J Sched (2013) 16:539–547
DOI 10.1007/s10951-012-0288-7

Lower bounds for online makespan minimization on a small number of related machines

Łukasz Jeż · Jarett Schwartz · Jiří Sgall · József Békési

Published online: 12 September 2012

Abstract In online makespan minimization, the jobs characterized by their processing time arrive one-by-one and each has to be assigned to one of the m uniformly related machines. The goal is to minimize the length of the schedule. We prove new combinatorial lower bounds for $m = 4$ and $m = 5$, and computer-assisted lower bounds for $m \leq 11$.

Keywords Online algorithms · Scheduling · Makespan · Related machines

Ł. Jeż (✉)
Institute of Computer Science, University of Wrocław, ul. Joliot-Curie 15, 50-383 Wrocław, Poland
e-mail: lje@cs.uni.wroc.pl

Ł. Jeż
Institute of Mathematics, Academy of Sciences of the Czech Republic, Žitná 25, 115 67 Praha 1, Czech Republic

J. Schwartz
Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Malostranské nám. 25, 11800 Praha 1, Czech Republic
e-mail: jarett@kam.mff.cuni.cz

J. Sgall
Faculty of Mathematics and Physics, Computer Science Inst. of Charles University, Malostranské nám. 25, 11800 Praha 1, Czech Republic
e-mail: sgall@iuuk.mff.cuni.cz

J. Békési
Department of Applied Informatics, Gyula Juhász Faculty of Education, University of Szeged, POB 396, 6701 Szeged, Hungary
e-mail: bekesi@jgypk.u-szeged.hu

1 Introduction

In this article we prove new lower bounds for online makespan minimization for a small number of uniformly related machines. The competitive ratio (as a function of the number of machines) for this fundamental problem is upper bounded by a constant but there has been little progress on determining its exact value, as well as the values for a fixed m.

The instance of this problem consists of a sequence of machines with possibly different speeds and a sequence of jobs specified by their processing times. A *schedule* assigns each job to one of the machines; the time needed to process a job is equal to its processing time divided by the speed of the machine where it is assigned. The objective is to minimize the *makespan* (also called the length of the schedule, or the maximal completion time). Usually a schedule also needs to specify the timing of each job (its starting and completion times) so that the jobs on each machine do not overlap. Due to the simplicity of the problem we consider, this is not necessary and it is sufficient to specify the assignment to the machines, silently assuming that each job is started as soon as all the previous jobs on its machine are processed. Instead of calculating the completion times individually for each job, we can calculate the completion time of each machine as the total processing time of the jobs allocated to it divided by the speed of the machine; the makespan is then the maximum of the completion times over all machines.

In the online version of the problem, jobs appear online *one-by-one*. When a job appears, an online algorithm has to make an irrevocable decision and assign the job to a machine. This decision is permanent and made without the knowledge of the future jobs; the algorithm is not even aware of whether any future jobs exist or not. An online algorithm is *R-competitive* if for each instance it produces

Table 1 Previous and our bounds for online makespan minimization on uniformly related machines; LS stands for List Scheduling

m	Lower bounds		Algorithms	
	Previous	New	Best known	LS
2	$\phi \approx 1.618$	–	–	$\phi \approx 1.618$
3	2	–	–	2
4	–	2.141391	–	2.2248
5	–	2.314595	–	2.4143
6	2.2880	2.347312	–	2.5812
7	–	2.439957	–	<2.7321
8	–	2.439957	–	<2.8709
9	2.4380	2.462775	–	≤3
10	–	2.483120	–	<3.1214
11	–	2.502672	–	<3.2361
∞	2.5648		5.8284	$\Theta(\log m)$

a schedule with makespan at most R times the optimal makespan.

We prove lower bounds of 2.141391 for $m = 4$ and 2.314595 for $m = 5$. Our lower bound is based on an instance where the processing times are a geometric sequence, similarly as in previous work (Berman et al. 2000; Epstein and Sgall 2000; Ebenlendr and Sgall 2012). The speeds are chosen so that any online algorithm can use only the two fastest machines, called the active machines. The bound is then obtained by carefully analyzing the possible patterns of scheduling the jobs on these machines.

Generalizing this to larger values of m, we use computer search for elimination of possible patterns and give instances with up to five active machines.

Our and previous bounds are summarized in Table 1.

Related work It is known that for two and three machines, the tight bounds are 1.618 and 2, respectively (see the next paragraph). Two other lower bounds for a small number of machines were 2.2880 for $m = 6$ and 2.4380 for $m = 9$ by Berman et al. (2000). That paper is not interested in the exact number of machines, as its focus is on the overall bounds, and consequently their choice of the speeds of machines is not optimal. Their lower bounds can be modified using our Lemma 1 to work for $m = 5$ and $m = 7$, respectively; see an explanation at the end of Sect. 2. For an arbitrary (large) number of machines, the current lower bound is 2.5648 (Ebenlendr and Sgall 2012), the only improvement of the mentioned lower bound of 2.4380 for $m = 9$.

Naturally, the lower bounds need to be compared to the existing algorithms. For a small number of machines the best currently known algorithm is the greedy List Scheduling (LS). Here List Scheduling is defined so that the next job is always scheduled so that it will finish as early as possible. Its competitive ratio for $m = 2$ is exactly $\phi \approx 1.618$, the golden ratio, and for $m \geq 3$ it is at most $1 + \sqrt{(m-1)/2}$; this bound is tight for $3 \leq m \leq 6$ (Cho and Sahni 1980). Moreover, for $m = 2, 3$ it can be checked easily that there is no better deterministic algorithm. For $m = 2$ it is even possible to give the exact optimal ratio for any speed combination and it is always achieved by greedy List Scheduling (Epstein et al. 2001). Already for three machines, it is not known exactly for which speed combinations List Scheduling is optimal, even though we know it is optimal in the worst case. Some recent progress is reported in Cai and Yang (2012), Han et al. (2011). Another special case when some partial results about optimality of List Scheduling are known is the case when $m - 1$ machines have the same speed, see, e.g., Han et al. (2011), Musitelli and Nicoletti (2011).

For an arbitrary (large) number of machines, the greedy algorithm is far from optimal: its competitive ratio is $\Theta(\log m)$ (Aspnes et al. 1997). The first constant competitive algorithm for non-preemptive scheduling on related machines was developed in Aspnes et al. (1997). The currently best algorithms are $3 + \sqrt{8} \approx 5.828$-competitive deterministic and 4.311-competitive randomized ones (Berman et al. 2000). For an alternative very nice presentation see Bar-Noy et al. (2000). All these algorithms use doubling, i.e., strategies that work with some estimate of the optimal makespan and when it turns out that the estimate is too low, it is multiplied by 2 or some other constant. While this is a standard technique for obtaining a constant competitive ratio, it would be surprising if it led to optimal algorithms. Designing better algorithms both for small and large number of machines remains one of the central open problems in this area.

Very little is known for randomized algorithms in addition to the 4.311-competitive algorithm mentioned above. For an arbitrary m, the current lower bound is 2, see Epstein and Sgall (2000). For two machines, we know that the best randomized algorithm has competitive ratio between 1.5 and 1.527 (Epstein et al. 2001). We are not aware of any other studies of randomized algorithms for online makespan minimization on related machines.

The problem of non-preemptive scheduling can be formulated in the language of online load balancing as the case where the jobs are permanent and the load is their only parameter corresponding to our processing time. Consequently, there are many results on load balancing that extend the basic results on online scheduling in a different direction, see, e.g., Azar (1998).

It is interesting to compare our results to the related problem of preemptive scheduling on uniformly related machines. In that problem, each job can be divided into several pieces that may be scheduled on different machines so that the corresponding time slots are non-overlapping; the time slots may also be non-consecutive, so that at some times a partially completed job is not being processed at all. Contrary to our (non-preemptive) problem, with preemption it

is necessary to specify the schedule completely, including the timing information. In the online version, for each job we have to specify its schedule completely before the next job is revealed. Interestingly, for this preemptive problem, it is possible to provide an optimal online algorithm for any combination of the speeds and its competitive ratio is between 2.1163 and $e \approx 2.7182$, see Ebenlendr et al. (2009), Ebenlendr (2011). Similar results seem to be out of reach for non-preemptive scheduling, as the combinatorial structure is much more difficult and the value of the optimum is NP-hard to compute, while for the preemptive scheduling it is efficiently computable, in fact given by an easy formula.

The state of the art should be also compared with makespan minimization on identical machines (i.e., the special case with all the speeds equal). There the competitive ratio of List Scheduling is known to be equal to $2 - 1/m$ (Graham 1966), and it is optimal for $m = 2$ and $m = 3$ (Faigle et al. 1989). Better algorithms are known for $m = 4$ (Chen et al. 1994) and larger (Galambos and Woeginger 1993). For an arbitrary m, the optimal competitive ratio is between 1.88 (Rudin 2001) and 1.92 (Fleischer and Wahl 2000).

2 Preliminaries

We number the machines as well as the jobs from 0 (to obtain simpler formulas). Thus we have machines $M_0, M_1, \ldots, M_{m-1}$ and jobs $J_0, J_1, \ldots, J_{n-1}$. The speed of machine M_i is denoted s_i; we order the machines so that their speeds are non-increasing. The processing time of job J_j is denoted p_j; thus the job takes time p_j/s_i to be processed on M_i.

For a given sequence of jobs $\mathcal{J}$, let $\mathcal{J}_i$ be the set of indices of jobs scheduled on machine M_i. The completion time of the machine is then simply the sum of processing times of the jobs scheduled to the machine divided by its speed: $C_i = \frac{1}{s_i} \sum_{j \in \mathcal{J}_i} p_j$. We compare the maximum completion time in the output of the algorithm with the maximum completion time of the optimal schedule.

In all our lower bounds, we let $\mathcal{J}$ denote an infinite sequence of jobs $j_0, j_1, \ldots$, where j_i has processing time $p_i = \alpha^i$ for some $\alpha > 1$. We shall consider the algorithm's assignment of jobs for the prefixes of $\mathcal{J}$; we denote the prefix of $\mathcal{J}$ consisting of the first $t + 1$ jobs $j_0, j_1, \ldots, j_t$ by $\mathcal{J}[t]$.

2.1 Three machines

We first briefly review the proof of the tight lower bound of 2 for $m = 3$, to introduce the main ideas.

The machine M_0 has speed 1, the machines M_1 and M_2 have speed $1/2$. We set $\alpha = 2$, i.e., $p_i = 2^i$. We observe that the optimal makespan for $\mathcal{J}[t]$ is $p_t = 2^t$: j_t is scheduled on M_0, j_{t-1} on M_1 and all the remaining jobs on M_2. Assume we have an algorithm with a competitive ratio smaller than 2. If a job j_t is scheduled on one of the slow machines, the makespan on $\mathcal{J}[t]$ is $2p_t$, twice the optimum. Thus the algorithm schedules all jobs on M_0. But then the makespan on $\mathcal{J}[t]$ is $2p_t - 1$, by taking t sufficiently high we get a contradiction again.

2.2 More machines

To obtain a lower bound of R for more machines, we arrange the instance so that an R-competitive algorithm is forced to schedule all jobs on the k fastest machines (instead of one for $m = 3$), for some k. With $k = 2$ we obtain combinatorial bounds for $m = 4, 5$ while for larger values of k we use computer search and obtain bounds for larger k. The value of α will be chosen later, separately for each case.

The machines $M_0, M_1, \ldots, M_{k-1}$ are called *active* machines; we set the speed of the active machine M_i to $s_i = \alpha^{-i}$. All the remaining machines, i.e., $M_k, M_{k+1}, \ldots, M_{m-1}$ are called *inactive* and have the same speed s_k. This speed is chosen so that:

(A) $s_k \le 1/R$ and
(B) for any t, the optimal makespan for $\mathcal{J}[t]$ equals p_t.

The condition (A) limits the possible lower bound on R and needs to be verified separately in each case. As we shall see later, for some values of α, k and m it gives the tightest bound on R, while for other values not. In any case, it is desirable to choose s_k as small possible, so that we have a chance of proving a large bound on R.

Thus we investigate what is the smallest possible speed s_k such that (B) holds, and it turns out that a simple argument gives an exact bound. Due to our choice of speeds of active machines, the k largest jobs of $\mathcal{J}[t]$ fit on the k active machines so that they all complete exactly at time p_t. Thus it is necessary that all the remaining jobs together do not overflow the capacity of all the inactive machines. With this in mind, we set the speeds

$$\begin{aligned} s_k = s_{k+1} &= \cdots = s_{m-1} \\ &= \alpha^{-k} \cdot \max\left\{1, \frac{1}{m-k} \sum_{i \ge 0} \alpha^{-i}\right\} \\ &= \alpha^{-k} \cdot \max\left\{1, \frac{\alpha}{(m-k)(\alpha-1)}\right\}, \end{aligned} \tag{1}$$

and in Lemma 1 we prove that this choice of s_k indeed guarantees that (B) holds.

Lemma 1 *The optimum makespan for $\mathcal{J}[t]$ is $p_t = \alpha^t$.*

Proof First observe that the optimum makespan is at least p_t because processing j_t alone takes p_t on M_0, the fastest machine. To prove that a schedule with the makespan p_t indeed exists, we assign the jobs greedily in the decreasing order of their processing times, i.e., in a reverse order, from j_t, $j_{t-1}, \ldots,$ to j_0. The job j_{t-i}, for $i = 0, 1, \ldots, k-1$ is assigned to M_i. Thus M_i will take $p_{t-i}/s_i = \alpha^t = p_t$ to process its only job. If $t < k$, we are done.

If $t \geq k$, define the *available work* of a machine M as p_t multiplied by M's speed minus the total processing time of jobs already assigned to M. Thus M's available work is the maximum total processing time of jobs that can yet be assigned to M while keeping its completion time no larger than p_t. We prove the following claim.

Claim *When the job j_n is to be assigned, the total available work on the $m-k$ inactive machines is at least $p_n \cdot \max\{m-k, \frac{\alpha}{\alpha-1}\}$.*

Assuming the claim holds, one of the machines has available work of at least p_n, thus j_n can be assigned to it. Applying this to all jobs yields the desired schedule.

It remains to prove the claim. First we observe that it holds for $n_0 = t - k$: At this point, no job is yet assigned to inactive machines, so using (1), the total available work is

$$(m-k)p_t s_k = (m-k)p_t\alpha^{-k} \cdot \max\left\{1, \frac{1}{m-k} \cdot \frac{\alpha}{\alpha-1}\right\} = p_{n_0} \cdot \max\left\{m-k, \frac{\alpha}{\alpha-1}\right\}.$$

To prove the claim for $n < n_0$, note that before scheduling j_n, the available work decreases by exactly $p_{n+1} = \alpha^{n+1}$ once j_{n+1} is scheduled. Thus to prove the claim, assuming its validity for $n+1$, it is sufficient to show that the bound in the claim decreases by at least α^{n+1}. As the actual decrease is

$$\begin{aligned}&(p_{n+1} - p_n) \cdot \max\left\{m-k, \frac{\alpha}{\alpha-1}\right\}\\ &= \left(\alpha^{n+1} - \alpha^n\right) \cdot \max\left\{m-k, \frac{\alpha}{\alpha-1}\right\}\\ &\geq \alpha^n(\alpha-1)\frac{\alpha}{\alpha-1} = \alpha^{n+1},\end{aligned}$$

this completes the proof of both the claim and the lemma. □

The lower bound proofs in each case continue by examining the possible patterns of scheduling jobs on the active machines. Intuitively it is clear that the best algorithm uses an eventually periodic pattern, which then matches the lower bound. For the actual proof we proceed by gradually excluding more and more patterns. For $k = 2$ active machines this requires only a few steps and thus can be performed by hand. For larger k we use computer search.

We note that our lower bounds, both combinatorial and computer-assisted ones, are optimized in the following sense: For every choice of α, assuming the speeds and the jobs in the instance are as above, and assuming that R equals our lower bound, there either exists a periodic pattern so that the sequence can be scheduled on the active machines with makespan at most R times the optimum or the speed of the inactive machines is larger than $1/R$ and thus the algorithm can use them.

Finally, let us explain our previous remark concerning the lower bounds from Berman et al. (2000). In these bounds, the authors set the speeds of inactive machines to continue the geometric sequence of the speeds of the active machines, only the last machine has the speed equal to the sum of the tail of the geometric sequence, so that all the small jobs exactly fit. Thus, in their analysis the analog of our Lemma 1 holds trivially. However, using Lemma 1 we can change the speeds of the inactive machines to be equal and thus decrease their number from $m = 6$ to $m = 5$ for the bound of 2.288 for two active machines and from $m = 9$ to $m = 7$ for the bound of 2.438 for three active machines, without any other changes in their proofs.

3 Combinatorial lower bounds

We give improved lower bounds for $m = 4$ and $m = 5$ machines. In both cases we use only $k = 2$ active machines. However, having three inactive machines instead of only two allows us to use smaller α and consequently obtain stronger lower bound for five machines.

3.1 Four machines

Theorem 2 *Let $\alpha \approx 1.72208$ be the largest real root of $z^4 - z^3 - z^2 - z + 1$. For makespan minimization on $m = 4$ uniformly related machines there exists no online algorithm with competitive ratio smaller than*

$$R = \frac{\alpha^4}{\alpha^3 - 1} = 1 + \frac{\alpha(\alpha+1)}{\alpha^3 - 1} \approx 2.141391. \quad (2)$$

Proof Assume for the sake of contradiction that an online algorithm has competitive ratio smaller than R. Lemma 1 then implies that, for every t, the algorithm's makespan on $\mathcal{J}[t]$ is strictly smaller than $R \cdot p_t$. Consider the algorithm's assignment of jobs to machines while serving $\mathcal{J}$.

Note that $\alpha/(2(\alpha-1)) \approx 1.1476 > 1$, so $s_2 = s_3 = 1/(2\alpha(\alpha-1)) \approx 0.3654$ by (1). Thus $s_2 \leq 1/R \approx 0.4670$, so the algorithm never assigns any job to any inactive machine, as otherwise the makespan on $\mathcal{J}[t]$ is at least $p_t/s_2 \geq p_t \cdot R$ after scheduling some j_t on an inactive machine.

Fact 3 *The algorithm never assigns two out of three consecutive jobs to M_1.*

Proof Suppose that the algorithm does assign two out of three consecutive jobs to M_1, and that j_t (for some t) is the later of those two jobs. Then the completion time of M_1 on $\mathcal{J}[t]$ is at least $(p_t + p_{t-2})/s_1 = p_t \cdot (1 + 1/\alpha^2) \cdot \alpha = p_t \cdot (\alpha + 1/\alpha) \approx 2.3 \cdot p_t > R \cdot p_t$, a contradiction. □

Using Fact 3, we further notice that the assignment to M_0 is also constrained.

Fact 4 *There exists an integer t_0 such that the algorithm never assigns three consecutive jobs, j_{t-2}, j_{t-1}, and j_t to M_0 for any $t \geq t_0$.*

Proof Suppose that the algorithm does assign j_{t-2}, j_{t-1}, and j_t to M_0. The algorithm never assigns any job to an inactive machine, and by Fact 3 it assigned at most every third job from $\mathcal{J}[t-3]$ to M_1. Thus the completion time of M_0 on $\mathcal{J}[t]$ divided by p_t is at least

$$\frac{p_t + p_{t-1} + p_{t-2} + p_{t-4} + p_{t-5} + p_{t-7} + p_{t-8} + \cdots}{p_t \cdot s_0},$$

which for $t \to \infty$ tends to

$$1 + \left(\frac{1}{\alpha} + \frac{1}{\alpha^2}\right) \cdot \frac{\alpha^3}{\alpha^3 - 1} = R,$$

where equality follows from (2), a contradiction. □

Together, Facts 3 and 4 imply that after the initial t_0 steps, the algorithm's behavior is fixed: its assignment has period 3, within which the algorithm assigns the first job to M_1 and the other two to M_0.

Take a large enough t such that the algorithm assigned j_t to M_1. As $t \to \infty$, the ratio of the completion time of M_1 on $\mathcal{J}[t]$ to p_t tends to

$$\frac{1}{p_t \cdot s_1}(p_t + p_{t-3} + p_{t-6} + \cdots) \xrightarrow[t\to\infty]{} \alpha \cdot \frac{\alpha^3}{\alpha^3 - 1} = R,$$

where equality follows from (2). This is the final contradiction. □

3.2 Five machines

Theorem 5 *Let $\alpha \approx 1.52138$ be the only real root of $z^3 - z - 2$. For makespan minimization on $m = 5$ uniformly related machines there exists no online algorithm with competitive ratio smaller than $R = \alpha^2 \approx 2.314595$.*

Proof Assume for the sake of contradiction that an online algorithm has competitive ratio smaller than R. Lemma 1 then implies that, for every t, the algorithm's makespan on $\mathcal{J}[t]$ is strictly smaller than $R \cdot p_t$. Consider the algorithm's assignment of jobs to machines while serving $\mathcal{J}$.

The choice of α implies that

$$\frac{\alpha + 1}{\alpha^3 - 1} = 1 \quad \text{and} \quad R = \frac{\alpha^2(\alpha + 1)}{\alpha^3 - 1} \approx 2.3146. \tag{3}$$

Note that $\alpha/(3(\alpha - 1)) \approx 0.9727 < 1$, so $s_2 = s_3 = s_4 = 1/\alpha^2 = 1/R$ by (1). Since $s_2 = 1/R$, the algorithm never assigns any job to any inactive machine, as otherwise the makespan on $\mathcal{J}[t]$ is at least $p_t/s_2 = p_t \cdot R$ after scheduling some j_t on an inactive machine.

Fact 6 *The algorithm never assigns two consecutive jobs, j_{t-1} and j_t, to M_1.*

Proof Suppose that the algorithm does assign j_{t-1} and j_t to M_1 for some t. Then the completion time of M_1 on $\mathcal{J}[t]$ is at least $(p_t + p_{t-1})/s_1 = p_t \cdot (1 + 1/\alpha) \cdot \alpha = p_t \cdot (\alpha + 1) \approx 2.5214 \cdot p_t > R \cdot p_t$, a contradiction. □

Using Fact 6, we further notice that the assignment to M_0 is also constrained.

Fact 7 *There exists an integer t_0 such that the algorithm never assigns three consecutive jobs, j_{t-2}, j_{t-1}, and j_t, to M_0 for any $t \geq t_0$.*

Proof Suppose that the algorithm does assign j_{t-2}, j_{t-1}, and j_t to M_0. The algorithm never assigns any job to an inactive machine, and by Fact 6 it assigned at most every other job from $\mathcal{J}[t-3]$ to M_1. Thus the completion time of M_0 on $\mathcal{J}[t]$ divided by p_t is at least

$$\frac{1}{p_t \cdot s_0}\big((p_t + p_{t-1} + p_{t-2}) + (p_{t-4} + p_{t-6} + \cdots)\big),$$

which for $t \to \infty$ tends to

$$1 + \frac{1}{\alpha} + \frac{1}{\alpha^2 - 1} \approx 2.4179 > R,$$

a contradiction. □

Now with Fact 7 in turn we realize that assignment to M_1 is further constrained.

Fact 8 *There exists an integer t_1 such that the algorithm never assigns two jobs j_{t-2} and j_t to M_1 for any $t \geq t_1$.*

Proof Let $t_1 \geq t_0 + 3$. Suppose that the algorithm does assign j_{t-2} and j_t to M_1. As $t \geq t_0 + 3$, the algorithm did not assign all three jobs j_{t-3}, j_{t-4}, j_{t-5} to M_0. Therefore, the completion time of M_1 on $\mathcal{J}[t]$ is at least $(p_t + p_{t-2} +$

$p_{t-5})/s_1 = p_t \cdot (1 + 1/\alpha^2 + 1/\alpha^5) \cdot \alpha \approx 2.365 p_t > R \cdot p_t$, a contradiction. □

Together, Facts 6 up to 8 imply that after the initial t_1 steps, the algorithm's behavior is fixed: its assignment has period 3, within which the algorithm assigns the first job to M_1 and the other two to M_0.

Take large enough t such that the algorithm assigned j_t and j_{t-1} to M_0 and j_{t-2} to M_1. As $t \to \infty$, the ratio of the completion time of M_0 on $\mathcal{J}[t]$ to p_t tends to

$$\frac{1}{p_t}(p_t + p_{t-1} + p_{t-3} + p_{t-4} + \cdots) \xrightarrow[t\to\infty]{} \left(1 + \frac{1}{\alpha}\right) \cdot \frac{\alpha^3}{\alpha^3 - 1} = R,$$

where the last equality follows from (3). This is the final contradiction. □

4 Computer search

The Lower Bounds described in Berman et al. (2000) were found via search through possible assignments of jobs to $k = 3$ active machines. We present a generalization of their computer search technique in order to find the exact pattern of allocations that maximizes the lower bound obtainable for $k = 3, 4, 5$ active machines. It also confirms the previous section's results for $k = 2$ active machines.

First, we describe our technique. Fix some competitive ratio R that we want to achieve. We maintain a vector $S = (S_0, \ldots, S_{k-1})$ that tracks the relative load on each of the active machines. Now, we build a graph with states representing all such vectors such that $S_i \le R\alpha^{-i}$ and edges being the possible vectors after a single job is scheduled. Our initial state is $(0, \ldots, 0)$. To get the relative load after a job scheduled on machine i on a given state S, we first divide each entry of our relative load vector S by α and then add 1 to the ith entry of our vector. So, we get up to k edges out of each vertex.

This is the same infinite graph as the one considered by Berman et al. (2000), though they only considered $k = 3$ active machines. We also consider all possible competitive ratios R, whereas they always set R to equal α^k. Clearly, there is an R-competitive deterministic scheduling algorithm if and only if there is an infinite path in the graph starting from the initial state. In order to make this graph computer searchable, Berman et al. made the graph finite by discretizing the S vector. This, however, led to large rounding errors except for large choices of n, the factor of discretization. So, we use a different method for which it is sufficient to search only a part of the graph.

Rather than building the entire graph, we only build the tree generated by scheduling r jobs for a small choice of r. Normally, this would generate a tree with k^r states, but since we do not include states with $S_i > R\alpha^{-i}$, many branches of the tree are pruned. From this graph, we can determine if there is a path of length r in the infinite graph. Thus, for a given choice of r, α, and k, if there is no path of length r, then there is no deterministic scheduling algorithm with competitive ratio R, giving us a lower bound.

As we are proving a lower bound of R, the common speed of the inactive machines has to be at most $1/R$, or else it would be possible to schedule at least one job on the inactive machines. Hence, by Lemma 1, we find that the number of inactive machines, $m - k$, is no less than $(R/\alpha^k) \sum_{i \ge 0} \alpha^{-i} = R \cdot \alpha^{1-k}(\alpha - 1)^{-1}$. So, this limits our choice of α to a certain range for a given combination of k and m. We maximize the choice of R over this range to obtain our lower bound.

We can also calculate the pattern that the online algorithm can follow to achieve a competitive ratio close to our lower bound. We select an R slightly larger than the maximum R for which the graph was finite. Then, any sufficiently long path is guaranteed to follow an optimal pattern; in principle there could be several optimal patterns, actually we did not even prove that there exists one. However, it is sufficient for us to find a single one matching the lower bound, since any such pattern shows that our lower bound cannot be improved further—and we have done this for each m for the optimal value of α.

As we saw in the analysis of $m = 4$ and $m = 5$, such pattern does not necessarily make the completion times of all machines equal, i.e., for some i the ith entry might be much smaller than α^{k-i}. While for our choice of α all paths in the tree will follow the pattern after some point, there is some flexibility in the allocation of the first few jobs, as they are very small. As we are using DFS to find a single long path in the tree in order to find the optimal pattern, our search is likely to find the pattern after inspecting only a small number out of many feasible initial allocations. After finding a cyclic pattern of length ℓ, it is simple to check if it attains the desired ratio by inspecting the relative load vector that it yields. Our method works for values of k up to 5. We are unable to find lower bounds and matching patterns for $k \ge 6$, as the tree becomes too large.

As described, we are able to verify for a given k, m, α and R if there is an R-competitive algorithm for our sequences. For the optimal values of α, it is feasible to find R by binary search. To speed up computations, we have not searched the whole tree, but only a random sample of the initial branches. This works in practice, since if there is an infinite branch and R is not too close to the real bound, most initial branches can be extended. If an infinite branch is found, it is proven that the bound is smaller. If none is found, we first verify the result by extending the random sample and then we have verified the results by a complete search for selected values of α.

Table 2 The results of the computer search for lower bounds, together with cyclic patterns that attain upper bounds slightly larger than these lower bounds

k	m	α	Pattern	h	R
2	4	1.722081	001	1	2.141391
2	5	1.521380	001	0	2.314595
3	6	1.450217	00102100102010010201001020 1	2	2.347312
3	7	1.346256	0010201012	1	2.439957
4	8	1.346256	0010201012	1	2.439957
4	9	1.255564	010210301201021030120 1023	2	2.462775
5	10	1.222412	010321041230012013021041023012	0	2.483120
5	11	1.209132	010213020140312010230412010321040120310210340120132010423… …0102103410201301204120310201 34	4	2.502672

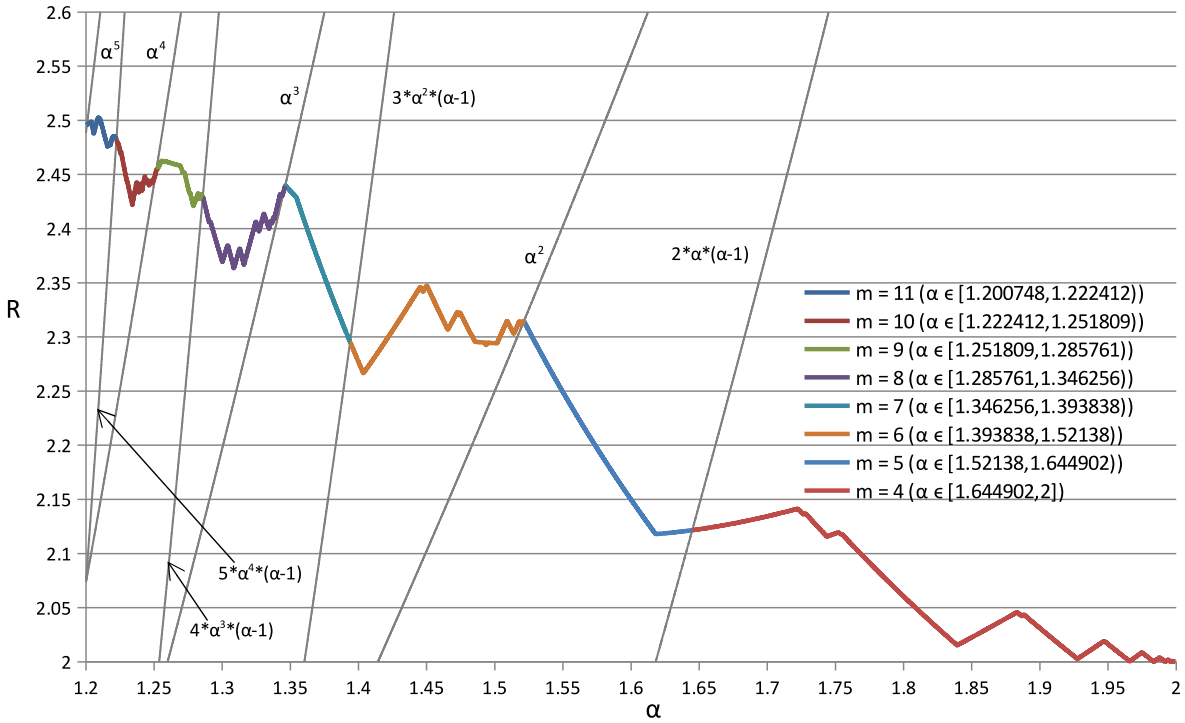

Fig. 1 The graph presents what lower bounds our strategy yields for $\alpha \in (1.2, 2.0)$

Our computer search results are presented in Table 2 and, in more detail, in Figs. 1, 2 and 3. The data in Table 2 are presented as follows. First, the number of active machines, k, then the optimal number of machines, m, then the approximation to the optimal value of α. This is followed by the optimal pattern, presented as follows. The integer i denotes M_i, the ith fastest machine (counting from 0), and the pattern repeatedly assigns jobs to the machines in the order given. The next column gives the number h of the machine M_h that attains the highest load for this pattern and this value of α. Finally, the last column gives the value of the lower bound ratio R given by the computer search and matched by the pattern found.

Figure 1 shows the bounds for all values of m and k. For each value of α, the largest lower bound R is displayed. The values of m and k are implicit, as given by the condition (A) and setting of the speeds. Most importantly, for a given α and R, k is the smallest number such that $\alpha^k \geq R$, since $s_k \geq \alpha^{-k}$ and (A) would be violated otherwise. Similarly, since $(m-k)s_k \geq \alpha^{1-k}/(\alpha-1)$, the number of inactive machines $(m-k)$ needs to satisfy $(m-k)\alpha^{k-1}(\alpha-1) \geq R$. These bounds on R are drawn as thin, fast increasing functions. As α decreases, the number of active or inactive ma-

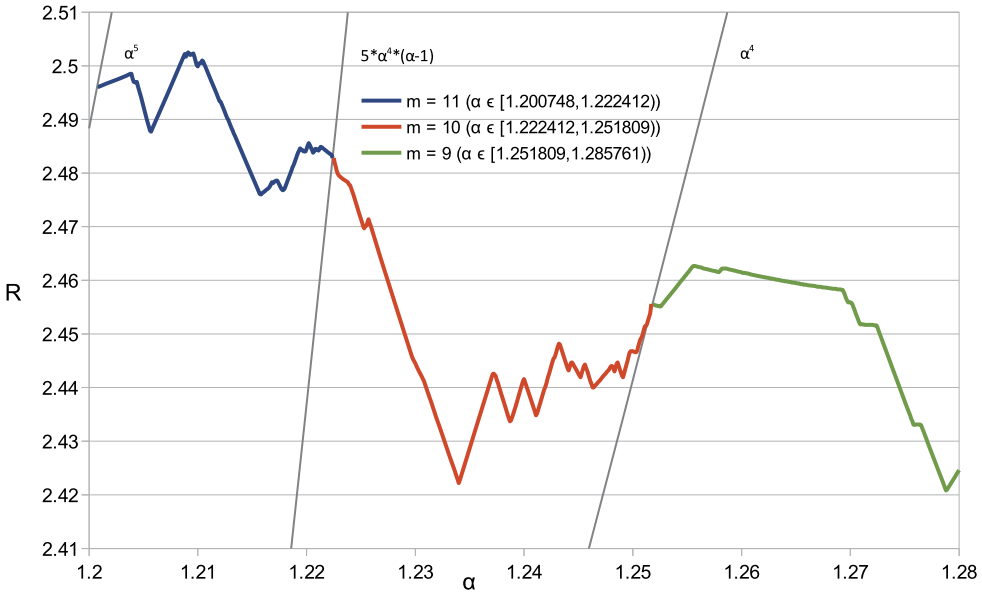

Fig. 2 A zoom-in on the *left part* of Fig. 1, i.e., $\alpha \in (1.20, 1.28)$

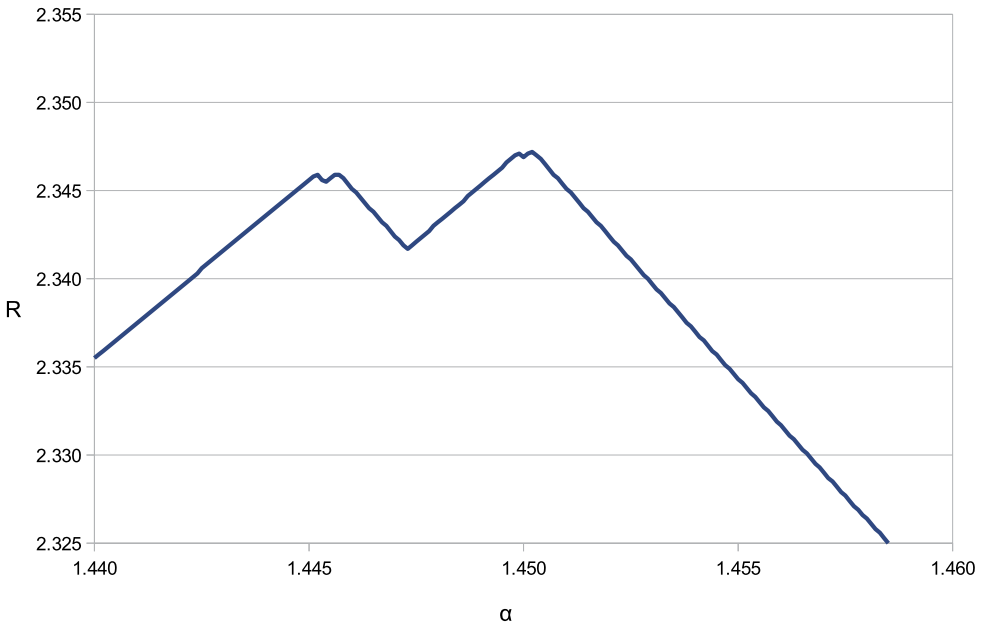

Fig. 3 A zoom-in on the *center part* of Fig. 1, i.e., $\alpha \in (1.44, 1.46)$

chines increases once one of these curves is crossed. The largest values of α correspond to $m = 4$ (and $k = 2$), but as α decreases, the curve for R crosses $1/s_2 = 2\alpha(\alpha - 1)$, so an additional inactive machine is required for smaller α. Hence at that point the region corresponding to $m = 5$ and $k = 2$ starts, and extends to the left until the R curve crosses $1/s_2 = \alpha^2$; at that point an active machine is added and the region corresponding to $m = 6$ and $k = 3$ starts, and so on.

The patterns found for $k = 2$ match the patterns found in our analysis of $m = 4, 5$. The lower bound we found for $k = 3$ is slightly better than the previous lower bound found by computer search for three active machines (2.438).

In general, the lower bounds increase with increasing number of machines. However, as the figures show, the exact dependence on α is complicated.

While the optimal R is at most α^k in general, sometimes it is strictly smaller: This is the case for all even $m \leq 10$, including $m = 4$, for which we gave a combinatorial proof. Thus, trying all possible values of R, rather than fixing it at α^k (as Berman et al. 2000 did) allowed us to obtain new bounds for even k. Additionally, it is interesting that for $m = 9$ and $m = 11$, the optimal R is strictly smaller than α^k, leading to improved bounds for odd m as well. Another anomaly is that for $m = 8$, the best bound matches the bound

for $m = 7$. Thus, it does not always help to add an active machine without an accompanying inactive machine.

Our program for finding these lower bounds as well as a spreadsheet summarizing its results can be found at http://iuuk.mff.cuni.cz/~sgall/ps/smallm/.

5 Conclusions

We have shown new lower bounds for online makespan scheduling on a small number of uniformly related machines. In contrast to the recent asymptotic bound from Ebenlendr and Sgall (2012), for small m we are able to take an advantage of the combinatorial structure of the problem and of the fact that even for the optimal online algorithm (pattern), the completion times of the active machines will be uneven. The gaps between the current lower and upper bounds are still significant and existing methods are limited to using geometric sequences of processing times for the lower bounds and doubling techniques for the algorithms. Finding new techniques seems necessary for further progress.

Acknowledgements Łukasz Jeż was partially supported by MNiSW grant N N206 368839, 2010–2013, a scholarship co-financed by an ESF project *Human Capital*, by the grant IAA100190902 of GA AV ČR, and by EUROGIGA project GraDR. Jarett Schwartz was partially supported by the National Science Foundation Graduate Research Fellowship and the Fulbright Fellowship. Jiří Sgall was supported by the Center of Excellence—Inst. for Theor. Comp. Sci., Prague (project P202/12/G061 of GA ČR). József Békési was partially supported by Stiftung Aktion Österreich-Ungarn, project No. 82öu9.

References

Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., & Waarts, O. (1997). On-line load balancing with applications to machine scheduling and virtual circuit routing. *Journal of the ACM*, *44*, 486–504.

Azar, Y. (1998). On-line load balancing. In A. Fiat & G. J. Woeginger (Eds.), *Online algorithms: the state of the art* (pp. 178–195). Berlin: Springer.

Bar-Noy, A., Freund, A., & Naor, J. (2000). New algorithms for related machines with temporary jobs. *Journal of Scheduling*, *3*, 259–272.

Berman, P., Charikar, M., & Karpinski, M. (2000). On-line load balancing for related machines. *Journal of Algorithms*, *35*, 108–121.

Cai, S. Y., & Yang, Q. F. (2012). Online scheduling on three uniform machines. *Discrete Applied Mathematics*, *160*(3), 291–302.

Chen, B., van Vliet, A., & Woeginger, G. J. (1994). New lower and upper bounds for on-line scheduling. *Operations Research Letters*, *16*, 221–230.

Cho, Y., & Sahni, S. (1980). Bounds for list schedules on uniform processors. *SIAM Journal on Computing*, *9*, 91–103.

Ebenlendr, T. (2011). *Combinatorial algorithms for online problems: semi-online scheduling on related machines*. Ph.D. Thesis, Charles University, Prague.

Ebenlendr, T., & Sgall, J. (2012). A lower bound on deterministic online algorithms for scheduling on related machines without preemption. In *Lecture notes in comput. sci.: Vol. 7164. Proc. 9th international workshop in approximation and online algorithms (WAOA 2011)* (pp. 102–108). Berlin: Springer.

Ebenlendr, T., Jawor, W., & Sgall, J. (2009). Preemptive online scheduling: optimal algorithms for all speeds. *Algorithmica*, *53*, 504–522.

Epstein, L., & Sgall, J. (2000). A lower bound for on-line scheduling on uniformly related machines. *Operations Research Letters*, *26*, 17–22.

Epstein, L., Noga, J., Seiden, S. S., Sgall, J., & Woeginger, G. J. (2001). Randomized on-line scheduling for two uniform machines. *Journal of Scheduling*, *4*, 71–92.

Faigle, U., Kern, W., & Turán, G. (1989). On the performance of online algorithms for partition problems. *Acta Cybernetica*, *9*, 107–119.

Fleischer, R., & Wahl, M. (2000). On-line scheduling revisited. *Journal of Scheduling*, *3*, 343–353.

Galambos, G., & Woeginger, G. J. (1993). An on-line scheduling heuristic with better worst case ratio than Graham's list scheduling. *SIAM Journal on Computing*, *22*, 349–355.

Graham, R. L. (1966). Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal*, *45*, 1563–1581.

Han, F., Tan, Z., & Yang, Y. (2011, to appear). On the optimality of list scheduling for online uniform machines scheduling. *Optimization Letters*. doi:10.1007/s11590-011-0335-x.

Musitelli, A., & Nicoletti, J. M. (2011). Competitive ratio of list scheduling on uniform machines and randomized heuristics. *Journal of Scheduling*, *14*, 89–101.

Rudin, J. F. III (2001). *Improved bound for the online scheduling problem*. Ph.D. Thesis, The University of Texas at Dallas.