A General Framework for Handling Commitment in Online Throughput Maximization

Lin Chen[1], Franziska Eberle[2(✉)], Nicole Megow[2], Kevin Schewior[3,4], and Cliff Stein[5]

[1] Department of Computer Science, University of Houston, Houston, TX, USA
chenlin198662@gmail.com
[2] Department for Mathematics/Computer Science, University of Bremen, Bremen, Germany
{feberle,nicole.megow}@uni-bremen.de
[3] Fakultät für Informatik, Technische Universität München, München, Germany
kschewior@gmail.com
[4] Département d'Informatique, École Normale Supérieure, Paris, France
[5] Department of IEOR, Columbia University, New York, USA
cliff@ieor.columbia.edu

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 scheduler 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.

N. Megow—Supported by the German Science Foundation (DFG) Grant ME 3825/1.
K. Schewior—Supported by CONICYT Grant PII 20150140 and DAAD PRIME program.
C. Stein—Research partly supported by NSF Grants CCF-1714818 and CCF-1822809.

A. Lodi and V. Nagarajan (Eds.): IPCO 2019, LNCS 11480, pp. 141–154, 2019.
https://doi.org/10.1007/978-3-030-17953-3_11

1 Introduction

Many modern computing environments, such as internal clusters and public clouds, involve a centralized system for managing the resource allocation of a large diverse workload [21] with a heterogeneous mixture of jobs. In this paper, we will study scheduling policies, evaluated by the commonly used notion of *throughput* which is the number of jobs completed, or if jobs have weights, the total weight of jobs completed. 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 [11]. However, for many industrial applications, service providers have to *commit to complete* admitted jobs since without such a guarantee, some applications will fail or customers may be unhappy and choose another environment.

Formally, we consider a fundamental single-machine scheduling 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 p_j units of processing time in the interval $[r_j, d_j)$. If a schedule completes a set S of jobs, then the *throughput* is $|S|$ while the weighted throughput is $\sum_{j \in S} w_j$. To measure the quality of an online algorithm, we use standard *competitive analysis* where its performance is compared to that of an optimal offline algorithm with full knowledge of the future.

Deadline-based objectives are typically much harder to optimize than other Quality-of-Service (QoS) metrics such as response time or makespan. Indeed, the problem becomes hopeless when *preemption* (interrupting a job and resuming it later) 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. 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 $d_j - r_j \approx p_j$. To bar such jobs from a system, we require that any submitted job contains some *slack*. 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. This slackness parameter captures certain aspects of QoS provisioning and admission control, see, e.g., [13,19], and it has been considered in previous work, e.g., in [2,4,12,14,21,23]. Other results for scheduling with deadlines use speed scaling, which can be viewed as adding slack to the schedule, e.g., [1,3,15,22]. In this paper we quantify the impact that different commitment requirements have on the performance of online algorithms.

1.1 Our Results and Techniques

Our main contribution is a general algorithmic framework, called the **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. Thus, our algorithms are all for unit weights $w_j = 1$.

Optimal Algorithm for Scheduling Without Commitment. We show that the region algorithm achieves a competitive ratio of $\mathcal{O}(\frac{1}{\varepsilon})$, and give 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. We show that no (randomized) online algorithm admits a bounded competitive ratio. Such a lower bound has only been shown by exploiting job weights [21,25]. Hence, we do not consider this model 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. In the second model, an online algorithm must commit to complete a job when its slack has reduced from the original slack requirement of an ε-fraction of the size to a δ-fraction for $0 < \delta < \varepsilon$, modeling an early-enough commitment for mission-critical jobs. We show that implementations of the region algorithm yield a 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—for any commitment model (excluding $w_j = p_j$).

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. (See Sect. 5 for more details.) We rule out bounded performance guarantees for $\varepsilon \in (0,1)$. For sufficiently large slackness ($\varepsilon > 3$), an online algorithm is provided in [2] that has bounded competitive ratio. Our new lower bound answers affirmatively the open question of whether 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 algorithms for weighted throughput.

Our Techniques. Once a job j is admitted to the system, its slack becomes a scarce resource: To complete the job on time 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 the similarity of job lengths.

A major difficulty in the analysis is understanding the complex interval structure formed by feasible time windows, regions, and processing time intervals.

Here, the key ingredient is that regions are defined independently of scheduling decisions. 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 aptly 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 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 as there might be many ($\gg \Theta(\frac{1}{\varepsilon^2})$) 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. 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*).

1.2 Previous Results

Preemptive online scheduling and admission control have been studied rigorously, see, e.g., [5,12,14] and references therein. Impossibility results for jobs with hard deadlines and without slack have been known for decades [6,7,17,18,20].

Most research on online scheduling does not address commitment. The only results independent of slack (or other job-dependent parameters) concern weighted throughput for the special case $w_j = p_j$, where a constant competitive ratio is possible [6,17,18,24]. In the unweighted setting, a randomized $\mathcal{O}(1)$-competitive algorithm is known [16]. For instances with ε-slack, an $\mathcal{O}(\frac{1}{\varepsilon^2})$-competitive algorithm in the general weighted setting is given in [21]. To the best of our knowledge, no lower bound was known to date.

Much less is known for scheduling with commitment. In the most restrictive model, *commitment upon job arrival*, Lucier et al. [21] 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 without a rigorous worst-case bound. Azar et al. [2] show that no bounded competitive ratio is possible for weighted throughput maximization for small ε. For the *δ-commitment model*, [2] design (in the context of truthful mechanisms) an online algorithm that is $\mathcal{O}(\frac{1}{\varepsilon^2})$-competitive for large slack ε. They left open if this latter condition is an inherent property of any committed scheduler in this model which we answer affirmatively. The machine utilization variant ($w_j = p_j$) is better tractable as greedy algorithms achieve the best possible competitive ratio $\Theta(\frac{1}{\varepsilon})$ [10,12] in all mentioned commitment models.

2 Our General Framework

2.1 The Region Algorithm

We now present our general algorithmic framework for scheduling with and without commitment. We assume that the slackness constant $\varepsilon > 0$ and, in the δ-commitment model, $0 < \delta < \varepsilon$ are known to the online algorithm.

Algorithm 1.1. 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 or 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
1. admit job i and reserve region $R(i) = [t, t + \alpha p_i)$,
2. update remaining regions $R(j)$ with $R(j) \cap [t, \infty) \neq \emptyset$ as described below

We first describe informally three underlying design principles. The third principle is crucial to improve on existing results that only use the first two [21].

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

The region algorithm has two parameters, $\alpha \geq 1$ and $0 < \beta < 1$. A *region*, $R(j)$ for job j, is a union of time intervals associated with j, and the size of the region is the sum of sizes of the intervals. 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. 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 $R(j)$ and all later regions are delayed.

Formally, 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 decisions then are independent of the regions: at any point in time, schedule the shortest admitted job that has not completed its processing, i.e., schedule admitted jobs in *Shortest Processing Time (SPT)* order. The 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 creating the region $R(1) := [r_1, r_1 + \alpha p_1)$. Two events – the release of a job and the end of a region– trigger the **region preemption** subroutine. This subroutine 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 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 $j \in J(t) \setminus \{k\}$, 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. See the summary Algorithm 1.1.

We apply the region 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 $\delta = \frac{\varepsilon}{2}$.

If the region algorithm commits to a job, it does so upon admission, which is, for our algorithm, the same as its start time. 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's execution for commitment upon admission ($\delta = \frac{\varepsilon}{2}$) 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 ($0 < \varepsilon \leq 1$) as for $\varepsilon > 1$ we run our algorithm simply by setting $\varepsilon = 1$ and obtain constant competitive ratios.

Without commitment, we give an optimal online algorithm which is an exponential improvement upon a previous result [21] (given for weighted throughput). For scheduling with commitment, we give the first rigorous upper bound.

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

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 $\beta = \frac{\varepsilon}{8}$.*

2.3 Interruption Trees

To analyze the performance of the region algorithm, we retrospectively consider the final schedule and the final regions. Let a_j be the admission date of job j which was not changed while executing the algorithm. Let b_j denote the end point of j's region. Then, the convex hull of $R(j)$ is given by $\text{conv}(R(j)) = [a_j, b_j)$.

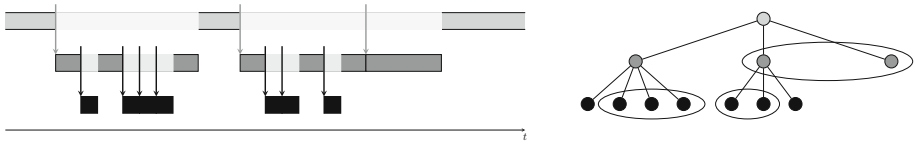

Fig. 1. Gantt chart of the regions (left) and the interruption tree (right)

Our analysis crucially relies on understanding the interleaving structure of the regions that the algorithm constructs. We use 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.

Let $\pi(j)$ denote the *parent* of j. 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 abusing notation, we denote the tree/forest as well as its jobs by T_*. A key property of this tree is that the processing times on a path are geometrically decreasing.

Lemma 1. *Let $j_1, \dots, j_\ell$ 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 $\sum_{i=1}^{\ell} p_{j_i} \leq \sum_{i=1}^{\ell} \beta^i p_j \leq \frac{\beta}{1-\beta} \cdot p_j$.*

3 Successfully Completing Sufficiently Many Jobs

We show that the region algorithm completes sufficiently many jobs among the admitted jobs on time, when the parameters α, β, and δ are chosen properly.
Scheduling Without Commitment. Let $\delta = \frac{\varepsilon}{2}$ for $0 < \varepsilon \leq 1$.

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, due to the scheduling order SPT, a job is always prioritized within its own region and, in the model without commitment, a job does not need to block extra time in the future to ensure the completion of

preempted jobs. In order to prove Theorem 3, we show that a late job j implies that the subtree T_j rooted in j contains more finished than unfinished jobs.

Scheduling With Commitment. For both models, commitment at admission and δ-commitment, we give conditions on the choice of α, β, and δ such that every admitted job will complete before its deadline. We restrict in the analysis to the δ-commitment model since the algorithm otherwise runs with $\delta = \frac{\varepsilon}{2}$.

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

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

any job j admitted by the algorithm at time $a_j \leq d_j - (1 + \delta)p_j$ finishes by d_j.

For any admitted job j, we consider two types of descendants in the interruption subtree T_j whose regions intersect $[a_j, d_j)$: *(i)* jobs k with $d_j \in conv(R(k))$ form a path in T_j and, thus, Lemma 1 bounds their total processing volume from above by $\frac{\beta}{1-\beta} p_j$, *(ii)* jobs k with $R(k) \subset [a_j, d_j)$ reserve an $(\frac{\alpha-1}{\alpha})$-fraction of $R(k)$ for processing j. Thus, a straightforward calculation implies Theorem 4.

4 Competitiveness: Admission of Sufficiently Many Jobs

Theorem 5. *The number of jobs that an optimal (offline) algorithm can complete on time is by at most a multiplicative 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 5.

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 2, starting with the leaves in the interruption tree as base case (Appendix, Lemma 4). For the construction of $(Y_j)_{j \in J}$, we heavily rely on the key property (Volume Lemma 3) 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 1

1. *Any job $x \in X$ that is scheduled by* Opt *and that is 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 1, $X = X_M^S$ 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 2. *For all $j \in J \cup \{M\}$, $|X_j^S| \leq \lambda \tau_j$.*

A proof sketch can be found in the appendix. We highlight the main steps here. 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 1.

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 *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 3 (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.

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 2 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.

Proof of Theorem 5. 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 1, $|X_M^S| \leq \lambda |J|$ implies $|X| \leq \lambda |J|$. This holds by applying Lemma 2 to the machine job M. □

Finalizing the Proofs of Theorems 1 and 2

Proof of Theorem 1. Set $\alpha = 1$ and $\beta = \frac{\varepsilon}{4}$. By Theorem 3 at least half of all admitted jobs complete on time. Theorem 5 implies the competitive ratio $16/\varepsilon$. □
Proof of Theorem 2. Theorem 4, $\alpha = \frac{8}{\delta}$ and $\beta = \frac{\delta}{4}$ imply that the algorithm completes all admitted jobs. Theorem 5 implies the competitive ratio $32/((\varepsilon - \delta)\delta^2 + 1)$. □

5 Lower Bounds on the Competitive Ratio

Theorem 6 (Scheduling Without Commitment). *Every deterministic online algorithm has a competitive ratio* $\Omega(\frac{1}{\varepsilon})$.

Theorem 7 (Commitment Upon Arrival). *No randomized online algorithm has a bounded competitive ratio for commitment upon arrival.*

Theorem 8 (δ-Commitment). *Consider weighted jobs in the δ-commitment model. For any $\delta > 0$ and ε with $\delta \leq \varepsilon < 1 + \delta$, no deterministic online algorithm has a bounded competitive ratio.*

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 an upper bound for sufficiently large slackness, i.e., $\varepsilon > 3$. We answer affirmatively the open question in [2] if high slackness is indeed required. Again, this strong impossibility result clearly separates the weighted and the unweighted problem as we show in the unweighted setting a bounded competitive ratio for any $\varepsilon > 0$ (Theorem 2).

6 Concluding Remarks

We provide a general framework for online scheduling of deadline-sensitive jobs 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*. 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, if scheduling with commitment is truly harder than without. It is

somewhat surprising that essentially the same algorithm performs well for both commitment models, 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.

Our focus on unit-weight jobs is justified by strong impossibility results (Theorem 7, 8, [2,21,25]). Thus, for weighted throughput a rethinking of the model is needed. A major difficulty seems to be the interleaving structure of time intervals as special structures (laminar or agreeable intervals) have been proven to be substantially better tractable in related research [8,9].

Finally, while 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 [21] can be improved. Future research on generalizations to multi-processors seems highly relevant. We believe that our general framework is a promising starting point.

A Appendix

Lemma 4. *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 (Sketch). Observe that $[a_f, b_g) = \bigcup_{j=f}^{k} R(g)$ because the leaves $f,\dots,g$ form a string of jobs. Thus, by showing that there is a job $x \in X_f^g := \bigcup_{j=f}^{g} X_j$ satisfying (V), we prove the lemma with the Volume Lemma. We show that for every job $f \le j \le g$ there is a set Y_j such that the processing volume of Y_j covers the interval $[a_j, b_j)$ at least $\frac{\varepsilon}{\varepsilon-\delta}$ times. More precisely, $Y_f,\dots,Y_g$ satisfy

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

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 $x \notin \bigcup_{j=f}^{g} Y_j$ and x is among those jobs in X_f^g that Opt completes last, (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, $|X_f| > \lambda$. Index the jobs in $X_f = \{x_1,\dots,x_\lambda,x_{\lambda+1},\dots\}$ in increasing completion times C_x^*. Define $Y_f := \{x_1,\dots,x_\lambda\}$ and $L_f := X_f \backslash Y_f$. Let $Y_f,\dots,Y_j$ and L_j be defined for $f < j+1 \le g$. By assumption, $|X_{j+1} \cup L_j| > \lambda$ since $|Y_i| = \lambda$ for $f \le i \le j$. We again index the jobs in $X_{j+1} \cup L_j = \{x_1,\dots,x_\lambda,x_{\lambda+1},\dots\}$ in increasing 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 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 $a_{i(x)}$ and x completes after

every $y \in \bigcup_{i=i(x)}^{j} Y_i$, then the Volume Lemma 3 implies that $x \in L_j$ is big w.r.t. $j+1$. Indeed, then $\sum_{i=i(x)}^{j} \sum_{y \in X_i : C_y^* \le C_x^*} p_y \ge p_x + \sum_{i=i(x)}^{j} \sum_{y \in Y_i} p_y \ge \frac{\varepsilon}{\varepsilon - \delta}(b_j - a_{i(x)}) + p_x$. By induction, we show the existence of such an index $i(x)$.

By the same argumentation for $j = g$, Corollary 1 implies the lemma. □

Lemma 2. *For all $j \in J \cup \{M\}$, $|X_j^S| \le \lambda \tau_j$.*

Proof (Sketch). Recall that T_j is the subtree of the interruption tree rooted in $j \in J$ while the forest T_{-j} is T_j without its root j. We show that for all $j \in J \cup \{M\}$ there exists a partition $(Y_k)_{k \in T_{-j}}$ with

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

Then, $|X_j^S| = |\bigcup_{k \in T_{-j}} Y_k| = \sum_{k \in T_{-j}} |Y_k| \le \tau_j \lambda$ 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 $j \in J$. Let $\varphi_{\max} := \max\{\varphi(i) : i \in J\}$. 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 4).

Let $j \in J$ with $\varphi(j) = \varphi_{\max} - 1$. By Observation 1, $X_j^S = \bigcup_{k:\pi(k)=j} X_k$, where all $k \in T_{-j}$ are leaves at distance $\varphi_{\max}$ from M. To define Y_k for $k \in T_{-j}$ satisfying (*i*) to (*iii*), we distinguish three cases:

Case 1. If $k \in T_{-j}$ is isolated, $|X_k| \le \lambda$ follows directly from the Volume Lemma as otherwise $\sum_{x \in X_k} p_x \ge \lambda \beta p_k + p_x = \frac{\varepsilon}{\varepsilon - \delta}(b_k - a_k) + p_x$ 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 2. For $k \in T_{-j}$ with $|X_k| > \lambda$, we find $Y_f, \ldots, Y_g$ with Lemma 4 and set $Y_{g+1} := X_{g+1} \cup L_g$ where $f \le k \le g$ (maximal) satisfy Lemma 2.

Case 3. Consider jobs k in a string with $|X_k| \le \lambda$ without 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 $i \ne k$ by the **Push Forward** procedure in Case 2. For such $k \in T_{-j}$ we define $Y_k := X_k$ as in Case 1.

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

We use induction to extend the claim for $\varphi = \varphi_{\max}$ to all $0 \le \varphi \le \varphi_{\max}$. Let $\varphi < \varphi_{\max}$ such that $(Y_k)_{k \in T_{-j}}$ satisfying (*i*) to (*iii*) exists for all $j \in J$ with $\varphi(j) \ge \varphi$. Fix $j \in J$ with $\varphi(j) = \varphi - 1$. By induction and Observation 1, 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 $(Y_k)_{k \in T_{-j}}$. We fix k with $\pi(k) = j$ and distinguish similar three cases as in the base case:

Case 1. If k is isolated, we show that $|X_k| \le (\tau_k + 1)\lambda$ and develop a procedure to find $(Y_i)_{i \in T_k}$.

By induction, $|X_k^S| \le \tau_k \lambda$. In the full version of the paper, we prove that $|X_k^B| \le \lambda + (\tau_k \lambda - |X_k^S|)$. 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 2. If $|X_k| > (\tau_k+1)\lambda$, k must belong to a string with similar properties as in Lemma 4, i.e., there is a maximal string of jobs $f, \ldots, g$ containing k such that $\sum_{j=f}^{i} |X_j| > \lambda \sum_{j=f}^{i} \tau_j$ for $f \leq i \leq g$ and $b_j = a_{j+1}$ for $f \leq j < g$.

If the Volume Condition (V) is satisfied, there exists another sibling $g+1$ that balances the sets $X_f, \ldots, X_g, X_{g+1}$ due to Corollary 1. This is shown by using **Push Down** 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 apply **Push Down** to Z_k and $(Y_i)_{i \in T_{-k}}$ in order to obtain $(Y_i)_{i \in T_k}$ such that they will satisfy $Z_k = \bigcup_{i \in T_k} Y_i$, $Y_i \subset \{x \in X_j : p_x \geq \beta p_i\}$, and $|Y_i| = \lambda$ for $i \in T_k$. Thus, the sets $X_f, \ldots, X_g$ satisfy (V) and we can apply Corollary 1.

Case 3. Any job k with $\pi(k) = j$ that was not yet considered as part of a string must satisfy $|X_k| \leq (\tau_k + 1)\lambda$. We use **Push Down** of Case 1 to get $(Y_i)_{i \in T_k}$. Hence, we have found $(Y_k)_{k \in T_{-j}}$ with the properties (*i*) to (*iii*). □

References

1. Agrawal, K., Li, J., Lu, K., Moseley, B.: Scheduling parallelizable jobs online to maximize throughput. In: Bender, M.A., Farach-Colton, M., Mosteiro, M.A. (eds.) LATIN 2018. LNCS, vol. 10807, pp. 755–776. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77404-6_55
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: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 28–39. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73420-8_5
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., et al.: 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., 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)
9. 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)
10. DasGupta, B., Palis, M.A.: Online real-time preemptive scheduling of jobs with deadlines. In: Jansen, K., Khuller, S. (eds.) APPROX 2000. LNCS, vol. 1913, pp. 96–107. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44436-X_11

11. 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)
12. 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)
13. 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)
14. 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)
15. 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)
16. Kalyanasundaram, B., Pruhs, K.: Maximizing job completions online. J. Algorithms **49**(1), 63–85 (2003)
17. 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)
18. 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)
19. 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)
20. Lipton, R.: Online interval scheduling. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 302–311 (1994)
21. 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)
22. 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)
23. 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)
24. Woeginger, G.J.: On-line scheduling of jobs with fixed start and end times. Theor. Comput. Sci. **130**(1), 5–16 (1994)
25. Yaniv, J.: Job scheduling mechanisms for cloud computing. Ph.D. thesis, Technion, Israel (2017)