A Lower Bound for Randomized On-Line Multiprocessor Scheduling

Jiří Sgall*

Abstract

We significantly improve the previous lower bounds on the performance of randomized algorithms for on-line scheduling jobs on m identical machines. We also show that a natural idea for constructing an algorithm with matching performance does not work.

Keywords

combinatorial problems, on-line algorithms, randomization, scheduling, worst case bounds.

1 Introduction

We study the model for scheduling introduced [7] and studied recently in [6, 1, 8]. This model is essentially a modified version of the game of Tetris. We have some fixed number of columns. Rectangles arrive one by one, each of them is one column wide and extends over one or more rows. We have to put each rectangle in one of the columns. The goal is to minimize the total number of rows that are at least partially used by the rectangles. In this scenario the columns represent the machines, rows represent the time steps and the rectangles represent the jobs with a running time corresponding to the height of a rectangle.

More precisely, we have m machines and a sequence of jobs arriving one by one; there are no dependencies between the jobs. When a job arrives, we know its running time, which is the same on all the machines (i.e., the machines are identical). We have to assign the job to one of the machines

*Mathematical Institute, AV ČR, Žitná 25, 115 67 Praha 1, Czech Republic, e-mail: `sgall@math.cas.cz`. Partially supported by grants A119107 and A1019602 of AV ČR; part of this work was done at Carnegie-Mellon University, Pittsburgh, PA, U.S.A.

immediately, without knowledge of the jobs that arrive later. The jobs cannot be preempted, i.e., once a job is scheduled (assigned to a machine), it cannot be moved to another machine. Our goal is to minimize the makespan, the time when the last job is completed. In the randomized model we measure the expected makespan. The performance of an on-line algorithm is measured by the competitive ratio—a randomized algorithm is σ-competitive (w.r.t. makespan) if for each input the expected makespan of the generated schedule is at most σ times larger than the optimal makespan.

In Section 2 we significantly improve the previously known lower bounds on the competitive ratio of randomized on-line algorithm for this problem. This result was reported in [10]. It was also proved independently by Chen, van Vliet and Woeginger [3], however our proof is more explicit and therefore it gives more insight for a possible construction of an improved algorithm.

This insight leads to a natural invariant that should be preserved by any algorithm matching our lower bound. An algorithm based on this invariant would be a natural generalization of the optimal algorithm for two processors from [1]; it would also follow the suggestion from [4]. In Section 3 we demonstrate that it is impossible to preserve this invariant inductively. Therefore for a design of such an algorithm we would have to use a stronger condition; at the present time we do not know if this is possible.

Let us survey the previous related results. While the deterministic case has been studied extensively [7, 6, 1, 8], much less was known about the randomized case; all of the following results are by Bartal, Fiat, Karloff, and Vohra [1]. Only the case of $m = 2$ is solved completely; an optimal 4/3-competitive algorithm is known and provably better than any deterministic algorithm. For $m = 3$ a nontrivial lower bound of 1.4 was proved. For $m > 3$, the best known lower bound was the easy 4/3 bound, not even matching the bound for $m = 3$. For small values of $m \geq 3$ randomized algorithm with a better competitive ratio than the best deterministic algorithm were discovered very recently by Seiden [9]. For large m no randomized algorithm with a better competitive ratio than the best deterministic algorithm is known, which means that we do not know how to make use of randomization at all.

For a long time the best deterministic algorithm was the $(2 - \frac{1}{m})$-competitive Graham's algorithm [7]. For $m = 2$ and $m = 3$, Graham's algorithm is provably optimal. For larger m, it was improved several times during the last few years [6, 1, 8]. For sufficiently large m the best known algorithm is 1.945-competitive [8]. For $m > 3$, an algorithm slightly better than Graham's is presented in [6]. The best lower bound on the competitive ratio of the deterministic scheduling algorithm for large m is approximately 1.837 [2]; an earlier bound of $1 + 1/\sqrt{2} \approx 1.707$ is valid for any $m > 3$ [5].

Interestingly, for a related model, deterministic preemptive on-line scheduling on m identical machines, Chen, van Vliet and Woeginger [4] not only proved the same lower bound as we prove for randomized nonpreemptive algorithms, but also gave an algorithm matching this bound.

2 An improved lower bound

We prove that the competitive ratio of any randomized on-line scheduling algorithm for m machines is at least $1+1/((\frac{m}{m-1})^m - 1)$. This matches the known tight bound of $4/3$ for $m = 2$ and improves the previous bounds for all $m > 2$. If m is large, this bound approaches $1 + 1/(e - 1) \approx 1.582$. Table 1 compares the bounds for a few values.

number of machines	our bound	previous lower bound	upper bound
2	$4/3 \approx 1.333$	$4/3$	$4/3$
3	$27/19 \approx 1.421$	1.4	1.559
4	$256/175 \approx 1.463$	$4/3$	1.675
5	$3125/2101 \approx 1.487$	$4/3$	1.741
6	$46656/31031 \approx 1.504$	$4/3$	1.788
∞	$1 + 1/(e - 1) \approx 1.582$	$4/3$	1.945

Table 1: *Known bounds on the competitive ratio for randomized on-line scheduling of sequential jobs.*

The intuition for our lower bound can be better understood if we look at the algorithm and the lower bound for two machines from [1]. Their algorithm guarantees that the ratio of the expected loads on the two machines is 2 : 1 most of the time (unless there is a job with a very long running time), where the load of a machine is defined as the total running time of all jobs assigned to the machine, also called height in [1]. Their lower bound essentially shows that any optimal algorithm has to maintain this ratio of expected loads, and that it is not possible to maintain a better ratio. The optimal algorithm can balance the loads exactly, hence the competitive ratio is $2/1.5 = 4/3$.

In the case of m machines we show that the best ratio of loads which might be possible to maintain is $1 : x : x^2 : \ldots : x^{m-1}$, where $x = m/(m-1)$. The optimal schedule can balance the loads to be $(1 + x + \cdots + x^{m-1})/m = (x^m - 1)/m(x - 1)$, hence the competitive ratio is at least $mx^{m-1}(x - 1)/(x^m - 1)$, which gives our bound. See Figures 1 and 2 for an illustration.

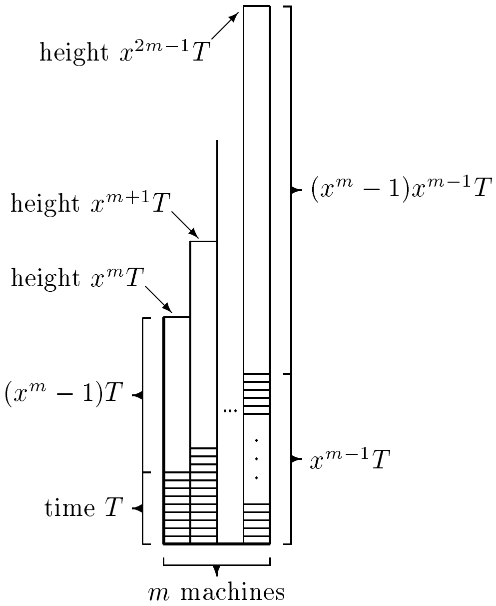

Figure 1: *An optimal on-line schedule for the sequence of jobs used for the lower bound.*

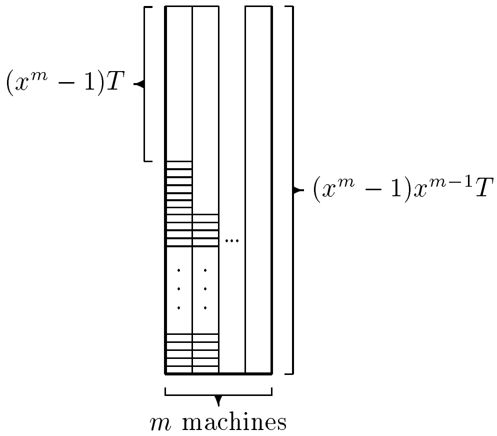

Figure 2: *An optimal off-line schedule for the sequence of jobs used for the lower bound.*

We first prove a general lemma which applies to any sequence of jobs. For our proof the last m jobs of the sequence are most important: the m different input sequences we use for lower bound are the initial segments of the sequence with fewer than m jobs from the end deleted. In fact in the particular sequence that we use later, the only purpose of the other jobs is to pad the sequence so that the optimal schedule can always balance the loads exactly.

Let a sequence of jobs $\mathcal{J}$ be given. Denote the last m jobs of $\mathcal{J}$ by $J_1, \ldots, J_m$ and their running times by $t_1, \ldots, t_m$. Let $\mathcal{J}_i$ be an initial segment of $\mathcal{J}$ ending by J_i (i.e., $\mathcal{J}_m = \mathcal{J}$). Let S_i be the sum of the running times of all jobs in $\mathcal{J}_i$ and let $T_{\text{opt}}(\mathcal{J}_i)$ be the length of an optimal schedule for $\mathcal{J}_i$. For a given randomized algorithm A, $\mathrm{E}[T_A(\mathcal{J}_i)]$ denotes the expected length of the schedule it generates on $\mathcal{J}_i$.

The definition of the competitive ratio σ implies that for any choice of the sequences of jobs $\mathcal{J}_i$ we have

$$\frac{\sum_{i=1}^m \mathrm{E}[T_A(\mathcal{J}_i)]}{\sum_{i=1}^m T_{\mathrm{opt}}(\mathcal{J}_i)} \leq \frac{\sum_{i=1}^m \sigma T_{\mathrm{opt}}(\mathcal{J}_i)}{\sum_{i=1}^m T_{\mathrm{opt}}(\mathcal{J}_i)} = \sigma.$$

The following lemma gives an upper bound on the expectation in the left-hand side.

Lemma 2.1 *For any randomized on-line algorithm A for scheduling on m machines we have $\sum_{i=1}^m \mathrm{E}[T_A(\mathcal{J}_i)] \geq S_m$.*

Proof. Fix a sequence of random bits used by the algorithm A. Let T_i be the makespan of the schedule generated by the algorithm A for the jobs in $\mathcal{J}_i$ with the fixed random bits. Since the algorithm is on-line, the schedule for $\mathcal{J}_i$ is obtained from the schedule for $\mathcal{J}_{i-1}$ by adding J_i to one of the machines.

Order the machines so that the load of ith machine does not change after J_i is scheduled. There always exists such an ordering: Designate a machine to be the ith one, if J_i is the last job scheduled on it. Clearly at most one machine is designated as the ith one. If no machine is designated as the jth one for some j, pick one of the remaining machines arbitrarily; note that no job J_i is scheduled on these machines. This defines an ordering satisfying the condition.

Let L_i be the load of ith machine after scheduling all jobs. Obviously $\sum_{i=1}^m L_i = S_m$. The load of ith machine is L_i already after scheduling $\mathcal{J}_i$ because of our condition on the order of the machines. Therefore $T_i \geq L_i$ and $\sum_{i=1}^m T_i \geq \sum_{i=1}^m L_i = S_m$ for any choice of the random bits.

By the linearity of expectation we have

$$\sum_{i=1}^m \mathrm{E}[T_A(\mathcal{J}_i)] \geq \mathrm{E}[\sum_{i=1}^m T_A(\mathcal{J}_i)] = \mathrm{E}[\sum_{i=1}^m T_i] \geq S_m$$

□

Let $x = m/(m-1)$. We choose our sequence of jobs so that the following property is satisfied. If all the jobs preceding J_i are scheduled so that the ratio of loads is $1 : x : \ldots : x^{m-1}$, then after scheduling J_i on the least loaded machine the ratio of the loads is preserved.

Let $T = (m-1)^{2m-1}$. The sequence $\mathcal{J}$ consists of $(1 + \cdots + x^{m-1})T$ jobs of running time 1 followed by m jobs with running times $t_i = (x^m - 1)x^{i-1}T$. The value of T is chosen so that all running times are integers. Optimal on-line and off-line schedules are illustrated on Figures 1 and 2.

After scheduling the jobs preceding J_i so that the ratio is as described above, the loads of the machines are $x^{i-1}T, \ldots, x^{i+m-2}T$. For $i = 1$ this is obvious. It is maintained inductively, since by our choice of running times we have $x^{i-1}T + t_i = x^{i+m-1}T$, which is exactly the condition we need to preserve the ratio of the loads. The next theorem says that no on-line algorithm can do better, even if it is randomized.

Theorem 2.2 *For any randomized on-line scheduling algorithm for m machines, the competitive ratio σ_m is at least $1 + 1/((\frac{m}{m-1})^m - 1)$.*

Proof. First we prove that $T_{\mathrm{opt}}(\mathcal{J}_i) = t_i$. The total running time of all jobs in $\mathcal{J}_i$ is

$$\begin{aligned} S_i &= (1 + \cdots + x^{m-1})T + (x^m - 1)(1 + \cdots + x^{i-1})T \\ &= (x^i + \cdots + x^{m-1})T + x^m(1 + \cdots + x^{i-1})T \\ &= x^i(1 + \cdots + x^{m-1})T \\ &= x^i \frac{x^m - 1}{x - 1} T = \frac{x}{x-1} t_i = m t_i. \end{aligned}$$

Since running times of all jobs are integers, at most m of them are greater than 1 and t_i is the maximal running time, the loads of the m machines can be balanced perfectly and $T_{\mathrm{opt}}(\mathcal{J}_i) = t_i$. (More explicitely, the schedule will use distinct processors for all jobs with running time greater than 1, and distribute the jobs with running time 1 so that the load of each processor is exactly t_i.)

Using Lemma 2.1 and $S_m = x^m(1+\cdots+x^{m-1})T$ from the previous computation, the competitive ratio σ_m for any randomized on-line algorithm A is bounded by

$$\begin{aligned} \sigma_m &\geq \frac{\sum_{i=1}^m \mathrm{E}[T_A(\mathcal{J}_i)]}{\sum_{i=1}^m T_{\mathrm{opt}}(\mathcal{J}_i)} \geq \frac{S_m}{\sum_{i=1}^m t_i} = \\ &= \frac{x^m(1 + \cdots + x^{m-1})T}{(1 + \cdots + x^{m-1})(x^m - 1)T} = \frac{x^m}{x^m - 1} = 1 + \frac{1}{(\frac{m}{m-1})^m - 1}. \end{aligned}$$

□

3 A note on a possible matching upper bound

The lower bound from the previous section suggests a natural generalization of the approach used in [1] in the optimal algorithm for two processors.

After each sequence of jobs we have a certain probability distribution on the possible configurations of the machines, where the configuration consists of the loads of the individual machines. We assume that in each configuration the machines are ordered by their load, starting with the smallest one. Now we take the average (expected) load of each machine. In [1] it is proved for two processors that the ratio of the average loads can be maintained inductively at $1:2$, as long as there is no large job. That means that if the ratio is $1:2$ and a job arrives, it can be randomly assigned to one of the machines so that the ratio remains unchanged.

A natural generalization, also suggested in [3], is to maintain the ratio of the average loads at $1:x:x^2:\cdots:x^{m-1}$ where $x=m/(m-1)$ and m is the number of processors, as long as the jobs are small. In fact, the lower bound shows that this has to be the case at least at all times when the jobs in the sequence together with the jobs used in the lower bound can be balanced exactly on m processors.

Extending the definitions from [1] we say that a job is *critical*, if its running time is $1/(m-1)$ times the sum of the previous running times; any job with longer running time is a *large* job. Thus, if a large job arrives, we know that the optimal schedule cannot balance the jobs exactly, and the optimal length of the schedule is at least its running time. For the rest of the paper we consider only sequences with no large jobs (except the first few ones); large jobs have to be handled separately from the above invariant, similarly as in [1].

We now demonstrate that the invariant ratio above cannot be maintained inductively even for sequences with no large jobs. The lower bound proof in Section 2 also implies that when a critical job arrives then in any configuration with non-zero probability the difference of the loads of the first and the last machine must be at most the running time of the critical job: if the difference between the loads is larger, the average load of the most loaded machine increases too much, and the invariant is not preserved.

Now it is easy to construct a concrete counterexample for three machines. Consider a sequence of $2\cdot 9\cdot 19=342$ jobs of running time 1 and 1 job of running time $9\cdot 19=171$; note that the last job is critical. Assume that after scheduling all jobs with running time 1 the loads of the machines are $(2 \cdot 38, 3 \cdot 38, 4 \cdot 38)$ with probability $18/19$ and $(0, 0, 9 \cdot 38)$ with probability $1/19$. The average loads are $(4 \cdot 18, 6 \cdot 18, 9 \cdot 18)$, and thus this distribution satisfies the invariant. After the last job with running time $9 \cdot 19$ is scheduled, the average loads should be $(4 \cdot 27, 6 \cdot 27, 9 \cdot 27)$ to preserve the invariant. However, after scheduling the last job in the first configuration, the largest load will be at least $2 \cdot 38 + 9 \cdot 19 = 13 \cdot 19$ and in the second configuration it will be at least $9 \cdot 38$. Thus the average is at least $\frac{18}{19} \cdot 13 \cdot 19 + \frac{1}{19} \cdot 9 \cdot 38 = 9 \cdot 28 > 9 \cdot 27$ and the invariant cannot be preserved; moreover, the schedule for this particular instance does not have the desired competitive ratio. To make the example more convincing we shall show that the distribution above can be achieved by a sequence where the expected loads satisfy the invariant at each step, except the first few ones. After scheduling 1, 2, 4, or 5 jobs of running time 1, no distribution satisfies the invariant.[1] A straightforward but somewhat tedious calculation gives a legal sequence of distributions on configurations for a sequence of jobs of running time 1 where the invariant is preserved at all other steps, after i jobs are scheduled the probability of the configuration $(0, 0, i)$ is always $1/19$, and after $9i$ jobs (i integer) the probability of the configuration $(2i, 3i, 4i)$ is $18/19$; this includes our desired configuration.

We conclude that to construct an algortihm matching our lower bound, we would have to use a more refined construction, perhaps with a stronger invariant which would account for the possibility above. We still conjecture that an algorithm matching the lower bound from Section 2 exists, however, the most intuitive approach does not work.

References

[1] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. *J. Comput. Syst. Sci.*, 51(3):359–366, 1995.

[1] For 1 and 2 this is obvious, for 4 and 5 jobs an easy computation is required. (In contrast, for two machines after the first job the desired invariant can be maintained at all times.) The fact that for 4 or 5 jobs the invariant is not preserved has no direct implication, since due to the small number of jobs, the optimal solution cannot balance them exactly as is needed in the lower bound proof. Thus it would be reasonable to assume that the algorithm can treat a first few jobs in a special way. For that reason we need to present the above example, where the invariant cannot be preserved for more substantial reasons.

[2] Y. Bartal, H. Karloff, and Y. Rabani. A new lower bound for m-machine scheduling. *Inf. Process. Lett.*, 50:113–116, 1994.

[3] B. Chen, A. van Vliet, and G. J. Woeginger. A lower bound for randomized on-line scheduling algorithms. *Inf. Process. Lett.*, 51:219–222, 1994.

[4] B. Chen, A. van Vliet, and G. J. Woeginger. An optimal algorithm for preemptive on-line scheduling. *Oper. Res. Lett.*, 18:127–131, 1995.

[5] U. Faigle, W. Kern, and G. Turan. On the performance of on-line algorithms for partition problems. *Acta Cybernetica*, 9:107–119, 1989.

[6] G. Galambos and G. J. Woeginger. An on-line scheduling heuristic with better worst case ratio than Graham's list scheduling. *SIAM J. Comput.*, 22(2):349–355, 1993.

[7] R. L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical J.*, 45:1563–1581, Nov. 1966.

[8] D. R. Karger, S. J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. In *Proc. of the 5th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 132–140. ACM-SIAM, 1994.

[9] S. Seiden. A randomized algorithm for that ancient scheduling problem. Manuscript, 1996.

[10] J. Sgall. *On-Line Scheduling on Parallel Machines.* PhD thesis, Technical Report CMU-CS-94-144, Carnegie-Mellon University, Pittsburgh, PA, U.S.A., 1994.