Eberle, Franziska ; Megow, Nicole ; Schewior, Kevin

Online Throughput Maximization on Unrelated Machines: Commitment is No Burden

Journal Article as: peer-reviewed accepted version (Postprint)

DOI of this document*(secondary publication): https://doi.org/10.26092/elib/3189

Publication date of this document: 01/08/2024

* for better findability or for reliable citation

Recommended Citation (primary publication/Version of Record) incl. DOI:

Franziska Eberle, Nicole Megow, and Kevin Schewior. 2023. Online Throughput Maximization on Unrelated Machines: Commitment is No Burden. ACM Trans. Algorithms 19, 1, Article 10 (January 2023), 25 pages. https://doi.org/10.1145/3569582.

ONLINE THROUGHPUT MAXIMIZATION ON UNRELATED MACHINES: COMMITMENT IS NO BURDEN*

FRANZISKA EBERLE†, NICOLE MEGOW†, AND KEVIN SCHEWIOR‡

Abstract. We consider a fundamental online scheduling problem in which jobs with processing times and deadlines arrive online over time at their release dates. The task is to determine a feasible preemptive schedule on a single or multiple possibly unrelated machines that maximizes the number of jobs that complete before their deadline. Due to strong impossibility results for competitive analysis on a single machine, we require that jobs contain some *slack* $\varepsilon > 0$, which means that the feasible time window for scheduling a job is at least $1 + \varepsilon$ times its processing time on each eligible machine. Our contribution is two-fold: (i) We give the first non-trivial online algorithms for throughput maximization on unrelated machines, and (ii), this is the main focus of our paper, we answer the question on how to handle commitment requirements which enforce that a scheduler has to guarantee at a certain point in time the completion of admitted jobs. This is very relevant, e.g., in providing cloud-computing services, and disallows last-minute rejections of critical tasks. We present an algorithm for unrelated machines that is $\Theta\big(\frac{1}{\varepsilon}\big)$-competitive when the scheduler must commit upon starting a job. Somewhat surprisingly, this is the same optimal performance bound (up to constants) as for scheduling without commitment on a single machine. If commitment decisions must be made before a job's slack becomes less than a δ-fraction of its size, we prove a competitive ratio of $\mathcal{O}\big(\frac{1}{\varepsilon-\delta}\big)$ for $0 < \delta < \varepsilon$. This result nicely interpolates between commitment upon starting a job and commitment upon arrival. For the latter commitment model, it is known that no (randomized) online algorithm admits any bounded competitive ratio. While we mainly focus on scheduling without migration, our results also hold when comparing against a migratory optimal solution in case of identical machines.

Key words. Deadline scheduling, throughput, online algorithms, competitive analysis, unrelated machines, migration

AMS subject classifications. 68W27, 90B35, 68W40, 68Q25

1. Introduction.

We consider the following online scheduling problem: there are given m unrelated parallel machines. Jobs from an unknown job set arrive online over time at their *release dates* r_j. Each job j has a *deadline* d_j and a *processing time* $p_{ij} \in \mathbb{R}_+ \cup \{\infty\}$, which is the execution time of j when processing on machine i; both job parameters become known to an algorithm at job arrival. We denote a machine i with $p_{ij} < \infty$ as *eligible* for job j. If all machines are identical, $p_{ij} = p_j$ holds for every job j, and we omit the index i. When scheduling these jobs or a subset of them, we allow *preemption*, i.e., the processing of a job can be interrupted at any time and may resume later without any additional cost. We mainly study scheduling without *migration* which means that a job must run completely on one machine. In case that we allow migration, a preempted job can resume processing on any machine, but no job can run simultaneously on two or more machines.

In a feasible schedule, two jobs are never processing at the same time on the same machine. A job is said to *complete* if it receives p_{ij} units of processing time

*Submitted to the editors June 9, 2021. A preliminary version of the results on online scheduling with commitment on parallel identical machines was published at the European Symposium of Algorithms 2020 [15].

Funding: The work of the second author was partially supported by the German Science Foundation (DFG) under contract ME 3825/1. The work of the third author was partially supported by the DAAD within the PRIME program using funds of BMBF and the EU Marie Curie Actions.

†Faculty of Mathematics and Computer Science, University of Bremen, Germany (franziska.eberle@posteo.de, nicole.megow@uni-bremen.de).

‡Department of Mathematics and Computer Science, Universität zu Köln, Cologne, Germany (kschewior@gmail.com).

on machine i within the interval $[r_j, d_j)$ if j is processed by machine i. The number of completed jobs in a feasible schedule is called *throughput*. The task is to find a feasible schedule with maximum throughput. We refer to this problem as *throughput maximization*.

As jobs arrive online and scheduling decisions are irrevocable, we cannot hope to find an optimal schedule even when scheduling on a single machine [12]. To assess the performance of online algorithms, we resort to standard *competitive analysis*. This means, we compare the throughput of an online algorithm with the throughput achievable by an optimal offline algorithm that knows the job set in advance.

On a single machine, it is well-known that "tight" jobs with $d_j - r_j \approx p_j$ prohibit competitive online decision making as jobs must start immediately and do not leave a chance for observing online arrivals [7]. Thus, it is commonly required that jobs contain some *slack* $\varepsilon > 0$, i.e., every job j satisfies $d_j - r_j \geq (1+\varepsilon)p_j$. In the more general setting with unrelated machines, we assume that each job j satisfies $d_j - r_j \geq (1+\varepsilon)p_{ij}$ for each machine i that is eligible for j, i.e., each machine i with $p_{ij} < \infty$. The competitive ratio of our online algorithm will be a function of ε; the greater the slack, the better should the performance of our algorithm be. This slackness parameter has been considered in a multitude of previous work, e.g., in [2, 5, 10, 17, 18, 31, 34]. Other results for scheduling with deadlines use speed scaling, which can be viewed as another way to add slack to the schedule, see, e.g., [1, 3, 22, 24, 32].

In this paper, we focus on the question how to handle *commitment* requirements in online throughput maximization. Modeling commitment addresses the issue that a high-throughput schedule may abort jobs close to their deadlines in favor of many shorter and more urgent tasks [16], which may not be acceptable for the job owner. Consider a company that starts outsourcing mission-critical processes to external clouds and that needs a guarantee that jobs complete before a certain time point when they cannot be moved to another computing cluster anymore. In other situations, a commitment to complete jobs might be required even earlier just before starting the job, e.g., for a faultless copy of a database [10].

Different commitment models have been formalized [2,10,31]. The requirement to commit at a job's release date has been ruled out for online throughput maximization by strong impossibility results (even for randomized algorithms) [10]. We distinguish two commitment models.

(i) *Commitment upon job admission*: an algorithm may discard a job any time before its start, we say its admission. This reflects a situation such as the faultless copy of a database.

(ii) *δ-commitment*: given $0 < \delta < \varepsilon$, an algorithm must commit to complete a job while the job's remaining slack is at least a δ-fraction of its original processing time. This models an early enough commitment (parameterized by δ) for mission-critical jobs. For identical parallel machines, the latest time for committing to job j is then $d_j - (1+\delta)p_j$. When given unrelated machines, such a commitment model might be arguably less relevant. We consider it only for non-migratory schedules and include also the choice of a processor in the commitment; we define the latest time point for committing to job j as $d_j - (1+\delta)p_{ij}$ when processing j on machine i.

Recently, a first unified approach has been presented for these models for a single machine [10]. In this and other works [2,31], there remained gaps in the performance bounds and it was left open whether scheduling with commitment is even "harder" than without commitment. Moreover, it remained unsettled whether the problem is tractable on multiple identical or even heterogeneous machines.

In this work, we give tight results for online throughput maximization on unrelated parallel machines and answer the "hardness" question to the negative. We give an algorithm that achieves the provably best competitive ratio (up to constant factors) for the aforementioned commitment models. Somewhat surprisingly, we show that the same competitive ratio of $\mathcal{O}\big(\frac{1}{\varepsilon}\big)$ can be achieved for both, scheduling *without* commitment and *with* commitment upon admission. For unrelated machines, this is the first nontrivial result for online throughput maximization with and without commitment. For identical parallel machines, this is the first online algorithm with bounded competitive ratio for arbitrary slack parameter ε. Interestingly, for this machine environment, our algorithm does not require job migration in order to be competitive against a migratory algorithm.

1.1. Related work. Preemptive online scheduling and admission control have been studied rigorously. There are several results regarding the impact of deadlines on online scheduling; see, e.g., [6, 17, 18] and references therein. In the following we give an overview of the literature focused on (online) throughput maximization.

Offline scheduling. In case that the jobs and their characteristics are known to the scheduler in advance, the notion of commitment is irrelevant as an offline algorithm only starts jobs that will be completed on time; there is no benefit in starting jobs without completing them. The offline problem is well understood: For throughput maximization on a single machine, there is a polynomial-time algorithm by Lawler [29]. The model where jobs have *weights* and the task is to maximize the total weight of jobs completed on time (*weighted throughput*) is NP-hard and we do not expect polynomial time algorithms. The algorithm by Lawler solves this problem optimally in time $\mathcal{O}(n^5 w_{\max})$, where $w_{\max} = \max_j w_j$, and can be used to design a fully polynomial-time approximation scheme (FPTAS) [33].

When given multiple identical machines, (unweighted) throughput maximization becomes NP-hard even for identical release dates [30]. Kalyanasundaram and Pruhs [25] show a 6-approximate reduction to the single-machine problem which implies a $(6 + \delta)$-approximation algorithm for weighted throughput maximization on identical parallel machines, for any $\delta > 0$, using the FPTAS for the single-machine problem [33]. Preemptive throughput maximization on unrelated machines is much less understood from an approximation point of view. The problem is known to be strongly NP-hard [14], even without release dates [35]. We are not aware of any approximation results for preemptive throughput maximization on unrelated machines. The situation is different for non-preemptive scheduling. In this case, throughput maximization is MAX-SNP hard [4] and several approximation algorithms for this general problem as well as for identical parallel machines and other special cases are known; see, e.g., [4, 9, 21].

Online scheduling without commitment. For single-machine throughput maximization, Baruah, Haritsa, and Sharma [6] show that, in general, no deterministic online algorithm achieves a bounded competitive ratio. Thus, their result justifies our assumption on ε-slackness of each job. Moreover, they consider special cases such as unit-size jobs or agreeable deadlines where they provide constant-competitive algorithms even without further assumptions on the slack of the jobs. Here, deadlines are agreeable if $r_j \leq r_{j'}$ for two jobs j and j' implies $d_j \leq d_{j'}$. In our prior work [10], we develop a $\Theta\big(\frac{1}{\varepsilon}\big)$-competitive algorithm and show a matching lower bound for deterministic algorithms. While this is ruled out for deterministic algorithms, Kalyanasundaram and Pruhs [26] give a *randomized* $\mathcal{O}(1)$-competitive algorithm for throughput maximization on a single machine without slackness assumption.

For maximizing weighted throughput, Lucier et al. [31] give an $\mathcal{O}\big(\frac{1}{\varepsilon^2}\big)$-competitive online algorithm for scheduling on identical parallel machines. In a special case of this problem, called *machine utilization* the goal is to maximize the total processing time of completed jobs. This problem is much more tractable. On a single machine, Baruah et al. [7,8] provide a best-possible online algorithm achieving a competitive ratio of 4, even without any slackness assumptions. Baruah and Haritsa [5] are the first to investigate the problem under the assumption of ε-slack and give a $\frac{1+\varepsilon}{\varepsilon}$-competitive algorithm which is asymptotically best possible. For parallel identical machines (though without migration), DasGupta and Palis [11] give a simple greedy algorithm that achieves the same performance guarantee of $\frac{1+\varepsilon}{\varepsilon}$ and give an asymptotically matching lower bound. Schwiegelshohn and Schwiegelshohn [34] show that migration helps an online algorithm and improves the competitive ratio to $\mathcal{O}\big(\sqrt[m]{1/\varepsilon}\big)$ for m machines.

In a line of research without slackness assumption, Baruah et al. [8] show a lower bound of $(1+\sqrt{k})^2$ for deterministic single-machine algorithms, where $k = \frac{\max_j w_j/p_j}{\min_j w_j/p_j}$ is the *importance ratio* of a given instance. Koren and Shasha give a matching upper bound [28] and generalize it to $\Theta(\ln k)$ for parallel machines if $k > 1$ [27].

Online scheduling with commitment upon job arrival. In our prior work [10], we rule out bounded competitive ratios for any (even randomized) online algorithm for throughput maximization with commitment upon job arrival, even on a single machine. Previously, such impossibility results where only shown exploiting weights [31].

Again, the special case $w_j = p_j$, or machine utilization, is much more tractable than weighted or unweighted throughput maximization. A simple greedy algorithm already achieves the best possible competitive ratio $\frac{1+\varepsilon}{\varepsilon}$ on a single machine, even for commitment upon arrival, as shown by DasGupta and Palis [11] and the matching lower bound by Garay et al. [17]. For scheduling with commitment upon arrival on m parallel identical machines, there is an $\mathcal{O}(\sqrt[m]{1/\varepsilon})$-competitive algorithm and an almost matching lower bound by Schwiegelshohn and Schwiegelshohn [34]. Suprisingly, this model also allows for bounded competitive ratios when preemption is not allowed. In this setting, Goldwasser and Kerbikov [19] give a best possible $\big(2+\frac{1}{\varepsilon}\big)$-competitive algorithm on a single machine. Very recently, Jamalabadi, Schwiegelshohn, and Schwiegelshohn [23] extend this model to parallel machines; their algorithm is near optimal with a performance guarantee approaching $\ln\frac{1}{\varepsilon}$ as m tends to infinity.

Online scheduling with commitment upon admission and δ-commitment. In our previous work [10], we design an online single-machine algorithm, called the *region algorithm*, that simultaneously (with the respective choice of parameters) achieves the first non-trivial upper bounds for both commitment models. For commitment upon job admission, our bound is $\mathcal{O}\big(\frac{1}{\varepsilon^2}\big)$, and in the δ-commitment model it is $\mathcal{O}\big(\frac{\varepsilon}{(\varepsilon-\delta)\delta^2}\big)$, for $0 < \delta < \varepsilon$. For scheduling on identical parallel machines and commitment upon admission, Lucier et al. [31] give a heuristic that empirically performs very well but for which they cannot show a rigorous worst-case bound. In fact, Azar et al. [2] show that no bounded competitive ratio is possible for weighted throughput maximization for small ε. For $\delta = \frac{\varepsilon}{2}$ in the δ-commitment model, they design (in the context of truthful mechanisms) an online algorithm for weighted throughput maximization that is $\Theta\big(\frac{1}{\sqrt[3]{1+\varepsilon}-1} + \frac{1}{(\sqrt[3]{1+\varepsilon}-1)^2}\big)$-competitive if the slack ε is sufficiently large, i.e., if $\varepsilon > 3$. For weighted throughput, this condition on the slack is necessary as is shown by a strong general lower bound, even on a single machine [10]. For the unweighted setting, we give the first rigorous upper bound for arbitrary ε in this paper for both models, commitment upon admission and δ-commitment, in the identical and even in the unrelated machine environment.

Machine utilization is again better understood. As commitment upon arrival is more restrictive than commitment upon admission and δ-commitment, the previously mentioned results immediately carry over and provide bounded competitive ratios.

1.2. Our results and techniques. Our main result is an algorithm that computes a non-migratory schedule that is best possible (up to constant factors) for online throughput maximization with and without commitment on identical parallel machines and, more generally, on unrelated machines. This is the first non-trivial online result for unrelated machines and it closes gaps for identical parallel machines. Our algorithm is universally applicable (by setting parameters properly) to both commitment models as well es scheduling without commitment.

THEOREM 1.1. *Consider throughput maximization on unrelated machines without migration. There is an* $\mathcal{O}\big(\frac{1}{\varepsilon-\delta'}\big)$*-competitive non-migratory online algorithm for scheduling with commitment, where* $\delta' = \frac{\varepsilon}{2}$ *in the model with commitment upon admission and* $\delta' = \max\{\delta, \frac{\varepsilon}{2}\}$ *in the* δ*-commitment model.*

For scheduling with commitment upon admission, this is (up to constant factors) an optimal online algorithm with competitive ratio $\Theta\big(\frac{1}{\varepsilon}\big)$, matching the lower bound of $\Omega\big(\frac{1}{\varepsilon}\big)$ for $m = 1$ [10]. For scheduling with δ-commitment, our result interpolates between the models with commitment upon starting a job and commitment upon arrival. If $\delta \leq \frac{\varepsilon}{2}$, the competitive ratio is $\Theta\big(\frac{1}{\varepsilon}\big)$, which is again best possible [10]. For $\delta \to \varepsilon$, the commitment requirements essentially implies commitment upon job arrival which has unbounded competitive ratio [10].

In our analysis, we compare a non-migratory schedule, obtained by our algorithm, with an optimal non-migratory schedule. However, in the case of identical machines the throughput of an optimal migratory schedule can only be larger by a constant factor than the throughput of an optimal non-migratory schedule. In fact, Kalyanasundaram and Pruhs [25] showed that this factor is at most $\frac{6m-5}{m}$. Thus, the competitive ratio for our non-migratory algorithm, when applied to identical machines, holds (up to this constant factor) also in a migratory setting.

COROLLARY 1.2. *Consider throughput maximization with or without migration on parallel identical machines. There is an* $\mathcal{O}\big(\frac{1}{\varepsilon-\delta'}\big)$*-competitive non-migratory online algorithm for scheduling with commitment, where* $\delta' = \frac{\varepsilon}{2}$ *in the model with commitment upon admission and* $\delta' = \max\{\delta, \frac{\varepsilon}{2}\}$ *in the* δ*-commitment model.*

The challenge in online scheduling with commitment is that, once we committed to complete a job, the remaining slack of this job has to be spent very carefully. The key component is a job admission scheme which is implemented by different parameters. The high-level objectives are:

(i) never start a job for the first time if its remaining slack is too small (parameter δ),
(ii) during the processing of a job, admit only significantly shorter jobs (parameter γ), and
(iii) for each admitted shorter job, block some time period (parameter β) during which no other jobs of similar size are accepted.

While the first two goals are quite natural and have been used before in the single and identical machine setting [10,31], the third goal is crucial for our new tight result. The intuition is the following: Think of a single eligible machine in a non-migratory schedule. Suppose we committed to complete a job with processing time 1 and have only a slack of $\mathcal{O}(\varepsilon)$ left before the deadline of this job. Suppose that c substantially smaller jobs of size $\frac{1}{c}$ arrive where c is the competitive ratio we aim for. On the one hand, if we do not accept any of them, we cannot hope to achieve c-competitiveness. On the other hand, accepting too many of them fills up the slack and, thus, leaves no room for even smaller jobs. The idea is to keep the flexibility for future small jobs by only accepting an ε-fraction of jobs of similar size (within a factor two).

We distinguish two time periods that guide the acceptance decisions. During the *scheduling interval* of a job j, we have a more restrictive acceptance scheme that ensures the completion of j whereas in the *blocking period* we guarantee the completion of previously accepted jobs. We call our algorithm *blocking* algorithm. This acceptance scheme is much more refined than the one of the known region algorithm in [10] that uses one long region with a uniform acceptance threshold and is then too conservative in accepting jobs.

Given that we consider the non-migratory version of the problem, a generalization from a single to multiple machines seems natural. It is interesting, however, that such a generalization works, essentially on a per-machine basis, even for unrelated machines and comes at no loss in the competitive ratio.

Clearly, scheduling with commitment is more restrictive than without commitment. Therefore, our algorithm is also $O\big(\frac{1}{\varepsilon}\big)$-competitive for maximizing the throughput on unrelated machines without any commitment requirements. Again, this is optimal (up to constant factors) as it matches the lower bound on the competitive ratio for deterministic online algorithms on a single machine [10].

COROLLARY 1.3. *There is a $\Theta\big(\frac{1}{\varepsilon}\big)$-competitive algorithm for online throughput maximization on unrelated machines without commitment requirements and without migration.*

However, for scheduling without commitment, we are able to generalize the simpler region algorithm presented for the single-machine problem in [10] to scheduling on unrelated machines.

THEOREM 1.4. *A generalization of the region algorithm is $\Theta\big(\frac{1}{\varepsilon}\big)$-competitive for online throughput maximization on unrelated machines without commitment requirements and without migration.*

Besides presenting a simpler algorithm for throughput maximization without commitment, we show this result to present an additional application of our technical findings for the analysis of the blocking algorithm. We give details later. On a high level, we show a key lemma on the size of non-admitted jobs for a big class of online algorithms which results in an upper bound on the throughput of an optimal (offline) non-migratory algorithm. This key lemma can be used in the analysis of both algorithms, blocking and region. In fact, also the analysis of the original region algorithm for a single machine [10] becomes substantially easier.

In case of identical machines, again, we can apply the result by Kalyanasundaram and Pruhs [25] that states that the throughput of an optimal migratory schedule is larger by at most a constant factor than the throughput of an optimal non-migratory schedule. Thus, the result in Theorem 1.4 holds also in a migratory setting when scheduling on identical machines.

COROLLARY 1.5. *A generalization of the region algorithm is $\Theta\big(\frac{1}{\varepsilon}\big)$-competitive for online throughput maximization on multiple identical machines without commitment requirements, with and without migration.*

Outline of the paper. In Section 2, we describe and outline the analysis of our new non-migratory algorithm. It consists of two parts, which are detailed in Sections 3 and 4: firstly, we argue that all jobs admitted by our algorithm can complete by their deadline and, secondly, we prove that we admit "sufficiently many" jobs. In Section 5, we generalize the known region algorithm, developed for a single machine in our prior work [10], to a non-migratory algorithm without commitment on unrelated machines. We show how to apply a new key technique developed for the analysis in Section 4 to analyze it and prove the same competitive ratio (up to constant factors) as for a single machine.

2. The blocking algorithm. In this section, we describe the *blocking algorithm* for scheduling with commitment. We assume that the slackness constant $\varepsilon > 0$ and, in the δ-commitment model, $\delta \in (0, \varepsilon)$ are given. If δ is not part of the input or if $\delta \leq \frac{\varepsilon}{2}$, then we set $\delta = \frac{\varepsilon}{2}$.

The algorithm never migrates jobs between machines, i.e., a job is only processed by the machine that initially started to process it. In this case, we say the job has been *admitted* to this machine. Moreover, our algorithm commits to completing a job upon admission (even in the δ-commitment model). Hence, its remaining slack has to be spent very carefully on admitting other jobs to still be competitive. As our algorithm does not migrate jobs, it transfers the admission decision to the shortest admitted and not yet completed job on each machine. A job only admits significantly shorter jobs and prevents the admission of too many jobs of similar size. To this end, the algorithm maintains two types of intervals for each admitted job, a *scheduling interval* and a *blocking period*. A job can only be processed in its scheduling interval. Thus, it has to complete in this interval while admitting other jobs. Job j only admits jobs that are smaller by a factor of at least $\gamma = \frac{\delta}{16} < 1$. For an admitted job k, job j creates a blocking period of length at most βp_{ik}, where $\beta = \frac{16}{\delta}$, which blocks the admission of similar-length jobs (cf. Figure 1). The scheduling intervals and blocking periods of jobs admitted by j will always be pairwise disjoint and completely contained in the scheduling interval of j.

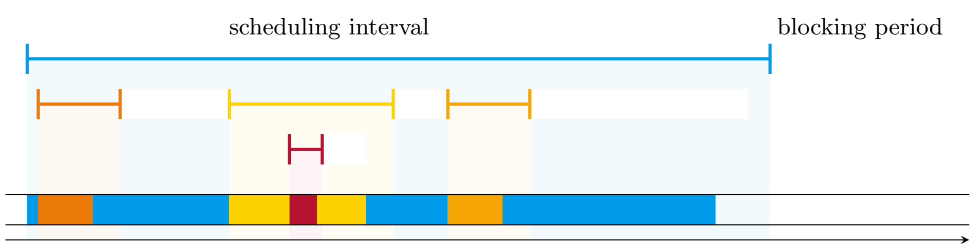

FIG. 1. *Scheduling interval, blocking period, and processing intervals*

Scheduling jobs. Independent of the admission scheme, the blocking algorithm follows the SHORTEST PROCESSING TIME (SPT) order for the set of uncompleted jobs assigned to a machine. SPT ensures that a job j has highest priority in the blocking periods of any job k admitted by j.

Admitting jobs. The algorithm keeps track of *available* jobs at any time point τ. A job j with $r_j \leq \tau$ is called available for machine i if it has not yet been admitted to a machine by the algorithm and its deadline is not too close, i.e., $d_j - \tau \geq (1+\delta)p_{ij}$.

Whenever a job j is available for machine i at a time τ such that time τ is not contained in the scheduling interval of any other job admitted to i, the shortest such job j is immediately admitted to machine i at time $a_j := \tau$, creating the scheduling interval $S(j) = [a_j, e_j)$, where $e_j = a_j + (1+\delta)p_{ij}$ and an empty blocking period $B(j) = \emptyset$. In general, however, the blocking period of a job j is a finite union of time intervals associated with j, and its size is the sum of lengths of the intervals, denoted by $|B(j)|$. Both, blocking period and scheduling interval, depend on machine i but we omit i from the notation as it is clear from the context; both periods are created after job j has been assigned to machine i.

Four types of events trigger a decision of the algorithm at time τ: the release of a job, the end of a blocking period, the end of a scheduling interval, and the admission of a job. In any of these four cases, the algorithm calls the *admission routine*. This subroutine iterates over all machines i and checks if j, the shortest job on i whose scheduling interval contains τ, can admit the currently shortest job $j^\star$ available for machine i.

To this end, any admitted job j checks whether $p_{ij^\star} < \gamma p_{ij}$. Only such jobs qualify for admission by j. Upon admission by j, job $j^\star$ obtains two disjoint consecutive intervals, the *scheduling interval* $S(j^\star) = [a_{j^\star}, e_{j^\star})$ and the *blocking period* $B(j^\star)$ of size at most $\beta p_{ij^\star}$. At the admission of job $j^\star$, the blocking period $B(j^\star)$ is planned to start at $e_{j^\star}$, the end of $j^\star$'s scheduling interval. During $B(j^\star)$, job j only admits jobs k with $p_{ik} < \frac{1}{2}p_{ij^\star}$.

Hence, when job j decides if it admits the currently shortest available job $j^\star$ at time τ, it makes sure that $j^\star$ is sufficiently small and that no job k of similar (or even smaller) processing time is blocking τ, i.e., it verifies that $\tau \notin B(k)$ for all jobs k with $p_{ik} \leq 2p_{ij^\star}$ admitted to the same machine. In this case, we say that $j^\star$ is a *child* of j and that j is the *parent* of $j^\star$, denoted by $\pi(j^\star) = j$. If job $j^\star$ is admitted at time τ by job j, the algorithm sets $a_{j^\star} = \tau$ and $e_{j^\star} = a_{j^\star} + (1+\delta)p_{ij^\star}$ and assigns the scheduling interval $S(j^\star) = [a_{j^\star}, e_{j^\star})$ to $j^\star$.

If $e_{j^\star} \leq e_j$, the routine sets $f_{j^\star} = \min\{e_j, e_{j^\star} + \beta p_{ij^\star}\}$ which determines $B(j^\star) = [e_{j^\star}, f_{j^\star})$. As the scheduling and blocking periods of children k of j are supposed to be disjoint, we have to *update the blocking periods*. First consider the job k with $p_{ik} > 2p_{ij^\star}$ admitted to the same machine whose blocking period contains τ (if it exists), and let $[e'_k, f'_k)$ be the maximal interval of $B(k)$ containing τ. We set $f''_k = \min\{e_j, f'_k + (1+\delta+\beta)p_{ij^\star}\}$ and replace the interval $[e'_k, f'_k)$ by $[e'_k, \tau) \cup [\tau + (1+\delta+\beta)p_{ij^\star}, f''_k)$. For all other jobs k with $B(k) \cap [\tau, \infty) \neq \emptyset$ admitted to the same machine, we replace the remaining part of their blocking period $[e'_k, f'_k)$ by $[e'_k + (1+\delta+\beta)p_{ij^\star}, f''_k)$ where $f''_k := \min\{e_j, f'_k + (1+\delta+\beta)p_{ij^\star}\}$. In this update, we follow the convention that $[e, f) = \emptyset$ if $f \leq e$. Observe that the length of the blocking period might decrease due to such updates.

Note that $e_{j^\star} > e_j$ is also possible as j does not take the end of its own scheduling interval e_j into account when admitting jobs. Thus, the scheduling interval of $j^\star$ would end outside the scheduling interval of j and inside the blocking period of j. During $B(j)$, the parent $\pi(j)$ of j, did not allocate the interval $[e_j, e_{j^\star})$ for completing jobs admitted by j but for ensuring its own completion. Hence, the completion of both $j^\star$ and $\pi(j)$ is not necessarily guaranteed anymore. To prevent this, we *modify all scheduling intervals* $S(k)$ (including $S(j)$) that contain time τ of jobs admitted to the same machine as $j^\star$ and their blocking periods $B(k)$. For each job k admitted to the same machine with $\tau \in S(k)$ (including j) and $e_{j^\star} > e_k$, we set $e_k = e_{j^\star}$. We also update their blocking periods (in fact, single intervals) to reflect their new starting points. If the parent $\pi(k)$ of k does not exist, $B(k)$ remains empty; otherwise we set $B(k) := [e_k, f_k)$ where $f_k = \min\{e_{\pi(k)}, e_k + \beta p_{ik}\}$. Note that, after this update, the blocking periods of any but the largest such job will be empty. Moreover, the just admitted job $j^\star$ does not get a blocking period in this special case.

During the analysis of the algorithm, we show that any admitted job j still completes before $a_j + (1+\delta)p_{ij}$ and that $e_j \leq a_j + (1+2\delta)p_{ij}$ holds in retrospect for all admitted jobs j. Thus, any job j that admits another job $j^\star$ tentatively assigns this job a scheduling interval of length $(1+\delta)p_{ij^\star}$ but, for ensuring its own completion, it is prepared to lose $(1+2\delta)p_{ij^\star}$ time units of its scheduling interval $S(j)$. We summarize the blocking algorithm in the following.

Algorithm Blocking algorithm

Scheduling Routine: At all times τ and on all machines i, run the job with shortest processing time that has been admitted to i and has not yet completed.

Event: Release of a new job at time τ
  Call Admission Routine.

Event: End of a blocking period or scheduling interval at time τ
  Call Admission Routine.

Admission Routine:
$i \leftarrow 1$
$j^\star \leftarrow$ a shortest job available at τ for machine i, i.e., $j^\star \in \arg\min\{p_{ij} \,|\, j \in \mathcal{J}, r_j \leq \tau \text{ and } d_j - \tau \geq (1+\delta)p_{ij}\}$
while $i \leq m$ **do**
  $K \leftarrow$ the set of jobs on machine i whose scheduling intervals contain τ
  if $K = \emptyset$ **then**
    admit job $j^\star$ to machine i, $a_{j^\star} \leftarrow \tau$, and $e_{j^\star} \leftarrow a_{j^\star} + (1+\delta)p_{ij^\star}$
    $S(j^\star) \leftarrow [a_{j^\star}, e_{j^\star})$ and $B(j^\star) \leftarrow \emptyset$
    call Admission Routine
  else
    $j \leftarrow \arg\min\{p_{ik} \,|\, k \in K\}$
    if $j^\star < \gamma p_{ij}$ **and** $\tau \notin B(j')$ for all j' admitted to i with $p_{ij'} \leq 2p_{ij^\star}$ **then**
      admit job $j^\star$ to machine i, $a_{j^\star} \leftarrow \tau$, and $e_{j^\star} \leftarrow a_{j^\star} + (1+\delta)p_{ij^\star}$
      if $e_{j^\star} \leq e_j$ **then**
        $f_{j^\star} \leftarrow \min\{e_j, e_{j^\star} + \beta p_{ij^\star}\}$
        $S(j^\star) \leftarrow [a_{j^\star}, e_{j^\star})$ and $B(j^\star) \leftarrow [e_{j^\star}, f_{j^\star})$
      else
        $S(j^\star) \leftarrow [a_{j^\star}, e_{j^\star})$ and $B(j^\star) \leftarrow \emptyset$
        modify $S(k)$ and $B(k)$ for $k \in K$
        update $B(j')$ for j' admitted to machine i with $B(j') \cap [\tau, \infty) \neq \emptyset$
        call Admission Routine
      end if
    else
      $i \leftarrow i + 1$
      $j^\star \leftarrow$ a shortest job available at τ for machine i, i.e., $j^\star \in \arg\min\{p_{ij} \,|\, j \in \mathcal{J}, r_j \leq \tau \text{ and } d_j - \tau \geq (1+\delta)p_{ij}\}$
    end if
  end if
end while

Roadmap for the analysis. During the analysis, it is sufficient to concentrate on instances with small slack, as also noted in [10]. For $\varepsilon > 1$ we run the blocking algorithm with $\varepsilon = 1$, which only tightens the commitment requirement, and obtain constant competitive ratios. Thus, we assume $0 < \varepsilon \leq 1$. For $0 < \delta < \varepsilon$, in the δ-commitment model an online scheduler needs to commit to the completion of a job j no later than $d_j - (1+\delta)p_{ij}$. Hence, committing to the completion of a job j at an earlier point in time clearly satisfies committing at a remaining slack of δp_{ij}. Therefore, we may assume $\delta \in [\frac{\varepsilon}{2}, \varepsilon)$.

The blocking algorithm does not migrate any job. In the analysis, we compare the throughput of our algorithm to the solution of an optimal non-migratory schedule. To do so, we rely on a key design principle of the blocking algorithm, which is that, whenever the job set admitted to a machine is fixed, the scheduling of the jobs follows the simple SPT order. This enables us to split the analysis into two parts.

In the first part, we argue that the scheduling routine can handle the admitted jobs sufficiently well. That is, every admitted jobs is completed on time; see Section 3. Here, we use that the blocking algorithm is non-migratory and consider each machine individually.

For the second part, we observe that the potential admission of a new job $j^\star$ to machine i is solely based on its availability and on its size relative to j_i, the job currently processed by machine i. More precisely, given the availability of $j^\star$, if $p_{ij^\star} < \gamma p_{ij_i}$, the time does not belong to the blocking period of a job k_i admitted to machine i with $p_{ij^\star} \geq \frac{1}{2}p_{ik_i}$ and i is the first machine (according to machine indices) with this property, then $j^\star$ is admitted to machine i. This implies that $\min\left\{\gamma p_{ij_i}, \frac{1}{2}p_{ik_i}\right\}$ acts as a *threshold*, and only available jobs with processing time less than this threshold qualify for admission by the blocking algorithm on machine i. Hence, any available job that the blocking algorithm does not admit has to exceed the threshold.

Based on this observation, we develop a general charging scheme for *any* non-migratory online algorithm satisfying the property that, at any time τ, the algorithm maintains a time-dependent threshold and the shortest available job smaller than this threshold is admitted by the algorithm. We formalize this description and analyze the competitive ratio of such algorithms in Section 4 before applying this general result to our particular algorithm.

3. Completing all admitted jobs on time. We show that the blocking algorithm finishes every admitted job on time in Theorem 3.1.

THEOREM 3.1. *Let $0 < \delta < \varepsilon$ be fixed. If $0 < \gamma < 1$ and $\beta \geq 1$ satisfy*

$$\frac{\beta/2}{\beta/2 + (1+2\delta)}\left(1 + \delta - 2(1+2\delta)\gamma\right) \geq 1, \tag{3.1}$$

then the blocking algorithm completes any job j admitted at $a_j \leq d_j - (1+\delta)p_{ij}$ on time.

Recall that we chose $\gamma = \frac{\delta}{16}$ and $\beta = \frac{16}{\delta}$, which guarantees that Equation (3.1) is satisfied.

As the blocking algorithm does not migrate jobs, it suffices to consider each machine individually in this section. The proof relies on the following observations: (i) The sizes of jobs admitted by job j that interrupt each others' blocking periods are geometrically decreasing, (ii) the scheduling intervals of jobs are completely contained in the scheduling intervals of their parents, and (iii) scheduling in SPT order guarantees that job j has highest priority in the blocking periods of its children. We start by proving the following technical lemma about the length of the final scheduling interval of an admitted job j, denoted by $|S(j)|$. In the proof, we use that $\pi(k) = j$ for two jobs j and k implies that $p_{ik} < \gamma p_{ij}$.

LEMMA 3.2. *Let $0 < \delta < \varepsilon$ be fixed. If $\gamma > 0$ satisfies $(1+2\delta)\gamma \leq \delta$, then $|S(j)| \leq (1+2\delta)p_{ij}$. Moreover, $S(j)$ contains the scheduling intervals and blocking periods of* *all descendants of j.*

Proof. Consider a machine i and let j be a job admitted to machine i. By definition of the blocking algorithm, the end point e_j of the scheduling interval of job j is only modified when j or one of j's descendants admits another job. Let us consider such a case: If job j admits a job k whose scheduling interval does not fit into the scheduling interval of j, we set $e_j = e_k = a_k + (1+\delta)p_{ik}$ to accommodate the scheduling interval $S(k)$ within $S(j)$. The same modification is applied to any ancestor j' of j with $e_{j'} < e_k$. This implies that, after such a modification of the scheduling interval, neither j nor any affected ancestor j' of j are the smallest jobs in their scheduling intervals anymore. In particular, no job whose scheduling interval was modified in such a case at time τ is able to admit jobs after τ. Hence, any job j can only admit other jobs within the interval $[a_j, a_j + (1+\delta)p_{ij})$. That is, $a_k \leq a_j + (1+\delta)p_{ij}$ for every job k with $\pi(k) = j$.

Thus, by induction, it is sufficient to show that $a_k + (1+2\delta)p_{ik} \leq a_j + (1+2\delta)p_{ij}$ for admitted jobs k and j with $\pi(k) = j$. Note that $\pi(k) = j$ implies $p_{ik} < \gamma p_{ij}$. Hence,

$$a_k + (1+2\delta)p_{ik} \leq (a_j + (1+\delta)p_{ij}) + (1+2\delta)\gamma p_{ij} \leq a_j + (1+2\delta)p_{ij},$$

where the last inequality follows from the assumption $(1+2\delta)\gamma \leq \delta$. Due to the construction of $B(k)$ upon admission of some job k by job j, we also have $B(k) \subseteq S(j)$. ∎

Proof of Theorem 3.1. Let j be a job admitted by the blocking algorithm to machine i with $a_j \leq d_j - (1+\delta)p_{ij}$. Showing that job j completes before time $d'_j := a_j + (1+\delta)p_{ij}$ proves the theorem. Due to scheduling in SPT order, each job j has highest priority in its own scheduling interval if the time point does not belong to the scheduling interval of a descendant of j. Thus, it suffices to show that at most δp_{ij} units of time in $[a_j, d'_j)$ belong to scheduling intervals $S(k)$ of descendants of j. By Lemma 3.2, the scheduling interval of any descendant k' of a child k of j is contained in $S(k)$. Hence, it is sufficient to only consider K, the set of children of j.

In order to bound the contribution of each child $k \in K$, we impose a *class structure* on the jobs in K depending on their size relative to job j. More precisely, we define $(\mathcal{C}_c(j))_{c\in\mathbb{N}_0}$, where $\mathcal{C}_c(j)$ contains all jobs $k \in K$ that satisfy $\frac{\gamma}{2^{c+1}}p_{ij} \leq p_{ik} < \frac{\gamma}{2^c}p_{ij}$. As $k \in K$ implies $p_{ik} < \gamma p_{ij}$, each child of j belongs to exactly one class and $(\mathcal{C}_c(j))_{c\in\mathbb{N}_0}$ in fact partitions K.

Consider two jobs $k, k' \in K$ where, upon admission, k interrupts the blocking period of k'. By definition, we have $p_{ik} < \frac{1}{2}p_{ik'}$. Hence, the chosen class structure ensures that k belongs to a strictly *higher* class than k', i.e., there are $c, c' \in \mathbb{N}$ with $c > c'$ such that $k \in \mathcal{C}_c(j)$ and $k' \in \mathcal{C}_{c'}(j)$. In particular, the admission of a job $k \in \mathcal{C}_c(j)$ implies either that k is the first job of class $\mathcal{C}_c(j)$ that j admits or that the blocking period of the previous job in class $\mathcal{C}_c(j)$ has completed. Based on this distinction, we are able to bound the loss of scheduling time for j in $S(j)$ due to $S(k)$ of a child k. Specifically, we partition K into two sets. The first set K_1 contains all children of j that where admitted as the first jobs in their class $\mathcal{C}_c(j)$. The set K_2 contains the remaining jobs.

We start with K_2. Consider a job $k \in \mathcal{C}_c(j)$ admitted by j. By Lemma 3.2, we know that $|S(k)| = (1+\mu\delta)p_{ik}$, where $1 \leq \mu \leq 2$. Let $k' \in \mathcal{C}_c(j)$ be the previous job admitted by j in class $\mathcal{C}_c(j)$. Then, $B(k') \subseteq [e_{k'}, a_k)$. Since scheduling and blocking periods of children of j are disjoint, j has highest scheduling priority in $B(k')$. Hence, during $B(k') \cup S(k)$ job j is processed for at least $|B(k')|$ units of time. In other words, j is processed for at least a $\frac{|B(k')|}{|B(k')\cup S(k)|}$-fraction of $B(k') \cup S(k)$. We rewrite this ratio as

$$\frac{|B(k')|}{|B(k')\cup S(k)|} = \frac{\beta p_{ik'}}{\beta p_{ik'} + (1+\mu\delta)p_{ik}} = \frac{\nu\beta}{\nu\beta + (1+\mu\delta)},$$

where $\nu := \frac{p_{ik'}}{p_{ik}} \in (\frac{1}{2}, 2]$. By differentiating with respect to ν and μ, we observe that the last term is increasing in ν and decreasing in μ. Thus, we lower bound this expression by

$$\frac{|B(k')|}{|B(k')\cup S(k)|} \geq \frac{\beta/2}{\beta/2 + (1+2\delta)}.$$

Therefore, j is processed for at least a $\frac{\beta/2}{\beta/2+(1+2\delta)}$-fraction in $\bigcup_{k\in K} B(k) \cup \bigcup_{k\in K_2} S(k)$.

We now consider the set K_1. The total processing volume of these jobs is bounded from above by $\sum_{c=0}^{\infty} \frac{\gamma}{2^c} p_{ij} = 2\gamma p_{ij}$. By Lemma 3.2, $|S(k)| \leq (1+2\delta)p_{ik}$. Combining these two observations, we obtain $\left|\bigcup_{k\in K_1} S(k)\right| \leq 2(1+2\delta)\gamma p_{ij}$. Combining the latter with the bound for K_2, we conclude that j is scheduled for at least

$$\left|[a_j, d'_j) \setminus \bigcup_{k\in K} S(k)\right| \geq \frac{\beta/2}{\beta/2 + (1+2\delta)}\big((1+\delta) - 2(1+2\delta)\gamma\big)p_{ij} \geq p_{ij}$$

units of time, where the last inequality follows from Equation (3.1). Therefore, j completes before $d'_j = a_j + (1+\delta)p_{ij} \leq d_j$, which concludes the proof. ∎

4. Competitiveness: admitting sufficiently many jobs. This section shows that the blocking algorithm admits sufficiently many jobs to be $\mathcal{O}\big(\frac{1}{\varepsilon-\delta}\big)$-competitive. As mentioned before, this proof is based on the observation that, at time τ, the blocking algorithm admits any job available for machine i if its processing time is less than γp_{ij_i}, where j_i is the job processed by machine i at time τ, and this time point is not blocked by another job k_i previously admitted by j_i to machine i. We start by formalizing this observation for a class of non-migratory online algorithms before proving that this enables us to bound the number of jobs any feasible schedule successfully schedules during a particular period. Then, we use it to show that the blocking algorithm is indeed $\mathcal{O}\big(\frac{1}{\varepsilon-\delta}\big)$-competitive.

4.1. A class of online algorithms. In this section, we investigate a class of non-migratory online algorithms. Recall that a job j is called available for machine i at time τ if it is released before or at time τ, $d_j - \tau \geq (1+\delta)p_{ij}$, and is not yet admitted.

We consider a non-migratory online algorithm $\mathcal{A}$ with the following properties.

(P1) $\mathcal{A}$ only admits available jobs.

(P2) Retrospectively, for each time τ and each machine i, there is a threshold $u_{i\tau} \in [0,\infty]$ such that any job j that was available for machine i and not admitted to machine i by $\mathcal{A}$ at time τ satisfies $p_{ij} \geq u_{i\tau}$. The function $u^{(i)} : \mathbb{R} \to [0,\infty], \tau \mapsto u_{i\tau}$ is piece-wise constant and right-continuous for every machine $i \in \{1,\ldots,m\}$. Further, there are only countably many points of discontinuity. (This last property is used to simplify the exposition.)

Key lemma on the size of non-admitted jobs. For the proof of the main result in this section, we rely on the following strong, structural lemma about the volume processed by a feasible non-migratory schedule in some time interval and the size of jobs admitted by a non-migratory online algorithm satisfying (P1) and (P2) in the same time interval.

Let σ be a feasible non-migratory schedule. Without loss of generality, we assume that σ completes all jobs that it started on time. Let X^σ be the set of jobs completed by σ and not admitted by $\mathcal{A}$. For $1 \le i \le m$, let X_i^σ be the set of jobs in X^σ processed by machine i. Let C_x be the completion time of job $x \in X^\sigma$ in σ.

LEMMA 4.1. *Let $0 \le \vartheta_1 \le \vartheta_2$ and fix $x \in X_i^\sigma$ as well as $Y \subset X_i^\sigma \setminus \{x\}$. If*

(R) $r_x \ge \vartheta_1$ as well as $r_y \ge \vartheta_1$ for all $y \in Y$,

(C) $C_x \ge C_y$ for all $y \in Y$, and

(P) $\sum_{y \in Y} p_{iy} \ge \frac{\varepsilon}{\varepsilon-\delta}(\vartheta_2 - \vartheta_1)$

hold, then $p_{ix} \ge u_{i\vartheta_2}$, where $u_{i\vartheta_2}$ is the threshold imposed by $\mathcal{A}$ at time ϑ_2. In particular, if $u_{i,\vartheta_2} = \infty$, then no such job x exists.

Proof. We show the lemma by contradiction. More precisely, we show that, if $p_{ix} < u_{i\vartheta_2}$, the schedule σ cannot complete x on time and, hence, is not feasible.

Remember that $x \in X_i^\sigma$ implies that $\mathcal{A}$ did not admit job x at any point ϑ. At time ϑ_2, there are two possible reasons why x was not admitted: $p_{ix} \ge u_{i\vartheta_2}$ or $d_x - \vartheta_2 < (1+\delta)p_{ix}$. In case of the former, the statement of the lemma holds. Toward a contradiction, suppose $p_{ix} < u_{i\vartheta_2}$ and, thus, $d_x - \vartheta_2 < (1+\delta)p_{ix}$ has to hold. As job x arrives with a slack of at least εp_{ix} at its release date r_x and $r_x \ge \vartheta_1$ by assumption, we have

$$\vartheta_2 - \vartheta_1 \ge \vartheta_2 - d_x + d_x - r_x > -(1+\delta)p_{ix} + (1+\varepsilon)p_{ix} = (\varepsilon - \delta)p_{ix}. \tag{4.1}$$

Since all jobs in Y complete earlier than x by Assumption (C) and are only released after ϑ_1 by (R), the volume processed by σ in $[\vartheta_1, C_x)$ on machine i is at least $\frac{\varepsilon}{\varepsilon-\delta}(\vartheta_2 - \vartheta_1) + p_{ix}$ by (P). Moreover, σ can process at most a volume of $(\vartheta_2 - \vartheta_1)$ on machine i in $[\vartheta_1, \vartheta_2)$. These two bounds imply that σ has to process job parts with a processing volume of at least

$$\frac{\varepsilon}{\varepsilon-\delta}(\vartheta_2 - \vartheta_1) + p_{ix} - (\vartheta_2 - \vartheta_1) > \frac{\delta}{\varepsilon-\delta}(\varepsilon-\delta)p_{ix} + p_{ix} = (1+\delta)p_{ix}$$

in $[\vartheta_2, C_x)$, where the inequality follows using Inequality (4.1). Thus, $C_x \ge \vartheta_2 + (1+\delta)p_{ix} > d_x$, which contradicts the feasibility of σ.

Observe that, by (P1) and (P2), the online algorithm $\mathcal{A}$ admits a job available for machine i if it satisfies $p_{ij} < u_{i\tau}$. In particular, if $u_{i\tau} = \infty$ for some time point τ, then $\mathcal{A}$ admits any job available for machine i. Hence, for $0 \le \vartheta_1 \le \vartheta_2$ with $u_{i\vartheta_2} = \infty$, there does not exist a job $x \in X_i^\sigma$ and a set $Y \subset X_i^\sigma \setminus \{x\}$ satisfying (R), (C), and (P) for machine i. ∎

Bounding the number of non-admitted jobs. In this section, we use the Properties (P1) and (P2) to bound the throughput of a non-migratory optimal (offline) algorithm. To this end, we fix an instance as well as an optimal schedule with job set OPT. Let $\mathcal{A}$ be a non-migratory online algorithm satisfying (P1) and (P2).

Let X be the set of jobs in OPT that the algorithm $\mathcal{A}$ did not admit. We assume without loss of generality that all jobs in OPT complete on time. Since OPT as well as $\mathcal{A}$ are non-migratory, we compare the throughput machine-wise. To this end, we fix one machine i. Let $X_i \subset X$ be the set of jobs scheduled on machine i by OPT.

Assumption (P2) guarantees that the threshold $u_{i,\tau}$ is piece-wise constant and right-continuous, i.e., $u^{(i)}$ is constant on intervals of the form $[\tau_t, \tau_{t+1})$. Let $\mathcal{I}$ represent the set of maximal intervals $I_t = [\tau_t, \tau_{t+1})$ where $u^{(i)}$ is constant. That is, $u_{i,\tau} = u_t$ holds for all $\tau \in I_t$ and $u_{i,\tau_{t+1}} \neq u_t$, where $u_t := u_{i,\tau_t}$, The main result of this section is the following theorem.

THEOREM 4.2. *Let X_i be the set of jobs that are scheduled on machine i in the optimal schedule. Let $\mathcal{I} = \{I_1, \dots, I_T\}$ be the set of maximal intervals on machine i of $\mathcal{A}$ such that the machine-dependent threshold is constant for each interval and has the value u_t in interval $I_t = [\tau_t, \tau_{t+1})$. Then,*

$$|X_i| \leq \sum_{t=1}^{T} \frac{\varepsilon}{\varepsilon - \delta} \frac{\tau_{t+1} - \tau_t}{u_t} + T,$$

where we set $\frac{\tau_{t+1}-\tau_t}{u_t} = 0$ if $u_t = \infty$ and $\frac{\tau_{t+1}-\tau_t}{u_t} = \infty$ if $\{\tau_t, \tau_{t+1}\} \cap \{-\infty, \infty\} \neq \emptyset$ and $u_t < \infty$.

We observe that $T = \infty$ trivially proves the statement as X_i contains at most finitely many jobs. The same is true if $\frac{\tau_{t+1}-\tau_t}{u_t} = \infty$ for some $t \in [T]$. Hence, for the remainder of this section we assume without loss of generality that $\mathcal{I}$ only contains finitely many intervals and that $\frac{\tau_{t+1}-\tau_t}{u_t} < \infty$ holds for every $t \in [T]$.

To prove this theorem, we develop a charging scheme that assigns jobs $x \in X_i$ to intervals in $\mathcal{I}$. The idea behind our charging scheme is that OPT does not contain arbitrarily many jobs that are available in I_t since u_t provides a natural lower bound on their processing times. In particular, the processing time of any job that is *released* during interval I_t and not admitted by the algorithm exceeds the lower bound u_t. Thus, the charging scheme relies on the release date r_x and the size p_{ix} of a job $x \in X_i$ as well as on the precise structure of the intervals created by $\mathcal{A}$.

The charging scheme we develop is based on a careful modification of the following partition $(F_t)_{t=1}^T$ of the set X_i. Fix an interval $I_t \in \mathcal{I}$ and define the set $F_t \subseteq X_i$ as the set that contains all jobs $x \in X_i$ released during I_t, i.e., $F_t = \{x \in X_i : r_x \in I_t\}$. Since, upon release, each job $x \in X_i$ is available and not admitted by $\mathcal{A}$, the next fact directly follows from Properties (P1) and (P2).

FACT 4.3. *For all jobs $x \in F_t$ it holds $p_{ix} \geq u_t$. In particular, if $u_t = \infty$, then $F_t = \emptyset$.*

In fact, the charging scheme maintains this property and only assigns jobs in X_i to intervals I_t if $p_{ix} \geq u_t$. In particular, no job will be assigned to an interval with $u_t = \infty$.

We now formalize how many jobs in X_i are assigned to a specific interval I_t. Let

$$\varphi_t := \left\lfloor \frac{\varepsilon}{\varepsilon - \delta} \frac{\tau_{t+1} - \tau_t}{u_t} \right\rfloor + 1$$

if $u_t < \infty$, and $\varphi_t = 0$ if $u_t = \infty$. We refer to φ_t as the *target number* of I_t. As discussed before, we assume $\frac{\tau_{t+1}-\tau_t}{u_t} < \infty$, and, thus, the target number is well-defined. If each of the sets F_t satisfies $|F_t| \leq \varphi_t$, then Theorem 4.2 immediately follows. In general, $|F_t| \leq \varphi_t$ does not have to be true since jobs in OPT may be preempted and processed during several intervals I_t. Therefore, for proving Theorem 4.2, we show that there always exists another partition $(G_t)_{t=1}^T$ of X_i such that $|G_t| \leq \varphi_t$ holds.

The high-level idea of this proof is the following: Consider an interval $I_t = [\tau_t, \tau_{t+1})$. If F_t does not contain too many jobs, i.e., $|F_t| \leq \varphi_t$, we would like to set $G_t = F_t$. Otherwise, we find a later interval $I_{t'}$ with $|F_{t'}| < \varphi_{t'}$ such that we can assign the excess jobs in F_t to $I_{t'}$.

Proof of Theorem 4.2. As observed before, it suffices to show the existence of a partition $\mathcal{G} = (G_t)_{t=1}^T$ of X_i such that $|G_t| \le \varphi_t$ in order to prove the theorem.

In order to repeatedly apply Lemma 4.1, we only assign excess jobs $x \in F_t$ to $G_{t'}$ for $t < t'$ if their processing time is at least the threshold of $I_{t'}$, i.e., $p_{ix} \ge u_{t'}$. By our choice of parameters, a set $G_{t'}$ with $\varphi_{t'}$ many jobs of size at least $u_{t'}$ "covers" the interval $I_{t'} = [\tau_{t'}, \tau_{t'+1})$ as often as required by (P) in Lemma 4.1, i.e.,

$$\sum_{x \in G_{t'}} p_{ix} \ge \varphi_{t'} \cdot u_{t'} = \left(\left\lfloor \frac{\varepsilon}{\varepsilon - \delta} \frac{\tau_{t'+1} - \tau_{t'}}{u_{t'}} \right\rfloor + 1 \right) \cdot u_{t'} \ge \frac{\varepsilon}{\varepsilon - \delta} (\tau_{t'+1} - \tau_{t'}). \tag{4.2}$$

The proof consists of two parts: the first one is to inductively (on t) construct the partition $\mathcal{G} = (G_t)_{t=1}^T$ of X_i, where $|G_t| \le \varphi_t$ holds for $t \in [T-1]$. The second one is the proof that a job $x \in G_t$ satisfies $p_{ix} \ge u_t$ which will imply $|G_T| \le \varphi_T$. During the construction of $\mathcal{G}$ we define temporary sets $A_t \subset X_i$ for intervals I_t. The set G_t is chosen as a subset of $F_t \cup A_t$ of appropriate size. In order to apply Lemma 4.1 to each job in A_t individually, alongside A_t, we construct a set $Y_{x,t}$ and a time $\tau_{x,t} \le r_x$ for each job $x \in X_i$ that is added to A_t. Let C_y^* be the completion time of some job $y \in X_i$ in the optimal schedule OPT. The second part of the proof is to show that x, $\tau_{x,t}$, and $Y_{x,t}$ satisfy

(R) $r_y \ge \tau_{x,t}$ for all $y \in Y_{x,t}$,
(C) $C_x^* \ge C_y^*$ for all $y \in Y_{x,t}$, and
(P) $\sum_{y \in Y_{x,t}} p_{iy} \ge \frac{\varepsilon}{\varepsilon - \delta} (\tau_t - \tau_{x,t})$.

This implies that x, $Y = Y_{x,t}$, $\vartheta_1 = \tau_{x,t}$, and $\vartheta_2 = \tau_t$ satisfy the conditions of Lemma 4.1, and thus the processing time of x is at least the threshold at time τ_t, i.e., $p_{ix} \ge u_{i\tau_t} = u_t$.

Constructing $G = (G_t)_{t=1}^T$. We inductively construct the sets G_t in the order of their indices. We start by setting $A_t = \emptyset$ for all intervals I_t with $t \in T$. We define $Y_{x,t} = \emptyset$ for each job $x \in X_i$ and each interval I_t. The preliminary value of the time $\tau_{x,t}$ is the minimum of the starting point τ_t of the interval I_t and the release date r_x of x, i.e., $\tau_{x,t} := \min\{\tau_t, r_x\}$. We refer to the step in the construction where G_t was defined by *step t*.

Starting with $t = 1$, let I_t be the next interval to consider during the construction with $t < T$. Depending on the cardinality of $F_t \cup A_t$, we distinguish two cases. If $|F_t \cup A_t| \le \varphi_t$, then we set $G_t = F_t \cup A_t$.

If $|F_t \cup A_t| > \varphi_t$, then we order the jobs in $F_t \cup A_t$ in increasing order of completion times in the optimal schedule. The first φ_t jobs are assigned to G_t while the remaining $|F_t \cup A_t| - \varphi_t$ jobs are added to A_{t+1}. In this case, we might have to redefine the times $\tau_{x,t+1}$ and the sets $Y_{x,t+1}$ for the jobs x that were newly added to A_{t+1}. Fix such a job x. If there is no job z in the just defined set G_t that has a smaller release date than $\tau_{x,t}$, we set $\tau_{x,t+1} = \tau_{x,t}$ and $Y_{x,t+1} = Y_{x,t} \cup G_t$. Otherwise let $z \in G_t$ be a job with $r_z < \tau_{x,t}$ that has the smallest time $\tau_{z,t}$. We set $\tau_{x,t+1} = \tau_{z,t}$ and $Y_{x,t+1} = Y_{z,t} \cup G_t$.

Finally, we set $G_T = F_T \cup A_T$. We observe that $u_T < \infty$ implies $\varphi_T = \infty$ because $\tau_{T+1} = \infty$. Since this contradicts the assumption $\varphi_t < \infty$ for all $t \in [T]$, this implies $u_T = \infty$. We will show that $p_x \ge u_T$ for all $x \in G_T$. Hence, $G_T = \emptyset$. Therefore $|G_T| = \varphi_T = 0$.

Bounding the size of jobs in G_t. We consider the intervals again in increasing order of their indices and show by induction that any job x in G_t satisfies $p_{ix} \ge u_t$ which implies $G_t = \emptyset$ if $u_t = \infty$. Clearly, if $x \in F_t \cap G_t$, Fact 4.3 guarantees $p_{ix} \ge u_t$. Hence, in order to show the lower bound on the processing time of $x \in G_t$, it is sufficient to consider jobs in $G_t \setminus F_t \subset A_t$. To this end, we show that for such jobs (R), (C), and (P) are satisfied. Thus, Lemma 4.1 guarantees that $p_{ix} \geq u_{i\tau_t} = u_t$ by definition. Hence, $A_t = \emptyset$ if $u_t = \infty$ by Lemma 4.1.

By construction, $A_1 = \emptyset$. Hence, (R), (C), and (P) are satisfied for each job $x \in A_1$.

Suppose that the Conditions (R), (C), and (P) are satisfied for all $x \in A_s$ for all $1 \leq s < t$. Hence, for $s < t$, the set G_s only contains jobs x with $p_{ix} \geq u_s$. Fix $x \in A_t$. We want to show that $p_{ix} \geq u_t$. By the induction hypothesis and by Fact 4.3, $p_{iy} \geq u_{t-1}$ holds for all $y \in G_{t-1}$. Since x did not fit in G_{t-1} anymore, $|G_{t-1}| = \varphi_{t-1}$.

We distinguish two cases based on G_{t-1}. If there is no job $z \in G_{t-1}$ with $r_z < \tau_{x,t-1}$, then $\tau_{x,t} = \tau_{x,t-1}$, and (R) and (C) are satisfied by construction and by the induction hypothesis. For (P), consider

$$\begin{aligned}
\sum_{y \in Y_{x,t}} p_{iy} &= \sum_{y \in Y_{x,t-1}} p_{iy} + \sum_{y \in G_{t-1}} p_{iy} \\
&\geq \frac{\varepsilon}{\varepsilon - \delta}(\tau_{t-1} - \tau_{x,t-1}) + u_{t-1} \cdot \varphi_{t-1} \\
&\geq \frac{\varepsilon}{\varepsilon - \delta}(\tau_{t-1} - \tau_{x,t-1}) + \frac{\varepsilon}{\varepsilon - \delta}(\tau_t - \tau_{t-1}) \\
&= \frac{\varepsilon}{\varepsilon - \delta}(\tau_t - \tau_{x,t}),
\end{aligned}$$

where the first inequality holds due to the induction hypothesis. By Lemma 4.1, $p_{ix} \geq u_{\tau_t} = u_t$.

If there is a job $z \in G_{t-1}$ with $r_z < \tau_{x,t-1} \leq \tau_{t-1}$, then $z \in A_{t-1}$. In step $t-1$, we chose z with minimal $\tau_{z,t-1}$. Thus, $r_y \geq \tau_{y,t-1} \geq \tau_{z,t-1}$ for all $y \in G_{t-1}$ and $r_x \geq \tau_{x,t-1} > r_z \geq \tau_{z,t-1}$ which is Condition (R) for the jobs in G_{t-1}. Moreover, by the induction hypothesis, $r_y \geq \tau_{z,t-1}$ holds for all $y \in Y_{z,t-1}$. Thus, $\tau_{x,t}$ and $Y_{x,t}$ satisfy (R). For (C), consider that $C^*_x \geq C^*_y$ for all $y \in G_{t-1}$ by construction and, thus, $C^*_x \geq C^*_z \geq C^*_y$ also holds for all $y \in Y_{z,t-1}$ due to the induction hypothesis. For (P), observe that

$$\begin{aligned}
\sum_{y \in Y_{x,t}} p_{iy} &= \sum_{y \in Y_{z,t-1}} p_{iy} + \sum_{y \in G_{t-1}} p_{iy} \\
&\geq \frac{\varepsilon}{\varepsilon - \delta}(\tau_{t-1} - \tau_{z,t-1}) + u_{t-1} \cdot \varphi_{t-1} \\
&\geq \frac{\varepsilon}{\varepsilon - \delta}(\tau_{t-1} - \tau_{z,t-1}) + \frac{\varepsilon}{\varepsilon - \delta}(\tau_t - \tau_{t-1}) \\
&\geq \frac{\varepsilon}{\varepsilon - \delta}(\tau_t - \tau_{x,t}).
\end{aligned}$$

Here, the first inequality follows from the induction hypothesis and the second from the definition of u_{t-1} and φ_{t-1}. Hence, Lemma 4.1 implies $p_{ix} \geq u_{\tau_t} = u_t$.

We note that $p_{ix} \geq u_t$ for all $x \in G_t$ and for all $t \in [T]$.

Bounding $|X_i|$. By construction, we know that $\bigcup_{t=1}^{T} G_t = X_i$. We start with considering intervals I_t with $u_t = \infty$. Then, I_t has an unbounded threshold, i.e., $u_{i\tau} = \infty$ for all $\tau \in I_t$, and $F_t = \emptyset$ by Fact 4.3. In the previous part we have seen that the conditions for Lemma 4.1 are satisfied. Hence, $G_t = \emptyset$ if $u_t = \infty$. For t with $u_t < \infty$, we have $|G_t| \leq \varphi_t = \left\lfloor \frac{\varepsilon}{\varepsilon - \delta} \frac{\tau_{t+1} - \tau_t}{u_t} \right\rfloor + 1$. As explained before, this bounds the number of jobs in X_i. ∎

4.2. The blocking algorithm admits sufficiently many jobs. Having the powerful tool that we developed in the previous section at hand, it remains to show that the blocking algorithm admits sufficiently many jobs to achieve the competitive ratio of $\mathcal{O}\big(\frac{1}{\varepsilon-\delta'}\big)$ where $\delta' = \frac{\varepsilon}{2}$ for commitment upon admission and $\delta' = \max\left\{\frac{\varepsilon}{2}, \delta\right\}$ for δ-commitment. To this end, we show that the blocking algorithm belongs to the class of online algorithms considered in Subsection 4.1. Then, Theorem 4.2 provides a bound on the throughput of an optimal non-migratory schedule.

We begin by showing that the blocking algorithm satisfies Properties (P1) to (P2). The first property is clearly satisfied by the definition of the blocking algorithm. For the second and the third property, we observe that a new job $j^\star$ is only admitted to a machine i during the scheduling interval of another job j admitted to the same machine if $p_{ij^\star} < \gamma p_{ij}$. Further, the time point of admission must not be blocked by a similar- or smaller-size job k previously admitted during the scheduling interval of j. This leads to the bound $p_{ij^\star} < \frac{1}{2}p_{ik}$ for any job k whose blocking period contains the current time point. Combining these observations leads to a machine-dependent threshold $u_{i,\tau} \in [0,\infty]$ satisfying (P2).

More precisely, fix a machine i and a time point τ. Using $j \to i$ to denote that j was admitted to machine i, we define $u_{i,\tau} := \min_{j\colon j\to i, \tau\in S(j)} \gamma p_{ij}$ if there is no job k admitted to machine i with $\tau \in B(k)$, with $\min\emptyset = \infty$. Otherwise, we set $u_{i,\tau} := \frac{1}{2}p_{ik}$. We note that the function $u^{(i)}$ is piece-wise constant and right-continuous due to our choice of right-open intervals for defining scheduling intervals and blocking periods. Moreover, the points of discontinuity of $u^{(i)}$ correspond to the admission of a new job, the end of a scheduling interval, and the start as well as the end of a blocking period of jobs admitted to machine i. Since we only consider instances with a finite number of jobs, there are at most finitely many points of discontinuity of $u^{(i)}$. Hence, we can indeed apply Theorem 4.2.

Then, the following theorem is the main result of this section.

THEOREM 4.4. *An optimal non-migratory (offline) algorithm can complete at most a factor $\alpha + 5$ more jobs on time than admitted by the blocking algorithm, where $\alpha := \frac{\varepsilon}{\varepsilon-\delta}\big(2\beta + \frac{1+2\delta}{\gamma}\big)$.*

Proof. We fix an instance and an optimal solution OPT. We use X to denote the set of jobs in OPT that the blocking algorithm did not admit. Without loss of generality, we can assume that all jobs in OPT complete on time. If J is the set of jobs admitted by the blocking algorithm, then $X \cup J$ is a superset of the jobs successfully finished in the optimal solution. Hence, showing $|X| \leq (\alpha+4)|J|$ suffices to prove Theorem 4.4.

For each machine i, we compare the throughput of the optimal solution to the throughput on machine i of the blocking algorithm. More precisely, let $X_i \subseteq X$ be the jobs in OPT scheduled on machine i and let $J_i \subseteq J$ be the jobs scheduled by the blocking algorithm on machine i. With Theorem 4.2, we show $|X_i| \leq (\alpha+4)|J_i|$ to bound the cardinality of X in terms of $|J|$.

To this end, we retrospectively consider the interval structure created by the algorithm on machine i. Let $\mathcal{I}$ be the set of maximal intervals $I_t = [\tau_t, \tau_{t+1})$ such that $u_{i,\tau} = u_{i,\tau_t}$ for all $\tau \in I_t$. We define $u_t = u_{i,\tau_t}$ for each interval I_t. As discussed above, the time points τ_t for $t \in [T]$ correspond to the admission, the end of a scheduling interval, and the start as well as the end of a blocking period of jobs admitted to machine 1. As the admission of a job adds at most three time points, we have that $|\mathcal{I}| \leq 3|J_i| + 1$.

As the blocking algorithm satisfies Properties (P1) to (P2), we can apply Theorem 4.2 to obtain

$$|X_i| \leq \sum_{t=1}^{T} \frac{\varepsilon}{\varepsilon - \delta} \frac{\tau_{t+1} - \tau_t}{u_t} + |\mathcal{I}| \leq \sum_{t=1}^{T} \frac{\varepsilon}{\varepsilon - \delta} \frac{\tau_{t+1} - \tau_t}{u_t} + (3|J_i| + 1).$$

It remains to bound the first part in terms of $|J_i|$. If $u_t < \infty$, let $j_t \in J_i$ be the *smallest* job j with $\tau_t \in S(j) \cup B(j)$. Then, at most $\frac{\varepsilon}{\varepsilon-\delta}\frac{\tau_{t+1}-\tau_t}{u_t}$ (potentially fractional) jobs will be charged to job j_t because of interval I_t. By definition of u_t, we have $u_t = \gamma p_{ij_t}$ if $I_t \subseteq S(j_t)$, and if $I_t \subseteq B(j_t)$, we have $u_t = \frac{1}{2} p_{ij_t}$. The total length of intervals I_t for which $j = j_t$ holds sums up to at most $(1 + 2\delta)p_{ij}$ for $I_t \subseteq S(j)$ and to at most $2\beta p_{ij}$ for $I_t \subseteq B(j)$. Hence, in total, the charging scheme assigns at most $\frac{\varepsilon}{\varepsilon-\delta}(2\beta + \frac{1+2\delta}{\gamma}) = \alpha$ jobs in X_i to job $j \in J_i$. Therefore,

$$|X_i| \leq \big(\alpha + 3\big)|J_i| + 1.$$

If $J_i = \emptyset$, the blocking algorithm admitted all jobs scheduled on machine i by OPT, and $|X_i| = 0 = |J_i|$ follows. Otherwise, $|X_i| \leq \big(\alpha + 4\big)|J_i|$, and we obtain

$$|\mathrm{OPT}| \leq |X \cup J| = \sum_{i=1}^{m} |X_i| + |J| \leq \sum_{i=1}^{m} (\alpha + 4)|J_i| + |J| \leq (\alpha + 5)|J|,$$

which concludes the proof. ∎

4.3. Finalizing the proof of Theorem 1.1.

Proof of Theorem 1.1. In Theorem 3.1 we show that the blocking algorithm completes all admitted jobs J on time. This implies that the blocking algorithm is feasible for the model commitment upon admission. As no job $j \in J$ is admitted later than $d_j - (1 + \delta)p_{ij}$, the blocking algorithm also solves scheduling with δ-commitment. In Theorem 4.4, we bound the throughput $|\mathrm{OPT}|$ of an optimal non-migratory solution by $\alpha + 5$ times $|J|$, the throughput of the blocking algorithm, where $\alpha = \frac{\epsilon}{\varepsilon-\delta}(2\beta + \frac{1+2\delta}{\gamma})$. Our choice of parameters $\beta = \frac{16}{\delta}$ and $\gamma = \frac{\delta}{16}$ implies that the blocking algorithm achieves a competitive ratio of $c \in \mathcal{O}\big(\frac{\varepsilon}{(\varepsilon-\delta)\delta}\big)$. For commitment upon arrival or for δ-commitment in the case where $\delta \leq \frac{\varepsilon}{2}$, we run the algorithm with $\delta' = \frac{\varepsilon}{2}$. Hence, $c \in \mathcal{O}\big(\frac{1}{\varepsilon-\delta'}\big) = \mathcal{O}\big(\frac{1}{\varepsilon}\big)$. If $\delta > \frac{\varepsilon}{2}$, then we set $\delta' = \delta$ in our algorithm. Thus, $\frac{\varepsilon}{\delta'} \in \mathcal{O}(1)$ and, again, $c \in \mathcal{O}\big(\frac{1}{\varepsilon-\delta'}\big)$. ∎

5. Scheduling without commitment. This section considers online throughput maximization without commitment requirements. We show how to exploit also in this setting our key lemma on the size of non-admitted jobs for a big class of online algorithms and the resulting upper bound on the throughput of an optimal (offline) non-migratory algorithm from Subsection 4.1.

We consider the *region algorithm* that was designed by [10] for scheduling on a single machine and we generalize it to parallel identical machines. We prove that it has a competitive ratio of $\mathcal{O}\big(\frac{1}{\varepsilon}\big)$, which is best possible on a single machine and improves substantially upon the best previously known parallel-machine algorithm (for weighted throughput) with a competitive ratio of $\mathcal{O}\big(\frac{1}{\varepsilon^2}\big)$ by Lucier et al. [31]. For a single machine, this matches the guarantee proven in [10]. However, our new analysis is much more direct.

5.1. The region algorithm. Originally, the region algorithm was designed for online scheduling with and without commitment on a single machine. We extend it to unrelated machines by never migrating jobs between machines and per machine using the same design principles that guide the admission decisions of the region algorithm, as developed in [10]. Since we do not consider commitment in this section, we can significantly simplify the exposition of the region algorithm when compared to [10].

As in the previous section, a job is only processed by the machine it initially was started on. We say the job has been *admitted* to this machine. Moreover, a running job can only be preempted by significantly smaller-size jobs, i.e., smaller by a factor of at least $\frac{\varepsilon}{4}$ with respect to the processing time, and a job j cannot start for the first time on machine i when its remaining slack is too small, i.e., less than $\frac{\varepsilon}{2}p_{ij}$.

Formally, at any time τ, the region algorithm maintains two sets of jobs: *admitted jobs*, which have been started before or at time τ, and *available jobs*. A job j is available for machine i if it is released before or at time τ, is not yet admitted, and τ is not too close to its deadline, i.e., $r_j \leq \tau$ and $d_j - \tau \geq \big(1 + \frac{\varepsilon}{2}\big)p_{ij}$. The intelligence of the region algorithm lies in how it admits jobs. The actual scheduling decision then is simple and independent of the admission of jobs: at any point in time and on each machine, schedule the shortest job that has been admitted to this machine and has not completed its processing time. In other words, we schedule admitted jobs on each machine in Shortest Processing Time (SPT) order. The region algorithm never explicitly considers deadlines except when deciding whether to admit jobs. In particular, jobs can even be processed after their deadline.

At any time τ, when there is a job j available for an *idle* machine i, i.e., i is not processing any previously admitted job j', the shortest available job $j^\star$ is immediately admitted to machine i at time $a_j^\star := \tau$. There are two events that trigger a decision of the region algorithm: the release of a job and the completion of a job. If one of these events occurs at time τ, the region algorithm invokes the preemption subroutine. This routine iterates over all machines and compares the processing time of the smallest job $j^\star$ *available* for machine i with the processing time of job j_i that is currently scheduled on machine i. If $p_{ij^\star} < \frac{\varepsilon}{4}p_{ij_i}$, job $j^\star$ is admitted to machine i at time $a_j^\star := \tau$ and, by the above scheduling routine, immediately starts processing. We summarize the region algorithm below.

The proof of the analysis splits again naturally into two parts: The first part is to show that the region algorithm completes at least half of all admitted jobs, and the second is to use Theorem 4.4 to compare the number of admitted jobs to the throughput of an optimal non-migratory algorithm.

5.2. Completing sufficiently many admitted jobs. The main result of this section is the following theorem.

Theorem 5.1. *Let $0 < \varepsilon \leq 1$. Then the region algorithm completes at least half of all admitted jobs before their deadline.*

The proof of Theorem 5.1 relies on two technical results that enable us to restrict to instances with one machine and further only consider jobs that are admitted by the region algorithm in this instance. Then, we can use the analysis of the region algorithm in [10] to complete the proof.

We start with the following observation. Let $\mathcal{I}$ be an instance of online throughput maximization with the job set $\mathcal{J}$ and let $J \subseteq \mathcal{J}$ be the set of jobs admitted by the region algorithm at some point. It is easy to see that a job $j \notin J$ does not influence the scheduling or admission decisions of the region algorithm. The next lemma formalizes this statement and follows immediately from the just made observations.

Lemma 5.2. *For any instance $\mathcal{I}$ for which the region algorithm admits the job*

Algorithm Region algorithm

Scheduling Routine: At any time τ and on any machine i, run the job with shortest processing time that has been admitted to i and has not yet completed.

Event: Release of a new job at time τ
  Call Threshold Preemption Routine.

Event: Completion of a job at time τ
  Call threshold preemption routine.

Threshold Preemption Routine:
$i \leftarrow 1$
$j^\star \leftarrow$ a shortest job available for machine i at τ, i.e., $j^\star \in \arg\min\{p_{ij} \mid j \in \mathcal{J}, r_j \leq \tau \text{ and } d_j - \tau \geq (1+\frac{\varepsilon}{2})p_{ij}\}$
while $i \leq m$ **do**
  $j \leftarrow$ job processed on machine i at time τ
  if $j = \emptyset$ **then**
    admit job $j^\star$ to machine i
    call Threshold Preemption Routine
  else if $p_{ij^\star} < \frac{\varepsilon}{4}p_{ij}$ **then**
    admit job $j^\star$ to machine i
    call Threshold Preemption Routine
  else
    $i \leftarrow i+1$
    $j^\star \leftarrow$ a shortest job available for machine i at τ, i.e., $j^\star \in \arg\min\{p_{ij} \mid j \in \mathcal{J}, r_j \leq \tau \text{ and } d_j - \tau \geq (1+\frac{\varepsilon}{2})p_{ij}\}$
  end if
end while

set $J \subseteq \mathcal{J}$, the reduced instance $\mathcal{I}'$ containing only the jobs J forces the region algorithm with consistent tie breaking to admit all jobs in J and to create the same schedule as produced for the instance $\mathcal{I}$.

The proof of the main result compares the number of jobs finished on time, $F \subseteq J$, to the number of jobs unfinished by their respective deadlines, $U = J \setminus F$. To further simplify the instance, we use that the region algorithm is non-migratory and restrict to single-machine instances. To this end, let $F^{(i)}$ and $U^{(i)}$ denote the finished and unfinished, respectively, jobs on machine i.

LEMMA 5.3. *Let $i \in \{1,\ldots,m\}$. There is an instance $\mathcal{I}'$ on one machine with job set $\mathcal{J}' = F^{(i)} \cup U^{(i)}$. Moreover, the schedule of the region algorithm for instance $\mathcal{I}'$ with consistent tie breaking is identical to the schedule of the jobs $\mathcal{J}'$ on machine i. In particular, $F' = F^{(i)}$ and $U' = U^{(i)}$.*

Proof. By Lemma 5.2, we can restrict to the jobs admitted by the region algorithm. Hence, let $\mathcal{I}$ be such an instance with $F^{(i)} \cup U^{(i)}$ being admitted to machine i. As the region algorithm is non-migratory, the sets of jobs scheduled on two different machines are disjoint. Let $\mathcal{I}'$ consist of the jobs in $\mathcal{J}' := F^{(i)} \cup U^{(i)}$ and one machine. We set $p'_j = p_{ij}$ for $j \in \mathcal{J}'$. The region algorithm on instance $\mathcal{I}$ admits all jobs in $\mathcal{J}$. In particular, it admits all jobs in $\mathcal{J}'$ to machine i.

We inductively show that the schedule for the instance $\mathcal{I}'$ is identical to the schedule on machine i in instance $\mathcal{I}$. To this end, we index the jobs in $\mathcal{J}'$ in increasing admission time points in instance $\mathcal{I}$.

It is obvious that job $1 \in \mathcal{J}'$ is admitted to the single machine at its release date r_1 as happens in instance $\mathcal{I}$ since the region algorithm uses consistent tie breaking. Suppose that the schedule is identical until the admission of job $j^\star$ at time $a_j^\star = \tau$. If $j^\star$ does not interrupt the processing of another job, then $j^\star$ will be admitted at time τ in $\mathcal{I}'$ as well. Otherwise, let $j \in \mathcal{J}'$ be the job that the region algorithm planned to process at time τ *before* the admission of job $j^\star$. Since $j^\star$ is admitted at time τ in $\mathcal{I}$, $j^\star$ is available at time τ, satisfies $p'_{j^\star} = p_{ij^\star} < \frac{\varepsilon}{4} p_{ij} = \frac{\varepsilon}{4} p'_j$, and did not satisfy both conditions at some earlier time τ' with some earlier admitted job j'. Since the job set in $\mathcal{I}'$ is a subset of the jobs in $\mathcal{I}$ and we use consistent tie breaking, no other job $j^* \in \mathcal{J}'$ that satisfies both conditions is favored by the region algorithm over $j^\star$. Therefore, job $j^\star$ is also admitted at time τ by the region algorithm in instance $\mathcal{I}'$. Thus, the schedule created by the region algorithm for $\mathcal{J}'$ is identical to the schedule of $\mathcal{J}$ on machine i in the original instance. ∎

For proving Theorem 5.1, we consider a worst-case instance for the region algorithm where "worst" is with respect to the ratio between admitted and successfully completed jobs. Since the region algorithm is non-migratory, there exists at least one machine in such a worst-case instance that "achieves" the same ratio as the whole instance. By the just proven lemma, we can find a worst-case instance on a single machine. However, on a single machine, the region algorithm algorithm in this paper is identical to the algorithm designed in [10]. Therefore, we simply follow the line of proof developed in [10] to show Theorem 5.1.

More precisely, in [10] we show that the existence of a late job j implies that the the set of jobs admitted by j or by one of its children contains more finished than unfinished jobs. Let F_j denote the set of jobs admitted by j or by one of its children that *finish on time*. Similarly, we denote the set of such jobs that complete after their deadlines, i.e., that are *unfinished at their deadline*, by U_j. We restate the following lemma, which was originally shown in a single-machine environment but clearly also holds for unrelated machines.

Lemma 5.4 (Lemma 3 in [10]). *Consider some job j admitted to some machine $i \in \{1, \dots, m\}$. If $C_j - a_j \geq (\ell + 1)p_{ij}$ for $\ell > 0$, then $|F_j| - |U_j| \geq \lfloor \frac{4\ell}{\varepsilon} \rfloor$.*

Proof of Theorem 5.1. Let U be the set of jobs that are unfinished by their deadline but whose ancestors have all completed on time. Every job $j \in U$ was admitted by the algorithm at some time a_j with $d_j - a_j \geq \left(1 + \frac{\varepsilon}{2}\right)p_{ij}$. Since j is unfinished, we have $C_j - a_j > d_j - a_j \geq \left(1 + \frac{\varepsilon}{2}\right)p_{ij}$. By Lemma 5.4, $|F_j| - |U_j| \geq \left\lfloor \frac{4 \cdot \varepsilon/2}{\varepsilon} \right\rfloor = 2$. Thus,

$$|F_j| + |U_j| \leq 2|F_j| - 2 < 2|F_j|.$$

Since every ancestor of such a job j finishes on time, this completes the proof. ∎

5.3. The region algorithm admits sufficiently many jobs. In this section, we show the following theorem and give the proof of Theorem 1.4.

Theorem 5.5. *An optimal non-migratory (offline) algorithm completes at most a factor $\left(\frac{8}{\varepsilon} + 4\right)$ more jobs on time than admitted by the region algorithm.*

Proof. As in the previous section, fix an instance and an optimal solution Opt. Let X be the set of jobs in Opt that the region algorithm did not admit. We assume without loss of generality that all jobs in Opt finish on time. Further, let J denote the set of jobs that the region algorithm admitted. Then, $X \cup J$ is a superset of the jobs in Opt. Thus, $|X| \leq \left(\frac{8}{\varepsilon} + 3\right)|J|$ implies Theorem 5.5.

Consider an arbitrary but fixed machine i. We compare again the throughput of the optimal schedule on machine i to the throughput of the region algorithm on machine i. Let $X_i \subseteq X$ denote the jobs in OPT scheduled on machine i and let J_i denote the jobs scheduled by the region algorithm on machine i. Then, showing $|X_i| \leq \big(\frac{8}{\varepsilon} + 3\big)|J_i|$ suffices to prove the main result of this section. Given that the region algorithm satisfies Properties (P1) and (P2), Theorem 4.2 already provides a bound on the cardinality of X_i in terms of the *intervals* corresponding to the schedule on amchine i. Thus, it remains to show that the region algorithm indeed qualifies for applying Theorem 4.2 and that the bound developed therein can be translated to a bound in terms of $|J_i|$.

We start by showing that the region algorithm satisfies the assumptions necessary for applying Theorem 4.2. Clearly, as the region algorithm only admits a job j at time τ if $d_j - \tau \geq \big(1 + \frac{\varepsilon}{2}\big)p_{ij}$, setting $\delta = \frac{\varepsilon}{2}$ proves that the region algorithm satisfies (P1). For (P2), we retrospectively analyze the schedule generated by the region algorithm. For a time τ, let j_i denote the job scheduled on machine i. Then, setting $u_{i,\tau} := \frac{\varepsilon}{4}p_{ij_i}$ or $u_{i,\tau} = \infty$ if no such job j_i exists, indeed provides us with the machine-dependent threshold necessary for (P2). This discussion also implies that $u^{(i)}$ has only countably many points of discontinuity as there are only finitely many jobs in the instance, and that $u^{(i)}$ is right-continuous.

Hence, let $\mathcal{I}$ denote the set of maximal intervals $I_t = [\tau_t, \tau_{t+1})$ for $t \in [T]$ of constant threshold $u_{i\tau}$. Thus, by Theorem 4.2,

$$|X_i| \leq \sum_{t=1}^{T} \frac{\varepsilon}{\varepsilon - \delta} \frac{\tau_{t+1} - \tau_t}{u_t} + T. \tag{5.1}$$

As the threshold $u_{i,\tau}$ is proportional to the processing time of the job currently scheduled on machine i, the interval I_t either represents an idle interval of machine i (with $u_{i\tau} = \infty$) or corresponds to the uninterrupted processing of some job j on machine i. We denote this job by j_t if it exists. We consider now the set $\mathcal{I}_j \subseteq \mathcal{I}$ of intervals with $j_t = j$ for some particular job $j \in J_i$. As observed, these intervals correspond to job j being processed which happens for a total of p_{ij} units of time. Combining with $u_t = \frac{\varepsilon}{4}p_{ij}$ for $I_t \in \mathcal{I}_j$, we get

$$\sum_{t:I_t \in \mathcal{I}_j} \frac{\tau_{t+1} - \tau_t}{u_t} = \frac{p_{ij}}{\frac{\varepsilon}{4}p_{ij}} = \frac{4}{\varepsilon}.$$

As $\delta = \frac{\varepsilon}{2}$, we additionally have that $\frac{\varepsilon}{\varepsilon - \delta} = 2$. Hence, we rewrite Equation (5.1) by

$$|X_i| \leq \frac{8}{\varepsilon}|J_i| + T.$$

It remains to bound T in terms of $|J_i|$ to conclude the proof. To this end, we recall that the admission of a job j to a machine interrupts the processing of at most one previously admitted job. Hence, the admission of $|J_i|$ jobs to machine 1 creates at most $2|J_i| + 1$ intervals.

If the region algorithm does not admit any job to machine i, i.e., $|J_i| = 0$, then $u_{i\tau} = \infty$ for each time point τ. Hence, there exists no job scheduled on machine i by OPT that the region algorithm did not admit. In other words, $X_i = \emptyset$ and $|X_i| = 0 = |J_i|$. Otherwise, $2|J_i| + 1 \leq 3|J_i|$. Therefore,

$$|X_i| \leq \left(\frac{8}{\varepsilon} + 3\right)|J_i|.$$

Combining with the observation about X_i and J_i previously discussed, we obtain

$$|\textsc{Opt}| \leq |X \cup J| = \sum_{i=1}^{m} |X_i| + |J| \leq \left(\frac{8}{\varepsilon} + 3\right) \sum_{i=1}^{m} |J_i| + |J| = \left(\frac{8}{\varepsilon} + 4\right) |J|,$$

which concludes the proof. □

5.4. Finalizing the proof of Theorem 1.4.

Proof of Theorem 1.4. In Theorem 5.1 we show that the region algorithm completes at least half of all admitted jobs J on time. In Theorem 4.2, we bound the throughput $|\textsc{Opt}|$ of an optimal non-migratory solution by $\left(\frac{8}{\varepsilon} + 4\right)|J|$. Combining these theorems shows that the region algorithm achieves a competitive ratio of $c = 2 \cdot \left(\frac{8}{\varepsilon} + 4\right) = \frac{16}{\varepsilon} + 8$. □

6. Conclusion. In this paper, we close the problem of online single-machine throughput maximization with and without commitment requirements. For both commitment settings, we give an optimal online algorithm. Further, our algorithms run in a multiple-machine environment, even on heterogenous machines. Our algorithms compute non-migratory schedules on unrelated machines with the same competitive ratio $\mathcal{O}\left(\frac{1}{\varepsilon}\right)$ as for a single machine and improve substantially upon the state of the art.

It remains open whether the problem with a large number of machines admits an online algorithm with a better competitive ratio. For $m \geq 2$, it is not known whether slack is actually needed to design algorithms with bounded competitive ratios, even without commitment requirements and identical machines. In fact, results in [26] (used to show a $\mathcal{O}(1)$-competitive randomized algorithm on a single machine) imply an $\mathcal{O}(1)$-competitive algorithm for scheduling jobs without slack and without commitment on $m \in \mathcal{O}(1)$ identical machines. Further, for machine utilization, i.e., weighted throughput with $p_j = w_j$, [23, 34] improve upon the factor of $\mathcal{O}\left(\frac{1}{\varepsilon}\right)$ for commitment upon arrival and jobs satisfying the ε-slack assumption.

In fact, there are examples in the literature in which the worst-case ratio for a scheduling problem improves with an increasing number of machines. Consider, e.g., the non-preemptive offline variant of our throughput maximization problem on identical machines. There is an algorithm with approximation ratio of 1.55 for any m which is improving with increasing number of machines, converging to 1 as m tends to infinity [20]. The second part of the result also holds for the weighted problem.

Another interesting question asks whether randomization allows for improved results. Recall that there is a $\mathcal{O}(1)$-competitive randomized algorithm for scheduling on a single machine without commitment and without slack assumption [26]. Therefore is seems plausible that randomization also helps designing algorithms with improved competitive ratios for the different commitment models, for which only weak lower bounds are known [10], and on multiple machines as discussed above.

Further, we leave migratory scheduling on unrelated machines as an open problem. Allowing migration in this setting means that, on each machine i, a certain fraction of the processing time p_{ij} is executed, and these fractions must sum to one. Generalizing the result we leverage for identical machines [25], it is conceivable that any migratory schedule can be turned into a valid non-migratory schedule of the same jobs by adding a constant number of machines *of each type*. Such a result would immediately allow to transfer our competitive ratios to the migratory setting (up to constant factors). Devanur and Kulkarni [13] show a weaker result that utilizes speed rather than additional machines. Note that the strong impossibility result of Im and

Moseley [22] does not rule out the desired strengthening because we make the ε-slack assumption for every job and machine eligible for it. Further, we – as well as Devanur and Kulkarni [13] – assume that the processing time of each job j satisfies $p_{ij} \leq d_j - r_j$ on any eligible machine i, whereas the lower bound in [22] requires jobs that violate this reasonable assumption.

Further research directions include generalizations such as weighted throughput maximization. While strong lower bounds exist for handling weighted throughput with commitment [10], there remains a gap for the problem without. The known lower bound of $\Omega\left(\frac{1}{\varepsilon}\right)$ already holds for unit weights [10]. A natural extension of the region algorithm bases its admission decisions on the density, i.e., the ratio of the weight of a job to its processing time. The result is an algorithm similar to the $\mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$-competitive algorithm by Lucier et al. [31]. Both algorithms only admit available jobs and interrupt currently running jobs if the new job is denser by a certain factor. However, we can show that there is a lower bound of $\Omega\left(\frac{1}{\varepsilon^2}\right)$ on the competitive ratio of such algorithms. Hence, in order to improve the upper bound for online weighted throughput maximization, one needs to develop a new type of algorithm.

REFERENCES

[1] K. Agrawal, J. Li, K. Lu, and B. Moseley, *Scheduling parallelizable jobs online to maximize throughput*, in LATIN, vol. 10807, Springer, 2018, pp. 755–776, https://doi.org/10.1007/978-3-319-77404-6_55.

[2] Y. Azar, I. Kalp-Shaltiel, B. Lucier, I. Menache, J. Naor, and J. Yaniv, *Truthful online scheduling with commitments*, in EC, ACM, 2015, pp. 715–732, https://doi.org/10.1145/2764468.2764535.

[3] N. Bansal, H. Chan, and K. Pruhs, *Competitive algorithms for due date scheduling*, Algorithmica, 59 (2011), pp. 569–582, https://doi.org/10.1007/s00453-009-9321-4.

[4] A. Bar-Noy, S. Guha, J. Naor, and B. Schieber, *Approximating the throughput of multiple machines in real-time scheduling*, SIAM J. Comput., 31 (2001), pp. 331–352, https://doi.org/10.1137/S0097539799354138, https://doi.org/10.1137/S0097539799354138.

[5] S. K. Baruah and J. R. Haritsa, *Scheduling for overload in real-time systems*, IEEE Trans. Computers, 46 (1997), pp. 1034–1039, https://doi.org/10.1109/12.620484.

[6] S. K. Baruah, J. R. Haritsa, and N. Sharma, *On-line scheduling to maximize task completions*, in RTSS, IEEE Computer Society, 1994, pp. 228–236, https://doi.org/10.1109/REAL.1994.342713.

[7] S. K. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. E. Rosier, D. E. Shasha, and F. Wang, *On the competitiveness of on-line real-time task scheduling*, Real-Time Systems, 4 (1992), pp. 125–144, https://doi.org/10.1007/BF00365406.

[8] S. K. Baruah, G. Koren, B. Mishra, A. Raghunathan, L. E. Rosier, and D. E. Shasha, *On-line scheduling in the presence of overload*, in FOCS, IEEE Computer Society, 1991, pp. 100–110, https://doi.org/10.1109/SFCS.1991.185354.

[9] P. Berman and B. DasGupta, *Improvements in throughout maximization for real-time scheduling*, in STOC, ACM, 2000, pp. 680–687.

[10] L. Chen, F. Eberle, N. Megow, K. Schewior, and C. Stein, *A general framework for handling commitment in online throughput maximization*, Math. Prog., 183 (2020), pp. 215–247, https://doi.org/10.1007/s10107-020-01469-2.

[11] B. DasGupta and M. A. Palis, *Online real-time preemptive scheduling of jobs with deadlines*, in APPROX, vol. 1913 of Lecture Notes in Computer Science, Springer, 2000, pp. 96–107, https://doi.org/10.1007/3-540-44436-X_11.

[12] M. L. Dertouzos and A. K. Mok, *Multiprocessor on-line scheduling of hard-real-time tasks*, IEEE Trans. Software Eng., 15 (1989), pp. 1497–1506, https://doi.org/10.1109/32.58762.

[13] N. R. Devanur and J. Kulkarni, *A unified rounding algorithm for unrelated machines scheduling problems*, in SPAA, C. Scheideler and J. T. Fineman, eds., ACM, 2018, pp. 283–290, https://doi.org/10.1145/3210377.3210384, https://doi.org/10.1145/3210377.3210384.

[14] J. Du and J. Y. Leung, *Minimizing the number of late jobs on unrelated machines*, Oper. Res. Lett., 10 (1991), pp. 153–158.

[15] F. Eberle, N. Megow, and K. Schewior, *Optimally handling commitment issues in online*

throughput maximization, in Proceedings of ESA, vol. 173 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[16] A. D. Ferguson, P. Bodík, S. Kandula, E. Boutin, and R. Fonseca, *Jockey: Guaranteed job latency in data parallel clusters*, in EuroSys, ACM, 2012, pp. 99–112, https://doi.org/10.1145/2168836.2168847.

[17] J. A. Garay, J. Naor, B. Yener, and P. Zhao, *On-line admission control and packet scheduling with interleaving*, in INFOCOM, IEEE Computer Society, 2002, pp. 94–103, https://doi.org/10.1109/INFCOM.2002.1019250.

[18] M. H. Goldwasser, *Patience is a virtue: The effect of slack on competitiveness for admission control*, J. Sched., 6 (2003), pp. 183–211, https://doi.org/10.1023/A:1022994010777.

[19] M. H. Goldwasser and B. Kerbikov, *Admission control with immediate notification*, J. Sched., 6 (2003), pp. 269–285, https://doi.org/10.1023/A:1022956425198.

[20] S. Im, S. Li, and B. Moseley, *Breaking 1 - 1/e barrier for non-preemptive throughput maximization*, in IPCO, F. Eisenbrand and J. Könemann, eds., vol. 10328 of Lecture Notes in Computer Science, Springer, 2017, pp. 292–304, https://doi.org/10.1007/978-3-319-59250-3_24, https://doi.org/10.1007/978-3-319-59250-3_24.

[21] S. Im, S. Li, and B. Moseley, *Breaking 1 - 1/e barrier for nonpreemptive throughput maximization*, SIAM J. Discret. Math., 34 (2020), pp. 1649–1669, https://doi.org/10.1137/17M1148438, https://doi.org/10.1137/17M1148438.

[22] S. Im and B. Moseley, *General profit scheduling and the power of migration on heterogeneous machines*, in SPAA, vol. 10807 of Lecture Notes in Computer Science, Springer, 2018, pp. 755–776, https://doi.org/10.1007/978-3-319-77404-6_55.

[23] S. Jamalabadi, C. Schwiegelshohn, and U. Schwiegelshohn, *Commitment and slack for online load maximization*, in SPAA, ACM, 2020, pp. 339–348, https://doi.org/10.1145/3350755.3400271.

[24] B. Kalyanasundaram and K. Pruhs, *Speed is as powerful as clairvoyance*, J. ACM, 47 (2000), pp. 617–643, https://doi.org/10.1145/347476.347479.

[25] B. Kalyanasundaram and K. Pruhs, *Eliminating migration in multi-processor scheduling*, J. Algorithms, 38 (2001), pp. 2–24, https://doi.org/10.1006/jagm.2000.1128.

[26] B. Kalyanasundaram and K. Pruhs, *Maximizing job completions online*, J. Algorithms, 49 (2003), pp. 63–85, https://doi.org/10.1016/S0196-6774(03)00074-9.

[27] G. Koren and D. E. Shasha, *MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling*, Theor. Comput. Sci., 128 (1994), pp. 75–97, https://doi.org/10.1016/0304-3975(94)90165-1.

[28] G. Koren and D. E. Shasha, *D^{over}: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems*, SIAM J. Comput., 24 (1995), pp. 318–339, https://doi.org/10.1137/S0097539792236882.

[29] E. Lawler, *A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs*, Ann. Oper. Res., 26 (1990), pp. 125–133.

[30] E. L. Lawler, *Recent results in the theory of machine scheduling*, in Mathematical Programming The State of the Art, A. Bachem, B. Korte, and M. Grötschel, eds., Springer, 1982, pp. 202–234, https://doi.org/10.1007/978-3-642-68874-4_9, https://doi.org/10.1007/978-3-642-68874-4_9.

[31] B. Lucier, I. Menache, J. Naor, and J. Yaniv, *Efficient online scheduling for deadline-sensitive jobs: Extended abstract*, in SPAA, ACM, 2013, pp. 305–314, https://doi.org/10.1145/2486159.2486187.

[32] K. Pruhs and C. Stein, *How to schedule when you have to buy your energy*, in APPROX, vol. 6302 of Lecture Notes in Computer Science, Springer, 2010, pp. 352–365, https://doi.org/10.1007/978-3-642-15369-3_27.

[33] K. Pruhs and G. J. Woeginger, *Approximation schemes for a class of subset selection problems*, Theor. Comput. Sci., 382 (2007), pp. 151–156, https://doi.org/10.1016/j.tcs.2007.03.006, https://doi.org/10.1016/j.tcs.2007.03.006.

[34] C. Schwiegelshohn and U. Schwiegelshohn, *The power of migration for online slack scheduling*, in ESA, vol. 57 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, pp. 75:1–75:17, https://doi.org/10.4230/LIPIcs.ESA.2016.75.

[35] R. Sitters, *Complexity of preemptive minsum scheduling on unrelated parallel machines*, Journal of Algorithms, 57 (2005), pp. 37–48, https://doi.org/https://doi.org/10.1016/j.jalgor.2004.06.011, https://www.sciencedirect.com/science/article/pii/S0196677404001130.