General Profit Scheduling and the Power of Migration on Heterogeneous Machines

Sungjin Im* Benjamin Moseley†

July 3, 2015

Abstract

In this paper we consider the power of migration in heterogenous machines settings and general profit scheduling. We begin by showing that on related machines or on related machines with restricted assignment that any migratory algorithm can be simulated by a non-migratory algorithm given $1+\epsilon$ speed augmentation and $O(\frac{1}{\epsilon})$ and $O(\frac{1}{\epsilon^2})$ machine augmentation, respectively, for any $0 < \epsilon \leq 1$. Similar results were only known in the case of identical machines and our results effectively show that migration does not give too much additional power to an algorithm, even in heterogeneous environments. Our results are constructive and can be computed efficiently in the offline setting. We complment our result by showing that there exists migratory schedules on related machines which require $\Omega(\frac{1}{\epsilon})$ machine augmentation with $(1+\epsilon)$-speed to be simulated by any non-migratory scheduler for any $0 < \epsilon \leq 1/2$, showing that machine augmentation without speed augmentation is insufficient for a non-migratory scheduler to simulate a migratory scheduler.

We then use these results to study general profit scheduling where a set of n jobs arrive over time online and every job i has a function $g_i(t)$ specifying the profit of completing job i at time t. The goal of the schedule is to maximize the total profit obtained. We give a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive algorithm in the *unrelated* machines setting for any $\epsilon > 0$ when comparing against a non-migratory adversary. Previous results were only known in the identical machines setting. As an example of the usefulness of the previous results on migration, they with the results on genial profit scheduling give a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^4})$-competitive algorithm for general profit scheduling when comparing against a migratory algorithm on related machines with restricted assignment for any $\epsilon > 0$.

*Electrical Engineering and Computer Science, University of California, 5200 N. Lake Road, Merced CA 95344. sim3@ucmerced.edu. Supported in part by NSF grant CCF-1409130.

†Department of Computer Science and Engineering, Washington University in St. Louis, 1 Brookings Drive, St. Louis, MO 63130. bmoseley@wustl.edu.

1 Introduction

In online scheduling theory a fundamental area of research focuses on designing algorithms for completing a set of jobs to maximize the amount of profit obtained. In one of the most basic formulations of this problem, there are n jobs that arrive over time and each job i arrives at some time r_i. Each of these jobs i is associated with a profit w_i and a deadline d_i. If the job i is completed by its deadline then a profit of w_i is obtained and no profit is obtained for the job otherwise. This problem is known as maximizing throughput.

Throughput maximization has been extensively studied in the case where the jobs are to be scheduled on a single machine and preemption is allowed. In this case, every job has some processing time p_i. This line of work has resolved the complexity of the problem and it is known that the optimal deterministic competitive ratio is $\Theta(\delta)$ where δ is ratio of the maximum to minimum density of a job [3, 4, 11, 13]. The density of a job is $\frac{w_i}{p_i}$, which intuitively is the profit obtained for each unit of the job processed. The optimal randomized competitive ratio is $\Theta(\min\{\log \delta, \log \Delta\})$ where Δ is the ratio of the maximum to minimum job size [6, 10].

Unfortunately, these results can be viewed as unsatisfying since they depend on δ and Δ, which could be super polynomial in the size of the problem instance. In this face of these strong lower bound, previous work has resorted to a resource augmentation analysis where the algorithm is given extra speed over the adversary [7]. An algorithm is said to be s-speed c-competitive if it can process jobs an s factor faster than the adversary and achieves a profit within $1/c$ factor of the optimal solution. Using resource augmentation, a $(1+\epsilon)$-speed $O(f(\epsilon))$-competitive algorithm is known for any fixed $\epsilon > 0$ where f is a function of only ϵ [7].

Work has also considered this problem in more general machine environments [10, 6, 12, 8]. One such environment is the *identical* machines setting. In this case, the jobs can be scheduled on a set of m machines and every job has the same processing time no matter which machine the job is processed on. In this setting, a $(1+\epsilon)$-speed $O(1)$ competitive algorithms are known for any fixed constant $\epsilon > 0$ [12, 8]. Interestingly, as far as the authors are aware, competitive online algorithms have not been discovered in more general environments, such as related machine, restricted assignment or unrelated machines. In the related machines setting each machine i runs at a speed s_i and the processing time of a job on a machine is $\frac{p_i}{s_i}$. In restricted assignment setting, a job i can only be processed on some subset of the machine and has processing time p_i on each machine it can be processed on. Finally, the unrelated machines setting is the most general model where the processing time of job i on machine j is $p_{i,j}$ and the processing times of jobs on machines are arbitrary.

Relatively recently, the throughput problem was generalized to the following problem, which we refer to as *general profit scheduling*. In this problem, instead of each job i having a weight, it is associated with its own *individual* function $g_i(t)$. The value $g_i(t)$ specifies the profit of completing job i at time t and the goal is to maximize $\sum_{i\in[n]} g_i(C_i)$ where C_i is the completion time of job i under some schedule. In past work, the only assumption made of the functions $g_i(t)$ is that they are non-increasing or, in other words, there should be no incentive to complete a job later. For this very general problem, $(1+\epsilon)$-speed $O(\frac{1}{\epsilon})$-competitive algorithm is known on a single machine [2] as well as on identical machines [12].

A question that is intimately tied with the problems of maximizing throughput, general profit scheduling, and many other problems is what power does an algorithm gain by using *migration*? That is, when scheduling on more than one machine, a preemptive algorithm could be allowed to migrate a job between the machines. The question is can an algorithm leverage this power to obtain significantly better schedules than an algorithm which must process each job on exactly one machine?

In general, a migratory algorithm can feasibly complete more jobs than a non-migratory scheduler in the throughput setting. Interestingly, it has been shown that this power is fairly limited though. In particular, an algorithm was shown in [8] which can convert any feasible migratory schedule on m identical machines to a a scheduler that is non-migratory using $6m$ machines. The schedule ensures that every job is completed only earlier than in the migratory schedule. This immediately implies that there exists a non-migratory scheduler

can achieve profit within $1/6$ of any non-migratory scheduler for throughput and general profit scheduling without any additional machine or speed augmentation. To see this, consider doing the conversion and then only choosing to schedule jobs on the m machine that achieve the largest profit. In general this fundamental result shows that for effectively any reasonable objective, one can convert a migratory scheduler to a non-migratory scheduler with a small amount of machine augmentation. Other results that study the power of migration can be found in [5, 9].

This line of work on profit scheduling problems and on the power of migration has left open many intriguing questions. Do there exist competitive algorithms for throughput maximization in heterogeneous machine settings such as related machine, restricted assignment or unrelated machines? Could these be extended to find algorithms for general profit scheduling in these settings? Finally, what is the power of migration on heterogeneous machines? In this paper, we work towards answering these questions.

Results: Our work begins by addressing the question of converting migratory schedules to non-migratory schedules on heterogeneous machines. We say that a non-migratory schedule *simulates* a migratory schedule if the following three properties hold (1) every job is processed only after its arrival; (2) every job is completed only possibly earlier in the non-migratory schedule; and (3) each job is processed on at most one machine. Our main result for the conversion is for the related machines with restricted assignment setting where machines have different speeds and each job can only be processed on some subset of the machines.

Theorem 1.1. *For any migratory schedule on m related machines with restricted assignment there exists a non-migratory schedule that simulates this schedule when every machine i is given a $(1+\epsilon)$ factor additional speed and there are $O(\frac{1}{\epsilon^2})$ copies of each machine for any $0 < \epsilon \leq 1$.*

While a similar result for the related machines setting (with no restricted assignment) (see Theorem 2.4) easily follows with a small observation from the previous work [1] on minimizing maximum weighted flowtime on related machines, we use different techniques to obtain a more general result. Succinctly, the result in [1] can be viewed as a clever generalization of the 'slow-fit' algorithm on minimizing makespan on related machines where each job is scheduled on the slowest machine it can be – to handle jobs arriving over time, a laminar family of intervals is defined, and a careful volume argument is performed. On the other hand, we cannot use a greedy rule to assign jobs to machines since jobs can be placed on different subsets of machines. Due to this, we create a network flow for jobs of similar sizes, and obtain an integral flow which readily translates into an actual assignment. Here we carefully use the constraints imposed on intervals that form a laminar family, and show no machines get too large volume on any interval in the laminar family. Here the main challenge is to ensure the volume does not accrue too much over multiple assignments coming from different network flows. We note that the conversion is constructive and offline.

To complement our positive results, we further show that machine augmentation is insufficient without speed augmentation on related machines (even without restricted assignment) unlike the identical machine result of [8]. This shows that our conversion results are essentially tight for any fixed ϵ.

Theorem 1.2. *There exists a migratory schedule on related machines such that any non-migratory schedule with $(1+\epsilon)$-speed requires $\Omega(\frac{1}{\epsilon})$-machine augmentation for any $0 \leq \epsilon \leq 1/2$.*

We leave the case for unrelated machines as an interesting open question, but note that it appears to require new technique beyond those used in our work.

With these results in place, we work towards discovering competitive non-migratory schedulers for throughput and general profit scheduling on heterogeneous machines. We show that the novel results of [2] on a single machine, can be generalized to the full extent of unrelated machines.

Theorem 1.3. *There exists a non-migratory schedule on unrelated machines with $(1+\epsilon)$-speed that achieves a $O(\frac{1}{\epsilon^2})$-competitive for general profit scheduling when compared against a non-migratory adversary.*

With these results, we can compare against a non-migratory adversary on related machines with restricted assignment using only $(1+\epsilon)$ speed augmentation and no machine augmentation. Indeed, for any migratory optimal solution one can use Theorem 1.2 to convert it to a migratory schedule with $(1+\epsilon)$ speed augmentation and $O(\frac{1}{\epsilon^2})$ machine augmentation. Then, for each machine of the original schedule, choose the copy which achieves the highest profit in the non-migratory schedule to get a $(1+\epsilon)$-speed $O(\frac{1}{\epsilon^2})$-competitive non-migratory scheduler. One can then compare against migratory schedule to derive the following corollary.

Corollary 1.4. *There exists a non-migratory schedule on related machines with restricted assignment with $(1+\epsilon)$-speed that achieves a $O(\frac{1}{\epsilon^4})$-competitive for general profit scheduling when compared against a migratory adversary.*

Organization: We begin by studying how to covert migratory schedules to non-migratory schedules in Section 2 and in Section 2.2 we show a lower bound on the machine augmentation needed for converting a migratory schedule to a non-migratory schedule in the related machines setting. In Section 3 we study the general profit scheduling problem on unrelated machines when comparing against a non-migratory adversary.

2 Relationship Between Migratory and Non-Migratory Schedules

In this section we begin by showing how to covert a migratory schedule to a non-migratory schedule in the related machines setting. Later in the section, we will discuss the related machines with restricted assignment setting. Throughout this section, we assume that we are given a migratory schedule on either related machines or related machines with restricted assignment. Given this migratory we define d_i to be the time the schedule completes job i. We will refer to this as job i's deadline and the time interval $[r_i, d_i]$ as job i's window. The window for a job, is the feasible times our scheduler can execute the job. Note that this time is completely dependent on the given migratory schedule. Our goal is to show a scheduler which completes all jobs by their deadlines, while ensuring every job is executed on only one machine.

We begin by defining formally the definition of the type of speed and machine augmentation we will use in the conversion.

Definition 2.1. *We say that a schedule is feasible with s-speed μ-machine augmentation if all jobs are completed during their respective windows (and at any point in time no job is scheduled on more than one machine) when each machines has μ identical copies and all machines run s times faster than their original speed.*

In the following lemma, we consider the following situation. Say one has a schedule which ensures that all jobs are scheduled without using migration, but using an infeasible schedule where a job can be executed before its arrival and after its deadline, but the interval the job can be executed in can be bounded by its window length $(d_i - r_i)$. Then we can use such a schedule to construct a feasible non-migratory schedule.

Lemma 2.2. *For any constant $\delta < 1$, consider a schedule on a* single *machine with speed s where each job j is scheduled during time interval $[r_j - \frac{1}{\delta}(d_j - r_j), d_j + \frac{1}{\delta}(d_j - r_j)]$, which we call j's $(1/\delta)$-extended window. Further, assume that in this schedule $d_j - r_j \geq cp_j$ for all jobs j for some constant $c > 0$. Then, for any $\gamma > \frac{1}{c} - 1$, there exists a non-migratory schedule on $\lfloor (2/\delta + 3)s/(1+\gamma-\frac{1}{c}) + 1 \rfloor = O(\frac{s}{(1+\gamma-\frac{1}{c})\delta})$ identical parallel machines running with speed $1+\gamma$ where each job is scheduled on a single machine (i.e., in a non-migratory way) during the time interval $[r_j, d_j]$. Further, such a schedule can be found in polynomial time.*

Proof. For notational simplicity, let $\ell_j := d_j - r_j$ and $m := \lfloor (2/\delta + 3)s/(1+\gamma-\frac{1}{c}) \rfloor + 1$. We construct a desired non-migratory schedule as follows. Consider jobs in increasing order of their window sizes, and

schedule each job j on an arbitrary machine that has enough idle times to preemptively process the job before its deadline and after its arrival. Observe that job j can not be assigned to machine i only if the machine processes at least $(1+\gamma-\frac{1}{c})\ell_j$ volume of work for jobs of smaller windows during $[r_j, d_j]$ since machines run with speed $(1+\gamma)$ and $d_j - r_j \leq cp_j$. We show that we can always find a machine to process each job j.

For the sake of contradiction, say j is the first job we couldn't schedule. Then, every machine must process at least $(1+\gamma-\frac{1}{c})\ell_j$ volume of work during $[r_j, d_j]$ for jobs of smaller window than job j. Such jobs arrive no earlier than $r_j - \ell_j$ and no later than d_j, meaning that they must be completed by time $d_j + \ell_j(1+1/\delta)$ under the given schedule on a single machine. Hence we found a set of jobs of total volume at least $(1+\gamma-\frac{1}{c})\ell_j m$ that must be completed during $[r_j - (1+1/\delta)\ell_j, d_j + \ell_j(1+1/\delta)]$ on a *single* machine of speed s. Since the given schedule can process at most $(2/\delta+3)s\ell_j$ volume of work during the interval, we obtain $m \leq (2/\delta+3)s/(1+\gamma-\frac{1}{c})$, which is a contradiction. □

We note that the relationship between migratory and non-migratory schedules on related machines easily follows from the following theorem from [1] and Lemma 2.2.

Theorem 2.3. *[1] There is a $O(1/\epsilon^4)$-competitive non-migratory algorithm to minimize maximum weighted flow time on related machines with $1+\epsilon$ speed augmentation for any $0 < \epsilon \leq 1$. The competitive ratio holds even against a migratory optimal schedule. Furthermore, every job i is scheduled on a machine j such that $\frac{p_i}{s_j} \leq \frac{1}{w_i}$, where w_i is the weight of job i and assuming that the optimal solution objective is at most 1.*

Theorem 2.4. *For any migratory schedule on m related machines there exists a non-migratory schedule that simulates this schedule when every machine i is given $O(1+\epsilon)$ factor additional speed and there are $O(\frac{1}{\epsilon})$ copies of each machine for any $0 < \epsilon \leq 1$.*

Proof. For each job j, define its weight w_j to be $1/(d_j - r_j)$. Note that the given migratory schedule has the maximum weighted flow time of $(d_j - r_i)w_j = 1$. Theorem 2.3 implies the existence of a non-migratory schedule with 2-speed that has the maximum weighted flow time of at most $O(1)$. Note that in this non-migratory schedule each job must complete by time $r_j + O(1)/w_j = r_j + O(1)\cdot(d_j - r_j)$. Applying Lemma 2.2 with $\delta = O(1)$, $c = 1$ and $s = 2$ on each machine together with jobs on it, we can find a non-migratory schedule with $1+\gamma$ speed and $O(1/\gamma)$-machine augmentation. □

2.1 Related Machines with Restricted Assignment

We now consider the more restricted case where machines have different speeds and each job only can be assigned to a subset of machines. We call this setting as related machines setting with restricted assignment. The remainder of this section is devoted to proving Theorem 1.1. As before, we consider an arbitrary migratory schedule and set d_i to be the time job i is completed in the migratory schedule. Our goal will be to construct a non-migratory schedule that completes each job i during $[r_i, d_i]$.

For simplicity, we will use slowness factor instead of speeds – a machine with slowness factor s requires sp time steps to complete a job of size p. To begin with, we change the original migratory schedule in the following manner. Let $0 \leq \epsilon < 1$ be some constant. We want to convert the schedule to another schedule that ensures a job i is only scheduled on machines j where $d_i - r_i \geq \frac{s_j p_i}{1+\epsilon}$. This is so that later we can apply Lemma 2.2. Note that $s_j p_i$ is the processing time of job i on machine j. To do this, notice that the total aggregate amount any job i can be processed on all machines j where $d_i - r_i < \frac{s_j p_i}{1+\epsilon}$ is $\frac{1}{1+\epsilon}p_i$ because a job can only be processed on one machine at any point in time and the job is completed during $[r_i, d_i]$. Using this, we can remove any portion of a job that is processed on any machine where $d_i - r_i < \frac{s_j p_i}{1+\epsilon}$. Then we use $\frac{1+\epsilon}{\epsilon}$ speed augmentation on each machine to ensure every job i is fully processed on machines j where $d_i - r_i \geq \frac{s_j p_i}{1+\epsilon}$. Then the total amount a job is processed on these faster machines is at least $p_i(1-\frac{1}{1+\epsilon})\frac{1+\epsilon}{\epsilon} = p_i$.

To clarify notation for remainder of the proof, we scale the slowness of machines such that machine j has slowness $s'_j := s_j \cdot \frac{\epsilon}{1+\epsilon}$ and work with these new speeds. Note that now $d_i - r_i \geq \frac{s'_j p_i}{\epsilon}$ for any machine j job i is processed on. Now, we assume that each job i is further restricted so that it cannot be processed on a machine j where $d_i - r_i < \frac{s'_j p_i}{\epsilon}$. We will work with the modified instance. To show our main theorem, we will later reduce this $O(\frac{1}{\epsilon})$ speed augmentation by using additional machine augmentation.

Consider any machine j. For each machine j we create a laminar family of intervals: Intervals of length $1, (0,1), (1,2), (2,3), ...$ are at the lowest level, and those of length $2^h, (0, 2^h), (1 \cdot 2^h, 2 \cdot 2^h), (2 \cdot 2^h, 3 \cdot 2^h), ...$ are at the hth lowest level – the top level interval should be long enough to cover the whole schedule. The created intervals for machine j are denoted as $\mathcal{I}_j$. We let $I(j, k', l)$ denote the interval $[(l-1) \cdot 2^{k'}, l \cdot 2^{k'}]$ in $\mathcal{I}_i$.

We create a single-source-single-sink network flow graph G_k for *all* jobs with size in $[2^{k-1}, 2^k)$ as follows. The source and sink nodes are denoted as $q_{k,s}$ and $q_{k,t}$, respectively. There is a unique node for each job i where $p_i \in [2^{k-1}, 2^k)$; for notational simplicity, we also refer to the node as job node i. The source node $q_{k,s}$ is connected to each job node j with capacity 1. For each machine j we create a node for each interval in $\mathcal{I}_j$. Again, we abuse the notation $I(j, k', l)$ to denote the node created for the interval $I(j, k', l) \in \mathcal{I}_j$. Each job node i is connected to the interval node $I(j, k', l)$ if the following conditions hold: (1) The interval $I(j, k', l)$ intersects $[r_i, d_i]$; (2) job i is allowed to be scheduled on machine j (this is, j is not a restricted machine for job i); and $s'_j 2^k \in [2^{k'-1}, 2^{k'})$. The capacity of these edges are 1. We note that the set of interval nodes a job is connected to corresponds to the machines and time intervals where the job can be feasibly processed in the modified migratory schedule and the length of these intervals are within a factor 4 of the processing time of i on j.

Before we move to explaining how interval nodes are connected, we define $\tilde{x}_{kjI}$ to be the total fractional number of jobs of size in $[2^{k-1}, 2^k)$ processed on machine j during time interval I. In other words, $\tilde{x}_{kjI} = \sum_{i \,:\, p_i \in [2^{k-1}, 2^k)} \frac{p'_{i,j}}{p_i}$ where $p'_{i,j}$ is the amount job i is processed on machine j in the migratory schedule. Each interval node $I(i, k', l)$ is connected to its unique parent interval node $I(j, k'+1, \lceil l/2 \rceil)$ with capacity $\lceil \tilde{x}_{kjI} \rceil$ where $I = I(j, k', l)$. The top level interval node I' for machine j is connected to the sink node $q_{k,t}$ with capacity equal to $\lceil \tilde{x}_{kjI'} \rceil$. This completes the description of G_k.

We now explain how to find a non-migratory schedule using the network flow graphs we constructed above. We observe that there is a fractional flow respecting the capacity constraints for each graph G_k such that one unit of flow is pushed through each job node. Such a flow is achieved by a natural flow where an edge from each job node i with size in $[2^{k-1}, 2^k)$ sends flow directly to an interval node $I = I(j, k', l)$, for machine j, equal to the fractional amount job i processed during $I(j, k', l)$ on machine j. A total of unit flow is pushed from each job node i. The flow on other edges are defined following flow conservation. The total flow passing through an interval node I will be exactly $\tilde{x}_{kjI}$. For each G_k we find an integral max flow respecting the capacities. This integral flow can be interpreted as an assignment $\mathcal{A}$ of jobs i with size in $[2^{k-1}, 2^k)$ to intervals $I(j, k', l)$ where $s'_j 2^k \in [2^{k'-1}, 2^{k'})$. That is, a job i is matched to an interval I such that node i is directly connected to the interval node I. We say job i is matched to interval I if the node for I is directly connected to the node for j and a unit of flow passes from node i to node I. Our goal is to use this assignment to construct a feasible non-migratory schedule.

We observe that the discovered assignment $\mathcal{A}$ has the following properties. Recall that $\tilde{x}_{kjI}$ is the fractional number of jobs of size 2^k assigned to interval I on machine j.

Claim 2.5. *Let $\bar{x}_{kjI}$ denote the integer number of jobs of size in $[2^{k-1}, 2^k)$ assigned to an interval I on machine j under the assignment $\mathcal{A}$. Then, $\mathcal{A}$ has the following properties.*

1. $\bar{x}_{kjI} - \tilde{x}_{kjI} \leq 1$.
2. *$\mathcal{A}$ assigns each job i with size $p_i \in [2^{k-1}, 2^k)$ to a unique machine j and interval $I(j, k', l)$ whose length $2^{k'}$ is such that $s'_j 2^k \in [2^{k'-1}, 2^{k'})$. Further j is a feasible machine for i.*

3. If $\mathcal{A}$ assigns job i to I, then the interval I is completely contained in $[r_j - 4(d_j - r_j), d_j + 4(d_j - r_j)]$.

Proof. The first property is immediate from the fact that $\mathcal{A}$ is derived from a flow respecting the capacity constraints. The second property follows from the fact that job node i is only directly connected to intervals nodes corresponding to machines that i can be feasibly scheduled on and have the desired length. To see why the third property holds, recall that in the modified migratory schedule job i can only be processed on machines j where $d_i - r_i \geq \frac{s'_j p_i}{\epsilon}$. Thus if job i is matched to an interval $I(j, k', l)$ on machine j then the length of $I(j, k', l)$, $2^{k'}$, must be less than $2s'_j 2^k \leq 4s'_j p_i \leq 4\epsilon(d_i - r_i) \leq 4(d_i - r_i)$. By definition of the flow networks, a job node i is only connected to interval nodes where $I(j, k', l)$ intersects $[r_j, d_j]$. Therefore, i's 4-extended window must include I. □

We now show that any interval $I(j, k', l)$ does not get assigned a lot more volume of jobs than it can handle over the assignment generated by combining the matchings from each graph G_k.

Lemma 2.6. *Fix any machine j and consider an interval $I(j, k', l)$. Let $J(j, k', l)$ be the set of all jobs matched to an interval $I(j, k, l) \subseteq I(j, k', l)$, including $I(j, k', l)$ itself over all matchings found. The total processing time of the jobs in $J(j, k', l)$ on machine j, $s'_j \sum_{i \in J(j,k',l)} p_i$, is less than $4s'_j 2^{k'}$.*

Proof. Fix any machine j and an interval $I(j, k', l)$. Consider a flow graph G_k and let k'' be such that $s'_j 2^k \in [2^{k''-1}, 2^{k''})$ and $k'' \leq k'$. Let $I = I(j, k', l)$. Notice that the total number of jobs with $p_i \in [2^{k-1}, 2^k)$ matched in G_k to an interval $I(j, k'', l) \subseteq I(j, k', l)$ is at most $\lceil \tilde{x}_{kjI} \rceil$. This is because node $I = I(j, k', l)$ in G_k has one outgoing edge of capacity $\lceil \tilde{x}_{kjI} \rceil$ and all flow through nodes $I(j, k'', l)$ where $I(j, k'', l) \subseteq I(j, k', l)$ must pass through node $I(j, k', l)$. We see that, $s'_j \sum_{i \in J(j,k',l), p_i \in [2^{k-1}, 2^k)} p_i \leq 2^{k''} \lceil \tilde{x}_{kjI} \rceil \leq 2^{k''}(\tilde{x}_{kjI} + 1)$.

Now our goal is to aggregate the processing time of all jobs over all matchings found to bound the processing time of jobs in $J(j, k', l)$. We see that,

$$\begin{aligned} s'_j \sum_{i \in J(j,k',l)} p_i &\leq \sum_{k \,:\, s'_j 2^k \in [2^{k''-1}, 2^{k''}), k'' \leq k'} 2^{k''}(\tilde{x}_{kjI} + 1) \leq 2^{k'+1} + \sum_{k \,:\, s'_j 2^k \in [2^{k''-1}, 2^{k''}), k'' \leq k'} 2^{k''} \tilde{x}_{kjI} \\ &\leq 2^{k'+1} + 2^{k'+1} \end{aligned}$$

The last inequality follows from the fact that the migratory schedule was able to processed a $\tilde{x}_{kjI}$ fraction of jobs of size in $[2^{k''-1}, 2^{k''})$ during $I(j, k', l)$ where $s'_j 2^k \in [2^{k''-1}, 2^{k''})$. □

We now apply Lemma 2.2 for each machine j. Recall that we scaled each machine's slowness to be $s'_j = s_j \frac{\epsilon}{1+\epsilon}$ at the beginning of the proof. We now want to convert back to the original problem instance while ensuring we have a feasible schedule. Notice that if each machine j runs with slowness $s'_j/4$ it can complete all jobs i such that i is only processed during $[r_j - 4(d_j - r_j), d_j + 4(d_j - r_j)]$ by the previous lemma. Further, we know by definition of the problem instance $d_i - r_i \geq \frac{s'_j p_i}{\epsilon} = \frac{s_j p_i}{1+\epsilon}$. We can then applying Lemma 2.2 for each machine j by assuming job i has processing time $s_j p_i$ on machine j, and setting $\delta = \frac{1}{4}$, $s = \frac{4(1+\epsilon)}{\epsilon}$, $c = \frac{1}{1+\epsilon}$ and $\gamma = 1 + 2\epsilon$. This give a desired non-migratory schedule where each machine j has speed- $(1 + 2\epsilon)s_j$ and there are $O(1/\epsilon^2)$ copies of machine j, which completes the proof of Theorem 1.1.

2.2 Lower Bound on Machine Augmentation

In this section, our goal is to show Theorem 1.2, showing that at least $\Omega(\frac{1}{\epsilon})$ machine augmentation is needed to covert a migratory schedule to a non-migratory schedule in the related machines setting.

Proof of [Theorem 1.2] Fix any constant $0 < \epsilon \leq 1/2$ and consider the following problem instance. There are n jobs of size 1 that are all released at time 0. There is a set M_1 of n speed-1 machines and a set M_2 of ϵn speed-5 machines. This completes the description of the instance.

First we show that there exists a migratory schedule that completes all of the jobs by time $1-2\epsilon$. Partition the jobs into groups $G_1, G_2, \ldots G_{\epsilon n}$ such that each group contains at most $\frac{1}{\epsilon}$ jobs. In the schedule, the jobs in group G_i is associated with a unique machine in M_2. Additionally, every job is associated with a unique machine in M_1. Each job in G_i is processed on the associated machine in M_2 for $\frac{\epsilon}{2}$ time steps during $[0, \frac{1}{2}]$. This can be done by say round robining the jobs in G_i on the machine. Every time step where a job is not being processed on a machine in M_2 the job is processed on its unique machine in M_1. Now we show that the jobs are competed by time $1 - 2\epsilon$. Indeed, each job receives $\frac{\epsilon}{2} \cdot 5$ amount of processing by a machine in M_2 and in the remaining $1 - \frac{\epsilon}{2} - 2\epsilon$ time steps during $[0, 1 - 2\epsilon]$ the jobs are processed on a machine in M_1 of speed 1. Thus, the amount of processing completed on a job is $\frac{\epsilon}{2}5 + 1 - \frac{\epsilon}{2} - 2\epsilon = 1$.

Now consider any non-migratory schedule with speed augmentation $1 + \epsilon$ and at most $\frac{1}{10\epsilon}$ machine augmentation. Let M_1' denote all of machines of speed $1 + \epsilon$ in this schedule and M_2' denote the machines of speed $5(1+\epsilon)$. We will show that this schedule will not be able to complete all of the jobs by time $1 - 2\epsilon$. First notice that no job can be scheduled on a machine in M_1' because then the job will not complete until time $\frac{1}{1+\epsilon} > 1 - 2\epsilon$ for $0 < \epsilon \leq 1$. Thus, all of the jobs are scheduled on machines in M_2'. There are $\frac{1}{10\epsilon} \cdot \epsilon n = \frac{n}{10}$ such machines. Thus there must be some machine that is assigned 10 jobs. However, these jobs then do not finish until time $10/(5(1 + \epsilon)) > 1 - 2\epsilon$. □

We note that in the proof that one machines with two different speeds are needed and the speeds are bounded by a constant factor of each other. Notice that this theorem implies the following corollary.

Corollary 2.7. *Machine augmentation (and no speed augmentation) is insufficient to convert a migratory schedule to a non-migratory schedule.*

3 Profit Scheduling on Unrelated Machines

In this section, we consider the following general profit scheduling problem in the online setting. There is a set of m unrelated machines. There are n jobs which arrive over time where job i has release time is r_i and each job i is associated with a function $g_i(t)$. The function $g_i(t)$ specifies the profit of completing job i at time t and the only assumption made on g_i is that it is non-increasing. That is, there should be no additional profit for making a job wait longer to be completed. Notice the each job has its own *individual* profit function. Each job i has a processing time $p_{i,j}$ on machine j. The goal of scheduler, which completes each job i at a time C_i, is to maximize $\sum_{i \in [n]} g_i(C_i)$.

We now define our algorithm which is inspired by [2, 12]. We assume the algorithm is given $1 + \epsilon$ speed augmentation for some $\epsilon > 0$ and we compare against a non-migratory adversary. The algorithm is non-migratory and when a job arrives it commits to the machine that will schedule the job. However, since we are in the profit setting, the scheduler may later decide not to schedule the job at all since there maybe higher profit jobs to schedule. When a job arrives the scheduler will associate it with a set of intervals I_i, which is intuitively are the times the scheduler is allowed to schedule the job. These intervals will be fixed the moment a job arrives as well as the machine the job is to be scheduled on. Let $A_j(t)$ be the set of all jobs which have been assigned to machine j by the algorithm by time t and let m_i denote the machine job i is assigned to. Note that the processing time of job i in the algorithm's schedule is then p_{i,m_i}. The total aggregate length of the intervals in I_i will be exactly $\frac{1+\epsilon/2}{1+\epsilon} p_{i,m_i}$. Implicit in the intervals I_i will be a time t_i where the algorithm hopes to complete job i by. We call this the *tentative* completion time. Given the tentative completion time of job i, let $w_i = g_i(t_i)$ be the potential profit of job i if we complete it before this completion time. Note that since g_i is non-decreasing, if we complete i before t_i the algorithm obtains at least w_i profit for job i. Finally, we define the *density* of job i to be $\frac{w_i}{p_{i,m_i}}$, the potential profit of the job in the algorithm's schedule divided by the processing time.

Now consider when job i arrives at time r_i. The algorithm considers each machine j and time t. When considering i and t, the algorithm tests if on machine j the time t is a good candidate tentative completion time. Let $w_i = g_i(t)$ be the potential profit of completing job i at time t. For a fixed constant α, to be set later, let $S(\frac{w_i}{\alpha})$ be the set of jobs assigned to machine j with density at least $\frac{w_i}{\alpha}$. Let I be the set of maximal time intervals during $[r_j, t]$ that do not overlap with any interval in $I_{i'}$ for a job $i' \in S(\frac{w_i}{\alpha})$. The time t is feasible if the total length of intervals in I is at least $\frac{1+\epsilon/2}{1+\epsilon} p_{i,j}$. After testing this for all machines and times, the algorithm assigns job i to the machine j with the *earliest* feasible tentative completion time and sets I_i to be the set of associated intervals for the machine and time. Note that there is always such a feasible time by allowing t to be sufficiently large.

So far, we have specified how to assign jobs to machine. As for which job to execute on each machine, at time t on machine j the algorithm schedules the job i which has the highest density *and* t is contained in some interval in I_i.

3.1 Analysis

We begin by making a few observations about the algorithm, similar to what was shown in [2].

Claim 3.1. *Consider any job i and machine j (which i may or may not have been assigned to). Let t be any time where $t \geq r_i + p_{i,j}$ and the algorithm sets the tentative completion time of i to be strictly later than t. Let L be the amount of time during $[r_i, t]$ which is contained in time intervals in any $I_{i'}$ associated with a job i' assigned to machine j which has density at least $\frac{g_i(t)}{\alpha p_{i,j}}$. It must be the case that $L \geq \frac{\epsilon/2}{1+\epsilon}(t - r_i)$.*

Proof. If L is less than $\frac{\epsilon/2}{1+\epsilon}(t - r_i)$ than t is a feasible time for job i. Since t is earlier than the tentative completion time job i, this contradicts the definition of the algorithm. □

Claim 3.2. *For any two jobs i and i' assigned to the same machine j where I_i and $I_{i'}$ contain intervals that overlap then $u_i > \alpha \frac{w_{i'}}{p_{i',j}}$ or $u_{i'} > \alpha \frac{w_i}{p_{i,j}}$.*

Proof. The proof of the previous claim follows by definition of I_i and $I_{i'}$. □

Let C be the set of jobs completed by the algorithm. We begin by showing that the profit of the job's completed by the algorithm is close to the profit that algorithm would have obtained if it completed all jobs by their tentative completion times.

Lemma 3.3. *The total profit obtained for jobs in C is at least as much as $(1 - \frac{2+\epsilon}{4\epsilon(\alpha-1)}) \sum_{i \in [n]} w_i$, which is $(1 - \frac{2+\epsilon}{4\epsilon(\alpha-1)})$ multiplied by the total profit the algorithm would have obtained if it completed all jobs by the tentative completion time.*

Proof. We will utilize a charging scheme separately for each machine j. For each job i in C we allocated w_i units of profit. This profit will be transferred to other jobs ensuring that every job i' receives $(1 - \frac{2+\epsilon}{4\epsilon(\alpha-1)}) w_{i'}$ units of profit, proving the lemma. For any time t, let $X_{t,j}$ be the set of jobs i assigned to machine j that has an interval in I_i which contains t. The job i with the highest density in $X_{t,j}$ transfers profit to the other jobs in $X_{t,j}$ an amount of $(\frac{1+\epsilon}{2\epsilon}) \frac{w_i}{p_{i,j}}$ for t.

The proof will follow by showing that each job must have w_i units of profit assigned to it and by showing that not too much profit is transferred out of the job. For each job in C, it is assigned w_i units of profit. For any job i assigned to machine j which is not completed, it must be the case that the algorithm was executing a more dense job for $\frac{\epsilon}{2(1+\epsilon)} p_{i,j}$ units of time contained in I_i by definition of the algorithm. Knowing that any job i' is only executed at a time during an interval $I_{i'}$, the total credit transferred to i is at least $\frac{1+\epsilon}{2\epsilon} p_{i,j} (\frac{1+\epsilon}{2\epsilon}) \frac{w_i}{p_{i,j}} = w_i$. Thus, every job is assigned w_i profit.

Now, we show that no job i has $\frac{2+\epsilon}{4\epsilon(\alpha-1)} w_i$ profit transferred to other jobs. Consider any time t and machine j. By Claim 3.2 it is the case that the jobs i' in $X_{t,j}$ have geometrically decreasing densities on

machine j. Let i be the job in $X_{t,j}$ with the highest density. The total profit transferred from job i at time t is $(\frac{1+\epsilon}{2\epsilon})\frac{w_i}{p_{i,j}}\sum_{k=1}^{\infty}\frac{1}{\alpha^k} \leq (\frac{1+\epsilon}{2\epsilon(\alpha-1)})\frac{w_i}{p_{i,j}}$. Knowing that the total amount of time in I_i is $\frac{1+\epsilon/2}{1+\epsilon}p_{i,j}$ the total amount transferred out of job i is $\frac{1+\epsilon/2}{1+\epsilon}p_{i,j}(\frac{1+\epsilon}{2\epsilon(\alpha-1)})\frac{w_i}{p_{i,j}} = \frac{2+\epsilon}{4\epsilon(\alpha-1)}w_i$. Thus, every job i is finally assigned $(1-\frac{2+\epsilon}{4\epsilon(\alpha-1)})w_i$ profit. □

We have now shown that the total profit the algorithm obtains is close to the total profit the algorithm would have received if it completed jobs by their tentative completion times. Our final goal is to bound the total profit the algorithm would have obtained if it completed jobs by their tentative completion time by the profit the adversary obtains, which will complete the analysis. Let $P^*(X)$ be the profit he adversary obtains for a set of jobs X and $P(X)$ be the profit the algorithm obtains for a set of jobs in X if it completed the jobs in X by their tentative completion times. Let A_1 be the set of jobs i where the algorithm sets the tentative completion time i to earlier than when the adversary completes i. Let A_2 be the remaining jobs. Clearly $P(A_1) \geq P^*(A_1)$ since the jobs' profit functions are non-increasing. Thus, our goal is to bound $P^*(A_2)$ by the profit the algorithm obtains.

Consider some machine j. Let $A^*_{2,j}$ (resp. $A_{2,j}$) be the set of jobs the adversary (resp. algorithm) processes on machine j that are contained in A_2 and similarly let $A^*_{1,j}$ (resp. $A_{1,j}$) be the jobs the adversary (resp. algorithm) processes on machine j that are contained in A_1. Our goal will be to bound the profit the adversary receives for these jobs by the total profit the algorithm would have received if it completed all jobs assigned to machine j by their tentative deadline. Let C^*_i be the completion time of job i in the adversary's schedule and let $w^*_i = g_i(C^*)$ be the profit the adversary obtains for job i. Let $L^*_j(u)$ denote the total length of time where the adversary is scheduling a job on machine j whose density in the adversary schedule is at least u. That is, $\frac{w^*_i}{p_{i,j}} \geq u$ for a job i assigned to machine j in the adversary's schedule. Let $L_j(\frac{u}{\alpha})$ be the total length of times t such that there is at least one job i in the algorithm's schedule assigned to j where t is contained in an interval in I_i and $\frac{w_i}{p_{i,j}} \geq \frac{u}{\alpha}$.

Lemma 3.4. *For all machines j and all $u > 0$, $L^*_j(u) \leq \frac{4(1+\epsilon)}{\epsilon}L(\frac{u}{\alpha})$*

Proof. Fix a machine j. We define $[r_i, C^*_i]$ to be the window of job i in the adversary's schedule. Let W^*_j be the set of windows for jobs in $A^*_{2,j}$ whose density $\frac{w^*_j}{p_{i,j}}$ is at least u in the adversary's schedule. Let M be a minimum subset of jobs in $A^*_{2,j}$ where $\frac{w^*_j}{p_{i,j}} \geq u$ whose windows span every time contained in a window in W^*_j. By minimality, it is the case that no three job in M contain the same time in their window. Thus, we can partition M into two sets M_1 and M_2 such that no two jobs in M_1 (respectively M_2) have overlapping windows. Without loss of generality, assume that the total length of the windows for jobs in M_1 is longer. Note that the length of the windows in M_1 is at least half of the total amount of time contained in the union of the windows in W^*_j.

Now consider any job i in M_1. We know that the adversary scheduled job i on machine j during $[r_i, C^*_i]$ and therefore $C^*_i - r_i \geq p_{i,j}$. Further, we know that since $i \in A^*_{2,j}$ the algorithm's tentative completion time of i is later than C^*_i. Thus, by Claim 3.1 it must be the case that the amount of time during $[r_i, C^*_i]$ which is contained in some interval in $I_{i'}$ for a job i' assigned to machine j by the algorithm whose density is at least $\frac{w^*_i}{\alpha p_{i,j}}$, is at least $\frac{\epsilon/2}{1+\epsilon}(C^*_i - r_i)$. Knowing that this holds for all jobs in M_1 and M_1 is at least half of the total amount of time content in the union of jobs in W^*_j, we have the lemma.

□

Now we are ready to bound $P^*(A^*_{2,j})$, the total profit the adversary obtains for jobs in A_2 processed on machine j by $P(A_{1,j} \cup A_{2,j})$ the profit the algorithm would obtain if it completed all jobs by their tentative completion time which were assigned to machine j.

Lemma 3.5. $P^*(A^*_{2,j}) \leq \frac{\alpha 4(1+\epsilon)}{\epsilon} P(A_{1,j} \cup A_{2,j})$

Proof. Consider the set of jobs in $A^*_{2,j}$. Let $k = |A^*_{2,j}|$ and consider sorting the jobs in $A^*_{2,j}$ from $1, 2, \ldots k$ by their densities in the adversary's schedule $\frac{w^*_i}{p_{i,j}}$ in decreasing order. For ease of notation let there be a job $k+1$ where $\frac{w^*_{k+1}}{p_{k+1,j}} = 0$. By Lemma 3.4 we know that for every u, $L^*_j(u) \leq \frac{4(1+\epsilon)}{\epsilon} L(\frac{u}{\alpha})$. In particular, for $u = \frac{w^*_i}{p_{i,j}}$ for a job $i \in A^*_{2,j}$, it is the case that

$$\sum_{i' \in A^*_{2,j}, i' \leq i} p_{i',j} = L^*_j(\frac{w^*_i}{p_{i,j}}) \leq \frac{4(1+\epsilon)}{\epsilon} L(\frac{\frac{w^*_i}{p_{i,j}}}{\alpha}) = \frac{4(1+\epsilon)}{\epsilon} \sum_{i' \in A_{1,j} \cup A_{2,j}, \frac{w^*_i}{p_{i,j}\alpha} \leq \frac{w_{i'}}{p_{i',j}}} p_{i',j}. \tag{1}$$

Using this, we have the following,

$$\begin{aligned}
P^*(A^*_{2,j}) &= \sum_{i \in A^*_{2,j}} w^*_i \\
&= \sum_{i \in A^*_{2,j}} p_{i,j} \sum_{i' \in A^*_{2,j}, i' \geq i} (\frac{w^*_{i'}}{p_{i',j}} - \frac{w^*_{i'+1}}{p_{i'+1,j}}) \quad \text{[Telescoping summation and } \tfrac{w^*_{k+1}}{p_{k+1,j}} = 0\text{]} \\
&= \sum_{i \in A^*_{2,j}} (\frac{w^*_i}{p_{i,j}} - \frac{w^*_{i+1}}{p_{i+1,j}}) \sum_{i' \in A^*_{2,j}, i' \leq i} p_{i',j} \\
&\leq \frac{4(1+\epsilon)}{\epsilon} \sum_{i \in A^*_{2,j}} (\frac{w^*_i}{p_{i,j}} - \frac{w^*_{i+1}}{p_{i+1,j}}) \sum_{i' \in A_{1,j} \cup A_{2,j}, \frac{w^*_i}{p_{i,j}\alpha} \leq \frac{w_{i'}}{p_{i',j}}} p_{i',j} \quad \text{[Equation (1)]} \\
&= \frac{4(1+\epsilon)}{\epsilon} \sum_{i \in A_{1,j} \cup A_{2,j}} p_{i,j} \sum_{i' \in A^*_{2,j}, \frac{w^*_{i'}}{\alpha p_{i',j}} \leq \frac{w_i}{p_{i,j}}} (\frac{w^*_{i'}}{p_{i,j}} - \frac{w^*_{i'+1}}{p_{i'+1,j}}) \\
&\leq \frac{4(1+\epsilon)}{\epsilon} \sum_{i \in A_{1,j} \cup A_{2,j}} p_{i,j} \alpha \frac{w_i}{p_{i,j}} \\
&\leq \frac{\alpha 4(1+\epsilon)}{\epsilon} \sum_{i \in A_{1,j} \cup A_{2,j}} w_i \\
&= \frac{\alpha 4(1+\epsilon)}{\epsilon} P(A_{1,j} \cup A_{2,j})
\end{aligned}$$

□

Theorem 3.6. *There is a* $(1+\epsilon)$*-speed* $O(\frac{1}{\epsilon^2})$*-competitive algorithm for any* $0 < \epsilon \leq 1$ *on related machine.*

Proof. The total profit obtained by the adversary is $\sum_{j=1}^m P^*(A^*_1) + P^*(A^*_2)$. By definition, we know that $\sum_{j=1}^m P^*(A^*_1)$ is a most the profit the algorithm obtains for jobs in A_1, $\sum_{j=1}^m P(A_{1,j})$. By Lemma 3.5 we know that $P^*(A^*_{2,j}) \leq \frac{\alpha 4(1+\epsilon)}{\epsilon} P(A_{1,j} \cup A_{2,j})$. Thus, $\sum_{j=1}^m P^*(A^*_1) + P^*(A^*_2) \leq (1 + \frac{\alpha 4(1+\epsilon)}{\epsilon}) \sum_{j=1}^m P(A_{1,j} \cup A_{2,j})$.

Now recall that $P(A_{1,j} \cup A_{2,j})$ is the total profit the algorithm would obtain for all jobs assigned to machine j if they were completed by their tentative completion time and C is the set of jobs completed by the algortihm. By Lemma 3.3 we know that $\sum_{j=1}^m P(A_{1,j} \cup A_{2,j}) \leq (\frac{4\epsilon(\alpha-1)-(2+\epsilon)}{4\epsilon(\alpha-1)}) \sum_{i \in C} w_i$. Thus, we have that $\sum_{j=1}^m P^*(A^*_1) + P^*(A^*_2) \leq (1 + \frac{\alpha 4(1+\epsilon)}{\epsilon})(\frac{4\epsilon(\alpha-1)-(2+\epsilon)}{4\epsilon(\alpha-1)}) \sum_{i \in C} w_i$. Setting $\alpha = 1 + \frac{1}{\epsilon}$ completes the proof. □

References

[1] S. Anand, Karl Bringmann, Tobias Friedrich, Naveen Garg, and Amit Kumar. Minimizing maximum (weighted) flow-time on related and unrelated machines. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013*, pages 13–24, 2013.

[2] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Competitive algorithms for due date scheduling. *Algorithmica*, 59(4):569–582, 2011.

[3] Sanjoy K. Baruah, Gilad Koren, Decao Mao, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, Dennis Shasha, and Fuxing Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4(2):125–144, 1992.

[4] Sanjoy K. Baruah, Gilad Koren, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, and Dennis Shasha. On-line scheduling in the presence of overload. In *Symposium on Foundations of Computer Science*, pages 100–110, 1991.

[5] Ho-Leung Chan, Tak Wah Lam, and Kar-Keung To. Non-migratory online deadline scheduling on multiprocessors. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 970–979, 2004.

[6] Bala Kalyanasundaram and Kirk Pruhs. Fault-tolerant real-time scheduling. *Algorithmica*, 28(1):125–144, 2000.

[7] Bala Kalyanasundaram and Kirk Pruhs. Speed is as powerful as clairvoyance. *J. ACM*, 47(4):617–643, 2000.

[8] Bala Kalyanasundaram and Kirk Pruhs. Eliminating migration in multi-processor scheduling. *J. Algorithms*, 38(1):2–24, 2001.

[9] Chiu-Yuen Koo, Tak Wah Lam, Tsuen-Wan Ngan, and Kar-Keung To. Extra processors versus future information in optimal deadline scheduling. *Theory Comput. Syst.*, 37(3):323–341, 2004.

[10] Gilad Koren and Dennis Shasha. MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling. *Theor. Comput. Sci.*, 128(1&2):75–97, 1994.

[11] Gilad Koren and Dennis Shasha. Dover: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM J. Comput.*, 24(2):318–339, 1995.

[12] Kirk Pruhs and Clifford Stein. How to schedule when you have to buy your energy. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 13th International Workshop, APPROX 2010, and 14th International Workshop, RANDOM 2010, Barcelona, Spain, September 1-3, 2010. Proceedings*, pages 352–365, 2010.

[13] Gerhard J. Woeginger. On-line scheduling of jobs with fixed start and end times. *Theor. Comput. Sci.*, 130(1):5–16, 1994.