# A lower bound for scheduling of unit jobs with immediate decision on parallel machines

Tomáš Ebenlendr[1] and Jiří Sgall[1]

Institute of Mathematics, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic.
ebik@math.cas.cz, sgall@math.cas.cz

**Abstract.** Consider scheduling of unit jobs with release times and deadlines on $m$ identical machines with the objective to maximize the number of jobs completed before their deadlines. We prove a new lower bound for online algorithms with immediate decision. This means that the jobs arrive over time and the algorithm has to decide the schedule of each job immediately upon its release. Our lower bound tends to $e/(e-1) \approx 1.58$ for many machines, matching the performance of the best algorithm.

## 1 Introduction

Suppose that we have unit jobs that arrive over time. Each job arrives at its release time and has a deadline, these times are integers. The goal is to schedule as many jobs as possible before their deadlines, on $m$ identical machines. In the online setting, at each time $t$ the algorithm chooses at most $m$ jobs to be started at time $t$ (among the jobs released before or at $t$, with a deadline strictly after $t$ and not scheduled yet). This is a very simple online problem: At each time $t$ we schedule $m$ jobs with the earliest deadlines. This generates an optimal schedule.

In this note, we study a modification of this problem called scheduling with *immediate decision*, introduced and studied in [5, 4]. In this variant, the online algorithm has to decide the schedule of the newly released jobs immediately after they are released. This means that at time $t$, the schedule of jobs with release time $t$ is fixed, and even if a job is scheduled to start only at time $t' > t$, its schedule cannot be changed later. Obviously, this is harder for the online algorithm, and, for example, the optimal algorithm described above does not work in this model.

In [4], Ding *et al.* presented an online algorithm with immediate decision with the competitive ratio decreasing to $e/(e-1)$ for $m \to \infty$. It works even for the more general case when the processing times are equal (but possibly larger than 1), with the same competitive ratio. This algorithm is actually very simple: The machines are kept sorted by decreasing completion times, i.e., the first machine is the one that would complete the currently assigned jobs latest. The newly released jobs are processed one by one, so that each job is scheduled on the first machine at the completion time of that machine; if that would violate the deadline, try the second machine, and so on; if no machine works, the job is rejected.

The obvious question is: Is there a better algorithm with immediate decision at least for unit jobs?

**Our results.** We prove that no algorithm for unit jobs with immediate decision on $m$ machines has a competitive ratio smaller than

$$r_m = \frac{e^{\frac{m-1}{m}}}{e^{\frac{m-1}{m}} - \frac{m}{m+1}} .$$

For $m \to \infty$, $r_m$ decreases to $e/(e-1) \approx 1.582$. For $m = 2$, $r_2 \approx 1.678$, and a few more values are listed in Table 1.

Our lower bound shows that the simple algorithm from [4] is optimal at least in the limit for large $m$. This is true even for unit jobs, showing that for scheduling with immediate decision, handling the unit jobs is almost as hard as scheduling jobs with equal processing times. In most online settings, scheduling unit jobs is significantly easier compared to jobs with equal processing times, thus we find the new lower bound quite surprising.

Note also that for our problem, as well as for the basic variant without immediate decision, it is natural that more machines allow algorithms with better competitive ratio, because it is possible to keep a fraction of machines available for the jobs that arrive later. In fact, for unit jobs, this can be formalized: The case of $m$ machines is equivalent to the case of a single machine with an additional restriction that all release times and all deadlines are multiples of $m$. Thus the competitive ratio for $m$ is at least the ratio for $m'$ whenever $m$ divides $m'$.

| m | 2 | 3 | 4 | 5 | $\cdots$ | $\to \infty$ |
|---|---|---|---|---|---|---|
| **a lower bound with immediate decision, unit jobs [new result]** | 1.678 | 1.626 | 1.607 | 1.598 | | 1.582 |
| an algorithm with immediate decision, equal processing times [4] | 1.8 | 1.730 | 1.694 | 1.672 | | 1.582 |
| a lower bound with immediate decision, equal processing times [4] | 1.8 | – | – | – | | 1.333 |
| an algorithm without immediate decision, equal processing times [5, 7] | 1.5 | – | – | – | | – |

**Table 1.** Summary of new and previous results.

**Previous results.** The exact competitive ratio of the algorithm from [4] with immediate decision is

$$R_m = \frac{1}{1 - \left(\frac{m}{m+1}\right)^m} ,$$

see Table 1 for a few values. The only previous lower bound for scheduling of unit jobs with immediate decision is a bound of 1.6 for $m = 2$ [4].

Another related and more general model, as we already mentioned, assumes that all jobs have the same processing time $p$. This is a significantly harder problem, as the release times and deadlines do not need to be multiples of $p$. Thus, for example, a new job can arrive when all the machines are committed to process other jobs. All the results below are for equal processing times.

It is known that a greedy algorithm is 2-competitive for any number of machines and this is optimal among the deterministic algorithms for a single machine [1]. For a single machine without immediate decision there also exists a 5/3-competitive randomized algorithm [3] and no better than 4/3-competitive randomized algorithm is possible [6].

For the case of two machines, two 1.5-competitive deterministic algorithms without immediate decision were designed independently in [5, 7]. This competitive ratio is optimal for $m = 2$ without immediate decision. So, for $m = 2$ and equal processing times, immediate decision increases the competitive ratio from 1.5 to 1.8; the lower bound of 1.8 for immediate decision is from [4].

For $m \geq 3$, the algorithm of [4] is the best algorithm currently known (and in fact the only one better than the 2-competitive greedy algorithm). We have no better algorithm even without immediate decision for equal processing times. A standard example gives a lower bound of 4/3 with immediate decision, as noticed in [4], while for general algorithms the lower bound approaches 6/5 for large $m$ [5].

**Preliminaries and notations.** We are given $m$, the number of the machines, and $n$ jobs. Each job is described by a pair of numbers: a release time $r_j$ and a deadline $d_j$; these numbers are integers. Each job has a unit processing time. The goal is to maximize the number of jobs completed by the deadline (the number of early jobs). We allow the algorithm to reject some jobs; in fact, w.l.o.g., we restrict ourselves to schedules where each job is either rejected or completed by its deadline.

In the *online* version, each job is released at its release time $r_j$, at this time also its deadline $d_j$ becomes known. The algorithm does not know that the job exists before this time. Moreover, if the online algorithm has the *immediate decision* property, it must decide the schedule of the job as soon as it is released and cannot change this decision later. Unrestricted online algorithms decide which jobs to start at each time, leaving the remaining jobs uncommitted. We use the standard competitive analysis: The algorithm is *c-competitive* if, for every input, it schedules at least $1/c$ fraction of the number of jobs scheduled by the optimum schedule.

We denote the starting time of a job (that is not rejected) by $s_j$. The job then occupies the time interval $[s_j, s_j + 1)$ on some machine. This means that each job (that is not rejected) must be scheduled so that $r_j \leq s_j \leq d_j - 1$. With unit processing times and integral release times, we restrict ourselves to integral starting times, both for the optimum and the online algorithms without loss of generality: Whenever non-integral starting times would occur, we can move the jobs forward one by one to start at $\lfloor s_j \rfloor$, with no loss in the performance. Since the jobs are aligned, we do not need to know which particular machine a job is assigned to. A valid machine assignment is available if and only if $|\{j \mid s_j = t\}| \leq m$ for each time $t$. Our goal is to maximize the number of properly scheduled jobs.

## 2 The idea of the proof

In this section we describe the idea of the lower bound. The exact proof is given in the next section.

As usual, our lower bound is formulated as an adversary strategy in a game between a deterministic algorithm and an adversary who has the power to react to the actions of the algorithm. The adversary releases some jobs at time $t$ (thus $r_j = t$ for these jobs and $d_j$ is set by the adversary for each job independently). Once the jobs are released, the algorithm schedules (or rejects) all these jobs and then the time advances to the next release time decided by the adversary.

**Adversary strategy.** The adversary starts with a sufficiently long interval $[0, T)$. This means that the adversary first releases a few jobs with the release time 0 and the deadline $T$. Due to the immediate decision property, the algorithm has to commit to the schedule of these jobs. By averaging, we can find a big part of the interval where the algorithm schedules at least the average number of jobs and such that the adversary can schedule all the jobs outside of this part. Then the adversary uses the same procedure recursively.

Now we do a few rough calculations to see how this idea gives the lower bound of $e/(e-1)$, disregarding various rounding issues. So now we describe the recursive process in more detail. For simplicity, let us also assume that the algorithm always schedules the released jobs so that they are spread uniformly over the feasible interval. (Later we show that no other algorithm performs much better against our adversary.)

During the process, at time $t$, the adversary has scheduled all the previously released jobs before $t$, while the algorithm has already scheduled $x$ jobs in the remaining interval $[t, T)$ of length $l = T - t$. We call $[t, T)$ the active interval and we say that its density is $\rho = x/(ml)$. Then the adversary at time $t$ releases $\varepsilon ml$ jobs with deadlines equal to $T$, for a small $\varepsilon$. The adversary schedules them before time $t' = t + \varepsilon l$. The density increases to $\rho + \varepsilon$ on $[t, T)$ as well as on the interval $[t', T)$ (due to the uniform spreading assumption). The adversary then increases time to $t'$ and continues until the density increases to 1.

We express the length $l$ of the active interval as a function of the density $\rho$. When $\rho$ increases by $\varepsilon$, then $l$ decreases by $\varepsilon l$. Taking $\varepsilon$ infinitesimally small, we get a differential equation $dl/d\rho = -l$. We have the initial condition $l(0) = T$, and thus the equation is solved by the function $l(\rho) = e^{-\rho} \cdot T$. So, starting with the length $T$, the adversary ends with an interval of length at least $l = l(1) = T/e$, during which all time steps have $m$ jobs scheduled in the schedule of the algorithm but no jobs in the schedule of the adversary. At this point, both the adversary and the algorithm have scheduled $m(T - l) = (1 - 1/e)mT$ jobs, as all the released jobs exactly fit before the active interval.

Now the adversary simply releases the final batch of $lm$ jobs that cannot be scheduled outside the active interval. The adversary schedules all of these jobs while the algorithm has to reject them. The adversary schedules the total of $mT$ jobs, while the algorithm only $(1 - 1/e)mT$ jobs, yielding the lower bound of $e/(e - 1)$.

**Technical issues.** The sketch of the proof above needs to be properly formalized. In the end, the proof is somewhat technical, as there are many issues that we have to deal with. Here we sketch the main issues and the ways to solve them.

**Finding the dense part.** First, we cannot assume that the algorithm spreads the jobs evenly. We need to find a dense part of a given length on which we focus in the recursion. This is done essentially by an averaging argument. Unfortunately, the dense part does not necessarily form a single interval. Instead, it can be composed of two non-overlapping intervals (and this is sufficient). This, in turn, makes the recursion more difficult. The number of intervals increases exponentially. The recursive procedure arranges them naturally in a tree of nested intervals. At any time, we have a list of active intervals instead of just one, and we release jobs corresponding to the interval which starts first. This corresponds to traversing the tree of intervals in the depth-first order. To analyze the length of the intervals, however, we always need to argue about the total length of the intervals on one level of the tree.

**Discretization and rounding.** We need to argue that by taking a small $\varepsilon$, the bound obtained from the continuous version of the recursion can be approximated arbitrarily well. We also need to account for various rounding errors and the fact that the adversary cannot release two batches of jobs at the same time. To this end, we use an initial active interval of length exponential in $1/\varepsilon$ and carefully bound the errors. In general, we release slightly more jobs than can fit in the adversary schedule and let the adversary reject some of them, to make the calculations simpler.

**Improving the bound for small $m$.** To improve the bound for small $m$, we stop the iterative process at density $(m - 1)/m$ instead of 1. Instead of increasing the density by $\varepsilon$, we increase it by almost $1/m$ in a single last phase. Then we present the final set of jobs as $m$ tight jobs for each time step in the schedule where the density is 1.

**Handling the rejected jobs.** So far we have assumed that the algorithm schedules all the released jobs. This is not necessarily true, and with immediate decision, it may seem that it could possibly be an advantage for an algorithm to reject a job and keep more options for later steps. However, a simple exchange argument shows that every algorithm can be modified so that no jobs are rejected, unless all machines are occupied during the whole feasible interval.

Taking all this into account, we give an adversary strategy which for any $\varepsilon > 0$ generates an instance showing that the competitive ratio is at least $e^{\frac{m-1}{m}}/(e^{\frac{m-1}{m}} - \frac{m}{m+1} + O(\varepsilon))$.

## 3 The lower bound

We first define the density and state the lemma which ensures the adversary to find dense intervals after releasing some jobs.

The density of an interval is defined as the number of jobs scheduled in it by the adversary, divided by the maximal number of jobs that can fit. Notice that the density depends on the schedule of the algorithm; in particular, it may increase during the online process but it can never decrease.

**Definition 1.** *Suppose that the algorithm has scheduled $x$ jobs during the time interval $[t_1, t_2)$. Then the density of the interval is $\rho_{[t_1,t_2)} = x/(m(t_2 - t_1))$. For an empty interval, i.e., $t_2 = t_1$, we define the density to be 1.*

**Lemma 1.** *Given an interval $[t_1, t_2)$ with density $\rho$ and an integer $l \leq t_2 - t_1$, we can find one or two non-overlapping intervals with total length $l$, each of them having the density at least $\rho$.*

*Proof.* Let $t$ be the smallest time such that $t \geq t_2 - l$ and $\rho_{[t,t_2)} \geq \rho$. (Note that $t = t_2$ is always eligible.) If $t = t_2 - l$, then we have a single dense interval of length $l$ and we are done.

Otherwise we take $[t, t_2)$ as one of the intervals and look for another interval of length $l' = l - (t_2 - t)$. Let $q = \lceil (t - t_1)/l' \rceil - 1$; we have $q > 0$ as $l < t_2 - t_1$. By the choice of $q$, we have $t_1 + ql' \geq t - l' = t_2 - l$. Thus the interval $[t_1 + ql', t_2)$ has density less than $\rho$ as otherwise we would choose $t \leq t_1 + ql'$. Considering the density of the whole interval, $[t_1, t_1 + ql')$ has density at least $\rho$. It can be covered by disjoint intervals $[t_1 + (i-1)l', t_1 + il')$ for $i = 1, \ldots, q$; thus one of these intervals has density at least $\rho$ as well, and it can be chosen as the desired second interval of length $l'$. □

Now we are ready to prove the main result.

**Theorem 1.** *Let $A$ be a deterministic online algorithm for scheduling unit jobs with release times and deadlines on $m$ machines with the objective to maximize the number of accepted jobs. If $A$ satisfies the restriction of immediate decision then its competitive ratio is at least $r_m = e^{\frac{m-1}{m}} / (e^{\frac{m-1}{m}} - \frac{m}{m+1})$.*

*Proof.* We proceed in several major steps roughly following the sketch in the previous section.

**Handling rejections.** First we observe that any algorithm can be modified so that no job is rejected, unless all machines are occupied during the whole feasible interval of that job.

Suppose that the online algorithm $A$ does not satisfy this restriction. Using $A$, we construct a new algorithm $B$ which never rejects a job, unless it has to, and accepts at least the same number of jobs. At a given time, $B$ schedules all the newly released jobs at the same time and the same machine as $A$, if that slot is empty. The remaining jobs, both those rejected by $A$ and those not yet scheduled by $B$ because the corresponding slot was not empty, are processed one by one in an arbitrary order. If there is an empty slot (i.e., a pair of machine and time) during the feasible interval of a job, it is scheduled to any such slot. Otherwise the job is rejected.

We claim that at any moment, the algorithm $B$ has scheduled a job to any slot where $A$ has scheduled one. For a given slot, this property can be violated only at the time when $A$ schedules a job to that slot. However, if $B$ has this slot still empty, it puts the same job there. Thus, at the end, $B$ has scheduled at least as many jobs as $A$.

From now on, we assume that $A$ is the modified algorithm which never rejects a job unless during its feasible interval all the machines are full. This is important as our adversary strategy relies on gradually increasing the density.

**An overview of the adversary strategy.** Choose $k$ a sufficiently large integer. Let $\varepsilon = (m-1)/(mk)$ and $T = \lceil 2^k/\varepsilon^2 \rceil$.

The adversary starts with one active interval $[0,T)$. Throughout the process, the adversary maintains a list of disjoint non-empty active intervals such that all these intervals start at the current time or later. Upon reaching the start of the interval, after some action of the adversary (typically releasing some jobs and waiting for the algorithm to schedule them), the interval is removed from the list and possibly replaced by one or two disjoint subintervals with strictly larger starting time. Then we let the time advance and continue the process as long as there is any active interval on the list.

Each interval has a level: The initial interval $[0,T)$ has level 0. Each subsequent interval created while processing an interval at level $i$ has level $i+1$. During the process we guarantee that each active interval at level $i$ has density at least $i\varepsilon$. The maximal level of an interval is $k$; at this point the density is at least $(m-1)/m$. Overall, we create and process less than $2^{k+1}$ intervals during the whole process.

The action of the adversary at the start time of the interval depends on the level of the interval. If the level is less than $k$, the adversary increases the density as described below, with the exception of the intervals of length at most $2/\varepsilon$ that are ignored. If the level of the processed interval is $k$, we do a more complicated phase described later, which guarantees that the algorithm rejects many jobs; in this case no new interval is introduced.

**Increasing the density by $\varepsilon$.** Suppose that the first active interval is $[t_1, t_2)$ at level $i < k$. Thus its density is at least $\rho_{[t_1,t_2)} \geq i\varepsilon$. Denote the length of the interval by $l = t_2 - t_1$. If $l \leq 2/\varepsilon$, the adversary removes this interval without any further action.

Otherwise, the adversary submits $\varepsilon lm + m$ jobs with $r_j = t_1$ and $d_j = t_2$. The density $\rho_{[t_1+1,t_2)}$ increases to at least $\rho_{[t_1,t_2)} + \varepsilon \geq (i+1)\varepsilon$ after the algorithm schedules the released jobs, as at most $m$ jobs (old or new) may be scheduled at time $t_1$.

Let $l' = \lceil e^{-\varepsilon} l \rceil$ be the desired length of the dense subintervals. Note that we use a factor of $e^{-\varepsilon}$ in place of $1-\varepsilon$ in the intuitive description; this approximation is good for small $\varepsilon$ and makes it possible to bound the error from discretization. Using elementary calculus we verify that $1 - e^{-x} \geq x(1-x)$ for all $x$, and we obtain

$$l - l' > (1 - e^{-\varepsilon})l - 1 \geq \varepsilon(1-\varepsilon)l - 1. \tag{1}$$

In particular, we have $l > l'$ due to our restriction $l > 2/\varepsilon$ and $\varepsilon < 1/2$ (which can be guaranteed by taking a large $k$).

Now the adversary applies Lemma 1 to find one or two disjoint subintervals of $[t_1 + 1, t_2)$ with total length $l' = \lceil le^{-\varepsilon} \rceil$ with density at least $(i+1)\varepsilon$; we can apply the lemma to this shorter interval as $l' < l$. These one or two intervals are added to the list of active intervals, and $[t_1, t_2)$ is removed. Note that the new active intervals are at level $i + 1$ and they have the density $(i+1)\varepsilon$ required for this level.

The adversary schedules $(l - l')m$ of the new jobs during $[t_1, t_2)$ but outside the new active intervals and rejects the remaining jobs (if any). Using (1), it follows that the number of jobs rejected by the adversary is at most

$$\varepsilon lm + m - (l - l')m \leq \varepsilon lm + m - (\varepsilon(1-\varepsilon)lm - m) = 2m + \varepsilon^2 lm. \tag{2}$$

**The final phase (at level $k$).** In the remaining case, the first active interval is $[t_1, t_2)$ at level $k$. Denote its length by $l = t_2 - t_1$. Note that the density is at least $(m-1)/m$.

The adversary releases $\lceil lm/(m+1) \rceil$ jobs with $r_j = t_1$ and $d_j = t_2$. Considering the total number of jobs and the fact that at most $m$ jobs may be scheduled during each time step, it follows that after the algorithm schedules the new jobs, there are at least $\lceil lm/(m+1) \rceil - 1$ time steps in the interval $[t_1+1, t_2)$ where the algorithm scheduled $m$ jobs. The adversary chooses exactly $\lceil lm/(m+1) \rceil - 1$ of these full time steps, and for each chosen time step $[t, t+1)$, it releases $m$ tight jobs with $r_j = t$ and $d_j = t + 1$. All these jobs are rejected by the algorithm.

The adversary schedules the jobs released at $t_1$ during $[t_1, t_2)$ but outside the new active intervals. The number of available time steps is

$$l - \left\lceil \frac{lm}{m+1} \right\rceil + 1 = \left\lfloor \frac{l}{m+1} \right\rfloor + 1 \geq \left\lceil \frac{l}{m+1} \right\rceil,$$

thus no job is rejected. The tight jobs are then scheduled during their corresponding time steps, so the adversary does not reject any of these jobs, either.

**Bounding the competitive ratio.** To bound the competitive ratio, we first bound the number of jobs rejected by the adversary and by the algorithm $A$; we denote these by $R_{adv}$ and $R_A$, respectively.

Let $L$ be the total length of the intervals at level $k$. The total length of all intervals that are removed because they are too short, over all levels, is at most $2^k \cdot 2/\varepsilon$. The total length of the remaining intervals decreases by at most a factor of $e^{-\varepsilon}$ at each level. Thus the overall length is at least

$$L \geq e^{-k\varepsilon}T - \frac{2^{k+1}}{\varepsilon} = e^{-\frac{m-1}{m}}T - \frac{2^{k+1}}{\varepsilon}.$$

At a level smaller than $k$, using (2), the adversary rejects at most $2m$ jobs per interval plus $\varepsilon^2 lm$ for each interval of length $l$. Since the intervals at the same level are disjoint, their total length is at most $T$ for each level. No jobs are rejected at level $k$. Thus, overall, the number of jobs rejected by the adversary is at most

$$R_{adv} \ \leq \ 2m \cdot 2^k + k \cdot \varepsilon^2 mT \ = \ 2m \cdot 2^k + \varepsilon(m-1)T \ \leq \ 3\varepsilon \cdot mT,$$

where the last inequality follows by our choice of $T$.

The algorithm $A$ rejects jobs at the level $k$, for an interval of length $l$ it rejects at least $lm^2/(m+1) - m$ jobs. Since there are at most $2^k$ intervals at the level $k$, the number of rejected jobs is at least

$$\begin{aligned} R_A &\geq \frac{m^2}{m+1}L - m2^k \ \geq \ \frac{m}{m+1}e^{-\frac{m-1}{m}} \cdot mT - \frac{m2^{k+1}}{\varepsilon} - m2^k \\ &\geq \left(\frac{m}{m+1}e^{-\frac{m-1}{m}} - 3\varepsilon\right) mT, \end{aligned}$$

where the last inequality follows by our choice of $T$ again.

The competitive ratio is at least $(n - R_{adv})/(n - R_A)$, where $n$ is the number of released jobs. As the ratio is larger than 1, the bound decreases with $n$, and we need to upper bound $n$. Using a simple fact that the adversary schedules at most $mT$ jobs, we have $n \leq mT + R_{adv}$. Thus the competitive ratio is at least

$$\begin{aligned} \frac{n - R_{adv}}{n - R_A} &\geq \frac{mT}{mT + R_{adv} - R_A} \\ &\geq \frac{mT}{mT + 3\varepsilon \cdot mT - (\frac{m}{m+1}e^{-\frac{m-1}{m}} - 3\varepsilon)mT} \ = \ \frac{1}{1 - \frac{m}{m+1}e^{-\frac{m-1}{m}} + 6\varepsilon}. \end{aligned}$$

For a sufficiently large $k$, we have $\varepsilon$ arbitrarily small, and thus the lower bound approaches the claimed bound of

$$r_m = \frac{1}{1 - \frac{m}{m+1}e^{-\frac{m-1}{m}}} = \frac{e^{\frac{m-1}{m}}}{e^{\frac{m-1}{m}} - \frac{m}{m+1}},$$

and the proof is complete. □

## 4 Conclusions

With immediate decision, our lower bound still leaves a gap for small $m$, for scheduling unit jobs. It gives an intuition what is needed for an algorithm to perform better than the algorithm of [4]: If a number of jobs with the same deadline is released, they should be spread uniformly between the release time and the deadline, and not packed starting from the release time. However, it seems quite hard to define and analyze such an algorithm once there are many different deadlines.

It is interesting to consider randomized algorithms instead of deterministic ones. Our problem can be viewed as a special case of online bipartite matching (matching jobs to slots in the schedule). Thus there exists an $e/(e-1)$-competitive algorithm. (A simplified proof and further references can be found in [2]. Their proof actually gives only an asymptotic result, but a simple padding argument proves the bound of $e/(e-1)$ for any instance.) Our lower bound can be modified to yield a lower bound of $e/(e-1)$ for randomized algorithms for any number of machines $m$. Since it is based on averaging arguments, it can be used for a randomized algorithm if we simply replace density by expected density. The only change that is needed is that we omit the last phase which is used to improve the deterministic lower bound for small $m$. We postpone the details to the journal version of this paper.

More fundamental problem in this area is to close the gap for general algorithms. It is quite surprising that for $m \geq 3$ we have no algorithms that would go beyond the immediate decision restriction.

**Acknowledgments**

We are grateful to an anonymous referee to draw our attention to the randomized case. We thank Marek Chrobak and Rob van Stee for pointers and discussions regarding online matching.

Partially supported by Institutional Research Plan No. AV0Z10190503, by Inst. for Theor. Comp. Sci., Prague (project 1M0545 of MŠMT ČR), and grant IAA1019401 of GA AV ČR.

## References

1. S. K. Baruah, J. Haritsa, and N. Sharma: *On-line scheduling to maximize task completions.* J. Comb. Math. Comb. Comput., **39** (2001), pp. 65–78. A preliminary version appeared in Proc. 15th Real-Time Systems Symp., IEEE, 1994, pp. 228–236.
2. Benjamin E. Birnbaum, Claire Mathieu: *On-line bipartite matching made simple.* SIGACT News **39** (2008): pp. 80-87.
3. M. Chrobak, W. Jawor, J. Sgall, and T. Tichý: *Online scheduling of equal-length jobs: Randomization and restarts help.* SIAM J. Comput., **36** (2007), pp. 1709-1728. A preliminary version appered in Proc. 31st International Colloquium on Automata, Languages, and Programming (ICALP), vol. 3142 of Lecture Notes in Comput. Sci., Springer, 2004, pp. 358–370.
4. J. Ding, T. Ebenlendr, J. Sgall and G. Zhang: *Online scheduling of equal-length jobs on parallel machines.* In Proc. 15th European Symp. on Algorithms (ESA), vol. 4698 of Lecture Notes in Comput. Sci., Springer, 2007, pp. 427-438.
5. J. Ding and G. Zhang: *Online scheduling with hard deadlines on parallel machines.* In Proc. 2nd International Conf. on Algorithmic Aspects in Information and Management (AAIM), vol. 4041 of Lecture Notes in Comput. Sci., Springer, 2006, pp. 32–42.
6. S. A. Goldman, J. Parwatikar, and S. Suri: *Online scheduling with hard deadlines.* J. Algorithms, **34** (2000), pp. 370–389.
7. M. H. Goldwasser and M. Pedigo: *Online, non-preemptive scheduling of equal-length jobs on two identical machines.* In Proc. 10th Scandinavian Workshop on Algorithm Theory (SWAT), vol. 4059 of Lecture Notes in Comput. Sci., Springer, 2006, pp. 113–123. To appear in ACM Transactions on Algorithms.