A Better Lower Bound for On-Line Scheduling

Yair Bartal * Howard Karloff † Yuval Rabani ‡

Abstract

We consider the on-line version of the original m-machine scheduling problem: given m machines and n positive real jobs, schedule the n jobs on the m machines so as to minimize the makespan, the completion time of the last job. In the on-line version, as soon as a job arrives, it must be assigned immediately to one of the m machines.

We study the competitive ratio of the best algorithm for m-machine scheduling. The largest prior lower bound was that if $m \geq 4$, then every algorithm has a competitive ratio at least $1+1/\sqrt{2} \approx 1.707$. We show that if m is large enough, the competitive ratio of every algorithm exceeds 1.837. The best upper bound on the competitive ratio is now 1.945.

1 Introduction

The m-machine scheduling problem is one of the most widely-studied problems in computer science ([4, 6, 7, 8] are surveys), with an almost limitless number of variants. In this note, we study one of the simplest and earliest m-machine scheduling problems ever studied, the scheduling problem of Graham, introduced in 1966 [3]. This is the variant in which each job consists of exactly one task, which requires the same execution time on each of the m machines. Jobs cannot be preempted and are independent of each other. The goal is to minimize the *makespan*, the completion time of the last job. Formally, the problem is this: Given a sequence of positive reals $a_1, a_2, ..., a_n$ and an integer m, for each j assign a_j to a machine i, $1 \leq i \leq m$, so as to minimize the maximum, over i, of the sum of all reals assigned to machine i. Even the special case of $m = 2$ is NP-Hard, as it is at least as hard as PARTITION.

We study the on-line version of the problem: at the time when job j is scheduled, the scheduling algorithm knows only $a_1, a_2, ..., a_j$. As soon as a_j appears, the scheduling algorithm,

*Computer Science Department, School of Mathematics, Tel-Aviv University, Tel-Aviv 69978, Israel.

†College of Computing, Georgia Institute of Technology, Atlanta, Georgia 30332-0280. The author was supported in part by NSF grant CCR 9107349.

‡International Computer Science Institute, 1947 Center Street, Berkeley, CA 94704-1105. Part of this work was done while the author was a student at Tel-Aviv University. The work at ICSI was supported in part by a Rothschild postdoctoral fellowship.

with only the knowledge of $a_1, a_2, ..., a_j$, must irrevocably assign job j to one of the machines.

For a sequence σ of jobs, let $A(\sigma)$ denote the makespan of the schedule generated by algorithm A, and let $OPT(\sigma)$ denote the minimum makespan among all m-machine schedules for σ. A's *competitive ratio* is then

$$c_A := \sup_\sigma \frac{E[A(\sigma)]}{OPT(\sigma)},$$

where the supremum is over all nonempty sequences of jobs.

Graham showed in [3] that the naive List algorithm—always place the next job on the most lightly loaded machine—has competitive ratio $2-1/m$ for each m. Until recently, no algorithm was known whose performance ratio, as a function of m, was bounded above by $2-\epsilon$ for all m (for a fixed positive ϵ). Bartal, Fiat, Karloff and Vohra recently exhibited an on-line algorithm with competitive ratio at most $2 - 1/70$ for all m [1]. Currently the best algorithm for large m is due to Karger, Phillips and Torng [5], and has competitive ratio bounded by 1.945.

But how *small* can c_A be? Prior to this note, the best known lower bounds were due to Faigle, Kern and Turan [2], who proved in 1989 that no on-line algorithm could have a competitive ratio that is smaller than 3/2 if $m = 2$ or smaller than 5/3 if $m = 3$, and that for no $m \geq 4$ could the competitive ratio be less than $1 + \frac{1}{\sqrt{2}} = 1.707\ldots$. This last fact was proven by examining the job sequence consisting of m jobs of size 1, m jobs of size $1+\sqrt{2}$, and then one job of size $2+2\sqrt{2}$.

We improve the lower bound due to Faigle, Kern and Turan to $\alpha \approx 1.837$. The proof is quite simple: we give the algorithm one sequence of judiciously-chosen jobs, and then argue that on at least one nonempty prefix of the sequence, the makespan obtained by algorithm A divided by the optimal makespan for those jobs is at least α. Since the algorithm is on-line, its schedule for jobs $\sigma_1, \sigma_2, ..., \sigma_i$ is obtained by adding job σ_i to its schedule for jobs $\sigma_1, \sigma_2, ..., \sigma_{i-1}$.

2 The Construction

We will give the on-line algorithm a sequence of jobs, whose sizes depend on the positive reals x, y, z, whose values we will choose later. Rather than give all the constraints that must be satisfied by x, y, and z now, we give them "on-the-fly" when they are needed. These parameters are chosen so that any algorithm that has competitive ratio less than α must schedule the jobs in a unique way. In fact, it must schedule each new job on the lowest machine available.

The job sequence is as follows.

1. m jobs of size $1/(x+1)$.

2. m jobs of size $x/(x+1)$.

3. m jobs of size x.

4. $\lfloor m/2 \rfloor$ jobs of size y.

5. $\lfloor m/3 \rfloor - 2$ jobs of size z.

6. $m + 3 - \lfloor m/2 \rfloor - \lfloor m/3 \rfloor$ jobs of size $2y$.

Now we study the performance of deterministic on-line algorithm A. We assume, for a contradiction, that its competitive ratio is less than α (where α is as yet undetermined). Let us use On_i and Opt_i to denote the on-line and optimal makespan, respectively, for the job sequence consisting of blocks $1, 2, ..., i$, $1 \leq i \leq 6$.

If the m jobs of size $1/(x+1)$ are not placed on separate machines, then $On_1 \geq 2/(x+1)$. Clearly $Opt_1 = 1/(x+1)$, so that in this case the competitive ratio is at least α if $\alpha \leq 2$. So we stipulate that

$$\alpha \leq 2, \tag{1}$$

thereby forcing A to place the m $1/(x+1)$'s on separate machines.

Now if the m $x/(x+1)$'s aren't on separate machines, then $On_2 \geq 2x/(x+1) + 1/(x+1)$ $= (2x+1)/(x+1)$, and clearly $Opt_2 = 1$. So let us stipulate that

$$\frac{2x+1}{x+1} \geq \alpha, \tag{2}$$

to force $On_2 = 1$.

We do the same with the x's. Equation (2) fortuitously forces A to place the x's on separate machines. This means that $On_3 = 1 + x$; clearly $Opt_3 = 1 + x$ also.

Now we have to force A to place the $\lfloor m/2 \rfloor$ y's on separate machines. What is Opt_4? After the y's arrive, we can place one $1/(x+1)$ and one $x/(x+1)$ on each machine, for a total height of 1. Next, we can place the x's on $\lceil m/2 \rceil$ machines, at most two x's per machine, so that those machines have height $1 + 2x$. On the remaining $m - \lceil m/2 \rceil = \lfloor m/2 \rfloor$ machines, we place one y per machine. This means that $Opt_4 \leq \max\{2x+1, y+1\}$. We will want to use $Opt_4 \leq 2x + 1$, so we add the condition

$$y \leq 2x. \tag{3}$$

If A places any two of the y's atop one another, then $On_4 \geq 2y + x + 1$. However, $Opt_4 \leq 2x + 1$. By stipulating that

$$\frac{2y+x+1}{2x+1} \geq \alpha, \tag{4}$$

we force A to place the y's on separate machines.

Now the z's, of which we have $\lfloor m/3 \rfloor - 2$. We want A to place them alongside each other, atop the x's. If it doesn't, then its makespan will be at least $1 + x + y + z$. We will ensure that

$Opt_5 \leq 3x$. Once this is done, we will stipulate that

$$\frac{z+y+x+1}{3x} \geq \alpha, \tag{5}$$

to force A to place the z's alongside each other, atop the x's.

One way to pack the items in blocks $1,2,3,4,5$ is as follows. $\lceil m/6 \rceil$ machines have three x's each. $\lceil m/2 \rceil$ machines have a y, an x, an $x/(x+1)$, and two $1/(x+1)$'s. $\lfloor m/3 \rfloor - 2$ machines have a z and two $x/(x+1)$'s. The reader can verify that no more than m machines have been used, and that all jobs in blocks 1-5 have been scheduled, provided that $m \geq 32$. The makespan of this schedule is $\max\{z+2x/(x+1), y+x+x/(x+1)+2/(x+1), 3x\}$. We want this to be $3x$. So we stipulate that

$$z + 2\frac{x}{x+1} \leq 3x \tag{6}$$

and

$$y + x + 1 + \frac{1}{x+1} \leq 3x. \tag{7}$$

Now, the last block, the $(2y)$'s. Notice that the number of y's, z's and $(2y)$'s, summed up, is $m+1$. This means that A must place at least one of the $(2y)$'s above either a y or a z. In any case, $On_6 \geq 3y + x + 1$, if we stipulate that

$$y \leq z. \tag{8}$$

We will add enough constraints to ensure that $Opt_6 = 2y$. So we will have a contradiction if we stipulate that

$$\frac{3y+x+1}{2y} \geq \alpha. \tag{9}$$

Here is a partial schedule at the end: There are $m + 3 - \lfloor m/2 \rfloor - \lfloor m/3 \rfloor$ $(2y)$'s, which go on separate machines. The $\lfloor m/2 \rfloor$ y's go on $\lceil m/4 \rceil$ machines, with up to two y's apiece. $\lceil m/3 \rceil$ machines have one z and one x. $\lceil 2m/9 \rceil$ machines have three x's and three $x/(x+1)$'s apiece.

On the remaining machines (which number at least $-8 + m/36$), we must schedule m $1/(x+1)$'s and $m - 3\lceil \frac{2m}{9} \rceil$ jobs of size $x/(x+1)$. On $\lceil m/63 \rceil$ machines, we place 21 $x/(x+1)$'s apiece. On $\lceil m/111 \rceil$ machines we place 111 $1/(x+1)$'s. This is a valid schedule, provided that we have $\lceil m/63 \rceil + \lceil m/111 \rceil \leq -8 + m/36$. This is certainly true if $m \geq 3454$.

The makespan of this schedule is

$$\max\{2y, z+x, 3x + 3\frac{x}{x+1}, 21\frac{x}{x+1}, 111\frac{1}{x+1}\}.$$

Since we want the makespan to be $2y$, we stipulate that

$$z + x \leq 2y, \tag{10}$$

$$3x + 3\frac{x}{x+1} \leq 2y, \tag{11}$$

$$21\frac{x}{x+1} \leq 2y, \tag{12}$$

$$111\frac{1}{x+1} \leq 2y. \tag{13}$$

What remains is to find x, y, z, α satisfying all the inequalities above, with α as large as possible. We will arbitrarily choose a set of inequalities to make tight, solve them, and show numerically that the remaining constraints are satisfied. We choose inequalities (2), (5), (9), and (10) to be satisfied with equality; the others will not be. From equations (2), (5), and (10), we get

$$\frac{2x+1}{x+1} = \alpha = \frac{3y+1}{3x},$$

which yields

$$y = \frac{1}{3}\left(\frac{6x^2+2x-1}{x+1}\right). \tag{14}$$

From equations (5), (9), and (10) we get

$$\frac{3y+1}{3x} = \frac{3y+x+1}{2y}. \tag{15}$$

Eliminating y from equations (14) and (15) we get

$$3x^3 - 13x^2 - 12x - 2 = 0.$$

It is not hard to verify that there are roots in the intervals $(-0.58023953020, -0.58023953019)$, $(-0.2236520327, -0.2236520326)$, and $(5.137224896, 5.137224897)$. We choose x to be the root in the final interval. In fact, the exact value for x is

$$\frac{13}{9} + \frac{2}{9}\sqrt{277}\cos\left(\frac{1}{3}\arccos\frac{4546}{277\sqrt{277}}\right).$$

This yields $y \approx 9.104056552$ and $z \approx 13.070888209$. This gives us $\alpha = (2x+1)/(x+1) = 2 - 1/(x+1) \in (1.8370599062, 1.8370599063)$. It is now easy to verify numerically that the remaining constraints are all strictly satisfied. This gives us a lower bound exceeding 1.837.

3 Acknowledgments

We are grateful to Steven Phillips and Eric Torng for improving our earlier results.

References

[1] Y. Bartal, A. Fiat, H. Karloff and R. Vohra, "New Algorithms for an Ancient Scheduling Problem," *Proc. 24th ACM Symposium on the Theory of Computing*, Victoria, Canada, 1992, 51-58.

[2] U. Faigle, W. Kern and György Turán, "On the Performance of On-Line Algorithms for Partition Problems," *Acta Cybernetica* 9 (1989), 107-119.

[3] R. L. Graham, "Bounds for Certain Multiprocessing Anomalies," *Bell System Technical Journal* 45 (1966), 1563-1581.

[4] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey," *Annals of Discrete Mathematics* 5 (1979), 287-326.

[5] D. Karger, S. Phillips, and E. Torng, "A Better Algorithm for an Ancient Scheduling Problem," manuscript, Stanford University.

[6] E. L. Lawler, "Recent Results in the Theory of Machine Scheduling," in A. Bachem, M. Grotschel, and B. Korte (eds.), *Math Programming: The State of the Art (Bonn 1982)*, Springer-Verlag, New York, 1983, 202-234.

[7] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, "Sequencing and Scheduling: Algorithms and Complexity," to appear in *Handbook of Operations Research and Management Science, Volume IV: Production Planning and Inventory*, S. C. Graves, A. H. G. Rinnooy Kan, and P. Zipkin (eds.), North-Holland.

[8] J. K. Lenstra and A. H. G. Rinnooy Kan, "An Introduction to Multiprocessor Scheduling," Technical Report, CWI, Amsterdam, 1988.