

Online Scheduling of Bounded Length Jobs to Maximize Throughput

Christoph Dürr¹, Łukasz Jeż², and Nguyen Kim Thang¹

¹ CNRS, LIX UMR 7161, Ecole Polytechnique, 91128 Palaiseau, France. *

² Institute of Computer Science, University of Wrocław, 50-383 Wrocław, Poland. **

Abstract. We consider an online scheduling problem, motivated by the following online auction setting. A non-storable perishable good is produced at a regular rate. It has to be sold immediately to customers whose offers arrive on-line. More formally, we consider the online scheduling problem with preemptions, where each job j is revealed at release time r_j , has processing time p_j , deadline d_j and weight w_j . The goal is to maximize the total weight of all jobs completed on time. Our main result is that if all jobs have processing time not more than k , then the deterministic competitive ratio of this problem is $\Theta(k/\log k)$.

1 Introduction

Consider a production site that produces at the regular rate of one item per time unit. The items have to be immediately delivered to some customer, think for example of electricity which can hardly be stored. In this online scenario, every customer i arrives at some release time $r_i \in \mathbb{N}$, and announces that he will pay $w_i \in \mathbb{R}$ euros if he gets $p_i \in \mathbb{N}^+$ items before the deadline $d_i \in \mathbb{N}$ (i is *satisfied*), otherwise he pays nothing. The objective of this optimization problem is to maximize the *social welfare*, which is the total value $\sum_i w_i$ over all satisfied customers i .

This is an online-scheduling problem on a single machine, where jobs arrive online at their release times, have some processing time, deadline and weight, and the objective is to maximize the total weight of jobs completed on time. Preemption of the jobs is allowed, meaning that a job i could be scheduled in different separated time intervals, as long as the total processing time is p_i . This problem could be denoted as 1|online- r_i ; pmtn| $\sum w_i(1 - U_i)$, according to the notation of [5].

This model is also called the *preemptive model with resume*, which contrasts with the *preemptive model with restarts* [6], where a job can be interrupted, but when it is scheduled again, it must be scheduled from the beginning.

It is known that the problem has an unbounded deterministic competitive ratio [3], so different directions of research were considered. One is to see if

* Supported by ANR Alpage.

** Supported by MNiSW grant number N N206 1723 33, 2007–2010, and COST 295 “DYNAMO”.

randomization helps, and indeed in [8] a constant competitive randomized algorithm was given, although with a big constant. Another direction of research is to consider resource augmentation, and in [9] a deterministic online algorithm was presented which has constant competitive ratio provided that the algorithm is allowed a constant speedup of its machine compared to the adversary.

In this paper we consider a different approach. We investigate deterministic algorithms only, and we do not allow resource augmentation. Instead we restrict ourselves to instances in which the processing times do not exceed some constant k . We analyze the competitive ratio as a function of k , and consider different cases.

Bounded processing time, unit weights (Case $\forall j : p_j \leq k, w_j = 1$) The offline problem can be solved in time $O(n^4)$ [1] already when the processing time is unbounded. We show that any deterministic online algorithm is $\Omega(\log k / \log \log k)$ -competitive. In [3] a similar bound has been shown for a generalization of this model, where processing times, release times and deadlines can be arbitrary fractions, whereas we restrict to integers. To force a ratio R , construct a sequence with largest to smallest processing time ratio $\frac{1}{2}(2R+1)^R$, and rational values of job parameters (release times, deadlines, processing times). In our construction all these parameters are integers and yet the largest processing time of job in our lower bound construction is only $R \cdot R!$. Thus using more restricted sequences we get a better constant in the $\Omega(\log k / \log \log k)$ lower bound. Additionally, we give a concise proof of the known fact that algorithm SHORTEST REMAINING PROCESSING TIME FIRST is $O(\log k)$ -competitive, which is claimed by [8]. The same paper provides a constant competitive randomized algorithm, however with a large constant.

Bounded processing time, arbitrary weights (Case $\forall j : p_j \leq k$) For fixed k the offline problem has not been studied to our knowledge, and when the processing times are unbounded, the offline problem is NP-hard by trivial reduction from the Knapsack Problem. It is known that any deterministic online algorithm for this case has competitive ratio at least $k/2 \ln k - 1$ [12] and we refine this lower bound, by showing that asymptotically the ratio is at least $k/\ln k$. This proof is moved to the appendix. In addition we show that the standard SMITH RATIO ALGORITHM has ratio $\Theta(k)$, and provide an optimal online algorithm that reaches the ratio $O(k/\log k)$. For the variant with only tight jobs, [4] provide an $O(\log k)$ -competitive randomized online algorithm and show a $\Omega(\sqrt{\log k / \log \log k})$ lower bound for any randomized competitive algorithm against an oblivious adversary.

Equal processing time, unit weights (Case $\forall j : p_j = k, w_j = 1$) The offline problem can be solved in time $O(n \log n)$ [10], and it is well known that the same algorithm can be turned into a 1-competitive online algorithm, see for example [13].

Equal processing time, arbitrary weights (Case $\forall j : p_j = k$) The offline problem can be solved in time $O(n^4)$ [2]. For $k = 1$ the problem is well studied, and the deterministic competitive ratio is between 1.618 and 1.83,

see for example [11, 7]. We show that for $k \geq 2$ the ratio is between $1.5\sqrt{3} \approx 2.598$ and 6.83.

2 Preliminaries

For a job i we denote its release time by r_i , its deadline by d_i , its processing time by p_i and its weight by w_i . All these quantities, except w_i , are integers. Let $q_i(t)$ be the remaining processing time of job i for the algorithm at time t . When there is no confusion, we simply write q_i . We say that job i is *pending* for the algorithm at time t if it has not been completed before and $r_i \leq t$ and $t + q_i(t) < d_i$. Let j be a job which is not completed by the algorithm. The *critical time* of j is the latest time when j was still pending for the algorithm. In other words, the critical time s of job j for the algorithm is such moment s that if the algorithm does not schedule j at time s , it cannot finish j anymore, i.e. $s = \max\{\tau : \tau + q_j(\tau) = d_j\}$. A direct observation from the definition is that (w.l.o.g.) at the critical time of a job j , which is not completed by the algorithm, the algorithm schedules some unit of another job.

Throughout the paper we will analyze our algorithms with similar charging schemes, that share the following outline: For every job j completed by the adversary we consider its p_j units. Each unit of job j will charge w_j/p_j to some job i_0 completed by the algorithm. The charging scheme will satisfy that every job i_0 completed by the algorithm receives no more than Rw_{i_0} in total, which implies competitive ratio R for the algorithm.

More precisely we distinguish individual units scheduled by both the algorithm and the adversary, where unit (i, a) stands for algorithm's execution of job i when its remaining processing time was a . In particular a job i completed by the algorithm consists of the units $(i, p_i), (i, p_i - 1), \dots, (i, 1)$. With every algorithm's unit (i, a) we associate a *capacity* $\pi(i, a)$ that depends on w_i and a , the exact value will be different from proof to proof. The algorithms, with their capacity, will be designed in such a way that they satisfy these following properties.

Monotonicity: If the algorithm schedules (i, a) at t and (i', a') at $t + 1$ with $a' > a$, then $\pi(i', a') \geq \pi(i, a)$,

Validity: Let (i, a) be the unit scheduled by the algorithm while j was pending. Then, $\pi(i, a) \geq w_j/p_j$.

In general there will be 3 types of charges in the charging scheme. Let (j, b) be a unit of job j scheduled by the adversary at time t .

Type 1: If the algorithm already completed j by time t , then charge w_j/p_j to j .

Type 2: Otherwise if the algorithm schedules a job unit (i, a) at time t that has capacity at least w_j/p_j then we charge w_j/p_j to i_0 , where i_0 is the next job completed by the algorithm from time t on (including t).

Type 3: In the remaining case, the job j is not pending anymore for the algorithm, as follows from algorithm's validity and monotonicity. Let s be the

critical time of j . We charge w_j/p_j to i_0 , where i_0 is the first job completed by the algorithm from time s on (including s).

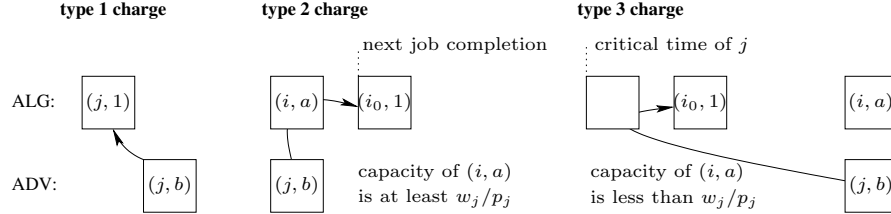


Fig. 1. The general charging scheme

Clearly every job i_0 completed by the algorithm can get at most w_{i_0} type 1 charges in total. We can bound the other types as well.

Lemma 1. *Let \mathcal{J} be the set of job units that are type 3 charged to a job i_0 completed by the algorithm, as described above. Then there are strictly less than p units $(j, b) \in \mathcal{J}$ with $p_j \leq p$. In particular $|\mathcal{J}| \leq k - 1$, if all jobs have processing time at most k . Moreover for each $(j, b) \in \mathcal{J}$ it holds that $w_j/p_j \leq \pi(i_0, 1)$.*

Proof. To be more precise we denote the elements of \mathcal{J} by triplets (s, t, j) such that a job unit (j, b) scheduled at time t by the adversary is type 3 charged to i_0 and its critical time is s . Let $t_0 \geq s$ be the completion time of i_0 by the algorithm. Between s and $t_0 - 1$ there is no idle time, nor a job completion, so by monotonicity and validity of the algorithm the capacities of all units in $[s, t_0]$ are at least w_j/p_j . However by definition of type 3 charges, the algorithm schedules at t some unit with capacity strictly smaller than w_j/p_j , so $t_0 < t$.

Since s is the critical time we have $s + q_j(s) = d_j$. However since the adversary schedules j at time t we have $t < d_j$. Thus $t - s < q_j(s) \leq p_j$. Note that all triplets $(s, t, j) \in \mathcal{J}$ have distinct times t . The first part of the lemma follows from the observation that there can be at most $c - 1$ pairs (s, t) with distinct t that satisfy $s \leq t_0 < t$ and $t - s < c$.

Since j was pending at time s , whatever the algorithm scheduled then had capacity at least w_j/p_j . By monotonicity of the algorithm this holds also at time t_0 , so $\pi(i_0, 1) \geq w_j/p_j$. \square

Actually all our algorithms, except the one from the next section, will be *priority based algorithms*, in the sense that the capacity π also plays the role of a priority, and every time the highest priority job is scheduled. More formally, the priority of j is $\pi(j, q_j)$. For such algorithms, monotonicity and validity are properties of the function π rather than the algorithm itself.

Lemma 2. *Suppose there is a constant $0 < \rho < 1$ such that $\rho\pi(i, a) \geq \pi(i, a+1)$. Then the total charge of type 2 a job i_0 receives is at most $\pi(i_0, 1)/(1 - \rho)$.*

Proof. The algorithm has the property that as long as it does not complete a job, it schedules units with capacities increasing at least by factor $1/\rho$ each. More formally, let t_0 be the completion time of i_0 , and let s the smallest time such that $[s, t_0)$ contains no idle time and no job completion. Then every unit scheduled at time $t_0 - i$ for $0 \leq i \leq t_0 - s$ has capacity at most $\pi(i_0, 1)\rho^i$. Thus the total type 2 charge is bounded by a geometric series,

$$\pi(i_0, 1)(1 + \rho + \rho^2 + \rho^3 \dots) = \pi(i_0, 1)/(1 - \rho).$$

□

We will adapt this general scheme to individual algorithms in the next sections.

3 Identical Processing Time Model

In this section we consider instances where each job has the same processing time $k \geq 2$ and arbitrary weight. Let $\beta > 1$ be a constant that we define later.

THE CONSERVATIVE ALGORITHM: If the previous time step was idle or a job completion, schedule the highest weight pending job. Otherwise if in the previous step some job i was scheduled that is still pending now, then continue executing i , until a new job j is released that is at least β times heavier than i , in which case j is executed (choosing the heaviest such job j if there is a choice).

Theorem 1. *The CONSERVATIVE ALGORITHM has ratio $2 + 2\beta + 1/(\beta - 1)$ which is at most 6.83 when $\beta = 1 + 1/\sqrt{2}$.*

Proof. We adapt the general charging scheme from the previous section. A job unit (i, a) scheduled by the algorithm has capacity $\beta w_i/k$. Note that the algorithm is monotone, since any job may only be interrupted by a heavier job. Now we turn to the algorithm's validity. Suppose the algorithm schedules (i, a) in some step t . Then any job j pending for the algorithm in that step has weight $w_j < \beta w_i$ by the algorithm's definition. As $\pi(i) = \beta w_i/k$, the algorithm is valid. Now we will bound the total charge received by a job i_0 completed by the algorithm, say at time t_0 . Total type 1 charge is at most w_{i_0} . For the type 2 charge, let s be the smallest time such that there is no idle time and no completion time in $[s, t_0)$. Let the jobs scheduled by the algorithm in this interval be i_b, i_{b-1}, \dots, i_0 , respectively. Then for every $c: b > c \geq 0$ the job i_{c+1} was interrupted by i_c , hence $\beta w_{i_{c+1}} \leq w_{i_c}$. So the total type 2 charge to i_0 is at most

$$\beta w_{i_0} \sum_{j=0}^b \frac{1}{\beta^j} \leq \beta w_{i_0} / (1 - 1/\beta) = w_{i_0} \beta^2 / (\beta - 1) = w_{i_0} \left(\beta + 1 + \frac{1}{\beta - 1} \right).$$

For the type 3 charge, we simply apply Lemma 1, and conclude that every job unit j type 3 charged to i_0 is bounded by $w_j/k \leq \beta w_{i_0}/k$, and since there are at most $k - 1$ type 3 charges, that makes at most $(k - 1)\beta w_{i_0}/k < \beta w_{i_0}$ in total. Adding the charges of all 3 types gives the required bound on the ratio. □

4 Bounded Processing Time, Unit Weight Model

In this section we consider instances over jobs j such that $p_j \leq k$ for some k and $w_j = 1$.

THE SHORTEST REMAINING PROCESSING TIME FIRST Algorithm is a greedy online algorithm that schedules at every step the pending job with the smallest remaining processing time. In other words, it is a priority-based algorithm with the priority function $\pi(i, a) = w_i/a = 1/a$. It was analyzed in [8], but we provide a concise proof for completeness.

Proposition 1 ([8]). SHORTEST REMAINING PROCESSING TIME FIRST is $2H_k$ -competitive, where H_k denotes the k -th harmonic number, $1+1/2+1/3+\dots+1/k$.

Proof. We use our general charging scheme. The algorithm with the definition of capacity satisfies properties of monotonicity and validity. A simple observation of the algorithm is that whenever the algorithm schedules some job i at time t , then some job will complete in $[t, t+k)$, either i itself or some job with smaller processing time. In particular if t_0 is the completion time of some job i_0 by the algorithm, and s is the smallest time such that $[s, t_0)$ contains no idle time nor completion, then $t_0 - s < k$ and the unit scheduled at time $t_0 - i$ for $0 \leq i \leq t_0 - s$ has capacity at most $1/(i+1)$. As a result the total type 2 charge to i_0 is at most H_k .

Lemma 1 states that there are at most $p-1$ type 3 charges to i_0 from jobs units j with $p_j \leq p$. The worst case is when there is exactly one job unit j with $p_j = p$ charging $1/p$ to i_0 for every $p = 2, 3, \dots, k$. Therefore the total type 3 charge to i_0 is at most $H_k - 1$.

Total type 1 charge is at most $w_{i_0} = 1$, so this concludes the proof. \square

5 Bounded Processing Time, Arbitrary Weight Model

This time we consider instances with arbitrary weights, starting with a straightforward generalization of SHORTEST REMAINING PROCESSING TIME FIRST. It is very easy to analyze using our charging scheme, but the algorithm is suboptimal. Afterwards we present an optimal algorithm.

THE SMITH RATIO ALGORITHM schedules at every step the pending job j that maximizes the Smith ratio w_j/q_j . In other words, it is a priority-based algorithm with the following priority function: $\pi(j, q_j) = w_j/q_j$.

Theorem 2. The SMITH RATIO ALGORITHM is $2k$ -competitive.

Proof. We use the general charging scheme. Monotonicity and validity of the algorithm easily follow from its definition and the priority function it uses. A job i_0 completed by the algorithm receives at most w_{i_0} type 1 charge in total. Lemma 2 implies that each i_0 receives at most kw_{i_0} type 2 charges in total, as for $\pi(i, a) = w_i/a$ the value of ρ is $1-1/k$. By Lemma 1 i_0 receives at most $k-1$ type 3 charges, and each such charge is at most $\pi(i_0, 1) = w_{i_0}$. This concludes the proof. \square

It is fairly easy to come up with an instance on which SMITH RATIO ALGORITHM achieves ratio arbitrarily close to k , thus showing our analysis is tight up to a constant factor of 2. Now we present a deterministic online algorithm that is optimal up to a constant.

THE EXPONENTIAL PRIORITY ALGORITHM is a priority-based algorithm with priority function of the form $\pi(j, a) = w_j \cdot \alpha^{a-1}$ for some $\alpha < 1$ to be specified later.

In fact, the constant α depends on k , seemingly making EXPONENTIAL PRIORITY ALGORITHM a semi-online algorithm. However, the $\alpha(k)$ we use is an increasing function of k , so we can make the algorithm fully online by using the value $\alpha(k^*)$ in each step, where k^* is the maximum processing time among all jobs released up to that step. One can check that this fully online algorithm is still monotone — in the sense of section 2 — and one can derive bounds on competitive ratio of the algorithm by analyzing only the π function defined by $\alpha(k^*)$ for the final value of k^* .

Theorem 3. *The EXPONENTIAL PRIORITY ALGORITHM is $(3 + o(1))k/\ln k$ -competitive.*

Proof. As before, we use the general charging scheme. Let us define the proper value of $\alpha(k)$ now: $\alpha(k) = 1 - c^2 \cdot \ln k/k$, where $c = 1 - \epsilon$ for arbitrarily small $\epsilon > 0$. Since $\alpha < 1$, the algorithm is monotone.

To prove validity it is sufficient to prove that $p\alpha^{p-1} \geq 1$ for all $p \leq k$, as this implies $w_j/p_j \leq w_j\alpha^{p_j-1}$, and, by monotonicity and the choice of π , the following holds at any time step t and job j pending at t .

$$w_j\alpha^{p_j-1} \leq \pi(j, q_j(t)) \leq \pi(h, q_h(t)) \leq \pi(i_0, 1) = w_{i_0} , \quad (1)$$

where h is the job scheduled by the algorithm at t , and i_0 is the next job completed by it from time t on. Hence we introduce the function $f(x) = x\alpha^{x-1}$, and claim the following holds for any large enough k and any $x \in \{1, 2, \dots, k\}$.

$$f(x) \geq 1 \quad \text{for } 1 \leq x \leq \frac{k}{c^2 \ln k} , \quad (2)$$

$$f(x) \geq \ln k \quad \text{for } \frac{k}{c^2 \ln k} < x \leq k . \quad (3)$$

In particular $f(x) \geq 1$ for $x \in \{1, 2, \dots, k\}$, hence the algorithm is valid by (1).

Now we bound the total charge of type 3 any job i_0 can receive. Let \mathcal{J} denote the set of job units that are type 3 charged to i_0 . For each $(j, b) \in \mathcal{J}$ the charge from it is w_j/p_j , while $w_j\alpha^{p_j-1} \leq w_{i_0}$, by (1). Thus $w_j/p_j \leq w_{i_0}/(p_j\alpha^{p_j-1}) = w_{i_0}/f(p_j)$. Recall that Lemma 1 states that for every $p \leq k$ the number of $(j, b) \in \mathcal{J}$ such that $p_j \leq p$ is at most $p - 1$. Applying it for $p = k/(c^2 \ln k)$, and using (2) and (3), we get

$$\sum_{(j,b) \in \mathcal{J}} 1/f(p_j) \leq \frac{k}{c^2 \ln k} + \frac{k}{\ln k} = \frac{k}{\ln k} \left(1 + \frac{1}{c^2} \right) .$$

Putting things together, each job i_0 completed by the algorithm receives a type 1 charge of at most w_{i_0} . By Lemma 2 for $\rho = \alpha$ it can receive at most $w_{i_0} k / c^2 \ln k$ type 2 charges in total. And we have just shown that type 3 charges are, for large k , at most $w_{i_0} (1 + 1/c^2) k / \ln k$ in total. Together, this is $w_{i_0} (1 + 2/c^2) \cdot k / \ln k = w_{i_0} (3 + o(1)) \cdot k / \ln k$.

It remains to prove the claims (2) and (3). First let us observe that for every constant $c < 1$ and large enough x ,

$$\left(1 - \frac{c}{x}\right)^x \geq \frac{1}{e}, \quad (4)$$

as for x tending to infinity the left hand side tends to $e^{-c} > e^{-1}$.

Clearly $f(1) = 1$ and, by (4),

$$\begin{aligned} f(k) &= k \left(1 - \frac{c^2 \ln k}{k}\right)^{k-1} = k \left(1 - \frac{c^2 \ln k}{k}\right)^{\frac{k}{c \ln k} (k-1) \frac{c \ln k}{k}} \\ &\geq k \left(\frac{1}{e}\right)^{(k-1) \frac{c \ln k}{k}} = k \cdot k^{c(1-k)/k} = k^{(1-\epsilon+k\epsilon)/k} \geq \ln k, \end{aligned}$$

if k is sufficiently large.

Now we observe that the sequence $(f(x))_{x=1}^k$ is non-decreasing for $x \leq k/(c^2 \ln k)$ and decreasing for $x > k/(c^2 \ln k)$. For this we analyze the ratio $f(x)/f(x-1) = \alpha x/(x-1)$, and see that it is at least 1 if and only if $x \geq k/(c^2 \ln k)$. Inequalities (2) and (3) follow. This completes the proof. \square

6 Lower Bound, Bounded Processing Time, Unweighted Model

In this section we consider instances where every job has processing time at most k and unit weight.

Theorem 4. *Any deterministic online algorithm has ratio $\Omega(\log k / \log \log k)$.*

Proof. Fix some deterministic algorithm. We will define an instance denoted $I(\ell, 0, 0)$ from which the algorithm can complete at most a single job, and the adversary can complete ℓ jobs. Moreover all jobs have processing time at most $(\ell + 1)!$. So if we choose $\ell = \lfloor \ln k / \ln \ln k \rfloor - 1$, then the processing time is not more than

$$\begin{aligned} (\ell + 1)! &\leq \left\lfloor \frac{\ln k}{\ln \ln k} \right\rfloor! \leq \left(\frac{\ln k}{2 \ln \ln k} \right)^{\frac{\ln k}{\ln \ln k}} \\ &= \exp \left((\ln \ln k - \ln 2 - \ln \ln \ln k) \cdot \frac{\ln k}{\ln \ln k} \right) \leq \exp(\ln k) = k. \end{aligned}$$

Let $\ell \geq 1, s, e \geq 0$ be integers. Let f be a function defined as $f(1, e) = e + 1$ and for $\ell > 1$,

$$f(\ell, e) = \max\{e, f(\ell - 1, 0)\} + f(\ell - 1, 0) + f(\ell - 1, \max\{e, f(\ell - 1, 0)\}) . \quad (5)$$

We construct an instance $I(\ell, s, e)$ with the following properties.

- The adversary can schedule ℓ jobs from this instance.
- The algorithm can schedule at most one job from this instance, and if he does then he spends strictly more than e units on jobs from this instance, including jobs that he does not complete.
- All jobs i from the instance satisfy $s \leq r_i$ and $d_i \leq s + f(\ell, e)$, and therefore also $p_i \leq f(\ell, e)$.

The basis case is easy, for $I(1, s, e)$ at time s we release a tight job of length $e + 1$. It satisfies the required properties.

Now we show how to construct $I(\ell + 1, s, e)$. Let $b = f(\ell, 0)$, $a = \max\{e, b\}$ and $c = f(\ell, a)$. At time s we release a job A of length $a + c$ and deadline $s + a + b + c$, as well as a job B of length $a + b$ and tight deadline. At time $s + a$, if the algorithm scheduled only B in $[s, s + a)$, then we release instance $I(\ell, s + a, 0)$. Otherwise at time $s + a + b$ we release $I(\ell, s + a + b, a)$, see figure 2.

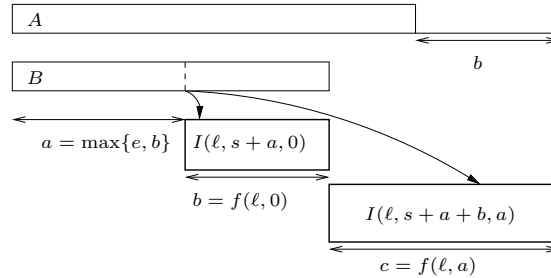


Fig. 2. The construction of $I(\ell + 1, s, e)$

Let's verify that the construction satisfies the required properties, by induction on ℓ . We already settled the basis case $\ell = 1$, so assume the claim holds for instances $I(\ell, s', e')$ for all $s', e' \geq 0$, and we will show it holds for $I(\ell + 1, s, e)$ as well. By construction and induction each job i from instance $I(\ell + 1, s, e)$ is not released before s and its deadline does not exceed $s + a + b + c = s + f(\ell + 1, e)$, so the third property is satisfied.

In case the algorithm scheduled only B in $[s, s + a)$: At this point, if the algorithm completes A or B , then in the interval $[s + a, s + a + b)$ there is not a single idle step left for another job. Therefore by induction hypothesis the algorithm can only schedule a single job. The algorithm already spent a units on B , so if he does complete a job, then he spends strictly more than $a \geq e$ units on jobs from this instance. By induction hypothesis, the adversary can schedule ℓ jobs from the subinstance in the interval $[s + a, s + a + b)$, and schedule A in the remaining time units $[s, s + a) \cup [s + a + b, s + a + c)$.

Otherwise: The algorithm cannot complete B , since the job is tight. Also if the algorithm completes some job from $I(\ell, s + a + b, a)$, then by induction hypothesis he spends strictly more than $a \geq e$ units on jobs from the subinstance. This does not leave enough space to complete job A in addition.

And if the algorithm completes job A , then he spends $a + c > e$ units on it. Now the adversary can complete B plus ℓ jobs from the sub-instance.

To complete the proof of the theorem, it remains to show that all jobs from $I(\ell, 0, 0)$ have processing time at most $(\ell + 1)!$. To this end, we prove by induction that

$$f(\ell, e) = \ell \max \{ \ell!, (\ell - 1)! + e \}, \quad (6)$$

which implies that all jobs from $I(\ell, 0, 0)$ have processing time at most $\ell \cdot \ell! < (\ell + 1)!$. Note that (6) trivially holds for $\ell = 1$. Now assume it holds for $\ell - 1$, and in particular $f(\ell - 1, 0) = (\ell - 1)(\ell - 1)!$. Then

$$\begin{aligned} f(\ell, e) &= \max\{e, f(\ell - 1, 0)\} + f(\ell - 1, 0) + f(\ell - 1, \max\{e, f(\ell - 1, 0)\}) \\ &= \max\{e, (\ell - 1) \cdot (\ell - 1)!\} + (\ell - 1) \cdot (\ell - 1)! + \\ &\quad + (\ell - 1) \max\{(\ell - 1)!, (\ell - 2)! + \max\{e, (\ell - 1) \cdot (\ell - 1)!\}\} \\ &= \max\{e, (\ell - 1) \cdot (\ell - 1)!\} + (\ell - 1) \cdot (\ell - 1)! + \\ &\quad + (\ell - 1) \cdot ((\ell - 2)! + \max\{e, (\ell - 1) \cdot (\ell - 1)!\}) \\ &= \ell \cdot \max\{e, (\ell - 1) \cdot (\ell - 1)!\} + \ell! \\ &= \ell \cdot \max\{e + (\ell - 1)!, \ell!\} \end{aligned} \quad (7)$$

The equality (7) follows from $(\ell - 1)! < (\ell - 2)! + \max\{e, (\ell - 1) \cdot (\ell - 1)!\}$. \square

7 Lower Bound, Equal Processing Time Model

We consider the model, where each job j has the same processing time $p_j = k$, but an arbitrary weight. We show a $1.5\sqrt{3} \approx 2.598$ lower bound on competitive ratio of any deterministic algorithm. This counterpart for the algorithm from Section 3 clearly indicates that the problem becomes harder when jobs do not have unit length and preemption matters; compare to the unit length variant [11, 7].

Lemma 3. *Any deterministic online algorithm for the equal processing time with $k \geq 2$ has competitive ratio at least $1.5\sqrt{3} \approx 2.598$.*

Proof. We describe the adversary's strategy for $k = 2$ only, as it can be easily adapted to larger values of k . Every job j will have processing time 2 and will be tight, i.e. $d_j = r_j + p_j = r_j + 2$. W.l.o.g. the adversary completes the heaviest feasible subset of jobs, which can be specified once the sequence is finished. For the time being we need only describe what jobs are released in each step. We also assume that when there are pending jobs with positive weights, ALG will process one of them, and that it will never process a job with non-positive weight.

Initially ($t = 0$) the adversary releases a job with weight $x_0 = 1$. In every step $t > 0$ the adversary releases a job with weight x_t that we specify later, unless the the algorithm has already completed one job (this has to be the one with weight x_{t-2}). In that case the adversary releases no job at time t and the

sequence is finished. The adversary completes every other job starting from the last one, for a total gain of

$$X_{t-1} = x_{t-1} + x_{t-3} + \dots + x_{b+2} + x_b ,$$

where $b = t - 1 \bmod 2$, while ALG's gain is only x_{t-2} .

Now we describe the sequence x_i that forces ratio at least $R = 1.5\sqrt{3} - \epsilon$ for arbitrarily small epsilon. If ALG completed a job released in step t , the ratio is

$$R_t = \frac{X_{t+1}}{x_t} = \frac{X_{t+1}}{X_t - X_{t-2}} ,$$

assuming $X_{-2} = X_{-1} = 0$. As we want to force ratio R , we let $R_t = R$, i.e.

$$X_{t+1} = R(X_t - X_{t-2})$$

for each $t > 0$. Note that this defines the sequence x_i , as $x_i = X_i - X_{i-2}$.

Now we claim that there exists an i such that $x_i = X_i - X_{i-2} \leq 0$. Notice that if i_0 is the minimum i such that $x_i \leq 0$, this implies that (w.l.o.g.) any algorithm will complete a task released before step i_0 .

Let us introduce two sequences: $q_i = R \cdot X_{i-1}/X_{i+1}$ and $s_i = R - q_i = R(1 - X_{i-1}/X_{i+1})$. We shall derive a recursive formula defining q_i and s_i , and then prove that s_i is a strictly decreasing sequence. Next we shall prove that $s_i \leq 0$ for some i . That will conclude the proof, as (assuming both X_{i-1} and X_{i+1} are positive)

$$s_i \leq 0 \iff q_i \geq R \iff \frac{X_{i-1}}{X_{i+1}} \geq 1 \iff X_{i-1} \geq X_{i+1} \iff x_{i+1} \leq 0 .$$

Of course, if $X_{i-1} > 0$ and $X_{i+1} \leq 0$, then $x_{i+1} < 0$ as well.

To prove existence of appropriate i first observe that

$$X_i = R(X_{i-1} - X_{i-3}) = X_{i-1} \left(R - R \frac{X_{i-3}}{X_{i-1}} \right) = X_{i-1} (R - q_{i-2}) ,$$

which implies

$$q_i = R \cdot \frac{X_{i-1}}{X_{i+1}} = \frac{R}{(R - q_{i-1})(R - q_{i-2})} . \quad (8)$$

Rewriting (8) in terms of s_i we get

$$s_i = R \left(1 - \frac{1}{s_{i-1}s_{i-2}} \right) , \quad (9)$$

and one can calculate that $s_0 = R$, $s_1 = R - 1/R$ and $s_2 = R(R^2 - 2)/(R^2 - 1)$, in particular $s_0 > s_1 > s_2 > 0$.

We prove by induction that s_i is a decreasing sequence. Observe that

$$s_{i+1} - s_i = R \left(\frac{1}{s_{i-1}s_{i-2}} - \frac{1}{s_i s_{i-1}} \right) = R \cdot \frac{s_i - s_{i-2}}{s_i s_{i-1} s_{i-2}} < 0 ,$$

since by induction hypothesis $s_{i-2} > s_{i-1} > s_i$. Hence there is i such that $s_i \leq 0$, unless the sequence s_i is bounded and converges to $g = \inf s_i$, s.t. $g \geq 0$. Suppose that is the case. Then s_i converges to g and (9) holds for $s_i = s_{i-1} = s_{i-2} = g$. Thus

$$g = R \left(1 - \frac{1}{g^2} \right) ,$$

or, equivalently,

$$P(g) = g^3 - Rg^2 + R = 0 \tag{10}$$

Since $R = 1.5\sqrt{3} - \epsilon$, the discriminant of P , which is $4R^2(R^2 - 27/4)$, is negative, i.e. P has a single real root. As $P(-1) = -1$ and $P(0) = R > 0$, the sole real root of P lies in $(-1, 0)$. In particular, it is negative, which proves s_i is not lower-bounded by any non-negative constant. \square

8 Conclusion

It remains open to determine the best competitive ratio a deterministic algorithm can achieve for the equal processing time model. Already for $k = 1$ the question is not completely answered.

How much the competitive ratio can be improved by use of randomization remains unknown. The only paper [4] we are aware of studies the case of oblivious adversary and tight weighted jobs only. It provides a lower bound of $\Omega(\sqrt{\log k / \log \log k})$ and an upper bound of $O(\log k)$ on competitive ratio in that setting. Can a similar ratio be achieved when jobs are not tight?

We would like to thank Artur Jež for his valuable comments.

Bibliography

- [1] Philippe Baptiste. An $O(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Oper. Res. Lett.*, 24(4):175–180, 1999.
- [2] Philippe Baptiste, Marek Chrobak, Christoph Dürr, Wojciech Jawor, and Nodari Vakhania. Preemptive scheduling of equal-length jobs to maximize weighted throughput. *Operations Research Letters*, 32(3):258–264, 2004.
- [3] S.K. Baruah, J. Haritsa, and N. Sharma. On-line scheduling to maximize task completions. *Real-Time Systems Symposium*, pages 228–236, Dec 1994.
- [4] Ran Canetti and Sandy Irani. Bounding the power of preemption in randomized scheduling. *SIAM J. Comput.*, 27(4):993–1015, 1998.
- [5] Bo Chen, Chris N. Potts, and Gerhard J. Woeginger. *Handbook of Combinatorial Optimization*, volume 3, chapter A review of machine scheduling: Complexity, algorithms and approximability, pages 21–169. Kluwer Academic Publishers, 1998.
- [6] Marek Chrobak, Wojciech Jawor, Jiri Sgall, and Tomáš Tichý. Online scheduling of equal-length jobs: Randomization and restarts help. *SIAM J. Comput.*, 36(6):1709–1728, 2007.
- [7] Matthias Englert and Matthias Westermann. Considering suppressed packets improves buffer management in QoS switches. In *Proc. 18th Symp. on Discrete Algorithms (SODA)*, pages 209–218. ACM/SIAM, 2007.
- [8] Bala Kalyanasundaram and Kirk R. Pruhs. Maximizing job completions online. *J. Algorithms*, 49(1):63–85, 2003.
- [9] Chiu-Yuen Koo, Tak-Wah Lam, Tsuen-Wan Ngan, Kunihiko Sadakane, and Kar-Keung To. On-line scheduling with tight deadlines. *Theoretical Computer Science*, 295(1-3):251 – 261, 2003.
- [10] E. L. Lawler. Knapsack-like scheduling problems, the Moore-Hodgson algorithm and the “tower of sets” property. *Mathl. Comput. Modelling*, 20(2):91–106, 1994.
- [11] Fei Li, Jay Sethuraman, and Clifford Stein. Better online buffer management. In *Proc. 18th Symp. on Discrete Algorithms (SODA)*, pages 199–208. ACM/SIAM, 2007.
- [12] Hing-Fung Ting. A near optimal scheduler for on-demand data broadcasts. *Theoretical Computer Science*, 401(1-3):77 – 84, 2008.
- [13] Nodari Vakhania. A fast on-line algorithm for the preemptive scheduling of equal-length jobs on a single processor. In *Proceedings of the 2nd WSEAS International Conference on Computer Engineering and Applications*, pages 158–161, 2008.

A Lower Bound, Bounded Processing Time Model

In this section we consider instances where all jobs have processing time at most k and arbitrary weights. Ting [12] showed that competitive ratio of any deterministic algorithm in this setting is at least $k/(2 \ln k) - 1$, while we improve it to $k/\ln k - o(1)$.

Lemma 4. *For any deterministic algorithm its competitive ratio is at least $k/\ln k - o(1)$. In particular it is at least $k/\ln k - 0.06$ for $k \geq 16$.*

Proof. For convenience denote $R = k/\ln k$, $r = \lceil R \rceil - 1$, and assume $k \geq 16$. Fix any deterministic algorithm and consider the following instance, depicted in figure 3. At time 0, the adversary releases a big job B with weight $w_B = R$, processing time k and deadline k , as well as a small job A_1 with weight, processing time and deadline all 1. Moreover, at each moment $0 \leq t \leq k - 1$, if the algorithm scheduled only job B in $[0, t)$, then the adversary releases a tight job A_{t+1} of unit processing time at time t , and does not release any new job otherwise. The jobs A_t have weights:

$$w(A_t) := \begin{cases} 1 & \text{if } t < R, \\ e^{t/R-1} & \text{if } t \geq R. \end{cases}$$

Note, job A_t is released at time $t - 1$.

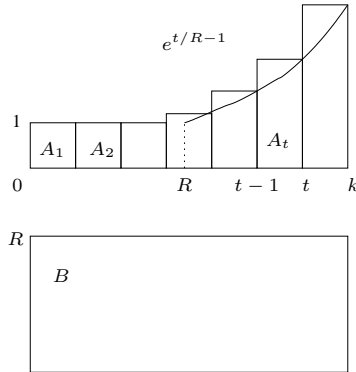


Fig. 3. The construction of the lower bound

If the algorithm schedules a job A_{t_0} with $t_0 < R$, then the adversary schedules job B and the ratio is R .

If the algorithm schedules a job A_{t_0} with $t_0 \geq R$, then the adversary schedules all jobs A_t for $t = 1, \dots, t_0$. The adversary's gain is

$$\begin{aligned}
\lceil R \rceil - 1 + \sum_{t=\lceil R \rceil}^{t_0} e^{t/R-1} &= r + \sum_{t=r+1}^{t_0} e^{t/R-1} \geq r + \int_r^{t_0} e^{t/R-1} dt \\
&= r + \left[R e^{t/R-1} \right]_r^{t_0} = r - R e^{r/R-1} + R e^{t_0/R-1} \\
&\geq f(R, r) + R e^{t_0/R-1} = f(R, r) + R w(A_{t_0}) , \quad (11)
\end{aligned}$$

where the inequalities follow from monotonicity of the function $e^{t/R-1}$, and

$$f(R, r) = r - R e^{r/R-1} .$$

So the adversary gain is at least $k/\ln k$ times the algorithm's gain plus $f(R, r)$.

If the algorithm schedules job B , gaining $k/\ln k$, the adversary schedules all k jobs A_t from $t = 0$ to $k - 1$. In that case, by (11) its gain is at least

$$f(R, r) + R e^{k/R-1} = f(R, r) + R e^{\ln k-1} = f(R, r) + R \cdot k/e,$$

and we need it to be more than $f(R, r) + R w(B) = f(R, r) + R^2$. This is true if $e \leq \ln k$ which holds for $k \geq e^e$, in particular when $k \geq 16$.

Now we analyze the function $f(R, r)$. Recall that $R = k/\ln k$ and $r = \lceil R \rceil - 1$, so in particular $R - r \in (0, 1]$. As $e^x \geq 1 + x$ and both sides converge to 1 as x tends to 0, we have

$$f(R, r) = r - R e^{r/R-1} \leq r - R \cdot \frac{r}{R} = 0 ,$$

and $f(R, r)$ tends to 0 as k grows.

In particular, it is straightforward to check that $f(R(k), r(k)) \geq -0.06$ for $k = 16, 17, \dots, 21$, and that $f(R(k), r(k)) \geq f(7, 1) > -0.06$ for larger k . As the algorithm's gain is (w.l.o.g.) at least 1, $f(R, r)$ divided by that gain is at least $f(R, r)$, which concludes the proof. \square