_2 \leq \cdots \leq s_m$. Damit ist die erste Maschine die langsamste Maschine. Wir werden das Resultat der Jobs den Spielern, die sich in einer terminalen Historie $h \in Z$ nicht an die Strategie von Algorithmus $\mathcal{A}$ halten, um genau die Zeit

$$\varepsilon + \sum_{k=1}^{|h_0|} \frac{x_k(h_0)}{s_1}$$

später mitteilen, als der letzte Job durch Algorithmus $\mathcal{A}$ seine Antwort erhält. Dadurch bleibt auch der langsamsten Maschine genügend Zeit, alle ihr zugeteilten Jobs vor diesem Zeitpunkt fertig zu stellen.

Das Maschinenmodell ist nun folgendermaßen definiert. Wir bearbeiten die Jobs auf den Maschinen so, wie es in dem von $\mathcal{A}$ generierten Schedule definiert wird, falls die Spieler sich an die Strategie von $\mathcal{A}$ halten. Ansonsten verzögern wir die Antwort an die Spieler bis zum Zeitpunkt X_ε, der wie folgt definiert ist:

$$X_\varepsilon := \varepsilon + \sum_{k=1}^{|h_0|} \frac{x_k(h_0)}{s_1} + \max_{l=1,2,\ldots,|h_0|} C_{p_l(h_0)}(S_{\mathcal{A}}(h_0))$$

Das heißt, sei $\varepsilon > 0$ beliebig, aber fest gewählt, dann sind die Antwortzeit-

funktionen durch

$$C_j(h) := \begin{cases} 0, & \text{falls } \nexists l : 1 \leq l \leq |h_0| \text{ mit } p_l(h_0) = j \\ C_j(S_{\mathcal{A}}(h_0)), & \text{falls } \exists l : p_l(h_0) = j \text{ und } h_l = S_{\mathcal{A}}(h_0)(j) \\ X_\varepsilon, & \text{sonst} \end{cases} \tag{4.2}$$

definiert.

Wenn Spieler sich nicht an Algorithmus $\mathcal{A}$ halten, kann es vorkommen, dass mehrere Jobs auf der gleichen Maschine die gleiche Antwortzeit haben. Dies kann dadurch erreicht werden, dass die Bearbeitung eines Jobs von der Übermittlung des Ergebnisses an den Spieler getrennt wird. Für den Spieler ist der Zeitpunkt, an dem er das Ergebnis seines Jobs erhält, ausschlaggebend und nicht der Zeitpunkt an dem die Maschine den Job fertig gestellt hat. Damit ist die Antwortzeit nicht die Fertigstellungszeit des Jobs, sondern der Zeitpunkt an dem das Ergebnis übermittelt wird. X_ε ist gerade so gewählt, dass jeder Maschine genügend Zeit bleibt, um alle Jobs rechtzeitig fertig zu stellen.

Wir halten somit die Antworten der Jobs, die sich nicht an den Algorithmus $\mathcal{A}$ halten, zurück, bis die langsamste Maschine alle Jobs bearbeitet haben könnte. Nach Definition brauchen die Jobs damit länger, als wenn sie sich an $\mathcal{A}$ gehalten hätten.

Das durch (4.2) definierte Maschinenmodell erzeugt Schedules, wie in Abbildung 4.1 exemplarisch aufgeführt, falls sich die Spieler nicht an den Algorithmus $\mathcal{A}$ halten. Wie man sieht, werden viele unnötigen Leerlaufzeiten eingeführt.

Ferner bezeichnen wir mit $H' \subseteq H$ die größte Teilmenge von H, so dass für alle $h \in H'$ ein vollständig gemischtes Profil ϕ' existiert, bei dem die Wahrscheinlichkeit, dass das Spiel in h endet, größer Null ist. Das heißt, $\Pr[h|\phi'] > 0$. Damit beinhaltet H' alle Historien, die in einem Spiel erreicht werden können und nicht schon durch die Natur bzw. die Wahrscheinlichkeitsverteilung $\rho_{\mathfrak{n}}$ ausgeschlossen werden.

Lemma 4.1. *Seien μ ein Modell, wie es oben definiert wurde, und $H' \subseteq H$ die Menge aus allen Historien, welche bei einem vollständig gemischten Profil erreicht werden können.*

Dann gilt für alle $n \in \mathbb{N}$, dass bei dem online Scheduling-Spiel $\mu(n)$ nur die Profile ϕ perfekte Gleichgewichte sind, bei denen die Spieler für alle Historien $h \in H'$ die gleichen Aktionen ausführen, die auch der Algorithmus $\mathcal{A}$ durchführen würde.

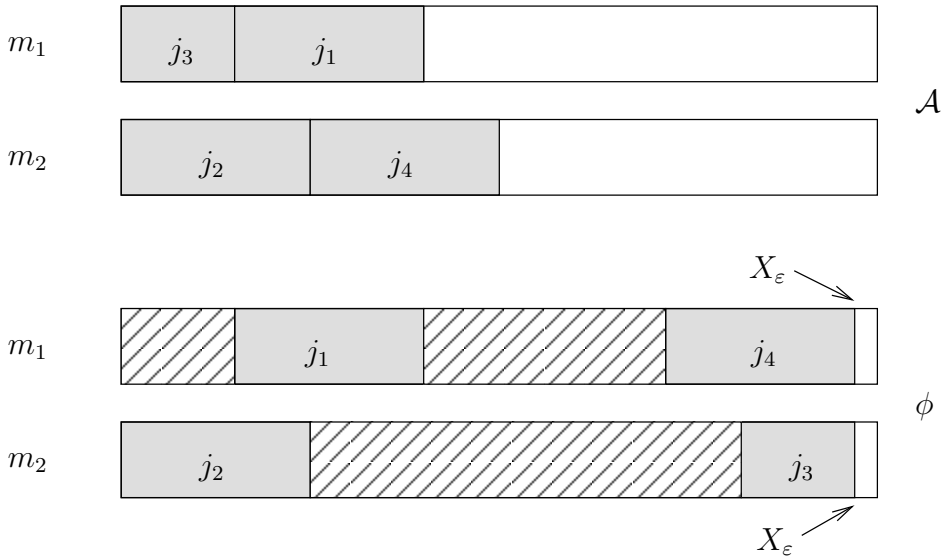

Abbildung 4.1: Beispiel für das Maschinenmodell aus Lemma 4.1

Die Einschränkung auf die Historien H' ist dadurch bedingt, dass wir auch Online-Algorithmen betrachten möchten, die nur eine Teilmenge $\overline{\mathcal{E}} \subseteq \mathcal{E}(n)$ von Eingaben verarbeiten. In unserem Modell wird dies dadurch dargestellt, dass die unerlaubten Eingaben mit der Wahrscheinlichkeit Null von der Natur ausgewählt werden.

Beweis. Wir wählen zuerst ein $n \in \mathbb{N}$ beliebig, aber fest. Sei das dazugehörige Spiel $\Gamma = \langle N, M, (s_i), H, Z, \rho_{\mathfrak{n}}, (\mathcal{I}_j), (C_j) \rangle = \mu(n)$. Wir zeigen zunächst, dass das Profil ϕ, bei dem alle Spieler der Strategie von Algorithmus $\mathcal{A}$ folgen, ein perfektes Gleichgewicht ist. Das bedeutet, in Profil ϕ wählt Spieler $j \in N$ die Maschine i in Informationsmenge $I \in \mathcal{I}_j$, falls $S_{\mathcal{A}}(I_0)(l) = i$, wobei l der Index von Spieler j in Eingabe I_0 ist. Betrachten wir hierzu eine Folge ϕ^k von vollständig gemischten Profilen, mit $\lim_{k\to\infty} \phi^k = \phi$. Wir wählen des Weiteren eine beliebige terminale Historie $h \in Z \cap H'$ und einen Spieler $j \in N$, so dass er in Historie h einen Zug durchgeführt hat. Das heißt, dass Spieler j in der Eingabe h_0 vorkommt. Ohne Beschränkung der Allgemeinheit führe Spieler j den j-ten Zug durch. Wir können dies immer erreichen, indem wir die Spieler umnummerieren.

Betrachten wir nun eine beliebige Testfolge $(\Gamma, \eta^1), (\Gamma, \eta^2), \ldots$. Seien $I \in \mathcal{I}_j$ mit $I \cap H' \neq \emptyset$, $i \in M$ und $k \in \mathbb{N}$ beliebig gewählt. Wir definieren die Strategie $\phi_j^k(I, \cdot)$ nun so, dass Spieler j genau die Maschine i mit größtmöglicher Wahrscheinlichkeit wählt, auf die auch der Algorithmus $\mathcal{A}$ den Job von Spie-

ler j platziert hätte. Diese Maschine wird von Spieler j in I gerade mit Wahrscheinlichkeit Eins in Profil ϕ gewählt. Das heißt,

$$\phi_j^k(I,i) := \begin{cases} 1 - \sum_{i' \neq i} \eta_j^k(I,i'), & \text{falls } \phi_j(I,i) = 1 \\ \eta_j^k(I,i), & \text{sonst.} \end{cases}$$

Dies definiert uns eine gültige Strategie für Spieler j in Spielzustand I, da $\mathcal{A}$ ein deterministischer Algorithmus ist und es für I eine eindeutige Maschine gibt, die $\mathcal{A}$ auswählt. Diese Maschine ist alleine durch die Eingabe I_0 festgelegt.

Aus der Definition von $\phi_j^k(\cdot,\cdot)$ folgt, dass $\lim_{k\to\infty} \phi^k = \phi$. Wir müssen nun zeigen, dass ϕ_j^k eine bestmögliche Antwort auf ϕ_{-j}^k ist. Dazu betrachten wir eine beliebige Strategie ϕ_j' von Spieler j für das Spiel (Γ, η^k). Sei $\psi = (\phi_{-j}^k, \phi_j')$. Betrachten wir nun die erwartete Antwortzeit $\mathrm{E}_\psi[C_j]$ für Spieler j.

$$\mathrm{E}_\psi[C_j] = \sum_{\substack{h \in Z \cap H' \\ h_j \neq S_\mathcal{A}(h_0)(j)}} \Pr_\psi[h] X_\varepsilon + \sum_{\substack{h \in Z \cap H' \\ h_j = S_\mathcal{A}(h_0)(j)}} \Pr_\psi[h] C_j(S_\mathcal{A}(h_0))$$

mit

$$X_\varepsilon := \varepsilon + \sum_{k=1}^{|h|} \frac{x_k(h)}{s_1} + \max_{1 \leq l \leq |h_0|} C_{p_l(h_0)}(S_\mathcal{A}(h_0))$$

Nach der Definition von X_ε gilt $X_\varepsilon > C_j(S_\mathcal{A}(h_0))$ und damit wird $\mathrm{E}_\psi[C_j]$ minimiert, wenn man in Informationsmenge $I \in \mathcal{I}_j$ die Wahrscheinlichkeit die Maschine $S_\mathcal{A}(h_0)(j)$ zu wählen maximiert. Das ist gerade die Maschine $i \in M$ mit $\phi_j(I,i) = 1$ und ϕ_j^k ist entsprechend gewählt. Das heißt, ϕ_j^k minimiert unter allen Strategien ϕ_j' für Spiel (Γ, η^k) die Antwortzeit von Spieler j. Damit ist ϕ ein Grenzwertgleichgewicht der Testfolge $(\Gamma, \eta^1), (\Gamma, \eta^2), \ldots$ und somit auch ein perfektes Gleichgewicht.

Es bleibt nun zu zeigen, dass jedes perfekte Gleichgewicht auf H' sich so verhält, wie der Algorithmus $\mathcal{A}$. Da ein perfektes Gleichgewicht auch immer ein Nash-Gleichgewicht ist, reicht es zu zeigen, dass jedes andere Profil, das sich nicht an Algorithmus $\mathcal{A}$ hält, kein Nash-Gleichgewicht ist. Seien nun ϕ ein perfektes Gleichgewicht, bei dem die Spieler sich an die Strategie von Algorithmus $\mathcal{A}$ halten, und ψ ein Profil mit $\psi|_{H'} \not\equiv \phi|_{H'}$. Das heißt, dass sich mindestens ein Spieler $j \in N$ in ψ bei mindestens einer Informationsmenge $I \in \mathcal{I}_j$ mit $I \cap H' \neq \emptyset$ nicht so verhält wie in ϕ. Da mindestens ein Spieler in ψ von der Strategie, die durch $\mathcal{A}$ vorgegeben ist, abweicht, wird eine terminale Historie, bei der dieser Spieler eine andere Aktion mit einer Wahrscheinlichkeit > 0 wählt, erreicht. Durch die Definition von C_j ist sichergestellt, dass

der Spieler eine höhere Antwortzeit haben wird, als wenn er sich an $\mathcal{A}$ gehalten hätte. Damit könnte er seinen Nutzen erhöhen, indem er seine Strategie zu ϕ_j verändert. Daraus folgt, dass ψ kein Nash-Gleichgewicht ist und damit auch kein perfektes Gleichgewicht. ■

Da im Beweis gezeigt wurde, dass jedes Profil, bei dem die Spieler sich an den Algorithmus halten, ein perfektes Gleichgewicht ist und alle anderen Profile kein Nash-Gleichgewicht sind, erhalten wir zudem folgenden Satz:

Satz 4.1. *Bei dem Modell μ aus Lemma 4.1 für einen c-kompetitiven Online-Algorithmus bezüglich einer Zielfunktion $f \in \mathfrak{F}$, die die Monotonieeigenschaft erfüllt, gelten folgende Eigenschaften:*

1. *Für jedes Spiel $\mu(n)$ gilt, dass die Menge der perfekten Gleichgewichte identisch zu der Menge der Nash-Gleichgewichte ist.*
2. $\mathrm{CR}(\mu, f) \leq c$.
3. $\mathrm{PA}(\mu, f) = 1$.

Die letzte Eigenschaft ist eine Folgerung aus der Konstruktion. Der durch (C_j) von $\mu(n)$ festgelegte Schedule sorgt dafür, dass bei keiner Strategie die Antwortzeit eines Spielers kleiner als die Antwortzeit ist, die von dem Algorithmus für diesen Spieler generiert würde. Das bedeutet, dass für jede Eingabe $a_0 \in \mathcal{E}(n)$, jeden Spieler $j \in N$, alle perfekten Gleichgewichte ϕ und beliebige Profile ψ

$$\sum_{\substack{h \in Z \\ h_0 = a_0}} \Pr_\phi[h|a_0] C_j(h) \leq \sum_{\substack{h \in Z \\ h_0 = a_0}} \Pr_\psi[h|a_0] C_j(h)$$

gilt. Mit der Monotonieeigenschaft der Zielfunktionen folgt dann die Aussage.

Das bedeutet, bezüglich des durch (C_j) festgelegten Maschinenmodells, ist ein perfektes Gleichgewicht das bestmöglich zu erreichende Ergebnis. Damit kann $\mathrm{CR}(\mu, f)$ deutlich größer als $\mathrm{PA}(\mu, f)$ sein. Dies zeigt, wie schlecht dieses Modell ist. Es ist eine künstliche Konstruktion, die, wie auch das Beispiel aus Abbildung 4.1 auf Seite 46 zeigt, Leerlaufzeiten auf den Maschinen erzeugt, falls ein Profil kein perfektes Gleichgewicht ist.

Eine große Differenz zwischen $\mathrm{CR}(\mu, f)$ und $\mathrm{PA}(\mu, f)$ zeigt im Allgemeinen, dass das durch μ gewählte Maschinenmodell die Spieler dazu zwingt, besondere Aktionen durchzuführen oder keine gute Strategie bezüglich der Zielfunktion f erlaubt. Wir können somit das Verhältnis $\mathrm{PA}(\mu, f)$ zu $\mathrm{CR}(\mu, f)$ verwenden, um die Modelle bezüglich einer Zielfunktion zu bewerten.

Ein weiterer Nachteil von dem in Gleichung (4.2) definierten Maschinenmodell ist, dass die Maschinen alle Jobs, auch die, die auf andere Maschinen verteilt wurden, kennen müssen. Es ist durchaus vorstellbar, dass man dies nicht garantieren kann. Insbesondere ist dies der Fall, wenn die Maschinen nicht von einem einzelnen Anbieter sind, sondern jede Maschine einen Anbieter darstellt.

Kapitel 5

List Scheduling

Während wir im vorigen Kapitel beliebige Online-Algorithmen betrachtet haben und ein Verfahren vorgestellt haben, mit dem das kompetitive Verhältnis auf das Koordinationsverhältnis übertragen werden kann, widmen wir uns nun einem speziellen Online-Algorithmus. Dies erlaubt es uns, ein Maschinenmodell zu entwerfen, bei dem die Probleme aus dem vorigen Kapitel nicht auftreten, da wir die speziellen Eigenschaften dieses Algorithmus ausnutzen können.

Der List Scheduling Algorithmus ist ein Greedy-Algorithmus, der die Jobs der Reihe nach auf die Maschinen verteilt. Dabei werden die Jobs so den Maschinen zugewiesen, dass die nachfolgenden Jobs die Antwortzeiten nicht mehr beeinflussen können. Dies werden wir ausnutzen, um ein einfaches und auch natürliches Maschinenmodell zu definieren, welches uns garantiert, dass der durch ein perfektes Gleichgewicht definierte Schedule auch durch den List Scheduling Algorithmus entstanden sein kann.

Betrachten wir hierzu erst einmal die Situation der klassischen Online-Algorithmen bei der die Einlastzeiten alle Null sind. Es ist bekannt, dass der List Scheduling Algorithmus ein $2 - \frac{1}{m}$-kompetitiver Online-Algorithmus im Fall von identischen Maschinen bezüglich der Zielfunktion $f_{[C_{\max}]}$ ist [Gra66]. Für beliebige Maschinen ist List Scheduling ein $\Theta(\log m)$-kompetitiver Online-Algorithmus [AAF+97]. Für den Beweis der unteren Schranke wird eine Eingabe konstruiert, bei der die Geschwindigkeiten der Maschinen von der Anzahl der Maschinen abhängen. Mit der von Graham [Gra66] präsentierten Analyse lässt sich für beliebige Maschinen die Schranke $1 + \frac{m-1}{m} \frac{s_{\max}}{s_{\min}}$ ableiten. Die Arbeit von Cho und Sahni [CS80] führt weitere Schranken für Spezialfälle von List Scheduling auf. Weiterhin ist eine Folgerung aus der Arbeit von Faigle et al. [FKT89], dass für zwei und drei identische Maschinen, List

Scheduling ein optimaler Online-Algorithmus für die Zielfunktion $f_{[C_{\max}]}$ ist.

Man kann sich den List Scheduling Algorithmus auch so vorstellen, dass die Jobs der Reihe nach, in der sie auftreten, in eine Schlange eingefügt werden. Sobald eine Maschine frei ist, wird der nächste Job aus der Schlange entfernt und auf dieser Maschine bearbeitet. Sind mehrere Maschinen frei, wird eine Maschine mit maximaler Geschwindigkeit unter allen freien Maschinen ausgewählt. Das bedeutet, wir können uns den List Scheduling Algorithmus als FIFO-Algorithmus vorstellen. Dieser lässt sich dann leicht auf Probleme mit Einlastzeiten ungleich Null erweitern. Für diese Erweiterung sind Ergebnisse bezüglich der Zielfunktion $f_{[F_{\max}]}$ bekannt. Der FIFO-Algorithmus ist ein optimaler Algorithmus im Fall von einer Maschine und ist $(3-2/m)$-kompetitiv für m identische Maschinen bezüglich dieser Zielfunktion [BCM98]. Wir werden im Folgenden auch die Erweiterung um Einlastzeiten weiterhin als List Scheduling Algorithmus bezeichnen.

Da der kompetitive Faktor des List Scheduling Algorithmus zumindest im Fall von identischen Maschinen durch die Konstante Zwei bzw. Drei beschränkt ist, werden wir nun versuchen, ein Modell zu definieren, bei dem die Spieler in allen perfekten Gleichgewichten sich an den List Scheduling Algorithmus halten, ohne Leerlaufzeiten auf den Maschinen zu generieren. Abbildung 5.1 auf der nächsten Seite definiert den List Scheduling Algorithmus formal mit der Notation von online Scheduling-Spielen.

Diesen Greedy-Algorithmus kann man direkt auf ein Maschinenmodell übertragen, indem man die Jobs in der Reihenfolge ihrer Verteilung unter Berücksichtigung der Einlastzeiten auf den Maschinen bearbeitet. Das heißt,

$$C_j(h) := \begin{cases} \widetilde{C}(l, h_l, h), & \text{falls es ein } l : 1 \leq l \leq |h| \text{ mit } p_l = j \text{ gibt} \\ 0, & \text{sonst,} \end{cases} \tag{5.1}$$

wobei $\widetilde{C}(l, i, h)$ der Fertigstellungszeitpunkt von dem letzten Job auf der Maschine i ist, dessen Index in der Eingabe h_0 kleiner oder gleich l ist. Falls ein solcher Job nicht existiert, ist $\widetilde{C}(l, i, h) = 0$. Formal ist $\widetilde{C}(\cdot, \cdot, \cdot)$ folgendermaßen definiert:

$$\widetilde{C}(l, i, h) := \begin{cases} 0, & \text{falls } l = 0 \\ \max\{\widetilde{C}(l-1, i, h), r_l(h)\} + x_l(h)/s_i, & \text{falls } l > 0 \text{ und } h_l = i \\ \widetilde{C}(l-1, i, h), & \text{sonst.} \end{cases}$$

Dieses Maschinenmodell erzeugt die gleichen Antwortzeiten für eine terminale Historie $h \in Z$ wie der List Scheduling Algorithmus in einer Ausführung

Eingabe: Menge von Maschinen $M = \{1, 2, \ldots, m\}$, Geschwindigkeiten der Maschinen $(s_i) = (s_1, s_2, \ldots, s_m)$ und eine Eingabe $a_0 = ((p_1, r_1, x_1), (p_2, r_2, x_2), \ldots, (p_k, r_k, x_k)) \in \mathcal{E}(n)$

Ausgabe: Schedule S

1. $\forall i \in M : L_i^0 := 0$ (L_i^j ist der Zeitpunkt, an dem Maschine i wieder frei wird, wenn schon die Jobs $1, 2, \ldots, j$ verteilt wurden)
2. **for** $j := 1$ **to** k **do**
   (a) $\forall i \in M : X_i^j := \max\{L_i^{j-1}, r_j\} + x_j/s_i$ (X_i^j ist der Zeitpunkt, an dem Maschine i den Job j fertig stellen kann)
   (b) Wähle nun eine Maschine $i \in M$ so, dass der Job schnellstmöglich fertig gestellt wird. Das heißt,
   $$X_i^j = \min_{i' \in M} X_{i'}^j.$$
   (c) $m_j := i$ (Job j wird auf Maschine i gesetzt)
   (d) $C_j := X_i^j$ (die Antwortzeit von Job j)
   (e) $\forall i' \in M : X_{i'}^j := \begin{cases} X_i^j, & \text{falls } i = i' \\ X_{i'}^{j-1}, & \text{sonst} \end{cases}$
3. **return** $((m_1, C_1, 1), (m_2, C_2, 2), \ldots, (m_k, C_k, k))$

Abbildung 5.1: List Scheduling Algorithmus ($\mathcal{LS}$)

$S = \mathcal{LS}(M, (s_i), h_0)$, falls für alle $1 \leq l \leq |h_0|$ gilt, dass die Jobs auf die gleichen Maschinen platziert wurden ($m_l = h_l$).

Aus der Definition folgt, dass $C_j(h)$ sich folgendermaßen vereinfacht, falls alle Einlastzeiten Null sind:

$$C_j(h) = \begin{cases} \sum_{\substack{k \leq l \\ h_k = h_l}} \frac{x_k}{s_{h_l}}, & \text{falls es ein } l : 1 \leq l \leq |h| \text{ mit } p_l = j \text{ gibt} \\ 0, & \text{sonst.} \end{cases} \tag{5.2}$$

Da bei diesem Maschinenmodell die Jobs von nachfolgenden Jobs nicht beeinflusst werden, ist zu erwarten, dass der List Scheduling Algorithmus ($\mathcal{LS}$) perfekte Gleichgewichte erzeugt.

Lemma 5.1. *Sei μ ein Modell, so dass die Informationspartitionen durch* (4.1) *gegeben sind und die Antwortzeiten durch* (5.1)*. Dann gilt für alle $n \in \mathbb{N}$, dass ein perfektes Gleichgewicht ϕ des Spieles $\mu(n)$ mit $\mu(n) = \langle N, M, (s_i), H, Z, \rho_{\mathfrak{n}}, (\mathcal{I}_j), (C_j)\rangle$ durch den List Scheduling Algorithmus generiert wird. Das heißt, für jedes perfekte Gleichgewicht ϕ und jede terminale Historie $h \in Z$ mit $\rho_{\mathfrak{n}}(h_0) > 0$ gilt:*

$$\Pr_\phi[h] > 0 \Rightarrow \exists S = \mathcal{LS}(M, (s_i), h_0) : \forall 1 \leq k \leq |h| : m_k = h_k, \tag{5.3}$$

wobei m_k die Maschine ist, die der Algorithmus $\mathcal{LS}$ für den k-ten Job in S ausgewählt hat.

Der List Scheduling Algorithmus hat Freiheit bei der Wahl der Maschine für einen Job, da es mehrere Maschinen geben kann, die bei der Verteilung eines Jobs die Antwortzeit dieses Jobs minimieren. Aus der Gleichung (5.3) und der Definition des Maschinenmodells erhalten wir direkt für alle perfekten Gleichgewichte ϕ

$$\Pr_\phi[h] > 0 \Rightarrow \exists S = \mathcal{LS}(M, (s_i), h_0) : \forall j \in N : C_j(h) = C_j(S).$$

Dies ist so zu verstehen, dass es einen gültigen Ablauf von $\mathcal{LS}$ gibt, der einen Schedule S erzeugt, dem die Historie h bei dem Maschinenmodell entspricht.

Intuitiv sollte dieses Lemma korrekt sein, da der Fertigstellungszeitpunkt von Spieler $j \in N$ nicht von der Strategie der Spieler, die nach j am Zug sind, abhängt und Spieler j die Wahl der Maschine und Größe der Jobs der vorangegangenen Spieler kennt.

Beweis. Zum Beweis betrachten wir eine terminale Historie $h \in Z$ und ein perfektes Gleichgewicht ϕ mit $\Pr_\phi[h] > 0$. Das heißt, das Spiel endet mit positiver Wahrscheinlichkeit in h. Es muss gezeigt werden, dass dieser Schedule durch $\mathcal{LS}$ entstanden sein kann.

Nehmen wir dazu an, dass der von h und $C_j(h)$ implizit festgelegte Schedule nicht durch $\mathcal{LS}$ entstehen kann. Das bedeutet, es gibt einen Zug $k : 1 \leq k \leq |h|$, so dass $\mathcal{LS}$ bei Eingabe $(M, (s_i), h_0)$ für den Job von Spieler $j = p_k(h)$ niemals die Maschine h_k ausgewählt hätte. Das bedeutet aber auch, dass es eine Maschine $i \in M$ gibt, so dass

$$X_i^k < X_{h_k}^k,$$

wobei X_i^j die entsprechenden Werte aus dem Lauf von Algorithmus $\mathcal{LS}$ sind. Nach der Definition des Maschinenmodells (C_j) und der Definition von $\mathcal{LS}$ beeinflussen die nachfolgenden Jobs die Antwortzeit von Spieler j nicht mehr.

Sei nun k so gewählt, dass Spieler $j := p_k(h)$ der erste Spieler ist, der keine Aktion in h gespielt hat, die für $\mathcal{LS}$ gültig wäre. Das heißt, es existiert kein Spieler $j' = p_{k'}(h)$ mit $1 \leq k' < k$, der sich nicht an $\mathcal{LS}$ hält.

Wir werden nun zeigen, dass Spieler j seinen Nutzen verbessern bzw. seine Antwortzeit verringern kann, indem er anstelle von h_k eine Maschine wie Algorithmus $\mathcal{LS}$ wählt. Dazu müssen wir alle Historien betrachten, die zur Informationspartition $I(h^{k-1})$ gehören. $I(h^{k-1})$ ist die Informationsmenge von Spieler j, so dass h^{k-1} in $I(h^{k-1})$ enthalten ist. Das heißt, sei $I(h^{k-1}) \in \mathcal{I}_j$ mit $h^{k-1} \in I(h^{k-1})$. Da aber nach der Definition des Maschinenmodells

$$C_j(h) = X^k_{h_k} > \min_{i' \in M} X^k_{i'}$$

gilt und alle Spieler $j' = p_{k'}(h)$ mit $k' > k$ die Antwortzeit von Spieler j nicht beeinflussen, folgt, dass Spieler j seine erwartete Antwortzeit verkürzen kann, wenn er in $I(h^{k-1})$ niemals die Maschine h_k wählt. Damit gilt für jedes Nash-Gleichgewicht ψ, dass $\Pr_\psi[h] = 0$. Damit ist ϕ kein Nash-Gleichgewicht und auch kein perfektes Gleichgewicht. Das ist ein Widerspruch, womit das Lemma bewiesen ist. ■

Durch Lemma 5.1 haben wir noch nicht gezeigt, dass es ein perfektes Gleichgewicht gibt. Falls es allerdings ein perfektes Gleichgewicht gibt, so folgt aus diesem Lemma, dass für jede Zielfunktion $f \in \mathfrak{F}$

$$\mathrm{PA}(\mu, f) \leq \mathrm{CR}(\mu, f) \leq c_f$$

gilt, falls $\mathcal{LS}$ ein c_f-kompetitiver Online-Algorithmus bezüglich der Zielfunktion f ist.

Wir sind nun an der Existenz von perfekten Gleichgewichten interessiert. Dazu zeigen wir die umgekehrte Richtung von Lemma 5.1. Das bedeutet, dass für jede Eingabe und jeden Schedule, der von Algorithmus $\mathcal{LS}$ für diese Eingabe konstruiert wird, ein perfektes Gleichgewicht existiert, bei dem die Spieler sich an die Verteilung auf die von $\mathcal{LS}$ vorgegebenen Maschinen halten.

Im Folgenden werden wir ein Profil ϕ durch den Algorithmus $\mathcal{LS}$ bestimmen lassen. Wir werden dann zeigen, dass dieses Profil ein perfektes Gleichgewicht ist. Sei dazu $X^j_i(I)$ die Last der Maschine i, falls Spieler j in Informationsmenge $I \in \mathcal{I}_j$ diese Maschine auswählt. Da Spieler, die nach dem Spieler j einen Zug durchführen, die Antwortzeit von Spieler j nicht mehr erhöhen, gilt für alle terminalen Historien $h \in Z$, die einen Präfix in I haben und bei denen Spieler j Maschine i auswählt, dass

$$X^j_i(I) = C_j(h) = \widetilde{C}(l, i, h),$$

wobei Spieler j den l-ten Index in Eingabe h_0 hat ($p_l(h_0) = j$). Zur Erinnerung, $\widetilde{C}(\cdot,\cdot,\cdot)$ ist nicht über die Spielernummer, sondern den Index des Zuges definiert.

In dem Profil ϕ wählt jeder Spieler $j \in N$ nun in einer Informationsmenge $I \in \mathcal{I}_j$ eine Maschine $i \in M$ aus, so dass die Last $X_i^j(I)$ minimiert wird. Das bedeutet, Maschine i wird so gewählt, dass

$$X_i^j(I) = \min_{i' \in M} X_{i'}^j(I).$$

Es kann mehrere solche Maschinen geben. Wir wählen eine beliebige Maschine aus und bezeichnen diese mit $i_j(I)$. Formal ist nun das Profil ϕ durch

$$\phi_j(I,i) := \begin{cases} 1, & \text{falls } i = i_j(I) \\ 0, & \text{sonst} \end{cases}$$

definiert. Dies entspricht exakt der Regel mit der der List Scheduling Algorithmus eine Maschine für einen Job auswählt. Das heißt, dass die Spieler sich an den List Scheduling Algorithmus halten. Durch das Profil werden aber auch Strategien für Informationsmengen festgelegt, die durch den Algorithmus nicht erreicht wurden. Das heißt, dass in einer solchen Informationsmenge ein vorangegangener Spieler nicht eine Maschine ausgewählt hat, die $X_i^j(I)$ minimiert. Eine solche Informationsmenge wird bei dem Profil ϕ nicht erreicht, wir benötigen aber dennoch eine Definition für die Aktion des Spielers. Insbesondere können wir keine beliebige wählen, da in einem Spiel (Γ, η) eine solche Informationsmenge erreicht wird.

Da es für einen Spieler $j \in N$ und eine Informationsmenge $I \in \mathcal{I}_j$ mehrere Maschinen geben kann, die $X_i^j(I)$ minimieren, gibt es mehrere Profile, die so konstruiert werden können. Sei $\Phi^P(\Gamma)$ die Menge dieser Profile für das Spiel Γ. Diese Menge entspricht der Menge der Profile, bei denen sich die Spieler an die Strategie der List Scheduling Algorithmus in jeder Informationsmenge halten. Wir werden in dem folgenden Lemma zeigen, dass jedes dieser Profile ein perfektes Gleichgewicht ist.

Lemma 5.2. *Sei μ ein Modell, wie in Lemma 5.1. Dann gilt für alle $n \in \mathbb{N}$ und dazugehörigen Spiele $\Gamma = \mu(n)$:*

1. *Es gibt mindestens ein Profil in $\Phi^P(\Gamma)$.*

2. *Jedes Profil $\phi \in \Phi^P(\Gamma)$ ist ein perfektes Gleichgewicht.*

Eine wichtige Folgerung aus diesem Lemma ist, dass es perfekte Gleichgewichte gibt. Zudem wissen wir, dass wir perfekte Gleichgewichte mit dem Algorithmus $\mathcal{LS}$ generieren können.

Aus den Lemmata 5.1 und 5.2 folgt nicht, dass jedes Nash-Gleichgewicht auch ein perfektes Gleichgewicht ist. Dies liegt daran, dass in einem Nash-Gleichgewicht ϕ ein Spieler $j \in N$ eine beliebige Strategie für eine Informationsmenge $I \in \mathcal{I}_j$ wählen kann, sofern I nicht erreicht wird ($\Pr_\phi[I] = 0$). Allerdings betrachten wir bei der Definition von perfekten Gleichgewichten Testfolgen. In den durch die Testfolgen definierten Spielen müssen die Spieler jede Aktion mit positiver Wahrscheinlichkeit ausführen, so dass jede Historie $I \in \mathcal{I}_j$, deren Eingabe I_0 von der Natur ausgewählt wird ($\rho_\mathfrak{n}(I_0) > 0$), auch mit positiver Wahrscheinlichkeit erreicht wird. Damit können die Spieler in perfekten Gleichgewichten keine beliebigen Strategien für solche Informationsmengen wählen, auch wenn diese in einem perfekten Gleichgewicht ϕ nicht erreicht werden.

Beweis. Wir wählen zunächst $n \in \mathbb{N}$ beliebig, aber fest. Sei das dazugehörige Spiel $\Gamma = \langle N, M, (s_i), H, Z, \rho_\mathfrak{n}, (\mathcal{I}_j), (C_j)\rangle = \mu(n)$.

Zum Beweis betrachten wir ein beliebiges Profil $\phi \in \Phi^P(\Gamma)$ für das Spiel Γ. Das Profil hat nach der Definition von $\Phi^P(\Gamma)$ die Eigenschaft

$$\forall j \in N : \forall I \in \mathcal{I}_j : \left(\phi_j(I, i) = 1 \;\Rightarrow\; X_i^j(I) = \min_{i' \in M} X_{i'}^j(I) \right).$$

Da es mindestens eine Maschine gibt, die die Last $X_i^j(I)$ minimiert, gibt es mindestens ein Profil in $\Phi^P(\Gamma)$, das diese gewünschte Eigenschaft erfüllt. Damit ist der erste Teil bewiesen.

Es fehlt noch zu zeigen, dass ϕ ein perfektes Gleichgewicht ist. Dazu reicht es aus, eine Testfolge $(\Gamma, \eta^1), (\Gamma, \eta^2), \ldots$ zu finden, so dass ϕ ein Grenzwertgleichgewicht dieser Testfolge ist.

Wir wählen dazu eine beliebige Testfolge $(\Gamma, \eta^1), (\Gamma, \eta^2), \ldots$ und ein festes $k \in \mathbb{N}$. Wir definieren nun ϕ^k und zeigen, dass ϕ^k ein Nash-Gleichgewicht für (Γ, η^k) ist. ϕ^k wird so gewählt sein, dass $\lim_{k\to\infty} \phi^k = \phi$ gilt. Womit dann das Lemma bewiesen wäre.

Wir wählen ϕ^k so, dass für jeden Spieler $j \in N$ und jede Informationsmenge $I \in \mathcal{I}_j$ die Maschine $i \in M$ mit $\phi_j(I, i) = 1$ mit größtmöglicher Wahrscheinlichkeit ausgewählt wird. Sei nun ϕ^k durch

$$\phi_j^k(I, i) = \begin{cases} 1 - \sum_{i' \neq i} \eta^k(I, i'), & \text{falls } \phi(I, i) = 1 \\ \eta^k(I, i), & \text{sonst} \end{cases}$$

definiert. Aus der Definition der Antwortzeiten und $L_j^i(I)$ folgt, dass für jede terminale Historie $h \in Z$, die nach I erreicht wird ($h^{|I|} \in I$) und in der Spieler j die Maschine i wählt ($h_{|I|+1} = i$)

$$C_j(h) = X_i^j(I)$$

gilt. Damit minimiert Spieler j in I seine Antwortzeit in Profil ϕ. Da Spieler j in Profil ϕ^k gerade die in ϕ gewählte Maschine, mit größtmöglicher Wahrscheinlichkeit wählt und allen anderen Maschinen die minimale durch η^k festgelegte Wahrscheinlichkeit zuweist, minimiert der Spieler auch in Profil ϕ^k für das Spiel (Γ, η^k) seine Antwortzeit.

Dies gilt für jeden Spieler und jede Informationsmenge. Damit ist für jeden Spieler $j \in N$ die Strategie ϕ_j^k eine bestmögliche Antwort auf ϕ^k. Das bedeutet, dass ϕ^k ein Nash-Gleichgewicht von dem Spiel (Γ, η^k) ist.

Da $\lim_{k\to\infty} \eta^k = 0$ gilt, folgt, dass $\lim_{k\to\infty} \phi^k = \phi$. Damit ist das Lemma bewiesen. ■

Lemma 5.2 zeigt uns, dass sich die unteren Schranken für die Kompetitivität von $\mathcal{LS}$ bezüglich einer Zielfunktion $f \in \mathfrak{F}$ auf das Koordinationsverhältnis übertragen. Formal erhalten wir aus den Lemmata 5.1 und 5.2 direkt folgenden Satz:

Satz 5.1. *Sei μ ein Modell wie in Lemma 5.1. Sei $f \in \mathfrak{F}$ eine Zielfunktion. Falls $\mathcal{LS}$ ein α-kompetitiver Online-Algorithmus für f bezüglich der durch μ definierten Maschinen und Geschwindigkeiten ist, dann gilt*

$$\mathrm{PA}(\Gamma, f) \leq \mathrm{CR}(\Gamma, f) = \alpha.$$

Weiterhin gilt für die Zielfunktion $f_{[C_{\max}]}$, dass der Preis der Anarchie identisch zum Koordinationsverhältnis ist, falls von der Natur nur Eingaben ausgewählt werden, bei denen die Einlastzeiten Null sind. Dies gilt, da die Reihenfolge der Jobs bei dieser Zielfunktion $f_{[C_{\max}]}$ nicht ausschlaggebend ist, sondern nur die Antwortzeit des letzten Jobs. Da in diesem Modell mit Einlastzeiten Null keine Leerlaufzeiten auf den Maschinen erzeugt werden, kann für jedes $n \in \mathbb{N}$, jedes Spiel $\mu(n)$ und jede Eingabe $a_0 \in \mathcal{E}(n)$ ein Profil ϕ existieren, das einen optimalen Schedule für $f_{[C_{\max}]}$ generiert. Damit erhalten wir aus Satz 5.1 folgendes Korollar.

Korollar 5.1. *Sei μ ein Modell wie in Lemma 5.1. Dann gelten folgende Aussagen:*

- *Falls von der Natur nur Eingaben ausgewählt werden, bei denen die Einlastzeiten Null sind, dann gilt:*

  1. $\mathrm{PA}(\mu, f_{[C_{\max}]}) = \mathrm{CR}(\mu, f_{[C_{\max}]})$.
  2. *Falls die Geschwindigkeiten der Maschinen aus μ identisch sind (das heißt, $s_1 = s_2 = \cdots = s_m = 1$), dann gilt:*

     (a) *Der Preis der Anarchie und das Koordinationsverhältnis sind*

     $$\mathrm{PA}(\mu, f_{[C_{\max}]}) = \mathrm{CR}(\mu, f_{[C_{\max}]}) = 2 - \frac{1}{m}. \qquad \textit{[Gra66]}$$

     (b) *Für $m \in \{2, 3\}$ kann es kein Modell μ' mit Informationspartitionen wie in* (4.1) *geben, bei dem das Koordinationsverhältnis kleiner $2 - \frac{1}{m}$ ist. Das heißt, für alle Modelle μ', bei denen die Informationspartitionen durch* (4.1) *gegeben und die Geschwindigkeiten der Maschinen identisch sind, gilt:*

     $$\mathrm{PA}(\mu', f_{[C_{\max}]}) = \mathrm{CR}(\mu', f_{[C_{\max}]}) \geq 2 - \frac{1}{m}. \qquad \textit{[FKT89]}$$

  3. *Für beliebige Geschwindigkeiten der Maschinen gilt:*

     $$\mathrm{PA}(\mu, f_{[C_{\max}]}) = \mathrm{CR}(\mu, f_{[C_{\max}]}) \in \Theta(\log m). \quad \textit{[AAF+97, CS80]}$$

- *Für Spiele mit beliebigen Einlastzeiten und identischen Maschinen gilt:*

  $$\mathrm{PA}(\mu, f_{[F_{\max}]}) \leq \mathrm{CR}(\mu, f_{[F_{\max}]}) = 3 - \frac{2}{m}. \qquad \textit{[BCM98]}$$

Für die Zielfunktionen $f_{[\sum C_j]}$ und $f_{[\sum F_j]}$ kann gezeigt werden, dass das Koordinationsverhältnis für ein Spiel $\mu(n)$ in $\Omega(n)$ liegt. Das liegt daran, dass es besonders ungünstig ist, falls die Jobs in nicht-aufsteigender Reihenfolge bezüglich der Jobgröße bearbeitet werden. Ein Beispiel, welches verdeutlicht, dass das Koordinationsverhältnis in $\Omega(n)$ liegt, kann mittels m identischen Maschinen konstruiert werden. Sei dazu ein $q > 1$ vorgegeben und ein $n \in \mathbb{N}$, so dass $n \geq qm$. Betrachten wir nun die Eingabe a_0, die aus genau qm Jobs besteht. Die Einlastzeiten aller Jobs sind Null. Die ersten m Jobs haben die Größe 2^{q-1}, die nächsten m die Größe 2^{q-2}, usw. Die letzten m Jobs haben die Größe $2^0 = 1$. Da in einem perfekten Gleichgewicht die Jobs wie bei dem List Scheduling Algorithmus verteilt werden, werden zuerst alle m Jobs der Größe 2^{q-1} auf die m Maschinen verteilt. Auch die weiteren Jobs werden gleichmäßig auf die Maschinen aufgeteilt, so dass auf jeder Maschine ein Job jeder Größe platziert ist. Die Jobs werden in absteigender Reihenfolge ihrer

Größe bearbeitet. Optimal ist die Bearbeitung in aufsteigender Reihenfolge. Daraus ergibt sich

$$\mathrm{CR}(\mu(n), a_0, f_{[\sum C_j]}) \geq \frac{m \cdot n2^{n+1}}{m \cdot 2^{n+2}} = \frac{n}{2}.$$

Da die Einlastzeiten Null sind, ist die Antwortzeit identisch zur Flusszeit und damit gilt

$$\mathrm{CR}(\mu(n), a_0, f_{[\sum F_j]}) = \mathrm{CR}(\mu(n), a_0, f_{[\sum C_j]}) \geq \frac{n}{2}.$$

Allerdings ist für eine bestimmte Klasse von Eingaben die Verteilung, die durch den List Scheduling Algorithmus vorgegeben wird, eine sehr gute Verteilung. Falls nur Eingaben vorkommen, bei denen die Größe der Jobs monoton wachsend ist, erhalten wir im Fall von identischen Maschinen mit Einlastzeiten Null als Folgerung aus Satz 5.1 bessere Ergebnisse.

Korollar 5.2. *Sei μ ein Modell wie in Lemma 5.1 für das zusätzlich gilt:*

1. $\forall i \in M : s_i = 1$ *(identische Maschinen),*
2. $\forall n \in \mathbb{N} : \forall a_0 \in \mathcal{E}(n) : \rho_{\mathfrak{n}}(a_0) > 0 \Rightarrow \not\exists 1 \leq j < l(a_0) : x_j(a_0) \geq x_{j+1}(a_0))$ *(die Jobs sind immer in nicht-absteigender Reihenfolge gegeben) und*
3. $\forall n \in \mathbb{N} : \forall a_0 \in \mathcal{E}(n) : (\rho_{\mathfrak{n}}(a_0) > 0 \Rightarrow \forall 1 \leq l \leq |a_0| : r_l(a_0) = 0)$ *(die Einlastzeiten sind Null).*

Dann gilt für den Preis der Anarchie und das Koordinationsverhältnis

$$\mathrm{PA}(\mu, f_{[C_{\max}]}) = \mathrm{CR}(\mu, f_{[C_{\max}]}) = 2 - \frac{1}{m} \qquad \textit{[Gra66]}$$

und

$$\mathrm{PA}(\mu, f_{[\sum C_j]}) = \mathrm{CR}(\mu, f_{[\sum C_j]}) = 1. \qquad \textit{[CMM67]}$$

Wenn wir von diesem Spezialfall absehen, müssen wir ein anderes Modell finden, um auch gute Resultate für den Preis der Anarchie und das Koordinationsverhältnis im Fall der Zielfunktionen $f_{[\sum C_j]}$ und $f_{[\sum F_j]}$ zu erhalten. Das heißt, wir suchen ein Maschinenmodell, bei dem der Preis der Anarchie und das Koordinationsverhältnis für beide Zielfunktionen nahe der Eins liegen.

Kapitel 6

Priorisierung der Jobs

Wir haben gesehen, dass das Modell aus Kapitel 5 für identische Maschinen gute Ergebnisse bei der Zielfunktion $f_{[F_{\max}]}$ und im Fall von Spielen, bei denen die Einlastzeiten der Jobs Null sind, auch für die Zielfunktion $f_{[C_{\max}]}$ liefert. Das Problem allerdings ist, dass es ein sehr schlechtes Modell bezüglich der Zielfunktionen $f_{[\sum C_j]}$ und $f_{[\sum F_j]}$ darstellt.

In diesem Kapitel werden wir uns, sofern nicht explizit anders angegeben, auf Spiele, bei denen die Einlastzeiten der Jobs Null sind, beschränken. Um die Notation zu vereinfachen, werden wir in den Eingaben die Einlastzeiten nicht aufführen.

Wir wollen nun das bisherige Modell soweit abändern, so dass wir möglichst gute Ergebnisse für $f_{[C_{\max}]}$ und $f_{[\sum C_j]}$ erhalten und dabei das ganze System trotzdem ein gutes Modell für die Spieler bleibt. Das heißt, es soll für die Spieler einfach sein, eine Strategie zu berechnen. Auch sollen nicht unnötig Jobs zurückgehalten werden, wie dies bei dem Modell aus Kapitel 4 geschieht. Wir werden sehen, dass wir Leerlaufzeiten einfügen müssen, allerdings werden wir dafür sorgen, dass ein Job nicht unnötig lange im Vergleich zu seiner Größe von diesen Leerlaufzeiten beeinträchtigt wird.

6.1 Einführende Überlegungen

Eine Idee für eine Verbesserung ist die Verwendung der SPT-Regel (Shortest-Processing-Time-Regel) für die einzelnen Maschinen. Diese Regel besagt, dass eine Maschine immer von den noch zu bearbeitenden Jobs denjenigen auswählt, dessen Bearbeitungszeit am geringsten ist.

Werden auch die Jobs ihrer Größe nach – zuerst die kleinen und dann die großen Jobs – den Maschinen zugewiesen, so werden die Jobs einer Größe ungefähr gleichmäßig auf die Maschinen verteilt. Diese Verteilung entspricht dem Verhalten des List Scheduling Algorithmus auf einer monoton wachsend sortierten Eingabe.

Conway, Maxwell und Miller [CMM67] haben gezeigt, dass bei identischen Maschinen die SPT-Regel eine optimale Lösung bezüglich der durchschnittlichen Antwortzeit $f_{[\sum C_j]}$ generiert. Graham [Gra66] hat bewiesen, dass bei identischen Maschinen auch bei der SPT-Regel der List Scheduling Algorithmus ein Approximationsalgorithmus mit Approximationsgüte $2 - 1/m$ bezüglich der maximalen Antwortzeit $f_{[C_{\max}]}$ ist. Das bedeutet, zur Verbesserung des Koordinationsverhältnisses bezüglich der durchschnittlichen Antwortzeit $f_{[\sum C_j]}$ macht es Sinn, die Jobs nach der SPT-Regel auf den Maschinen zu bearbeiten.

Bei den in dieser Arbeit beschriebenen Scheduling-Problemen ist dieses Modell, bei dem die Jobs ihrer Größe nach auf den Maschinen bearbeitet werden, möglich, da wir den Ablauf eines Spiels in zwei Teile unterteilen: die Verteilung der Jobs auf die Maschinen gefolgt von der Abarbeitung der Jobs. Das heißt, die Maschinen kennen alle zu bearbeitenden Jobs, wenn sie anfangen den ersten Job abzuarbeiten.

Dieses Modell hat aber einen Nachteil. Die Spieler versuchen ihre erwartete Antwortzeit zu minimieren. Das kann bedeuten, dass ein Spieler in diesem Modell eine Maschine auswählt, die ungünstig bezüglich der schon bekannten Jobs ist, da er erwartet, dass einer der nächsten Spieler einen kleineren Job besitzt und diesen dann auf die bezüglich der bekannten Jobs günstigeren Maschine platziert. Auch wenn dieses Verhalten die erwartete Antwortzeit des Spielers minimiert, folgt daraus nicht, dass der Preis der Anarchie dadurch auch gering ausfällt. Der Grund dafür ist, dass wir beim Preis der Anarchie eine klassische Worst-Case-Analyse durchführen und es eine Eingabe geben kann, bei der dieses Verhalten der Spieler besonders schlecht ist.

Betrachten wir dazu ein einfaches Beispiel. Seien zwei Maschinen mit den Geschwindigkeiten $s_1 = 1$ und $s_2 = 1 - \varepsilon$ für ein $\varepsilon > 0$ gegeben. Wir betrachten ein Spiel, bei dem es deutlich mehr als zwei verschiedene Spieler gibt. $\rho_{\mathfrak{n}}$ ist nun so gewählt, dass bei allen Eingaben, die von der Natur mittels $\rho_{\mathfrak{n}}$ ausgewählt werden, alle Jobs eine Größe kleiner oder gleich $x_{\max} \in \mathbb{N}$ haben. $\rho_{\mathfrak{n}}$ wählt jede dieser Eingaben mit gleicher Wahrscheinlichkeit aus. Damit haben wir ein endliches Scheduling-Spiel und wissen, dass es mindestens ein perfektes Gleichgewicht gibt.

Betrachten wir nun die Aktionen der Spieler, die die ersten zwei Züge machen.

Mit einer positiven Wahrscheinlichkeit wird von der Natur die Eingabe $h_0 = ((x_{j_1}, j_1), (x_{j_2}, j_2))$ der Länge Zwei ausgewählt, bei der der Spieler des ersten Zuges einen Job der Größe $x_{j_1} = x_{\max}$ und der Spieler des zweiten Zuges einen Job der Größe $x_{j_2} = x_{\max}$ hat. Falls jede Eingabe, bei denen die Jobs eine maximale Größe von $x_{\max}$ haben, mit gleicher Wahrscheinlichkeit ausgewählt wird und falls es genügend viele Spieler gibt ($n >> 2$), so muss der Spieler j_1 damit rechnen, dass die meisten anderen Jobs vor ihm bearbeitet werden. Das heißt, in einem perfekten Gleichgewicht muss er damit rechnen, dass Spieler mit kleineren Jobs ihren Job auf die Maschine mit Geschwindigkeit s_1 platzieren. Das heißt, bei geeigneter Wahl von n, ε und $x_{\max}$ wählt Spieler j_1 die zweite Maschine mit der Geschwindigkeit $s_2 = 1 - \varepsilon$.

Diese Überlegungen kann man nun auf Spieler j_2 übertragen. Falls es genügend Spieler gibt und die möglichen Eingaben gleichverteilt ausgewählt werden, wird auch dieser Spieler davon ausgehen, dass es für ihn günstiger ist, die Maschine mit der Geschwindigkeit s_2 auszuwählen. Das heißt, für diese Eingabe erhalten wir einen Schedule, bei dem beide Spieler die gleiche Maschine auswählen und die andere Maschine keinen einzigen Job bearbeitet. Entsprechend kann man dieses Beispiel auf Spiele mit mehr Maschinen und Eingaben, bei denen mehr Spieler teilnehmen, übertragen.

Das Problem entsteht dadurch, dass die Spieler die erwartete Antwortzeit für alle möglichen zukünftigen Ereignisse minimieren. Das führt dazu, dass in einem perfekten Gleichgewicht die Jobs der Größe 1 eine erwartete gleichmäßige Auslastung der Maschinen erzeugen. Es gibt keinen anderen Job der kleiner ist und vor diesen Jobs bearbeitet würde. Wäre dies nicht der Fall, könnte ein Spieler mit einem Job der Größe 1 seine Strategie ändern und die erwartete Antwortzeit verbessern. Ähnlich folgt dies dann für Spieler mit einem Job der Größe 2, da sie davon ausgehen können, dass die Spieler mit Jobs der Größe 1 schon ihre Jobs gleichmäßig in einem perfekten Gleichgewicht verteilt haben. Für eine feste Eingabe aber muss der daraus resultierende Schedule, wie wir am Beispiel oben gesehen haben, nicht die Jobs gleichmäßig verteilen.

Diese ungleichmäßige Verteilung entsteht dadurch, dass die Spieler erwarten, dass die Maschinen ungefähr zeitgleich mit der Bearbeitung kürzerer Jobs fertig werden, dies aber für eine feste Eingabe nicht mehr garantiert werden kann. Dieses Problem können wir umgehen, indem wir den Spielern garantieren, dass die Maschinen zeitgleich mit der Bearbeitung der Jobs einer Größe beginnen. In diesem Fall wird die Strategie der Spieler unabhängig von der Strategie der Spieler mit einem kleineren Job. Der direkte Weg dies zu erreichen besteht darin, dass eine Maschine nur mit der Bearbeitung eines Jobs der Größe x beginnt, falls alle Maschinen die ihnen zugewiesenen Jobs der

Größen $< x$ fertig bearbeitet haben. Damit wird Leerlauf auf den Maschinen erzeugt. Dies wird aber dafür sorgen, dass in einem perfekten Gleichgewicht Spieler j_1 seinen Job auf der schnelleren und Spieler j_2 seinen Job auf der langsameren Maschinen platziert.

Der Leerlauf erzeugt allerdings Schwierigkeiten bei dem Koordinationsverhältnis. Betrachten wir dazu ein Spiel mit m identischen Maschinen. Sei eine Eingabe gegeben, bei der n Jobs der Größen $1, 2, \ldots, n$ vorkommen. Die Jobs würden durch das Modell nacheinander ausgeführt. Ein optimaler Schedule könnte die Leerlaufzeiten ausnutzen und dadurch deutlich bessere Antwortzeiten generieren. In einem optimalen Schedule würde man die Jobs gleichmäßig der Reihenfolge nach auf die Maschinen verteilen. Seien nun die Jobs $(j_{l,k})$ für $1 \leq l \leq m$ und $0 \leq k \leq k_{\max}$ gegeben, wobei die Größe $x_{l,k}$ von Job $j_{l,k}$ durch $x_{l,k} := k \cdot m + j$ definiert ist. Das heißt, dass wir $(k_{\max} + 1) \cdot m$ Jobs der Größen $1, 2, \ldots, (k_{\max} + 1) \cdot m$ verteilen müssen. Nach unserem Modell werden diese nacheinander abgearbeitet und wir erhalten für Job $j_{l,k}$ die Antwortzeit in einer terminalen Historie h mit dieser Eingabe (es ist hier zu beachten, dass für diese Eingabe die Antwortzeiten unabhängig von der Wahl der Maschine sind, da jede Jobgröße nur einmal vorkommt)

$$C_{j_{l,k}}(h) = \sum_{r=1}^{x_{l,k}} r = \frac{x_{l,k}(x_{l,k}+1)}{2} = \frac{(km+l)(km+l+1)}{2} \in \Theta\left(k^2 m^2\right).$$

In einem von SPT generierten Schedule wird der Job $j_{l,k}$ auf Maschine l platziert. Das heißt, wir erhalten als Antwortzeit

$$\begin{aligned} C_{j_{l,k}}^{\mathrm{SPT}} &= \sum_{r=0}^{k} x_{l,r} = \sum_{r=0}^{k} (r \cdot m + l) \\ &= kl + m \cdot \sum_{r=0}^{k} r = kl + m \cdot \frac{k(k+1)}{2} \in \Theta\left(k^2 m\right). \end{aligned}$$

Da jeder Job in diesem Fall um $\Omega(m)$ schneller in einer optimalen Lösung bearbeitet wird, ist das Koordinationsverhältnis bei der durchschnittlichen Antwortzeit mindestens $\Omega(nm)$. Das heißt, wir haben durch dieses Modell keine Verbesserung erzielt. Falls wir allerdings die Leerlaufzeiten verringern könnten, dann könnten wir bessere Ergebnisse erreichen.

Die Idee ist, dass wir die Jobs in Klassen aufteilen. Die Maschinen bearbeiten die Jobs klassenweise. Zuerst werden die Klassen mit Jobs kleiner Größe bearbeitet. Innerhalb der Klasse wird die Reihenfolge der Jobs aus der Eingabe eingehalten. Durch eine entsprechende Definition der Klassifizierung kann

erreicht werden, dass die relative Verzögerung der Jobs einer späteren Klasse durch die Leerlaufzeiten bezüglich deren Jobgröße durch eine Konstante beschränkt ist.

6.2 Einteilung der Jobs in Klassen

Wir unterteilen die Jobs folgendermaßen in Klassen. Für $k \in \mathbb{N}_0$ besteht die *Klasse* k aus allen Jobs der Größen $\left[2^k, 2^{k+1}\right)$. Das heißt, die Jobgrößen verdoppeln sich von Klasse zu Klasse. Wir werden zeigen, dass wir durch diese Einteilung in Klassen sicher stellen, dass die Verzögerung eines Jobs, die durch die Leerlaufzeiten erzeugt wird, im Verhältnis zu seiner Jobgröße beschränkt ist. Mit dieser Klassifizierung umgehen wir die oben beschriebenen Probleme sowohl bei dem Preis der Anarchie als auch dem Koordinationsverhältnis.

Wir werden einen ähnlichen Ansatz wie Avrahami und Azar [AA03] verfolgen, betrachten hier allerdings keine zentrale Verteilung der Jobs, führen Leerlaufzeiten ein und erlauben auch kein präemptives Scheduling. Der Ansatz von Avrahami und Azar kann nicht direkt übertragen werden, da im Allgemeinen die Schedules, die von dem in Avrahami und Azar [AA03] vorgestellten Algorithmus generiert werden, in dem entsprechend definierten Spiel keine perfekten oder Nash-Gleichgewichte sind.

Zur Einteilung der Jobs in Klassen betrachten wir nun einen beliebigen Zustand, bzw. Historie, $h \in H$ des Spiels. Zur Erinnerung, h besteht aus der Eingabe h_0, die von der Natur ausgewählt wurde, und den schon getätigten Zügen $h_1, h_2, \ldots, h_{|h|}$ der Spieler $p_1(h), p_2(h), \ldots, p_{|h|}(h)$. Der Zug eines Spielers ist die Wahl einer Maschine. Da in der Eingabe alle Jobs und deren Größen aufgeführt sind, ist durch eine Historie h implizit die Klasse der Jobs aller Spieler festgelegt.

Wir führen nun folgende Hilfsbezeichnungen ein, um die Notation zu vereinfachen:

- $\kappa(h, j)$ ist die Klasse des Jobs von Spieler $j \in N$ bei Historie h. Das heißt, der Job j hat eine Größe von

  $$2^{\kappa(h,j)} \leq x_l(h) < 2^{\kappa(h,j)+1},$$

  wobei l der Index des Spielers j in der Eingabe h_0 ist. Das heißt, $p_l(h_0) = j$.

- $N_k^i(h)$ ist die Menge aller Jobs aus der Klasse $k \in \mathbb{N}_0$, die in Historie h schon auf die Maschine i verteilt wurden.

$$N_k^i(h) := \{1 \leq l \leq |h| \mid \kappa(h, p_l(h)) = k, h_l = i\}.$$

- $N_k(h)$ ist die Menge aller Jobs der Klasse $k \in \mathbb{N}_0$, die schon auf eine Maschine verteilt wurden.

$$N_k(h) := \bigcup_{i \in M} N_k^i(h).$$

- $L_k^i(h)$ ist die Last auf Maschine $i \in M$ bei Betrachtung nur der Jobs aus der Klasse $k \in \mathbb{N}_0$, die in Historie h schon verteilt wurden. Das heißt,

$$L_k^i(h) := \sum_{l \in N_k^i(h)} \frac{x_l(h)}{s_i}.$$

Wir werden im Folgenden ein Modell für online Scheduling-Spiele konstruieren. Das bedeutet, dass die Informationspartitionen dadurch festgelegt sind, dass die Spieler sowohl ihren eigenen Job als auch die Jobs und Züge der vorangegangen Spieler kennen. Formal sind die Informationspartitionen in einem online Scheduling-Spiel, wie gehabt, durch

$$\mathcal{I}_j := \{\{h \in H_j \mid \mathfrak{A}(h) = \mathfrak{a}, \mathfrak{X}(h) = \mathfrak{x}\} \mid \mathfrak{a} \in M^{|h|}, \mathfrak{x} \in \mathbb{N}^{|h|+1}\}$$

definiert, wobei

$$\mathfrak{A}(h) = \big(h_1, h_2, \ldots, h_{|h|}\big)$$

das Tupel der schon gespielten Aktionen ist und

$$\mathfrak{X}(h) = \big(x_1(h), x_2(h), \ldots, x_{|h|+1}(h)\big)$$

das Tupel der Jobgrößen der Spieler, die den ersten bis $(|h| + 1)$-ten Zug durchführen, darstellt. Nach den Definitionen von $\kappa(\cdot, \cdot)$, $N_k^i(\cdot)$, $N_k(\cdot)$ und $L_k^i(\cdot)$ sind die Werte dieser Funktionen identisch für alle Historien aus einer Informationsmenge $I \in \mathcal{I}_j$ eines beliebigen Spielers j. Aus diesem Grund vereinfachen wir die Notation und definieren für alle Informationsmengen I und ein beliebiges $h \in I$

$$\kappa(I, j) := \kappa(h, j), \qquad N_k^i(I) := N_k^i(h),$$
$$N_k(I) := N_k(h) \qquad \text{und} \qquad L_k^i(I) := L_k^i(h).$$

Wir definieren das Maschinenmodell nun so, dass zuerst alle Jobs einer Klasse abgearbeitet werden. Wir wollen damit erreichen, dass die Jobs innerhalb einer Klasse, wie beim List Scheduling Algorithmus verteilt werden. Dies erreichen wir, indem wir erst auf einer Maschine mit der Berechnung der Jobs der Klasse k beginnen, wenn diese und alle anderen Maschinen die Jobs der Klassen $< k$ fertig bearbeitet haben. Das erzeugt allerdings Leerlauf auf den Maschinen. Wir werden aber zeigen, dass diese Leerlaufzeit gering genug ist, so dass dieses Modell nicht zu stark die Bearbeitungszeit verlängert. Zudem erfordert es, dass die Maschinen sich koordinieren oder ihnen die komplette Verteilung mitgeteilt wird, so dass sie den richtigen Leerlauf erzeugen können.

Innerhalb einer Klasse werden die Jobs in der Reihenfolge, in der sie einer Maschine zugewiesen wurden, bearbeitet.

Sei $h \in Z$ eine beliebige terminale Historie. Das heißt, dass alle Spieler, die einen Job in der Eingabe h_0 haben, ihren Zug bereits durchgeführt haben und somit das Spiel beendet ist. Sei $j \in N$ ein beliebiger Spieler, der einen Job bei Eingabe h_0 besitzt. Dann bezeichnen wir mit $\mathrm{Pref}(h, j)$ die Zeit, die von den Maschinen benötigt wird, um die Jobs der Klassen kleiner als $\kappa(h, j)$ zu bearbeiten inklusive der generierten Leerlaufzeiten. Dies bedeutet, dass $\mathrm{Pref}(h, j)$ auch der Zeitpunkt ist, an dem die Maschinen mit der Bearbeitung der Jobs der Klasse $\kappa(h, j)$ beginnen. Es gilt

$$\mathrm{Pref}(h, j) := \sum_{k=0}^{\kappa(h,j)-1} \max_{i \in M} \left(\sum_{p \in N_k^i(h)} \frac{x_p(h)}{s_i} \right).$$

Sei nun wieder $j \in N$ ein Spieler, der in Eingabe h_0 einen Job besitzt. Der Index von Spieler j in Eingabe h_0 sei l und somit gilt $p_l(h_0) = j$. Wir bezeichnen nun mit $\mathrm{Suff}(h, j, l)$ den Anteil der Bearbeitungszeit, der von dem Job des Spielers j und allen anderen Jobs der gleichen Klasse auf der gleichen Maschine vor Durchführung des Jobs von Spielers j verursacht wird. Das heißt, dass $\mathrm{Pref}(h, j) + \mathrm{Suff}(h, j, l)$ die gesamte Antwortzeit von Spieler j ist. Formal ist $\mathrm{Suff}(h, j, l)$ durch

$$\mathrm{Suff}(h, j, l) := \sum_{p \in N_{\kappa(I,j)}^{h_l}(h), p \leq l} \frac{x_p(h)}{s_{h_l}}$$

definiert, wobei $N_k^i(h)$, wie oben beschrieben, die Menge aller Indizes von Jobs in Eingabe h_0 ist, deren Jobs in h aus der Klasse k sind und auf Maschine i bearbeitet werden.

Damit erhalten wir folgende Definition der Antwortzeitfunktionen:

$$C_j(h) := \begin{cases} \text{Pref}(h,j) + \text{Suff}(h,j,l), & \text{falls } \exists l : 1 \leq l \leq |h_0| \text{ mit } p_l(h_0) = j \\ 0, & \text{sonst.} \end{cases} \tag{6.1}$$

Betrachten wir nochmals genauer das Maschinenmodell. Sei nun wieder $h \in Z$ eine beliebige terminale Historie und damit ein Ausgang des Spiels. Sei $j \in N$ ein Spieler, der bei Eingabe h_0 einen Job besitzt. Dann wird die Antwortzeit dieses Spielers von folgenden Jobs beeinflusst:

1. Alle Jobs, die zu einer kleineren Klasse gehören als Spieler j, beeinflussen die Antwortzeit des Jobs von Spieler j. Durch die Definition von $\text{Pref}(h,j)$ beeinflussen alle Jobs dieser Klassen direkt oder indirekt die Antwortzeit. Nicht nur die Jobs, die auf der gleichen Maschine bearbeitet werden, sind ausschlaggebend. Das bedeutet, die Antwortzeit von Spieler j hängt von allen Spielern aus der Menge

$$\bigcup_{k=0}^{\kappa(h,j)-1} N_k(h)$$

   ab.

2. Alle Jobs, die zu der gleichen Klasse $\kappa(h,j)$ gehören, auf der gleichen Maschine bearbeitet werden und vor Spieler j ihren Zug getätigt haben, beeinflussen die Antwortzeit des Jobs von Spieler j. Sei l der Index von dem Spieler j in der Eingabe h_0. Das heißt, Spieler j macht den l-ten Zug ($p_l(h_0) = j$). Dann beeinflussen nur die Jobs gleicher Klasse aus der Menge $N^{h_l}_{\kappa(h,j)}(h^{l-1})$ die Antwortzeit von Spieler j, wobei $h^l = h_0 h_1 \dots h_l$ der l-te Präfix von h ist.

Das bedeutet aber auch, dass die Jobs einer Klasse größer als $\kappa(h,j)$ die Antwortzeit des Spielers j nicht beeinflussen können. Sie werden nach der Definition des Maschinenmodells nach Spieler j bearbeitet. Auch die Spieler der gleichen Klasse, die nach Spieler j am Zug sind, beeinflussen die Antwortzeit nicht.

Wie wir schon beschrieben haben, beginnen alle Maschinen gleichzeitig mit der Bearbeitung der Jobs einer Klasse. Dadurch werden Leerlaufzeiten auf den Maschinen hinzugefügt. Die so hinzugefügten Leerlaufzeiten auf den Maschinen sollen den Effekt haben, dass ein Spieler sich alleine auf die Jobs der gleichen Klasse, die ihm schon bekannt sind, konzentrieren kann. Wir werden

im Folgenden eingeschränkte Informationspartitionen betrachten, bei denen der Spieler nur die anderen Jobs der eigenen Klasse kennt, die schon einen Zug durchgeführt haben. Da nur die Last $L^i_{\kappa(I,j)}$, die von den Jobs der eigenen Klasse $\kappa(I,j)$ auf Maschine i verursacht wird, für Spieler j von Bedeutung ist, reduzieren wir die Informationen auf die Kenntnis der Last $L^i_{\kappa(I,j)}$. Die Informationspartitionen $\widetilde{\mathcal{I}}_1, \widetilde{\mathcal{I}}_2, \ldots, \widetilde{\mathcal{I}}_n$ beschreiben diese Informationen.

$$\widetilde{\mathcal{I}}_j := \{\{h \in H_j \mid \forall i \in M : L^i_{\kappa(h,j)}(h) = l_i, \\ x_{|h|+1}(h) = x\} \mid (l_i) \in \mathbb{Q}^M, x \in \mathbb{N}\}. \quad (6.2)$$

Diese Informationspartitionen $(\widetilde{\mathcal{I}}_j)$ stellen eine Reduzierung der Informationen gegenüber denen, die durch die Informationspartitionen $(\mathcal{I}_j)$ eines online Scheduling-Spiels bekannt sind, dar. Die Informationspartitionen $(\widetilde{\mathcal{I}}_j)$ sind eine gröbere Partitionierung der Historien (H_j) gegenüber den Informationspartitionen $(\mathcal{I}_j)$. Das heißt, dass es zu jeder Informationsmenge $I \in \mathcal{I}_j$ eine Informationsmenge $\widetilde{I} \in \widetilde{\mathcal{I}}_j$ gibt, die diese beinhaltet $(I \subset \widetilde{I})$.

Wir definieren nun bezüglich dieser Inklusion eine Äquivalenzrelation. Die Informationspartitionen $I, I' \in \mathcal{I}_j$ sind äquivalent – $I \equiv I'$ – genau dann, wenn es ein $\widetilde{I} \in \widetilde{\mathcal{I}}_j$ gibt, so dass $I \subset \widetilde{I}$ und $I' \subset \widetilde{I}$. Das heißt, $I \equiv I'$, wenn die Lasten auf allen Maschinen, die von den vorangegangenen Spielern aus der gleichen Klasse verursacht wurden, identisch sind.

Wir werden im Folgenden erläutern, dass wir uns bei online Scheduling-Spielen mit dem hier eingeführten Maschinenmodell und identischen Maschinen auf ein Spiel mit den durch $\widetilde{\mathcal{I}}_1, \widetilde{\mathcal{I}}_2, \ldots, \widetilde{\mathcal{I}}_n$ definierten Informationspartitionen beschränken können. Auch wenn es einsichtig erscheint, dass wir uns auf diese Informationspartitionen konzentrieren können, ist dies nicht notwendigerweise der Fall. Es existieren Abhängigkeiten zwischen den Spielern verschiedener Klassen. Auch wenn ein Spieler der Klasse k nicht von der Strategie eines Spielers der Klasse $k' > k$ beeinflusst wird, so kann er bei den Informationspartitionen $(\mathcal{I}_j)$ seine Strategie in der Abhängigkeit von diesem Spieler wählen. Deswegen könnte es sein, dass eine Reduzierung der Informationen auf die Partitionen $(\widetilde{\mathcal{I}}_j)$ besonders ungünstige perfekte Gleichgewichte aus der Betrachtung entfernt.

Wir vermuten, dass wir im allgemeinen Fall die durch diese Abhängigkeiten erzeugten perfekten Gleichgewichte für die Analyse nicht betrachten müssen. Wir werden in Satz 6.1 dies für den Fall von identischen Maschinen zeigen. Dazu benötigen wir weitere Hilfsdefinitionen. Den Beweis für Satz 6.1 verschieben wir auf Kapitel 6.6.

Sei $\mathcal{G}(\Gamma)$ die Menge aller perfekten Gleichgewichte für ein Spiel Γ und $\mathcal{G}_{\equiv}(\Gamma)$ die Menge aller perfekten Gleichgewichte von Γ, für die gilt, dass die Spieler die gleiche Strategie für äquivalente Informationsmengen spielen. Das heißt,

$$\forall \phi \in \mathcal{G}_{\equiv}(\Gamma) : \forall j \in N : \forall I, I' \in \mathcal{I}_j : I \equiv I' \Rightarrow \phi_j(I, \cdot) = \phi_j(I', \cdot).$$

Sei nun $\Gamma = \langle N, M, (s_i), H, Z, (\mathcal{I}_j), \rho_{\mathfrak{n}}, (C_j) \rangle$ ein online Scheduling-Spiel. $\widetilde{\Gamma} = \left\langle N, M, (s_i), H, Z, (\widetilde{\mathcal{I}}_j), \rho_{\mathfrak{n}}, (C_j) \right\rangle$ ist das abgewandelte Spiel, das mit Γ bis auf die Wahl der Informationspartitionen übereinstimmt. Bei dem Spiel $\widetilde{\Gamma}$ sind diese, wie oben, durch $\widetilde{\mathcal{I}}_1, \widetilde{\mathcal{I}}_2, \ldots, \widetilde{\mathcal{I}}_n$ definiert. Sei $\widetilde{\mu}$ das Modell dieser Spiele $\widetilde{\Gamma}$.

Die Profile aus $\mathcal{G}_{\equiv}(\Gamma)$ können direkt auf Profile für ein Spiel $\widetilde{\Gamma}$ übertragen werden. Entsprechend kann jedes Profil ϕ von $\widetilde{\Gamma}$ als ein Profil von Γ aufgefasst werden, bei dem die Spieler in äquivalenten Spielzuständen die gleiche Strategie spielen.

Wir werden zeigen, dass wir uns auf perfekte Gleichgewichte aus der Menge $\mathcal{G}(\widetilde{\Gamma})$ beschränken können. Dazu müssen zwei Dinge gezeigt werden. Zum einen muss es in $\widetilde{\Gamma}$ solche perfekten Gleichgewichte geben. Das heißt, dass die Menge $\mathcal{G}(\widetilde{\Gamma})$ nicht leer sein darf. Des Weiteren muss gezeigt werden, dass für alle Eingaben $a_0 \in \mathcal{E}(n)$ mit $\rho_{\mathfrak{n}}(a_0) > 0$ (alle Eingaben, die von der Natur ausgewählt werden können) und Zielfunktionen $f \in \left\{ f_{[C_{\max}]}, f_{[\sum C_j]} \right\}$

$$\sup\{ \mathrm{E} f(\Gamma, a_0, \phi) \mid \phi \in \mathcal{G}(\Gamma) \} = \sup\{ \mathrm{E} f(\widetilde{\Gamma}, a_0, \psi) \mid \psi \in \mathcal{G}(\widetilde{\Gamma}) \} \tag{6.3}$$

gilt, wobei $\mathrm{E} f(\Gamma', a_0', \phi')$ der erwartete Wert der Zielfunktion f bei Spiel Γ' für Eingabe a_0' und Profil ϕ' ist. Zur Erinnerung, wir vergleichen bei dem Preis der Anarchie und dem Koordinationsverhältnis das perfekte Gleichgewicht, das die Zielfunktion maximiert, mit einer optimalen Strategie bzw. einem optimalen Schedule. Das bedeutet, wenn Gleichung (6.3) erfüllt ist, dann gilt direkt $\mathrm{CR}(\mu, f) = \mathrm{CR}(\widetilde{\mu}, f)$ (das Koordinationsverhältnis von dem Spielmodell μ bezüglich der Zielfunktion f ist identisch zu dem Koordinationsverhältnis von dem Spielmodell $\widetilde{\mu}$ bezüglich der gleichen Zielfunktion f). Das gleiche gilt auch für den Preis der Anarchie $\mathrm{PA}(\mu, f) = \mathrm{PA}(\widetilde{\mu}, f)$. Hier werden zwar die perfekten Gleichgewichte mit einer optimalen Strategie, die sich an die gleichen Spielregeln halten muss wie die Spieler, verglichen, aber der Vergleich wird immer für eine feste Eingabe durchgeführt und so werden die optimalen Strategien immer unabhängig von den Informationspartitionen bestimmt. Siehe dazu auch Gleichungen (2.1) und (2.2) auf Seite 33.

Der folgende Satz fasst die Aussagen zusammen und besagt, dass jedes Spiel $\widetilde{\Gamma}$ des Modells $\widetilde{\mu}$ mindestens ein perfektes Gleichgewicht besitzt.

Satz 6.1. *Seien μ und $\widetilde{\mu}$ Modelle, wie oben beschrieben, und sei $n \in \mathbb{N}$ beliebig, aber fest gewählt. Dann gilt für $\Gamma = \mu(n)$ und $\widetilde{\Gamma} = \widetilde{\mu}(n)$:*

- $\mathcal{G}(\Gamma) \neq \emptyset$, $\mathcal{G}_{\equiv}(\Gamma) \neq \emptyset$, $\mathcal{G}(\widetilde{\Gamma}) \neq \emptyset$,
- *jedes perfekte Gleichgewicht $\phi \in \mathcal{G}(\widetilde{\Gamma})$ ist auch ein perfektes Gleichgewicht von Γ ($\phi \in \mathcal{G}(\Gamma)$) und*
- *für identische Maschinen – $\forall i \in M : s_i = 1$ –, jede Zielfunktion $f \in \{f_{[C_{\max}]}, f_{[\sum C_j]}\}$ und jede Eingabe $a_0 \in \mathcal{E}(n)$ mit $\rho_{\mathfrak{n}}(a_0) > 0$ gilt*

$$\sup\{\, \mathrm{E}f(\Gamma, a_0, \phi) \mid \phi \in \mathcal{G}(\Gamma)\} = \sup\{\, \mathrm{E}f(\Gamma, a_0, \phi) \mid \phi \in \mathcal{G}_{\equiv}(\Gamma)\}$$

und

$$\sup\{\, \mathrm{E}f(\Gamma, a_0, \phi) \mid \phi \in \mathcal{G}_{\equiv}(\Gamma)\} = \sup\{\, \mathrm{E}f(\widetilde{\Gamma}, a_0, \phi) \mid \phi \in \mathcal{G}(\widetilde{\Gamma})\}.$$

Wir werden uns nun auf Spiele konzentrieren, die eine eingeschränkte Informationspartition, wie in Gleichung (6.2), besitzen. In den Kapiteln 6.3 bis 6.5 werden wir diese Spiele analysieren. Danach präsentieren wir in Kapitel 6.6 den Beweis von Satz 6.1. Für die in den Kapitel 6.3 bis 6.5 durchgeführte Analyse wird nicht benötigt, dass Satz 6.1 gültig ist. Im Gegenteil, wir werden Lemmata, die wir in Kapitel 6.3 beweisen, zum Beweis von Satz 6.1 verwenden.

Wir werden im Folgenden das Spielmodell $\widetilde{\mu}$ betrachten. Das heißt, das Modell $\widetilde{\mu}$ ist so definiert, dass für jede Anzahl an Spielern $n \in \mathbb{N}$ und das dazugehörige Scheduling-Spiel $\widetilde{\Gamma} = \langle N, M, (s_i), H, Z, (\mathcal{I}_j), \rho_{\mathfrak{n}}, (C_j)\rangle = \widetilde{\mu}(n)$

$$\widetilde{\mathcal{I}}_j := \{\{h \in H_j \mid \forall i \in M : L^i_{\kappa(h,j)}(h) = l_i, x_{|h|+1}(h) = x\} \mid (l_i) \in \mathbb{Q}^M, x \in \mathbb{N}\}$$

gilt. Dies entspricht der Definition der Informationspartitionen aus Gleichung (6.2).

Als Maschinenmodell verwenden wir das am Anfang dieses Kapitels beschriebene Modell, bei dem die Jobs klassenweise bearbeitet werden. Wir wiederholen der Vollständigkeit wegen hier nochmals die Definition des Anfang dieses Kapitels beschriebenen Maschinenmodells. Formal ist es durch

$$C_j(h) := \begin{cases} \mathrm{Pref}(h, j) + \mathrm{Suff}(h, j, l), & \text{falls } \exists l : 1 \leq l \leq |h_0| \text{ mit } p_l(h_0) = j \\ 0, & \text{sonst} \end{cases}$$

mit

$$\mathrm{Pref}(h,j) := \sum_{k=0}^{\kappa(h,j)-1} \max_{i\in M} \left(\sum_{p\in N_k^i(h)} \frac{x_p(h)}{s_i} \right) \qquad \text{und}$$

$$\mathrm{Suff}(h,j,l) := \sum_{p\in N^{h_l}_{\kappa(I,j)}(h), p\le l} \frac{x_p(h)}{s_{h_l}}$$

definiert.

6.3 Perfekte Gleichgewichte

Wir werden im Folgenden zeigen, dass in einem perfekten Gleichgewicht die Spieler eine Strategie auswählen, die der List Scheduling Algorithmus ausgewählt hätte, wenn man dem Algorithmus nur die Jobs der gleichen Klasse präsentiert hätte. Das bedeutet, ein Spieler wählt nur solche Maschinen aus, die die Last der Jobs der eigenen Klasse minimieren.

Nach der Definition der Informationsmengen $(\widetilde{\mathcal{I}}_j)$ wählt jeder Spieler eine Strategie nur abhängig von den vorangegangenen Jobs der eigenen Klasse aus. Das bedeutet insbesondere, dass ein Zug eines Spielers der Klasse k nicht die Strategie eines anderen Spielers der Klasse $\neq k$ beeinflusst. Daraus folgt, dass der Zeitpunkt, an dem die Maschinen beginnen die Jobs einer Klasse k zu bearbeiten, nicht von dem Zug eines Spielers der Klasse k abhängt. Damit erhalten wir direkt folgendes Lemma:

Lemma 6.1. *Seien $n \in \mathbb{N}$, $\widetilde{\Gamma} = \widetilde{\mu}(n)$, ein Spieler $j \in N$, eine Informationsmenge $I \in \widetilde{\mathcal{I}}_j$ und ein Profil ϕ von Γ beliebig gegeben. Dann gilt*

$$\forall i, i' \in M : \mathrm{E}_\phi[\mathrm{Pref}(\cdot, j)|Y_{I,i}] = \mathrm{E}_\phi[\mathrm{Pref}(\cdot, j)|Y_{I,i'}],$$

wobei $Y_{I,i}$ die Menge aller terminalen Historien ist, die einen Präfix in I haben und bei denen der Spieler $p_{|I|+1}(I)$, der in I seinen Zug durchführt, die Maschine i gewählt hat. Das heißt,

$$Y_{I,i} := \{h \in Z \mid h^{|I|} \in I, h_{|I|+1} = i\}.$$

Die Definition von $\mathrm{Suff}(h,j,l)$ entspricht gerade der Definition des Maschinenmodells aus Kapitel 5 eingeschränkt auf eine einzelne Klasse. Damit erhalten wir aus den Lemmata 5.1, 5.2 und 6.1 folgendes Lemma:

Lemma 6.2. *Für jedes $n \in \mathbb{N}$, Spiel $\widetilde{\Gamma} = \widetilde{\mu}(n)$ und perfekte Gleichgewicht $\phi \in \mathcal{G}(\widetilde{\Gamma})$ gilt, dass jeder Schedule der durch eine terminale Historie $h \in Z$, die bei Profil ϕ erreicht wird ($\Pr_\phi[h] > 0$), erzeugt wird, auch durch die Anwendung des List Scheduling Algorithmus auf die einzelnen Klassen erzeugt hätte werden können.*

Zudem besitzt das Spiel mindestens ein perfektes Gleichgewicht. Das bedeutet, $\mathcal{G}(\widetilde{\Gamma}) \neq \emptyset$.

Anders ausgedrückt, besagt das Lemma, dass ein Spieler sich in einem perfekten Gleichgewicht an die Strategie des List Scheduling Algorithmus eingeschränkt auf die eigene Klasse hält.

Lemma 6.2 erleichtert uns nun die Analyse der Spiele des Modells $\widetilde{\mu}$. Wir müssen nicht perfekte Gleichgewichte betrachten, sondern können stattdessen eine klassenweise Verteilung durch den List Scheduling Algorithmus analysieren.

6.4 Zielfunktion $f_{[\sum C_j]}$

Wir werden folgenden Satz in diesem Abschnitt beweisen.

Satz 6.2.

$$\mathrm{PA}(\widetilde{\mu}, f_{[\sum C_j]}) \leq 16\frac{s_{\max}^2}{s_{\min}^2} + 10\frac{s_{\max}}{s_{\min}} + 1,$$

wobei $s_{\max}$ die Geschwindigkeit der schnellsten und $s_{\min}$ die Geschwindigkeit der langsamsten Maschine ist.

Beweis. Seien im Folgenden $n \in \mathbb{N}$, Spiel $\widetilde{\Gamma} = \left\langle N, M, (s_i), (C_j), \rho_{\mathfrak{n}}, (\widetilde{\mathcal{I}}_j) \right\rangle = \widetilde{\mu}(n)$, Eingabe $a_0 \in \mathcal{E}(n)$ mit $\rho_{\mathfrak{n}}(a_0) > 0$ und ein perfektes Gleichgewicht ϕ von $\widetilde{\Gamma}$ beliebig, aber fest gewählt.

Weiterhin wählen wir ein $h \in Z$ mit $\Pr_{\rho_{\mathfrak{n}},\phi}[h] > 0$ und $h_0 = a_0$ beliebig, aber fest. Das heißt, h ist eine terminale Historie, die als Eingabe a_0 besitzt und, falls die Spieler das Profil ϕ spielen, mit Wahrscheinlichkeit größer Null erreicht wird.

Wir benötigen noch weitere Hilfsdefinitionen. Sei dazu ein Zeitpunkt $t \in \mathbb{R}$ gegeben. Hierbei ist $t = 0$ der Zeitpunkt, an dem die Maschinen mit der Bearbeitung der Jobs beginnen.

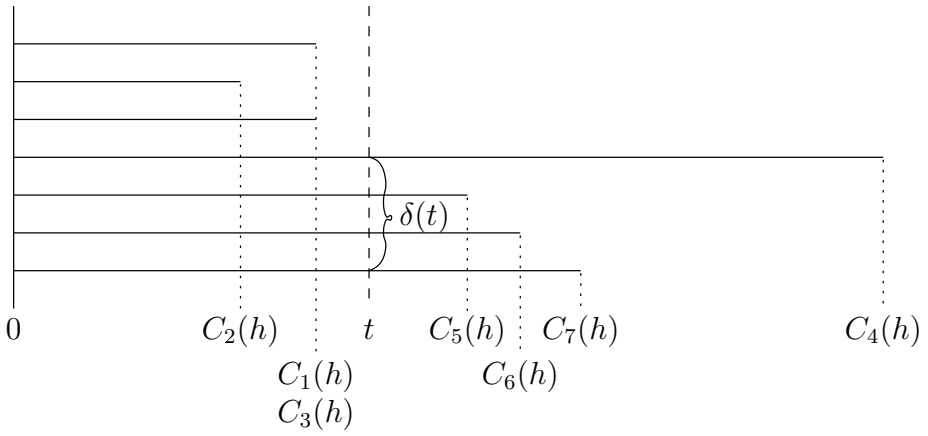

Abbildung 6.1: Die Funktion $\delta(t)$

- $\delta(t)$: $\delta(t)$ ist die Anzahl der aktiven Jobs zu einem Zeitpunkt t in dem Spiel Γ, falls das Spiel in der terminalen Historie h endet. Ein Job ist aktiv, falls er noch nicht bearbeitet wurde oder gerade bearbeitet wird. Das heißt, der Job des Spielers j ist in Zeitintervall $[0, C_j(h))$ aktiv. Die Abbildung 6.1 beschreibt diese Funktion.

  Entsprechend ist $\delta_k(t)$ die Anzahl der aktiven Jobs der Klasse $k \in \mathbb{N}_0$ zum Zeitpunkt t und $\delta_k^i(t)$ ist die Anzahl der aktiven Jobs der Klasse $k \in \mathbb{N}_0$ auf Maschine i zum Zeitpunkt t.

  Mit $\delta_k^{ii'}(t)$ bezeichnen wir die Differenz der Anzahl der aktiven Jobs der Klasse k auf den Maschinen i und i'. Das heißt, $\delta_k^{ii'}(t) := |\delta_k^i(t) - \delta_k^{i'}(t)|$.

- $R^i(t)$: $R^i(t)$ ist die Restbearbeitungszeit zum Zeitpunkt t auf der Maschine i. Dies ist die Zeit, die die Maschine i noch benötigt, um alle Jobs fertig zu stellen. Wir definieren analog zu $\delta(t)$ auch $R_k^i(t)$ und $R_k^{ii'}(t)$.

- Wir erweitern die Definition für $L_k^i(h)$, wie bei $\delta(t)$ und $R^i(t)$, um $L_k^{ii'}(h)$ für den Unterschied der Last auf den Maschinen i und i'. Das heißt,

$$L_k^{ii'}(h) := |L_k^i(h) - L_k^{i'}(h)|.$$

Betrachten wir nun die Definition von aktiven Jobs. Ein Job ist zum Zeitpunkt t aktiv, falls $t \in [0, C_j(h))$. Wir können nun die Antwortzeit umschreiben. Sei dazu $\Delta_j(t)$ die Funktion, die im Intervall $[0, C_j(h))$ Eins ist und

ansonsten Null. Dann gilt nach Definition

$$\int_{\mathbb{R}_{\geq 0}} \Delta_j(t)\mathrm{d}t = C_j(h).$$

Weiterhin gilt nach Definition von $\Delta_j(t)$ und $\delta(t)$, dass

$$\delta(t) = \sum_{j \in N} \Delta_j(t).$$

Wir können dies nun verwenden, um die Zielfunktion $f_{[\sum C_j]}$ umzuschreiben

$$f_{[\sum C_j]}(C(h)) = \sum_{j \in N} C_j(h) = \sum_{j \in N} \int_{\mathbb{R}_{\geq 0}} \Delta_j(t)\mathrm{d}t = \int_{\mathbb{R}_{\geq 0}} \delta(t)\mathrm{d}t.$$

Dies wird uns helfen den Preis der Anarchie für die Zielfunktion $f_{[\sum C_j]}$ abzuschätzen, da wir die Funktion $\delta(t)$ bzw. die Funktionen $\delta_k(t)$ abschätzen können.

Sei dazu $x_{\max}$ die Größe des größten Jobs in a_0. Sei weiterhin $k_{\max}$ die Klasse von $x_{\max}$. Nach Definition der Klasse heißt dies, dass $2^{k_{\max}} \leq x_{\max} < 2^{k_{\max}+1}$ und damit $k_{\max} \leq \log x_{\max} < k_{\max} + 1$.

Sei $\mathcal{T}_k$ das Zeitintervall, in dem die Jobs von Klasse k bearbeitet werden. Des Weiteren zerlegen wir das Intervall $\mathcal{T}_k$ in zwei Teile: $\widetilde{\mathcal{T}}_k$ und $\overline{\mathcal{T}}_k$. $\widetilde{\mathcal{T}}_k$ ist der Teil von $\mathcal{T}_k$, in dem alle Maschinen arbeiten. $\overline{\mathcal{T}}_k$ ist der Teil von $\mathcal{T}_k$, in dem mindestens eine Maschine keinen weiteren Job der Klasse k mehr bearbeiten muss. Für alle Klassen k, für die es in der Eingabe a_0 keinen Job gibt, definieren wir $\mathcal{T}_k := \emptyset$.

Da $\delta(t)$ die Anzahl der aktiven Jobs ist, gilt für alle $t \notin \mathcal{T} := \mathcal{T}_0 \cup \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_{k_{\max}}$, dass $\delta(t) = 0$. Das bedeutet, anstelle von ganz $\mathbb{R}_{\geq 0}$ reicht es aus, das Intervall $\mathcal{T}$ zu betrachten.

Nach dieser Definition gilt für alle $k \neq k'$: $\mathcal{T}_k \cap \mathcal{T}_{k'} = \emptyset$. Das heißt, wir haben eine Partition der Zeit, in der Jobs bearbeitet werden, durch $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_{k_{\max}}$ definiert.

Wir teilen nun das Integral auf die Klassenbestandteile auf und schätzen diese für jede Klasse einzeln ab.

$$\int_{\mathbb{R}_{\geq 0}} \delta(t)\mathrm{d}t = \sum_{k=0}^{k_{\max}} \int_{\mathcal{T}} \delta_k(t)\mathrm{d}t.$$

Sei nun $k \in \mathbb{N}_0$ fest gewählt. Wir zerlegen das Intervall $\mathcal{T}$ in

$$\mathcal{T}_{<k} = \bigcup_{k'<k} \mathcal{T}_{k'}, \quad \widetilde{\mathcal{T}}_k, \quad \overline{\mathcal{T}}_k \quad \text{und} \quad \mathcal{T}_{>k} = \bigcup_{k'>k} \mathcal{T}_{k'}.$$

Da im Intervall $\mathcal{T}_k$ alle Jobs der Klasse k abgearbeitet werden, gilt für alle $t \in \mathcal{T}_{>k}$, dass $\delta_k(t) = 0$.

Bevor wir mit dem Beweis von diesem Satz fortfahren, werden wir verschiedene Lemmata aufstellen und beweisen. Mit Hilfe dieser Lemmata erhalten wir obere Schranken für $\delta_k^i(t)$ für die drei Zeitabschnitte $\mathcal{T}_{<k}$, $\widetilde{\mathcal{T}}$ und $\overline{\mathcal{T}}$. Die folgende Liste gibt einen Überblick über die verwendeten Lemmata und deren Zusammenhang.

- Als erstes erhalten wir mit Lemma 6.3 eine obere Schranke für den Unterschied der Last $L_k^{ii'}(h)$, die von den Jobs der Klasse k auf den Maschinen i und i' verursacht wird. Daraus kann man direkt eine obere Schranke für $R_k^{ii'}(t)$ ableiten. Diese obere Schranke wird mehrfach für die eigentlichen Abschätzungen benötigt.

- Lemma 6.4 gibt uns eine obere Schrank für $\delta_k^i(t)$ für den Zeitabschnitt $\overline{\mathcal{T}}_k$.

- Das Lemma 6.5 wird im folgenden Verlauf für die weiteren Lemmata benötigt. Es gibt eine obere Schranke für die Leerlaufzeit der Maschinen an.

- Mit Hilfe von Lemma 6.5 erhalten wir in den Lemmata 6.6, 6.7 und 6.8 eine Abschätzung für $\Delta R_k^i(t)$ für die Zeitabschnitte $\mathcal{T}_{<k}$ und $\widetilde{\mathcal{T}}_k$. $\Delta R_k^i(t)$ ist der Unterschied zwischen der Restbearbeitungszeit in der terminalen Historie h und einer terminalen Historie h^{OPT}, die eine optimale Lösung bezüglich $f_{[\sum C_j]}$ darstellt.

- Aus der oberen Schranke von $\Delta R_k^i(t)$ leiten wir in Lemma 6.9 eine obere Schranke für $\delta_k^i(t)$ für die Zeitabschnitte $\mathcal{T}_{<k}$ und $\widetilde{\mathcal{T}}_k$ her.

- Wir werden des Weiteren wegen der von uns bewiesenen oberen Schranken von $\delta_k^i(t)$ eine obere Schranke für $\int_{\mathcal{T}_{<k} \cup \mathcal{T}_k} 1 \mathrm{d}t$ benötigen. Diese liefert uns Lemma 6.11.

- Nachdem wir diese Lemmata formal aufgestellt und bewiesen haben, werden wir den Beweis des Satzes 6.2 fortsetzen und die oberen Schranken für $\delta_k^i(t)$ aus den Lemmata 6.4 und 6.9 verwenden, um die Beschränkung des Preises der Anarchie zu beweisen.

Der Beweis von Satz 6.2 wird später fortgesetzt. □

Aus Lemma 6.2 können wir nun folgendes Lemma ableiten. Dies werden wir im Folgenden mehrfach benötigen. Es gibt uns eine obere Schranke für den Unterschied der klasseninternen Last zweier Maschinen.

Lemma 6.3. *Für alle Maschinen $i \neq i'$ gilt*

$$L_k^{ii'}(h) < \frac{2^{k+1}}{s_{\min}},$$

wobei $s_{\min}$ die Geschwindigkeit der langsamsten Maschine ist.

Beweis. Für den Beweis dieser Ungleichung betrachte man den letzten Job $j_k^i(h)$ aus Klasse k, der auf Maschine i in h gesetzt wurde. Sei $I_k^i(h)$ die dazugehörige Informationsmenge, in der dieser Job auf die Maschine gesetzt wird. Nach Definition folgt

$$L_k^i(h) = L_k^i(I_k^i(h)) + \frac{x_{|I_k^i(h)|+1}(I_k^i(h))}{s_i}. \tag{6.4}$$

Da nach Job $j_k^i(h)$ noch weitere Jobs der Klasse k auf Maschine i' verteilt werden können, gilt für die von uns zu Anfang beliebig, aber fest gewählte terminale Historie $h \in Z$

$$L_k^{i'}(h) \geq L_k^{i'}(I_k^i(h)). \tag{6.5}$$

Da ϕ ein perfektes Gleichgewicht ist, folgt aus Lemma 6.2 (jedes perfekte Gleichgewicht kann von dem List Scheduling Algorithmus erzeugt werden)

$$L_k^i(I_k^i(h)) + \frac{x_{|I_k^i(h)|+1}(I_k^i(h))}{s_i} \leq L_k^{i'}(I_k^i(h)) + \frac{x_{|I_k^i(h)|+1}(I_k^i(h))}{s_{i'}}.$$

Wir kombinieren nun diese Ungleichung mit der Gleichung (6.4), der Ungleichung (6.5) und der Tatsache, dass $x_{|I_k^i(h)|+1}(I) < 2^{k+1}$ ($j_k^i(h)$ ist aus der Klasse k) und $s_{\min} \leq s_{i'}$ gelten, und erhalten

$$\begin{aligned}
L_k^i(h) &= L_k^i(I_k^i(h)) + \frac{x_{|I_k^i(h)|+1}(I_k^i(h))}{s_i} \\
&\leq L_k^{i'}(I_k^i(h)) + \frac{x_{|I_k^i(h)|+1}(I_k^i(h))}{s_{i'}} \\
&\leq L_k^{i'}(h) + \frac{x_{|I_k^i(h)|+1}(I_k^i(h))}{s_{i'}} \\
&< L_k^{i'}(h) + \frac{2^{k+1}}{s_{\min}}.
\end{aligned}$$

Wir können das Gleiche mit vertauschten Rollen von i und i' wiederholen und erhalten dann

$$L_k^{i'}(h) < L_k^i(h) + \frac{2^{k+1}}{s_{\min}}.$$

Somit gilt

$$L_k^{ii'}(h) < \frac{2^{k+1}}{s_{\min}}.$$

■

Jede Maschine fängt gleichzeitig an, die Jobs der Klasse k zu bearbeiten. Das heißt, dass auf jedem Maschinenpaar die Restbearbeitungszeit sich maximal um den Unterschied in der Gesamtlast unterscheidet. Das heißt, eine Folgerung aus Lemma 6.3 ist

$$\forall t \in \mathbb{R}_{\geq 0} : R_k^{ii'}(t) < \frac{2^{k+1}}{s_{\min}}. \tag{6.6}$$

Wir beginnen nun, den Zeitraum $\overline{\mathcal{T}}_k$ zu analysieren und eine Abschätzung für $\delta_k(t)$ herzuleiten. $\overline{\mathcal{T}}_k$ ist der Zeitabschnitt in dem Jobs der Klasse k bearbeitet werden, aber mindestens eine Maschine alle ihre zugewiesenen Jobs dieser Klasse bereits fertiggestellt hat. Dazu betrachten wir jede Maschine getrennt voneinander und bestimmen eine Abschätzung für $\delta_k^i(t)$.

Lemma 6.4. *Für alle $t \in \overline{\mathcal{T}}_k$ gilt*

$$\delta_k^i(t) < 2\frac{s_{\max}}{s_{\min}} + 1,$$

wobei $s_{\max}$ die Geschwindigkeit der schnellsten und $s_{\min}$ die Geschwindigkeit der langsamsten Maschine ist.

Beweis. Da $t \in \overline{\mathcal{T}}_k$ gilt, existiert eine Maschine $i \in M$ mit $R_k^i(t) = 0$. Nach Lemma 6.3 und Gleichung (6.6) gilt aber für alle $i' \neq i$, dass

$$R_k^{ii'}(t) < \frac{2^{k+1}}{s_{\min}} \quad \text{und damit} \quad R_k^{i'}(t) < \frac{2^{k+1}}{s_{\min}}.$$

Da alle Jobs der Klasse k mindestens eine Größe 2^k haben und die schnellste Maschine die Geschwindigkeit $s_{\max}$ hat, braucht die Maschine i' mindestens $2^k/s_{\max}$ Zeiteinheiten für jeden noch zu bearbeitenden Job. Maschine i' hat zudem maximal einen Job, der zum Zeitpunkt t schon teilweise bearbeitet

und noch nicht fertiggestellt wurde. Daraus folgt, dass während der Restbearbeitungszeit $R_k^{i'}(t)$ maximal

$$\delta_k^{i'}(t) \leq \frac{R_k^{i'}(t)}{\frac{2^k}{s_{\max}}} + 1$$

Jobs auf der Maschine i' aus der Klasse k noch fertiggestellt werden können. Wir setzen für $R_k^{i'}(t)$ obige obere Schranke ein und erhalten

$$\delta_k^{i'}(t) < \frac{2^{k+1}}{2^k}\frac{s_{\max}}{s_{\min}} + 1 = 2\frac{s_{\max}}{s_{\min}} + 1.$$

■

Bevor wir die Zeitabschnitte $\widetilde{\mathcal{T}}_k$ und $\mathcal{T}_{<k}$ betrachten, führen wir folgendes Lemma ein, das uns eine obere Schranke der Leerlaufzeit LL_k^i der Maschine i in dem Zeitintervall $\mathcal{T}_k$ bestimmt. Dieses Lemma wird später mehrfach benötigt.

Lemma 6.5.

$$\mathrm{LL}_k^i < \frac{2^{k+1}}{s_{\min}}.$$

Beweis. Wir zeigen die obere Schranke, indem wir beweisen, dass dies für die Maschine $i \in M$ gilt, deren Leerlaufzeit am größten ist. Das heißt, das Maschine i nach der Definition von $\overline{\mathcal{T}}_k$ zum Zeitpunkt $t = \min \overline{\mathcal{T}}_k$ aufhört, Jobs der Klasse k zu bearbeiten. Sei i' die Maschine, die als letztes einen Job der Klasse k bearbeitet. Die Leerlaufzeit der Maschine i wird gerade durch diese Maschine i' bestimmt, da erst mit der Bearbeitung der Jobs der nächsten Klasse begonnen wird, wenn i' alle Jobs fertiggestellt hat. Nach Gleichung (6.6) gilt für alle $t' \in \mathbb{R}_{\geq 0}$

$$R_k^{ii'}(t') < \frac{2^{k+1}}{s_{\min}}.$$

Das heißt, $R_k^{ii'}(t) < 2^{k+1}/s_{\min}$. Da nach Definition $R_k^i(t) = 0$ gilt, folgt $R_k^{i'}(t) < 2^{k+1}/s_{\min}$ und $R_k^{i'}(t)$ ist gerade die Leerlaufzeit von Maschine i und es folgt $\mathrm{LL}_k^i < 2^{k+1}/s_{\min}$. Da alle anderen Maschinen $i'' \in M$ nicht vor Maschine i mit der Bearbeitung von Jobs der Klasse k fertig sind, gilt für alle Maschinen $i'' \in M$

$$\mathrm{LL}_k^{i''} < \frac{2^{k+1}}{s_{\min}}$$

und somit ist das Lemma bewiesen. ■

Bevor wir nun Lemma 6.5 anwenden, benötigen wir ein weiteres Lemma und eine weitere Hilfsdefinition. Betrachten wir das Intervall $\widetilde{\mathcal{T}}_k := \mathcal{T}_k \setminus \overline{\mathcal{T}}_k$. Das heißt, $\widetilde{\mathcal{T}}_k$ ist der Zeitraum, in dem alle Maschinen Jobs der Klasse k bearbeiten. Definieren wir dazu mit $\Delta R^i(t) = R^i(t) - R^{i,\mathrm{OPT}}(t)$ den Unterschied in der Restbearbeitungszeit auf Maschine i zwischen dem Schedule, der durch h repräsentiert wird, und dem Schedule einer terminalen Historie $h^{\mathrm{OPT}} \in Z$ mit $h_0^{\mathrm{OPT}} = a_0$, die $f_{[\sum C_j]}$ minimiert. Wir betrachten in Satz 6.2, den wir gerade beweisen wollen, den Preis der Anarchie, bei dem wir die perfekten Gleichgewichte nicht mit beliebigen Schedules vergleichen, sondern mit beliebigen Strategien. Das heißt, dass der daraus resultierende Schedule durch das gleiche Maschinenmodell eingeschränkt wird, durch das auch die Spieler eingeschränkt werden. So entsteht auch in h^{OPT} Leerlauf auf den Maschinen, da alle Maschinen warten, bis die anderen die Jobs einer Klasse fertig bearbeitet haben, bevor diese beginnen die Jobs der nächsten Klasse abzuarbeiten. Entsprechend sind $\Delta R_k^i(t)$ und $\Delta R_k^{ii'}(t)$ definiert. Wir werden nun eine obere Schranke für $\Delta R_k^i(t)$ bestimmen. Dazu zeigen wir in dem folgenden leicht zu beweisenden Lemma, dass wir uns auf den Zeitpunkt, an dem wir beginnen Jobs der Klasse k zu bearbeiten, konzentrieren können.

Lemma 6.6. *Angenommen $\widetilde{\mathcal{T}}_k \neq \emptyset$. Sei $t_k = \min \mathcal{T}_k$ der erste Zeitpunkt, zu dem ein Job aus der Klasse k bearbeitet wird. Dann gilt für alle $t \in \widetilde{\mathcal{T}}_k$ und $i \in M$*

$$\Delta R_k^i(t) \leq \Delta R_k^i(t_k).$$

Beweis. Sei $i \in M$ fest, aber beliebig, gewählt. Es gilt $\min \mathcal{T}_k = \min \widetilde{\mathcal{T}}_k$, da nach Voraussetzung am Anfang des Intervalls, in dem die Jobs der Klasse k bearbeitet werden, alle Maschinen arbeiten. Das heißt, dass $t_k \leq t$. Zum Zeitpunkt t_k gilt weiterhin $R_k^i(t_k) = L_k^i(h)$, da noch kein Job der Klasse k bearbeitet wurde. Dabei ist h die terminale Historie, die wir gerade betrachten und $L_k^i(h)$ entspricht der gesamten Last aller Jobs der Klasse k auf Maschine i. Während des ganzen Intervalls $\widetilde{\mathcal{T}}_k$ werden von allen Maschinen nur Jobs der Klasse k bearbeitet. Das heißt, dass ein optimaler Algorithmus auch keinen größeren Anteil von Jobs der Klasse k abarbeiten kann. Damit folgt die Behauptung. ■

Zur Bestimmung einer oberen Schranke für $\Delta R_k^i(t)$ mit $t \in \widetilde{\mathcal{T}}_k$ genügt es demnach, eine obere Schranke für $\Delta R_k^i(t_k)$ zu finden. Wir wenden nun Lemma 6.5 an, um eine solche obere Schranke für $\Delta R_k^i(t_k)$ herzuleiten.

Lemma 6.7. *Für alle $t \in \widetilde{\mathcal{T}}_k$ und $i \in M$ gilt*

$$\Delta R_k^i(t) < \frac{2^{k+2}}{s_{\min}}.$$

Beweis. Wegen Lemma 6.6 reicht es zu zeigen, dass

$$\Delta R_k^i(t_k) < \frac{2^{k+2}}{s_{\min}}.$$

In der optimalen Lösung h^{OPT} muss für jede Klasse k' mindestens die Zeit $|\widetilde{\mathcal{T}}_{k'}|$ aufgebracht werden, um alle Jobs aus dieser Klasse zu bearbeiten. Das liegt daran, dass in h zu jedem Zeitpunkt $t \in \widetilde{\mathcal{T}}_{k'}$ alle Maschinen einen Job der Klasse k' bearbeiten.

Das heißt, dass die optimale Lösung höchstens die Leerlaufzeiten der Maschinen einsparen kann, bis Jobs der Klasse k bearbeitet werden. Dies entspricht gerade

$$\sum_{k'=0}^{k-1} |\overline{\mathcal{T}}_{k'}| = \sum_{k'=0}^{k-1} \max_{i' \in M} \mathrm{LL}_{k'}^{i'}.$$

Zudem könnten die Maschinen die Jobs der Klasse k in der optimalen Lösung h^{OPT} schneller bearbeiten, als in der durch ein perfektes Gleichgewicht generierten Lösung h. Diese Zeitersparnis ist aus den gleichen Gründen durch die Leerlaufzeiten während der Bearbeitung der Jobs der Klasse k beschränkt. Das heißt, wir haben insgesamt eine Zeitersparnis für die Bearbeitung der Klasse k bei der optimalen Lösung h^{OPT} von

$$\begin{aligned} \Delta R_k^i(t_k) &\leq \sum_{k'=0}^{k-1} |\overline{\mathcal{T}}_{k'}| + |\overline{\mathcal{T}}_k| = \sum_{k'=0}^{k} |\overline{\mathcal{T}}_{k'}| = \sum_{k'=0}^{k} \max_{i' \in M} \mathrm{LL}_{k'}^{i'} \\ \text{(nach Lemma 6.5)} \quad &< \sum_{k'=0}^{k} \frac{2^{k'+1}}{s_{\min}} \leq \frac{2^{k+2}}{s_{\min}} \end{aligned}$$

und damit ist das Lemma bewiesen. ■

Wir betrachten nun $t \in \mathcal{T}_{<k} = \mathcal{T}_0 \cup \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_{k-1}$. $\Delta R_k^i(t)$ kann nicht größer als $\Delta R_k^i(t_k)$ sein, da ansonsten zu einem Zeitpunkt $t < t' < t_k$ in dem Schedule, der durch h gegeben ist, eine Maschine einen Job der Klasse k bearbeiten müsste. t_k ist allerdings der früheste Zeitpunkt, an dem ein solcher Job bearbeitet wird. Damit erhalten wir folgende Abwandlung des obigen Lemmas.

Lemma 6.8. *Für alle $t \in \mathcal{T}_{<k} \cup \widetilde{\mathcal{T}}_k$ und $i \in M$ gilt*

$$\Delta R_k^i(t) < \frac{2^{k+2}}{s_{\min}}.$$

Dies verwenden wir, um im Folgenden die Anzahl an aktiven Jobs zum Zeitpunkt $t \in \mathcal{T}_{<k} \cup \widetilde{\mathcal{T}}_k$ abzuschätzen. Mittels dieses und Lemma 6.4 können wir dann $f_{[\sum C_j]}(C(h))$ abschätzen.

Lemma 6.9. *Für alle $t \in \mathcal{T}_{<k} \cup \widetilde{\mathcal{T}}_k$ und $i \in M$ gilt*

$$\delta_k^i(t) < 2\frac{s_{\max}}{s_{\min}}\left(2 + \delta_k^{\mathrm{OPT},i}(t)\right) + 1,$$

wobei $\delta_k^{\mathrm{OPT},i}(t)$ die Anzahl der aktiven Jobs auf Maschine i bei dem von der optimalen Strategie h^{OPT} induzierten Schedule zum Zeitpunkt t ist.

Beweis. Jeder Job der Klasse k benötigt mindestens die Zeit $2^k/s_{\max}$, um bearbeitet zu werden, da 2^k die kleinste Jobgröße der Jobs in Klasse k und $s_{\max}$ die Geschwindigkeit der schnellsten Maschine ist. Da es zum Zeitpunkt t maximal einen Job gibt, der schon teilweise bearbeitet wurde, erhalten wir folgende Abschätzung für $\delta_k^i(t)$:

$$\delta_k^i(t) \leq \frac{s_{\max}}{2^k} R_k^i(t) + 1.$$

Sei $R_k^{\mathrm{OPT},i}(t)$ die Restbearbeitungszeit der Jobs der Klasse k auf Maschine i bei dem gewählten optimalen Schedule h^{OPT}. Wir setzen nun $R_k^i(t) = \Delta R_k^i(t) + R_k^{\mathrm{OPT},i}(t)$ in der Gleichung ein und erhalten

$$\delta_k^i(t) \leq \frac{s_{\max}}{2^k} R_k^i(t) + 1 \leq \frac{s_{\max}}{2^k}\left(\Delta R_k^i(t) + R_k^{\mathrm{OPT},i}(t)\right) + 1.$$

Wir wenden nun Lemma 6.8 an und erhalten

$$\delta_k^i(t) < \frac{s_{\max}}{s_{\min}}\frac{2^{k+2}}{2^k} + \frac{s_{\max}}{2^k} R_k^{\mathrm{OPT},i}(t) + 1 = 4\frac{s_{\max}}{s_{\min}} + \frac{s_{\max}}{2^k} R_k^{\mathrm{OPT},i}(t) + 1. \quad (6.7)$$

Betrachten wir nun $R_k^{\mathrm{OPT},i}(t)$. Da nach der Definition der Klassen jeder Job der Klasse k höchstens eine Größe 2^{k+1} hat, müssen in dem durch h^{OPT} implizit festgelegten Schedule auf Maschine i mindestens $R_k^{\mathrm{OPT},i}(t) \cdot s_i/2^{k+1}$ Jobs bearbeitet. Damit erhalten wir

$$\delta_k^{\mathrm{OPT},i}(t) \geq R_k^{\mathrm{OPT},i}(t)\frac{s_i}{2^{k+1}} \geq R_k^{\mathrm{OPT},i}(t)\frac{s_{\min}}{2^{k+1}}$$

$$\Leftrightarrow$$

$$R_k^{\mathrm{OPT},i}(t) \leq \delta_k^{\mathrm{OPT},i}(t)\frac{2^{k+1}}{s_{\min}}.$$

Wir setzen nun dies für $R_k^{\mathrm{OPT},i}(t)$ in Gleichung (6.7) ein und erhalten

$$\delta_k^i(t) < 4\frac{s_{\max}}{s_{\min}} + 2\frac{s_{\max}}{s_{\min}}\delta_k^{\mathrm{OPT},i}(t) + 1 = 2\frac{s_{\max}}{s_{\min}}\left(2 + \delta_k^{\mathrm{OPT},i}(t)\right) + 1.$$

■

Fortsetzung des Beweises von Satz 6.2. Wir verwenden nun die oberen Schranken aus den Lemmata, um für die drei verbliebenen Intervalle $\mathcal{T}_{<k}$, $\widetilde{\mathcal{T}}_k$ und $\overline{\mathcal{T}}_k$ das Integral abzuschätzen. Sei dazu $M_k \subseteq M$ die Menge der aktiven Maschinen für Klasse k. Das heißt, $i \in M_k$ genau dann, wenn es einen Job der Klasse k gibt, der auf Maschine i in h platziert wurde. Das heißt, dass $M_k = M$ ist, falls $\widetilde{\mathcal{T}}_k \neq \emptyset$.

Beginnen wir mit $\mathcal{T}_{<k}$ und $\widetilde{\mathcal{T}}_k$. Für beide erhalten wir die gleiche Abschätzung aus Lemma 6.9. Sei nun $\mathcal{T}' := \mathcal{T}_{<k} \cup \widetilde{\mathcal{T}}_k$.

$$\begin{aligned}
\int_{\mathcal{T}'} \delta_k(t)\mathrm{d}t &= \sum_{i \in M_k} \int_{\mathcal{T}'} \delta_k^i(t)\mathrm{d}t \\
\text{(nach Lemma 6.9)} \quad &< \sum_{i \in M_k} \int_{\mathcal{T}'} \left(2\frac{s_{\max}}{s_{\min}} \left(2 + \delta_k^{\mathrm{OPT},i}(t)\right) + 1 \right) \mathrm{d}t \\
&= \sum_{i \in M_k} \int_{\mathcal{T}'} \left(4\frac{s_{\max}}{s_{\min}} + 1 + 2\frac{s_{\max}}{s_{\min}} \delta_k^{\mathrm{OPT},i}(t) \right) \mathrm{d}t.
\end{aligned}$$

Wir teilen nun das Integral in zwei Teile auf. Einen Teil, der $\delta_k^{\mathrm{OPT},i}$ beinhaltet, und einen ohne $\delta_k^{\mathrm{OPT},i}$.

$$\begin{aligned}
\int_{\mathcal{T}'} \delta_k(t)\mathrm{d}t &= \sum_{i \in M_k} \int_{\mathcal{T}'} \left(4\frac{s_{\max}}{s_{\min}} + 1 + 2\frac{s_{\max}}{s_{\min}} \delta_k^{\mathrm{OPT},i}(t) \right) \mathrm{d}t \\
&= \sum_{i \in M_k} \int_{\mathcal{T}'} \left(1 + 4\frac{s_{\max}}{s_{\min}} \right) \mathrm{d}t + \sum_{i \in M_k} \int_{\mathcal{T}'} 2\frac{s_{\max}}{s_{\min}} \delta_k^{\mathrm{OPT},i}(t)\mathrm{d}t \\
&= \sum_{i \in M_k} \int_{\mathcal{T}'} \left(1 + 4\frac{s_{\max}}{s_{\min}} \right) \mathrm{d}t + \int_{\mathcal{T}'} \sum_{i \in M_k} 2\frac{s_{\max}}{s_{\min}} \delta_k^{\mathrm{OPT},i}(t)\mathrm{d}t \\
\text{(da } M_k \subseteq M\text{)} \quad &\leq \sum_{i \in M_k} \int_{\mathcal{T}'} \left(1 + 4\frac{s_{\max}}{s_{\min}} \right) \mathrm{d}t + \int_{\mathcal{T}'} \sum_{i \in M} 2\frac{s_{\max}}{s_{\min}} \delta_k^{\mathrm{OPT},i}(t)\mathrm{d}t \\
&= \sum_{i \in M_k} \int_{\mathcal{T}'} \left(1 + 4\frac{s_{\max}}{s_{\min}} \right) \mathrm{d}t + \int_{\mathcal{T}'} 2\frac{s_{\max}}{s_{\min}} \delta_k^{\mathrm{OPT}}(t)\mathrm{d}t. \qquad (6.8)
\end{aligned}$$

Für den Zeitabschnitt $\overline{\mathcal{T}}_k$ verwenden wir Lemma 6.4. Nach diesem Lemma gilt für alle $t \in \overline{\mathcal{T}}_k$

$$\delta_k^i(t) < 2\frac{s_{\max}}{s_{\min}} + 1.$$

Wir setzen dies nun in $\int_{\overline{\mathcal{T}}_k} \delta_k(t)\mathrm{d}t$ ein. Dazu teilen wir die Funktion $\delta_k(t)$ bezüglich der Maschinen auf. Hierbei müssen wir nur Maschinen betrachten,

die auch Jobs der Klasse k bearbeiten, denn für alle $i \notin M_k$ und $t \in \mathbb{R}_{\geq 0}$ gilt, dass $\delta_k^i(t) = 0$ und damit $\int_{\mathbb{R}_{\geq 0}} \delta_k^i(t)\mathrm{d}t = 0$. Somit erhalten wir für das Intervall $\overline{\mathcal{T}_k}$:

$$\begin{aligned}\int_{\overline{\mathcal{T}}_k} \delta_k(t)\mathrm{d}t &= \sum_{i \in M_k} \int_{\overline{\mathcal{T}}_k} \delta_k^i(t)\mathrm{d}t \\ \text{(nach Lemma 6.4)} \quad &< \sum_{i \in M_k} \int_{\overline{\mathcal{T}}_k} \left(2\frac{s_{\max}}{s_{\min}} + 1\right) \mathrm{d}t \\ &\leq \sum_{i \in M_k} \int_{\overline{\mathcal{T}}_k} \left(4\frac{s_{\max}}{s_{\min}} + 1\right) \mathrm{d}t. \end{aligned} \tag{6.9}$$

Das heißt, wenn wir die Gleichungen (6.8) und (6.9) einsetzen, erhalten wir insgesamt für die durchschnittliche Antwortzeit bezüglich der Klasse $k \in \mathbb{N}_0$

$$\begin{aligned} f_{[\sum C_j]}(C(h)) &= \int_{\mathcal{T}} \delta_k(t)\mathrm{d}t \\ &= \int_{\mathcal{T}_{<k}} \delta_k(t)\mathrm{d}t + \int_{\widetilde{\mathcal{T}}_k} \delta_k(t)\mathrm{d}t + \int_{\overline{\mathcal{T}}_k} \delta_k(t)\mathrm{d}t \\ &< \sum_{i \in M_k} \int_{\mathcal{T}_{<k} \cup \mathcal{T}_k} \left(4\frac{s_{\max}}{s_{\min}} + 1\right) \mathrm{d}t + \int_{\mathcal{T}_{<k} \cup \widetilde{\mathcal{T}}_k} 2\frac{s_{\max}}{s_{\min}} \delta_k^{\mathrm{OPT}}(t)\mathrm{d}t. \end{aligned} \tag{6.10}$$

Es fehlt nur noch, den ersten Teil

$$\sum_{i \in M_k} \int_{\mathcal{T}_{<k} \cup \mathcal{T}_k} \left(4\frac{s_{\max}}{s_{\min}} + 1\right) \mathrm{d}t$$

abzuschätzen, da wir für den zweiten Teil direkt aus der Definition von $\delta_k^{\mathrm{OPT}}(t)$

$$\int_{\mathcal{T}_{<k} \cup \widetilde{\mathcal{T}}_k} 2\frac{s_{\max}}{s_{\min}} \delta_k^{\mathrm{OPT}}(t)\mathrm{d}t = 2\frac{s_{\max}}{s_{\min}} \left[f_{[\sum C_j]}\right]_k^{\mathrm{OPT}}$$

erhalten, wobei $\left[f_{[\sum C_j]}\right]_k^{\mathrm{OPT}}$ die Summe der Antwortzeiten aller Jobs der Klasse k in der optimalen Lösung h^{OPT} ist. Für die Abschätzung des ersten Teils verwenden wir Lemma 6.11. Es liefert eine obere Schranke für $\int_{\mathcal{T}_{<k} \cup \mathcal{T}_k} 1\mathrm{d}t$. Wir werden nun zuerst dieses Lemma formulieren und dann beweisen, bevor wir mit dem Beweis von Satz 6.2 fortfahren. □

Für die formale Definition von Lemma 6.11 benötigen wir weitere Hilfsdefinitionen und das Lemma 6.10.

Sei $j \in N_k(h)$ ein Spieler mit einem Job der Klasse k, dessen Bearbeitung in der optimalen Lösung h^{OPT} zum Zeitpunkt $t_j := C_j(h^{\mathrm{OPT}})$ endet. Zudem habe der Job die Eigenschaft, dass zum Zeitpunkt t_j mindestens eine Maschine in der optimalen Lösung keinen Job der Klasse k bearbeitet oder gerade aufgehört hat, Jobs dieser Klasse zu bearbeiten. Das heißt, falls in dem durch h^{OPT} definierten Schedule für die Bearbeitung der Jobs der Klasse k die Maschinen M'_k verwendet werden, so gibt es mindestens $|M'_k|$ Jobs mit dieser Eigenschaft. Denn der letzte Job j jeder Maschine $i \in M'_k$ besitzt gerade diese Eigenschaft, dass es mindestens eine Maschine gibt, die zum Zeitpunkt t_j keinen weiteren Job der Klasse k bearbeitet. Sei nun $\widetilde{N}_k$ die Menge aller Jobs $j \in N_k(h^{\mathrm{OPT}})$ mit dieser Eigenschaft.

Wir zeigen nun folgendes Lemma, das uns garantiert, dass es mindestens $|M_k|$ Jobs mit dieser Eigenschaft gibt.

Lemma 6.10. $|\widetilde{N}_k| \geq |M_k|$.

Beweis. Nehmen wir an, es gelte $|\widetilde{N}_k| < |M_k|$. Sei, wie oben, M'_k die Menge aller Maschinen, die in der optimalen Lösung h^{OPT} zur Bearbeitung von Jobs der Klasse k verwendet werden. Wie wir oben erläutert haben, gibt es mindestens $|M'_k|$ Jobs mit dieser Eigenschaft. Das heißt, es muss $|M'_k| \leq |\widetilde{N}_k| < |M_k|$ gelten. Das bedeutet aber, dass $|M'_k| < |M|$ und damit ist immer eine Maschine unbenutzt und somit gilt $\widetilde{N}_k = N_k(h^{\mathrm{OPT}}) = N_k(h)$. Da aber jeder Maschine aus M_k mindestens ein Job aus $N_k(h)$ in h zugewiesen ist, folgt $|M_k| \leq |N_k(h)|$. Damit haben wir

$$|N_k(h)| = |\widetilde{N}_k| < |M_k| \leq |N_k(h)|$$

und das ist ein Widerspruch und zeigt das Lemma. ■

Mit Hilfe von Lemma 6.10 zeigen wir nun folgendes Lemma, dass uns eine Abschätzung für $\int_{\mathcal{T}_{<k} \cup \mathcal{T}_k} 1 \mathrm{d}t$ liefert.

Lemma 6.11. *Für alle Jobs $j \in \widetilde{N}_k$ gilt*

$$\int_{\mathcal{T}_{<k} \cup \mathcal{T}_k} 1 \mathrm{d}t < \left(4\frac{s_{\max}}{s_{\min}} + 1\right) C_j(h^{\mathrm{OPT}}).$$

Beweis. Betrachten wir nun einen Job $j \in \widetilde{N}_k$. Nach der Definition von $\widetilde{N}_k$ gibt es zum Zeitpunkt $t_j = C_j(h^{\mathrm{OPT}})$ mindestens eine Maschine, die keine weiteren Jobs bearbeitet. Wie wir in dem Beweis von Lemma 6.7 gesehen

haben, werden die Maschinen in dem durch die optimale Strategie h^{OPT} induzierten Schedule höchstens um die Leerlaufzeiten der Maschinen in dem durch das perfekte Gleichgewicht induzierten Schedule früher fertig. Sei j nun der letzte Job auf einer der Maschinen. Nach der Definition von $\widetilde{N}_k$ gehört jeder Job, der in h^{OPT} als letzter auf einer Maschine bearbeitet wird, zu $\widetilde{n}_k$. Damit können wir $C_j(h^{\mathrm{OPT}})$ folgendermaßen abschätzen:

$$C_j(h^{\mathrm{OPT}}) \geq \sum_{k'=0}^{k} |\widetilde{\mathcal{T}}_{k'}|.$$

Diese Summe ist nach der Definition von $\widetilde{\mathcal{T}}_{k'}$ und den Leerlaufzeiten LL_k^i identisch zu

$$\sum_{k'=0}^{k} |\mathcal{T}_{k'} \setminus \overline{\mathcal{T}}_{k'}| = \sum_{k'=0}^{k} |\mathcal{T}_{k'}| - \sum_{k'=0}^{k} |\overline{\mathcal{T}}_{k'}| = \int_{\mathcal{T}_{<k} \cup \mathcal{T}_k} 1\mathrm{d}t - \sum_{k'=0}^{k} \max_{i \in M} \mathrm{LL}_k^i.$$

Dies setzen wir nun für die Summe ein und lösen nach $\int_{\mathcal{T}_{<k} \cup \mathcal{T}_k} 1\mathrm{d}t$ auf:

$$C_j(h^{\mathrm{OPT}}) \geq \int_{\mathcal{T}_{<k} \cup \mathcal{T}_k} 1\mathrm{d}t - \sum_{k'=0}^{k} \max_{i \in M} \mathrm{LL}_k^i$$

$$\Leftrightarrow \qquad \int_{\mathcal{T}_{<k} \cup \mathcal{T}_k} 1\mathrm{d}t \leq C_j(h^{\mathrm{OPT}}) + \sum_{k'=0}^{k} \max_{i \in M} \mathrm{LL}_k^i. \tag{6.11}$$

Nach Lemma 6.5 ist die Leerlaufzeit $\mathrm{LL}_{k'}^i$ der Maschine $i \in M$ in dem Zeitintervall $\mathcal{T}_{k'}$ durch

$$\mathrm{LL}_{k'}^i < \frac{2^{k'+1}}{s_{\min}}$$

beschränkt. Das setzen wir in die Ungleichung (6.11) ein und erhalten:

$$\begin{aligned}
\int_{\mathcal{T}_{<k} \cup \mathcal{T}_k} 1\mathrm{d}t &\leq C_j(h^{\mathrm{OPT}}) + \sum_{k'=0}^{k} \max_{i \in M} \mathrm{LL}_{k'}^i \\
&< C_j(h^{\mathrm{OPT}}) + \sum_{k'=0}^{k} \frac{2^{k'+1}}{s_{min}} \\
&\leq C_j(h^{\mathrm{OPT}}) + \frac{2^{k+2}}{s_{\min}}.
\end{aligned}$$

Da ein Job $j \in \widetilde{N}_k$ mindestens die Größe 2^k hat und auf der schnellsten Maschine mindestens $2^k/s_{\max}$ Zeit benötigt, folgt

$$1 \leq C_j(h^{\mathrm{OPT}}) \cdot \frac{s_{\max}}{2^k}.$$

Das setzen wir oben ein und erhalten

$$\begin{aligned}\int_{\mathcal{T}_{<k}\cup\mathcal{T}_k} 1\mathrm{d}t &< C_j(h^{\mathrm{OPT}}) + \frac{2^{k+2}}{s_{\min}} \\ &\leq C_j(h^{\mathrm{OPT}}) + 4\frac{s_{\max}}{s_{\min}} C_j(h^{\mathrm{OPT}}) \\ &= \left(4\frac{s_{\max}}{s_{\min}} + 1\right) C_j(h^{\mathrm{OPT}}).\end{aligned}$$

■

Fortsetzung des Beweises von Satz 6.2. Wir verwenden nun Lemmata 6.10 und 6.11, um aus der Ungleichung (6.10) den Beweis für diesen Satz zu erhalten.

Sei dazu nun $l := |M_k|$ die Anzahl der aktiven Maschinen und $N'_k(h) := \{j_1, j_2, \ldots, j_l\} \subseteq \widetilde{N}_k$ eine Menge von l Jobs mit der gewünschten Eigenschaft, so dass wir Lemma 6.11 anwenden können. Nach Lemma 6.10 gilt $|M_k| \leq |\widetilde{N}_k|$, so dass es mindestens l solche Jobs gibt. Da alle Jobs aus $N'_k(h)$ aus der Klasse k sind, gilt $N'_k(h) \subseteq N_k(h)$. Nun können wir den ersten Teil unserer oberen Schranke aus Gleichung (6.10), wie folgt, abschätzen:

$$\begin{aligned}\sum_{i\in M_k} \int_{\mathcal{T}_{<k}\cup\mathcal{T}_k} \left(4\frac{s_{\max}}{s_{\min}} + 1\right) \mathrm{d}t &\leq \sum_{j\in N'_k(h)} \left(4\frac{s_{\max}}{s_{\min}} + 1\right) \int_{\mathcal{T}_{<k}\cup\mathcal{T}_k} 1\mathrm{d}t \\ &< \left(4\frac{s_{\max}}{s_{\min}} + 1\right) \sum_{j\in N'_k(h)} \left(4\frac{s_{\max}}{s_{\min}} + 1\right) C_j(h^{\mathrm{OPT}}) \\ (\text{da } N'_k(h) \subseteq N_k(h)) \quad &\leq \left(4\frac{s_{\max}}{s_{\min}} + 1\right) \sum_{j\in N_k(h)} \left(4\frac{s_{\max}}{s_{\min}} + 1\right) C_j(h^{\mathrm{OPT}}) \\ (\text{nach Def. von } N_k(h)) \quad &= \left(4\frac{s_{\max}}{s_{\min}} + 1\right) \left(4\frac{s_{\max}}{s_{\min}} + 1\right) \left[f_{[\sum C_j]}\right]_k^{\mathrm{OPT}}.\end{aligned}$$

Setzen wir dies in Gleichung (6.10) ein, erhalten wir für alle Jobs der Klasse k die Abschätzung

$$\begin{aligned}\int_{\mathcal{T}} \delta_k(t)\mathrm{d}t &< \left(4\frac{s_{\max}}{s_{\min}} + 1\right) \left(4\frac{s_{\max}}{s_{\min}} + 1\right) \left[f_{[\sum C_j]}\right]_k^{\mathrm{OPT}} + 2\frac{s_{\max}}{s_{\min}} \left[f_{[\sum C_j]}\right]_k^{\mathrm{OPT}} \\ &= \left(16\frac{s_{\max}^2}{s_{\min}^2} + 10\frac{s_{\max}}{s_{\min}} + 1\right) \left[f_{[\sum C_j]}\right]_k^{\mathrm{OPT}}\end{aligned}$$

und damit insgesamt

$$\begin{aligned}\int_{\mathbb{R}_{\geq 0}} \delta(t)\mathrm{d}t &= \sum_{k=0}^{k_{\max}} \int_{\mathcal{T}} \delta_k(t)\mathrm{d}t \\ &< \sum_{k=0}^{k_{\max}} \left(16\frac{s_{\max}^2}{s_{\min}^2} + 10\frac{s_{\max}}{s_{\min}} + 1\right) \left[f_{[\sum C_j]}\right]_k^{\mathrm{OPT}} \\ &= \left(16\frac{s_{\max}^2}{s_{\min}^2} + 10\frac{s_{\max}}{s_{\min}} + 1\right) \left[f_{[\sum C_j]}\right]^{\mathrm{OPT}}.\end{aligned}$$

Hiermit ist nun Satz 6.2 bewiesen. ■

Für identische Maschinen erhalten wir aus Satz 6.2 folgendes Korollar.

Korollar 6.1. *Sei $\widetilde{\mu}$ ein Modell mit den Antwortzeitfunktionen und Informationsmengen wie gehabt. Seien die Geschwindigkeiten der Maschinen gleich, das heißt, für alle $i \in M$ ist $s_i = 1$. Dann gilt*

$$\mathrm{PA}(\widetilde{\mu}, f_{[\sum C_j]}) \leq 27.$$

Betrachten wir für die Analyse des Koordinationsverhältnisses die Analyse von dem Algorithmus IMD aus der Arbeit von Avrahami und Azar [AA03]. Der Algorithmus IMD ist ein Algorithmus für präemptives Scheduling von Jobs mit beliebigen Einlastzeiten und identischen Maschinen. Setzt man die Einlastzeiten für alle Jobs auf Null, so verteilt der IMD Algorithmus die Jobs auf die Maschinen, wie es die Spieler in unserem Modell in einem perfekten Gleichgewicht tun würden. Zudem werden in diesem Fall die Jobs nicht unterbrochen, wodurch wir den Aspekt, dass es sich um einen Algorithmus für präemptives Scheduling handelt, ignorieren können.

Avrahami und Azar zeigen in der Arbeit folgenden Satz:

Satz 6.3 ([AA03])**.** *Das kompetitive Verhältnis von dem Algorithmus IMD bezüglich der Zielfunktion $f_{[\sum F_j]}$ beträgt $O(\log n)$.*

Betrachtet man den Beweis von diesem Satz, wird nicht der Algorithmus IMD analysiert, sondern eine Variante, bei der die Jobs in genau der gleichen Reihenfolge abgearbeitet werden, wie dies unser Maschinenmodell durchführt, sofern alle Einlastzeiten Null sind. Der einzige Unterschied zwischen dieser Variante IMD' und dem von einem perfekten Gleichgewicht und dem Maschinenmodell induzierten Schedule besteht in den Leerlaufzeiten auf den Maschinen.

Da in unserem Fall (alle Einlastzeiten sind Null) für jeden Job j die Flusszeit identisch zu der Antwortzeit ist, können wir aus Satz 6.3 eine Analyse für das Koordinationsverhältnis ableiten, sofern wir den Verlust durch die Leerlaufzeiten beschränken können.

Nach Lemma 6.5 haben wir für jede Klasse k auf einer Maschine $i \in M$ höchstens einen Leerlauf LL_k^i von

$$\mathrm{LL}_k^i < 2^{k+1}.$$

Das heißt, jeder Job der Klasse k wird maximal um

$$\sum_{k'=0}^{k-1} \max_{i \in M} \mathrm{LL}_{k'}^i < \sum_{k'=0}^{k-1} 2^{k'+1} < 2^{k+1}$$

verzögert. Da ein Job $j \in N$ der Klasse k mindestens die Größe 2^k hat, verschlechtert sich die Antwortzeit höchstens um den Faktor $2^{k+1}/2^k = 2$. Da dies für jeden Job gilt, folgt somit aus Satz 6.3 folgendes Lemma.

Satz 6.4. *Für ein Modell $\widetilde{\mu}$, wie oben definiert, mit identischen Maschinen – $\forall i \in M : s_i = 1$ – gilt*

$$\mathrm{CR}(\widetilde{\mu}, f_{[\sum C_j]}) \in O\left(\log n\right).$$

Im Gegensatz zu dem einfachen Modell, das den List Scheduling Algorithmus nachbildet, hat dieses Modell den Vorteil, dass im Fall von der Zielfunktion $f_{[\sum C_j]}$ der Preis der Anarchie unabhängig von der Anzahl der Jobs oder der Größe der Jobs ist und nur von dem relativen Verhältnis der Geschwindigkeit der Maschinen abhängt. Bei dem den List Scheduling Algorithmus nachbildenden Modell aus Kapitel 5 konnten wir zudem die untere Schranke $\Omega(n)$ für das Koordinationsverhältnis bei identischen Maschinen angeben. Für das in diesem Kapitel beschriebene Modell haben wir eine obere Schranke von $O(\log n)$ für das Koordinationsverhältnis bei identischen Maschinen. Diese Verbesserung haben wir durch die Klasseneinteilung erhalten. Die Klasseneinteilung so zu wählen, dass die Größen der Jobs innerhalb einer Klasse von Klasse zu Klasse sich exponentiell steigern, sorgt dafür, dass wir hier keinen linearen Faktor bezüglich der Anzahl der Jobs erhalten, sondern nur einen logarithmischen.

6.5 Zielfunktion $f_{[C_{\max}]}$

Für den Preis der Anarchie müssen wir nur beachten, dass ein Spieler nach Lemma 6.2 in einem perfekten Gleichgewicht nur solche Maschinen auswählen würde, die auch der List Scheduling Algorithmus auswählen könnte,

falls man ihm nur die Jobs der gleichen Klasse k präsentieren würde. Daraus folgt, dass der Schedule jeder terminalen Historie, die mit positiver Wahrscheinlichkeit erreicht wird, auch durch die Ausführung des List Scheduling Algorithmus für jede Klasse einzeln entstanden sein könnte.

Die maximale Antwortzeit wird minimiert, falls die Zeit, die die Maschinen für die Bearbeitung einer einzelnen Klasse benötigen, minimiert wird. Dies folgt direkt aus der Definition des Maschinenmodells, da die Maschinen erst mit der Bearbeitung der nächsten Klasse beginnen, falls alle Jobs der aktuellen Klasse auf allen Maschinen fertig abgearbeitet wurden. Wenn wir die Jobs einer einzelnen Klasse betrachten, so werden diese nach dem List Scheduling Algorithmus verteilt. $T_k(a_0, \xi)$ ist die erwartete Zeit, die bei der Strategie ξ für die Bearbeitung der Jobs der Klasse k benötigt wird. Das heißt, in einem perfekten Gleichgewicht ϕ benötigen die Maschinen nach Graham [Gra66] für eine feste Eingabe $a_0 \in \mathcal{E}(n)$ maximal

$$T_k(a_0, \phi) \leq \left(1 + \frac{m-1}{m} \cdot \frac{s_{\max}}{s_{\min}}\right) T_k(a_0, \psi)$$

mehr Zeit gegenüber einer optimalen Strategie ψ bezüglich $f_{[C_{\max}]}$, um die Jobs einer Klasse k zu bearbeiten. Dies erhalten wir direkt aus dem List Scheduling Algorithmus, da die Spieler sich innerhalb einer Klasse wie der List Scheduling Algorithmus angewendet auf die Jobs dieser Klasse verhalten.

Daraus folgt, dass der erwartete Wert der Zielfunktion $f_{[C_{\max}]}$ für die Eingabe a_0 und das perfekte Gleichgewicht ϕ folgendermaßen abgeschätzt werden kann:

$$\begin{aligned}
\mathrm{E} f_{[C_{\max}]}(\widetilde{\Gamma}, a_0, \phi) &= \sum_{k=0}^{k_{\max}(a_0)} T_k(a_0, \phi) \\
&\leq \sum_{k=0}^{k_{\max}(a_0)} \left(1 + \frac{m-1}{m} \cdot \frac{s_{\max}}{s_{\min}}\right) T_k(a_0, \psi) \\
&= \left(1 + \frac{m-1}{m} \cdot \frac{s_{\max}}{s_{\min}}\right) \sum_{k=0}^{k_{\max}(a_0)} T_k(a_0, \psi) \\
&= \left(1 + \frac{m-1}{m} \cdot \frac{s_{\max}}{s_{\min}}\right) \mathrm{E} f_{[C_{\max}]}(\widetilde{\Gamma}, a_0, \psi).
\end{aligned}$$

Hierbei bezeichnet $k_{\max}(a_0)$ die größte Klasse der Jobs, die in Eingabe a_0 vorkommen.

Damit erhalten wir direkt folgenden Satz:

Satz 6.5. *Sei $\widetilde{\mu}$ ein Modell, wie gehabt, dann gilt*

- *für beliebige Maschinen*

$$\text{PA}(\widetilde{\mu}, f_{[C_{\max}]}) \leq 1 + \frac{m-1}{m} \cdot \frac{s_{\max}}{s_{\min}} \leq \frac{s_{\max}}{s_{\min}} \left(2 - \frac{1}{m}\right)$$

- *und für identische Maschinen*

$$\text{PA}(\widetilde{\mu}, f_{[C_{\max}]}) \leq 2 - \frac{1}{m}.$$

Im Fall des Koordinationsverhältnisses erhalten wir eine andere Abschätzung, da die Maschinen bei einem optimalen Schedule, der sich nicht an das Maschinenmodell hält, vollständig ausgelastet werden können und nicht pausieren, bis alle anderen Maschinen die Jobs der gleichen Klasse bearbeitet haben.

Seien nun $n \in \mathbb{N}$, $\widetilde{\Gamma} = \widetilde{\mu}(n)$, ein perfektes Gleichgewicht ϕ von $\widetilde{\Gamma}$, $a_0 \in \mathcal{E}(n)$ und $h \in Z$ mit $h_0 = a_0$ und $\Pr_{\rho_{\mathfrak{n}},\phi}[h] > 0$ beliebig, aber fest gewählt. Sei $k_{\max}$ wieder die Klasse des größten Jobs in a_0.

Da für diese Zielfunktion die Abarbeitungsreihenfolge auf den Maschinen nicht von Bedeutung ist, können wir davon ausgehen, dass die Maschinen die Jobs der Größe nach abarbeiten. Aber auch im optimalen Schedule müssen alle Maschinen mindestens $|\widetilde{\mathcal{T}}_0 \cup \widetilde{\mathcal{T}}_1 \cup \cdots \cup \widetilde{\mathcal{T}}_{k_{\max}}|$ Zeit an allen Jobs arbeiten. Das heißt, dass in dem Schedule, der durch h definiert wird, höchstens die Leerlaufzeiten mehr an Arbeit verrichtet werden. Sei nun $x_j(h)$ die Größe des Jobs, der zuletzt in h fertiggestellt wird. Nach der Definition des Maschinenmodells ist dieser Job aus der größten Klasse $k_{\max}(a_0)$. Dieser Job benötigt in h höchstens um die Leerlaufzeiten der Maschinen mehr Zeit als der letzte Job in einem optimalen Schedule, um fertig zu werden. Der optimale Algorithmus benötigt mindestens $x_j(h)/s_{\max}$ Zeit, um diesen Job zu bearbeiten. Sei S^{OPT} der optimale Schedule und $f_{[C_{\max}]}(S^{\text{OPT}})$ die größte Antwortzeit in S^{OPT}. Entsprechend sei $f_{[C_{\max}]}(h)$ die größte Antwortzeit bei dem Schedule, der durch h impliziert wird. Damit erhalten wir folgende Ungleichungen

$$f_{[C_{\max}]}(S^{\text{OPT}}) \geq \frac{x_j(h)}{s_{\max}} \geq \frac{2^{k_{\max}(a_0)}}{s_{\max}}$$

und

$$\begin{aligned} f_{[C_{\max}]}(h) &\leq f_{[C_{\max}]}(S^{\text{OPT}}) + \sum_{k=0}^{k_{\max}(a_0)} \max_{i \in M} \text{LL}_k^i \\ \text{(nach Lemma 6.5)} \quad &< f_{[C_{\max}]}(S^{\text{OPT}}) + \sum_{k=0}^{k_{\max}(a_0)} \frac{2^{k+1}}{s_{\min}}, \end{aligned}$$

wobei LL_k^i die Leerlaufzeit der Maschine i bei der Bearbeitung der Jobs der Klasse k ist.

Wir setzen nun beide Gleichung ineinander ein, um das Koordinationsverhältnis auszurechnen.

$$\begin{aligned}
f_{[C_{\max}]}(h) &< f_{[C_{\max}]}(S^{\mathrm{OPT}}) + \sum_{k=0}^{k_{\max}(a_0)} \frac{2^{k+1}}{s_{\min}} \\
&\leq f_{[C_{\max}]}(S^{\mathrm{OPT}}) + \frac{2^{k_{\max}(a_0)+2}}{s_{\min}} \\
&= f_{[C_{\max}]}(S^{\mathrm{OPT}}) \left(1 + \frac{2^{k_{\max}(a_0)+2}}{s_{\min}} \frac{1}{f_{[C_{\max}]}(S^{\mathrm{OPT}})}\right) \\
&\leq f_{[C_{\max}]}(S^{\mathrm{OPT}}) \left(1 + \frac{2^{k_{\max}(a_0)+2}}{s_{\min}} \frac{s_{\max}}{2^{k_{\max}(a_0)}}\right) \\
&= f_{[C_{\max}]}(S^{\mathrm{OPT}}) \left(1 + 4\frac{s_{\max}}{s_{\min}}\right).
\end{aligned}$$

Damit haben wir folgenden Satz bewiesen.

Satz 6.6. *Sei $\widetilde{\mu}$ ein Modell wie oben, dann gilt*

- *für beliebige Maschinen*

$$\mathrm{CR}(\widetilde{\mu}, f_{[C_{\max}]}) \leq 1 + 4\frac{s_{\max}}{s_{\min}}$$

- *und für identische Maschinen*

$$\mathrm{CR}(\widetilde{\mu}, f_{[C_{\max}]}) \leq 5.$$

Wie wir hier sehen, ist durch die Wahl der Klassen noch immer sichergestellt, dass das Koordinationsverhältnis im Fall von identischen Maschinen durch eine Konstante beschränkt ist.

Im ersten Moment mag es verwundern, dass in die Abschätzung nicht die Anzahl der Maschinen eingeht, wenn man an eine Eingabe denkt, bei der keine zwei Jobs zur gleichen Klasse gehören. In diesem Fall werden alle Jobs von der schnellsten Maschine der Reihe nach bearbeitet. Allerdings vergrößern sich durch die Einteilung der Klassen die Jobs von Klasse zu Klasse um den Faktor Zwei. Dadurch ist die Verzögerung eines Jobs der Klasse k in einem solchen Fall durch 2^{k+1} beschränkt. Da der Job der Klasse k mindestens die Größe 2^k hat, brauchen die Maschinen höchstens den Faktor Zwei mehr an Zeit, um diesen Job zu bearbeiten.

6.6 Informationen der Spieler

Wir werden nun Satz 6.1 beweisen. Betrachten wir dazu erneut die Modelle μ und $\widetilde{\mu}$, die sich nur durch die Informationspartitionen unterscheiden. Zur Erinnerung, μ ist das online Scheduling-Spiel mit den dazugehörigen Informationspartitionen

$$\mathcal{I}_j := \{\{h \in H \mid \mathfrak{A}(h) = \mathfrak{a}, \mathfrak{X}(h) = \mathfrak{x}\} \mid \mathfrak{a} \in M^{|h|}, \mathfrak{x} \in \mathbb{N}^{|h|+1}\}$$

und $\widetilde{\mu}$ ist das modifizierte Spiel, bei denen die Spieler nur noch die Last, die von den schon verteilten Jobs der eigenen Klasse auf den Maschinen verursacht werden, kennen. Hierbei ist $\mathfrak{A}(h)$ der Vektor der Züge der Spieler und $\mathfrak{X}(h)$ der Vektor der Jobgrößen der schon verteilten Jobs und des aktuellen Jobs. Das Maschinenmodell ist unverändert durch (6.1) definiert.

Wir werden nun den folgenden Satz zeigen:

Satz 6.1. *Seien μ und $\widetilde{\mu}$ Modelle, wie gehabt, und sei $n \in \mathbb{N}$ beliebig, aber fest gewählt. Dann gilt für $\Gamma = \mu(n)$ und $\widetilde{\Gamma} = \widetilde{\mu}(n)$:*

- $\mathcal{G}(\Gamma) \neq \emptyset$, $\mathcal{G}_{\equiv}(\Gamma) \neq \emptyset$, $\mathcal{G}(\widetilde{\Gamma}) \neq \emptyset$,
- *jedes perfekte Gleichgewicht $\phi \in \mathcal{G}(\widetilde{\Gamma})$ ist auch ein perfektes Gleichgewicht von Γ ($\phi \in \mathcal{G}(\Gamma)$) und*
- *für identische Maschinen – $\forall i \in M : s_i = 1$ –, jede Zielfunktion $f \in \{f_{[C_{\max}]}, f_{[\sum C_j]}\}$ und jede Eingabe $a_0 \in \mathcal{E}(n)$ mit $\rho_{\mathfrak{n}}(a_0) > 0$ gilt*

  $$\sup\{\, \mathrm{E} f(\Gamma, a_0, \phi) \mid \phi \in \mathcal{G}(\Gamma)\} = \sup\{\, \mathrm{E} f(\Gamma, a_0, \phi) \mid \phi \in \mathcal{G}_{\equiv}(\Gamma)\}$$

  und

  $$\sup\{\, \mathrm{E} f(\Gamma, a_0, \phi) \mid \phi \in \mathcal{G}_{\equiv}(\Gamma)\} = \sup\{\, \mathrm{E} f(\widetilde{\Gamma}, a_0, \phi) \mid \phi \in \mathcal{G}(\widetilde{\Gamma})\}.$$

Beweis. Wir unterteilen den Beweis in folgende Schritte:

1. Aus Lemma 6.2 auf Seite 73 erhalten wir, dass es ein perfektes Gleichgewicht in $\widetilde{\Gamma}$ gibt. Das heißt, $\mathcal{G}(\widetilde{\Gamma}) \neq \emptyset$.
2. Dann zeigen wir in Lemma 6.12, dass jedes perfekte Gleichgewicht von $\widetilde{\Gamma}$ auch ein perfektes Gleichgewicht von Γ ist. Aus der Existenz von perfekten Gleichgewichten in $\widetilde{\Gamma}$ folgt dann direkt die Aussage, dass $\mathcal{G}_{\equiv}(\Gamma)$ und $\mathcal{G}(\Gamma)$ nicht leer sind.

3. Als nächstes zeigen wir in Lemma 6.13, dass bei dem Spiel Γ mit identischen Maschinen jedes perfekte Gleichgewicht durch den List Scheduling Algorithmus entstehen kann.

4. Die Aussage von Lemma 6.13 wird dann kombiniert mit dem Fakt 6.1. Fakt 6.1 besagt, dass bei jedem von $\mathcal{LS}$ generierten Schedule die Antwortzeiten aller Spieler identisch ist. Aus der Kombination aller drei Aussagen und Lemma 6.2 (jedes perfekte Gleichgewicht von $\widetilde{\Gamma}$ kann durch den List Scheduling Algorithmus erzeugt werden) erhalten wir dann direkt, dass

$$\begin{aligned}\sup\{\,\mathrm{E}f(\Gamma, a_0, \phi) \mid \phi \in \mathcal{G}(\Gamma)\} &= \sup\{\,\mathrm{E}f(\Gamma, a_0, \phi) \mid \phi \in \mathcal{G}_{\equiv}(\Gamma)\} \\ &= \sup\{\,\mathrm{E}f(\widetilde{\Gamma}, a_0, \phi) \mid \phi \in \mathcal{G}(\widetilde{\Gamma})\}.\end{aligned}$$

■

Lemma 6.12. *Jedes perfekte Gleichgewicht von $\widetilde{\Gamma}$ ist auch ein perfektes Gleichgewicht von Γ.*

Lemma 6.13. *Sei Γ ein Spiel, wie in Satz 6.1, mit identischen Maschinen. Dann kann jedes perfekte Gleichgewicht durch den List Scheduling Algorithmus entstehen. Dabei wird der Algorithmus auf jede Klasse von Jobs einzelnd angewendet.*

Fakt 6.1. *Sei eine feste Reihenfolge von Jobs gegeben, dann gilt für alle Schedules S und S', die von dem List Scheduling Algorithmus bei gleicher Reihenfolge der Eingabe erzeugt werden, dass*

$$\forall j : C_j(S) = C_j(S').$$

Beweis von Lemma 6.12. Wir können auf natürliche Art und Weise die Profile von $\widetilde{\Gamma}$ auf Γ übertragen. Sei $\widetilde{\phi}$ ein Profil von $\widetilde{\Gamma}$, dann ist das dazugehörige Profil von Γ, wie folgt, definiert:

$$\phi_j(I, i) := \phi_(\widetilde{I}, i) \quad \text{mit } I \subset \widetilde{I}.$$

Nach Definition von $\mathcal{I}_j$ und $\widetilde{\mathcal{I}}_j$ ist für alle $I \in \mathcal{I}_j$ die Informationsmenge $\widetilde{I} \in \widetilde{\mathcal{I}}_j$ mit der Eigenschaft $I \subset \widetilde{I}$ eindeutig definiert. Entsprechend können wir auch die Testfolgen $(\widetilde{\Gamma}, \eta^1), (\widetilde{\Gamma}, \eta^2), \ldots$ auf die Testfolgen $(\Gamma, \eta^1), (\Gamma, \eta^2), \ldots$ übertragen.

Sei nun ψ ein perfektes Gleichgewicht von $\widetilde{\Gamma}$. Dann gibt es eine Testfolge $(\widetilde{\Gamma}, \eta'^1), (\widetilde{\Gamma}, \eta'^2), \ldots$, so dass ψ Grenzwertgleichgewicht dieser Testfolge mit

den Nash-Gleichgewichten $\psi^1, \psi^2, \ldots$ ist. Aus der Testfolge für $\widetilde{\Gamma}$ erhalten wir direkt die Testfolge $(\Gamma, \eta^1), (\Gamma, \eta^2), \ldots$ für das Spiel Γ, so dass

$$\forall j \in N : \forall I, I' \in \mathcal{I}_j : I \equiv I' \Rightarrow \eta_j^k(I, \cdot) = \eta_j^k(I', \cdot).$$

Sei ϕ das Profil von Γ, das ψ entspricht. Nun konstruieren wir ein Profil ϕ^k für das Spiel (Γ, η^k). Wir wählen ϕ^k so, dass eine Strategie die minimal mögliche Wahrscheinlichkeit einer Maschine i zuweist, die in Profil ϕ mit Wahrscheinlichkeit Null ausgewählt wird. Da diese Maschinen nun mit einer größeren Wahrscheinlichkeit ausgewählt werden, müssen wir in ϕ^k bei den restlichen Maschinen die Wahrscheinlichkeit, dass diese ausgewählt werden, reduzieren. Wir werden diese Reduzierung gleichmäßig auf diese Maschinen verteilen. ϕ^k ist damit dann so definiert, dass die Folge (ϕ^k) ϕ konvergiert. Formal ist ϕ^k durch

$$\phi^k(I, i) := \begin{cases} \phi(I, i) - \frac{\sum_{i' \in M, \phi(I,i')=0} \eta^k(I,i')}{|\{i' \in M | \phi(I,i') > 0\}|}, & \text{falls } \phi(I, i) > 0 \\ \eta^k(I, i), & \text{falls } \phi(I, i) = 0 \end{cases}$$

definiert. Da $\lim_{k\to\infty} \eta^k = 0$ gilt, existiert ein $K \in \mathbb{N}$, so dass für alle $k \geq K$ das Profil ϕ^k wohldefiniert ist. Ohne Beschränkung der Allgemeinheit können wir alle Glieder $< K$ der Testfolge verwerfen und den Rest der Folge umnummerieren, so dass für alle $k \in \mathbb{N}$ das Profil ϕ^k wohldefiniert ist.

Aus der Definition von ϕ^k folgt direkt, dass $\lim_{k\to\infty} \phi^k = \phi$. Es fehlt damit nur noch, zu zeigen, dass ϕ^k ein Nash-Gleichgewicht für (Γ, η^k) ist. Das heißt, wir müssen zeigen, dass für alle Spieler $j \in N$ die Strategie ϕ_j^k eine bestmögliche Antwort auf ϕ^k ist. Dazu betrachten wir erneut die Definition von ϕ^k. Sie stellt sicher, dass für alle $j \in N$ und $I, I' \in \mathcal{I}_j$

$$I \equiv I' \Rightarrow \phi^k(I, \cdot) = \phi^k(I', \cdot)$$

gilt.

Betrachten wir nun die erwarteten Antwortzeiten bezüglich ϕ^k für die einzelnen Aktionen eines festen Spielers $j \in N$ an einer festen Spielposition $I \in \mathcal{I}_j$. Aus der Definition von C_j und, da in ϕ^k die Spieler einer Klasse k unabhängig von den Zügen der Spieler in allen anderen Klassen ihre Strategie auswählen, folgt, dass für alle $h, h' \in Z$ mit $h^{|I|}, h'^{|I|} \in I$

$$\text{Pref}(h, j) = \text{Pref}(h', j).$$

Das heißt, dass Spieler $j \in N$ seine erwartete Antwortzeit minimiert, falls das Spiel in Informationsmenge I gelangt und die anderen Spieler nach ϕ_{-j}^k

spielen, indem er nur Maschinen $i \in M$ mit $\phi_j^k(I,i) > \eta_j^k(I,i)$ auswählt, für die

$$\sum_{\substack{1 \le j' \le |I|+1 \\ j' \in N^i_{\kappa(I,j)^{(h)}}}} \frac{x_{j'}(I)}{s_i} = \min_{i' \in M} \sum_{\substack{1 \le j' \le |I|+1 \\ j' \in N^{i'}_{\kappa(I,j)^{(h)}}}} \frac{x_{j'}(I)}{s_{i'}}$$

gilt.

Nach Lemma 6.2 auf Seite 73 folgt, dass nur solche Maschinen mit einer Wahrscheinlichkeit $\phi_j(I,i) > 0$ in einem perfekten Gleichgewicht von Γ' ausgewählt werden. Das heißt, dass ϕ^k ein Nash-Gleichgewicht von (Γ, η^k) darstellt. Damit ist ϕ ein perfektes Gleichgewicht von Γ. ■

Bevor wir Lemma 6.13 beweisen, werden wir Fakt 6.1 zeigen. Zur Erinnerung, die Aussage von Fakt 6.1 ist, dass, wenn eine feste Reihenfolge von Jobs gegeben ist, dann gilt für alle Schedules S und S', die von dem List Scheduling Algorithmus bei gleicher Reihenfolge der Eingabe erzeugt werden, dass

$$\forall j : C_j(S) = C_j(S').$$

Das heißt, dass die Antwortzeiten der einzelnen Jobs nicht durch die Wahlfreiheit des List Scheduling Algorithmus beeinflusst werden.

Dies kann mittels Induktion über die Anzahl der Jobs eingesehen werden. Für einen einzelnen Job ist es offensichtlich wahr. Gelte es nun für $j-1$ Jobs und wir betrachten die Verteilung des j-ten Jobs. Die Lasten auf den Maschinen sind für alle gültigen Verteilungen der ersten $j-1$ Jobs bis auf eine Permutation der Maschinen identisch. Ansonsten wären die Antwortzeiten der ersten $j-1$ Jobs nicht gleich. List Scheduling wählt eine Maschine mit minimaler Last für Job j aus. Da die Maschinen minimaler Last bei allen Verteilungen die gleiche Last haben, folgt, dass Job j in allen gültigen Verteilungen die gleiche Antwortzeit hat.

Dieser Fakt wird nun für den Beweis von Lemma 6.13 ausgenutzt.

Beweis von Lemma 6.13. Wir zeigen nun, dass in einem perfekten Gleichgewicht alle Spieler eine Strategie spielen, die auch der List Scheduling Algorithmus auswählen würde, wenn man ihm nur die Jobs der gleichen Klasse präsentiert hätte. Es genügt uns zu zeigen, dass dies für jedes Nash-Gleichgewicht gilt, da jedes perfekte Gleichgewicht auch ein Nash-Gleichgewicht ist.

- Betrachten wir zuerst die Jobs der kleinsten Klasse. Es gibt für diese Spieler keinen Job, der zu einer kleineren Klasse gehört. Damit beeinflussen nur die schon verteilten Jobs der gleichen Klasse die Antwortzeiten der Spieler. Wir benutzen eine ähnliche Begründung wie bei dem Beweis von Lemma 6.2. Falls sich ein Spieler nun nicht an die Strategie von List Scheduling hält, kann er seine Antwortzeit verbessern, indem er die Maschine wechselt.

- Gehen wir nun davon aus, dass für die Klasse $< k$ die Spieler dem List Scheduling Algorithmus folgen. Betrachten wir nun Klasse k. Da für alle Jobs der kleineren Klassen sich die Spieler an List Scheduling halten, gilt, dass die Verzögerung, die durch diese Jobs verursacht wird, nur abhängig von der Reihenfolge der Jobs in der Eingabe ist. Mit anderen Worten wir erhalten, wie in Lemma 6.1 auf Seite 72, dass für alle Spieler $j \in N$ der Klasse k, Informationsmengen $I \in \mathcal{I}_j$ und ein perfektes Gleichgewicht ϕ von Γ die erwartete Verzögerung durch die Jobs einer kleineren Klasse wegen Fakt 6.1 unabhängig von der Wahl der Maschine von Spieler j ist. Das heißt,

$$\forall i, i' \in M : \mathrm{E}_\phi[\mathrm{Pref}(\cdot, j)|Y_{I,i}] = \mathrm{E}_\phi[\mathrm{Pref}(\cdot, j)|Y_{I,i'}],$$

  wobei $Y_{I,i} \subseteq Z$ die Menge der terminalen Historien ist, bei denen das Spiel die Informationsmenge I erreicht und Spieler j Maschine i ausgewählt hat. Dann können wir direkt die Analyse aus Lemma 5.1 anwenden und erhalten, dass sich dieser Spieler j auch an den List Scheduling Algorithmus gehalten hätte. Dies gilt für alle Spieler dieser Klasse. Damit werden auch die Jobs der Klasse k in einem perfekten Gleichgewicht nur auf die Maschinen verteilt, die auch der List Scheduling Algorithmus ausgewählt hätte. ■

Es ist uns nicht bekannt, wie man die Aussage von Satz 6.1 auf beliebige Maschinen erweitert. Allerdings haben wir folgende Vermutung:

Vermutung 6.1 (Verallgemeinerung von Satz 6.1)**.** *Seien μ und $\widetilde{\mu}$ Modelle, wie gehabt, bei denen aber beliebige Einlastzeiten erlaubt sind, und sei $n \in \mathbb{N}$ beliebig, aber fest gewählt. Dann gilt für $\Gamma = \mu(n)$ und $\widetilde{\Gamma} = \widetilde{\mu}(n)$:*

- $\mathcal{G}(\Gamma) \neq \emptyset$, $\mathcal{G}_{\equiv}(\Gamma) \neq \emptyset$, $\mathcal{G}(\widetilde{\Gamma}) \neq \emptyset$,

- *jedes perfekte Gleichgewicht $\phi \in \mathcal{G}(\widetilde{\Gamma})$ ist auch ein perfektes Gleichgewicht von Γ ($\phi \in \mathcal{G}(\Gamma)$) und*

- *für jede Zielfunktion* $f \in \{f_{[C_{\max}]}, f_{[\sum C_j]}\}$ *und jede Eingabe* $a_0 \in \mathcal{E}(n)$ *mit* $\rho_{\mathfrak{n}}(a_0) > 0$

$$\sup\{\, \mathrm{E}f(\Gamma, a_0, \phi) \mid \phi \in \mathcal{G}(\Gamma)\} = \sup\{\, \mathrm{E}f(\Gamma, a_0, \phi) \mid \phi \in \mathcal{G}_{\equiv}(\Gamma)\}$$

und

$$\sup\{\, \mathrm{E}f(\Gamma, a_0, \phi) \mid \phi \in \mathcal{G}_{\equiv}(\Gamma)\} = \sup\{\, \mathrm{E}f(\widetilde{\Gamma}, a_0, \phi) \mid \phi \in \mathcal{G}(\widetilde{\Gamma})\}.$$

6.7 Beliebige Einlastzeiten

Das hier vorgestellte Modell und dessen Analyse lassen sich nicht einfach auf beliebige Einlastzeiten erweitern. Das Problem entsteht dadurch, dass die Maschinen in dem hier vorgestellten Modell die Information benötigen, welche weiteren Jobs der gleichen Klasse noch auf der Maschine platziert werden, bevor mit der Ausführung begonnen wird.

Es gibt zwei naheliegende Möglichkeiten das Modell zu erweitern. Wir werden im Folgenden begründen, warum wir bei beiden Erweiterungen nicht erwarten können, dass wir ähnliche Ergebnisse wie bei dem in diesem Kapitel vorgestellten Modell erhalten.

Falls wir von präemptiven Scheduling ausgehen, dann könnten wir zu einem Zeitpunkt $t \in \mathbb{R}_{\geq 0}$ die Bearbeitung der Jobs der Klasse k unterbrechen, falls ein Job j einer Klasse $< k$ mit Einlastzeit $r_j = t$ auf eine Maschine verteilt wird. Wir müssten wieder alle Maschinen unterbrechen, um sicherzustellen, dass die Antwortzeiten der Jobs der Klasse k auf allen Maschinen im gleichen Maße durch die Jobs der Klassen $< k$ beeinflusst werden. Dadurch können weitere zusätzliche Leerlaufzeiten entstehen. So kann im ungünstigsten Fall ein Job der Klasse k immer wieder durch Jobs der kleineren Klassen ausgebremst werden. Dies ist zum Beispiel der Fall, wenn während der Ausführung des Jobs der Klasse k Jobs einer kleineren Klasse $k' < k$ eingelastet werden. Gehen wir davon aus, dass zum Zeitpunkt t der Job der Klasse k bearbeitet wird. Die Größen der Jobs $j_1, j_2, \ldots, j_l$ der Klasse k' seien gerade $2^{k'}$. Sei die Einlastzeit von Job j_λ genau $t+(\lambda-1)2^k$. Damit werden diese Jobs der Klasse k' nacheinander bearbeitet und bremsen so den Job der Klasse k aus. Falls nun l sehr groß ist, wird der Job der Klasse k sehr lange im Verhältnis zu seiner Größe verzörgert. Damit wird das Ziel der gewählten Klassifizierung nicht erreicht, da durch das exponentielle Wachstum der Klassen verhindert werden sollte, dass die Bearbeitung eines Jobs einer größeren Klasse durch die kleineren Jobs in einem besonderen Maße verzögert wird. Wie wir an

diesen Überlegungen sehen, funktioniert diese Idee in Kombination mit dem Einfügen von Leerlaufzeiten nur, falls alle Einlastzeiten der Jobs Null sind.

Die zweite Möglichkeit besteht in der Betrachtung von nicht-präemptiven Scheduling. In diesem Fall müsste ein Job der Klasse k, der zu einem Zeitpunkt $t \in \mathbb{R}_{\geq 0}$ bearbeitet wird, bis zum Ende bearbeitet werden, falls ein Job j der Klasse $< k$ mit Einlastzeit $r_j = t$ auf eine der Maschinen verteilt wird. Da dieser Job schon bekannt ist, können wir direkt auf den anderen Maschinen mit den Jobs der Klasse $< k$ fortfahren, würden allerdings die Bearbeitung weiterer Jobs der Klasse k zurückhalten, damit wir eine ähnliche Struktur für die perfekten Gleichgewichte erhalten wie bei dem in diesem Kapitel vorgestellten Priorisierungsmodell. Bei ungünstigen Einlastzeiten kann nun der Effekt eintreten, dass alle Jobs in nicht-aufsteigender Reihenfolge ihrer Größe bearbeitet werden. Die Einlastzeiten liegen aber so dicht beieinander, dass es für die Zielfunktionen $f_{[\sum F_j]}$ und $f_{[\sum C_j]}$ günstiger gewesen wäre, Leerlaufzeiten am Anfang einzufügen und in umgekehrter Reihenfolge die Jobs abzuarbeiten. Wir erhalten damit die gleiche untere Schranke für das Koordinationsverhältnis für $f_{[\sum C_j]}$ und $f_{[\sum F_j]}$ von $\Omega(n)$, wie in dem einfachen Modell aus Kapitel 5. Somit sehen wir, dass die naheliegenden Erweiterungen nicht die gewünschte Verbesserung gegenüber dem Modell aus Kapitel 5 erreichen.

Kapitel 7

Zusammenfassung und Ausblick

In dieser Arbeit haben wir erläutert, wie man online Scheduling-Spiele darstellen kann. Diese Modellierung hat den Vorteil, dass sie die Möglichkeit bietet, verschiedene Varianten abzubilden, bei denen die Spieler mehr oder aber auch weniger Informationen erhalten, als einem Online-Algorithmus zur Verfügung stehen würden.

Sei ein Online-Algorithmus gegeben, dann ist es immer möglich, ein Spiel zu konstruieren, bei dem die Spieler sich an die Strategien dieses Online-Algorithmus halten. Man erreicht dies dadurch, indem man alle Spieler, die sich nicht an den Algorithmus halten, bestraft. Das resultierende Spiel hat dann allerdings eine solche Struktur, dass es keinen Sinn mehr machen würde, nicht direkt eine zentrale Verwaltung der Jobs durchzuführen.

Wir haben gesehen, dass, wenn man sich keiner unnatürlichen Konstruktion der Maschinenmodelle bedienen will, die Spieler auf einfache Art und Weise dazu gebracht werden können, sich wie der List Scheduling Algorithmus zu verhalten. Das wird erreicht, indem man die Jobs auf den Maschinen in der Reihenfolge, in der die Jobs verteilt werden, abarbeitet. Dies liegt an der Struktur des Algorithmus, da er in jedem Schritt die aktuell günstigste Maschine auswählt und damit auch die Reihenfolge der Jobs auf den Maschinen aus dem einfachen Modell nachbildet. Auch wenn dieser Algorithmus gut für die Optimierung der maximalen Antwortzeit geeignet ist, konnten wir zeigen, dass bezüglich der durchschnittlichen Antwortzeit das Koordinationsverhältnis in $\Omega(n)$ liegt.

Weiterhin wurde aufgeführt, dass dieses Problem umgangen werden kann, wenn man eine Klassifizierung der Jobs nach ihrer Größe und eine Priorisierung dieser Klassen einführt. Dies hat allerdings auch den Nachteil, dass nun eine gewisse Kommunikation zwischen den Maschinen stattfinden muss.

Zudem gelten die Ergebnisse nur für Spiele bei denen die Einlastzeiten aller Jobs immer Null sind. Um dafür zu sorgen, dass die Spieler in perfekten Gleichgewichten vernünftige Strategien spielen, beginnen die Maschinen mit der Bearbeitung eines Jobs einer Klasse k erst, wenn alle Jobs der Klassen $< k$ schon bearbeitet wurden.

Für identische Maschinen wurde von uns bewiesen, dass wir die Informationen in dem Spiel reduzieren können. Die Spieler kennen nur noch die schon verteilten Jobs der eigenen Klasse. Für diese eingeschränkten Spiele haben wir gezeigt, dass der Preis der Anarchie bezüglich der Zielfunktionen maximale Antwortzeit $f_{[C_{\max}]}$ und durchschnittliche Antwortzeit $f_{[\sum C_j]}$ durch eine Konstante beschränkt ist. Die Analyse dieses eingeschränkten Spielmodells wurde für beliebige Maschinen durchgeführt. Außerdem haben wir bewiesen, dass im Fall der durchschnittlichen Antwortzeit der Preis der Anarchie in $O\left(s_{\max}^2/s_{\min}^2\right)$ und im Fall der maximalen Antwortzeit in $O\left(s_{\max}/s_{\min}\right)$ liegt.

Aus der hier vorgestellten Modellierung und der Analyse des Preises der Anarchie ergeben sich unter anderem folgende interessante Fragestellungen, die noch offen sind.

- Kann Vermutung 6.1 gezeigt werden?
- Wie ist das Koordinationsverhältnis bei dem vorgestellten Prioritätsmaschinenmodell für die durchschnittliche Antwortzeit bei beliebigen Maschinen?
- Gibt es bessere Modelle für die in dieser Arbeit beschriebenen Zielfunktionen, bei denen nicht unnatürlicher Zwang auf die Spieler, wie in Kapitel 4 beschrieben, ausgeübt wird?
- Wie verhalten sich die in dieser Arbeit vorgestellten Spiele bezüglich des Preises der Anarchie und des Koordinationsverhältnisses bei anderen Zielfunktionen? Betrachten wir dabei zum Beispiel den Fall, dass die Antwortzeit in das Verhältnis zu der Jobgröße gesetzt wird, was durch

  $$f^n_{[C_{\max}]}(C(h)) := \max_{j \in N} \frac{C_j(h)}{x_{p_j(h)}(h)}$$

  und

  $$f^n_{[\sum C_j]}(C(h)) := \sum_{j \in N} \frac{C_j(h)}{x_{p_j(h)}(h)}$$

  repräsentiert wird.

  Was sind gute Maschinenmodelle bezüglich dieser Zielfunktionen?

- Was ist ein gutes Modell mit beliebigen Einlastzeiten bezüglich der Zielfunktionen $f_{[\sum C_j]}$ und $f_{[\sum F_j]}$?

Des Weiteren kann diese Form der Modellierung mit Spielen in Extensivform mit imperfekten Informationen auch auf andere Probleme übertragen werden. So stellt sich die Frage, welche Ergebnisse dort erreicht werden können, wenn einzelne Spieler Handlungen vornehmen und keine zentrale Kontrolle stattfindet.

Die Art und Weise wie Informationen modelliert werden, kann man auch als Basis für das Design von Algorithmen verwenden. So können an Stelle von online und semi-online Modellen andere Formen dargestellt werden, bei denen zum Beispiel weniger Informationen zur Verfügung stehen. Wie auch schon verschiedene semi-online Modelle in der Arbeit von Ebenlendr und Sgall [ES11] verallgemeinert wurden, stellt sich die Frage, ob auch bei unserer Verallgemeinerung des Modells ein Meta-Algorithmus gefunden werden kann, der gute Ergebnisse erzielt.

Literaturverzeichnis

[AA03] Nir Avrahami and Yossi Azar. Minimizing total flow time and total completion time with immediate dispatching. In *In Proc. 15th Symp. on Parallel Algorithms and Architectures (SPAA*, pages 11–18. ACM, 2003.

[AAF+97] James Aspnes, Yossi Azar, Amos Fiat, Serge A. Plotkin, and Orli Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *J. ACM*, 44(3):486–504, 1997.

[AR98] Yossi Azar and Oded Regev. On-line bin-stretching. In Michael Luby, José D. P. Rolim, and Maria J. Serna, editors, *RANDOM*, volume 1518 of *Lecture Notes in Computer Science*, pages 71–81. Springer, 1998.

[BCM98] Michael A. Bender, Soumen Chakrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In Howard J. Karloff, editor, *SODA*, pages 270–279. ACM/SIAM, 1998.

[BGGM04] P. Berenbrink, L.A. Goldberg, P.W. Goldberg, and R.A. Martin. Utilitarian resource assignment. *Computing Research Repository (CoRR)*, cs.GT/0410018, 2004.

[CMM67] R.W. Conway, W.L. Maxwell, and L.W. Miller. *Theory of Scheduling*. Addison Wesley, 1967.

[CS80] Yookun Cho and Sartaj Sahni. Bounds for list schedules on uniform processors. *SIAM J. Comput.*, 9(1):91–103, 1980.

[CV02] A. Czumaj and B. Vöcking. Tight bounds for worst-case equilibria. *Proc. of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 413–420, 2002.

[ES11] Tomás Ebenlendr and Jiri Sgall. Semi-online preemptive scheduling: One algorithm for all variants. *Theory Comput. Syst.*, 48(3):577–613, 2011.

[FGL+03a] R. Feldmann, M. Gairing, T. Lücking, B. Monien, and M. Rode. Nashification and the coordination ratio for a selfish routing game. In *Proc. of the 30th Internaltional Colloquium on Automata, Languages and Programming (ICALP)*, volume 2719 of *Lecture Notes in Computer Science (LNCS)*, pages 514–526, 2003.

[FGL+03b] R. Feldmann, M. Gairing, T. Lücking, B. Monien, and M. Rode. Selfish routing in non- cooperative networks: A survey. In *Proc. of the 28th International Symposium Mathematical Foundations of Computer Science*, volume 2747 of *Lecture Notes in Computer Science (LNCS)*, pages 21–45, Spinger Verlag, Heidelberg, 2003.

[FKT89] U. Faigle, W. Kern, and G. Turán. On the performance of online algorithms for partition problems. *Acta Cybernetica*, 9:107–119, 1989.

[FW99] Amos Fiat and Gerhard J. Woeginger. On-line scheduling on a single machine: Minimizing the total completion time. *Acta Inf.*, 36(4):287–293, 1999.

[GLM+03] M. Gairing, T. Lücking, M. Mavronicolas, B. Monien, and P.G. Spirakis. Extreme nash equilibria. In *Proc. of the 8th Italian Conference on Theoretical Computer Science (ICTCS)*, volume 2841 of *Lecture Notes in Computer Science (LNCS)*, pages 1–20, 2003.

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

[ILMS09] Nicole Immorlica, Li Li, Vahab S. Mirrokni, and Andreas S. Schulz. Coordination mechanisms for selfish scheduling. *Theor. Comput. Sci.*, 410(17):1589–1598, 2009.

[KP99] E. Koutsoupias and C.H. Papadimitriou. Worst-case equilibira. In *Proc. of the 16th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1563 of *Lecture Notes in Computer Science (LNCS)*, pages 404–413, 1999.

[Kre06] M. Kretschmer. Worstcase Nashgleichgewichte in Lastbalancierungs-Spielen. *Diplomarbeit, Rheinische Friedrich-Wilhelm-Universität Bonn, Informatik V*, 2006.

[LKA04] Joseph Leung, Laurie Kelly, and James H. Anderson. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis.* CRC Press, Inc., Boca Raton, FL, USA, 2004.

[Nas50] J.F. Nash. Equilibrium points in n-person games. *Proc. of the National Academy of Science of the United States of America*, 36:48–49, 1950.

[OR94] M.J. Osborne and A. Rubinstein. *A Course in Game Theory.* MIT Press, 1994.

[PST03] Kirk Pruhs, Jiri Sgall, and Eric Torng. Online scheduling. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, pages 115–124. CRC Press, 2003.

[Ras89] Eric Rasmussen. *Games and Information - An Introduction to Game Theory.* Blackwell Publishers, Oxford, 3 edition, 1989.

[Sel75] R. Selten. Reexamination of the perfectness concept for equilibrium points in extensive games. *International Journal of Game Theory*, 4:25–55, 1975. 10.1007/BF01766400.

[Sga98] Jirí Sgall. On-line scheduling - a survey. In *Online Algorithms: The State of the Art*, Lecture Notes in Computer Science 1442, pages 196–231. Springer, 1998.