$O(1/\varepsilon)$ Is the Answer in Online Weighted Throughput Maximization

Franziska Eberle
Technische Universität Berlin, Germany

Abstract

We study a fundamental online scheduling problem where jobs with processing times, weights, and deadlines arrive online over time at their release dates. The task is to preemptively schedule these jobs on a single or multiple (possibly unrelated) machines with the objective to maximize the weighted throughput, the total weight of jobs that complete before their deadline. To overcome known lower bounds for the competitive analysis, we assume that each job arrives with some slack $\varepsilon > 0$; that is, the time window for processing job j on any machine i on which it can be executed has length at least $(1 + \varepsilon)$ times j's processing time on machine i.

Our contribution is a best possible online algorithm for weighted throughput maximization on unrelated machines: Our algorithm is $\mathcal{O}\left(\frac{1}{\varepsilon}\right)$-competitive, which matches the lower bound for unweighted throughput maximization on a single machine. Even for a single machine, it was not known whether the problem with weighted jobs is "harder" than the problem with unweighted jobs. Thus, we answer this question and close weighted throughput maximization on a single machine with a best possible competitive ratio $\Theta\left(\frac{1}{\varepsilon}\right)$.

While we focus on non-migratory schedules, on identical machines, our algorithm achieves the same (up to constants) performance guarantee when compared to an optimal migratory schedule.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Scheduling algorithms

Keywords and phrases Deadline scheduling, weighted throughput, online algorithms, competitive analysis

Digital Object Identifier 10.4230/LIPIcs.STACS.2024.32

Related Version *Full Version*: https://doi.org/10.48550/arXiv.2310.16697

Funding Supported by the Dutch Research Council (NWO), Netherlands Vidi grant 016.Vidi.189.087.

1 Introduction

We consider an online scheduling problem with m parallel *unrelated* machines. Online over time, job j arrives at its *release date* r_j. Upon arrival of job j, its *processing time*, sometimes also referred to as *size*, $p_{ij} \in \mathbb{R}_{>0} \cup \{\infty\}$ on machine $i \in [m] := \{1, \dots, m\}$, its *weight* w_j, and its *deadline* d_j are revealed to the online algorithm. The *density* of j on machine i is given by $\rho_{ij} := \frac{w_j}{p_{ij}}$. A machine i is *eligible* for job j if $p_{ij} < \infty$. If $p_{ij} = p_j$ holds for all i and all j, we call the machines *identical* and omit the index.

The processing of a job is allowed to be interrupted, we say *preempted*, and resumed at any later point in time, also on a different machine; that is, we allow *migration*. A job j completes if $\sum_{i=1}^{m} \frac{q_{ij}}{p_{ij}} = 1$, where j receives a total of q_{ij} time units on machine i. In a feasible schedule, at any point in time, every machine can process at most one job, and every job can be processed by at most one machine. The objective is to find a feasible schedule that maximizes the *weighted throughput*, the total weight of jobs that meet their deadlines.

This model captures a resource allocation problem, e.g., encountered in public cloud computing environments or large internal clusters, where the available hardware needs to be allocated to (often) time-sensitive and mission-critical jobs [18]. Focusing on the objective of maximizing weighted throughput allows us to factor in deadlines of time-sensitive jobs, account for the variety in the importance of jobs, and overall maximize total utility [8]. As explained in [6], allowing preemption in multiprocessor computer systems is cost-wise quite reasonable, while migration, which our algorithm will not use, can be quite expensive due to additional communication requirements.

Due to the lack of information because of the online nature of the problem, there are instances where an online algorithm cannot find the schedule with maximum throughput as shown in Section 3 of [3] for single-machine instances. Thus, we employ standard competitive analysis to measure the performance of an online algorithm, where we compare the weighted throughput of an online algorithm to the weighted throughput of an optimal offline algorithm, that has complete knowledge about the instance in advance and uses this information to make scheduling decisions.

It has been known for 30 years that in this general setting no deterministic algorithm can achieve a bounded competitive ratio on identical machines [4,16]. In fact, even when allowing randomization on a single machine, the competitive ratio for weighted throughput maximization remains unbounded [7]. All of the aforementioned lower bounds heavily rely on "tight" jobs, that is, on jobs that need to be processed immediately and continuously upon release in order to finish before their deadlines.

To overcome these lower bounds, we make a standard *slackness* assumption that the window for processing job j has some slack: we assume that $d_j - r_j \geq (1+\varepsilon)p_{ij}$ holds if machine i is eligible for j, i.e., if $p_{ij} < \infty$. We expect our algorithm to perform better with larger slack parameter $\varepsilon > 0$. This trade-off between slack and competitive ratio has successfully been studied before [1,2,8,10–12,18,21]. The slackness assumption does not pose a real obstacle for applications since deadlines are typically not tight [13] and slack can even be introduced by lowering the operating level or increasing the speed [9,20].

Recently, unweighted throughput maximization has been solved on a single machine with a competitive ratio of $\mathcal{O}\big(\frac{1}{\varepsilon}\big)$ [8], on identical machines with a $\mathcal{O}(1)$-competitive algorithm [19], and on unrelated machines with a competitive ratio of $\mathcal{O}\big(\frac{1}{\varepsilon}\big)$ [10]. For weighted throughput maximization it is known that $\mathcal{O}(1)$-competitive algorithms are not possible, independent of the machine environment, even when allowing randomization [16]. Even on a single machine, there remained a gap between the performance bound $\mathcal{O}\big(\frac{1}{\varepsilon^2}\big)$ of the algorithm by [18] and the lower bound $\Omega\big(\frac{1}{\varepsilon}\big)$ carried over from the unweighted setting [8].

In this work, we close this gap and essentially the line of research concerned with online weighted throughput maximization started in this form by [18]. We give an (up to constant factors) best possible online algorithm for weighted throughput maximization on unrelated machines with competitive ratio $\mathcal{O}\big(\frac{1}{\varepsilon}\big)$. On identical machines, our non-migratory algorithm remains $\mathcal{O}\big(\frac{1}{\varepsilon}\big)$-competitive against the optimal schedule that is allowed to use migration. In particular, we solve the problem on a single machine, matching the known lower bound for unweighted throughput maximization [8].

Related work

Online throughput maximization gained a lot of interest during the last years [8,10,13,19], but research has been active for decades [2–5].

For tight jobs with $d_j - r_j = p_j$, there are non-constant lower bounds on the competitive ratios of deterministic and randomized algorithms [4,7], which justify our slackness assumption. Baruah et al. [4] give a lower bound of $(1+\sqrt{\mu})^2$ on the competitive ratio of any deterministic single-machine algorithm, where $\mu = \frac{\max_j \rho_j}{\min_j \rho_j}$, while Koren and Shasha [17] give an algorithm with matching competitive ratio. On identical machines, their algorithm achieves the best possible competitive ratio of $\Theta(\ln \mu)$ [16]. Canetti and Irani [7] consider randomized algorithms and show a lower bound of $\Theta\big(\frac{\log \nu}{\log \log \nu}\big)$, where $\nu = \min\big\{\frac{\max p_j}{\min p_j}, \frac{\max w_j}{\min w_j}\big\}$. They also give an almost matching upper bound.

Since both parameters, μ and ν, can become arbitrarily large, research started to investigate instances satisfying a slackness assumption [18]. Most relevant to our work is the $\mathcal{O}\big(\frac{1}{\varepsilon^2}\big)$-competitive algorithm by Lucier et al. [18] for weighted throughput maximization on identical machines under the slackness assumption. Azar et al. [1] study the same problem and give a truthful mechanism with a competitive ratio of $2 + \Theta\Big(\frac{1}{\sqrt[3]{1+\varepsilon}-1} + \frac{1}{(\sqrt[3]{1+\varepsilon}-1)^3}\Big)$.

The special case of maximizing machine utilization, where the weight of each job equals its processing time, allows for $\mathcal{O}(1)$-competitive algorithms, even in settings without slack. On a single machine, the algorithm by Baruah et al. [4,5] is 4-competitive, and on identical machines, Koren and Shasha [16] claim an $\mathcal{O}(1)$-competitive algorithm.

In the *unweighted* setting, Baruah et al. [3] show that non-trivial competitive ratios are impossible in the presence of tight jobs. However, randomization allows for a competitive ratio of $\mathcal{O}(1)$ [15]. If every job arrives with a slack of ε, the (deterministic) algorithm by Chen et al. [8] achieves the provably best competitive ratio of $\Theta\big(\frac{1}{\varepsilon}\big)$ on a single machine. On at least two machines, the algorithm by Moseley et al. [19] is $\mathcal{O}(1)$-competitive, even for instances without slack. Eberle et al. [10] design a $\Theta\big(\frac{1}{\varepsilon}\big)$-competitive algorithm for throughput maximization on unrelated machines when each job has ε-slack.

Our results

As our main result, we present an $\mathcal{O}\big(\frac{1}{\varepsilon}\big)$-competitive algorithm for online weighted throughput maximization.

▶ **Theorem 1.** *For weighted throughput maximization on unrelated machines without migration, there is an $\mathcal{O}\big(\frac{1}{\varepsilon}\big)$-competitive online algorithm.*

This generalizes and improves the $\mathcal{O}\big(\frac{1}{\varepsilon^2}\big)$-competitive algorithm by [18]. It matches the known lower bound of $\Omega\big(\frac{1}{\varepsilon}\big)$ on a single machine [8] and, thus, closes the gap that remained.

During the analysis, we focus on comparing the non-migratory schedule obtained by our algorithm to an optimal, non-migratory schedule. On identical machines, it is known that the throughput achievable without migration is within a constant multiplicative factor of the throughput achievable using migration by Kalyanasundaram and Pruhs [14]. Thus, our result also holds in the migratory setting.

▶ **Theorem 2.** *For weighted throughput maximization on identical machines with migration, there is an $\mathcal{O}\big(\frac{1}{\varepsilon}\big)$-competitive online algorithm.*

One threshold cannot beat $\Theta(1/\varepsilon^2)$

Previous results for throughput maximization use a threshold-based policy to decide about the admission of newly released and the preemption of currently running jobs [1,8,10,18]. Crucially, these algorithms rely on a *single* density threshold $\gamma \in \Theta(\varepsilon)$ to determine if a currently running job is preempted in favor of a newly released job with higher density.

The following two examples give an intuition why a single-threshold algorithm cannot break the $\mathcal{O}\big(\frac{1}{\varepsilon^2}\big)$-barrier. Let $\varepsilon < 1$ and suppose that $\gamma \in (0,1]$ is the threshold which an algorithm uses to discard currently running jobs in favor of newly released jobs with density higher by a factor at least $\frac{1}{\gamma}$. In both examples, $\delta \ll 1$ is a small constant, for which we will eventually consider the limit $\delta \to 0$, and the jobs are *tight*, i.e., they satisfy $r_j + (1+\varepsilon)p_j = d_j$.

▶ **Example 3.** There is a single machine and $n+1$ tight jobs with the parameters $r_0 = 0$, $p_0 = w_0 = \rho_0 = 1$ and $r_j = r_{j-1} + (1-\delta)p_{j-1}$, $p_j = (\varepsilon + \delta)p_{j-1}$ and $\rho_j = \frac{1+\delta}{\gamma}\rho_j$ for $j \in [n] := \{1, \ldots, n\}$.

By our choice of parameters, job j interrupts the execution of job $j-1$ immediately upon arrival. It is easy to calculate that $d_j \geq d_{j-1}$ and $C_j = d_{j-1}$ for $j \in [n]$ if the processing of job j is not interrupted. Combining these two observations implies that the algorithm can only complete the last job on time if the constant δ is chosen sufficiently small. Hence, the algorithm obtains a total weight of $\rho_n p_n = \left(\frac{(1+\delta)(\varepsilon+\delta)}{\gamma}\right)^n \to \left(\frac{\varepsilon}{\gamma}\right)^n$ as $\delta \to 0$. Conversely, by scheduling only job 1, one can obtain a total weight of 1, implying that the competitive ratio is at least $\left(\frac{\gamma}{\varepsilon}\right)^n$. Hence, $\gamma \leq \varepsilon$ is necessary to achieve a bounded competitive ratio.

▶ **Example 4.** We have a single-machine instance with two tight jobs 1 and 2. The parameters are $p_1 = w_1 = 2$, $r_1 = 0$ and $p_2 = \frac{1}{\varepsilon}$, $w_2 = \frac{1}{\varepsilon\gamma - \delta}$, $r_2 = \delta$. Since both jobs are tight, no feasible schedule can complete both jobs on time. Hence, it is optimal to schedule only the second job upon release and obtain a total weight of $\frac{1}{\varepsilon\gamma-\delta}$. However, the parameters are chosen such that an algorithm with threshold γ admits the first job upon release and cannot discard it in favor of the second job, implying a competitive ratio of $\Omega\left(\frac{1}{\varepsilon\gamma-\delta}\right)$, which goes to $\Omega\left(\frac{1}{\gamma\varepsilon}\right)$ as $\delta \to 0$.

What becomes apparent in the examples is that by relying on a single threshold to guide the admission decisions, an algorithm is both too careless (Example 3) and too conservative (Example 4) in admitting jobs. In fact, such an algorithm does not distinguish the reasons for a job having a relatively high density: it might be caused by a large weight or by a small processing time.

We show that, by using *two* distinct thresholds, a simple greedy algorithm achieves a competitive ratio of $\Theta\left(\frac{1}{\varepsilon}\right)$, which is optimal up to constants even in the unweighted setting [8]. Our algorithm compares the sizes of a newly released job $j^\star$ and a currently running job j, in order to decide whether to abandon the latter in favor of the former. In Example 3, we have already established that if j is preempted in favor of $j^\star$ with $p_{ij^\star} \in \mathcal{O}(\varepsilon)p_{ij}$, then the density of $j^\star$ should be greater by a factor $\Omega\left(\frac{1}{\varepsilon}\right)$. Conversely, to avoid the issue present in Example 4, if the new job $j^\star$ is larger than the currently running job j, then j should be interrupted in favor of processing $j^\star$ if it has a similar density as j. For technical reasons, our algorithm employs a third admission rule that smoothly interpolates between the threshold $\Theta(\varepsilon)$ for jobs smaller by a factor $\mathcal{O}(\varepsilon)$ and a threshold $\Theta(1)$ for larger jobs.

In the following section, we formally describe our algorithm before analyzing its competitive ratio in the two subsequent sections.

2 The two-threshold algorithm

In this section, we design the two-threshold algorithm. We assume without loss of generality that $\varepsilon \leq 1$ as otherwise we can simply run the algorithm with $\varepsilon = 1$ and obtain a constant competitive ratio.

The two-threshold algorithm starts job j on machine i for the first time only before $d_j - \left(1 + \frac{\varepsilon}{2}\right)p_{ij}$. If machine i starts processing job j at time a_j, we say j is *admitted* to machine i at time a_j. For each machine i, the algorithm maintains the set of jobs that are active at time τ. A job j is *active* at time τ on machine i if it was admitted to i before time τ, is not yet completed and can still complete before time $a_j + \left(1 + \frac{\varepsilon}{2}\right)p_{ij}$, i.e., the remaining processing time of j on i is at most $a_j + \left(1 + \frac{\varepsilon}{2}\right)p_{ij} - \tau$.

On a high-level, our algorithm uses two independent subroutines: the scheduling routine and the admission routine. The admission routine merely assigns jobs to machines. Among the jobs assigned to a machine, the scheduling routine chooses which job to actually process.

Scheduling routine. At time τ and on each machine i, the algorithm simply processes the job j which is active for i at τ and has the highest density $\rho_{ij} = \frac{w_j}{p_{ij}}$ among all such jobs.

Admission routine. There are two events that trigger the admission routine at time τ: the release of a new job and the completion of a job. The admission routine loops over the machines and decides whether the currently running job j should be preempted for a job with higher density.

To this end, it considers the jobs that have been released, have not yet been admitted, and whose deadline is sufficiently far in the future, i.e., $d_{j^\star} - \tau \geq \left(1 + \frac{\varepsilon}{2}\right) p_{ij^\star}$, in decreasing order of machine-dependent density $\rho_{ij^\star}$. Let $j^\star$ be the job with highest density that has not been considered for admission to machine i before. The algorithm compares $j^\star$'s processing time with that of the job j that is currently processed by machine i.

If no such job exists, then $j^\star$ is admitted to machine i and starts executing immediately. If such a job j exists and its processing time is at least $\frac{2}{\varepsilon}p_{ij^\star}$, then the first density-threshold $\frac{8}{\varepsilon}$ is invoked: if $\rho_{ij^\star} \geq \frac{8}{\varepsilon}\rho_{ij}$, then $j^\star$ is admitted to i at time $a_{j^\star} = \tau$. If j exists and $\frac{\varepsilon}{2}p_{ij} < p_{ij^\star} \leq p_{ij}$, then we use a smooth transition between the two thresholds: if $w_{j^\star} \geq 4w_j$, then $j^\star$ is admitted to i at time $a_{j^\star} = \tau$. Otherwise, that is, j is currently running on machine i and its processing time is smaller than $p_{ij^\star}$, then the second density-threshold 4 is invoked: if $\rho_{ij^\star} \geq 4\rho_{ij}$, then $j^\star$ is admitted to i at time $a_{j^\star} = \tau$.

If job $j^\star$ interrupts the execution of job j, we say that j is the *parent* $\pi(j^\star)$ of $j^\star$ and $j^\star$ is a *child* of j. Note that by construction, a newly admitted job has highest density on its machine and starts processing immediately. We summarize our algorithm in Algorithm 1.

The following observation formalizes the "smooth transition" between the two density thresholds.

▶ **Observation 5.** *Consider jobs j and k with $\frac{\varepsilon}{2}p_{ij} < p_{ik} \leq p_{ij}$ for some machine i. If $w_k \geq 4w_j$, then $\rho_{ik} = \frac{w_k}{p_{ik}} \geq \frac{4w_j}{p_{ij}} = 4\rho_{ij}$. If $w_k < 4w_j$, then $\rho_{ik} < \frac{4w_j}{\varepsilon/2p_{ij}} = \frac{8}{\varepsilon}\rho_{ij}$. Further, if $p_{ik} > p_{ij}$ and $\rho_{ik} \geq 4\rho_{ij}$, then $w_k = \rho_{ik}p_{ik} \geq 4w_j$.*

Roadmap of the analysis

The analysis of the two-threshold algorithm naturally splits into two parts. In Section 3, we show that the highest-density rule used for scheduling active jobs guarantees that the total weight of jobs completed before their deadlines is at least half of the total weight of jobs admitted by the admission routine. In Section 4, we compare the total weight of the jobs admitted by the two-threshold algorithm to the weighted throughput of an optimal solution before proving Theorem 1.

3 Weight of finished jobs vs. weight of admitted jobs

In this section, we show that the two-threshold algorithm obtains at least half of the total weight of the jobs that were admitted. We prove the following theorem where J denotes the set of jobs admitted by our algorithm and $F \subseteq J$ the set of jobs completed before their respective deadlines.

Algorithm 1 Two-threshold algorithm.

```
Initialize: If $\varepsilon > 1$, then reset $\varepsilon \leftarrow 1$.
Scheduling Routine: At all times $\tau$ and on all machines $i$, run the job with highest
  density that is active for $i$.
Event: Release of a new job at time $\tau$
    Call Admission Routine
Event: Completion of a job at time $\tau$
    Call Admission Routine

Admission Routine:
for $i = 1$ to $m$ do
    $J^\star \leftarrow \{j \,|\, r_j \le \tau, d_j - \tau \ge \left(1+\frac{\varepsilon}{2}\right) p_{ij}, j \text{ not yet admitted}\}$   // eligible jobs
    $K \leftarrow \{k : k \text{ active on machine } i \text{ at time } \tau\}$   // jobs currently active for $i$
    for $j^\star \in J^\star$ in order of decreasing $\rho_{ij^\star}$ do   // select highest-density job
        if $K = \emptyset$ then
            admit $j^\star$ to $i$ and $a_{j^\star} \leftarrow \tau$
            $\pi(j^\star) \leftarrow \emptyset$   // $j^\star$ does not have a parent
            break for-loop
        else
            $j \leftarrow \arg\max\{\rho_{ik} \,|\, k \in K\}$   // currently running job
            if $p_{ij^\star} \le \frac{\varepsilon}{2} p_{ij}$ and $\rho_{ij^\star} \ge \frac{8}{\varepsilon}\rho_{ij}$ then
                admit $j^\star$ to $i$ and $a_{j^\star} \leftarrow \tau$
                $\pi(j^\star) \leftarrow j$   // parent of $j^\star$
                break for-loop
            else if $\frac{\varepsilon}{2} p_{ij} < p_{ij^\star} \le p_{ij}$ and $w_{j^\star} \ge 4 w_j$ then
                admit $j^\star$ to $i$ and $a_{j^\star} \leftarrow \tau$
                $\pi(j^\star) \leftarrow j$   // parent of $j^\star$
                break for-loop
            else if $p_{ij^\star} > p_{ij}$ and $\rho_{ij^\star} \ge 4\rho_{ij}$ then
                admit $j^\star$ to $i$ and $a_{j^\star} \leftarrow \tau$
                $\pi(j^\star) \leftarrow j$   // parent of $j^\star$
                break for-loop
```

▶ **Theorem 6.** *Let J and F be the set of jobs admitted and finished, respectively, by the two-threshold algorithm. Then,*

$$\sum_{j \in F} w_j \ge \frac{1}{2} \sum_{j \in J} w_j.$$

For intuition, consider an instance that only consists of a job j and the set K of j's children. Suppose that j does not finish on time as otherwise the theorem trivially holds. Recall that j was admitted at $a_j \le d_j - \left(1+\frac{\varepsilon}{2}\right) p_{ij}$ to machine i. (Jobs that are not interrupted complete before $a_j + \left(1+\frac{\varepsilon}{2}\right) p_{ij} \le d_j$.) This implies that the total processing time of jobs interrupting j is at least $\frac{\varepsilon}{2} p_{ij}$. If there is at least one job k with $p_{ik} > \frac{\varepsilon}{2} p_{ij}$, Observation 5 and the admission rule for jobs with $\frac{\varepsilon}{2} p_{ij} < p_{ik} \le p_{ij}$ imply that $w_k \ge 4 w_j$ showing the statement. If all jobs k have processing time at most $\frac{\varepsilon}{2} p_{ij}$, their densities are bounded from below by $\frac{8}{\varepsilon}\rho_{ij}$, and their total weight is again at least $4 w_j$.

In the formal proof of Theorem 6, we assume the existence of an instance that violates the statement and restrict to one that is minimal with respect to the total number of jobs. The above intuition tells us that sub-instances consisting of a job and its children cannot cause the violation. In fact, we show that we can carefully merge such sub-instances into one job without changing the fact that the complete instance violates the theorem statement, which contradicts the minimality of the original instance.

Proof of Theorem 6. Let $U = J \setminus F$ be the set of jobs admitted by the two-threshold algorithm that were discarded, i.e., that did not complete by time $a_j + \left(1 + \frac{\varepsilon}{2}\right) p_{ij}$. In order to show the theorem, it suffices to prove $\sum_{j \in F} w_j \geq \sum_{j \in U} w_j$.

For the sake of contradiction, we assume that there is an instance with $\sum_{j \in F} w_j < \sum_{j \in U} w_j$. Among all such instances, we consider an instance with the smallest number of jobs. In particular, this implies that there are no jobs in the instance that were not admitted by the algorithm and there is only one machine in the instance. We show that there is another instance with strictly fewer jobs that still satisfies the above inequality, contradicting the choice of the instance.

Without loss of generality, for all jobs j, we can assume that $r_j = a_j$ and $d_j = r_j + (1+\varepsilon)p_j$ holds. Indeed, since the first assumption does not change the availability of a job j at time a_j or the density, the two-threshold algorithm still admits j at time a_j. Further, as the algorithm discards jobs when they cannot be completed by time $a_j + \left(1 + \frac{\varepsilon}{2}\right) p_j < d_j$, the second assumption does not change whether a job is completed on time by the algorithm.

Observe that a job that is not interrupted completes on time. Hence, the assumption $\sum_{j \in F} w_j < \sum_{j \in U} w_j$ implies that there are jobs whose processing is interrupted. Fix a job j that is preempted but whose children's processing is not interrupted. Let K be the set of children of j, and let $\pi = \pi(j)$ if it exists. Let $C'_{j'}$ be the last point in time that the two-threshold algorithm works on job j', which is either the completion time of j' or the point when j' was discarded because of jobs with higher densities. Denote by $C' := \max\{\max_{k \in K} C'_k, C'_j\}$, the last point in time when j or one of its children were processed. Observe that during $[a_j, C')$ only j and j's children are processed.

Our goal is to create a new instance where j and its children are replaced by a new job $j^\star$. Let F' and U' denote the finished and unfinished jobs, respectively, after the replacement. We will show that the new instance satisfies the following properties:

(i) Job $j^\star$ is admitted at a_j and completes at time C'.
(ii) $\sum_{j' \in F'} w_{j'} < \sum_{j' \in U'} w_{j'}$.
(iii) There are strictly fewer jobs.

By assumption j has at least one child. Hence, property (iii) follows trivially from our replacement. We do not make any changes to a job $j' \notin K \cup \{j\}$. Thus, property (i) and the assumptions on the instance imply that our changes will not influence whether such a job j' is discarded or completed by the algorithm.

We set $p_{j^\star} = \bar{p}_j + \sum_{k \in K} p_k$, where $\bar{p}_j \leq p_j$ is the actual amount that the two-threshold algorithm processed j in the original instance. For ensuring that $j^\star$ is available at a_j, we set $r_{j^\star} = a_j$ and $d_{j^\star} = r_{j^\star} + (1+\varepsilon)p_{j^\star}$. This choice of parameters implies that, if $j^\star$ is admitted at time $r_{j^\star}$, it will complete at time

$$r_{j^\star} + p_{j^\star} = a_j + \bar{p}_j + \sum_{k \in K} p_k = C' < d_{j^\star}$$

since no other job is released during the interval $[a_j, C')$ in the new instance. Thus, in order to show property (i), it suffices to show that $j^\star$ interrupts job π at $r_{j^\star}$. Recall that the two-threshold algorithm compares $p_{j^\star}$ with p_π in order to decide upon admission of $j^\star$. There are three possibilities: $p_{j^\star} \le \frac{\varepsilon}{2}p_\pi$, $p_{j^\star} \in \left(\frac{\varepsilon}{2}p_\pi, p_\pi\right]$, and $p_{j^\star} > p_\pi$. Depending on the interval, different admission rules apply.

For defining the weight of job $j^\star$, we distinguish two cases based on job j.

Case I. If j completes on time, set $w_{j^\star} = w_j + \sum_{k\in K} w_k \ge w_j$. We observe that $\sum_{k\in K} p_k \le \frac{\varepsilon}{2}p_j$ as j completes on time. Further,

$$\rho_{j^\star} = \frac{w_j + \sum_{k\in K} w_k}{p_j + \sum_{k\in K} p_k} \ge \frac{\rho_j p_j + \frac{8}{\varepsilon}\rho_j \sum_{k\in K} p_k}{p_j + \sum_{k\in K} p_k} \ge \rho_j .$$

Thus, if $p_{j^\star}$ and p_j belong to the same interval with respect to p_π, $j^\star$ is admitted upon release and property (i) is satisfied. If they belong to different intervals, we note that

$$\left(1 + \frac{\varepsilon}{2}\right) p_j \ge p_{j^\star} = p_j + \sum_{k\in K} p_k \ge p_j \tag{1}$$

and distinguish two cases.

- $\boldsymbol{p_j \le \frac{\varepsilon}{2}p_\pi < p_{j^\star}}$ **:** We note that (1) and $\varepsilon \le 1$ imply $p_{j^\star} \le p_\pi$. Thus,

  $$w_{j^\star} = \rho_{j^\star} p_{j^\star} \ge \rho_j \frac{\varepsilon}{2} p_\pi \ge \frac{8}{\varepsilon}\rho_\pi \frac{\varepsilon}{2} p_\pi = 4 w_\pi ,$$

  which guarantees property (i).
- $\boldsymbol{p_j \le p_\pi < p_{j^\star}}$ **:** We have $p_{j^\star} = (1+\delta)p_j \le (1+\delta)p_\pi$ for some $\delta \in \left(0, \frac{\varepsilon}{2}\right]$. Further, $w_{j^\star} \ge w_j + \delta\frac{8}{\varepsilon}w_j \ge 4(1+\delta)w_\pi$. Thus, $\rho_{j^\star} = \frac{w_{j^\star}}{p_{j^\star}} \ge \frac{4(1+\delta)w_\pi}{(1+\delta)p_\pi} \ge 4\rho_\pi$, and property (i) holds.

Hence, the total weight of the jobs completed by the two-threshold algorithm and the total weight of the discarded jobs does not change in all cases, which implies property (ii).

Case II. If j does not complete on time, we set $w_{j^\star} = \frac{3}{4}\sum_{k\in K} w_k$. If some $k^\star \in K$ satisfies $p_{k^\star} > \frac{\varepsilon}{2}p_j$, then $\sum_{k\in K} w_k \ge w_{k^\star} \ge 4w_j$ by definition of the two-threshold algorithm. Using that j does not finish on time, we know that $\sum_{k\in K} p_k > \frac{\varepsilon}{2}p_j$. Thus, if the processing times for all $k \in K$ are bounded from above by $\frac{\varepsilon}{2}p_j$, then $\sum_{k\in K} w_k \ge \frac{8}{\varepsilon}\rho_j \sum_{k\in K} p_k \ge 4w_j$.

For property (i), we start by bounding $w_{j^\star}$ and $\rho_{j^\star}$. Using the observation above, $w_{j^\star} = \frac{3}{4}\sum_{k\in K} w_k \ge 3w_j$. By Observation 5, $k \in K$ with $p_k \in \left(\frac{\varepsilon}{2}p_j, p_j\right]$ satisfy $\rho_k \ge 4\rho_j$. Hence, $\sum_{k\in K} w_k = \sum_{k\in K} \rho_k p_k \ge 4\rho_j \sum_{k\in K} p_k$. Using again that $\sum_{k\in K} w_k \ge 3w_j$, we have

$$\rho_{j^\star} = \frac{w_{j^\star}}{p_{j^\star}} \ge \frac{1/2\sum_{k\in K} w_k + w_j}{\bar{p_j} + \sum_{k\in K} p_k} \ge \frac{2\rho_j \sum_{k\in K} p_k + \rho_j p_j}{\sum_{k\in K} p_k + \bar{p_j}} \ge \rho_j .$$

As before, if p_j and $p_{j^\star}$ are in the same interval with respect to p_π, these observations guarantee that $j^\star$ interrupts π at a_j, which implies property (i). If they belong to different intervals, we distinguish five cases.

- $\boldsymbol{p_j \le \frac{\varepsilon}{2}p_\pi < p_{j^\star} \le p_\pi}$ **:** We have $w_{j^\star} = \rho_{j^\star} p_{j^\star} \ge \rho_j \frac{\varepsilon}{2} p_\pi \ge \frac{8}{\varepsilon}\rho_\pi \frac{\varepsilon}{2} p_\pi = 4w_\pi$.
- $\boldsymbol{p_j \le \frac{\varepsilon}{2}p_\pi < p_\pi < p_{j^\star}}$ **:** We have $\rho_{j^\star} \ge \rho_j > 4\rho_\pi$.
- $\boldsymbol{\frac{\varepsilon}{2}p_\pi < p_j \le p_\pi < p_{j^\star}}$ **:** We know that $\rho_{j^\star} \ge \rho_j = \frac{w_j}{p_j} \ge \frac{4w_\pi}{p_\pi} = 4\rho_\pi$.
- $\boldsymbol{p_{j^\star} \le \frac{\varepsilon}{2}p_\pi < p_j}$ **:** We note that $p_{j^\star} \ge \sum_{k\in K} p_k > \frac{\varepsilon}{2}p_j$, which implies $p_j \le p_\pi$. We have $\rho_{j^\star} = \frac{w_{j^\star}}{p_{j^\star}} \ge \frac{3w_j}{\varepsilon/2p_\pi} \ge \frac{12w_\pi}{\varepsilon/2p_\pi} = \frac{24}{\varepsilon}\rho_\pi$.
- $\boldsymbol{\frac{\varepsilon}{2}p_\pi < p_{j^\star} \le p_\pi < p_j}$ **:** We have $w_{j^\star} \ge 3w_j = 3\rho_j p_j > 3 \cdot 4\rho_\pi p_\pi = 12w_\pi$.

Hence, in all cases, $j^\star$ satisfies the conditions of the two-threshold algorithm to interrupt the processing of π at $r_{j^\star}$.

Recall that $\sum_{k\in K} w_k \geq 4w_j$. After replacing $K \cup \{j\}$ with $j^\star$, it holds that

$$\sum_{j'\in F'} w_{j'} = \sum_{j'\in F} w_{j'} - \frac{1}{4}\sum_{k\in K} w_k < \sum_{j'\in F} w_{j'} - w_j < \sum_{j'\in U} w_{j'} - w_j = \sum_{j'\in U'} w_{j'},$$

which implies property (ii).

As argued above, this contradicts the choice of the instance, which concludes the proof. ◀

4 Weight of admitted jobs vs. weight of Opt

In this section, we show that the total weight of jobs finished by an optimal solution is up to a factor $\mathcal{O}\big(\frac{1}{\varepsilon}\big)$ comparable to the total weight of jobs admitted by the two-threshold algorithm:

▶ **Theorem 7.** *Let* Opt *and J be the set of jobs admitted by an optimal, non-migratory solution and the two-threshold algorithm, respectively. Then, $\sum_{x\in \text{OPT}} w_x \leq \mathcal{O}\big(\frac{1}{\varepsilon}\big)\sum_{j\in J} w_j$.*

For proving this statement, it is sufficient to focus on X, the set of jobs scheduled by Opt that the two-threshold algorithm did not admit since $\text{OPT} \subseteq X \cup J$.

Fix a job $x \in X$ that Opt schedules on machine i. The two-threshold algorithm admits a job j during the interval $[r_j, d_j - \big(1+\frac{\varepsilon}{2}\big)p_{ij})$ if it is sufficiently dense. Since x is not admitted by our algorithm, the algorithm is processing jobs J_x on machine i during the interval $[r_x, d_x - \big(1+\frac{\varepsilon}{2}\big)p_{ix})$ with densities that are large and prevent interruption by x. That is, for jobs $j \in J_x$ with $\frac{\varepsilon}{2}p_{ij} \geq p_{ix}$ it holds that $\rho_{ij} > \frac{\varepsilon}{8}\rho_{ix}$, for jobs $j \in J_x$ with $p_{ix} \in \big(\frac{\varepsilon}{2}p_{ij}, p_{ij}\big]$ Observation 5 implies $\rho_{ij} > \frac{\varepsilon}{8}\rho_{ix}$ and for jobs $p_{ij} < p_{ix}$ it holds that $\rho_{ij} > \frac{1}{4}\rho_{ix}$. We say that the jobs J_x *block* the admission of x. We will charge the weight w_x to the weight of the jobs in J_x. Exploiting the two thresholds which the algorithm uses to make admission decisions, we show that the algorithm "obtains" a weight from partially finished jobs of at least $\Omega(\varepsilon)w_x$ in the interval $[r_x, d_x)$.

Proof idea. We give an intuition by considering a single-machine instance where the two-threshold algorithm admits exactly one job j. Consider the jobs in X whose admission was blocked by j: We know that the interval $[r_x, d_x - \big(1+\frac{\varepsilon}{2}\big)p_x)$ is completely covered by the processing of job j, i.e., by the interval $[a_j, C_j)$.

Now consider a job x with $p_x \leq p_j$. Thus, the deadline of x is at most $C_j + \big(1+\frac{\varepsilon}{2}\big)p_x \leq a_j + \big(2+\frac{\varepsilon}{2}\big)p_j$. This implies that Opt can schedule such jobs only during $\big[a_j, a_j + \big(2+\frac{\varepsilon}{2}\big)p_j\big)$, an interval of length $\big(2+\frac{\varepsilon}{2}\big)p_j$. The admission rule for the case $p_x \leq \frac{\varepsilon}{2}p_j$ and Observation 5 for $p_x \in \big(\frac{\varepsilon}{2}p_j, p_j\big]$ imply $\rho_x < \frac{8}{\varepsilon}\rho_j$. Hence,

$$\sum_{\substack{x\in X\\ p_x\leq p_j}} w_x = \sum_{\substack{x\in X\\ p_x\leq p_j}} \rho_x p_x \leq \frac{8}{\varepsilon}\rho_j \sum_{\substack{x\in X\\ p_x\leq p_j}} p_x \leq \frac{8}{\varepsilon}\rho_j\left(2+\frac{\varepsilon}{2}\right)p_j = \left(\frac{16}{\varepsilon}+4\right)w_j.$$

For a job x with $p_x > p_j$, the slacknes assumption guarantees that $r_x \leq d_x - (1+\varepsilon)p_x$. Further, the interval $\big[r_x, d_x - \big(1+\frac{\varepsilon}{2}\big)p_x\big)$ is contained in $[a_j, C_j)$. This allows us to upper bound the processing time p_x by $\frac{2}{\varepsilon}p_j$. Thus, Opt can schedule such jobs only during $\big[a_j, a_j + \big(1+\frac{2}{\varepsilon}\big)p_j\big)$. Using the admission rule of our algorithm in this case gives

$$\sum_{\substack{x\in X\\ p_x> p_j}} w_x = \sum_{\substack{x\in X\\ p_x> p_j}} \rho_x p_x < 4\rho_j \sum_{\substack{x\in X\\ p_x> p_j}} p_x < 4\rho_j\left(1+\frac{2}{\varepsilon}\right)p_j = \left(4+\frac{8}{\varepsilon}\right)w_j.$$

Combining the above two calculations yields that $\sum_{x\in X} w_x \in \mathcal{O}\big(\frac{1}{\varepsilon}w_j\big)$.

Proof outline. In this particular instance, each job $x \in X$ is blocked by a job with either larger or smaller processing time. In general, this is not necessarily true. Hence, in order to extend this idea to arbitrary instances, we partition the jobs in X according to whether at least half of their *availability interval* $\left[r_x, d_x - \left(1+\frac{\varepsilon}{2}\right)p_x\right)$ is covered by jobs in J with smaller or larger processing times. We then show that by losing an additional factor 3, we can assume that only one type covers the availability interval of each job in X. This is done in Lemma 10.

An additional technical challenge poses the fact that, even after we assume that a job is blocked by either shorter or longer jobs, the densities of these jobs are still not uniform enough to directly generalize the above idea to arbitrary instances. In Lemma 8, we show that, at the loss of an additional factor 4, we can partition the jobs in X according to their densities and bound the weight for each density level separately. We then upper bound OPT's available time for scheduling jobs of a certain density level in Lemmas 13 and 14 depending on the size of the blocking jobs.

Proof of Theorem 7. Since OPT and the two-threshold algorithm are non-migratory, we fix one particular machine i and only consider jobs that either OPT or the two-threshold algorithm scheduled on machine i. For simplicity, we drop the index i for the remainder.

In order to partition jobs according to their densities, we geometrically partition the range of potential job densities, $(0, \max_j \rho_j]$, into intervals of the form $(2^{\ell-1}, 2^{\ell}]$ and call $\ell \in \mathbb{Z}$ a *density level.* We say that a job $j \in J$ has *density level* ℓ if $2^{\ell-1} < \rho_j \leq 2^{\ell}$. (For jobs $x \in X$ we are interested in the density levels of the jobs that block them, which we define later.)

For a job $j \in J$ with density level $\lceil \log_2 \rho_j \rceil$, we first argue that we can separately charge j at the levels $\ell \leq \lceil \rho_j \rceil$ at the loss of a constant factor, formalized in the next lemma. This allows us to focus on one density level ℓ at a time.

▶ **Lemma 8.** *Suppose there is a scheme that charges job $j \in J$ a weight of at most $2^{\ell} c p_j$ at level $\ell \leq \lceil \log_2 \rho_j \rceil$ and no weight at level $\ell > \lceil \log_2 \rho_j \rceil$, where $c > 0$ is a constant. Then, the total weight charged to j is at most $4cw_j$.*

Proof. The total weight charged to j is upper bounded by

$$\sum_{-\infty}^{\lceil \log_2 \rho_j \rceil} 2^{\ell} c p_j = c \cdot p_j \left(1 + 2^{\lceil \log_2 \rho_j \rceil + 1} - 1\right) \leq 4c\rho_j p_j = 4cw_j,$$

as desired. ◀

Having this lemma at hand, we now restrict to one density level $\ell \in \mathbb{Z}$ and define $J_\ell := \{j \in J : \rho_j \geq 2^{\ell}\}$. Next, we remove the technical challenge that a job $x \in X$ can be blocked by jobs with smaller and larger processing times. To this end, we carefully modify the intervals where jobs in J_ℓ are scheduled such that the availability interval of a job $x \in X$ blocked by jobs in J_ℓ is completely contained in the modified intervals. To this end, we fix a level ℓ and let $\mathcal{S}_\ell$ denote the set of processing intervals of the jobs in J_ℓ, that is, the intervals during which jobs in J_ℓ are processed.

The modification works as follows: We copy each interval in $I \in \mathcal{S}_\ell$ twice and call one copy the *early* and the other the *late* copy. For each original interval $I = [\alpha, \omega)$, we move the early copy earlier such that it ends at α and we move the late copy later such that it begins at ω. If a copy intersects with another original (even if only partially), by potentially splitting the copy, we shift the part that intersects further into the indicated direction; that is, for the early copy, we move the part earlier and for the late copy, we move the part later. We treat the time points where multiple copies overlap similarly. More precisely, if the interval $[t, t')$ is currently contained in k different copies, we use a $\frac{1}{k}$-fraction from every copy to cover $[t, t')$ and send the remaining $\frac{k-1}{k}$-fraction into the directions indicated by their name.

We denote the resulting set of intervals (including the original ones) by $\mathcal{I}_\ell$. Next, we prove some structural properties about $\mathcal{I}_\ell$ and, for each job $x \in X$, relate its availability interval $\left[r_x, d_x - \left(1 + \frac{\varepsilon}{2}\right) p_x\right)$ to $\mathcal{I}_{\ell_x}$ for some carefully chosen $\ell_x \in \mathbb{Z}$.

▶ **Observation 9.** *Let $\mathcal{S}$ and $\mathcal{T}$ be two sets of intervals such that for each $S \in \mathcal{S}$ there is a $T \in \mathcal{T}$ with $S \subseteq T$. Then, the result of the modification of $\mathcal{T}$ covers all time points covered by the result of the modification of $\mathcal{S}$.*

▶ **Lemma 10.** *For each job $x \in X$, let $\ell_{1x} = \frac{1}{4} \cdot 2^{\lfloor \log_2 \rho_x \rfloor}$ and $\ell_{2x} = \frac{\varepsilon}{8} \cdot 2^{\lfloor \log_2 \rho_x \rfloor}$. Then, $\left[r_x, d_x - \left(1 + \frac{\varepsilon}{2}\right) p_x\right) \subseteq \bigcup_{I \in \mathcal{I}_{\ell_{1x}}} I$ or $\left[r_x, d_x - \left(1 + \frac{\varepsilon}{2}\right) p_x\right) \subseteq \bigcup_{I \in \mathcal{I}_{\ell_{2x}}} I$.*

Proof. Fix a job $x \in X$ and the two levels ℓ_{1x} and ℓ_{2x} from the lemma statement. By assumption, x is blocked by jobs in J at all times in $\left[r_x, d_x - \left(1 + \frac{\varepsilon}{2}\right) p_x\right)$.

Assume that x is blocked for at least half of the time by jobs j with $p_j < p_x$. By definition of Algorithm 1, this implies that $4\rho_j > \rho_x$ holds for these jobs j. Hence, $j \in J_{\ell_{1x}}$. We will show that in this case ℓ_{1x} satisfies the lemma; for simplicity, set $\ell = \ell_{1x}$.

Conversely, suppose that x is blocked for at least half of the time by jobs j with $p_x \le p_j$. Observation 5 for $\frac{\varepsilon}{2} p_j < p_x \le p_j$ and the admission threshold for $\frac{\varepsilon}{2} p_j \ge p_x$ guarantee $\rho_x < \frac{8}{\varepsilon} \rho_j$. Hence, $j \in J_{\ell_{2x}}$ holds if $p_x \le p_j$ and j blocks x. For simplicity, set $\ell = \ell_{2x}$ in this case.

Using Observation 9, it suffices to focus on the set $\mathcal{S}$ of intervals that actually cover the interval $\left[r_x, d_x - \left(1 + \frac{\varepsilon}{2}\right) p_x\right)$ and correspond to scheduling times of jobs that block x at level ℓ. By truncating, we assume that the earliest interval in $\mathcal{S}$ starts not earlier than r_x and that the latest interval ends no later than $d_x - \left(1 + \frac{\varepsilon}{2}\right) p_x$. We index the intervals in $\mathcal{S}$ by starting point and let $K = |\mathcal{S}|$. Denote by α_k and ω_k the start and end point, respectively, of the kth interval.

By assumption, $\mathcal{S}$ covers at least half of $\left[r_x, d_x - \left(1 + \frac{\varepsilon}{2}\right) p_x\right)$. Thus,

$$\omega_K - \alpha_1 \le d_x - \left(1 + \frac{\varepsilon}{2}\right) p_x - r_x \le 2 \sum_{k=1}^{K} (\omega_k - \alpha_k).$$

This implies that $\mathcal{S}$ and its copies cover the intervals $[\alpha_1, \alpha_1 + 2\sum_{k=1}^{K}(\omega_k - \alpha_k))$ and $[\omega_K - 2\sum_{k=1}^{K}(\omega_k - \alpha_k), \omega_K)$ because $\mathcal{S}$ and the late copies would suffice to cover the former and $\mathcal{S}$ and the early copies would suffice to cover the latter interval.

Hence, the lemma follows if we show that $r_x \ge \omega_K - 2\sum_{k=1}^{K}(\omega_k - \alpha_k)$ and $d_x - \left(1 + \frac{\varepsilon}{2}\right) p_x \le \alpha_1 + 2\sum_{k=1}^{K}(\omega_k - \alpha_k)$. To this end, we observe that

$$\begin{aligned}(\alpha_1 - r_x) + \left(\left(d_x - \left(1 + \frac{\varepsilon}{2}\right) p_x\right) - \omega_K\right) &= \left(\left(d_x - \left(1 + \frac{\varepsilon}{2}\right) p_x\right) - r_x\right) - (\omega_K - \alpha_1) \\ &\le 2 \sum_{k=1}^{K} (\omega_k - \alpha_1) - (\omega_K - \alpha_1),\end{aligned}$$

where we used that $\alpha_1 \ge r_x$ and $\omega_K \le d_x - \left(1 + \frac{\varepsilon}{2}\right) p_x$ by assumption on $\mathcal{S}$. This implies that both summands on the left hand side are bounded by the term on the right hand side. Rearranging shows the above bounds on r_x and $d_x - \left(1 + \frac{\varepsilon}{2}\right) p_x$ and proves the lemma. ◀

▶ **Lemma 11.** *For each job $j \in J_\ell$, j's processing intervals are contained in one contiguous interval of $\bigcup_{I \in \mathcal{I}_\ell} I$.*

Proof. The statement holds trivially if j only has one processing interval as this interval is in $\mathcal{I}_\ell$. If j is preempted at some time τ and resumed at some later time τ', then the two-threshold algorithm processes higher-density jobs in the interval $[\tau, \tau')$. By definition, these higher-density jobs are in J_ℓ if $j \in J_\ell$. Hence, the processing intervals of j together with these higher-density jobs form a contiguous interval in $\bigcup_{I\in\mathcal{I}_\ell} I$. ◀

Consider subsets of jobs of J_ℓ that are inclusion-wise maximal with respect to the processing intervals of the corresponding jobs and their copies forming exactly one contiguous interval in $\bigcup_{I\in\mathcal{I}_\ell} I$. Let $J_{\ell,k}$ for $1 \le k \le K$ and $K \in \mathbb{N}$ be those maximal subsets of J_ℓ, i.e., for each k, the processing intervals of the jobs in $J_{\ell,k}$ form a contiguous interval and adding one more job to $J_{\ell,k}$ would create at least one more interval. Lemma 11 and Lemma 10 imply the following corollary.

▶ **Corollary 12.** *The above defined sets $J_{\ell,1}, \dots, J_{\ell,K}$ partition J_ℓ. If job $x \in X$ is blocked at level ℓ, then there exists exactly one index $k \in \{1, \dots, K\}$ such that $J_{\ell,k}$ blocks x.*

We now partition X as follows: Let $X_L \subseteq X$ and $X_S \subseteq X$ be the jobs in Opt that are, for at least half of their availability interval $\left[r_x, d_x - \left(1+\frac{\varepsilon}{2}\right)p_x\right)$ blocked by jobs with larger and smaller processing times, respectively. Let $X_{*\ell} \subset X_*$ for $* \in \{L, S\}$ be the jobs that are blocked at level ℓ. The previously proven structural properties allow us to upper bound the total time that Opt has available for processing jobs in $X_{S\ell}$ and $X_{L\ell}$ in the following two lemmas.

▶ **Lemma 13.** *For each level $\ell \in \mathbb{Z}$, the total time that Opt has available for processing jobs in $X_{L\ell}$ is at most $\left(4+\frac{\varepsilon}{2}\right)\sum_{j\in J_\ell} p_j$.*

Proof. By Corollary 12 it suffices to separately show the lemma for each maximal subset J'.

Consider a job x that is blocked by a subset of J' for at least half of $\left[r_x, d_x - \left(1+\frac{\varepsilon}{2}\right)p_x\right)$, where the jobs in J' blocking x have a larger processing time than x. By the definition of the two-threshold algorithm, this implies that there is a job $j \in J'$ with $p_x \le p_j$ that is processed during $\left[r_x, d_x - \left(1+\frac{\varepsilon}{2}\right)p_x\right)$. By Lemma 10, $\left[r_x, d_x - \left(1+\frac{\varepsilon}{2}\right)p_x\right) \subseteq \bigcup_{I\in\mathcal{I}_\ell} I$ and, therefore, $\left[r_x, d_x - \left(1+\frac{\varepsilon}{2}\right)p_x\right)$ is contained in the interval $I = [\alpha, \omega)$ associated with the jobs in J'. Combining these two observations implies that

$$[r_x, d_x) = \left[r_x, d_x - \left(1+\frac{\varepsilon}{2}\right)p_x\right) \cup \left[d_x - \left(1+\frac{\varepsilon}{2}\right)p_x, d_x\right) \subseteq I \cup \left[\omega, \omega + \left(1+\frac{\varepsilon}{2}\right)\sum_{j\in J'} p_j\right).$$

Using that the length of I is at most $3\sum_{j\in J'} p_j$ concludes the proof. ◀

▶ **Lemma 14.** *For each level $\ell \in \mathbb{Z}$, the total time that Opt has available for processing jobs in $X_{S\ell}$ is at most $\left(\frac{4}{\varepsilon}+5\right)\sum_{j\in J_\ell} p_j$.*

Proof. By Corollary 12 it suffices to separately show the lemma for each maximal subset J'. Let $I = [\alpha, \omega)$ be the interval associated with J'.

Consider a job x that is blocked by a subset of J' for at least half of $\left[r_x, d_x - \left(1+\frac{\varepsilon}{2}\right)p_x\right)$, where the jobs in J' blocking x have a smaller processing time than x. By definition of the two-threshold algorithm, this implies that there is a job $j \in J'$ that is processed during $\left[r_x, d_x - \left(1+\frac{\varepsilon}{2}\right)p_x\right)$.

By our slackness assumption, it holds that $d_x - r_x \geq (1+\varepsilon)p_x$ or equivalently, $p_x \leq \frac{2}{\varepsilon}\left(d_x - \left(1+\frac{\varepsilon}{2}\right)p_x - r_x\right)$. Since x is blocked for at least half of $\left[r_x, d_x - \left(1+\frac{\varepsilon}{2}\right)p_x\right)$ by jobs in J', this implies $p_x \leq \frac{4}{\varepsilon}\sum_{j\in J'} p_j$. Thus,

$$\begin{aligned}[r_x, d_x) &= \left[r_x, d_x - \left(1+\frac{\varepsilon}{2}\right)p_x\right) \cup \left[d_x - \left(1+\frac{\varepsilon}{2}\right)p_x, d_x\right) \\ &\subseteq I \cup \left[\omega, \omega + \left(\frac{4}{\varepsilon}+2\right)\sum_{j\in J'} p_j\right).\end{aligned}$$

Using again that the length of I is at most $3\sum_{j\in J'} p_j$ concludes the proof. ◀

Proof of Theorem 7. We now bound the weight of the sets $X_{S\ell}$ and $X_{L\ell}$ separately for each $\ell \in \mathbb{Z}$.

By Lemma 14, the time available for processing jobs in $X_{S\ell}$ is bounded from above by $\left(\frac{4}{\varepsilon}+5\right)\sum_{j\in J_\ell} p_j$. Being blocked at level ℓ by smaller jobs implies that $\rho_x \leq 4\cdot 2\cdot 2^\ell = 8\cdot 2^\ell$. Hence,

$$\sum_{x\in X_{S\ell}} w_x = \sum_{x\in X_{S\ell}} \rho_x p_x \leq 8\cdot 2^\ell \left(\frac{4}{\varepsilon}+5\right)\sum_{j\in J_\ell} p_j.$$

Similarly, by Lemma 13, the time available for processing jobs in $X_{L\ell}$ is upper bounded by $\left(4+\frac{\varepsilon}{2}\right)\sum_{j\in J_\ell} p_j$ and being blocked at level ℓ by larger jobs implies $\rho_x \leq \frac{8}{\varepsilon}\cdot 2\cdot 2^\ell$, where we used Observation 5 for a blocking job j with $p_x \in \left(\frac{\varepsilon}{2}p_j, p_j\right]$. Hence,

$$\sum_{x\in X_{L\ell}} w_x = \sum_{x\in X_{L\ell}} \rho_x p_x \leq \frac{16}{\varepsilon}\cdot 2^\ell \left(4+\frac{\varepsilon}{2}\right)\sum_{j\in J_\ell} p_j.$$

Combining the last two equations with Lemma 8 gives

$$\sum_{x\in X} w_x \leq 4\left(8\cdot\left(\frac{4}{\varepsilon}+5\right) + \frac{16}{\varepsilon}\left(4+\frac{\varepsilon}{2}\right)\right)\sum_{j\in J} w_j = 192\left(\frac{2}{\varepsilon}+1\right)\sum_{j\in J} w_j = \mathcal{O}\left(\frac{1}{\varepsilon}\right)\sum_{j\in J} w_j\,.$$

◀

Proof of main result

Proof of Theorem 1. Recall that Opt is the set of jobs scheduled in an optimal non-migratory solution and that F is the set of jobs that the two-threshold algorithm completes on time. Combining Theorems 6 and 7, we obtain

$$\sum_{x\in \textsc{Opt}} w_x \leq \sum_{x\in X} w_x + \sum_{x\in J} w_x \leq \mathcal{O}\left(\frac{1}{\varepsilon}\right)\sum_{j\in J} w_j \leq \mathcal{O}\left(\frac{1}{\varepsilon}\right)\sum_{j\in F} w_j\,, \tag{2}$$

which concludes the proof of our main result. ◀

Proof of Theorem 2. On identical machines, it is known that the optimal throughput achievable without migration is within a constant multiplicative factor of the throughput achievable using migration by Kalyanasundaram and Pruhs [14]. More precisely, as before, let Opt be the subset of jobs finished by an optimal (offline) non-migratory schedule, and let $\textsc{Opt}_{\text{mig}}$ be the subset of jobs finished by an optimal (offline) schedule that is allowed to use migration. Then, Theorem 1.1 in [14] shows that $\frac{1}{6}\sum_{j\in\textsc{Opt}_{\text{mig}}} w_j \leq \sum_{j\in\textsc{Opt}} w_j$. Combining this with (2), we immediately obtain

$$\sum_{j\in\textsc{Opt}_{\text{mig}}} w_j \leq \mathcal{O}(1)\sum_{j\in\textsc{Opt}} w_j \leq \mathcal{O}\left(\frac{1}{\varepsilon}\right)\sum_{j\in F} w_j\,,$$

which proves Theorem 2. ◀

5 Conclusion

We have presented a provably best possible non-migratory algorithm for online weighted throughput maximization on unrelated machines, that is $\mathcal{O}\big(\frac{1}{\varepsilon}\big)$-competitive against an optimal non-migratory schedule. Even for a single machine, only an $\mathcal{O}\big(\frac{1}{\varepsilon^2}\big)$-competitive algorithm was previously known [18] while the lower bound was $\Omega\big(\frac{1}{\varepsilon}\big)$ [8]. Our result closes this gap on a single machine.

In contrast to special cases such as maximizing throughput with unit weights [19] or maximizing machine utilization ($w_j = p_j$) [16], it is known that $\mathcal{O}(1)$-competitive algorithms are not possible even on identical machines and even when using randomization [7]. It is conceivable that $o\big(\frac{1}{\varepsilon}\big)$-competitive algorithms are possible for $m \geq 2$ identical machines, which we leave as an interesting open problem.

References

1 Yossi Azar, Inna Kalp-Shaltiel, Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Truthful online scheduling with commitments. In *EC*, pages 715–732. ACM, 2015. doi:10.1145/2764468.2764535.

2 Sanjoy K. Baruah and Jayant R. Haritsa. Scheduling for overload in real-time systems. *IEEE Trans. Computers*, 46(9):1034–1039, 1997. doi:10.1109/12.620484.

3 Sanjoy K. Baruah, Jayant R. Haritsa, and Nitin Sharma. On-line scheduling to maximize task completions. In *RTSS*, pages 228–236. IEEE Computer Society, 1994. doi:10.1109/REAL.1994.342713.

4 Sanjoy K. Baruah, Gilad Koren, Decao Mao, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, Dennis E. Shasha, and Fuxing Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4(2):125–144, 1992. doi:10.1007/BF00365406.

5 Sanjoy K. Baruah, Gilad Koren, Bhubaneswar Mishra, Arvind Raghunathan, Louis E. Rosier, and Dennis E. Shasha. On-line scheduling in the presence of overload. In *FOCS*, pages 100–110. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185354.

6 Luca Becchetti, Stefano Leonardi, and S. Muthukrishnan. Scheduling to minimize average stretch without migration. In *SODA*, pages 548–557. ACM/SIAM, 2000.

7 Ran Canetti and Sandy Irani. Bounding the power of preemption in randomized scheduling. *SIAM J. Comput.*, 27(4):993–1015, 1998. doi:10.1137/S0097539795283292.

8 Lin Chen, Franziska Eberle, Nicole Megow, Kevin Schewior, and Cliff Stein. A general framework for handling commitment in online throughput maximization. *Math. Prog.*, 183:215–247, 2020. doi:10.1007/s10107-020-01469-2.

9 Bhaskar DasGupta and Michael A. Palis. Online real-time preemptive scheduling of jobs with deadlines. In *APPROX*, volume 1913 of *Lecture Notes in Computer Science*, pages 96–107. Springer, 2000. doi:10.1007/3-540-44436-X_11.

10 Franziska Eberle, Nicole Megow, and Kevin Schewior. Online throughput maximization on unrelated machines: Commitment is no burden. *ACM Trans. Algorithms*, 19(1), February 2023. doi:10.1145/3569582.

11 Juan A. Garay, Joseph Naor, Bülent Yener, and Peng Zhao. On-line admission control and packet scheduling with interleaving. In *INFOCOM*, pages 94–103. IEEE Computer Society, 2002. doi:10.1109/INFCOM.2002.1019250.

12 Michael H. Goldwasser. Patience is a virtue: The effect of slack on competitiveness for admission control. *J. Sched.*, 6(2):183–211, 2003. doi:10.1023/A:1022994010777.

13 Samin Jamalabadi, Chris Schwiegelshohn, and Uwe Schwiegelshohn. Commitment and slack for online load maximization. In *SPAA*, pages 339–348. ACM, 2020. doi:10.1145/3350755.3400271.

14 Bala Kalyanasundaram and Kirk Pruhs. Eliminating migration in multi-processor scheduling. *J. Algorithms*, 38(1):2–24, 2001. doi:10.1006/jagm.2000.1128.

15 Bala Kalyanasundaram and Kirk Pruhs. Maximizing job completions online. *J. Algorithms*, 49(1):63–85, 2003. `doi:10.1016/S0196-6774(03)00074-9`.

16 Gilad Koren and Dennis E. Shasha. MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling. *Theor. Comput. Sci.*, 128(1&2):75–97, 1994. `doi:10.1016/0304-3975(94)90165-1`.

17 Gilad Koren and Dennis E. Shasha. $\mathrm{D}^{\mathrm{over}}$: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM J. Comput.*, 24(2):318–339, 1995. `doi:10.1137/S0097539792236882`.

18 Brendan Lucier, Ishai Menache, Joseph Naor, and Jonathan Yaniv. Efficient online scheduling for deadline-sensitive jobs: Extended abstract. In *SPAA*, pages 305–314. ACM, 2013. `doi:10.1145/2486159.2486187`.

19 Benjamin Moseley, Kirk Pruhs, Clifford Stein, and Rudy Zhou. A competitive algorithm for throughput maximization on identical machines. In *IPCO*, volume 13265 of *Lecture Notes in Computer Science*, pages 402–414. Springer, 2022.

20 Kirk Pruhs and Clifford Stein. How to schedule when you have to buy your energy. In *APPROX*, volume 6302 of *Lecture Notes in Computer Science*, pages 352–365. Springer, 2010. `doi:10.1007/978-3-642-15369-3_27`.

21 Chris Schwiegelshohn and Uwe Schwiegelshohn. The power of migration for online slack scheduling. In *ESA*, volume 57 of *LIPIcs*, pages 75:1–75:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPIcs.ESA.2016.75`.