Competitive Algorithms from Competitive Equilibria: Non-Clairvoyant Scheduling under Polyhedral Constraints

Sungjin Im, University of California, Merced
Janardhan Kulkarni, Microsoft Research
Kamesh Munagala, Duke University

We introduce and study a general scheduling problem that we term the Polytope Scheduling problem (PSP). In this problem, jobs can have different arrival times and sizes; and the rates assigned by the scheduler to the jobs are subject to arbitrary packing constraints. The PSP framework captures a variety of scheduling problems, including the classical problems of unrelated machines scheduling, broadcast scheduling, and scheduling jobs of different parallelizability. It also captures scheduling constraints arising in diverse modern environments ranging from individual computer architectures to data centers. More concretely, PSP models multidimensional resource requirements and parallelizability, as well as network bandwidth requirements found in data center scheduling.

We show a surprising result – there is a *single* algorithm that is $O(1)$ competitive for *all* PSP instances when the objective is total completion time, and $O(1)$ competitive for a large sub-class of PSP instances when the objective is total flow time. This algorithm simply uses the well-known Proportional Fairness algorithm (PF) to perform allocations each time instant. Though PF has been extensively studied in the context of maximizing fairness in resource allocation, we present the *first* analysis in adversarial and general settings for optimizing job latency. Further, PF is non-clairvoyant, meaning that the algorithm doesn't need to know jobs sizes until their completion. We establish our positive results by making novel connections with Economics, in particular the notions of market clearing, Gross Substitutes, and Eisenberg Gale markets.

We complement these positive results with a negative result: We show that for the total flow time objective, any non-clairvoyant algorithm for *general* PSP has a strong lower bound on the competitive ratios unless given a poly-logarithmic speed augmentation. This motivates the need to consider sub-classes of PSP when studying flow time. The sub-class for which we obtain positive results not only captures several well-studied models such as scheduling with speedup curves and related machine scheduling, but also captures as special cases hitherto unstudied scheduling problems such as single source flow routing, routing multicast (video-on-demand) trees, and resource allocation with substitute resources.

Additional Key Words and Phrases: Online scheduling, polytope constraints, total completion time, total flow time, non-clairvoyant, market equilibrium, proportional fairness, adversarial input

1. INTRODUCTION

In a typical *non-clairvoyant* scheduling problem, jobs arrive online in an adversarial fashion, with the scheduler knowing job characteristics, but not its size. Algorithms

---

This work combines two conference papers, appearing in ACM STOC 2014 [Im et al. 2014] and IEEE FOCS 2015 [Im et al. 2015] respectively.
Im was supported in part by NSF grants CCF-1008065 and CCF-1409130; part of this work was done when Im was at Duke. This work was done while Kulkarni was at Duke, supported by NSF grants CCF-1008065, CCF-0745761, CCF-1348696, and IIS-0964560. Munagala was supported by an award from Cisco, by NSF grants CCF-0745761, CCF-1008065, CCF-1348696, CCF-1408784, IIS-0964560, IIS-1447554, and by grant W911NF-14-1-0366 from the Army Research Office (ARO).
Author's addresses: S. Im, Electric Engineering and Computer Science, University of California, Merced, CA 95343, sim3@ucmerced.edu; J. Kulkarni , Microsoft Research, One Microsoft Way, Redmond, WA 98052-6399, janardhan.kulkarni@gmail.com; K. Munagala, Computer Science Department, Duke University, Durham NC 27708, kamesh@cs.duke.edu.

DOI: 0000001.0000001

with good performance guarantees (that is, *constant competitive* in scheduling parlance) in terms of total flow time (latency) or total completion time are known for disparate problems, such as scheduling jobs on identical parallel machines [Chekuri et al. 2004], scheduling pages to broadcast [Bansal et al. 2010], or scheduling on parallel machines when a job's rate of execution depends on how many machines it is allocated [Robert and Schabanel 2008]. However, the techniques for solving these problems seem quite specialized, and we hit a roadblock when we try to extend these techniques to more general problems, such as scheduling unrelated parallel machines, or scheduling parallel machines when a job's rate of execution depends on the total CPU *and* memory it is allocated, and so on.

In this paper, we present a unified model for several widely studied scheduling problems, both classical and modern. We term this general model *Polytope Scheduling Problem* (PSP). This model takes a *instantaneous resource allocation* view of scheduling – after all, any scheduling algorithm allocates limited resources (broadcast slot, machine, CPU, etc.) among competing jobs every time instant. The advantage of this view is that it naturally connects to work in Economics, where there is a significant body of work on designing resource allocation algorithms with several desirable properties, such as efficiency or fairness. We focus on one such scheme, termed *Proportional Fairness* or *Nash Product*. We show that if the scheduler simply executes this resource allocation scheme every time instant, then the resulting scheduling algorithm is constant competitive for total completion time on *all* PSP instances, and is constant competitive for total job latency on a large sub-class of PSP instances. The very existence of *one* unifying algorithm that has good guarantees for so many scheduling problems is indeed quite surprising in itself!

1.1. The Polytope Scheduling Problem (PSP)

In the PSP problem, 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 $\{y_j(t)\}$ to the current jobs in the system. Let $y_j^{\mathcal{A}}(t)$ denote the rate at which job j is processed at time t by a scheduler/algorithm $\mathcal{A}$. Job j's completion time $C_j^{\mathcal{A}}$ under the schedule of $\mathcal{A}$ is defined to be the first time t' such that $\int_{t=r_j}^{t'} y_j^{\mathcal{A}}(t)\mathrm{d}t \geq p_j$. Similarly, we define job j's flow time as $F_j^{\mathcal{A}} = C_j^{\mathcal{A}} - r_j$, which is the length of time job j waits to be completed since its arrival. When the algorithm $\mathcal{A}$ and time t are clear from the context, we may drop them from the notation.

We assume the vector of rates $\mathbf{y}$ is constrained by a packing polytope $\mathcal{P}$ given by:

$$\mathcal{P} = \{B\mathbf{y} \leq \mathbf{1}; \qquad \mathbf{y} \geq 0\}, \tag{1}$$

where B has non-negative entries.

Problems Modeled by PSP. As mentioned before, the PSP problem captures and generalizes several widely studied scheduling problems. We discuss these problems in detail in Sections 1.2 and 2. Without going into details, PSP captures the following.

Multi-dimensional scheduling. Here a job's rate is an arbitrary concave function of the amount of resources it obtains in each dimension;
Unrelated machine scheduling. Here a job can execute at different speeds on different machines, and can be pre-empted and migrated across machines;
Generalized broadcast scheduling. Here jobs receive different speedups from different pages that are broadcast;
Routing flows. Here each job needs to route flow of given volume (size) between an arbitrary source and destination in a capacitated network; and

Multicast scheduling. Here each job must multicast a given amount of content to the entire network.

Online, Non-clairvoyant Scheduling Objectives. The class of scheduling algorithms we consider are constrained by several properties, all of which are naturally motivated by scheduling applications, both new and old.

— It is *online* and learns about job j only when it arrives. Before this point, $y_j = 0$.
— 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 $\mathbf{y}(t)$ at any real time t arbitrarily often. This allows for preemption as well as migration across physical machines contributing to resources 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 matrix B is known in advance to the scheduler and are independent of time, so that $\mathcal{P}$ 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 matrix B are defined over this universe. This is purely done to simplify our description and notation – in our applications, the polytope $\mathcal{P}$ 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 the above assumptions, we will investigate non-clairvoyant online algorithms that minimize the overall job latency, *i.e.*, the total weighted completion time $\sum_j w_j C_j$ and the 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. We will use the standard notion of *competitive ratio* for analyzing our algorithms. An online algorithm is said to be α-competitive if for *every* finite input instance that can be even *adversarial*, the cost incurred by the algorithm is at most α times the cost of an optimal offline solution to the instance.

As mentioned above, our main result in this paper is the analysis of a simple, non-clairvoyant algorithm termed *Proportional Fairness*, which we show is constant competitive for total completion time on *all* PSP instances, and is constant competitive for total job latency on a large sub-class of PSP instances.

Technical Note.. For the purpose of modeling problems as special cases of PSP, it will be more convenient to present $\mathcal{P}$ using auxiliary variables $\mathbf{x}$ as:

$$\mathcal{P} = \Big\{\mathbf{y} \;\mid\; \mathbf{y} = A\mathbf{x}; \qquad H\mathbf{x} \le \mathbf{1}; \qquad \mathbf{x} \ge 0\Big\}$$

where A and H have non-negative entries. It is easy to check that both representations capture general packing polytopes, and are hence equivalent. In fact, our results do not depend on the dimension of these polytopes, and hold for general concave rates, so that $y_j = f_j(\mathbf{x})$ for non-decreasing concave functions f_j, subject to $H\mathbf{x} \le \mathbf{1}$ and $\mathbf{x} \ge \mathbf{0}$. It is easy to check that under reasonable smoothness assumptions, such rates can be encoded by a packing polytope on $\mathbf{y}$ to arbitrary precision.

1.2. Running Example: Multidimensional Scheduling

Before proceeding further, we will present a concrete instantiation of PSP motivated by modern computing systems, which will serve as a running example in this section. 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, 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.

This general scheduling scenario termed *Multidimensional scheduling* 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 D different types of resources. In the context 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. At each time instant, the resources must be feasibly allocated among the jobs. If job j is allocated resource vector $\mathbf{x}_j = (x_{j1}, x_{j2}, \dots, x_{jD})$, then these must satisfy $\sum_j x_{jd} \leq R_d$ for all d. The rate y_j at which job j executes is determined by a non-decreasing concave function $y_j = u_j(\mathbf{x}_j)$ with $u_j(\mathbf{0}) = 0$. Borrowing from Economics, we term these functions as *utility functions*. Multidimensional scheduling can therefore be modeled by

$$\mathcal{P} = \left\{ y_j = u_j(\mathbf{x}_j) \ \ \forall j; \qquad \sum_j x_{jd} \leq R_d \ \ \forall d \in [D]; \qquad \mathbf{x} \geq 0 \right\} \tag{2}$$

We discuss several utility functions as we go along. In the special case of *Leontief* utilities, job j is associated with resource demand vector $\mathbf{f}_j = (f_{j1}, f_{j2}, ..., f_{jD})$ so that it requires f_{jd} amount of the d^{th} resource. When allocated resources $\mathbf{x}_j$, it is processed at a rate that is determined by its bottleneck resource, so that its rate is $y_j = \min_d (x_{jd}/f_{jd})$.

The multi-dimensional scheduling problem 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 paths, each having its own source and destination. 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 multidimensional 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 y 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. Note that the schedulers in a data center context typically have access to instantaneous resource requirements (the vectors $\mathrm{f_j}$), but are not typically able to estimate how large the jobs are in advance, *i.e.*, they are *non-clairvoyant*. They further are only aware of jobs when they arrive, so that they are *online* schedulers. This motivates our scheduling model.

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 very little theoretical analysis of this aspect; see [Bonald et al. 2006; Kelly et al. 2009] for recent queueing-theoretic analysis of network routing policies. This is the starting point of our paper – we formalize non-clairvoyant, online scheduling under packing constraints on rates as the general PSP problem, and present competitive algorithms for problems in this framework.

1.3. The Proportional Fairness Algorithm

We show positive results for both the completion time and flow time metrics using a simple algorithm that has been widely studied in the context of fairness in resource allocation, dating back to Nash [Nash 1950]. This is the Proportional Fairness (PF) algorithm [Nash 1950; Kelly et al. 1998; Ghodsi et al. 2011], which has also been called the *Nash Product*. Though the algorithm is ancient, our work is the first analysis of such an algorithm motivated by Economics in the context of adversarial scheduling theory.

Let $\mathcal{A}_t$ denote the set of jobs alive at time t. At time t, the rates are set using the solution to the following convex program, called the *Eisenberg-Gale convex program*. We term it the PF program. (See Section 3.1 for more details.)

$$\mathbf{y}^*(t) = \mathbf{argmax}\Big\{ \sum_{j \in \mathcal{A}_t} w_j \log y_j \;\mid\; B\mathbf{y} \le \mathbf{1}; \qquad \mathbf{y} \ge 0 \Big\}$$

At any time instant, the online scheduling algorithm simply solves the PF program and performs rate allocation according to its output. We note that the PF program does not require knowledge of job size p_j, and is hence a *non-clairvoyant* algorithm.

Though the PF algorithm is well-defined for any PSP instance, for multi-dimensional scheduling, the program has an intuitive explanation in terms of *market clearing*. Consider multidimensional scheduling (Eq (2)) with Leontief utility function $u_j(\mathbf{x_j}) = \min_d \left\{ \frac{x_{jd}}{f_{jd}} \right\}$. This utility function is a special case of homogeneous, concave utility functions of degree 1, meaning that $u_j(\alpha \mathbf{x_j}) = \alpha u_j(\mathbf{x_j})$ for all $\alpha \ge 0$. For such utilities, the PF algorithm implements a *competitive equilibrium* on the jobs [Jain and Vazirani 2010]. The KKT conditions on the convex program imply resource d has price λ_d per unit quantity. Job j has budget w_j, and sets its rate y_j by purchasing the best possible resources subject to these prices, *i.e.*,

$$y_j = \max \left\{ u_j(\mathbf{x_j}) \mid \sum_d \lambda_d x_{jd} \le w_j \right\}$$

The convex program optimum guarantees that there exists a set of prices $\{\lambda_d\}$ so that the market clears – all resources with non-zero price are completely allocated, i.e.,

$$\lambda_d > 0 \Rightarrow \sum_j x_{jd} = R_d \;\; \forall d$$

and $\mathbf{x}$ is feasible for $\mathcal{P}$. It is known from Economics [Varian 1976] that the resulting competitive equilibrium is Pareto-efficient (meaning all jobs cannot increase their rates simultaneously) and envy-free (meaning that no job prefers the allocation of resources that another job gets to its own allocation). In that sense, such an allocation is *fair* every time instant. We don't elaborate on this since it is not our main focus.

In order to show our positive results, we will crucially use the dual interpretation of the PF program via KKT conditions, and its connections to market clearing literature from Economics.

Monotone PSP. An important sub-class of the PSP problem is one where the rates found by the above convex program are monotone in the set of jobs present currently in the system. This class termed MONOTONE PSP is formally defined as follows. When the current set of jobs is S, let $y_j(S)$ denote the rate allocated by PF to job $j \in S$.

Definition 1.1 (*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.

Roughly speaking, in the subclass we define, each job's rate (or utility) can only decrease when more jobs are introduced into the system, thus into competition for limited resources. MONOTONE PSP captures several application scenarios, which we outline in Section 2.2, and has a natural connection to the notions of Gross Substitutes [Gul and Stacchetti 1999] and Eisenberg Gale markets [Jain and Vazirani 2010] in Economics. An important sub-class of MONOTONE PSP is polymatroidal utilities. This not only captures related machine scheduling [Gupta et al. 2012a; Im et al. 2014; Im et al. 2014], but also captures as special cases hitherto unstudied problems such as *single-sink flow routing* and *routing multicast trees* (video-on-demand). We defer the details to Section 2.2.1. It further includes multidimensional scheduling with *substitutes* utility functions; we highlight this special case since it continues our discussion of multidimensional scheduling with homogeneous utilities of degree 1. Such utilities are closely related to *Constant Elasticity of Scale* (CES) utilities, defined as:

$$u_j(\mathbf{x}_j) = \left(\sum_{d=1}^{D} c_{jd} x_{jd}^{\rho_j} \right)^{1/\rho_j} \tag{3}$$

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 ρ captures the extent of substitutability. Intuitively, this type of utilities capture substitute resources where deficit in one resource can be compensated by other resources. A special case as $\rho \to 0$ is termed *Cobb-Douglas* utilities: $u_j(\mathbf{x}_j) = \prod_{d=1}^{D} x_{jd}^{\alpha_{jd}}$, 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*.

These utilities have a natural connection to the concept of *Gross Substitutes* in Economics, and we use this connection to show in Theorem 1.7 that CES utility functions when $\rho \in [0, 1]$, along with some generalizations, belong to MONOTONE PSP. The generalization that we term *Resource Allocation with Substitutes* (RA-S) is presented in Section 2.2.2 and the result that such functions belong to MONOTONE PSP is presented in Appendix A[1].

1.4. Our Results

Completion Time Metric. For the weighted completion time metric, our main result is the *first* constant competitive non-clairvoyant algorithm for PSP. It also yields the first analysis of the Proportional Fairness algorithm in a general scheduling context.

THEOREM 1.2 (SECTION 3). *For the total weighted completion time objective for PSP, the PF algorithm is $O(1)$-competitive.*

This implies the first such result for a variety of applications that are special cases of the general PSP framework; these special cases have each been well-studied in their own right, and are found in Section 2. We summarize our new results in the following corollary.

COROLLARY 1.3. *The PF algorithm is an $O(1)$ competitive non-clairvoyant scheduling policy for the total weighted completion time metric for all the following problems:*

[1]We note that Leontief utilities correspond to CES utilities when $\rho \to -\infty$, and are therefore not monotone.

— *Unrelated machine scheduling;*
— *Multidimensional scheduling where the rate of a job is an arbitrary concave function of the resources it obtains;*
— *Multicommodity flow scheduling, where each job needs to route flow of given volume between a given source and sink; and*
— *Broadcast scheduling, where the speedup of jobs can be different for different pages that are broadcast.*

We now present some intuition for why our result is quite surprising. Consider multidimensional scheduling with Leontief utilities (Eq (2)) and recall that D is the number of resources (or the dimension). When there is only $D = 1$ 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). 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. It is not obvious how to generalize the most intuitively fair algorithm Round Robin (or Max-Min Fairness) to unrelated machine scheduling (see Section 2 for a definition) – in [Gupta et al. 2012a], 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 (Section 2) [Bansal et al. 2010]. Hence, we find the very existence of a unified, competitive, and fair algorithm for PSP quite surprising!

Flow Time Metric. We next consider the weighted flow time objective for PSP. We note that even for classical single machine scheduling, any deterministic algorithm is $\omega(1)$-competitive [Bansal and Chan 2009]. Further, in the unrelated machine setting, there is no online algorithm with a bounded competitive ratio [Garg and Kumar 2007]. Hence to obtain positive results, we appeal to speed augmentation which is a popular relaxation of the worst case analysis framework for online scheduling [Kalyanasundaram and Pruhs 2000]. Here, the online algorithm is given speed $s \geq 1$, and is compared to an optimal scheduler which is given a unit speed. More precisely, we compare our algorithm against an optimal omniscient solution which is constrained by the tighter constraint $H\mathbf{x} \leq \frac{1}{s}$. Note that speed augmentation is done purely for the sake of analysis and the algorithm is oblivious to it.

THEOREM 1.4 (SECTION 5). *For the total weighted flow time objective, 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.*

Given the generality of the PSP problem leads to strong lower bounds, we seek to find a subclass of problems that admit positive results. This is where we bring in the class MONOTONE PSP. Our main result is the following theorem.

THEOREM 1.5 (SECTION 4). *For the* MONOTONE PSP *problem, for any constant $\epsilon \in (0, 1/2)$, PF is $(e + \epsilon)$-speed, $O\left(1/\epsilon^2\right)$ competitive for minimizing total weighted flow time.*

This yields the first constant competitive algorithms for several problems presented in Section 2.2. We summarize the new results in the following corollary.

COROLLARY 1.6. *For any constant* $\epsilon \in (0, 1/2)$*, PF is* $(e + \epsilon)$*-speed,* $O\left(1/\epsilon^2\right)$ *competitive for minimizing weighted flow time for the following problems:*

— *Multidimensional scheduling when the utility functions are CES with* $\rho_j \in [0, 1]$*, and its generalization that we term RA-S (defined in Section 2.2.2);*
— *The setting where jobs need to route flow of given magnitude to the same sink in a capacitated network; and*
— *Scheduling multicast trees, where each job requires to multicast content to the entire network.*

The PF algorithm also yields an entirely different (and perhaps more intuitive) constant competitive algorithm for related machine scheduling, for which the first such result was only obtained recently [Im et al. 2014].

The multicast tree, single-sink flow, and related machine scheduling problems are special cases where the utility functions form a polymatroid. We present the definition and details in Section 2.2.1. Jain and Vazirani [Jain and Vazirani 2010] generalize the notion of market clearing to polymatroidal utilities and term the resulting markets Submodular Utility Allocation (SUA) markets. For these markets, the PF algorithm computes the market clearing solution. 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 SUA markets are competition monotone, which directly implies that polymatroidal utilities fall within MONOTONE PSP. This implies Theorem 1.5 holds for these problems.

As a final contribution, in Appendix A, we show that the class of CES utility functions and its generalization that we term RA-S (defined in Section 2.2.2) do indeed fall within MONOTONE PSP, so that Theorem 1.5 holds for these problems.

THEOREM 1.7 (APPENDIX A). *The* PF *algorithm is monotone for CES utility functions defined in Eq (3), and more generally for the RA-S utility functions defined in Eq (5) in Section 2.2.2.*

1.5. Our Techniques

The analysis of Theorem 1.2 is based on dual fitting. Dual fitting is popular for design and analysis of approximation and online algorithms, but two elegant works in [Anand et al. 2012; Gupta et al. 2012b] initiated dual fitting approach for online scheduling. As the name suggests, the key step in dual fitting based analysis is setting dual variables. In the simplest single machine setting, there is one type of dual variable to be set for each time, and the sum of those dual variables becomes the completion time when set to the total weight of unsatisfied jobs at the moment. This idea was successfully used and adapted even in the multiple machines setting [Anand et al. 2012].

However, in PSP, our task is more complicated – we are required to distribute the total weight of unsatisfied 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 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.

We next discuss how we prove Theorem 1.5. Our algorithm involves solving a convex optimization problem each time instant on the set of jobs in the system, in order to perform rate allocation. In contrast with single-machine scheduling settings, there is no closed form for the rates obtained by the PF algorithm. We start with the primal optimality condition (Proposition 4.1) and the monotonicity of PF algorithm in order to connect the rates found by PF against any other rate vector. We then use the framework of amortized local competitiveness [Im et al. 2011], where we define a potential function on the difference between the algorithm's job set and the optimal solution's job set. Our potential naturally generalizes potentials used for single resource scheduling [Fox et al. 2013]; however, our analysis becomes different in that it uses the optimality conditions, sometimes iteratively, to show competitive ratio. In hindsight, we believe primal optimality conditions naturally unify, simplify, and generalize many such analyses for single resource/machine scheduling.

This approach is different from the dual fitting we use to prove Theorem 1.2. Currently, we do not know how to extend dual fitting for analyzing the flow time objective, since this requires highly structured dual variables for the PF convex program, which may not exist. In contrast, the potential function approach directly works with the primal optimality conditions of the PF convex program, which yields a new analysis framework to the best of our knowledge.

1.6. Related Work

We present related work for special cases of PSP in Section 2. At a higher level, 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. 2011]; 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 the completion time metric; see [Williamson and Shmoys 2011] for a nice overview. In the online setting, several works [Chadha et al. 2009; Anand et al. 2012] give competitive clairvoyant algorithms for the weighted flow time objective on unrelated machines. Linear (or convex) programs and dual fitting approaches have been popular for online scheduling [Anand et al. 2012; Gupta et al. 2012b; Devanur and Huang 2014; Im et al. 2014; Angelopoulos et al. 2015]; 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. Therefore, both the model and techniques are very different.

We note that multidimensional scheduling has indeed been studied in the special case of one-dimension. In cluster computing, 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, a theoretical model a.k.a. arbitrary speed-up curves was introduced by Edmonds et al. [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 Γ_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. 2011; Edmonds et al. 2011; Edmonds and Pruhs 2012; Fox et al. 2013]. However, no previous work addresses parallelizability in multiple dimensions, and this is exactly the multidimensional scheduling model.

2. APPLICATIONS OF THE PSP FRAMEWORK

In this section, we present several concrete problems that fall in the PSP framework. For several applications, we present a mapping to the constraints in $\mathcal{P}$ to showcase the flexibility of our PSP framework. We note that our framework can handle various combinations of these problems as well, and achieves $O(1)$ competitive ratio for the weighted completion time objective for all of them!

2.1. Applications of the General PSP Framework

We first discuss applications that are captured by the general PSP, for which we obtain positive results for the weighted completion time objective, and subsequently discuss applications that belong to MONOTONE PSP, for which we obtain positive results for the weighted flow time objective. We have already described multidimensional scheduling in detail; we therefore focus on the other applications below.

All-or-nothing Resource Allocation. In the case of multidimensional scheduling with Leontief utilities in Eq (2), we 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 may need 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 $\mathcal{S}$ denote the collection of subsets of jobs that can be scheduled simultaneously. Let x_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.

$$\mathcal{P} = \left\{ y_j = \sum_{S : j \in S} x_S \ \ \forall j; \qquad \sum_{S \in \mathcal{S}} x_S \le 1; \qquad \mathbf{x} \ge 0 \right\} \tag{4}$$

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

Multicommodity Flow Scheduling. In this problem, we are given a capacitated graph $G(V, E)$, where the capacity of edge $e \in E$ is $c(e)$. Each job j requires routing p_j amount of flow from s_j to t_j in the graph. The flows have to satisfy the capacity constraints. Let P_j denote the set of paths between s_j and t_j. We can express this problem as a special case of PSP as follows:

$$\mathcal{P} = \left\{ y_j = \sum_{P \in P_j} x_P \ \ \forall j; \qquad \sum_{P | e \in P} x_P \le c(e) \ \ \forall e \in E; \qquad \mathbf{x} \ge 0 \right\}$$

A classical result of Kelly *et al.* [Kelly et al. 1998] shows that the TCP congestion control algorithm can be viewed as an implementation of proportional fairness in a distributed fashion. We note that the above problem does not specify the paths taken by the flow, so that job j could use different paths at different time steps. If the path that j needs to use is fixed, then it is easy to check that this problem becomes a special case of multi-dimensional scheduling with Leontief utilities (by treating each edge as a separate resource).

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. 2012a]. 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 x_{ij} denote the fraction of job j that is scheduled on machine i. Then:

$$\mathcal{P} = \Big\{ y_j = \sum_i s_{ij} x_{ij} \;\; \forall j; \qquad \sum_j x_{ij} \le 1 \;\; \forall i; \qquad \sum_i x_{ij} \le 1 \;\; \forall j; \qquad \mathbf{x} \ge 0 \Big\}$$

Note that any feasible x 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 $\mathsf{s_j}$ are only revealed when job j arrives. Our result gives the first $O(1)$-competitive algorithm for this problem for the total weighted completion time objective. Prior to our work, a variant of Round Robin was considered for the setting where machines are related and jobs are unweighted [Gupta et al. 2012a]; 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. After our work, another $O(1)$-competitive algorithm was found for unrelated machines [Im et al. 2014], but our result is still interesting since our algorithm is very different and much more general.

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 $\mathsf{s_j}$ of rates is only revealed when job j arrives. Therefore:

$$\mathcal{P} = \Big\{ y_j = \sum_{i \in [M]} s_{ij} x_i \;\; \forall j; \qquad \sum_{i \in [M]} x_i \le 1; \qquad \mathbf{x} \ge 0 \Big\}$$

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. 2012] 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 received considerable attention in theory [Gandhi et al. 2006; Bansal et al. 2008; Bansal et al. 2010; Edmonds et al. 2011; Im and Moseley 2012; Bansal et al. 2014] and has applications in multicast systems, LAN and wireless systems [Wong 1988; Acharya et al. 1995; Aksoy and Franklin 1999].

2.2. Applications of MONOTONE PSP

We now shift our discussion to applications of monotone PSP for which we show that PF is $O(1)$-speed $O(1)$-competitive for the weighted flow time metric (Theorem 1.5). As discussed in Section 1, we identify two subclasses of utility functions that induce monotone PSP– polymatroids and RA-S. These are described below.

2.2.1. Polymatroidal Utilities. This sub-class of PSP is given by the following polyhedron:

$$\mathcal{P} = \left\{ \sum_{j \in S} y_j \le v(S) \qquad \forall \text{ subsets of jobs } S \right\}$$

where the function $v(S)$ is a non-decreasing submodular function with $v(\phi) = 0$. The feasible region $\mathcal{P}$ is therefore a *polymatroid* [Schrijver 2003].

As mentioned before, Jain and Vazirani [Jain and Vazirani 2010] generalize the notion of market clearing to polymatroidal utilities and term the resulting markets Submodular Utility Allocation (SUA) markets. For these markets, the PF algorithm computes the market clearing solution. 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 SUA markets are competition monotone, which directly implies that polymatroidal utilities fall within MONOTONE PSP. This implies Theorem 1.5 holds for these problems.

Many natural resource allocation problems define polymatroids:

Single-sink Flow Scheduling. 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]. In conjunction with the classical result of Kelly *et al.* [Kelly et al. 1998] connecting the TCP congestion control algorithm to proportional fairness, 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 $G(V, E)$. 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 $\sum_j x_{jm} \le 1$ for all m; and each job is allocated at most one unit of machines, so that $\sum_m x_{jm} \le 1$ for all j. The rate of job j is $u_j(\mathbf{x}) = \sum_m s_m x_{jm}$. It is shown in [Feldman et al. 2008] that the space $\mathcal{P}$ of feasible rates define a polymatroid[2]. The aforementioned recent work [Im et al. 2014] gives a $O(1)$-speed $O(1)$-competitive algorithm for this problem, but our result is still interesting since our algorithm is very different and more natural.

2.2.2. Multidimensional Resource Allocation with Substitutes (RA-S). Recall the Multidimensional Scheduling setting from Eq (2). There are D divisible resources. We assume

[2]This is not stated as such in their paper, but follows as a simple corollary.

by scaling that each of which is available in unit supply, so that $R_d = 1$. Recall the Constant Elasticity of Scale (CES) utilities (Eq. 3) we discussed before. In this paper, we generalize CES functions to a broader class that we term *resource allocation with substitutes* or RA-S. These are given by:

$$u_j(\mathbf{x}_j) = \left(\sum_{d=1}^{D} (f_{jd}(x_{jd}))^{\rho_j} \right)^{1/\rho'_j} \qquad \text{where } \rho_j \in (0,1] \text{ and } \rho'_j \geq \rho_j \tag{5}$$

Here, the $\{f_{jd}\}$ are increasing, smooth, strictly concave functions, with $f_{jd}(0) = 0$. As before, the constraints $\mathcal{P}$ 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 $u_j(\mathbf{x}_j) = \prod_{d=1}^{D} (f_{jd}(x_{jd}))^{\alpha_{jd}}$, 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 paper, our algorithmic results also extend to a slightly different class of utilities of the form: $u_j(\mathbf{x}_j) = g_j\left(\sum_{d=1}^{D} f_{jd}(x_{jd})\right)$, where g_j is increasing, smooth, and strictly concave, with $g_j(0) = 0$.

We show in Appendix A that the RA-S problem belongs to MONOTONE PSP. We highlight the challenge in this proof. At a high level, the challenge is that the market equilibrium PF computes may not be equivalent to the more standard *Fisher market*. Suppose we view each job j as an agent who will use her budget w_j to maximize her utility in response to resource prices. In a Fisher market, each resource d is set to a price p_d that clears market: No resource is over allocated; for each resource with non-zero price, supply equals demand; and each agent spends its entire budget. The CES utilities for $\rho \in [0, 1]$ satisfy a property termed *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 (Definition 1.1).

For the RA-S utilities (Eq. (5)), 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 $\log u_j(\mathbf{x})$ 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 $\log u_j(\mathbf{x})$ is not zero when $\mathbf{x} = 0$; in fact it can be unbounded. We therefore need to show a stronger condition than the usual GS property in order to establish monotonicity.

3. WEIGHTED COMPLETION TIME FOR PSP: PROOF OF THEOREM 1.2

3.1. The Proportional Fairness (PF) Algorithm and Dual Prices

Recall the definition of $\mathcal{P}$ in Eq (1). We first set up useful notation that will be used throughout this paper. We will refer to our algorithm Proportional Fairness (PF) simply as $\mathcal{A}$. We let $\mathcal{A}_t := \{j \mid r_j \leq t < C_j^{\mathcal{A}}\}$ denote the set of outstanding/alive jobs at time t in the algorithm's schedule. Similarly, let $U_t := \{j \mid t < C_j^{\mathcal{A}}\}$ 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 $\mathbf{v_1}$ and $\mathbf{v_2}$ by $\mathbf{v_1} \cdot \mathbf{v_2}$. For a matrix B, $B_{i\cdot}$ denotes the i^{th} row (vector) of matrix B. Likewise, $B_{\cdot i}$ denotes the i^{th} column vector of matrix B. The indicator variable $\mathtt{1}()$ becomes 1 iff the condition in the parentheses is satisfied, otherwise 0.

As mentioned before, $C_j^{\mathcal{A}}$ denotes job j's completion time in $\mathcal{A}$'s schedule. Let $F_j^{\mathcal{A}} := C_j^{\mathcal{A}} - r_j$ 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.

At each time t (more precisely, either when a new job arrives or a job is completed), PF solves the following convex program.

$$\begin{aligned} \max \sum_{j \in \mathcal{A}_t} w_j \log y_j & & \qquad (\mathsf{CP_{PF}}) \\ s.t. \quad B\mathbf{y} \le \mathbf{1} & & \\ y_j = 0 & \qquad \forall j \notin \mathcal{A}_t & \end{aligned}$$

Then (PF) processes each job j at a rate of y^*_{jt} where y^*_{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, $\mathcal{A}_t$ does. For compact notation, we use a vector changing over time by adding t to subscript – for example, $\mathbf{y}^*_t$ denotes the vector $\{y^*_{jt}\}_j$. Observe that the constraint $\mathbf{y} \ge \mathbf{0}$ is redundant since $y^*_j > 0$ for all $j \in \mathcal{A}_t$.

The dual of $\mathsf{CP_{PF}}$ has variables $\gamma_{dt}, d \in [D]$ corresponding to the primal constraints $B_{d\cdot} \cdot \mathbf{y} \le 1$. Let $\gamma_t := (\gamma_{1t}, \gamma_{2t}, ..., \gamma_{Dt})$. By the KKT conditions [Boyd and Vandenberghe 2004], any optimal solution $\mathbf{y}^*$ for $\mathsf{CP_{PF}}$ must satisfy the following conditions for some γ^*:

$$\gamma^*_{dt} \cdot (B_{d\cdot} \cdot \mathbf{y}^*_t - 1) = 0 \qquad \forall t, d \in [D] \qquad (6)$$

$$\frac{w_j}{y^*_{jt}} = B_{\cdot j} \cdot \gamma^*_t \qquad \forall t, j \in \mathcal{A}_t \qquad (7)$$

$$\gamma^*_t \ge 0 \qquad \forall t \qquad (8)$$

3.2. Main Analysis

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].

$$\begin{aligned} \min \sum_{t,j} w_j \cdot \frac{t}{p_j} \cdot y_{jt} & & \qquad (\mathsf{PRIMAL}) \\ s.t. \quad \sum_{t \ge r_j} \frac{y_{jt}}{p_j} \ge 1 & \qquad \forall j \in U_0 & \\ B \cdot \mathbf{y}_t \le \mathbf{1} & \qquad \forall t \ge 0 & \\ y_{jt} \ge 0 & \qquad \forall t, j & \end{aligned}$$

The variable y_{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 1.2, will not need speed augmentation. The optimal solution with speed $1/s$ must satisfy the following LP.

$$\min \sum_{t,j} w_j \cdot \frac{t}{p_j} \cdot y_{jt} \qquad (\mathsf{PRIMAL}_s)$$

$$\begin{aligned} s.t. \quad \sum_{t \geq r_j} \frac{y_{jt}}{p_j} &\geq 1 && \forall j \in U_0 \\ B \cdot (s\mathbf{y}_t) &\leq \mathbf{1} && \forall t \geq 0 \\ y_{jt} &\geq 0 && \forall j, t \geq 0 \end{aligned}$$

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

$$\max \sum_j \alpha_j - \sum_{d,t} \beta_{dt} \qquad (\mathsf{DUAL_s})$$

$$\begin{aligned} s.t. \quad \frac{\alpha_j}{p_j} - sB_{\cdot j} \cdot \beta_t &\leq w_j \cdot \frac{t}{p_j} && \forall j, t \geq r_j && (9) \\ \alpha_j &\geq 0 && \forall j && (10) \\ \beta_{dt} &\geq 0 && \forall d, t && (11) \end{aligned}$$

We will set the dual variables α_j and β_{dt} using the optimal solution of $\mathsf{CP}_{\mathsf{PF}}$, y^*_{jt}, and the corresponding dual variables γ^*_{dt}. The following proposition shows the outcome we will derive by dual fitting.

PROPOSITION 3.1. *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 $\mathcal{A}$. Then $\mathcal{A}$ 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 any feasible solution $\mathbf{y}_t$ for PRIMAL is also feasible for PRIMAL_s when the $\mathbf{y}_t$ is stretched out horizontally by a factor of s – the new schedule $\mathbf{y}'_t$ is defined as $\mathbf{y}'_{(st)} = (1/s)\mathbf{y}_t$ 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 $U_t := \{j \mid j < C^{\mathcal{A}}_j\}$ 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 $\mathcal{A}_t := \{j \mid r_j \leq j < C^{\mathcal{A}}_j\}$ 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 $\mathbf{y}^*_t$ of $\mathsf{CP}_{\mathsf{PF}}$, and its dual variables γ^*_t. We will define α_{jt}, and set $\alpha_j := \sum_t \alpha_{jt}$ for all j.

Let q_{jt} denote the size of job j processed at time t. Define ζ_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.

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

We continue to define β_{dt} as $\beta_{dt} := \sum_{t' \geq t} \frac{1}{s} \zeta_{t'} \gamma^*_{dt'}$. We now show that this definition of α_{jt} and β_{dt} makes DUAL_s's objective to be at least $\Omega(1)$ times the objective of our algorithm.

LEMMA 3.2. $\sum_j \alpha_j \geq (1/2) \sum_j w_j C_j^{\mathcal{A}}$.

PROOF. At each time t, jobs in U_t contribute to $\sum_j \alpha_{jt}$ by at least half of the total weight of jobs in U_t. □

LEMMA 3.3. *For any time t, $\sum_d \gamma^*_{dt} = \sum_{j \in \mathcal{A}_t} w_j \leq W_t$.*

PROOF.

$$\sum_d \gamma^*_{dt} = \sum_d \gamma^*_{dt} (B_{d\cdot} \cdot \mathbf{y}^*_t) = \sum_d \gamma^*_{dt} \sum_{j \in \mathcal{A}_t} B_{dj}\, y^*_{jt} = \sum_{j \in \mathcal{A}_t} y^*_{jt} (B_{\cdot j} \cdot \gamma^*_t) = \sum_{j \in \mathcal{A}_t} y^*_{jt} \frac{w_j}{y^*_{jt}} \leq W_t$$

The first and last equalities are due to the KKT conditions (6) and (7), respectively. □

LEMMA 3.4. *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 $\Big((\frac{1}{2})^k W_t, (\frac{1}{2})^{k-1} W_t \Big]$. Now consider any fixed $k \geq 1$. We upper bound the contribution of M_k to $\sum_d \beta_{dt}$, that is $\frac{1}{s} \sum_{t' \in M_k} \sum_d \zeta_{t'} \gamma^*_{dt'}$. Towards this end, we first upper bound $\sum_{t' \in M_k} \zeta_{t'} \leq 4$. 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'.

$$\begin{aligned} \sum_{t' \in M_k} \zeta_{t'} &\leq \sum_{t' \in M_k} 2 \sum_{j \in \mathcal{A}_{t'}} \frac{w_j}{W_{t'}} \cdot \mathbb{1}\Big(\frac{q_{jt'}}{p_j} \geq \zeta_{t'} \Big) \cdot \frac{q_{jt'}}{p_j} \leq 2 \frac{1}{(1/2)^k W_t} \sum_{t' \in M_k} \sum_{j \in U_{t'}} w_j \frac{q_{jt'}}{p_j} \\ &\leq 2 \frac{1}{(1/2)^k W_t} (1/2)^{k-1} W_t = 4 \end{aligned}$$

The first inequality follows from the definition of $\zeta_{t'}$: the total weight of jobs in $U_{t'}$ with $\frac{q_{jt'}}{p_j} \geq \zeta_{t'}$ is at least half the total weight of jobs in $U_{t'}$. The second inequality is due to the fact that $W_{t'} \geq (\frac{1}{2})^k W_t$ 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,

$$\begin{aligned}
\sum_d \beta_{dt} &= \frac{1}{s}\sum_{t'\geq t}\sum_d \zeta_{t'}\gamma^*_{dt'} = \frac{1}{s}\sum_{k\geq 1}\sum_{t'\in M_k}\zeta_{t'}\sum_d \gamma^*_{dt'} \\
&\leq \frac{1}{s}\sum_{k\geq 1}\sum_{t'\in M_k}\zeta_{t'}W_{t'} && \text{[By Lemma 3.3]} \\
&= \frac{1}{s}\sum_{k\geq 1}4(1/2)^{k-1}W_t && \text{[By definition of } M_k \text{ and the fact } \textstyle\sum_{t'\in M_k}\zeta_{t'}\leq 4] \\
&\leq \frac{8}{s}W_t
\end{aligned}$$

□

COROLLARY 3.5. $\sum_{d,t}\beta_{dt} \leq \frac{8}{s}\sum_j w_j C_j^{\mathcal{A}}$.

From Lemma 3.2 and Corollary 3.5, we derive that the objective of DUAL_s is at least half of PF' total weighted completion time when $s = 32$. By Lemma 3.1, it follows that the algorithm PF is 64-competitive for the objective of minimizing total weighted completion time.

It now remains to show all the dual constraints are satisfied. Observe that the dual constraint (10) is trivially satisfied. Also the constraint (11) is satisfied due to the KKT condition (8).

We now focus on the more interesting dual constraint (9) to complete the analysis of Theorem 1.2.

LEMMA 3.6. *The dual constraint (9) is satisfied.*

PROOF.

$$\begin{aligned}
\frac{\alpha_j}{p_j} - w_j\frac{t}{p_j} &\leq \sum_{t'\geq t}\frac{\alpha_{jt'}}{p_j} && \text{[Since } \alpha_{jt'}\leq w_j \text{ for all } t'] \\
&= \sum_{t'\geq t}\frac{w_j}{p_j}\cdot \mathbb{1}\Big(\frac{q_{jt'}}{p_j}\leq \zeta_{t'}\Big) = \sum_{t'\geq t}\frac{w_j}{q_{jt'}}\cdot\frac{q_{jt'}}{p_j}\cdot \mathbb{1}\Big(\frac{q_{jt'}}{p_j}\leq\zeta_{t'}\Big) \\
&= \sum_{t'\geq t}\frac{w_j}{y^*_{jt'}}\cdot\frac{q_{jt'}}{p_j}\cdot \mathbb{1}\Big(\frac{q_{jt'}}{p_j}\leq\zeta_{t'}\Big) && \text{[Since } q_{jt'} = y^*_{jt'}] \\
&\leq \sum_{t'\geq t} B_{\cdot j}\cdot(\zeta_{t'}\gamma^*_{t'}) && \text{[By the KKT condition (7)]} \\
&= sB_{\cdot j}\cdot\beta_t && \text{[By definition of } \beta_t]
\end{aligned}$$

□

4. WEIGHTED FLOW TIME FOR MONOTONE PSP: PROOF OF THEOREM 1.5

As before, let $\mathcal{A}_t$ 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 $t\in[r_j, C_j]$. Recall that Proportional Fairness algorithm computes a rate vector $\mathbf{y}_t$ that maximizes $\sum_{j\in A_t} w_j\log y_j$ subject to $\mathbf{y}\in\mathcal{P}$.

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 4.1 (OPTIMALITY CONDITION). *Let $\mathbf{y} \in \mathcal{P}$ 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 $f(\mathbf{y}) = \sum_{j \in S} w_j \log y_j$. We have $\frac{\partial f(\mathbf{y}^*)}{\partial y_j} = \frac{w_j}{y_j^*}$. The optimality of $\mathbf{y}^*$ implies $\nabla f(\mathbf{y}^*) \cdot (\mathbf{y} - \mathbf{y}^*) \leq 0$ for all $\mathbf{y} \in \mathcal{P}$. 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 1.1 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.

We use amortized local competitiveness to show Theorem 1.5. 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 4.1 to bound how the potential function changes when the algorithm processes jobs.

Focus on some time instant t, and define the following quantities. 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_{jt}^O denote the remaining size of the job in OPT's schedule. Define a job j's *lag* as $\tilde{p}_{jt} = \max(0, p_{jt} - p_{jt}^O)$. The quantity $\tilde{p}_{jt}$ indicates how much our algorithm is behind the optimal schedule in terms of job j's processing. Let $L_t = \{j \in \mathcal{A}_t \mid \tilde{p}_{jt} > 0\}$. Note that $\mathcal{A}_t \setminus L_t \subseteq O_t$.

Consider the jobs in increasing order of arrival times, and number them $1, 2, \ldots$ in this order. Let $\mathcal{A}_t^{\leq j} = \mathcal{A}_t \cap \{1, 2, \ldots, j\}$. 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:

$$\Phi(t) = \frac{1}{\epsilon} \sum_{j \in \mathcal{A}_t} w_j \frac{\tilde{p}_{jt}}{y_j^*(\mathcal{A}_t^{\leq j})}$$

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 completeness.

CLAIM 4.2. *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 $y_j^*(\mathcal{A}_t^{\leq j})$ 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 $\mathcal{A}_t$ plus the job j' that just arrived, and $j' \notin A_t$. For this job, $\tilde{p}_{j't} = 0$. Furthermore, this job does not affect $y_j^*(\mathcal{A}_t^{\leq j})$ for any $j \in \mathcal{A}_t$, since $j' \notin \mathcal{A}_t^{\leq j}$. Therefore, the potential does not change when a job arrives. Similarly, suppose a job j' is completed by OPT but $\mathcal{A}_t$ 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 $\mathcal{A}_t$. We have $\tilde{p}_{j't} = 0$. This departure can change $y_j^*(\mathcal{A}_t^{\leq j})$ for $j \in \mathcal{A}_t$ 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. □

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,

$W(S) = \sum_{j \in S} w_j$.

$$W(\mathcal{A}_t) + \frac{d}{dt}\Phi(t) \leq \frac{2}{\epsilon^2} W(O_t) \tag{12}$$

Suppose all jobs are completed by PF and OPT by time T. Observe that $\int_{t=0}^{T} \frac{d}{dt}\Phi(t)dt \geq 0$ from the facts that the discontinuous changes to Φ are all non-positive, and $\Phi(0) = \Phi(T) = 0$. Then integrating the above inequality over time, we have:

$$\int_{t=0}^{T} W(\mathcal{A}_t)dt + \int_{t=0}^{T} \frac{d}{dt}\Phi(t)dt \leq \frac{2}{\epsilon^2} \int_{t=0}^{T} W(O_t)dt$$

Note that the first term above is the weighted flow time of PF, the second term is non-negative, and the RHS is the weighted flow time of OPT. This will complete the proof of Theorem 1.5.

4.1. Proving Inequality (12)

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 $\frac{d}{dt}\Phi|_{\mathcal{A}}$ denote the potential changes due to OPT's processing and PF's processing respectively. Note that $\frac{d}{dt}\Phi = \frac{d}{dt}\Phi|_{\mathcal{A}} + \frac{d}{dt}\Phi|_O$.

LEMMA 4.3. $\frac{d}{dt}\Phi|_O \leq \frac{1}{\epsilon} W(\mathcal{A})$.

PROOF. For job $j \in \mathcal{A}$, suppose OPT assigns rate y_j^O. Then, $\frac{d}{dt}\tilde{p}_j \leq y_j^O$ for $j \in \mathcal{A}$, due to OPT's processing. Therefore, the change in potential is upper bounded by:

$$\frac{d}{dt}\Phi|_O \leq \frac{1}{\epsilon} \sum_{j \in \mathcal{A}} w_j \frac{y_j^O}{y_j^*(\mathcal{A}_t^{\leq j})} \leq \frac{1}{\epsilon} \sum_{j \in \mathcal{A}} w_j \frac{y_j^O}{y_j^*(\mathcal{A})}$$

The inequality above follows from the monotonicity of the PF algorithm, since $\mathcal{A}^{\leq j} \subseteq \mathcal{A}$. Using Proposition 4.1, the RHS is at most $W(A)$. This completes the proof. □

We now bound $\frac{d}{dt}\Phi|_{\mathcal{A}}$, 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 $W(L) \leq (1-\epsilon)W(\mathcal{A})$. Since $\mathcal{A} \setminus L \subseteq O$, we have $W(O) \geq \epsilon W(\mathcal{A})$. Since $\frac{d}{dt}\Phi|_{\mathcal{A}} \leq 0$, we have:

$$W(\mathcal{A}) + \frac{d}{dt}\Phi \leq W(\mathcal{A}) + \frac{d}{dt}\Phi|_O \leq \frac{2}{\epsilon} W(\mathcal{A}) \leq \frac{2}{\epsilon^2} W(O)$$

where the second inequality follows from Lemma 4.3.

Case 2.. The more interesting case is when $W(L) \geq (1-\epsilon)W(\mathcal{A})$. For $j \in L$, we have $\frac{d}{dt}\tilde{p}_j = y_j^*(\mathcal{A})$ due to PF's processing, by the definition of $y_j^*(\mathcal{A})$. Therefore,

$$\epsilon \cdot \frac{d}{dt}\Phi|_{\mathcal{A}} \leq -\sum_{j \in L} w_j \frac{y_j^*(\mathcal{A})}{y_j^*(\mathcal{A}^{\leq j})}$$

For notational convenience, let $|S| = \kappa$, and number the jobs in $\mathcal{A}$ in increasing order of arrival time as $1, 2, \ldots, \kappa$. For $k > j$ and $k \leq \kappa$, let $\alpha_{jk} = \frac{y_j^*(\mathcal{A}^{\leq k-1})}{y_j^*(\mathcal{A}^{\leq k})}$. 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 4.1 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 $y_j^*(\mathcal{A}^{\leq k})$, and this goes into the denominator in Proposition 4.1. We consider $y_j^*(\mathcal{A}^{\leq k-1})$ for $j < k$, and $y_k^*(\mathcal{A}^{\leq k-1}) = 0$ as a different set of rates that go into the numerator in Proposition 4.1. This yields:

$$\sum_{j=1}^{k-1} w_j \frac{y_j^*(\mathcal{A}^{\leq k-1})}{y_j^*(\mathcal{A}^{\leq k})} \leq \sum_{j=1}^{k} w_j$$

Observing that $\frac{y_j^*(\mathcal{A}^{\leq k-1})}{y_j^*(\mathcal{A}^{\leq k})} = 1 + \delta_{jk}$, 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:

$$\sum_{k=1}^{\kappa}\sum_{j=1}^{k-1} w_j \delta_{jk} = \sum_{j=1}^{\kappa} w_j \left(\sum_{k=j+1}^{\kappa} \delta_{jk}\right) \leq W(\mathcal{A}) \qquad \Longrightarrow \qquad \sum_{j\in L} w_j \left(\sum_{k=j+1}^{\kappa} \delta_{jk}\right) \leq W(\mathcal{A})$$

Let $\Delta_j = \sum_{k=j+1}^{\kappa} \delta_{jk}$, so that the above inequality becomes $\sum_{j\in L} w_j \Delta_j \leq W(\mathcal{A})$. Now observe that

$$\frac{y_j^*(\mathcal{A})}{y_j^*(\mathcal{A}^{\leq j})} = \prod_{k=j+1}^{\kappa} \frac{1}{\alpha_{jk}} = \prod_{k=j+1}^{\kappa} \frac{1}{1+\delta_{jk}} \geq \exp\left(-\sum_{k=j+1}^{\kappa} \delta_{jk}\right) = \exp(-\Delta_j)$$

We used the fact that $\delta_{jk} \geq 0$ for all j,k. Therefore,

$$\epsilon \cdot \frac{d}{dt}\Phi|_{\mathcal{A}} \leq -\sum_{j\in L} w_j \frac{y_j^*(\mathcal{A})}{y_j^*(\mathcal{A}^{\leq j})} \leq -\sum_{j\in L} w_j \exp(-\Delta_j)$$

Since $\sum_{j\in L} w_j \Delta_j \leq W(\mathcal{A})$, the RHS is maximized when $\Delta_j = W(\mathcal{A})/W(L) \leq 1/(1-\epsilon)$. This implies:

$$\epsilon \cdot \frac{d}{dt}\Phi|_{\mathcal{A}} \leq -\sum_{j\in L} w_j \exp(-W(\mathcal{A})/W(L)) \leq -W(L)\exp(-1/(1-\epsilon)) \leq -\frac{1-2\epsilon}{e} W(\mathcal{A})$$

for $0 < \epsilon < 1/2$. Therefore, if we run PF at speed $(e + 3\epsilon)$, we have: $\frac{d}{dt}\Phi|_{\mathcal{A}} \leq -\left(1+\frac{1}{\epsilon}\right) W(\mathcal{A})$. Therefore,

$$W(\mathcal{A}) + \frac{d}{dt}\Phi|_O + \frac{d}{dt}\Phi|_{\mathcal{A}} \leq W(\mathcal{A}) + \frac{1}{\epsilon}W(\mathcal{A}) - \left(1+\frac{1}{\epsilon}\right) W(\mathcal{A}) \leq 0 \leq W(O)$$

This completes the proof of Inequality (12) and hence of Theorem 1.5.

5. FLOW TIME LOWER BOUND: PROOF OF THEOREM 1.4

In this section, we prove the lower bound claimed in Theorem 1.4. Towards this end, we will first prove a lower bound for makespan.

THEOREM 5.1. *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 5.1 implies the desired Theorem 1.4.

Proof of [Theorem 1.4] Let $\mathcal{I}_0$ denote the lower bound instance consisting of N unweighted jobs with arrival times 0 that establishes the lower bound stated in Theorem 5.1. By scaling, we can w.l.o.g. assume that the optimal (offline) makespan for $\mathcal{I}_0$ is 1. For any fixed $\epsilon > 0$, we create $N^{1/\epsilon}$ copies of instance $\mathcal{I}_0$, $\{\mathcal{I}_e\}_{e \in \{0,1,2,\ldots,N^{1/\epsilon}-1\}}$ where all jobs in $\mathcal{I}_e$ arrive at time e. There is a global constraint across all instances $\{\mathcal{I}_e\}$ that enforces the scheduler to pick only one instance to work on at any point in time. More precisely, if each instance is constrained by polytope $B_e \mathbf{x}_{et} \leq 1$, then using auxiliary variables s_{et} we connect the polytopes as follows:

$$B_e \mathbf{x}_{et} \leq s_e \qquad \forall t, e \tag{13}$$

$$\sum_e s_{et} \leq 1 \qquad \forall t, e \tag{14}$$

$$s_{et} \geq 0 \qquad \forall t, e, \tag{15}$$

Here, for simplicity we omitted the obvious constraints on $\mathbf{x}_{et}$ we have due to jobs arrival, and $\mathbf{x}_{et} \geq 0$. It is easy to check that this forms a downward closed polytope by observing that the feasible solution set is downward closed and convex. Due to Eq. (14), the scheduler is forced to work on only one instance at any instantaneous time. Or equivalently, if the scheduler chooses to work on $\mathcal{I}_e$ using s_e fractional of its total capacity at an instantaneous moment, then it can process jobs in $\mathcal{I}_e$ at a rate of s_e times that it could if it used its whole capacity on processing $\mathcal{I}_e$. Note that we are not imposing precedence constraints; the scheduler can work on instances $\{\mathcal{I}_e\}$ in arbitrary order.

Then, any deterministic non-clairvoyant algorithm that is given speed less than half the lower bound on makespan stated in Theorem 5.1 cannot complete all jobs in each $\mathcal{I}_e$ within 2 time steps. It is easy to see that there are at least $e/2$ jobs alive during $[e, e+1)$ for any $e \in \{0, 1, 2, ..., N^{1/\epsilon} - 1\}$. Hence any deterministic non-clairvoyant algorithm has total flow time $\Omega(N^{2/\epsilon})$. In contrast, the optimal offline scheduler 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 $\mathcal{I}_e$, completing the proof of Theorem 1.4. □

Henceforth, we will focus on proving Theorem 5.1. Our lower bound instance comes from single source routing in a tree network with "multiplicative speed propagation". 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 fist discuss the one-level tree instance, and then describe the full lower bound instance. Throughout this section, we refer to an arbitrary non-clairvoyant algorithm as $\mathcal{A}$.

One-level instance: $\mathcal{I}(1)$. The root ρ has $\Delta_1 := 4$ routers where only one router has 2-speed and the other routers have 1-speed. There are Δ_1 packets (or equivalently jobs) to be routed to the root ρ. 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 non-clairvoyant $\mathcal{A}$ 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 size-3 job at time $\Delta_1/(\Delta_1 + 1) + 1 = 9/5$. Observe that giving more 1-speed routers does not help the online algorithm since the main challenge comes from finding the big job and processing it using a faster router.

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

At any time, a feasible scheduling decision is a matching between routers $\mathcal{R}_v$ and nodes $\mathcal{C}_v$ for all non-leaf nodes v; when some jobs complete, this naturally extends to an injective mapping from $\mathcal{C}_v$ to $\mathcal{R}_v$. 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 sequence of routers. Let z_g denote the indicator variable for g. Let $\eta_j(g)$ denote the number of 2-speed routers in the sequence of routers given to j in schedule g where job j is processed at a rate of $2^{\eta_j(g)}$. We can formally describe this setting using PSP as follows:

$$\mathcal{P} = \left\{ y_j \leq \sum_g 2^{\eta_j(g)} z_g \ \forall j; \quad \sum_g z_g \leq 1; \quad \mathbf{y} \geq 0; \quad \mathbf{z} \geq 0 \right\} \tag{16}$$

We now describe job sizes. Recall that $\mathcal{A}$ is non-clairvoyant, meaning that it faces the challenge of finding big jobs. Each non-leaf node v of depth less than $h-1$ has one special "big" child amongst its $\Delta_h = 4^h$ children $\mathcal{C}_v$ – a big child will have bigger jobs in its subtree, which will be formally stated shortly. For each node v of depth $h-1$, define η_v to be the number of "big" nodes on the unique path from v to the root including v itself. Then v's children (jobs), $\mathcal{C}_v$ 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 $\mathcal{I}(h)$.

The final instance will be $\mathcal{I}(H)$. Since $\mathcal{I}(H)$ has 4^{H^2} jobs, we have $H = \Theta(\sqrt{\log n})$. For a visualization of the instance, see Figure 1.

LEMMA 5.2. *There is an offline schedule that completes all jobs by time 2.*

PROOF. For each node v of depth less than $H-1$, we assign its unique faster router in $\mathcal{R}_v$ to its unique big node in $\mathcal{C}_v$. At the bottom level, jobs can be mapped to any routers. Consider any non-leaf node v of depth $H-1$. The lemma follows since all jobs in $\mathcal{C}_v$ have size at most $2^{\eta_v+1} - 1$, and all of them are processed at a rate of at least 2^{η_v}. $\square$

We now discuss how $\mathcal{A}$ performs for the instance $\mathcal{I}(H)$. We first give a high-level overview of the adversary's strategy which forces $\mathcal{A}$ 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 $\mathcal{A}$ comes from the fact that $\mathcal{A}$ does not know which jobs/nodes are big, hence cannot assign big nodes to faster routers – for example, all subtrees induced by the root's children are indistinguishable to $\mathcal{A}$. Such sub-optimal decisions will accrue over layers and will yield a gap $\Omega(H)$. To simplify our argument,

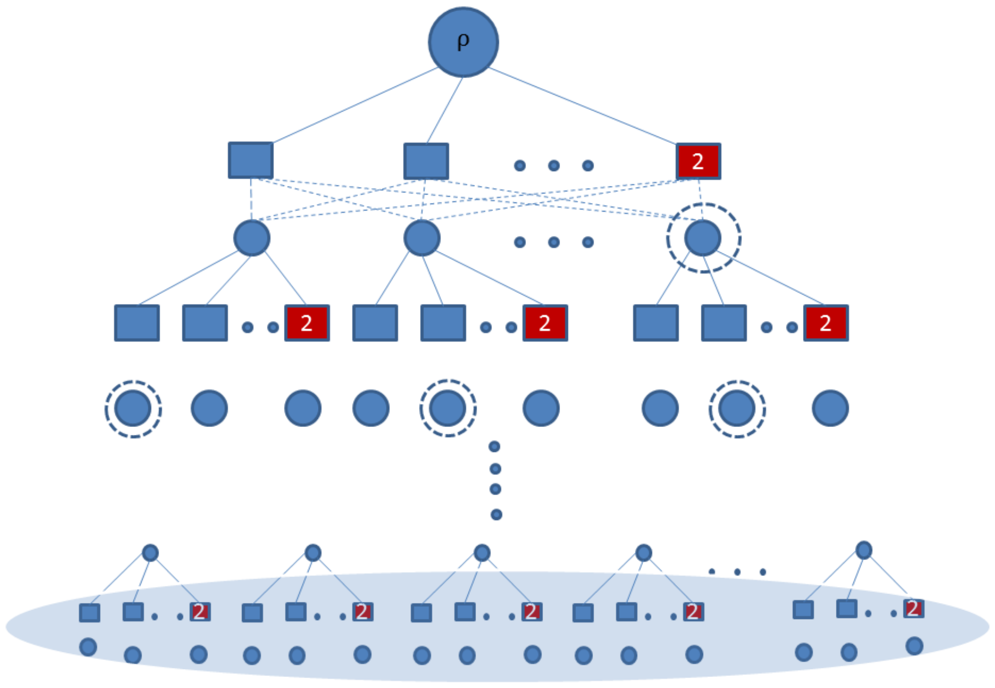

Fig. 1. Routers are represented by rectangles and jobs/nodes by circles; jobs are leaf nodes. The tree has height $H = \Theta(\sqrt{\log n})$. Each non-leaf node has 4^H children, and for each node of depth less than $H-1$, there is only one big child node, shown by dotted circle, which is hidden from the online algorithm. Each node/job (except the root) has to be mapped to a router attached to its parent node. The sizes of jobs having the same parent node depend on the number of big nodes on the path from the job to the root.

we allow the adversary to *decrease* job sizes. That is, at any point in time, the adversary observes the non-clairvoyant algorithm $\mathcal{A}$'s schedule, and can decrease the size of any alive job, or can even remove it. This is a valid strategy for the adversary since the algorithm $\mathcal{A}$ is non-clairvoyant, and can only be better off for smaller jobs. Also, this does not increase the optimal solution's makespan.

Consider any non-leaf node $v \neq \rho$. Let us say that the subtree T_v rooted at v is big (resp. small) if the node v is big (resp. small). If the node v has used the unique 2-speed router in $\mathcal{R}_{u(v)}$ for $1/2^{H+1}$ 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 that is essentially equivalent to $\mathcal{I}(H-1)$. By repeating this, the online algorithm will be forced to have a makespan of at least $H/2$. We show two crucial properties.

(1) At time $1/2$, each alive non-leaf node has at least 4^{H-1} children.
(2) Any alive job has been processed by strictly less than 1.

The first property easily follows since each non-leaf node has 4^H children, and at most 2^H of its children can be removed by time 1/2. To see why the second property holds, consider any job j. Observe that each of j's ancestors (including j itself) used the 2-speed router only for $1/2^{H+1}$ time steps. Here the maximum processing for job j can be achieved when j's all ancestors use the 2-speed routers simultaneously for $1/2^{H+1}$ time steps, which is most $1/2$. Also note that the total length of times job j is processed by the combination of 1-speed routers only is at most $1/2$, and strictly less than 1/2 if any 2-speed routers were used. 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 T_{v^*} of those alive, and declare it is big. The adversary keeps only the big subtree incident to the root. By the same argument, for any non-leaf node $v \neq \rho$ of depth less than $H-1$, the adversary can remove all nodes in $\mathcal{C}_v$ except exactly 4^{H-1} including the big node. The adversary can assume w.l.o.g. that all the remaining jobs have been processed exactly by one unit by decreasing job sizes appropriately. Then, of all jobs sharing the same parent node, the adversary keeps exactly 4^{H-1} jobs with sizes greater than 1. This is well defined since there are exactly $4^H - 4^{H-1}$ unit-sized jobs sharing the same parent node. Observe that each alive job has remaining size $2^1 + 2^2 + ... + 2^k = 2(2^k - 1)$ for some $k \geq 1$.

Since T_{v^*} is the only subtree incident to the root, we can assume w.l.o.g. that $\mathcal{A}$ assigns the 2-speed router to v from now on. This has the effect of decreasing each job's remaining size by half. Hence, the subtree T_{v^*} exactly coincides with the instance $\mathcal{I}(H-1)$. This will allow the adversary recurse on the instance $\mathcal{I}(H-1)$, thereby making $\mathcal{A}$'s makespan no smaller than $H/2 = \Omega(\sqrt{\log n})$. This, together with Lemma 5.2, establishes the claimed lower bound on makespan, thus proving Theorem 5.1 and Theorem 1.4.

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 $\mathcal{S} := \mathcal{S}(0)$ of possible instances that the adversary can use. The adversary will gradually decrease the instance space $\mathcal{S}(t)$ depending on the algorithm's choices till time t. The adversary can only reduce $\mathcal{S}(t)$ in time t. The deterministic algorithm $\mathcal{A}$ cannot distinguish between instances in $\mathcal{S}(t)$ at time t, and hence must behave exactly the same by time t for all the instances in $\mathcal{S}(t)$. In this sense, all instances in $\mathcal{S}(t)$ are indistinguishable to $\mathcal{A}$ by time t.

All instances in $\mathcal{S}$ follow the same polytope constraints for $\mathcal{I}(H)$, but there are two factors that make the pool of instances, $\mathcal{S}$ rich. The first factor is "hidden" job IDs: Each non-leaf node v of depth less than $H-1$ has only one big child, and it can be any of its children, $\mathcal{C}_v$. In other words, this is completely determined by a function ψ that maps each of such non-leaf nodes, v to one of v's children, $\mathcal{C}_v$. Consider any fixed ψ. Then for each non-leaf node v of depth $H-1$, 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's ID, we mean the job in the common instance that corresponds to job j. The second factor is "flexible" job sizes. Note that in $\mathcal{I}(H)$, a job j's ID determines its size completely. This is not the case in $\mathcal{S}$, and we can let each job have any size up to that determined by its ID.

The adversary will start with set $\mathcal{S}(0, H)$ – here we added H since the set is constructed from $\mathcal{I}(H)$. The adversary's goal is to have $\mathcal{S}(1/2, H)$ that essentially includes $\mathcal{S}(0, H-1)$ with jobs arrival times changed to 1/2. By recursively applying this strategy, the adversary will be able to force $\mathcal{A}$ to have a makespan of at least $H/2$. As observed in Lemma 5.2, for any instance in $\mathcal{S}(0)$, all jobs in the instance can be completed by time 2 by the optimal solution, and this will complete the proof of Theorem 5.1.

We make use of the two crucial properties we observed above to show that $\mathcal{S}(1/2, H)$ still is rich enough to fool the algorithm. For any non-leaf node v, the adversary removes a subtree rooted at v as soon as the algorithm has assigned the 2-speed router to v for $1/2^{H+1}$ time steps; all jobs in the subtree disappear from each of the remaining possible instances. Since all the possible instances are indistinguishable to the algorithm, the adversary can choose any alive subtree T_{v^*} incident to the root, and delete other sibling subtrees. This will reduce the set of possible instances. From time $1/2$, all instances in $\mathcal{S}(1/2, H)$ must satisfy the restriction that v^* is big.

Also, at time 1/2, the adversary deletes nodes/jobs in T_{v^*} so that each non-leaf node has 4^{H-1} children. All jobs of size 1 are removed, and no big node is removed in this process. The adversary can still choose any mapping (the adversary can pick any node from each set of the alive sibling nodes (not jobs) and set it to be big). Note that and any remaining instance in the pool has the same "hidden Job ID" flexibility as $\mathcal{S}(0, H-1)$. Hence, by decreasing job sizes or removing some jobs, the adversary can make $\mathcal{S}(1/2, H)$ become essentially equivalent to $\mathcal{S}(0, H-1)$. In other words, all instances in $\mathcal{S}(1/2, H)$ have the big subtree T_{v^*} incident to the root (v^* is assigned the 2-speed router), and the subtree encodes all instances in $\mathcal{S}(0, H-1)$ – more precisely, the root of all instances in $\mathcal{S}(0, H-1)$ is replaced with v^* and jobs arrival times with 1/2. While the job sizes are also doubled, the effect is nullified by v^* being assigned the 2-speed router. This allows the adversary to apply his strategy recursively. Hence we derive the following lemma which completes the proof of Theorems 5.1 and 1.4.

LEMMA 5.3. *For any instance in $\mathcal{S}(H)$, there is a way to complete all jobs in the instance by time 2. In contrast, for any deterministic non-clairvoyant algorithm $\mathcal{A}$, there is an instance in $\mathcal{S}(H)$ for which $\mathcal{A}$ has a makespan of at least $H/2$.*

Finally, we discuss the lower bound in terms of the number of constraints that define the polytope over jobs processing rates, $\{y_j\}_j$. We first rewrite the polytope in Eq. (16) only in terms of $\{y_j\}_j$. Let G denote the set of all possible schedules, g. Note that $|G| < ((4^H))^{n/H} = 4^n$ since for each non-leaf node v, one can map exactly one of its 4^H children to the unique 2-speed router in $\mathcal{R}_v$, and there are less than $1 + 4^H + 4^{2H} + ... + 4^{H(H-1)} < n/H$ non-leaf nodes; recall that $n = 4^{H^2}$. Hence, the polytope over $\{g\}$ subject to $\sum_g z_g \leq 1$ consists of at most 4^n vertices, denoted as $v_1, v_2, ..., v_K$. Each job j's processing rate, y_j is expressed as a linear combination of z_g, so we can compactly express it as $\mathbf{y} \leq A\mathbf{z}$ where A is an appropriate matrix consisting of non-negative entries. Then, the polytope over $\{y_j\}_j$ is a downward closure of convex hull of $Av_1, Av_2, ..., Av_K$, so it has at most 2^{4^n} constraints.

We now shift out discussion to how to express the polytope (13)-(15) in terms of $\{y_j\}$. Note that each B_e (omitting t for notational simplicity) has $L = 2^{4^n}$ constraints from the above discussion. To remove s_e variables, we choose one row from each B_e and add up the chosen rows – the right-hand-side is replaced with one. It is easy to see that this expanded expression with $D := L^{N^{1/\epsilon}} = L^{N^2}$ constraints is equivalent to the original polytope; here we set $\epsilon = 1/2$ for simplicity.

Recall that we have shown that no non-clairvoyant algorithm has a competitive ratio $O(N)$ when given speed less than $H/8$ where $N = n$ and $H = \Theta(\sqrt{\log n})$. This translates into a lower bound of $\Omega(\log \log D)$ when given speed less than $O(\sqrt{\log \log \log D})$.

To derive a lower bound when the algorithm is given a constant speed, for some constant H, we use $\mathcal{I}(H)$ as $\mathcal{I}_e$ (with job sizes halved). Note that the number of jobs in $\mathcal{I}(H)$ is $O(1)$, hence $L = O(1)$. If we concatenate E instances, $\mathcal{I}_0, \mathcal{I}_1, ..., \mathcal{I}_{E-1}$, then when the algorithm is given less than $H/8$ speed, the algorithm will have total flow time at least $\Omega(E^2)$ while the optimal scheduler has total flow time at most $O(E)$ resulting in a lower bound of $\Omega(E)$. Since there are at least $D = \Theta(1)^E$ constraints, we derive a lower bound of $\Omega(\log D)$.

COROLLARY 5.4. *Let D denote the number of constraints. For the total flow time objective, any non-clairvoyant deterministic algorithm is $\Omega(\log\log D)$-competitive when given speed less than $o(\sqrt{\log\log\log D})$, and is $\Omega(\log D)$-competitive when given any constant speed.*

6. CONCLUSIONS AND OPEN QUESTIONS

We conclude with some open questions. An immediate open question is to extend Theorem 1.5 so that the speed is $1 + \epsilon$ for any $\epsilon > 0$. Though this can be done in the single-machine setting [Fox et al. 2013], that analysis crucially required a closed form for the allocations and rates. Our analysis is based on optimality conditions of PF, and extending it seems to encounter fundamental roadblocks.

Next, our lower bound for flow time for general PSP requires speed $\omega(1)$ to show any bounded competitive ratio. However, as stated in Corollary 5.4, the lower bound grows very slowly in the number of dimensions/constraints. This brings up the open question of designing a constant-speed algorithm for multidimensional scheduling with Leontief utilities, whose competitive ratio for flow time is poly(D), where D is the number of dimensions (or resources). We believe this will require fundamentally new ideas. A related question is to show lower bounds for flow time of PSP even with clairvoyance – the lower bound we presented holds only for non-clairvoyant algorithms. We note that a clairvoyant $O(1)$-speed $O(\log D)^D$-competitive algorithm is known [Im et al. 2015].

A more open-ended question is to characterize the class MONOTONE PSP and study its precise connection to market clearing for a suitably defined Fisher market, extending the work of [Jain and Vazirani 2010]. As we emphasized, we crucially require the optimality condition of PF. In many cases, even if a market clearing solution exists, the PF algorithm will not find this solution, and therefore we cannot use monotonicity characterizations from market clearing literature. A related question will be to study other algorithms whose optimality conditions have simple characterizations.

ACKNOWLEDGMENTS

The authors would like to thank Nikhil Bansal for the helpful discussion on deriving the lower bound in Corollary 5.4.

REFERENCES

Swarup Acharya, Michael Franklin, and Stanley Zdonik. 1995. Dissemination-based data delivery using broadcast disks. *Personal Communications, IEEE* 2, 6 (1995), 50–60.

Faraz Ahmad, Srimat T Chakradhar, Anand Raghunathan, and T. N. Vijaykumar. 2012. Tarazu: optimizing mapreduce on heterogeneous clusters. In *ACM SIGARCH Computer Architecture News*, Vol. 40. 61–74.

Demet Aksoy and Michael J. Franklin. 1999. "RxW: A scheduling approach for large-scale on-demand data broadcast. *IEEE/ACM Trans. Netw.* 7, 6 (1999), 846–860.

S. Anand, Naveen Garg, and Amit Kumar. 2012. Resource augmentation for weighted flow-time explained by dual fitting. In *Proc. 23rd Annual ACM-SIAM Symp. on Discrete Algorithms*. 1228–1241.

Spyros Angelopoulos, Giorgio Lucarelli, and Nguyen Kim Thang. 2015. Primal-Dual and Dual-Fitting Analysis of Online Scheduling Algorithms for Generalized Flow Time Problems. In *Proc. 23rd European Symposium on Algorithms*. 35–46.

Yossi Azar, Umang Bhaskar, Lisa Fleischer, and Debmalya Panigrahi. 2013. Online Mixed Packing and Covering. In *Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms*. 85–100.

Yossi Azar and Iftah Gamzu. 2011. Ranking with Submodular Valuations. In *Proc. 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*. 1070–1079.

Nikhil Bansal and Ho-Leung Chan. 2009. Weighted Flow Time Does Not Admit O(1)-competitive Algorithms. In *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms*. 1238–1244.

Nikhil Bansal, Moses Charikar, Ravishankar Krishnaswamy, and Shi Li. 2014. Better algorithms and hardness for broadcast scheduling via a discrepancy approach. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms*. 55–71.

Nikhil Bansal, Don Coppersmith, and Maxim Sviridenko. 2008. Improved Approximation Algorithms for Broadcast Scheduling. *SIAM J. Comput.* 38, 3 (2008), 1157–1174.

Nikhil Bansal, Ravishankar Krishnaswamy, and Viswanath Nagarajan. 2010. Better scalable algorithms for broadcast scheduling. In *Proc. 37th Intl. Colloq. on Automata, Languages and Programming (ICALP)*. Springer-Verlag, 324–335.

Ted Bergstrom. Manuscript, 2014. Proving that a Cobb-Douglas function is concave if the sum of exponents is no bigger than 1. (Manuscript, 2014).

Sushil Bikhchandani, Sven de Vries, James Schummer, and Rakesh V Vohra. 2011. An ascending vickrey auction for selling bases of a matroid. *Operations research* 59, 2 (2011), 400–413.

Thomas Bonald, Laurent Massoulié, Alexandre Proutiere, and Jorma Virtamo. 2006. A queueing analysis of max-min fairness, proportional fairness and balanced fairness. *Queueing systems* 53, 1-2 (2006), 65–84.

Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press, New York, NY, USA.

Jivitej S. Chadha, Naveen Garg, Amit Kumar, and V. N. Muralidhara. 2009. A competitive algorithm for minimizing weighted flow time on unrelatedmachines with speed augmentation. In *Proc. ACM Symposium on Theory of Computing (STOC)*. 679–684.

Ho-Leung Chan, Jeff Edmonds, and Kirk Pruhs. 2011. Speed Scaling of Processes with Arbitrary Speedup Curves on a Multiprocessor. *Theory Comput. Syst.* 49, 4 (2011), 817–833.

Chandra Chekuri, Ashish Goel, Sanjeev Khanna, and Amit Kumar. 2004. Multi-processor scheduling to minimize flow time with epsilon resource augmentation. In *Proc. ACM Symposium on Theory of Computing*. 363–372.

Richard Cole, Vasilis Gkatzelis, and Gagan Goel. 2013. Mechanism design for fair division: allocating divisible items without payments. In *Proc. 14th ACM conference on Electronic commerce*. 251–268.

Nikhil R. Devanur and Zhiyi Huang. 2014. Primal Dual Gives Almost Optimal Energy Efficient Online Algorithms. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms*. 1123–1140.

Jeff Edmonds, Donald D. Chinn, Tim Brecht, and Xiaotie Deng. 2003. Non-Clairvoyant Multiprocessor Scheduling of Jobs with Changing Execution Characteristics. *J. Scheduling* 6, 3 (2003), 231–250.

Jeff Edmonds, Sungjin Im, and Benjamin Moseley. 2011. Online Scalable Scheduling for the ℓ_k-norms of Flow Time Without Conservation of Work. In *22nd Annual ACM-SIAM Symposium on Discrete Algorithms*.

Jeff Edmonds and Kirk Pruhs. 2012. Scalably scheduling processes with arbitrary speedup curves. *ACM Transactions on Algorithms* 8, 3 (2012), 28.

Jon Feldman, S. Muthukrishnan, Evdokia Nikolova, and Martin Pál. 2008. A truthful mechanism for offline ad slot scheduling. In *Algorithmic Game Theory*. Springer, 182–193.

Kyle Fox, Sungjin Im, and Benjamin Moseley. 2013. Energy Efficient Scheduling of Parallelizable Jobs. In *Proc. 24th ACM-SIAM Symposium on Discrete Algorithms*. 948–957.

Kyle Fox and Madhukar Korupolu. 2013. Weighted Flowtime on Capacitated Machines. In *Proc. 24th Annual ACM-SIAM Symp. on Discrete Algorithms*. 129–143.

Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. 2006. Dependent rounding and its applications to approximation algorithms. *J. ACM* 53, 3 (2006), 324–360.

Naveen Garg and Amit Kumar. 2007. Minimizing Average Flow-time : Upper and Lower Bounds. In *Proc. 48th IEEE Symposium on Foundations of Computer Science (FOCS)*. 603–613.

Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Proc. 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

Faruk Gul and Ennio Stacchetti. 1999. Walrasian equilibrium with gross substitutes. *Journal of Economic theory* 87, 1 (1999), 95–124.

Anupam Gupta, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, and Kirk Pruhs. 2012a. Scheduling heterogeneous processors isn't as easy as you think. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms*. 1242–1253.

Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. 2012b. Online Primal-Dual for Non-linear Optimization with Applications to Speed Scaling. In *Proc. 10th International Workshop on Approximation and Online Algorithms (WAOA)*. 173–186.

Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. 1997. Scheduling to Minimize Average Completion Time: Off-Line and On-Line Approximation Algorithms. *Math. of Oper. Res.* 22, 3 (1997), 513–544.

Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. 2014. Competitive Algorithms from Competitive Equilibria: Non-Clairvoyant Scheduling under Polyhedral Constraints. In *Proc. 46th ACM Symposium. on Theory of Computing (STOC)*. 313–322.

Sungjin Im, Janardhan Kulkarni, and Kamesh Munagala. 2015. Competitive Flow Time Algorithms for Polyhedral Scheduling. In *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*. 506–524.

Sungjin Im, Janardhan Kulkarni, Kamesh Munagala, and Kirk Pruhs. 2014. SelfishMigrate: A Scalable Algorithm for Non-clairvoyantly Scheduling Heterogeneous Processors. In *IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS)*. 531–540.

Sungjin Im and Benjamin Moseley. 2012. An online scalable algorithm for average flow time in broadcast scheduling. *ACM Transactions on Algorithms* 8, 4 (2012), 39.

Sungjin Im, Benjamin Moseley, and Kirk Pruhs. 2011. A tutorial on amortized local competitiveness in online scheduling. *SIGACT News* 42, 2 (2011), 83–97.

Sungjin Im, Viswanath Nagarajan, and Ruben van der Zwaan. 2012. Minimum Latency Submodular Cover. In *Proc. 39th Intl. Colloq. on Automata, Languages and Programming (ICALP)*. 485–497.

K. Jain and V. V. Vazirani. 2010. Eisenberg-Gale markets: Algorithms and game-theoretic properties. *Games and Economic Behavior* 70, 1 (2010), 84–106.

Bala Kalyanasundaram and Kirk Pruhs. 2000. Speed is as powerful as clairvoyance. *JACM* 47, 4 (2000), 617–643.

Frank P. Kelly, Laurent Massoulié, and Neil S. Walton. 2009. Resource pooling in congested networks: proportional fairness and product form. *Queueing Systems* 63, 1-4 (2009), 165–194.

Frank P. Kelly, Aman K. Maulloo, and David K. H. Tan. 1998. Rate control for communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research society* (1998), 237–252.

Gunho Lee, Byung-Gon Chun, and Randy H. Katz. 2011. Heterogeneity-aware resource allocation and scheduling in the cloud. In *Proc. 3rd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud*, Vol. 11.

Nimrod Megiddo. 1974. Optimal flows in networks with multiple sources and sinks. *Mathematical Programming* 7, 1 (1974), 97–107.

John Nash. 1950. The Bargaining Problem. *Econometrica* 18, 2 (1950), 155–162.

Lucian Popa, Gautam Kumar, Mosharaf Chowdhury, Arvind Krishnamurthy, Sylvia Ratnasamy, and Ion Stoica. 2012. FairCloud: sharing the network in cloud computing. In *ACM SIGCOMM*. 187–198.

Kirk Pruhs, Jiri Sgall, and Eric Torng. 2004. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapter Online Scheduling.

Maurice Queyranne and Maxim Sviridenko. 2002. A (2+epsilon)-approximation algorithm for the generalized preemptive open shop problem with minsum objective. *J. Algorithms* 45, 2 (2002), 202–212.

Julien Robert and Nicolas Schabanel. 2008. Non-clairvoyant scheduling with precedence constraints. In *Proc. 19th Annual ACM-SIAM symposium on Discrete algorithms*. 491–500.

Alexander Schrijver. 2003. *Combinatorial optimization: polyhedra and efficiency*. Vol. 24. Springer Science & Business Media.

Andreas S. Schulz and Martin Skutella. 1997. Random-based scheduling new approximations and LP lower bounds. In *Proc. International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*. Springer, 119–133.

Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. In *Proc. IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. 1–10.

Hal R. Varian. 1976. Two problems in the theory of fairness. *Journal of Public Economics* 5, 3-4 (1976), 249–260.

David P. Williamson and David B. Shmoys. 2011. *The Design of Approximation Algorithms*. Cambridge University Press. I–XI, 1–504 pages.

Joel Wolf, Deepak Rajan, Kirsten Hildrum, Rohit Khandekar, Vibhore Kumar, Sujay Parekh, Kun-Lung Wu, and Andrey Balmin. 2010. Flex: A slot allocation scheduling optimizer for mapreduce workloads. In *Middleware*. Springer, 1–20.

John W. Wong. 1988. Ilivery. *Proc. IEEE* 76, 12 (1988), 1566–1577.

Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. 2008. Improving MapReduce performance in heterogeneous environments. In *Proc. 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 29–42.

Seyed Majid Zahedi and Benjamin C. Lee. 2014. REF: resource elasticity fairness with sharing incentives for multiprocessors. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 145–160.

A. MONOTONICITY OF RA-S: PROOF OF THEOREM 1.7

We will show the following theorem, which will immediately imply Theorem 1.7.

THEOREM A.1. *The RA-S utility functions defined in Eq (5) are concave (which implies the space $\mathcal{P}$ 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 (3), 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 $\mathcal{P}$ 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 $\mathcal{P}$ is given by the following (where $\rho \in [0,1]$ and $\rho' \geq \rho$):

$$\mathcal{P} = \left\{ y_j = u_j(\mathbf{y}_j) = \left(\sum_{d=1}^{D} (f_{jd}(x_{jd}))^{\rho_j} \right)^{1/\rho'_j}, \qquad \sum_j x_{jd} \leq 1 \ \forall d \right\}$$

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:

$$v_j(\mathbf{y}_j) = w_j \log u_j(\mathbf{y}_j) = \frac{w_j}{\rho'_j} \log \left(\sum_{d=1}^{D} h_{jd}(x_{jd}) \right)$$

For price vector $\mathbf{p} = \{p_1, p_2, \ldots, p_D\} \geq \mathbf{0}$, define the demand function $X_j(\mathbf{p})$ as follows:

$$X_j(\mathbf{p}) = \text{argmax}_{\mathbf{y}_j \geq 0} \left(v_j(\mathbf{y}) - \mathbf{p} \cdot \mathbf{y}_j \right) \qquad \text{and} \qquad U_j(\mathbf{p}) = u_j(X_j(\mathbf{p}))$$

Note that $X_j(\mathbf{p})$ is uniquely defined for given $\mathbf{p}$ due to the strict concavity of $v_j(\mathbf{y})$ (see the first part of Theorem A.1), so $U_j(\mathbf{p})$ is well-defined.

LEMMA A.2. *Consider an arbitrary price vector $\mathbf{p}$, and a different price vector $\mathbf{p}'$ that only differs from $\mathbf{p}$ in the r^{th} dimension. Assume $p'_r > p_r$. Let $\mathbf{y}_j = X_j(\mathbf{p})$ and $\mathbf{y}'_j = X_j(\mathbf{p}')$. Then:*

(*1*) *If $x_{jr} > 0$, then $U_j(\mathbf{p}') < U_j(\mathbf{p})$. Furthermore, $x'_{jr} < x_{jr}$, and for all $d \neq r$, $x'_{jd} \geq x_{jd}$.*
(*2*) *If $x_{jr} = 0$, then $U_j(\mathbf{p}') = U_j(\mathbf{p})$. Furthermore, for all d, $x'_{jd} = x_{jd}$.*

Further, we have a stronger property that if $x_{jr} > c$, then $U_j(\mathbf{p}) - U_j(\mathbf{p}') \geq c'(p_r - p'_r)$ 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 $\mathbf{p}$ and $\mathbf{p}'$ 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 $W(\mathbf{y}) = (U_j(\mathbf{y}))^{\rho'_j}$; since $U_j(\mathbf{y})$ is strictly monotone in $\mathbf{y}$, the same holds for $W(\mathbf{y})$. Note that $W(\cdot)$ also can be viewed as a function of $\mathbf{p}$ since $\mathbf{p}$ uniquely determines $\mathbf{x}$. Partially differentiating $v_j(\mathbf{x}) - \mathbf{p} \cdot \mathbf{x}_j$ w.r.t. x_d, we have the following (sufficient and necessary) optimality condition:

$$x_d > 0 \;\Rightarrow\; \frac{q}{W(\mathbf{y})}h'_d(x_d) = p_d \qquad \text{and} \qquad x_d = 0 \;\Rightarrow\; \frac{q}{W(\mathbf{y})}h'_d(x_d) \leq p_d$$

Consider price vector $\mathbf{p}'$ 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 $W(\mathbf{p}') \geq W(\mathbf{p})$. 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 h_r. The same argument shows that for all dimensions $d \neq r$, $x'_d \leq x_d$. But this implies $W(\mathbf{p}') < W(\mathbf{p})$, which is a contradiction. Therefore, we must have $W(\mathbf{p}') < W(\mathbf{p})$.

Now consider any dimension $d \neq r$. Since $p'_d = p_d$ and $W(\mathbf{p}') < W(\mathbf{p})$, 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 $x'_r < x_r$.

It now remains to show the stronger property under the extra conditions. Imagine that we increase $\mathbf{p}$ to $\mathbf{p}'$ 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 $W(\mathbf{x})$, let's focus on d with non-zero p_d.

Observe that boundedness of $\mathbf{p}$ implies boundedness of $\mathbf{x}$ since $\mathbf{x}$ minimizes $v_j(\mathbf{x}) - \mathbf{p} \cdot \mathbf{x}_j$ and $v_j(\mathbf{x})$ 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 $\mathbf{p}, \mathbf{p}', \mathbf{x}, \mathbf{x}'$. For the sake of contradiction, suppose $W_j(\mathbf{p})$ and $W_j(\mathbf{p}')$ 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 $W_j(\mathbf{p}')$ and $W_j(\mathbf{p})$ are significantly different, which is a contradiction. An easy algebra gives the desired claim. $\square$

Proof of Theorem A.1.. We now use Lemma A.2 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 $\mathbf{p}$ such that $\{X_j(\mathbf{p})\}$ define the optimal solution to PF.
(2) For this price vector $\mathbf{p}$, if $p_d > 0$, then $\sum_j x_{jd} = 1$.

Start with this optimal solution. Suppose a new job arrives. At the price vector $\mathbf{p}$, compute the quantities $X_j(\mathbf{p})$. If some resource is over-demanded, we continuously increase its price. We perform this tatonnement process until no resource is over-demanded. By Lemma A.2, 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 A.2, 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 increae the price of over-demanded resources, the boundness of $\mathbf{p}$ follows. Hence by the stronger property of Lemma A.2, 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 $\mathbf{p}'$, and let $\mathbf{y}'_j = X_j(\mathbf{p}')$. 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 A.1.

A similar proof to the above shows the following; we omit the details.

COROLLARY A.3. *The PF algorithm is monotone for utility functions of the form* $u_j(\mathbf{y}_j) = g_j\left(\sum_{d=1}^{D} f_{jd}(x_{jd})\right)$, *where* g_j, f_{jd} *are increasing, smooth, and strictly concave functions.*