Online Competitive Algorithms for Maximizing Weighted Throughput of Unit Jobs

Francis Y. L. Chin [a,1] Marek Chrobak [b,2] Stanley P. Y. Fung [a,1] Wojciech Jawor [b,2] Jiří Sgall [c,*,3] Tomáš Tichý [c,3]

[a] *Department of Computer Science, The University of Hong Kong, Hong Kong*

[b] *Department of Computer Science, University of California, Riverside, CA 92521, U.S.A.*

[c] *Mathematical Institute, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic*

Abstract

We study an online unit-job scheduling problem arising in buffer management. Each job is specified by its release time, deadline, and a nonnegative weight. Due to overloading conditions, some jobs have to be dropped. The goal is to maximize the total weight of scheduled jobs. We present several competitive online algorithms for various versions of unit-job scheduling, as well as some lower bounds on the competitive ratios.

We first give a randomized algorithm RMIX with competitive ratio of $e/(e-1) \approx 1.582$. This is the first algorithm for this problem with competitive ratio smaller than 2.

Then we consider s-bounded instances, where the span of each job (deadline minus release time) is at most s. We give a 1.25-competitive randomized algorithm for 2-bounded instances, matching the known lower bound. We also give a deterministic algorithm EDF_α, whose competitive ratio on s-bounded instances is $2-2/s+o(1/s)$. For 3-bounded instances its ratio is $\phi \approx 1.618$, matching the known lower bound.

In s-uniform instances, the span of each job is exactly s. We show that no randomized algorithm can be better than 1.25-competitive on s-uniform instances, if the span s is unbounded. For $s = 2$, our proof gives a lower bound of $4 - 2\sqrt{2} \approx 1.172$. Also, in the 2-uniform case, we prove a lower bound of $\sqrt{2} \approx 1.414$ for deterministic memoryless algorithms, matching a known upper bound.

Finally, we investigate the multiprocessor case and give an $1/(1 - (\frac{m}{m+1})^m)$-competitive algorithm for m processors. We also show improved lower bounds for the general and s-uniform cases.

Key words: Scheduling, online algorithms, buffer management.

1 Introduction

The online *bounded delay buffer problem* has been recently introduced [14,2,15] to model the trade-offs arising in managing buffers for storing packets in QoS networks. In this formulation, packets arrive and are buffered at network switches. At each integer time step, only one packet can be sent along the link. Each packet is characterized by its QoS value, which can be thought of as a benefit gained by forwarding the packet. Network switches can use this QoS value to prioritize the packets. In order to control the end-to-end delay, each packet has also a deadline that specifies the latest time when the packet can be sent. In overload conditions, some packets will not be sent by their deadline. Such packets do not contribute to the benefit value, and can as well be dropped. The objective is to maximize the total value of the forwarded packets, that is the *weighted throughput*.

It is easy to see that this buffer management problem is equivalent to the following *unit-job scheduling* problem. We are given a set of n unit-length jobs, with each job j specified by a triple (r_j, d_j, w_j) where r_j and d_j are integral release times and deadlines, and w_j is a non-negative real weight. We have a single machine, i.e., one job can be processed at each integer time. We use the term *weighted throughput* or *gain* for the total weight of the jobs completed by their deadline. The goal is to compute a schedule for the given set of jobs that maximizes the *weighted throughput*. In Graham's notation, this problem can be described as $1|p_j = 1, r_j| \sum_j w_j U_j$.

In this paper we focus on the online version of unit-job scheduling, where each job arrives at its release time. At each time step, an online algorithm needs to schedule one of the pending jobs, without the knowledge of the jobs that will be released later in the future. An online algorithm $\mathcal{A}$ is called *R-competitive* if its gain on any instance is at least $1/R$ times the optimum (offline) gain on this instance. The smallest such value R is called the *competitive ratio* of $\mathcal{A}$. The competitive ratio is commonly used as a performance measure for online algorithms, and we adopt this measure in this paper.

---

* Corresponding author.

Email addresses: `chin@cs.hku.hk` (Francis Y. L. Chin), `marek@cs.ucr.edu.` (Marek Chrobak), `pyfung@cs.hku.hk` (Stanley P. Y. Fung), `wojtek@cs.ucr.edu.` (Wojciech Jawor), `sgall@math.cas.cz` (Jiří Sgall), `tichy@math.cas.cz` (Tomáš Tichý).

[1] Supported by an RGC Grant HKU7142/03E.

[2] Supported by NSF grants CCR-9988360 and CCR-0208856.

[3] Partially supported by Institute for Theoretical Computer Science, Prague (project 1M0021620808 of MŠMT ČR), grant IAA1019401 of GA AV ČR and grant 201/05/0124 of GA ČR.

For unit jobs, some restrictions on instances have been proposed in the literature [14,2,9,15]. In *s-bounded instances*, the span of the jobs (defined as the difference between the deadline and the release time) is at most s, and in *s-uniform instances* the span of each job is exactly s. In the context of QoS buffer management, these cases would correspond to models where, in order to reduce the end-to-end delay, only a small amount of delay is allowed at each node [14].

The unit-job scheduling problem is related to another scheduling problem which also arises from QoS applications. In *metered-task model* [6,10], each job is specified by four real numbers: release time, deadline, processing time (not necessarily unit), and weight. Preemptions are allowed. Unlike in classical scheduling, even non-completed jobs contribute to the overall gain. Specifically, the gain of a job is proportional to the amount of this job that was processed. This problem arises as a QoS problem of transferring large images over a network of low bandwidth [6], where the amount of processed tasks (images) corresponds to their resolution. It is also related to a problem called *imprecise computation* in real-time systems, where some tasks are allowed to be executed only partially. Partial execution degrades the quality of some results but it allows to meet the deadlines (see [16]), resulting in an overall benefit for the whole system.

Past work. As shown by Kesselman et al. [14] and, independently, by Hajek [13], the naive greedy algorithm for unit-job scheduling, that always schedules the heaviest job, is 2-competitive. For the deterministic case, a lower bound of $\phi \approx 1.618$ appeared in [13]. Lower bound proofs with the same value appeared also in [2,9]. In the randomized case, Chin and Fung [9] give a lower bound of 1.25. (The proof in [9] was for metered tasks, but it carries over to unit jobs.) Both of these lower bounds apply even to 2-bounded instances.

For the 2-bounded case, a ϕ-competitive (and thus optimal) algorithm was presented in [14]. Deterministic algorithms for 2-uniform instances were studied by Zhu et al. [2], who established a lower bound of $\frac{1}{2}(\sqrt{3}+1) \approx 1.366$ and an upper bound of $\sqrt{2} \approx 1.414$.

Kesselman et al. [14] also studied a version of the buffer management problem in which the output port has *bandwidth* m (that is, m packets at a time can be sent). This corresponds to the problem of scheduling unit-time jobs on m identical processors. The results in [14] include two lower bounds (valid for any m): a lower bound of $4 - 2\sqrt{2} \approx 1.172$ that applies even to the 2-bounded case, and a lower bound of $10/9$ for the 2-uniform case.

Other models of buffer management were considered in the literature [17,1,15]. Closely related to our work is the FIFO model, the buffer has finite capacity, and packets must be forwarded in the same order as they arrive, although some packets can be dropped (in the preemptive version). An online algorithm needs to decide which packets to admit into the buffer, and then which packets to drop. It is known that an R-competitive algorithm for the FIFO model can be modified to obtain a competitive algorithm for the s-uniform case with the same competitive ratio R [15]. Therefore the 1.75-competitive algorithm for the FIFO model given in [3] implies the existence of a 1.75-competitive algorithm for the s-uniform model. More general switches with more output queues also received considerable attention during the last few years; see [12] for a recent survey.

Our results. We present several competitive online algorithms for various versions of unit-job scheduling, as well as some lower bounds on the competitive ratios.

First, in Section 3, we give a randomized algorithm with competitive ratio $e/(e-1) \approx 1.582$, which is the first algorithm for this problem with competitive ratio below 2. In fact, its competitive ratio is smaller than the lower bound of ϕ for deterministic algorithms. Our algorithm has been inspired by the techniques developed in [10] for metered tasks.

For 2-bounded instances, in Section 4 we give a 1.25-competitive randomized algorithm—a substantial improvement over the 1.618 bound for the deterministic case. This ratio is optimal, as it matches a lower bound that follows from the work in [9].

Both results above illustrate the power of randomization for the problem of scheduling unit jobs. We hope that our techniques will contribute to a better understanding of randomization in the context of online scheduling.

In Section 5 we give a deterministic algorithm $\textsc{Edf}_\alpha$ whose competitive ratio on 3-bounded instances is $\phi = 1.618$, matching the lower bound. This result extends previous results from the literature for 2-bounded instances [14], and it provides evidence that a ϕ-competitive deterministic algorithm might be possible for the general case. For 4-bounded instances, $\textsc{Edf}_\alpha$ is $\sqrt{3} \approx 1.732$ competitive, and for s-bounded instances it is $2 - 2/s + o(1/s)$ competitive. However, without the restriction on the span, it is only 2-competitive.

In Section 6 we study the case of s-uniform instances. For randomized algorithms, we prove a lower bound that is increasing with s and approaching 1.25 for large s. Comparing it with our 1.25-competitive randomized algorithm for 2-bounded instances, this gives the first evidence that the competitive ratio for s-uniform instances with large s is not better than for small s. For 2-uniform instances, this gives a lower bound of $4 - 2\sqrt{2} \approx 1.172$ for randomized algorithms. In the deterministic case, we prove a lower bound of $\sqrt{2} \approx 1.414$ on memoryless algorithms for 2-uniform instances. (For the definition of memoryless algorithms, see Section 2.) This matches the previously known upper bound from [2]. We remark that all competitive algorithms for unit-job scheduling in the past literature, as well as in this paper, are memoryless.

The bounds from the previous work and this paper are summarized in Table 1. Our results are marked with [$*$]. A blank entry indicates that the bound in this entry follows from another bound in the same column.

Table 1
Bounds achieved in the previous work and in this paper.

	deterministic		randomized	
	upper	lower	upper	lower
General	2 [14]		1.582... [$*$]	
s-bounded	$2 - 2/s + o(1/s)$ [$*$]			
s-uniform	1.75 [3]			1.25 for $s \to \infty$ [$*$]
4-bounded	1.732... [$*$]			
3-bounded	1.618... [$*$]			
2-bounded	1.618... [14]	1.618... [2,9]	1.25 [$*$]	1.25 [9]
2-uniform	1.414... [2]	1.366... [2]		1.172.... [$*$]

Finally, in Section 7, we study online algorithms for the multiprocessor case, namely $Pm|p_j = 1, r_j| \sum_j w_j U_j$ in Graham's notation. This corresponds to the buffer management problem in which the output port has *bandwidth* m, meaning that it can send m packets at a time. We give a $1/(1 - (\frac{m}{m+1})^m)$-competitive algorithm for the case of m processors. For randomized algorithms, we also show improved lower bounds of 1.25 for the general and s-uniform cases (with $s \to \infty$).

This paper is a full version of the extended abstract [4].

2 Preliminaries

Unit job scheduling. As we noted in the introduction, the QoS buffer management problem is equivalent to the unit-job scheduling problem. We will henceforth use job scheduling terminology in this paper. We number the jobs $1, 2, \ldots, n$. Each job j is specified by a triple (r_j, d_j, w_j), where r_j and d_j are integral release times and deadlines, and w_j is a non-negative real weight.

To simplify terminology and notation, we will sometimes use the weights of jobs to identify jobs. Thus, we will say "job w" meaning "the job with weight w". Even if several jobs of the same weight w are present, it will always be clear from context which job we refer to.

Whenever ties between jobs of equal weight need to be broken, we always break it in favor of lower-indexed jobs. More specifically, we will say that a job j is heavier than a job j', if either $w_j > w_{j'}$, or $w_j = w_{j'}$ and $j < j'$.

A *schedule* S specifies which jobs are executed, and for each executed job j it specifies an integral time t when it is scheduled, where $r_j \leq t < d_j$. Only one job can be scheduled at any given time step. The *throughput* or *gain* of a schedule S on instance I, denoted $gain_S(I)$, is the total weight of the jobs in I that are executed in S. Similarly, if $\mathcal{A}$ is a scheduling algorithm, $gain_{\mathcal{A}}(I)$ is the gain of the schedule computed by $\mathcal{A}$ on I. The optimal gain on I is denoted by $opt(I)$.

We say that an instance is s-*bounded* if $d_j - r_j \leq s$ for all jobs j. Similarly, an instance is s-*uniform* if $d_j - r_j = s$ for all jobs j. The difference $d_j - r_j$ is called the *span* of a job j. A job j is *pending* in schedule S at time t if $r_j \leq t < d_j$ and j has not been scheduled in S before t.

To simplify some arguments, we assume that there is always a pending job of weight 0 and that at each step in which a job is scheduled. Obviously, adding such a job is always possible and does not change the gain of the online algorithm or the optimal schedule.

Earliest-deadline schedules. If X is a set of pending jobs, a job $j \in X$ is called the *earliest-deadline* job in X if, for all $i \in X$, either $d_j < d_i$, or $d_j = d_i$ and $w_j > w_i$, or $d_j = d_i$, $w_j = w_i$, and $j < i$. (The last rule is needed only to break ties in a manner consistent with that for ties between job weights.) The earliest-deadline job in a non-empty set X of pending jobs is always uniquely defined.

We often assume that (offline) schedules are *canonical earliest-deadline*. In such schedules, for each time t, the job that is scheduled at t is chosen as the earliest-deadline job from the set of pending jobs that are executed later in the schedule. Each schedule can be easily converted into an earliest-deadline schedule by rearranging its jobs.

Online algorithms. An algorithm $\mathcal{A}$ is called *online* if, at each time step, it determines which job to execute at this step based only on the information about the already released jobs. Online algorithms are of great importance in real-time scheduling applications, including the buffer management problem.

We will say that $\mathcal{A}$ is *memoryless* if (i) the decision as to which job to execute is based only on the weights of the pending jobs, and (ii) the algorithm is invariant under scaling, that is, multiplying the weights of the pending jobs by the same positive constant does not affect the choice of the job to execute.

We often view the behavior of an online algorithm $\mathcal{A}$ as a game between $\mathcal{A}$ and an adversary. Both algorithms schedule jobs released by the adversary whose objective is to maximize the ratio $opt(I)/gain_{\mathcal{A}}(I)$. Several of the upper bound proofs are based on a *potential function* argument. In such proofs, we define a potential function Φ that maps all possible configurations into real numbers. (In general, a configuration at a given step may include all information about the computation before and after this step, both for the algorithm and the adversary. In most arguments, however, it is sufficient to include only the set of pending jobs in both schedules.) Intuitively, the potential represents $\mathcal{A}$'s savings at a given step. At each time step, an online algorithm and the adversary execute a job. The proofs are based on the following lemma which can be proven by a simple summation over all steps.

Lemma 2.1 *Let $\mathcal{A}$ be an online algorithm for scheduling unit jobs. Let Φ be a potential function that is 0 on configurations with no pending jobs, and at each step satisfies*

$$R \cdot \Delta gain_{\mathcal{A}} \geq \Delta adv + \Delta\Phi, \tag{1}$$

where $\Delta\Phi$ represents the change of the potential, and $\Delta gain_{\mathcal{A}}$, Δadv represent $\mathcal{A}$'s and the adversary gain in this step. Then $\mathcal{A}$ is R-competitive.

The lemma above applies to randomized algorithms as well. In that case we need to prove the inequality on average with respect to the algorithm's random choices at the given step, i.e., to prove that $\mathbf{Exp}[R \cdot \Delta gain_{\mathcal{A}} - \Delta\Phi] \geq \Delta adv$, as both the gain of the algorithm and the change of the potential are influenced by the random choices.

In some proofs, in particular for deterministic algorithms, we use a different approach called *charging*. In a charging scheme, the weight of each of the jobs in the adversary schedule is charged to some time in our schedule, in such a way that for each time t the weight of all jobs charged to t is at most R times the total gain of the job(s) scheduled in our schedule at t. If such a charging scheme exists, by simple summation over all time steps, it implies that our algorithm is R-competitive. A charging scheme can be transformed into a potential method argument, with the potential function at a given time equal to the sum of the charges going backward in time across this time minus the sum of the charges going forward. However, proofs based on charging schemes tend to be more illuminating.

Metered tasks. As discussed in the introduction, our problem is related to the metered-task model. Consider the *discrete* metered-task model, in which jobs have integral release times, deadlines and processing lengths, and the algorithm can only start and preempt jobs at integral times. In the multiprocessor cases, as in classical preemptive scheduling, jobs are allowed to migrate from one machine to another. (In [9] this model is called *non-timesharing.*) Then:

Theorem 2.2 (a) *The unit-job scheduling problem with a single processor is equivalent to the single processor discrete metered-task model.* (b) *The unit-job scheduling problem with m processors is a special case of the m-processor discrete metered-task model; they are equivalent when, in addition, all jobs in the metered-task model are of unit length.*

The theorem can be easily proven by observing that each job in the discrete metered-task model of length p can be transformed into p unit-length jobs, each having the same release time, deadline and weight as the original job. A valid unit-job schedule on the transformed instance then corresponds to a valid discrete metered-task model schedule with the same total value, and vice versa. The transformation in the reverse direction is trivial. For multiprocessor unit-length jobs the transformation is trivial, too. Note that for the multiprocessor case and arbitrary jobs, the problems are not equivalent because a long job cannot be processed by several processors simultaneously, while a number of different unit-length jobs can be processed in this way.

The continuous version of the metered-task model [6,10,9] bears some resemblance to the randomized case of unit-job scheduling. By Theorem 2.2, any randomized algorithm for unit-job scheduling can be transformed into a randomized algorithm for the metered-task model, and it is known that in the continuous version of the metered-task model deterministic and randomized algorithms are equivalent. On the other hand, it is not clear whether the algorithms for the continuous metered-tasks can be automatically translated into randomized algorithms for unit jobs. One may attempt to convert a deterministic algorithm $\mathcal{D}$ for metered tasks into a randomized $\mathcal{R}$ algorithm for unit jobs, by setting the probability of $\mathcal{R}$ executing a given job j to be equal to $\mathcal{D}$'s fraction of the processor power devoted to j. It is, however, not clear how to extend this into a full specification of an algorithm that would match the performance of $\mathcal{D}$.

3 Randomized Algorithm RMix

In this section we give a randomized algorithm for scheduling unit jobs with competitive ratio $e/(e-1) \approx 1.582$.

As noted in the introduction, the greedy algorithm has competitive ratio 2. The example on which its ratio approaches 2 exploits the fact that this algorithm prefers the heaviest job, even in presence of more urgent jobs with almost the same weight. One natural idea for an improvement is to schedule the earliest-deadline pending job among those with weight at least some fixed fraction of the heaviest job. This yields the algorithm EDF_α (see Section 5) which gives an improved competitive ratio for the s-bounded case. Our algorithm RMix presented below sets the threshold randomly; this yields an improved competitive ratio for general instances.

Algorithm RMix. At each step, let h be the heaviest pending job. Select a real $x \in [-1, 0]$ uniformly at random. Let f be the earliest-deadline pending job with $w_f \geq e^x w_h$. Execute f.

Theorem 3.1 *Algorithm* RMix *is* $\frac{e}{e-1} \approx 1.582$*-competitive.*

PROOF. Without loss of generality, we assume that the adversary schedule is canonical earliest-deadline. At a given time step, let X be the set of pending jobs in RMix, and let Y be the set of pending jobs in the adversary schedule that the adversary will schedule in the future. Define the potential $\Phi = \sum_{j \in Y-X} w_j$.

Job arrivals and expirations cannot increase the potential as these jobs are not in $Y - X$: the arriving job is always in X and the expiring job is never in Y by the definition of Y. So we only need to analyze how the potential changes after job execution.

Consider a given time step in which the adversary schedules a job j. By Lemma 2.1, it is sufficient to prove that $\mathbf{Exp}[\frac{e}{e-1} w_f - \Delta\Phi] \geq w_j$, where f is a random variable denoting the job chosen by RMix.

Suppose first that $j \in Y - X$. Executing j by the adversary decreases Φ by w_j. At the same time f is removed from X, which increases Φ by at most w_f. So $\frac{e}{e-1} w_f - \Delta\Phi \geq w_f - \Delta\Phi \geq w_j$ for each f and the same holds on average.

Now assume that $j \in Y \cap X$. Then Φ increases by at most w_f and $w_f - \Delta\Phi \geq 0$, regardless of the choice of random x in RMix. In addition, if $x \leq \ln(w_j/w_h)$, which is equivalent to $w_j \geq e^x w_h$, then we claim that Φ does not change, and thus $w_f - \Delta\Phi = w_f$. In this case, j is sufficiently heavy to be considered for f. By the definition of RMix, either f is before j in the earliest-deadline ordering or $f = j$. The adversary schedule is canonical earliest-deadline and executes j at this step, thus it cannot execute f later. It follows that $f \notin Y$ after this step and Φ does not change. Denoting $z = \max\{-1, \ln(w_j/w_h)\}$, we conclude that, for $x \in [-1, z]$, we have $w_f - \Delta\Phi = w_f$, and for $x \in [z, 0]$, we have $w_f - \Delta\Phi \geq 0$. Averaging over all f we get

$$\begin{aligned}
\mathbf{Exp}\left[\frac{e}{e-1} w_f - \Delta\Phi\right] &= \frac{1}{e-1}\mathbf{Exp}[w_f] + \mathbf{Exp}[w_f - \Delta\Phi] \\
&\geq \frac{1}{e-1}\int_{-1}^{0} e^x w_h\, dx + \int_{-1}^{z} e^x w_h\, dx \\
&= e^z w_h \\
&\geq w_j,
\end{aligned}$$

completing the proof. □

4 An Optimal Randomized Algorithm for 2-Bounded Instances

In this section we give a randomized algorithm for 2-bounded instances with competitive ratio 1.25. This matches the lower bound from [9], and thus completely resolves the 2-bounded case. In addition, our algorithm is memoryless.

For $a, b \geq 0$, define

$$p_{ab} = \begin{cases} 1 & \text{if } a \geq b \\ \frac{4a}{5b} & \text{otherwise} \end{cases}$$

Also, let $q_{ab} = 1 - p_{ab}$. Note that p_{ab} satisfies the following properties for any $a, b \geq 0$:

$$5p_{ab}a \geq 4a - b \tag{2}$$

$$5(p_{ab}a + q_{ab}b) \geq 4b \tag{3}$$

$$5p_{ab}a + 2q_{ab}b \geq 4a \tag{4}$$

$$5p_{ab}a + 2q_{ab}b \geq b \tag{5}$$

Algorithm R2B. Let a and b denote the heaviest jobs of span 1 and span 2, respectively, released at this time step, and c the heaviest pending job (of span 2) issued in the previous step. Let $u = \max(c, a)$. Execute u with probability p_{ub} and b with probability q_{ub}.

Theorem 4.1 *Algorithm* R2B *is a* 1.25*-competitive randomized algorithm for* 2*-bounded instances.*

PROOF. Without loss of generality, we can assume that at each step exactly one job of span 1 is released. All jobs of span 1 except the heaviest one can be simply ignored, and if no job is released, we can introduce a job of weight 0. Similarly, we can assume that at each step (except last) exactly one job of span 2 is released. This can be justified as follows: If, at a given time t, the optimal schedule contains a job of span 2 released at t, we can assume that it is the heaviest such job. A similar statement holds for Algorithm R2B, since its decision at each step depends only on the heaviest job of span 2. Thus all the other jobs of span 2 can be ignored in this step, and treated as if they are released with span 1 in the following time step.

At a given step, the state of R2B is given by a pair $\langle x, \sigma \rangle$, where x is the job of span 2 released in the previous step, and σ is the probability that x was executed in the previous step. Denote by $\bar{\sigma} = 1 - \sigma$ the probability that x is pending in the current step. In other words, the value of c in the algorithm is 0 with probability σ and x with probability $\bar{\sigma}$. To describe the state of the adversary, let $z \in \{0, x\}$ be a variable such that $z = x$ if the adversary has not scheduled x (i.e., x is pending in the adversary schedule) and $z = 0$ if the adversary has no pending job.

We define the potential function for each configuration described by a triple $\langle x, \sigma, z \rangle$. Note that this slightly deviates from the use of the potential method in the randomized case, as discussed after Lemma 2.1. In our case, the potential is a function of the distribution of R2B's current state, and is not a random variable. Nevertheless, Lemma 2.1 still applies.

Let $\Phi_{x\sigma z}$ denote the potential function in the configuration $\langle x, \sigma, z \rangle$. We put $\Phi_{x\sigma z} = 0$ if $z = 0$ and $\Phi_{x\sigma z} = \frac{1}{4} x \cdot \max(5\sigma - 1, 3\sigma)$ if $z = x$.

Consider one step, where the configuration is $\langle x, \sigma, z \rangle$, for $z \in \{0, x\}$, and two jobs a, b are released, of span 1 and span 2, respectively. The new configuration is $\langle b, \sigma', z' \rangle$, where $\sigma' = \sigma q_{ab} + \bar{\sigma} q_{vb}$, for $v = \max(a, x)$, and $z' \in \{0, b\}$. Using Lemma 2.1, we need to show that for each adversary move:

$$R \cdot \mathbf{Exp}[\Delta gain_{\mathrm{R2B}}] - \Phi_{b\sigma' z'} + \Phi_{x\sigma z} \;\geq\; \Delta adv \tag{6}$$

where $\Delta gain_{\mathrm{R2B}}$ is the weight of the randomly chosen job scheduled by R2B and Δadv the weight of the job scheduled by the adversary.

Case 1: Adversary schedules b. Then $\Delta adv = b$ and $z' = 0$. For a fixed value of u in the algorithm, the expected gain of the algorithm is $p_{ub}u + q_{ub}b$ and (3) implies $\frac{5}{4}(p_{ub}u + q_{ub}b) \geq b$. By averaging over $u \in \{a, v\}$ we get $R \cdot \mathbf{Exp}[\Delta gain_{\mathrm{R2B}}] \geq b$. This, together with $\Phi_{x\sigma z} \geq 0$ and $\Phi_{b\sigma' z'} = 0$, implies (6).

Case 2: The adversary does not schedule b. Then $z' = b$ and $\Phi_{b\sigma' z'} = \frac{1}{4} b \cdot \max(5\sigma' - 1, 3\sigma')$. The algorithm executes b with probability $\sigma q_{ab} + \bar{\sigma} q_{vb} = \sigma'$, a with probability σp_{ab}, and v with probability $\bar{\sigma} p_{vb}$, so $\mathbf{Exp}[\Delta gain_{\text{R2B}}] = \sigma' b + \sigma p_{ab} a + \bar{\sigma} p_{vb} v$. Substituting into (6), it is sufficient to prove that

$$\min(b, 2\sigma' b) + 5\sigma p_{ab} a + 5\bar{\sigma} p_{vb} v + 4 \cdot \Phi_{x\sigma z} \geq 4 \cdot \Delta adv \tag{7}$$

Case 2.1: The adversary schedules a. Then $\Delta adv = a \leq v$. Since $\Phi_{x\sigma z} \geq 0$, it is sufficient to show (7) with $\Phi_{x\sigma z}$ replaced by 0. For the first term of the minimum, we use (2) twice and get

$$\begin{aligned} b + 5\sigma p_{ab} a + 5\bar{\sigma} p_{vb} v \ &= \ \sigma(b + 5p_{ab} a) + \bar{\sigma}(b + 5p_{vb} v) \\ &\geq \ 4\sigma a + 4\bar{\sigma} v \ \geq \ 4a. \end{aligned}$$

For the second term of the minimum, we use (4) twice and get

$$\begin{aligned} 2\sigma' b + 5\sigma p_{ab} a + 5\bar{\sigma} p_{vb} v \ &= \ \sigma(5p_{ab} a + 2q_{ab} b) + \bar{\sigma}(5p_{vb} v + 2q_{vb} b) \\ &\geq \ 4\sigma a + 4\bar{\sigma} v \ \geq \ 4a. \end{aligned}$$

Case 2.2: $z = x$ and the adversary schedules z. It must be the case that $v = x \geq a$, as otherwise the adversary would prefer to schedule a. We have $\Delta adv = x$.

If $x \geq b$, then $p_{xb} = 1$. We use $4\Phi_{x\sigma z} = 4\Phi_{x\sigma x} \geq (5\sigma - 1)x$ and obtain

$$5\bar{\sigma} p_{xb} x + 4\Phi_{x\sigma z} \ \geq \ 5\bar{\sigma} x + 5\sigma x - x = 4x,$$

which implies (7).

It remains to consider the case $x < b$. Using (2), (5) and (4) we obtain

$$b + 5\bar{\sigma} p_{xb} x \ \geq \ b + \bar{\sigma}(4x - b) \ = \ \sigma b + 4\bar{\sigma} x$$

and

$$\begin{aligned} 2\sigma' b + 5\sigma p_{ab} a + 5\bar{\sigma} p_{xb} x \ &= \ \sigma(5p_{ab} a + 2q_{ab} b) + \bar{\sigma}(5p_{xb} x + 2q_{xb} b) \\ &\geq \ \sigma b + 4\bar{\sigma} x \end{aligned}$$

Together with $4\Phi_{x\sigma z} = 4\Phi_{x\sigma x} \geq 3\sigma x$ and $x < b$ this implies

$$\min(b, 2\sigma' b) + 5\sigma p_{ab} a + 5\bar{\sigma} p_{xb} x + 4\Psi_{x\sigma z} \ \geq \ \sigma b + 4\bar{\sigma} x + 3\sigma x \ \geq \ 4x,$$

and (7) follows. $\square$

5 Deterministic Algorithms for s-Bounded Instances

The 2-bounded (deterministic) case is now well understood: there exists an online algorithm with competitive ratio ϕ, and no better ratio is possible [2,9,13]. In this section, we extend the upper bound of ϕ to 3-bounded instances by proving that Algorithm $\text{EDF}_{\phi-1}$ is ϕ-competitive.

Algorithm EDF_α: Let h be the heaviest pending job and f be the earliest-deadline pending job such that $w_f \geq \alpha w_h$. Execute f.

Theorem 5.1 *$\text{EDF}_{\phi-1}$ is ϕ-competitive for 3-bounded instances.*

PROOF. We fix a canonical earliest-deadline adversary schedule A. Let E be the schedule computed by $\text{EDF}_{\phi-1}$. We use the following charging scheme: Suppose that j is the job scheduled by the adversary at time t. If j is executed in E before time t, charge j to its copy in E. Otherwise, charge j to the job in E scheduled at time t.

Fix some time step t. Let f and j be the jobs scheduled at time t in E and A, respectively. By the definition of $\text{EDF}_{\phi-1}$, let h be the heaviest pending job in E at time t, and let f be the earliest-deadline job that is pending at time t and satisfies $w_f \geq (\phi - 1)w_h = w_h/\phi$.

Job f receives at most two charges: one from j and one from itself, if f is executed in A at some later time. Ideally, we would like to prove that the sum of the charges is at most ϕw_f. It turns out that in some cases this is not true, and, if so, we then show that for the job g scheduled by E in the next step, the total of all charges to f and g is at most $\phi(w_f + w_g)$. Summing over all such groups of one or two jobs, the ϕ-competitiveness of $\text{EDF}_{\phi-1}$ follows.

If f receives only one charge, it is at most ϕw_f: If this charge is from f, it is trivially at most w_f. If the charge is from j (not scheduled before t in E), then j is pending at t in E and thus $w_j \leq w_h \leq \phi w_f$, by the definition of $\text{EDF}_{\phi-1}$. In this case the group consist of a single job and we are done.

It remains to handle the case when f receives both charges. In this case, obviously, $j \neq f$ and j is pending in E at time t. Since in the canonical earliest-deadline schedule A job j is strictly before f, yet f is chosen by $\text{EDF}_{\phi-1}$, it follows that $w_j < (\phi - 1)w_h$.

If $w_f = w_h$, then f is charged at most $w_f + w_j \leq (1 + \phi - 1)w_h = \phi w_f$, and we have a group with a single job again.

Otherwise, $w_f < w_h$ and by the rule of $\text{EDF}_{\phi-1}$, it follows that $d_h > d_f$. Furthermore, since the adversary does not schedule f at time t, we have $d_f \geq t+2$. The span is bounded by 3, and thus the only possible case is that $d_h = t+3$ and $d_f = t+2$. Thus the adversary schedules f at time $t+1$. The weight of the job g scheduled at time $t+1$ in E is $w_g \geq (\phi - 1)w_h$, as $h \neq f$ is still pending in E. Furthermore, g gets only the charge from itself, as the adversary at time $t+1$ schedules f which is charged to itself. The total weight of the jobs charged to f and g is at most $w_j + w_f + w_g \leq (\phi - 1)w_h + w_f + w_g \leq \frac{3}{2}(w_f + w_g)$, since both w_f and w_g are at least $(\phi - 1)w_h$. In this last case we have a group of two jobs. $\square$

A more careful analysis yields an upper bound of $2-\Theta(1/s)$ on the competitive ratio of EDF_α on s-bounded instances, for an appropriately chosen α. More precisely, for each $s \geq 4$, let λ_s be the unique non-negative solution of the equation

$$(2 - \lambda_s)(\lambda_s^2 + \left\lfloor \frac{s}{3} \right\rfloor \lambda_s + s - 2 - 2\left\lfloor \frac{s}{3} \right\rfloor) \;=\; \lambda_s^2 - \lambda_s.$$

Theorem 5.2 *For each $s \geq 4$, the competitive ratio of* EDF_{1/λ_s} *for s-bounded instances is equal to λ_s defined above.*

PROOF. Throughout the proof, we write λ instead of λ_s. For any time t, let M_t be the maximal weight of a job available to $\text{EDF}_{1/\lambda}$ at time t; define $M_t = 0$ if no job is available.

First we show that we can restrict ourselves to instances where all the jobs have weights λ^i for some integer i. For these instances, however, we assume that the algorithm has no control over how the ties are resolved, and given two jobs of equal weight, the adversary can dictate which one should be considered heavier. Still, the ties are resolved in a consistent manner for both algorithms. More precisely, a *valid run* for such instances is defined so that at each time step, we schedule the earliest deadline job (applying our usual tie-breaking convevntion) from the set of pending jobs that contains all jobs with weight strictly bigger than M_t/λ and an arbitrary subset of jobs (chosen by the adversary) with weight equal to M_t/λ.

Claim A: Without loss of generality, it is sufficient to prove the theorem for valid runs on instances where all the jobs have weights of the form λ^i, for some integer i.

Call a job *bad* if its weight is not equal to λ^i for an integral i. Now we show that any instance with some bad jobs can be converted to an instance with a smaller number of bad jobs and with the same or larger competitive ratio on some valid run. Express each weight of a bad job j as $w_j = a_j \lambda^{e_j}$ for integral e_j and $1 < a_j < \lambda$. Let $b_{\min} = \min_j(a_j - 1)$ and $b_{\max} = \min_j(\lambda - a_j)$ (the minima are taken over all bad jobs j). Now replace the weight of each bad job j by $(a_j + b)\lambda^{e_j}$, for some $b \in [-b_{\min}, b_{\max}]$. From the definition of $b_{\min}$ and $b_{\max}$, it follows that the order of the weights of jobs does not change, as well as the result of comparisons of one weight to another weight divided by λ, except possibly for creating new ties. Consequently, any valid run on the original instance is also a valid run on the new instance, and the set of jobs scheduled in the original optimal solution gives also an optimal solution of the new instance. As the total weights of jobs scheduled both in the valid run and the optimal schedule are linear in b, their ratio is monotone in $[-b_{\min}, b_{\max}]$ and thus it is maximized either for $b = -b_{\min}$ or $b = b_{\max}$. Choose the appropriate b of these two possibilities and the corresponding modified instance. By the definition of $b_{\min}$ and $b_{\max}$, the number of bad jobs has decreased. After repeating this process a sufficient number of times we will convert the initial instance into one without bad jobs, and the ratio between the weight of the optimal schedule and the weight of $\text{EDF}_{1/\lambda}$'s schedule will not decrease. This completes the proof of Claim A.

Fix a valid run of $\text{EDF}_{1/\lambda}$ on an instance with no bad jobs and denote the resulting schedule by E. At any time t, either a job of weight M_t or M_t/λ is scheduled. If a job of weight M_t/λ is scheduled, the job with weight M_t remains pending at time $t+1$ (if its deadline were $t+1$, the valid run would schedule such a job at time t); thus in this case we have $M_{t+1} \geq M_t$.

Fix an earliest-deadline adversary schedule A. We define the charging scheme as follows. For any integer time t, let j be the job scheduled at t in A and f be the job scheduled in E. If j is completed in E before time t and $w_j \geq M_t$, charge j to j in E. Otherwise, charge j to f in E.

By the charging scheme, each job f in E receives at most two charges. Denoting by t the time when f is scheduled in E, f can receive a charge from the job scheduled in A at time t, and also from itself, if f is scheduled in A at or after time t.

It remains to prove that the charging scheme works correctly. The idea is similar to the proof of Theorem 5.1. We partition E into segments such that in each segment the total of all charges to the jobs in the segment is at most λ times their total weight. Summing over all such segments, this will imply λ-competitiveness of $\text{EDF}_{1/\lambda}$. Therefore, to complete the upper bound proof, it is now sufficient to prove the following claim.

Claim B: Schedule E can be partitioned into disjoint contiguous segments of jobs such that in each segment the total of all charges to the jobs in the segment is at most λ times their total weight.

We now prove Claim B. Let f be a job scheduled in E at time t. We start by some general observations.

(I) If f receives only one charge, then this charge is at most λ times its weight. If this charge is from f in A, it is trivially at most w_f. Otherwise, this charge is from a job j scheduled at time t in A. If j is scheduled before t in E, the charge is at most $M_t \leq \lambda w_f$ by the definition of the charging scheme. If j is not scheduled before t in E, then j is pending at t in E and thus $w_j \leq M_t \leq \lambda w_f$, by the definition of $\text{EDF}_{1/\lambda}$.

(II) If f receives both charges, the charge from the job j scheduled in A at time t is at most M_t/λ. It could be more only if j is not scheduled before t in E and $w_j > M_t/\lambda$. In that case, however, j is pending for $\text{EDF}_{1/\lambda}$ and has sufficiently large weight. In A, both j and f are pending at t and the adversary selects j. Since the ties are broken consistently, EDF must also prefer j and cannot schedule f.

We split E into segments starting from the beginning. Suppose that the currently processed time is t and E schedules a job f at time t. If f receives a single charge or $w_f = M_t$, we create a segment with a single job f. By the observations above, this segment is charged at most λ times its weight: if $w_f = M_t$, then f is charged at most $(1 + 1/\lambda)w_f \leq \lambda w_f$, by (II) and the inequality $\lambda \geq \phi$.

It remains to handle the case when f receives two charges and $w_f = M_t/\lambda$. For $i \geq t$, let f_i be the job scheduled in E at time i, and let $m \geq t$ be the smallest index such that $w_{f_m} = M_m$. (Such m exists, as eventually a maximal job is scheduled.) Thus $w_{f_i} = M_i/\lambda$ for $t \leq i < m$ and $M_t \leq M_{t+1} \leq \cdots \leq M_m$. Let Z be the set of jobs $f_t, \ldots, f_{m-1}$. We create a segment of jobs $f_t, \ldots, f_m$ and prove that its charging ratio is at most λ.

Let $k \geq 0$ be such that $M_m = \lambda^{k+1}$. For $i = 1, \ldots, k$, let X_i be the set of all jobs in Z with weight M_m/λ^i that receive two charges and let $x_i = |X_i|$. Also, let $X = \bigcup X_i$ and $x = |X| = \sum_{i=1}^{k} x_i$.

By the definition of Z, $\text{EDF}_{1/\lambda}$ schedules first all jobs in X_k, then X_{k-1}, and so on, up to X_1 (with possibly some jobs in $Z - X$ scheduled in-between the jobs from X.) Since every f_r in X receives also its own charge, it is scheduled in A after time r. Furthermore, it cannot be scheduled at time r' such that $r < r' \leq m$, for otherwise we would have $w_{f_r} < M_r \leq M_{r'}$, and by the definition of the charging scheme f_r is not charged to itself in such a case. Thus all jobs in X are scheduled at time $m + 1$ or later in A.

We claim that for each i,

$$x_i + x + 2 \leq s. \tag{8}$$

For a fixed i, let $i' \geq i$ be such that the last job finished by A from $X_i \cup \cdots \cup X_k$ belongs to $X_{i'}$. Let j be a job of maximal weight available when $\textsc{Edf}_{1/\lambda}$ schedules the first job of $X_{i'}$. Since j is scheduled in E only after all jobs in $X_{i'}$ despite the fact that its weight is larger, it must have strictly larger deadline than all the jobs in $X_{i'}$. Between the start of the first job in $X_{i'}$ in E and time $m+1$, $\textsc{Edf}_{1/\lambda}$ schedules all jobs in $X_{i'} \cup X_{i'-1} \cup \cdots \cup X_1$ and f_m. Between time $m+1$ and the time A finishes the last job of $X_{i'}$, A schedules all jobs in $X_i \cup \cdots \cup X_k$. Job j is available at all times from the start of the first job in $X_{i'}$ in E, until at least one time step after A finishes the last job of $X_{i'}$. Therefore we have

$$s \geq x_{i'} + x_{i'-1} + \cdots + x_1 + 1 + x_i + \cdots + x_k \geq x_i + x + 2.$$

Using (I), each job in $Z - X$ is charged at most λ times its weight. Let

$$W = \sum_{i=1}^{k} \frac{x_i}{\lambda^i}, \tag{9}$$

i.e., $WM_m = \lambda^{k+1} W$ is the total weight of jobs in X. Using (II), the jobs in X and f_m are charged a total of at most $2WM_m + (1 + 1/\lambda)M_m$ and their weight is $WM_m + M_m$. To finish the proof of λ-competitiveness, it is sufficient to show that

$$\lambda \geq \frac{2W + 1 + \frac{1}{\lambda}}{W + 1} = 2 - \frac{1 - \frac{1}{\lambda}}{W + 1}. \tag{10}$$

The right-hand side increases with W. Thus, we need to determine the largest possible value of W.

Suppose that integers x_i satisfy (8) and maximize W. Then we claim that this optimal solution satisfies the following conditions:

- **(a)** $x_i \geq x_{i+1}$ for any $i \geq 1$. Otherwise, we could switch the values of x_i and x_{i+1}, preserving inequality (8) and increasing W. Furthermore, $x_1 > 0$, as otherwise $W = 0$ but $x_1 = 1$, $x_2 = x_3 = \ldots = 0$ is a feasible solution with $W > 0$.
- **(b)** $x_i = 0$ for any $i \geq 3$. Otherwise, we could decrease both x_{i-1} and x_i by 1 and increase x_1 by 1. Since $\lambda \geq \phi$, this increases W by at least $1/\lambda - 1/\lambda^2 - 1/\lambda^3 \geq 0$, and it preserves (8) and (a).

(c) $2x_1 + x_2 + 2 = s$. Otherwise (8) implies a strict inequality and we could modify x_1, x_2 as follows: If $x_2 = 0$, increase x_2 to 1. If $x_2 > 0$, increase x_1 by 1 and decrease x_2 by 1. This increases W, and it preserves (8), (a), and (b).

(d) $x_1 \le x_2 + 2$. Otherwise, we could increase x_2 by 2 and decrease x_1 by 1. This increases W, and it preserves (8), (a), (b) and (c).

By (a), (c) and (d), we get $(s-2)/3 \le x_1 \le s/3$. For any $s \ge 4$, the only integer in this range is $x_1 = \lfloor s/3 \rfloor$. Thus $x_2 = s - 2 - 2\lfloor s/3 \rfloor$, and we have

$$
\begin{aligned}
2 - \frac{1 - \frac{1}{\lambda}}{W+1} &\le 2 - \frac{1 - \frac{1}{\lambda}}{1 + \frac{\lfloor \frac{s}{3} \rfloor}{\lambda} + \frac{s - 2 - 2\lfloor \frac{s}{3} \rfloor}{\lambda^2}} \\
&= 2 - \frac{\lambda^2 - \lambda}{\lambda^2 + \lfloor \frac{s}{3} \rfloor \lambda + s - 2 - 2\lfloor \frac{s}{3} \rfloor} \;=\; \lambda,
\end{aligned}
$$

by the definition of λ. This completes the proof of Claim B and the the upper bound.

Claim C: The competitive ratio of $\text{EDF}_{1/\lambda}$ is no better than λ.

To prove Claim C we present instances on which the competitive ratio of $\text{EDF}_{1/\lambda}$ approaches λ. Intuitively, the bad instance consists of exactly one segment corresponding to the worst case from the proof of Claim B above. Let x_1 and x_2 be the optimal values as defined in that proof. Let $\epsilon > 0$ be arbitrarily small. The instance contains the following jobs, written as (r_j, d_j, w_j): x_2 jobs $(0, x_2, 1-\epsilon)$, $x_1 + 1$ jobs $(x_2, x_1 + x_2 + 1, \lambda - \epsilon)$, x_2 jobs $(0, s, 1)$, 1 job $(0, s, \lambda)$, $x_1 - 1$ jobs $(x_2, x_2 + s, \lambda)$, and 1 job $(x_2, x_2 + s, \lambda^2)$. It is easy to check that the adversary schedules all the jobs in the given order, while $\text{EDF}_{1/\lambda}$ schedules only the jobs with weights 1, λ, and λ^2. The total weight obtained by $\text{EDF}_{1/\lambda}$ is $\lfloor s/3 \rfloor \lambda + (s - 2 - 2\lfloor s/3 \rfloor) + \lambda^2$ and the total weight obtained by the adversary approaches $(2\lfloor s/3 \rfloor + 1)\lambda + 2(s - 2 - \lfloor 2s/3 \rfloor) + \lambda^2$ for $\epsilon \to 0$. Hence the competitive ratio approaches λ. □

For $s = 4$, we get $\lambda_4 = \sqrt{3} \approx 1.732$. For larger s, the equation is cubic. It can be verified that $2 - 2/s \le \lambda_s \le 2 - 1/s$, and in the limit for $s \to \infty$, $\lambda_s = 2 - 2/s + o(1/s)$.

Recall that, by Theorem 2.2, results for discrete metered tasks can be applied to unit-job scheduling. Here we describe two such results. We say a pending job i *dominates* another pending job j if $d_i < d_j$ and i is heavier than j. A pending job is *dominant* if no other pending job dominates it. In [8], the authors considered the case of the metered-task model when there are at most

s dominant jobs at each time, and proposed an online algorithm GAP for this case. In s-bounded instances there can be at most s pending dominant jobs at any time, since there can be at most one dominant job for each deadline. Thus, the results from [8] imply that GAP is r_s-competitive for s-bounded instances, where r_s is the unique positive real root of the equation $r_s = 1 + r_s^{-1/(s-1)}$. It can be shown that $r_s = 2 - \Theta(\frac{1}{s})$. Table 2 gives a comparison of $\text{EDF}_{1/\lambda}$ (with $\lambda = \lambda_s$) and GAP. $\text{EDF}_{1/\lambda}$ has a smaller competitive ratio for s-bounded instances. On the other hand GAP can be applied to the more general set of instances that have at most s dominant jobs at any time. GAP can also be slightly modified to give the same performance without knowing the value of s in advance.

Table 2
Comparison of the upper bounds for $\text{EDF}_{1/\lambda}$ and GAP.

s	2	3	4	5	6	7	10	20	∞
$\text{EDF}_{1/\lambda}$	1.618	1.618	1.732	1.769	1.791	1.813	1.856	1.917	2
GAP	1.618	1.755	1.819	1.857	1.881	1.899	1.930	1.965	2

In [7], an algorithm FIT was presented for the discrete metered-task model. Its competitive ratio is better than 2 when the *importance ratio*, that is the ratio of maximum to minimum job weights, is at most ξ (without any restriction on s). By Theorem 2.2, we obtain that FIT is $(2 - 1/(\lceil \log_2 \xi \rceil + 2))$-competitive for unit-job scheduling.

6 s-Uniform Instances

In this section we consider s-uniform instances, where $d_j = r_j + s$ for each job j. We first prove a lower bound on the competitive ratio of randomized algorithms which increases with s and tends to 1.25 for large s. This improves the (deterministic) lower bound of $4 - 2\sqrt{2} \approx 1.172$ from [14] for $s \to \infty$.

Theorem 6.1 *Let*

$$R_s \;=\; 1 + \frac{s-1}{2s - 1 + 2\sqrt{s^2 - s}} \,.$$

No randomized algorithm can be better than R_s-competitive for s-uniform instances.

PROOF. We use Yao's minimax principle, in the form applicable to the lower bounds on the competitive ratios [5]. Following this principle, it is sufficient to give a distribution on instances for which the ratio between the expected gain of any deterministic online algorithm and the expected optimal gain is no better than R_s.

We generate an instance randomly as follows. Fix a large integer n and let $a = 1 + \sqrt{s/(s-1)}$ and $p = 1/a$. (Note that $a = 1 + \sqrt{2}$ for $s = 2$ and $a \to 2$ for $s \to \infty$.) Each instance consists of stages $0, 1, \ldots$, where in stage i we have s jobs of weight a^i released at time si and $s - 1$ jobs of weight a^{i+1} released one by one at times $si + 1$, $si + 2$, $\ldots$, $si + s - 1$. After each stage $i \le n$, we continue with probability p or stop with probability $1 - p$. After stage n, if the process has not yet terminated, then at time $(s + 1)n$, we release s jobs of weight a^{n+1} and stop.

Fix a deterministic online algorithm $\mathcal{A}$. We compute the expected gain of $\mathcal{A}$ and the adversary in stage $i \le n$, conditioned on stage i being reached. More precisely, in stage i we include the contributions of jobs scheduled at times si, $si + 1$, $\ldots$, $si + s - 1$ plus the expected gain of the jobs that remain pending at time $s(i + 1)$ in case this is the last stage.

Suppose that $\mathcal{A}$ reaches stage i. Let x be the number of jobs with weight a^i executed by $\mathcal{A}$. Then the gain of $\mathcal{A}$ is $xa^i + (s - x)a^{i+1}$. In addition, there are $x-1$ pending jobs of weight a^{i+1} at the end of the stage, which contribute if the instance ends by this stage, i.e., with probability $1 - p$. Since the probability of reaching stage i is p^i, the expected gain for stage i is

$$
\begin{aligned}
&p^i(xa^i + (s - x)a^{i+1} + (1 - p)(x - 1)a^{i+1}) \\
&= x + sa - xa + (a - 1)(x - 1) \;=\; 1 + (s - 1)a.
\end{aligned}
$$

Note that this is independent of x and i.

We now calculate the expected adversary gain in stage i. If we stop after this stage, the contribution of stage i towards adversary's gain is $sa^i + (s - 1)a^{i+1}$, otherwise it is $a^i + (s - 1)a^{i+1}$, so the expected contribution of stage i is

$$p^i(a^i + (s - 1)a^{i+1} + (1 - p)(s - 1)a^i) \;=\; 1 + (s - 1)(a + 1 - p).$$

Summarizing, for each stage, except the last one, the contributions towards the expected value are constant. The contributions of stage $n+1$ are different, but they are also constant (independent of n). So the overall ratio will be, in the limit for $n \to \infty$, the same as the ratio of the contributions of stages $0, \ldots, n$, which is (after some calculation)

$$\frac{1 + (s - 1)(a + 1 - p)}{1 + (s - 1)a} \;=\; 1 + \frac{(s - 1)(1 - p)}{1 + (s - 1)a} \;=\; R_s. \qquad \square$$

Deterministic algorithms for 2-uniform instances were studied in [2], where an upper bound of $\sqrt{2}$ was given. As we show below, it is not possible to beat ratio $\sqrt{2}$ with any deterministic memoryless algorithm. Recall that we define an online algorithm $\mathcal{A}$ to be *memoryless* if its decision at each step depends only on the weights of the pending jobs (including those released at the current step) and its decisions are scale-invariant.

Theorem 6.2 *No deterministic memoryless algorithm can be better than $\sqrt{2}$-competitive for 2-uniform instances.*

PROOF. Let $\mathcal{A}$ be a deterministic memoryless algorithm. We show an adversary strategy that forces $\mathcal{A}$'s competitive ratio to be at least $\sqrt{2}$. In this strategy, at time 0 two jobs of weight 1 are released. Afterwards, $\mathcal{A}$ will have exactly one pending job from the previous step, except for the end when no new jobs are released. Suppose that $y > 0$ is a pending job (or, more precisely, the weight of the pending job). The adversary will choose one of the two moves:

(a) Release two jobs with weight y.
(b) Release one job with weight $b > y$.

Note that in step (a), $\mathcal{A}$ has no choice but to execute a job of weight y. Further, it does not matter which job it chooses for execution.

Consider steps of type (b). $\mathcal{A}$'s decision is only a function of b/y. For any $\epsilon > 0$ there are two constants $a_1, a_2 > 1$ with $|a_1 - a_2| \leq \epsilon$, such that $\mathcal{A}$ executes y when $b/y = a_1$ and b when $b/y = a_2$. (If one of a_1, a_2 does not exist, then it's easy to see that $\mathcal{A}$ is not competitive at all.) To simplify the proof, we will further assume that $a_1 = a_2 = a$, and allow the adversary to break the tie and determine which job will be executed by $\mathcal{A}$ when $b/y = a$. To fully formalize this argument, all we need to do is to replace a by a_1 or a_2 to force $\mathcal{A}$ to make the desired decision, and then take the limit with $\epsilon \to 0$.

Choose some large integer n. The adversary uses three instances. In instance I_1 we release two jobs of weight 1 at step 0 and one job of weight a at step 1. The adversary forces $\mathcal{A}$ to execute job a at step 1. The gain of $\mathcal{A}$ is $1 + a$, the adversary gains $2 + a$, so the ratio is $R_1(a) = (a + 2)/(a + 1)$.

In instance I_2, after the jobs of weight 1 at step 0, at each time $j = 1, \ldots, n$ we release a job j of weight a^j, and at time $n + 1$ we release two jobs n', n'' of weight a^n. The adversary breaks the tie so that $\mathcal{A}$ executes job j at time $j + 1$, for $j = 1, \ldots, n$. So the gain of $\mathcal{A}$ is $1 + \sum_{j=0}^{n} a^j + a^n$, and the gain of the adversary is $\sum_{j=0}^{n} a^j + 2a^n$. Taking the limit with $n \to \infty$, so that we can ignore low-order terms, the ratio is $R_2(a) = (3a - 2)/(2a - 1)$.

The last instance I_3 has the same jobs as I_2, except that at time $n+2$ we also release job n^* of weight a^{n+1}. The adversary forces $\mathcal{A}$ to execute this job at time $n+2$. The gain of $\mathcal{A}$ is $1+\sum_{j=0}^{n} a^j + a^{n+1}$, and the gain of the adversary is $\sum_{j=0}^{n} a^j + 2a^n + a^{n+1}$. In the limit, the ratio is $R_3(\alpha) = 1 + 2(a-1)/a^2$.

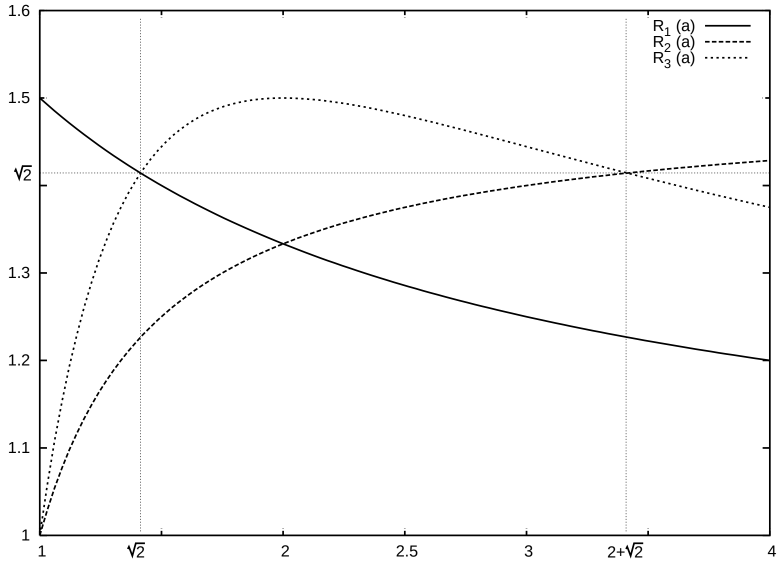

Fig. 1. Lower bounds for memoryless algorithm

The overall ratio is $R(a) = \max\{R_1(a), R_2(a), R_3(a)\}$. The graph in Figure 1 shows the three functions $R_1(a)$, $R_2(a)$, and $R_3(a)$. By routine calculations, this ratio is minimized for $a \in \left\{\sqrt{2}, 2+\sqrt{2}\right\}$, and for those values $R(a) = \sqrt{2}$. $\square$

7 The Multiprocessor Case

In this section we consider the multiprocessor case. The greedy 2-competitive algorithm [14,13] applies to both uniprocessor and multiprocessor cases. We show an algorithm that for m processors achieves competitive ratio $R = (1 - (\frac{m}{m+1})^m)^{-1}$. When $m \to \infty$, this ratio tends to $e/(e-1) \approx 1.582$, beating even the $\phi \approx 1.618$ lower bound for $m = 1$ [2,9,13].

The basic idea of our algorithm is similar to algorithm Mixed for metered tasks in [9], and it also bears some similarity to our randomized algorithm RMix. We want to divide the processing effort between the m processors, such that the first one executes the heaviest job, and each of the successive processors executes the earliest-deadline job with weight above a certain threshold, with the threshold decreasing geometrically for each processor. The difficulty that arises in multiprocessor scheduling is that after choosing several jobs there may not be any jobs with weight above the current threshold. In

MIXED or RMIX, this would result in assigning more resources (processing speed or higher probability, respectively) to the previously selected jobs. This approach cannot be emulated in a multiprocessor setting. Instead, we simply select the heaviest remaining job, and reset the threshold to this job's weight. The detailed algorithm is given below.

Throughout this section, $\beta = m/(m+1)$ and $R = (1-\beta^m)^{-1}$. To simplify presentation, we assume that at each step there are at least m pending jobs. (Otherwise, the algorithm may create some dummy jobs with weight 0.)

Algorithm DMIX-m**.** Let P be the set of pending jobs at a given time t. Schedule jobs $h_1, h_2, \ldots, h_m$ chosen according to the following procedure:

(1) $q \leftarrow 1$
(2) $h_q \leftarrow$ the heaviest job in $P - \{h_1, \ldots, h_{q-1}\}$
(3) $k \leftarrow q$
(4) `while` $q < m$ `do`
(5) $\quad q \leftarrow q + 1$
(6) $\quad h_q \leftarrow$ the earliest-deadline job f in $P - \{h_1, \ldots, h_{q-1}\}$ with $w_f \geq \beta^{q-k} w_{h_k}$
(7) $\quad$ `if` no such job exists `then` goto (2)

Theorem 7.1 DMIX-m *is R-competitive for m processors, where*

$$R = \left(1 - \left(\frac{m}{m+1}\right)^m\right)^{-1}.$$

PROOF. We use a charging scheme. For a given input instance, let A be a canonical earliest-deadline adversary schedule. A job j scheduled in A at time t is charged (i) to time t' when j is scheduled in DMIX-m, if such t' exists and $t' \leq t$, or (ii) to time t otherwise.

For a given time t, let $h_1, \ldots, h_m$ be the jobs executed by DMIX-m at t, listed in the order chosen by the algorithm. Let l be the largest index such that h_l is chosen in step (2) of the algorithm and let us denote $v = w_{h_l}$. From the description of DMIX-m it follows that

$$w_{h_q} \geq \beta^{q-l} v \tag{11}$$

for all $q = 1, \ldots, m$. This can be verified first by a backward induction for the subset of the jobs $h_l, \ldots, h_1$ which are selected in step (2) of DMIX-m, and then it follows immediately for all the remaining jobs.

Let J be the set of jobs charged to t. Observe that by the definition of the charging scheme, all jobs in J are pending in DMix-m at time t. Let p be the largest index such that $p \geq l$ and h_p is executed in A after time t; if no such p exists, set $p = l$. Denote $L = J - \{h_1, \ldots, h_p\}$.

If $h' \in L$ then, by the choice of p, h' is scheduled in A at time t. In particular, this implies that $|L| \leq m$. Furthermore, we claim that

$$w_{h'} \leq \beta^{p-l} v \tag{12}$$

for all $h' \in L$. If $p = l$ then (12) is trivial, as $h_p = h_l$ is chosen as the heaviest job of the remaining ones, including h'. If $p > l$ then h' is scheduled in A at time t while h_p is scheduled later, by the choice of p. This implies that h' is before h_p in the earliest-deadline ordering; so if $w_{h'} > \beta^{p-l} v$, then h' would be chosen in place of h_p. (We use here the fact that $p \geq l$ and l is the last job chosen in step (2), so it determines the threshold for h_p in the algorithm.)

To show the correctness of the charging scheme, we need to show that the total weight charged to time t is at most R times the gain of DMix-m at t, that is $\sum_{j \in J} w_j \leq R \cdot \sum_{q=1}^{m} w_{h_q}$. Using inequalities (11), (12), and $|L| \leq m$, we have

$$\begin{aligned}
& R \cdot \sum_{q=1}^{m} w_{h_q} - \sum_{j \in J} w_j \\
& \geq R \cdot \sum_{q=1}^{m} w_{h_q} - \sum_{q=1}^{p} w_{h_q} - \sum_{h' \in L} w_{h'} \\
& \geq R \cdot \sum_{q=p+1}^{m} \beta^{q-l} v + (R-1) \cdot \sum_{q=1}^{p} \beta^{q-l} v - m\beta^{p-l} v \\
& = \left(\frac{1}{1-\beta^m} \cdot \frac{\beta^{p+1} - \beta^{m+1}}{1-\beta} + \frac{\beta^m}{1-\beta^m} \cdot \frac{\beta - \beta^{p+1}}{1-\beta} - m\beta^p \right) \beta^{-l} v \\
& = \left(\frac{\beta^{p+1}}{1-\beta} - m\beta^p \right) \beta^{-l} v \\
& = 0.
\end{aligned}$$

The first equality follows by the definition of R and summing the geometric sequences, the second equality results from routine algebraic cancellation, and the last one follows from the definition of β, as $\beta/(1-\beta) = m$. □

The analysis in Theorem 7.1 is tight. The bound is attained by the following instance. Consider $2m$ jobs all released at time 0: m jobs of deadline 2, with weights $1, \beta, \beta^2, \ldots, \beta^{m-1}$, and m jobs of deadline 1 and weight $\beta^{m-1} - \epsilon$ for small $\epsilon > 0$. The adversary can schedule all jobs while DMIX-m only schedules the m jobs with deadline 2.

The lower bound proofs for randomized algorithms in [9] and Theorem 6.1 can easily be generalized to the multiprocessor case by including m copies of each job used in the lower bound instance. This improves the bounds in [14], which are $4 - 2\sqrt{2}$ for the general case, and $10/9$ for the 2-uniform case.

Theorem 7.2 (a) *No deterministic or randomized algorithm can be better than* 1.25*-competitive, for any number of processors* m*.*

(b) *For* s*-uniform instances, no deterministic or randomized algorithm can be better than* R_s*-competitive, for any number of processors* m*, where*

$$R_s = 1 + \frac{s-1}{2s-1+2\sqrt{s^2-s}}.$$

8 Final Remarks

The main question regarding unit-job scheduling, of whether there exists a ϕ-competitive deterministic algorithm, remains open. Very recently, a deterministic 1.94-competitive algorithm was obtained in [11], but closing the gap appears to be a challenging problem. Similarly, in the randomized case, there is still a wide gap between the lower bound of 1.25 [9] and our upper bound of 1.582.

By presenting the randomized 1.25-competitive algorithm, we have completely settled the 2-bounded case in this paper. It is quite surprising, in our view, that the seemingly simpler 2-uniform case is so much harder to solve. Recently, in [11], the 2-uniform deterministic case was completely resolved by showing tight bounds approximately equal to 1.377. This is less than our lower bound of $\sqrt{2}$ for memoryless algorithms, showing that the general algorithms are provably more powerful in this case. Needless to say, the randomized case is even harder, and our intuitions are not strong enough to formulate any conjectures.

References

[1] W. A. Aiello, Y. Mansour, S. Rajagopolan, and A. Rosen. Competitive queue policies for differentiated services. In *Proc. of the IEEE INFOCOM*, pages 431–440, 2000.

[2] N. Andelman, Y. Mansour, and A. Zhu. Competitive queueing policies in QoS switches. In *Proc. 14th Symp. on Discrete Algorithms (SODA)*, pages 761–770. ACM/SIAM, 2003.

[3] N. Bansal, L. Fleischer, T. Kimbrel, M. Mahdian, B. Schieber, and M. Sviridenko. Further improvements in competitive guarantees for QoS buffering. In *Proc. 31st International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 3142 of *Lecture Notes in Comput. Sci.*, pages 196–207. Springer, 2004.

[4] Y. Bartal, F. Y. L. Chin, M. Chrobak, S. P. Y. Fung, W. Jawor, R. Lavi, J. Sgall, and T. Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. In *Proc. 21st Symp. on Theoretical Aspects of Computer Science (STACS)*, volume 2996 of *Lecture Notes in Comput. Sci.*, pages 187–198. Springer, 2004.

[5] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[6] E.-C. Chang and C. Yap. Competitive online scheduling with level of service. In *Proc. 7th Annual International Computing and Combinatorics Conference (COCOON)*, volume 2108 of *Lecture Notes in Comput. Sci.*, pages 453–462. Springer, 2001.

[7] F. Y. L. Chin and S. P. Y. Fung. Online scheduling with partial job values and bounded importance ratio. In *Proc. of International Computer Symposium*, pages 787–794, 2002.

[8] F. Y. L. Chin and S. P. Y. Fung. Improved competitivness algorithms for online scheduling with partial job values. In *9th International Computing and Combinatorics Conference*, pages 425–434, 2003.

[9] F. Y. L. Chin and S. P. Y. Fung. Online scheduling for partial job values: Does timesharing or randomization help? *Algorithmica*, 37:149–164, 2003.

[10] M. Chrobak, L. Epstein, J. Noga, J. Sgall, R. van Stee, T. Tichý, and N. Vakhania. Preemptive scheduling of tasks with hard deadlines. *J. Comput. Systems Sci.*, 67:183–197, 2003.

[11] M. Chrobak, W. Jawor, J. Sgall, and T. Tichý. Improved online algorithms for buffer management in QoS switches. In *Proc. 12th European Symp. on Algorithms (ESA)*, volume 3221 of *Lecture Notes in Comput. Sci.*, pages 204–215. Springer, 2004.

[12] L. Epstein and R. van Stee. Buffer management problems. *SIGACT News*, 35:58–66, 2004.

[13] B. Hajek. On the competitiveness of online scheduling of unit-length packets with hard deadlines in slotted time. In *Conference in Information Sciences and Systems*, pages 434–438, 2001.

[14] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer overflow management in QoS switches. In *Proc. 33rd Symp. Theory of Computing (STOC)*, pages 520–529. ACM, 2001.

[15] A. Kesselman, Y. Mansour, and R. van Stee. Improved competitive guarantees for QoS buffering. In *Proc. 11th European Symp. on Algorithms (ESA)*, volume 2832 of *Lecture Notes in Comput. Sci.*, pages 361–372. Springer, 2003.

[16] J. W. S. Liu, K.-J. Lin, W.-K. Shih, A. C. shi Yu, J.-Y. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. *IEEE Computer*, 24(5):58–68, 1991.

[17] Z. Lotker and B. Patt-Shamir. Nearly optimal FIFO buffer management for DiffServ. In *In Proc. of the 21st Symp. on Principles of Distributed Computing*, pages 134–143. ACM, 2002.