Mathematical Programming (2020) 183:215–247
https://doi.org/10.1007/s10107-020-01469-2

FULL LENGTH PAPER

Series B

A general framework for handling commitment in online throughput maximization

Lin Chen[1] · **Franziska Eberle**[2] · **Nicole Megow**[2] · **Kevin Schewior**[3] · **Cliff Stein**[4]

Received: 29 May 2019 / Accepted: 13 January 2020 / Published online: 3 February 2020

Abstract
We study a fundamental online job admission problem where jobs with deadlines arrive online over time at their release dates, and the task is to determine a preemptive single-server schedule which maximizes the number of jobs that complete on time. To circumvent known impossibility results, we make a standard slackness assumption by which the feasible time window for scheduling a job is at least $1+\varepsilon$ times its processing time, for some $\varepsilon > 0$. We quantify the impact that different provider commitment requirements have on the performance of online algorithms. Our main contribution is one universal algorithmic framework for online job admission both with and without commitments. Without commitment, our algorithm with a competitive ratio of $\mathcal{O}(1/\varepsilon)$ is the best possible (deterministic) for this problem. For commitment models, we give the first non-trivial performance bounds. If the 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 $\mathcal{O}(\varepsilon/((\varepsilon-\delta)\delta^2))$, for $0 < \delta < \varepsilon$. When a provider must commit upon starting a job, our bound is $\mathcal{O}(1/\varepsilon^2)$. Finally, we observe that for scheduling with commitment the restriction to the "unweighted" throughput model is essential; if jobs have individual weights, we rule out competitive deterministic algorithms.

Mathematics Subject Classification 90B35 · 68M20 · 68W25 · 68W40

1 Introduction

Many modern computing environments involve a centralized system for managing the resource allocation for processing many different jobs. Such environments are varied,

An extended abstract of this paper was published at the Conference on Integer Programming and Combinatorial Optimization (IPCO) 2019 [8].

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s10107-020-01469-2) contains supplementary material, which is available to authorized users.

Extended author information available on the last page of the article

including, for example, internal clusters and public clouds. These systems typically handle a diverse workload [22] with a mixture of jobs including short time-sensitive jobs, longer batch jobs, and everything in between. By centralizing computing and scheduling decisions, one can potentially better utilize resources.

The challenge for a system designer is to implement scheduling policies that trade off between these different types of jobs and obtain good performance. There are many ways to define good performance and in this paper, we will focus on the commonly used notion of *throughput* which is the number of jobs completed, or if jobs have weights, the total weight of jobs completed.

In general, throughput is a "social welfare" objective that tries to maximize total utility. To this end, a solution may abort jobs close to their deadlines in favor of many shorter and more urgent tasks [12]. As companies start to outsource mission critical processes to external clouds, they may require a certain provider-side guarantee, i.e., service providers have to *commit to complete* admitted jobs before they cannot be moved to other computing clusters anymore. Moreover, companies tend to rely on business analytics to support decision making. Analytical tools, that usually work with copies of databases, depend on faultless data. This means, once such a copy process started, its completion must be guaranteed.

Formally, we consider a model in which jobs arrive online over time at their *release date* r_j. Each job has a *processing time* $p_j \geq 0$, a *deadline* d_j, and possibly a *weight* $w_j > 0$. In order to complete, a job must receive a total of p_j units of processing time in the interval $[r_j, d_j)$. We allow *preemption*, that is, the processing time does not need to be contiguous. If a schedule completes a set S of jobs, then the *throughput* is $|S|$, while the weighted throughput is $\sum_{j \in S} w_j$. We analyze the performance of algorithms using standard *competitive analysis* in which the performance of an algorithm is compared to that of an optimal offline algorithm with full knowledge of the future. More precisely, an online algorithm ALG is called c-competitive if it achieves for any input instance I a total value of $\text{ALG}(I) \geq \frac{1}{c}\text{OPT}(I)$, where OPT is the value of an optimal offline algorithm.

Deadline-based objectives are typically much harder to optimize than other Quality-of-Service metrics such as makespan or total completion time. Indeed, the problem becomes hopeless when preemption is not allowed: whenever an algorithm starts a job j without being able to preempt it, it may miss the deadlines of an arbitrary number of jobs that would have been schedulable if j had not been started. For scheduling with commitment, we provide a similarly strong lower bound for the preemptive version of the problem in the presence of weights. Therefore, we focus on *unweighted preemptive online* throughput maximization.

Hard examples for online algorithms tend to involve jobs that arrive and then *must* immediately be processed since $d_j - r_j \approx p_j$. It is entirely reasonable to bar such jobs from a system, requiring that any submitted job contains some *slack*, that is, we must have some separation between p_j and $d_j - r_j$. To that end we say that an instance has ε-slack if every job satisfies $d_j - r_j \geq (1+\varepsilon)p_j$. We develop algorithms whose competitive ratio depends on ε; the greater the slack, the better we expect the performance of our algorithm to be. This slackness parameter captures certain aspects of Quality-of-Service provisioning and admission control, see e.g. [14,20], and it has been considered in previous work, e.g., in [2,4,13,15,22,24]. Other results

for scheduling with deadlines use speed scaling, which can be viewed as another way to add slack to the schedule, e.g. [1,3,16,23]. In this paper we quantify the impact that different job commitment requirements have on the performance of online algorithms. We parameterize our performance guarantees by the slackness of jobs.

1.1 Our results and techniques

Our main contribution is a general algorithmic framework, called *region algorithm*, for online scheduling with and without commitments. We prove performance guarantees which are either tight or constitute the first non-trivial results. We also answer open questions in previous work. We show strong lower bounds for the weighted case and therefore our algorithms are all for the unweighted case, $w_j \equiv 1$.

Optimal algorithm for scheduling without commitment We give an implementation of the region algorithm that achieves a competitive ratio of $\mathcal{O}(\frac{1}{\varepsilon})$. We prove that this is *optimal* by giving a matching lower bound (ignoring constants) for any deterministic online algorithm.

Impossibility results for commitment upon job arrival In this most restrictive model, an algorithm must decide immediately at a job's release date if the job will be completed or not. We show that no (randomized) online algorithm admits a bounded competitive ratio. Such a lower bound has only been shown by exploiting arbitrary job weights [22,26]. Given our strong negative result, we do not consider this commitment model any further.

Scheduling with commitment We distinguish two different models: (i) *commitment upon job admission* and (ii) *δ-commitment*. In the first model, an algorithm may discard a job any time before its start, its admission. This reflects the situation when the start of a process is the critical time point after which the successful execution is essential (e.g., faultless copy of a database). In the second model, δ-commitment, an online algorithm must commit to complete a job when its slack has reduced from the original slack requirement of at least an ε-fraction of the job size to a δ-fraction for $0 < \delta < \varepsilon$. Then, the latest time for committing to job j is $d_j - (1+\delta)p_j$. This models an early enough commitment (parameterized by δ) for mission critical jobs.

For both models, we show that implementations of the region algorithm allow for the first non-trivial performance guarantees. We prove an upper bound on the competitive ratio of $\mathcal{O}(1/\varepsilon^2)$ for commitment upon admission and a competitive ratio of $\mathcal{O}(\varepsilon/((\varepsilon-\delta)\delta^2))$, for $0 < \delta < \varepsilon$, in the δ-commitment model. These are the first rigorous non-trivial upper bounds in any commitment model (excluding the special weighted setting with $w_j = p_j$ that has been resolved; see related work).

Instances with arbitrary weights are hopeless without further restrictions. We show that there is no deterministic online algorithm with bounded competitive ratio, neither for commitment upon admission (also shown in [2]) nor for δ-commitment. Informally, our construction implies that there is no deterministic online algorithm with bounded competitive ratio in *any commitment model* in which a scheduler may have to commit to a job before it has completed. (This is hard to formalize but may give guidance for

Table 1 Summary of the state-of-the-art

	No commitment	Time point of commitment		
		at admission	δ-laxity	at arrival
$w_j \equiv 1$	$\Theta(\frac{1}{\varepsilon})$ Theorems 1 and 5	$\Omega(\frac{1}{\varepsilon}), \mathcal{O}(\frac{1}{\varepsilon^2})$ Theorems 2 and 5	$\Omega(\frac{1}{\varepsilon}), \mathcal{O}(\frac{\varepsilon}{(\varepsilon-\delta)\delta^2})$ Theorems 2 and 5	No $f(\varepsilon)$ Theorem 6
$w_j = p_j$	$\mathcal{O}(1)$ [19]	$\Theta(\frac{1}{\varepsilon})$ [11,13]	$\Theta(\frac{1}{\varepsilon})$ Theorem 8, [11,13]	$\Theta(\frac{1}{\varepsilon})$ [11,13]
General w_j	$\Omega(\frac{1}{\varepsilon}), \mathcal{O}(\frac{1}{\varepsilon^2})$ Theorem 5, [22]	No $f(\varepsilon)$ [2]	No $f(\varepsilon)$ Theorem 7	No $f(\varepsilon)$ [22]

the design of alternative commitment models.) Our lower bound for δ-commitment is actually more fine-grained: for any $\delta > 0$ and any ε with $\delta \leq \varepsilon < 1 + \delta$, no deterministic online algorithm has a bounded competitive ratio for weighted throughput. In particular, this rules out bounded performance guarantees for $\varepsilon \in (0, 1)$. We remark that for sufficiently large slackness ($\varepsilon > 3$), Azar et al. [2] provide an online algorithm that has bounded competitive ratio. Our new lower bound answers affirmatively the open question if high slackness is indeed required.

Finally, our impossibility result for weighted jobs and the positive result for instances without weights clearly separate the weighted from the unweighted setting. Hence, we do not consider weights in this paper. We summarize in Table 1 the state of the art regarding competitive analysis for online throughput maximization with and without commitment.

Our techniques Once a job j is admitted to the system, its slack becomes a scarce resource: to complete the job before its deadline (which may be mandatory depending on the commitment model, but is at least desirable), one needs to carefully "spend" the slack on admitting jobs to be processed before the deadline of j. Our general framework for admission control, the region algorithm, addresses this issue by the concept of "responsibility": whenever a job j' is admitted while j could be processed, j' becomes responsible for not admitting similar-length jobs for a certain period, its *region*. The intention is that j' reserves time for j to complete. To balance between reservation (commitment to complete j) and performance (loss of other jobs), the algorithm uses the parameters α and β, which specify the length of a region and similarity of job lengths.

A major difficulty in the analysis of the region algorithm is understanding the complex interval structure formed by feasible time windows, regions, and time intervals during which jobs are processed. Here, we rely on a key design principle of our algorithm: regions are defined independently of the actual execution of jobs. Thus, the analysis can be naturally split into two parts.

In the first part, we argue that the scheduling routine can handle the admitted jobs sufficiently well for suitably chosen parameters α and β. That means that the respective commitment model is obeyed and, if not implied by that, an adequate number of the admitted jobs is completed.

In the second part, we can disregard how jobs are actually scheduled by the scheduling routine and argue that the region algorithm admits sufficiently many jobs to be competitive with an optimum solution. The above notion of "responsibility" suggests a proof strategy mapping jobs that are completed in the optimum to the corresponding job that was "responsible" due to its region. Transforming this idea into a charging scheme is, however, a non-trivial task: There might be many ($\gg \mathcal{O}(\frac{1}{\varepsilon^2})$) jobs released within the region of a single job j and completed by the optimum, but not admitted by the region algorithm due to many consecutive regions of varying size. It is unclear where to charge these jobs—clearly not all of them to j.

We develop a careful charging scheme that avoids such overcharging. We handle the complex interval structure by working on a natural tree structure (*interruption tree*) related to the region construction and independent of the actual schedule. Our charging scheme comprises two central routines for distributing charge: moving charge along a sequence of consecutive jobs (*Push Forward*) or to children (*Push Down*).

We show that our analysis of the region algorithm is tight up to a constant factor.

1.2 Previous results

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., [5,13,15] and references therein. Impossibility results for jobs with hard deadlines and without slack have been known for decades [6,7,18,19,21].

Scheduling without commitment Most research on online scheduling does not address commitment. The only results independent of slack (or other job-dependent parameters) concern the machine utilization, i.e., weighted throughput for the special case $w_j = p_j$, where a constant competitive ratio is possible [6,18,19,25]. In the unweighted setting, a randomized $\mathcal{O}(1)$-competitive algorithm is known [17]. For instances with ε-slack, Lucier et al. [22] give an $\mathcal{O}(\frac{1}{\varepsilon^2})$-competitive algorithm in the most general weighted setting. To the best of our knowledge, no lower bound was known to date.

Scheduling with commitment Much less is known for scheduling with commitment. In the most restrictive model, *commitment upon job arrival*, Lucier et al. [22] rule out competitive online algorithms for any slack parameter ε when jobs have arbitrary weights. For *commitment upon job admission*, they give a heuristic that empirically performs very well but for which they cannot show a rigorous worst-case bound. In fact, later Azar et al. [2] show that no bounded competitive ratio is possible for weighted throughput maximization for small ε. For the *δ-commitment model*, Azar et al. [2] design (in the context of truthful mechanisms) an online algorithm that is $\mathcal{O}(\frac{1}{\varepsilon^2})$-competitive if the slack ε is sufficiently large. They call an algorithm in this model *β-responsive algorithm*. They left open if this latter condition is an inherent property of any committed scheduler in this model and we answer this affirmatively.

Again, the machine utilization variant ($w_j = p_j$) is much more tractable than weighted or unweighted throughput maximization. Simple greedy algorithms achieve the best possible competitive ratio $\Theta(\frac{1}{\varepsilon})$ [11,13] in all aforementioned commitment models, even commitment upon arrival.

2 Our general framework

2.1 The region algorithm

In this section we present our general algorithmic framework which we apply to scheduling with and without commitment. We assume that an online algorithm is given the slackness constant $\varepsilon > 0$ and, in the δ-commitment model, $0 < \delta < \varepsilon$.

To gain some intuition for our algorithm, we first describe informally the three underlying design principles. The third principle is crucial to improve on existing results that only use the first two [22].

1. A running job can be preempted only by significantly smaller jobs (parameter β).
2. A job cannot start for the first time when its remaining slack is too small (constant δ which is part of the input in the δ-commitment model and otherwise set to $\delta = \frac{\varepsilon}{2}$).
3. If a job preempts other jobs, then it has to take "responsibility" for a certain time interval (parameter α) with which it assures that the jobs it preempted can complete on time.

We implement it in the following way. The region algorithm has two parameters, $\alpha \geq 1$ and $0 < \beta < 1$. A *region* is a union of time intervals associated with a job, and the size of the region is the sum of sizes of the intervals. We denote the region of job j by $R(j)$. Region $R(j)$ will always have size αp_j, although the particular time intervals composing the region may change over time. Regions are always disjoint, i.e., for any $i \neq j$, $R(i) \cap R(j) = \emptyset$. Informally, whenever our algorithm starts a job i (we say i is *admitted*) that arrives during the region of an already admitted job j, then the current interval of j is split into two intervals and the region $R(j)$ as well as all later regions are delayed.

Formally speaking, at any time t, the region algorithm maintains two sets of jobs: *admitted jobs*, which have been started before or at time t, and *available jobs*. A job j is available if it is released before or at time t, is not yet admitted, and it is not too close to its deadline, i.e., $r_j \leq t$ and $d_j - t \geq (1 + \delta)p_j$. The intelligence of the region algorithm lies in admitting jobs and (re)allocating regions. The actual scheduling decision then is simple and independent of the regions: at any point in time, schedule the shortest admitted job that has not completed its processing time, i.e., we schedule admitted jobs in *Shortest Processing Time (SPT)* order. The region algorithm never explicitly considers deadlines except when deciding whether to admit jobs.

The region algorithm starts by admitting job 1 at its release date and creates the region $R(1) := [r_1, r_1 + \alpha p_1)$. There are two events that trigger a decision of the region algorithm: the release of a job and the end of a region. If one of these events occurs at time t, the region algorithm invokes the **region preemption** subroutine. This routine compares the processing time of the smallest *available* job i with the processing time of the *admitted* job k whose region contains t. If $p_i < \beta p_k$, job i is admitted and the region algorithm reserves the interval $[t, t + \alpha p_i)$ for processing i. Since regions must be disjoint, the algorithm then modifies all other remaining regions, i.e., the parts of regions that belong to $[t, \infty)$ of other jobs j. We refer to the set of such jobs j whose regions have not yet completed by time t as $J(t)$. Intuitively, we preempt the interval of the region containing t and delay its remaining part as well

Algorithm 1 Region algorithm

Scheduling routine: At any time t, run an admitted and not yet completed job with shortest processing time.

Event: Upon release of a new job at time t:
Call region **preemption routine**.

Event: Upon ending of a region at time t:
Call region **preemption routine**.

Region preemption routine:
$k \leftarrow$ the job whose region contains t
$i \leftarrow$ a shortest available job at t, i.e., $i = \arg\min\{p_j \mid r_j \leq t \text{ and } d_j - t \geq (1+\delta)p_j\}$
If $p_i < \beta p_k$, then
admit job i and reserve region $R(i) = [t, t + \alpha p_i)$,
update all remaining regions $R(j)$ with $R(j) \cap [t, \infty) \neq \emptyset$ as described below.

as the remaining regions of all other jobs. Formally, this **update of all remaining regions** is defined as follows. Let k be the one job whose region is interrupted at time t, and let $[a'_k, b'_k)$ be the interval of $R(k)$ containing t. Interval $[a'_k, b'_k)$ is replaced by $[a'_k, t) \cup [t + \alpha p_i, b'_k + \alpha p_i)$. For all other jobs $j \in J(t)\backslash\{k\}$, the remaining region $[a'_j, b'_j)$ of j is replaced by $[a'_j + \alpha p_i, b'_j + \alpha p_i)$. Observe that, although the region of a job may change throughout the algorithm, the starting point of a region for a job will never be changed. We summarize the region algorithm in Algorithm 1.

We apply the algorithm in different commitment models with different choices of parameters α and β, which we derive in the following sections. In the δ-commitment model, δ is given as part of the input. In the other models, i.e., without commitment or with commitment upon admission, we simply set $\delta = \frac{\varepsilon}{2}$.

Commitment The region algorithm always commits upon admission of a job, i.e., at its first start. This is possibly earlier than required in the δ-commitment model. The parameter δ determines the latest possible start time of a job, which is then for our algorithm also the latest time the job can be admitted. Thus, for the analysis, the algorithm execution for commitment upon admission (with $\delta = \frac{\varepsilon}{2}$) is a special case of δ-commitment. This is true only for our algorithm, not in general.

2.2 Main results on the region algorithm

In the analysis we focus on instances with small slack as they constitute the hard case. Notice that instances with large slack clearly satisfy a small slack assumption. In such a case, we simply run our algorithm by setting $\varepsilon = 1$ and obtain constant competitive ratios. Therefore, we assume for the remainder that $0 < \varepsilon \leq 1$.

Our main results are as follows. Without commitment, we present an optimal online algorithm.

Theorem 1 (Scheduling without commitment) *Let $0 < \varepsilon \leq 1$. With the choice of $\alpha = 1$, $\beta = \frac{\varepsilon}{4}$, and $\delta = \frac{\varepsilon}{2}$, the region algorithm is $\Theta(\frac{1}{\varepsilon})$-competitive for scheduling without commitment.*

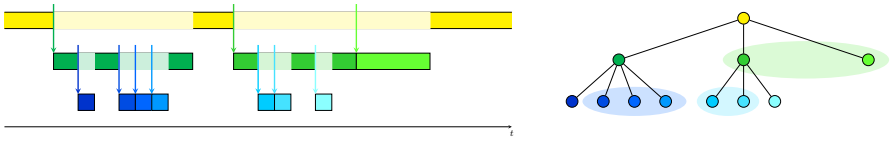

Fig. 1 Gantt chart of the regions (left) and the interruption tree (right) generated by the region algorithm (color figure online)

This is an exponential improvement upon the previously best known upper bound [22] (given for weighted throughput). For scheduling with commitment, we give the first rigorous upper bound.

Theorem 2 (Scheduling with commitment) *Let* $0 < \delta < \varepsilon \leq 1$. *Choosing* $\alpha = \frac{8}{\delta}$, $\beta = \frac{\delta}{4}$, *the region algorithm is* $\mathcal{O}(\frac{\varepsilon}{(\varepsilon-\delta)\delta^2})$*-competitive in the* δ*-commitment model. When the scheduler has to commit upon admission, the region algorithm has a competitive ratio* $\mathcal{O}(\frac{1}{\varepsilon^2})$ *for* $\alpha = \frac{4}{\varepsilon}$ *and* $\beta = \frac{\varepsilon}{8}$.

In Sect. 5, we show that the analysis of our framework is tight up to constants.

2.3 Interruption trees

To analyze the performance of the region algorithm on a given instance, we consider the final schedule and the final regions and investigate them retrospectively. Let a_j be the admission date of job j which remained fixed throughout the execution of the algorithm. Let b_j denote the final end point of j's region. Then, the convex hull of $R(j)$ is given by $\text{conv}(R(j)) = [a_j, b_j)$.

Our analysis crucially relies on understanding the interleaving structure of the regions that the algorithm constructs. This structure is due to the interruption by smaller jobs and can be captured well by a tree or forest in which each job is represented by one vertex. A job vertex is the child of another vertex if and only if the region of the latter is interrupted by the first one. The leaves correspond to jobs with non-interrupted regions. By adding a machine job M with $p_M := \infty$ and $a_M = -\infty$, we can assume that the instance is represented by a tree which we call *interruption tree*. This idea is visualized in Fig. 1, where the vertical arrows indicate the interruption of a region by another job and intervals of the same color belong to one job.

Let $\pi(j)$ denote the *parent* of j. Further, let T_j be the subtree of the interruption tree rooted in job j and let the forest T_{-j} be T_j without its root j. By slightly abusing notation, we denote the tree/forest as well as its job vertices by T_*.

A key property of this tree is that the processing times on a path are geometrically decreasing.

Lemma 1 *Let* $j_1, \ldots, j_\ell$ *be* ℓ *jobs on a path in the interruption (sub)tree* T_j *rooted in* j *such that* $\pi(j_{i+1}) = j_i$. *Then,* $p_{j_\ell} \leq \beta p_{j_{\ell-1}} \cdots \leq \beta^{\ell-1} p_{j_1} \leq \beta^\ell p_j$ *and the total processing volume is*

$$\sum_{i=1}^{\ell} p_{j_i} \leq \sum_{i=1}^{\ell} \beta^i p_j \leq \frac{\beta}{1-\beta} \cdot p_j.$$

Proof Let the jobs $j_1, \ldots, j_\ell$ be indexed in decreasing order of processing times, i.e., $p_{j_\ell} \leq p_{j_{\ell-1}} \leq \cdots \leq p_{j_1}$.

Observe that $p_{j_1} < \beta p_j$ as otherwise the region of j shall not be preempted by j_1. Furthermore, for any $2 \leq i \leq \ell$, we claim that job j_i is released after j_{i-1}. Suppose the claim is not true, then for some i job j_i is released before j_{i-1}. Consider the point in time t when job j_{i-1} is admitted. The time t either belongs to the region of j_i, or belongs to the region of some job j' which interrupts the region of j_i, and consequently $p_{j'} < \beta p_{j_i}$. In both cases the algorithm will not admit j_{i-1}, and therefore the claim is true. At any point in time when the algorithm admits a job j_i, then it interrupts the region of j_{i-1} and $p_{j_i} < \beta p_{j_{i-1}}$. Thus, we have

$$p_{j_\ell} < \beta p_{j_{\ell-1}} < \beta^2 p_{j_{\ell-2}} < \cdots < \beta^{\ell-1} p_{j_1} < \beta^\ell p_j.$$

We conclude by observing that the total processing volume of the jobs $j_1, \ldots, j_\ell$ is

$$\sum_{i=1}^{\ell} p_{j_i} < \sum_{i=1}^{\ell} \beta^i p_j = \frac{\beta(1-\beta^\ell)}{1-\beta} \cdot p_j \leq \frac{\beta}{1-\beta} \cdot p_j.$$

□

3 Successfully completing sufficiently many admitted jobs

We show that the region algorithm completes sufficiently many jobs among the admitted jobs before their deadline. For scheduling without commitment, we show how to choose α, β, and δ to ensure that *at least half* of all admitted jobs are completed on time. For scheduling with commitment, we provide a choice of α, β, and δ such that *every* admitted job is guaranteed to complete on time.

Recall that the region algorithm schedules admitted and yet not completed jobs independently of the regions in SPT order. This guarantees the following.

Observation 1 *For $\alpha \geq 1$, the region algorithm always prioritizes a job within its own region.*

3.1 Scheduling without commitment

In this section we fix $\delta = \frac{\varepsilon}{2}$ for $0 < \varepsilon \leq 1$. We show the following result.

Theorem 3 *Let $\alpha = 1$ and $\beta = \frac{\varepsilon}{4}$. Then the region algorithm completes at least half of all admitted jobs before their deadline.*

The intuition for setting $\alpha = 1$ and thus reserving regions of minimum size $|R(j)| = p_j$, for any j, is that in the model without commitment, we do not need to block

extra time in the future to ensure the completion of earlier admitted jobs. Because of Observation 1, for $\alpha = 1$, every job j completes at the end of the region at b_j. Thus, j completes on time if and only if the region $R(j)$ ends before d_j, i.e., $b_j \leq d_j$. We prove Theorem 3 by showing that at least half of all regions end before the deadline of their respective jobs. Formally, we prove the following lemma.

Lemma 2 *For any instance $\mathcal{I}$ and some job j, for which the region algorithm generates an interruption tree T_j with regions in $[a_j, b_j)$, there is an instance $\mathcal{I}'$ with at most $|T_j| + 1$ jobs such that the regions in $[a_j, b_j)$ and the tree T_j are identical.*

Proof Consider an instance $\mathcal{I}$ of the non-committed scheduling problem, and let T_j be the interruption tree constructed by the region algorithm with its root in j. Let the interval $[a_j, b_j)$ be the convex hull of the subintervals belonging to the region of j. Our goal is to modify the instance $\mathcal{I}$ such that we can remove jobs outside of T_j without changing the interruption tree of the algorithm. We do so by setting $\mathcal{I}'$ to contain the set of jobs in T_j and changing only parameters of job j (and possibly adding one auxiliary job). Note that j is, by definition, the largest job in $\mathcal{I}'$.

If $d_j - a_j \geq (1+\varepsilon)p_j$ then we set $r'_j := a_j$. Otherwise, we add an auxiliary job 0 to $\mathcal{I}'$ that is tight and blocks the machine until a_j. This means $r_0 = d_j - (1+\varepsilon)p_j$, $p_0 = (1+\varepsilon)p_j - (d_j - a_j)$, and $d_0 = r_0 + (1+\varepsilon)p_0$. Moreover, we modify the release date of j to $r'_j := r_0$. Since the auxiliary job is the smallest job in instance $\mathcal{I}'$ at time r_0, the region algorithm admits this job and delays job j. Let $\mathcal{R}$ and $\mathcal{R}'$ be the schedule of regions in $[a_j, b_j)$ generated by the region algorithm when applied to $\mathcal{I}$ and $\mathcal{I}'$, respectively. We show that $\mathcal{R}$ and $\mathcal{R}'$ are identical in $[a_j, b_j)$.

Consider the time $t = a_j$. Clearly, j's region starts in $\mathcal{R}$ by assumption. If no auxiliary job was used, job j is the only available job in $\mathcal{I}'$. Thus, the region algorithm admits j. In contrast, if $0 \in \mathcal{I}'$, it finishes at a_j by definition. Since j is admitted in $\mathcal{R}$, it must hold that $d_j - a_j \geq (1+\delta)p_j$. Thus, its regions also begins in $\mathcal{R}'$ at a_j.

Let $a_j < t < b_j$ be the first time when the two region schedules $\mathcal{R}$ and $\mathcal{R}'$ are different. Since both schedules are generated by the region algorithm, any change in the structure of the regions is due to one of the two decision events of the region algorithm. Recall that these events where the end of a job's region and the release of a new job. We distinguish two cases based on the job k that caused the difference in $\mathcal{R}$ and $\mathcal{R}'$: $k \in T_j$ or $k \notin T_j$.

By definition, the region of any job outside of T_j has empty intersection with $[a_j, b_j)$. Thus, the release of such a job can neither change $\mathcal{R}$ nor $\mathcal{R}'$. Of course, the region of such job cannot end within $[a_j, b_j)$. Thus, a job $k \in T_j$ is the reason for the difference in $\mathcal{R}$ and $\mathcal{R}'$. Let k be the job that owns time t in $\mathcal{R}$. If the processor is idle in $\mathcal{R}$ after t let k be the job that owns t in $\mathcal{R}'$. As the two schedules are identical in $[a_j, t)$, let $i \in T_j$ be the unique job that owns the time right before t.

Consider the event that the region of i ended at t. If k is an ancestor of i, then there is no sufficiently small job available in $\mathcal{I}$ that prevents k from being restarted at t. Additionally, the amount of time that belongs to $R(k)$ in $[a_j, t)$ is identical in $\mathcal{R}$ and $\mathcal{R}'$. Thus, k also resumes processing in $\mathcal{R}'$. If k is admitted at t in $\mathcal{R}$, it is sufficiently small to (further) preempt the ancestor of i and it is available for admission. Hence, these two properties are also satisfied in $\mathcal{I}'$ and k is admitted at t in $\mathcal{R}'$ as well.

Therefore $\mathcal{R}$ contains idle time at t while the region of k is scheduled in $\mathcal{R}'$. Since the jobs in $\mathcal{I}'$ are a subset of the jobs in $\mathcal{I}$ (except for 0), job k is also admitted and unfinished or available at t in $\mathcal{I}$. This is a contradiction. □

We show that the existence of a late job j implies that the subtree T_j rooted in j contains more finished than unfinished jobs. We fix a job $j \in J$ that was admitted by the region algorithm at time a_j and whose region completes at time b_j. We want to analyze the structure of all regions $\mathcal{R}$ in $[a_j, b_j)$, i.e., the regions of all jobs in T_j. Let F_j denote the set of jobs in T_j that *finish on time*. Similarly, we denote the set of jobs in T_j that complete after their deadlines, i.e., that are *unfinished at their deadline*, by U_j.

Lemma 3 *Let* $\alpha = 1$ *and* $\beta = \frac{\varepsilon}{4}$, *with* $\varepsilon > 0$. *If* $b_j - a_j \geq (\ell + 1)p_j$ *for* $\ell > 0$, *then* $|F_j| - |U_j| \geq \lfloor \frac{4\ell}{\varepsilon} \rfloor$.

As this proof is rather technical without new insights, we give only the proof sketch here and refer to the Appendix for the full details.

Proof (Proof sketch) Assume for the sake of contradiction that there is an instance such that the interruption tree generated by the region algorithm contains a subtree T_j with $b_j - a_j \geq (\ell+1)p_j$ and $|F_j| - |U_j| < \lfloor \frac{4\ell}{\varepsilon} \rfloor$. Let $\mathcal{I}$ be such an instance that uses a minimal number of jobs in total. The goal is to construct an instance $\mathcal{I}'$ that satisfies $b_j - a_j \geq (\ell + 1)p_j$ and $|F_j| - |U_j| < \lfloor \frac{4\ell}{\varepsilon} \rfloor$ although it uses less jobs than $\mathcal{I}$.

To this end, we modify $\mathcal{I}$ in several steps such that we can merge three jobs to one larger job without violating $b_j - a_j \geq (\ell + 1)p_j$, changing $|F_j|$ or $|U_j|$, or making the instance infeasible. The three jobs will be leaves with the same parent i in T_j. If i is an unfinished job that has children which are all leaves, then there have to be at least three jobs that interrupt i. After merging the three jobs, we adapt the release date and deadline of i to guarantee that the modified instance remains feasible. For all these modification steps, it is crucial that we can restrict to instances in which all jobs appear in the interruption tree (Lemma 2).

However, this modification might lead to $b_i \leq d'_i$ which implies that i finishes on time. This changes the values of $|F_j|$ and $|U_j|$. Clearly, in this case, $|U'_j| = |U_j| - 1$. By a careful analysis, we see that the number of finished jobs decreases by one as well because the three children of i are replaced by only one finished job. Hence, $|F'_j| - |U'_j| = |F_j| - |U_j|$. If i does not finish by d'_i, then $|F'_j| - |U'_j| = (|F_j| - 2) - |U_j|$. Thus, the modified instance $\mathcal{I}'$ also violates $|F'_j| - |U'_j| \geq \lfloor \frac{4\ell}{\varepsilon} \rfloor$ but uses less jobs than $\mathcal{I}$ does; a contradiction. □

Proof (Theorem 3) Let U be the set of jobs that are unfinished by their deadline but whose ancestors (except machine job M) have all completed on time. Every job $j \in U$ was admitted by the algorithm at some time a_j with $d_j - a_j \geq (1 + \delta)p_j$. With $\delta = \frac{\varepsilon}{2}$ this implies $b_j - a_j > d_j - a_j \geq (1 + \frac{\varepsilon}{2})p_j$. By Lemma 3, it follows that $|F_j| - |U_j| \geq \lfloor \frac{4 \cdot \varepsilon/2}{\varepsilon} \rfloor = 2$. Then, $|T_j| = |F_j| + |U_j| \leq 2|F_j| - 2 < 2|F_j|$. This completes the proof. □

3.2 Scheduling with commitment

We analyze the region algorithm for scheduling with commitment. For both models, commitment at admission and δ-commitment, we show that there is a choice of α and β such that every job that has started processing will be completed before its deadline. Recall that we can restrict to analyzing the algorithm in the δ-commitment model since it runs with $\delta = \frac{\varepsilon}{2}$ for commitment at admission.

Lemma 4 *Let $\varepsilon, \delta > 0$ be fixed with $\delta < \varepsilon$. If $\alpha \geq 1$ and $0 < \beta < 1$ satisfy the condition that*

$$\frac{\alpha - 1}{\alpha} \cdot \left(1 + \delta - \frac{\beta}{1 - \beta}\right) \geq 1, \tag{1}$$

then any job j that is admitted by the algorithm at time $a_j \leq d_j - (1 + \delta)p_j$ will be finished by d_j.

Proof Consider a job j that is admitted (and simultaneously accepted for completion) by time a_j. It holds that $d_j - a_j \geq (1 + \delta)p_j$. We show that j receives at least p_j units of time within $[a_j, d_j)$. Let $|R(k)|$ denote the total length of intervals in $R(k)$, the region of job k.

Let $D_j \subseteq T_{-j}$ be the set of jobs whose region delays the region of job j, and has nonempty intersection with $[a_j, d_j)$. Notice that a job $k \in D_j$ can only be released after time a_j. Let $D'_j \subseteq D_j$ be the subset of jobs whose region is completely contained in $[a_j, d_j)$ and $D''_j = D_j \backslash D'_j$.

Consider D'_j. Notice that $\left|\bigcup_{k \in D'_j} R(k)\right| = \alpha \sum_{k \in D'_j} p_k$. Thus, within regions $R(k)$ of jobs $k \in D'_j$, an $\frac{\alpha-1}{\alpha}$-fraction of the total time is available for processing job j.

Consider $D''_j = \{j_1, j_2, \ldots, j_\ell\}$ and assume that $p_{j_1} \geq p_{j_2} \geq \cdots \geq p_{j_\ell}$. Any interval $[a_{j_i}, b_{j_i})$ of such a job j_i in D''_j contains d_j. This implies that $\pi(j_{i+1}) = j_i$ for $0 \leq i < \ell$ where $j_0 := j$ for simplicity. Applying Lemma 1 gives an upper bound on the total processing volume of jobs in D''_j, i.e., $\sum_{i=1}^{\ell} p_{j_i} \leq \frac{\beta}{1-\beta} \cdot p_j$.

To determine the amount of time for processing j within $[a_j, d_j)$, we first subtract the total processing time for jobs in D''_j. The remaining interval may be covered with regions of D'_j within which we can use an $\frac{\alpha-1}{\alpha}$-fraction as shown above. Recall that $d_j - a_j \geq (1+\delta)p_j$. Thus, the amount of time that we can process job j within $[a_j, d_j)$ is at least

$$\frac{\alpha - 1}{\alpha} \cdot \left((d_j - a_j) - \sum_{j_i \in D''_j} p_{j_i}\right) \geq \frac{\alpha - 1}{\alpha} \cdot \left(1 + \delta - \frac{\beta}{1 - \beta}\right) \cdot p_j.$$

This bound is now independent of the actual schedule. We can conclude, if α and β satisfy Condition (1), then job j can process for p_j units of time within $[a_j, d_j)$ and completes before its deadline. □

4 Competitiveness: admission of sufficiently many jobs

We show that the region algorithm admits sufficiently many jobs, independently of the commitment model.

Theorem 4 *The number of jobs that an optimal (offline) algorithm can complete on time is at most a factor* $\lambda + 1$ *larger than the number of jobs admitted by the region algorithm, where* $\lambda := \frac{\varepsilon}{\varepsilon-\delta}\frac{\alpha}{\beta}$, *for* $0 < \delta < \varepsilon \leq 1$.

To prove the theorem, we fix an instance and an optimal offline algorithm Opt. Let X be the set of jobs that Opt scheduled and the region algorithm did not admit. We can assume that Opt completes all jobs in X on time. Let J denote the jobs that the region algorithm admitted. Then, $X \cup J$ is a superset of the jobs scheduled by Opt. Thus, showing $|X| \leq \lambda|J|$ implies Theorem 4.

To this end, we develop a charging procedure that assigns each job in X to a unique job in J such that each job $j \in J$ is assigned at most $\lambda = \frac{\varepsilon}{\varepsilon-\delta}\frac{\alpha}{\beta}$ jobs. For a job $j \in J$ admitted by the region algorithm we define the subset $X_j \subset X$ based on release dates. Then, we inductively transform the laminar family $(X_j)_{j\in J}$ into a partition $(Y_j)_{j\in J}$ of X with $|Y_j| \leq \lambda$ for all $j \in J$ in the proof of Lemma 5, starting with the leaves in the interruption tree as base case (Lemma 7). For the construction of $(Y_j)_{j\in J}$, we heavily rely on the key property (Volume Lemma 6) and Corollary 1.

More precisely, for a job $j \in J$ let X_j be the set of jobs $x \in X$ that were released in the interval $[a_j, b_j)$ and satisfy $p_x < \beta p_{\pi(j)}$. Let $X_j^S := \{x \in X_j : p_x < \beta p_j\}$ and $X_j^B := X_j \setminus X_j^S$ denote the *small* and the *big* jobs, respectively, in X_j. Recall that $[a_j, b_j)$ is the convex hull of the region $R(j)$ of job j and that it includes the convex hulls of the regions of all descendants of j in the interruption tree, i.e., jobs in T_j. In particular, $X_k \subset X_j$ if $k \in T_j$.

Observation 2
1. *Any job that is scheduled by* Opt *and not admitted by the region algorithm is released within the region of some job* $j \in J$, *i.e.,* $\bigcup_{j\in J} X_j = X$.
2. *As the region algorithm admits any job that is small w.r.t.* j *and released in* $R(j)$, *it holds that* $X_j^S = \bigcup_{k:\pi(k)=j} X_k$.

Recall that M denotes the machine job. By Observation 2, $X = X_M^S$ and, thus, it suffices to show that $|X_M^S| \leq \lambda|J|$. In fact, we show a stronger statement for each job $j \in J$: the number of small jobs in X_j is bounded by $\lambda\tau_j$ where τ_j is the number of descendants of j in the interruption tree, i.e., $\tau_j := |T_{-j}|$.

Lemma 5 *For all* $j \in J \cup \{M\}$, $|X_j^S| \leq \lambda\tau_j$.

Before proving the lemma, we will highlight the main steps in the following. The fine-grained definition of the sets X_j in terms of the release dates and the processing times allows us to show that any job j with $|X_j| > (\tau_j + 1)\lambda$ has *siblings* $j_1, \ldots, j_k$ such that $|X_j| + \sum_{i=1}^{k} |X_{j_i}| \leq \lambda(\tau_j + 1 + \sum_{i=1}^{k}(\tau_{j_i} + 1))$. We call i and j siblings if they have the same parent in the interruption tree. Simultaneously applying this charging idea to *all* descendants of a job h already proves $|X_h^S| \leq \lambda\tau_h$ as $X_h^S = \bigcup_{j:\pi(j)=h} X_j$ by Observation 2.

We prove that this "balancing" of X_j between jobs only happens between siblings $j_1, \ldots, j_k$ with the property that $b_{j_i} = a_{j_{i+1}}$ for $1 \leq i < k$. We call such a set of jobs a *string* of jobs. The ellipses in Fig. 1 visualize the maximal strings of jobs. A job j is called *isolated* if $b_i \neq a_j$ and $b_j \neq a_i$ for all children $i \neq j$ of $\pi(j)$.

The next (technical) lemma is a key ingredient for the "balancing" of X_j between a string of jobs. For any subset of J, we index the jobs in order of increasing admission points a_j. Conversely, for a subset of X, we order the jobs in increasing order of completion times, C_x^*, in the optimal schedule.

Lemma 6 (Volume Lemma) *Let $f, \ldots, g \in J$ be jobs with a common parent in the interruption tree. Let $x \in \bigcup_{j=f}^{g} X_j$ such that*

$$\sum_{j=f}^{g} \sum_{y \in X_j : C_y^* \leq C_x^*} p_y \geq \frac{\varepsilon}{\varepsilon - \delta}(b_g - a_f) + p_x. \tag{V}$$

Then, $p_x \geq \beta p_{j^*}$, where $j^* \in J \cup \{M\}$ is the job whose region contains b_g, i.e., $b_g \in R(j^*)$.

Proof Let $f, \ldots, g, x$, and j^* as in the lemma. Since $x \in X$, the region algorithm did not accept x at time b_g. There are two possible reasons for this behavior: either $p_x \geq \beta p_{j^*}$ or x was not available for admission at time b_g anymore.

Assume for the sake of contradiction that $p_x < \beta p_{j^*}$ and, thus, $d_x - b_g < (1+\delta)p_x$. By assumption, $r_x \geq a_f$ and $d_x - r_x \geq (1+\varepsilon)p_x$. Hence,

$$b_g - a_f \geq b_g - d_x + d_x - r_x > -(1+\delta)p_x + (1+\varepsilon)p_x = (\varepsilon - \delta)p_x.$$

By (V), the volume OPT processes between b_g and C_x^* is at least $\frac{\delta}{\varepsilon - \delta}(b_g - a_f) + p_x$. By applying the above calculated lower bound, we get that

$$\frac{\delta}{\varepsilon - \delta}(b_g - a_f) + p_x > \delta p_x + p_x = (1+\delta)p_x$$

and, hence, that $C_x^* \geq b_g + (1+\delta)p_x > d_x$, which contradicts that OPT is a feasible schedule. □

The next corollary follows directly from the Volume Lemma applied to a string of jobs or to a single job $j \in J$ (let $f = j = g$). To see this, recall that X_j contains only jobs that are small w.r.t. $\pi(j)$, i.e., all $x \in X_j$ satisfy $p_x < \beta p_{\pi(j)}$.

Corollary 1 *Let $\{f, \ldots, g\} \subset J$ be a string of jobs and let $x \in \bigcup_{j=f}^{g} X_j$ satisfy (V). Then, the interruption tree contains a sibling j^* of g with $b_g = a_{j^*}$.*

The main part of the proof of Lemma 5 is to show (V) for a string of jobs only relying on $\sum_{j=f}^{g} |X_j| > \lambda \sum_{j=f}^{g} (\tau_j + 1)$. Then, Corollary 1 allows us to charge the "excess" jobs to a subsequent sibling $g+1$. The relation between processing volume and size of job sets is possible due to the definition of X_j based on T_j.

We inductively prove Lemma 5 where the induction is on the distance $\varphi(j)$ of a job j from the machine job M, i.e., $\varphi(M) := 0$ and $\varphi(j) := \varphi(\pi(j)) + 1$ for $j \in J$. Moreover, let $\varphi_{\max} := \max\{\varphi(j) : j \in J\}$ be the maximal distance or, equivalently, the height of the interruption tree. Any job j at maximal distance from the machine job is a leaf in the interruption tree. The following lemma serves as base case in the proof of Lemma 5.

Lemma 7 *Let $\{f, \dots, g\} \subset J$ be jobs at maximal distance from M such that $\sum_{j=f}^{i} |X_j| > \lambda(i + 1 - f)$ holds for all $f \le i \le g$. If g is the last such job, there is a sibling j^* of g with $b_g = a_{j^*}$ and $\sum_{j=f}^{j^*} |X_j| \le \lambda(j^* + 1 - f)$.*

Proof Observe that $[a_f, b_g) = \bigcup_{j=1}^{k} R(j)$ because the leaves $f, \dots, g$ form a string of jobs. Thus, by showing that there is a job $x \in X_f^g := \bigcup_{j=f}^{g} X_j$ that satisfies (V), we prove the statement with the Volume Lemma. To this end, we show that for every job $f \le j \le g$ there exists a set Y_j such that the processing volume within Y_j is sufficient to cover the interval $[a_j, b_j)$ at least $\frac{\varepsilon}{\varepsilon-\delta}$ times. More precisely, $Y_f, \dots, Y_g$ will satisfy

(i) $\bigcup_{j=f}^{g} Y_j \subset X_f^g$,
(ii) $|Y_j| = \lambda$, and
(iii) $Y_j \subset \{x \in X_f^g : p_x \ge \beta p_j\}$ for every $f \le j \le g$.

Then, (ii) and (iii) imply $\sum_{y \in Y_j} p_y \ge \lambda\beta p_j = \frac{\varepsilon}{\varepsilon-\delta}(b_j - a_j)$. Thus, if we choose x among those jobs in X_f^g that OPT completes last and guarantee that $x \notin \bigcup_{j=f}^{g} Y_j$, the volume condition (V) is satisfied. We first describe how to find $Y_f, \dots, Y_g$ before we show that these sets satisfy (i) to (iii).

By assumption, $|X_f| > \lambda$. Let $X_f = \{x_1, \dots, x_\lambda, x_{\lambda+1}, \dots\}$ be indexed in increasing completion times C_x^*. Define $Y_f := \{x_1, \dots, x_\lambda\}$ and $L_f := \{x_{\lambda+1}, \dots\} = X_f \setminus Y_f$, i.e., Y_f contains the λ jobs in X_f that OPT completes first and L_f contains the last jobs. For $f < j + 1 \le g$, let $Y_f, \dots, Y_j$ and L_j be defined. By assumption, $|X_{j+1} \cup L_j| > \lambda$ since $|Y_i| = \lambda$ for $1 \le i \le j$. The jobs in $X_{j+1} \cup L_j = \{x_1, \dots, x_\lambda, x_{\lambda+1}, \dots\}$ are again indexed in increasing order of optimal completion times. Then, $Y_{j+1} := \{x_1, \dots, x_\lambda\}$ and $L_{j+1} := \{x_{\lambda+1}, \dots\}$. Since we move jobs only horizontally to later siblings, we call this procedure **Push Forward**.

By definition, (i) and (ii) are satisfied. Since $f, \dots, g$ are leaves, the jobs in $Y_j \cap X_j$ are big w.r.t. j. Thus, it remains to show that the jobs in L_j are big w.r.t. the next job $j + 1$.

To this end, we observe the following. Assume that the jobs in $Y_f, \dots, Y_j$ are big w.r.t. $f, \dots, j$, respectively. If we find an index $f \le i(x) \le j$ such that x as well as the jobs in $\bigcup_{i=i(x)}^{j} Y_i$ are released after $a_{i(x)}$, i.e.,

$$a_{i(x)} \le r_y \quad \text{for } y = x \text{ or } y \in \bigcup_{i=i(x)}^{j} Y_i, \tag{2}$$

and x completes after every $y \in \bigcup_{i=i(x)}^{j} Y_i$, i.e.,

$$C_y^* \le C_x^* \quad \text{for } y \in \bigcup_{i=i(x)}^{j} Y_i, \tag{3}$$

then we can apply the Volume Lemma to show that $x \in L_j$ is big w.r.t. $j+1$. Indeed, then

$$\sum_{i=i(x)}^{j} \sum_{y \in X_i : C_y^* \le C_x^*} p_y \ge p_x + \sum_{i=i(x)}^{j} \sum_{y \in Y_i} p_y \ge p_x + \sum_{i=i(x)}^{j} \frac{\varepsilon}{\varepsilon - \delta}(b_i - a_i)$$
$$= \frac{\varepsilon}{\varepsilon - \delta}(b_j - a_{i(x)}) + p_x.$$

We show by induction that such an index $i(x)$ exists for every $x \in L_j$.

Since $Y_f \subset X_f$, we set $i(x) := f$ for $x \in L_f$. By definition of L_f, $C_y^* \le C_x^*$ for $y \in Y_f$ and $x \in L_f$. Hence, applying the Volume Lemma as explained above shows $p_x \ge \beta p_{f+1}$.

Let $f < j < g$. Assume that $Y_f, \ldots, Y_j$ and L_j are defined as described above. For jobs $x \in L_j \backslash X_j \subset L_{j-1}$, we have $i(x)$ with the Properties (ii) and (iii) by induction. For $x \in L_j \cap X_j$, we temporarily set $i(x) := j$ for simplification. We have to distinguish two cases: $i(x)$ also satisfies (ii) and (iii) for j or we have to adjust $i(x)$. Fix $x \in L_j$.

- $L_i \cap Y_j = \emptyset$ for every $f \le i < i(x)$. Since only jobs in L_i are shifted to some later job j, this implies $\bigcup_{i=f}^{i(x)-1} X_i \cap Y_j = \emptyset$. Thus, the jobs in Y_j are released after $a_{i(x)}$ and by definition, $C_y^* \le C_x^*$ for $y \in Y_j$. By induction, x and the jobs in $Y_{i(x)} \cup \cdots \cup Y_{j-1}$ satisfy (ii) and (iii). Hence, $i(x)$ is a suitable choice for x and j.
- $L_i \cap Y_j \neq \emptyset$ for some $f \le i < i(x)$. Choose the job $y \in L_{j-1} \cap Y_j$ with the smallest $i(y)$. By a similar argumentation as before, $\bigcup_{i=f}^{i(y)-1} X_i \cap Y_j = \emptyset$, which implies (ii) for $z \in Y_j$. Again by induction, y and the jobs in $Y_{i(y)} \cup \cdots \cup Y_{j-1}$ satisfy (ii) and (iii). Since $x \in L_j$, $C_x^* \ge C_z^*$ for all $z \in Y_j$. This implies $C_x^* \ge C_z^*$ for $z \in \bigcup_{i=i(y)}^{j-1} Y_i$ because $y \in L_{j-1} \cap Y_j$. Set $i(x) := i(y)$.

As explained above, the Volume Lemma implies $p_x \ge \beta p_{j+1}$.

The same argumentation holds for $j = g$ although in this special case, Corollary 1 implies the statement. □

We can now generalize the above described procedure to arbitrary strings of jobs in the interruption tree and, thus, prove Lemma 5.

Proof (Lemma 5) We show that for every $j \in J \cup \{M\}$, there exists a partition $(Y_k)_{k \in T_{-j}}$ with

(i) $\bigcup_{k \in T_{-j}} Y_k = X_j^S$,
(ii) $Y_k \subset \{x \in X_j : p_x \ge \beta p_k\}$, and
(iii) $|Y_k| \le \lambda$ for every $k \in T_{-j}$.

Then, it holds that $|X_j^S| = |\bigcup_{k \in T_{-j}} Y_k| = \sum_{k \in T_{-j}} |Y_k| \le \lambda \tau_j$ and, thus, the lemma follows.

The proof consists of an outer and an inner induction. The outer induction is on the distance $\varphi(j)$ of a job j from machine job M, i.e., $\varphi(M) := 0$ and $\varphi(j) := \varphi(\pi(j))+1$ for $j \in J$. The inner induction uses the idea about pushing jobs $x \in X_j$ to some later sibling of j in the same string of jobs (see proof of Lemma 7).

Let $j \in J$ with $\varphi(j) = \varphi_{\max} - 1 := \max\{\varphi(i) : i \in J\} - 1$. By Observation 2, $X_j^S = \bigcup_{k:\pi(k)=j} X_k$, where all $k \in T_{-j}$ are leaves at maximal distance from M. We distinguish three cases for $k \in T_{-j}$:

Case I If $k \in T_{-j}$ is isolated, $|X_k| \leq \lambda$ follows directly from the Volume Lemma as otherwise $\sum_{x \in X_k} p_x \geq \lambda\beta p_k + p_x = \frac{\varepsilon}{\varepsilon-\delta}(b_k - a_k) + p_x$ contradicts Corollary 1, where $x \in X_k$ is the last job that OPT completes from the set X_k. Since all jobs in X_k are big w.r.t. k, we set $Y_k := X_k$.

Case II If $k \in T_{-j}$ with $|X_k| > \lambda$ is part of a string, let $f, \ldots, g$ be the *maximal* string satisfying Lemma 7 with $k \in \{f, \ldots, g\}$. With this lemma, we find $Y_f, \ldots, Y_g$ and set $Y_{g+1} := X_{g+1} \cup L_g$.

Case III We have not yet considered jobs k in a string with $|X_k| \leq \lambda$ that do not have siblings $f, \ldots, g$ in the same string with $b_g = a_k$ and $\sum_{i=f}^{g} |X_j| > (g - f)\lambda$. This means that such jobs do not receive jobs $x \in X_i$ for $i \neq k$ by the **Push Forward** procedure in Case II. For such $k \in T_{-j}$ we define $Y_k := X_k$.

Then, $X_j^S = \bigcup_{k:\pi(k)=j} X_k = \bigcup_{k \in T_{-j}} X_k = \bigcup_{k \in T_{-j}} Y_k$ and, thus, (i) to (iii) are satisfied.

Let $\varphi < \varphi_{\max}$ such that $(Y_k)_{k \in T_{-j}}$ satisfying (i) to (iii) exists for all $j \in J$ with $\varphi(j) \geq \varphi$. Fix $j \in J$ with $\varphi(j) = \varphi - 1$. By induction and Observation 2, it holds that $X_j^S = \bigcup_{k:\pi(k)=j} \left(X_k^B \cup \bigcup_{i \in T_{-k}} Y_i\right)$. Now, we use the partitions $(Y_i)_{i \in T_{-k}}$ for k with $\pi(k) = j$ as starting point to find the partition $(Y_k)_{k \in T_{-j}}$. Fix k with $\pi(k) = j$ and distinguish again the same three cases as before.

Case I If k is isolated, we show that $|X_k| \leq \lambda(\tau_k + 1)$ and develop a procedure to find $(Y_i)_{i \in T_k}$. Assume for sake of contradiction that $|X_k| > \lambda(\tau_k + 1)$ and index the jobs in X_k in increasing order of completion times, i.e., $X_k = \{x_1, \ldots, x_{\lambda(\tau_k+1)}, x_{\lambda(\tau_k+1)+1}, \ldots\}$, and set $L := \{x_{\lambda(\tau_k+1)+1}, \ldots\}$. Then,

$$|X_k^B \backslash L| = |X_k \backslash L| - |X_k^S \backslash L| = (\tau_k + 1)\lambda - \sum_{i \in T_{-k}} |Y_i \backslash L| = \lambda + \sum_{i \in T_{-k}} (\lambda - |Y_i \backslash L|).$$

By induction hypothesis, $\lambda - |Y_i \backslash L| \geq 0$ for $i \in T_{-k}$. Let Y_k contain λ arbitrary big jobs in $X_k^B \backslash L$ and assign each Y_i for $i \in T_{-k}$ exactly $\lambda - |Y_i \backslash L|$ of the remaining (big) jobs in $X_k^B \backslash L$. This is possible because the jobs in X_k^B are big for any descendant of k, i.e., they satisfy (ii). By choice of λ, each of the just obtained sets covers the region of the corresponding job at least $\frac{\varepsilon}{\varepsilon-\delta}$ times. Thus, the jobs in $X_k \backslash L$ have a total processing volume of at least $\frac{\varepsilon}{\varepsilon-\delta}(b_k - a_k)$. Therefore, any job $x \in L$ satisfies (V) which contradicts the fact that k is isolated by Corollary 1. Thus, $|X_k| \leq \lambda(\tau_k + 1)$.

To construct $(Y_i)_{i \in T_k}$, we assign $\min\{\lambda, |X_k^B|\}$ jobs from X_k^B to Y_k. If $|X_k^B| > \lambda$, distribute the remaining jobs according to $\lambda - |Y_i|$ among the descendants of k. Then, $X_k = \bigcup_{i \in T_k} Y_i$. Because a job that is big w.r.t job k is also big w.r.t. all descendants of k, every (new) set Y_i satisfies (ii) and (iii). We refer to this procedure as **Push Down** since jobs are shifted vertically to descendants.

Case II If $|X_k| > \lambda(\tau_k + 1)$, k must belong to a string with similar properties as described in Lemma 7, i.e., there are jobs $f, \ldots, g$ containing k such that

1. $\sum_{j=f}^{i} |X_j| > \lambda \sum_{j=f}^{i} \tau_j$ for all $f \leq i \leq g$ and
2. $b_j = a_{j+1}$ for all $f \leq j < g$.

Choose $\{f, \ldots, g\}$ maximal with those two properties. We show that the Volume Lemma implies the existence of another sibling $g + 1$ that balances the sets $X_f, \ldots, X_g, X_{g+1}$. This is done by using the **Push Down** procedure within a generalization of the **Push Forward** procedure.

As the jobs $f, \ldots, g$ may have descendants, we use **Push Forward** to construct the sets $Z_f, \ldots, Z_g$ and $L_f, \ldots, L_g$ with $|Z_k| = \lambda(\tau_k + 1)$. Then, we show that we can apply **Push Down** to Z_k and $(Y_i)_{i \in T_{-k}}$ in order to obtain $(Y_i)_{i \in T_k}$. This means the newly obtained partition satisfies

(iv) $Y_k \cup \bigcup_{i \in T_{-k}} Y_i = Z_k$,
(v) $Y_i \subset \{x \in X_j : p_x \geq \beta p_i\}$ and
(vi) $|Y_i| = \lambda$ for every $i \in T_k$.

This implies that the set Z_k covers $[a_k, b_k)$ at least $\frac{\varepsilon}{\varepsilon - \delta}$ times. Thus, the sets X_k with $f \leq k \leq g$ satisfy (V) and we can apply Corollary 1.

To define $Z_f, \ldots, Z_g$, we index the jobs in $X_f = \{x_1, \ldots, x_{\lambda_f}, x_{\lambda(\tau_f+1)+1}, \ldots\}$ in increasing order of optimal completion times and set $Z_f := \{x_1, \ldots, x_{\lambda(\tau_f+1)}\}$ and $L_f = X_f \backslash Z_f$. Assume that $Z_f, \ldots, Z_k$ and $L_f, \ldots, L_k$ are defined. Index the jobs in $X_{k+1} \cup L_k = \{x_1, \ldots, x_{\lambda(\tau_{k+1}+1)}, x_{\lambda(\tau_{k+1}+1)+1}, \ldots\}$ in increasing order of completion times and set $Z_{k+1} := \{x_1, \ldots, x_{\lambda(\tau_{k+1}+1)}\}$ and $L_{k+1} = (X_{k+1} \cup L_k) \backslash Z_{k+1}$. Use the **Push Down** procedure to obtain the partition $(Y_i)_{i \in T_k}$.

If we can show that any job $x \in L_k$ is big w.r.t. $k + 1$, we have that $Z_{k+1} \backslash X_{k+1}^S$ only contains big jobs w.r.t. $k + 1$, which are also big w.r.t. every $i \in T_{-(k+1)}$. As in Case I,

$$|Z_{k+1} \backslash X_{k+1}^S| = |Z_{k+1}| - |X_{k+1}^S \backslash L_{k+1}| = \lambda + \sum_{i \in T_{-(k+1)}} (\lambda - |Y_i \backslash L_{k+1}|).$$

Hence, the just defined partition $(Y_i)_{i \in T_k}$ satisfies (iv) to (vi).

As in the proof for Lemma 7, we show by induction that every $x \in L_k$ exhibits an index $i(x)$ with

$$a_{i(x)} \leq r_y \tag{4}$$
$$C_y^* \leq C_x^* \tag{5}$$

for $y = x$ or $y \in \bigcup_{i=i(x)}^{j} Z_i$. Then, the Volume Lemma implies that $p_x \geq \beta p_{k+1}$.

For $x \in L_f$, set $i(x) = f$. Thus, Eqs. (4) and (5) are trivially satisfied. Since $Z_f \subset X_f$, we have that $Z_f \backslash X_f^S$ only contains big jobs w.r.t. f.

Let $f < k < g$. Assume that $Z_f, \ldots, Z_k$ and L_k are defined as described above. For jobs $x \in L_k \backslash X_k$, we have $i(x)$ with the Properties (iv) and (v) by induction. For $x \in L_k \cap X_k$, we temporarily set $i(x) := k$ for simplification. We have to distinguish two cases: $i(x)$ also satisfies (iv) and (v) for k or we have to adjust $i(x)$. Fix $x \in L_k$.

- $L_i \cap Z_k = \emptyset$ for every $f \leq i < i(x)$. Since only jobs in L_i are shifted to some later job k, this implies $\bigcup_{i=f}^{i(x)-1} X_i \cap Z_k = \emptyset$. Thus, the jobs in Z_k are released after $a_{i(x)}$ and by definition, $C_y^* \leq C_x^*$ for $y \in Z_k$. By induction, x and the jobs in $Z_{i(x)} \cup \cdots \cup Z_{k-1}$ satisfy (iv) and (v). Hence, $i(x)$ is a suitable choice for x and k.
- $L_i \cap Z_k \neq \emptyset$ for $f \leq i < i(x)$. Choose the job $y \in L_{k-1} \cap Z_k$ with the smallest $i(y)$. By a similar argumentation as before, $\bigcup_{i=f}^{i(y)-1} X_i \cap Z_k = \emptyset$, which implies (iv) for $z \in Z_k$. Again by induction, y and the jobs in $Z_{i(y)} \cup \cdots \cup Z_{k-1}$ satisfy (iv) and (v). Since $x \in L_k$, $C_x^* \geq C_z^*$ for all $z \in Z_k$. This implies $C_x^* \geq C_z^*$ for $z \in \bigcup_{i=i(y)}^{k-1} Z_i$ because $y \in L_{k-1} \cap Y_k$. Set $i(x) := i(y)$.

As explained above, the Volume Lemma implies $p_x \geq \beta p_{k+1}$.

For $k+1 = g$, the above argumentation can be combined with Corollary 1 to prove that the sibling $g+1$ indeed exists. Set $Z_{g+1} := X_{g+1} \cup L_g$ and use **Push Down** to construct $(Y_i)_{i \in T_{(g+1)}}$.

Case III Any job k with $\pi(k) = j$ that is part of a string and was not yet considered must satisfy $|X_k| \leq (\tau_k + 1)\lambda$. We use the **Push Down** procedure for isolated jobs to get the partition $(Y_i)_{i \in T_k}$.

Hence, we have found $(Y_k)_{k \in T_{-j}}$ with the properties (iv) to (vi). □

We can now prove the main result of this section.

Proof (Theorem 4) As explained before, the job set scheduled by Opt clearly is a subset of $X \cup J$, the union of jobs only scheduled by Opt and the jobs admitted by the region algorithm. Thus, it suffices to prove that $|X| \leq \lambda |J|$. By Observation 2, $X = X_M^S$ and, hence, $|X_M^S| \leq \lambda |J|$ implies $|X| \leq \lambda |J|$. This is true as Lemma 5 also holds for the machine job M. □

Finalizing the proofs of Theorems 1 and 2

Proof (Theorem 1) Set $\alpha = 1$ and $\beta = \frac{\varepsilon}{4}$. Theorem 3 shows that our algorithm completes at least half of all admitted jobs on time. Theorem 4 implies that the region algorithm is $\frac{16}{\varepsilon}$-competitive. □

Proof (Theorem 2) By Lemma 4, the choice $\alpha = \frac{8}{\delta}$ and $\beta = \frac{\delta}{4}$ implies that the region algorithm completes all admitted jobs. Theorem 4 implies that our algorithm is $(\frac{32\varepsilon}{(\varepsilon-\delta)\delta^2} + 1)$-competitive. □

5 Tightness of the region algorithm

In this section, we consider scheduling with commitment. We show that the analysis of the region algorithm is tight in the sense that the competitive ratio of the region algorithm is $\Omega(\alpha/\beta)$. Moreover, we give examples that show that for the commitment upon admission model the choice $\alpha \in \Omega(1/\varepsilon)$ and $\beta \in O(1/\varepsilon)$ is best possible.

In Sect. 6.1 we show that the region algorithm is best possible (up to constants) for scheduling without commitment.

Lemma 8 *Let $0 < \varepsilon \leq 1$, $\alpha \geq 1$, and $0 < \beta < 1$. Then, the competitive ratio of the region algorithm is bounded from below by α/β.*

Proof We consider an instance where a job 0 with processing time $p_0 = 1$ and a huge scheduling interval $[r_0, r_0 + \alpha + 2)$ is released first. Then, the region algorithm blocks the region $[r_0, r_0 + \alpha]$ for this job. During this interval, $\lfloor \alpha/\beta \rfloor$ jobs of size $p_j = \beta$ arrive. They all fit into $R(0)$ but the jobs are to big relative to 0 to be admitted. Then, an offline optimum would process all small jobs until $r_0 + \alpha$ before starting job 0. Hence, the region algorithm completes one job while it is optimal to complete $\lfloor \alpha/\beta \rfloor + 1$ jobs.

More formally, let $r_0 = 0$, $p_0 = 1$ and $d_0 = \alpha + 1$. Fix $0 < \varphi < \beta < 1$. For $1 \leq j \leq \lfloor \alpha/\beta \rfloor$ let $r_j = (j-1)\beta + \varphi$, $p_j = \beta$ and $d_j = r_j + (1+\varepsilon)p_j$. The region algorithm admits job 0 at time 0 and blocks the interval $[0, \alpha)$ for 0. Thus, the region algorithm cannot admit any of the small jobs and completes only job 0. This behavior does not depend on the commitment model.

An optimal offline algorithm processes the jobs $1, \ldots, \lfloor \alpha/\beta \rfloor$ one after the other in the interval $[\varphi, \lfloor \alpha/\beta \rfloor \beta + \varphi) \subset [0, \alpha + 1)$. At the latest at time $\alpha + 1$ job 0 starts processing and finishes on time.

Thus, the competitive ratio of the algorithm is bounded from below by $\lfloor \alpha/\beta \rfloor + 1 \geq \alpha/\beta$. □

Lemma 9 *The competitive ratio of the region algorithm in the scheduling with commitment model is bounded from below by* $\Omega(1/\varepsilon^2)$.

Proof The proof consists of two parts. First we show an upper bound on the choice of β in terms of δ. Then, we use this observation to show an upper bound on β depending on α.

It is obvious that $\beta \leq \delta$ must hold as otherwise a job that is admitted at $d_j - (1+\delta)p_j$ and interrupted by another job i with $p_i = \beta p_j$ cannot finish on time. Hence, $\beta \leq \delta \leq 1$ must hold.

We define a family of instances $\mathcal{I}_m(c)$ that depends on two natural numbers $m, c \in \mathbb{N}$ where c is chosen such that

$$\frac{1}{\beta(c+1)} < \alpha \leq \frac{1}{\beta c}. \tag{6}$$

Each instance consists of four types of jobs, a job 0 that cannot be finished unless α and β satisfy certain bounds, an auxiliary job -1 that guarantees that 0 is not admitted before $d_0 - (1+\delta)p_0$ and two sets of jobs, $B(c)$ and $G(m)$, that block as much time in $[a_0, d_0)$ as possible. A visualization of the instance can be seen in Fig. 2.

More precisely, at time $t = 0$, an auxiliary job -1 is released with $p_{-1} = 1$ and $d_{-1} = (1+\varepsilon)p_{-1}$. The region algorithm admits this job and assigns it the region $R(-1) = [0, \alpha)$. At time $\alpha - (\varepsilon - \delta)$ job 0 is released with $p_0 = 1$ and $d_0 = \alpha + 1 + \delta$. Obviously, this job is admitted at time α as it is still available. Fix $\varphi > 0$ sufficiently small.

At time $\alpha + \varphi$ the sequence $B(c)$ of c identical jobs is released one after the other such that the release date of one job coincides with the end of the region of the previous job. For $0 \leq i \leq c-1$, a tight job is released at $r_i := \alpha + i/c + \varphi$ with processing time $p_i = \beta - \varphi$ and deadline $d_i = r_i + (1+\varepsilon)p_i$. Since

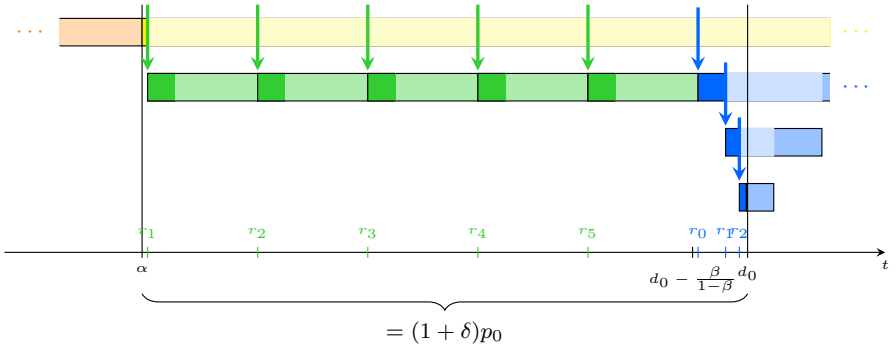

Fig. 2 The structure of the regions and the schedule generated by the region algorithm when faced with the instance $\mathcal{I}_m(c)$. The darkest shades of a color mean that jobs are scheduled there. The light yellow and blue parts show that the region is currently interrupted. The only time slots where 0 can be processed are the lighter parts of the green regions, i.e., the regions belonging to $B(c)$ (color figure online)

$$r_i + \alpha p_i \leq \alpha + i/c + \varphi + \beta/(\beta c) - \alpha\varphi < \alpha + (i+1)/c + \varphi = r_{i+1}$$

each of these jobs is admitted by the region algorithm at their release date. The last of these regions ends at $\alpha + (c-1)/c + \varphi + \alpha(\beta - \varphi) = \alpha + \frac{c-1}{c} + \varphi + \frac{1}{c} - \alpha\varphi \leq \alpha + 1$. Thus, in the limit $\varphi \to 0$, they block $c\beta$ units of time in $[a_j, a_j + d_j)$.

At time $d_0 - \frac{\beta}{1-\beta}$, a sequence of m geometrically decreasing jobs $G(m)$ is released. For $1 \leq j \leq m$, job j is released at $r_j = d_0 - \frac{\beta}{1-\beta} + \sum_{i=1}^{j} \beta^i$ with processing time $p_j = (\beta - \varphi)^j$ and deadline $d_j = r_j + (1+\varepsilon)p_j$. Then, $p_{j+1} = (\beta - \varphi)p_j < \beta p_j$. Thus, the region algorithm admits each of the m jobs. Again, in the limit $m \to \infty$ and $\varphi \to 0$, the processing volume of $G(m)$ sums up to $\frac{\beta}{1-\beta}$.

Putting the two observations together, we obtain

$$\frac{\beta}{1-\beta} + c\beta \leq \delta$$

as otherwise job 0 cannot finish on time. Hence, $0 \leq c\beta^2 - (1 + c + \delta)\beta + \delta$. Solving for the two roots, β_+ and β_-, we obtain

$$\beta_+ = \frac{1 + c + \delta + \sqrt{(1+c+\delta)^2 - 4c\delta}}{2c} \geq \frac{1 + c + \delta + \sqrt{c^2 + \delta^2 - 2c\delta}}{2c} > 1.$$

As we have seen by the first example, $\beta \leq 1$ must hold. Thus, we conclude that the only valid choice for β is in the interval $(0, \beta_-)$. By a similar calculation, it follows that $\beta_- \leq \frac{\delta}{c}$. As we know by Lemma 8, the competitive ratio is bounded from below by α/β. Combined with the two bounds on α, $\frac{1}{\beta(c+1)} < \alpha \leq \frac{1}{\beta c}$, we obtain

$$\frac{\alpha}{\beta} \geq \frac{1}{\beta(c+1)} \frac{1}{\beta} = \frac{c^2}{\delta^2(c+1)}.$$

Since the right hand side is increasing in c for positive c, the expression is minimized for $c = 1$. This implies that $\beta \in \mathcal{O}(\varepsilon)$ and therefore $\alpha \in \Omega(1/\varepsilon)$. □

6 Lower bounds on the competitive ratio

In this section we give a collection of lower bounds on the competitive ratio in the different commitment models and for different problem settings. To simplify notation, we formally introduce the notion of *laxity*. Let j be a job with processing time p_j, deadline d_j, and r_j. The laxity ℓ_j is defined as $d_j - r_j - p_j$.

6.1 Scheduling without commitment

We give a lower bound matching our upper bound in Theorem 2. This shows that the region algorithm is best possible for scheduling without commitment.

Theorem 5 *Every deterministic online algorithm has a competitive ratio* $\Omega(\frac{1}{\varepsilon})$.

Proof The proof idea is as follows: we release $\Omega(\frac{1}{\varepsilon})$ *levels* of jobs. In each level, the release date of any but the first job is the deadline of the previous job. Whenever an online algorithm decides to complete a job from level i (provided no further jobs are released), then the release of jobs in level i stops and a sequence of $\mathcal{O}(\frac{1}{\varepsilon})$ jobs in level $i + 1$ is released. Jobs in level $i + 1$ have processing time that is too large to fit in the slack of a job of level i. Thus, an algorithm has to discard the job started at level i to run a job of level $i + 1$. This implies that it can only finish one job, while the optimum can finish a job from every other level.

Formally, let $\varepsilon < \frac{1}{10}$ such that $\frac{1}{8\varepsilon} \in \mathbb{N}$ and suppose there is an online algorithm with competitive ratio $c < \frac{1}{8\varepsilon}$, from which it is sufficient to deduce a contradiction. We construct an adversarial instance in which each job j belongs to one of $2 \cdot \lceil c + 1 \rceil$ levels and fulfills $d_j = r_j + (1 + \varepsilon) \cdot p_j$. The processing time for any job j in level i are $p_j = p^{(i)} = (2\varepsilon)^i$.

This (along with the interval structure) makes sure that no two jobs from consecutive levels can both be completed by a single schedule, which we will use to show that the online algorithm can only complete a single job throughout the entire instance. The decrease in processing times between levels, however, also makes sure that the optimum can finish a job from every other level, resulting in an objective value of $\lceil c + 1 \rceil$, which is a contradiction to the algorithm being c-competitive.

The sequence starts in level 0 at time 0 with the release of one job j with processing time $p^{(0)} = 1$ and, thus, deadline $d_j = 1 + \varepsilon$. We will show inductively that, for each level i, there is a time t_i when there is only a single job j_i left that the algorithm can still finish, and this job is from the current level i (and, thus, $p_{j_i} = p^{(i)} = (2\varepsilon)^i$). We will also make sure that at t_i at most a $(\frac{2}{3})$-fraction of the time window of j_i has passed. From t_i on, no further jobs from level i are released, and jobs from level $i + 1$ start being released (or, if $i = 2 \cdot \lceil c + 1 \rceil - 1$, we stop releasing jobs altogether). It is clear that t_0 exists.

Consider some time t_i, and we will release jobs from level $i+1$ so as to create time t_{i+1}. The first job j from level $i+1$ has release date t_i and, by the above constraints, $d_j = t_i + (1+\varepsilon)\cdot p_j$ where $p_j = p^{(i+1)} = (2\varepsilon)^{i+1}$. As long as no situation occurs that fits the above description of t_{i+1}, we release an additional job of level $i+1$ at the deadline of the previous job from this level (with identical time-window length and processing time). We show that we can find time t_{i+1} before $\frac{1}{8\varepsilon}$ jobs from level $i+1$ have been released. Note that the deadline of the $\frac{1}{8\varepsilon}$th job from level $i+1$ is $t_i + \frac{1}{8\varepsilon}\cdot(1+\varepsilon)\cdot 2\varepsilon\cdot p^{(i)}$, which is smaller than the deadline of d_{j_i} since by induction $d_{j_i} - t_i \geq \frac{2}{3}\cdot p^{(i)}$ and $\varepsilon < \frac{1}{10}$. This shows that, unless more than $\frac{1}{8\varepsilon}$ jobs from level $i+1$ are released (which will not happen as we will show), all time windows of jobs from layer $i+1$ are contained in that of j_i.

Note that there must be a job $j^\star$ among the $\frac{1}{8\varepsilon}$ first ones in level $i+1$ that the algorithm completes if no further jobs are released within the time window of $j^\star$: by induction, the algorithm can only hope to finish a single job released before time t_i and the optimum could complete $\frac{1}{8\varepsilon}$ jobs from level $i+1$, so $j^\star$ must exist for the algorithm to be c-competitive. Now we can define j_{i+1} to be the first such job $j^\star$ and find t_{i+1} within its time window: at the release date of $j^\star$, the algorithm could only complete j_i. However, since the algorithm finishes j_{i+1} if there are no further jobs released, and $\varepsilon < \frac{1}{10}$, it must have worked on j_{i+1} for more than $\frac{p^{(i+1)}}{2}$ units of time until $r_{i+1} + \frac{2}{3}\cdot p^{(i+1)} =: t_{i+1}$. This quantity, however, exceeds the laxity of j_i, meaning that the algorithm cannot finish j_i any more. (Recall that the laxity of j_i is $\varepsilon p^{(i)} = 2^i\varepsilon^{i+1}$.) So t_{i+1} has the desired properties.

This defines $t_{2\cdot\lceil c+1\rceil}$, and indeed the algorithm will only finish a single job. We verify that an optimal algorithm can schedule a job from every other level. Note that, among levels of either parity, processing times are decreasing by a factor of $4\varepsilon^2$ between consecutive levels. So, for any job j, the total processing time of jobs other than j that need to be processed within the time window of j adds up to less than

$$\begin{aligned}\sum_{i=1}^{\infty}(4\varepsilon^2)^i\cdot p_j &= 4\varepsilon^2\cdot\sum_{i=0}^{\infty}(4\varepsilon^2)^i\cdot p_j\\ &= 4\varepsilon^2\cdot\frac{1}{1-4\varepsilon^2}\cdot p_j \leq \varepsilon\cdot\frac{4}{10}\cdot\frac{1}{1-\frac{4}{100}}\cdot p_j < \varepsilon\cdot p_j = \ell_j.\end{aligned}$$

This completes the proof. □

6.2 Scheduling with commitment

6.2.1 Commitment upon arrival

We strengthen earlier results for weighted jobs [22,26] and show that the model is hopeless even in the unweighted setting and even for randomized algorithms.

Theorem 6 *No randomized online algorithm has a bounded competitive ratio for commitment upon arrival.*

In the proof of the theorem, we use the following algebraic fact.

Lemma 10 *Consider positive numbers $n_1, \ldots, n_k, c \in \mathbb{R}_+$ with the following properties:*

(i) $\sum_{i=1}^{k} n_i \leq 1$,
(ii) $\sum_{i=1}^{j} n_i \cdot 2^{i-1} \geq \frac{2^{j-1}}{c}$ *for all* $j = 1, \ldots, k$.

Then it holds that $c \geq \frac{k+1}{2}$.

Proof We take a weighted sum over all inequalities in (ii), where the weight of the inequality corresponding to $j < k$ is 2^{k-j-1} and the weight of the inequality corresponding to $j = k$ is 1. The result is

$$\sum_{i=1}^{k} n_i \cdot 2^{k-1} \geq \frac{(k+1) \cdot 2^{k-2}}{c} \quad \Leftrightarrow \quad \sum_{i=1}^{k} n_i \geq \frac{(k+1)}{2c}.$$

If $c < \frac{k+1}{2}$, this contradicts (i). □

We proceed to the proof of the theorem.

Proof (Theorem 6) Consider any $\varepsilon > 0$ and arbitrary $\gamma \in (0, 1)$. Suppose there is a (possibly randomized) c-competitive algorithm, where c may depend on ε.

We will choose some $k \in \mathbb{N}$ later. The adversary releases at most k *waves* of jobs, but the instance may end after any wave. Wave i has 2^i jobs. Each job from the ith wave has release date $\frac{i}{k} \cdot \gamma$, deadline 1, and processing time $\frac{1}{2^i} \cdot \frac{1-\gamma}{1+\varepsilon}$. Note that choosing $p_j \leq \frac{1-\gamma}{1+\varepsilon}$ for all jobs j makes sure that indeed $\ell_j \geq \varepsilon \cdot p_j$, and observe that the total volume of jobs in wave i adds up to no more than $1 - \gamma$.

Define n_i to be the expected total processing time of jobs that the algorithm accepts from wave i. We observe:

(i) Since all accepted jobs have to be scheduled within the interval $[0, 1]$, we must have $\sum_{i=1}^{k} n_i \leq 1$.
(ii) For each i, possibly no further jobs are released after wave i. Since, in this case, the optimum schedules all jobs from wave i and the jobs' processing times decrease by a factor of 2 from wave to wave, it must hold that $\sum_{i=1}^{j} n_i \cdot 2^{i-1} \geq \frac{2^{j-1}}{c}$.

This establishes the conditions necessary to apply Lemma 10 to $n_1, \ldots, n_k$, which shows that choosing $k \geq 2c$ yields a contradiction. □

6.2.2 Commitment on job admission and δ-commitment

Since these models are more restrictive than scheduling without commitment, the lower bound $\Omega(\frac{1}{\varepsilon})$ from Theorem 5 holds. In the present setting we can provide a much simpler (but asymptotically equally strong) lower bound.

Commitment upon admission For scheduling with arbitrary weights, Azar et al. [2] rule out any bounded competitive ratio for deterministic algorithms. Thus, our bounded competitive ratio for the unweighted setting (Theorem 2) gives a clear separation between the weighted and the unweighted setting.

Scheduling with δ-commitment We give a lower bound depending on parameters ε and δ.

Theorem 7 *Consider scheduling weighted jobs in the δ-commitment model. For any $\delta > 0$ and any ε with $\delta \leq \varepsilon < 1+\delta$, no deterministic online algorithm has a bounded competitive ratio.*

Proof We reuse the idea of [2] to release the next job upon admission of the previous one while heavily increasing the weights of subsequent jobs. However, the scheduling models differ in the fact that the δ-commitment model allows for processing before commitment which is not allowed in the commitment-upon-admission model.

Assume for the sake of contradiction, that there is a c-competitive algorithm. We consider the following instance that consists of n tight jobs with the same deadline $d := 1+\varepsilon$. Job j has a weight of $w_j := (c+1)^j$ which implies that any c-competitive algorithm has to admit job j at some point even if all jobs $1, \ldots, j-1$ are admitted. In the δ-commitment model, the admission cannot happen later than $d - (1+\delta)p_j$ which is the point in time when job $j+1$ is released.

More precisely, the first job is released at $r_1 = 0$ with processing time $p_1 = 1$. If jobs $1, \ldots, j$ have been released, job $j+1$ is released at $r_{j+1} = d - (1+\delta)p_j$ and has processing time

$$\frac{d - r_{j+1}}{1+\varepsilon} = \frac{d - (d - (1+\delta)p_j)}{1+\varepsilon} = \frac{1+\delta}{1+\varepsilon} p_j = \cdots = \left(\frac{1+\delta}{1+\varepsilon}\right)^{j-1}.$$

An instance with n such jobs has a total processing volume of

$$\sum_{j=1}^{n} p_j = \sum_{j=0}^{n-1} \left(\frac{1+\delta}{1+\varepsilon}\right)^j = \frac{1 - \left(\frac{1+\delta}{1+\varepsilon}\right)^n}{1 - \frac{1+\delta}{1+\varepsilon}}.$$

Any c-competitive algorithm has to complete the n jobs before $d = 1+\varepsilon$. This also holds for $n \to \infty$ and, thus, $\frac{1+\varepsilon}{\varepsilon-\delta} \leq 1+\varepsilon$ is implied. This is equivalent to $\varepsilon \geq 1+\delta$. In other words, if $\varepsilon < 1+\delta$, there is no deterministic c-competitive online algorithm. □

In particular, there is no bounded competitive ratio possible for $\varepsilon \in (0, 1)$. A restriction for ε appears to be necessary as Azar et al. [2] provide such a bound when the slackness is sufficiently large, i.e, $\varepsilon > 3$. In fact, our bound answers affirmatively the open question in [2] if high slackness is indeed required. Again, this strong impossibility result gives a clear separation between the weighted and the unweighted problem as we show in the unweighted setting a bounded competitive ratio for any $\varepsilon > 0$ (Theorem 2).

Proportional weights ($w_j = p_j$) For scheduling with commitment, it is known that simple greedy algorithms achieve the best possible competitive ratio $\Theta(1/\varepsilon)$ [11,13]. In this section, we show a weaker lower bound for randomized algorithms.

Theorem 8 *Consider proportional weights* ($w_j = p_j$). *For commitment on job admission and the δ-commitment model, the competitive ratio of any randomized algorithm is $\Omega(\log \frac{1}{\varepsilon})$.*

Proof Let $k = \lfloor \log(\frac{1}{8\varepsilon}) \rfloor$, and consider a c-competitive algorithm. The adversary releases at most k jobs, where job $j = 1, \ldots, k$ arrives at $r_j = 2\varepsilon \sum_{i=1}^{j-1} 2^{i-1}$, has processing time 2^{j-1} and slack $\varepsilon 2^{j-1}$.

Denote by n_i the probability that the algorithm commits to job i. We make the following observations:

(i) The release date of job j is

$$2\varepsilon \sum_{i=1}^{j-1} 2^{i-1} < 2\varepsilon \cdot 2^{\log(\frac{1}{8\varepsilon})} \leq \frac{1}{4},$$

at which time any job $j' < j$ that the algorithm has committed to has at least $p_1 - 1/4 = 3/4$ processing left. The slack of j is however only at most

$$\varepsilon \cdot 2^{j-1} \leq \varepsilon \cdot 2^{\lfloor \log(\frac{1}{8\varepsilon}) \rfloor - 1} \leq \frac{1}{16}.$$

This implies that no two jobs can both be committed to at the same time. Hence, $\sum_{i=1}^{k} n_i \leq 1$.

(ii) The algorithm has to commit to $j < k$ at the latest at

$$r_j + \varepsilon 2^{j-1} = 2\varepsilon \sum_{i=1}^{j-1} 2^{i-1} + \varepsilon 2^{j-1} < 2\varepsilon \sum_{i=1}^{j} 2^{i-1} = r_{j+1},$$

that is, unknowingly whether $j+1$ will be released or not, so it has to be competitive with the optimum that only schedules j. Hence, we have $\sum_{i=1}^{j} n_i \cdot 2^{i-1} \geq \frac{2^{j-1}}{c}$.

This allows us to apply Lemma 10 to $n_1, \ldots, n_k$, showing $c \geq \frac{k+1}{2} \Omega(\log \frac{1}{\varepsilon})$. □

7 Concluding remarks

We provide a general framework for online deadline-sensitive scheduling with and without commitment. This is the first unifying approach and we believe that it captures well (using parameters) the key design principles needed when scheduling *online, deadline-sensitive* and *with commitment*.

We give the first rigorous bounds on the competitive ratio for maximizing throughput in different commitment models. Some gaps between upper and lower bounds

remain and, clearly, it would be interesting to close them. In fact, the lower bound comes from scheduling without commitment and it is unclear whether scheduling with commitment is truly harder than without. It is somewhat surprising that essentially the same algorithm (with the same parameters and commitment upon admission) performs well for both, commitment upon admission and δ-commitment, whereas a close relation between the models does not seem immediate. It remains open, if an algorithm can exploit the seemingly greater flexibility of δ-commitment.

We restrict our investigations to unit-weight jobs which is justified by strong impossibility results (Theorems 6, 7, [2,22,26]). Thus, for weighted throughput a rethinking of the model is needed. A major difficulty is the interleaving structure of time intervals which makes the problem intractable in combination with weights. However, practically relevant restrictions to special structures such as laminar or agreeable intervals have been proven to be substantially better tractable in related online deadline scheduling research [9,10].

Furthermore, we close the problem of scheduling unweighted jobs without commitment with a best-achievable competitive ratio $\Theta(\frac{1}{\varepsilon})$. It remains open if the weighted setting is indeed harder than the unweighted setting or if the upper bound $\mathcal{O}(\frac{1}{\varepsilon^2})$ in [22] can be improved. Future research on generalizations to multi-processors seems highly relevant. We believe that our general framework is a promising starting point.

Acknowledgements Open access funding provided by Projekt DEAL. Research of the first author was partly supported by NSF Grant 1756014. Research of the second and third author was partly supported by the German Science Foundation (DFG) under contract ME 3825/1. Research of the fourth author was partly supported by CONICYT Grant PII 20150140 and the DAAD PRIME program. Research of the fifth author was partly supported by NSF Grants CCF-1714818 and CCF-1822809.

Appendix

In this section we give the technical details of the modifications used in the proof of Lemma 3.

Lemma 3 *Let* $\alpha = 1$ *and* $\beta = \frac{\varepsilon}{4}$, *with* $\varepsilon > 0$. *If* $b_j - a_j \geq (\ell + 1)p_j$ *for* $\ell > 0$, *then* $|F_j| - |U_j| \geq \lfloor \frac{4\ell}{\varepsilon} \rfloor$.

Proof Assume for the sake of contradiction that there is an instance such that the interruption tree generated by the region algorithm contains a subtree T_j with $b_j - a_j \geq (\ell + 1)p_j$ and $|F_j| - |U_j| < \lfloor \frac{4\ell}{\varepsilon} \rfloor$. Let $\mathcal{I}$ be such an instance with minimal number of jobs in total. By Lemma 2, we restrict to the jobs contained in T_j. The goal is to construct an instance $\mathcal{I}'$ that satisfies $b_j - a_j \geq (\ell + 1)p_j$ and $|F_j| - |U_j| < \lfloor \frac{4\ell}{\varepsilon} \rfloor$

although it uses less jobs than $\mathcal{I}$. To this end, we modify $\mathcal{I}$ in several steps such that we can merge three jobs to one larger job without violating $b_j - a_j \geq (\ell+1)p_j$, changing $|F_j|$ or $|U_j|$, or making the instance infeasible. The three jobs will be leaves with the same parent i in T_j. In fact, if i is an unfinished job, then $b_i - a_i \geq (1+\frac{\varepsilon}{2})p_i$. Any job k that may postpone i satisfies $p_k < \beta p_i = \frac{\varepsilon}{4}p_i$. Hence, if the children of i are all leaves, there have to be at least three jobs that interrupt i.

In the following argumentation we use the position of jobs in the interruption tree. To that end, we define the height of an interruption tree to be the length of a longest path from root to leaf and the height of the node j in the tree to be the height of T_j.

The roadmap of the proof is as follows: first, we show that there is an instance $\mathcal{I}'$ with an unfinished job of height one by proving the following facts.

1. The height of the interruption tree T_j is at least two.
2. Any finished job of height one can be replaced by a leaf.

Let i be an unfinished job of height one. We show that three of its children can be merged and become a sibling of their former ancestor. To this end, we prove that we can assume w.l.o.g. the following two properties of the children of i.

3. No job is completely scheduled in $[d_i, b_i)$.
4. The children of i form a string of jobs.

Ad 1. We show that the height of T_j is at least two. Assume for the sake of contradiction that T_j is a star centered at j. Since any leaf finishes by definition of the region algorithm, the root j is the only job that could possibly be unfinished. As $|F_j| - |U_j| \leq \lfloor \frac{4\ell}{\varepsilon} \rfloor - 1$, this implies that there are at most $\lfloor \frac{4\ell}{\varepsilon} \rfloor$ leaves in T_j. Then,

$$b_j - a_j = \sum_{i \in T_j} p_i < p_j + \left\lfloor \frac{4\ell}{\varepsilon} \right\rfloor \cdot \frac{\varepsilon}{4} p_j \leq p_j + \ell p_j,$$

where we used $p_i < \beta p_j$ for each leaf $i \in T_j$. This contradicts the assumption $b_j - a_j \geq (\ell+1)p_j$.

Ad 2. We show that w.l.o.g. any region of height one ends after the corresponding deadline. Let i be a finished job of height one, i.e., $b_i \leq d_i$, let l be the last completing child of i, and let $\pi(i)$ denote the parent of i. This job $\pi(i) \in T_j$ must exist because the height of T_j is at least two by 1. We create a new instance by replacing l by a new job l' that is released at $r_{l'} := b_i - p_l$ and that has the same processing time as l, i.e., $p_{l'} := p_l$. The deadline is $d_{l'} := r_{l'} + (1+\varepsilon)p_{l'}$. We argue that i finishes at $r_{l'}$ in the new instance and that l' finishes at b_i.
As l is not interrupted, $r_{l'} - a_l = b_i - p_l - a_l = b_i - b_l$, which is the remaining processing time of i at a_l. If we can show that i is not preempted in $[a_l, r_{l'})$ in the new instance, i completes at $a_l + b_i - b_l = r_{l'}$. Since l is the last child of i, any job k released within $[a_l, b_i)$ is scheduled later than b_i. (Recall that we restrict to jobs in the interruption tree T_j by Lemma 2.) Thus, $p_k \geq \beta p_i > p_l$. Hence, i is not interrupted in $[a_l, r_{l'})$ and completes at $r_{l'} < b_i \leq d_i$. At time $r_{l'}$, job l' is the smallest available job and satisfies $p_{l'} < (\frac{\varepsilon}{4})p_i < (\frac{\varepsilon}{4})^2 p_{\pi(i)}$. Thus, l' is admitted at $r_{l'}$ and is not interrupted until $r_{l'} + p_l = b_i$ by the

same argumentation about the jobs k that are released in $[a_l, b_i)$. Hence, its region ends at $r_{l'} + p_{l'} < d_{l'}$. Moreover, outside the interval $[a_l, b_i)$ neither the instance nor the schedule of the regions are changed. Since l' is now released outside of the region of i, l' becomes a child of $\pi(i)$, i.e., l' is directly appended to $\pi(i)$. This modification does not alter the length of $[a_j, b_j)$ or the number of finished and unfinished jobs. Inductively applying this modification to any finished job of height one proves the claim.

Next, we prove the simplifying assumptions on jobs of height one. Because of the just proven statements, T_j must contain at least one unfinished job of height one. Let i be such a job. Since i is unfinished, it must hold that $d_i < b_i$. For simplicity, let $T := T_{-i}$ denote the set of children of i and let $\tau := |T|$.

Ad 3. We start by showing that no region of child of i is completely contained in $[d_i, b_i)$. If there is a child c with $d_i \leq a_c < b_c < b_i$, it does not prevent the algorithm from finishing i. Hence, it could be appended to $\pi(i)$ in the same way as we appended the last child of an finished job in the previous claim. That is, we can create a new instance in which c is appended to $\pi(i)$ and i is still unfinished.

Ad 4. We show that the regions of the children of i form an interval with endpoint $\max\{d_i, b_{\max}\}$ where $b_{\max} := \max_{c \in T} b_c$. We further prove that they are released and admitted in increasing order of their processing times. More formally, we index the children in increasing order of their processing times, i.e, $p_{c_1} \leq p_{c_2} \leq \cdots \leq p_{c_\tau}$. Then, we create a new instance with modified release dates such that one child is released upon completion of the previous child. This means $r_{c'_\tau} := \max\{d_i, b_{\max}\} - p_{c_\tau}$ and $r_{c'_{h-1}} := r_{c'_h} - p_{c'_{h-1}}$ for $1 < h \leq \tau$ where the processing times are not changed, i.e., $p_{c'_h} = p_{c_h}$. In order to ensure that the modified instance is still feasible, we adapt the deadlines $d_{c'_h} := r_{c'_h} + (1+\varepsilon)p_{c'_h}$.
It is left to show that the modifications did not affect the number of finished or unfinished jobs. Obviously, the region algorithm still admits every job in T'. A job $k \notin T'$ released in $[a_i, b_i)$ satisfies $p_k \geq \beta p_i > p_c > \beta p_c$ for all $c \in T$. Hence, these jobs do not interrupt either i or any of the children. They are still scheduled after b_i and every child $c \in T$ completes before its deadline. We also need to prove that i still cannot finish on time. If $b_{\max} \leq d_i$, the region of every child is completely contained in $[a_i, d_i)$. Hence, job i is still interrupted for the same amount of time before d_i in $\mathcal{I}'$ as it is in $\mathcal{I}$. Thus, $b'_i = a_i + p_i + \sum_{h=1}^{\tau} p_{c'_h} = a_i + p_i + \sum_{c \in T} p_c = b_i > d_i$. If $b_{\max} > d_i$, let l denote the child in $\mathcal{I}$ with $b_l = b_{\max}$. Then, $r_{c'_\tau} = b_{\max} - p_{c'_\tau} \leq b_l - p_l < d_i$, where we used that no child is completely processed in $[d_i, b_i)$ and that c'_τ is the child of i with the largest processing time. Thus, the delay of i in $[a_i, d_i)$ is identical to $\sum_{c \in T} p_c - (b_l - d_i)$. Hence, i still cannot finish on time. In this case, $b'_i = b_i$ holds as well. Hence, the modified jobs in $\mathcal{I}'$ still cover the same interval $[a_i, b_i)$.

So far we have modified $\mathcal{I}$ such that it remains an instance which achieves $|F_j| - |U_j| < \lfloor \frac{4\ell}{\varepsilon} \rfloor$ with a minimum total number of jobs. In the following, we show that

the considered instance does not use a minimal number of jobs in total which implies a contradiction and thus the lemma is proved. We do so by modifying the instance in three steps. In the first step, we merge three jobs in T_{-i} where $i \in T_j$ is an unfinished job of height one such that its children satisfy the Assumptions 3 and 4. In the second step, we replace i by a similar job i' to ensure that the instance is still feasible. In the third step, we adapt jobs $k \notin T_{-i}$ to guarantee that i' is admitted at the right point in time. Then, we we show that the resulting instance covers the same interval $[a_i, b_i)$

Since i is admitted at $a_i \leq d_i - (1+\frac{\varepsilon}{2})p_i$ and not finished by the region algorithm on time, $b_i - a_i \geq (1+\frac{\varepsilon}{2})p_i$. As any job that may postpone the region of i satisfies $p_k < \beta p_i = \varepsilon/4 p_i$, there have to be at least three jobs that interrupt i. Among these, consider the first three jobs c_1, c_2, and c_3 (when ordered in increasing release dates). We create a new instance by deleting c_1, c_2, and c_3 and adding a new job c' such that c' is released at the admission date of i in $\mathcal{I}$ and it merges c_1, c_2 and c_3, i.e., $r_{c'} := a_i$, $p_{c'} := p_{c_1} + p_{c_2} + p_{c_3}$, and $d_{c'} := r_{c'} + (1+\varepsilon)p_{c'}$. In exchange, we remove the jobs c_1, c_2, and c_3 from the instance. This completes our first step of modification.

Second, we replace i by a new job i' that is released at $r_{i'} := a_i + p_{c'}$, has the same processing time, i.e., $p_{i'} = p_i$, and has a deadline $d_{i'} := \max\{d_i, r_{i'} + (1+\varepsilon)p_{i'}\}$.

In the third step of our modification, we replace every job k with $r_k \in [a_i, r_i]$ and $p_k \leq p_i$ by a new job k' that is released slightly after i', i.e., $r_{k'} := r_{i'} + \varrho$ for $\varrho > 0$ sufficiently small. It is important to note that we do not change the processing time or the deadline of k', i.e., $p_{k'} = p_k$ and $d_{k'} = d_k$. This ensures that k' finishes on time if and only if k finishes on time. This modification is feasible, i.e., $d_{k'} - r_{k'} \geq (1+\varepsilon)p_{k'}$, because of two reasons. First,

$$b_i - r_{i'} = b_i - (a_i + p_{c'}) = b_i - a_i - (p_{c_1} + p_{c_2} + p_{c_3}) \geq p_i$$

as c_1, c_2, and c_3 postponed the region of i by their processing times in $\mathcal{I}$. Second, $d_k - b_i \geq (1+\frac{\varepsilon}{2})p_k$ because we only consider jobs that were admitted at some point later than b_i by the region algorithm. Then,

$$\begin{aligned} d_{k'} - r_{k'} = d_k - b_i + b_i - r_{i'} - \varrho &\geq \left(1+\frac{\varepsilon}{2}\right)p_k + p_i - \varrho \\ &\geq \left(2+\frac{\varepsilon}{2}\right)p_k - \varrho \geq (1+\varepsilon)p_{k'}, \end{aligned}$$

where the last but one inequality follows from the fact that only jobs with $p_k \leq p_i$ were affected by the modification and the last inequality is due to the sufficiently small choice of ϱ.

So far, we have already seen that the resulting instance is still feasible. It is left to show that c' completes at $r_{c'} + p_{c'}$ as well as that i' is admitted at $r_{i'}$ and its region ends at b_i.

Since it holds that $p_{c'} < \frac{3\varepsilon}{4}p_i < p_i$, at $a_i = r_{c'}$ the new job c' is the smallest available job and any job that was interrupted by i is preempted by c' as well. The jobs in T_{-i} are released one after the other by Assumption 4 and $r_{c_1} > a_i$. Thus, if i' has at least one child left after the modification, it holds that $r_{c_4} = r_{c_1} + p_{c_1} + p_{c_2} + p_{c_3} = a_i + p_{c'} + (r_{c_1} - a_i) > r_{i'}$. Hence, no remaining child is released in $[r_{c'}, r_{i'}]$ in the

modified instance. Any other job $k \in T_j$ released in $[r_{c'}, r_{i'}]$ satisfies $p_k \geq \frac{\varepsilon}{4} p_i$ as $k \notin T_{-i}$. Because $p_{c'} < p_i$, this implies that $p_k \geq \frac{\varepsilon}{4} p_{c'}$ holds as well, i.e., no such job k interrupts c'. Therefore, c' completes at r'_i.

Job i' is admitted at $r_{i'}$ if it is the smallest available job at that time. We have already proven that none of the remaining children of i is released in $[a_i, r_{i'}]$ that might prevent the region algorithm from admitting i at $r_{i'}$. Furthermore, the third step of our modification guarantees that any job $k \in T_j \backslash T_i$ that is smaller than p_i is released after $r_{i'}$. Therefore, i' is the smallest available job at time $r_{i'}$ by construction, and it is admitted. As argued above, the modified instance is still feasible and the interval $[a_{\pi(i)}, b_{\pi(i)})$ is still completely covered by regions of jobs in $T_{\pi(i)}$.

However, the second step of our modification might lead to $b_{i'} \leq d_{i'}$ which implies that i' finishes on time while i does not finish on time. This changes the values of $|F_j|$ and $|U_j|$. Clearly, in the case that i' completes before $d_{i'}$, $|U'_j| = |U_j| - 1$. By a careful analysis, we see that in this case the number of finished jobs decreases by one as well because the three (on time) jobs c_1, c_2 and c_3 are replaced by only one job that finishes before its deadline. Formally, we charge the completion of c' to c_1, and the completion of i' to c_2 which leaves c_3 to account for the decreasing number of finished jobs. Hence, $|F'_j| - |U'_j| = |F_j| - |U_j|$. If i' does not finish by $d_{i'}$, then $|F'_j| - |U'_j| = (|F_j| - 2) - |U_j|$. Thus, the modified instance $\mathcal{I}'$ also satisfies $|F'_j| - |U'_j| < \lfloor \frac{4\ell}{\varepsilon} \rfloor$ but uses less jobs than $\mathcal{I}$ does. This is a contradiction. □

References

1. Agrawal, K., Li, J., Lu, K., Moseley, B.: Scheduling parallelizable jobs online to maximize throughput. In: Proceedings of the Latin American Theoretical Informatics Symposium (LATIN), pp. 755–776 (2018)
2. Azar, Y., Kalp-Shaltiel, I., Lucier, B., Menache, I., Naor, J., Yaniv, J.: Truthful online scheduling with commitments. In: Proceedings of the ACM Symposium on Economics and Computations (EC), pp. 715–732 (2015)
3. Bansal, N., Chan, H.-L., Pruhs, K.: Competitive algorithms for due date scheduling. In: *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, pp. 28–39 (2007)
4. Baruah, S.K., Haritsa, J.R.: Scheduling for overload in real-time systems. IEEE Trans. Comput. **46**(9), 1034–1039 (1997)
5. Baruah, S.K., Haritsa, J.R., Sharma, N.: On-line scheduling to maximize task completions. In: Proceedings of the IEEE Real-Time Systems Symposium (RTSS), pp. 228–236 (1994)
6. Baruah, S.K., Koren, G., Mao, D., Mishra, B., Raghunathan, A., Rosier, L.E., Shasha, D.E., Wang, F.: On the competitiveness of on-line real-time task scheduling. Real-Time Syst. **4**(2), 125–144 (1992)
7. Canetti, R., Irani, S.: Bounding the power of preemption in randomized scheduling. SIAM J. Comput. **27**(4), 993–1015 (1998)
8. Chen, L., Eberle, F., Megow, N., Schewior, K., Stein, C.: A general framework for handling commitment in online throughput maximization. In: Proceedings of the Conference on Integer Programming and Combinatorial Optimization (IPCO), pp. 141–154 (2019)
9. Chen, L., Megow, N., Schewior, K.: An $\mathcal{O}(\log m)$-competitive algorithm for online machine minimization. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 155–163 (2016)
10. Chen, L., Megow, N., Schewior, K.: The power of migration in online machine minimization. In: Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 175–184 (2016)

11. DasGupta, B., Palis, M.A.: Online real-time preemptive scheduling of jobs with deadlines. In: *Proceedings of the International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pp. 96–107 (2000)
12. Ferguson, A.D., Bodík, P., Kandula, S., Boutin, E., Fonseca, R.: Jockey: guaranteed job latency in data parallel clusters. In: Proceedings of the European Conference on Computer Systems (EuroSys), pp. 99–112 (2012)
13. Garay, J.A., Naor, J., Yener, B., Zhao, P.: On-line admission control and packet scheduling with interleaving. In: Proceedings of the IEEE International Conference on Computer Communications (INFOCOM), pp. 94–103 (2002)
14. Georgiadis, L., Guérin, R., Parekh, A.K.: Optimal multiplexing on a single link: delay and buffer requirements. IEEE Trans. Inf. Theory **43**(5), 1518–1535 (1997)
15. Goldwasser, M.H.: Patience is a virtue: the effect of slack on competitiveness for admission control. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 396–405 (1999)
16. Im, S., Moseley, B.: General profit scheduling and the power of migration on heterogeneous machines. In: Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 165–173 (2016)
17. Kalyanasundaram, B., Pruhs, K.: Maximizing job completions online. J. Algorithms **49**(1), 63–85 (2003)
18. Koren, G., Shasha, D.E.: MOCA: a multiprocessor on-line competitive algorithm for real-time system scheduling. Theor. Comput. Sci. **128**(1–2), 75–97 (1994)
19. Koren, G., Shasha, D.E.: D^{over}: an optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. SIAM J. Comput. **24**(2), 318–339 (1995)
20. Liebeherr, J., Wrege, D.E., Ferrari, D.: Exact admission control for networks with a bounded delay service. IEEE/ACM Trans. Netw. **4**(6), 885–901 (1996)
21. Lipton, R.: Online interval scheduling. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 302–311 (1994)
22. Lucier, B., Menache, I., Naor, J., Yaniv, J.: Efficient online scheduling for deadline-sensitive jobs: extended abstract. In: Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 305–314 (2013)
23. Pruhs, K., Stein, C.: How to schedule when you have to buy your energy. In: Proceedings of the International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APROX), pp. 352–365 (2010)
24. Schwiegelshohn, C., Schwiegelshohn, U.: The power of migration for online slack scheduling. In: Proceedings of the European Symposium of Algorithms (ESA), Vol. 57, pp. 75:1–75:17 (2016)
25. Woeginger, G.J.: On-line scheduling of jobs with fixed start and end times. Theor. Comput. Sci. **130**(1), 5–16 (1994)
26. Yaniv, J.: Job scheduling mechanisms for cloud computing. Ph.D. thesis, Technion, Israel (2017)

Affiliations

Lin Chen[1] · Franziska Eberle[2] · Nicole Megow[2] · Kevin Schewior[3] · Cliff Stein[4]

✉ Franziska Eberle
feberle@uni-bremen.de

Lin Chen
chenlin198662@gmail.com

Nicole Megow
nicole.megow@uni-bremen.de

Kevin Schewior
kschewior@gmail.com

Cliff Stein
cliff@ieor.columbia.edu

1 Department of Computer Science, Texas Tech University, Lubbock, TX, USA

2 Department for Mathematics and Computer Science, University of Bremen, Bremen, Germany

3 Department of Mathematics and Computer Science, Universität zu Köln, Cologne, Germany

4 Department of Industrial Engineering and Operations Research, Columbia University, New York, USA