The Design of Scheduling Algorithms Using Game Theoretic Ideas

by

Janardhan Kulkarni

Department of Computer Science
Duke University

Date: __________________

Approved: __________________

Kamesh Munagala, Supervisor

Vincent Conitzer

Bruce Maggs

Kirk Pruhs

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2015
ABSTRACT

The Design of Scheduling Algorithms Using Game Theoretic Ideas

by

Janardhan Kulkarni

Department of Computer Science
Duke University

Date: ____________________

Approved:

________________________
Kamesh Munagala, Supervisor

________________________
Vincent Conitzer

________________________
Bruce Maggs

________________________
Kirk Pruhs

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University 2015
Abstract

Scheduling a set of jobs over a collection of machines to optimize some quality-of-service measure is one of the most important research topics in computer science theory and practice. In this thesis, we design algorithms that optimize flow-time (or delay) of jobs for scheduling problems that arise in a wide range of applications. We consider the classical model of unrelated machine scheduling and resolve several long standing open problems; we introduce new models that capture the novel algorithmic challenges in scheduling jobs in data centers or large clusters; we study the effect of selfish behavior in distributed and decentralized environments; we design algorithms that strive to balance the energy consumption and performance.

The main technical contribution of our work is the connections we establish between the design of approximation and online algorithms for scheduling problems and economics, game theory, and queueing theory. The interplay of ideas from these different areas lies at the heart of most of the algorithms presented in this thesis. The main contributions of the thesis can be placed in one of the following categories.

• Classical Unrelated Machine Scheduling: We give the first polylogarithmic approximation algorithms for minimizing the average flow-time and minimizing the maximum flow-time in the offline setting. In the online and non-clairvoyant setting, we design the first non-clairvoyant algorithm for minimizing weighted flow-time in the resource augmentation model. Our work introduces the iterated rounding technique for the offline flow-time optimization, and gives the first duality based framework to analyze non-clairvoyant algorithms for unrelated machines.
• **Polytope Scheduling Problem:** To capture the multidimensional nature of the scheduling problems that arise in practice, we introduce the Polytope Scheduling Problem (PSP). The PSP problem not only generalizes almost all classical scheduling models, but also captures hitherto unstudied scheduling problems such as routing multi-commodity flows, routing multicast (video-on-demand) trees, and multi-dimensional resource allocation. We design several competitive algorithms for the PSP problem and its variants for the objectives of minimizing flow-time and completion time. Our work establishes many connections between scheduling and market equilibrium concepts, fairness and non-clairvoyant scheduling, Lyapunov function method and the potential function method, and queueing theoretic notion of stability and the resource augmentation analysis.

• **Energy Efficient Scheduling:** We give the first non-clairvoyant algorithm for minimizing the total flow-time + energy in the online and resource augmentation model for the most general setting of unrelated machines. Further, we initiate the theoretical study of cost aware scheduling.

• **Selfish Scheduling:** We study the effect of selfish behavior in scheduling and routing problems. We define a fairness index for scheduling policies called bounded stretch, and show that for the objective of minimizing the average (weighted) completion time, policies with small stretch lead to equilibrium outcomes with small price of anarchy. Our work gives the first linear/ convex programming duality based framework to bound the price of anarchy for equilibrium concepts such as coarse correlated equilibrium.
To my parents and all my teachers.
## Contents

Abstract ix

Acknowledgements xiv

1 Introduction 1

1.1 Overview of Our Results 3

1.1.1 Iterated Rounding For Flow-Time Optimization 3

1.1.2 Dual-Fitting Framework for Non-Clairvoyant Scheduling 4

1.1.3 Polytope Scheduling: Machine Scheduling Meets Markets 5

1.1.4 Selfish Behavior, Price of Anarchy, and Duality 6

1.2 Preliminaries 6

1.2.1 Basics of Scheduling Theory 7

1.2.2 Machine Environments 7

1.2.3 Quality Of Service (QoS) 8

1.2.4 Popular Scheduling Policies 9

1.2.5 Measuring Performance: Approximation Factor, Competitive Ratio 12

1.2.6 Measuring Performance Amidst Selfish Behavior: Price of Anarchy (PoA) 13

1.3 Road Map 15

2 Approximation Algorithms For Minimizing Total Flow-Time 17

2.1 Introduction 17

2.1.1 Our Results 18

2.1.2 History 19
2.2 Alternate LP Relaxation and The High-Level Idea ......................... 20
2.3 Tentative Schedule to Actual Schedule ..................................... 24
2.4 Iterated Rounding of LP\textsubscript{new} ................................. 28
  2.4.1 Analysis .......................................................... 30
  2.4.2 The $O(\log^2 n)$ approximation .................................. 34

3 Minimizing the Maximum Flow-time ............................................. 36
  3.1 Introduction ............................................................ 36
  3.1.1 Our Results ......................................................... 37
  3.1.2 History .............................................................. 37
  3.2 Overview of Techniques .................................................. 38
  3.3 A Linear Programming Relaxation ....................................... 39
  3.4 Iterated Rounding ...................................................... 41

4 Online and Non-Clairvoyant Algorithms For Minimizing Flow-Time .... 46
  4.1 Introduction ............................................................ 46
  4.1.1 Our Results ......................................................... 48
  4.2 Technical Contributions: Selfish Migration and Nash Equilibrium ...... 49
    4.2.1 \textsc{SelfishMigrate} Framework ................................. 50
  4.3 History ............................................................... 53
  4.4 Non-clairvoyant Scheduling ............................................. 55
    4.4.1 The \textsc{SelfishMigrate} Algorithm ............................ 56
  4.5 Analysis of \textsc{SelfishMigrate} .................................... 59
  4.6 Polynomial Time Algorithm and Minimizing Reassignments. ............. 64

5 Energy Efficient Scheduling to Minimize Flow-Time ......................... 67
  5.1 Introduction ........................................................... 67
  5.1.1 Our Results ....................................................... 68
  5.2 The \textsc{SelfishMigrate-Energy} Algorithm .......................... 69
5.3 Analysis of **SelfishMigrate-Energy** ........................................ 71

6 Cost Aware Scheduling ................................................. 77

6.1 Introduction .......................................................... 77

6.1.1 Our Model ......................................................... 79

6.1.2 Our Results ....................................................... 80

6.1.3 History .......................................................... 82

6.2 Online Algorithms for Weighted Flow Time ......................... 83

6.2.1 Lower Bounds .................................................... 84

6.2.2 The Two-Cost Problem .......................................... 86

6.2.3 Proof Outline .................................................... 87

6.2.4 Simplifying the Input ........................................... 88

6.2.5 Online Algorithm **BALANCE** .................................. 89

6.2.6 Analysis of **BALANCE** ........................................ 89

6.3 Offline Problem: Weighted Completion Time ....................... 95

6.3.1 No Release Dates: 1|\text{pmtn}| \sum_{j}(w_jC_j + E(j)) ................. 96

6.4 Minimizing Weighted Completion Time with Release Dates ........ 99

6.4.1 \(K = 2\) Case: Quasi-polynomial Time Algorithm ................. 99

6.4.2 LP Relaxation and Analysis via Speed Augmentation .......... 101

7 Polytope Scheduling .................................................. 105

7.1 Introduction .......................................................... 105

7.2 General Polytope Scheduling Framework ............................ 107

7.3 Applications of the PSP Framework ................................ 108

7.3.1 All-or-nothing Multidimensional Scheduling .................... 108

7.3.2 Scheduling Jobs with Different Parallelizability over Multiple Resources ........................................... 109

7.3.3 Non-clairvoyant Scheduling for Unrelated Machines .......... 109
11 Queueing View of PSP

11.1 Introduction .................................................. 160
    11.1.1 Our Results ........................................... 161
11.2 Applications of PSP-Q ...................................... 163
    11.2.1 Flow Routing ......................................... 163
    11.2.2 Resource Allocation with Complementarities (RA-C) ...... 163
11.3 Non-Clairvoyant Algorithm for PSP-Q .......................... 164
    11.3.1 Algorithm Description ................................. 164
    11.3.2 Analysis ............................................... 166
11.4 Another Algorithm for PSP-Q .................................. 170
    11.4.1 Algorithm Description ................................. 171
    11.4.2 Analysis ............................................... 172
11.5 Resource Allocation with Complementarities (RA-C) .......... 174

12 Coordination Mechanisms For Selfish Scheduling .......... 179

12.1 Introduction .................................................. 179
    12.1.1 Model .................................................. 180
    12.1.2 Our Results ........................................... 181
12.2 History .......................................................... 187
12.3 Preliminaries .................................................... 189
    12.3.1 Smooth Games and Robust Price of Anarchy .............. 189
12.4 Robust Price of Anarchy Bound via Potential Function ........ 190
    12.4.1 Proof of Theorem 97 ................................... 194
12.5 Price of Anarchy using Dual Fitting ........................ 197
    12.5.1 Properties of Scheduling Policies ...................... 198
    12.5.2 LP-relaxation ......................................... 200
    12.5.3 Analysis ............................................... 201
14.4.4 Tree Topologies ................................................. 261
14.5 Energy Minimization Games in Machine Scheduling ............. 262
  14.5.1 Robust PoA bound ........................................... 263
14.6 Summary and Open Problems ..................................... 269

15 Conclusions .......................................................... 271
Bibliography ............................................................ 273
Biography ............................................................... 284
Acknowledgements

As a fledgling graduate student in theoretical computer science, I wanted to design algorithms of deep mathematical beauty that can have an impact on the lives of millions of people. Surely, I had no problems in having lofty ambitions. But, there was a small hiccup: I did not know how to do research, nor did I know where to begin.

Today if I am able to write this thesis it is because I was fortunate to get help from a lot of people. And the person who is most responsible for my transformation is my advisor Kamesh Munagala. Kamesh was a great advisor: he was encouraging when things were going slow, and appreciative when I did well. He was there for me whenever I needed him: for technical discussions, for feedback, and for conference travel money! Thanks a lot Kamesh for everything, and I hope that I will continue to receive your advice for years to come.

I had the privilege of having some of the most extraordinary professors, Vince Conitzer, Bruce Maggs, and Kirk Pruhs, as my thesis committee members. I would like to thank them for sharing their time and invaluable experience in academic and non-academic matters. My interactions with Bruce, at numerous parties at his beautiful house, and Vince, two of the most accomplished people I have known in my life, will remain as some of the fondest memories of my time at Duke. I would like say special thanks to Kirk for hosting me at Pittsburgh, early in my graduate studies, that deepened my interest in scheduling theory.

When I started my research work in algorithms, I read a lot of elegant papers by Nikhil Bansal. He was the researcher I wanted to emulate, hence, it was very special when I eventually got a chance to work with him. Befittingly, my favorite result of the thesis is in
collaboration with him. The time I spent at Eindhoven working with Nikhil was a great learning experience, and I would like thank him for all the advice and inspiration.

A large part of this thesis work is in collaboration with Sungjin Im. Sungjin was there to discuss ideas 24/7, literally, and his work ethic is one thing I hope I can inculcate. Sungjin also gave me a very simple mantra for doing good research: “Do not be greedy for papers; just solve one problem a month”. Working with Sungjin is a pleasure, and I am looking forward for many years of fruitful collaboration.

Special thanks to Pankaj Agarwal, one of my personal heroes, for always being there to offer guidance and help. I joined Duke because of him, and I hope I will continue to receive his guidance long after I leave Duke.

I cannot thank enough Debmalya Panigrahi, Sreenivas Gollapudi, Nikhil Devanur, Tim Roughgarden, Deeparnab Chakrabarti, Anupam Gupta, Ashwin Machanavajjhala, Aravind Srinivasan, and Naveen Garg for their generosity in sharing research experiences, for championing my work, for giving me invaluable feedback, and above all, for recommending me at various points during my graduate studies. Thank you all so much for being such amazing mentors!

While I was lucky to have great mentors, I was equally fortunate have wonderful collaborators - Ben Moseley, Kyle Fox, Vahab Mirrokni, Sayan Bhattacharya, Xiaoming Xu, Elias Koutsoupias, and Stefano Leonardi. Thank you all! You guys made research a lot more fun!

I would like to thank my family members: my brother Sanjeev, my sister-in-law Shailaja, and my favorite person in the universe, my five year old niece Shreya; my teachers: sister Supriya, Mary teacher, SVS, LNR, MSN, BNP; my friends from forever: SB, Manu, Holla, Archana Patel, Suvarna, Anjana, Nanda, Varun; my friends and extended family at Duke: Marilyn, Rob, Bala, Wuzhou, Pablo, Sharath, Jiangwei, Nagababu, Mayuresh, Animesh, Abhishek, Salman, Jacopo, Racheal, Bing, Rozy, Rohit, Jenny, Harish, Dina, Marisabel, Songchun, Vishwa, Kathleen, Sarah, Jewel, Pam, Claire, Reza; without their affection my life wouldn’t be the same!
I would like to acknowledge the special role played in my life by my primary school teacher sister Benvita. She taught me many things about life, none more important than loving people for what they are. No one rejoices in my successes as much as she does, and her selfless life will always be an inspiration to me.

In Indian tradition, children do not explicitly thank their parents as it is believed that words cannot adequately express the feelings of gratitude. Yet, having lived away from India for more than half a decade, I feel I can deviate from this tradition for once and include them here. Needless to say, all I am today is because of them. It is my parents' sacrifices, their vision for my future, the unconditional love only they could have shown me, that form the building blocks of my work and my life. I dedicate this thesis to them.
From massive data centers to portable computing devices, scheduling a set of jobs over a collection of machines (cores) to optimize some quality of service (QoS) measure is one of the fundamental problems that arises in a broad range of applications. Besides the practical relevance, scheduling and resource allocation problems are also some of the most important topics in the theory of algorithm design. The origins of many foundational algorithm design principles, such as LP rounding methods, online amortized potential functions etc., can be traced to the study of these problems.

In this thesis, we design algorithms to optimize delay or flow-time objective for various scheduling problems: we consider the classical models of unrelated machine scheduling and resolve several important open problems; we introduce new models that capture the novel algorithmic challenges that arise in modern computing environments such as data centers or large clusters; we study scheduling problems in decentralized and selfish settings; we design algorithms that strive to balance energy consumption and performance.

A technically interesting aspect of our results is the surprising connections they show between the design of approximation and online algorithms and economics, game theory, and queueing theory: Our resolution of a classical online scheduling problem crucially uses the idea of Nash equilibrium in the design and analysis of the online algorithm. Our work
Figure 1.1: Many scheduling problems considered in this thesis use concepts from different areas both in the design and analysis of algorithms. The figure illustrates various connections established in this thesis along with their chapter numbers.

in game theory shows that LP duality theory, a fundamental technique in the area of approximation and online algorithms, is equally powerful in the price of anarchy analysis. Fairness as an algorithm design ingredient plays a critical role in our work on the general polytope scheduling problem, which also shows new ways of looking at scheduling problems through the lens of market equilibrium concepts. Queueing view of the polytope scheduling problem reveals an intrinsic connection between queuing theoretic notion of stability to that of resource augmentation analysis in the adversarial setting, and connections to the Lyapunov function argument and the amortized potential argument. This interplay of ideas from various different areas lies at the heart of many of the algorithms presented in this thesis. The figure (1.1) gives an illustration.
1.1 Overview of Our Results

A typical scheduling instance consists of a set of machines and a set of jobs. Each job is characterized by a processing requirement, a release time, and a weight. The goal of a scheduling algorithm is to assign jobs machines and specify the order in which jobs are executed so as to optimize some QoS measure. This seemingly simple model has given rise to some very elegant algorithm design principles and many unyielding open problems. In this thesis, we consider scheduling to optimize the delay or flow-time objective.

1.1.1 Iterated Rounding For Flow-Time Optimization

The unrelated machine setting models the heterogeneity found in modern computer architectures: jobs have different processing requirements on different machines. The problem of minimizing flow-time on unrelated machines is one of the central questions in the theory of approximation algorithms. The flow-time measures the amount of time a job spends in the system before it completes, and is one of the most natural measures of quality-of-service. The flow-time objective has been extensively studied for decades (Garg and Kumar, 2006b,a; Sitters, 2008; Garg et al., 2008; Leonardi and Raz, 2007; Awerbuch et al., 2002; Avrahami and Azar, 2003), and yet, several important questions remain open. A fundamental difficulty with the flow-time objective is that even a slightest change to the input can lead to drastic changes in the optimal solution value. Hence, despite several attempts, the problem of scheduling jobs to minimize the average flow-time on unrelated machines remained an important open problem. More surprisingly, the problem of minimizing the maximum flow-time, a natural generalization of the load balancing problem, also had no non-trivial approximation algorithms.

In this thesis we answer these questions by designing algorithms that approximate the optimal solution within logarithmic factors, and thus, resolving a long standing conjecture. Our algorithms and analyses use the iterated rounding technique on new linear programming relaxations of the problems that surmount the main technical hurdles faced by the previous approaches. Our results give a general framework based on the iterated round-
ing method for the analysis of flow-time objectives that seems applicable to several other problems.

We present these results in Chapters 2 and 3, and they are based on joint work with Nikhil Bansal (Bansal and Kulkarni, 2015).

1.1.2 Dual-Fitting Framework for Non-Clairvoyant Scheduling

In many practical scenarios it is difficult to get accurate knowledge of the processing requirements of jobs. Moreover, scheduling problems that arise in data-centers are also online in nature since we get to know the properties of a job only upon its arrival. From an engineering standpoint, online and non-clairvoyant algorithms are of fundamental importance. Hence, a key question was to design non-clairvoyant algorithms for minimizing the average weighted flow-time objective for the unrelated machine setting (Anand et al., 2012; Chadha et al., 2009; Devanur and Huang, 2014; Im et al., 2014a; Gupta et al., 2012b; Im et al., 2014a). In this thesis, we resolve this question and give an online algorithm that has an almost optimal competitive ratio in the resource augmentation model. Our work also introduces a unified LP-duality based approach to analyze non-clairvoyant scheduling problems that simplifies and generalizes all the previous works on the topic. Using our framework, we also design the first non-clairvoyant energy efficient scheduling algorithm for minimizing the weighted flow-time for unrelated machines.

In the last decade, we have witnessed the primal-dual and dual-fitting methods emerge as the main unifying principles in the design of online algorithms. Our work on non-clairvoyant scheduling takes another step in this direction. Interestingly, our framework also uses ideas from game theory; as we will see in the next paragraphs, the duality theory also seems to lie at the heart of analysis of games.

We cover these results in Chapters 4,5 and 6, and they are based on joint work with Sungjin Im, Kamesh Munagala and Kirk Pruhs (Im et al., 2014b).
1.1.3 Polytope Scheduling: Machine Scheduling Meets Markets

The classical scheduling models such as the unrelated machine model, do not capture the multidimensional nature of problems that arise in data centers or large clusters. In these modern computing setups, each job requires a vector of resources such as CPU, memory, bandwidth, etc., to execute. A scheduling decision at each time step consists of partitioning the available set of resources among the jobs that are alive. Given a partition of the resources to the jobs, each job executes at a rate which is some specified function of its allocation. The polytope scheduling problem (PSP), introduced in this thesis, captures the scheduling problems that arise in these multidimensional settings. The PSP problem encodes many well studied problems including unrelated machine scheduling, switch scheduling, broadcast scheduling etc., and hitherto unstudied scheduling problems such as routing multi-commodity flows, routing multicast (video-on-demand) trees, and multidimensional resource allocation. We design several competitive algorithms for the PSP problem and its special cases for the objective of minimizing the flow-time and completion time.

Our key contribution in these works is to view scheduling from an economic perspective. Our algorithm for the PSP problem is a widely used algorithm in the context of fairness in resource allocation problems, and is called Proportional Fairness (PF). PF computes a competitive equilibrium between the set of jobs and the resources. The analysis then proceeds by exploiting several interesting aspects of the competitive equilibrium such as existence of the market clearing prices and connections to the Eisenberg-Gale convex program and its dual. In hindsight, the market equilibrium concepts seem to give intuitive and systematic ways to approach design of algorithms for the multidimensional settings.

We cover these results in Chapters 7,8,9,10, and 11, and they are based on joint work with Sungjin Im and Kamesh Munagala (Im et al., 2014a). At the time of writing this thesis, a part of this work is still under a conference review.
1.1.4 Selfish Behavior, Price of Anarchy, and Duality

We also consider scheduling problems in game theoretic settings. The main assumption here is that the jobs are selfish agents and choose machines which give them the best utilities - that is, the machines which minimize their completion times. Here, our goal is to design distributed and local scheduling policies, also called coordination mechanisms, such that the PoA is minimized. Our main focus in these works is to minimize the average completion time of jobs. We define a notion of fairness called \textit{bounded stretch} and show that if an algorithm satisfies our condition, the PoA of the resulting game is small. We also generalize these results to more complex settings such as routing over graphs.

In the process of obtaining these results, we develop a new framework to analyze PoA using dual-fitting. The key idea in this approach is to exploit the properties of equilibrium outcomes to construct a dual solution that gives the bound on the PoA via the weak duality theorem. The dual-fitting technique seems to be a natural tool to analyze PoA of games, yet, before our work it was not used in the PoA literature. We show the wide applicability of our framework by giving alternate proofs of several important PoA results known in the literature such as those for congestion games, simultaneous second price auctions etc. Furthermore, using our framework, we also give the first coordination mechanisms with bounded PoA for temporal routing games over general graphs. We believe that these results hold promise for unifying the price of anarchy analysis with that of the rich literature on linear and convex programing based methods in approximation and online algorithms.

We present these results in Chapters 12, 13, and 14. Parts of this work are based on joint work with Sayan Bhattacharya, Sungjin Im, Kamesh Munagala and Vahab Mirrokni (Bhattacharya et al., 2014b,a; Kulkarni and Mirrokni, 2015).

1.2 Preliminaries

Before we present our results in full detail, we give a brief introduction to some the important (classical) scheduling models and problems considered in the literature. Please see
(Williamson and Shmoys, 2011; Vazirani, 2001; Borodin and El-Yaniv, 1997; Buchbinder and Naor, 2009; Pruhs et al., 2004; Karger et al., 2010; Leung et al., 2004; Im et al., 2011a; Nisan et al., 2007) for a more detailed introduction to approximation and online algorithms design, scheduling theory, and game theory.

1.2.1 Basics of Scheduling Theory

A typical scheduling instance consists of a set of machines $M$ and a set of jobs $J$. Each job $j \in J$ is characterized by its processing requirement, release time and weight. The processing requirement of a job denotes the length of time the job needs to execute before it completes if the job is allocated the entire machine without any interruptions. The release time of a job denotes the time at which the job is first available for processing; the importance or priority of a job $j$ is captured by the parameter $w_j$. Often, the processing requirement of a job depends on the machine on which it is executing. This dependence is captured by different machine environments.

1.2.2 Machine Environments

- **Single Machine Setting:** In this simplest model, there is only one machine and all the jobs need to be scheduled completely on the machine. A valid single machine schedule is simply any preemptive schedule which completes the processing requirements of all the jobs.

- **Identical Machine Setting:** In this case, there are multiple machines and each job has *same* processing requirement on all of them. In other words, processing length $p_j$, release time $r_j$ and weight $w_j$ of the job $j$ remains unchanged on all the machines in $M$.

- **Related Machine Setting:** Here, each machine $i \in M$ runs at a speed $s_i$, and can finish $s_i$ units of processing for unit time step. Therefore, a job $j$ takes $\frac{p_j}{s_i}$ time units to complete on the machine $i$. Observe that the related machine setting generalizes the identical machine case.
• Restricted Assignment Setting: In this machine environment, each job $j$ can be processed only on a subset of machines $S_j \subseteq M$. The processing length of the job, however, remains same on all the machines in the set $S_j$. The restricted assignment setting strictly generalizes the identical machine setting.

• Unrelated Machine Setting: In this very general model, each job $j$ has a machine dependent processing requirement $p_{ij}$. Clearly, this model generalizes all the above machine models.

Broadly speaking, a scheduling algorithm in multiple machine settings has two main components: 1) An assignment policy that determines the machine a job is assigned to. 2) A policy that determines the ordering of the jobs assigned to a machine. A schedule where a job can be interrupted and resumed at a later time without any loss in the work already completed and without incurring any cost are called preemptive schedules. Schedules where jobs cannot be preempted are called non-preemptive schedules. In this thesis we are mostly concerned with preemptive schedules. When there is more than one machine, we also differentiate between migratory and non-migratory schedules. In a migratory schedule, a job can be processed on multiple machines whereas in a non-migratory schedule a job needs to be processed completely on a single machine. Unless stated otherwise, we demand schedules to be non-migratory schedules.

1.2.3 Quality Of Service (QoS)

Scheduling problems have been studied with the objective of optimizing various QoS metrics. Below, we discuss some of the most important objective functions considered in the literature.

• Minimizing Makespan: In this problem, the objective is to assign jobs to machines such that completion time of the last job is as small as possible. Completion time of a job is defined as the earliest time instant at which the job is completely processed. In the single machine setting, the problem is trivial as any work conserving algorithm...
minimizes the makespan. Therefore, this problem is interesting only in multiple settings. Minimizing makespan problem is also referred to as load balancing problem.

- Minimizing Completion-Time: One of the most commonly studied objective functions is minimizing the average (weighted) completion time of jobs. If there is only one machine and if the release times of all jobs are 0, then, scheduling jobs in the decreasing order of their density, which is the ratio of processing length of a job to its weight, is an optimal algorithm. This widely used policy is also called Smith’s rule.

- Minimizing Flow-Time: This objective functions measures the delay seen by jobs. Flow-Time of a job is defined as the amount of time a job spends in the system, and is equal to completion time - release time. Two most popular objective functions concerning flow-time are minimizing the average (weighted) flow-time and minimizing the maximum flow-time. Note that minimizing the maximum flow-time generalizes the makespan minimization problem. In the single machine setting, Shortest Remaining Processing Time (SRPT) is an optimal algorithm for minimizing the total (average) flow-time of jobs.

- Deadline Scheduling: In these problems, each job has a processing length \( p_j \), a release time \( r_j \), and a deadline \( d_j \). The objective is assign jobs to machines, and schedule them such that each job finishes by its deadline. In a single machine setting, it is easy to verify that Earliest Deadline First (EDF) is an optimal algorithm. Another important objective function considered in deadline scheduling problems is to maximize the throughput. Note that deadline scheduling problem generalizes the problem of minimizing the maximum flow-time.

1.2.4 Popular Scheduling Policies

Here, we describe some well known scheduling policies. Fix a machine \( i \) which has to process a set of jobs \( Q_i \). Recall that \( p_{ij} \) denotes the processing length of job \( j \) on the machine \( i \). The symbol \( J_i(t) \) denotes the set of unfinished jobs that are available for processing at time
$t$. All the policies described below are simple, in the sense that they can be implemented in an “online” environment (defined in the next section). Thus, at any time $t$, the processing decision depends only on the jobs in $J_i(t)$, their weights, and remaining processing lengths. In particular, the decision is independent of the jobs that are going to arrive in future.

**Highest Density First (HDF).** At any time $t$, the machine works on the job $j \in J_i(t)$ which has the highest density $w_j/p_{ij}$. When the jobs are unweighted, this policy is known as “Shortest Job First” (SJF).

**Highest Residual Density First (HRDF).** At any time $t$, the machine works on the job $j \in J_i(t)$ which has the highest residual density $w_j/p_{ij}(t)$. When the jobs are unweighted, this policy is known as “Shortest Remaining Processing Time First” (SRPT).

**Weighted Round Robin (WRR).** At any time $t$, the machine works on the jobs in $J_i(t)$ in proportion to their weights. Specifically, for all jobs $j \in J_i(t)$, we have:

$$\frac{d}{dt} (p_{ij}(t)) = -\frac{w_j}{\sum_{j' \in J_i(t)} w_{j'}}.$$  

We note that WRR is an example of a *non-clairvoyant* scheduling policy. This refers to the property that a scheduling policy like WRR can be used even when the machine does not know the processing lengths in advance, and a job’s processing length is revealed only when the machine finishes the job.

**Weighted Shortest Elapsed Time First (WSETF).** At any time $t$, the machine works on the job $j \in J_i(t)$ which maximizes the ratio $w_j/(p_{ij} - p_{ij}(t))$. Unweighted version of this scheduling policy is known as “Shortest Elapsed Time First”, and here the machine works on the job which has been least processed so far. WSETF is a non-clairvoyant scheduling policy.
**Weighted Latest Arrival Processor Sharing (WLAPS(\(\epsilon\)).** This scheduling policy takes a parameter \(\epsilon \in [0, 1]\) as input. Let \(J_i^\epsilon(t)\) denote \(\epsilon |J(t)|\) jobs in \(J(t)\) with the highest release dates. At any time \(t\), the machine works on the jobs \(j \in J_i^\epsilon(t)\) in proportion to their weights, so that for all \(j \in J_i^\epsilon(t)\) we have:

\[
\frac{d}{dt}(p_{ij}(t)) = -\frac{w_j}{\sum_{j' \in J_i^\epsilon(t)} w_{j'}}.
\]

This scheduling policy gives preference to the recently released jobs, and is non-clairvoyant. Note that if \(\epsilon = 1\), then this policy reduces to **Weighted Round Robin**. When the jobs are unweighted, this policy is known as **Latest Arrival Processor Sharing (LAPS)**.

**Weighted Ranked Processor Sharing, WSLAPS(\(k\)).** In this thesis, we introduce a new scheduling policy called Weighted Ranked Processor Sharing. This policy is parametrized by an integer \(k\) (that we will later set to \(1/\epsilon\)) and \(\eta > 1\) that captures the speed augmentation. Fix some machine \(i\) and time instant \(t\). Recall that \(J_i(t)\) denotes the set of jobs assigned to this machine at time \(t\). Let \(\pi_j(t)\) denote the rank of \(j\) at time \(t\). The ranks of the jobs are input to the algorithm. Let \(W(i, t)\) denote their total weight, i.e.,

\[
W(i, t) = \sum_{j \in J_i(t)} w_j.
\]

Let \(S_j(t)\) denote the set of jobs with the ranks less than the rank of job \(j\); That is,

\[
S_j(t) = \{j' \mid j', j \in J_i(t) \text{ and } \pi_{j'}(t) < \pi_j(t)\}
\]

Let \(W_j(t) = \sum_{j' \in S_j(t)} w_{j'}\) denote the total weight of jobs in the set \(S_j(t)\). At time instant \(t\), the total processing rate of the machine \(i\) is divided among the jobs in \(J_i(t)\) as follows. Job \(j \in J_i(t)\) is assigned processing power \(\nu_j(t)\) as follows:

\[
\nu_j(t) := \eta \cdot \frac{(W_j(t) + w_j)^{k+1} - W_j^{k+1}}{W(i, t)^{k+1}}
\]

The rate at which job \(j \in J_i(t)\) is processed at time \(t\) is therefore \(\ell_{ij} \nu_j(t)\). Note that \(\sum_{j \in J_i(t)} \nu_j(t) = \eta\) at all time instants \(t\) and for all machines \(i\). Note that if \(k = 0\), this is
exactly weighted round robin. As \( k \) becomes larger, this gives higher rate to jobs in the tail of the queue, taking the weights \( w_j \) into account.

Observe that the algorithms WSLAPS\((k)\) and WLAPS\((\epsilon)\), unlike HDF or RR, take into account the speed augmentation. This will become more clear in the Chapter 4.

1.2.5 Measuring Performance: Approximation Factor, Competitive Ratio

Most of the optimization problems considered in this thesis are NP-hard. Hence, we adopt the standard notions of approximation factor and competitive ratio to compare the performance of our algorithms. An algorithm is \textit{offline} if the decisions taken by the algorithm rely on knowing the \textit{entire input} in advance. However, in several applications input is revealed one step at-a-time, hence, it is desirable to have algorithms that are online; that is, algorithms which make \textit{irrevocable} decisions only looking at the current input. All the scheduling algorithms described above are online.

Consider a cost minimization problem. An offline algorithm for the problem has an \textit{approximation factor} of \( \alpha \), if for every input instance \( I \), the cost incurred by the algorithm is at most \( \alpha \) times the cost of optimal solution to the instance. We call \( \alpha \) the \textit{competitive ratio} of the algorithm if the algorithm is \textit{online}. However, for many problems there are strong lowerbounds on the competitive ratios (or approximation factors) any randomized algorithm can achieve. For such problems we do the \textit{resource augmentation} analysis. In the resource augmentation analysis, we assume that our algorithm has slightly more power than the optimal algorithm. (An equivalent way to think about the resource augmentation analysis is to assume that the optimal solution has more constraints than the online algorithm.) To give an example, consider the problem of minimizing weighted flow-time on a single machine. For this problem, it is easy to show that the scheduling algorithm Highest Density First has unbounded competitive ratio. We can prove, however, that HDF is \((1 + \epsilon)\)-speed \( O(\frac{1}{\epsilon}) \)-competitive. The result is interpreted as follows: if the HDF algorithm is run on a machine which can process \((1 + \epsilon)\)-units of jobs for each time instant and the offline optimum is only allowed process 1 unit of job per time step, then the cost incurred
by HDF is at most $O\left(\frac{1}{\epsilon}\right)$ times the cost of optimal solution for every input instance. Note that the algorithm HDF itself does not explicitly take into account the resource augmentation. This is true for many problems we consider in this thesis; in most cases, resource augmentation is purely an analysis tool.

### 1.2.6 Measuring Performance Amidst Selfish Behavior: Price of Anarchy (PoA)

We also study scheduling models that incorporate strategic and selfish behavior observed in real world applications. In these cases, we adopt game theoretic concepts to measure the performance of our algorithms amidst selfish behavior. A canonical problem we consider in this setting is as follows: Consider the problem of minimizing the average completion of times jobs on unrelated machines. Although several constant factor approximation algorithms are known for this problem, all the algorithms assume that a centralized scheduler can make the both the decisions of assigning jobs to machines and the order in which jobs are processed on each machine. In many distributed applications, however, it is not possible to have such centralized schedulers. In contrast with the centralized view of classical scheduling models, selfish scheduling models assume that each job is a self-interested and autonomous agent free to select its own machine. Every machine declares its scheduling policy in advance, and this induces a simultaneous-move game between the jobs. The strategy of a job consists of choosing the machine where it will get processed. Each job wants to minimize its own disutility, which is the completion time of the job. A Nash equilibrium of this game is a stable outcome where no job can reduce its disutility by switching to another machine. A standard assumption made in game theoretic settings is that selfish behavior converges to an equilibrium outcome (not necessarily pure NE). Thus, the goal of a system designer is to design a scheduling policy on each machine (such as HDF, Round Robin etc), such that inefficiency resulting from the selfish behavior is as small as possible.

We adopt the Price of Anarchy framework to measure the degradation in the quality of schedules due to selfish behavior. The price of anarchy measures the worst case ratio of the cost of a solution in an equilibrium outcome to the optimal solution. We briefly discuss
some of the equilibrium concepts we consider in this thesis.

Let us first set up some notation. Consider a cost minimization game $G$ with $N$ players and each player $j \in N$ having a strategy space $S_j$. Let $\theta = (\theta_1, \ldots, \theta_n)$, where $\theta_j \in S_j$, denote a strategy profile of strategies taken by the players. We refer to a strategy profile $\theta$ as an outcome.

- **Pure NE:** An outcome $\theta$ is in pure NE if for every $j \in N$ and for all $i \in S_j$,
  $$\text{Cost}_j(\theta) \leq \text{Cost}_j(i, \theta_{-j}).$$

- **Mixed NE:** A set of independent probability distributions $(\sigma_1, \sigma_2, \ldots, \sigma_n)$ over strategy sets of players is a mixed Nash equilibrium of a game if for all $j \in N$, and for all $i \in S_j$,
  $$\mathbb{E}_{\theta \sim \sigma} [\text{Cost}_j(\theta)] \leq \mathbb{E}_{\theta_{-j} \sim \sigma_{-j}} [\text{Cost}_j(i, \theta_{-j})].$$
  Here, $\sigma$ denotes the product distribution. The concept of mixed NE strictly generalizes the pure NE.

- **Correlated Equilibrium:** A joint probability distribution $\sigma$ over the outcomes of a game is said to be a correlated equilibrium, if for all $j \in N$, and for all $i, i' \in S_j$,
  $$\mathbb{E}_{\theta \sim \sigma} [\text{Cost}_j(\theta)|i] \leq \mathbb{E}_{\theta \sim \sigma} [\text{Cost}_j(i', \theta_{-j})|i].$$
  Correlated equilibrium is a strict generalization of mixed NE.

- **Coarse Correlated Equilibrium:** A further relaxation of correlated equilibrium is coarse correlated equilibrium. A distribution $\sigma$ over outcomes of a game is a coarse correlated equilibrium (CCE) if for all $j \in N$, and for all $i \in S_j$,
  $$\mathbb{E}_{\theta \sim \sigma} [\text{Cost}_j(\theta)] \leq \mathbb{E}_{\theta \sim \sigma} [\text{Cost}_j(i, \theta_{-j})].$$
  In the context of repeated games, a CCE corresponds to the limiting distribution of a sequence of plays that are no regret for every player.

Finally, the robust PoA is the worst case ratio of expected social cost of a distribution in CCE to the optimal solution to the instance (in non-strategic settings). Let $OPT$ denote the optimal solution to the instance $(G, N, \cup_j S_j)$. Then, Robust PoA =
$$\max_{\sigma} \frac{\sum_j \mathbb{E}_{\theta \sim \sigma} [\text{Cost}_j(\theta)]}{OPT},$$
where $\sigma$ is in CCE.
1.3 Road Map

The thesis is organized into four main parts. In the first part (Chapters 2-4) we consider the problem of minimizing the flow-time on unrelated machines in both offline and online settings. In the second part (Chapters 5-6), we design energy efficient algorithms for the minimizing the weighted flow-time. We study the polytope scheduling problems and its various applications in the third part of thesis (Chapters 7-11). The last part of the thesis (Chapters 12-14) is devoted to exploring the game theoretic aspects of scheduling, and applications of dual fitting technique for PoA analysis.
PART I

Classical Scheduling Problems
Approximation Algorithms For Minimizing Total Flow-Time

2.1 Introduction

Many computer architects believe that architectures consisting of heterogeneous processors will be the dominant architectural design in the future: Simulation studies indicate that, for a given area and power budget, heterogeneous multiprocessors can offer an order of magnitude better performance for typical workloads (Bower et al., 2008; Kumar et al., 2006; Merritt, 2010; Koufaty et al., 2010). Looking at the consequences of Moore’s Law even further in the future, some computer architectures are projecting that we will transition from the current era of multiprocessor scaling to an era of “dark silicon”, in which switches become so dense that it is not economically feasible to cool the chip if all switches are simultaneously powered. (Esmaeilzadeh et al., 2011). One possible architecture in the dark silicon era would be many specialized processors, each designed for a particular type of job. The processors that are on any point of time should be those that are best suited for the current tasks. Hence, in modern computing environments, be it web-servers, data-centers, clusters of machines, or personal computers, heterogeneity of the processors and architectures is ubiquitous. The most general and widely studied model that incorporates the heterogeneity of jobs and machines is the so-called unrelated machines setting. Here,
there is a set $J$ of $n$ jobs and a set $M$ of $m$ machines. Each job $j$ is specified by its release time (or arrival time) $r_j$, which is the first time instant it is available for processing, and a machine-dependent processing requirement $p_{ij}$, which is the time taken to process $j$ on machine $i$.

Besides the practical motivation, exploring basic scheduling problems in the unrelated machines setting has also led to the development of several fundamental techniques in algorithm design, for example (Lenstra et al., 1990; Skutella, 2001; Bansal and Sviridenko, 2006; Chakrabarty et al., 2009; Svensson, 2012; Garg, 2009; Anand et al., 2012). However, one such problem that has long resisted non-trivial approximation ratios, despite several efforts (Garg and Kumar, 2006b,a; Sitters, 2008), is the problem of minimizing the total flow-time on unrelated machines. The flow-time of a job, defined as the amount of time the job spends in the system, is one of the most natural measures of quality of service, and is also sometimes referred to as response time or sojourn time. More precisely, if a job $j$ completes its processing at time $C_j$, then flow-time of the job $F_j$ is defined $C_j - r_j$; i.e., its completion time minus arrival time.

2.1.1 Our Results

In this chapter, we consider a natural problem of minimizing total flow-time on unrelated machines. The problem has been studied quite extensively (as we discuss below). Yet, all these results hold in more restricted settings and obtaining a non-trivial approximation algorithm in the general unrelated machines setting was a key open question Garg and Kumar (2006a); Sitters (2008). In this chapter we will give the first polylogarithmic approximation algorithm to the problem. Our main result is:

**Theorem 1.** There exists a polynomial time $O(\log n \cdot \log P)$-approximation algorithm for minimizing the total flow time in the unrelated machine setting.

Using a standard trick this implies an $O(\log^2 n)$ approximation, which may be better if $P$ is super-polynomial in $n$. An approximation hardness of $\Omega(\log P)$ is also known for the problem even in the much simpler setting of identical machines (Garg et al., 2008).
Our algorithm is based on applying the iterated rounding technique and is quite different from the previous approaches to the problem. The key idea is to write a new time-indexed linear programming (LP) formulation for the problem. The formulation we consider is different from those considered previously, and has much fewer constraints than the standard time-indexed LP formulation. Having fewer constraints is crucial in being able to use of iterated rounding. We describe the new formulation and give an overview of the algorithm in section 4.4. Theorem 1 is proved in section 4.4.

2.1.2 History

Scheduling to minimize flow-time has been extensively studied in the literature under various different models and objective functions and we only describe the work that is most relevant to our results. A more comprehensive survey of various flow-time related results can be found in (Garg, 2009; Im et al., 2011a; Pruhs et al., 2004).

Single Machine. Both total flow-time and maximum flow-time are well understood in the single machine case. The SRPT (Shortest Remaining Processing Time) algorithm is optimal for total flow-time if preemption is allowed, that is, when a job can be interrupted arbitrarily and resumed later from the point of interruption. Without preemptions, the problem becomes hard to approximate within $O(n^{1/2-\alpha(1)})$ (Kellerer et al., 1996). We will consider only preemptive algorithms in this chapter.

Multiple Machines. For multiple machines, various different settings have been studied. The simplest is the identical machines setting, where a job has identical size on each machine ($p_{ij} = p_j$ for all $i$). A more general model is the related machines setting, where machine $i$ has speed $s_i$ and job $j$ has size $p_j$ ($p_{ij} = p_j/s_i$). Another model is the restricted assignment setting, where a job $j$ has a fixed size, but it can only be processed on a subset $S_j$ of machines ($p_{ij} \in \{p_j, \infty\}$). Clearly, all these are special cases of the unrelated machines setting. As in most previous works, we will consider the non-migratory setting where a job must be executed on a single machine.
Leonardi and Raz (2007) obtained the first poly-logarithmic guarantee for identical machines and showed that SRPT is an $\Theta(\log(\min(\frac{n}{m}, P)))$ competitive algorithm. Subsequently, other algorithms with similar competitive ratio, but other desirable properties such as no-migration and immediate dispatch were also obtained in Awerbuch et al. (2002); Avrahami and Azar (2003). Later, polylogarithmic offline and online guarantees were obtained for the related machines setting Garg and Kumar (2006b,a). As mentioned previously, an $\Omega(\log^{1-\epsilon} P)$ hardness of approximation is known even for identical machines Garg et al. (2008). The above approaches do not work for the restricted assignment case, which is much harder. In an important breakthrough, Garg and Kumar (2007) gave a $O(\log P)$ approximation, based on an elegant and non-trivial LP rounding approach. They consider a natural LP relaxation of the problem, and round it based on computing certain unsplittable flows Dinitz et al. (1998) on an appropriately defined graph.

To extend these ideas to the unrelated machines case, (Garg et al., 2008) introduce a $(\alpha, \beta)$-variability setting (see (Garg et al., 2008) for details) and prove a general result that implies logarithmic approximations for both restricted assignment and related machines setting. For the unrelated setting, their result gives an $O(k)$ approximation where $k$ is the number of different possible values of $p_{ij}$ in the instance. Sitters (2008) also independently obtained a similar result using different techniques. In general however these guarantees are polynomial in $n$ and $m$.

Interestingly, with job weights, approximating total weighted flow-time is $n^{\Omega(1)}$-hard even for identical machines Chekuri and Khanna (2002). However, several interesting results are known for this measure in the resource augmentation setting Chekuri et al. (2004); Chadha et al. (2009); Anand et al. (2012). In this chapter we only consider the unweighted setting.

2.2 Alternate LP Relaxation and The High-Level Idea

Before describing the new LP formulation that we use, we first describe the standard time-indexed linear programming relaxation for the problem that was used, for example in Garg
and Kumar (2007); Garg et al. (2008).

**Standard LP formulation:** There is a variable $x_{ijt}$ for each machine $i \in [m]$, each job $j \in [n]$ and each unit time slot $t \geq r_j$. The $x_{ijt}$ variables indicate the amount to which a job $j$ is processed on machine $i$ during the time slot $t$. The first set of constraints (service constraints) says that every job must be completely processed. The second set of constraints (capacity constraints) enforces that a machine cannot process more than one unit of job during any time slot. Note that this LP allows a job to be processed on multiple machines, and even at the same time.

$$\min \sum_{i,j,t} \left( \frac{t - r_j}{p_{ij}} + \frac{1}{2} \right) \cdot x_{ijt}$$

s.t. (1) $\sum_i \sum_{t \geq r_j} \frac{x_{ijt}}{p_{ij}} \geq 1 \ \forall j$,
(2) $\sum_{j : t \geq r_j} x_{ijt} \leq 1 \ \forall i, t$, and $x_{ijt} \geq 0 \ \forall i, j, t \geq 0$

**Fractional flow-time:** The objective function needs explanation. The term $\sum_{i,t} x_{ijt}$ is the total amount of processing done on job $j$. The term $\sum_{i,t} (t - r_j) \cdot \frac{x_{ijt}}{p_{ij}}$ is the fractional flow-time of job $j$ and we denote it by $f_j$. Recall that the (integral) flow-time of a job $j$ can be viewed as summing up 1 over each time step that $j$ is alive, i.e. $\sum_{t \geq r_j} 1_{(j \text{ is alive at } t)}$. Similarly, the fractional flow-time is the sum over time of the remaining fraction of job $j$.

On machine $i$, the fraction of job $j$ unfinished at time $t$ is $\sum_{t' > t} \frac{x_{ijt'}}{p_{ij}}$ (the numerator is the work done on $j$ on machine $i$ after $t$). Thus the fractional flow-time on machine $i$ is $\sum_{t \geq r_j} \sum_{t' > t} \frac{x_{ijt'}}{p_{ij}}$, which can be written as $\sum_t (t - r_j) \cdot \frac{x_{ijt}}{p_{ij}}$. Note that the integral flow-time is at least the fractional flow-time plus half the size of a job, and thus the objective function in the LP above is valid lowerbound on flow-time. For more details on the LP above, see Garg and Kumar (2007).

We assume that $\min_{i,j} p_{ij} \neq 0$ (otherwise $j$ can be scheduled on machine $i$ right upon arrival), and hence by scaling we assume henceforth that $\min_{i,j} p_{ij} = 1$. Define $P = \max_{i,j} p_{ij} / \min_{i,j} p_{ij}$. For $k = 0, 1, \ldots, \log P$, we say that a job $j$ belongs to class $k$ on machine $i$ if $p_{ij} \in (2^{k-1}, 2^k]$. Note that the class of a job depends on the machine.
We now describe the new LP relaxation for the problem. The main idea is to ignore the capacity constraints (1) at each time slot, and instead only enforce them over carefully chosen intervals of time. Even though the number of constraints is fewer, as we will see, the quality of the relaxation is not sacrificed much.

**New LP formulation:** There is a variable \( y_{ijt} \) (similar to \( x_{ijt} \) before) that denotes the total units of job \( j \) processed on machine \( i \) at time \( t \). However, unlike the time-indexed relaxation, we allow \( y_{ijt} \) to take values greater than one. In fact, we will round the new LP in such a way that eventually \( y_{ijt} = p_{ij} \) for each job, which will have a natural interpretation that job \( j \) is scheduled at time \( t \) on machine \( i \).

For each class \( k \) and each machine \( i \), we partition the time horizon \([0, T]\) into intervals of size \( 4 \cdot 2^k \). Without loss of generality we can assume that \( T \leq nP \) (otherwise the input instance can be trivially split into two disjoint non-overlapping instances). For \( a = 1, 2, \ldots \), let \( I(i, a, k) \) denote the \( a \)-th interval of class \( k \) on machine \( i \). That is, \( I(i, 1, k) \) is the time interval \([0, 4 \cdot 2^k]\) and \( I(i, a, k) = ((4 \cdot 2^k)(a-1), (4 \cdot 2^k) a] \). We write the new LP relaxation.

\[
\begin{align*}
\min & \sum_i \sum_j \sum_k \sum_{t \geq r_j} \sum_{j \in [2^k-1, 2^k]} \left( \frac{t-r_j}{p_{ij}} + \frac{1}{2} \right) \cdot y_{ijt} \\
\text{s.t.} & \sum_i \sum_j \sum_{t \geq r_j} y_{ijt} / p_{ij} \geq 1 \quad \forall j \tag{2.1} \\
& \sum_j \sum_{t : p_{ij} \leq 2^k} y_{ijt} \leq \text{Size}(I(i, a, k)) \quad \forall i, k, a \tag{2.2} \\
& y_{ijt} \geq 0 \quad \forall i, j, t : t \geq r_j 
\end{align*}
\]

Here, \( \text{Size}(I(i, a, k)) \) denotes the size of the interval \( I(i, a, k) \) which is \( 4 \cdot 2^k \) (but would change in later iterations of the LP when we apply iterated rounding). Observe that in (2.2) only jobs of class \( \leq k \) contribute to the left hand side of constraints corresponding to intervals of class \( k \).

Clearly, \((\text{LP}_{\text{new}})\) is a relaxation of the time indexed LP formulation considered above, as any valid solution there is also a valid solution to \((\text{LP}_{\text{new}})\) (by setting \( y_{ijt} = x_{ijt} \)). Therefore,
we conclude that an optimum solution to \((\text{LP}_{\text{new}})\) lower bounds the value of an optimal solution.

**Remark:** When we apply iterative rounding and consider subsequent rounds, we will refer the intervals \(I(i, a, k)\) as \(I(i, a, k, 0)\).

**The high-level approach:** The main idea of our algorithm is the following. Let us call a job \(j\) to be *integrally assigned* to machine \(i\) at time \(t\), if \(y_{ijt} = p_{ij}\) (note that this job will be completely executed on machine \(i\)). Let us view this as processing the job \(j\) during \([t, t+p_{ij}]\). In the algorithm, we first find a tentative integral assignment of jobs to machines (at certain times) such that the total flow-time of this solution is at most the LP value. This solution is tentative in the sense that multiple jobs could use the same time slot; however we will ensure that the effect of this overlap is negligible (in the sense of Lemma 2 below). More precisely, we show the following result.

**Lemma 2.** There exists a solution \(y^* = \{y^*_{ijt}\}_{i,j,t}\) satisfying the following properties.

- **(Integrality:)** For each job \(j\), there is exactly one non-zero variable \(y_{ijt}^*\) in \(y^*\), which takes value \(p_{ij}\). That is, each job is assigned integrally to exactly one machine, and one time slot: \(y^*_{ijt} = p_{ij}\).

- **(Low cost:)** The cost of \(y^*\) is at most the cost of an optimal solution to \(\text{LP}_{\text{new}}\).

- **(Low overload:)** For any interval of time \([t_1, t_2]\), every machine \(i\) and for every class \(k\),

\[
\sum_{j: p_{ij} \leq 2^k} \sum_{t \in [t_1, t_2]} y^*_{ijt} \leq (t_2 - t_1) + O(\log n) \cdot 2^k.
\]

That is, the total size of jobs of class at most \(k\) assigned integrally in any time interval \([t_1, t_2]\) exceeds the size of the interval by at most \(O(\log n) \cdot 2^k\).

Lemma 2 is the core of our algorithm, which will be proved using iterated rounding. In particular, we show using a counting argument that in each round a basic feasible optimum solution assigns at least a constant fraction of jobs integrally in each round. Therefore,
after $O(\log n)$ rounds every job is integrally assigned to some machine. In each round as some jobs get integrally assigned, we will fix them permanently and reduce the free space available in those intervals. Then, we merge these intervals greedily to ensure that the free space in an interval corresponding to class $k$ stays $O(1) \cdot 2^k$. This merging process adds an overload of at most $O(1) \cdot 2^k$ to any time interval in each round. This ensures that the total error added for any time interval is $O(\log n) \cdot 2^k$.

The next step is to show that the tentative schedule can be converted to a valid preemptive schedule by increasing the total flow-time of jobs by $O(\log P \log n)$ times the LP$_{new}$ value. To this end, we use ideas similar to those used by Garg and Kumar (2006a, 2007) for the related or restricted machines case. In particular, we schedule the jobs on each machine in the order given by the tentative schedule, while prioritizing the jobs in the shortest job first (SJF) order. The low overload property of the tentative schedule ensures that a job of class $k$ is additionally delayed by at most $O(\log n) \cdot 2^k$ due to jobs that arrive before it, or is delayed by smaller jobs (of strictly lower class) that arrive after the time when it is tentatively scheduled. In either case, we show that this delay can be charged to the total flow-time of other jobs.

2.3 Tentative Schedule to Actual Schedule

We show how Theorem 1 follows given a solution $y^*$ satisfying the conditions of Lemma 2. Recall that in the solution $y^*$, for each job $j$, we have $y_{ijt} = p_{ij}$ for some time instant $t$ and some machine $i$, but this is not necessarily a valid schedule. We convert $y^*$ into a valid preemptive schedule $S$ as follows. Fix a machine $i$ and let $J(i, y^*)$ denote the set of jobs which are scheduled on machine $i$ in the solution $y^*$ (i.e. jobs $j$ such that $y_{ijt} = p_{ij}$ for some time instant $t$). In the schedule $S$, for each machine $i$, we imagine that a job $j$ in $J(i, y^*)$ becomes available for $S$ at the time $t$ where $y_{ijt} = p_{ij}$. We schedule the jobs in $S$ (after they become available) using Shortest Job First (SJF) (where jobs in the same class are viewed as having the same size); for two jobs belonging to same class we schedule the jobs in the order given by $y^*$. Let $J_k(i, S)$ denote the set of jobs of class $k$ which are
assigned to machine $i$ in schedule $S$, and let $J(i, S) = \cup_k J_k(i, S)$ denote the set of jobs scheduled by $S$ on $i$. Clearly, $J_k(i, S) = J_k(i, y^*)$. We also observe that, since jobs within a class are considered in order, for each class $k$ and on each machine $i$, there is at most one job belonging to class $k$ which is partially processed (due to preemptions by jobs of a smaller class). This directly implies the following relation between the fractional and integral flow-time of jobs in $S$. Let $F_j^S$ denote the flow-time of job $j$ in schedule $S$ and $f_j^S$ denote the fractional flow-time.

**Lemma 3.** Fix a machine $i$ and the set of jobs belonging to class $k$. Then,

$$
\sum_{j \in J_k(i, S)} F_j^S \leq \sum_{j \in J_k(i, S)} f_j^S + \sum_{j \in J(i, S)} p_{ij}.
$$

**Remark:** Note that first two summations are over $J_k(i, S)$, while the third summation is over $J(i, S)$.

**Proof.** We use the alternate view of integral and fractional flow-times. Let $C_j^S$ denote the completion time of job $j$ in the schedule $S$. Then, the integral flow-time of $j$ is $F_j^S = \int_{t=r_j}^{C_j^S} 1 \cdot dt$ and the fractional flow-time is $f_j^S = \int_{t=r_j}^{C_j^S} p_{ij}(t)/p_{ij} dt$, where $p_{ij}(t)$ denotes the remaining processing time of job $j$ on machine $i$.

Let $J_k(i, S, t)$ denote the set of jobs available for processing at time $t$ of class $k$ on machine $i$ in $S$, which have not been completed, and $T(i, k)$ denote the set of time instants where $J_k(i, S, t) \geq 1$, i.e. at least one job of class $k$ is alive. Then,

$$
\sum_{j \in J_k(i, S)} F_j^S = \int_{t \in T(i, k)} |J_k(i, S, t)| dt \leq \int_{t \in T(i, k)} \left( 1 + \sum_{j \in J_k(i, S)} \frac{p_{ij}(t)}{p_{ij}} \right) dt
$$

$$
\leq \sum_{j \in J(i, S)} p_{ij} + \sum_{j \in J_k(i, S)} f_j^S.
$$

The first inequality follows as there is at most one partially processed job of class $k$ at
any time in \( S \). The second inequality follows by observing that \( \int_{t \in \mathcal{T}(i,k)} 1 dt \) is simply the time units when at least one class \( k \) job is alive. This can be at most the time when any job (of any class) is alive, which is precisely equal to \( \sum_{j \in J(i,S)} p_{ij} \), the total processing done on machine \( i \) (as the schedule \( S \) is never idle if there is work to be done). Thus, \( \int_{t \in \mathcal{T}(i,k)} 1 dt \leq \sum_{j \in J(i,S)} p_{ij} \). Moreover, \( \int_{t \in \mathcal{T}(i,k)} \sum_{j \in J_k(i,S)} \frac{p_{ij}(t)}{p_{ij}} dt = \sum_{j \in J_k(i,S)} \int_{t \geq r_j} \frac{p_{ij}(t)}{p_{ij}} dt \) which is exactly the total fractional flow-time \( \sum_{j \in J_k(i,S)} f_j^S \).

Let \( V_k(y^*, i, t) \) denote the total remaining processing time (or volume) of jobs of class \( k \) alive at time \( t \) on machine \( i \) in the schedule defined by \( y^* \) (i.e. these are precisely the jobs that are released but not yet scheduled by \( t \)); similarly, let \( V_k(S, i, t) \) denote the total remaining processing time of jobs of class \( k \) that have \( r_j \leq t \), but are unfinished at time \( t \) on machine \( i \) in the schedule \( S \). As a job is available for \( S \) only after it is scheduled in \( y^* \), we make the following simple observation.

**Observation 1.** For any \( k \), \( V_k(y^*, i, t) \leq V_k(S, i, t) \). Moreover, \( V_k(S, i, t) - V_k(y^*, i, t) \) is the volume of precisely those jobs of class \( k \) that are available to \( S \) (i.e. already scheduled in \( y^* \)), but have not been completed by \( S \).

Using the above observation we show that \( V_k(y^*, i, t) \) and \( V_k(S, i, t) \) do not deviate by too much, which is very crucial for our analysis.

**Lemma 4.** For machine \( i \) and class \( k \), \( \forall t, V_k(S, i, t) - V_k(y^*, i, t) \leq O(\log n) \cdot 2^k \)

**Proof.** By Observation 1, \( V_k(S, i, t) - V_k(y^*, i, t) \) is the total processing time of jobs of class \( k \) that are available for processing in \( S \) at time \( t \) and not yet completed. As \( V_k(S, i, t) - V_k(y^*, i, t) \leq V_{\leq k}(S, i, t) - V_{\leq k}(y^*, i, t) \) (this follows by Observation 1 as \( V_{k'}(S, i, t) \geq V_{k'}(y^*, i, t) \) for each \( k' \)), it suffices to bound the latter difference. Let \( t' \leq t \) be the last time before \( t \) when machine \( i \) was idle in \( S \), or was processing a job of class strictly greater than \( k \). This means that no jobs of class \( \leq k \) are available to \( S \) (as they have either not arrived or have not yet been made available by \( y^* \)). Thus, \( V_{\leq k}(S, i, t') = V_{\leq k}(y^*, i, t') \) or equivalently \( V_{\leq k}(S, i, t') - V_{\leq k}(y^*, i, t') = 0 \). By the low overload property, the total
volume of jobs belonging to class at most \( k \) that becomes available during \((t', t]\) is at most \((t - t') + O(\log n)2^k\). Since \( S \) processes only jobs of class at most \( k \) during \((t', t]\) (by definition of \( t' \)), \( S \) completes precisely \((t - t')\) volume of jobs belonging to class at most \( k \). This implies \( V_{\leq k}(S, i, t) - V_{\leq k}(y^*, i, t) = O(\log n)2^k \).

We are now ready to show how this implies Theorem 1

**Proof of Theorem 1.** We first compare the fractional flow-times of schedules defined by \( y^* \) and \( S \) and then use Lemma 3 to complete the argument.

Define \( y^S_{ijt} \) variables corresponding to the schedule \( S \) by setting \( y^S_{ijt} \) to amount of processing done on job \( j \) on machine \( i \) at time \( t \) in the schedule \( S \). Let \( P(S, i) = \sum_{j \in I(i, S)} \sum_{t} y^S_{ijt} \) denote the total processing time of the jobs scheduled on machine \( i \) in \( S \).

 Clearly, since the set of jobs on machine \( i \) in \( y^* \) and \( S \) is identical, we have \( P(S, i) = P(y^*, i) \).

Let \( T(i, k) \) be the times when there is at least one available but unfinished job in \( S \). Recall that \( \int_{t \in T(i, k)} 1 \cdot dt = P(i, S) \).

Then, the difference between the fractional flow-times of jobs in \( S \) and \( y^* \) can be bounded by

\[
\sum_j (f^S_j - f^*_j) = \sum_i \sum_t \sum_k \sum_{j: p_{ij} \in (2^{k-1}, 2^k]} (y^S_{ijt} - y^*_{ijt}) \cdot \left( \frac{t - r_j}{p_{ij}} \right)
\]

\[
\leq \sum_i \sum_t \sum_k \sum_{j: p_{ij} \in (2^{k-1}, 2^k]} (y^S_{ijt} - y^*_{ijt}) \cdot \left( \frac{t - r_j}{2^{k-1}} \right)
\]

\[
= \sum_i \sum_t \sum_k \sum_{j: p_{ij} \in (2^{k-1}, 2^k]} \frac{1}{2^{k-1}} (V_k(S, i, t) - V_k(y^*, i, t)) \tag{2.3}
\]

\[
\leq \sum_i \sum_k \sum_{t \in T(i, k)} O(\log n) = \sum_i \sum_k O(\log n)P(i, S) \text{ [Lemma (4)]}
\]

\[
\leq \sum_i O(\log n \cdot \log P)P(i, S) = O(\log n \cdot \log P)P(S)
\]

Here (2.3) follows as for any schedule \( S \), the quantity \( \sum_{j: p_{ij} \in (2^{k-1}, 2^k]} \sum_{t \geq r_j} y^S_{ijt} \cdot (t - r_j) \) is
exactly equal to $\sum V_k(S, i, t)$ (by the two different ways of looking at fractional flow-time). Next, we can bound the total flow-time as

$$\sum_j F_j^S = \sum_i \sum_k \sum_{j \in J_k(i, S)} F_j^S \leq \sum_i \sum_k \left( \sum_{j \in J_k(i, S)} f_j^S + \sum_{j \in J(i, S)} p_{ij} \right)$$

which is at most $O(\log n \cdot \log P)$ times the value of optimal solution to LP$_{new}$.

We show an approximation guarantee of $\min\left( O(\log^2 n), O(\log n \cdot \log P) \right)$ at the last section of this chapter.

2.4 Iterated Rounding of LP$_{new}$

In this section we prove the Lemma 2 using iterated rounding. In the iterated rounding technique, we successively relax the LP$_{new}$ with a sequence of linear programs, each having fewer constraints than the previous one while ensuring that optimal solutions to the linear programs is at most the cost of optimal solution to LP$_{new}$. An excellent reference for various applications of this technique is Lau et al. (2011).

We denote the successive relaxations of LP$_{new}$ by LP$\ell$ for $\ell = 0, 1, \ldots$. Let $J(\ell)$ denote the set of jobs that appear in LP$\ell$. Linear program LP$0$ is same as LP$_{new}$, and $J(0) = J$. We define LP$\ell$ for $\ell > 0$ inductively as follows.

- **Computing a basic optimal solution:** Find a basic optimal solution $y^*(\ell - 1) = \{y^\ell_{ijt} \}_{i,j,t}$ to LP$\ell - 1$. We use $y^\ell_{ijt}$ to indicate the value taken by the variable $y_{ijt}$ in the solution $y^*(\ell - 1)$. Let $S_{\ell - 1}$ be the set of variables in the support of $y^*(\ell - 1)$. We initialize $J(\ell) = J(\ell - 1)$.

- **Eliminating 0-variables:** The variables $y_{ijt}$ for LP$\ell$ are defined only for the variables in $S_{\ell - 1}$. That is, if $y^\ell_{ijt} = 0$ in $y^*(\ell - 1)$, then these variables are fixed to 0 forever,
and do not appear in $LP(\ell)$.

- **Fixing integral assignments:** If a variable $y_{ijt}^{\ell-1} = p_{ij}$ in $y^* (\ell - 1)$ for some job $j$, then $j$ is permanently assigned to machine $i$ at time $t$ in $y^*$ (as required by Lemma 2), and we update $J(\ell) = J(\ell) \setminus \{j\}$. We drop all the variables corresponding to the job $j$ in $LP(\ell)$, and also drop the service constraint (2.5) for the job $j$. We use $A(\ell - 1)$ to denote the set of jobs which get integrally assigned in $(\ell - 1)$-th iteration. We redefine the intervals based on the unassigned jobs next.

**Remark:** It will be convenient below not to view an interval as being defined by its start and end times, but by the $y_{ijt}$-variables it contains.

- **Defining intervals for $\ell$-th iteration:** Fix a class $k$ and machine $i$. We define the new intervals $I(i, *, k, \ell)$ and their sizes as follows.

Consider the jobs in $J(\ell)$ (those not yet integrally assigned) belonging to classes $\leq k$, and order the variables $y_{ijt}$ in increasing order of $t$ (in case of ties, order them lexicographically). Greedily group consecutive $y_{ijt}$ variables (starting from the beginning) such that sum of the $y_{ijt}^{\ell-1}$ values of the variables in that group first exceeds $4 \cdot 2^k$.

Each such group will be an interval (which we view as a subset of $y_{ijt}$ variables). Define the size of an interval $I = I(i, *, k, \ell)$ as

$$\text{Size}(I) = \sum_{y_{ijt} \in I} y_{ijt}^{\ell-1}. \quad (2.4)$$

As $y_{ijt}^{\ell-1} \leq 2^k$ for jobs of class $k$, clearly $\text{Size}(I) \in [4 \cdot 2^k, 5 \cdot 2^k]$ for each $I$ (except possibly the last, in which case we can add a couple of extra dummy jobs at the end).

Note that the intervals formed in $LP(\ell)$ for $\ell > 0$ are not related to time anymore (unlike $LP(0)$), and in particular can span much longer duration of time that $4 \cdot 2^k$. All we ensure is that the amount of unassigned volume in an interval is $\Omega(2^k)$.  

29
Defining the LP for $\ell$-th iteration: With the above definition intervals $I(i, a, k, \ell)$ and the $y_{ijt}$ variables defined for the $\ell$-th iteration, we write the linear programming relaxation for $\ell$-th round, $LP(\ell)$.

$$\min \sum_i \sum_{t \geq r_j} \sum_k \sum_{j \in J(\ell): j \in (2^{k-1}, 2^k]} \left( \frac{t - r_j}{p_{ij}} + \frac{1}{2} \right) \cdot y_{ijt} \quad (LP(\ell))$$

$$\text{s.t.} \quad \sum_i \sum_{t \geq r_j} \frac{y_{ijt}}{p_{ij}} \geq 1 \quad \forall j \in J(\ell) \quad (2.5)$$

$$\sum_{y_{ijt} \in I(i,a,k,\ell)} y_{ijt} \leq \text{Size}(I(i,a,k,\ell)) \quad \forall i, k, a \quad (2.6)$$

$$y_{ijt} \geq 0 \quad \forall i, j \in J(\ell), t : t \geq r_j$$

2.4.1 Analysis

We note that $LP(\ell)$ is clearly a relaxation of $LP(\ell - 1)$ (restricted to variables corresponding to jobs in $J(\ell)$). This follows as setting $y_{ijt} = y_{ijt}^{\ell-1}$ is a feasible solution for $LP(\ell)$ (by the definition of Size($I$)). Moreover, the objective function of $LP(\ell)$ is exactly the objective of $LP(\ell - 1)$ when restricted to the variables in $J(\ell)$. Let $y^*$ denote the final integral assignment (assuming it exists) obtained by applying the algorithm iteratively to $LP(0), LP(1), \ldots$. Then this implies

**Lemma 5.** The cost of the integral assignment $\text{cost}(y^*)$ is at most the cost of optimal solution to $LP_{\text{new}}$.

Bounding the number of iterations: We now show that the sequence of $LP(\ell)$ relaxations terminate after some small number of rounds. Let $N_\ell = |J(\ell)|$ denote the number of jobs in $LP(\ell)$ (i.e. the one unassigned after solving $LP(\ell - 1)$).

**Lemma 6.** After each iteration, the number of unassigned jobs decreases by a constant factor. In particular, for each $\ell$: $N_\ell \leq N_{\ell-1}/2$.

**Proof.** Consider the basic optimal solution $y^*(\ell - 1)$ to $LP(\ell - 1)$. Let $S_{\ell-1}$ denote the non-zero variables in this solution, i.e. $y_{ijt}^{\ell-1}$ such that $y_{ijt}^{\ell-1} > 0$. Consider a linearly
independent family of tight constraints in $LP(\ell - 1)$ that generate the solution $y^*(\ell - 1)$. As tight constraints $y_{ijt}^{\ell-1} = 0$ only lead to 0 variables, it follows that $|S_{\ell-1}|$ is at most the number of tight constraints (2.5) or tight capacity constraints (2.6). Let $C_{\ell-1}$ denote the number of tight capacity constraints. Thus,

$$|S_{\ell-1}| \leq N_{\ell-1} + C_{\ell-1}. \quad (2.7)$$

Recall that $A(\ell - 1)$ denotes the set of jobs that are assigned integrally in the solution $y^*(\ell - 1)$. As each job not in $A(\ell - 1)$ contributes at least two to $|S_{\ell-1}|$, we also have

$$|S_{\ell-1}| \geq |A(\ell - 1)| + 2(N_{\ell-1} - |A(\ell - 1)|) = N_{\ell-1} + N_{\ell}. \quad (2.8)$$

The equality above follows as $N_{\ell} = N_{\ell-1} - |A(\ell - 1)|$ is the number of the (remaining) jobs considered in $LP(\ell)$. Together with (2.7) this gives

$$N_{\ell} \leq C_{\ell-1}. \quad (2.9)$$

We now show that $C_{\ell-1} \leq N_{\ell-1}/2$, which together with (2.9) would imply the claimed result. We do this by a charging scheme. Assign two tokens to each job $j$ in $N_{\ell-1}$. The jobs redistribute their tokens as follows.

Fix a job $j$ and let $k(i)$ denote the class of $j$ on machine $i$. For each machine $i$, time $t$ and class $k' \geq k(i)$, the job $j$ gives $\frac{1}{2^{k'-k(i)}} y_{ijt}^{\ell-1} p_{ij}$ tokens to the class $k'$ interval $I(i, a, k', \ell - 1)$ on machine $i$ containing $y_{ijt}$. If there are multiple time slots $t$ in an interval $I(i, a, k', \ell - 1)$ with $y_{ijt}^{\ell-1} > 0$, then $I(i, a, k', \ell - 1)$ receives a contribution from each of these slots. This is a valid token distribution scheme as the total tokens distributed by the job $j$ is at most

$$\sum_{i} \sum_{t} \sum_{k' \geq k(i)} \frac{y_{ijt}^{\ell-1}}{2^{k'-k(i)} p_{ij}} = \sum_{i} \sum_{t} \left( \sum_{k' \geq k(i)} \frac{y_{ijt}^{\ell-1}}{p_{ij}} \cdot \sum_{k' \geq k(i)} \frac{1}{2^{k'-k(i)}} \right) \leq 2 \cdot \sum_{i} \sum_{t} \frac{y_{ijt}^{\ell-1}}{p_{ij}} = 2.$$

Next, we show that each tight constraint of type (2.6) receives at least 4 tokens. If an interval $I(i, a, k', \ell - 1)$ of class $k'$ on machine $i$ is tight, this means that $\sum_{y_{ijt} \in I(i,a,k',\ell-1)} y_{ijt}^{\ell-1} = 2$. This gives

$$\sum_{y_{ijt} \in I(i,a,k',\ell-1)} y_{ijt}^{\ell-1} = 2.$$
Size$(I(i, a, k', \ell - 1))$ which is at least $4 \cdot 2^{k'}$. Now, the tokens given by a variable $y_{ijt}$ in $I(i, a, k', \ell - 1)$ where $j$ is of class $k(i) \leq k'$ are

$$\frac{y_{ijt}^{\ell-1}}{(2^{k'-k(i)} \cdot p_{ij})} \geq \frac{y_{ijt}^{\ell-1}}{(2^{k'} \cdot 2^{k(i)})} = \frac{y_{ijt}^{\ell-1}}{2^{k'}}.$$

Thus, the tokens obtained by $I(i, a, k', \ell - 1)$ are at least $\sum_{y_{ijt} \in I(i, a, k', \ell - 1)} y_{ijt}^{\ell-1}/2^{k'} \geq 4 \cdot 2^{k'}/2^{k'} = 4$. As each job distributes at most 2 tokens and each tight interval receives at least 4 tokens, we conclude that $C_{\ell-1} \leq N_{\ell-1}/2$. 

Bounding the backlog: To complete the proof of Lemma 2, it remains to show that for any time period $[t_1, t_2]$ and for any class $k$, the total volume of jobs belonging to class at most $k$ assigned to $[t_1, t_2]$ in $y^*$ is at most $(t_2 - t_1) + O(\log n)2^k$. Recall that $A(\ell)$ denotes the set of jobs which get integrally assigned in the $\ell$-th round. We use $A(t_1, t_2, i, k, \ell)$ to denote the set of jobs of class $\leq k$ which get integrally assigned to the machine $i$ in the interval $[t_1, t_2]$.

Given the solution $y^*(\ell)$ to LP$(\ell)$ and a time interval $[t_1, t_2]$, let us define

$$\text{Vol}(t_1, t_2, i, k, \ell) := \sum_{j \in J(\ell): p_{ij} \leq 2^k} \sum_{t \in [t_1, t_2]} y_{ijt}^{\ell} + \sum_{\ell' \leq (\ell-1)} \sum_{j \in A(t_1, t_2, i, k, \ell')} p_{ij},$$

as the total size of jobs of class $\leq k$, assigned either integrally or fractionally to the period $[t_1, t_2]$ after $\ell$ rounds. The following key lemma controls how much Vol can get worse in each round.

**Lemma 7.** For any period $[t_1, t_2]$, machine $i$, class $k$, and round $\ell$,

$$\text{Vol}(t_1, t_2, i, k, \ell) \leq O(1) \cdot 2^k + \text{Vol}(t_1, t_2, i, k, \ell - 1).$$

**Proof.** By the definition of Vol this is equivalent to showing that

$$\sum_{j \in J(\ell): p_{ij} \leq 2^k} \sum_{t \in [t_1, t_2]} y_{ijt}^{\ell} + \sum_{j \in A(t_1, t_2, i, k, \ell-1)} p_{ij} \leq O(1) \cdot 2^k + \sum_{j \in J(\ell-1): p_{ij} \leq 2^k} \sum_{t \in [t_1, t_2]} y_{ijt}^{\ell-1} \quad (2.10)$$
Fix a time period \([t_1, t_2]\). The main idea is that in each round \(\ell\), the error to \(\text{Vol}\) can be introduced only due to the two class \(k\) intervals overlapping with the boundary of \([t_1, t_2]\).

Consider the maximal set of contiguous intervals \(I(i, b, k, \ell), I(i, b + 1, k, \ell), \ldots I(i, b + h, k, \ell)\), for some \(b, h \geq 0\), that contain the period \([t_1, t_2]\). More precisely, \(b\) is the smallest index such that \(I(i, b, k, \ell)\) contains some \(y_{ijt}\) with \(t \in [t_1, t_2]\), and \(h\) is the largest index such that \(I(i, b + h, k, \ell)\) contains some \(y_{ijt}\) with \(t \in [t_1, t_2]\). As these intervals have size at most \(5 \cdot 2^k\), we have

\[
\sum_{y_{ijt} \in I(i, b, k, \ell)} y_{ijt}^\ell + \sum_{y_{ijt} \in I(i, b + h, k, \ell)} y_{ijt}^\ell \leq 10 \cdot 2^k. \tag{2.11}
\]

Now, consider the intervals \(I(i, b', k, \ell) \in \{I(i, b + 1, k, \ell), I(i, b + 2, k, \ell), \ldots I(i, b + h - 1, k, \ell)\}\) that are completely contained in \([t_1, t_2]\) (i.e. for all \(y_{ijt} \in I(i, b', k, \ell), t \in [t_1, t_2]\)). By definition of these intervals and capacity constraints of \(\text{LP}(\ell)\) we have,

\[
\sum_{b'=b+1}^{b+h-1} \sum_{y_{ijt} \in I(i, b', k, \ell)} y_{ijt}^\ell \leq \sum_{b'=b+1}^{b+h-1} \text{Size}(I(i, b', k, \ell)) \leq \sum_{b'=b+1}^{b+h-1} \sum_{y_{ijt} \in I(i, b', k, \ell)} y_{ijt}^\ell - \sum_{j \in J(\ell); p_{ij} \leq 2^k} \sum_{t \in [t_1, t_2]} y_{ijt}^{\ell - 1} \tag{2.12}
\]

The first inequality follows from the constraints (2.6) of \(\text{LP}(\ell)\), where as the second one follows from the definition (2.4) of \(\text{Size}\). We now prove (2.10). Consider,

\[
\sum_{j \in J(\ell); p_{ij} \leq 2^k} \sum_{t \in [t_1, t_2]} y_{ijt}^\ell \\
\leq \sum_{b'=b}^{b+h} \sum_{y_{ijt} \in I(i, b', k, \ell)} y_{ijt}^\ell \leq 10 \cdot 2^k + \sum_{j \in J(\ell); p_{ij} \leq 2^k} \sum_{t \in [t_1, t_2]} y_{ijt}^{\ell - 1} \quad [\text{by (2.11) and (2.12)}] \\
\leq 10 \cdot 2^k + \sum_{j \in J(\ell-1); p_{ij} \leq 2^k} \sum_{t \in [t_1, t_2]} y_{ijt}^{\ell - 1} - \sum_{j \in A(t_1, t_2, i, k, \ell - 1)} p_{ij}
\]

The last step follows as \(J(\ell) = J(\ell - 1) \setminus A(\ell - 1)\) and as \(\sum_{j \in A(t_1, t_2, i, k, \ell - 1)} y_{ijt}^{\ell - 1} = \sum_{j \in A(t_1, t_2, i, k, \ell - 1)} p_{ij}\). \(\square\)
This directly implies the following bound on the total error in any period \([t_1, t_2]\) in \(y^*\).

**Lemma 8.** For a given time period \([t_1, t_2]\), machine \(i\) and class \(k\), the total volume of jobs of class at most \(k\), assigned to the interval is at most \((t_2 - t_1) + O(\log n)2^k\).

**Proof.** Recall the definition of an interval \(I(i, a, k, 0)\) in \(LP(0)\). Each interval \(I(i, a, k, 0) = (t', t'')\) has size \(4 \cdot 2^k\) and contains all the \(y_{ijt}\) variables for jobs of class at most \(k\) and \(t \in (t', t'')\). Therefore, for any period \([t_1, t_2]\), by considering the capacity constraints (2.2) of \(LP(0)\) for the overlapping intervals \(I(i, *, k, 0)\), we obtain

\[
\text{Vol}(t_1, t_2, i, k, 0) = \sum_{j : p_{ij} \leq 2^k} \sum_{t \in [t_1, t_2]} y_{ijt}^0 \leq (t_2 - t_1) + O(1) \cdot 2^k \tag{2.13}
\]

Applying lemma 7 inductively (for the term \(\text{Vol}\) in the above equation) over the \(O(\log n)\) iterations of the algorithm gives the result. \(\square\)

**Proof of Lemma 2.** Consider the final solution \(y^*\) at the end of the algorithm. By our construction each job is integrally assigned in \(y^*\). By Lemma (5), \(\text{cost}(y^*)\) is no more than the cost of an optimal solution to \(LP_{\text{new}}\). By Lemma (8), for any time period \([t_1, t_2]\), machine \(i\) and class \(k\), the total volume of jobs assigned of jobs in class \(\leq k\) is at most \((t_2 - t_1) + O(\log n)2^k\). This concludes the proof. \(\square\)

### 2.4.2 The \(O(\log^2 n)\) approximation

The \(O(\log^2 n)\) approximation follows directly by observing that jobs much small \(p_{\text{max}}\) essentially have no effect.

The algorithm guesses \(p_{\text{max}}\), the value of the maximum job size in an optimal solution (say, by trying out all possible \(mn\) choices), and considers a modified instance \(J'\) where we set \(p_{ij} = p_{\text{max}}/n^2\) whenever \(p_{ij} < p_{\text{max}}/n^2\), and applies the previous algorithm for \(J'\). Clearly, \(P \leq n^2\) for \(J'\). Moreover \(\text{OPT}(J') \leq 2 \text{OPT}(J)\). Indeed, consider the optimum solution for \(J\) and for each job \(j\) assigned to machine \(i\) with size \(p_{ij} < p_{\text{max}}/n^2\), increase its size to \(p_{\text{max}}/n^2\) and push all the jobs behind it by the amount by which the size increases. This gives a valid schedule for \(J'\). Each job can be pushed by at most \(n\) jobs, and hence its
flow time increases by at most \( n \cdot p_{\text{max}}/n^2 \). Thus the total flow-time increases by at most \( p_{\text{max}} \) which is at most \( \text{OPT}(J) \).

2.5 Summary and Open Problems

In this chapter, we gave the first polylogarithmic approximation algorithm for minimizing the average flow-time on unrelated machines. An interesting open problem here is to get rid of the dependence on \( \log n \) in the approximation factor. An important open problem in this area is to obtain better approximation algorithm for the problem of minimizing the total \textit{weighted} flow-time for the \textit{single} machine case. Several polylogarithmic approximation algorithms are known for the problem, yet, improving the approximation factor has long been an important open problem.

2.6 Notes

3

Minimizing the Maximum Flow-time

3.1 Introduction

We continue our study of minimizing flow-time objective in the unrelated machine setting and consider the problem of minimizing the maximum flow-time. In this problem, there is a set $J$ of $n$ jobs and a set $M$ of $m$ machines. Each job $j$ is specified by its release time (or arrival time) $r_j$, which is the first time instant it is available for processing, and a machine-dependent processing requirement $p_{ij}$, which is the time taken to process $j$ on machine $i$. The flow-time of a job is defined as the amount of time the job spends in the system. Formally, if a job $j$ completes its processing at time $C_j$, then the flow-time of the job $F_j$ is defined as $C_j - r_j$; i.e., its completion time minus arrival time. The objective is to assign jobs to machines, and specify the order in which jobs are executed such that we minimize the maximum flow-time of a job. In other words, $\min \{ \max_{j \in J} F_j \}$. By doing binary search, we assume that we know the value of optimum solution (OPT), say $\text{OPT} = D$. Thus, we can reduce the problem of minimizing the maximum flow-time to verifying if there is a feasible schedule of jobs such that every job is assigned to a single machine and completes by the time $r_j + D$. Hence, minimizing the maximum flow-time is also closely related to deadline scheduling problems: maximum flow-time is $D$ if and only if each job $j$ is
completed by time $r_j + D$. However, usually the focus in deadline problems is to maximize the throughput (the jobs completed by their deadlines) and hence the results there do not translate to our setting. Also, observe that maximum flow-time is a (substantially harder) generalization of minimizing makespan objective. In particular, maximum flow-time is equal to the makespan if all the jobs are released at the same time. (See (Aspnes et al., 1997) for more details on the makespan minimization problem.) For the remainder of this chapter, we assume without loss of generality that no two jobs arrive at the same time. We also index the jobs by their release times.

3.1.1 Our Results

In this chapter we prove the following theorem.

**Theorem 9.** There is an $O(\log n)$-approximation algorithm for minimizing the maximum flow-time in the unrelated machine setting. In fact, the maximum flow-time exceeds the optimum value by an additive $O(\log n)p_{\text{max}}$ term, where $p_{\text{max}}$ is the maximum size of a job in the optimum schedule.

Our result gives the first non-trivial approximation algorithm for the problem in the unrelated machine setting. Our technique for obtaining the above result is similar to the problem of minimizing the total flow-time: We first formulate the problem as a linear program with few constraints and then apply the iterated rounding technique. However, there are some crucial differences between the two results, particularly in the rounding step as for maximum flow-time we must ensure that no job is delayed by too much. We give an overview of our techniques in the section 3.2.

3.1.2 History

For the single machine case, it is easy to see that First In First Out (FIFO) is an optimal (online) algorithm for the problem. However, relatively fewer results are known in the multiple machine setting. For identical machines, (Ambühl and Mastrolilli, 2005) showed that FIFO is a 3 competitive online algorithm. More general settings were considered recently
by (Anand et al., 2013), but all their positive results are in the resource augmentation setting. For unrelated machines, Bansal (2005) gave a polynomial time approximation scheme for the case of $m = O(1)$. Prior to our work, no non-trivial approximation algorithm was known even for the substantially simpler related machine setting.

3.2 Overview of Techniques

Unlike other commonly studied measures such as makespan and completion time, one problem with approximating flow-time is that it can be very sensitive to small changes in the schedule, and small errors can add up over time. The following example is instructive.

Consider some hard instance $I$ of the makespan problem on unrelated machines with $n$ jobs and $m$ machines, such that in the optimum schedule all machines have load exactly $T$ (add dummy jobs if necessary). As the problem is strongly NP hard, $T = \text{poly}(n,m)$. On the other hand, in any schedule computed by an efficient algorithm, at least one unit of work will be left unfinished at time $T$.

Make $N$ copies of $I$, and create an instance of maximum flow-time problem by releasing the jobs in the $i$-th copy at time $(i-1)T$, for $i = 1, 2, \ldots, N$. Clearly, the optimum maximum flow-time is $T$ as the optimum can finish jobs in the $i$-th copy before the next copy arrives. On the other hand, any polynomial time algorithm must ensure that a backlog of work does not build up over the copies. Otherwise, the accumulated error at the end will be $\Omega(N)$, leading to a maximum-flow time of $\Omega(N/mT)$ which can be made arbitrarily large (say $N = n^2m^2T^2 = \text{poly}(n,m)$).

To get around such issues we adopt a two-step approach. First, we determine a coarse schedule by computing an assignment of each job to some machine and a time slot, and ensure that for any machine, no overload is created in any reasonably large time interval. This is done by formulating a suitable LP to model these constraints, and then applying iterated rounding. A key property that enables us to apply iterated rounding is that our LP has few constraints. In the second step, we determine an actual schedule on each machine by scheduling the jobs according to SRPT or FIFO (depending on the problem),

38
and show that the no-overload property ensures that the quality of the solution does not worsen substantially.

We believe that this approach is quite modular and should be useful for many other scheduling problems, where currently we can only handle easier objectives such as makespan or completion time.

3.3 A Linear Programming Relaxation

We first write a linear programming relaxation for the problem. In this relaxation, there is a variable $x_{ij}$ denoting the total processing done on job $j$ on machine $i$. If $p_{ij} > D$ for a job $j$ on machine $i$, then we set $x_{ij} = 0$, as $j$ cannot be scheduled on machine $i$. The first set of constraints (3.1) ensure that each job is completely processed. To see the second constraint (3.2), we note that any job released during the interval $[t, t']$ must be completed by time $t' + D$. Thus the total size of jobs released in $[t, t']$ that are assigned to $i$ can be at most $(t' - t) + D$. Moreover, it suffices to consider intervals such that $t, t'$ are release dates of some jobs (as this gives the tightest constraints).

\[
\sum_i \frac{x_{ij}}{p_{ij}} \geq 1 \quad \forall j \quad (3.1)
\]

\[
\sum_{r_j \in [t, t']} x_{ij} \leq (t' - t) + D \quad \forall i, \forall t, t' \in \{r_1, \ldots, r_n\} \quad (3.2)
\]

\[
x_{ij} \geq 0 \quad \forall i, j \quad (3.3)
\]

\[
x_{ij} = 0 \quad \forall i, j \quad \text{with } p_{ij} > D. \quad (3.4)
\]

**Remark:** Note that the variables $x_{ij}$ do not specify the time at which job $j$ is assigned to machine $i$. However, it is instructive to view $x_{ij}$ units of work being assigned at time $r_j$ (the release time of $j$).

We say that a job is *integrrally* assigned to machine $i$ in the interval $[t_1, t_2]$ if $x_{ij} = p_{ij}$ and $r_j \in [t_1, t_2]$. Similarly, if $x_{ij} > 0$ and $x_{ij} \neq p_{ij}$, then job is assigned fractionally to
machine \( i \). Let \( p_{\text{max}} \) denote the maximum value of \( p_{ij} \) is some optimum schedule (note that \( p_{ij} \leq D \)). For convenience of description later, let us also assume that the release times are distinct (say, by perturbing them by some infinitesimally small amount).

As previously, we prove Theorem 9 using iterated rounding. To this end, we will show how to create a “tentative” schedule satisfying the following properties.

**Lemma 10.** There exists a solution \( x^* = \{ x_{ij} \}_{i,j} \) with the following properties:

- \( x^* \) integrally assigns each job \( j \) to a single machine \( i \); i.e., \( x_{ij} \) is equal to \( p_{ij} \) for some machine \( i \).

- For any time interval \([t_1, t_2]\), the total volume of jobs assigned in \( x^* \) is at most \((t_2 - t_1) + D + O(\log n) \cdot p_{\text{max}}\). That is,

\[
\sum_{j: r_j \in [t_1, t_2]} x_{ij} \leq (t_2 - t_1) + D + O(\log n) \cdot p_{\text{max}}.
\]

We first show Theorem 9 follows easily from the above lemma.

**Proof of Theorem 9.** Given a solution \( x^* \) satisfying the properties of Lemma 10, we construct a valid schedule such that flow-time of each job is at most \( D + O(\log n) \cdot p_{\text{max}} \) as follows. Fix a machine \( i \). Consider the jobs \( J(i, x^*) = \{ j \mid x_{ij} = p_{ij} \} \) assigned to machine \( i \), and schedule them in First In First Out (FIFO) order.

To see that a job \( j \) is completed by time \( r_j + D + O(\log n) \cdot p_{\text{max}} \), consider the interval \([0, r_j]\). Let \( t' \in [0, r_j] \) be the latest time instant when the machine \( i \) is idle. This implies that all the jobs in \( J(i, x^*) \) released in the interval \([0, t']\) are completed by \( t' \). As the machine is busy during \((t', r_j]\) and the total volume of jobs assigned in the interval is at most \((r_j - t') + D + O(\log n) \cdot p_{\text{max}}\) (as promised by Lemma 10), the total volume of jobs alive at \( r_j \) is at most \( D + O(\log n) \cdot p_{\text{max}} \). Since we schedule the jobs using FIFO, the job completes by time \( r_j + D + O(\log n) \cdot p_{\text{max}} \).

Henceforth, we focus on proving Lemma 10.
3.4 Iterated Rounding

We prove Lemma 10 using iterated rounding. Similar to the proof of Lemma (1), we write successive relaxations of the LP (3.1-3.3) denoted by $LP(\ell)$ (3.5-3.7), for $\ell = 0, 1, 2..., $ such that number of constraints drop by a constant fraction on each iteration. Finally, we obtain a solution where each job is integrally assigned to a single machine. $LP(0)$ is same as LP (3.1-3.3). Let $J(\ell)$ denote the set of jobs which are yet to be integrally assigned at the beginning of iteration $\ell$. Let $J(0) = J$. We now define $LP(\ell)$ for $\ell \geq 1$.

- **Computing a basic feasible solution:** Solve $LP(\ell - 1)$ and find a basic feasible solution $x^*(\ell - 1) = \{x^ {\ell - 1}_{ij}\}_{i,j}$ to $LP(\ell - 1)$. We use $x^ {\ell - 1}_{ij}$ to indicate the value taken by variable $x_{ij}$ in the solution $x^*(\ell - 1)$. Initialize $J(\ell) = J(\ell - 1)$.

- **Eliminating zero variables:** Variables $x_{ij}$ of $LP(\ell)$ are defined with respect to set of positive variables in the basic feasible solution to $LP(\ell - 1)$. In other words, if $x^ {\ell - 1}_{ij} = 0$ in $x^*(\ell - 1)$, then $x_{ij}$ is not defined in $LP(\ell)$.

- **Fixing integral assignments:** If $x^ {\ell - 1}_{ij} = p_{ij}$ for some job $j$, then $j$ is permanently assigned to machine $i$ in the solution $x^*$, and we update $J(\ell) = J(\ell) \setminus \{j\}$.

We drop all the variables involving job $j$ in $LP(\ell)$, and the constraint (3.5). Moreover, we update the constraints of type (3.6) as follows.

- **Defining Intervals:** For each machine $i$ and for each iteration $\ell$, we define the notion of intervals $I(i, a, \ell)$ as follows: Consider the variables $x_{ij}$ for jobs $j \in J(\ell)$ (i.e. the ones not assigned integrally thus far), in the order of non-decreasing release times. Greedily group consecutive $x_{ij}$ variables (starting from the beginning) such that sum of the $x^ {\ell - 1}_{ij}$ values in that group first exceeds $2p_{\text{max}}$. We call these groups intervals, and denote the $a$-th group by $I(i, a, \ell)$. We say $j \in I(i, a, \ell)$ if $x_{ij} \in I(i, a, \ell)$, and define $\text{Size}(I(i, a, \ell)) = \sum_{j \in I(i, a, \ell)} x^ {\ell - 1}_{ij}$. Note that $\text{Size}(I(i, a, \ell)) \in [2 \cdot p_{\text{max}}, 3 \cdot p_{\text{max}})$ (except possibly for the last interval, in
which case we add a dummy job of size $2p_{\text{max}}$.

\textbf{LP($\ell$):} We are now ready to write $LP(\ell)$.

\begin{align*}
\sum_{i} \frac{x_{ij}}{p_{ij}} & \geq 1 \quad \forall j \in J(\ell) \quad (3.5) \\
\sum_{j \in I(i,a,\ell)} x_{ij} & \leq \text{Size}(I(i,a,\ell)) \quad \forall i, a, \ell \quad (3.6) \\
x_{ij} & \geq 0 \quad \forall i, j \geq 0 \quad (3.7)
\end{align*}

By the definition of intervals and their sizes, it is clear that the feasible solution $x^*(\ell - 1)$ to $LP(\ell - 1)$ also is a feasible solution to $LP(\ell)$. Next we show that each job is integrally assigned after $O(\log n)$ iterations.

\textit{Bounding the number of iterations:} Let $N_{\ell}$ denote the number of jobs during the $\ell$-th iteration.

\textbf{Lemma 11.} For all $\ell > 1$, $N_{\ell} \leq \frac{N_{\ell-1}}{2}$.

\textbf{Proof.} Consider the basic optimal solution $x^*(\ell - 1)$ to $LP(\ell - 1)$. Let $S_{\ell-1}$ denote the non-zero variables in this solution, i.e. $x_{ij}$ such that $x_{ij}^{\ell-1} > 0$. Consider a linearly independent family of tight constraints in $LP(\ell - 1)$ that generate the solution $x^*(\ell - 1)$. Since tight constraints of the type of $x_{ij}^{\ell-1} = 0$ only lead to 0 variables, it follows that $|S_{\ell-1}|$ is at most the number of tight constraints (3.5) or tight capacity constraints (3.6). Let $C_{\ell-1}$ denote the number of tight capacity constraints. Thus,

$$ |S_{\ell-1}| \leq N_{\ell-1} + C_{\ell-1} \quad (3.8) $$

Recall that $A(\ell - 1)$ denotes the set of jobs that are assigned integrally in the solution $x^*(\ell - 1)$. Then, $N_{\ell} = N_{\ell-1} - |A(\ell - 1)|$ is the number of remaining jobs that are considered in $LP(\ell)$. As each job not in $A(\ell - 1)$ contributes at least a value of two to $|S_{\ell-1}|$, we also have

$$ |S_{\ell-1}| \geq |A(\ell - 1)| + 2(N_{\ell} - |A(\ell - 1)|) = N_{\ell-1} + N_{\ell} \quad (3.9) $$
Together with (3.8) this gives
\[ N_\ell \leq C_{\ell-1} \]  
(3.10)

We now show that \( C_{\ell-1} \leq N_{\ell-1}/2 \), which together with (3.10) would imply the claimed result. We know that size of each interval in \((\ell - 1)\)-th iteration is at least \( 2 \cdot p_{\text{max}} \). As each tight interval \( I(i,a,\ell - 1) \) has \( \sum_{j \in I(i,a,\ell - 1)} x_{ij}^{\ell - 1} = \text{Size}(I(i,a,\ell)) \), we have

\[ N_{\ell-1} \geq \frac{\sum_{i,j} x_{ij}^{\ell - 1}}{p_{\text{max}}} \geq \frac{2 \cdot p_{\text{max}} \cdot C_{\ell-1}}{p_{\text{max}}} \geq 2C_{\ell-1} \]

Thus we get \( C_{\ell-1} \leq N_{\ell-1}/2 \).

Therefore, the number of jobs which are integrally assigned at each iteration \( \ell \) is at least \( N_{\ell}/2 \). Note that number of constraints in \( LP(1) \) is at most \( n/2 \) since size of each interval is at least \( 2 \cdot p_{\text{max}} \). Hence, the algorithm terminates in \( O(\log n) \) rounds.

Bounding the overload: It remains to show that for any time interval \([t_1, t_2]\), the total size of jobs assigned in the interval \([t_1, t_2]\) in \( x^* \) is at most \( (t_2 - t_1) + O(\log n) \cdot p_{\text{max}} + D \).

Let \( \text{Vol}(t_1, t_2, i, \ell) \) be the total volume of jobs assigned (both fractionally and integrally) during the period \([t_1, t_2]\) at the end of \( \ell \)-th iteration. Moreover, let \( A(t_1, t_2, i, \ell - 1) \) be the set of jobs assigned in the period \([t_1, t_2]\) in the \((\ell - 1)\)-th iteration, i.e. \( x_{ij}^{\ell - 1} = p_{ij} \) and \( r_j \in [t_1, t_2] \).

Given the solution \( x^*(\ell) \) to \( LP(\ell) \). Clearly,

\[ \text{Vol}(t_1, t_2, i, \ell) = \sum_{r_j \in [t_1, t_2]} x_{ij}^{\ell} + \sum_{\ell' < \ell, j \in A(t_1,t_2,i,\ell')} p_{ij}. \]  
(3.11)

The following lemma shows that for any time period, the volume does not increase much in each round.

**Lemma 12.** For any iteration \( \ell \), machine \( i \), and any time period \([t_1, t_2]\),

\[ \text{Vol}(t_1, t_2, i, \ell) \leq \text{Vol}(t_1, t_2, i, \ell - 1) + 6 \cdot p_{\text{max}} \]
**Proof.** Consider the maximal contiguous set of intervals \( \mathcal{I} = \{I(i, b, \ell), I(i, b+1, \ell), \ldots I(i, b+h, \ell)\} \) such that for every interval \( I(i, b', \ell) \in \mathcal{I} \), there exists a job \( j \in I(i, b', \ell) \) and \( r_j \in [t_1, t_2] \). Recall that size of each interval in \( LP(\ell) \) is at most \( 3 \cdot p_{\text{max}} \). Hence, the intervals \( I(i, b, \ell) \) and \( I(i, b+h, \ell) \) which overlap \([t_1, t_2]\) at the left and right boundaries respectively, contribute at most \( 6 \cdot p_{\text{max}} \) to the interval \([t_1, t_2]\). Therefore,

\[
\sum_{r_j \in [t_1, t_2]} x_{ij}^\ell \leq \sum_{a=b+1}^{b+h-1} \text{Size}(I(i, a, \ell)) + 6 \cdot p_{\text{max}} \quad \text{[from (3.6)]}
\]

\[
\leq \sum_{r_j \in [t_1, t_2]} x_{ij}^{\ell-1} - \sum_{j \in A(t_1, t_2, i, \ell-1)} p_{ij} + 6 \cdot p_{\text{max}} \quad \text{[from def. of intervals]}
\]

\[
\leq \text{Vol}(t_1, t_2, i, \ell - 1) - \sum_{\ell' \leq (\ell-1)} \sum_{j \in A(t_1, t_2, i, \ell')} p_{ij} + 6 \cdot p_{\text{max}} \quad \text{[from (3.11)]}
\]

The lemma now follows by rearranging the terms and using (3.11).

**Lemma 13.** In the solution \( x^* \), the total volume of jobs assigned in any interval \([t_1, t_2]\) is at most \((t_2 - t_1) + D + O(\log n) \cdot p_{\text{max}}\).

**Proof.** Consider the interval \([t_1, t_2]\). From the constraints of \( LP(0) \) over the interval \([t_1, t_2]\) and definition of \( \text{Vol}(i, a, 0) \) (equation 3.11), we have,

\[
\text{Vol}(t_1, t_2, i, 0) = \sum_{r_j \in [t_1, t_2]} x_{ij}^0 \leq t_2 - t_1 + D
\]

The result now follows by applying Lemma 12 for the \( O(\log n) \) iterations of the algorithm.

**Proof of Lemma 10.** From lemma 11 we know that each job is integrally assigned to a single machine. Lemma 13 guarantees that total volume of jobs assigned in each time interval \([t_1, t_2]\) is bounded by \((t_2 - t_1) + D + O(\log n) \cdot p_{\text{max}}\). This gives us the desired \( x^* \) and concludes the proof.
3.5 Summary and Open Problems

In this chapter, we gave the first polylogarithmic approximation algorithm for minimizing the maximum flow-time on unrelated machines. An interesting open problem here is to obtain a better approximation factor. I believe that the problem admits an $O(1)$ approximation algorithm. Furthermore, it is possible that the integrality gap of our LP relaxation for the problem is at most a constant.

3.6 Notes

The chapter is based on joint work with Nikhil Bansal, and a preliminary version of this work appeared in the Proceedings of the 47th ACM Symposium on Theory of Computing (STOC 2015) (Bansal and Kulkarni, 2015).
4

Online and Non-Clairvoyant Algorithms For Minimizing Flow-Time

4.1 Introduction

We shift our focus on designing algorithms for flow-time objective in the online settings. From a practical viewpoint, online algorithms are of fundamental importance as in many applications we do not get to know the entire input in advance. In this chapter, we consider the problem of minimizing (weighted) flow-time on the unrelated machines in the online and non-clairvoyant setting, where the online algorithm is not aware of the processing lengths of the jobs. Unfortunately, however, minimizing the average flow-time, even when jobs are of unit length, has an unbounded competitive ratio in the online setting even for the restricted assignment case (Chadha et al., 2009). Thus, we focus on designing algorithms that have small competitive ratios in the resource augmentation model (Kalyanasundaram and Pruhs, 2000). Note that the results obtained in the last two chapters are in the offline setting, hence, do not extend in any natural way to the online setting. The algorithm and the analysis presented in this chapter are very different from the algorithms we saw the previous chapters. We begin with a brief motivation for our problem.

As the heterogeneous architectures become more popular, it is recognized by the com-
puter systems community (Bower et al., 2008) and the algorithms community that scheduling these future heterogeneous multiprocessor architectures is a major challenge in the online and non-clairvoyant setting, where the scheduler is not aware of the processing lengths of the jobs. It is known that some of the standard scheduling algorithms for single processors and homogeneous processors can perform quite badly on heterogeneous processors (Gupta et al., 2012b). A scalable algorithm (defined below) is known if somehow the scheduler is clairvoyant (able to know the size of a job when it arrives) (Chadha et al., 2009); their algorithm, however, needs to know the precise processing lengths of jobs, which is generally not available in general purpose computing settings. A scalable algorithm is also known if all jobs are of equal importance (Im et al., 2014a); however, the whole raison d’être for heterogeneous architectures is that there is generally heterogeneity among the jobs, most notably in their importance/priorities.

Therefore, a major open question in the area of online scheduling, both in theory and practice, is the design of scalable online algorithms for scheduling heterogeneous processors when job sizes are not known in advance (non-clairvoyant setting), and jobs have different priorities (weighted setting). The typical objective that has been studied in this context is minimizing weighted delay, or weighted flow-time. This problem generalizes most previous work (Anand et al., 2012; Chadha et al., 2009; Devanur and Huang, 2014; Im et al., 2014a; Gupta et al., 2012b; Im et al., 2014a) in online single and multiple machine scheduling considered recently by the algorithms community. As we discuss below, the algorithmic techniques developed for these problems (clairvoyant setting, unweighted setting, etc) do not extend in any natural way to the most general setting with weights, non-clairvoyance, and energy, leading to a fundamental gap in our understanding of classical scheduling algorithms. In particular, we ask: Do different variants of multiple machine scheduling considered in literature require different algorithmic techniques and analyses, or is there one unifying technique for them all?

In this chapter, we close this gap, and obtain one unifying algorithmic technique for
them all, achieving the first scalable non-clairvoyant algorithms for scheduling jobs of varying importance on heterogeneous machines. The interesting aspect of our work, as we discuss in Section 4.2 below, is that it provides an algorithmically simple and conceptually novel framework for multiple machine scheduling as a coordination game (see (Christodoulou et al., 2009; Bhattacharya et al., 2014b)), where jobs have (virtual) utility functions for machines based on delays they contribute to, and machines announce scheduling policies, and treat migration of jobs into them as new arrivals. In hindsight, we believe this provides the correct, unifying way of viewing all recent algorithmic results (Anand et al., 2012; Devanur and Huang, 2014; Im et al., 2014a; Gupta et al., 2012b) in related and simpler models.

4.1.1 Our Results

We adopt the competitive analysis framework in online algorithms. We say that an online schedule is $s$-speed $c$-competitive if it is given $s$ times faster machines and is $c$-competitive when compared to the optimal scheduler with no speed augmentation. The goal is to design a $O(1)$-competitive algorithm with the smallest extra speed. In particular, a scalable algorithm, which is $(1 + \epsilon)$-speed $O(1)$-competitive for any fixed $\epsilon > 0$, is considered to be essentially the best result one can hope for in the competitive analysis framework for machine scheduling (Kalyanasundaram and Pruhs, 2000). (It is known that without resource augmentation no online algorithm can have bounded competitive ratio for our problem (Garg and Kumar, 2007; Chadha et al., 2009).)

We consider the general unrelated machine model where each job $j$ can be processed at rate $\ell_{ij} \geq 0$ on each machine $i$; if $\ell_{ij} = 0$, then job $j$ cannot be processed on machine $i$. In a feasible schedule, each job can be scheduled by at most one machine at any point in time. Preemption is allowed without incurring any cost, and so is migration to a different machine. A job $j$’s flow-time $F_j := C_j - r_j$ measures the length of time between when the job arrives at time $r_j$ and when the job is completed at $C_j$. When each job $j$ has weight $w_j$, the total weighted flow-time is $\sum_j w_j F_j$; the unweighted case is where all jobs weights
are one. The online scheduler does not know the job size $p_j$ until the job completes. We give a formal statement of the problem in Section 4.4.

Our main result of the chapter is the following. For comparison, we note that no constant speed, constant competitive result was known for the weighted case, except when all machines are identical.

**Theorem 14.** For any $\epsilon > 0$, there is a $(1 + \epsilon)$-speed $O(1/\epsilon^2)$-competitive non-clairvoyant algorithm for the problem of minimizing the total weighted flow-time on unrelated machines. Furthermore, each job migrates at most $O((\log W + \log n)/\epsilon)$ times, where $W$ denotes the ratio of the maximum job weight to the minimum.

It is well known (Gupta et al., 2012b) that in the non-clairvoyant setting, jobs need to be migrated to obtain $O(1)$ competitive ratio. Our algorithm migrates each job relatively small number of times. Reducing the number of migrations is not only theoretically interesting but also highly desirable in practice (Chan et al., 2013; Gupta et al., 2014).

### 4.2 Technical Contributions: Selfish Migration and Nash Equilibrium

Our main technical contribution is a new conceptually simple game-theoretic framework for multiple machine scheduling that unifies, simplifies and generalizes previous work, both in terms of algorithm design as well as analysis using dual fitting. Before presenting this framework, we present some difficulties an online scheduler has to overcome in the non-clairvoyant settings we consider.

An online scheduler for multiple machines consists of two scheduling components: A single-machine scheduling policy on each machine, and the global machine assignment rule which assigns jobs to machines. In the context of clairvoyant scheduling (Chadha et al., 2009; Anand et al., 2012), the authors in (Anand et al., 2012) show via a dual fitting analysis that the following algorithm is scalable: Each machine runs a scalable single-machine scheduling policy such as Highest Density First (HDF); this is coupled with a simple greedy dispatch rule that assigns arriving jobs to the machine on which they cause
the least increase in flow-time to previously assigned jobs. This simple yet elegant algorithm has been very influential. In particular, the greedy dispatch rule has become standard, and been used in various scheduling settings (Im and Moseley, 2011; Gupta et al., 2010b; Anand et al., 2012; Thang, 2013; Devanur and Huang, 2014). The analysis proceeds by setting dual variables corresponding to a job to the marginal increase in total delay due to the arrival of this job, and showing that this setting is not only feasible but also extracts a constant fraction of the weighted flow-time in the objective. The immediate-dispatch greedy rule is necessary for analysis in all the aforementioned work, since they require the algorithm to measure each job’s effect on the system at the moment it arrives.

In a non-clairvoyant context, there are main two hurdles that arise. First, to use the greedy dispatch rule, it is crucial to measure how much a new job affects the overall delay, that is, how much the job increases the objective. To measure the increase, the scheduler must know the job size, which is not allowed in the non-clairvoyant setting. Secondly, as mentioned before, jobs must migrate to get a $O(1)$-competitive algorithm even with any $O(1)$-speed, and this makes it more difficult to measure how much each job is responsible for the overall delay. This difficulty appears in the two analysis tools for online scheduling, potential function (Im et al., 2011a) and dual fitting method (Anand et al., 2012; Gupta et al., 2012a; Devanur and Huang, 2014). Due to these difficulties, there have been very few results for non-clairvoyant scheduling on heterogeneous machines (Gupta et al., 2010a, 2012b; Im et al., 2014a). Further, there has been no work in any heterogeneous machines setting for the weighted flow-time objective.

4.2.1 SelfishMigrate Framework

We demonstrate a simple framework SelfishMigrate that addresses the above two issues in one shot. Our algorithm can be best viewed in a game theoretic setting where jobs are selfish agents, and machines declare their scheduling policies in advance.
Machine Behavior. Each machine maintains a virtual queue on the current set of jobs assigned to it; newly arriving jobs are appended to the tail of this queue. In a significant departure from previous work (Anand et al., 2012; Devanur and Huang, 2014; Gupta et al., 2010a, 2012b; Im et al., 2014a), each machine treats a migration of a job to it as an arrival, and a migration out of it as a departure. This means a job migrating to a machine is placed at the tail of the virtual queue.

Each machine runs a scheduling policy that is a modification of weighted round robin (WRR) that smoothly assigns larger speed to jobs in the tail of the queue, taking weights into account. This is a smooth extension of the algorithm Latest Arrival Processor Sharing (LAPS or WLAPS) (Edmonds and Pruhs, 2012; Edmonds et al., 2011). We note that the entire analysis also goes through with WRR, albeit with $(2 + \epsilon)$-speed augmentation.

The nice aspect of our smooth policies (unlike WLAPS) is that we can approximate the instantaneous delay introduced by this job to jobs ahead of it in its virtual queue, even without knowing job sizes. This will be critical for our analysis.\footnote{The best known competitive ratio for LAPS is $O(1/\epsilon^2)$ with $(1 + \epsilon)$-speed, which shows our overall analysis is tight unless one gives a better analysis of LAPS.}

Job Behavior. Each job $j$ has a virtual utility function, which roughly corresponds to the inverse of the instantaneous weighted delay introduced by $j$ to jobs ahead of it in its virtual queue, and their contribution to $j$’s weighted delay. Using these virtual utilities, jobs perform sequential best response (SBR) dynamics, migrating to machines (and get placed in the tail of their virtual queue) if doing so leads to larger virtual utility. Therefore, at each time instant, the job migration achieves a Nash equilibrium of the SBR dynamics on the virtual utilities. We show that our definition of the virtual utilities implies they never decrease due to migrations, arrivals, or departures, so that at any time instant the Nash equilibrium exists and is computable. (We note that at each time step, we simulate SBR dynamics and migrate each job directly to the machine that is predicted by the Nash equilibrium.)

When a job migrates to a machine, the virtual utility starts off being the same as the
real speed the job receives. As time goes by, the virtual queue ahead of this job shrinks, and that behind it increases. This lowers the real speed the job receives, but its virtual utility, which measures the inverse of the impact to jobs ahead in the queue and vice versa, does not decrease. Our key contribution is to define the coordination game on the virtual utilities, rather than on the actual speed improvement jobs receive on migration. A game on the latter quantities (utility equals actual speed) need not even admit to a Nash equilibrium.

Given the above framework, our analysis proceeds by setting the dual variable for a job to the increase in overall weighted delay it causes on jobs ahead of it in its virtual queue. Our key insight is to show that Nash dynamics on virtual utilities directly corresponds to our setting of dual variables being feasible for the dual constraints, implying the desired competitive ratio. This overall approach is quite general, even extending to energy constraints, and requires two key properties from the virtual utility:

- The virtual utility should correspond roughly to the inverse of the instantaneous delay induced by a job on jobs ahead of it in its virtual queue.

- SBR dynamics should monotonically improve virtual utility, leading to a Nash equilibrium that corresponds exactly to satisfying the dual constraints.

Our main (and perhaps quite surprising) contribution is to show the existence of such a virtual utility function for WRR and its scalable modifications, when coupled with the right notion of virtual queues. In hindsight, we believe this framework is the right way to generalize the greedy dispatch rules and dual fitting analysis from previous works (Anand et al., 2012; Im et al., 2014a), and we hope it finds more applications in complex scheduling settings.

We would like to emphasize that the online problem of minimizing the total weighted flow-time has no game theoretic component to it, and is concerned purely with online optimization (without any selfish behavior of jobs). However, we use game theoretic ideas
in design and analysis of the online algorithm. In the last part of this thesis, we will study scheduling problems where jobs behave selfishly, and the selfish behavior is taken into consideration in design of algorithms.

4.3 History

As mentioned above, the algorithmic idea based on coordination games is very different from the previous greedy dispatch rules (Chadha et al., 2009; Anand et al., 2012; Devanur and Huang, 2014) for clairvoyant scheduling, and also from previous work on non-clairvoyant scheduling (Gupta et al., 2010a, 2012b; Im et al., 2014a). We contrast our algorithm with these, highlighting the necessity of new techniques.

It is instructive to compare this framework with the scalable non-clairvoyant algorithm for unweighted flow-time (Im et al., 2014a). This algorithm has the seeds of several ideas we develop here – it introduces virtual queues, a smooth variant of LAPS (Edmonds and Pruhs, 2012; Edmonds et al., 2011) for single-machine scheduling, as well as migration based on the delay a job contributes to jobs ahead in its queue. However, this algorithm always assigns priorities to jobs in order of original arrival times, and migrates jobs to machines preserving this ordering. In essence, this algorithm mimics the clairvoyant algorithms (Chadha et al., 2009; Anand et al., 2012) that account for delay a job contributes to those that arrived ahead in time. This idea of arrival time ordering is specific to unweighted jobs, and does not extend to the weighted case or to energy constraints. In contrast, we let each machine treat migrations as new arrivals, leading us to view job migration through a game-theoretic lens. This leads to a more natural framework via instantaneous Nash equilibria, with a simple dual fitting analysis. The simplicity makes the framework extend easily to energy constraints. The resulting accounting using the delay a job \( j \) induces to those ahead of it in its virtual queue is novel – in contrast to previous work (Anand et al., 2012; Devanur and Huang, 2014; Gupta et al., 2010a, 2012b; Im et al., 2014a), the virtual queue changes with time and could possibly include jobs whose original arrival time \( r_{j'} \) is later than that of job \( j \).
To further illustrate the technical difficulty of the weighted case, let us consider Round-Robin (RR) and its variants. The work of (Gupta et al., 2010a) consider unweighted jobs, and gives \((2 + \epsilon)\)-speed \(O(1)\)-competitive algorithm for the total (unweighted) flow-time on related machines. The work of (Gupta et al., 2012b) improves this to a scalable algorithm, but for simpler illustration, we focus on RR. In the RR used in (Gupta et al., 2010a), each of \(n\) fastest machines is distributed to all \(n\) jobs uniformly. It is not difficult to see that this fractional schedule can be decomposed into a convex combination of feasible actual schedules. Hence, RR allows us to view multiple machines as a more powerful single machine, and this is the crux of the analysis in (Gupta et al., 2010a). In contrast, the work of (Gupta et al., 2012b) argues that the weighted case is not as simple: in fact, they show that natural extensions of weighted round robin for related machines fail to be competitive.

For a survey on online scheduling, please see (Pruhs et al., 2004). As alluded to above, for weighted flow-time on unrelated machines, the best clairvoyant result is a \((1 + \epsilon)\)-speed \(O(1/\epsilon)\)-competitive algorithm (Anand et al., 2012; Devanur and Huang, 2014). For the version with power functions, the best corresponding clairvoyant result is a \((1 + \epsilon)\)-speed \(O(1/\epsilon)\)-competitive algorithm (Devanur and Huang, 2014). In the most basic setting of multiple machines where machines are all identical, the work of (Chekuri et al., 2004) gives the first analysis of scalable algorithms for weighted flow-time. Intuitively, machine assignment rule should have a spirit of load balancing. Indeed, the work of (Chekuri et al., 2004) shows two machine assignment rules can be combined with various single machine scheduling policies to yield a scalable algorithm. One is random assignment rule, and the other is based on volume of each job class.

For the problem of non-clairvoyantly scheduling a single machine, the WLAPS (Weighted Latest Arrival Processor Sharing) algorithm (Edmonds and Pruhs, 2012; Bansal et al., 2010; Edmonds et al., 2011) is scalable for the total weighted flow even when jobs have arbitrary speedup curves. Other non-clairvoyant scalable algorithms for the unweighted case include Shorted Elapsed Time First (SETF) and Multi-level Feedback (Kalyanasundaram and Pruhs, 2000). The work of (Bansal and Pruhs, 2004) extends SETF to its weighted
version. While Shortest Remaining Processing Time (SRPT) is optimal for the total flow
time objective on a single machine with clairvoyance, even slightly more general settings
(non-clairvoyant or weighted or multiple machine settings) do not admit a \( O(1) \)-competitive
algorithm without resource augmentation (Becchetti and Leonardi, 2004; Bansal and Chan,
2009; Leonardi and Raz, 2007).

4.4 Non-clairvoyant Scheduling

In this problem, there are \( m \) unrelated machines. Job \( j \) is processed at rate \( \ell_{ij} \in [0, \infty) \) on
each machine \( i \). Each job has processing length \( p_j \) and weight \( w_j \). The online algorithm is
allowed to preempt and migrate jobs at any time with no penalty. The important constraint
is that at any instantaneous time, a job can be processed only on a single machine. Job
\( j \) is released at time \( r_j \). In the non-clairvoyant online scheduling model we consider, the
scheduler knows the values of \( \ell_{ij} \) and \( w_j \) when the job arrives, but is not aware of the
processing length \( p_j \).\(^2\) Without loss of generality we assume that weights \( w_j \) are integers.

Fix some scheduling policy \( P \). At each time instant \( t \), each active job \( j \) with \( r_j \leq t \) is
assigned to some machine \( i \). Let \( J_i(t) \) denote the set of jobs assigned to machine \( i \). Machine
\( i \) splits its processing power among the jobs in \( J_i(t) \). Let \( \nu_j(t) \) denote the processing power
assigned to job \( j \in J_i(t) \). We enforce that \( \sum_{j \in J_i(t)} \nu_j(t) \leq 1 \) for all \( i, t \). Then, \( j \in J_i(t) \)
exectes at rate \( q_j(t) = \ell_{ij} \nu_j(t) \). The completion time \( C_j \) is defined as the earliest time \( t_j \)
such that

\[
C_j = \arg\min_{t_j} \left( \int_{t=r_j}^{t_j} q_j(t) dt \geq p_j \right)
\]

At this time, the job finishes processing and departs from the system. The objective is to
find a scheduling policy that minimizes the sum of weighted flow-times \( \sum_j w_j F_j \), where
\( F_j = C_j - r_j \) is the flow-time of job \( j \).\(^{2} \)

In the speed augmentation setting, we assume the online algorithm can process job \( j \) at

\(^2\) It is easy to show that if \( \ell_{ij} \) values are not known, then no online algorithm can have a bounded
competitive ratio even with any constant speed augmentation.
rate \((1 + \epsilon)\ell_{ij}\) on machine \(i\), where \(\epsilon > 0\). We will compare the resulting flow-time against an offline optimum that knows \(p_j\) and \(r_j\) at time 0, but is not allowed the extra speed. Our main result is a scalable algorithm that, for any \(\epsilon > 0\), is \(O(1/\epsilon^2)\) competitive with speed augmentation of \((1 + \epsilon)\).

4.4.1 The SelfishMigrate Algorithm

Our algorithm can be best viewed as a coordination mechanism between the machines and the jobs. Each machine declares a single machine policy that it uses to prioritize and assign rates to arriving jobs. Given these policies, jobs migrate to machines that give them the most instantaneous utility (in a certain virtual sense). We will now define the single machine scheduling policy, and the utility function for jobs.

**Single Machine Policy: Weighted Ranked Processor Sharing, S-LAPS \((k)\)**

This policy is parametrized by an integer \(k\) (that we will later set to \(1/\epsilon\)) and \(\eta > 1\) that captures the speed augmentation (and that we set later to \(1 + 3\epsilon\)). Fix some machine \(i\) and time instant \(t\). Recall that \(J_i(t)\) denotes the set of jobs assigned to this machine at time \(t\). Let \(W(i,t)\) denote their total weight, i.e., \(W(i,t) = \sum_{j \in J_i(t)} w_j\). The machine maintains a virtual queue on these jobs.

We now introduce some notation based on these virtual queues. Let \(\sigma(j,t)\) denote the machine to which job \(j\) is assigned at time \(t\). Therefore, \(i = \sigma(j,t)\) if and only if \(j \in J_i(t)\). Let \(\pi_j(t)\) denote the rank of \(j\) in the virtual queue of \(i = \sigma(j,t)\), where the head of the virtual queue has rank 1 and the tail of the queue has rank \(|J_i(t)|\). Let \(J_j(t)\) denote the set of jobs ahead of job \(j \in J_i(t)\) in the virtual queue of machine \(i\). In other words

\[
J_j(t) = \{ j' \mid \sigma(j',t) = \sigma(j,t) \text{ and } \pi_{j'}(t) < \pi_j(t) \}
\]

Let \(W_j(t) = \sum_{j' \in J_j(t)} w_{j'}\) denote the total weight of jobs ahead of job \(j\) in its virtual queue.

**Rate Assignment.** At time instant \(t\), the total processing rate of the machine \(i\) is divided among the jobs in \(J_i(t)\) as follows. Job \(j \in J_i(t)\) is assigned processing power \(\nu_j(t)\) as
follows:

\[ \nu_j(t) := \eta \cdot \frac{(W_j(t) + w_j)^{k+1} - W_i^{k+1}}{W(i, t)^{k+1}} \]  

(4.1)

The rate at which job \( j \in J_i(t) \) is processed at time \( t \) is therefore \( \ell_{ij} \nu_j(t) \). Note that \( \sum_{j \in J_i(t)} \nu_j(t) = \eta \) at all time instants \( t \) and for all machines \( i \). Note that if \( k = 0 \), this is exactly weighted round robin. As \( k \) becomes larger, this gives higher rate to jobs in the tail of the queue, taking the weights \( w_j \) into account. This ensures that small jobs arriving later do not contribute too much to the flow-time, hence reducing the speed augmentation.

One important property of S-LAPS \((k)\) is that if a new job is added to the tail of the virtual queue, then all the old jobs are slowed down by the same factor. This is one of the important characteristics of weighted round robin which ensures that for a pair of jobs weighted delay induced by each other is exactly same. S-LAPS\((k)\) preserves this property to a factor of \( O(k) \), and this will be crucial to our analysis.

We note that using the natural setting of \( k = 0 \) (weighted round robin) gives a competitive algorithm with speedup \((2 + \epsilon)\), and this is tight even for a single machine. We use a larger value of \( k \) to reduce the amount of speed augmentation needed. (We believe that S-LAPS\((k)\) gives a black-box reduction from any \((2 + \epsilon)\)-speed algorithm using WRR into a scalable algorithm.)

Arrival Policy. The behavior of the policy is the same when a job \( j \) either arrives to the system and chooses machine \( i \), or migrates from some other machine to machine \( i \) at time \( t \). In either case, the job \( j \) is placed at the tail of the virtual queue. In other words, if \( J_i(t^-) \) is the set of jobs just before the arrival of job \( j \), then we set \( \sigma(j, t) = i \) and \( \pi_j(t) = |J_i(t^-)|+1 \). Therefore, the virtual queue sorts the jobs in order in which they arrive onto this machine. Since a job could also arrive due to a migration, this is not the same as ordering on the \( r_j \) — a job with smaller \( r_j \) that migrates at a latter point onto machine \( i \) will occupy a relatively later position in its virtual queue.
Departure Policy. If job $j$ departs from machine $i$ either due to completion or due to migrating to a different machine, the job simply deletes itself from $i$’s virtual queue, keeping the ordering of other jobs the same. In other words, for all jobs $j' \in J_i(t)$ with $\pi_{j'}(t) > \pi_j(t)$, the value $\pi_{j'}(t)$ decreases by 1.

Virtual Utility of Jobs and Selfish Migration

The virtual queues define a virtual utility of job as follows. Let $j \in J_i(t)$ at time $t$. Then its virtual utility is defined as:

$$\phi(j, t) = \frac{\ell_{ij}}{W_j(t) + w_j}$$

We interpret this utility as follows: The inverse of this quantity will be roughly in proportion to the marginal increase in instantaneous weighted delay that job $j$ induces on jobs $J_j(t)$ that are ahead of it in its virtual queue, and their contribution to the weighted delay of $j$. We will establish this in the Delay Lemmas below. This marginal increase is exactly what we need in order to define dual variables in our proof, and in some sense, the virtual utility is defined keeping this in mind.

At every time instant, job $j \in J_i(t)$ behaves as follows: If it were to migrate to machine $d \neq i$, it would be placed at the tail of $d$’s queue and would obtain virtual utility $\frac{\ell_{dj}}{W(d, t) + w_j}$. If this quantity is larger than $\phi(j, t)$, then job $j$ migrates to machine $d$. This leads to the corresponding changes to the virtual queues of machine $i$ (job $j$ is deleted), machine $d$ (job $j$ is added to the tail), and the virtual utility $\phi(j, t)$ of job $j$ (which is set to $\frac{\ell_{dj}}{W(d, t) + w_j}$). At every time instant $t$, this defines a game on the jobs, and starting with the configuration at the previous step, the jobs simulate sequential best response dynamics, where they sequentially migrate to improve virtual utility, till the system reaches a Nash equilibrium. In this configuration, each job is on a machine that is locally optimal for $\phi(j, t)$.

Note that if a job departs from a machine, the virtual utilities of other jobs on that machine either stay the same or increase. Further, if a job migrates to a machine, it is placed on the tail of the virtual queue, so that the virtual utilities of other jobs on the machine
remain the same. This shows that sequential best response dynamics guarantees that the virtual utilities of all jobs are monotonically non-decreasing with time, converging to a Nash equilibrium. (Note that jobs don’t actually need to execute best response dynamics since they can directly migrate to the machines corresponding to the resulting Nash equilibrium.)

When a new job arrives to the system, it simply chooses the machine \(i\) which maximizes its virtual utility, \(\ell_{ij} \over W(i,t) + w_j\), where \(W(i,t)\) is the weight of jobs assigned to \(i\) just before the arrival of job \(j\). This completes the description of the algorithm.

The following lemma is an easy consequence of the description of the algorithm.

**Lemma 15.** For all jobs \(j\), \(\phi(j,t)\) is non-decreasing over the interval \(t \in [r_j, C_j]\).

### 4.5 Analysis of SelfishMigrate

We first write a linear programming relaxation of the problem \(LP_{\text{new}}\) described below which was first given by Anand et al. (2012); Garg and Kumar (2007). It has a variable \(x_{ijt}\) for each machine \(i \in [m]\), each job \(j \in [n]\) and each unit time slot \(t \geq r_j\). If the machine \(i\) processes the job \(j\) during the whole time slot \(t\), then this variable is set to 1. The first constraint says that every job has to be completely processed. The second constraint says that a machine cannot process more than one unit of jobs during any time slot. Note that the LP allows a job to be processed simultaneously across different machines.

\[
\begin{align*}
\text{Min} \quad & \sum_j \sum_i \sum_{t \geq r_j} \left( \frac{\ell_{ij} (t - r_j)}{p_j} + 1 \right) \cdot w_j \cdot x_{ijt} \\
\sum_i \sum_{t \geq r_j} \frac{\ell_{ij} \cdot x_{ijt}}{p_j} & \geq 1 \quad \forall j \\
\sum_{j : t \geq r_j} x_{ijt} & \leq 1 \quad \forall i, t \\
x_{ijt} & \geq 0 \quad \forall i, j, t : t \geq r_j
\end{align*}
\]

It is easy to show that the above LP lower bounds the optimal flow-time of a feasible schedule within factor 2. We use the dual fitting framework to analyze SelfishMigrate. We write the dual of \(LP_{\text{new}}\) as,
\[
\max \sum_j \alpha_j - \sum_i \sum_t \beta_{it} \quad \text{(LP}_{\text{dual}}) \\
\frac{\ell_{ij} \cdot \alpha_j}{p_j} - \beta_{it} \leq \frac{w_j \ell_{ij}(t - r_j)}{p_j} + w_j \quad \forall i, j, t : t \geq r_j \\
\alpha_j \geq 0 \quad \forall j \\
\beta_{it} \geq 0 \quad \forall i, t
\]

We will show that there is a feasible solution to \(\text{LP}_{\text{dual}}\) that has objective \(O(\epsilon^2)\) times the total weighted flow-time of SELFISHMIGRATE, provided we augment the speed by \(\eta = (1 + 3\epsilon)\). From now on, we will assume that each processor in SELFISHMIGRATE has \(\eta\) extra speed when processing jobs.

**Instantaneous Delay and Setting Dual Variables**

Recall that each machine runs S-LAPS \((k)\) with \(k = 1/\epsilon\), and we assume without loss of generality that \(1/\epsilon\) is an integer. We define the *instantaneous weighted delay* induced by job \(j\) on jobs ahead of \(j\) in its virtual queue (the set \(J\_j(t)\)) as follows:

\[
\delta_j(t) = \frac{1}{\eta} \left( \sum_{j' \in J\_j(t)} (w_{j'} \cdot \nu_j(t) + w_j \cdot \nu_{j'}(t)) + w_j \cdot \nu_j(t) \right)
\]

This quantity sums the instantaneous weighted delay that \(j' \in J\_j(t)\) induces on \(j\) and vice versa, plus the delay seen by \(j\) due to itself. Note that \(\delta_j(t)\) is equal to \(\frac{1}{\eta} \left( (W_j(t) + w_j) \cdot \nu_j(t) + w_j \cdot \sum_{j' \in J\_j(t)} \nu_{j'}(t) \right)\)

Define,

\[
\Delta_j = \int_{t=r_j}^{C_j} \delta_j(t) dt
\]

as the cumulative weighted delay induced by \(j\) on jobs ahead of it in its virtual queue and vice versa. Note that the set \(J\_j(t)\) changes with \(t\) and can include jobs that are released after job \(j\). It is an easy exercise to check that \(\sum_j w_j F_j = \sum_j \Delta_j\). Our way of accounting for weighted delay is therefore a significant departure from previous work that either keeps \(J\_j(t)\) the same for all \(t\) (clairvoyant algorithms), or preserves orderings based on arrival time.
We now perform the dual fitting. We set the variables of the LP\textsubscript{dual} as follows. We set $\beta_{it}$ proportional to the total weight of jobs alive on machine $i$ at time $t$, i.e., $\beta_{it} = \frac{1}{k+3}W(i,t)$. We set $\alpha_j = \frac{1}{k+2}\Delta_j$, i.e., proportional to the cumulative weighted delay induced by $j$ on jobs ahead of it in its virtual queue.

We first bound the dual objective as follows (noting $k = 1/\epsilon$ and $\eta = 1 + 3\epsilon$):

$$
\sum_j \alpha_j - \sum_{i,t} \beta_{it} = \sum_j \frac{\Delta_j}{k+2} - \sum_{i,t} \frac{W(i,t)}{k+3} = \epsilon \left( \sum_j \frac{\Delta_j}{1+2\epsilon} - \sum_{i,t} \frac{W(i,t)}{1+3\epsilon} \right)
$$

$$
= \epsilon \cdot \sum_j w_j F_j \cdot \left( \frac{1}{1+2\epsilon} - \frac{1}{1+3\epsilon} \right) = O(\epsilon^2) \sum_j w_j F_j \quad (4.2)
$$

In the rest of the analysis, we show that this setting of dual variables satisfies the dual constraints.

**Delay Lemmas**

The dual constraints need us to argue about the weighted delay induced by $j$ till any point $t$. For this purpose, we define for any $t^* \in [r_j, C_j]$ the following:

$$
\Delta^1_j(t^*) = \int_{t=r_j}^{t^*} \delta_j(t)dt \quad \text{and} \quad \Delta^2_j(t^*) = \int_{t=t^*}^{C_j} \delta_j(t)dt
$$

The following propositions have elementary proofs which have been omitted.

**Proposition 16.** Consider any integer $k \geq 0$, and $0 \leq w \leq 1$, then $(1-w)^k \geq 1 - kw$.

**Proposition 17.** Consider any integer $k \geq 0$, and $w,w' \geq 0$, then $(w + w')^k \geq w^k + kw^{k-1}w'$.

The first Delay Lemma bounds the quantity $\Delta^1_j(t^*)$ as follows:

**Lemma 18 (First Delay Lemma).** For any time instant $t^* \in [r_j, C_j]$ and for any job $j$,

$$
\Delta^1_j(t^*) \leq (k+2) \cdot w_j \cdot (t^* - r_j)
$$
Proof.

\[
\Delta_{j}^{2}(t^{*}) = \frac{1}{\eta} \int_{t=r_{j}}^{t^{*}} \left( \nu_{j}(t) \cdot (W_{j}(t) + w_{j}) + w_{j} \cdot \left( \sum_{j' \in J_{j}(t)} \nu_{j'}(t) \right) \right) dt \\
\leq \frac{1}{\eta} \int_{t=r_{j}}^{t^{*}} \left( \eta \cdot \frac{(W_{j}(t) + w_{j}) W_{j}(t) + w_{j} + w_{j} \cdot \eta}{(W_{j}(t) + w_{j}) k+1} \right) dt \quad \text{[Def. } \nu] \\
\leq \int_{t=r_{j}}^{t^{*}} \left( \frac{(W_{j}(t) + w_{j}) W_{j}(t) + w_{j} + w_{j}}{k+1} \right) dt \\
= \int_{t=r_{j}}^{t^{*}} \left( 1 - \left( 1 - \frac{w_{j}}{W_{j}(t) + w_{j}} \right)^{k+1} \right) dt \\
= \int_{t=r_{j}}^{t^{*}} \left( w_{j} \cdot (k + 1) \right) dt \quad \text{[Proposition 16]} \\
= (k + 2) \cdot w_{j} \cdot (t^{*} - r_{j})
\]

The inequality (4.3) follows because \( W(\sigma(j, t), t) \geq W_{j}(t) + w_{j} \). Let \( p_{j}(t^{*}) = \int_{t=t^{*}}^{C_{j}} \ell_{\sigma(j, t), j} \cdot \nu_{j}(t) dt \) denote the residual size of job \( j \) at time \( t^{*} \). The second Delay Lemma states that total marginal increase in the algorithm’s cost due to job \( j \) till its completion is upper bounded by the marginal increase in the algorithm’s cost if the job \( j \) stays on machine \( \sigma(j, t^{*}) \) till its completion. However, as noted before, marginal increase in the cost of the algorithm on a single machine is inversely proportional to the job’s virtual speed. The proof of the second Delay Lemma hinges crucially on the fact that a job selfishly migrates to a new machine only if its virtual utility increases. In fact, the statement of this lemma implies the correctness of our setting of virtual utility.

Lemma 19 (Second Delay Lemma). For any time instant \( t^{*} \in [r_{j}, C_{j}] \) and for any job \( j \), let \( i^{*} = \sigma(j, t^{*}) \) denote the machine to which job \( j \) is assigned at time \( t^{*} \). Then:

\[
\Delta_{j}^{2}(t^{*}) \leq \frac{1}{\eta} \cdot \frac{k + 2}{k + 1} \cdot \frac{p_{j}(t^{*})}{\phi(j, t^{*})} \leq \frac{1}{\eta} \cdot \frac{k + 2}{k + 1} \cdot \frac{W_{j}(t^{*}) + w_{j}}{l_{i^{*}j}}
\]
Proof.

\[
\Delta_j^2(t^*) = \frac{1}{\eta} \int_{t=t^*}^{C_j} \left( \nu_j(t) \cdot (W_j(t) + w_j) + w_j \cdot \left( \sum_{j' \in J(t)} \nu_{j'}(t) \right) \right) dt \\
= \frac{1}{\eta} \int_{t=t^*}^{C_j} \nu_j(t) \cdot (W_j(t) + w_j) + w_j \cdot \frac{W_j(t)^{k+1}}{W(\sigma(j, t), t)^{k+1}} dt \\
= \frac{1}{\eta} \int_{t=t^*}^{C_j} \nu_j(t) \cdot \left( W_j(t) + w_j \cdot \frac{W_j(t)^{k+1}}{(W_j(t) + w_j)^{k+1} - W_j(t)^{k+1}} \right) dt \\
\leq \frac{1}{\eta} \int_{t=t^*}^{C_j} \nu_j(t) \cdot \left( W_j(t) + w_j + \frac{W_j(t) + w_j}{k + 1} \right) dt \quad \text{[Proposition 17]} \\
= \frac{1}{\eta} \cdot k + 2 \cdot \int_{t=t^*}^{C_j} \ell_{\sigma(j, t)} \cdot \nu_j(t) \cdot \frac{1}{\phi(j, t)} dt \\
\leq \frac{1}{\eta} \cdot k + 2 \cdot \frac{1}{\phi(j, t^*)} \int_{t=t^*}^{C_j} \ell_{\sigma(j, t)} \nu_j(t) dt \quad \text{[Lemma 15]} \\
= \frac{1}{\eta} \cdot k + 2 \cdot \frac{p_j(t^*)}{\phi(j, t^*)} \leq \frac{1}{\eta} \cdot k + 2 \cdot \frac{W_j(t^*) + w_j}{l_{i^*j}} \\
\square
\]

Note that the previous two lemmas imply the following by summation:

**Lemma 20.** For any time instant \( t \in [r_j, C_j] \) and job \( j \) that is assigned to machine \( i^* = \sigma(j, t) \), we have:

\[
\Delta_j = \Delta_j^1(t) + \Delta_j^2(t) \leq (k + 2) \cdot w_j \cdot (t - r_j) + \frac{1}{\eta} \cdot k + 2 \cdot p_j \cdot \frac{W_j(t) + w_j}{l_{i^*j}}
\]

**Checking the Feasibility of Constraints**

Now it remains to prove that constraints of \( \text{LP}_{\text{dual}} \) are satisfied. To see this, fix job \( j \) and time instant \( t \). We consider two cases.
Case 1: Machine \( i = \sigma(j, t) \). Then

\[
\alpha_j - \frac{p_j}{\ell_{ij}} \beta_{it} = \frac{\Delta_j}{k+2} - \frac{p_j}{\ell_{ij}} \cdot \frac{W(i, t)}{k+3}
\]

\[
\leq w_j \cdot (t - r_j) + \frac{p_j}{\eta \cdot (k+1)} \cdot \frac{W_j(t) + w_j}{\ell_{ij}} - \frac{p_j}{\ell_{ij}} \cdot \frac{W(i, t)}{k+3} \quad \text{[Lemma 20]}
\]

\[
\leq w_j \cdot (t - r_j) \quad \text{[since } \eta = 1 + 3\epsilon, k = 1/\epsilon]\]

Case 2: Machine \( i \neq \sigma(j, t) \). Then

\[
\alpha_j - \frac{p_j}{\ell_{ij}} \beta_{it} = \frac{\Delta_j}{k+2} - \frac{p_j}{\ell_{ij}} \cdot \frac{W(i, t)}{k+3}
\]

\[
\leq w_j \cdot (t - r_j) + \frac{1}{\eta} \cdot \frac{p_j}{k+1} \cdot \frac{W_j(t) + w_j}{\ell_{ij}} - \frac{p_j}{\ell_{ij}} \cdot \frac{W(i, t)}{k+3} \quad \text{[Lemma 20]}
\]

\[
\leq w_j \cdot (t - r_j) + \left( \frac{p_j}{\eta \cdot (k+1)} \frac{W(i, t) + w_j}{\ell_{ij}} - \frac{p_j}{k+3} \cdot \frac{W(i, t)}{\ell_{ij}} \right)
\]

\[
\leq w_j \cdot (t - r_j) + \frac{w_j p_j}{\ell_{ij}} \quad \text{[since } \eta = 1 + 3\epsilon, k = 1/\epsilon]\]

The penultimate inequality follows since the machine \( \sigma(j, t) \) maximizes the virtual utility of job \( j \) at time \( t \). Therefore, the dual constraints are satisfied for all time instants \( t \) and all jobs \( j \), and we derive that \textsc{SelfishMigrate} is \((1 + \epsilon)\)-speed augmentation, \(O(1/\epsilon^2)\)-competitive against \( \text{LP}_{\text{new}} \), completing the proof of the first part of Theorem 14.

4.6 Polynomial Time Algorithm and Minimizing Reassignments.

A careful observation of the analysis reveals that to satisfy dual constraints, each job need not be on the machine which gives the highest virtual utility. We can change the policy \textsc{SelfishMigrate} so that a job migrates to a different machine only if its virtual utility increases by a factor of at least \((1 + \epsilon)\). Note that this does not change the monotonicity properties of the virtual utility of a job, hence the entire analysis follows (with the speed augmentation \( \eta \) increased by a factor of \( 1 + \epsilon \)). This also implies that for any job \( j \), the total number of migrations is at most \((\log(1+\epsilon) W + \log(1+\epsilon) n)\), where \( W \) is the ratio of
the maximum weight of all jobs to the minimum weight. Omitting the simple details, we complete the proof of Theorem 14.

4.7 Summary and Open Problems

In this chapter we presented a new framework based on game theoretic ideas for the design and analysis of non-clairvoyant scheduling algorithm for minimizing weighted flow-time. An important open question here is to extend the results to general $\ell_k$-norms of flow-time.

4.8 Notes

The chapter is based on joint work with Sungjin Im, Kamesh Munagala and Kirk Pruhs. A preliminary version of this work appeared in the 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014 Im et al. (2014b).
PART II

Energy Efficient Scheduling
5.1 Introduction

Modern data centers are big consumers of electricity, and large providers of cloud services see huge cost savings by even modest savings in energy consumption. A standard approach taken by system designers has been to build systems which are energy efficient, via technologies such as speed scalable processors, dynamic power-down and power-up mechanisms, new cooling technologies, and multi-core servers. These approaches have been investigated widely both in practice and theory; see (Pruhs, 2011; Albers, 2009; Murugesan, 2008) for more details. In the next chapter, we propose a new approach to reducing the energy costs by exploiting the variable pricing of electricity.

We focus on the dynamic speed scaling model in this chapter. In this model, introduced in a seminal work of (Yao et al., 1995), the goal is to dynamically scale the speed of a processor to optimize power consumed (which is a convex function of speed) and some QoS metric like deadlines or flow time. This model has a rich literature in online algorithms and potential function design; see (Bansal et al., 2011; Pruhs et al., 2008; Bansal et al., 2009a; Albers and Fujiwara, 2007; Bansal et al., 2009b; Pruhs, 2011; Gupta et al., 2012b) for more details.
In this chapter, we show that a simple extension to SELFISHMIGRATE algorithm, presented in the previous chapter, gives a scalable algorithm for minimizing the sum of weighted flow-time and energy in the dynamic speed scaling model. The problem formulation is the same as in Chapter 4.4, with an added feature. Each machine \( i \) can be run at a variable rate \( S(i,t) \) by paying an energy cost of \( f_i(S(i,t)) \), where \( f_i \) is a machine dependent, convex increasing function (also called as power function). The rate \( S(i,t) \) can be partitioned among the jobs \( J_i(t) \), so that \( \sum_{j \in J_i(t)} \nu_j(t) \leq S(i,t) \). As before, job \( j \in J_i(t) \) runs at speed \( q_j(t) = \nu_j(t) \times \ell_{ij} \).

We define the completion time \( C_j \) of job \( j \) to satisfy \( \int_{t-\tau_j}^{C_j} q_j(t)dt = p_j \). Both preemption and migration of jobs are allowed without any penalty, but each job must be assigned to a single machine at every instant of time. The scheduler is not aware of the processing lengths \( p_j \). Our objective is to minimize sum of weighted flow-time and energy consumed:

\[
\text{Objective} = \sum_j w_j F_j + \sum_i \int_t f_i(S(i,t))dt
\]

### 5.1.1 Our Results

In this chapter we show the first scalable algorithm for the problem in the non-clairvoyant setting. The main result of the chapter is,

**Theorem 21.** For any \( \epsilon > 0 \), there is a \( (1 + \epsilon) \)-speed \( O(1/\epsilon^2) \)-competitive non-clairvoyant algorithm for the problem of minimizing the total weighted flow-time plus total energy consumption on unrelated machines. This result holds even when each machine \( i \) has an arbitrary strictly-convex power function \( f_i : [0, \infty) \to [0, \infty) \) with \( f_i(0) = 0 \).

The theorem implies a \( O(\gamma^2) \)-competitive algorithm (without resource augmentation) when each machine \( i \) has a power function \( f(s) = s^\gamma \) for some \( \gamma > 1 \), perhaps most important power functions in practice. Our result also implies a scalable algorithm in the model where at each time instant a processor \( i \) can either run at speed \( s_i \) consuming a power \( P_i \), or be shutdown and consume no energy.
We note that no $O(1)$-speed $O(1)$-competitive non-clairvoyant algorithm was known prior to our work even in the related machine setting for any nontrivial classes of power functions.

In a resource augmentation analysis, we assume that the online algorithm gets $(1 + \epsilon)$ more speed, for any $\epsilon > 0$, for consuming the same energy. Alternatively, the offline benchmark has to pay a cost of $f_i((1+\epsilon)s)$ if it runs machine $i$ at a rate of $s$. Speed augmentation is required to achieve meaningful competitive ratios for the case of arbitrary power functions. To elaborate on this point, consider a function $f(s)$ that takes an infinitesimal value in the interval $0 \leq s \leq 1$ and sharply increases when $s > 1$. For such a power function, any competitive online scheduler has to be optimal at each instant of time unless we give it more resources. A scalable algorithm in the speed augmentation setting implies algorithms with small competitive ratios when the energy cost function can be approximated by polynomials. In particular, the result translates to an $O(\gamma^2)$-competitive algorithm (without any resource augmentation) when the power function is $f_i(s) = s^\gamma$.

Let $g_i$ be the inverse of power function $f_i$. Note that $g$ is an increasing concave function.

Before we describe our algorithm, we make the following simple observation regarding concave functions.

**Proposition 22.** For any increasing concave function $g$ with $g(0) = 0$, $\frac{g(w)}{w}$ is decreasing in $w$.

### 5.2 The SelfishMigrate-Energy Algorithm

Our algorithm **SelfishMigrate-Energy**, is very similar to the algorithm **SelfishMigrate**. Please see the chapter 4.4 for more details on the **SelfishMigrate**. The most important difference is the policy that sets the speeds of the machines.

**Speed Scaling Policy:** We set the speed of machine $i$ at time $t$, denoted by $S(i,t)$, such that the total energy cost is equal to the total weight of jobs at time $t$.

$$f_i(S(i,t)) = W(i,t) \quad \text{or equivalently,} \quad S(i,t) = g_i(W(i,t)) \quad (5.1)$$
This is same as the speed scaling policy used in (Bansal et al., 2009a; Anand et al., 2012). Our speed scaling policy easily implies that the total energy cost of the schedule is equal to the weighted flow-time. Hence, we will only be concerned with the total weighted flow-time of our schedule.

**Single Machine Policy.** The remaining components of the algorithm remain similar to SELFISHMIGRATE algorithm. We briefly mention the differences.

Each machine runs S-LAPS \((k)\) where \(k = \frac{1}{\epsilon}\). In this policy, the notions of virtual queues, rank of a job, and the arrival and departure policies (with associated notation) remain the same. In particular, a job that arrives or migrates to a machine are placed at the tail of the virtual queue and assigned the highest rank on the machine. At time instant \(t\), the total processing rate \(S(i, t) = g_i(W(i, t))\) of the machine \(i\) is divided among the jobs in \(J_i(t)\) as follows.

\[
\nu_j(t) := g_i(W(i, t)) \cdot \frac{(W_j(t) + w_j)^{k+1} - W_j^{k+1}}{W(i, t)^{k+1}} 
\]

As before, this implies job \(j \in J_i(t)\) is processed at rate \(\ell_{ij} \nu_j(t)\).

**Virtual Utility of Jobs and Selfish Migration.** Consider a job \(j \in J_i(t)\) at time \(t\). Its virtual utility is defined as:

\[
\phi(j, t) = g_i(W_j(t) + w_j) \cdot \frac{\ell_{ij}}{W_j(t) + w_j}
\]

Using this virtual utility, the jobs perform sequential best response dynamics, migrating to other machines if it improves its virtual utility. As before, this leads to a Nash equilibrium every step. Note that unlike SELFISH-MIGRATE where migrations can only happen on departure of jobs from the system, migrations can now also happen on arrival of jobs into the system. If a job moves out of a machine, the weights \(W_j(t)\) of other jobs on the machine either stay the same or decrease. Using Proposition 22, this implies the virtual utility of other jobs on the machine either remains the same or increases. Therefore, similar to Lemma 15, we easily get the monotonicity of the virtual utilities of jobs.
Lemma 23. For all jobs $j$, $\phi(j, t)$ is non-decreasing over the interval $t \in [r_j, C_j]$.

5.3 Analysis of SelfishMigrate-Energy

Since our speed scaling policy ensures that total weighted flow-time of the schedule is equal to the energy cost, we focus on bounding the total weighted flow-time of jobs.

Convex Programming Relaxation. Consider the following convex programming relaxation for the problem due to (Anand et al., 2012; Devanur and Huang, 2014). In this relaxation, there is a variable $s_{ijt}$ which indicates the speed at which job $j$ is processed at time $t$ on machine $i$. The constraints of $\text{CP}_{\text{primal}}$ state that each job needs to be completely processed and the speed of machine $i$ at time $t$ is equal to the sum of individual speeds of the jobs.

To describe the objective function we set up some notation.

\[
\text{TERM1} = \sum_j \sum_i \int_{t \geq r_j} (t - r_j) \cdot \frac{\ell_{ij}w_j}{p_j} \cdot s_{ijt} \, dt
\]

\[
\text{TERM2} = \sum_i \int_t f_i((1 + 3 \epsilon) s_{it}) \, dt + \sum_i \sum_j \int_{t \geq r_j} (f_i^*)^{-1}(w_j)s_{ijt} \, dt
\]

\[
\text{TERM3} = \sum_i \sum_j \int_{t \geq r_j} (f_i^*)^{-1}(w_j)s_{ijt} \, dt
\]

We now give a brief explanation on why the objective function lower bounds the optimal schedule within a factor of 2. See (Devanur and Huang, 2014; Anand et al., 2012) for a complete proof of this claim. The first term in the objective function lower bounds the weighted flow-time of jobs and is similar to the term in the $\text{LP}_{\text{new}}$. The second term corresponds to the energy cost of the schedule. Here we use the fact that we analyse our algorithm in the resource augmentation model. Hence, the offline benchmark pays a cost of $f_i((1 + \epsilon) \cdot s_{it})$ for running at a speed of $s_{it}$. The third term is a lowerbound on the total cost any optimal solution has to pay to schedule a job $j$, assuming that the job $j$ is the only job present in the system. This term is needed, as we do not explicitly put any
constraints to forbid simultaneous processing of job $j$ across machines. Clearly, without this term, $\text{CP}_{\text{primal}}$ has a huge integrality gap as a single job can be processed to an extent of $\frac{1}{m}$ simultaneously on all machines. The function $f^*_i$ is the Legendre-Fenchel conjugate of function $f$ and is defined as $f^*(\beta) = \max_s\{s \cdot \beta - f(s)\}$. See Devanur and Huang (2014) for more details.

Minimize $\text{TERM1} + \text{TERM2} + \text{TERM3}$ \hspace{1cm} (\text{CP}_{\text{primal}})

$$\sum_i \int_{t \geq r_j} \frac{\ell_{ij} \cdot s_{ijt}}{p_j} \geq 1 \quad \forall j$$

$$\sum_{j, t \geq r_j} s_{ijt} = s_{it} \quad \forall i, t$$

$$s_{ijt} \geq 0 \quad \forall i, j, t : t \geq r_j$$

We write the dual of $\text{CP}_{\text{primal}}$ following the framework given in (Devanur and Huang, 2014). Similar to the dual of weighted flow-time, we have a variable $\beta_{it}$ for each machine $i$ and time instant $t$, and a variable $\alpha_j$ for each job $j$.

Max $\sum_j \alpha_j - \sum_i \int_i f^*_i \left(\frac{\beta_{it}}{1 + 3\epsilon}\right) dt$ \hspace{1cm} (\text{CP}_{\text{dual}})

$$\frac{\ell_{ij} \alpha_j}{p_j} - \beta_{it} \leq \frac{\ell_{ij} w_j}{p_j} (t - r_j) + (f^*_i)^{-1}(w_j) \quad \forall i, j, t : t \geq r_j$$

$$\alpha_j \geq 0 \quad \forall j$$

$$\beta_{it} \geq 0 \quad \forall i, t$$

We need the following simple observation regarding $f$ and $f^*$ for the rest of the analysis.

**Lemma 24.** For any increasing strictly convex function $f$ with $f(0) = 0$, let $g = f^{-1}$ and let $f^*$ be its Legendre-Fenchel conjugate. Then $f^* \left(\frac{w}{g(w)}\right) \leq w$, or $\frac{w}{g(w)} \leq (f^*)^{-1}(w)$.

**Proof.** From the definition of $f^*$, it is enough to show that for all $x, w$, we have: $\frac{wx}{g(w)} - f(x) \leq w$. Consider the following two cases. If $x \leq g(w)$, then the condition is trivially true. Now consider the case when $x \geq g(w)$. It follows that $f(x)/x$ is non-decreasing from...
convexity of $f$ and the fact $f(0) = 0$. Hence we have $f(x)/x \geq f(g(w))/g(w) = w/g(w)$, which completes the proof.

**Instantaneous Delay and Setting Variables.** Recall that each machine runs S-LAPS ($k$) with $k = 1/\epsilon$. Let $i^* = \sigma(j, t)$ denote the machine to which job $j$ is assigned at time $t$. We define the instantaneous weighted delay induced by job $j$ on jobs ahead of $j$ in its virtual queue (the set $J_j(t)$) and the weighted delay incurred by job $j$ due to jobs in $J_j(t)$ as follows:

$$
\delta_j(t) = \frac{1}{g_{i^*}(W(i^*, t))} \cdot \left( \sum_{j' \in J_j(t)} (w_{j'} \cdot \nu_j(t) + w_j \cdot \nu_{j'}(t)) + w_j \cdot \nu_j(t) \right)
$$

Define $\Delta_j = \int_{t=r_j}^{t=C_j} \delta_j(t)dt$ as the cumulative weighted delay induced by $j$ on jobs ahead of it in its virtual queue. Again, it is easy to see that $\sum_j w_j F_j = \sum_j \Delta_j$.

We now perform the dual fitting. We set $\beta_{it} = \frac{1}{k} \cdot \frac{W(i, t)}{g(i, W(i, t))}$ and $\alpha_j = \frac{1}{k+2} \Delta_j$. As before, we have:

$$
\sum_j \alpha_j - \sum_{i,t} f^*_i \left( \frac{\beta_{it}}{1 + 3\epsilon} \right) \geq \sum_j \frac{\Delta_j}{k+2} - \sum_{i,t} \epsilon \cdot \frac{W(i, t)}{1 + 3\epsilon} \quad [f^*_i \text{ is convex and Lemma 24}]
$$

$$
= \epsilon \left( \sum_j \frac{\Delta_j}{1 + 2\epsilon} - \sum_{i,t} \frac{W(i, t)}{1 + 3\epsilon} \right)
$$

$$
= \epsilon \cdot \sum_j w_j F_j \cdot \left( \frac{1}{1 + 2\epsilon} - \frac{1}{1 + 3\epsilon} \right) = O(\epsilon^2) \sum_j w_j F_j
$$

Since the energy cost of SefishMigrate-Energy is equal to total weighted flow-time, we get a $O(1/\epsilon^2)$-competitive algorithm with a speed-augmentation of $(1 + \epsilon)$.

**Delay Lemmas.** Similar to the delay lemmas for total weighted flow-time, we establish corresponding delay lemmas. Define for any $t^* \in [r_j, C_j]$ the following:

$$
\Delta_j^1(t^*) = \int_{t=r_j}^{t=t^*} \delta_j(t)dt \quad \text{and} \quad \Delta_j^2(t^*) = \int_{t=t^*}^{t=C_j} \delta_j(t)dt
$$
The following lemma bounds the quantity $\Delta_j^1(t^*)$; the proof is identical to the proof of Lemma 18:

**Lemma 25.** For any time instant $t^* \in [r_j, C_j]$ and for any job $j$,

$$
\Delta_j^1(t^*) \leq (k + 2) \cdot w_j \cdot (t^* - r_j)
$$

Let $p_j(t^*) = \int_{t^*}^{C_j} \ell_{\sigma(j,t^*)} \cdot \nu_j(t)dt$ denote the residual size of job $j$ at time $t^*$. Next we establish the corresponding Second Delay Lemma.

**Lemma 26.** For any time instant $t^* \in [r_j, C_j]$ and for any job $j$, let $t^* = \sigma(j,t^*)$ denote the machine to which job $j$ is assigned at time $t^*$. Then:

$$
\Delta_j^2(t^*) \leq \frac{k + 2}{k + 1} \cdot \frac{p_j(t^*)}{\phi(j,t^*)} \leq \frac{k + 2}{k + 1} \cdot \frac{p_j}{l_{t^*j}} \cdot \frac{W_j(t^*) + w_j}{g_{t^*}(W_j(t^*) + w_j)}
$$

**Proof.**

$$
\Delta_j^2(t^*) = \int_{t^*}^{C_j} \frac{1}{g_{t^*}(W(i^*, t))} \cdot \left( \nu_j(t) \cdot (w_j + W_j(t)) + w_j \cdot \left( \sum_{j' \in J_j(t)} \nu_j(t) \right) \right) dt
$$

$$
= \int_{t^*}^{C_j} \frac{1}{g_{t^*}(W(i^*, t))} \cdot \nu_j(t) \cdot \left( w_j + W_j(t) + \frac{w_j \cdot W_j(t)^{k+1}}{(W_j(t) + w_j)^{k+1} - W_j(t)^{k+1}} \right) dt
$$

$$
\leq \int_{t^*}^{C_j} \frac{1}{g_{t^*}(W(i^*, t))} \cdot \nu_j(t) \cdot \left( w_j + W_j(t) + \frac{w_j + W_j(t)}{k + 1} \right) dt \quad \text{[Prop. 17]}
$$

$$
\leq \frac{k + 2}{k + 1} \cdot \int_{t^*}^{C_j} \frac{1}{g_{t^*}(W_j(t) + w_j)} \cdot \nu_j(t) \cdot (w_j + W_j(t)) dt \quad \text{[g is increasing]}
$$

$$
= \frac{k + 2}{k + 1} \cdot \int_{t^*}^{C_j} \ell_{t^*j} \cdot \nu_j(t) \cdot \frac{1}{\phi(j,t)} dt \leq \frac{k + 2}{k + 1} \cdot \frac{1}{\phi(j,t^*)} \cdot \int_{t^*}^{C_j} \ell_{t^*j} \cdot \nu_j(t) dt \quad \text{[Lem 23]}
$$

$$
= \frac{k + 2}{k + 1} \cdot \frac{p_j(t^*)}{\phi(j,t^*)} \leq \frac{k + 2}{k + 1} \cdot \frac{p_j}{l_{t^*j}} \cdot \frac{W_j(t^*) + w_j}{g_{t^*}(W_j(t^*) + w_j)}
$$

\[ \square \]

From the previous two lemmas we get:
Lemma 27. For any time instant \( t \in [r_j, C_j] \) and job \( j \) that is assigned to machine \( i^* = \sigma(j, t) \), we have:

\[
\Delta_j = \Delta_j^1(t) + \Delta_j^2(t) \leq (k + 2) \cdot w_j \cdot (t - r_j) + \frac{k + 2}{k + 1} \cdot \frac{p_j}{\ell_{ij}} \cdot \frac{W_j(t) + w_j}{g_{t^*}(W_j(t) + w_j)}
\]

Checking the Feasibility of \( CP_{\text{dual}} \) Constraints. Now it remains to prove that constraints of \( CP_{\text{dual}} \) are satisfied. To see this, fix job \( j \) and time instant \( t \). We consider two cases.

Case 1: Machine \( i \neq \sigma(j, t) \). Then

\[
\alpha_j - \frac{p_j}{\ell_{ij}} \beta_{it} = \frac{\Delta_j}{k + 2} - \frac{p_j}{\ell_{ij}} \cdot \frac{1}{k} \cdot \frac{W(i, t)}{g_i(W(i, t))}
\]

\[
\leq w_j \cdot (t - r_j) + \frac{p_j}{\ell_{ij}} \cdot \frac{1}{k + 1} \cdot \frac{W_j(t) + w_j}{g_{\sigma(j, t)}(W_j(t) + w_j)} - \frac{p_j}{\ell_{ij}} \cdot \frac{1}{k} \cdot \frac{W(i, t)}{g_i(W(i, t))} \tag{5.3}
\]

\[
\leq w_j \cdot (t - r_j) + \frac{p_j}{\ell_{ij}} \cdot \frac{1}{k + 1} \cdot \frac{W_j(t) + w_j}{g_{\sigma(j, t)}(W_j(t) + w_j)} - \frac{p_j}{\ell_{ij}} \cdot \frac{1}{k} \cdot \frac{(W(i, t) + w_j - w_j)}{g_i(W(i, t) + w_j)} \tag{5.4}
\]

\[
\leq w_j \cdot (t - r_j) + \frac{p_j}{\ell_{ij}} \cdot \frac{1}{k + 1} \cdot \frac{w_j}{g_i(W(i, t) + w_j)} \tag{5.5}
\]

\[
\leq w_j \cdot (t - r_j) + \frac{p_j}{\ell_{ij}} \cdot \frac{1}{k + 1} \cdot \frac{w_j}{g_i(w_j)}
\]

\[
\leq w_j \cdot (t - r_j) + \frac{p_j}{\ell_{ij}} \cdot (f^*_i)^{-1}(w_j)
\]

The inequality 5.3 holds due to Lemma 27. The equation 5.4 is true because \( g \) is increasing. The inequality 5.5 follows as the machine \( \sigma(j, t) \) maximizes virtual utility.

Next, we consider the more easier case.

Case 2: Machine \( i = \sigma(j, t) \). Then

\[
\alpha_j - \frac{p_j}{\ell_{ij}} \beta_{it} = \frac{\Delta_j}{k + 2} - \frac{p_j}{\ell_{ij}} \cdot \frac{1}{k} \cdot \frac{W(i, t)}{g_i(W(i, t))}
\]

\[
\leq w_j \cdot (t - r_j) + \frac{p_j}{\ell_{ij}} \cdot \frac{1}{k + 1} \cdot \frac{W_j(t) + w_j}{g_i(W_j(t) + w_j)} - \frac{p_j}{\ell_{ij}} \cdot \frac{1}{k} \cdot \frac{W(i, t)}{g_i(W(i, t))}
\]

\[
\leq w_j \cdot (t - r_j) \quad \text{[Proposition 22]}
\]
The inequality holds because of Lemma 27. This completes the proof.

5.3.1 Polynomial Power Functions

As a corollary of the above result, we get a $O(\gamma^2)$-competitive algorithm when each machine follows the power function $s^\gamma$.

**Corollary 28.** There is $O(\gamma^2)$-competitive non-clairvoyant algorithm for minimizing weighted flow-time plus energy, when each machine follows a polynomial power function $f(s) = s^\gamma$.

**Proof.** Consider the convex programming formulation $\text{CP}_{\text{primal}}$ and set $\epsilon = 1/\gamma$. Then, the CP is only $O(1)$ more than any optimal solution. Thus, we get $O(\gamma^2)$-competitive algorithm. \qed

5.4 Summary and Open Problems

In this chapter we presented a simple extension to the SELFISHMIGRATE framework for the objective of minimizing flow-time in the dynamic speed scaling setting. An important open question here is to extend the results to general $\ell_k$-norms of flow-time.

5.5 Notes

The chapter is based on joint work with Sungjin Im, Kamesh Munagala and Kirk Pruhs. A preliminary version of this work appeared in the 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014 (Im et al., 2014b).
6

Cost Aware Scheduling

6.1 Introduction

In the previous chapter we considered energy efficient scheduling in the dynamic speed scaling model. In this chapter, we take a different look at minimizing energy costs in data center settings. We introduce a new model where there is a cost involved in processing jobs that varies as a function of time. Before defining the problems formally and discussing the technical novelty, we present a few technological motivations for introducing the model.

Demand Response Models:

The electricity markets in large parts of United states are moving towards variable pricing. As noted in (Qureshi et al., 2009), in those parts of the U.S. with wholesale electricity markets, prices vary on an hourly basis and are often not well correlated at different locations. Moreover, these variations are substantial, as much as a factor of ten from one hour to the next. Several suppliers offer “Time Of Use” plans\(^1\), and often the electricity markets are too complex to give a simple thumb rule on which variation in cost depends. However, in general, the price depends on the resource used for generation: As the demand peaks, the cost goes up disproportionately as the suppliers have to rely on

\(^1\) http://www.pge.com/tariffs/electric.shtml
expensive and nonrenewable resources like coal to meet the demand. In this sense, the cost of electricity is also an indicator of how “green” its generation is and its impact on environment.

This variation in prices of electricity offers opportunities for large scale system designers to cut down their electricity expenses by moving their work load both in space and time. Note that in contrast to energy efficient computing, the purpose of this line of work is not to reduce the amount of energy consumed per unit of work, but to reduce the cost for doing the work. As noted above, this often translates to a greener way of getting the work done. In (Qureshi et al., 2009), the authors exploit the spatial nature of variation in electricity cost for scheduling, while in (Chase, 2010), the authors analyse a simple model to exploit the temporal nature of variation in electricity prices. The latter work is particularly relevant to this work: They consider a system at a single location executing a workload that is delay tolerant, such as processing batch jobs. They further consider a pricing model where cost of electricity varies between two levels, a base price and a peak price. They propose simple schemes to defer the workload to less expensive base price periods, and show experimentally that it smoothly trades off costs for delay.

In this chapter we theoretically model and analyze the impact of the temporal nature of variation in electricity prices on scheduling decisions. One of the important contributions of our work is to analyze the simplest model considered by authors in (Chase, 2010) from the theoretical perspective with worst case bounds.

Spot Pricing in Data Centers: An entirely different technological motivation for cost-aware scheduling comes from the Amazon EC2 cloud computing system, which allows users to rent virtual machines on the cloud for computational needs. Amazon offers various pricing schemes to rent machines, one of which is spot pricing. Spot pricing enables the users to bid for unused capacity, and prices get set based on supply and demand. Again, as in the previous example, the cost of renting the machine on EC2 varies dynamically over time,

http://www.eia.doe.gov

offering opportunities for optimizing the cost and QoS of batch jobs on such a system.

6.1.1 Our Model

In this chapter, we consider the single-machine preemptive scheduling framework. There is a set \( J \) of \( n \) jobs, where each job \( J_j \) has processing time \( p_j \), release time \( r_j \), and weight \( w_j \). We assume time is discrete, and the processing times and release times of the jobs are integers. There is a \textit{processing cost} function \( e(t) \): If a job \( J_j \) is scheduled at time \( t \), it incurs processing cost \( e(t) \). We assume \( e(t) \) is piecewise constant with polynomially many break-points, and takes \( K \) distinct values; our algorithmic results will depend on \( K \).

Given any schedule, the \textit{processing cost} of \( J_j \) denoted by \( E(j) \) is given by \( \int_{\tau} e(t)x_j(t)dt \), where \( x_j(t) \) indicates whether job \( J_j \) was scheduled at the instant \( t \); since we assume time is discrete, \( E(j) = \sum_{\tau} e(t)x_j(t) \). The completion time of job \( J_j \) denoted by \( C_j \) is the last time instant when this job is scheduled. The flow time of \( J_j \) is defined as \( F_j = C_j - r_j \). Our optimization objective will be to find a preemptive schedule that minimizes the sum of a QoS metric such as flow time or completion time, plus the total processing cost of all jobs. We consider the following variants, motivated by the application scenarios discussed above.

\textit{Online Problem: Weighted Flow Time:} In this setting, jobs arrive online and the cost function \( e(t) \) changes in an online fashion. The objective is to minimize \( \sum_{J_j \in J}(w_jF_j + E(j)) \) – the sum of weighted flow time and processing cost.

\textit{Offline Problem: Weighted Completion Time:} In the offline case, the job arrival times and the function \( e(t) \) are known in advance. We assume all release dates are poly-bounded. Our objective is to minimize the sum of weighted completion times and processing cost. We consider the problem without release dates, \( 1|\text{pmtn}| \sum_{J_j \in J}(w_jC_j + E(j)) \) and with release dates, \( 1|r_j, \text{pmtn}| \sum_{J_j \in J}(w_jC_j + E(j)) \).
6.1.2 Our Results

In this chapter, we initiate the study of scheduling problems with the objective of minimizing the processing costs plus well-known QoS guarantees. We show that these problems are significantly different from their counterparts without processing costs, hence requiring new algorithmic techniques.

Online Problem. We first consider the online problem of minimizing weighted flow time plus processing cost. Our upper bounds assume the algorithm does not know the future job arrivals and the future cost function \( e(t) \), and proceed via speed augmentation analysis, where we give the algorithm extra processing speed compared to \( \text{OPT} \) for the purpose of analysis. The holy grail of speed augmentation analysis is to design a so-called scalable algorithm: For any \( \epsilon > 0 \), the algorithm is \((1 + \epsilon)-speed\), \( O(\text{poly}(\frac{1}{\epsilon})) \)-competitive. Recall that \( K \) is the number of distinct values taken by the function \( e(t) \).

- In Section 6.2.1, we show that no deterministic online algorithm can be constant competitive even for unit length, unit weight jobs, when \( e(t) \) is known in advance, and when it takes only \( K = 2 \) distinct values.

- In Section 6.2.2, we present our main result. We show a scalable \((1 + \epsilon)-speed\), \( O(\text{poly}(\frac{1}{\epsilon})) \)-competitive algorithm for \( K = 2 \); we term this problem Two-Cost.

- We complement the positive result by showing (in Section 6.2.1) that when \( e(t) \) takes \( K > 2 \) distinct values, there can be no scalable algorithm: In order to achieve a competitive ratio of \( O(1) \), the algorithm needs speed strictly greater than \( K - 1 \).

Our algorithm for Two-Cost in Section 6.2.2 is the most natural one: Always schedule at low cost time instants. For high cost time instants, if the total flow time accumulated since the last scheduling decision is at least the cost of processing, then schedule using the highest density first priority rule, else wait. We outline the idea behind the analysis in Section 6.2.3. The analysis is complicated by the non-uniform nature of the problem - the behavior of the algorithm is different in the high and low cost instants, and these instants
themselves arrive in an online fashion. The key decision that the algorithm has to make is whether to schedule a job at the current step in a high cost time instant, or wait for a low cost time instant that may arrive soon in the future. However, waiting poses a risk in that jobs could arrive in the future and create a huge backlog at the low cost time instants. To partially mitigate the backlogging effect, we resort to speed augmentation for the analysis, and show that this is necessary as well.

Speed augmentation and potential functions have proven to be useful techniques in the analysis of online algorithms for weighted flow time problems. See for example, scheduling policies on unrelated machines (Chadha et al., 2009) and for the speed scaling problems (Bansal et al., 2007a, 2011; Gupta et al., 2010b). These potential functions follow a similar template (the so-called standard potential function Im et al. (2011b)), and are defined in terms of the future online cost (or cost-to-go) of the algorithm assuming no more jobs arrive in the system and how far the online algorithm is behind the optimal schedule in work processed. Due to the online nature of the cost function $e(t)$, it is not clear how to even define a standard potential function for our problem. We instead proceed via simplifying the input, revealing structural properties of OPT. We then show a majorization property (Theorem 37) of our schedule relative to the optimal schedule, in the sense that the schedule always lags the optimal one. We use this characterization repeatedly and split the time horizon into suitably defined phases, and use a simple potential function argument within each phase. In effect we show that an algorithm that cannot estimate cost-to-go for its scheduling decision is competitive \textit{even if} the costs in the future vary arbitrarily. We also believe Theorem 37 will be of independent interest in other scheduling problems.

\textit{Offline Problem.} In the offline case, we consider the objective of minimizing weighted completion time plus processing cost. We assume the weights and processing times are poly-bounded. We show the following results.

- In 6.3.1, we present a poly-time optimal algorithm for $1|\text{pmtn}|\sum_j(C_j+E(j))$, and a 4 -approximation to $1|\text{pmtn}|\sum_j(w_jC_j+E(j))$. 

81
• In 6.4.1, we present a quasi-polynomial time $O(1)$-approximation algorithm for $1|r_j, pmtn| \sum_j (w_jC_j + E(j))$ when $e(t)$ takes $K = 2$ distinct values (the Two-Cost case). Our framework can be extended to get similar bounds when $K > 2$; we omit the details.

• In 6.4.2, we present a $O(\frac{1}{\epsilon})$ approximation to $1|r_j, pmtn| \sum_j (w_jC_j + E(j))$ with $1 + \epsilon$ speed augmentation.

Our algorithms involve careful dynamic programming to compute the periods of time when the optimal algorithm schedules jobs (Lemma 44). In order to decide the order in which to schedule jobs, we observe that our problem generalizes scheduling problems with limited machine availability (Epstein et al., 2010; Schmidt, 1998). In these problems, the machine is unavailable for certain intervals of time that is known in advance and the goal is to optimize weighted completion time. In the absence of release dates and for the objective of minimizing the weighted completion time, Epstein et al. (Epstein et al., 2010) design a universal sequence, which is oblivious to the intervals in which the machine fails, and which is a 4-approximation to the optimal algorithm that knows the machines failures in advance. We combine some of the ideas used in (Epstein et al., 2010) with several structural properties of optimal schedule we reveal to obtain our approximation guarantees (Theorems 47, 49). For the final result (with speed augmentation), we formulate a linear relaxation very similar to those for completion time on parallel machines due to (Hall et al., 1996). In our case, this LP has unbounded integrality gap without speed augmentation, highlighting the technical difference to the version without processing costs (Theorem 50).

6.1.3 Histrory

Use of speed augmentation for the analysis of online scheduling problems, particularly flow time objectives, was first considered in (Kalyanasundaram and Pruhs, 2000). Since then, several as have shown speed augmentation results for highest density first (HDF) (Becchetti et al., 2006), $L_p$ norms of flow time (Bansal and Pruhs, 2004), and flow time on unrelated machines (Chadha et al., 2009).
The line of research closest to our model is dynamic speed scaling and its variants that we saw in the last chapter. As observed above, however, our model is fundamentally different from speed scaling (or power down) models, since in our case, electricity costs vary with time in a non-monotone fashion. It would be interesting to study the effect of combining speed scaling with our model.

There has been some prior work on modeling time-varying costs in scheduling. Im, Moseley and Pruhs (Im et al., 2012b) study online scheduling to minimize $\sum_j w_j g(F_j)$ where $g$ is an arbitrary non-decreasing cost function. We differ in two fundamental ways: First, our cost function is not non-decreasing; it is easy to see that finishing a job earlier does not necessarily incur less cost. Secondly, cost of processing a job not only depends on the completion time, but also on the time periods where the processing is done. In (Pruhs and Stein, 2010) Pruhs and Stein consider the problem where jobs arrive over time on a set of speed scalable processors and lost income is a function of job delay. They consider maximizing income obtained minus the energy costs. There are two main differences with our work: Unlike us, they assume the cost of energy is fixed and known, and furthermore, since they are maximizing profit, they have the option of not processing jobs.

6.2 Online Algorithms for Weighted Flow Time

In this section, we devise online algorithms for minimizing the sum of weighted flow time and processing cost on a single machine (with preemption). Recall that we denote the weights of the jobs by $w_i$, and the processing times by $p_i$. Our algorithms will use speed augmentation to be competitive - this means that to show a $O(1)$-competitive ratio, we pretend the algorithm runs on a faster machine than the optimal solution; the extra speed trades off with the competitive ratio. The precise notion of speed augmentation will be defined later.

We first present lower bounds on the achievable competitive ratio. In showing our lower bounds, we only assume the job arrivals are online; the cost function $e(t)$ is known in advance. These lower bounds lead us to consider the case where $e(t)$ takes on only two
distinct values; we call this special case Two-Cost. For this problem, we show that for any $\epsilon > 0$, there is a $(1 + \epsilon)$ speed, $O(1/\epsilon^3)$ competitive online algorithm where we assume both job arrivals and the cost function $e(t)$ are online in nature.

6.2.1 Lower Bounds

In this section, we first show that no online algorithm can have competitive ratio independent of the values taken by cost function $e(t)$, even when all jobs have the same weight and unit processing length, and when the cost function $e(t)$ is known in advance. Furthermore, we show that when there are $K$ distinct values taken by the cost function $e(t)$, any online algorithm has to be augmented with speed at least $K - 2$ in order to achieve a competitive ratio that is independent of values taken by $e(t)$.

**Theorem 29.** No deterministic online algorithm can have a competitive ratio independent of the values taken by $e(t)$, even when all jobs have unit length and equal weight and $e(t)$ takes only $K = 2$ distinct values.

**Proof.** Let $e(t) = \beta$ in the interval $[1, \ldots \sqrt{\beta}]$ and $e(t) = 1$ elsewhere. The adversary releases a unit length, unit weight job at each time instant $t \in [1, \ldots \sqrt{\beta}]$. Let $A$ be any online algorithm. Consider the number of jobs in the queue of $A$ at time $t = \sqrt{\beta}$. If $A$ has more than $\beta^{1/4}$ jobs then the adversary releases one job at each time instant $t > \sqrt{\beta}$. For this input, the optimal offline algorithm will process each job released in the interval $[1, \ldots \sqrt{\beta}]$ by paying a processing cost of $\beta$, hence number of jobs it has at any time is at most one. However, $A$ accumulates $\beta^{1/4}$ jobs by the time $t = \sqrt{\beta}$ which it cannot clear subsequently. Hence there are at least $\beta^{1/4}$ jobs in its queue at every time instant $t > \sqrt{\beta}$, incurring a cost of $\beta^{1/4}$ towards flowtime at each time step. Therefore, the competitive ratio of $A$ is at least $\beta^{1/4}$. Next, consider the case when $A$ has less than $\beta^{1/4}$ jobs at time $t = \sqrt{\beta}$. In this case, the adversary will not release any more jobs. For this instance, the optimal offline algorithm will not process any jobs in the interval $[1, \ldots \sqrt{\beta}]$ and processes all jobs in the low cost time instants following $t > \sqrt{\beta}$ incurring a total cost of $O(\beta)$. The competitive ratio of $A$
is at least $\beta^\frac{1}{2}$, since the algorithm pays at least $\beta^\frac{5}{4}$ towards the processing cost.

**Definition 30.** The speed of a machine is the number of units of jobs it processes per time step (where we assume the optimal solution uses a machine of speed one). The speed threshold of an online algorithm is the minimum speed a machine should have, so that the algorithm run on this machine has a competitive ratio independent of the magnitude of the $e(t)$ values.

**Theorem 31.** When $e(t)$ takes $K$ distinct values, the speed threshold of any online deterministic algorithm is at least $K - 1$. This holds even when $e(t)$ is known in advance and all jobs have unit length and equal weight.

**Proof.** For $\beta > 2 \log K$, consider the cost function $e(t)$ defined as follows. $e(t) = (2^\beta)^t$ for $t \in [0, 1, \ldots K - 3]$. $e(t) = 0$ at $t = K - 2$ and $e(t)$ takes value $\infty$ elsewhere. Let $\epsilon > 0$ be a constant. Let $A$ be any online algorithm running on a machine with a speed of $K - 1 - \epsilon$; assume that $A$ processes at least $\epsilon$ units per time instant if it decides to process at all.

The adversary injects one job per time step for each $t \in [0, 1, \ldots K - 2]$. Suppose $A$ does not schedule any jobs before time $K - 2$. In this case, it will have $K - 1$ jobs in its queue at the time $K - 2$. Since only $K - 1 - \epsilon$ jobs can be processed at time $K - 2$, at least $\epsilon > 0$ fraction of a job has be processed at $t > K - 2$, hence paying a cost $\infty$.

Therefore, we can assume that $A$ will processes at least an $\epsilon$ fraction of some job at time $t < K - 2$; consider the earliest such time instant and denote it as $t'$. Given this value of $t'$, the adversary stops injecting jobs after time $t'$. On this new input, the processing cost paid by $A$ running on a machine with a speed of $K - 1 - \epsilon$ is at least $\epsilon 2^{\beta t'}$. On the other hand, the optimal offline algorithm pays a cost of $\sum_{t=0}^{t'-1} (2^\beta)^t + \Theta(K)$ on this input, since it processes jobs at times $t < t'$ and keeps the last job for time $t = K - 2$. For $\beta > 2 \log K$, it is easy to check that the competitive ratio of $A$ on this input sequence is $\Omega(2^\beta/2)$. \qed
6.2.2 The Two-Cost Problem

Motivated by the negative result in Theorem 31, we consider the the case where the function $e(t)$ takes only two distinct values. We assume without loss of generality that $e(t)$ takes either a value of 1 or $\beta$ at all the time instants. However, Theorem 29 implies that even in this case, there is no algorithm with competitive ratio independent of the $e(t)$ values. We therefore resort to speed augmentation to analyze the performance of our algorithm.

We first describe the analysis technique, which is implicit in previous work (Chadha et al., 2009; Bansal et al., 2007b, 2011). Let $OPT$ denote both the optimal offline algorithm, as well as its value.

Given any online algorithm and input sequence, there are two decisions the algorithm has to make every step:

**Time slot selection:** This policy decides which time slots to schedule jobs - we term these *active* time slots.

**Job selection policy:** Decides which job to schedule in each active time slot.

**Definition 32.** Given an algorithm $A$ and speed $s$, we say that algorithm $B$ is a $s$-speed simulation of $A$ if:

- The active time slots of $A$ and $B$ are the same (or $B$ simulates $A$).
- The job selection policy of $B$ is same as $A$; however, $B$ can process $s$ units of jobs in every active time slot.

**Definition 33.** An online algorithm $A$ is said to be $s$-speed, $c$-competitive if there is a $s$-speed simulation of $A$ that is $c$ competitive against $OPT$.

We will show the following theorem in the sequel.

**Theorem 34.** Two-Cost has a $(1 + \epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive algorithm.
6.2.3 Proof Outline

We design our algorithm for the case when jobs have unit length with arbitrary weights and at each time step a single job is released. We later show how to convert this algorithm to handle jobs with arbitrary lengths using the ideas which have become standard now. For unit length jobs, the job selection policy of any algorithm is simple: Schedule that job $J_i$ from the current queue with highest density or weight. This is the well-known Highest Density First (HDF) policy. Our overall online algorithm for unit length jobs is the most natural one: Always schedule using HDF in low cost instants. For high cost instants, if the total flow time accumulated since the last scheduling decision is at least $\beta$, then schedule using HDF, else wait. We call this algorithm Balance. The hard part in defining a (standard) potential function is the non-uniformity in the processing cost. Instead, we first transform and simplify the input so that we only have to deal with unit length jobs, only one of which arrives per step.

The crux of our analysis is a majorization property of the HDF schedule, Theorem 37: If an online algorithm processing using HDF always lags another algorithm in terms of number of units processed, but eventually catches up, then if the initial weight of jobs in the queue of the first algorithm was smaller, the final weight will be smaller as well. We show that Balance always lags $OPT$ in terms of number of jobs processed, hence the majorization result directly bounds the processing cost paid by Balance (Lemma 39).

To bound the flow time, we perform a speed augmentation analysis. Again, the analysis is complicated by the non-uniformity in processing costs between low and high cost time instants. We instead divide time into phases where the augmented Balance lags $OPT$. Using our majorization result, we show it is sufficient to analyze each phase separately. We now construct a simple potential function and argue about the amortized cost, completing the proof(Lemma 40).
6.2.4 Simplifying the Input

By scaling the input, we can assume that $e(t)$ takes values either 1 or $\beta$. We also assume that processing times and release times of jobs take integer values. We assume jobs are released at the beginning of a time slot. Our scheduling policies will be based on considering the weight of jobs in the queue during the time slot, and the processing happens at the end of the time slot.

We call a time instant $t$ as high cost time instant if $e(t) = \beta$ in the interval $[t, t+1)$. Otherwise, we call it as low cost time instant. We assume without loss of generality that $e(t)$ changes only at integral values of $t$. Thus every time instant is either a low cost time instant or a high cost time instant.

**Step 1: Unit Length Jobs.** The following lemma is an easy consequence of similar results in Chadha et al. (2009); Bansal et al. (2011); Becchetti and Leonardi (2004). The proof follows by replacing each job $J_i$ with $p_i$ jobs of unit length and weight $w_i/p_i$.

**Lemma 35.** If an online algorithm $A$ is $s$-speed $c$-competitive for minimizing the objective $\sum_j w_j F_j$ for unit length jobs, then it is $(1 + \epsilon)s$-speed $(1 + \frac{1}{c})c$-competitive when jobs have arbitrary length.

The above lemma allows us to focus on unit length jobs in designing the online algorithm. For unit length jobs, given the set of active slots, it is easy to characterize the job selection policy: If a slot is active, the algorithm will simply schedule that job $J_i$ from its queue with highest density or weight per unit length, $w_i$. This is the well-known Highest Density First (HDF) policy. Note however that even for unit length jobs, Theorem 29 shows there is no 1-speed algorithm with competitive ratio independent of $\beta$. We therefore need to use a speed augmentation analysis even for this case. We redefine $OPT$ to be the optimal offline algorithm for this new problem instance (with unit length jobs).

**Step 2: Modifying Release Dates.** From now on, we assume unit length jobs. Consider a SUPER-OPT that ignores processing costs, and simply schedules jobs at all time instants
using HDF. Let $r_i$ denote the release time of job $J_i$ and let $q_i$ denote the processing time on Super-Opt. It is easy to check that $J_i$ will only get processed later than $q_i$ in any algorithm that uses HDF as its job selection policy. Furthermore, $q_i$ is readily computable in an online fashion. Therefore, we can assume the release time of the job is $q_i$ instead of $r_i$. Clearly, any competitive algorithm with this assumption is also competitive assuming the original release times. Therefore we assume at most one job is released per time step.

**Step 3: Modifying the optimal schedule.** Given any algorithm $A$, let $W^A(t)$ denote the total weight of jobs in $A$’s queue during time $t$. The proof of the following claim follows easily from the observation that making $OPT$ process a job has cost $\beta$, while the total weight of jobs in the queue contributes to the flow time. As a consequence, if $W^{OPT}(t) \geq \beta$, we can assume $OPT$ schedules at time $t$.

**Claim 36.** With $O(1)$ loss in competitive ratio, we can assume that in any interval $I = [s, d]$ where $OPT$ does not process jobs, $\sum_{t \in I} W^{OPT}(t) < \beta$.

### 6.2.5 Online Algorithm Balance

The online algorithm Balance is characterized by the following two rules. Here $W^A(t)$ denotes the total weight of jobs in Balance’s queue during time $t$.

**Time slot selection Policy:** If $e(t) = 1$, then mark $t$ as active. If $e(t) = \beta$ then let $t'$ be the last active time instant. Mark $t$ as active if $\sum_{u \in (t', t]} W^A(u) \geq \beta$.

**Job selection Policy (HDF):** If $t$ is active, then among the set of jobs available at the time $t$, schedule the one with highest weight.

### 6.2.6 Analysis of Balance

We begin this section by showing some important structural properties of the schedule produced by Balance, which we use subsequently in our analysis.

**Majorization Property of HDF.** Before we analyze Balance, we establish an important property of scheduling jobs in HDF. For a scheduling algorithm $A$ processing unit
length jobs in the interval \([t_1, t_2]\), let \(Q^A(t)\) denote the set of jobs \(A\) has at the time \(t\). Let \(Q^A_{\geq w}(t)\) denote the subset of those jobs with weight at least \(w\). Let \(N^A(t_1, t_2)\) denote the number of jobs \(A\) has scheduled in the interval \([t_1, t_2]\). Then we have the following theorem about scheduling jobs in HDF, which may be of independent interest.

**Theorem 37.** Let \(A\) be a scheduling algorithm which processes unit length jobs using the HDF job selection policy, and \(B\) be any other scheduling algorithm on the same input. Suppose \(\forall J_i \in Q^B(t_1), |Q^A_{\geq w_i}(t_1)| \leq |Q^B_{\geq w_i}(t_1)|\). If \(N^A(t_1, t_2) = N^B(t_1, t_2)\) and \(\forall t \in [t_1, t_2], N^A(t_1, t) \leq N^B(t_1, t)\), then \(\forall J_i \in Q^B(t_2), |Q^A_{\geq w_i}(t_2)| \leq |Q^B_{\geq w_i}(t_2)|\). Further, \(W^A(t_2) \leq W^B(t_2)\).

**Proof.** We prove this by contradiction. Suppose at time \(t_2\), there is a job \(J_i \in Q^B(t_2)\) such that \(|Q^A_{\geq w_i}(t_2)| > |Q^B_{\geq w_i}(t_2)|\). Consider the set of jobs processed by \(A\) in the interval \([t_1, t_2]\). If the weight of all these jobs is at least \(w_i\), then since both algorithms process equal number of jobs in \([t_1, t_2]\) and \(B\) had more jobs initially of weight at least \(w_i\), it must have more jobs with at least weight \(w_i\) at \(t_2\). This is an immediate contradiction.

Next, consider the case where \(A\) processes a job of weight less than \(w_i\) in the interval \([t_1, t_2]\). Let \(t' \in [t_1, t_2]\) be the last time instant when \(A\) scheduled a job with weight less than \(w_i\). Since \(A\) schedules jobs using HDF, it must be the case that \(|Q^A_{\geq w_i}(t')| = 0\). Now observe that \(A\) processes at least as many jobs as \(B\) in the interval \([t', t_2]\). If \(J_{\geq w_i}(t', t_2)\) denotes the set of jobs with weight greater than \(w_i\) released in the interval \([t', t_2]\) then \(|Q^A_{\geq w_i}(t_2)| = |J_{\geq w_i}(t', t_2)| - N^A(t', t_2)\). Since, \(N^A(t', t_2) \geq N^B(t', t_2)\), we have \(|Q^A_{\geq w_i}(t_2)| \leq |Q^B_{\geq w_i}(t_2)|\). Hence we get a contradiction.

Finally, summing over weights of all the jobs in the queues of \(A\) and \(B\) we obtain \(W^A(t_2) \leq W^B(t_2)\).

Bounding the Processing Cost. From this point on, we will use \(\mathcal{A}\) to denote the schedule produced BALANCE. The following lemma is the crucial property of BALANCE: Compared to \(OPT\), \(\mathcal{A}\) always lags in total processing done.
Lemma 38. In the schedule $\mathcal{A}$ produced by BALANCE, $\forall t \in [0, t], N^A(0, t) \leq N^{OPT}(0, t)$.

**Proof.** For the sake of contradiction, let $t_1$ be the first time instant when $N^A(0, t_1) > N^{OPT}(0, t_1)$ and $t_2 < t_1$ be the last time instant when $A$ scheduled a job. By the definition of $t_1$ and $t_2$, we note that both $OPT$ and $A$ do not process any jobs in the interval $(t_2, t_1]$ and $N^A(0, t_2) = N^{OPT}(0, t_2)$. Moreover, in the interval $[0, t_2]$ $A$ lags $OPT$; that is, $\forall t \in [0, t_2], N^A(0, t) \leq N^{OPT}(0, t)$. Since BALANCE processes jobs in HDF, we apply Lemma 37 over the interval $[0, t_2]$ to claim that $W^A(t_2) \leq W^{OPT}(t_2)$. If $t_1$ is a low cost time instant, we conclude that $OPT$ will also process a job since $W^{OPT}(t_1) > 0$. Next, consider the case when $t_1$ is a high cost time instant. Since $A$ is processing a job at $t_1$, from the description of BALANCE, we have $\sum_{t=t_2+1}^{t_1} W^A(t) \geq \beta$. However, $W^{OPT}(t_2) \geq W^A(t_2)$ and $OPT$ does not process jobs in the interval $(t_2, t_1]$, then it must be the case that $\sum_{t=t_2+1}^{t_1} W^{OPT}(t) \geq \beta$. Therefore, by Claim 36 $OPT$ also processes at $t_1$. This completes the proof. □

Let $E^A, E^{OPT}$ denote the total processing cost of BALANCE and $OPT$ respectively. The processing cost $E^A$ can now be bounded using the above lemmas.

**Lemma 39.** Total processing cost of $\mathcal{A}$, $E^A \leq E^{OPT}$.

**Proof.** Proof follows from Lemma 38. For the sake of contradiction, let $t$ be the first time instant when total processing cost of $\mathcal{A}$ is more than $OPT$. We note that in the interval $[0, t]$, when there are jobs to process, $\mathcal{A}$ has no idle time slots during the low cost time instants. Therefore, $\mathcal{A}$ schedules at least as many jobs as $OPT$ in the low cost time instants of the interval $[0, t]$. Since the total processing cost of $\mathcal{A}$ is more than $OPT$ at time $t$, $\mathcal{A}$ must have scheduled more jobs in high cost time instants compared to $OPT$. This implies that total number of jobs scheduled by BALANCE in the interval $[0, t]$ is greater than that of $OPT$, which contradicts Lemma 38. □

**Flow Time via Speed Augmentation.** We now analyze the schedule produced by BALANCE using speed augmentation. In particular, we consider a class of algorithms, SIMULATE-BALANCE($s$) that uses the same active time-slots as BALANCE (in that sense, it
simulates BALANCE). However, on each active time slot of BALANCE, the new algorithm schedules at most $s$ units of jobs from its own queue using the HDF policy.

It follows directly from Lemma 39 that the processing cost of SIMULATE-BALANCE$(s)$ is at most $s \cdot E^{OPT}$. In the sequel, we bound the flow time of SIMULATE-BALANCE$(1 + \epsilon)$ against the flow time of OPT, which we denote $F^{OPT}$. We will use SIMULATE-BALANCE to mean SIMULATE-BALANCE$(1 + \epsilon)$, where the factor $(1 + \epsilon)$ will be implicit.

In order to bound the weighted flow time, we first split the contribution of the weighted flow time to individual time steps (and hence time intervals). We treat time as a discrete quantity here with each time instant $t$ denoting the time interval $[t, t+1)$. For an algorithm $B$, let $F^B$ denote the total weighted flow time, and let $W^B(t)$ denote the weight of jobs in queue of $B$ at time instant $t$ (this excludes the job getting processed at the time step $t$). Then, we have:

$$F^B = \sum_{t \geq 0} W^B(t) + \sum_j w_j.$$

Let $F^A$ denote the total weighted flow time of SIMULATE-BALANCE. Let $W^A(t)$ denote the total weight of the jobs in the queue of SIMULATE-BALANCE at time $t$. Recall that $F^{OPT}$ and $E^{OPT}$ are the weighted flow time and processing cost of OPT, respectively. We will prove the following lemma in the sequel, which will complete the proof of Theorem 34.

**Lemma 40.** $F^A \leq O\left(\frac{1}{\epsilon^2}\right) (F^{OPT} + E^{OPT})$.

**Proof of Lemma 40:** Let $W^A(t)$ denote the fractional weight of jobs in the queue of SIMULATE-BALANCE; $W^A(t) = \sum_{J_i \in Q^A} w_i x_i(t)$, where $x_i(t) \in [0, 1]$ denotes the remaining processing time of $J_i$ at the time $t$. Then, weighted fractional flow time of SIMULATE-BALANCE is defined as: $f^A = \sum_{J_j \in J} \frac{w_j}{2} + \sum_{t \geq 0} W^A(t)$.

We start by bounding the weighted fractional flow time $f^A$ of SIMULATE-BALANCE. For an interval $[t_1, t_2]$, let $P^A(t_1, t_2)$, $P^{OPT}(t_1, t_2)$ denote the total units of processing done by SIMULATE-BALANCE and OPT in the interval $[t_1, t_2]$.

**Definition 41.** An interval $[s, d)$ is a lag-interval if $P^A(s, d) \geq P^{OPT}(s, d)$, but for all $t \in [s, d)$, $P^A(s, t) < P^{OPT}(s, t)$. 

92
• An interval \([s,d]\) is a lead-interval if for all \(t \in [s,d]\), Simulate-Balance processes more units that time instant than OPT, but at time \(d\), it processes less units than OPT.

• The entire time horizon partitions into a sequence of alternating lag and lead intervals of the form \([0,d_1],[d_1,d_2],\ldots\), where \(0 < d_1 < d_2\cdot\cdot\cdot\). We call the interval \([d_i,d_{i+1})\) as the \(i^{th}\) phase.

The following lemma follows by a repeated application of Theorem 37.

**Lemma 42.** For any phase \([d_i,d_{i+1})\), \(W^A(d_i) \leq W^{OPT}(d_i)\). Furthermore, for any \(t\) in a lead phase \([d_i,d_{i+1})\), we have \(W^A(t) \leq W^{OPT}(t)\).

**Proof.** We prove this by induction on \(i\). If the phase is a lead phase, the induction is trivial since at each step, Simulate-Balance processes more units than OPT, hence for any \(t \in [d_i,d_{i+1})\), we have \(W^A(t) \leq W^{OPT}(t)\). For a lag phase \([d_i,d_{i+1})\), Simulate-Balance processes jobs in HDF, always lags OPT in terms of number of units processed within the phase, but catches up with OPT at time \(d_{i+1}\). We simply invoke the Theorem 37 on the jobs processed within the phase to argue that if \(W^A(d_i) \leq W^{OPT}(d_i)\), then \(W^A(d_{i+1}) \leq W^{OPT}(d_{i+1})\). The details are straightforward and omitted.

We will bound the fractional flow time of Simulate-Balance against that of OPT within each phase. For a lead-phase, the bound is simple since at every time instant \(t\) within the phase, we have \(W^A(t) \leq W^{OPT}(t)\). We therefore focus only on lag-phases. Since Simulate-Balance always lags OPT in terms of number of units processed within a lag phase, we have the following easy consequences for any time instant in a lag phase:

• If Simulate-Balance processes, it processes at least \(\epsilon\) more units of jobs than OPT.

• If Simulate-Balance does not process, then it has to be a high cost instant.

We will now use a potential function argument to show that the fractional flow time of Simulate-Balance within the lag-phase is at most \(O(1/\epsilon)\) times the total cost spent by OPT within the phase.
Fix a lag-phase $i$. We use $P^{OPT}(t)$ and $P^{A}(t)$ to abbreviate $P^{OPT}(d_i, t)$ and $P^{A}(d_i, t)$ respectively. For each time instant $t \in [d_i, d_{i+1})$ we define the following potential function.

$$\Phi(t) = \frac{2\beta}{\epsilon}(P^{OPT}(t) - P^{A}(t))$$

(6.1)

The amortized cost paid by Simulate-Balance is defined as:

$$\theta(t) = W^{A}(t) + \Phi(t) - \Phi(t - 1)$$

(6.2)

Define $[t, t')$ as an idle period if neither OPT nor Simulate-Balance schedule jobs in that interval. The following lemma follows from a careful analysis of the potential function in Equation (6.1).

**Lemma 43.** For any lag-phase $[d_i, d_{i+1})$ and any idle period $X = [t, t')$ so that $t, t' \in [d_i, d_{i+1})$, we have:

$$\sum_{t \in X} \theta(t) \leq O\left(\frac{1}{\epsilon}\right) \left(\sum_{t \in X} (W^{OPT}(t) + e(t) \cdot I^{OPT}(t))\right)$$

(6.3)

where $I^{OPT}(t)$ is the indicator variable denoting whether OPT schedules at $t$.

**Proof.** By the description of Balance and since at most one job arrives each time step, we have:

$$\sum_{t \in X} W^{A}(t) \leq 2\beta$$

At time $t'$, there are two cases:

**Simulate-Balance schedules:** In this case, it schedule $\epsilon$ more units than OPT, so the potential drops by at least $2\beta$. The sum of flow time over the idle period is at most $2\beta$ and hence, the amortized cost is at most 0.

**Simulate-Balance does not schedule:** In this case, this time instant is a high cost time instant. OPT pays at least $\beta$ in processing cost. The potential increases by at most $\frac{2\beta}{\epsilon}$, while Simulate-Balance pays at most $2\beta$ in flow time. Therefore, the amortized cost of Simulate-Balance is at most $O(\beta/\epsilon)$. 


In either case, the amortized cost paid by Simulate-Balance is at most $O(1/\epsilon)$ times $OPT$’s flow time plus processing cost.

Next, note that $\Phi(d_i) - \Phi(d_{i+1} - 1)$ is non-negative in the entire time interval $[d_i, d_{i+1})$ since Simulate-Balance lags $OPT$. From Lemma 42 we know that $W^A_{\epsilon}(d_i) \leq W^{OPT}(d_i)$. Hence we conclude that over the interval $[d_i, d_{i+1})$, total weighted fractional flow time of Simulate-Balance is upper bounded by:

$$\sum_{t = d_i}^{d_{i+1}-1} W^A(t) \leq O\left(\frac{1}{\epsilon}\right) \sum_{t = d_i}^{d_{i+1}-1} (W^{OPT}(t) + e(t) \cdot I^{OPT}(t))$$

Equation (6.3) is true for lead-phases directly from Lemma 42. Therefore, summing over all phases, we conclude that the weighted fractional flow time of Simulate-Balance is at most $O(1/\epsilon)$ times the total cost of $OPT$. We then convert the weighted fractional flow time into weighted flow time by augmenting Simulate-Balance with another $(1+\epsilon)$-speed using ideas similar to that in Lemma 35. Therefore, we conclude that on a machine with $(1+\epsilon)$-speed, $F^A_{\epsilon} \leq O\left(\frac{1}{\epsilon^2}\right)(F^{OPT} + E^{OPT})$. This concludes the proof of Lemma 40.

**Proof of Theorem 34:** We first bound the competitive ratio of Simulate-Balance for the objective minimizing $\sum_j (w_j F_j + E(j))$ when jobs are of unit length. From Lemma 39, it follows that total processing cost of Simulate-Balance is at most $(1+\epsilon)E^{OPT}$. Lemma 40 shows that on a machine with $(1+\epsilon)$-speed, $F^A_{\epsilon} \leq \frac{1}{\epsilon^2}(F^{OPT} + E^{OPT})$. Putting all the pieces together, we conclude that Balance is $(1+\epsilon)$-speed $O\left(\frac{1}{\epsilon^2}\right)$-competitive for unit length jobs with arbitrary weights. Using Lemma 35 we finally conclude that Balance is $(1+\epsilon)$-speed $O\left(\frac{1}{\epsilon^3}\right)$-competitive for arbitrary length jobs.

6.3 Offline Problem: Weighted Completion Time

In this section, we study the offline problem of minimizing the average weighted completion time plus the processing cost. As before, there is a set $J$ of $N$ jobs with processing times
\( p_1, p_2, \ldots, p_n \), with release times \( r_1, r_2, \ldots, r_n \), with weights \( w_1, w_2, \ldots, w_n \); we assume these values are integers. The cost of processing on a machine is given as a function \( e(t) \), which we assume is a piecewise constant function with polynomially many break-points. We specify the function as a pair \( \{I_i, e(I_i)\} \), where \( I_i = [s_i, d_i] \) is the \( i \)th interval and \( \forall t \in I_i, e(t) = e(I_i) \) remains constant; assume these values are also integers. Let \( I = \{I_i | i \geq 0\} \) denote the set of all intervals. Recall that the processing cost of job \( J_j \) denoted by \( E(j) \) is defined as \( \sum_t e(t) x_j(t) \), where \( x_j(t) \) is an indicator function denoting whether job \( J_j \) was scheduled at time instant \( t \), where \( t \) refers to the interval \( [t, t+1) \). Our objective is to minimize the function \( \sum_j (w_j C_j + E(j)) \). Here, we study following two problems:

1. \( |pmtn| \sum_j (w_j C_j + E(j)) \) and \( |r_j, pmtn| \sum_j (w_j C_j + E(j)) \).

Similar to online version, an algorithm for the problem needs to make two policy decisions. The time slot selection policy of an algorithm decides which time instants the algorithm schedules the jobs – the processor could decide to idle in some time instants where the processing cost is too high. The job selection policy of an algorithm decides which job to schedule at any given time instant. Hence, an algorithm for minimizing the objective \( \sum_j (w_j C_j + E(j)) \), will have periods of processing (active periods) and periods where it idles (idle periods).

We note upfront that without the assumption on preemption, approximating the problem \( 1|| \sum_j (C_j + E(j)) \) to any function of \( n \) is strongly NP-hard. We also note that \( 1|r_j| \sum_j (w_j C_j + E(j)) \) can be solved optimally in polynomial time, when jobs are of unit length (with release dates) by computing bipartite matching on an appropriately defined graph. We omit the details here.

6.3.1 No Release Dates: \( 1|pmtn| \sum_j (w_j C_j + E(j)) \)

We first consider the case when all the jobs are released at \( t = 0 \). In this case, the goal becomes to compute an ordering (or permutation) in which to process the jobs, and the time instants in which to schedule the jobs (and those in which to idle). Let \( \pi \) be an ordering of the jobs. Let \( W^\pi(t|p') \) denote total weight of the jobs remaining at the time \( t \)
given that a total of \( p' \) units of processing has happened in the interval of time \([0, t]\). Since the jobs are processed in the fixed order \( \pi \), we have following two simple observations.

**Observation 2.** On a single machine scheduling jobs according to a sequence \( \pi \), for a fixed value \( p' \), the quantity \( W^\pi(t|p') \) is uniquely defined and can be calculated in polynomial time.

**Observation 3.** If an algorithm \( A \) does \( p' \) units of processing in an interval \( I_i = [s_i, d_i] \), then it uses time instants in the period \([s_i, s_i + p']\). This claim holds true only in the absence of release dates, since all the jobs are available for processing at time \( s_i \), hence using the period \([s_i, s_i + p']\) can only decrease the completion time of the jobs.

Observations 2 and 3 lets us design a simple dynamic program for solving the time slot selection policy of \( A \). Let \( F^\pi(P, i) \) denote the minimum cost of completing \( P \in \{1\ldots \sum_j p_j\} \) units of jobs by the end of interval \( I_i \), when jobs are scheduled according to the sequence \( \pi \). Then, from the Observation 2, we have:

\[
F^\pi(P, i) = \min_{p'} \left\{ F^\pi(P - p', i - 1) + p'e(I_i) + \sum_{t=s_i}^{d_i} W^\pi(t) \right\}
\]  \hspace{1cm} (6.4)

where \( p' \in \{0, 1, \ldots \min (P, d_i - s_i)\} \). However, for a particular sequence \( \pi \) of the jobs, \( \sum_{t=s_i}^{d_i} W^\pi(t) \) can be calculated as follows.

\[
\sum_{t=s_i}^{d_i} W^\pi(t) = \sum_{x=0}^{p'-1} W^\pi(s_i + x|P - p' + x + 1) + \sum_{t=s_i+p'+1}^{d_i} W^\pi(t|P)
\]  \hspace{1cm} (6.5)

Once the table of \( F^\pi(P, i) \) values is computed using equations (6.4) and (6.5), \( \min_{I_i \in I} F^\pi(\sum_j p_j, i) \) gives the optimal solution. Running time of the above dynamic program is \( O(|I|(|\sum_j p_j|^2)) \). Hence we have the Lemma:

**Lemma 44.** For the problem \( 1|\text{pmtn}\| \sum_j(w_j C_j + E(j)) \), time slot selection policy of an algorithm \( A \) which schedules the jobs according to ordering \( \pi \), can be computed in time \( O(|I|(|\sum_j p_j|^2)) \)
The dynamic programming formulation described above lets us focus on designing job selection policies for the problems. We now describe job selection policies for $1|\text{pmtn}| \sum_j (C_j + E(j))$ and $1|\text{pmtn}| \sum_j (w_j C_j + E(j))$.

**Lemma 45.** The job selection policy of any algorithm for the problem $1|\text{pmtn}| \sum_j (C_j + E(j))$ is Shortest Job First(SJF).

**Proof.** Fix the intervals of time where the algorithm processes the jobs. This fixes the processing cost $\sum_j E(j)$ incurred by the algorithm; hence it remains to minimize the total completion time. It is well-known that in the absence of release dates, SJF minimizes the total completion time. \hfill \Box

Unfortunately, scheduling jobs in HDF does not guarantee good performance for the problem $1|\text{pmtn}| \sum_j (w_j C_j + E(j))$. To design the job selection policy for this problem, we use ideas from universal sequencing problems studied in (Epstein et al., 2010). Consider an unreliable machine, which can experience unexpected failures. During a failure period, the machine cannot process jobs and such failure periods are not known in advance. The universal sequencing considers problem of designing a single ordering (or schedule) that performs well, regardless of failure periods of the machine. In absence of the release dates, for minimizing the objective $\sum_{J_i \in J} w_i C_i$ (Epstein et al., 2010) design a universal sequence which gives $4$-approximation to the optimal schedule, which runs in pseudo-polynomial time.

**Lemma 46.** Let $A$ denote the algorithm that uses universal sequencing of Epstein et al. as the job selection policy and the dynamic program in Lemma 44 as the time slot selection policy. For $1|\text{pmtn}| \sum_j (w_j C_j + E(j))$, $A$ is $4$-approximation.

**Proof.** Let $\theta^t_A$ denote the total cost incurred by algorithm $A$. Consider the periods of time where the optimal algorithm processes jobs. Let these intervals be denoted by $T^{OPT}$. Let $B$ denote an algorithm which uses universal sequence of Epstein et al. as job selection policy and uses the time slot selection policy of $OPT$. That is, it processes the jobs in universal
sequence of Epstein et al., in the intervals $T^{OPT}$. Let $\theta^U_B$ denote the cost incurred by $B$. From (Epstein et al., 2010) we know that, $\theta^U_B \leq 4OPT$. However, $A$ and $B$ both process jobs in the same sequence, hence it follows from Lemma 44 that $\theta^U_A \leq \theta^U_B$. Therefore, $\theta^U_A \leq 4OPT$.

To summarize the above results,

**Theorem 47.** There is a $O(|I|(|I|p_j)^2)$ time algorithm to optimally solve the problem 1|\(p|\sum_j p_j\)|\(\sum_j W_j C_j + E(j)\) and a 4-approximation to the objective 1|\(p|\sum_j W_j C_j + E(j)\)

### 6.4 Minimizing Weighted Completion Time with Release Dates

We now consider the above problems when there are release dates on the jobs. In this setting, it is easy to verify that there can be no fixed ordering of jobs that can provide good performance guarantees. In this section, we present two results: (1) When $e(t)$ takes on only $K = 2$ distinct values, we present a quasi-polynomial time constant approximation; we think it can be extended to get constant approximation in time $O(n^k \log n)$ for arbitrary values of $K$. (2) For the general problem, we show an LP rounding algorithm that yields a constant approximation with $(1 + \epsilon)$-speed.

#### 6.4.1 $K = 2$ Case: Quasi-polynomial Time Algorithm

Let's consider the problem of minimizing the weighted completion time when a machine is not available for processing during certain time periods. The non-availability periods are known in advance and the objective is to schedule jobs to minimize the weighted completion time and ensure that no job is scheduled when the machine is not available for processing. Although not explicitly mentioned in their paper, the algorithm DOUBLE-R in (Epstein et al., 2010) can be adapted to get a constant approximation to this problem.

**Theorem 48.** DOUBLE-R is a 4-approximation to 1|\(r_j, p|\sum_j w_j C_j\) when a machine has non-availability periods that are known in advance.
We now consider our problem when $K = 2$. Similar to the online version of the problem, we assume that $e(t)$ takes either a value $\beta$ or 1 at all time instants. We call the time instants when $e(t) = \beta$ as high cost time instants and the rest as low cost instants. When length of jobs and $\beta$ are polynomial in $n$, we present a quasi-polynomial time $O(1)$-approximation to the problem $1|r_j, pmtn| \sum j(w_jC_j + E(j))$.

Let $T(w)$ denote the earliest time instant $t$ when total weight of the unfinished jobs in $OPT$ is at most $w$. (We drop the superscript $OPT$ in this section, as all the variables refer to optimal schedule.) Let $H(t_1, t_2)$ denote total units of processing done by $OPT$ in high cost time instants within the interval $[t_1, t_2]$. Consider an interval $[T(w), T(w/\beta)]$ where total remaining weight of jobs in $OPT$’s schedule decreases by a factor of $c$. Let $t' \in [T(w), T(w/\beta)]$ denote the time instant such that total number of high cost times instants in the interval $[t', T(w/\beta)]$ is equal to $H(T(w), T(w/\beta))$. Within the interval $[T(w), T(w/\beta)]$, if $OPT$ uses all high cost time instants in the time period $[t', T(w/\beta)]$, then weight of jobs remaining at time $T(w/\beta)$ will be at most the weight of remaining jobs at $T(w/\beta)$ had $OPT$ used the high cost time instants optimally. This claim follows from the observation that the set of jobs available for processing does not change with this modification. Further, the cost incurred by $OPT$ towards total weighted completion time increases at most by a factor of $c$. Therefore, we make the following observation regarding the optimal schedule.

**Observation 4.** With a factor $c$ loss in approximation ratio, within an interval $[T(w), T(w/\beta)]$, we assume $OPT$ does all $H(T(w), T(w/\beta))$ units of processing in the high cost time instants in decreasing order of time.

With this characterization of $OPT$, we now present a constant approximation to $1|r_j, pmtn| \sum j(w_jC_j + E(j))$ when $K = 2$. To simplify the notation, for $i = \{0, 1 \ldots \lceil \log \beta \rceil \}$, let $T_i$ denote the earliest time instant when total weight of unfinished jobs in $OPT$’s schedule is at most $\frac{\beta}{2^i}$. Similarly, let $H(i)$ denote $H(T_i + 1, T_{i+1})$ in the subsequent discussion. Further, with $O(1)$ loss in the approximation ratio, we assume that $OPT$ processes both in low and high cost time instants till total weight of the unfinished jobs is at least $\beta$. Given the
values of $T_i$ and $H(i)$ for $i \in \{0, 1 \ldots \lceil \log \beta \rceil \}$ we compute the schedule which minimizes the objective $\sum_j (w_j C_j + E(j))$ as follows. First, we reduce the problem of minimizing the objective $\sum_j (w_j C_j + E(j))$ into a problem of minimizing the weighted completion time with non-availability periods. We now describe the reduction.

For an interval $(T_i, T_{i+1}]$, let $t' \in (T_i, T_{i+1}]$ be the earliest time instant such that total number of high cost time instants after $t'$ is equal to $H(i)$. Define all high cost time instants in the interval $(T_i, t']$ as unavailable periods. With this reduction, we run DOUBLE-R to compute a schedule which minimizes the weighted completion time; from Theorem 48, we know that the solution will be a 4-approximation to the optimal solution.

Finally, to determine the values of $T_i$ and $H(i)$ for $i \in \{0, 1 \ldots \lceil \log \beta \rceil \}$, we do an exhaustive search on all feasible values. It is easy to see that there can be at most $O((\beta \sum_j p_j)^{\log \beta})$ estimates for the values of $T_i$ and $H(i)$. We run DOUBLE-R on at most $O((\beta \sum_j p_j)^{\log \beta})$ feasible sets and take the schedule with minimum cost, which from Theorem 48, gives us a $O(1)$-approximation to the optimal solution to the problem $1|r_j, pmtn| \sum_j (w_j C_j + E(j))$.

Thus, we conclude:

**Theorem 49.** When values of $p_j$’s and $\beta$ are polynomially bounded, there is a $O(1)$-approximation to the problem $1|r_j, pmtn| \sum_j (w_j C_j + E(j))$ in $O(n^{\log^2 n})$ time.

### 6.4.2 LP Relaxation and Analysis via Speed Augmentation

Consider the following time indexed Linear Programming relaxation for the problem adapted from (Hall et al., 1996). In this LP, for every time unit $t$ and every job $j$ we have a variable $x_{jt}$ indicating whether the job is scheduled at that instant. For a job $j$ with release time $r_j$, $x_{jt} = 0, \forall t < r_j$. Let $y_{jt}$ be the indicator variable denoting whether job $j$ finishes before time $t$. It is easy to check that any schedule is a feasible solution to the LP-MIN and LP optimum is a lower bound on the cost incurred by any schedule.

Minimize $\sum_{j,t} w_j (1 - y_{jt}) + \sum_{j,t} e(t) x_{jt}$ \hspace{1cm} (LP-MIN)
\[
\sum_{t} x_{jt} = p_i \quad \forall j \quad (1)
\]

\[
\sum_{t=0}^{t'} x_{jt} \geq p_j y_{jt'} \quad \forall j, t' \quad (2)
\]

\[
\sum_{j} x_{jt} \leq 1 \quad \forall t \quad (3)
\]

\[
x_{jt} \in [0, 1] \quad \forall j, t \quad (4)
\]

\[
y_{jt} \in [0, 1] \quad \forall j, t \quad (5)
\]

The following example shows that the LP has unbounded integrality gap. Hence, we resort to resource augmented analysis. Let \( \epsilon < 1 \) be any constant. Let \( e(t) = 1 \) in the interval \([0, 1 - \epsilon]\). \( e(t) \) takes \( \infty \) in the interval \((1 - \epsilon, L]\) where \( L \) is some constant. \( e(t) = 1 \) every where else. The optimal LP solution to the instance has a cost \( O(\epsilon L) \) where as the optimal integral solution to this instance has to pay a cost of \( L \). By making \( \epsilon \) arbitrarily close to zero, we can make the integrality gap unbounded.

Hence, we analyze the schedule produced by the LP-MIN in the resource augmentation setting. We show that with \( 0 < \epsilon < 1 \)-speed augmentation, the schedule produced by LP is \( O(\frac{1}{\epsilon}) \) approximation to the optimal solution. We modify the LP solution on a machine with augmented speed of \( 1 + \epsilon \) to get schedule \( A \) as follows. Schedule the job \( J_j \) in the first \( \frac{p_j}{1+\epsilon} \) time instants in which LP schedules the job. Following theorem bounds the the cost of \( A \).

**Theorem 50.** \( A \) is a \( O(\frac{1}{\epsilon}) \)-approximation to optimal schedule with \( 1 \)-speed.

**Proof.** Let \( x^*_{jt}, y^*_{jt} \) denote the solution to LP-MIN. For a job \( J_j \), let \( t' \) denote the earliest time instant when \( y^*_{jt'} \geq \frac{1}{1+\epsilon} \). Therefore, on a machine with \( (1 + \epsilon) \)-speed \( J_j \) completes by the time \( t' \). Let \( \delta_{j}^{LP}, \delta_{j}^{A} \) denote total cost incurred by the optimal LP solution towards the Weighted completion time of a job \( J_j \) and the cost incurred by scheduling jobs on machine with \( 1 + \epsilon \)-speed. Then,

\[
\delta_{j}^{LP} = \sum_{t}^{t'} w_{j}(1 \- y_{jt}) \geq \sum_{t=0}^{t'} w_{j}(1 - y_{jt}) = \left( \frac{\epsilon}{1+\epsilon} \right) w_{j} \cdot t' = \delta_{j}^{A}
\]
Similarly, the total processing cost of a job \( J_j \) increases at most by a factor of \( 1 + \epsilon \). Hence, \( A \) is \((\frac{1}{1+\epsilon})\)-approximation to LP solution. Hence, \( A \) is \((1 + \epsilon)\)-speed \( O((\frac{1}{1+\epsilon})\)-approximation to optimal solution to the problem \( 1|r_j, pmtn| \sum_j (w_j C_j + E(j)) \). We omit the details of making the LP polynomial in size.

6.5 Summary and Open Problems

We initiated the study of cost aware scheduling, where there is a cost involved to schedule jobs at each time instant, and gave some initial results. Several questions remain open: In the offline setting, we do not know any approximation algorithms (without speed augmentation) that run in polynomial time. Also, it would be interesting to consider other objective functions in our model.

6.6 Notes

This chapter is based on joint work with Kamesh Munagala, and appeared in Approximation and Online Algorithms - 10th International Workshop, WAOA 2012, Ljubljana, Slovenia, September 13-14, 2012, (Kulkarni and Munagala, 2012).
PART III

New Models For Multidimensional Scheduling
7

Polytope Scheduling

7.1 Introduction

In the previous chapters we considered classical models of scheduling. All the models we considered till now have one major drawback: they assume that each job wants only one resource, typically CPU. However, resource allocation and scheduling challenges that arise in modern data center settings are multidimensional in nature. Consider a typical data center setting, where there is a cluster of machines with a distributed file system implementation (such as HDFS Shvachko et al. (2010)) layered on top of the cluster. Users submit executables (or jobs) to this cluster. In a typical MapReduce implementation such as Hadoop \(^1\), each job is a collection of parallel map and reduce tasks requiring certain CPU, disk space, and memory to execute. The job therefore comes with a request for resources in each dimension; these can either be explicitly specified, or can be estimated by the task scheduler from a high-level description of the job.

In a general scheduling scenario that has gained a lot of attention recently (see Ghodsi et al., 2011) and followup work (Cole et al., 2013; Zaharia et al., 2008; Ahmad et al., 2012; Popa et al., 2012; Lee et al., 2011)), there are \(M\) different types of resources. In the context

\(^1\) http://hadoop.apache.org
of a data center, these could be CPU, disk, memory, network bandwidth, and so on. The resources are assumed to be infinitely divisible due to the abundance of resources, and there is $R_d$ amount of resource $d$.

Each job $j$ is associated with resource demand vector $f_j = (f_{j1}, f_{j2}, ..., f_{jM})$ so that it requires $f_{jd}$ amount of the $d^{th}$ resource. At each time instant, the resources must be feasibly allocated among the jobs. If job $j$ is allocated resource vector $(a_{j1}, a_{j2}, ..., a_{jM})$ where $a_{jd} \leq f_{jd}$, it is processed at a rate that is determined by its bottleneck resource, so that its rate is $x_j = \min_d(a_{jd}/f_{jd})$. Put differently, the rate vector $x$ needs to satisfy the set of packing constraints:

$$\mathcal{P} = \left\{ \sum_j x_j f_{jd} \leq R_d \ \forall d \in [M]; \quad x \leq 1; \quad x \geq 0 \right\}$$

The above resource allocation problem, that we term Multi-dimensional Scheduling is not specific to data centers – the same formulation has been widely studied in network optimization, where resources correspond to bandwidth on edges and jobs correspond to flows. The bandwidth on any edge must be feasibly allocated to the flows, and the rate of a flow is determined by its bottleneck allocation. For instance, see (Kelly et al., 1998) and copious followup work in the networking community.

The focus of such resource allocation has typically been instantaneous throughput (Ghodsi et al., 2011), fairness (Ghodsi et al., 2011; Popa et al., 2012; Lee et al., 2011), and truthfulness (Ghodsi et al., 2011; Cole et al., 2013) – at each time instant, the total rate must be as large as possible, the vector $x$ of rates must be “fair” to the jobs, and the jobs should not have incentive to misreport their requirements. The scheduling (or temporal) aspect of the problem has largely been ignored. Only recently, in the context of data center scheduling, has response time been considered as an important metric – this corresponds to the total completion time or total flow time of the jobs in scheduling parlance. Note that the schedulers in a data center context typically have access to instantaneous resource requirements (the vectors $f_j$), but are not typically able to estimate how large the jobs are
in advance—in scheduling parlance, they are *non-clairvoyant*. They further are only aware of jobs when they arrive, so that they are *online* schedulers.

Though there has been extensive empirical work measuring response times of various natural resource allocation policies for data center scheduling (Ghodsi et al., 2011; Zaharia et al., 2008; Ahmad et al., 2012; Popa et al., 2012; Lee et al., 2011), there has been no theoretical analysis of any form. This is the starting point of this chapter—we formalize non-clairvoyant, online scheduling under packing constraints on rates as a general framework that we term *General Polytope Scheduling Problem* (PSP), and present competitive algorithms for problems in this framework.

### 7.2 General Polytope Scheduling Framework

In this chapter, we consider a generalization of the multi-dimensional scheduling problem discussed above. In this framework that we term *General Polytope Scheduling Problem* (PSP), the packing constraints on rates can be arbitrary. We show below (Section 7.3) that this framework not only captures multi-dimensional scheduling, but also captures classical scheduling problems such as unrelated machine scheduling (with preemption and migration), fractional broadcast scheduling, as well as scheduling jobs with varying parallelizability—only some special cases have been studied before.

In PSP, a scheduling instance consists of *n* jobs, and each job *j* has weight *w* _j_, size *p*_ j, and arrives at time *r*_ j. At any time instant *t*, the scheduler must assign rates \{*x*_ j\} to the current jobs in the system. Let *x*_ j^A*(t)* denote the rate at which job *j* is processed at time *t* by a scheduler/algorithm *A*. Job *j*’s completion time *C*_ j^A* under the schedule of *A* is defined to be the first time *t’* such that \(\int_{t=r_j}^{t’} x_j^A(t) \, dt \geq p_j\). Similarly, we define job *j’* flow time as *F*_ j^A* = C*_ j^A* − *r*_ j, which is the length of time job *j* waits to be completed since its arrival. When the algorithm *A* and time *t* are clear from the context, we may drop them from the notation.

We assume the vector of rates *x* is constrained by a packing polytope *P*, where the
matrices $H$, $Q$ have non-negative entries.

$$\mathcal{P} = \{ x \mid x \leq Qz; \quad Hz \leq 1; \quad x \geq 0; \quad z \geq 0 \} \quad (7.1)$$

### 7.3 Applications of the PSP Framework

In this section we present several concrete problems that fall in the PSP framework. In each case, we present a mapping to the constraints in $\mathcal{P}$. We have already seen the special case of multi-dimensional scheduling. We note that our framework can handle combinations of these problems as well.

#### 7.3.1 All-or-nothing Multidimensional Scheduling.

In multidimensional scheduling, we have assumed that a job needs all resources to execute, and given a fraction of all these resources, it executes at a fraction of the rate. However, in practice, a job often needs to receive its entire requirement in order to be processed (Zaharia et al., 2008) – this can be necessitated by the presence of indivisible virtual machines that need to be allocated completely to jobs. Therefore, a job $j$ is processed at a rate of 1 when it receives the requirement $f_j$, otherwise not processed at all. This all-or-nothing setting was studied recently in (Fox and Korupolu, 2013) when there is only one dimension. To see how this problem is still captured by PSP, define variables that encode feasible schedules. Let $S$ denote the collection of subsets of jobs that can be scheduled simultaneously. Let $z_S$ denote the indicator variable which becomes 1 if and only if $S$ is exactly the set of jobs currently processed. We observe this setting is captured by the following polytope.

$$\mathcal{P} = \{ x_j \leq \sum_{S: j \in S} z_S \forall j; \quad \sum_{S \in S} z_S \leq 1; \quad x \geq 0; \quad z \geq 0 \} \quad (7.2)$$

The solution to $\mathcal{P}$ is a set of preemptive schedules that process jobs in $S$ for $z_S$ fraction of time.

---

7.3.2 Scheduling Jobs with Different Parallelizability over Multiple Resources.

In most cluster computing applications, a job is split into several tasks that are run in parallel. However, jobs may have different parallelizability depending on how efficiently it can be decomposed into tasks (Wolf et al., 2010). To capture varying degree of parallelizability, an elegant theoretical model a.k.a. arbitrary speed-up curves was introduced by (Edmonds et al., 2003). In this model, there is only one type of resources, namely homogeneous machines, and a job \( j \) is processed at a rate of \( \Gamma_j(m_j) \) when assigned \( m_j \) machines. The parallelizability function \( \Gamma_j \) can be different for individual jobs \( j \), and is assumed to be non-decreasing, and sub-linear (\( \Gamma_j(m_j)/m_j \) is non-increasing). Due to the simplicity and generality, this model has received considerable amount of attention (Robert and Schabanel, 2008; Chan et al., 2009; Edmonds et al., 2011; Edmonds and Pruhs, 2012; Fox et al., 2013). However, no previous work addresses parallelizability in multiple dimensions and heterogeneous machines. Here we extend \( \Gamma_j \) to be a multivariate function that takes the resource vector \( \mathbf{z}_j := (z_{j1}, z_{j2}, ..., z_{jM}) \) of dimension \( M \) job \( j \) is assigned, and outputs the maximum speed job \( j \) can get out of the assignment. The function \( \Gamma_j \) is restricted to be concave in any positive direction. Observe that \( x_j \leq \Gamma_j(\mathbf{z}_j) \) can be (approximately) expressed by a set of packing constraints over \( \mathbf{a}_j \) that upper bound \( x_j \). (The PSP framework can be generalized to a convex polytope, and our results carry over). Then the obvious extra constraints is \( \sum_j z_{j} \leq 1 \). This extension can also capture tradeoff between resources (complements or substitutes) that can be combinatorial in nature. For example, a job can boost its execution by using more CPU or memory in response to the available resources.

7.3.3 Non-clairvoyant Scheduling for Unrelated Machines.

In this problem there are \( M \) unrelated machines. Job \( j \) is processed at rate \( s_{ij} \in [0, \infty) \) on each machine \( i \). (Unrelated machines generalize related machines where machines have different speeds independent of jobs). The online algorithm is allowed to preempt and migrate jobs at any time with no penalty – without migration, any online algorithm has an arbitrarily large competitive ratio for the total completion time (Gupta et al., 2012b).
The important constraint is that at any instantaneous time, each machine can schedule only one job, and a job can be processed only on a single machine.

We can express this problem as a special case of PSP as follows. Let $z_{ij}$ denote the fraction of job $j$ that is scheduled on machine $i$. Then:

$$
\mathcal{P} = \left\{ x_j \leq \sum_i s_{ij} z_{ij} \ \forall j; \quad \sum_j z_{ij} \leq 1 \ \forall i; \quad \sum_i z_{ij} \leq 1 \ \forall j; \quad x \geq 0; \quad z \geq 0 \right\}
$$

Note that any feasible $z$ can be decomposed into a convex combination of injective mappings from jobs to machines preserving the rates of all jobs. Therefore, any solution to $\mathcal{P}$ can be feasibly scheduled with preemption and reassignment. As before, the rates $s_j$ are only revealed when job $j$ arrives. No non-trivial result was known for this problem before our work. The only work related to this problem considered the setting where machines are related and jobs are unweighted (Gupta et al., 2012b). The algorithm used in (Gupta et al., 2012b) is a variant of Round Robin; however, as pointed out there, it is not clear how to extend these techniques to take job weights and heterogeneity of machines into account, and this needs fundamentally new ideas.

### 7.3.4 Generalized Broadcast Scheduling.

There are $M$ pages of information (resources) that is stored at the server. The server broadcasts a unit of pages at each time step. When a page $i$ is broadcast, each job $j$ (of total size $p_j$) is processed at rate $s_{ij}$. The vector $s_j$ of rates is only revealed when job $j$ arrives. Therefore:

$$
\mathcal{P} = \left\{ x_j \leq \sum_{i \in [M]} s_{ij} z_i \ \forall j; \quad \sum_{i \in [M]} z_i \leq 1; \quad x \geq 0; \quad z \geq 0 \right\}
$$

This setting strictly generalizes classical fractional broadcast scheduling where it is assumed that for each job $j$, the rate $s_{ij} = 0$ for all pages except one page $i$, and for the page $i$, $s_{ij} = 1$. In general, $s_{ij}$ can be thought of as measuring how much service $i$ makes happy client $j$ – for motivations, see (Azar and Gamzu, 2011; Im et al., 2012a) where more general
submodular functions were considered for clairvoyant schedulers in a different setting. We note that fractional classical broadcast scheduling is essentially equivalent to the integral case since there is an online rounding procedure (Bansal et al., 2010) that makes the fractional solution integral while increasing each job’s flow time by at most a constant factor (omitting technicalities). The unique feature of broadcast scheduling is that there is no limit on the number of jobs that can be processed simultaneously as long as they ask for the same resource. It has therefore received considerable attention in theory (Gandhi et al., 2006; Bansal et al., 2008, 2010; Edmonds et al., 2011; Im and Moseley, 2012; Bansal et al., pear) and has abundant applications in practice such as multicast systems, LAN and wireless systems (Wong, 1988; Acharya et al., 1995; Aksoy and Franklin, 1999).

7.4 Notes

This chapter is based on joint work with Sungjin Im and Kamesh Munagala. A preliminary version containing some of the results presented in this chapter appeared in Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014, (Im et al., 2014a).
8.1 Introduction

As we saw in the last chapter, a PSP input instance consists of $n$ jobs, and each job $j$ has weight $w_j$, size $p_j$, and arrives at time $r_j$. At any time instant $t$, the scheduler must assign rates $\{x_j\}$ to the current jobs in the system. Let $x_j^A(t)$ denote the rate at which job $j$ is processed at time $t$ by a scheduler/algorithm $A$. Job $j$'s completion time $C_j^A$ under the schedule of $A$ is defined to be the first time $t'$ such that $\int_{t=r_j}^{t'} x_j^A(t) dt \geq p_j$. Similarly, we define job $j$'s flow time as $F_j^A = C_j^A - r_j$, which is the length of time job $j$ waits to be completed since its arrival. When the algorithm $A$ and time $t$ are clear from the context, we may drop them from the notation.

We assume the vector of rates $x$ is constrained by a packing polytope $\mathcal{P}$, where the matrices $H, Q$ have non-negative entries.

$$\mathcal{P} = \left\{ x \mid x \leq Qz; \quad Hz \leq 1; \quad x \geq 0; \quad z \geq 0 \right\} \quad (8.1)$$

The class of scheduling algorithms we consider are constrained by several properties, all of which are naturally motivated by modern scheduling applications.

- It is online and learns about job $j$ only when it arrives. Before this point, $x_j = 0$. 
• It is non-clairvoyant, i.e., does not know a job’s size $p_j$ until completing the job.

• It is allowed to re-compute $x(t)$ at any real time $t$ arbitrarily often. As we will see below, this allows for pre-emption as well as migration across machines at no cost. Though we technically allow infinitely many re-computations, our algorithms will perform this computation only when jobs either arrive or complete.

Without loss of generality, we will assume the matrices $H, Q$ are known in advance to the scheduler and are independent of time, so that $P$ itself is time-invariant. One way of enforcing this is to assume that jobs arrive online from a subset of a (possibly countably infinite) universe $U$ of possible jobs, and the matrices $H, Q$ are defined over this universe. This is purely done to simplify our description and notation – in our applications, the polytope $P$ will indeed be defined only over the subset of jobs currently in the system, and the algorithms we design will make no assumptions over future jobs.

Under these assumptions, we will investigate non-clairvoyant online algorithms that minimize the overall job latency, i.e., the total weighted completion time $\sum_j w_j C_j$ (resp. total weighted flow time $\sum_j w_j F_j$). We will compare our algorithm against the optimal offline scheduler that knows the scheduling instance $(w_j, p_j, r_j$ for all jobs $j$) in advance, using the standard notion of competitive ratio.

8.1.1 Our Results

Our main result is the following; it also yields the first such result for all the applications discussed in the previous chapter 7.3.

**Theorem 51.** For the weighted completion time objective, there exists a $O(1)$-competitive non-clairvoyant scheduling algorithm for PSP.

We show this result by a simple algorithm that has been widely studied in the context of fairness in resource allocation, dating back to (Nash, 1950). This is the Proportional Fairness (PF) algorithm (Nash, 1950; Kelly et al., 1998; Ghodsi et al., 2011). Let $A_t$ denote the set of jobs alive at time $t$. At time $t$, the rates are set using the solution to the following
convex program (See Section 8.4 for more details).

\[
x^*(t) = \arg\max \left\{ \sum_{j \in A_t} w_j \log x_j \mid x \in \mathcal{P} \right\}
\]

To develop intuition, in the case of multi-dimensional scheduling with resource vector \( f_j \) for job \( j \), the PF algorithm implements a competitive equilibrium on the jobs. Resource \( d \) has price \( \lambda_d \) per unit quantity. Job \( j \) has budget \( w_j \), and sets its rate \( x_j \) so that it spends its budget, meaning that \( x_j = \frac{w_j}{\sum_d \lambda_d f_{jd}} \). The convex program optimum guarantees that there exists a set of prices \( \{\lambda_d\} \) so that the market clears, meaning that all resources with non-zero price are completely allocated.

In the same setting, when there is \( K = 1 \) dimension, the PF solution reduces to Max-Min Fairness – the resource is allocated to all jobs at the same rate (so that the increase in \( f_j x_j \) is the same), with jobs dropping out if \( x_j = 1 \). Such a solution makes the smallest allocation to any job as large as possible, and is fair in that sense. Viewed this way, our result seems intuitive – a competitive non-clairvoyant algorithm needs to behave similarly to round-robin (since it needs to hedge against unknown job sizes), and the max-min fair algorithm implements this idea in a continuous sense. Therefore, fairness seems to be a requirement for competitiveness. However this intuition can be misleading – in a multi-dimensional setting, not all generalizations of max-min fairness are competitive – in particular, the popular Dominant Resource Fair (DRF) allocation and its variants (Ghodsi et al., 2011) are \( \omega(1) \)-competitive. Therefore, though fairness is a requirement, not all fair algorithms are competitive.

Multidimensional scheduling is not the only application where the “right” notion of fairness is not clear. As discussed before, it is not obvious how to generalize the most intuitively fair algorithm Round Robin (or Max-Min Fairness) to unrelated machine scheduling – in (Gupta et al., 2012b), a couple of natural extensions of Round Robin are considered, and are shown to be \( \omega(1) \)-competitive for total weighted completion time. In hindsight, fairness was also a key for development of online algorithms in broadcast scheduling (Bansal
et al., 2010). Hence, we find the very existence of a unified, competitive, and fair algorithm for PSP quite surprising!

8.2 Our Techniques

Our analysis is based on dual fitting, which was first introduced by (Anand et al., 2012) for online scheduling. This work considered (clairvoyant) unrelated machine scheduling for the total weighted flow time. Their approach formulates the natural LP relaxation for weighted flow time, and sets feasible dual variables of this program so that the dual objective is within a constant of the primal objective. Their algorithm couples natural single-machine scheduling policies with a greedy rule that assigns each arriving job to a machine that increases the objective the least assuming that no more jobs arrive. The algorithm is immediate-dispatch and non-migratory – it immediately assigns an arriving job to a machine, and the job never migrates to other machines. However, such nice properties require that the algorithm should be clairvoyant. In fact, there is a simple example that shows that any non-clairvoyant and immediate dispatch algorithm has an arbitrarily large competitive ratio if migration is not allowed (Gupta et al., 2012b). Ironically, migration, which seems to give more flexibility to the algorithm, makes the analysis significantly more challenging. Hence it is no surprise that essentially all online algorithms for heterogeneous machine scheduling have been non-migratory – the only exception being (Gupta et al., 2012b), which gives a scalable algorithm for related machines for the flow time objective. For the same reasons, there has been very little progress in non-clairvoyant heterogeneous machine scheduling which is in sharp contrast to the recent significant progress in the clairvoyant counterpart (Chadha et al., 2009; Anand et al., 2012).

Since PSP captures non-clairvoyant scheduling on unrelated machines, the algorithms need to be migratory. Since migration disallows reduction to single machine scheduling, this precludes the types of dual variable settings considered in (Anand et al., 2012). To develop intuition, in dual fitting, we are required to distribute the total weight of unsatisfied (resp. alive) jobs to the dual variables corresponding to constraints in \( \mathcal{P} \). We therefore connect
the dual values found by the KKT condition to the dual variables of the completion (resp.
flow) time LP for PSP. This is a challenging task since the duals set by KKT are obtained
by instantaneous (resource allocation) view of PF while the duals in the LP should be
globally set considering each job’s completion time. For the completion time objective we
manage to obtain $O(1)$-competitiveness by reconciling these two views using the fact that
the contribution of the unsatisfied jobs to the objective only decreases over time.

8.3 History

We only summarize related work that have not been discussed before. We note that PSP
is NP-hard even when all jobs arriving are known a priori – this follows from the well-
known NP-hardness of the problem of minimizing the total weighted completion time on
a single machine. In the offline setting, it is easy to obtain a $O(1)$-approximation for
PSP in the metric $\sum_j w_j C_j$. It can be achieved by LP rounding, for example, see (Im
et al., 2011a); similar ideas can be found in other literature (Schulz and Skutella, 1997;
Queyranne and Sviridenko, 2002). Tight upper bounds have been developed for individual
scheduling problems in completion time metric; see (Williamson and Shmoys, 2011) for a
nice overview. In the online setting, (Chadha et al., 2009; Anand et al., 2012) give a scalable
(clairvoyant) algorithm for the weighted flow time objective on unrelated machines. Linear
(or convex) programs and dual fitting approaches have been popular for online scheduling;
for an overview of online scheduling see (Pruhs et al., 2004). Though (Azar et al., 2013)
study a general online packing and covering framework, it does not capture temporal
aspects of scheduling and is very different from our framework.

8.4 The Proportional Fairness (PF) Algorithm and Dual Prices

We first set up useful notation that will be used throughout this chapter. We will refer to
our algorithm Proportional Fairness (PF) simply as $\mathcal{A}$. We let $\mathcal{A}_t := \{ j \mid r_j \leq t < C_j^A \}$
denote the set of outstanding/alive jobs at time $t$ in the algorithm’s schedule. Similarly,
let $U_t := \{ j \mid t < C_j^A \}$ denote the set of unsatisfied jobs. Note that $\mathcal{A}_t \subseteq U_t$, and $U_t$ can
only decrease as time $t$ elapses. We let $U_0$ denote the entire set of jobs that actually arrive. We denote the inner product of two vectors $u$ and $v$ by $u \cdot v$. For a matrix $B$, $B_i$ denotes the $i^{th}$ row (vector) of matrix $B$. Likewise, $B_i$ denotes the $i^{th}$ column vector of matrix $B$. The indicator variable $1()$ becomes $1$ iff the condition in the parentheses is satisfied, otherwise $0$.

As mentioned before, $C_A^j$ denotes job $j$’s completion time in $A$’s schedule. Let $F_A^j := C_A^j - r_j$ denote job $j$’s flow time; recall that $r_j$ denotes job $j$’s release time. For notational simplicity, we assume that times are slotted, and each time slot is sufficiently small compared to job sizes. By scaling, we can assume that each time slot has size $1$, and we assume that jobs arrive and complete only at integer times. These simplifying assumptions are w.l.o.g. and will make notation simpler.

To present our algorithm and analysis more transparently, we take a simpler yet equivalent view of the PSP by projecting the polytope $\mathcal{P}$ into $x$:

$$\mathcal{P} = \{Bx \leq 1; \quad x \geq 0\}, \quad (8.2)$$

where $B$ has no negative entries. The equivalence of these two expressions can be easily seen by observing that the definition in (9.1) is equivalent to that of general packing polytopes. We assume that $B$ has $D$ rows.

Recall that $\mathcal{A}_t := \{j \mid r_j \leq t < C_A^j\}$ denotes the set of outstanding/alive jobs at time $t$ in our algorithm’s schedule. At each time $t$ (more precisely, either when a new job arrives or a job is completed), the algorithm Proportional Fairness (PF) solves the following convex program.

$$\begin{align*}
\max & \sum_{j \in \mathcal{A}_t} w_j \log x_j \\
\text{s.t.} & Bx \leq 1 \\
& x_j = 0 \quad \forall j \notin \mathcal{A}_t
\end{align*}$$

Then $(PF)$ processes each job $j$ at a rate of $x_{jt}^*$ where $x_{jt}^*$ is the optimal solution of the convex program at the current time $t$. Here the time $t$ is added to subscript since the
scheduling decision changes over time as the set of outstanding jobs, \( A_t \) does. For compact notation, we use a vector changing over time by adding \( t \) to subscript – for example, \( \mathbf{x}_t^* \) denotes the vector \( \{x_{jt}^*\}_j \). Observe that the constraint \( \mathbf{x} \geq 0 \) is redundant since \( x_{jt}^* > 0 \) for all \( j \in A_t \).

The dual of \( \text{CP}_{PF} \) has variables \( y_d, d \in [D] \) corresponding to the primal constraints \( B_d \cdot \mathbf{x} \leq 1 \). Let \( \mathbf{y}_t := (y_{1t}, y_{2t}, ..., y_{Dt}) \). By the KKT conditions (Boyd and Vandenberghe, 2004), any optimal solution \( \mathbf{x}^* \) for \( \text{CP}_{PF} \) must satisfy the following conditions for some \( \mathbf{y}^* \):

\[
\begin{align*}
    y_{dt}^* \cdot (B_d \cdot \mathbf{x}_t^* - 1) &= 0 & \forall t, d \in [D] \quad (8.3) \\
    \frac{w_j}{x_{jt}^*} &= B_{jt} \cdot \mathbf{y}_t^* & \forall t, j \in A_t \quad (8.4) \\
    \mathbf{y}_t^* &\geq 0 & \forall t \quad (8.5)
\end{align*}
\]

We emphasize that the new definition of (8.2) of \( \mathcal{P} \) is only for ease of analysis; in reality, we will solve \( \text{CP}_{PF} \) over the original polytope given in (9.1) – this is entirely equivalent to the above discussion.

8.5 Analysis of PF

The analysis will be based on linear programming and dual fitting. Consider the following LP formulation, which is now standard for the weighted completion time objective (Hall et al., 1997).

\[
\begin{align*}
    \min & \quad \sum_{t,j} w_j \cdot \frac{t}{p_j} \cdot x_{jt} \\
    \text{s.t.} & \quad \sum_{t \geq r_j} \frac{x_{jt}}{p_j} \geq 1 & \forall j \in U_0 \\
    & \quad B \cdot \mathbf{x}_t \leq 1 & \forall t \geq 0 \\
    & \quad x_{jt} \geq 0 & \forall j, t \geq 0
\end{align*}
\]

The variable \( x_{jt} \) denotes the rate at which job \( j \) is processed at time \( t \). The first constraint ensures that each job must be completed. The second is the polytope constraint.
It is easy to see that the objective lower bounds the actual total weighted flow time of any feasible schedule.

For a technical reason which will be clear soon, we will compare our algorithm to the optimal schedule with speed $1/s$, where $s$ will be set to 32 later – this is only for the sake of analysis, and the final result, as stated in Theorem 51, will not need speed augmentation.

The optimal solution with speed $1/s$ must satisfy the following LP.

\[
\begin{align*}
\min \sum_{t,j} w_j \cdot \frac{t}{p_j} \cdot x_{jt} & \quad \text{(PRIMAL$_s$)} \\
\text{s.t.} \quad \sum_{t \geq r_j} \frac{x_{jt}}{p_j} & \geq 1 \quad \forall j \in U_0 \\
B \cdot (sx_t) & \leq 1 \quad \forall t \geq 0 \\
x_{jt} & \geq 0 \quad \forall j, t \geq 0
\end{align*}
\]

Note that the only change made in PRIMAL$_s$ is that $x$ is replaced with $sx$ in the second constraint. We take the dual of this LP; here $\beta_t := (\beta_{1t}, \beta_{2t}, \ldots, \beta_{Dt})$.

\[
\begin{align*}
\max \sum_j \alpha_j - \sum_{d,t} \beta_{dt} & \quad \text{(DUAL$_s$)} \\
\text{s.t.} \quad \frac{\alpha_j}{p_j} - sB \cdot \beta_t & \leq w_j \cdot \frac{t}{p_j} \quad \forall j, t \geq r_j \quad (8.6) \\
\alpha_j & \geq 0 \quad \forall j \quad (8.7) \\
\beta_{dt} & \geq 0 \quad \forall d, t \quad (8.8)
\end{align*}
\]

We will set the dual variables $\alpha_j$ and $\beta_{dt}$ using the optimal solution of CP$_{PF}$, $x^*_j$, and the corresponding dual variables $y^*_d$. The following proposition shows the outcome we will derive by dual fitting.

**Proposition 52.** Suppose there exist \{\alpha_j\}_j and \{\beta_{dt}\}_{d,t} that satisfy all constraints in DUAL$_s$ such that the objective of DUAL$_s$ is at least $c$ times the total weighted completion time of algorithm $A$. Then $A$ is $(s/c)$-competitive for minimizing the total weighted completion time.
Proof. Observe that the optimal objective of PRIMAL_s is at most s times that of PRIMAL. This is because that any feasible solution \( x_t \) for PRIMAL is also feasible for PRIMAL_s when the \( x_t \) is stretched out horizontally by a factor of s – the new schedule \( x'_t \) is defined as \( x'_t(st) = (1/s)x_t \) for all \( t \geq 0 \). The claim easily follows from the fact that PRIMAL is a valid LP relaxation of the problem, weak duality, and the condition stated in the proposition.

We will first show that the dual objective is a constant times the total weighted completion time of our algorithm, and then show that all dual constraints are satisfied. Recall that \( U_t := \{ j \mid j < C_j^A \} \) denote the set of unsatisfied jobs at time \( t \) – it is important to note that \( U_t \) also includes jobs that have not arrived by time \( t \), hence could be different from the set \( \mathcal{A}_t := \{ j \mid r_j \leq j < C_j^A \} \) of alive jobs at time \( t \). Let \( W_t := \sum_{j \in U_t} w_j \) denote the total weight of unsatisfied jobs at time \( t \).

We now show how to set dual variables using the optimal solution \( x^*_t \) of CP_PF, and its dual variables \( y^*_t \). We will define \( \alpha_j, t, \) and set \( \alpha_j := \sum_t \alpha_j, t, \) for all \( j \).

Let \( q_{jt} \) denotes the size of job \( j \) processed at time \( t \). Define \( \zeta_t \) to be the ‘weighted’ median of \( \frac{q_{jt}}{p_j} \) amongst all jobs \( j \) in \( U_t \) – that is, the median is taken assuming that each job \( j \) in \( U_t \) has \( w_j \) copies.

\[
\alpha_j, t := \begin{cases} w_j & \forall j, t \text{ s.t. } j \in U_t, \frac{q_{jt}}{p_j} \leq \zeta_t \\ 0 & \text{otherwise} \end{cases}
\]

We continue to define \( \beta_{dt} \) as \( \beta_{dt} := \sum_{t' \geq t} \frac{1}{s} \zeta_{t'}y^*_{dt'} \). We now show that this definition of \( \alpha_j, t \) and \( \beta_{dt} \) makes DUAL_s’s objective to be at least \( O(1) \) times the objective of our algorithm.

**Lemma 53.** \( \sum_j \alpha_j \geq (1/2) \sum_j w_j C_j^A \).

**Proof.** At each time \( t \), jobs in \( U_t \) contribute to \( \sum_j \alpha_j, t \) by at least half of the total weight of jobs in \( U_t \). □

**Lemma 54.** For any time \( t \), \( \sum_d y^*_dt = \sum_{j \in \mathcal{A}_t} w_j \leq W_t \).
Proof.

\[
\sum_d y_d^* = \sum_d y_d^* (B_d \cdot x_t^*) = \sum_d y_d^* \sum_{j \in A_t} B_{dj} x_{jt}^* = \sum_{j \in A_t} x_{jt}^* (B_j \cdot y_t^*) = \sum_{j \in A_t} x_{jt}^* \frac{w_j}{x_{jt}^*} \leq W_t
\]

The first and last equalities are due to the KKT conditions (8.3) and (8.4), respectively. □

Lemma 55. At all times \( t \), \( \sum_d \beta dt \leq \frac{8}{s} W_t \).

Proof. Consider any fixed time \( t \). We partition the time interval \([t, \infty)\) into subintervals \( \{M_k\}_{k \geq 1} \) such that the total weight of unsatisfied jobs at all times during in \( M_k \) lies in the range \( \left( \frac{1}{2} W_t, \left( \frac{1}{2} \right)^{k-1} W_t \right] \). Now consider any fixed \( k \geq 1 \). We upper bound the contribution of \( M_k \) to \( \sum_d \beta dt \), that is \( \frac{1}{s} \sum_{t' \in M_k} \sum_d \zeta_{dt'}y_{dt'} \). Towards this end, we first upper bound \( \sum_{t' \in M_k} \zeta_{dt'} \leq 4 \). The key idea is to focus on the total weighted throughput processed during \( M_k \). Job \( j \)'s fractional weighted throughput at time \( t' \) is defined as \( w_j \frac{q_{jt'}}{p_j} \), which is job \( j \)'s weight times the fraction of job \( j \) that is processed at time \( t' \); recall that \( q_{jt'} \) denotes the size of job \( j \) processed at time \( t' \).

\[
\sum_{t' \in M_k} \zeta_{dt'} \leq \sum_{t' \in M_k} \sum_{j \in A_{t'}} \frac{w_j}{W_{t'}} \cdot 1 \left( \frac{q_{jt'}}{p_j} \geq \zeta_{dt'} \right) \cdot \frac{q_{jt'}}{p_j} \leq 2 \frac{1}{(1/2)^k W_t} \sum_{t' \in M_k} \sum_{j \in U_{t'}} \frac{w_j}{p_j} \frac{q_{jt'}}{p_j}
\]

\[
\leq 2 \frac{1}{(1/2)^k W_t} \left( 1/2 \right)^{k-1} W_t = 4
\]

The first inequality follows from the definition of \( \zeta_{dt'} \): for jobs \( j \) with total weight at least half the total weight of jobs in \( U_{t'} \), \( \frac{q_{jt'}}{p_j} \geq \zeta_{dt'} \). The second inequality is due to the fact that \( W_{t'} \geq (1/2)^k W_t \) for all times \( t' \in M_k \). The last inequality follows since the total weighted throughput that can be processed during \( M_k \) is upper bounded by the weight of
unsatisfied jobs at the beginning of \( M'_k \), which is at most \( (\frac{1}{2})^{k-1} W_t \). Therefore,

\[
\sum_d \beta_{dt} = \frac{1}{s} \sum_{t' \geq t} \sum_d \zeta_{t'} y^*_{dt'} = \frac{1}{s} \sum_{k \geq 1} \sum_{t' \in M_k} \zeta_{t'} \sum_d y^*_{dt'}
\]

\[
\leq \frac{1}{s} \sum_{k \geq 1} \sum_{t' \in M_k} \zeta_{t'} W_{t'} \quad \text{[By Lemma 54]}
\]

\[
= \frac{1}{s} \sum_{k \geq 1} 4(\frac{1}{2})^{k-1} W_t \quad \text{[By definition of } M_k \text{ and the fact } \sum_{t' \in M_k} \zeta_{t'} \leq 4]
\]

\[
\leq \frac{8}{s} W_t
\]

\[\square\]

**Corollary 56.** \( \sum_{d,t} \beta_{dt} \leq \frac{8}{s} \sum_j w_j C_j' \).

From Lemma 53 and Corollary 56, we derive that the objective of \( \text{DUAL}_s \) is at least half of \( \text{PF}' \) total weighted completion time for \( s = 32 \). By Lemma 52, it follows that the algorithm \( \text{PF} \) is 64-competitive for the objective of minimizing the total weighted completion time.

It now remains to show all the dual constraints are satisfied. Observe that the dual constraint (8.7) is trivially satisfied. Also the constraint (8.8) is satisfied due to KKT condition (8.5).

We now focus on the more interesting dual constraint (8.6) to complete the analysis of Theorem 51.

**Lemma 57.** *The dual constraint (8.6) is satisfied.*
Proof.

\[
\frac{\alpha_j}{p_j} - w_j \frac{t}{p_j} \leq \sum_{t' \geq t} \frac{\alpha_{jt'}}{p_j} \quad \text{[Since } \alpha_{jt'} \leq w_j \text{ for all } t']
\]

\[
= \sum_{t' \geq t} \frac{w_j}{p_j} \cdot 1 \left( \frac{q_{jt'}}{p_j} \leq \zeta_{t'} \right) = \sum_{t' \geq t} \frac{w_j}{q_{jt'}} \cdot \frac{q_{jt'}}{p_j} \cdot 1 \left( \frac{q_{jt'}}{p_j} \leq \zeta_{t'} \right)
\]

\[
= \sum_{t' \geq t} \frac{w_j}{x_{jt'}^+} \cdot \frac{q_{jt'}}{p_j} \cdot 1 \left( \frac{q_{jt'}}{p_j} \leq \zeta_{t'} \right) \quad \text{[Since } q_{jt'} = x_{jt'}^+ \text{]}
\]

\[
\leq \sum_{t' \geq t} B_{jt} \cdot (\zeta_{t'} y_{t'}^*) \quad \text{[By the KKT condition (8.4)]}
\]

\[
= sB_{jt} \cdot \beta_t \quad \text{[By definition of } \beta_t]\]

\[
\square
\]

8.6 Notes

This chapter is based on joint work with Sungjin Im and Kamesh Munagala. A preliminary version containing some of the results presented in this chapter appeared in Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014, (Im et al., 2014a).
9

Minimizing Flow-Time for PSP

9.1 Introduction

We continue our study of PSP problem, and consider a more challenging problem of minimizing the total weighted flow time for PSP. For more details about PSP problem see Chapters 14.1, 8. For the sake of completeness, we briefly describe the problem again. A PSP scheduling instance consists of $n$ jobs, and each job $j$ has weight $w_j$, size $p_j$, and arrives at time $r_j$. At any time instant $t$, the scheduler must assign rates $\{x_j\}$ to the current jobs in the system. Let $x_j^A(t)$ denote the rate at which job $j$ is processed at time $t$ by a scheduler/algorithm $A$. Job $j$’s completion time $C_j^A$ under the schedule of $A$ is defined to be the first time $t'$ such that $\int_{t-r_j}^{t'} x_j^A(t)dt \geq p_j$. We define job $j$’s flow time as $F_j^A = C_j^A - r_j$, which is the length of time job $j$ waits to be completed since its arrival. When the algorithm $A$ and time $t$ are clear from the context, we may drop them from the notation.

We assume the vector of rates $\mathbf{x}$ is constrained by a packing polytope $\mathcal{P}$, where the matrices $H, Q$ have non-negative entries.

$$\mathcal{P} = \left\{ \mathbf{x} \mid \mathbf{x} \leq Q\mathbf{z}; \quad Hz \leq 1; \quad \mathbf{x} \geq 0; \quad \mathbf{z} \geq 0 \right\} \quad (9.1)$$
9.1.1 Our Results

Our goal in this chapter is to design algorithms that are online and non-clairvoyant for minimizing the total weighted flow-time of jobs. We prove the following theorem in this chapter.

**Theorem 58.** For PSP, the PF algorithm is \(O(\log n)\)-speed, \(O(\log n)\)-competitive for minimizing the total weighted flow time. Furthermore, there exists an instance of PSP for which no deterministic non-clairvoyant algorithm is \(O(n^{1-\epsilon})\)-competitive for any constant \(0 < \epsilon < 1\) with \(o(\sqrt{\log n})\)-speed.

Similar to our result for minimizing the total weighted completion time of jobs, we use PF algorithm, and denote it by \(A\). We give a brief description of PF algorithm here; please see Chapter 8.4 for more details about the algorithm. Let \(A_t\) denote the set of jobs alive at time \(t\). At time \(t\), the rates are set using the solution to the following convex program (See Section 8.4 for more details).

\[
x^*(t) = \arg\max \left\{ \sum_{j \in A_t} w_j \log x_j \mid x \in P \right\}
\]

9.2 High-level Overview of The Analysis

We first give a high-level overview of the analysis. Similar to the analysis we did for the completion time objective in the previous chapter, we will use dual fitting technique. The mathematical programs should be modified accordingly. The only changes made in PRIMAL and PRIMAL\(_s\) are in the objective: \(\sum_{t,j \in U(t)} w_j \cdot \frac{t}{p_j} \cdot x_{jt}\) should be changed to \(\sum_{t \in r_j,t} w_j \cdot \frac{t-r_j}{p_j} \cdot x_{jt}\). Recall that PRIMAL\(_s\) gives a valid lower bound to the optimal adversarial scheduler with \(1/s\)-speed, which is equivalent to the algorithm being given \(s\)-speed and the optimal scheduler being given 1-speed. Then the dual of PRIMAL\(_s\) is as follows.

\[
\max \sum_j \alpha_j - \sum_{d,t} \beta_{dt} \tag{DUAL\(_s\)}
\]
Let’s recall the high-level idea of the analysis for the completion time objective. In dual fitting, we set dual variables so that (i) the dual objective is at least $O(1)$ times $A'$ total weighted completion time of PF, and (ii) all dual constraints are satisfied. Regarding (i), we had two things to keep in mind: to make $\sum_j \alpha_j$ comparable to $A$’s total weighted completion time, and make $\sum_d t \beta_{dt}$ smaller than $\sum_j \alpha_j$. On the other hand, in order to satisfy dual constraints, we have to “cover” the quantity $\frac{\alpha_j}{p_j} - \frac{r_j}{p_j} \cdot \frac{t - r_j}{p_j}$ by $\beta_{dt}$. In this sense, (i) and (ii) compete each other.

To give an overview of the analysis of the flow result, for simplicity, let’s assume that all jobs are unweighted. We will construct a laminar family $L$ of intervals, and fit jobs into intervals with similar sizes. By ignoring jobs of small flow time, say less than $\frac{1}{2} \cdot \text{max weighted flow time of all jobs}$, we can assume that there are at most $O(\log n)$ levels of intervals in the family. Now we find the level such that the total weighted flow time of jobs in the level is maximized. We crucially use the fact that all jobs in the level have similar flow times. Namely, we will be able to define $\beta_{dt}$ for each laminar interval in the level we chose, via a linear combination of $y_t$ over the laminar interval.

The actual analysis is considerably more subtle particularly due to jobs with varying weights. In the weighted case, we will have to create multiple laminar families. Further, there could be many different laminar families, even more than $\text{polylog}(n)$. It is quite challenging to define $\beta_{dt}$ using a linear combination of the duals of $\text{CP}_{PF}$ while trying to minimize the side-effect between different laminar families. Also “ignoring” jobs is not as easy as it looks since such ignored jobs still contribute the duals of $\text{CP}_{PF}$.
9.3 Analysis of PF

In this section we give a formal analysis of the upper bound claimed in Theorem 58. To make our analysis more transparent, we do not optimize constants. To set dual variables, we perform the following sequence of preprocessing steps. To keep track of changes made in each step, we define a set \( A'_t \subseteq A_t \) of “active” jobs at time \( t \), which will help us to set dual variables later. Also we will maintain the set of “globaly active” jobs \( A' \). The set \( A' \) is global in the sense that if \( j \notin A' \) then \( j \notin A'_t \) for all \( t \). Intuitively, jobs in \( A' \) will account for a large fraction \( (\Omega(1/\log n)) \) of the total weighted flow time. Initially, we set \( A'_t := A_t \) for all \( t \) (recall that \( A_t \) denote jobs alive at time \( t \)), and \( A' \) to be the entire set of jobs. When we remove a job \( j \) from \( A' \), the job will be automatically removed from \( A'_t \) for all \( t \). Since \( A \) will change over the preprocessing steps, we will let \( A^i \) denote the current set \( A' \) just after completing \( i \)th step; \( A^i_t \) is defined similarly. Also we will refer to the quantity \( \sum_{t,j \in A_t^i} w_j F_j \) as job \( j \)’s residual flow time, which will change over preprocessing steps. A job’s weighted residual flow time is similarly defined. Between steps, we will formally state some important changes made. For a subset \( S \) of jobs, let \( W(S) \) denote the total weight of jobs in \( S \).

**Step 1. Discard jobs with small weighted flow time.** For all jobs \( j \) such that \( w_j F_j^A \leq \frac{1}{2n} \max_j w_j F_j^A \), remove them from \( A' \).

**Proposition 59.** \( \sum_t W(A_t^1) = \sum_{j \in A^1} w_j F_j \geq \frac{1}{2} \sum_j w_j F_j \).

**Step 2. Group jobs with similar weights: odd/even classes.** Let \( w_{\max} \) denote the maximum job weight. A job \( j \in A' \) is in class \( C_h, h \geq 1 \) if the job has weight in the range of \( (w_{\max}/n^8, w_{\max}/n^{8(h-1)}) \). Classes \( C_1, C_3, \ldots \) are said to be odd, and the other classes even. Note that \( \{C_h\}_{h \geq 1} \) is a partition of all jobs in \( A' \).

**Step 3. Keep only odd [even] classes.** Between odd and even classes, keep the classes that give a larger total weighted flow time. More precisely, if \( \sum_t \sum_{h:\text{odd}} W(C_h \cap A'_t) \geq \sum_t \sum_{h:\text{even}} W(C_h \cap \)
\( \mathcal{A}' \), we keep only odd classes, i.e. remove all jobs in even classes from \( \mathcal{A}' \). Since the other case can be handled analogously, we will assume throughout the analysis that we kept odd classes.

**Proposition 60.** \( \sum_t W(\mathcal{A}'_t^3) \geq \frac{1}{2} \sum_t W(\mathcal{A}'_t^2) = \frac{1}{2} \sum_t W(\mathcal{A}_t^1) \geq \frac{1}{4} \sum_j w_j F_j. \)

**Step 4. Black out times with extra large weights for odd classes.** We say that time \( t \) has extra large weights for odd classes if \( \sum_{h: \text{odd}} W(\mathcal{A}'_t \cap \mathcal{C}_h) \leq \frac{1}{8} W(\mathcal{A}_t) \). For all such times \( t \), we remove all jobs from \( \mathcal{A}'_t \). Let’s call such time steps “global black out” times, which we denote by \( \mathcal{T}_B \).

We now repeatedly remove jobs with a very small portion left – we say that a job \( j \) is almost-removed if the job \( j \in \mathcal{A}' \) has global black out times for at least \( 1 - \frac{1}{n^2} \) fraction of times during its window, \( [r_j, C_A^j] \). For each time \( t \in \mathcal{A}'_t \), if \( w_j \geq \frac{1}{n^2} W(\mathcal{A}'_t) \), we remove all jobs from \( \mathcal{A}'_t \), and add the time \( t \) to \( \mathcal{T}_B \). Also we remove the almost-removed job \( j \) from \( \mathcal{A}'_t \). We repeat this step until we have no almost-removed job in \( \mathcal{A}' \).

**Proposition 61.** For any time \( t \notin \mathcal{T}_B \), we have \( \sum_{h: \text{odd}} W(\mathcal{A}'_t^4 \cap \mathcal{C}_h) \geq \frac{1}{16} W(\mathcal{A}_t) \).

**Proof.** Before the second operation of repeatedly removing almost removed jobs, we have that for all \( t \notin \mathcal{T}_B \), \( \sum_{h: \text{odd}} W(\mathcal{A}'_t^4 \cap \mathcal{C}_h) \geq \frac{1}{8} W(\mathcal{A}_t) \). Hence if a time step \( t \) has “survived” to the end of Step 4, i.e. \( t \notin \mathcal{T}_B \), it means that the quantity \( W(\mathcal{A}'_t^4) \) decreased by a factor of at most \( 1 - 1/n^2 \) when a job is removed by the second operation. Assuming that \( n \) is greater than a sufficiently large constant, the proposition follows.

**Proposition 62.** \( \sum_t W(\mathcal{A}'_t^4) \geq \frac{1}{4} \sum_t W(\mathcal{A}_t^1) \geq \frac{1}{16} \sum_j w_j F_j. \)

**Proof.** Define \( V_j^1, V_j^2 \) to be the decrease of job \( j \)'s residual weighted flow time due to operation 1 [2]. Define \( R \) to be all jobs \( j \) in \( \mathcal{A}^3 \) such that \( \frac{1}{100} V_j^1 \leq V_j^2 \) and \( V_j^1 + V_j^2 \geq \frac{1}{4} F_j \). We claim that the decrease in the total residual weighted flow time due to jobs in \( \mathcal{A}^3 \setminus R \) is at most \( \frac{1}{2} \sum_t W(\mathcal{A}_t^3) \). Firstly, the decrease due to jobs \( j \) such that \( V_j^1 + V_j^2 \leq \frac{1}{4} F_j \) is at most \( \frac{1}{4} \sum_t W(\mathcal{A}_t^3) \). Secondly, the decrease due to jobs \( j \) such that \( \frac{1}{100} V_j^1 \geq V_j^2 \) is at most
the decrease due to the first operation in Step 4 times $101/100$, which is upper bounded by $\frac{101}{100} \cdot \frac{1}{16} \sum_j w_j F_j$. This follows from the observation that the total weighted flow time that jobs in $A^3_t$ accumulate at black out times is at most $\frac{1}{16} \sum_j w_j F_j \leq \frac{1}{4} \sum_t W(A^3_t)$ by Proposition 60. Hence the claim follows.

We now focus on upper bounding the decrease in the total residual weighted flow time due to jobs $A^3 \cap R$. Observe the important property of jobs $j$ in $A^3 \setminus R$: job $j$’s residual flow time decreases by $\Omega\left(\frac{1}{n} F_j\right)$ when a certain job $j'$ is removed in the second operation in Step 4 (recall that removing $j'$ could lead to more times becoming blacked out, hence decreasing other jobs residual flow times). Motivated by this, we create a collection of rooted trees $F$, which describes which job is most responsible for removing each job in $R$. We let $j'$ become $j$’s parent; if there are more than one such $j'$ we break ties arbitrarily. Note that roots in $F$ are those jobs $j$ whose residual flow time is $\frac{1}{n} F_j$ just before the second operation starts. Now let’s consider two jobs, $j'$ and $j$, a child of $j'$. We claim that $w_j F_j \leq O\left(\frac{1}{n^2}\right) w_{j'} F_{j'}$. Just before we remove job $j'$, job $j$ had residual flow time at least $\Omega\left(\frac{1}{n} F_{j'}\right)$, and $j'$ had residual flow time at most $\frac{1}{n^2} F_{j'}$. Further it must be the case that $w_{j'} \leq n^2 w_j$. This implies that job $j$’s weighted flow time is only a fraction of that of job $j'$, hence the claim follows.

We complete the proof by charging a child’s weighted flow time to its parent’s weighted flow time. Eventually we will charge the total weighted flow time of all jobs in $F$ to that of the root jobs. Then it is easy to observe that the total weighted flow time of non-root jobs in $F$ is at most $O(1/n^2)$ times the total weighted flow time of the root jobs. Since we already upper bounded $(1-1/n^4)$ times the total weighted flow time of root jobs by $\frac{1}{16} w_j F_j$, hence the decrease in the total residual weighted flow time due to jobs $(A^3 \cap R$ and $A^3 \setminus R)$ in Step 3 is at most $\frac{n^4}{n^4 - 1} \frac{1}{16} \sum_j w_j F_j + \frac{1}{2} \sum_t W(A^3_t) \leq \frac{3}{4} \sum_t W(A^3_t)$ by Proposition 60. □

**Proposition 63.** For all jobs $j \in A^4$, we have $\sum_{t:j \in A^4_t} 1 \geq \frac{1}{n^4} F_j$. 

129
Step 5. Find jobs with similar flow times within each odd class $C_h$ that have the largest total weighted flow time. Consider each odd class $h$. Observe that all jobs in $C_h$ have flow times which are all within factor $2n^9$ – this follows from the facts that all jobs in the same class have weights, all within factor $n^8$, and all jobs have weighted flow times, all within factor $2n$. Let $F_h^{\text{max}}$ denote the maximum flow time of all jobs in $C_h$. We say that a job $j \in C_h$ is in $u^{th}$-level if $F_j^A \in \left(F_h^{\text{max}}/2^u, F_h^{\text{max}}/2^{u-1}\right]$, and denote all $u^{th}$-level jobs in $C_h$ as $C_{hu}$. Let $u^*_h$ be the $u$ that maximizes the total “residual” weighted flow time of jobs in $C_{hu}$, i.e. $\sum_{j \in C_{hu}} \sum_{t : j \in A^t} w_j$. Observe that $1 \leq u^*_h \leq \lfloor \log 2n^9 \rfloor \leq 2^4 \log n$ for all $n$ greater than a sufficiently large constant. We remove all jobs $j \in C_h \setminus C_{h,u^*_h}$ from $A^t$.

Proposition 64. For all $h \geq 1$, $\sum_t W(A^t \cap C_h) \geq \frac{1}{2^4 \log n} \sum_t W(A^t \cap C_h)$.

Step 6. Fit jobs in $C_{h,u^*_h}$ into a disjoint intervals of similar sizes. Consider each $h$. Define a set of intervals $L_{hu} := \{[0, F_h^{\text{max}}/2^u), [F_h^{\text{max}}/2^u, 2F_h^{\text{max}}/2^u), [2F_h^{\text{max}}/2^u, 3F_h^{\text{max}}/2^u), \ldots]\}$. Observe that jobs in $C_{h,u^*_h}$ have flow times similar to the size of an interval in $L_{hu}$. Associate each job $j$ in $C_{h,u^*_h}$ with the interval $L$ in $L_{hu}$ that maximizes $\sum_{t : j \in A^t, t \in L} 1$, breaking ties arbitrarily. Here we trim out job $j$’s window so that it completely fits into the interval $L$, i.e. for all times $t \notin L$, remove job $j$ from $A^t$. We let $L_j$ denote the unique interval in $L_h$ into which we fit job $j$. Here we will refer to $L_j$ as job $j$’s laminar window.

We say that a job $j \in A^t$ is left if $L_j$’s left end point lies in job $j$’s window, $[r_j, C_j]$. All other jobs in $A^t$ are said to be right. Between left and right jobs, we keep the jobs that give a larger total remaining weighted flow time, remove other jobs from $A^t$.

Proposition 65. For all jobs $j \in A^6$, we have $\sum_{t : j \in A^t} 1 \geq \frac{1}{4n} F_j$.

Proof. There are at most 4 intervals in $C_{h,u^*_h}$ that intersect job $j$’s window. This, together with Proposition 63 implies the claim.

$\square$

Proposition 66. $\sum_t W(A^t_6 \cap C_h) \geq \frac{1}{2^t} \sum_t W(A^t \cap C_h) \geq \frac{1}{2^4 \log n} \sum_t W(A^t_4 \cap C_h)$, for all odd values of $h \geq 1$. 130
Step 7. Define exclusive active times for each odd class. We say that a class $C_h$ is lower than $C_{h'}$ if $h < h'$. Observe that jobs in a lower odd class have flow time significantly smaller than jobs in a higher odd class. Consider odd classes $C_h$ in increasing order of $h$. For each job $j$, and for all times $t$ where a smaller odd class job $j'$ in $A'$ is processed ($j' \in A_t'$), remove job $j$ from $A_t'$. This makes different odd classes have disjoint “active” times – for any two jobs $j, j'$ from different odd classes, and for all times $t$, it is the case that $j \not\in A_t' \cap C_h$ or $j' \not\in A_t' \cap C_{h'}$. For an odd class $h$, define $T_h$ to be the set of times $t$ where no job in odd class lower than $h$ is processed, and there is a job in $A_t' \cap C_h$. We say that the times in $T_h$ are active for class $h$. Let $h^*_t$ denote the class that is active at time $t$ – by definition there is at most one such class, and we let $h^*_t := 0$ if no such class exists.

**Proposition 67.** $\sum_t W(A_t' \cap C_h) \geq \frac{1}{2} \sum_t W(A_t' \cap C_{h^*_t}) \geq \frac{1}{2^{q} \log n} \sum_t W(A_t' \cap C_{h^*_t})$, for all odd values of $h \geq 1$.

**Proof.** Observe that every job in $A_1$ of a lower class than $j$ has a flow time at most $2n/n^8$ times that of job $j$ – two jobs in different odd groups have different weights which differ by a factor of at least $n^8$ while all jobs in $A_1$ have the same weight flow time within factor $2n$. Hence jobs in smaller classes can create at most $2n^2 F_j$ additional inactive time slots for job $j$, which are negligible compared to $\sum_{t,j \in A_t'} 1 \geq \frac{1}{16n^4} F_j$ (See Proposition 65). The proposition follows assuming that $n$ is greater than a sufficiently large constant. \qed

Step 8. Black out times with extra large weights for $C_{h^*_t}$ at each time $t$. Consider each time $t$. We say that time $t$ has extra large weights for class $h^*_t$ if $\sum_{j \in A_t' \cap C_{h^*_t}} w_j \geq 2^9 \log n \sum_{j \in A_t' \cap C_h} w_j$. For all such times $t$, we remove all jobs $j \in C_{h^*_t}$ from $A_t'$. Let’s call such time steps “black out” times for class $C_{h^*_t}$. Also remove those times from $T_{h^*_t}$.

**Proposition 68.** For all odd $h \geq 1$,

$$\sum_t W(A_t' \cap C_{h^*_t}) \geq \frac{1}{2} \sum_t W(A_t' \cap C_{h^*_t}) \geq \frac{1}{2^{q} \log n} \sum_t W(A_t' \cap C_{h^*_t}).$$

**Proof.** From Proposition 67, we have that $\sum_t W(A_t' \cap C_{h^*_t}) \geq \frac{1}{2^{q} \log n} \sum_t W(A_t' \cap C_{h^*_t})$. At every black out time, the total weight of jobs in $A_t' \cap C_{h^*_t}$ is at most $\frac{1}{2^{q} \log n}$ times the
total weight of jobs in $A_t^4 \cap C_{h_t^*}$. This implies that in Step 8, the decrease in the quantity $\sum_t W(A_t^4 \cap C_{h_t^*})$ is at most $\frac{1}{2^7 \log n}$ times $\sum_t W(A_t^4 \cap C_{h_t})$, hence the second inequality follows.

We now show the last inequality. We can prove that $\sum_t W(A_t^4 \cap C_{h_t^*}) \geq \frac{1}{2} \sum_t \sum_{h: \text{odd}} W(A_t^4 \cap C_h)$ (See the proof of Proposition 69). Then no jobs in even class contribute to the right-hand-side quantity (See Step 3), we have $\sum_t \sum_{h: \text{odd}} W(A_t^4 \cap C_h) = \sum_t \sum_h W(A_t^4 \cap C_h)$. This, together with Proposition 62, completes the proof.

**Proposition 69.** At all times $t \notin T_B$, $W(A_t^8 \cap C_{h_t^*}) \geq \frac{1}{2^7 \log n} W(A_t^4 \cap C_{h_t^*}) \geq \frac{1}{2^{14} \log n} W(A_t)$.

**Proof.** We focus on the second inequality since the first inequality is obvious. By definition of $h_t^*$, we know that no job in odd class smaller than $h_t^*$ is processed by $A$ at time $t$. Also we know that there is at least one job in $A_t^8 \cap C_{h_t^*}$ whose weight is $n^4$ times larger than any job in a higher odd class. Hence we have that $W(A_t^4 \cap C_{h_t^*}) \geq \frac{1}{2} \sum_{h: \text{odd}} W(A_t^4 \cap C_h)$. Then the second inequality immediately follows from Proposition 61.

This completes the description of all preprocessing time steps. At this point, let’s recap what we have obtained from these preprocessing steps. Proposition 69 says that at each time $t \notin T_B$, we can just focus jobs in one class, and furthermore all those jobs are at the same level and the level is the same at all times for the class. This is because at each time $t$, the total weight of those jobs at the same level in a class is at least $\Omega(1/\log n)$ times the total weight of all jobs alive at the time. Also since we have fit jobs at the same level into intervals of the same size, and kept say only left jobs, we will be able to pretend that those jobs arrive at the same time. Then the analysis basically reduces to that of the weighted completion time objective.

We are almost ready to set dual variables. Let $q_{jt}$ denote the size of job $j$ processed at time $t$. For each $h$, we define $\zeta_t(h)$ as follows. Let $L$ be the unique interval in $L_{h, u^*_h}$ such that $t \subseteq L$. Let $\zeta_t(h)$ denote the weighted median from the multiset $M(h, L) := \{ \frac{q_{jt}}{p_j} \mid j \in A_t^8 \cap C_h \}$ – here the median is taken assuming that the quantity $\frac{q_{jt}}{p_j}$ has $w_j$ copies in the multiset $M(h, L)$. As before, we set dual variables using the optimal solution $x_t$ of $\text{CP}_\text{PF}$,
and its dual variables $y_t$. Recall that each time step $t$ is active for at most one class $h$ which is denoted as $h^*_t$; if no such class exists, let $h^*_t = 0$ and $\zeta_t(h^*_t) := -1$. Define,

$$
\alpha_{jt} := \begin{cases} 
    w_j & \forall t, j \in C_{h^*_t} \cap A_t^8 \text{ s.t. } \frac{q_{jt}}{p_j} \leq \zeta_t(h^*_t) \\
    0 & \text{otherwise}
\end{cases}
$$

and let

$$
\alpha_j := \sum_t \alpha_{jt} \quad \forall j
$$

We continue to define $\beta_{dt}$. We will first define $\beta_{dt}(h)$ for each odd $h$, and will let $\beta_{dt} := \sum_{h: odd} \beta_{dt}(h)$ for all $d,t$. Consider any odd $h$ and $L \in L_{h,u^*_h}$. Then for all times $t \in L$, define

$$
\beta_{dt}(h) := \frac{1}{s} \sum_{t' \geq t, t' \in T_h \cap L \setminus T_B} \zeta_{t'}(h) y_{dt'}
$$

Also for all odd $h$ and $t$ such that there is no $L \in L_{h,u^*_h}$ with $t \in L$, we let $\beta_{dt}(h) := 0$.

This completes setting dual variables.

As before, we will first lower bound the objective of $\text{DUAL}_s$. We start with lower bounding the first part in the objective.

**Lemma 70.** \( \sum_j \alpha_j \geq \frac{1}{2} \sum_t W(A_t^8 \cap C_{h^*_t}) \).

**Proof.** For any time $t$, it is easy to see from the definition of $\alpha_{jt}$ that the quantity $\sum_j \alpha_{jt}$ is at least $\frac{1}{2} W(A_t^8 \cap C_{h^*_t})$.

\qed

In the following lemma we lower bound the second part $\sum_{d,t} \beta_{dt}$ in the $\text{DUAL}_s$ objective. The proof is very similar to that of Lemma 55.

**Lemma 71.** For all odd $h$ and time $t \in T_h \setminus T_B$, \( \sum_d \beta_{dt}(h) \leq \frac{220 \log n}{s} W(A_t^8 \cap C_{h^*_t}) \).
**Proof.** Consider any fixed odd \( h \) and \( L \in \mathcal{L}_{h,u_h^*} \). Recall that amongst jobs \( j \in \mathcal{C}_{h,u_h^*} \) such that \( L_j = L \), we kept left or right jobs in Step 6. We assume that we kept left jobs since the other case can be handled similarly. Consider any \( t \in \mathcal{T}_h \setminus \mathcal{T}_B \). Let \( t_R \) denote the right end point of \( L \). It is important to note that jobs contributing to \( \mathcal{A}_{h}^8 \cap \mathcal{C}_h \) are consistent: the set of such jobs can only decrease in in time \( t' \in \mathcal{T}_h \cap L \setminus \mathcal{T}_B \) – so does \( W(\mathcal{A}_{h}^8 \cap \mathcal{C}_h) \). We partition the time interval \([t, t_R]\) into subintervals \( \{M_k\}_{k \geq 1} \) such that \( J \), the the quantity \( \sum_{h \in \mathcal{C}_h} (\frac{1}{2})^{k-1} W(\mathcal{A}_{h}^8 \cap \mathcal{C}_h) \). Now consider any fixed \( k \geq 1 \). We upper bound the contribution of \( M_k \) to \( \sum_{h} \beta_d(h) \), that is \( \frac{1}{2} \sum_{h \in \mathcal{C}_h} \left( 2 \sum_{j \in \mathcal{A}_{h}^8 \cap \mathcal{C}_h} \frac{w_j}{W(\mathcal{A}_{h}^8 \cap \mathcal{C}_h)} \right) \cdot 1(\frac{q_{j'}u'}{p_j} \geq \zeta_{h}(h)) \cdot \frac{q_{j'}u'}{p_j} \)

\[
\leq 2 \left( \frac{1}{2} \right)^{k} W(\mathcal{A}_{h}^8 \cap \mathcal{C}_h) \sum_{h \in \mathcal{C}_h} \sum_{j \in \mathcal{A}_{h}^8 \cap \mathcal{C}_h} w_j \frac{q_{j'}u'}{p_j} 
\leq 2 \left( \frac{1}{2} \right)^{k} W(\mathcal{A}_{h}^8 \cap \mathcal{C}_h) \left( \frac{1}{2} \right)^{k-1} W(\mathcal{A}_{h}^8 \cap \mathcal{C}_h) = 4
\]

The first inequality follows from the definition of \( \zeta_{h}(h) \): for jobs \( j \) with total weight at least half the total weight of jobs in \( \mathcal{A}_{h}^8 \cap \mathcal{C}_h \), \( \frac{q_{j'}u'}{p_j} \geq \zeta_{h}(h) \). The second inequality is due to the fact that \( W(\mathcal{A}_{h}^8 \cap \mathcal{C}_h) \geq \left( \frac{1}{2} \right)^{k} W(\mathcal{A}_{h}^8 \cap \mathcal{C}_h) \) for all times \( t' \in M_k \). The last inequality follows since the total weighted throughput that can be processed during \( M_k \) is upper bounded by the quantity \( W(\mathcal{A}_{h}^8 \cap \mathcal{C}_h) \) at the earliest time \( t' \in M_k \cap \mathcal{T}_h \setminus \mathcal{T}_B \), which is at most
\((\frac{1}{2})^{k-1}W(A^8_t \cap C_h)\). We are now ready to complete the proof.

\[
\sum_d \beta_{dt}(h) = \frac{1}{s} \sum_{t' \geq t, t' \in T_h \backslash T_B} \zeta_{t'}(h) y^*_{dt'} = \frac{1}{s} \sum_{k \geq 1} \sum_{t' \in M_k \cap T_h \backslash T_B} \zeta_{t'}(h) \sum_d y^*_{dt'}
\]

\[
= \frac{1}{s} \sum_{k \geq 1} \sum_{t' \in M_k \cap T_h \backslash T_B} \zeta_{t'}(h) W(A_{t'}) \quad \text{[By Lemma 54]}
\]

\[
= \frac{2^{17} \log n}{s} \sum_{k \geq 1} \sum_{t' \in M_k \cap T_h \backslash T_B} \zeta_{t'}(h) W(A^8_{t'} \cap C_h) \quad \text{[By Proposition 69]}
\]

\[
= \frac{2^{17} \log n}{s} \sum_{k \geq 1} 4(1/2)^{k-1} W(A^8_k \cap C_h)
\]

\[
\leq \frac{2^{20} \log n}{s} \cdot W(A^8_k \cap C_h)
\]

The penultimate inequality follows from the fact that \(\sum_{t' \in M_k} \zeta_{t'}(h) \leq 4\) and from the definition of \(M_k\). From Lemma 70 and Lemma 71, we derive that the DUAL\_s objective is at least \(\Omega(\sum_t W(A^8_t \cap C_h))\) with \(s = 2^{22} \log n\). By Proposition, we conclude that the DUAL\_s objective is \(\Omega(1/\log n)\) times the total weighted flow time.

To complete the proof of the upper bound result in Theorem 58, it only remains to show that all dual constraints are satisfied. Observe that the dual constraints (9.3) and (9.4) are trivially satisfied, hence we focus on dual constraint (9.2). Recall that \(q_{jt}\) denotes the size of job \(j\) that is processed by the algorithm at time \(t\). Note that \(\sum_t q_{jt} = p_j\).

**Lemma 72.** The dual constraint (9.2) is satisfied.

**Proof.** We only need to consider \(j \in A^8\). Say \(j \in C_h\) (\(h\) is odd as before). Also we only need to consider time \(t\) before the interval \(L_j\) ends. Observe that \(j \in C_{h'_t} \cap A^8\) only if
\[ h = h_t^* \text{ and } t \notin T_B. \]

\[
\frac{\alpha_j}{p_j} - w_j \frac{t - a_j}{p_j} \leq \sum_{t' \geq t, t' \in T_h \setminus T_B} \frac{\alpha_{jt'}}{p_j} \quad \text{[Since for all } t', \alpha_{jt'} \leq w_j]\]

\[
\leq \sum_{t' \geq t, t' \in T_h \cap L_j \setminus T_B} \frac{\alpha_{jt'}}{p_j} \quad \text{[Since } \alpha_{jt'} \leq w_j \text{ at all times } t' \notin L_j]\]

\[
= \sum_{t' \geq t, t' \in T_h \cap L_j \setminus T_B} w_{jt'} \frac{1}{p_j} \cdot 1 \left( \frac{q_{jt'}}{p_j} \leq \zeta_{jt'}(h) \right) \]

\[
= \sum_{t' \geq t, t' \in T_h \cap L_j \setminus T_B} w_{jt'} \frac{q_{jt'}}{x_{jt'}} \cdot 1 \left( \frac{q_{jt'}}{p_j} \leq \zeta_{jt'}(h) \right) \quad \text{[Since } q_{jt'} = x_{jt'}^*]\]

\[
\leq \sum_{t' \geq t, t' \in T_h \cap L_j \setminus T_B} B_{jt} \cdot y_{jt'}^* \cdot \zeta_{jt'}(h) \quad \text{[By the KKT condition (8.4)]}\]

\[
= B_{jt} \cdot \beta_t(h) \quad \text{[By definition of } \beta_t(h)]\]

\[
\leq B_{jt} \cdot \beta_t \quad \text{[By definition of } \beta_t]\]

\[ \square \]

9.4 Lower Bound

In this section, we prove the lower bound claimed in Theorem 58. Towards this end, we will first prove a lower bound for makespan.

**Theorem 73.** Any deterministic non-clairvoyant algorithm is \( \Omega(\sqrt{\log n}) \)-competitive for minimizing the makespan (the maximum completion time). Further, this is the case even when all jobs arrive at time 0.

We prove that Theorem 73 implies the desired result.

**Proof of** [the lower bound in Theorem 58] To see this, let \( I_0 \) denote the lower bound instance consisting of \( N \) unweighted jobs that establishes the lower bound stated in Theorem 73. By scaling, we can without loss of generality assume that the optimal (offline)
makespan for this instance is 1. For any fixed $\epsilon > 0$, we create $N^{1/\epsilon}$ copies of instance $I_0$, $\{I_e\}_{e \in \{0,1,2,...,N^{1/\epsilon}\}}$ where all jobs in $I_e$ arrive at time $e$. There is a global constraint across all instances $I_e$ – two jobs from different instances cannot be scheduled simultaneously. Then any deterministic non-clairvoyant algorithm that is given speed less than half the lower bound stated in Theorem 73 cannot complete all jobs in each $I_e$ within 2 time steps. It is easy to see that there are at least $e/2$ jobs not completed during $[e, e + 1)$ for any $e \in \{0, 1, 2, ..., N^{1/\epsilon}\}$. Hence any deterministic online algorithm has total flow time $\Omega(N^{2/\epsilon})$. In contrast, the optimal solution can finish all jobs within 1 time step, thus having total flow time $O(N \cdot N^{1/\epsilon})$. This implies that the competitive ratio is $\Omega(n^{(1-\epsilon)/(1+\epsilon)})$ where $n$ is the number of jobs in the entire instance concatenating all $I_e$, completing the proof of lower bound stated in Theorem 58.

□

Henceforth, we will focus on proving Theorem 73. Our lower bound instance comes from single source routing in a tree network with “multiplicative speed propagation”. As the name suggests, this network is hypothetical: a packet is transferred from node $v_a$ to $v_b$ at a rate equal to the multiplication of speeds of all routers that the packet goes through. To give a high-level idea of the lower bound, we first discuss one-level tree, and then describe the full lower bound instance. Throughout this section, we refer to an arbitrary non-clairvoyant algorithm as $A$.

One-level instance: $I(1)$. The root $\rho$ has $\Delta_1 := 4$ routers where only one router has 2-speed and the other routers have 1-speed. There are $\Delta_1$ packets (or equivalently jobs) to be routed to the root $\rho$. Only one job has size $2^2 - 1 = 3$, and the other jobs have size $2^1 - 1 = 1$. Each job must be completely sent to the root, and it can be done only using routers. At any time, each router can process only one job. This setting can be equivalently viewed as the related machine setting, but we stick with this routing view since we will build our lower bound instance by multilayering this one-level building block. Obviously, the optimal solution will send the big job via the 2-speed router, thus having makespan $3/2$. Also intuitively, the best strategy for $A$ is to send all jobs at the same rate
by equally assigning the 2-speed router to all jobs. Then it is easy to see that the online algorithm can complete all 1-size jobs only at time $\Delta_1/(\Delta_1 + 1)$, and complete the 2-size job at time $\Delta_1/(\Delta_1 + 1) + 1 = 9/5$. Observe that giving more 1-speed routers does not give any advantage to the online algorithm since the main challenge comes from finding the big job and processing it using a faster router.

**Multi-level instance:** $I(h), h \in [D = \Theta(\sqrt{\log n})]$. We create a tree $T_h$ with root $\rho$ where all jobs are leaves and each job $j$ can communicate with its parent node $u(j)$ via one of $u(j)$’s router, and the parent $u(j)$ can communicate with its parent node $u^{(2)}(j)$ via one of $u^2(j)$’s router, and so on; node/job $v$’s parent is denoted as $u(v)$. The tree $T_h$ has depth $h$. Every non-leaf node $v$ has $\Delta_h = 4^h$ children, which are denoted as $C_v$. Also each non-leaf node $v$ has a set $R_v$ of routers, whose number is exactly the same as that of $v$’s children, i.e. $|R_v| = |C_v| = \Delta_h$. All routers in $R_v$ have 1-speed except only one which has 2-speed.

At any time, a feasible scheduling decision is a matching between routers $R_v$ and nodes $C_v$ for all non-leaf nodes $v$; when some jobs complete, this naturally extends to an injective mapping from $C_v$ to $R_v$. To formally describe this, let $g$ denote each feasible scheduling decision. Note that each feasible schedule $g$ connects each job to the root by a unique sequence of routers. Let $z_g$ denote the indicator variable. Let $\eta_j(g)$ denote the number of 2-speed routers on the unique path from $j$ to the root $\rho$ for $g$. When the schedule follows $g$ each job $j$ is processed at a rate of $2^{\eta_j(g)}$. We can formally describe this setting by PSP as follows:

$$
\mathcal{P} = \left\{ x_j \leq \sum_g 2^{\eta_j(g)}z_g \forall j; \quad \sum_g z_g \leq 1; \quad x \geq 0; \quad z \geq 0 \right\}
$$

We now describe job sizes, which are hidden to $\mathcal{A}$. Each non-leaf node $v$ has one special “big” child amongst its $\Delta_h = 4^h$ children $C_v$ – roughly speaking, a big child can have bigger jobs in its subtree. Note that $\mathcal{A}$ is not aware of which child is big. For each node $v$ of depth $h - 1$, define $\eta_v$ to be the number of “big” children on the path from $v$ to the root possibly including $v$ itself. Then $v$’s children/jobs, $C_v$ have the following sizes: for any
integer $0 \leq k < \eta_v$, the number of jobs of size $2^{k+1} - 1$ is exactly $4^{h-\eta_v}(4^{\eta_v-k} - 4^{\eta_v-k-1})$; for $k = \eta_v$, there are $4^{h-\eta_v}$ jobs of size $2^\eta_v+1 - 1$. Note that there is only one job of size $2^{h+1} - 1$ in $T_h$ and it is the biggest job in the instance.

The final instance will be $\mathcal{I}(D)$. Since $\mathcal{I}(D)$ has $4^D$ jobs, we have $D = \Theta(\sqrt{\log n})$. For a visualization of the instance, see Figure 9.1.

**Figure 9.1:** Here, routers are represented by rectangles and jobs by circles. Height of the tree is $\Theta(\sqrt{\log n})$. Each job has $4^D$ children and for each level less than $D - 1$ there is one big node, shown by dotted circle, which is hidden from the online algorithm. Each job has to be mapped to a router. The sizes of jobs at the last level depend on the number of big nodes on the path connecting a job to the root.

**Lemma 74.** There is an offline schedule that completes all jobs by time 2.

**Proof.** This is achieved by assigning each big node/job to the 2-speed router at all levels. This is possible since each non-leaf node has exactly one faster router and one big node/job. Consider any non-leaf node $v$ of depth $D - 1$. Since all jobs in $\mathcal{C}_v$ have size at most $2^{\eta_v+1} - 1$, and all those jobs are processed at a rate of at least $2^{\eta_v}$, the claim follows. \qed
We now discuss how $A$ performs for the instance $I(D)$. We first give a high-level overview of the adversary’s strategy which forces $A$ to have a large makespan. Then, we will formalize several notions to make the argument clear – the reader familiar with online adversary may skip this part.

**A high-level overview of the adversary’s strategy.** As mentioned before, the main difficulty for the non-clairvoyant algorithm $A$ comes from the fact that $A$ does not know which jobs/nodes are big, hence cannot process big nodes using faster routers. This mistake will accrue over layers and will yield a gap $\Omega(D)$. To simplify our argument, we allow the adversary to decrease job sizes. That is, at any point in time, the adversary observes the non-clairvoyant algorithm $A$’s schedule, and can remove any alive job. This is without loss of generality since the algorithm $A$ is non-clairvoyant, and can only be better off for smaller jobs. Obviously, this does not increase the optimal solution’s makespan.

Consider any node $v \neq \rho$. Let us say that the subtree $T_{v'}$ rooted at $v'$ is big [small] if the node $v'$ is big [small]. If the node $v$ have used the unique 2-speed router in $R_{u(v)}$ for $1/2^{D+1}$ time steps, the adversary removes the subtree $T_{v'}$ rooted at $v'$ (including all jobs in $T_{v'}$). We now show that at time $1/2$, the adversary still has an instance as effective as $I(D - 1)$ – by repeating this, the online algorithm will be forced to have a makespan of at least $D/2$. We claim two properties.

1. At time $1/2$, each alive non-leaf node has at least $4^{D-1}$ children.

2. Any job has been processed by strictly less than 1.

The first property easily follows since each non-leaf node has $4^D$ children, and at most $2^D$ children are removed by time $1/2$. To see why the second property holds, consider any job $j$. Observe that each of $j$’s ancestor (including $j$ itself) used the 2-speed router only for $1/2^{D+1}$ time steps. Here the maximum processing for job $j$ can be achieved when $j$’s all ancestors use the 2-speed router simultaneously for $1/2^{D+1}$ time steps, which is most $1/2$. Also note that the length of time job $j$ is processed by a combination of 1-speed routers
only is strictly less than 1/2 time step. Hence the second property holds.

Due to the second property, the online algorithm cannot find the big subtree incident to the root. This is because all subtrees incident to the root are indistinguishable to the algorithm by time 1/2, hence the adversary can pick any alive one $T_v$ of those alive, and declare it is big. Likewise, for each alive non-leaf node $v'$, the adversary can keep alive the big child of $v'$. Hence the adversary can remove all nodes and jobs keeping only $4^{D-1}$ children including the big child for non-leaf node, and keeping only non-unit sized jobs. Also the adversary can pretend that all the remaining jobs have been processed exactly by one unit by decreasing job sizes. Observe that each alive job has remaining size $2^1 + 2^2 + ... + 2^k$ for some $k \geq 1$. Since $T_v$ is the only subtree incident to the root, we can assume that $A$ let $v$ use the 2-speed router from now on. This has the effect of decreasing each job’s remaining size by half, and this exactly coincides with the instance $I(D-1)$. This will allow the adversary recurse on the instance $I(D-1)$, thereby making $A$’s makespan no smaller than $D/2 = \Omega(\sqrt{\log n})$. This, together with, Lemma 74, establish a lower bound $\Omega(\Delta) = \Omega(\sqrt{\log n})$ for makespan, thus proving Theorem 73 and the lower bound claimed in Theorem 58.

We formalize several notions (such as decreasing job sizes, indistinguishable instances) we used above to make the argument more clear. To this end it will be useful to define the collection $S := S(0)$ of possible instances that the adversary can use. The adversary will gradually decrease the instance space $S(t)$ depending on the algorithm’s choice; $S(t)$ can only decrease in time $t$. Equivalently, the deterministic algorithm $A$ cannot distinguish between instances in $S(t)$ at the moment of time $t$, and hence must behave exactly the same by time $t$ for all the instances in $S(t)$. In this sense, all instances in $S(t)$ are indistinguishable to $A$ by time $t$. All instances in $S$ follow the same polytope constraints for $I(D)$.

There are two factors that make instances in $S$ rich. The first factor is “hidden” job IDs: Each non-leaf node $v$ has only one big child, and it can be any of its children $C_v$. In other words, this is completely determined by a function $\psi$ that maps each non-leaf node $v$ to one of $v$’s
children, $C_v$. Consider any fixed $\psi$. Then for each non-leaf node $v$ of depth $D - 1$, the sizes that $v$’s children can have are fixed – however, the actual mapping between jobs and job sizes can be arbitrary. So far, all instances can be viewed equivalent in the sense that they can be obtained from the same instance by an appropriate mapping. By a job $j'$ ID, we mean the job in the common instance which corresponds to job $j$ in the common instance. The second factor is “flexible” job sizes. Note that in $I(D)$, a job $j$’s ID determines its size completely. This is not the case in $S$, and we let each job have any size up to the size determined by its ID.

The adversary will start with set $S(0, D)$ – here we added $D$ since the set is constructed from $I(D)$. The adversary’s goal is to have $S(1/2, D)$ which essentially includes $S(0, D - 1)$. By recursively applying this strategy, the adversary will be able to force $A$ have a makespan of at least $D/2$. As observed in Lemma 74, for any instance in $S(0)$, all jobs in the instance can be completed by time 2 by the optimal solution, and this will complete the proof of Theorem 73.

We make use of the two crucial properties we observed above. In particular, the second property ensures that at time $1/2$, any job ID mapping remains plausible in the solution set $S(1/2, D)$. Hence the adversary can choose any alive subtree $T_v$ incident to the root, and delete other sibling subtrees. From time $1/2$, any instance in $S(1/2, D)$ must satisfy the constraint that $v$ is big. The adversary now deletes all nodes/jobs in $T_v$ so that each node has $4^{D-1}$ children. Here the adversary can still choose any mapping (from each set of the alive siblings, the adversary can set any node/job to be big), and this has the same structure that $S(0, D - 1)$ has regarding the “hidden Job ID” flexibility. Then, as mentioned before, by decreasing job sizes (more precisely, the corresponding instances are removed from $S(1/2, D)$) so that $S(1/2, D)$ becomes the same as $S(0, D - 1)$ – the only difference is that the root of the big subtree in $S(1/2, D)$ is processed via 2-speed router, however this difference is nullified by the fact that the each job in any instance in $S(1/2, D)$ has exactly the double size that the corresponding job in the corresponding instance in $S(0, D - 1)$ has. This allows the adversary to apply his strategy recursively. Hence we derive the following
lemma which completes the proof of Theorem 73 and the lower bound in Theorem 58.

**Lemma 75.** For any instance in $S(D)$, there is a way to complete all jobs in the instance within time 2. In contrast, for any deterministic non-clairvoyant algorithm $A$, there is an instance in $S(D)$ for which $A$ has a makespan of at least $D/2$.

9.5 Notes

This chapter is based on joint work with Sungjin Im and Kamesh Munagala (Im et al., 2014a). A preliminary version containing some of the results presented in this chapter appeared in Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014
10

Constant Competitive Flow-Time Algorithms for Monotone PSP

10.1 Introduction

The generality of the PSP problem leads to strong lower bounds: As we saw in the last chapter, there is an instance of PSP for which no deterministic algorithm is $O(n^{1-\epsilon})$-competitive on the flow time objective for any constant $0 < \epsilon < 1$, unless the algorithm runs with $\Omega(\sqrt{\log n})$ speed compared to the optimal algorithm. The above negative result is in sharp contrast with positive results for special cases such as broadcast scheduling and unrelated machine scheduling. This motivates the following question: Are there natural sub-classes of PSP for which it is possible to design competitive algorithms for flow time?

In this chapter, we consider fairly broad classes of scheduling problems for which we can prove better bounds. These problems not only generalize several well-studied models such as scheduling with speedup curves and related machine scheduling (Gupta et al., 2012b; Im et al., 2014a,b), but also capture as special cases hitherto unstudied problems such as single sink flow routing, routing multicast trees, etc.
10.1.1 Resource Allocation View and Monotone PSP

In the PSP problem, the instantaneous allocation of rates to jobs can be viewed as a resource allocation problem. A natural algorithm for performing resource allocation is the proportional fairness (PF) algorithm we considered in the previous chapters.

$$\max \sum_j w_j \log y_{jt} \quad \text{s.t.} \quad y_t \in \mathcal{P}$$

Note that $\mathcal{P}$ is a convex set in the definition of PSP. Though this algorithm is constant competitive for completion time objective (Im et al., 2014a), the same is not true for flow time; the best previous result requires the speed to depend on the number of jobs. We therefore analyze the PF algorithm under a natural restriction on the utility functions. Suppose the current set of jobs is $S$. Let $y_j(S)$ denote the rate allocated by PF to job $j \in S$.

**Definition 76** (Monotonicity of PF). The PF algorithm is said to be monotone if for any $S$ and $\ell \notin S$, we have the following condition. For all $j \in S$, $y_j(S) \geq y_j(S \cup \{\ell\})$. The class Monotone PSP is the sub-class of PSP for which the PF algorithm is monotone.

Our goal in this chapter is to design online, non-clairvoyant algorithms for the class Monotone PSP.

10.1.2 Our Results

Our main result is the following theorem, which we prove in Section 10.2.

**Theorem 77.** For the Monotone PSP problem, for any constant $\epsilon \in (0, 1/2)$, PF is $(e + \epsilon)$-speed, $O(1/\epsilon^2)$ competitive for minimizing weighted flow time.

We show the above theorem by amortized local competitiveness (Im et al., 2011a). The potential function we use is a natural generalization of the potential for single-machine weighted round robin (WRR) considered in (Fox et al., 2013); nevertheless, our analysis is very different since we don’t have a closed form for the allocations or rates. We crucially
use the optimality condition for the PF convex program (Proposition 80), which we apply iteratively considering the first $k$ jobs in arrival order versus the first $k+1$ jobs. This yields a simple analysis that works for any MONOTONE PSP instance.

The next question is to identify problem classes that belong to MONOTONE PSP. We present two broad sub-classes of PSP that are monotone. Showing monotonicity requires making connections to market clearing literature, particularly Walrasian equilibria and Gross Substitutes (Gul and Stacchetti, 1999), and the Submodular Utility Allocation markets defined in (Jain and Vazirani, 2010).

10.1.3 Resource Allocation with Substitutes (RA-S)

We revisit the multi-dimensional resource allocation problem considered above. Formally, there are $D$ divisible resources (or dimensions), numbered $1, 2, \ldots, D$. We assume w.l.o.g. (by scaling and splitting resources) that each resource is available in unit supply. If job $j$ is assigned a non-negative vector of resources $x = \{x_1, x_2, \ldots, x_D\}$, then the rate at which the job executes is given by $y_j = u_j(x)$, where $u_j$ is a concave utility function that is known to the scheduler. The constraints $P$ simply capture that each resource can be allocated to unit amount, so that $\sum_j x_{jd} \leq 1$ for all $d \in \{1, 2, \ldots, D\}$. A well-studied special class of utilities are the Constant Elasticity of Scale (CES) utilities, given by:

$$u_j(x_j) = \left( \sum_{d=1}^{D} c_{jd}^{\rho_j} x_{jd}^{\rho_j} \right)^{1/\rho_j} \tag{10.1}$$

A parameter range of special interest is when $\rho \in (0, 1]$ – these utility functions are widely studied in economics, and capture resources that are imperfect substitutes of each other, where the parameter $\rho$ captures the extent of substitutability. A special case as $\rho \to 0$ is termed Cobb-Douglas utilities: $u_j(x_j) = \prod_{d=1}^{D} x_{jd}^{\alpha_{jd}}$, where $\sum_d \alpha_{jd} \leq 1$ and $\alpha_{jd} \geq 0$ for all $j,d$. These utilities can be used to model task rates in heterogeneous microprocessor architectures (Zahedi and Lee, 2014). When $\rho = 1$, CES utilities reduce to linear utilities.

In this chapter, we generalize CES functions to a broader class that we term resource
allocation with substitutes or RA-S.

\[ u_j(x_j) = \left( \sum_{d=1}^{D} (f_{jd}(x_{jd}))^{\rho_j} \right)^{1/\rho_j} \] \hspace{1cm} \text{where } \rho_j \in (0, 1] \text{ and } \rho_j' \geq \rho_j \quad (10.2)

Here, the \( \{f_{jd}\} \) are increasing, smooth, strictly concave functions, with \( f_{jd}(0) = 0 \). As before, the constraints \( P \) simply capture that each resource can be allocated to unit amount, so that \( \sum_j x_{jd} \leq 1 \) for all \( d \in \{1, 2, \ldots, D\} \). The special case as \( \rho \to 0 \) corresponds to \( u_j(x_j) = \prod_{d=1}^{D} (f_{jd}(x_{jd}))^{\alpha_{jd}} \), where \( \sum_d \alpha_{jd} \leq 1 \) and \( \alpha_{jd} \geq 0 \) for all \( j, d \), which can be viewed as Generalized Cobb-Douglas utilities. The single-dimensional case \( (D = 1) \) corresponds to scheduling with concave speedup curves, which has been extensively studied in literature (Edmonds et al., 2011; Edmonds and Pruhs, 2012; Fox et al., 2013). Though we do not present details in this chapter, our algorithmic results also extend to a slightly different class of utilities of the form: \( u_j(x_j) = g_j \left( \sum_{d=1}^{D} f_{jd}(x_{jd}) \right) \), where \( g_j \) is increasing, smooth, and strictly concave, with \( g_j(0) = 0 \).

**Monotonicity of RA-S.** We prove that RA-S is a special case of MONOTONE PSP in Section 10.2.2. We present the intuition and new technical ideas here. First consider CES utilities given by Eq (10.1), when \( \rho \in [0, 1] \). Recall that for these utilities, the feasibility constraints \( P \) simply encode that each resource is allocated to at most the supply of one unit. CES utilities are homogeneous of degree one and quasi-concave. Therefore, the PF algorithm computes the market equilibrium to the equivalent Fisher market. Each job \( j \) is an agent with budget \( w_j \). Suppose resource \( d \) has price \( p_d \). Each agent buys resources to solve the following maximization problem:

\[
\text{Maximize } u_j(x_j) \quad \text{s.t. } \sum_{d=1}^{D} p_d x_{jd} \leq w_j
\]

A Fisher equilibrium (or market clearing solution) is a set of prices \( \{p_d\} \) such that the per-agent utility maximizing allocations clear the market: No resource is over allocated; for
each resource with non-zero price, supply equals demand; and each agent spends its entire budget. This follows from the KKT conditions. These utilities satisfy a property termed \textit{Gross Substitutability} (GS) (Gul and Stacchetti, 1999). The GS property means that when the price of a resource increases, the demand for resources whose prices did not increase only goes up. For utilities satisfying GS, it is easy to show that a market clearing solution will be monotone. Since the PF algorithm computes this solution, it satisfies monotonicity (Def. 12.5.2).

For the RA-S utilities (Eq. (10.2)), the PF algorithm no longer coincides with a Fisher equilibrium. We therefore prove monotonicity of the PF algorithm from first principles. Our proof proceeds by considering \(\log u_j(x)\) as a utility function, and viewing the PF algorithm as computing a Walrasian equilibrium (Gul and Stacchetti, 1999) of this utility function. The GS property would imply that the equilibrium can be computed by a monotone tatonnement process, and when a new agent arrives, the tatonnement only increases prices, therefore lowering utility. The key technical hurdle in our case is that the utility function \(\log u_j(x)\) is not zero when \(x = 0\); in fact it can be unbounded. We therefore need to show a stronger condition than the usual GS property in order to establish monotonicity. The end result is the following corollary to Theorem 77.

**Corollary 78.** For RA-S utilities, the PF algorithm is \((e + \epsilon)\)-speed, \(O\left(\frac{1}{\epsilon^2}\right)\) competitive for minimizing weighted flow time, where \(\epsilon\) is any constant in \((0, 1]\).

10.1.4 Polymatroidal Utilities

This sub-class of PSP is given by the following polyhedron:

\[
P = \left\{ \sum_{j \in S} y_j \leq v(S) \quad \forall \text{ subsets of jobs } S \right\}
\]

where the function \(v(S)\) is a non-decreasing submodular function with \(v(\emptyset) = 0\). The feasible region \(P\) is therefore a polymatroid. Many natural resource allocation problems define polymatroids:
Single-sink Flow Routing. We are given a directed capacitated graph $G(V, E)$, with capacities $c(e)$ on edge $e \in E$. Each job $j$ is characterized by a pair of source-sink vertices, $(s_j, t_j)$, as well as a total flow value $p_j$ and weight $w_j$. If we allocate flow value $y_{jt}$ for job $j$ at time $t$, then $y_{jt}$ should be a feasible flow from $s_j$ to $t_j$. The $\{y_{jt}\}$ values should satisfy the capacity constraints on the edges. In the case where all jobs need to route to the same sink node $t$, the rate region $\mathcal{P}$ is a polymatroid: For a subset of jobs $S$, let $v(S)$ denote the maximum total rate that can be allocated to jobs in $S$, then $v(S)$ is a submodular function (Megiddo, 1974). A classical result of (Kelly et al., 1998) shows that the TCP congestion control algorithm can be viewed as an implementation of proportional fairness in a distributed fashion. Our result shows that such an implementation is competitive on delays of the flows, assuming they are routed to a single sink.

Video-on-Demand (Multicast). Consider a video-on-demand setting (Bikhchandani et al., 2011), where different sources of video streams on a network need to stream content to all network vertices via spanning trees. Formally, there is a capacitated undirected graph $G(V, E)$ with a sink node $s \in V$. Job (video stream) $j$ arrives at node $v_j$. If job $j$ is assigned $x_T$ units of spanning tree $T$, the rate it gets is $x_T$; this rate is additive across trees. Any feasible allocation is therefore a fractional assignment of spanning trees to jobs, so that along any edge, the total amount of trees that use that edge is at most the capacity of the edge. This rate polytope $\mathcal{P}$ is a polymatroid (Bikhchandani et al., 2011).

Related Machine Scheduling. There are $M$ machines, where machine $m$ has speed $s_m$. The machines are fractionally allocated to jobs; let job $j$ be assigned $x_{jm}$ units of machine $m$. The feasibility constraints $\mathcal{P}$ require that each machine can be fractionally allocated by at most one unit, so that $\sum_j x_{jm} \leq 1$ for all $m$; and each job is allocated at most one unit of machines, so that $\sum_m x_{jm} \leq 1$ for all $j$. The rate of job $j$ is $u_j(x) = \sum_m s_m x_{jm}$. It is known (Feldman et al., 2008) that the space $\mathcal{P}$ of feasible rates define a polymatroid. While there already exists an $O(1)$-speed $O(1)$-competitive algorithm for this problem, we
find this result interesting since the algorithm is very different from (Im et al., 2014b).

**Monotonicity of Polymatroidal Utilities.** Jain and Vazirani (Jain and Vazirani, 2010) generalize Fisher markets to *polymatroidal utilities*, which they term Submodular Utility Allocation (SUA) markets. They show that the PF algorithm computes the market clearing solution. For such markets, they define the notion of *competition monotonicity*: A new agent entering the market leads to greater competition, and hence to lower utilities for existing agents. They show that this market is competition monotone, which directly implies the PF algorithm is monotone for polymatroidal utilities, leading to the following corollary of Theorem 77.

**Corollary 79.** For polymatroidal utilities, the PF algorithm is \((e+\epsilon)\)-speed \(O\left(\frac{1}{\epsilon^2}\right)\) competitive for minimizing weighted flow time, where \(\epsilon\) is any constant in \((0, 1]\).

10.2 The Proportional Fairness (PF) Algorithm and MONOTONE PSP

Let \(A_t\) denote the set of jobs that are alive at time \(t\); we will often drop \(t\) when the time \(t\) in consideration is clear from the context. These include jobs \(j\) for which \(t \in [r_j, C_j]\). The proportional fairness algorithm computes a rate vector \(y_t\) that optimizes:

\[
\text{Maximize } \sum_{j \in A_t} w_j \log y_j \quad \text{s.t. } y \in \mathcal{P}
\]

Let \(y_j^*(S)\) denote the optimal rate the PF algorithm allocates to job \(j \in S\) when working on a set of jobs, \(S\). We will use the following well-known proposition repeatedly in our analysis.

**Proposition 80** (Optimality Condition). Let \(y \in \mathcal{P}\) denote any feasible rate vector for the jobs in \(S\). If the space of feasible rates \(\mathcal{P}\) is convex, then

\[
\sum_{j \in S} w_j \frac{y_j}{y_j^*(S)} \leq \sum_{j \in S} w_j
\]
Proof. For notational simplicity, let \( y^*_j := y^*_j(S) \). Let \( f(y) = \sum_{j \in S} w_j \log y_j \). We have \( \frac{\partial f(y^*)}{\partial y_j} = \frac{w_j}{y^*_j} \). The optimality of \( y^* \) implies \( \nabla f(y^*) \cdot (y - y^*) \leq 0 \) for all \( y \in \mathcal{P} \). The proposition now follows by elementary algebra.

Recall that we analyze the PF algorithm under a natural restriction on the utility functions. Recall from Definition 12.5.2 that the PF algorithm is said to be monotone if for any \( S \) and \( \ell \notin S \), we have the following condition: for all \( j \in S \), \( y^*_j(S) \geq y^*_j(S \cup \{\ell\}) \). We term this class of PSP problems as MONOTONE PSP.

10.2.1 Competitive Analysis

We use amortized local competitiveness to show the Theorem 77. The potential function we use is the same as that for one-dimensional concave speedup curves (Fox et al., 2013); however, our analysis is different and repeatedly uses Proposition 80 to bound how the potential function changes when the algorithm processes jobs.

Focus on some time instant \( t \), and define the following quantities. Let \( A_t \) denote the subset of jobs alive in the PF schedule, and let \( O_t \) denote those alive in OPT’s schedule. For job \( j \), let \( p_{jt} \) denote the remaining size of the job in the PF’s schedule, and let \( p_{Ojt} \) denote the remaining size of the job in OPT’s schedule. Define a job \( j \)’s lag as \( \tilde{p}_{jt} = \max(0, p_{jt} - p_{Ojt}) \).

The quantity \( \tilde{p}_{jt} \) indicates how much our algorithm is behind the optimal schedule in terms of job \( j \)’s processing. Let \( L_t = \{ j \in A_t \mid \tilde{p}_{jt} > 0 \} \). Note that \( A_t \setminus L_t \subseteq O_t \).

Consider the jobs in increasing order of arrival times, and number them \( 1, 2, \ldots \) in this order. Let \( A_t^{\leq j} = A_t \cap \{1, 2, \ldots, j\} \). Recall that \( y^*_j(S) \) denote the optimal rate the PF algorithm allocates to job \( j \in S \) when working on a set of jobs, \( S \). We define the following potential function:

\[
\Phi(t) = \frac{1}{\epsilon} \sum_{j \in A_t} w_j \frac{\tilde{p}_{jt}}{y^*_j(A_t^{\leq j})}
\]

We first show the following simple claim, similar to the one in (Fox et al., 2013). This crucially needs the monotonicity of the PF algorithm, and we present the proof for com-
Claim 81. If $\Phi(t)$ changes discontinuously, this change is negative.

Proof. If no jobs arrive or is completed by PF or OPT, the $\tilde{p}$ values change continuously, and the $y_j^*(A_t^{\leq j})$ values do not change. Hence, the potential changes continuously. Suppose a job $j'$ arrives; for notational convenience, we assume that the current alive jobs are $A_t$ plus the job $j'$ that just arrived, and $j' \notin A_t$. For this job, $\tilde{p}_{jt} = 0$. Furthermore, this job does not affect $y_j^*(A_t^{\leq j})$ for any $j \in A_t$, since $j' \notin A_t^{\leq j}$. Therefore, the potential does not change when a job arrives. Similarly, suppose a job $j'$ is completed by OPT but $A_t$ remains unchanged. Then, none of the terms in the potential change, and hence $\Phi(t)$ does not change. Finally, consider the case where $j'$ departs from $A_t$. We have $\tilde{p}_{jt} = 0$. This departure can change $y_j^*(A_t^{\leq j})$ for $j \in A_t$ s.t. $j' \leq j$. By the monotonicity of the PF algorithm, these rates cannot decrease. Therefore, all terms in the potential are weakly decreasing, completing the proof. \qed

Assuming that PF uses a speed of $(e + \epsilon)$ compared to OPT, we will show the following at each time instant $t$ where no job arrives or is completed either by PF or OPT. Here, $W(S) = \sum_{j \in S} w_j$.

\begin{equation}
W(A_t) + \frac{d}{dt}\Phi(t) \leq \frac{2}{\epsilon^2} W(O_t) \tag{10.3}
\end{equation}

Suppose all jobs are completed by PF and OPT by time $T$. Then integrating the above inequality over time, and using the fact that the discontinuous changes to $\Phi$ are all negative, we have:

$$
\int_{t=0}^{T} W(A_t)dt + (\Phi(T) - \Phi(0)) \leq \frac{2}{\epsilon^2} \int_{t=0}^{T} W(O_t)dt
$$

Note that the first term above is the weighted flow time of PF, the second term is zero, and the RHS is the weighted flow time of OPT. This will complete the proof of Theorem 77.
Proving Inequality (10.3)

Consider a time instant $t$ when no job arrives or completes. To simplify notation, we omit the subscript $t$ from the proof. Let $\frac{d}{dt}\Phi|_O$ and $\frac{d}{dt}\Phi|_A$ denote the potential changes due to OPT’s processing and PF’s processing respectively. Note that $\frac{d}{dt}\Phi = \frac{d}{dt}\Phi|_A + \frac{d}{dt}\Phi|_O$.

Lemma 82. $\frac{d}{dt}\Phi|_O \leq \frac{1}{\epsilon} W(A)$.

Proof. For job $j \in A$, suppose OPT assigns rate $y^O_j$. Then, $\frac{d}{dt}\tilde{p}_j \leq y^O_j$ for $j \in A$, due to OPT’s processing. Therefore, the change in potential is upper bounded by:

$$\frac{d}{dt}\Phi|_O \leq \frac{1}{\epsilon} \sum_{j \in A} w_j \frac{y^O_j}{y^*(A^j)} \leq \frac{1}{\epsilon} \sum_{j \in A} w_j \frac{y^O_j}{y^*(A)}$$

The inequality above follows from the monotonicity of the PF algorithm, since $A^j \subseteq A$. Using Proposition 80, the RHS is at most $W(A)$. This completes the proof. 

We now bound $\frac{d}{dt}\Phi|_A$, the change in potential due to PF. We first assume PF runs at speed 1, and we will scale this up later. We consider two cases:

Case 1. Suppose $W(L) \leq (1 - \epsilon)W(A)$. Since $A \setminus L \subseteq O$, we have $W(O) \geq \epsilon W(A)$. Since $\frac{d}{dt}\Phi|_A \leq 0$, we have:

$$W(A) + \frac{d}{dt}\Phi \leq W(A) + \frac{d}{dt}\Phi|_O \leq \frac{2}{\epsilon} W(A) \leq \frac{2}{\epsilon^2} W(O)$$

where the second inequality follows from Lemma 82.

Case 2. The more interesting case is when $W(L) \geq (1 - \epsilon)W(A)$. For $j \in L$, we have $\frac{d}{dt}\tilde{p}_j = y^*_j(A)$ due to PF’s processing, by the definition of $y^*_j(A)$. Therefore,

$$\epsilon \cdot \frac{d}{dt}\Phi|_A \leq - \sum_{j \in L} w_j \frac{y^*_j(A)}{y^*_j(A^j)}$$
For notational convenience, let $|S| = \kappa$, and number the jobs in $A$ in increasing order of arrival time as $1, 2, \ldots, \kappa$. For $k > j$ and $k \leq \kappa$, let $\alpha_{jk} = \frac{y^*_j(A^{\leq k-1})}{y^*_j(A^{\leq k})}$. By the monotonicity of PF, we have $\alpha_{jk} \geq 1$. Define $\delta_{jk} = \alpha_{jk} - 1$. Note that $\delta_{jk} \geq 0$.

We now apply Proposition 80 to the set $\{1, 2, \ldots, k\}$ as follows: For jobs $j \in \{1, 2, \ldots, k\}$, the rate assigned by PF when executed on this set is $y^*_j(A^{\leq k})$, and this goes into the denominator in Proposition 80. We consider $y^*_j(A^{\leq k-1})$ for $j < k$, and $y^*_k(A^{\leq k-1}) = 0$ as a different set of rates that go into the numerator in Proposition 80. This yields:

$$\sum_{j=1}^{k-1} w_j y^*_j(A^{\leq k-1}) = \sum_{j=1}^{k} w_j$$

Observing that $\frac{y^*_j(A^{\leq k-1})}{y^*_j(A^{\leq k})} = 1 + \delta_{jk}$, we obtain $\sum_{j=1}^{k-1} w_j \delta_{jk} \leq w_k$ for $k = 1, 2, \ldots, \kappa$. Adding these inequalities for $k = 1, 2, \ldots, \kappa$ and changing the order of summations, we obtain:

$$\sum_{k=1}^{\kappa} \sum_{j=1}^{k-1} w_j \delta_{jk} = \sum_{j=1}^{\kappa} w_j \left( \sum_{k=j+1}^{\kappa} \delta_{jk} \right) \leq W(A) \quad \implies \quad \sum_{j \in L} w_j \left( \sum_{k=j+1}^{\kappa} \delta_{jk} \right) \leq W(A)$$

Let $\Delta_j = \sum_{k=j+1}^{\kappa} \delta_{jk}$, so that the above inequality becomes $\sum_{j \in L} w_j \Delta_j \leq W(A)$. Now observe that

$$\frac{y^*_j(A)}{y^*_j(A^{\leq j})} = \prod_{k=j+1}^{\kappa} \frac{1}{\alpha_{jk}} = \prod_{k=j+1}^{\kappa} \frac{1}{1 + \delta_{jk}} \geq \exp \left( - \sum_{k=j+1}^{\kappa} \delta_{jk} \right) = \exp(-\Delta_j)$$

We used the fact that $\delta_{jk} \geq 0$ for all $j, k$. Therefore,

$$\epsilon \cdot \frac{d}{dt} \Phi|_{A} \leq - \sum_{j \in L} w_j \frac{y^*_j(A)}{y^*_j(A^{\leq j})} \leq - \sum_{j \in L} w_j \exp(-\Delta_j)$$

Since $\sum_{j \in L} w_j \Delta_j \leq W(A)$, the RHS is maximized when $\Delta_j = W(A)/W(L) \leq 1/(1 - \epsilon)$. This implies:

$$\epsilon \cdot \frac{d}{dt} \Phi|_{A} \leq - \sum_{j \in L} w_j \exp(-W(A)/W(L)) \leq -W(L) \exp(-1/(1 - \epsilon)) \leq -\frac{1 - 2\epsilon}{e} W(A)$$

154
for $0 < \epsilon < 1/2$. Therefore, if we run PF at speed $(\epsilon + 3\epsilon)$, we have: $\frac{d}{dt} \Phi|_A \leq - \left(1 + \frac{1}{\epsilon}\right) W(A)$. Therefore,

$$W(A) + \frac{d}{dt} \Phi|_O + \frac{d}{dt} \Phi|_A \leq W(A) + \frac{1}{\epsilon} W(A) - \left(1 + \frac{1}{\epsilon}\right) W(A) \leq 0 \leq W(O)$$

This completes the proof of Inequality (10.3) and hence of Theorem 77.

10.2.2 Monotonicity of RA-S

We will show the following theorem, which when combined with Theorem 77 shows Corollary 78.

**Theorem 83.** The RA-S utility functions defined in Eq (10.2) are concave (which implies the space $P$ is convex). Furthermore, the PF algorithm is monotone for these functions.

The first part follows by easy algebra. The CES utility function given by Eq (10.1), when $\rho \in (0, 1]$, is homogeneous of degree one and quasi-concave. This implies it is concave (Bergstrom, 2014). The RA-S utilities are obtained by a monotone concave transformation of the variables and the entire function. This preserves concavity. This implies the space $P$ of feasible utilities is convex.

The remainder of this section is devoted to proving the second part of the theorem. Recall that for RA-S, the space $P$ is given by the following (where $\rho \in [0, 1]$ and $\rho' \geq \rho$):

$$P = \left\{ y_j = u_j(x_j) = \left( \sum_{d=1}^{D} (f_{jd}(x_{jd}))^{\rho_j} \right)^{1/\rho'_j}, \quad \sum_j x_{jd} \leq 1 \forall d \right\}$$

Let $h_{jd}(x_{jd}) = (f_{jd}(x_{jd}))^{\rho_j}$. This function is increasing and strictly concave, assuming the same is true for $f_{jd}$. Further $h_{jd}(0) = 0$. Define:

$$v_j(x_j) = w_j \log u_j(x_j) = \frac{w_j}{\rho'_j} \log \left( \sum_{d=1}^{D} h_{jd}(x_{jd}) \right)$$

For price vector $p = \{p_1, p_2, \ldots, p_D\} \geq 0$, define the demand function $X_j(p)$ as follows:

$$X_j(p) = \arg\max_{x_j \geq 0} (v_j(x) - p \cdot x) \quad \text{and} \quad U_j(p) = u_j(X_j(p))$$

155
Note that $X_j(p)$ is uniquely defined for given $p$ due to the strict concavity of $v_j(x)$ (see the first part of Theorem 83), so $U_j(p)$ is well-defined.

**Lemma 84.** Consider an arbitrary price vector $p$, and a different price vector $p'$ that only differs from $p$ in the $r$th dimension. Assume $p'_r > p_r$. Let $x_j = X_j(p)$ and $x'_j = X_j(p')$. Then:

1. If $x_{jr} > 0$, then $U_j(p') < U_j(p)$. Furthermore, $x'_{jr} < x_{jr}$, and for all $d \neq r$, $x'_{jd} \geq x_{jd}$.

2. If $x_{jr} = 0$, then $U_j(p') = U_j(p)$. Furthermore, for all $d$, $x'_{jd} = x_{jd}$.

Further, we have a stronger property that if $x_{jr} > c$, then $U_j(p) - U_j(p') \geq c'(p_r - p'_r)$ for a finite $c' > 0$ when the following conditions are satisfied:

- For all $j$, $d$, $h_{jd}$ has a bounded curvature over the domain $[c, C]$ for finite values $c, C > 0$, i.e. there exist $\gamma = \gamma(c, C)$ and $\delta = \delta(c, C)$ such that $\gamma(y_2 - y_1) \leq h'_{jd}(y_1) - h'_{jd}(y_2) \leq \delta(y_2 - y_1)$ for all $c \leq y_1 \leq y_2 \leq C$.

- Both vectors $p$ and $p'$ are upper bounded by a finite vector.

**Proof.** Since we focus on a single job $j$, we omit the subscript $j$ in the proof. Focus on dimension $r$. Let $q = w_j / \rho'_j$. Let $W(x) = (U_j(x))^{\rho'_j}$; since $U_j(x)$ is strictly monotone in $x$, the same holds for $W(x)$. Note that $W(\cdot)$ also can be viewed as a function of $p$ since $p$ uniquely determines $x$. Partially differentiating $v_j(x) - p \cdot x_j$ w.r.t. $x_d$, we have the following (sufficient and necessary) optimality condition:

$$x_d > 0 \implies \frac{q}{W(x)} h'_d(x_d) = p_d \quad \text{and} \quad x_d = 0 \implies \frac{q}{W(x)} h'_d(x_d) \leq p_d$$

Consider price vector $p'$ with $p'_r > p_r$ and $p'_d = p_d$ for all $d \neq r$. If $x_r = 0$, this does not change the optimality condition above for any dimension $d$, so that the second part of the lemma follows.

If $x_r > 0$, suppose $W(p') \geq W(p)$. This implies $h'(x_r) < h'(x'_r)$ since $p'_r > p_r$. To satisfy the optimality condition, we must therefore have $x'_r < x_r$ by the strict concavity of
The same argument shows that for all dimensions $d \neq r$, $x'_d \leq x_d$. But this implies $W(p') < W(p)$, which is a contradiction. Therefore, we must have $W(p') < W(p)$.

Now consider any dimension $d \neq r$. Since $p'_d = p_d$ and $W(p') < W(p)$, the optimality condition implies that $h'_d(x'_d) \leq h'_d(x_d)$. This implies $x'_d \geq x_d$. However, since the utility strictly decreased, this must imply $x'_d < x_r$.

It now remains to show the stronger property under the extra conditions. Imagine that we increase $p$ to $p'$ continuously by slowly increasing $p_r$ to $p'_r$. For simplicity, we assume that for all $d$, $p_d$ remains either non-zero or zero throughout this process, excluding the start and the end – the general case can be shown by starting a new process when $p_d$'s status, whether it is non-zero or zero, changes. Since if $p_d$ remains 0 in the process, $d$ has no effect on $W(x)$, let’s focus on $d$ with non-zero $p_d$.

Observe that boundedness of $p$ implies boundedness of $x$ since $x$ minimizes $v_j(x) - p \cdot x_j$ and $v_j(x)$ is strictly concave. Since the remaining proof follows from a tedious basic algebra, we only give a sketch here. In the following we crucially use the boundedness of $p, p', x, x'$. For the sake of contradiction, suppose $W_j(p)$ and $W_j(p')$ are very close such that the claim is not true for any fixed $c'$. Then, for all $d \neq r$ with non-zero $p_d$, the bounded curvature of $h_d$ and the optimality condition imply that $x_d$ and $x'_d$ are very close. Likewise, we can argue that $x_r$ and $x'_r$ are significantly different so that the difference is lower bounded by $c''(p'_r - p_r)$ for a fixed $c''$. This leads to the conclusion that $W_j(p')$ and $W_j(p)$ are significantly different, which is a contradiction. An easy algebra gives the desired claim.

Proof of Theorem 83. We now use Lemma 84 to show the second part of the theorem. The KKT conditions applied to the PF convex program imply the following:

1. There exists a price vector $p$ such that $\{X_j(p)\}$ define the optimal solution to PF.

2. For this price vector $p$, if $p_d > 0$, then $\sum_j x_{jd} = 1$.

Start with this optimal solution. Suppose a new job arrives. At the price vector $p$, compute the quantities $X_j(p)$. If some resource is over-demanded, we continuously increase
its price. We perform this tatonnement process until no resource is over-demanded. By Lemma 84, any job that demands a resource whose price is increasing, sees its overall utility strictly decrease, while jobs that do not demand this resource see their utility remain unchanged. Therefore, if we define the potential function to be the total utility of the jobs, this potential strictly decreases. Further, by Lemma 84, the total demand for the resource whose price is increasing strictly decreases, while the demands for all other resources weakly increase. Therefore, any resource with price strictly positive must have total demand at least one at all points of time.

Now parameterize the tatonnement process by the total price of resources. When the price of over-allocated resource $r$ is raised, there must exist a job $j$ such that $x_{jr} \geq 1/n$. This, when combined with the optimal condition, implies that $p_r$ is bounded. Since we only increase the price of over-demanded resources, the boundness of $p$ follows. Hence by the stronger property of Lemma 84, the potential must decrease by at least $c'$ times the increase of the total price for some finite $c' > 0$; the potential decreases at least as much as $j$’s utility does. This implies that the process must terminate since the potential is lower bounded by zero. When it terminates, suppose the price vector is $p'$, and let $x'_j = X_j(p')$. Any resource $d$ with $p'_d > 0$ must have $\sum_j x'_{jd} = 1$. If $p_d = 0$, we must have $\sum_j x'_{jd} \leq 1$. This therefore is the new optimal solution to the PF program. Since the utilities of all existing jobs either stay the same or decrease in the tatonnement process, this shows the PF algorithm is monotone. This completes the proof of Theorem 83.

A similar proof to the above shows the following; we omit the details.

**Corollary 85.** The PF algorithm is monotone for utility functions of the form $u_j(x_j) = g_j \left( \sum_{d=1}^{D} f_{jd}(x_{jd}) \right)$, where $g_j, f_{jd}$ are increasing, smooth, and strictly concave functions.

### 10.3 Summary and Open Problems

In this chapter we showed that for a fairly general subclass of the PSP problem the PF algorithm is constant competitive. One interesting research direction is to show more utility
functions for which PF allocation is monotone, which may be of independent interest besides yielding constant competitive algorithms for average flow-time. Also, our speed augmentation bound may not be the tightest one can get. Can we design $O(1 + \epsilon)$-speed $O(1)$-competitive algorithms for the MONOTONE PSP problem?

10.4 Notes

This chapter is based on joint work with Sungjin Im and Kamesh Munagala, and is under a conference submission (Sungjin Im and Munagala, 2015).
Queueing View of PSP

11.1 Introduction

Continuing the theme of the last chapter of identifying important special cases of the PSP problem for which we can get better algorithms, in this chapter, we show another fairly general sub-class of the PSP problem for which we can obtain better algorithms. Towards that, we take a queueing viewpoint of the PSP problem. One special case of the PSP problem that arises in practice is the case when the jobs can be grouped into a small number of types or queues. Jobs within a queue have different sizes and weights, but are interchangeable for the purpose of scheduling. Our goal will be to derive an algorithm whose competitive ratio depends on the number of queues instead of the number of jobs.

We define PSP-Q as follows: There are $K$ queues. Job $j$ has processing length $p_j$, weight $w_j$, and arrives at queue $q_j$. At each step $t$, let $y_{jt}$ denote the rate assigned to job $j$. Let $S_q$ denote the set of jobs in queue $q$ at time $t$. A feasible allocation at time $t$ is given by the following, where we drop subscript $t$:

$$\left\{ z_q = \sum_{j \in S_q} y_j \forall q; \mathrm{and} \; z \in P_q \right\}$$
where $\mathcal{P}_q$ is a downward closed convex space. Note jobs in the same queue are completely interchangeable, in that feasibility is determined by the total rate at which jobs are processed in each queue. In Section 11.2 we show that PSP-Q problem captures several interesting problems such as flow-routing and multidimensional scheduling problem.

Our algorithms for the PSP-Q problem are inspired by the algorithms considered in queueing literature for similar problems. The Max-Weight algorithm was first presented in the seminal work of (Tassiulas and Ephremides, 1992). As with most queueing literature, their work focuses on stability, and assumes jobs are unit-sized and unweighted. To generalize this algorithm for the PSP-Q problem, we need a normalization step that we describe below. While we believe our extension is flexible enough to be combined any single machine scheduling algorithm for each queue, for simplicity, we will only consider two algorithms: Highest Density First (HDF) and an extension of Weighted Round Robin (WRR).

The algorithm Max-Weight+WRR is defined as follows. Let $g_q = \max\{z_q | z \in \mathcal{P}_q\}$ be the maximum possible rate that can be assigned to queue $q$. At time $t$, let $S_{qt}$ denote the set of jobs in queue $q$. Let $W_{qt} = \sum_{j \in S_{qt}} w_j$. The Max-Weight algorithm assigns rates $z_{qt}$ as follows:

$$\max \sum_q W_q \frac{z_{qt}}{g_q} \quad \text{s.t.} \quad z_t \in \mathcal{P}_q$$

Given the rate $z_{qt}$ for queue $q$, the algorithm sets $y_{jt} = \frac{w_j}{W_q} z_{qt}$ for $j \in S_{qt}$. In contrast with queueing literature, the above algorithm maximizes the weight of the queue times the normalized rate $\frac{z_{qt}}{g_q}$.

11.1.1 Our Results

Our main result in this chapter is the following.

**Theorem 86.** For the PSP-Q problem with $K$ queues, for any constant $\epsilon > 0$, there is a non-clairvoyant algorithm that is $(1 + \epsilon)$-speed $O(K^3/\epsilon^3)$-competitive for weighted flow time. In addition, the Max-Weight+WRR algorithm with speed $(2 + \epsilon)$ is $O(K^3/\epsilon^3)$
competitive for \(PSP-Q\).

We give the proof of the theorem in Section 11.3. The \((1+\epsilon)\) speed algorithm is obtained by replacing WRR within a queue with a scalable single machine scheduling algorithm; see Section 11.3 for details. We can also replace the WRR algorithm for individual queues with the HDF algorithm, thereby obtaining a clairvoyant algorithm, Max-Weight+HDF which is also \((1+\epsilon)\)-speed, \(O(K/\epsilon^2)\)-competitive; see Section 11.4 for details. We note that even with clairvoyance, better previous results were not known.

The above results are quite surprising – it is easy to obtain algorithms whose speed depends on \(K\); we instead find an algorithm whose speed is an arbitrarily small constant. At a high level, this is similar to stability analysis in queueing, and indeed, we show the above result by using ideas from that field. A queueing system with an ergodic arrival process is said to be stable if the expected queue sizes are finite. The proof that the Max-Weight algorithm is stable (Tassiulas and Ephremides, 1992; McKeown et al., 1999; Maguluri and Srikant, 2013) uses a Lyapunov function argument: The Lyapunov function is the sum of the squares of the queue weights. The rate of change of this function is simply the sum over the queues of the weight of the queue times the rate of change of this weight. Since the arrival process is admissible, it is easy to show that the expected rate of change is larger for the Max-Weight algorithm than for the arrival process, hence showing that the drift in queue size is always negative.

Though the arrival process is adversarial in our case, we show that Lyapunov functions of a certain form can be converted to a potential function that is defined on the difference in weights between the algorithm’s and optimal solution’s queues, leading to an amortized local competitiveness analysis (Im et al., 2011a). This connection is not straightforward. The key challenge is that while in stability, we need to analyze the change of the Lyapunov function due to arrivals which are fixed and stochastic, in our setting, we have to analyze the change in potential due to the optimal solution’s processing. Therefore, though the Lyapunov function method shows stability not just for Max-Weight, but also for any
scheduling policy that maximizes some weighted function of rates (such as proportional
fairness, square of weights), this does not imply we can convert all these Lyapunov functions
into potential functions. In hindsight, our result is the first to use queueing theoretic ideas
(stability) to perform adversarial (competitive) analysis for a natural class of scheduling
problems.

Since the PSP-Q problem reduces to the PSP problem when each job defines its own
queue, the polynomial dependence on $K$ is unavoidable for all non-clairvoyant algorithms
if we insist on constant speed (Im et al., 2014a).

11.2 Applications of PSP-Q

Though any PSP instance has an equivalent PSP-Q instance, we present some problems
that can be naturally modeled by PSP-Q framework.

11.2.1 Flow Routing

Consider the Flow Routing problem (Kelly et al., 1998) from Section 10.1.4. When
the $(s_i, t_i)$ pairs for the jobs can be arbitrary, we can group jobs that require the same
source-sink pair into one queue, so that there are at most $K = |V|^2$ queues. This yields a
competitive ratio of $O(|V|^6)$ with $(1 + \epsilon)$ speed. Note that $|V|$ is the number of vertices in
the underlying graph $G(V, E)$, which is fixed and independent of $n$, the number of flows
that arrive and depart over time.

11.2.2 Resource Allocation with Complementarities (RA-C)

Consider the multi-dimensional resource allocation problem presented in Section 10.1.3.
The class of Leontief utilities is given by $u_j(x_j) = \min_{d=1}^{D} (c_{jd}x_{jd})$, where $c_j$ is the resource
vector of the job. Such utilities capture resources that are complements of each other.
We term this problem the Resource Allocation with Complementarities or RA-C problem.
These utilities are widely used to model job rates in data centers (Ghodsi et al., 2011;
Zaharia et al., 2008; Ahmad et al., 2012; Popa et al., 2012). In the RA-C problem, a job
is characterized by its resource requirement vector $c_j$. In practice (Maguluri and Srikant,
2013), the number of distinct resource vectors is a small number $K$, and it can be checked that this is a special case of PSP-Q with $K$ queues. We show in Section 11.5 that even in the full generality of RA-C, a simple preprocessing shows that $O((\log D)^D)$ queues suffice, albeit with the knowledge of job sizes, which makes the overall algorithm clairvoyant.

11.3 Non-Clairvoyant Algorithm for PSP-Q

This section is devoted to proving Theorem 86. We show that the algorithm NORMALIZED MAX-WEIGHT gives us the desired result when combined with WEIGHTED LATEST ARRIVAL PROCESSOR SHARING (WLAPS), which is a scalable generalization of WRR (Edmonds and Pruhs, 2012). We begin by describing the algorithm.

11.3.1 Algorithm Description

Recall that $z_{qt}$ denotes the rate assigned to $q$ at time $t$, and $g_q$ is the maximum rate that can be assigned to queue $q$ ($g_q = \max\{z_q \mid z \in \mathcal{P}_q\}$). For the sake of analysis we scale the input instance such that $g_q = 1$ for all queues $q \in [K]$. This can be done by scaling the polytope such that $g_q = 1$ for all queues and modifying the size of each job $j$ in $q$ to $p_j/g_q$. It is easy to verify that an optimal solution remains unchanged due to this scaling. We emphasize that this scaling and modification is done only for the sake of analysis, and our algorithm does not have to know the job sizes – this conversion can be thought as being done at the end of algorithm’s run. Indeed, our algorithm will only need to know the set of alive jobs in each queue along with their weights, but not their remaining sizes. Hence our algorithm will remain non-clairvoyant. For the remainder of this section we assume that $g_q = 1$ for all $q \in [K]$.

The NORMALIZED MAX-WEIGHT algorithm calculates a feasible set of rates $z_{qt}$ for each queue at each time instant $t$ by solving the following convex program. Recall that $W_{qt}$ is the total weight of jobs alive in the algorithm’s queue $q$.

$$\max \sum_{q \in [K]} W_{qt} \cdot z_{qt} \quad \text{s.t.} \quad z_t \in \mathcal{P}_q$$

164
The total rate allocated to a queue \( q \) is distributed among the jobs in the queue \( q \) using the WLAPS policy. Let \( A_{qt} \) denote the set of jobs that are in the algorithm’s queue \( q \) at time \( t \). Let \( A^\epsilon_{qt} \) denote the minimal set of latest arriving jobs in \( A_{qt} \) such that their total weight is at least \( \epsilon \cdot W_{qt} \), for some parameter \( \epsilon \in [0, 1] \). Our algorithm distributes the total rate \( z_{qt} \) among the jobs in the set \( A^\epsilon_{qt} \).

\[
y_{jt} = z_{qt} \cdot \frac{w_j}{\epsilon \cdot W_{qt}} \quad \forall j \in A^\epsilon_{qt},
\]

with the exception that the rate assigned to the earliest arriving job \( j' \) in \( A^\epsilon_{qt} \) is

\[
y_{j't} = z_{qt} \cdot \frac{\epsilon \cdot W_{qt} - \sum_{i \in A^\epsilon_{qt} \setminus \{j'\}} w_i}{\epsilon \cdot W_{qt}}
\]

In other words, job \( j' \) gets the remaining rate from \( z_{qt} \) after assigning rates to other jobs in \( A^\epsilon_{qt} \). Observe that if \( \epsilon = 1 \), then the algorithm distributes total available rate \( z_{qt} \) using WRR. This completes the description of our algorithm. To make our analysis more transparent, we will assume that \( \sum_{q \in [K]} W_{qt} \cdot z_{qt} = \epsilon W_{qt} \) which allows us to ignore the exception. This simplifying assumption can be easily removed.

We observe two simple properties. Let \( W_{\max,t} \) denote the highest weight over all queues; that is, \( W_{\max,t} = \max_{q \in [K]} W_{qt} \).

**Observation 5.** \( \sum_{q \in [K]} W_{qt} \cdot z_{qt} \geq W_{\max,t} \)

**Observation 6.** \( \sum_{q \in [K]} W_{qt} \cdot z_{qt} \geq \frac{1}{K} \cdot \sum_q W_{qt} \)

The above observations follow by considering two feasible schedules of assigning a rate of 1 to the highest weight queue, and assigning a rate of \( \frac{1}{K} \) to each queue.

Since our algorithm computes an optimal feasible solution to \( \max_{z'} \sum_q W_{qt} \cdot z'_{qt} \), we immediately have the following optimality condition. Recall that \( z_{qt} \) is the rate our algorithm assigns to queue \( q \) at time \( t \).

**Proposition 87** (Optimality Condition). For any \( z'_t \in P_q, \sum_{q \in [K]} W_{qt} \cdot z'_{qt} \geq \sum_{q \in [K]} W_{qt} \cdot z_{qt} \)
Next, we will show that this algorithm, when given a speed of \((1 + \epsilon)\), is \(O(K^3 \epsilon^{-3})\)-competitive for the PSP-Q problem.

11.3.2 Analysis

Our analysis is based on a potential function argument. Towards defining a potential function we will set up some notation. Define \(W_{qt}^O\) as the total weight of jobs that are alive in the queue \(q\) in a fixed optimal schedule \(OPT\). Let \(p_jt, p_jt^O\) denote the job \(j\)'s remaining size in our algorithm’s schedule and the optimal schedule at time \(t\), respectively. Define a job \(j\)'s lag as \(\tilde{p}_{jt} := \max(p_{jt} - p_{jt}^O, 0)\). The quantity \(\tilde{p}_{jt}\) indicates how much our algorithm is behind the optimal schedule in terms of job \(j\)’s processing. Note that jobs always have non-negative lags. Furthermore, if a job is alive in the algorithm’s schedule and has zero lag then it implies that the job is also alive in the optimal schedule.

It is assumed w.l.o.g. that no two jobs arrive at the same time. For a job \(j \in A_{qt}\), define \(A_{qt, \leq j}\) as the set of jobs in \(A_{qt}\) that arrive no later than job \(j\). That is, \(A_{qt, \leq j} = \{j' \mid j' \in A_{qt} \text{ and } r_{j'} \leq r_j\}\). Let \(W_{qt, \leq j}\) denote the total weight of jobs in \(A_{qt, \leq j}\). In other words, \(W_{qt, \leq j}\) denotes the total weight of jobs in our algorithm’s queue \(q\) at time \(t\) that arrive no later than job \(j\).

We now define the potential function:

\[
\Phi(t) = \frac{K}{\epsilon} \cdot \sum_{q \in [K]} \sum_{j \in A_{qt}} W_{t, \leq j} \cdot \tilde{p}_{jt} \tag{11.1}
\]

Since jobs have non-negative lags, \(\Phi(t) \geq 0\) at all time instants \(t\). Let \(T\) be a sufficiently large time when all jobs are completed by our algorithm and \(OPT\). Clearly, \(\Phi(0) = \Phi(T) = 0\).

We first consider non-continuous changes of the potential.

**Lemma 88.** When a job arrives or is completed by our algorithm or \(OPT\), the potential does not increase.

**Proof.** First, observe that arrival of a new job \(j\) does not change the potential as \(\tilde{p}_{jt} = 0\)
for the job $j$, and for all other jobs $j' \in A_{qt}$ the quantity $W_{t, \leq j'}$ remains the same since $r_{j'} < r_j$. OPT completing a job has no effect on the potential. Finally, when our algorithm completes a job $j$, the potential can only decrease as $W_{t, \leq j'}$ decreases for each $r_{j'} > r_j$ and remains the same for each $r_{j'} < r_j$.

Therefore, we have $\int_{t=0}^{T} \frac{d}{dt} \Phi(t) dt \geq 0$. Our main goal is to show the following lemma that involves continuous changes of the potential.

**Lemma 89.** For all time instants $t$ in the execution of our algorithm where no jobs arrive or are completed by our algorithm or OPT, $\sum_{q \in [K]} W_{qt} + \frac{d}{dt} \Phi(t) \leq O\left(\frac{K^3}{\epsilon^3}\right) \cdot \sum_{q \in [K]} W_{qt}^O$.

Indeed, integrating the inequality in the lemma over the time period $[0, T]$ yields Theorem 86 since $\int_{t=0}^{T} \sum_{q} W_{qt} dt$ and $\int_{t=0}^{T} \sum_{q} W_{qt}^O dt$ are our algorithm’s total weighted flow time and OPT’s, respectively.

It now remains to prove Lemma 89. Consider a fixed time instant $t$ where no discontinuous changes occur. Let $A_{\epsilon qt}$ denote the set of jobs in $A_{qt}$ that have a positive lag; recall that $A_{qt}$ denotes the set of latest arriving jobs whose total weight is $\epsilon \cdot W_{qt}$. We consider two cases depending on the total weight of jobs in the set $A_{\epsilon qt}$.

**Case 1:** For all queues $q \in [K], \sum_{j \in A_{\epsilon qt}} w_j \geq \epsilon \cdot W_{qt} - \frac{\epsilon^2}{K} \cdot W_{\text{max},t}$.

Roughly speaking, in this scenario, our algorithm is behind OPT for almost all jobs in every queue $q$. We first consider the decrease of the potential due to our algorithm’s processing.
\[
\frac{d}{dt} \Phi(t) |_A = \frac{K}{\epsilon} \cdot \sum_{q \in [K]} \sum_{j \in A_{qt}} W_{t \leq j} \cdot \frac{d}{dt} \tilde{p}_{jt} |_A
\]

\[
= -\frac{K}{\epsilon} \cdot \sum_{q \in [K]} \sum_{j \in \tilde{A}_{qt}} W_{t \leq j} \cdot (1 + 5\epsilon) \cdot z_{qt} \cdot \frac{w_j}{\epsilon \cdot W_{qt}} \quad \text{[def.WLAPS and speed aug.]} 
\]

\[
\leq -\frac{K}{\epsilon} \cdot \sum_{q \in [K]} \sum_{j \in \tilde{A}_{qt}} (1 - \epsilon) \cdot W_{qt} \cdot (1 + 5\epsilon) \cdot z_{qt} \cdot \frac{w_j}{\epsilon \cdot W_{qt}} 
\]

(11.2)

\[
\leq -\frac{K}{\epsilon^2} \cdot (1 + 3\epsilon) \cdot \sum_{q \in [K]} z_{qt} \cdot \sum_{j \in \tilde{A}_{qt}} w_j 
\]

The inequality 11.2 follows from the fact \( W_{t \leq j} \geq (1 - \epsilon)W_{qt}, \forall j \in \tilde{A}_{qt} \). We now use the condition of Case (1), as well as Observation 5 to complete the proof. Note that this part crucially requires the normalization step.

\[
\frac{d}{dt} \Phi(t) |_A \leq -\frac{K}{\epsilon^2} \cdot (1 + 3\epsilon) \cdot \sum_{q \in [K]} z_{qt} \cdot (\epsilon \cdot W_{qt} - \frac{\epsilon^2}{K} \cdot W_{max,t}) \quad \text{[Condition of Case 1]} 
\]

\[
\leq -\frac{K}{\epsilon} \cdot (1 + 3\epsilon) \cdot \left( \sum_{q \in [K]} z_{qt} \cdot W_{qt} - \epsilon \cdot W_{max,t} \right) \quad \text{[z_{qt} \leq 1 for all q \in [K]]} 
\]

\[
\leq -\frac{K}{\epsilon} \cdot (1 + 3\epsilon) \cdot \left( \sum_{q \in [K]} z_{qt} \cdot W_{qt} - \epsilon \cdot \sum_{q \in [K]} z_{qt} \cdot W_{qt} \right) \quad \text{[Observation 5]} 
\]

\[
\leq -\frac{K}{\epsilon} \cdot (1 + \epsilon) \cdot \sum_{q \in [K]} z_{qt} \cdot W_{qt} \quad \text{(11.3)} 
\]

Next, consider the increase of the potential due to OPT’s processing. Let \( z_{qt}^O \) denote the rate at which OPT processes \( q \). We note that the last inequality is where the optimality
condition of Proposition 87 plays a crucial role.

\[
\frac{d}{dt} \Phi(t)|_O = \frac{K}{\epsilon} \sum_{q \in [K]} \sum_{j \in A_{qt}} W_{t \leq j} \cdot \frac{d}{dt} \tilde{p}_{jt}|_O
\]

\[
\leq \frac{K}{\epsilon} \sum_{q \in [K]} \sum_{j \in A_{qt}} W_{qt} \cdot \frac{d}{dt} \tilde{p}_{jt}|_O \geq \sum_{j \in [A]} W_{qt} \cdot z_{qt} \quad \text{[Proposition 87]} \quad (11.4)
\]

Therefore, from the equations (11.3) and (11.4), the total decrease in the potential is at least

\[
\frac{d}{dt} \Phi(t) = \frac{d}{dt} \Phi(t)|_O + \frac{d}{dt} \Phi(t)|_A \leq -\frac{K}{\epsilon} \cdot (1 + \epsilon) \cdot \sum_{q \in [K]} z_{qt} \cdot W_{qt} + \frac{K}{\epsilon} \cdot \sum_{q \in [K]} z_{qt} \cdot W_{qt}
\]

\[
\leq -K \cdot \sum_{q \in [K]} z_{qt} \cdot W_{qt} \leq -K \cdot \sum_{q \in [K]} \frac{1}{K} \cdot W_{qt} \quad \text{[Observation 6]}
\]

\[
\leq -\sum_{q \in [K]} W_{qt}
\]

Therefore, we have \( \sum_{q \in [K]} W_{qt} + \frac{d}{dt} \Phi(t) \leq 0 \), proving Lemma 89 for the first case.

**Case 2:** For some \( q \in [K] \), \( \sum_{j \in \bar{A}_{qt}} w_j \leq \epsilon \cdot W_{qt} - \frac{\epsilon^2}{K} \cdot W_{max,t} \)

In this scenario, the total weight of jobs in OPT’s queue is comparable to that of our algorithm. So, we charge the cost incurred by our algorithm to OPT directly. For this case, we ignore the change of the potential due to our algorithm’s processing as it is always at most 0. Let \( q’ \) be a queue for which the above inequality in the condition holds. Rearranging terms, we have

\[
\frac{\epsilon^2}{K} \cdot W_{max,t} \leq \epsilon \cdot W_{qt} - \sum_{j \in \bar{A}_{qt}} w_j = \sum_{j \in \bar{A}_{qt}} w_j - \sum_{j \in \bar{A}_{qt}'} w_j \leq W_{qt}'
\]

since all jobs in \( A_{qt} \setminus \bar{A}_{qt} \) are alive in the optimal schedule’s queue \( q \); note that \( j \in A_{qt} \setminus \bar{A}_{qt} \) implies the algorithm is ahead of OPT in terms of job \( j \)’s processing but still
hasn’t completed the job. With this observation in mind, we derive

$$\sum_{q \in [K]} W_{qt} + \frac{d}{dt} \Phi(t)$$

$$\leq \sum_{q \in [K]} W_{qt} + \frac{K}{\epsilon} \cdot \sum_{q \in [K]} W_{qt} \cdot z_{qt} \quad \text{[Equation (11.4)]}$$

$$\leq \sum_{q \in [K]} W_{qt} + \frac{K}{\epsilon} \cdot \sum_{q \in [K]} W_{qt} \quad \text{[}z_{qt} \leq 1 \text{ for all } q \in [K]\text{]}$$

$$\leq \frac{K + 1}{\epsilon} \cdot K \cdot W_{\text{max},t} \quad \text{[}W_{qt} \leq W_{\text{max},t} \text{ for all } q \in [K]\text{]}$$

$$= O\left(\frac{K^3}{\epsilon^2}\right) \cdot \sum_{q \in [K]} W_{qt}^O$$

This completes the proof of Lemma 89, and gives us Theorem 86.

11.4 Another Algorithm for PSP-Q

In this section, we consider the algorithm NORMALIZED MAX-WEIGHT combined with HIGHEST DENSITY FIRST (HDF). This algorithm assigns rates $z_{qt}$ to queues giving each queue $q$ a weight that is equal to the total fractional weight of jobs in the queue $q$ while fully devoting the rate $z_{qt}$ to the highest density job in the queue. Our main goal is to show the following theorem.

**Theorem 90.** For the PSP-Q problem with $K$ queues, for any constant $\epsilon > 0$, the algorithm NORMALIZED MAX-WEIGHT+HIGHEST DENSITY FIRST with speed $(1+\epsilon)$ is $O(K/\epsilon)$ competitive for fractional weighted flow time. Therefore, there is an algorithm that is $(1+\epsilon)$ is $O(K/\epsilon^2)$ competitive for (integral) weighted flow time.

Fractional weighted flow time is a relaxation of its integral counterpart, letting a job $j$ only incur cost $w_j p_{jt}/p_j$ at time instant $t$ as opposed to $w_j$ incurred in the integral objective while $j$ is alive. Hence the fractional weighted flow lower bounds the integral weighted flow. Although the integral weighted flow can be significantly greater than the fractional weighted flow, it is known that an algorithm that is $c$-competitive for the fractional objective can
be converted online into an algorithm that is \( O(c/\epsilon) \)-competitive with an extra speed augmentation of \((1 + \epsilon)\). For example, see (Im et al., 2011a). Henceforth we will focus on proving the first part of Theorem 90.

### 11.4.1 Algorithm Description

For simplicity, assume w.l.o.g. that all jobs have size 1. This can be done following the standard conversion from arbitrary jobs sizes to unit jobs sizes: each job \( j \) is replaced with \( p_j \) jobs with weight \( w_j/p_j \). It is well known that the fractional objective remains the same after this conversion. Then, in this simplified instance, processing the highest density job implies processing the highest weight job. For notational convenience, we will assume that in the given instance, jobs have unit sizes, i.e. \( p_j = 1 \) for all \( j \) from the beginning. We assume w.l.o.g. that all jobs have distinct weights. Also as in the previous section, we assume w.l.o.g. that \( g_q = 1 \) for all \( q \in [K] \).

Let \( J_{qt} \) denote the jobs that have arrived into queue \( q \) by time \( t \), and \( p_{jt} \) job \( j \)'s remaining size at time \( t \) in the algorithm’s schedule. The **Normalized Max-Weight** algorithm calculates a feasible set of rates \( z_{qt} \) for each queue at each time instant \( t \) by solving the following mathematical program.

\[
\max \sum_{q \in [K]} \left( \sum_{j \in J_{qt}} w_j p_{jt} \right) \cdot z_{qt} \\
\text{s.t.} \quad z_t \in P_q
\]

We reserve \( z_{qt} \) throughout this section to denote the rate our algorithm assigns to queue \( q \) at time \( t \). Note that \( \sum_{j \in J_{qt}} w_j p_{jt} \) is the total fractional weight of jobs alive in the algorithm’s queue \( q \) since if \( j \in J_{qt} \setminus A_{qt} \), then \( p_{jt} = 0 \), so \( \sum_{j \in J_{qt}} w_j p_{jt} = \sum_{j \in A_{qt}} w_j p_{jt} \). Then, the algorithm processes the highest weight job \( 1_{qt} \) that is alive in each queue \( q \) at a rate of \( z_{qt} \), i.e. \( \frac{d}{dt} p_{1_{qt}} = -z_{qt} \).

Similar to the analysis in the previous section, we make the following observations. Let \( \hat{W}_{qt} = \sum_{j \in J_{qt}} w_j p_{jt} \) and \( \hat{W}^O_{qt} = \sum_{j \in J_{qt}} w_j p^O_{jt} \).

**Observation 7.** \( \sum_{q \in [K]} \hat{W}_{qt} \cdot z_{qt} \geq \frac{1}{K} \cdot \sum_{q \in [K]} \hat{W}_{qt} \)
**Proposition 91.** For any $z'_t \in {\mathcal P}_q$, $\sum_{q \in [K]} \hat{W}_{qt} \cdot z_{qt} \geq \sum_{q \in [K]} \hat{W}_{qt} \cdot z'_{qt}$

### 11.4.2 Analysis

Our analysis will be based on a potential function argument which is inspired by (Chadha et al., 2009). The potential is defined as,

$$\Phi(t) := \sum_{q \in [K]} \hat{C}_q(t)$$

where

$$\hat{C}_q(t) := \sum_{j \in J_{qt}} w_j p_{jt} \left( \sum_{j' \in J_{qt}, w_{j'} \geq w_j} p_{j't} - \sum_{j' \in J_{qt}, w_{j'} \geq w_j} p^O_{j't} \right) - \sum_{j \in J_{qt}} w_j p^O_{jt} \left( \sum_{j' \in J_{qt}, w_{j'} \geq w_j} p_{j't} \right)$$

We will proceed our analysis by taking a close look at how the potential changes over time. We first consider discontinuous changes. Say a new job $i$ arrives into queue $q$. So far jobs $J_{qt}$ and $i$ have arrived into queue $q$. For each $j \in J_{qt}$, the quantity in the first summation does not change since $p_{it} = p^O_{it}$ and either $p_{it} - p^O_{it}$ or nothing is added to the quantity in the first parenthesis. Also job $i$ appear in the first summation which can increase the potential by at most $w_i p_{it} \sum_{j' \in J_{qt}, w_{j'} \geq w_i} p_{j't}$. However, it is offset by $w_i p^O_{it} \sum_{j' \in J_{qt}, w_{j'} \geq w_i} p_{j't}$. Hence jobs arrival can only decrease the potential. Jobs completion either in the algorithm’s schedule or the optimal schedule has no effect on the potential. Hence the potential can only decrease when discontinuous changes occur.

We now focus on continuous changes of $\Phi(t)$ which occur when no jobs arrive or complete. As before, let $\frac{d}{dt}(\cdot)|_A$ denote the derivative of the quantity in the parenthesis due to $A$’s processing freezing OPT’s processing. We define $\frac{d}{dt}(\cdot)|_O$ similarly. So $\frac{d}{dt} \Phi(t) = \frac{d}{dt} \Phi(t)|_A + \frac{d}{dt} \Phi(t)|_O$. Knowing that the potential is 0 both at time 0 and at a time $T$ when all jobs have been completed by our algorithm and OPT, and no discontinuous changes increase the potential, we have $\int_{t=0}^{T} \frac{d}{dt} \Phi(t) dt \geq 0$. Our main goal is to show,

$$\int_{t=0}^{T} \frac{d}{dt} \Phi(t) dt \leq -\frac{\epsilon}{K} \int_{t=0}^{T} \sum_{q \in [K]} \hat{W}_{qt} dt + O(1) \int_{t=0}^{T} \sum_{q \in [K]} \hat{W}^O_{qt} dt$$

(11.5)
where \( \hat{W}_{qt} := \sum_{j \in J_{qt}} w_j p_{jt} \) and \( \hat{W}^O_{qt} := \sum_{j \in J_{qt}} w_j p^O_{jt} \). This, combined with the fact that \( \int_{t=0}^{T} \frac{d}{dt} \Phi(t) dt \geq 0 \), will immediately yield the first part of Theorem 90.

It now remains to show Equation (11.5). Assuming that the algorithm is given \((1 + \epsilon)\)-speed, it processes only the highest weight job \( 1_q \), at a rate of \((1 + \epsilon) z_{qt} \); here we omitted \( t \) from \( 1_q \) for notational simplicity. Considering each queue \( q \), we derive,

\[
\frac{d}{dt} \bar{C}_q(t) \mid_A \leq \left( \sum_{j \in J_{qt}} w_j p_{jt} \right) \cdot (-(1 + \epsilon) z_{qt}) + w_{1_q} (1 + \epsilon) z_{qt} \cdot \left( \sum_{j' \in J_{qt}, w_{j'} \geq w_{1_q}} p^O_{jt} \right)
\]

\[
+ \left( \sum_{j' \in J_{qt}, w_j \leq w_{1_q}} w_{j'} p^O_{jt} \right) (1 + \epsilon) z_{qt}
\]

\[
\leq -(1 + \epsilon) z_{qt} \cdot \hat{W}_{qt} + 2(1 + \epsilon) z_{qt} \cdot \hat{W}^O_{qt}
\]

Say OPT processes job \( i \) in queue \( q \) at a rate of \( z^O_{qt} \).

\[
\frac{d}{dt} \bar{C}_q(t) \mid_O \leq \left( \sum_{j \in J_{qt}, w_j \leq w_i} w_j p_{jt} \right) z^O_{qt} + w_i z^O_{qt} \left( \sum_{j' \in J_{qt}, w_{j'} \geq w_i} p^O_{jt} \right)
\]

\[
\leq z^O_{qt} \cdot \hat{W}_{qt} + z^O_{qt} \cdot w_i p_{it}
\]

For a while, we proceed our analysis ignoring the term \( z^O_{qt} \cdot w_i p_{it} \) – we will bring it in the picture at the end of the analysis. Then, we have,

\[
\frac{d}{dt} \Phi(t) = \frac{d}{dt} \Phi(t) \mid_A + \frac{d}{dt} \Phi(t) \mid_O
\]

\[
\leq -(1 + \epsilon) \sum_{q \in [K]} z_{qt} \cdot \hat{W}_{qt} + \sum_{q \in [K]} z^O_{qt} \cdot \hat{W}_{qt} + 2(1 + \epsilon) \sum_{q \in [K]} z_{qt} \cdot \hat{W}^O_{qt}
\]

\[
\leq -\epsilon \sum_{q \in [K]} z_{qt} \cdot \hat{W}_{qt} + 2(1 + \epsilon) \sum_{q \in [K]} z^O_{qt} \cdot \hat{W}^O_{qt} \quad \text{[Proposition 91]}
\]

\[
\leq -\epsilon \frac{1}{K} \sum_{q \in [K]} \hat{W}_{qt} + 2(1 + \epsilon) \sum_{q \in [K]} \hat{W}^O_{qt} \quad \text{[Observation 7 and } z_{qt} \leq 1]\]

Integrating this inequality gives Equation (11.5).
We now bring in the term \( z_{qt} \cdot w_i p_{it} \) we ignored. Note that we need to add this term to \( \frac{d}{dt} \Phi(t) \) only when the optimal scheduler processes job \( i \) at time \( t \). What this means is that OPT processing job \( i \) by \( \delta \) units contributes to \( \int_{t=0}^{T} \frac{d}{dt} \Phi(t) \) by at most \( \delta w_i \); recall that \( p_{it} \leq 1 \). Hence each job \( i \) can add at most \( w_i \) to \( \int_{t=0}^{T} \frac{d}{dt} \Phi(t) \) when the ignored term is considered. Observe that in OPT with 1-speed, each job \( i \)'s weighted fractional flow time is no smaller than \( w_i / 2 \); here we used the fact \( z_{qt} \leq 1 \). Hence the ignored term can only increase \( \int_{t=0}^{T} \frac{d}{dt} \Phi(t) dt \) by at most twice OPT’s fractional weighted flow time. This shows Equation (11.5), completing the proof of Theorem 90.

11.5 Resource Allocation with Complementarities (RA-C)

Recall the RA-C problem from Section 11.2. There are \( D \) divisible resources (or dimensions), numbered 1, 2, \ldots, \( D \). We assume w.l.o.g. (by scaling and splitting resources) that each resource is available in unit supply. If job \( j \) is assigned a non-negative vector of resources \( x = \{x_1, x_2, \ldots, x_D\} \), then the rate at which the job executes is given by \( y_j = u_j(x) \) where \( u_j(x_j) = \min_{d=1}^{D} (c_{jd} x_{jd}) \) are Leontief utilities. Here, \( c_j \) is the resource vector of the job.

When the number of distinct resource vectors (or job types) is bounded by \( K \), then this problem is clearly a special case of PSP-Q with \( K \) queues. In this section, we show that even when the resource vectors can be arbitrary, the RA-C problem can be reduced to PSP-Q with a bounded number of queues. In the reduction, jobs of “similar” resource vectors will be discretized to the same type.

**Lemma 92.** If we use a \((1 + \epsilon)\) factor more speed, then we can w.l.o.g. assume that for all jobs \( j \): (1) \( \min_{d=1}^{D} c_{jd} = 1 \); and (2) \( c_{jd} \in [1, D/\epsilon] \cup \{\infty\} \) for all \( d \in [D] \).

Here if \( c_{jd} = \infty \), \( c_{jd} x_{jd} \) is defined to be \( \infty \) for any value of \( x_{jd} \) (meaning that \( d \) is never a bottleneck in determining \( j \)'s utility). More precisely, if we have a \( s \)-speed \( f \)-competitive algorithm for this simplified instance, then we can derive an algorithm that is \( s(1 + \epsilon) \)-speed \( f \)-competitive for the original instance.
Proof. Fix a job $j$. Let $h = \min_{d=1}^{D} c_{jd}$. We first normalize $c_j$ so that the first property is satisfied. Towards this end, we scale down $c_j$ and $p_j$ by the same factor $h$. Note that these changes do not affect the schedule. More precisely, any feasible schedule $x_{j,t}$ for the original instance is also feasible for the new modified instance, and further all jobs completion times remain the same. So far we have shown the first property can be satisfied w.l.o.g. Henceforth, for notational convenience, let’s assume that the original instance of jobs $j$ with $c_{jd}$ satisfies the first property.

Now turning to the second property, if $c_{jd} \geq D/\epsilon$, then we simply set $c_{jd}$ to $\infty$. Let $c_j'$ denote the new vector for job $j$. To see why we can make this change w.l.o.g., consider any feasible resource allocation $x_{jd}'$ at a fixed time $t$ for this new instance. Here, we can assume w.l.o.g. that $x_{jd}' = 0$ if $c_{jd}' = \infty$, and $c_{jd}' x_{jd}'$ is the same for all $d$ with $c_{jd}' < \infty$ since otherwise resources would be wasted. We would like to construct $\{x_{jd}\}$ such that each job gets processed at the same rate under the schedule $\{x_{jd}\}$ for the original instance as it does under the schedule $\{x_{jd}'\}$ for the modified instance. Towards this end, for any $d$ with $c_{jd}' = \infty$ and $c_{jd} < \infty$, i.e. $D/\epsilon < c_{jd} < \infty$, we set $x_{jd}$ so that $c_{jd} x_{jd}$ are the same for all $d$ with $c_{jd} < \infty$. We say that a job $j'$ is tightest for dimension $d'$ if $c_{jd'} = 1$. The first property implies that every job is tightest for some dimensions; a job can be tightest for multiple dimensions. Let $J_{d'}$ denote the jobs that are the tightest for dimension $d'$.

The issue is that the resource allocation $\{x_{jd}\}$ may over-allocate resources. However, we will show that over-allocation can be fixed by a small amount of speed augmentation. We now show that $\sum_j (x_{jd} - x_{jd}') \leq \epsilon$ for all $d \in [D]$. This will imply that $\sum_j x_{jd} \leq \epsilon + \sum_j x_{jd}' \leq 1 + \epsilon$. Hence if we use a resource allocation $x_{jd}/(1 + \epsilon)$ with a $(1 + \epsilon)$ factor more speed, we will have a feasible schedule while processing each job’s rate. Consider any fixed $d^*$. Since $x_{jd^*} - x_{jd^*}' > 0$ only if $D/\epsilon \leq c_{jd^*} < \infty$, we have,
\[ \sum_j (x_{jd} - x'_{jd}) \leq \sum_{d' \in [D]} \sum_{j \in J_{d'}, \epsilon \leq c_{jd} < \infty} (x_{jd} - x'_{jd}) \]

\[ = \sum_{d' \in [D]} \sum_{j \in J_{d'}, \epsilon \leq c_{jd} < \infty} x_{jd} \quad [x'_{jd} = 0 \text{ if } D/\epsilon \leq c_{jd}] \]

\[ = \sum_{d' \in [D]} \sum_{j \in J_{d'}, \epsilon \leq c_{jd} < \infty} (1/c_{jd}) x_{jd} \quad [c_{jd} x_{jd} = c_{jd'} x_{jd'} = x_{jd}] \]

\[ \leq \sum_{d' \in [D]} \sum_{j \in J_{d'}} (\epsilon/D) x_{jd'} \leq (\epsilon/D) \sum_{d' \in [D]} 1 \leq \epsilon \]

We are now ready to describe the reduction. Assume that the given resource vectors satisfy the properties stated in the above lemma. We discretize each job’s resource vectors as follows. For each \( d \in [D] \), if \( c_{jd} < \infty \), then round up \( c_{jd} \) to the closest power of \((1 + \epsilon)\). We know that \( x_{jd} \) can have at most \( \lceil \log_{(1+\epsilon)} (D/\epsilon) \rceil + 2 \leq O((1/\epsilon) \log D) \) different values. Hence there are at most \( (O(1/\epsilon) \log D)^D \) distinct resource vectors. Jobs of the same resource vector are placed into the same queue. It is easy to see using another \((1 + \epsilon)\) factor more speed takes care of the rate loss due to the discretization. Hence we have shown a reduction of arbitrary RA-C instances with \( D \) dimensions to PSP-Q with \( K = ((2/\epsilon) \log D)^D \) queues by using \((1 + \epsilon)\) factor more speed.

11.6 Summary and Open Problems

In this chapter we identified a fairly general subclass of the PSP problem for which we can design better competitive algorithms. However, several challenging questions remain open:

Are there are any other classes of PSP problem for which we can circumvent the lower-bound of \( \Omega(\sqrt{\log n}) \) speed? Can we design \( O(1) \)-speed \( O(D) \)-competitive algorithms for multidimensional scheduling problem? Note that in this chapter we showed a clairvoyant algorithm that is \( O(1 + \epsilon) \)-speed \( O((\log D)^D) \)-competitive algorithm, but we do not have any evidence that this cannot be improved.
11.7 Notes

This chapter is based on joint work with Sungjin Im and Kamesh Munagala, and is under a conference submission (Sungjin Im and Munagala, 2015).
PART IV
Selfish Scheduling
12

Coordination Mechanisms For Selfish Scheduling

12.1 Introduction

Explosive growth of data has driven distributed computing to evolve at an unprecedented pace. In modern distributed systems, there are a large number of machines which are clustered and connected in a variety of topologies, and situated across different geographical locations. Hence routing jobs can incur considerable costs and communication delays. Machines are also inherently heterogeneous, having very different architectures and accesses to energy resources – some machines can process some jobs more efficiently and at cheaper costs. Due to the large scale of such systems, centralized algorithms for scheduling jobs are not very practical. Moreover, in many scenarios each job is a selfish agent that strategically selects a machine for getting processed. Can such a decentralized system perform well in spite of the strategic behaviors of the jobs? In next few chapters, we explore these questions under a mechanism design paradigm called coordination mechanisms (Christodoulou et al., 2009).
12.1.1 Model

There is a set $\mathcal{J}$ of $n$ jobs, and a set $\mathcal{M}$ of $m$ unrelated machines. A job $j$ has a weight $w_j$, and it needs $p_{ij}$ units of processing time if scheduled on machine $i$. The job $j$ has a communication delay (or release date) $r_{ij}$ on machine $i$, i.e., the machine can start processing the job only after time $r_{ij}$. Further, the job $j$ incurs an assignment cost $h_{ij}$ if it is assigned to the machine $i$. This, for example, captures the privacy concerns or energy costs.

In a sharp contrast with the centralized view of classical scheduling models, here each job is a self-interested and autonomous agent free to select its own machine. Every machine declares its scheduling policy in advance, and this induces a simultaneous-move game between the jobs. The strategy of a job consists of choosing the machine where it will get processed. Each job wants to minimize its own disutility, which is its weighted completion time plus its assignment cost. A Nash equilibrium of this game is a stable outcome where no job can reduce its disutility by switching to another machine. The strategic interactions among the jobs may lead to overall degradation in system performance. The standard benchmark to measure this deterioration is the Price of Anarchy (PoA), first introduced in (Koutsoupias and Papadimitriou, 1999). This is the worst case (maximum possible) ratio of total disutility of the jobs in a Nash Equilibrium to that in an optimal solution (which assumes centralized assignment and no strategic behavior).

We let different machines declare different scheduling policies. Each of these policies, however, must be strongly local since a machine $i$ only knows the $w_j$, $p_{ij}$, $r_{ij}$, and $h_{ij}$ values of the jobs $j$ that were assigned to it. In the absence of a global view of the input, a reasonable option for a machine is to declare a scheduling policy that (approximately) minimizes its own share of the objective, namely, the total weighted completion time of all the jobs assigned to it. This raises a compelling question:

---

1 Our analysis can be easily extended to unrelated weights $w_{ij}$.

2 If a job $j$ is assigned to a machine $i$, then the machine $i$ is not aware of the characteristics of the job $j$ (processing lengths, release dates etc.) on other machines $i' \neq i$.

3 The machine can ignore the assignment costs as their contribution to the objective is fixed once each job selects its strategy.
• Do all scheduling policies that are $O(1)$-approximate on a single machine result in coordination mechanisms with $O(1)$ price of anarchy? If the answer is no, is there a characterization of single-machine scheduling policies that induce $O(1)$ price of anarchy?

12.1.2 Our Results

In this chapter, we present a general recipe for designing coordination mechanisms for minimizing the total weighted completion time of the jobs plus their assignment costs. The machines need not agree upon a specific scheduling policy. Nevertheless, the system will have small constant price of anarchy as long as every machine selects a scheduling policy that satisfies a certain bounded stretch condition introduced in this chapter (see Sections 12.1.2, 12.1.2). We further show that almost all scheduling policies used in practice satisfy this condition. We complement this positive result by showing that there exists a widely used $O(1)$-approximate single-machine scheduling policy that does not satisfy the bounded stretch condition, and induces a game with large price of anarchy.

All the previous works on coordination mechanisms for completion time scheduling (Cole et al., 2011; Cohen et al., 2012) focused on the case without communication delays (release dates) and assignment costs, i.e., $r_{ij} = h_{ij} = 0$. We note that release dates introduce a significant complexity in scheduling, and algorithms for problems without release dates typically do not generalize to those with release dates. For example, the underlying optimization problem on a single machine is polynomial time solvable via a greedy algorithm without release dates, but is NP-HARD with release dates (Skutella and Woeginger, 1999). Furthermore, the previous works (Cole et al., 2011; Cohen et al., 2012) analyze very specific scheduling policies, and restrict all machines to announce the same scheduling policy. In contrast, we give a new dimension to the problem by allowing different machines to declare different scheduling policies. This generalization models the real world applications more accurately, since the machines (or data centers) are typically owned and operated by different entities.
Our results rely upon two novel techniques. (a) A potential function that leads to an instantaneous \textit{smoothness} condition; and (b) Linear programming and dual fitting.

\textit{Characterization of Good Single-machine Scheduling Policies}

We introduce the notion of a scheduling policy with \textit{bounded stretch} in the definition given below.\footnote{Our notion of stretch is different from the standard definition of stretch used in scheduling literature.}

\textbf{Definition 12.1.1.} Suppose that a machine is processing a given set of jobs. The scheduling policy followed by the machine has a stretch \( \alpha \) iff the completion time \( C_j \) of each job \( j \) (with weight \( w_j \), release date \( r_j \), and processing time \( p_j \)) satisfies the inequality:

\[
C_j \leq r_j + p_j + \sum_{j' \neq j : C_{j'} \geq r_j} \alpha \cdot \min(p_{j'}, (w_{j'}/w_j) \cdot p_j)
\]

To understand the above definition, assume for a while that all the jobs have unit weight, and note that the total \textit{delay} encountered by a job \( j \) is equal to \( C_j - (r_j + p_j) \). What should be a reasonable upper bound on the contribution (say \( \eta_{j'} \)) of some specific job \( j' \neq j \) towards this delay \( C_j - (r_j + p_j) \)? Without any loss of generality, we can assume that the job \( j' \) completes after the release date of the job \( j \), i.e. \( C_{j'} \geq r_j \); otherwise \( \eta_{j'} \) is zero. The \( \alpha \)-stretch condition says that \( \eta_{j'} \) can be at most \( \alpha \) times \( \min(p_{j'}, p_{j'}) \). Note that the job \( j' \) can delay the job \( j \) when it gets processed with a higher priority, and this can happen for an amount of time equal to the job \( j' \)'s size, which is \( p_{j'} \). Further, the \( \alpha \)-stretch condition requires that the policy is fair to both the jobs and hence the bound \( \alpha \cdot \min(p_j, p_{j'}) \). Finally, the weights of the jobs are also factored in.

This seemingly simple characterization turns out to be quite powerful as many commonly used scheduling have small stretch.

\textbf{Theorem 93.} The scheduling policies – Highest Density First, Highest Residual Density First, Weighted Round Robin, and Weighted Shortest Elapsed Time First – all have stretch \( \alpha = 1 \).
The reader may find it helpful to compare Definition 12.1.1 with the definition of the Weighted Round Robin (WRR) scheduling policy. An important distinction between the two definitions is that the stretch condition does not specify a scheduling policy but is only concerned with the final completion times of the jobs. On the other hand, the scheduling policies with bounded stretch behave similar to WRR, in the sense that each job delays another job by at most $\alpha$ times its own processing length. This provides an intuitive explanation as to why such policies should lead to equilibria with small PoA.

**Price of Anarchy Bounds for Coordination Mechanisms**

With the above definition of bounded stretch, we prove the following general result.

**Theorem 94.** Suppose that each machine declares a (possibly different) scheduling policy with stretch $\alpha$. Then the resulting game has a robust (smooth) price of anarchy of at most

$$\frac{1 + \alpha(\sqrt{\alpha^2 + 1} + \alpha)}{1 - \alpha(\sqrt{\alpha^2 + 1} - \alpha)} \leq 4\alpha^2 + 2\alpha = O(\alpha^2)$$

Particularly when $\alpha = 1$, the bound is at most 5.8284, and this holds for many popular scheduling policies.

The interesting aspect of the above result is the analysis. We present two analysis techniques which yield somewhat different bounds - potential functions and dual fitting. Conceptually, our techniques are inspired by the elegant work on online scheduling (Chadha et al., 2009; Anand et al., 2012; Im et al., 2011a) – the connection being that in both cases, we need to compare the decisions made by the optimal solution (that is non-strategic and omniscient) with the solution that arises due to the execution of the implemented policy. However, the similarity ends there - unlike online algorithms, in a coordination game, a job is selfish and cannot be forced to go to a specific machine and hence equilibrium state cannot be controlled by an algorithm. Moreover, a game can have multiple equilibria, and the analysis should hold for all them simultaneously.
Potential Functions. Our first technique uses a smoothness argument (see Section 12.3.1) via a carefully constructed potential function (see Section 12.4). The difficulty in a direct smoothness argument is that we have to compare the execution of two policies that make decisions over time, and these decisions could be very divergent. Note that unlike the case without release dates (Cole et al., 2011), we cannot write closed form expressions for the completion time induced by specific policies. Instead, we show that the derivative of the potential function (w.r.t. time) gives an instantaneous smoothness inequality that is easy to compute. We then integrate this inequality over time to derive the final smoothness bound. To our knowledge, this type of approach inspired by online algorithms has not been used before in the context of price of anarchy.

Dual Fitting. (See Section 12.5.) Consider the optimization problem underlying our game-theoretic framework, where the goal is to minimize the total weighted completion time of the jobs plus their assignment costs. We write a time-indexed LP-relaxation of the problem, similar to the one in (Anand et al., 2012). Using the dual of this LP, we bound the price of anarchy of the game induced between the jobs, when every machine declares a scheduling policy satisfying certain natural conditions (see Section 12.5.1). The idea is to take any Nash equilibrium of the game, and appropriately charge the disutility of each job to the dual variables in a way such that (a) all the dual constraints are satisfied, and (b) the dual objective is at least $\eta$ times the total disutility incurred by all the jobs in the Nash equilibrium, for some $\eta \in (0, 1]$. This shows that the price of anarchy is at most $1/\eta$ due to weak duality.

In contrast to a potential function based argument, one apparent drawback of the dual-fitting framework is that for technical reasons we need to impose two restrictions on the allowable class of scheduling policies (see Section 12.5.1) in addition to the bounded stretch condition. These restrictions, however, are fairly intuitive. We are not aware of any simple, combinatorial scheduling policy with bounded stretch that violates the additional assumptions required in the dual-fitting proof. Further, unlike the potential
function analysis, the dual-fitting proof only bounds the price of anarchy of pure, mixed Nash and correlated equilibria, and at present we do not see any way to extend the proof to get a robust (smooth) price of anarchy bound which, in addition to these three solution concepts, also applies to no regret sequences (Roughgarden, 2009).

Nevertheless, we feel the dual-fitting framework is interesting in its own right, as it establishes a connection between the price of anarchy of a coordination mechanism and the LP relaxation of the underlying optimization problem. We will see more applications of this technique in next two chapters. Further, it yields the following improved theorem:

**Theorem 95.** The price of anarchy (of pure, mixed, and correlated equilibria) of the Highest Density First scheduling policy is at most 4.

This matches the lower bound (Cole et al., 2011) known for non-preemptive scheduling policies when there are no release dates and assignment costs. The dual-fitting approach also helps us compare against an optimal migratory solution where a job is processed over multiple machines. To our knowledge, this type of approach to bound the price of anarchy of games has not been considered previously in the literature.

**Scheduling Policies with Large Stretch.** As our $\alpha$-stretch condition is fairly general and most of the popular scheduling policies have bounded stretch, the reader may be tempted to conjecture that all scheduling policies that are $O(1)$-approximation on a single machine lead to coordination mechanisms with $O(1)$ price of anarchy. However, we show in Section 12.6 that this is not the case. The scheduling policy Weighted Latest Arrival Processor Sharing (WLAPS) gives $O(1)$-approximation to the total weighted completion time on a single machine. But it does not induce a game with constant price of anarchy. The scheduling policy WLAPS generalizes Round Robin to favor more recent jobs, and has been extensively studied in scheduling theory – particularly in broadcast scheduling problems (Edmonds and Pruhs, 2009; Bansal et al., 2010). Not surprisingly, WLAPS fails our bounded stretch condition ($\alpha$ is $\Omega(n)$). Thus, our characterization seems to separate

---

5 Note that Highest Density First is a nonpreemptive policy in the absence of release dates.
scheduling policies which are good in non-strategic settings from those which lead to small price of anarchy.

Non-preemptive Scheduling Policies

In some applications, preempting a job can be costly, and non-preemptive scheduling policies are highly desirable. When the jobs arrive online, however, no natural non-preemptive scheduling policy gives $O(1)$ approximation to the weighted completion time, even on a single machine. This is particularly relevant since both our potential function and dual-fitting proofs are inspired by the frameworks developed for online scheduling problems (Chadha et al., 2009; Anand et al., 2012). Nevertheless, in section 12.7 we present a general black box reduction from preemptive scheduling policies to non-preemptive scheduling policies that preserve the stretch within a factor of two. This reduction is an adaptation of the idea used in (Hall et al., 1996).

**Theorem 96.** There exists a reduction that takes any preemptive scheduling policy with stretch $\alpha$ and outputs a non-preemptive scheduling policy with stretch at most $2\alpha$.

The above theorem, along with Theorem 93 and Theorem 94, shows how to construct non-preemptive (offline) scheduling policies that lead to small constant price of anarchy.

Pure Nash Equilibrium

Although a correlated equilibrium (which can be computed in polynomial time) and a mixed Nash equilibrium is guaranteed to exist in every finite game, not all games have pure Nash equilibrium (PNE). For example, when there are no communication delays (release dates) and assignment costs, it is known that the game induced by the Highest Density First policy might not have a PNE Cole et al. (2011). In contrast, a PNE is guaranteed to exist in the game induced by the Weighted Round Robin (WRR) policy. It is not clear if this property of WRR continues to hold in the presence of release dates and assignment costs. We address this concern by transforming the WRR policy. The idea is to run the WRR schedule but withhold the completion time of a job, forcing it to satisfy the following
condition.

\[ w_j C_j = \sum_{j'} \min(w_j p_{j'}, w_{j'} p_j) \]

One way to achieve this is to process the last \( \epsilon \) portion of job \( j \) at time \( t = (1/w_j) \cdot \sum_{j'} \min(w_j p_{j'}, w_{j'} p_j) \). It is easy to verify that the stretch of the resulting schedule is at most 2. The proof for WRR in (Cole et al., 2011) can be easily extended to show that this induces a potential game. Hence, a PNE is guaranteed to exist, and the Nash dynamics converges to a PNE in pseudo-polynomial time.

The above transformation may lead to unnecessary idle periods in the schedule produced. It will be interesting to find a natural scheduling policy which induces a game with PNE. Our generalized framework where each machine can declare a different scheduling policy raises another interesting question: Can a combination of different scheduling policies on different machines induce PNE that can be reached in polynomial steps? Note that here each component of a distributed system runs a different mechanism so as to force a better outcome. We leave it as an interesting open problem.

12.2 History

There has been a lot of work on approximation algorithms for minimizing the weighted sum of completion times (Hall et al., 1996; Schulz and Skutella, 2002; Skutella, 2001). Our potential function and dual-fitting techniques are inspired by the elegant framework developed for online scheduling in (Chadha et al., 2009; Anand et al., 2012). We note that their framework can be adapted to yield combinatorial scheduling policies for weighted completion time that are also \( O(1) \)-approximations. Here, each machine simply schedules the set of jobs assigned to it using, for example, the Highest Residual Density First policy. The algorithm considers the jobs in increasing order of their release dates and applies a greedy dispatch rule, assigning a job \( j \) to that machine \( i \) which increases the overall objective function (for the currently dispatched jobs) by the least amount. Although our potential function and dual-fitting proofs are inspired by this framework, as mentioned
above, the settings are actually very different. For instance, there is a scheduling policy (see Section 12.6) that gives $O(1)$-approximation to the online optimization problem when used in conjunction with the greedy dispatch rule, but induces a game with very large price of anarchy.

Coordination mechanisms were first introduced in (Christodoulou et al., 2009). See the survey (Immorlica et al., 2009) for various selfish scheduling models. The study of coordination mechanisms for completion time objective was initiated in (Cole et al., 2011; Correa and Queyranne, 2012). In the absence of release dates and assignment costs, they show tight constant factor price of anarchy bounds for three specific policies - Weighted Round Robin (WRR), Highest Density First (HDF), and Random (Random). They also show that, both WRR and RANDOM induce pure Nash Equilibrium while HDF does not. The $l_k$-norms of the completion time were considered (Cohen et al., 2012). They prove a price of anarchy of $O(k)$ for Shortest Job First (when there are no weights, release dates, and assignment costs), and show that no strongly local deterministic policy can achieve a price of anarchy better than $O(k/\log \log k)$.

Azar et al (Azar et al., 2008) design coordination mechanisms for the makespan objective. They show a lower bound of $\Omega(m)$ for any strongly local scheduling policy (see Section 12.1.1 for definition), where $m$ is the number of machines. In contrast, they present a weakly local scheduling policy (where a machine knows everything about the jobs assigned to it, including their processing lengths on other machines) that achieves a price of anarchy of $O(\log m)$, and a policy that induces a pure Nash Equilibrium with $\text{PoA}$ of $O(\log^2 m)$. These results were later extended by Caragiannis (Caragiannis, 2009). Similar to our work, he showed a strong connection between coordination mechanisms and online algorithms (Caragiannis, 2008). It will be interesting to study whether this is purely coincidental, or if there is a deeper connection between price of anarchy and competitive ratios.
12.3 Preliminaries

In this section, we introduce some concepts and notations that will be used throughout the rest of this chapter.

Recall the concepts and notations introduced in Section 12.1.1. We index a machine by \(i \in \mathcal{M}\), and a job by \(j \in \mathcal{J}\). Each machine \(i \in \mathcal{M}\) declares a strongly local scheduling policy \(A_i\). Let the symbol \(A = (A_1, \ldots, A_i, \ldots A_{|\mathcal{M}|})\) denote the profile of scheduling policies. Let \(\text{GAME}(A)\) denote the resulting game induced between the jobs. An outcome of this game is a strategy-profile \(\theta = (\theta_1, \ldots, \theta_j, \ldots, \theta_{|\mathcal{J}|})\), where \(\theta_j \in \mathcal{M}\) is the machine selected by the job \(j \in \mathcal{J}\). For notational convenience, we also define an assignment-vector \(Q\) that summarizes the outcome from the perspective of the machines. The vector \(Q = (Q_1, \ldots, Q_i, \ldots, Q_{|\mathcal{M}|})\) has \(|\mathcal{M}|\) components, and the \(i^{th}\) component of this vector refers to the set of jobs assigned to machine \(i \in \mathcal{M}\). Thus, we have \(Q_i = \{j \in \mathcal{J} : \theta_j = i\}\). The completion time of a job \(j\) under this outcome is given by \(C^A_j(\theta)\). The disutility of the job equals its assignment cost plus its weighted completion time, and this is denoted by \(\text{cost}^A_j(\theta) = h_{\theta_j,j} + w_j \cdot C^A_j(\theta)\). The outcome is a pure Nash equilibrium iff no job can reduce its disutility by switching to another machine, i.e., \(\text{cost}^A_j(\theta) \leq \text{cost}^A_j(i, \theta_{-j})\) for all \(j \in \mathcal{J}, i \in \mathcal{M}\), where \(\theta_{-j}\) is the strategy-profile of all the jobs except the job \(j\).

We reserve the term scenario for a triple \(S = (A, Q, \theta)\). This specifies the scheduling policy followed by every machine, and an outcome of the resulting game. The symbol \(p_{ij}(t)\) denotes the remaining processing length of a job \(j \in Q_i\) on machine \(i\) at time \(t\). We say that the job is unfinished at time \(t\) iff \(p_{ij}(t) > 0\). The symbol \(W_i(t)\) denotes the total weight of the unfinished jobs on machine \(i\) at time \(t\).

12.3.1 Smooth Games and Robust Price of Anarchy

Fix any profile of scheduling policies \(A\), and let \(\text{NE}(A)\) be the set of all pure Nash equilibria of \(\text{GAME}(A)\). The objective is to minimize the total disutility of the jobs. The price of
anarchy of this game is defined as:

$$\text{PoA}(A) = \frac{\max_{\theta \in \text{NE}(A)} \sum_{j} \text{cost}_{j}^{A}(\theta)}{\min_{A', \theta'} \sum_{j} \text{cost}_{j}^{A'}(\theta')}$$

Note that the above expression compares the worst Nash equilibrium of GAME(A) with the optimal solution to the underlying optimization problem, which may use an entirely different profile of scheduling policies.

The notion of price of anarchy as defined above is not applicable in games that do not admit any pure Nash equilibrium. To address this issue, Roughgarden (2009) introduced a smoothness framework which gives robust price of anarchy bounds for generalized solution concepts such as mixed Nash and correlated equilibria and no regret sequences. Adapting the smoothness framework to our context, we say that the GAME(A) is \((\lambda, \mu)\)-smooth iff the following inequality holds for any two scenarios \(S = (A, Q, \theta)\) and \(S' = (A', Q', \theta')\).

$$\sum_{j \in J} \text{cost}_{j}^{A}(\theta_{j}', \theta_{-j}) \leq \lambda \cdot \sum_{j \in J} \text{cost}_{j}^{A'}(\theta') + \mu \cdot \sum_{j \in J} \text{cost}_{j}^{A}(\theta)$$  \hspace{1cm} (12.1)

The reader may find it helpful to think of the scenario \(S\) as a pure Nash equilibrium of GAME(A), and the scenario \(S'\) as an optimal solution to the underlying optimization problem. It can be shown that the robust price of anarchy of any \((\lambda, \mu)\)-smooth game is at most \(\lambda/(1 - \mu)\).

12.4 Robust Price of Anarchy Bound via Potential Function

We devote this entire section to proving Theorem 94 by a potential function based argument. Throughout this section, we fix two scenarios \(S = (A, Q, \theta)\) and \(S' = (A', Q', \theta')\). Further, we assume that all the scheduling polices \(A_{i}\) (specified by \(A\)) under the scenario \(S\) has stretch \(\alpha\) (see Definition 12.1.1). We will prove Equation 12.1. The quantities \(\lambda\) and \(\mu\) will be decided later depending on \(\alpha\).
Zero Assignment Costs. For ease of analysis, we ignore the jobs’ assignment costs throughout this section, i.e. \( h_{ij} = 0 \). It is straightforward to extend our analysis to incorporate nonnegative assignment costs.\(^6\)

Note that the left hand side of Equation 12.1 results from mixing two completely different scenarios \( S \) and \( S' \). Hence, it is not easy to relate this quantity with the right hand side, which consists of the weighted completion times under the two individual scenarios. This is the case particularly when the jobs are released over time, as it becomes extremely difficult to derive mathematical expressions for the terms in Equation 12.1.

To circumvent this difficulty, we upper bound the left hand side by a carefully chosen potential function, and consider its derivative with respect to time. The advantage of this approach is that we have a good understanding of how an algorithm makes instantaneous scheduling decision. For example, the instantaneous increase in the total weighted completion time is simply the total weight of the unfinished jobs. In other words, each unfinished job \( j \) incurs a penalty of \( w_j \) at each time step.

For a technical reason that will become clear as we proceed with the proof, we slow down the schedule under \( S' \) by a (suitably chosen) constant factor \( \delta \in (0, 1) \). Let \( A'(\delta) \) denote the new profile of scheduling policies, and let \( S'(\delta) = (A'(\delta), Q', \theta') \) denote the new resulting scenario. More precisely, a job is processed on machine \( i \) at time \( t \) in \( S' \) iff it is processed on the same machine \( i \) at time \( t/\delta \) in \( S'(\delta) \). Note that the assignment vector and the strategy-profile remain the same across the two scenarios \( S' \) and \( S'(\delta) \).

We emphasize that \( S'(\delta) \) is not the schedule that results from the machines executing the policies \( A' \) with speed \( \delta \). Rather, \( S'(\delta) \) is obtained by stretching out the schedule \( S \) by a factor of \( 1/\delta \) over the time horizon. It is easy to see that this transformation increases the completion time of a job by a factor of \( 1/\delta \).

**Fact 12.4.1.** The completion time of every job \( j' \) under \( S'(\delta) \) is exactly \( 1/\delta \) times its completion time under \( S' \).

\(^6\) The dual fitting proof in Section 12.5 is presented in its full generality, and does not require this simplifying assumption.
Overview of our approach. Recall the notations introduced in Section 12.3. We will always index the jobs by $j$ under the scenario $S$, and by $j'$ under the scenario $S'(\delta)$. The remaining processing lengths will be denoted by $p_{ij}(t)$ under the scenario $S$, and by $p'_{ij}(t)$ under the scenario $S'(\delta)$. Similarly, the total weight of the unfinished jobs on a machine will be denoted by $W_i(t)$ under the scenario $S$, and by $W'_i(t)$ under the scenario $S'(\delta)$. With these notations in place, we are now ready to define our potential function.

\[
\Phi(t) = \sum_{i \in M} \sum_{j' \in Q_i} \sum_{j \in Q_i} \min \left( w_{j'} \cdot p_{ij}(t), w_j \cdot p'_{ij}(t) \right)
\]  

Let $\text{cost}^S = \sum_j \text{cost}_j^A(\theta)$ denote the total disutility of all the jobs under the scenario $S = (A, Q, \theta)$, which is the same as their total weighted completion time (assuming zero assignment costs). Hence, we can write this quantity as $\text{cost}^S = \int_0^\infty \frac{d}{dt} \text{cost}^S(t)$, where the derivative $\frac{d}{dt} \text{cost}^S(t)$ equals the total weight of the unfinished jobs at time $t$ under the scenario $S$. Similarly, let $\text{cost}^{S'}$ (resp. $\text{cost}^{S'\!(\delta)}$) denote the total disutility of all the jobs under the scenario $S'$ (resp. $S'(\delta)$). Fact 12.4.1 implies that $\text{cost}^{S'\!(\delta)} = (1/\delta) \cdot \text{cost}^{S'}$. We will show that $\Phi(t)$ is a good estimate of the left hand side of Equation 12.1. In particular, we will prove that $\Phi(t)$ satisfies the following conditions.

\[
\Phi(\infty) = 0 
\]  

\[
\sum_{i \in M} \sum_{j' \in Q_i} \text{cost}_j^A(i, \theta_{-j'}) \leq \text{cost}^{S'} + \alpha \cdot \Phi(0) 
\]  

\[
-\frac{d}{dt} \Phi(t) \leq \frac{d}{dt} \text{cost}^{S'\!(\delta)}(t) + \delta \cdot \frac{d}{dt} \text{cost}^S(t) \quad \text{at every time } t
\]

In Equation 12.4, the symbol $\alpha$ denotes the stretch of the scheduling policies declared by the machines (see Definition 12.1.1) under the scenario $S$. The proof of the next theorem appears in Section 12.4.1.

**Theorem 97.** The potential function as defined in Equation 12.2 satisfies Equations 12.3, 12.4, 12.5.
We now use Theorem 97 to derive:

Left hand side of Equation 12.1

\[ \sum_{i \in M} \sum_{j' \in Q'} \text{cost}^{\delta}_{j'}(i, \theta_{-j'}) \]

\[ \leq \text{cost}^{S'} + \alpha \cdot \Phi(0) \]

\[ = \text{cost}^{S'} - \alpha \cdot \int_{t=0}^{\infty} \frac{d}{dt} \Phi(t) dt \]

\[ \leq \text{cost}^{S'} + \alpha \cdot \int_{t=0}^{\infty} \left[ \frac{d}{dt} \text{cost}^{S'(\delta)}(t) + \delta \cdot \frac{d}{dt} \text{cost}^{S}(t) \right] dt \]

\[ \leq \text{cost}^{S'} + \alpha \cdot \text{cost}^{S'(\delta)} + (\alpha \delta) \cdot \text{cost}^{S} \]

\[ = \ (1 + \alpha/\delta) \cdot \text{cost}^{S'} + (\alpha \delta) \cdot \text{cost}^{S} \]

The last equality follows from Fact 12.4.1.

Thus, setting \( \lambda = (1 + \alpha/\delta) \) and \( \mu = \alpha \delta \), we get a robust price of anarchy bound of \( (1 + \alpha/\delta)/(1 - \alpha \delta) \). This leads to Theorem 94. More specifically, by setting \( \delta = 1/(2\alpha) \), we obtain a robust PoA bound of \( 2(1 + 2\alpha^2) \). The best bound we can obtain here is \( \frac{1+\alpha(\sqrt{\alpha^2+1}+\alpha)}{1-\alpha(\sqrt{\alpha^2+1}-\alpha)} \) when \( \delta = \sqrt{\alpha^2+1} - \alpha \). This bound becomes \( (\sqrt{2} + 1)/(\sqrt{2} - 1) \approx 5.8284 \) when \( \alpha = 1 \).

Remark. When all the release dates are zero, the robust price of anarchy bound improves to \( 4\alpha^2 \). This improvement follows from a better bound for Equation 12.4, namely, we can show that its left hand side is at most \( \alpha \cdot \Phi(0) \). Note that this bound is tight Cole et al. (2011) when \( \alpha = 1 \) (see Theorem 93).
12.4.1 Proof of Theorem 97

Let $\Phi_i(t)$ denote the contribution towards $\Phi(t)$ by machine $i \in M$, and let $\Phi_{ij'}(t)$ denote the contribution towards $\Phi_i(t)$ by job $j' \in Q'_i$. Thus, we have:

$$\Phi_{ij'}(t) = \sum_{j \in Q_i} \min\left(w_{j'} \cdot p_{ij}(t), w_j \cdot p_{ij'}(t)\right) \tag{12.6}$$

$$\Phi_i(t) = \sum_{j' \in Q'_i} \Phi_{ij'}(t) \tag{12.7}$$

$$\Phi(t) = \sum_{i \in M} \Phi_i(t) \tag{12.8}$$

The next two lemmas show that $\Phi(t)$ satisfies the boundary conditions at $t = 0$ and $t = \infty$.

**Lemma 98.** The potential function $\Phi(t)$ satisfies Equation 12.3.

**Proof.** Follows from the observation that each job has zero remaining processing length at time $t = \infty$. \hfill \Box

**Lemma 99.** The potential function $\Phi(t)$ satisfies Equation 12.4.

**Proof.** Fix any machine $i \in M$. For every job $j' \in Q'_i$, we have:

$$w_{j'} \cdot C^A_{j'}(i, \theta_{-j'}) \leq w_{j'} \cdot r_{ij'} + w_{j'} \cdot p_{ij'} + \alpha \cdot \sum_{j \in Q_i} \min\left(w_{j'} \cdot p_{ij}, w_j \cdot p_{ij'}\right)$$

$$= w_{j'} \cdot (r_{ij'} + p_{ij'}) + \alpha \cdot \sum_{j \in Q_i} \min\left(w_{j'} \cdot p_{ij}(0), w_j \cdot p_{ij'}(0)\right)$$

$$= w_{j'} \cdot (r_{ij'} + p_{ij'}) + \alpha \cdot \Phi_{ij'}(0)$$

$$\leq w_{j'} \cdot C^A_{j'}(\theta') + \alpha \cdot \Phi_{ij'}(0) \tag{12.9}$$

The first inequality holds since the scheduling policy $A_i$ has stretch $\alpha$ (see Definition 12.1.1). The last inequality holds since $r_{ij'} + p_{ij'}$ is at most the completion time of the job $j'$ under any feasible schedule. Finally, note that $w_{j'} \cdot C^A_{j'}(i, \theta_{-j'}) = \text{cost}^A_{j'}(i, \theta_{-j'})$ in the absence of
assignment costs. So the lemma follows when we sum both sides of Equation 12.9 over all machines \( i \in \mathcal{M} \) and jobs \( j' \in Q_i' \).

It remains to show that \( \Phi(t) \) satisfies Equation 12.5. We will first make some simple observations.

**Fact 12.4.2.** The functions \( p_{ij}(t), p^*_{ij'}(t) \), and \( \Phi_{ij'}(t) \) are all continuous and non-increasing in \( t \).

The following facts hold since the machines operate at speed \( \delta \) (resp. 1) under scenario \( S'(\delta) \) (resp. \( S \)).

**Fact 12.4.3.** Fix any machine \( i \in \mathcal{M} \), and any two jobs \( j' \in Q_i' \) and \( j \in Q_i \). At any time \( t \), we have:

\[
0 \geq \frac{d}{dt}(p^*_{ij'}(t)) \geq -\delta, \quad \text{and} \quad 0 \geq \frac{d}{dt}(p_{ij}(t)) \geq -1.
\]

**Fact 12.4.4.** At any time \( t \), on any machine \( i \in \mathcal{M} \), we have:

\[
0 \geq \frac{d}{dt}\left( \sum_{j' \in Q_i'} p^*_{ij'}(t) \right) \geq -\delta, \quad \text{and} \quad 0 \geq \frac{d}{dt}\left( \sum_{j \in Q_i} p_{ij}(t) \right) \geq -1.
\]

We now bound the rate of change in \( \Phi_i(t) \) due to any unfinished job under the scenario \( S'(\delta) \).

**Claim 100.** For every job \( j' \in Q_i' \) that completes after time \( t \) under the scenario \( S'(\delta) \), we have:

\[
\frac{d}{dt}(\Phi_{ij'}(t)) \geq -w_{j'} + W_i(t) \cdot \frac{d}{dt}(p^*_{ij'}(t)).
\]

**Proof.** Recall that \( \Phi_{ij'}(t) \) is a summation over a set of terms, each corresponding to a job \( j \in Q_i \) assigned to the same machine \( i \), but under a different scenario \( S \) (see Equation 12.6). Each such term is the minimum of two functions: \( w_{j'} \cdot p_{ij}(t) \) and \( w_j \cdot p^*_{ij'}(t) \). We partition all the jobs in \( Q_i \) into two subsets \( Y \) and \( Z \), depending on which of the two functions attain the minimum value:

\[
Y = \{ j \in Q_i : w_{j'} \cdot p_{ij}(t) \leq w_j \cdot p^*_{ij'}(t) \}, \quad \text{and} \quad Z = \{ j \in Q_i : w_{j'} \cdot p_{ij}(t) > w_j \cdot p^*_{ij'}(t) \}.
\]
The functions \( f^Y(t), f^Z(t) \) capture the respective contributions of the subsets \( Y \) and \( Z \) towards \( \Phi_{ij'}(t) \).

\[
\begin{align*}
  f^Y(t) &= \sum_{j \in Y} p_{ij}(t), & \text{and} & & f^Z(t) &= \left( \sum_{j \in Z} w_j \right) \cdot p^*_{ij'}(t).
\end{align*}
\]

Note that \( \Phi_{ij'}(t) = f^Y(t) + f^Z(t) \). Further, since the job \( j' \) completes after time \( t \) under the scenario \( S'(\delta) \), we have \( p^*_{ij'}(t) > 0 \). It follows that every job \( j \in Z \) has \( p_{ij}(t) > 0 \). In other words, every job in \( Z \) completes after time \( t \) under the scenario \( S \), which implies that \( \sum_{j \in Z} w_j \leq W_i(t) \). Hence, we conclude:

\[
\begin{align*}
  \frac{d}{dt} (\Phi_{ij'}(t)) &= \frac{d}{dt} f^Y(t) + \frac{d}{dt} f^Z(t) \\
  &= w_{j'} \cdot \frac{d}{dt} \sum_{j \in Y} p_{ij}(t) + \left( \sum_{j \in Z} w_j \right) \cdot \frac{d}{dt} (p^*_{ij'}(t)) \\
  &\geq -w_{j'} + \left( \sum_{j \in Z} w_j \right) \cdot \frac{d}{dt} (p^*_{ij'}(t)) \\
  &\geq -w_{j'} + W_i(t) \cdot \frac{d}{dt} (p^*_{ij'}(t)) 
\end{align*}
\]

Equation 12.10 follows from Fact 12.4.4. Equation 12.11 holds since \( \sum_{j \in Z} w_j \leq W_i(t) \) and \( \frac{d}{dt} (p^*_{ij'}(t)) \leq 0 \).

The next claim shows that we can ignore the jobs that finishes before time \( t \) under the scenario \( S'(\delta) \).

**Claim 101.** For every job \( j' \in Q'_i \) that completes before time \( t \) under the scenario \( S'(\delta) \), we have:

\[
\frac{d}{dt} (\Phi_{ij'}(t)) = 0.
\]

**Proof.** Follows from the observation that such a job \( j' \) has \( p^*_{ij'}(t') = 0 \) for all \( t' \geq t \).

The next claim bounds the overall rate of change of \( \Phi_i(t) \).
Claim 102. For any machine \(i \in \mathcal{M}\) and any time \(t\), we have:

\[
\frac{d}{dt} (\Phi_i(t)) \geq -W_i^*(t) - \delta \cdot W_i(t).
\]

Proof. We infer that:

\[
\frac{d}{dt} \Phi_i(t) = \sum_{j' \in Q'_i} \frac{d}{dt} \Phi_{ij'}(t)
\geq -W_i^*(t) + W_i(t) \cdot \sum_{j' \in Q'_i} \frac{d}{dt} \left(p_{ij'}^*(t)\right)
\geq -W_i^*(t) - \delta \cdot W_i(t)
\]

Equation 12.12 follows from Claim 100 and Claim 101. Equation 12.13 follows from Fact 12.4.4.

Now we are ready to bound the overall rate of change of \(\Phi(t)\).

Lemma 103. The potential function \(\Phi(t)\) satisfies Equation 12.5.

Proof. Follows from summing both sides of the inequality in Claim 102 over all machines \(i \in \mathcal{M}\), and recalling that \(\frac{d}{dt}\text{cost}^{S(\delta)}(t) = \sum_i W_i^*(t)\), and \(\frac{d}{dt}\text{cost}^S(t) = \sum_i W_i(t)\).

Theorem 97 follows from Lemma 98, Lemma 99, and Lemma 103.

12.5 Price of Anarchy using Dual Fitting

In this section, we improve the bound on the PoA of \(\alpha\)-stretch scheduling policies to \(4\alpha\) using dual fitting. For the scheduling policy Highest Density First, we get a PoA bound of 4 (since \(\alpha = 1\)), and this matches the lower bound known for non-preemptive policies in the absence of release dates and assignment costs Cole et al. (2011). Our approach is inspired by the framework developed in Anand et al. (2012) under a completely different context of online algorithms. For simplicity of exposition, we only derive the PoA of pure Nash equilibria.
Our main observation is that the dual variables of a LP relaxation to the underlying optimization problem capture the disutility incurred by each agent in an equilibrium state. This highlights the versatility of the framework developed in Anand et al. (2012).

12.5.1 Properties of Scheduling Policies

We require that the scheduling policies satisfy two properties in addition to the bounded stretch condition.

**Definition 12.5.1** (Myopic Policy). A scheduling policy is myopic iff its scheduling decision depends only on the status of the jobs available for processing at the present time instant. In particular, the decision is independent of the jobs that will be released in future.

Assume for a while that all the jobs have unit weights, and consider a machine which follows the SRPT scheduling policy. At time $t$, this machine looks at the set of jobs available in its queue, and works on the job $j$ with shortest remaining processing time $p_j(t)$. At time $t + 1$, it repeats the same process. This policy is myopic, since its scheduling decision depends only on the jobs currently available in the machine’s queue. The reader may find it helpful to keep this example in mind while going through the rest of this section.

**Definition 12.5.2** (Monotone Policy). A scheduling policy is monotone iff it satisfies three properties.

1. **Everything else remaining the same, the completion time of a job can never increase if it is released at an earlier date.**

2. **Everything else remaining the same, the completion time of a job $j$ can never decrease if the machine is asked to process an extra job $j'$.**

3. **For any two jobs $j$ and $j'$ with $w_j = w_{j'}$, $r_j \leq r_{j'}$, and $p_j \leq p_{j'}$, the completion time of job $j$ is at most the completion time of job $j'$, i.e., $C_j \leq C_{j'}$.**
Remark. All the scheduling policies that give \( O(1) \)-approximations on a single machine, with the exception of WLAPS, are myopic and monotone, and have stretch \( \alpha = 1 \).

We devote the rest of this section to the proof of the following theorem.

**Theorem 104.** Suppose that each machine declares a (possibly different) scheduling policy which is myopic, monotone and has stretch \( \alpha \geq 1 \). Then the price of anarchy of the induced game is at most \( 4\alpha \).

First we derive a bound on the completion time of a job.

**Lemma 105.** If a machine runs a scheduling policy with stretch \( \alpha \geq 1 \), then a job \( j \) (with weight \( w_j \), release date \( r_j \), processing length \( p_j \), and completion time \( C_j \)) on the machine satisfies the following condition.

\[
C_j \leq r_j + \alpha \cdot \left( \frac{W(r_j)}{w_j} \right) \cdot p_j.
\]

Here, the symbol \( W(r_j) \) denotes the total weight of the unfinished jobs at time \( t \).

**Proof.** Since the scheduling policy has stretch \( \alpha \geq 1 \), Definition 12.1.1 implies that:

\[
C_j \leq r_j + p_j + \sum_{j' \neq j : C_{j'} \geq r_j} \alpha \cdot \min(p_{j'}, \left( \frac{w_{j'}}{w_j} \right) \cdot p_j)
\]

\[
\leq r_j + \sum_{j' : C_{j'} \geq r_j} \alpha \cdot \left( \frac{w_{j'}}{w_j} \right) \cdot p_j
\]

\[
= r_j + \alpha \cdot \left( \frac{W(r_j)}{w_j} \right) \cdot p_j
\]

The above lemma justifies our use of the term “bounded stretch”, for traditionally the “stretch” of a job is defined as its completion time minus its release date divided by its processing length.
12.5.2 LP-relaxation

Consider the linear program Primal described below Anand et al. (2012). It has a variable \( x_{ijt} \) for each machine \( i \in M \), each job \( j \in J \) and each unit time-slot \( t \geq r_{ij} \). If the machine \( i \) processes the job \( j \) during the whole time-slot \( t \), then this variable is set to 1. The first constraint says that every job has to be completely processed. The second constraint says that a machine cannot process more than one unit of the jobs during any time-slot. Note that the LP allows a job to be processed simultaneously across different machines.

In the objective function, the term \( \sum_i \sum_{t \geq r_{ij}} h_{ij} \cdot (x_{ijt}/p_{ij}) \) gives the assignment cost incurred by the job \( j \). The term \( \sum_i \sum_{t \geq r_{ij}} w_j \cdot x_{ijt} \cdot (t/p_{ij}) \) is known as the fractional weighted completion time of the job \( j \). In a feasible schedule this quantity is no more than its integral weighted completion time, minus half of its weighted processing time. Finally, the remaining term \( \sum_i \sum_{t \geq r_{ij}} w_j \cdot x_{ijt} \cdot (t/p_{ij} + 1/2) \) equals half of the weighted processing time of the job. Thus, adding up these three terms, we see that the disutility of a job \( j \) is at least \( \sum_i \sum_{t \geq r_{ij}} h_{ij} \cdot (x_{ijt}/p_{ij}) + \sum_i \sum_{t \geq r_{ij}} w_j \cdot x_{ijt} \cdot (t/p_{ij} + 1/2) \). Hence, the linear program Primal is a valid relaxation of our problem.

\[
\text{Min } \sum_j \sum_i \sum_{t \geq r_{ij}} h_{ij} \cdot (x_{ijt}/p_{ij}) + \sum_j \sum_i \sum_{t \geq r_{ij}} w_j \cdot x_{ijt} \cdot (t/p_{ij} + 1/2) \quad \text{(Primal)}
\]

\[
\begin{align*}
\sum_i \sum_{t \geq r_{ij}} x_{ijt} / p_{ij} & \geq 1 & \forall j \\
\sum_j x_{ijt} & \leq 1 & \forall i, t \\
x_{ijt} & \geq 0 & \forall i, j, t : t \geq r_{ij}
\end{align*}
\]

Now, suppose that we constrain each machine to run at a reduced speed of \( 1/2\alpha \). In other words, each machine can process at most \( 1/2\alpha \) units of the jobs during one unit of time. It is easy to see that the modified LP described below is a valid relaxation under this new constraint. This transformation increases the objective by at most a factor of \( 2\alpha \).

\[
\text{Min } \sum_j \sum_i \sum_{t \geq r_{ij}} h_{ij} \cdot (x_{ijt}/p_{ij}) + \sum_j \sum_i \sum_{t \geq r_{ij}} w_j \cdot x_{ijt} \cdot (t/p_{ij} + \alpha) \quad \text{Primal(\alpha)}
\]
\[
\sum_{i} \sum_{t \geq r_{ij}} x_{ijt} / p_{ij} \geq 1 \quad \forall j
\]
\[
\sum_{j : t \geq r_{ij}} x_{ijt} \leq 1/(2\alpha) \quad \forall i, t
\]
\[
x_{ijt} \geq 0 \quad \forall i, j, t : t \geq r_{ij}
\]

**Lemma 106.** The optimal objective of the linear program PRIMAL(\(\alpha\)) is at most 2\(\alpha\) times the total disutility of the jobs in any feasible integral schedule.

Finally, we write down the dual of the above linear program.

\[
\text{Max} \quad \sum_{j} y_{j} - \sum_{i} \sum_{t} z_{it} \quad \text{DUAL(\(\alpha\))}
\]

\[
y_{j} \leq h_{ij} + w_{j} \cdot (t + \alpha \cdot p_{ij} + (2\alpha) \cdot (z_{it}/w_{j}) \cdot p_{ij}) \quad \forall i, j, t : t \geq r_{ij}
\]
\[
y_{j} \geq 0 \quad \forall j
\]
\[
z_{it} \geq 0 \quad \forall i, t
\]

12.5.3 Analysis

Fix a profile of scheduling policies \(A = (A_1, \ldots, A_{|\mathcal{M}|})\), where \(A_i\) gives the scheduling policy declared by the machine \(i\). For all \(i \in \mathcal{M}\), the scheduling policy \(A_i\) is myopic, monotone and has a stretch \(\alpha\). Recall the notations introduced in Section 12.3. For the rest of this section, we focus on any scenario \(S = (A, Q, \theta)\) where the strategy-profile \(\theta\) is a pure Nash equilibrium of GAME(\(A\)).

We will set the variables of the linear program DUAL(\(\alpha\)) so as to get a feasible dual solution, with an objective that is at least 1/2 times the total disutility of the jobs under the scenario \(S\). This, combined with Lemma 106 and weak duality, will imply that the price of anarchy of GAME(\(A\)) is at most 4\(\alpha\).

**Setting the dual variables:** The variable \(y_{j}\) is set to be the disutility of the job \(j\) under the scenario \(S\). Further, the variable \(z_{it}\) is set to be half of the total weight of the unfinished jobs on machine \(i\) at time \(t\), under the scenario \(S\).

\[
y_{j} = \text{cost}^{A}_{j}(\theta) = h_{\theta_{j}, j} + w_{j} \cdot C_{\theta_{j}}^{A}(\theta) \quad (12.14)
\]
\[
z_{it} = (1/2) \cdot W_{i}(t) \quad (12.15)
\]
Lemma 107. If the dual variables are set as in the equations 12.14, 12.15, then the objective of the linear program \( \text{Dual} (\alpha) \) is at least \((1/2) \cdot \sum_j \text{cost}_j^A (\theta) \).

**Proof.** We use the well-known fact that the total weighted completion time of all the jobs assigned to any specific machine \( i \) is equal to \( \sum_t W_i(t) \). Thus, we infer that:

\[
\sum_{i,t} z_{it} = (1/2) \cdot \sum_j w_j \cdot C^A_j (\theta) \leq (1/2) \cdot \sum_j \text{cost}_j^A (\theta).
\]

The lemma follows from the above inequality and the fact that \( \sum_j y_j = \sum_j \text{cost}_j^A (\theta) \).

Lemma 108. If the dual variables are set as in the equations 12.14, 12.15, then all the constraints of the linear program \( \text{Dual} (\alpha) \) are satisfied.

**Proof.** Clearly, the dual variables are set to nonnegative values. For the rest of the proof, we fix a job \( j \), a machine \( i \), and a time \( t \geq r_{ij} \), and show that the corresponding dual constraint is satisfied.

A Thought Experiment. We create a job \( j' \) with \( p_{ij'} = p_{ij} \), \( w_{j'} = w_j \), and \( r_{ij'} = t \). The machine \( i \) is now asked to process the set of jobs \( Q_i \cup \{j'\} \) using the scheduling policy \( A_i \).

Under this thought experiment, let \( C^*_{j'} \) denote the completion time of the job \( j' \). Recall that \( A_i \) is a myopic scheduling policy (Definition 12.5.1). Hence, under this thought experiment, the total weight of the unfinished jobs on machine \( i \) at time \( t \) is exactly \( W_i(t) + w_{j'} \). Since the policy \( A_i \) also has stretch \( \alpha \), Lemma 105 gives:

\[
C^*_{j'} \leq t + \alpha \cdot \left( \frac{W_i(t) + w_{j'}}{w_{j'}} \right) \cdot p_{ij'} = t + \alpha \cdot p_{ij'} + \alpha \cdot (W_i(t)/w_{j'}) \cdot p_{ij'}
\]

Plugging in the equalities \( W_i(t) = 2z_{it} \), \( w_{j'} = w_j \), and \( p_{ij'} = p_{ij} \), we get:

\[
C^*_{j'} \leq t + \alpha \cdot p_{ij} + (2\alpha) \cdot (z_{it}/w_j) \cdot p_{ij}
\]

(12.16)

We now consider two possible cases.
Case 1: Job \( j \) selects machine \( i \) under the outcome \( S \), so that \( \theta_j = i \).

Let \( C_j^* \) denote the completion time of the job \( j \) under the thought experiment. Recall that the machine follows a monotone scheduling policy. Hence, part 3 of Definition 12.5.2 implies that \( C_j^* \leq C_j^{*'} \). Further, part 2 of Definition 12.5.2 implies that \( C_j^A(\theta) \leq C_j^* \).

Accordingly, we have \( C_j^A(\theta) \leq C_j^{*'} \). Since \( y_j = h_{ij} + w_j \cdot C_j^A(\theta) \), we infer that:

\[
y_j \leq h_{ij} + w_j \cdot C_j^{*'}
\]  \((12.17)\)

Equation 12.16 and Equation 12.17 imply that the dual constraint is satisfied.

Case 2: Job \( j \) does not select machine \( i \) under the scenario \( S \), so that \( \theta_j \neq i \).

Consider the scenario \( S \). We want to bound the completion time of the job \( j \) when it switches to the machine \( i \), and everything else remains the same. This is denoted by \( C_j^A(i, \theta_{-j}) \). This also corresponds to a thought experiment, where the machine \( i \) is asked to schedule the jobs in \( Q_i \cup \{ j \} \) using the scheduling policy \( A_i \). The only difference between this thought experiment and the previous one is that here the job being inserted has an earlier release date (\( r_{ij} \leq r_{ij'} = t \), \( w_j = w_{j'} \), \( p_{ij} = p_{ij'} \)). Accordingly, part 1 of Definition 12.5.2 implies that \( C_j^A(i, \theta_{-j}) \leq C_j^{*'} \). Since \( \text{cost}_j^A(i, \theta_{-j}) = h_{ij} + w_j \cdot C_j^A(i, \theta_{-j}) \), we get:

\[
\text{cost}_j^A(i, \theta_{-j}) \leq h_{ij} + w_j \cdot C_j^{*'}
\]  \((12.18)\)

Finally, recall that the strategy-profile \( \theta \) is a pure Nash equilibrium of \( \text{GAME}(A) \). Hence, we have:

\[
y_j = \text{cost}_j^A(\theta) \leq \text{cost}_j^A(i, \theta_{-j})
\]  \((12.19)\)

Equation 12.16, Equation 12.18, and Equation 12.19 imply that the dual constraint is satisfied.

Theorem 104 now follows from Lemma 106, Lemma 107, and Lemma 108.
12.6 Price of Anarchy of a Scheduling Policy with Large Stretch

In this section, we consider a policy called Weighted Latest Arrival Processor Sharing (WLAPS) which has been extensively studied in scheduling theory Edmonds and Pruhs (2009); Bansal et al. (2010). Although this policy gives $O(1)$-approximation to weighted completion time on a single machine, we show that its induced game has large PoA (see Lemma 109). Further, we show that WLAPS fails our $\alpha$-stretch condition (see Lemma 110), hence making a case for bounded stretch policies.

Fix a machine $i$ which has to process a set of jobs, and let $J_i(t)$ denote the set of unfinished jobs that are available for processing at time $t$. The scheduling policy WLAPS takes a parameter $\epsilon \in [0,1]$ as input. Let $J_i^\epsilon(t)$ denote the $\epsilon|J_i(t)|$ jobs in $J_i(t)$ with the highest release dates. At any time $t$, the machine works on the jobs $j \in J_i^\epsilon(t)$ in proportion to their weights, so that for all $j \in J_i^\epsilon(t)$ we have:

\[
\frac{d}{dt}(p_{ij}(t)) = -\sum_{j' \in J_i^\epsilon(t)} w_{jj'} w_{j'j}.
\]

WLAPS is an example of a non-clairvoyant scheduling policy, and reduces to Weighted Round Robin when $\epsilon = 1$. For simplicity, we fix $\epsilon = 1/2$ in the following proof; however, the proof can be easily extended to any value of $\epsilon$.

**Lemma 109.** If every machine follows the scheduling policy WLAPS($\epsilon$) with $\epsilon = 1/2$, then the PoA of the resulting game is $\Omega(n)$, where $n$ denotes the total number of jobs.

**Proof.** Consider the following instance. There are $n$ jobs and $n$ machines. Each job has unit weight. Among the $n$ jobs, there is one big job $j^*$ which has processing length $p_{ij^*} = n$ and release date $r_{ij^*} = \kappa$ on all the machines. Here, $\kappa$ is an arbitrarily small positive value. The remaining $n-1$ small jobs have processing lengths $p_{ij} = \kappa + \beta$ (which is an arbitrarily small positive value) on machine $i = 1$, and processing length $p_{ij} = n$ on the other machines $i \neq 1$. Further, these small jobs have release dates $r_{ij} = 0$ on all the machines.

In an optimal solution, all the small jobs are assigned to machine 1, and the big job is assigned to any other machine. Hence, the sum of the completion times of the jobs is
at most $O(n)$. In contrast, there exists a bad equilibrium - the big job on machine 1, and each small job on a distinct machine. Here, the total completion time of all the jobs is at least $\Omega(n^2)$. It is easy to see that this is a Nash Equilibrium. In particular, no small job wants to deviate to machine 1, since that machine would make the small job wait till it finishes the big job.

Lemma 110. The scheduling policy WLAPS($\epsilon$) has stretch $\alpha = \Omega(n)$ when $\epsilon = 1/2$. Here, the symbol $n$ denotes the number of jobs processed by the machine.

Proof. Consider the following instance. A machine is processing a set of $n$ jobs $X_1 \cup X_2$ using the scheduling policy WLAPS(1/2). Each job $j \in X_1 \cup X_2$ has weight $w_j = 1$. The subset $X_1$ consists of $n/2$ jobs. Each job $j \in X_1$ has a processing length $p_j = 1$ and release date $r_j = 0$. The subset $X_2$ also consists of $n/2$ jobs. Each job $j \in X_2$ has a processing length $p_j = n$ and release date $r_j = \kappa$, where $\kappa$ is an arbitrarily small positive value.

It is easy to see that the machine starves the jobs in $X_1$ in favor of the jobs in $X_2$ from time $t = \kappa$ onwards. Hence, all the jobs in $X_2$ complete at time $t = n^2/2 + \kappa$, and the completion time of any job $j \in X_1$ is at least $\Omega(n^2)$. On the other hand, the bounded stretch condition (see Definition 12.1.1) requires the completion time of such a job to be at most $\alpha n$. Thus, WLAPS($\epsilon$) has stretch $\alpha = \Omega(n)$ when $\epsilon = 1/2$. 

12.7 Nonpreemptive Policies

In this section we develop non-preemptive coordination mechanisms, and prove Theorem 96. We show how to transform any preemptive policy into a non-preemptive policy in which the completion time of a job increases at most by a factor 2. This ensures that if the preemptive policy had a stretch $\alpha$ to begin with, then the resulting non-preemptive policy has a stretch $2\alpha$. This implies Theorem 96. Our transformation is similar in spirit to the one used in Hall et al. (1996).

Consider any $\alpha$-stretch scheduling policy $A_i$ declared by machine $i$. Recall that $Q_i$
denotes the set of jobs assigned to machine \( i \). Let \( C_j \) be completion time of job \( j \) in the schedule produced by policy \( A_i \) on the set \( Q_i \). Renumber the jobs in \( Q_i \) such that \( C_{j-1} < C_j \). We modify the preemptive schedule produced by \( A_i \) into a non-preemptive schedule in the following manner.

- Consider the jobs in \( j \in Q_i \) in increasing order of their completion time values \( C_j \).

  Schedule the jobs non-preemptively in this order.

Let \( \overline{C}_j \) denote the completion time of job \( j \) in this new schedule. The following theorem shows that, the completion time every job in the new schedule increases at most by a factor of 2.

**Theorem 111.** \( \overline{C}_j \leq 2C_j \)

**Proof.** Fix a job \( j \). The completion time of the job \( j \) in the non-preemptive schedule can be bounded by:

\[
\overline{C}_j \leq \max_{k \in [j]}\{r_{ik}\} + \sum_{k \in [j]} p_{ik}
\]

Recall that we renumbered the jobs in the set \( Q_i \) in the increasing order of \( C_j \) values.

The above inequality holds since there is no idle period after the job with highest release date in the set \([1 \ldots j]\) is released.

The proof of the theorem follows from the observations that \( C_j \geq \max_{k \in [j]} r_{ik} \) and \( C_j \geq \sum_{k \in [j]} p_{ik} \).

12.8 Summary and Open Problems

In this chapter, we designed coordination mechanisms that achieve near optimal PoA for the social objective of sum of completion times. We also gave a characterization of scheduling policies that lead to outcomes with small PoA. An interesting research direction in this area is to identify properties which are necessary to achieve small PoA. Another interesting research direction is to design coordination mechanisms that achieve small PoA for other
scheduling objectives such as minimizing average flow-time or minimizing the maximum flow-time etc.

12.9 Notes

This chapter is based on joint work with Sayan Bhattacharya, Sungjin Im, and Kamesh Munagala. A preliminary version of this result appeared in 5th Innovations in Theoretical Computer Science Conference, 2014, (Bhattacharya et al., 2014b).
Coordination Mechanisms for Selfish Routing over Time on a Tree

13.1 Introduction

As a central topic in algorithmic game theory, selfish routing problems have been studied extensively in the context of congestion games (Koutsoupias and Papadimitriou, 1999; Roughgarden, 2009; Roughgarden and Tardos, 2000). Being a representative class of potential games, network congestion games have served as a foundation for proving price of anarchy results. However they lack an important aspect of real network routing which is the fact that routing happens over time, and any realistic model should take this into account.

To address this issue, several new models have been proposed to capture the nature of realistic routing over time (Koutsoupias and Papakonstantinopoulou, 2012; Christodoulou et al., 2009, 2011; Koch et al., 2011; Peis et al., 2009; Farzad et al., 2008; Cole et al., 2011; Azar et al., 2008). Amongst these models, the concept of coordination mechanisms, first introduced in an influential paper in (Christodoulou et al., 2009), have been proposed to capture the queueing nature of routing. Coordination mechanisms model the decentralized nature of routing decisions made by machines and the selfish behavior of jobs: they do so by seeking local policies that achieve a good price of anarchy in the resulting equilibria in
a corresponding game. While these subjects have attracted a great amount of research, the problem of designing better coordination mechanisms for routing over time has remained a wide open problem, and results have been developed only for very special classes of networks such as parallel edges (corresponding to a multi-processor scheduling problem (Christodoulou et al., 2009; Immorlica et al., 2009; Cole et al., 2011; Azar et al., 2008)). In this chapter, we focus on tree networks, and provide the first coordination mechanisms with provable price of anarchy.

### 13.1.1 The Model

![Diagram of job assignment](image)

**Figure 13.1:** In our model, each edge $e$ has a speed $s(e)$. Each job $j$ starts at the root and needs to travel to one of the nodes in $L_j$, shown in dashed circles. To the right, we show how our model generalizes the related machines (top right) and the restricted assignment setting (bottom right).

We are given a tree $T = (V, E)$ rooted at node $r \in V$. Each edge $e \in E$ in the tree is a machine\(^1\) with speed $s(e)$. For the rest of this chapter, we use the terms “edge” and

\(^1\) The reason that we use the term “machine” to refer to the edges is to cope with the scheduling literature.
“machine” interchangeably. There is a set of jobs $J$ with unique identifiers, which will be used by our policies for breaking ties consistently. Each job $j \in J$ has weight $w_j$ and length $p_j$, and its processing time on edge $e$ is $p_{je} = p_j/s(e)$. Each edge can process at most one unit of the jobs during a unit time-step, and a job $j$ requires $p_{je}$ time-units of processing on an edge $e$. At time $t = 0$, all the jobs are located at the root.

A job $j$ can be served by only the nodes in the subset $L_j \subseteq V$. This models, for example, the fact that some servers in a data center may not have the necessary content to satisfy a request. The job $j$ starts at the root and wants to exit the tree through any one of the nodes in $L_j$, which are called the destination-nodes of $j$. Accordingly, the job $j$ selects a path $i = (e_1, \ldots, e_l)$ that begins at the root of the tree and terminates at some node in $L_j$. Here, $e_1$ is the first edge on the path (adjacent to the root), and $e_l$ is the last edge. The job can start getting processed on an edge $e_k$, $k \in \{2, \ldots, l\}$, only after it is processed completely by the preceding edge $e_{k-1}$. The job exits the tree when it gets completely processed on the last edge $e_l$, and the time at which this event takes place is called the sojourn time of the job. The weighted sojourn time of $j$ is equal to its weight $w_j$ times its sojourn time. Note that since all the jobs are at the root node at time $t = 0$, the average sojourn time is equivalent to the average flow-time in our context. A reader familiar with the scheduling literature can see that our model is a generalization of the related machine and the restricted assignment settings (see Figure 13.1).

The underlying optimization problem asks us to route every job $j$ through a root-to-destination path terminating in $L_j$, and provide a scheduling policy on each edge so as to minimize the sum of their weighted sojourn times. We allow preemption of jobs on an edge. The jobs, however, are selfish agents who cannot be forced to obey the dictate of a centralized authority. Thus, in our model, the machines declare their scheduling policies in advance, and this induces a simultaneous-move game between the jobs. We require that the scheduling policies be strongly local, in the sense that the scheduling decision by an edge at any time-instant depends only on the current set of jobs waiting to be processed.

One should think about these machines as a communication link.
on that edge, and is independent of the global state of the system.

In this game, the strategy of each job $j$ consists of selecting a path from the root to any one of the destination-nodes in $L_j$. The vector $\theta = (\theta_1, \ldots, \theta_{|J|})$ denotes an outcome (strategy-profile) of the game, where $\theta_j$ is the path selected by the job $j$. The symbol $(i, \theta_{-j})$ denotes an outcome where the job $j$ selects the path $i$, and every job $j' \neq j$ selects the path $\theta_{j'}$. The symbol $\text{Cost}_j(\theta)$ denotes the cost incurred by the job $j$ under the outcome $\theta$, which is equal to its weighted sojourn time. An outcome $\theta$ is in a (pure) Nash equilibrium iff no job can reduce its cost by unilaterally deviating to another path, i.e., iff $\text{Cost}_j(\theta) \leq \text{Cost}_j(i, \theta_{-j})$ for all jobs $j$ and root-to-destination paths $i$ terminating at a node in $L_j$.

The price of anarchy (PoA) of the game is the worst (maximum) possible ratio between the total cost of the agents in a Nash equilibrium and the optimal objective of the underlying optimization problem. Intuitively, it is a measure of the degradation in the overall system-performance due to the strategic interactions between the jobs.

We want to solve the following problem: Find a set of scheduling policies for the machines so as to minimize the PoA of the resulting game.

13.1.2 Our Results

We analyze the PoA of the game induced among the jobs when the machines follow a natural and easy to implement scheduling policy known as Shortest Job First. We prove the following theorem in this chapter.

**Theorem 112.** If every machine follows Shortest Job First (SJF) policy and every job has unit weight, then the price of anarchy of the induced game is $O(\log^2(s_{\text{max}}/s_{\text{min}}))$, where $s_{\text{max}}$ (resp. $s_{\text{min}}$) is the maximum (resp. minimum) speed among all the machines.

Here, $s_{\text{max}}$ (resp. $s_{\text{min}}$) denotes the speed of the fastest (resp. slowest) machine. Note that this implies constant PoA when all the machines are identical. Furthermore, this result generalizes the PoA results for scheduling game we considered in the previous chapter.

211
13.1.3 Our Techniques:

Both of the PoA upper-bounds in this chapter are obtained using the following simple technique: First, we find an LP relaxation for the underlying optimization problem and write down its dual. Next, we consider any arbitrary Nash equilibrium outcome $\theta$, and based on this outcome assign values to the dual variables in a way that satisfies all the dual constraints, thereby getting a feasible solution to the dual LP. Finally, we show that the objective of this feasible dual solution is at least $1/\alpha$ times the total cost incurred by the jobs under $\theta$. Weak duality implies that the PoA of the game is at most $\alpha$. Our overall approach is inspired by papers (Anand et al., 2012; Bhattacharya et al., 2014b). This technique is very powerful and can potentially be applied to bound the PoA in several other settings. Bilò et al. Bilò (2012) give another application of this technique to analyze PoA.

Apart from the overall idea of using the dual fitting technique to analyze the PoA of the game, writing a linear programming relaxation with small integrality gap turns be a significant challenge for our problem. A direct extension of the time-indexed LP (Anand et al., 2012) has a huge integrality gap in our setting. We circumvent this difficulty for the case of unweighted jobs by first finding a set of critical edges which play a crucial role in how the jobs delay each other. Then, we write a time-indexed LP relaxation taking into account only these edges which brings the integrality gap down. See Section 13.2 for details.

13.1.4 History.

Following the landmark paper of Christodoulou et al. (Christodoulou et al., 2009) who initiated the study of coordination mechanisms, several papers have been written on the topic for various problems. However, most of these results are for machine scheduling problems, either proving PoA results on the makespan objective function (Immorlica et al., 2009; Caragiannis, 2009; Azar et al., 2008) or recently for the weighted completion times (Cole et al., 2011; Bhattacharya et al., 2014b). In the context of selfish routing, the multi-
machine scheduling problem corresponds to a network of parallel edges and related machine scheduling is a special case of our model where the tree is a star, i.e., a tree of depth one (Figure 13.1). The only two results that go beyond the scheduling problem are by Hoefer et al (Hoefer et al., 2009) and by Christodoulou et al. (Christodoulou et al., 2011). The first paper only studies existence and computation of equilibria for various coordination mechanisms and leaves the PoA question as an open problem. The second paper discusses a quite different setting with non-atomic players.

Our work is also related to the literature on the PoA of selfish routing (Koutsoupias and Papadimitriou, 1999; Roughgarden and Tardos, 2000), and more specifically unsplittable selfish routing (Roughgarden, 2009). Here we extend the selfish routing model by incorporating a temporal component into the problem formulation. Other attempts to address this issue include (Koutsoupias and Papakonstantinopoulou, 2012; Koch et al., 2011; Farzad et al., 2008; Peis et al., 2009), but none of these results discuss coordination mechanisms using strongly local (decentralized) policies.

Scheduling over tree and line networks are considered in the online (and resource augmentation) setting in a recent work by (Im and Moseley, 2015). Finally, our work is related to approximation algorithms of classical optimization problems for minimizing the weighted sum of completion times and flow times (Karger et al., 2010; Chekuri and Khanna, 2004; Anand et al., 2012; Bansal and Kulkarni, 2015), none of which present an approximation algorithm for the problem minimizing average completion time in routing over a tree.

13.2 PoA Analysis

In this section, we assume that no two jobs have same length, and bound the PoA of the game where every edge follows the Shortest Job First (SJF) scheduling policy (Theorem 112). We start with a high-level overview of our approach. We say that a job \( j \) is delayed by another job \( j' \) on an edge \( e \) at time \( t \) iff two conditions are satisfied at the same time: (a) the job \( j \) is available for processing on edge \( e \), and (b) instead of \( j \), the edge is processing the job \( j' \). A machine following the SJF policy never sits idle when one or
more jobs are waiting in its queue. This ensures that the sojourn time of a job \( j \) is exactly equal to the total amount of processing done on \( j \) by all the edges, plus the total delay it encounters due to all other jobs. The former quantity is given by \( \sum_{e \in i} p_{je} \), where \( i \) is the path selected by the job. It is the latter quantity for which it is difficult to get a closed-form expression. We overcome this difficulty by showing that there is a small subset of critical edges on any path (Definition 114), and that one job can delay another only on one of these edges. This line of reasoning culminates in a key structural result (Theorem 116) that gives an upper bound on the maximum delay a job \( j \) can experience due to any single job \( j' \neq j \).

As alluded in Section 13.1.2, we now confront the task of designing a suitable LP relaxation for the underlying optimization problem. A straightforward extension of the time-indexed LP considered in Anand et al. (2012) to our setting leads to a large integrality gap. On the positive side, the time-indexed LP has one nice property: Its dual has variables that can be interpreted as decomposing the total delay incurred by a job across the edges on its path. The duals of other “natural” LP relaxations for our problem do not seem to be amenable to such a nice interpretation. Accordingly, we modify the time-indexed LP relaxation by only taking into account the critical edges of the tree. As the number of critical edges is small, this brings down the integrality gap of the LP. We then manage to fit its dual using Theorem 116.

To proceed with the technical details, let \( s_{\max} \) (resp. \( s_{\min} \)) denote the speed of the fastest (resp. slowest) of a machine, and let \( K = \lceil \log(s_{\max}/s_{\min}) \rceil \). For ease of exposition, we assume that the speeds of the machines are discretized in powers of two. By standard time stretching arguments, it is easy to show that this assumption can lead to at most a factor two loss in the PoA of our coordination mechanism.

**Assumption 113.** For all \( e \in E \), we have \( s(e) = 2^k \cdot s_{\min} \) for some \( k \in \{1, 2, \ldots, K\} \).

**Definition 114.** Let \( e_{i,k} \) denote the edge of minimum depth on path \( i \) that has speed \( 2^k s_{\min} \). We refer to such an edge as a critical edge on that path. We define \( E_i = \bigcup_k \{e_{ik}\} \) to be the...
set of all critical edges on path i.

Below, we describe our LP relaxation for the underlying optimization problem. For rest of the chapter, we overload the notation $i \in \mathcal{L}_j$ to denote a path $i$ that starts at the root and terminates at a node in $\mathcal{L}_j$.

Minimize $\sum_j \sum_{i \in \mathcal{L}_j} \sum_{t} x_{ij} \cdot \left( \frac{t}{p_{je}} \right) + \sum_j \sum_{i \in \mathcal{L}_j} \sum_{e \in i} \sum_{t} x_{ijet}$ \hspace{1cm} (13.1)

\[\sum_{i \in \mathcal{L}_j} x_{ij} \geq 1 \hspace{1cm} \forall \text{ jobs } j \] \hspace{1cm} (13.2)

\[\sum_{t} x_{ijet} \geq x_{ij} \hspace{1cm} \forall \text{ jobs } j, \text{ paths } i \in \mathcal{L}_j, \text{ edges } e \in i \hspace{1cm} (13.3)\]

\[\sum_{j} \sum_{i \in \mathcal{L}_j; e \in i} x_{ijet} \leq 1 \hspace{1cm} \forall \text{ edges } e, \text{ times } t \] \hspace{1cm} (13.4)

\[x_{ijet}, x_{ij} \geq 0 \hspace{1cm} \forall j, i \in \mathcal{L}_j, e \in i, t \] \hspace{1cm} (13.5)

In an integral feasible solution of the above linear program, the variable $x_{ij} \in \{0, 1\}$ indicates if the job $j$ takes the path $i \in \mathcal{L}_j$. The variable $x_{ijet} \in \{0, 1\}$ indicates if the job $j$ takes the path $i \in \mathcal{L}_j$ and is being processed on the edge $e \in i$ at time-step $t$.

Constraint 13.2 states that every job has to take some path. Constraint 13.3 states that if a job $j$ takes a path $i \in \mathcal{L}_j$, then it has to get completely processed on all the edges on this path. Finally, constraint 13.4 states that every edge can process at most one unit of the jobs during one time-step.

The second summation in the LP objective gives the total amount of processing done on all the jobs, which clearly is a lower bound on the sum of their sojourn-times. Now, fix any job $j$ which takes a path $i \in \mathcal{L}_j$, and consider an edge $e \in i$ on this path. The term $\sum_t x_{ijet} \cdot (t/p_{je})$ is known as the fractional completion time Anand et al. (2012) of the job $j$ on the edge $e$. This is at most the time at which the edge $e$ finishes processing the job, which, in turn, is at most the sojourn-time of $j$. We sum this quantity over all the critical
edges $E_i$ on path $i$, and the number of such critical edges is $O(K)$. Thus, the summation $\sum_{i \in L_j} \sum_{e \in E_i} \sum_{t} (t/p_{je}) \cdot x_{i ej t}$ is $O(K)$ times the sojourn-time of $j$. Summing over all the jobs, we see that the overall LP objective is $O(K)$ times the objective of the underlying optimization problem.

We now get a new LP by replacing the 1 in the right hand side of constraint 13.4 with $1/(4K)$. This imposes the condition that a machine can process at most $1/(4K)$ units of the jobs during one time-step, and, by standard scaling arguments, it is easy to show that this increases the LP objective by a factor of $4K$. The dual of this new LP is given by LP (13.6). By weak duality, we get Lemma 115.

\[
\text{Max } \sum_{j} y_j - \frac{1}{4K} \sum_{e, t} z_{et} \tag{13.6}
\]

\[
y_j \leq \sum_{e \in i} u_{ije} \quad \forall \text{ jobs } j, \text{ paths } i \in L_j \tag{13.7}
\]

\[
\frac{u_{ije}}{p_{je}} - z_{et} \leq 1 \quad \forall \text{ jobs } j, i \in L_j, e \in i \setminus E_i, \text{ times } t \tag{13.8}
\]

\[
\frac{u_{ije}}{p_{je}} - z_{et} \leq \frac{t}{p_{je}} + 1 \quad \forall \text{ jobs } j, i \in L_j, e \in E_i, \text{ times } t \tag{13.9}
\]

\[
y_j, u_{ije}, z_{et} \geq 0 \quad \forall j, t, i \in L_j, e \in i \tag{13.10}
\]

**Lemma 115.** The objective of any feasible solution to LP (13.6) is $O(K^2)$ times the optimal objective of the underlying optimization problem, where $K = \lfloor \log(s_{\text{max}}/s_{\text{min}}) \rfloor$.

For the rest of this section, we will assume that every edge follows SJF scheduling policy, and we will analyze the PoA of the resulting game. The theorem below bounds the maximum delay a job can encounter due to any other job.

**Theorem 116.** Suppose that every machine runs SJF scheduling policy. Consider any two jobs $j^*, j_1^*$, and fix any outcome $\theta$ (not necessarily a Nash Equilibrium) of the induced game. Let $e \in \theta_{j^*} \cap \theta_{j_1^*}$ be an edge with slowest speed that is common to the paths taken by the jobs. Then the total delay of the job $j^*$ due to the job $j_1^*$ is at most $2p_{j^*}/s(e)$.

216
Sketch. Consider the path $\theta_j^* = (e_1, \ldots, e_l)$, where $e_1$ (resp. $e_l$) is the edge adjacent to (resp. farthest away from) the root. We decompose this path in $w$ segments, for some natural number $w$, in the following manner. Consider a function $f : [1, w+1] \to [1, l+1]$ such that $1 = f(1) < f(2) < \cdots < f(w+1) = l+1$. Segment $k \in [1, w]$ corresponds to the sequence of edges $e_{f(k)}, e_{f(k)+1}, \ldots, e_{f(k+1)-1}$. The decomposition satisfies two properties.

- The speeds of the first edges of these segments form a strictly decreasing sequence.

  Hence, Assumption 113 implies that $s(e_{f(1)}) > s(e_{f(2)})/2 > \cdots > s(e_{f(w)})/2^{w-1}$.

- Within each segment, the speed of the first edge is at most the speed of any other edge. Thus, $s(e_{f(k)}) \leq s(e)$ for all $k \in [1, w]$ and $e \in \{e_{f(k)}, \ldots, e_{f(k+1)-1}\}$.

Note that the first edge $e_{f(k)}$ of every segment $k \in [1, w]$ is a critical edge on the path $\theta_j^*$. We show that the job $j^*$ can get delayed by other jobs only on this set of edges $\{e_{f(k)} : k \in [1, w]\}$. Now, under SJF policy, the job $j^*$ can get delayed by another job $j^*_1$ only if $p_{j^*_1} \leq p_{j^*}$, and, in this case, the delay experienced by $j^*$ due to $j^*_1$ on any edge $e_{f(k)}$ is at most $p_{j^*_1}/s(e_{f(k)}) \leq p_{j^*}/s(e_{f(k)})$. Thus, the total delay incurred by $j^*$ due to $j^*_1$ is upper bounded by the sum $\sum_{k=k^*}^1 p_{j^*_1}/s(e_{f(k)})$, where $k^*$ is the segment with the largest index whose first edge also belongs to $\theta_{j^*_1}$. This sum is part of a geometric series with common ratio $1/2$, and is at most $2p_{j^*}/s(e_{e_{f(k^*)}})$. The theorem follows from the fact that the edge $e_{f(k^*)}$ has the slowest speed among all the edges that are common to the paths $\theta_j^*$ and $\theta_{j^*_1}$.

Setting The Dual Variables:

Now, we give a procedure that takes any Nash equilibrium of the game (under SJF policy), and sets the variables in LP (13.6). Fix any outcome $\theta$ of the game (under SJF policy) that is in a Nash equilibrium.

- Set $y_j \leftarrow \text{Cost}_j(\theta)$.

- Consider an indicator variable $\lambda_{jyt} \in \{0, 1\}$ that is set to 1 iff : (a) the path $\theta_j$ contains the edge $e$ as a critical edge, i.e., $e \in E_{\theta_j}$ and (b) the sojourn-time of job $j$
is at least $t$ under the outcome $\theta$.

Set $z_{et} \leftarrow \sum_j 2 \cdot \lambda_{jet}$.

- Set the variable $u_{ije}$ as follows:
  
  (a) If $e \neq E_i$, then $u_{ije} \leftarrow p_{je}$.

  (b) Else if $e \in E_i$, then let $\Gamma_{ije}$ denote the set of all jobs $j' \neq j$ such that $e$ is the slowest edge in $i \cap \theta_{j'}$. Let $\delta_{ij}(j')$ be the total delay experienced by $j$ due to the job $j'$ under the outcome $(i, \theta_{-j})$. Set $u_{ije} \leftarrow p_{je} + \sum_{j' \in \Gamma_{ije}} \delta_{ij}(j')$.

Lemma 117. If the dual variables are assigned as described above, then for all jobs $j$ and root-to-leaf paths $i \in L_j$, we have $\text{Cost}_j(i, \theta_{-j}) = \sum_{e \in i} u_{ije}$.

**Proof.** Fix the outcome $(i, \theta_{-j})$. Now, the sojourn-time of $j$ equals the amount of time $j$ is processed, plus the amount of time it is delayed due to the other jobs. The former quantity is equal to $\sum_{e \in i} p_{je}$, the latter quantity being $\sum_{j' \neq j} \delta_{ij}(j')$. Thus, we get:

$$\text{Cost}_j(i, \theta_{-j}) = \sum_{e \in i} p_{je} + \sum_{j' \neq j} \delta_{ij}(j') = \sum_{e \in i} p_{je} + \left( \sum_{e \in E_i} \sum_{j' \in \Gamma_{ije}} \delta_{ij}(j') \right)$$

$$= \sum_{e \in i \setminus E_i} p_{je} + \sum_{e \in E_i} \left( p_{je} + \sum_{j' \in \Gamma_{ije}} \delta_{ij}(j') \right)$$

$$= \sum_{e \in i \setminus E_i} u_{ije} + \sum_{e \in E_i} u_{ije} = \sum_{e \in i} u_{ije}$$

(13.11)

To see why equation 13.11 holds, define $J^+$ to be the set of jobs which make positive contributions towards the sum $\sum_{j' \neq j} \delta_{ij}(j')$, i.e., $J^+ = \{j' \neq j : \delta_{ij}(j') > 0\}$. Each job $j' \in J^+$ has $i \cap \theta_{j'} \neq \emptyset$, and there is a unique critical edge on path $i$ that is also the slowest edge in $i \cap \theta_{j'}$. So each job $j' \in J^+$ is part of exactly one of the sets in $\{\Gamma_{ije} : e \in E_i\}$. In other words, $\{\Gamma_{ije} : e \in E_i\}$ induces a partition of the jobs in $J^+$.

Lemma 118. Our setting of the dual variables satisfy the constraints 13.7 and 13.8 of LP (13.6).
The right hand side of constraint 13.7, by Lemma 117, is equal to Cost\(_j(i, \theta_{-j})\). The left hand side, by definition of dual variables, is equal to Cost\(_j(\theta)\). The constraint is satisfied as the Nash equilibrium condition dictates that Cost\(_j(\theta) \geq Cost_j(i, \theta_{-j})\).

In constraint 13.8, the edge \(e\) is not a critical edge on the path \(i\). Hence, our setting of dual variables implies that the quantity \(z_{et}\) is zero at all times \(t\). Furthermore, the quantity \(u_{ije}\) is set to \(p_{je}\), so that the left hand side of the constraint is 1, which is equal to its right hand side.

Lemma 119. Fix any job \(j\), any path \(i \in L_j\), any critical edge \(e \in E_i\), and any job \(j' \in \Gamma_{ije}\). Let \(\delta_{ij}(j', t)\) be the total delay experienced by \(j\) due to the job \(j'\) (anywhere in the tree) on or after time \(t\), under the outcome \((i, \theta_{-j})\). We have \(\delta_{ij}(j', t) \leq 2\lambda^{*}_{j'et} \cdot p_{je}\).

Proof. The main difficulty in proving the lemma is that \(\delta_{ij}(j', t)\) refers to the outcome \((i, \theta_{-j})\), whereas \(\lambda^{*}_{j'et}\) refers to the outcome \(\theta\). So we introduce the quantity \(\lambda^{*}_{j'et}\), which is the exact analogue of \(\lambda_{j'et}\) under the outcome \((i, \theta_{-j})\). Note that the edge \(e\) already belongs to \(\theta_{j'}\) and is a critical edge. Hence, we drop condition (a) used in defining \(\lambda_{j'et}\) in our definition of the dual variables. We then consider two possible cases.

\[
\lambda^{*}_{j'et} = \begin{cases} 
1 & \text{if the sojourn-time of } j' \text{ under } (i, \theta_{-j}) \text{ is at least } t; \\
0 & \text{otherwise.}
\end{cases}
\]

By our assumption in the first paragraph of Section 13.1.2, either \(p_j < p_{j'}\) or \(p_j > p_{j'}\).

Case 1. Either \(\lambda^{*}_{j'et} = 0\) or \(p_j < p_{j'}\). Here, if \(\lambda^{*}_{j'et} = 1\), then the job \(j'\) is already out of the system by time \(t\) under the outcome \((i, \theta_{-j})\). Naturally, we have \(\delta_{ij}(j', t) = 0\), and the lemma holds. On the other hand, if \(p_j < p_{j'}\), then SJF scheduling policy ensures that the job \(j\) never gets delayed by the job \(j'\), and we again have \(\delta_{ij}(j', t) = 0\).

Case 2. \(\lambda^{*}_{j'et} = 1\) and \(p_j > p_{j'}\). In this case, first note that by Theorem 116, \(\delta_{ij}(j', t) \leq 2p_{je}\). Since \(\lambda^{*}_{j'et} = 1\), we get:

\[
\delta_{ij}(j', t) \leq 2\lambda^{*}_{j'et} \cdot p_{je} \tag{13.12}
\]
Since \( p_j > p_{j'} \), SJF scheduling policy ensures that the processing of \( j' \) is not affected if \( j \) switches its path. Specifically, the time-steps at which \( j' \) is processed by edge \( e \) remains unchanged under the two outcomes \((i, \theta_j - j)\) and \( \theta \). Thus, we have \( \lambda_{j'et} = \lambda^*_{j'et} \), and equation 13.12 implies that the lemma holds.

**Lemma 120.** Our setting of the dual variables satisfy all the constraints of LP (13.6).

**Proof.** By Lemma 118, the constraints 13.7, 13.8 are already satisfied. We focus on the remaining constraint 13.9. Fix any job \( j \), any path \( i \in \mathcal{L}_j \), and any edge \( e \in E_i \). By Lemma 119:

\[
\sum_{j' \in \Gamma_{je}} \delta_{ij}(j', t) \leq p_{je} \cdot \sum_{j' \in \Gamma_{je}} 2\lambda_{j'et} \leq p_{je} \cdot z_{et} \tag{13.13}
\]

Under the outcome \((i, \theta_j - j)\), the total delay experienced by \( j \) due to the jobs in \( \Gamma_{je} \) till time-step \( t \) is, by definition, at most \( t \). This leads to the following inequality.

\[
\sum_{j' \in \Gamma_{je}} (\delta_{ij}(j') - \delta_{ij}(j', t)) \leq t \tag{13.14}
\]

From our setting of the dual variables, equation 13.14 and equation 13.13, we see that constraint 13.9 is satisfied.

\[
u_{ije} = p_{je} + \sum_{j' \in \Gamma_{je}} \delta_{ij}(j') \leq p_{je} + t + \sum_{j' \in \Gamma_{je}} \delta_{ij}(j', t) \leq p_{je} + t + p_{je} \cdot z_{et}.
\]

**Lemma 121.** If the dual variables are set as described above, then the objective of LP (13.6) is at least \((1/2) \cdot \sum_j \text{Cost}_j(\theta)\).

**Proof.** Fix the outcome \( \theta \), and focus on any job \( j \) with sojourn-time \( \text{Cost}_j(\theta) \).

(Case 1) \( t \leq \text{Cost}_j(\theta) \). In this case, our definition of the dual variables implies that the job \( j \) contributes 2 to each of the \( z_{et} \)'s corresponding to the critical edges in the path \( \theta_j \), and it makes zero contribution to the remaining \( z_{et} \)'s. Since \( \theta_j \) has at most \( K \) critical edges, the total contribution of the job to the sum \( \sum_e z_{et} \) is at most \( 2K \).
(Case 2) \( t > \text{Cost}_j(\theta) \). Hence in our setting of dual variables the job \( j \) contributes 0 to the sum \( \sum_e z_{et} \).

Summing over all time-steps, the contribution of any single job \( j \) to the sum \( \sum_{e,t} z_{et} \) is at most \( 2K \cdot \text{Cost}_j(\theta) \). Next, summing over the contributions from all the jobs, we infer that \( \sum_{et} z_{et} \leq (2K) \cdot \sum_j \text{Cost}_j(\theta) \). Since \( y_j = \text{Cost}_j(\theta) \) for all jobs \( j \), we get:

\[
\text{LP-objective} = \sum_j y_j - \frac{1}{(4K)} \cdot \sum_{e,t} z_{et} \geq \frac{1}{2} \cdot \sum_j \text{Cost}_j(\theta).
\]

Theorem 112 now follows from Lemma 115, Lemma 120, and Lemma 121.

Interestingly, we get super-constant upper bound on the PoA because the speeds of the edges may keep on decreasing as we traverse farther along a path starting from the root-node. This is precisely the situation in the real-world fat-tree networks. In contrast, consider an instance where the edges adjacent to the root-node have the slowest speeds. In this instance, the proof of Theorem 116 implies that a job can get delayed by other jobs only on the first edge on its path. Thus, we can write a new time-indexed LP where we take into account the fractional completion times of the jobs only on these edges at depth one. This LP will give a constant approximation to the underlying optimization problem, as a path starting from the root contains exactly one edge of depth one. We can then execute the same analysis as outlined in this section to get constant PoA.

**Corollary 122.** If every machine follows Shortest Job First (SJF) policy and every job has unit weight, then the price of anarchy of the induced game is \( O(1) \) when all the machines have same speed.

13.2.1 Remarks and some observations:

A reader may think that one can obtain the constant PoA result for the case of identical machines by reducing the network setting to the classical multiple machine scheduling problem by observing that jobs delay each other only on the first edge of a path. Although
this observation is true, the length of the path itself has an effect on the total processing
done on the jobs. This in turn has an effect on the strategy profile of a job and is the reason
why SJF does not induce games with PoA 1 in tree networks (even with depth two!). On
the other hand, SJF has PoA of 1 in the case of identical machine setting in the classical
scheduling models.

13.3 Summary And Open Problems

In this chapter we gave the first coordination mechanisms for temporal routing games with
non-trivial PoA upperbounds for tree networks. Our work leaves open several interesting
questions: Are there mechanisms which achieve $O(1)$ upperbounds on PoA for trees? What
can be say about the lowerbounds on PoA? We only know a lowerbound of 2, so, it is
possible that there are coordination mechanisms that achieve $O(1)$ PoA.

13.4 Notes

This chapter is based on joint work Sayan Bhattacharya and Vahab Mirrokni. A prelimi-
nary version of this appeared in 41st International Colloquium, ICALP 2014, Copenhagen,
Denmark, July 8-11, 2014, (Bhattacharya et al., 2014a).
14

Robust PoA Bounds Via Duality

14.1 Introduction

Self-interested actions of agents in strategic settings often result in suboptimal equilibrium outcomes. In a seminal work, (Koutsoupias and Papadimitriou, 1999) proposed the concept of \textit{price of anarchy} (PoA) as a formal way to quantify this degradation. For a given social cost function, the PoA of a game is defined as the ratio of the worst-case social cost of an outcome in (pure) Nash equilibrium (NE) to the cost of an optimal solution. The definition can be naturally extended to include outcomes in more general notions of equilibria, such as mixed Nash equilibria (MNE), correlated equilibria (CE) and coarse correlated equilibria (CCE), all of which are strict generalizations of pure NE. In fact, for any game NE \( \subseteq \) MNE \( \subseteq \) CE \( \subseteq \) CCE. The notions of correlated equilibrium and coarse correlated equilibrium have several desirable properties: for any finite game, these equilibria always exist, and can be computed and learned in polynomial time (Gilboa and Zemel, 1989; Blum and Mansour, 2007). This is in contrast to pure NE which may not exist, or MNE which cannot always be efficiently computed.

A variety of techniques have been developed to prove the PoA bounds for a wide range of problems. Of particular interest in this context are the techniques that can be
used to bound the PoA for equilibrium notions such as CE or CCE. The most general among such techniques is the widely used smoothness framework of (Roughgarden, 2009). Roughgarden’s framework distills the essence of many PoA proofs known in the literature and gives a canonical proof template to establish PoA bounds which simultaneously extend to MNE, CE and CCE. Yet, for many games of practical and theoretical interest, such as temporal routing games, it is not clear how to apply the smoothness framework to obtain non-trivial bounds on the PoA.

14.1.1 Our Results

Building upon our ideas from the previous two chapters, here, we present a framework based on the concept of LP and Fenchel duality for establishing the PoA bounds for equilibrium concepts such NE, CCE etc. The main technical idea of the chapter is the following: For a wide class games where the social objective is the sum of costs incurred by players, one can formulate the underlying optimization problem as a linear or convex program such that the dual of the relaxation encodes the equilibrium condition. Further, the dual program has a variable for each player and each resource, which can be interpreted as the cost incurred by the player and utilization of the resource in an equilibrium outcome. Once the variables are defined this way, we appeal to the weak duality theorem to establish the PoA bounds. This is the essence of our proofs in the previous two chapters.

In this chapter, we demonstrate the broad applicability of this dual-fitting technique in two ways. First, we show that many classical PoA results easily follow from our framework. As representative examples, we consider weighted affine congestion games (Awerbuch et al., 2005; Christodoulou and Koutsoupias, 2005b), simultaneous second price auctions (Christodoulou et al., 2008), competitive facility-location games (Vetta, 2002) and give alternate proofs of tightness of the PoA bounds for CCE. Moreover, we show the technique easily extends to bounding the inefficiency for other equilibrium notions such as \(k\)-lookahead equilibrium (Mirrokni et al., 2012) or approximate NE.

Second, we use the dual-fitting framework to get the first PoA bounds for CCE for
some natural generalizations of the games we considered in the last two chapters.

1) Coordination Mechanisms For Temporal Routing Games. We study the temporal routing games in the framework of coordination mechanisms (Christodoulou et al., 2009). Unlike congestion games, which were also studied in the context of selfish routing, the temporal routing games model the queueing nature of routing in real networks and have been an active area of research in recent years (Christodoulou et al., 2011; Koutsoupias and Papakonstantinopoulou, 2012; Azar et al., 2008; Caragiannis, 2009; Cole et al., 2011; Bhattacharya et al., 2014b,a). The specific model we consider is the following: Given a graph $G = (V, E)$ and a set of packets, each packet has a size $p_j$, a weight or priority $w_j$, and wants to travel from some source $h_j \in V$ to some destination $o_j \in V$. The strategy set $S_j$ for a packet consists of a subset of all possible simple paths between vertices $h_j$ to $o_j$. Each edge $e \in E$ has a speed $\nu_e$, a packet $j$ takes $\frac{p_j}{\nu_e}$ units of time to traverse the edge. Our goal is to design local forwarding policies on each edge - which determine the packet which will be forwarded when multiple packets try to use the same edge - which lead to games with small PoA. Given a forwarding policy, each packet chooses the path which gives it the minimum weighted delay or the weighted sojourn time. The social cost is the sum of costs incurred by players. Let $|P_i|$ denote the length of path $P_i$. We first show the following result.

Theorem 123. The PoA anarchy of temporal routing games for coarse correlated equilibrium is at most $4 \cdot D^2$, where $D = \max_{P_i \in \bigcup_j S_j} |P_i|$ is the dilation of $G$, when each edge follows the Highest Density First (HDF) forwarding policy.

This is the first analysis of local forwarding policies for temporal routing games on arbitrary graphs and answers a question raised in (Cole et al., 2011). Our result implies that PoA is independent of number packets, and depends solely on structure of the network. Previous known results for this problem could only handle restricted topologies such as tree networks (Bhattacharya et al., 2014a) or parallel links (Cole et al., 2011; Bhattacharya
Further, our result matches the tight robust PoA bound of 4 for HDF policy obtained by (Cole et al., 2011) using smoothness framework, and generalizes the results in (Bhattacharya et al., 2014b,a) to CCE. Constant-factor approximation algorithms can be inferred from the works of (Kumar et al., 2005; Leighton et al., 1988) for the underlying optimization problem in the offline setting, however, no analysis of local forwarding policies (such as HDF) are known. Interestingly, we also show that the dependence on $D$ may be unavoidable, even for very simple settings.

**Theorem 124.** The price of anarchy of temporal routing games is at least $\Omega(D)$ when each edge uses a priority-based forwarding policy (such as Shortest Job First). This holds even for the special case when all packets have unit size, and same source and destination.

The lower bound is based on a carefully chosen example, and highlights the interplay between short paths and connectivity of graphs in NE outcomes. It also shows that coordination mechanisms with provable PoA are intrinsically harder for general graphs compared to trees. Finally, we show that there are NE with special combinatorial structures for which inefficiency bounds can be improved. This result is achieved by a careful understanding of the dual constraints and proving structural lemmas about a class of equilibrium points.

**Theorem 125.** The price of stability of temporal routing games is at most $4D$ when each edge follows the Shortest Job First forwarding policy, for the case when all packets have the same source and weight.

This result shows that dual fitting is a useful tool for the price of stability analysis and dual constraints can be used to find equilibria with good efficiency. As a corollary of this result, we improve the PoA of CCE for tree topologies considered in the previous chapter from $O(\log^2 \nu)$ to $O(\log \nu)$, where $\nu$ is the ratio of maximum speed of any edge to minimum speed.

2) **Energy Minimization Games in Machine Scheduling.** Minimizing energy consumption is a fundamental problem in wide range of applications - from data center scheduling to mobile
devices with small batteries - and has been an active area of research both in theory and practice (Albers, 2010; Bansal et al., 2007b; Anand et al., 2012; Devanur and Huang, 2014). One of the important theoretical models of this problem in machine scheduling literature is the following. We are given a set of machines and a set jobs. Each job $j$ has a processing length $p_{ij}$ on machine $i$ and a weight $w_{ij}$. Each machine can run at variable speed. If a machine is run at the speed level $s$ it consumes energy $s^\gamma$, where $\gamma$ is some fixed constant (typically 2 or 3 in practice.) The objective is to design scheduling and speed scaling algorithms which minimize the energy consumed while simultaneously guaranteeing certain quality of service (QoS), such as sum of weighted completion times or weighted delays of jobs. One of the most commonly used algorithm for this problem is the following: Each machine schedules the jobs using Highest Density First Scheduling policy (HDF) and chooses the speed level such that total power consumed is equal to the total weight of jobs which have not yet finished. This intuitive algorithm has been shown to perform well both in theory and practice (Bansal et al., 2007b; Anand et al., 2012; Devanur and Huang, 2014).

We take a look at this problem from a game theoretic lens. We treat jobs as selfish agents which choose the machine which gives them the smallest weighted completion time. Again, the social cost is sum of costs incurred by players and our objective is to understand the PoA of the resulting game.

**Theorem 126.** The price of anarchy of energy minimization games for coarse correlated equilibrium is at most $O(\gamma)$.

To the best of our knowledge, this is the first result (along with the affine weighted congestion games proof in this chapter) that uses Fenchel duality to obtain PoA bounds. As noted earlier, tight results are known for the optimization version of this problem (Devanur and Huang, 2014; Anand et al., 2012).
14.2 Dual-fitting Analysis of Congestion Games

All our results are obtained using the dual fitting technique. Instead of explaining the technique in abstract terms, we give a concrete example which captures the essence of our main idea. As a warm-up example, we consider the affine weighted atomic congestion games. All the variants of congestion games are extensively studied in the literature and tight results are known for very general settings (Awerbuch et al., 2005; Christodoulou and Koutsoupias, 2005b; Blum et al., 2008; Christodoulou and Koutsoupias, 2005a; Roughgarden, 2009; Bhawalkar et al., 2010; Aland et al., 2011; Roughgarden and Schoppmann, 2011).

A (slightly) generalized version of this cost minimization game with *unrelated weights* is as follows: consider a set of $\mathcal{N}$ players and a set of $\mathcal{E}$ resources. For each player $j \in \mathcal{N}$, the strategy set $\mathcal{S}_j \subseteq 2^\mathcal{E}$ is given. For the special case of network congestion games, the set of resources correspond to edges of a graph and strategies correspond to source-destination paths in the graph. Here, we will use edges and resources interchangeably. For an outcome $\mathbf{\theta} = (\theta_1, \theta_2, \ldots, \theta_n)$, where each $\theta_j \in \mathcal{S}_j$, the cost incurred by player $j$ is given by the affine cost function $\text{Cost}_j(\mathbf{\theta}) = \sum_{e \in \theta_j} w_{ej} \cdot (a_e \cdot \ell_e(\mathbf{\theta}) + b_e)$, where $\ell_e(\mathbf{\theta})$ is the load on the (resource) edge $e \in \mathcal{E}$. In weighted congestion games, the load on an edge is given by $\ell_e(\mathbf{\theta}) = \sum_{j': e \in \theta_{j'}} w_{e_j'}$, which is simply the total weight of players using that edge. The goal of each player is to minimize the cost incurred, and the social cost is the sum of cost incurred by players. Observe that the social cost, $\text{Cost}(\mathbf{\theta}) = \sum_j \text{Cost}_j(\mathbf{\theta})$, can also be written as

$$\sum_j \sum_{e \in \theta_j} w_{ej} \cdot (a_e \cdot \ell_e(\mathbf{\theta}) + b_e) = \sum_e \ell_e(\mathbf{\theta}) \cdot (a_e \cdot \ell_e(\mathbf{\theta}) + b_e)$$

$$= \sum_e a_e \cdot \ell_e^2(\mathbf{\theta}) + b_e \cdot \ell_e(\mathbf{\theta})$$

Even before the smoothness proof of Roughgarden (2009), the PoA for general equilibrium concepts such as MNE, CCE were obtained in Awerbuch et al. (2005); Christodoulou
and Koutsoupias (2005b); Blum et al. (2008); Christodoulou and Koutsoupias (2005a). We give a new proof of this result using Fenchel duality.

(Note: Although all our proofs are stated in terms of CCE, it is instructive to think of just pure NE to get the main idea.)

Step 1. Convex program and Fenchel dual: For every player $j$ and strategy $i \in S_j$, define $L_{ij} = \sum_{e \in i} w_{ej} \cdot (a_e w_{ej} + b_e)$. $L_{ij}$ is precisely the cost incurred by player $j$ for the strategy $i$, if she were the only player to play the strategy. Consider the following primal convex programming relaxation (Congestion – Primal) for the underlying optimization problem. Here, we have a variable $x_{ij}$ for every player $j$ and for strategy $i \in S_j$ in the strategy space for player $j$, which set to 1 if the player $j$ chooses the strategy $i$. The variable $y_e$ for every edge $e \in E$ indicates the load on edge $e$ in a solution.

$$\min \sum_{j \in N} \sum_{i \in S_j} (x_{ij} \cdot L_{ij}) + \sum_{e \in E} a_e \cdot y_e^2$$

$$s.t. \sum_{i \in S_j} x_{ij} \geq 1 \quad \forall j \in N$$

$$y_e \geq \sum_{j} \sum_{i \in S_j} w_{ej} \cdot x_{ij} \quad \forall e \in E$$

$$x_{ij} \geq 0 \quad \forall j, i \in S_j$$

The first constraint says that every player chooses a strategy. The second constraint enforces that $y_e$ should be at least the load on edge $e$ in any outcome. Consider the objective function: the second term in the objective function is at most the sum of the costs incurred by players, as noted earlier. The first term in the objective function states that the total cost incurred by a player $j$ for playing a strategy is at least the cost of playing the strategy assuming $j$ is the only player in the game. (In fact, without this term the convex program has a large integrality gap. Consider a single player choosing an edge in a graph with $n$ parallel edges. The convex program incurs a cost of $1/n^2$ by choosing each
edge to an extent 1/n, while, in any outcome player has to pay at least 1. On the other hand, this term is not needed for non-atomic games.) From the construction, it is not hard to see that the convex program is a valid relaxation of the problem and an optimal solution to this convex program is at most 2-approximation to the cost of any optimal solution. We lose a factor of 2 here due to the term \( \sum_j \sum_{i \in S_j} (x_{ij} \cdot L_{ij}) \). As explained earlier, the term lowerbounds the total cost incurred players, and is needed to fix the integrality gap of the convex program. We are ready to write the Fenchel dual of Congestion − Primal. For readers not familiar with Fenchel duality, we refer to Devanur and Huang (2014) for an excellent discourse on the topic.

\[
\begin{align*}
\max & \sum_{j \in \mathcal{N}} \alpha_j - \sum_{e \in \mathcal{E}} \frac{1}{4a_e} \cdot \mu_e^2 \\
\text{s.t} & \quad \alpha_j - \sum_{e \in \mathcal{E}} w_{e_j} \cdot \beta_e \leq L_{ij} \quad \forall j \in \mathcal{N}, i \in S_j \\
& \quad \beta_e \leq \mu_e \quad \forall e \in \mathcal{E} \\
& \quad \alpha_j \geq 0, \beta_e \geq 0 \quad \forall j \in \mathcal{N}, \forall e \in \mathcal{E}
\end{align*}
\]

We give a brief explanation on how to derive Fenchel dual of a convex program with linear constraints. As a first step, we ignore the convex part of the objective function and write a dual similar to a LP dual. Now, consider the convex part of objective function. Suppose \( f(y) \) is the convex part of the objective function. Then the dual objective will have the function \(-f^*(\mu)\), where \( f^*(\mu) = \max_y (y \cdot \mu - f(y)) \), is the conjugate of function \( f(y) \). Note that if \( f(y) = y^2 \) then \( f^*(\mu) = \frac{\mu^2}{4} \). Moreover, every variable which occurs in the convex part of the objective function has an extra variable \( \mu \) in the dual which appears in the right hand side of the constraint corresponding to it.

The second step of the proof is to show a setting of dual variables which gives a feasible solution to the dual. Consider any distribution \( \sigma \) over outcomes of the game, where no player can decrease her expected cost using unilateral deviations. Such a distribution \( \sigma \) is a coarse correlated equilibrium of the game. Now we show using a simple dual fitting
argument that for any such distribution the PoA for weighted congestion games is at most $1 + \phi$, where $\phi$ is the golden ratio.

**Step 2. Setting The Dual Variables.** We set the dual variables $\alpha_j$, $\beta_e$, and $\mu_e$ as follows:

- We set $\alpha_j$ to the expected cost incurred by player $j$ in $\sigma$: $\alpha_j = \mathbb{E}_{\theta \sim \sigma}[\text{Cost}_j(\theta)]$.
- We set $\beta_e$ and $\mu_e$ proportional to the expected load on the edge $e$: $\beta_e = \mu_e = a_e \cdot \mathbb{E}_{\theta \sim \sigma}[\ell_e(\theta)]$.

As we shall see, throughout this chapter, we interpret the dual variables corresponding to players as the cost incurred by players in an equilibrium. Similarly, we interpret the dual variables corresponding to resources (or edges) as the congestion on the resource. We encourage the readers to think of pure NE rather than CCE: then $\alpha_j$ is the cost incurred by the player $j$ and $\beta_e$ is the load on edge $e$ in a NE.

**Step 3. Bounding The Dual Objective.** From the definition of dual variables, the term $\sum_e \frac{1}{4a_e} \cdot \mu_e^2$ in the dual objective is equal to $\sum_e \frac{1}{4a_e} \cdot a_e^2 \cdot (\mathbb{E}_{\theta \sim \sigma}[\ell_e(\theta)])^2$, which is at most $1/4$ times the sum of expected cost of players in $\sigma$. This follows from Jensen’s inequality $^1$ and linearity of expectation as shown below.

\[
\sum_e a_e \cdot (\mathbb{E}_{\theta \sim \sigma}[\ell_e(\theta)])^2 \leq \sum_e a_e \cdot \mathbb{E}_{\theta \sim \sigma}[(\ell_e(\theta))^2] \\
\leq \mathbb{E}_{\theta \sim \sigma} \left[ \sum_e a_e \cdot (\ell_e(\theta))^2 \right] \\
\leq \mathbb{E}_{\theta \sim \sigma} \left[ \sum_j \text{Cost}_j(\theta) \right] \\
= \sum_j \mathbb{E}_{\theta \sim \sigma}[\text{Cost}_j(\theta)]
\]

---

$^1$ For any random variable $X$ and any convex function $g$, $g(\mathbb{E}[X]) \leq \mathbb{E}[g(X)]$
Therefore, by the weak duality theorem, the cost of primal optimum is at least

\[
\text{CP-Cost} \geq \sum_{j \in \mathcal{N}} \alpha_j - \sum_{e \in \mathcal{E}} \frac{1}{4a_e} \mu_e^2
\]

\[
\geq \sum_{j \in \mathcal{N}} \mathbb{E}_{\theta \sim \sigma} [\text{Cost}_j(\theta)] - \frac{1}{4} \sum_{j \in \mathcal{N}} \mathbb{E}_{\theta \sim \sigma} [\text{Cost}_j(\theta)]
\]

\[
= \frac{3}{4} \sum_{j} \mathbb{E}_{\theta \sim \sigma} [\text{Cost}_j(\theta)]
\]

Since the primal is at most 2-approximation to the optimal solution, we get a bound of \(8/3\) on the robust PoA. To get the tight bound of \(\frac{3+\sqrt{5}}{2}\), we first scale the term \(\sum_{i,j} (x_{ij} \cdot L_{ij})\) by \(\frac{1}{\lambda}\) (which means, the primal program is \((1+1/\lambda)\)-approximation to opt). Then we scale the dual variables \(\alpha_j, \beta_e\) and \(\mu_e\) also by \(\frac{1}{\lambda}\). Note that dual constraints are continued to be satisfied with this change. A routine calculation shows that this will give a bound of

\((1 + \frac{1}{\lambda}) \cdot (\frac{4\lambda^2}{4\lambda^2 - 1})\) on the robust PoA, which we can optimize to get the tight bound of \(\frac{3+\sqrt{5}}{2}\).

**Step 4. Verifying The Dual Constraints.** We will show that our definition of the dual variables satisfy the dual constraints. Recall that there is a constraint (14.1) for every pair of player \(j\) and strategy \(i \in \mathcal{S}_j\):

\[
\alpha_j \leq L_{ij} + \sum_{e \in i} w_{ej} \cdot \beta_e
\]

Consider a player \(j\) and a strategy \(i \in \mathcal{S}_j\). Let \(\mathbb{E}_{\theta \sim \sigma} [\text{Cost}_j(i, \theta - j)]\) be the expected cost incurred by player \(j\) for playing the strategy \(i\), fixing everyone else’s strategy. From the definition of \(L_{ij}\) we note that for every outcome \(\theta\),

\[
\text{Cost}_j(i, \theta - j) = \sum_{e \in i} w_{ej} \cdot (a_e \cdot \ell_e(\theta) + w_{ej} + b_e)
\]

\[
= L_{ij} + \sum_{e \in i} w_{ej} \cdot a_e \cdot \ell_e(\theta)
\]

232
for all $i$, where $i$ is not the strategy chosen by player $j$ in $\theta$. Similarly, $\text{Cost}_j(i, \theta_{-j}) = \sum_{e \in i} w_{e} \cdot (a_{e} \cdot \ell_{e}(\theta) + b_{e})$, if $i$ is equal to the strategy chosen by player $j$ in outcome $\theta$. In other words, for all outcomes $\theta$ and for all $i$, we have

\[
\text{Cost}_j(i, \theta_{-j}) \leq L_{ij} + \sum_{e \in i} w_{e} \cdot a_{e} \cdot \ell_{e}(\theta)
\]

Therefore, we have $E_{\theta \sim \sigma}[\text{Cost}_j(i, \theta_{-j})] \leq L_{ij} + \sum_{e \in i} w_{e} \cdot \beta_{e}$. This is true, since we set $\beta_{e}$ to $a_{e} \cdot E_{\theta \sim \sigma}[\ell_{e}(\theta)]$.

In order to verify that constraint (14.1) is satisfied, we do the following thought experiment: We first pretend that $\alpha_{j}$ is equal to $E_{\theta \sim \sigma}[\text{Cost}_j(i, \theta_{-j})]$. Then, the definition of dual variables imply that dual constraint (14.1) corresponding to player $j$ and strategy $i$ is satisfied. However, $\alpha_{j}$ is set to $E_{\theta \sim \sigma}[\text{Cost}_j(\theta)]$, and not $E_{\theta \sim \sigma}[\text{Cost}_j(i, \theta_{-j})]$. Now we use the fact that $\sigma$ is a coarse correlated equilibrium and $E_{\theta \sim \sigma}[\text{Cost}_j(\theta)] \leq E_{\theta \sim \sigma}[\text{Cost}_j(i, \theta_{-j})]$. In fact, $E_{\theta \sim \sigma}[\text{Cost}_j(\theta)] = \min_{i \in S_{j}} E_{\theta \sim \sigma}[\text{Cost}_j(i, \theta_{-j})]$ if $\sigma$ is a coarse correlated equilibrium. Therefore, if a constraint is satisfied by setting $\alpha_{j} = E_{\theta \sim \sigma}[\text{Cost}_j(i, \theta_{-j})]$, it is also satisfied by $\alpha_{j} = E_{\theta \sim \sigma}[\text{Cost}_j(\theta)]$. From this perspective, the dual constraints (14.1) encode the equilibrium definition that fixing the strategies of the other players, no player can increase her utility (or decrease cost) by unilateral deviations! From this fact, we will conclude that our setting of dual variables satisfy the dual constraints and we get robust bounds on PoA.

Observe that in order to prove that our setting of the dual variables satisfy the dual constraints, it is enough to verify the following: Consider any outcome $\theta = (\theta_{1}, \theta_{2}, \ldots, \theta_{n})$. We check that the dual constraints (14.1) corresponding to player $j$ and strategy $i \in S_{j}$ are satisfied if we assume that $\alpha_{j} = \text{Cost}_j(i, \theta_{-j})$ and $\beta_{e} = a_{e} \cdot \ell_{e}(\theta)$. Since $\sigma$ is a distribution over outcomes, the constraints will also be satisfied by $\alpha_{j} = E_{\theta \sim \sigma}[\text{Cost}_j(i, \theta_{-j})]$ and $\beta_{e} = a_{e} \cdot E_{\theta \sim \sigma}[\ell_{e}(\theta)]$. Finally, we use the fact that $\sigma$ is a coarse correlated equilibrium and hence $\alpha_{j} = E_{\theta \sim \sigma}[\text{Cost}_j(\theta)]$ satisfies all the dual constraints simultaneously.

Remark 127. • Note that our proof can be easily extended to non-atomic setting to obtain $4/3$ PoA. For this setting we do not need the $L_{ij}$ term in the convex program-
Readers familiar with Roughgarden’s (Roughgarden, 2009) smoothness framework can draw an analogy here. The dual argument replaces the smoothness inequality to verifying the dual constraints. It would be interesting to explore if there is a formal connection between these frameworks.

Next, we give a sketch of the proof to get a bound of $\frac{3+\sqrt{5}}{2} \cdot k^2$ on the inefficiency of for $k$-look ahead equilibria for affine unweighted congestion games. No results were known previously for this case, except when $k = 2$. See (Mirrokni et al., 2012; Bilb et al., 2013) for more details. First we observe that if a strategy profile $\theta = (\theta_1, \theta_2, \ldots, \theta_n)$ is in $k$-lookahead equilibrium, it satisfies the following inequality for every player $j$ and strategy $i \in S_j$:

$$\sum_{e \in i} a_e \cdot (\ell_e(\theta) + k) + b_e \geq \sum_{e \in \theta_j} a_e \cdot (\ell_e(\theta) - \max\{k, \ell_e(\theta)\}) + b_e$$

Note that $\theta_j$ here denotes the strategy of player $j$ in the outcome $\theta$ and $i$ is any strategy in the strategy space of player $j$. We emphasize that the above inequality is only a sufficient condition (and perhaps a weak one). However, this is enough to get the desired bound on PoA.

To show the bound on the PoA, we set $\alpha_j$ to be $\frac{1}{k} \cdot \sum_{e \in \theta_j} a_e \cdot (\ell_e(\theta) - \max\{k, \ell_e(\theta)\}) + b_e$. We set $\beta_e = \mu_e = \frac{1}{k} \cdot a_e \cdot \ell_e(\theta)$. Observe that $\sum_{e \in \theta_j} a_e \cdot (\ell_e(\theta) - \max\{k, \ell_e(\theta)\}) + b_e \geq \frac{1}{k} C_j(\theta)$. Therefore, this setting of dual variables will imply that the cost of primal program is at least $\frac{3+\sqrt{5}}{2k}$ the cost of $\theta$, which gives a bound of $8/3 \cdot k^2$ on the coordination ratio (a term used in (Mirrokni et al., 2012)). To get better the bound, we first scale primal and dual variables by $1/\lambda$ and the first term in the primal objective by $1/\lambda$ to get the coordination ratio as a function of $\lambda$ which we optimize to get the bound of $\frac{3+\sqrt{5}}{2} \cdot k^2$. It is easy to verify that dual constraints are satisfied with our definition of dual variables and this completes
the proof.

14.2.1 More related work

To the best of our knowledge, there are very few instances ((Bhattacharya et al., 2014b,a; Nadav and Roughgarden, 2010; Bilò, 2012; Bilò et al., 2013)) in the literature where dual-fitting or primal-dual ideas are used in proving the PoA bounds and none of them use Fenchel duality. The works of Bhattacharya et al. in ((Bhattacharya et al., 2014b,a)) come closest to the spirit in which we use the technique, however, they do not consider CCE. There are two other works (Nadav and Roughgarden, 2010; Bilò, 2012) in the literature which use primal-dual ideas which need a comparison. The main idea in (Nadav and Roughgarden, 2010; Bilò, 2012) is to write a LP to the process of bounding the PoA itself, whereas we write LP/CP to the underlying optimization problem. In fact, (Nadav and Roughgarden, 2010) study the problem of finding the largest class of equilibrium concept for which smoothness bounds apply. Hence, in these works, variables of LP represent strategies ((Nadav and Roughgarden, 2010)) or payoffs ((Bilò, 2012)) of players in an equilibrium and optimal solution, and constraints of primal enforce the equilibrium definition. On the other hand, our primal programs are independent of the equilibrium concepts and only lower bound the optimal social cost. Although techniques in (Nadav and Roughgarden, 2010; Bilò, 2012) are interesting in their own right, in our opinion, do not easily generalize to more complicated settings (such as temporal routing games or energy minimization games).

14.2.2 Limitations

Although the dual-fitting technique seems to be very useful in bounding the PoA for a wide range of problems, it has some intrinsic limitations. A major drawback of the technique is the integrality gap of the LP/CP relaxations, which impose an implicit restriction on the tightness of the results one can obtain using this technique. Another limitation of the technique, at least as developed in this work, is that it only works when the social cost is a convex function. Nevertheless, we hope that our technique finds more applications, espe-
cially in the context of analyzing games for which underlying optimization problems have nice duality based (approximation) algorithms, such as network design games (Anshelevich et al., 2004).

14.3 More Examples

14.3.1 Simultaneous Second-Price Auctions

In this problem, we are given a set $\mathcal{M}$ of items and a set $\mathcal{N}$ of users. Each user $j \in \mathcal{N}$ has a non-negative valuation $v_j(T)$ for every subset of items $T \subseteq \mathcal{M}$. We assume that valuation functions are submodular. Each user submits a bid $b_{ij}$ for every item $i \in \mathcal{M}$ such that $\sum_{i \in T} b_{ij} \leq v_j(T)$, $\forall T \subseteq \mathcal{M}$. Given a bid vector $b = (b_1, b_2, \ldots, b_n)$, where $b_j = (b_{1j}, b_{2j}, \ldots, b_{mj})$ corresponds to the bid by player $j$ for all the items $i \in \mathcal{M}$, the items are allocated independently using the second-price auction (giving the item to the highest bidder and charging the second highest bid). A strategy for a player $j$ corresponds to a bid $b_j = (b_{1j}, b_{2j}, \ldots, b_{mj})$.

Given a bid profile $b$, let $Z_j(b)$ denote the set of items agent $j$ wins. The payoff for an agent $j$ is defined as $\Delta_j(b) = v_j(Z_j(b)) - \sum_{i \in Z_j(b)} p(i, b)$, where $p(i, b)$ is the second highest bid for the item $i$ in $b$. The objective is to maximize the social welfare, which includes the sum of users payoffs and the revenue from the auctioneer. Let $W(b)$ denote the welfare of the bid vector $b$. Note that $W(b) = \sum_{j \in \mathcal{N}} v_j(Z_j(b))$.

Christodoulou et al. (2008) showed the game has a PoA of at most 2 (which is tight). Roughgarden (2009) reinterpreted this proof using the smoothness framework which automatically implies a bound of 2 for mixed NE, correlated equilibria (CE), and coarse CE. We now give a dual-fitting proof of these results, which matches the these bounds.

**LP Formulation and Dual:** Consider the configuration LP formulation for the problem given below. Here, we have a variable $x_{jT}$ for every subset of items $T \subseteq \mathcal{M}$ and every agent $j \in \mathcal{N}$, which indicates if the set of items $T$ is allocated to the agent $j$. The first constraint ensures that no item is over-allocated, while the second constraint says that each agent receives exactly one set. The objective finds the (fractional) allocation which maximizes
the social welfare.

\[
\max \sum_{T} \sum_{j \in N} x_{jT} \cdot v_j(T) \quad \text{(SSA – Primal)}
\]

s.t. \[
\sum_{j} \sum_{i \in T} x_{jT} \leq 1 \quad \forall i \in M
\]

\[
\sum_{T} x_{jT} \leq 1 \quad \forall j \in N
\]

\[
x_{jT} \geq 0 \quad \forall j, T \subseteq M
\]

Now consider the dual. Here we have a variable \(\alpha_j\) for every player \(j\) and a variable \(\beta_i\) for item \(i\).

\[
\min \sum_{j} \alpha_j + \sum_{i} \beta_i \quad \text{(SSA – Dual)}
\]

s.t. \[
\alpha_j \geq v_j(T) - \sum_{i \in T} \beta_i \quad \forall j, T \subseteq M \quad (14.3)
\]

\[
\alpha_j \geq 0, \beta_i \geq 0 \quad \forall j, i \quad (14.4)
\]

**Setting The Dual Variables:** Given a distribution \(\sigma\) which is in CCE, we set the dual variables in the most natural way.

- We set \(\alpha_j\) to the expected payoff of player \(j\) under the distribution \(\sigma\); \(\alpha_j = \mathbb{E}_{b \sim \sigma}[\Delta_j(b)]\).
- We set \(\beta_i\) to the expected value of highest bid for the item \(i\) in \(\sigma\). That is, \(\beta_i = \mathbb{E}_{b \sim \sigma}[\max_j \{b_{ij}\}]\)

**Bounding The Dual Objective:** Since bids are constrained to satisfy \(\sum_{i \in T} b_{ij} \leq V_j(T)\), for all subsets of items \(T\) and agents \(j\), we have \(\mathbb{E}_{b \sim \sigma}[\max_j \{b_{ij}\}] \leq \mathbb{E}_{b \sim \sigma}[W(b)]\). Therefore, from the weak duality theorem,

\[
\text{LP-cost} \leq \sum_{j} \alpha_j + \sum_{i} \beta_i
\]

\[
= \mathbb{E}_{b \sim \sigma}[\Delta_j(b)] + \mathbb{E}_{b \sim \sigma}[\max_j \{b_{ij}\}]
\]

\[
\leq 2 \cdot \mathbb{E}_{b \sim \sigma}[W(b)]
\]

237
Checking The Dual Constraints: Before we formally prove that the constraints are satisfied, observe that the constraint (14.3) encodes the equilibrium condition. The constraint states that for every outcome \( b \) in NE (or CCE), the payoff of a player \( j \) has to be at least \( v_j(T) - \sum_{i \in T} \beta_i \), \( \forall T \subseteq M \); otherwise, the player has an incentive to deviate from the current strategy (which contradicts the NE/CCE condition).

To show that our setting of dual variables satisfy the dual constraints, we prove the following claim.

**Lemma 128.** For any player \( j \), any subset \( T \subseteq M \) of items, and any bid vector \( b \), there is a bid \( b_j(T) \) for the player \( j \) such that the payoff is at least \( v_j(T) - \sum_{i \in T} \max_{j' \in N \setminus j} \{ b_{ij'} \} \), fixing the bids of other players.

Let \( \Delta_j(b_j(T), b_{-j}) \) denote this payoff of \( j \) in \( (b_j(T), b_{-j}) \). First we discuss why the above lemma implies that our setting of dual variables satisfy the dual constraints. The above claim implies that the dual constraint 14.3 corresponding to agent \( j \) and subset \( T \) is satisfied if \( \alpha_j \) is set to \( E_{b \sim \sigma}[\Delta_j(b_j(T), b_{-j})] \). This is because, for every outcome \( b \) we are guaranteed by the above lemma that

\[
\Delta_j(b_j(T), b_{-j}) \geq v_j(T) - \sum_{i \in T} \max_{j' \in N \setminus j} \{ b_{ij'} \}
\]

However, we note that \( E_{b \sim \sigma}[\Delta_j(b)] \geq E_{b \sim \sigma}[\Delta_j(b_j(T), b_{-j})] \), \( \forall T \subseteq M \), if \( \sigma \) is a coarse correlated equilibrium. Therefore, we conclude that all the dual constraints are satisfied if the lemma is true.

**Proof.** Now we prove the lemma. Proof of the lemma is essentially the idea used in the proof by Christodoulou et al. (2008). Number the items in the set \( T \) from 1, 2, \ldots, \( d \). Consider the following bid \( b_j(T) \) for the player \( j \): set \( b_{ij}^* = (v_j(1, 2, \ldots, i) - \)
\( v_j(1, 2, \ldots, i - 1) \) for item \( i \in T \) and zero for the rest. From the construction, it is true that \( \sum_{i \in T} b_{ij}^* = v_j(T) \) and from the submodularity of valuation function \( \sum_{i \in T'} b_{ij}^* \leq v_j(T'), \forall T' \subseteq M. \) Let \( T^* \subseteq T \) be set of items player \( j \) wins in the bid profile \((b_j(T), b_{-j})\) (Note that using our notation \( T^* = Z_j(b_j(T), b_{-j})\)). Consider,

\[
\Delta_j(b_j(T), b_{-j}) = v_j(T^*) - \sum_{i \in T^*} \max_{j' \in N \setminus j} \{ b_{ij'} \}
\]

\[
geq v_j(T^*) - \sum_{i \in T^*} \max_{j' \in N \setminus j} \{ b_{ij'} \}
\]

\[
+ \sum_{i \in T \setminus T^*} (b_{ij}^* - \max_{j' \in N \setminus j} \{ b_{ij'} \})
\]

\[
geq \sum_{i \in T^*} b_{ij}^* - \sum_{i \in T^*} \max_{j' \in N \setminus j} \{ b_{ij'} \}
\]

\[
+ \sum_{i \in T \setminus T^*} (b_{ij}^* - \max_{j' \in N \setminus j} \{ b_{ij'} \})
\]

\[
geq v_j(T) - \sum_{i \in T} \max_{j' \in N \setminus j} \{ b_{ij'} \}
\]

The first inequality is true from the fact that the right most term is at most zero. The remaining inequalities follow from the construction of the bid \( b_j(T) \). (Note that \( v_j(T^*) \geq \sum_{i \in T^*} b_{ij}^* \) and \( v_j(T) = \sum_{i \in T} b_{ij}^* \).) Therefore, we conclude that PoA of CCE is at most 2. The crux of the entire argument is that the dual constraint 14.3 encodes the equilibrium condition, once we treat the dual variables as the payoff of agents in an equilibrium. Hence, the bound applies to general equilibrium concepts such NE or CCE. \( \square \)

**Remark 129.** The above proof can be modified to show that PoA of first price auction for pure NE is 1, if it exists. In the construction of bid \( b_j(T) \) for the set \( T \), we will use the highest bid in \( b_{-j} \) plus \( \epsilon \) for some vanishingly small \( \epsilon \). But observe that \( b_j(T) \) is dependent on the prices paid in \( b \), hence does not extend to mixed NE or CCE.
14.3.2 Competitive Facility Location Games

In this problem, there is a set of $M$ clients, a set $N$ suppliers and a set of $K$ locations. Each client $i \in M$ needs a service and has a value $\pi_i$ for the service (which can be interpreted as the money she is willing to pay for the service). Each supplier $j \in N$ can provide the service to clients from a subset of locations $S_j \subseteq K$. There is a cost $c_{ik}$ for serving the client $i$ from the location $k$. We assume that $c_{ik} \leq \pi_i$, otherwise no one would serve the client $i$ from the location $k$. Each supplier chooses a single location $k \in S_j$ to set up the facility and offers prices to the clients. Given the prices set by suppliers, the clients choose the service provider who offers the least price (denoted by $p(i, \theta)$).

Hence, the game consists of suppliers choosing the locations and offering the prices. An outcome of this game $\theta = (\theta_1, \theta_2, \ldots, \theta_n)$ consists of supplier $j$ choosing a single location $\theta_j \in S_j$. Let $K(\theta)$ denote the set of locations chosen by the suppliers in the outcome $\theta$; $K(\theta) = \cup_j \{\theta_j\}$. A supplier who opens a facility at location $k$ can serve all the clients $i$ for which the service cost $c_{ik}$ is the least (among $K(\theta)$), and offer a price which is the second cheapest. Let $P_j(i, k, \theta - j)$ denote the profit a supplier can make serving the client $i$ from a location $k \in S_j$. Then,

$$P_j(i, k, \theta - j) = \begin{cases} 
\left( \min_{k' \in \{K(\theta) \setminus \theta_j\}} c_{ik'} \right) - c_{ik} & \text{if } c_{ik} \leq c_{ik'} \\
0 & \text{otherwise}
\end{cases}$$

Therefore, in the outcome $\theta$ each supplier $j$ serves the client $i$ if $c_{i\theta_j}$ has the least service cost and offers the second cheapest cost. Our objective is to maximize the social welfare, denoted by $W(\theta)$, which is equal to the sum of profit made by each supplier and the savings by each client. The savings made by a client $D_i(\theta)$ is equal to the difference between the value $\pi_i$ the client has for the service and the actual price paid by the client: $D_i(\theta) = \pi_i - p(i, \theta)$.

Vetta (2002) first established a tight bound of 2 for the PoA of pure NE in more general settings. This result was later extended to other equilibrium notions by Blum et al. (2008).
Roughgarden (2009) showed how to recast these arguments in the smoothness framework to obtain matching bounds. We give a new proof of these results using duality.

**LP relaxation and Dual.** We first write a LP relaxation for the optimization version of the problem, as shown below. Here, we have a variable $x_{jk}$ for every supplier $j \in \mathcal{N}$ and location $k \in \mathcal{S}_j$, which in the integral solution indicates if the supplier opens a facility at location $k$. The variable $x_{ijk}$ indicates if the supplier $j$ serves the client $i$ from the location $k \in \mathcal{S}_j$. The objective function measures the profit made by supplier $j$. Note that in the optimization version of the problem, there is no competition among the suppliers so each supplier can offer a price which is equal to the cost of serving the client $i$ from location $k$ and hence makes a profit of $(\pi_i - c_{ik})$.

Let us understand the constraints. The first constraint says that each client is served by at most one supplier (from some location). The second and third constraints enforce that at most one supplier opens the facility at each location and each supplier chooses at most one location respectively. The last constraint ensures that no supplier can serve the client $i$ from the location $k$ without opening the facility at $k$.

\[
\begin{align*}
\text{max} & \sum_j \sum_{k \in \mathcal{S}_j} \sum_i (\pi_i - c_{ik}) x_{ijk} & \quad \text{(Facility - Primal)} \\
\text{s.t.} & \sum_j \sum_k x_{ijk} \leq 1 & \forall i \in \mathcal{M} \\
& \sum_j x_{jk} \leq 1 & \forall k \in \mathcal{K} \\
& \sum_{k \in \mathcal{S}_j} x_{jk} \leq 1 & \forall j \in \mathcal{N} \\
& x_{ijk} \leq x_{jk} & \forall i \in \mathcal{M}, j \in \mathcal{N}, k \in \mathcal{S}_j \\
& x_{ijt} \geq 0 & \forall i, j, t
\end{align*}
\]

Now consider the dual of the LP. Here, we have a variable $\alpha_j$ for each supplier, a
variable $\beta_i$ for each client, a variable $\gamma_k$ for each location $k$ and a variable $z_{ijk}$ for each (client, supplier, location) triplet.

\[
\min \sum_{j \in N} \alpha_j + \sum_{i \in M} \beta_i + \sum_{k \in K} \gamma_k \quad \text{(Facility - Dual)}
\]

\[
\text{s.t.} \quad \beta_i + z_{ijk} \geq \pi_i - c_{ik} \quad \forall i \in M, j \in \mathcal{N}, k \in \mathcal{S}_j \quad (14.5)
\]

\[
\gamma_k + \alpha_j \geq \sum_{i \in M} z_{ijk} \quad \forall k \in \mathcal{S}_j, j \in \mathcal{N} \quad (14.6)
\]

\[
\alpha_j, \beta_i, \geq 0
\]

Observe that we see the pattern repeat: in the dual we get variables for each player, and each resource and a constraint (14.6) which beautifully encodes the equilibrium condition.

**Setting The Dual Variables.** For an outcome $\theta$, let $P_j(\theta)$ denote the profit made by the supplier $j \in \mathcal{N}$ and let $\rho_j(\theta)$ denote the set of clients the supplier $j$ serves. Recall that $P_j(i, k, \theta_{-j})$ denotes the profit the supplier $j$ can make from location $k$ serving the client $i$, fixing the strategies of other players. In this notation, $P_j(\theta) = \sum_{i \in \rho_j(\theta)} P_j(i, \theta_j, \theta_{-j})$. Recall that the savings made by client $i$ is $D_i(\theta) = \pi_i - p(i, \theta)$, where $p(i, \theta)$ denotes the price paid by client $i$ in the outcome $\theta$.

Given a distribution $\sigma$ in a coarse correlated equilibrium, we set the dual variables as follows.

- We set $\alpha_j$ to the expected profit made by the supplier $j$ under the distribution $\sigma$. That is, $\alpha_j = \mathbb{E}_{\theta \sim \sigma}[P_j(\theta)]$.

- We set $\beta_i$ to the expected savings made by the client $i$ under the distribution $\sigma$. So, $\beta_i = \mathbb{E}_{\theta \sim \sigma}[D_i(\theta)]$.

- We set $z_{ijk}$ to the expected profit the agent $j$ would make if she serves the client $i$ from the location $k$, fixing the strategies of other players. For an outcome $\theta$, recall
that \( P_j(i, k, \theta_j) \) denotes the profit made by the supplier \( j \) from the location \( k \) serving the client \( i \). Then, we set 
\[
z_{ijk} = \mathbb{E}_{\theta \sim \sigma}[P_j(i, k, \theta_j)].
\]

- Lastly, we set \( \gamma_k \) as the expected profit collected by the agents serving from the location \( k \) in the distribution \( \sigma \). For an outcome \( \theta \), define 
\[
\xi_k(\theta) = \begin{cases} 
P_j(\theta), & \text{iff } k \in K(\theta), \text{ and } \theta_j = k \\ 0, & \text{otherwise} \end{cases}
\]

Then, \( \gamma_k = \mathbb{E}_{\theta \sim \sigma}[\xi_k(\theta)] \).

**Bounding The Dual Objective.** From the definition of dual variables, \( \sum_{j \in N} \alpha_j + \sum_{i \in M} \beta_i \) is the expected social welfare under the distribution \( \sigma \). Further, \( \sum_{k \in K} \gamma_k \) is at most the social welfare. Therefore, from the weak duality theorem,

\[
\text{LP-Cost} \leq \sum_{j \in N} \alpha_j + \sum_{i \in M} \beta_i + \sum_{k \in K} \gamma_k \leq 2 \cdot \mathbb{E}_{\theta \sim \sigma}[W(\theta)]
\]

Thus we get a bound of 2 on the PoA for CCE if the dual constraints are satisfied.

**Checking The Dual Constraints.** We show that dual constraints are satisfied in a pure NE \( \theta \). The proof for CCE remains exactly same except that we need to argue in terms of expectations. From our definition of the dual variables \( \beta_i \) and \( z_{ijk} \), the constraints of type (14.5) are trivially satisfied. Note that the constraint is tight for the locations \( k, k' \) which have the first and second cheapest service cost to the client \( i \). For the remaining locations \( k \), we set \( z_{ijk} \) as zero, so \( \beta_i = \pi_i - p(i, \theta) \) is greater than \( \pi_i - c_{ik} \).

Now consider the constraints of type (14.6). Note that there is one constraint for each agent \( j \) and location \( i \). These constraints encode the equilibrium conditions. Fix an agent \( j \). We consider two cases.

**Case 1.** \( k \in K(\theta) \): In this case note that \( \gamma_k = \sum_{i \in M} z_{ijk} \), since we set \( z_{ijk} = P_j(i, k, \theta_j) \) (which is equal to the profit agent \( j \) can make from location \( k \), fixing the
strategies of others.) Note that in the outcome \( \theta \), the agent \( j \) may have chosen some other location. However, since \( k \in K(\theta) \), for some agent \( j' \in N \), \( \theta_j = k' \) and hence, 
\[
\gamma_k = P_j'(\theta) = \sum_{i \in M} z_{ijk}.
\]
Therefore, constraints are satisfied.

Case 2. \( k \notin K(\theta) \): Since \( \theta \) is a NE, it is true that \( \alpha_j \geq \sum_{i \in M} z_{ijk} \). Otherwise, agent \( j \) can switch to location \( k \) and increase his profit under the outcome \( \theta \). Therefore, constraints are satisfied.

This completes the proof. In all our proofs the dual constraints are satisfied precisely for the reason that outcomes are in a certain equilibrium. We will see more instances of this in future sections.

14.4 Coordination Mechanisms For Temporal Routing Over Graphs

In this problem, we are given a graph \( G = (V, E) \) and a set \( N \) of \( n \) packets. Each packet \( j \in N \) has a size of \( p_j \), a weight \( w_j \) and wants to travel from some source vertex \( h_j \in V \) to some destination vertex \( o_j \in V \). Each edge \( e \in E \) has a speed \( \nu_e \), which models the bandwidth or processing power of the edge \( e \). Hence, it takes \( p_j/\nu_e \) units of time to forward a packet \( j \) on edge \( e \). For each packet \( j \), we are also given the strategy space \( S_j \), which is a subset of all possible simple paths between the vertices \( h_j \) and \( o_j \). Each packet \( j \) selects a path \( P_i = (e_1, e_2, \ldots e_l) \), \( P_i \in S_j \), that begins at the source node \( h_j \) and ends at the destination node \( o_j \). Furthermore, packets can start getting processed on an edge \( e_k, k \in \{2, \ldots l\} \), only after it is processed completely by the preceding edge \( e_{k-1} \). A packet exits the graph when it gets completely processed on the last edge \( e_l \), which is incident on the destination vertex \( o_j \). The time it takes for a packet to travel from the source to the destination is called sojourn time of the packet. The weighted sojourn time of \( j \) is equal to its weight \( w_j \) times its sojourn time. We adopt the store-and-forward routing model (Leighton et al., 1988). In this model, when multiple packets want to use the same edge at the same time, a forwarding policy determines which packet goes first and rest of the packets are queued up. Moreover, we allow the forwarding policy to be preemptive in the sense that a packet being forwarded may be preempted by another packet and can be
resumed at a later point of time.

An outcome \( \theta = (\theta_1, \theta_2, \ldots, \theta_n) \) of this game consist of each packet choosing a path \( \theta_j \in S_j \). The cost incurred by a packet is the weighted sojourn time of the packet. In other words, if a packet \( j \) reaches the destination \( o_j \) at time \( C_j(\theta) \) in the outcome \( \theta \), then \( \text{Cost}_j(\theta) = w_j \cdot C_j(\theta) \). Given the forwarding policy on each edge, each packet strategically chooses the path which gives it the smallest weighted sojourn time. This induces a game among the packets and we want to understand efficiency of the outcomes in some equilibrium, such as NE or CCE. The social cost is the sum of player costs:

\[
\text{Cost}(\theta) = \sum_{j \in N} \text{Cost}_j(\theta)
\]

Recall that an outcome \( \theta \) is in pure NE if for every \( j \in N \) and for all \( P_i \in S_j \), \( \text{Cost}_j(\theta) \leq \text{Cost}_j(i, \theta_{-j}) \). Similarly, a distribution \( \sigma \) over outcomes of the game is a coarse correlated equilibrium if for all \( j \in N \), and for all \( P_i \in S_j \), \( E_{\theta \sim \sigma} [\text{Cost}_j(\theta)] \leq E_{\theta \sim \sigma} [\text{Cost}_j(i, \theta_{-j})] \). The robust PoA is the worst case ratio of expected social cost of a distribution in CCE to the optimal solution to the instance (in non-strategic settings). Let OPT denote the optimal solution to the instance \((G, N, \cup_j S_j)\). Then, robust PoA = \( \max_{\sigma} \frac{\sum_{j} E_{\theta \sim \sigma} [\text{Cost}_j(\theta)]}{\text{OPT}} \).

### 14.4.1 Lower bound on the PoA

We first show a lower bound on the PoA for priority based forwarding policies. The lower bound holds even for the special case when all the edges have same speed and each packet is of unit length. Moreover, in our lower bound instance each packet has the same source and destination.

**Definition 130.** A forwarding policy on an edge is a priority based policy, if for any two packets \( j, j' \in N \) competing to use the same edge at the same time, the policy always forwards \( j \) before \( j' \).

Forwarding policies such as Shortest Job First, Highest Weight First etc are priority based forwarding policies where as First In First Out, Shortest Remaining Length First are
not priority based. It is already known policies such as FIFO, proportional sharing etc lead to games with PoA which depends on $D$ (Bhattacharya et al., 2014a). Thus, forwarding policies which can lead to more efficient outcomes can be very complicated and difficult to analyze.

Let $S = \bigcup j S_j$ denote the union of all the strategy spaces of the users. Let $D = \max_{i \in S} |P_i|$, denote the dilation of $G$.

**Figure 14.1:** The bad path $P_\ell$ is shown in thick edges. Observe that the bad path is a shortest path.

**Proof of Theorem 124**

**Proof.** (Refer Figure 14.1). Consider the following graph $G$: $G$ consists of $\sqrt{n}/2$ edge disjoint paths $P_1, P_2, \ldots P_k$ where $k = \sqrt{n}/2$, each of length $\sqrt{n} + 1$, between vertices $h$ and $o$. Let $P_i := \{h, v_2(i), \ldots v_{\sqrt{n}+1}(i), o\}$ be the set of vertices on the path $P_i$. There is a *bad* short path $P_\ell$ in $G$ which intersects the paths $P_i, \forall i \in [k]$ at the $(2i - 1)$-th edge. Since $P_\ell$ is the shortest path between source and destination in the graph, it is also a shortest path to the every vertex on it. The bad path is defined $P_\ell$ as follows:

$$P_\ell := \{h, v_2(1), v_3(2), v_4(2), \ldots v_{2i-2}(i - 1), v_{2i-1}(i), \ldots v_{\sqrt{n}-1}(\sqrt{n}/2), v_{\sqrt{n}}(\sqrt{n}/2), o\}$$

The bad path intersects the path $P_i$ at the edge connecting vertices $(v_{2i-1}(i), v_{2i}(i))$ ($(2i - 1)$-the edge) and connects the vertices $(v_{2i}(i), (v_{2(i+1)-1}(i + 1))$ on the paths $P_i$. 

246
and $P_{i+1}$. Finally, there is an edge from the vertex $v_{\sqrt{n}-1}(\sqrt{n}/2)$ to the destination $o$. Therefore, length of the bad path is $\sqrt{n}$. Note that for every path $P_i$, the segment of the path from the vertex $v_{2i}(i)$ to $o$ has length greater than the length of the bad path from vertex $v_{2i}(i)$ to $o$.

There is a set of $n$ packets, each of which want to go from source $h$ to destination $o$. The strategy space for each packet consists of set of all simple paths from $h$ to $o$. Consider any priority based forwarding policy, which given any 2 packets $j, j'$ always forwards $j$. This implies that there is a strict ordering on the packets and we renumber the packets from $1, 2, \ldots n$ using these priorities. Consider the following outcome where all the packets take the bad path $P_\ell$. We argue by induction that this is a NE. The first packet takes $P_\ell$, which from the construction is the shortest path between $h$ and $o$, and hence it is in NE. Now consider $j$-th packet, for some $1 < j < n$. Note that $j$-th packet can delayed only by packets in the set $\{1, 2, \ldots j - 1\}$. Suppose $j$-th packet takes a path $P_i \neq P_\ell$. Then, it will be delayed by the packets $\{1, 2, \ldots j - 1\}$ at the edge where it intersects the path $P_\ell$ (connecting vertices $v_{2i-1}$ and $v_{2i}$). This is because, the length of path $P_\ell$ from $h$ to $v_{2i-1}$ is equal to the length of the path $P_i$ from $h$ to $v_{2i-1}$ (Recall that the bad path is also a shortest path). Therefore, the $j$-th packet is delayed by all the $j - 1$ packets. Observe that once the $(j - 1)$-th packet crosses the edge connecting $v_{2i-1}$ to $v_{2i}$, $j$-th packet can traverse without any further delays. Therefore, for any choice of path by packet $j$, the total sojourn time of packet $j$ is equal to $(j - 1) + L$, where $L$ is the length of the path chosen by $j$-th packet. On the other hand, if $j$-th packet takes the path $P_\ell$, its sojourn time is $|P_\ell| + (j - 1)$. Since $P_\ell$ is the shortest path, $j$-th packet is in NE. We conclude that all the packets taking $P_\ell$ is a NE.

**Bounding the cost of bad NE.** The social cost of this NE is $n \cdot \sqrt{n} + \sum_{j}^{n} j$, which is at least $\frac{n^2}{2}$. On the other hand, if $2\sqrt{n}$ packets are forwarded on each of the paths $P_i$ separately, then the social cost is $\sqrt{n}/2 \cdot (2\sqrt{n} \cdot (\sqrt{n} + 1) + \sum_{j=1}^{n} j)$, which is at most $O(n\sqrt{n})$. Hence, the PoA is at least $\sqrt{n}/8$ which is at least $D/16$.  

247
Remark 131. • Note that in our lower bound example, the length of the shortest path is also \( \sqrt{n} \). That is, \( \min_{i \in S} |P_i| = \sqrt{n} \). Hence for single source, single destination case one may obtain a bound on PoA as a function of shortest path. However, one can easily change the above instance such that \( \min_{i \in S} |P_i| \) is small if multiple destinations are allowed.

• The lowerbound example easily shows why Shortest Job First, Shortest Remaining Processing Time policies also fail. We simply change the lengths of packets to \( 1 + \epsilon_j \), where \( 0 < \epsilon_j < 1 \) is an arbitrary small constant.

• The example also brings out the intrinsic difficulty of selfish routing over graphs. Note that if one removes the bad path, then PoA of routing game is at most 4. This follows from the work of (Bhattacharya et al., 2014a).

14.4.2 Robust PoA

In this section, we prove the PoA of CCE to be at most \( 4 \cdot D^2 \) for the temporal routing games with arbitrary sources and destinations and packets with weights, when each edge follows the Highest Density First (HDF) forwarding policy. For a given packet \( j \), the density of the packet is defined as the ratio of weight over processing length. That is, \( d_{ij} = \frac{w_j}{p_j} \). Let \( h_j, o_j \) denote the source and destination vertices of the packet \( j \). The strategy space for a packet \( j \) denoted by \( S_j \) consists of a subset of all possible simple paths between \( h_j \) to \( o_j \).

Note that \( S_j \) need not include all possible simple paths between \( (h_j, o_j) \).

LP relaxation and Dual: We formulate the optimization version of the routing problem as a LP. Let \( p_{ej} = \frac{p_j}{s_e} \). Consider the LP relaxation for the problem WeightedRouting – Primal given below. This relaxation is a generalization of the time indexed LP relaxation given in (Anand et al., 2012; Bhattacharya et al., 2014a). Here, we have a variable \( x_{eijt} \) which indicates if the packet \( j \) is being forwarded on the edge \( e \) at time \( t \) if it chooses the path \( P_i \in S_j \).
\[
\min \sum_{j} \sum_{P_i \in S_j} \sum_{e \in P_i} \sum_{t} w_{eijt} \cdot \left( \frac{t}{p_{eij}} + 1 \right) \quad \text{(WeightedRouting – Primal)}
\]

s.t. \[
\sum_{P_i \in S_j} x_{ij} \geq 1 \quad \forall j \in \mathcal{N}
\]

\[
\sum_{t} \frac{x_{eijt}}{p_{eij}} \geq x_{ij} \quad \forall e \in P_i, j \quad \text{and} \quad P_i \in S_j
\]

\[
\sum_{j} \sum_{P_i \in S_j} x_{eijt} \leq \frac{1}{2D} \quad \forall e, t
\]

\[
x_{eijt} \geq 0 \quad \forall e, i, j, t
\]

Let us understand the constraints first. The first constraint says that every packet chooses a path, while the second constraint enforces that each packet if it chooses path \(P_i\), then on every edge along the path the packet should be forwarded for at least \(p_{eij}\) time units. Lastly, the third constraint says that at most \(1/2 \cdot D\) units (instead of 1) of packets can be forwarded at any time step. This slows down the LP schedule by a factor of \(1/2 \cdot D\).

By standard time-stretching arguments (see (Bhattacharya et al., 2014a) for more details) we note that this restriction on the LP solution only increases the cost of the solution by a factor of \(2 \cdot D\).

Lemma 132. **WeightedRouting – Primal is a \(2 \cdot D^2\)-approximation to the cost of optimal solution.**

**Proof.** We sketch an outline of the proof, see (Bhattacharya et al., 2014a) for more details. Fix a packet \(j\). Consider the term \(\sum_{P_i \in S_j} \sum_{e \in P_i} \sum_{t} x_{eijt}\) in the objective function (This corresponds to +1 term in the objective). The term counts the total units of time packet \(j\) is forwarded on all the edges and is clearly a lower bound on the sojourn time of packet \(j\). Note that this term does not include the delay a packet may suffer at the edges along its chosen path. This will be accounted by the term \(\sum_{P_i \in S_j} \sum_{e \in P_i} \sum_{t} x_{eijt} \cdot \frac{t}{p_{eij}}\). To understand this quantity, fix an edge \(e \in P_i\). We imagine the packet \(j\) as being made of \(p_{eij}\) unit length
packets. Then, \( \sum_t x_{eij} \cdot \frac{t}{p_{ej}} \) indicates the average of the time instants at which these unit length packets are being forwarded and is called fractional completion time of the packet \( j \).

Note that, if a packet is continuously forwarded in the interval \([t, t + p_{ej}]\), then the fractional completion time on \( e \) will be \( t + p_{ej}/2 \); on the other hand, the integral completion time will be \( t + p_{ej} \). Therefore, \( \sum_t x_{eij} \cdot t \) gives a lower bound on the completion time of \( j \) on \( e \).

Since the completion time of a packet on any edge is a lower bound on the sojourn time of the packet \( j \), \( \sum_j \sum_{P_i \in \mathcal{S}_j} \sum_{e \in P_i} \sum_t w_j \cdot x_{eij} \cdot \frac{t}{p_{ej}} \) is at most \( D \) times the cost incurred by \( j \). Further, we loose a factor \( 2 \cdot D \) due to the second constraint of \( \text{WeightedRouting} - \text{Primal} \). Therefore, we conclude that the LP is a \( 2 \cdot D \)-approximation to the optimal cost.

Now we write the dual. In \( \text{WeightedRouting} - \text{Dual} \), there is a variable \( \alpha_j \) for every packet \( j \), a variable \( \beta_{et} \) for every edge \( e \in E \) and time instant \( t \). There is a variable \( \vartheta_{eij} \) for every packet \( j \), edge \( e \) and the path \( P_i \in \mathcal{S}_j \).

\[
\max \sum_j \alpha_j - \frac{1}{2D} \sum_{e,t} \beta_{et} \quad \text{(WeightedRouting - Dual)}
\]

\[
s.t. \quad \alpha_j - \sum_{e \in P_i} \vartheta_{eij} \leq 0 \quad \forall j, P_i \in \mathcal{S}_j \quad (14.7)
\]

\[
\frac{\vartheta_{eij} - \beta_{et}}{p_{ej}} \leq w_j \cdot \frac{t}{p_{ej}} + w_j \quad \forall e, j, P_i \in \mathcal{S}_j, t \quad (14.8)
\]

\[
\alpha_j, \beta_{et}, \vartheta_{eij} \geq 0
\]

**Setting The Dual Variables:**

Fix an outcome \( \theta \). Let \( C_{ej}(i, \theta_{-j}), a_{ej}(i, \theta_{-j}) \) denote the completion time and arrival time of the packet \( j \) on the edge \( e \) if it takes the path \( P_i \in \mathcal{S}_j \), and rest of the packets follow paths in \( \theta_{-j} \). Let \( \delta_{eij}(i, \theta_{-j}) = C_{ej}(i, \theta_{-j}) - a_{ej}(i, \theta_{-j}) \), denote the total time the packet \( j \) waits at the edge \( e \). Note that \( \delta_{eij}(i, \theta_{-j}) \) may be different for different paths even if they share the same edge. This is because, depending upon the path a packet chooses it may arrive at the same edge at different times. Finally, let \( \text{Cost}_j(i, \theta_{-j}) = \sum_{e \in P_i} w_j \cdot \delta_{eij}(i, \theta_{-j}) \).

Note that in our notation, \( \text{Cost}_j(\theta) \) is simply \( \text{Cost}_j(\theta_j, \theta_{-j}) \).

250
For any time instant $t$ and edge $e$, let $z_{et}(\theta)$ denote the total weight of all the packets which use the edge $e$ (at some point of time) in the outcome $\theta$ and which are alive; i.e., the packets which have not reached the destination by time $t$. Let $C_j(\theta)$ denote the sojourn time of packet $j$. Then, $z_{et}(\theta) = \sum_{j: e \in \theta_j, t \leq C_j(\theta)} w_j$.

Note that $\text{Cost}_j(\theta)$ is also $w_j \cdot C_j(\theta)$. Let $\sigma$ be any distribution over the outcomes which is in coarse correlated equilibrium. We now set the dual variables.

- We set $\alpha_j$ to the expected cost of the packet $j$ in the distribution $\sigma$. That is, $\alpha_j = \mathbb{E}_{\theta \sim \sigma}[\text{Cost}_j(\theta)] = \mathbb{E}_{\theta \sim \sigma}[w_j \cdot C_j(\theta)]$.

- We set $\vartheta_{eij}$ to the expected weighted delay seen by the packet $j$ on the edge $e$, if $j$ chooses the path $P_i$ and rest of the packets follow the distribution $\sigma$. More precisely, $\vartheta_{eij} = \mathbb{E}_{\theta \sim \sigma}[w_j \cdot \delta_{ej}(i, \theta - j)]$

- We set $\beta_{et}$ to the expected total weight of alive packets at time $t$ which use the edge $e$ in $\sigma$; Formally, $\beta_{et} = \mathbb{E}_{\theta \sim \sigma}[z_{et}(\theta)]$.

Bounding The Dual Objective:

**Lemma 133.** If the dual variables are set as defined above, then the cost of $\text{WeightedRouting} - \text{Primal}$ is at least $1/2 \cdot \sum_j \mathbb{E}_{\theta \sim \sigma}[\text{Cost}_j(\theta)]$.

**Proof.** Consider an outcome $\theta$. For every time instant $t \in [0, C_j(\theta)]$, the packet $j$ contributes $w_j$ to $z_{et}$ for each edge $e$ in the path $\theta_j$. Observe that the total cost incurred by packet $j$ can be expressed as $\text{Cost}_j(\theta) = \sum_{t=0}^{C_j(\theta)} w_j$. Therefore, the total contribution by a packet $j$ to the term $\sum_{e \in \theta_j} \sum_{t} z_{et}$ is at most the $D \cdot w_j \cdot \text{Cost}_j(\theta)$, since $D$ is at least the length of the path $j$ chooses. Therefore, summing over all packets we have $\sum_{e} \sum_{t} z_{et} \leq D \cdot \sum_j \text{Cost}_j(\theta)$. Hence, $\sum_{e, t} \beta_{et} = \sum_{e, t} \mathbb{E}_{\theta \sim \sigma}[z_{et}(\theta)] \leq D \cdot \sum_j \mathbb{E}_{\theta \sim \sigma}[\text{Cost}_j(\theta)]$. Therefore, from the weak duality theorem
\[ \text{LP-Cost} \geq \sum_{j \in \mathcal{N}} \alpha_j - \frac{1}{2} D \cdot \sum_{e \in \mathcal{E}} \sum_{t} \beta_{et} \]
\[ = \frac{1}{2} \sum_{j \in \mathcal{N}} \mathbb{E}_{\theta \sim \sigma} (\text{Cost}_j(\theta)) \]

**Verifying The Dual Constraints:** Now we prove that for every packet \( j \in \mathcal{N} \) and every strategy \( P_i \in \mathcal{S}_j \), the constraints 14.7 and 14.8 are satisfied if \( \sigma \) is a coarse correlated equilibrium. Fix a packet \( j \), a path \( P_i \in \mathcal{S}_j \). To show that our definition of dual variables satisfy the constraints, it is enough if we verify the following: for every outcome \( \theta \) the constraints corresponding to packet \( j \) and path \( P_i \in \mathcal{S}_j \) are satisfied if we set \( \alpha_j = \text{Cost}_j(i, \theta_{-j}), \vartheta_{eij} = w_j \cdot \delta_{ej}(i, \theta_{-j}) \) and \( \beta_{et} = z_{et}(\theta) \). It is easy to see this for the constraint 14.8 as we set \( \vartheta_{eij}, \beta_{et} \) to the expected value of \( \delta_{ej}(i, \theta_{-j}), z_{et}(\theta) \) respectively. To show that constraints 14.7 will also be satisfied, we make use of the fact that \( \sigma \) is a coarse correlated equilibrium. Therefore, \( \alpha_j = \mathbb{E}_{\theta \sim \sigma} [\text{Cost}_j(\theta)] \leq \mathbb{E}_{\theta \sim \sigma} [\text{Cost}_j(i, \theta_{-j})] \forall P_i \in \mathcal{S}_j \).

Thus, we focus on showing that \( \alpha_j = \text{Cost}_j(i, \theta_{-j}), \vartheta_{eij} = \delta_{ej}(i, \theta_{-j}) \) and \( \beta_{et} = z_{et}(\theta) \) satisfy constraints 14.7 and 14.8 for every outcome \( \theta \). Clearly, constraint 14.7 is satisfied from the definition. Consider the constraint 14.8.

We will have two cases: 1) The constraint 14.8 corresponding to the path \( P_i \) which the packet \( j \) took in the outcome \( \theta \). 2) The constraints corresponding to paths \( P_i \neq \theta_j \). We focus on the second case since it is more tricky to prove and the proof subsumes the proof of first case.

The tricky part of showing constraints (14.8) are satisfied for the second case is that, we set \( \vartheta_{eij} = w_j \cdot \delta_{ej}(i, \theta_{-j}) \), which is the weighted delay seen by \( j \) on \( e \) in the hypothetical event that \( j \) chooses the path \( P_i \) while other players play \( \theta_{-j} \). On the other hand, the variables \( \beta_{et} \) correspond to the actual outcome \( \theta \). Note that \( \beta_{et} \) values may be very different in the outcomes \( \theta \) and \((\theta_j = i, \theta_{-j})\). However, as each edge follows HDF forwarding policy, a
packet $j$ gets delayed only by packets with density greater than the density of $j$. And these packets traverse the network oblivious to the path taken by the packet $j$. (This property is not true for non-priority based forwarding policies, which is a major hurdle in analyzing them.) Hence, their contribution to $\beta_{et}$ remains unaffected. We now argue that this is enough to show that dual constraints are satisfied.

Let $\mathcal{N}(e,j,i,\theta_{-j})$ denote the set of packets which delay packet $j$ at the edge $e$, when $j$ chooses the path $P_i$. Observe that some of these packets may be partially processed on the edge $e$ when the packet $j$ arrives. Let $q_{ej'}$ denote the remaining processing size of the packet $j' \in \mathcal{N}(e,j,i,\theta_{-j})$ when $j$ arrives at the edge $e$. From the definition of $\delta_{ej}(i,\theta_{-j})$ we have

$$\delta_{ej}(i,\theta_{-j}) = (p_{ej} + \sum_{j' \in \mathcal{N}(e,j,i,\theta_{-j})} q_{ej'})$$

(14.9)

Since the packets in $\mathcal{N}(e,j,i,\theta_{-j})$ delay $j$ at $e$, they are still alive and contribute their weight to $z_{et}$. Therefore, at time $t = 0$,

$$z_{et} \geq \sum_{j' \in \mathcal{N}(e,j,i,\theta_{-j})} w_{j'}$$

$$= \sum_{j' \in \mathcal{N}(e,j,i,\theta_{-j})} \frac{w_{j'}}{p_{ej'}} \cdot p_{ej'}$$

$$\geq \frac{w_j}{p_{ej}} \cdot \left( \sum_{j' \in \mathcal{N}(e,j,i,\theta_{-j})} p_{ej'} \right)$$

The last inequality follows from the fact that each packet in the set $\mathcal{N}(e,j,i,\theta_{-j})$ has a density greater than the density of packet $j$. Since in the interval $[0,t']$ at most $t'$ units of packets can be processed, we have
\[ z_{et'} \geq \frac{w_j}{p_{ej}} \left( \sum_{j' \in N(e,j,i,\theta_{-j})} (p_{ej'} - t') \right) \quad (14.10) \]

Fix a time instant \( t' \) and consider the constraint (14.8) corresponding to the job \( j \) and an edge \( e \). We have,

\[
\vartheta_{eij} - p_{ej} \cdot \beta_{et'} = w_j \cdot \delta_{ej}(i, \theta_{-j}) - p_{ej} \cdot z_{et'}
\]

[From eqns ((14.9), (14.10))]

\[
\leq w_j \cdot (p_{ej} - \sum_{j' \in N(e,j,i,\theta_{-j})} q_{ej'}) - w_j \cdot \sum_{j' \in N(e,j,i,\theta_{-j})} (p_{ej'} - t')
\]

\[
\leq w_j \cdot t' + w_j \cdot p_{ej} \quad \text{[since } q_{ej'} \leq p_{ej'}]\]

Hence, the dual constraints are satisfied. Therefore, from Lemma 132 and 133 we get the robust PoA of at most \( 4 \cdot D^2 \). This concludes the proof of Theorem 123.

### 14.4.3 Price of stability for single source, multiple destination case

In this section, we prove the price of stability of temporal routing games for the case when all packets have the same source and weights. The proof is obtained by a dual analysis that exploits an interesting combinatorial property of a special NE called Minimal-Shortest Path NE, which we describe below.

Recall that \( a_{ej}(\theta), C_{ej}(\theta) \) denote the arrival time and the completion time of the packet \( j \) at the edge \( e \) in the outcome \( \theta \).

**Definition 134** (Minimal-Shortest Path NE). *An outcome \( \theta \) for the temporal routing game is said to be a Minimal-Shortest Path NE, if for every packet \( j \in \mathcal{N} \) and every edge \( e \in \theta_j \), \( j \) arrives earlier than any packet \( j' \) when \( p_{j'} > p_j \) and \( e \in \theta_{j'} \); That is, \( a_{ej}(\theta) \leq a_{ej'}(\theta) \).*
Lemma 135. A Minimal-Shortest Path NE exists for every temporal routing game if the forwarding policy on each edge is Shortest Job First, every packet has the same source and the strategy space for each packet is set of all possible paths.

Proof. We prove this by induction. Let us relabel the packets in $\mathcal{N}$ in the increasing order of their size, the packet with the smallest size getting an index of 1 (we break ties arbitrarily). We assume that all packets start at the same vertex $h$ and want to travel to some destination vertex $o_j, j \in \mathcal{N}$. The first packet takes a shortest or fastest path $P_1$ to destination $o_1$ from $h$. For the first packet, fastest path is essentially a shortest path if weight of each edge is $p_1/s_e$. (We will stick with the term fastest path.) Here, the fastest path is the path which minimizes the sojourn time of packet 1. Observe that no packet $j$ can arrive at any edge $e \in P_1$ earlier than packet 1, since it would contradict $P_1$ being the fastest path. Suppose packets 1, 2, …, $k-1$ satisfy the property stated in the definition. Fixing the routes taken by packets 1, 2, …, $k-1$, we construct the route for packet $k$ as follows. We temporarily find a path $P_k$ from $h$ to its destination $o_k$ which minimizes the sojourn time of the packet, taking into consideration the paths taken by packets smaller than $k$. Let $P_k = \{h, v_1, v_2, \ldots v_b, o_k\}$, where $v_l$ are the vertices on the path $P_k$. Note that by our selection, $P_k$ minimizes the arrival time of packet $j$ at the vertex $o_k$. Next, we move the destination of packet $k$ from $o_k$ to $v_b$ and find the shortest path from $h$ to $v_b$. Suppose $v^1_b$ is the last vertex (other than $v_b$ on this path). In the next iteration we move the destination of packet $k$ to $v^1_b$. We repeat this construction till the destination of packet $k$ in an iteration coincides with $h$. The final path for packet $k$ is constructed by following the edges in the reverse order: $\{o_k, v_b, v^1_b, v^2_b, \ldots h\}$. Therefore, from the construction no packet which has size greater than packet $j$ can arrive at any edge on the path of $j$ earlier than the packet $j$. This concludes the proof.

Lemma 136. In a Minimal-Shortest Path NE $\theta$, no packet $j$ which arrives at an edge $e \in \theta_j$ at time $a_{ej}(\theta)$ is delayed by a packet $j'$ arriving at time $a_{ej'}(\theta) > a_{ej}(\theta)$.

Proof. Fix a packet $j$ and an edge $e \in \theta_j$. Since the forwarding policy on each edge is
SJF, j can only be delayed by packets j' which have smaller size. However, since θ is a Minimal-Shortest Path NE every such packet j' arrives at e earlier than j. This completes the proof.

We note that this property is not true for every NE when the forwarding policy on each edge is Shortest Job First or if packets have multiple sources.

**LP relaxation and Dual:** Consider the LP relaxation for the problem Routing – Primal given below. This relaxation is similar to the general case with weights except for one main difference: The second constraint of the primal program says that at each time instant at most 1/2 units of packet can be routed unlike 1/2D for the Routing – Primal.

\[
\begin{align*}
\min & \sum_j \sum_{P_i \in S_j} \sum_{e \in P_i} \sum_t x_{eijt} \cdot \left( \frac{t}{p_{ej}} + 1 \right) \\
\text{s.t.} & \sum_{P_i \in S_j} x_{ij} \geq 1 \quad \forall j \in \mathcal{N} \\
& \sum_t \frac{x_{eijt}}{p_{ej}} \geq x_{ij} \quad \forall e, j \text{ and } P_i \in S_j \\
& \sum_j x_{eijt} \leq 1/2 \quad \forall e, t
\end{align*}
\]

Therefore, we get the following lemma.

**Lemma 137.** Routing – Primal is a 2D-approximation to the cost of optimal solution.

Next consider the dual. In the dual, there is a variable \(\alpha_j\) for every player \(j\), a variable \(\beta_{et}\) for every edge \(e \in E\) and time instant \(t\). There is a variable \(\vartheta_{eij}\) for every packet \(j\), edge \(e\) and a path \(P_i \in S_j\).

\[
\max \sum_j \alpha_j - \sum_{e,t} \beta_{et} \tag{Routing – Dual}
\]
\[
s.t. \quad \alpha_j - \sum_{e \in P_i} \vartheta_{eij} \leq 0 \quad \forall j, i \in S_j \tag{14.11}
\]
\[
\vartheta_{eij} - p_{ej} \cdot \beta_{et} \leq t + p_{ej} \quad \forall e, j, i \in S_j, \text{ and } t \tag{14.12}
\]
\[
\alpha_j, \beta_{et}, \vartheta_{eij} \geq 0
\]

**Setting The Dual Variables:** Fix an outcome \( \theta \) which is in Minimal-Shortest Path NE. Our interpretation of the dual variables follow the standard template.

- We set \( \alpha_j \) to the cost incurred by the packet \( j \); in other words, \( \alpha_j = \text{Cost}_j(\theta) \), which is the sojourn time of packet \( j \) in \( \theta \).

- Consider the variable \( \vartheta_{eij} \) corresponding to the path \( P_i \) chosen by packet \( j \) in the outcome \( \theta \). That is, \( P_i = \theta_j \). We set \( \vartheta_{eij} \) to the total delay the packet \( j \) suffers at the edge \( e \) (which includes the time spent processing the packet). Let \( \delta_{eij}(\theta) = C_{eij}(\theta) - a_{eij}(\theta) \) denote the total delay seen by \( j \). Then, we set \( \vartheta_{eij} = \delta_{eij}(\theta) \). From the definition of the variables, we have \( \alpha_j = \sum_{e \in P_i} \vartheta_{eij} \).

- Consider the variables \( \vartheta_{eij} \) corresponding to the paths \( P_i \in S_j \) and \( P_j \neq \theta_j \). That is, the set of paths packet \( j \) didn’t choose in the outcome \( \theta \). We set \( \vartheta_{eij} \) as the total delay the packet \( j \) would suffer on \( e \), if the packet chose \( P_i \) fixing the paths chosen by other packets \( (\theta_{-j}) \). Recall that \( a_{eij}(i, \theta_{-j}) \) and \( C_{eij}(i, \theta_{-j}) \) denote the arrival and completion time of the packet \( j \) on the edge \( e \) if it takes the path \( P_i \) fixing the paths of other packets. Let \( \delta_{eij}(i, \theta_{-j}) = C_{eij}(i, \theta_{-j}) - a_{eij}(i, \theta_{-j}) \). We set \( \vartheta_{eij} = \delta_{eij}(i, \theta_{-j}) \). Define, \( \text{Cost}_j(i, \theta_{-j}) = \sum_{e \in P_i} \vartheta_{eij} \). From the definition of dual variables and the fact that \( \theta \) is a NE, we have \( \alpha_j \leq \text{Cost}_j(i, \theta_{-j}) \) \( \forall j, P_i \in S_j \).

- We set \( \beta_{et} \) as follows. Let \( \mathcal{N}(e, \theta) \subseteq \mathcal{N} \) be the set of packets which pass through the edge \( e \) (at some point of time) in the outcome \( \theta \). We imagine \( \beta_{et} \) as the sum of \( \beta_{ejt} \) for every packet \( j \in \mathcal{N}(e, \theta) \). For a packet \( j \), we set \( \beta_{ejt} \) to 1 for every time instant \( t \in [0, \delta_{eij}(\theta)] \). Finally, we set \( \beta_{et} = \sum_{j \in \mathcal{N}(e, \theta)} \beta_{ejt} \). This setting of \( \beta_{et} \) variables is different from the PoA analysis.
Bounding The Dual Objective:

Lemma 138. If the variables of the dual are set as described above, then cost of the Routing – Primal is at least $1/2 \cdot \sum_j \text{Cost}_j(\theta)$

Proof. From the definition of $\alpha_j$ we note that $\sum_j \alpha_j = \text{Cost}_j(\theta)$. Now consider,

$$\sum_{e,t} \beta_{et} = \sum_{e,t} \sum_{j \in N(e,\theta)} \beta_{ejt}$$

$$= \sum_{e} \sum_{j \in J(e,\theta)} \sum_{t=0}^{t=\delta_{ej}(\theta)} \sum_{t} 1$$

$$= \sum_{j} \sum_{e \in \theta_j} \delta_{ej}(\theta) = \sum_{j} \text{Cost}_j(\theta)$$

Therefore, from the weak duality theorem we get,

$$\text{LP-cost} \geq \sum_j \alpha_j - \sum_{e,t} \beta_{et} = 1/2 \cdot \sum_j \text{Cost}_j(\theta)$$

Checking The Dual Constraints: Fix a packet $j$. From the definition of dual variables and since $\theta$ is a NE, the constraints of type (14.11) (which encode the equilibrium condition) corresponding to $j$ are satisfied. This is because,

$$\alpha_j = \sum_{e \in \theta_j} \vartheta_{eij}(\theta) \leq \sum_{i \in S_j} \text{Cost}_j(i, \theta_{-j})$$

$$= \sum_{e \in P_i} \vartheta_{eij}(i, \theta_{-j}) \ \forall P_i \in S_j$$

Now consider the constraints of type (14.12). We consider two cases.

Case 1: Packet $j$ chooses path $P_i$ in $\theta$.

Fix an edge $e \in P_i$. Recall that $\delta_{ej}(\theta)$ denotes the total time packet $j$ spends waiting at the edge $e$. Consider the time interval $[0, \delta_{ej}(\theta)]$ and we need to verify that for all
$t \in [0, \delta_{ej}(\theta)]$, the constraints (14.12) are satisfied. Note that the constraints are trivially satisfied when $t > \delta_{ej}(\theta)$. Let $\mathcal{N}(e, j, \theta)$ be the set of packets which delay $j$ on edge $e$. As each edge follows SJF policy, for every packet $j' \in \mathcal{N}(e, j, \theta), p_{j'} \leq p_j$. From Lemma 136, all the packets in $\mathcal{N}(e, j, \theta)$ arrive at $e$ earlier than $j$. Let $j^*$ be the packet that is being processed at the edge $e$ when $j$ arrives at $e$. Let $p_{j^*}'$ denote the remaining size of packet $j^*$ when the packet $j$ arrives. This implies that every packet $j' \in \mathcal{N}(e, j, \theta)$ is delayed by packet $j^*$ at least by $p_{j^*}'$ time units. By applying Lemma 136 to every packet $j'$ in $\mathcal{N}(e, j, \theta)$ we conclude that

$$
\delta_{ej'}(\theta) \geq p_{j^*}' + \sum_{\{j'' \in \mathcal{N}(e, j, \theta) \setminus j^* \text{ and } p_{j''} \leq p_{j'}\}} p_{j''}
$$

(14.13)

(The above inequality is crucial in verifying the dual constraints and is true only if Lemma 136 holds. Unfortunately, this is not the case with every NE. Thus, we loose another factor $D$ in PoA.)

Consider the packets in the set $\mathcal{N}(e, j, \theta)$ and index them in the increasing order of their size breaking ties arbitrarily. Consider a time instant $t$ in the interval $[0, \delta_{ej}(\theta)]$. Let $k$ denote the smallest index of the packet in $\mathcal{N}(e, j, \theta)$ such that $\sum_{i=1}^k p_j \geq t$. We can express $\delta_{ej}(\theta)$ as follows.

$$
\delta_{ej}(\theta) \leq p_{ej} + \sum_{j''=1}^{k-1} p_{ej''} + \sum_{j''=k}^{|\mathcal{N}(e, j, \theta)|} p_{ej''}
$$

$$
\leq (t + p_{ej}) + \sum_{j''=k}^{|\mathcal{N}(e, j, \theta)|} p_{ej''}
$$

However, since each packet $j''$ in the set $\mathcal{N}(e, j, \theta)$ is smaller than the packet $j$,

$$
\sum_{j''=k}^{|\mathcal{N}(e, j, \theta)|} p_{ej''} \leq (|\mathcal{N}(e, j, \theta)| - (k - 1))p_{ej}
$$

259
Therefore, for every time instant \( t \) we can upper bound the total delay seen by \( j \) at \( e \) by

\[
\delta_{ej}(\theta) \leq t + p_{ej} + (|N(e, j, \theta)| - (k - 1))p_j
\]

(14.14)

From our definition of dual variable \( \beta_{et} \) and from the equation (14.13) it follows that

\[
\beta_{et} \geq |N(e, j, \theta)| + 1 - (k - 1)
\]

(14.15)

The ' +1' in the equation is due to the contribution of packet \( j \) to the term \( \beta_{et} \). We are ready to verify that constraints are satisfied at \( t \). Consider the constraint of type (14.12). From equations (14.14) and (14.15) we get,

\[
\vartheta_{eij} - p_{ej} \cdot \beta_{et} \leq t + p_{ej} \cdot (|N(e, j, \theta)| - (k - 1)) - p_{ej} \cdot (|N(e, j, \theta)| - (k - 1)) \leq t
\]

Case 2: Packet \( j \) does not choose path \( P_i \in S_j \) in \( \theta \). We need to verify that constraints (14.12) hold for the paths \( P_i \in S_j \) not chosen by the packet \( j \). Only difference between the this case and the first case is that \( \vartheta_{eij} \) is set to \( \delta_{ej}(i, \theta_{-j}) \) which is the total time packet \( j \) waits at the edge \( e \) in the hypothetical outcome where packet \( j \) takes path \( P_i \), while other packets follows paths in \( \theta_{-j} \). On the other hand \( \beta_{et} \) is defined with respect to the outcome \( \vartheta \) where job \( j \) takes path \( \theta_j \). In particular, if the forwarding policy on each edge is not SJF, then the set of packets seen by job \( j \) in the outcome \( \theta \) can be very different from the set of packets job \( j \) sees in the outcome where \( j \) takes the path \( P_i \) and remaining packets take paths in \( \theta_{-j} \).

However, since each edge follows SJF, the set of packets which can delay packet \( j \) remain unaffected by this hypothetical switching of strategy by the packet \( j \). In other words, the schedule of packets which are smaller than packet \( j \) remains exactly the same in
the outcome $\theta$ and the outcome where $j$ follows path $P_i$ and other packets follow paths in $\theta_{\sim j}$. Hence, the contribution of those packets to the dual variable $\beta_{et}$ remains unchanged.

We now argue that this is enough to show that constraints are satisfied.

Let $\mathcal{N}(e,j,i,\theta_{\sim j})$ be the set of packets which delay $j$ on the edge $e$ if $j$ switches to the path $P_i$. Repeating the arguments from the case (1), we can write for every time instant $t \in [0, \vartheta_{eij}]$ an equation similar to eqn(14.14):

$$
\delta_{ej}(i, \theta_{\sim j}) \leq t + p_{ej} + (|\mathcal{N}(e,j,i,\theta_{\sim j})| - (k - 1))p_{ej}
$$

(14.16)

Similarly we have,

$$
\beta_{et'} \geq |\mathcal{N}(e,j,i,\theta_{\sim j})| - (k - 1)
$$

(14.17)

Observe that unlike eqn(14.15) we do not have '+1' in above equation as $j$ does not take the path $P_i$ in $\theta$. Consider the dual constraints,

$$
\vartheta_{eij} - p_{ej} \cdot \beta_{et}
\leq t + p_{ej} + (|\mathcal{N}(e,j,i,\theta_{\sim j})| - (k - 1)) \cdot p_{ej}
$$

$$
- p_{ej} \cdot (|\mathcal{N}(e,j,i,\theta_{\sim j})| - (k - 1))
\leq t + p_{ej}
$$

Therefore, we get a bound of $4D$ from Lemma (137) and Lemma (138) on the inefficiency of Minimal-Shortest Path NE. This completes the proof of Theorem 125

14.4.4 Tree Topologies

A special case of our problem considered by Bhattacharya et al. (Bhattacharya et al., 2014a) is the following : We are given a rooted tree and all packets start at the root. For each packet $j$, we are also given a set $L_j$, which is simply a subset of nodes of the tree. A job $j$ starts at the root and wants to exit the tree through any one of the nodes in $L_j$. Given a forwarding policy, each job selects a destination vertex in $L_j$ which gives it the
minimum completion time and this induces a game among the packets. Our objective is to bound the PoA of resulting game. Bhattacharya et al. (Bhattacharya et al., 2014a) showed that the PoA of pure NE for the game is $O(\log^2 \nu)$, where $\nu$ is the ratio of maximum speed to minimum, for the case when packets have same weight.

We observe that for this problem, Lemma 136 applies for every NE. Therefore, we can set the dual variables as defined in the price of stability analysis and we get an improved bound of $O(\log \nu)$ for the problem. A closer look at the proof reveals that the entire analysis even generalizes to coarse correlated equilibrium. Thus, we conclude that the PoA of CCE for tree network is $O(\log \nu)$.

**Theorem 139.** The PoA of CCE for the temporal routing problem on rooted tree networks is at most $O(\log \nu)$.

We note that the analysis in (Bhattacharya et al., 2014a) only applies to pure NE.

14.5 Energy Minimization Games in Machine Scheduling

We study the problem of energy minimization in machine scheduling from a game theoretic perspective. In this problem, we are given a set $\mathcal{M}$ of machines and a set $\mathcal{N}$ of jobs. Each job $j$ has a processing requirement of $p_{ij}$, a weight $w_{ij}$ on machine $i$. Each machine $i$ can run at a variable speed $\eta(i,t)$ by paying an energy cost of $\eta(i,t)^\gamma$, $\gamma \geq 2$. (In practice $\gamma = 2, 3$. We can extend our results to arbitrary energy functions, but for the sake of clarity we consider polynomial functions.) The objective is to design a speed scaling policy and a machine scheduling policy such that we minimize the total energy consumed while simultaneously guaranteeing a certain quality of service, such as average completion times of jobs. One of the most commonly used algorithms for the problem is the following.

*Machine Scheduling Policy.* Each machine follows the Highest Density First (HDF) scheduling policy.
**Speed Scaling Policy.** We set the speed of machine $i$ at time $t$ such that the total energy cost is equal to the total fractional weight of jobs at time $t$. For a job $j$, let the fractional weight at time $t$ be defined as $\frac{p_{ij}(t)}{P_{ij}} \cdot w_{ij}$, where $p_{ij}(t)$ be the remaining processing length of job $j$ at time $t$. Let $W_{it}(\theta)$ denote the total fractional weight of jobs at time $t$ on machine $i$ in an outcome $\theta$. Then,

$$(\eta(i, t))^\gamma = W_{it}(\theta) \quad \text{or} \quad \eta(i, t) = (W_{it}(\theta))^{\frac{1}{\gamma}}$$

Note that since we use HDF, there can be at most one job that is partially processed at any given instant of time. The setting of speed equal to fractional weight is only for the sake of technical convenience.

We study this problem in the game theoretic setting. In this setting, we assume that each job is a strategic agent and chooses the machine which minimizes its weighted completion time. An outcome of this game consists of each job $j \in \mathcal{N}$ choosing a machine $i \in \mathcal{M}$. Given an outcome $\theta = (\theta_1, \theta_2, \ldots, \theta_n)$, where $\theta_j$ denotes the machine job $j$ chooses, let $C_j(\theta)$ denote the completion time of job $j$ in the outcome $\theta$. Then,

$$\text{Cost}_j(\theta) = w_{\theta_j} \cdot C_j(\theta)$$

The social cost for an outcome is the sum of players cost + energy consumed. Formally,

$$\text{Cost}(\theta) = \sum_{j \in \mathcal{N}} \text{Cost}_j(\theta) + \sum_{i \in \mathcal{M}} \int_t \eta(i, t)^\gamma dt$$

Our objective is to understand the PoA of this game for general equilibrium concepts such CCE.

14.5.1 **Robust PoA bound**

We first observe that for any outcome $\theta$, the sum of weighted completion time of jobs is at most $\sum_i \int_t W_{it}(\theta) dt$ (Indeed, it would have been equal $\sum_i \int_t W_{it}(\theta) dt$, if $W_{it}$ denoted
the total integral weight of jobs at time $t$). The speed scaling policy ensures that in any outcome the total energy cost is at most the total weighted completion time of jobs, so we focus on the latter quantity.

Convex Programming Relaxation and Dual. Let,

$$
\mathcal{F} = \sum_j \sum_i \int_t w_{ij} \cdot x_{ijt} \cdot \frac{t}{p_{ij}} dt
$$

and

$$
\mathcal{E}_1 = (1 + \epsilon)^{2\gamma} \cdot \sum_i \int_t x_{it}^\gamma dt
$$

$$
\mathcal{E}_2 = \sum_i \sum_j w_{ij}^{1-1/\gamma} \int_t x_{ijt} dt
$$

Consider the following convex programming relaxation for the problem due to Anand et al. (2012); Devanur and Huang (2014).

$$
\begin{align*}
\text{min} & \quad (\mathcal{F} + \mathcal{E}_1 + \mathcal{E}_2) \\
\text{s.t.} & \quad \sum_i \int_t x_{ijt}/p_{ij} \geq 1 \quad \forall j \\
& \quad \sum_j x_{ijt} = x_{it} \quad \forall i, t \\
& \quad x_{ijt} \geq 0 \quad \forall i, j, t
\end{align*}
$$

In this relaxation, there is a variable $x_{ijt}$ which indicates the speed at which the job $j$ is processed at time $t$ on machine $i$. The constraints of Energy – Primal state that each job needs to be completely processed and the speed of machine $i$ at time $t$ is equal to the sum of individual speeds of the jobs.

We give a brief explanation on why the objective function is an $O(1)$-approximation to the cost of optimal solution. See (Devanur and Huang, 2014; Anand et al., 2012) for
a complete proof of this claim. The first term in the objective function lower bounds
the weighted completion of jobs. The second term corresponds to the energy cost of the
schedule. We use a scaling factor of $1 + \epsilon$, where $\epsilon = \frac{\gamma}{\gamma - 1}$. This will be useful in the
dual analysis. Note that this only increases the cost of objective function by a constant
factor. The third term is a lower bound on the total cost any optimal solution has to pay
to schedule a job $j$, assuming that $j$ is the only job present in the system. This term is
needed, as we do not explicitly put any constraints on disallowing simultaneous processing
of jobs across machines. Without this term, Energy − Primal has a huge integrality gap as
a single job can be processed to an extent of $\frac{1}{m}$ simultaneously on all machines.

We write the dual of Energy − Primal following the framework given in (Devanur and
Huang, 2014). Just like all the LP and CP relaxations we wrote earlier, we have a variable
$\beta_{it}$ for each machine $i$ and time instant $t$, and a variable $\alpha_j$ for each job $j$.

\[
\max \sum_j \alpha_j - 1_\gamma \cdot \frac{1}{(1 + \epsilon)^2} \cdot \sum_i \int_t^{\gamma - 1} \beta_{it} \cdot dt \\
\text{s.t.} \quad \frac{\alpha_j}{p_{ij}} - \beta_{it} \geq \frac{w_{ij}}{p_{ij}} \cdot t + \frac{1}{\gamma^{1/\gamma}} \quad \forall i, j, t
\]

\[
\alpha_j \geq 0 \quad \forall j
\]

\[
\beta_{it} \geq 0 \quad \forall i, t
\]

Here, $1_\gamma = (\frac{1}{\gamma^{1/\gamma}} - \frac{1}{\gamma^{1/\gamma}})$ is a constant we get in the Fenchel dual of function $x^\gamma$.

**Setting The Dual Variables:** Let $\sigma$ be a distribution in a coarse correlated equilibrium. Our
interpretation of the dual variables is as follows.

- We set $\alpha_j$ proportional to the expected cost incurred by the player $j$ in $\sigma$. That is,

\[
\alpha_j = (1 - 1/\gamma) \cdot \mathbb{E}_{j \sim \sigma}[\text{Cost}_j(\theta)]
\]
We set $\beta_{it}$ to the total expected fractional weight of jobs at time $t$ divided by the speed of machine $i$. More precisely,

$$\beta_{it} = \mathbb{E}_{\theta \sim \sigma}[W_{it}^{\gamma - 1/\gamma}(\theta)]$$

**Remark 140.** For the readers familiar with the online version of the problem, we highlight the differences. Our setting of the dual variable $\alpha_j$ is different from the dual-fitting proof for the problem in the online setting used in (Devanur and Huang, 2014; Anand et al., 2012). In the online version of the problem, $\alpha_j$ is set to the marginal increase in the cost of the objective due to job $j$. Note that such a setting of dual variable won’t work for our problem. The jobs select machines depending on their own weighted completion time not the increase in the objective caused by them. However, some of the technical lemmas essentially follow from (Devanur and Huang, 2014; Anand et al., 2012).

**Bounding The Dual Objective:** From the weak duality theorem, the cost of primal program is

$$\text{CP-Cost} \geq \sum_j \alpha_j - 1/\gamma \cdot \frac{1}{(1+\epsilon)^2} \cdot \sum_i \int_t \beta_{it}^{\gamma - 1}$$

$$\geq \sum_j (1 - 1/\gamma) \cdot \mathbb{E}_{\theta \sim \sigma} \text{Cost}_j(\theta)$$

$$\frac{1}{(1+\epsilon)^2} \int_{i,t} \mathbb{E}_{\theta \sim \sigma}[W_{it}(\theta)]$$

$$\geq \frac{\gamma - 1}{\gamma^2} \cdot \sum_j \mathbb{E}_{\theta \sim \sigma}[\text{Cost}_j(\theta)] \quad \text{[since, } \epsilon = \frac{1}{\gamma - 1}]$$

The second inequality follows from Jensen’s inequality and also note that $\int_{i,t} W_{it}(\theta)$ is equal to the weighted completion-time of jobs in the outcome $\theta$. As already observed, the primal program is at most $O(1)$-approximation to the optimal cost. Since the energy cost of the schedule is equal to the total weighted completion time, we lose another factor of
Therefore, we get a bound of $O(\gamma)$ on the PoA of CCE \(^2\).

**Checking The Constraints.** It remains to verify that our setting of the dual variables satisfy the dual constraints. We note that technical details of the proofs in this part are similar to the proofs in (Anand et al., 2012; Devanur and Huang, 2014) and we omit few details.

Recall that for every job $j$, machine $i$, and time instant $t$ we have a constraint,

$$
\frac{\alpha_j}{p_{ij}} - \beta_{it} \leq \frac{w_{ij}}{p_{ij}} \cdot t + w_{ij}^{1-1/\gamma} \quad \forall i, j, t
$$

We first prove that for every outcome $\theta$, the constraint corresponding to $j, i$ and $t$ is satisfied if we set $\beta_{it} = W_{it}^{\gamma-1/\gamma}(\theta)$ and $\alpha_j = (1 - 1/\gamma) \cdot \text{Cost}_j(i, \theta_{-j})$. This would imply that the constraint corresponding to a fixed $j, i$ and $t$ triple is satisfied in expectation if $\alpha_j = (1 - 1/\gamma) \cdot \mathbb{E}_{\sigma \sim \sigma}[\text{Cost}_j(i, \theta_{-j})]$ and $\beta_{it} = \mathbb{E}_{\sigma \sim \sigma}[W_{it}^{\gamma-1/\gamma}(\theta)]$. Once we show this, we use the fact that $\sigma$ is a coarse correlated equilibrium and $\mathbb{E}_{\sigma \sim \sigma}[\text{Cost}_j(\theta)] = \min_i \mathbb{E}_{\sigma \sim \sigma}[\text{Cost}_j(i, \theta_{-j})]$, which will complete the proof.

Fix an outcome $\theta$. We focus on proving that $\beta_{it} = W_{it}^{\gamma-1/\gamma}(\theta)$ and $\alpha_j = (1 - 1/\gamma) \cdot \text{Cost}_j(i, \theta_{-j})$ satisfy the constraint 14.18 for the job $j$, machine $i$ and time instant $t$. Two cases arise: (a) the job $j$ chose machine $i$ in the outcome $\theta$ and (b) the job chose machine $i \neq \theta_j$. We prove case (b) since it subsumes case (a). Proving case (b) is tricky because $\alpha_j$ is defined with respect to the hypothetical event that the job $j$ chose machine $i$, while $\beta_{it}$ is defined with respect to the actual outcome $\theta$. Let $\theta' := (\theta_j = i, \theta_{-j})$ denote this hypothetical outcome. Note that $W_{it}(\theta') \neq W_{it}(\theta)$. This is because, in the outcome $\theta'$, the total weight of jobs on the machine $i$ is strictly more than the outcome $\theta$ and hence the machine runs faster in the outcome $\theta'$. Let $T_j$ denote the first time instant in $\theta'$ when the job $j$ gets executed.

**Lemma 141.** $W_{it}(\theta) \geq W_{it}(\theta') - w_{ij}, \quad \forall t < T_j.$

\(^2\) The constant is at most 32.
Figure 14.2: The figure illustrates the schedule on machine $i$ in the outcome $\theta$ (top) and $\theta'$ (bottom). The job $j$ in the outcome $\theta'$ is represented by the solid block. Note that in $\theta'$, the machine runs faster because of the weight of job $j$. Therefore, $W_{it}(\theta) \geq W_{it}(\theta') - w_{ij}$, till the job $j$ starts getting processed in $\theta'$.

**Proof.** Let $J(j, \theta)$ denote the set of jobs which have density more than the density of job $j$ on machine $i$. Hence, the jobs in the set $J(i, \theta)$ get executed first in both $\theta$ and $\theta'$. However, since the machine runs faster in $\theta'$ (we set the speed such that energy cost is equal to the total remaining fractional weight) the jobs in the set $J(i, \theta)$ finish earlier in $\theta'$ than $\theta$. Thus, $W_{it}(\theta) \geq W_{it}(\theta') - w_{ij}, \ \forall t < T_j$. See Figure 14.2 for an illustration.

Therefore, if we show that constraints are satisfied for $\theta'$, then the constraints are also satisfied for $\theta$ as $\beta_{it}$ can be only higher.

**Lemma 14.2.** For any time instant $t' \leq T_j$, $\frac{w_{ij}}{p_{ij}} \cdot (T_j - t') \leq \frac{W_{it}^{-1/\gamma}(\theta')}{W_{iu}^{1/\gamma}(\theta')}$.  

**Proof.** We give a brief outline of the proof here. See Anand et al. (2012) for more details. Consider any small interval of time $[u, u + du]$. In this interval, the total weight of jobs decreases by $dW \geq \frac{w_{ij}}{p_{ij}} \cdot W_{iu}^{1/\gamma}(\theta') \cdot du$. This is because, the density of the job being processed at time $u$ is at least density of the job $j$ and the machine runs at the speed $W_{iu}^{1/\gamma}(\theta')$. Therefore we get the following differential equation.

$$
\int_{t'}^{T_j} du \leq \int_{W_{it}(\theta')}^{W_{it}(\theta')} \left( \frac{p_{ij}}{w_{ij}} \cdot \frac{1}{W_{iu}^{1/\gamma}} \right) \cdot dW \\
\leq \frac{p_{ij}}{w_{ij}} \cdot \frac{W_{it}^{-1/\gamma}(\theta')}{1 - 1/\gamma}
$$
Now we upper bound the cost of job $j$ in the outcome $\theta'$.

**Lemma 143.** $\text{Cost}_j(\theta') \leq w_{ij} \cdot T_j + \frac{p_{ij}}{(1-\frac{1}{\gamma})} \cdot w_{ij}^{1-\frac{1}{\gamma}}$

**Proof.** From the definition, $T_j$ is the last time instant when a job of density at least the density of job $j$ gets scheduled on machine $i$ in $\theta'$. Let us upper bound the time it takes to finish $j$ (call it $y_{ij}$). Since $W_{iT_j}(\theta') \geq w_{ij}$, repeating the arguments similar to 142 we conclude that it takes at most $y_{ij} \leq \frac{p_{ij}}{w_{ij}(1-\frac{1}{\gamma})} \cdot w_{ij}^{1-\frac{1}{\gamma}}$ units of time to process $j$. Since, $\text{Cost}_j(\theta') \leq w_{ij} \cdot (T_j + y_{ij})$, we complete the proof. \qed

Now consider the dual constraint 14.18 for job $j$, machine $i$ and some time instant $t' \leq T_j + y_{ij}$.

\[
\frac{\alpha_j}{p_{ij}} - \frac{w_{ij}}{p_{ij}} \cdot t' - \beta_{it'} = (1 - \frac{1}{\gamma}) \cdot \text{Cost}_j(\theta') - \frac{w_{ij}}{p_{ij}} \cdot t' - W_{it'}^{1-\frac{1}{\gamma}}(\theta')
\leq \frac{w_{ij}}{p_{ij}} \cdot T_j + w_{ij}^{1-\frac{1}{\gamma}} - \frac{w_{ij}}{p_{ij}} \cdot t' - W_{it'}^{1-\frac{1}{\gamma}}(\theta')
\leq \frac{w_{ij}}{p_{ij}} \cdot (T_j - t') + w_{ij}^{1-\frac{1}{\gamma}} - W_{it'}^{1-\frac{1}{\gamma}}(\theta')
\leq w_{ij}^{1-\frac{1}{\gamma}} \quad \text{(From Lemma 142)}
\]

This completes the analysis and proof of Theorem 126. Therefore, the robust price of anarchy of energy minimization game is at most $O(\gamma)$.

### 14.6 Summary and Open Problems

In this chapter, we showed how to use LP and Fenchel duality to obtain PoA bounds for a wide class games for general equilibrium concepts such as CCE. The duality theory lies at the heart much of the approximation and online algorithms literature and seems to be equally applicable for analyzing efficiency of equilibrium outcomes, as demonstrated.
in this chapter. We believe that the technique is fairly general and we hope finds more applications.

In terms of more specific questions, there is a gap in our understanding of coordination mechanisms for graphs as we could only show an upper bound of $4D^2$. It would be nice to bridge this gap and perhaps get better upperbounds for special graph classes such as series-parallel graphs etc. Lastly, a tighter analysis of the PoS would shed more light on the structural complexity of the problem. Indeed, we believe that for the case of single source and single destination, one can get much better PoS than shown in this chapter.

Much of the coordination mechanism design literature for scheduling and routing games has focused on designing policies where jobs have complete information and select machines which minimize their overall delay or completion time. However, in many real world applications jobs do not have this information. Typically, these problems are modeled in the framework of non-clairvoyant scheduling in online and approximation algorithms literature. Correspondingly, these problems can be modeled as instantaneous coordination games, where each agent selects a machine which gives her the best utility at current instant of time. Here, a job may be change its machine over the course of time but at every instant of time it is in NE. This idea of instantaneous coordination games was used in a recent work by (Im et al., 2014b) in a non-game theoretic setting. We believe that our framework and ideas developed this chapter can be useful for these problems. However, we omit details and leave it as a future research direction.

14.7 Notes

This chapter is based on join work with Vahab Mirrokni, and appeared in Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015 (Kulkarni and Mirrokni, 2015).
We considered scheduling problems that arise in a wide range of applications, and designed algorithms with provable performance guarantees for the objective of minimizing delay or flow-time of jobs. Our work identifies two broad unifying principles: a) Usefulness of game theoretic and economic ideas in the design of scheduling algorithms. b) Power of linear programming and duality in the analysis of scheduling algorithms and games. We hope these principles will find more applications in future research related to these topics.

The research presented in this thesis can be extended in several directions (see specific chapters for more details): Our understanding of the classical unrelated machine scheduling is far from complete, and several important problems remain open including the problem of minimizing the weighted flow-time on a single machine. The polytope scheduling problem, introduced in this thesis, is just the first step in modeling scheduling problems that arise in data centers. The PSP problem models a single cluster setting; what happens if there multiple clusters? Data center scheduling is complex, but provides a rich opportunity for new algorithmic research. Our work in selfish scheduling problems led to the development of dual-fitting technique for the PoA analysis; an interesting research direction is to explore the power of dual analysis compared to other methods for bounding the PoA.

The above questions concern the topics explored in this thesis. The assumptions made
in this thesis, however, do not hold in many settings. All the scheduling problems considered in this thesis make the assumption that there are no dependencies among jobs, which is not true in many cases. Consider the example of a MapReduce system, where the reduce tasks cannot begin execution till the map tasks complete. These dependencies among jobs can be modeled in the framework of precedence constrained scheduling. Our understanding of scheduling with precedence constraints is minimal and is in need of fresh new ideas.

Another broad class problems of that arise in practice frequently are scheduling and resource allocation problems in presence of "real" money. Consider the cloud services such as Microsoft Azure, Google Cloud, Amazon Elastic Cloud, etc., where users can rent machines (by paying money) in the cloud to meet their computational demands. In these settings, the service providers need to make two decisions: 1) How to price the resources, 2) How to schedule jobs. These problems which are at the intersection of economics and scheduling theory would benefit from new algorithmic insights.

The above applications are just two examples where scheduling decisions play a crucial role. Scheduling and resource allocations problems, however, surface in almost every human endeavor. In a survey paper, Scheduling Algorithms, David Karger, Cliff Stein, and Joel Wein write:

"The practice of this field dates to the first time two humans contended for a shared resource and developed a plan to share it without bloodshed" 

We hope ideas in this thesis lead to less bloodshed.
Bibliography


Bergstrom, T. (Manuscript, 2014), “Proving that a Cobb-Douglas function is concave if the sum of exponents is no bigger than 1,”.


275


Im, S. and Moseley, B. (2015), “Scheduling in Bandwidth Constrained Tree Networks,” SPAA.


Biography

Janardhan Dattatreya Kulkarni is from Jog, a small rustic town known for its lush surroundings and magnificent water falls in the heart of Sahyadri moutain range, Karnataka, India. He spent memorable eighteen years in Jog before moving to Mysore to obtain bachelor’s degree in computer science from Shri Jayachamarajendra College of Engineering. Janardhan got his master’s degree from the Indian Institute of Science, Bangalore. He obtained his Ph.D from the department of computer science, Duke University, Durham, USA. Janardhan is a recipient of gold medal from the Indian Institute of Science for the academic year 2010, outstanding prelim award and outstanding teaching and mentoring award from the department of computer science at Duke University. When he is not designing algorithms, Janardhan spends his time playing tennis, running, and watching the night sky.