Minimizing Maximum Flow Time on Related Machines via Dynamic Posted Pricing[*]

Sungjin Im[†1], Benjamin Moseley[‡2], Kirk Pruhs[§3], and Clifford Stein[¶4]

1 Electrical Engineering and Computer Science, University of California at Merced, CA, USA
sim3@ucmerced.edu
2 Washington University in St. Louis, MO, USA
bmoseley@wustl.edu
3 Dept. of Computer Science, University of Pittsburgh, PA, USA
kirk@cs.pitt.edu
4 Department of Industrial Engineering and Operations Research, Columbia University, New York, NY, USA
cliff@ieor.columbia.edu

Abstract

We consider a setting where selfish agents want to schedule jobs on related machines. The agent submitting a job picks a server that minimizes a linear combination of the server price and the resulting response time for that job on the selected server. The manager's task is to maintain server prices to (approximately) optimize the maximum response time, which is a measure of social good. We show that the existence of a pricing scheme with certain competitiveness is equivalent to the existence of a monotone immediate-dispatch algorithm. Our main result is a monotone immediate-dispatch algorithm that is $O(1)$-competitive with respect to the maximum response time.

1998 ACM Subject Classification F.2.2 [Nonnumerical Algorithms and Problem]: Sequencing and scheduling

Keywords and phrases Posted pricing scheme, online scheduling, related machines, maximum flow time, competitiveness analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2017.51

1 Introduction

1.1 Motivation and Background

Many large companies foster a competitive internal environment to create flexibility, challenge the status quo, and motivate employees. However, it is recognized that internal competition has to be managed so that the costs do not outweigh the benefits [9, 6]. In this paper, we consider one such management task. Namely, we consider managing compute servers, used

[*] This work was partially done while the authors were participating in the Algorithms and Uncertainty program at the Simons Institute for the Theory of Computing at the University of California at Berkeley.
[†] Supported in part by NSF grants CCF-1409130 and CCF-1617653.
[‡] Supported in part by a Google Research Award, a Yahoo Research Award and NSF Grant CCF-1617724.
[§] Supported in part by NSF grants CCF-1421508 and CCF-1535755, and an IBM Faculty Award.
[¶] Supported in part by NSF grant CCF-1421161.

by competing self-interested agents, to optimize social good. As agents are self-interested, you would expect them to greedily choose the server that will finish their task first. In the setting that we consider, this can lead to schedules with highly suboptimal social good. Thus the manager might reasonably want to implement some mechanism that will incentivize the agents to produce a schedule with high social good. Following the lead of [10] we consider a dynamic posted price mechanism; that is, the manager maintains a dynamically changing price for each server (like Amazon's EC2). Thus a self-interested agent would take into account both response time and price when selecting a server.

In [10] various common models of compute servers are considered, and it is assumed that:

- all agents sequentially select servers at the same moment of time,
- jobs on one machine are scheduled in a First-Come-First-Served manner,
- agents greedily pick the server that minimizes a linear combination of the resulting response time of the agent's job and the current price for that server, and
- the social good is measured by (technically the inverse of) the makespan of the schedule.

The main result in [10] is a pricing scheme that guarantees that selfish agents construct a schedule that is $O(1)$-competitive with respect to makespan on related machines. In this scheme the prices are essentially set so that the resulting schedule is identical to the schedule produced by the (non-pricing based) online algorithm, called Slow-Fit in [4], that was shown to be $O(1)$-competitive in [3]. Slow-Fit assigns each job to the slowest machine that would not result in a response time greater than some constant times the current estimate of the optimal makespan (and doubles the estimate if this isn't possible).

[11] posed the question of whether this $O(1)$-approximation result could be extended to the (arguably more natural) setting where agents may submit jobs over time, and the social good is the natural generalization of makespan, namely the maximum response time of any job. The maximum response time is the maximum time a job waits in the system after its arrival to be completed.

The combinatorics of the scheduling of related machines with the objective of maximum flow are a bit tricky, and it wasn't until recently that any $O(1)$-competitive online algorithm (or even any polynomial-time $O(1)$-approximation offline algorithm) was known [5]. [5] called this online algorithm Double-Fit. Double-Fit delays assigning jobs, and collects them into batches. Periodically the jobs in a batch are assigned to machines. In assigning a batch of jobs, the jobs are considered in decreasing order of size. Each job is then assigned to a server using essentially a two level generalization of Slow-Fit. The conclusion of [5] states that:

> Note that our algorithm Double-Fit is not immediate dispatch, i.e., it does not dispatch a job to a machine immediately upon arrival. We are unable to extend the ideas here to obtain an $O(1)$-competitive immediate dispatch algorithm, and it is not clear to us whether such an algorithm exists.

Immediate dispatch algorithms have some practical advantages, most notably, they do not require the maintenance of a global queue of unprocessed work, which could be a potential bottleneck to scalability.

1.2 Our Results

We start by observing that the open questions from [11] and [5] are related. More precisely, we observe that a monotone immediate-dispatch algorithm can be converted into a dynamic posted price algorithm, preserving the competitive ratio. Similarly, we observe that a posted price algorithm can be converted into a monotone immediate-dispatch algorithm, preserving the competitive ratio. An algorithm is monotone if the speed of the server that an agent would pick is monotonically increasing in the size of a job. Monotonicity is a natural property in that the bigger a job is, the more critical it is that it be assigned to a fast server (in the extreme, an infinite sized job must be assigned to the fastest server if one is to achieve a competitive ratio independent of server speeds).

Using this equivalence, we establish the existence of an $O(1)$-competitive posted price scheme by giving a monotone immediate-dispatch $O(1)$-competitive algorithm, which we call Immediate-Double-Fit (IDF). *Thus we affirmatively answer both the open question from [11] and the open question from [5].*

The algorithm IDF immediately assigns jobs using the same strategy as Double-Fit does. After observing that IDF is monotone, we turn to analyzing IDF's competitiveness. The fact that Double-Fit assigns jobs within a batch in size order was critical to the analysis of Double-Fit in [5]. Intuitively the main difficulty of analyzing IDF is that there may be no relationship between the size of a job and when it is assigned a server, thus making the analysis of Double-Fit in [5] inapplicable. Not surprisingly, the key to our being able to analyze IDF was finding the "right" inductive hypothesis, which is substantially different than the inductive hypothesis used in [5]. Perhaps somewhat surprisingly, our inductive hypothesis is actually simpler than the one used in [5], and as a consequence, we also get a slightly better bound on the competitive ratio of IDF, namely 25/2, than the bound of 27/2 on the competitive ratio of Double-Fit obtained in [5].

One intuitive take away point from these results is that dynamic posted prices gives management essentially the same power as being able to impose arbitrary job to server assignments, when the setting is related machines and the objective is maximum flow time.

1.3 Other Related Work

[10] showed that for unrelated machines, every pricing scheme can lead to schedules that are $\Omega(m)$-competitive with respect to makespan. In the unrelated machine setting the processing time of a job is machine dependent. They also showed that static pricing schemes (where server prices do not change over time) are in some sense equivalent to the natural greedy algorithm.

Intuitively prices are necessary to achieve $O(1)$-competitiveness for maximum response time on related machine. To understand why, note that it is well-known that FIFO is optimal on a single machine for the objective of maximum response time. In addition to being optimal, FIFO is the unique scheduling policy that allows each agent to know with certainty the response time of its job on each server. However, [5] showed that the natural greedy algorithm is $\Omega(m)$-competitive for maximum response time on related machines.

There is a significant literature on mechanism design for scheduling, starting with the paper [14] that instigated the study of mechanism design within the algorithmic community. Much of this work focuses on finding and/or analyzing coordination mechanisms with respect to the price of anarchy, which compares the social good of some equilibrium to the optimal social good. We mention a few such results that seem most closely related to the results in this paper. A coordination mechanism for identical machines with constant price of anarchy with respect to makespan can be found in [7]. [13] studies coordination mechanisms for four classes of multiprocessor machine scheduling problems and derive upper and lower bounds for the price of anarchy with respect to makespan of these mechanisms. [1] considers coordination mechanisms for unrelated machines in which agents control subsets of jobs, and each player's objective is to minimize the weighted sum of completion time of her jobs.

There is a significant literature on mechanism design using posted prices, most of it focused on auctions and markets (see [15] for an overview). [8] focuses on server problems, motivated in part by the SFPark system (SF-park.org), which sets parking prices in San

Francisco based on parking congestion. [8] gives pricing schemes for the classical problems of k-server, metrical task systems, and metrical matching that in some cases achieve competitive ratios that are close to the optimal competitive ratios for general online algorithms.

There is significant literature on online scheduling. Good starting points into this literature are [16] and [12]. Probably the most relevant results come from [2], which gave scalable algorithms for minimizing maximum flow time on unrelated machines, and for minimizing weighted maximum flow time on related machines. Resource augmentation is required for both problems in order to achieve constant approximation.

2 Notation and Definitions

In the standard related machines environment the m servers/machines have associated speeds, $s_1, \ldots, s_m$. We assume without loss of generality that $s_1 \leq s_2 \leq \ldots \leq s_m$. A collection of n jobs arrive over time. The release time r_j of job j is when it is submitted to be scheduled. Further, each job j has a size p_j. A (nonpreemptive) schedule specifies for each job j, a starting time λ_j and an assigned server i_j, with the restriction the time intervals $[\lambda_j, C_j]$ should be disjoint for all jobs assigned to the same machine. Here $C_j = \lambda_j + p_j/s_{i_j}$ is the time that job j completes. If j has been run on server i for $\tau \leq p_j/s_i$ units of time, then its unprocessed volume is $p_j - s_i\tau$. The *flow/response time* of the job is defined as $F_j := C_j - r_j$, and the objective we consider is to minimize the maximum flow time $F_{\max} = \max_j F_j$. The makespan of a schedule is the maximum completion time.

In this paper, we assume that an online scheduler learns job j's size p_j at time r_j when it is released. An online scheduler is called *immediate dispatch* if it always assigns a job to a machine at the job's release time. The scheduler need not start job j at time r_j, but the scheduler must make an irrevocable decision about which machine the job will eventually run on. Let $A_t(p)$ be the speed of the machine that an algorithm A would assign a job of size p to if it was released at the current time t; here it is assumed that job identity plays no role in A's assignment decision. Then algorithm A is monotone if for all possible instances, and for all possible t, $A_t(p)$ is non-decreasing in p.

A dynamic posted pricing scheme is a special type of online scheduler. Jobs assigned to a server are processed in First-Come-First-Served order. So every processor is always processing the earliest released, uncompleted job assigned to it. The online schedule maintains a dynamically changing price for each server. Let $c_i(t)$ denote the price/cost for server i at time t. Let $L_i(t)$ denote the unprocessed volume of jobs previously assigned to server i at time t, divided by machine i's speed. In other words, it takes $L_i(t)$ units of time for machine i to complete its unprocessed workload assuming that no more jobs arrive. Then job j is assigned to the server i that minimizes $L_i(r_j) + p_j/s_i + c_i(t)$. Intuitively the job selfishly assigns itself to the machine that minimizes its flow time plus the machine cost. It is important that the prices $c_i(r_j)$ are posted prior to job j'arrival; that is, they cannot depend on the value of p_j, and may only depend on past events. For notational convenience we may drop the current time from the notation if it is clear from the context. For example, we may simply use c_i in place of $c_i(t)$.

One take away point from [10], as well as earlier work on posted price mechanisms, is that dealing with ties can be annoying. The issue of ties manifests itself in two ways in our setting. Firstly, in order to show that our pricing scheme is monotone, we need that the machine speeds are distinct, that is that $s_1 < s_2 < \ldots < s_m$. This can be achieved with probability one by decreasing each machine speed by some random infinitesimal amount, at the cost of raising the competitive ratio by an infinitesimal amount. The mechanism can simulate a slower machine by delaying the start date of each job appropriately. Thus technically our mechanism involves both pricing and slightly delaying some jobs.

The second way in which the issue of ties manifests itself is when there are two different servers that simultaneously minimize $L_i(r_j) + p_j/s_i + c_i(t)$. But this is also handled by the random decrement of the processor speeds, as this sort of tie will then arise with probability zero. Thus we assume in our analysis that there is a unique server i that minimizes $L_i(r_j) + p_j/s_i + c_i(t)$.

[10] takes a different approach to this second issue of ties. They assume that in the case of ties, the job may be adversarially assigned to any minimizing server. This has the advantage of imposing minimal assumptions on the actions of the agent, but it has the disadvantage of cluttering/complicating the algorithmic design/analysis process. The majority of the effort in [10] is related to handling ties.

3 Algorithm and Analysis

In Subsection 3.1 we establish the equivalence of posted price algorithms and monotone immediate-dispatch algorithms. In Subsection 3.2 we describe the Immediate-Double-Fit algorithm. In Subsection 3.3 we first note that the Immediate-Double-Fit algorithm is monotone, and then give an inductive argument bounding its competitiveness.

3.1 Equivalence between Monotonicity and Post-pricing Scheme

▶ **Lemma 1.** *An immediate-dispatch, monotone algorithm A can be converted into a posted pricing algorithm/scheme B. In particular, there is a pricing algorithm B where each job is assigned to the same machines in both A and B. Thus, both algorithms produce exactly the same schedule.*

Proof. We explain how to convert A into B. Assume that a job j is released at time t. Price any machine on which A would never run j no matter what its processing time is at infinite. For notational convenience, drop them from our ordering and assume that m machines remain. Assume according to algorithm A that at size p_i the speed of the selected processor changes from s_i to s_{i+1} for $i \in [1, m-1]$; more precisely, $A(p) \leq s_i$ for all $p < p_i$ and $A(p) \geq s_{i+1}$ for all $p > p_i$. Define $g_c(p)$ as the cost function that takes a job size p and returns the minimum cost the job has to pay under the pricing c. Let L_i denote the load on machine i just before j is assigned. By setting the price vector c so that the following is satisfied for all $1 \leq i \leq m-1$:

$$L_i + c_i + p_i/s_i = L_{i+1} + c_{i+1} + p_i/s_{i+1},$$

we get a cost function $g_c(p)$ where the cost for a job of size $p \in (p_{i-1}, p_i)$ is minimized on machine i. Hence under this post-pricing scheme, each job is assigned exactly to the same machine as it were by the given algorithm A. Also by setting c_1 to be sufficiently large and using the fact that $s_1 < s_2 < ... < s_m$, we can ensure that all prices are positive. ◀

Although it is not needed to establish our main results, we now prove the converse of Lemma 1.

▶ **Lemma 2.** *A pricing algorithm A is an immediate dispatch, monotone algorithm.*

Proof. It it obvious that it is an immediate dispatch algorithm. To establish monotonicity, consider the arrival of a job. Let c_i be the price for machine i. Note that job of size p pays $L_i + c_i + p/s_i$ if it chooses machine i. Let $g(p)$ denote the minimum cost a job of size p has to pay on any machine. Due to the greedy nature of clients, we have $g(p) := \min_i(L_i + c_i + p/s_i)$, which is a piece-wise linear function. This implies that if there is a value of p for which machine i minimizes the cost, the set of such values must form an interval. Let p_i be the size of a job where a greedy client would change from a machine with speed s_i to a machine with speed s_k when we increase p. Note that the uniqueness of p_i follows from the above observation. Knowing that $L_i + c_i + p_i/s_i = L_k + c_k + p_i/s_k$ and $L_i + c_i + p/s_i > L_k + c_k + p/s_k$ for $p > p_i$, we conclude that $s_i < s_k$, thus proving monotonicity of the algorithm. ◀

3.2 Description of the Immediate-Double-Fit Algorithm

We start by making some simplifying assumptions, and defining some concepts and notation. Assume for the moment that the algorithm knows Opt, the objective value of the optimal solution. If not, we show at the end of the analysis how to remove this assumption using the standard doubling trick. For simplicity, we assume that jobs arrive at distinct times. We can easily extend our analysis to remove this assumption by considering jobs released at the same time from the largest to smallest, but this would complicate the analysis.

Time is broken into epochs. The length of a epoch depends on when jobs are released. Define epochs to be of length $\epsilon\mathsf{Opt}$, where ϵ is an arbitrarily small parameter such that at most one job arrives in each epoch. The first epoch begins at time 0 before any job arrives. At the start of an epoch we assign the job that arrived in the last epoch. We now describe how to assign an arriving job j. Let $[i_j, m]$ be the machines i on which p_j/s_i is at most Opt. The algorithm is parameterized by constants $\alpha, \beta \geq 1$ which will be fixed later.

We are now ready to describe the Immediate-Double-Fit (IDF) Algorithm. When a new job j arrives, IDF does the following:

1. If there is a machine in $[i_j, m]$ with load less than $\alpha\mathsf{Opt}$, then schedule j on the slowest such machine. We say in this case that j was placed in the *saturation* phase.
2. Else if there is a machine in $[i_j, m]$ with load less than $\beta\mathsf{Opt}$ then schedule j on the slowest such machine. We say in this case that j was placed in the *slow fit* phase.
3. Else the algorithm admits failure.

3.3 Analysis

We begin by establishing in Lemma 3 that the IDF algorithm is monotone. We then turn to analyzing IDF's competitiveness. We show that IDF never admits failure for proper choice of α and β, under the assumption that its estimation of Opt is correct. Noting that the algorithm is immediate dispatch for sufficiently small ϵ the algorithm can be converted to a pricing algorithm as shown in Lemma 3. We then finish by showing how to apply the standard doubling trick to remove the assumption that the algorithm knows Opt.

▶ **Lemma 3.** *Algorithm A is a monotone algorithm.*

Proof. Let $p < q$ be two possible job sizes. Let $[i_p, m]$ be the machines on which a job of size p would run less than Opt time units. That is, $p/s_k \leq \mathsf{Opt}$ for $k \in [i_p, m]$. Similarly define $[i_q, m]$. Note that $i_p \leq i_q$. If a job of size q was placed on a machine i during the saturation phase, then this machine has load less than $\alpha\mathsf{Opt}s_i$. By definition of the algorithm, a job of size p would also be assigned during the saturation phase to a machine no faster than machine i, since machine i's load is less than $\alpha\mathsf{Opt}s_i$, and p can run on machine i. If instead a job of size q was placed on machine i during the slow fit phase, then all machines in the range $[i_q, m]$ have load at least $\alpha\mathsf{Opt}$. Thus a job of size p could either be placed on a machine slower than i_q during the saturation phase, or on a machine no faster than i in the slow fit phase by definition of the algorithm. ◀

We now turn to analyzing IDF's competitiveness. For notational compactness, we will now starting using A to denote the algorithm IDF. We will show that the following statements hold by induction on epochs:

$\mathcal{A}(i,k)$ is the statement $A_i(k) \leq A_i^*(k) + c\mathsf{Opt}S_i$

and

$\mathcal{B}(i,k)$ is the statement $B_i(k) \leq B_i^*(k) + c\mathsf{Opt}S_i$

where

- $S_i = \sum_{k=i}^m s_k$ is the total speed of machines $[i,m]$.
- $A_i(k)$ is the total load on machines $[i,m]$ under the algorithm A just before jobs are assigned at the start of epoch k.
- $B_i(k)$ is the total load on machines $[i,m]$ under the algorithm A just after all jobs are assigned by the algorithm at the start of epoch k.
- Define *restricted opt* to be the optimum under the restriction that jobs can only be assigned to machines at the start of the epoch. Note that by making ϵ sufficiently small, this is does not change the optimal solution.
- $A_i^*(k)$ is the total load on machines $[i,m]$ for the restricted opt just before jobs are assigned by restricted opt at the start of epoch k.
- $B_i^*(k)$ is the total load on machines $[i,m]$ for the restricted opt just after all jobs are assigned by restricted opt at the start of epoch k.

In order for our induction to go through, we will need the various parameters to satisfy the following inequalities:

- $\alpha \geq \beta - c + 1$
- $c \geq \alpha + 1$
- $1 + c + \epsilon \leq \beta$

We observe in Lemma 4 that these inductive statements imply that IDF/A is $\beta+1$ competitive. We then show in Lemma 5 that $\mathcal{B}(i,k)$ implies $\mathcal{A}(i,k+\epsilon)$. We then complete the inductive proof by showing in Lemma 6 that $\mathcal{A}(i,k)$ implies $\mathcal{B}(i,k)$.

▶ **Lemma 4.** *If* $\forall i \forall k\, [\mathcal{A}(i,k)$ *and* $\mathcal{B}(i,k)]$ *then* A *is* $(\beta+1)$*-competitive.*

▶ **Lemma 5.** $\forall i\ \ \mathcal{B}(i,k)$ *implies* $\forall i\ \ \mathcal{A}(i,k+\epsilon)$.

Proof. The proof is by reverse induction on i. For a base case, $i = m+1$, the claim is vacuously true. For the inductive case, assume that $\mathcal{A}(i+1,k+\epsilon)$ holds and our goal is to prove $\mathcal{A}(i,k+\epsilon)$.

For the first case, say that machine i at epoch $k+\epsilon$ has at most $c\mathsf{Opt}s_i$ work assigned to it. This implies that $A_i(k+\epsilon) \leq A_{i+1}(k+\epsilon) + c\mathsf{Opt}s_i$. Knowing that $A_{i+1}(k+\epsilon) \leq A_{i+1}^*(k+\epsilon) + c\mathsf{Opt}S_{i+1}$ is true (that is, $\mathcal{A}(i+1,k+\epsilon)$ holds), we have the following.

$$\begin{aligned} A_i(k+\epsilon) &\leq A_{i+1}(k+\epsilon) + c\mathsf{Opt}s_i \\ &\leq A_{i+1}^*(k+\epsilon) + c\mathsf{Opt}S_{i+1} + c\mathsf{Opt}s_i \\ &= A_{i+1}^*(k+\epsilon) + c\mathsf{Opt}S_i \end{aligned}$$

For the second case, machine i at epoch $k+\epsilon$ has strictly more than $c\mathsf{Opt}s_i$ assigned to it. Let a denote the last job assigned to machine i. We know that $p_a/s_i \leq \mathsf{Opt}$ by definition of the algorithm. Knowing this, it must be the case that machine i was loaded to more than

$(c-1)s_i\mathsf{Opt}$ when job a was assigned. Knowing that $c-1 \geq \alpha$ it is the case that job a was assigned by the slow-fit phase of the algorithm. Also we know that machine i is not ready to process job a at epoch $k+\epsilon$ since it hasn't completed all jobs assigned to it that have arrived before job a since at the epoch machine i has a strictly positive load excluding the last job a assigned to it. Since a was assigned by the slow-fit phase, when job a arrived, it must be the case that all machines i, $i+1$, ... ,m have load at least $\alpha\mathsf{Opt}$. This implies that at epoch $k+\epsilon$, all machines i, $i+1$, ... ,m have strictly positive loads.

Thus, we now know that $m, m-1, \ldots, i$ are busy processing some job between epoch k and $k+\epsilon$. We know that $B_i(k) \leq B_i^*(k) + c\mathsf{Opt}S_i$ since $\mathcal{B}(i,k)$ holds. We further know that $B_i^*(k)$ can decrease by at most ϵS_i to get $A_i^*(k+\epsilon)$ as this is the most work Opt can process on machines $m, m-1, \ldots, i$ between epoch k and $k+\epsilon$. The above argument implies that $B_i(k)$ decreases by ϵS_i since all machines i or greater are processing jobs during $[k, k+\epsilon]$. Thus, in the inequality $B_i(k) \leq B_i^*(k) + c\mathsf{Opt}S_i$ the left hand side decreases by at least as much as the right, giving the lemma. ◀

▶ **Lemma 6.** *$\forall i\mathcal{A}(i,k)$ implies $\forall i\mathcal{B}(i,k)$.*

Proof. Assume that a job j arrives in epoch $k-1$. By assumption, only one job arrives in epoch $k-1$. The proof is first by induction on k, and then by reverse induction on i, where i is the machine to which job j is assigned. We handle at the end the case where job j cannot be assigned and the algorithm declares failure.

We consider two cases. In the first case, assume that the load on machine i for the algorithm after jobs have been assigned at the start of epoch k is at most $c\mathsf{Opt}s_i$. Then in this case we know by induction that $B_{i+1}(k) \leq B_{i+1}^*(k) + c\mathsf{Opt}S_{i+1}$. Thus using the assumption that the load on machine i is at most $c\mathsf{Opt}s_i$, we know that

$$B_i(k) \leq B_{i+1}^*(k) + c\mathsf{Opt}S_{i+1} + c\mathsf{Opt}s_i \leq B_i^*(k) + c\mathsf{Opt}S_i$$

Now consider the case that the load on machine i is strictly more than $c\mathsf{Opt}s_i$. Thus we know that the last job put on machine i by the algorithm at the start of epoch k was assigned in the slow fit phase since $c \geq \alpha + 1$. If $p_j/s_{i-1} > \mathsf{Opt}$ or $i = 1$, then optimal cannot run j on a slower machine than i, and thus $\mathcal{A}(i,k)$ implies $\mathcal{B}(i,k)$ as $B_i(k) - A_i(k)$ and $B_i^*(k) - A_i^*(k)$ both increase by p_j. Otherwise let h be minimal such that all machines in the range $[h, i-1]$ have load at least $\beta\mathsf{Opt}$. Then we know that either $h = 1$ or $p_j/s_{h-1} > \mathsf{Opt}$, otherwise the algorithm would have put job j on machine $h-1$. In either case, optimal cannot put job j on a machine with index $\leq h-1$. Thus $\mathcal{A}(h,k)$ implies $\mathcal{B}(h,k)$ as $B_h(k) - A_h(k)$ and $B_h^*(k) - A_h^*(k)$ both increase by p_j.

Now consider what happens to $\mathcal{B}(g,k)$ as g increases from h to i. Assume $g \in (h, i]$. Then $B_{g-1}(k) - B_g(k) \geq \beta\mathsf{Opt}s_{g-1}$. So intuitively $B_g(k)$ decreases at a rate of at least β. Also $B_{g-1}^*(k) - B_g^*(k) \leq (1+\epsilon)\mathsf{Opt}s_{g-1}$, otherwise the load on machine $g-1$ for optimal would be greater than $(1+\epsilon)\mathsf{Opt}s_{g-1}$, contradicting the definition of Opt. Thus intuitively, $B_g^*(k)$ decreases at a rate of at most $\epsilon + 1$. Also $c\mathsf{Opt}S_{g-1} - c\mathsf{Opt}S_g = c\mathsf{Opt}s_{g-1}$. So intuitively this term decreases at a rate of exactly c. Thus using the fact that $1 + c + \epsilon \leq \beta$, $\mathcal{B}(h,k)$ implies $\mathcal{B}(i,k)$.

Now consider the case that the algorithm couldn't assign job j. Then machine m has load at least $\beta\mathsf{Opt}s_m$. Let h be minimal such that all machines in the range $[h, m]$ have load at least $\beta\mathsf{Opt}$. Then we know that either $h = 1$ or $p_j/s_{h-1} > \mathsf{Opt}$, otherwise the algorithm would have put job j on machine $h-1$. In either case, optimal cannot put job j on a machine with index $\leq h-1$. Thus $\mathcal{A}(h,k)$ implies $\mathcal{B}(h,k)$ as $B_h(k) - A_h(k)$ and $B_h^*(k) - A_h^*(k)$ both increase by p_j. Now we just repeat the argument in the last paragraph to prove that

$B_m(k) \leq B^*_m(k) + cs_m\mathsf{Opt}$. Since $B_m(k) \geq \beta s_m\mathsf{Opt}$, we have $B^*_m(k) \geq (\beta - c)s_m\mathsf{Opt}$, which is a contradiction to Opt if $\beta - c > \epsilon + 1$. ◀

▶ **Lemma 7.** *One can verify that $\alpha = 2$, $c = 3$ and $\beta = 4$ satisfies the stated inequalities when $\epsilon = 0$, and thus IDF with these parameters is 5-competitive, assuming that its estimate of* Opt *is correct.*

Now consider the case that the algorithm does not know Opt. It is easy to see that the whole analysis goes through as long as our estimate of Opt is no smaller than the actual Opt. If our algorithm fails to assign a job, the algorithm sets its new estimate of optimal, Opt', to be $\mathsf{Opt}(\beta + 1)/\alpha$. Then we know that $A_i \leq \alpha S_i\mathsf{Opt}'$ since $A_i \leq (\beta + 1)S_i\mathsf{Opt}$. Lemma 6 still goes through since all machines have load at most $\alpha\mathsf{Opt}'$. More precisely, the proof of Lemma 6 does not need to appeal to the slow fit phase to prove the invariants since no machine is currently saturated. Since our estimate of Opt can be at most $(\beta + 1)/\alpha$ larger than the true Opt, we derive a competitive ratio of $(\beta + 1)^2/\alpha$, which is 25/2 for the above choice of α and β.

Acknowledgements. We thank Amos Fiat for introducing us to this problem in a talk [11] under the auspices of the Algorithms and Uncertainty program at the Simons Institute for the Theory of Computing at the University of California at Berkeley, and for several subsequent helpful discussions.

References

1 Fidaa Abed, José R. Correa, and Chien-Chung Huang. Optimal coordination mechanisms for multi-job scheduling games. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 13–24, 2014.

2 S. Anand, Karl Bringmann, Tobias Friedrich, Naveen Garg, and Amit Kumar. Minimizing maximum (weighted) flow-time on related and unrelated machines. *Algorithmica*, 77(2):515–536, 2017.

3 James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44(3), May 1997.

4 Yossi Azar, Bala Kalyanasundaram, Serge A. Plotkin, Kirk Pruhs, and Orli Waarts. On-line load balancing of temporary tasks. *Journal of Algorithms*, 22(1):93–110, 1997.

5 Nikhil Bansal and Bouke Cloostermans. Minimizing maximum flow-time on related machines. *Theory of Computing*, 12(1):1–14, 2016.

6 Julian Birkinshaw. Strategies for managing internal competition. *California Management Review*, 44(1):21–38, 2001.

7 George Christodoulou, Elias Koutsoupias, and Akash Nanavati. Coordination mechanisms. In *International Colloquium on Automata, Languages and Programming*, pages 345–357, 2004.

8 Ilan Reuven Cohen, Alon Eden, Amos Fiat, and Lukasz Jez. Pricing online decisions: Beyond auctions. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 73–91, 2015.

9 Shelley DuBois. Internal competition at work: Worth the trouble? *Fortune*, February 25 2012.

10 Michal Feldman, Amos Fiat, and Alan Roytman. Makespan minimization via posted prices. unpublished, 2017.

11 Amos Fiat. Makespan minimization via posted prices, November 3 2016. talk given under the auspices of the Algorithms and Uncertainty program at the Simons Institute for the Theory of Computing at the University of California at Berkeley.

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

13 Nicole Immorlica, Erran L. Li, Vahab S. Mirrokni, and Andreas S. Schulz. Coordination mechanisms for selfish scheduling. In *International Workshop on Internet and Network Economics*, pages 55–69, 2005.

14 Noam Nisan and Amir Ronen. Algorithmic mechanism design (extended abstract). In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing, May 1-4, 1999, Atlanta, Georgia, USA*, pages 129–140, 1999.

15 Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay Vazirani. *Algorithmic Game Theory.* Cambridge University Press, New York, NY, USA, 2007.

16 Kirk Pruhs, Jirí Sgall, and Eric Torng. Online scheduling. In *Handbook of Scheduling - Algorithms, Models, and Performance Analysis.* 2004.