New Algorithms for an Ancient Scheduling Problem

Yair Bartal * Amos Fiat * Howard Karloff † Rakesh Vohra ‡

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 job j arrives, it must be assigned immediately to one of the m machines.

We present two main results. The first is a $(2-\epsilon)$-competitive deterministic algorithm for all m. The competitive ratio of all previous algorithms approaches 2 as $m \to \infty$. Indeed, the problem of improving the competitive ratio for large m had been open since 1966, when the first algorithm for this problem appeared.

The second result is an optimal randomized algorithm for the case $m = 2$. To the best of our knowledge, our 4/3-competitive algorithm is the first specifically randomized algorithm for the original, m-machine, on-line scheduling problem.

Introduction

The m-machine scheduling problem is one of the most widely-studied problems in computer science ([4, 5, 6, 7] are surveys), with an almost limitless number of variants. For example, there can be precedence constraints between jobs; the jobs can run at different

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

†Department of Computer Science, University of Chicago. Earlier drafts of this paper were written while the author visited DIMACS. This work was supported in part by NSF grant CCR 9107349.

‡Department of Management Science, Ohio State University. This work was supported in part by a Dean's Research Professorship of the College of Business of the Ohio State University.

speeds on different machines; each job can consist of one or many tasks; a job can become available only at a "release time"; a job may be "preemptable," so that it can be stopped and restarted, or nonpreemptable; the processing time of a job may be known in advance or known only when the job eventually terminates; and the goal can be to minimize the completion time of the last job, or the average (weighted or unweighted) completion time.

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 are nonpreemptable and 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 $a_1, a_2, ..., a_j$. If the scheduler can flip coins, the algorithm is *randomized*. As soon as a_j appears, the scheduling algorithm, with only the knowledge of $a_1, a_2, ..., a_j$, must irrevocably assign job j to one of the machines. (The list of jobs is chosen in advance, that is, the choice of job j cannot depend on how the randomized algorithm scheduled jobs $1, 2, ..., j-1$.)

For a sequence σ of jobs, let $A(\sigma)$, a random variable, denote the makespan of the schedule generated by (randomized or deterministic) 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 sequences of jobs.

How small can c_A be? In his 1966 paper [3], Graham proposed a simple heuristic now known as the List Processing Algorithm, or just List, for the on-line scheduling problem. When job j arrives, List sums the sequence of jobs assigned already to machine i, for each i, and assigns job j to the machine with least total processing time to date, breaking ties arbitrarily. Graham gave a very simple proof that $c_{List} = 2 - \frac{1}{m}$ when the number of machines is m. (Graham's 1966 analysis is one of the first worst-case analyses of a heuristic.) In 1989, Faigle, Kern and Turan [1] proved that no deterministic on-line

algorithm could have a smaller competitive ratio than $2 - \frac{1}{m}$ for $m = 2$ and 3, and that for no $m \geq 4$ could the competitive ratio be less than $1 + \frac{1}{\sqrt{2}} = 1.707\ldots$. Thus List is optimal for $m = 2$ and 3. Only in 1991 did improvements over List appear [2, 9]. These algorithms are $(2 - \frac{1}{m} - \epsilon_m)$-competitive for all m, where $\epsilon_m > 0$ for all $m \geq 4$. The drawback is that $\epsilon_m \to 0$ as $m \to \infty$, so that the improvement is negligible for large m.

This paper focuses on large values of m, as well as on the smallest nontrivial value, 2. We exhibit an $\epsilon > 0$ and a $(2 - \epsilon)$-competitive, deterministic, m-machine scheduling algorithm. The problem of improving the competitive ratio for m-machine scheduling for large m had been a longstanding open problem. Indeed, it is sufficiently difficult to find a $(2 - \epsilon)$-competitive algorithm for all m—the problem had been open since 1966, when Graham's original paper appeared—that it was plausible to conjecture that the limiting competitive ratio of any deterministic algorithm is at least 2.

We also initiate the study of randomization in the traditional, m-machine, on-line scheduling problem. We exhibit a randomized 2-machine scheduling algorithm which is 4/3-competitive, and prove it optimal. To the best of our knowledge, this is the first use of randomization for this problem.

A New m-Machine Algorithm

The intuition for our algorithm seems to be well known to fans of the Russian computer game Tetris [8]. The m-machine scheduling problem is a rather simplified version of the game. The idea is to try to prevent a situation where the load on all processors is approximately equal, if you can do so while keeping the makespan within the range permitted by the target competitive ratio. Graham's List algorithm simply tries to balance the load.

Consider a situation where the load is approximately balanced yet some processor p is loaded with many small jobs. A preferable configuration would be to have some processor with a relatively low load. With hindsight, this could have been attained by assigning the small jobs now scheduled on p to other processors. This alternate configuration is better because if a big job now arrives, it can be scheduled on the processor with low load, increasing the makespan far less than if it had been scheduled on a processor with close to average load.

Our m-machine algorithm tries to keep approximately 44.5% of the machines lightly loaded. If these machines have too small a load then new small jobs will be assigned

to them. If we've attained our target, then small new jobs will be divided among the processors so as to preserve the goal. Problems will arise when a big job arrives, since we do not want to assign it to a lightly loaded processor, as this would contradict our stated goal, nor do we wish to increase the makespan by assigning it to a heavily loaded processor. In this case, we have no choice but to assign the new job to a lightly loaded processor.

We define our algorithm for $m \geq m_0$ machines, m_0 to be defined later; the algorithm is $(2-\epsilon)$-competitive, where $\epsilon = 1/m_0$. Our algorithm also requires a parameter $\delta \in (0,1)$ (dependent on m), where δm must be integral. Further restrictions on ϵ and δ are given later.

We need the following definitions both to define and to analyze the algorithm.

Definitions. Define the *height* of a machine to be the sum of the lengths of the jobs already assigned to it. At all times, the algorithm will maintain a list of the machines sorted into nondecreasing order by current height. After exactly i jobs have been scheduled, define F_i to be both the sequence of the first δm machines on the list, and the sequence of their heights, abusing the notation slightly. L_i denotes the sequence of the last $m - \delta m$ machines, and their heights. A_i and M_i denote the average and minimum height (over all m machines), respectively, when i jobs have been scheduled. Where P is a sequence of heights, $A(P)$ and $M(P)$ denote the average and minimum, respectively, of the heights in P.

The m-Machine Algorithm. When job $i+1$ arrives, place a_{i+1} on the first machine in L_i, if

$$M(L_i) + a_{i+1} \leq (2-\epsilon)A(F_i);$$

otherwise, place a_{i+1} on the first machine on the list, the one with least height overall, breaking any ties arbitrarily. If necessary, permute the list of machines so that height remains nondecreasing.

We will prove

Theorem 1 *The algorithm is $(2-\epsilon)$-competitive.*

The analysis of our m-machine scheduling algorithm needs parameters τ, β_1, β_2, and α, all in $(0,1)$, which must satisfy certain conditions involving ϵ and δ for the proof to

go through. The parameter ϵ is a constant that does not depend on m. Parameters δ and τ do depend upon m, but only "slightly"; they vary only enough to ensure that δm and $\tau\delta m$ remain integral. The other parameters are completely determined by ϵ, δ and τ:

$$\begin{aligned}
\beta_1 &= [2-\epsilon+\frac{\delta}{1-\delta}](1-\tau)(1-\epsilon)-\frac{1}{1-\delta};\\
\beta_2 &= [2-\epsilon+\frac{\delta}{1-\delta}]\tau;\\
\alpha &= \frac{1-\beta_1}{1+\beta_2}+\epsilon\frac{1-\tau\delta}{\tau\delta(1+\beta_2)}.
\end{aligned}$$

We impose the conditions:

$$\beta_1 \geq \frac{1}{2(1-\epsilon)}; \tag{1}$$

$$\alpha \leq 1-\epsilon-\frac{1}{2(1-\epsilon)}. \tag{2}$$

One can verify that for $m_0 = 70$, $\epsilon = 1/m_0$, all $m \geq m_0$, all $\delta \in [0.445 - 1/(2m), 0.445 + 1/(2m)]$, and all $\tau \in [0.14 - 1/(2\delta m), 0.14 + 1/(2\delta m)]$, these conditions hold with $\beta_1 \approx 0.56$, $\beta_2 \approx 0.39$, and $\alpha \approx 0.47$. The allowable ranges for δ and τ ensure that they can be chosen so that δm and $\tau\delta m$ are integral. From now on we will assume δ and τ are chosen in this way.

We prove Theorem 1 by contradiction.

Definition. Let *time* i denote the time just after job i is scheduled. Let On_i and Opt_i denote the makespans of the on-line and optimal algorithms at time i, respectively.

If the algorithm is not $(2-\epsilon)$-competitive, then there is a shortest sequence of jobs $a_1, a_2, ..., a_{t+1}$ such that $On_{t+1} > (2-\epsilon)Opt_{t+1}$.

Lemma 2 $On_{t+1} = M_t + a_{t+1}$.

Proof. Since $a_1, a_2, \ldots, a_{t+1}$ is a shortest sequence on which the algorithm is not $(2-\epsilon)$-competitive, we know that the scheduling of job $t+1$ increases the makespan. If

$$M(L_t) + a_{t+1} \leq (2-\epsilon)A(F_t),$$

then

$$On_{t+1} = M(L_t) + a_{t+1} \leq (2-\epsilon)A(F_t) \leq (2-\epsilon)A_t$$

$$\leq (2-\epsilon)A_{t+1} \leq (2-\epsilon)Opt_{t+1}.$$

From this contradiction we infer that a_{t+1} is placed on the machine with least height overall. ■

Definition. A schedule is called *flat* at time i if $M_i > (1-\epsilon)A_i$.

Lemma 3 *The schedule at time t is flat.*

Proof. Suppose that the schedule at time t is not flat. Then since $M_t \leq (1-\epsilon)A_t$, we have

$$\begin{aligned} On_{t+1} &= M_t + a_{t+1} \\ &\leq (1-\epsilon)A_t + a_{t+1} \\ &\leq (1-\epsilon)A_{t+1} + a_{t+1} \\ &\leq (2-\epsilon)Opt_{t+1}, \end{aligned}$$

using $A_{t+1} \leq Opt_{t+1}$ and $a_{t+1} \leq Opt_{t+1}$. ■

Definition. At time r, the schedule is *half flat* if only the machines in F_r have height at most $(1-\epsilon)A_t$. Let S_i denote the sequence of $\tau\delta m$ machines of smallest height at time i, and the sequence of their heights. At time s, the schedule is *almost flat* if only the machines in S_s have height at most $(1-\epsilon)A_t$.

Observe that because the schedule is flat at time t, there must exist $r < s < t$ such that the schedule is half flat at time r and almost flat at time s.

Lemma 4 *At some point between times s and t, on each machine in S_s is scheduled at least one job of length at least $\beta = \beta_1 A_t + \beta_2 A(S_s)$.*

Proof. Since every machine in S_s has height at most $(1-\epsilon)A_t$ at time s, and the schedule at time t is flat, every machine in S_s must receive at least one more job. Fix some machine $p \in S_s$. Let job $i+1$ be the first job machine p receives after time s. Since a_{i+1} was put on the shortest machine overall at time $i+1$, we have

$$M(L_i) + a_{i+1} > (2-\epsilon)A(F_i),$$

so that

$$a_{i+1} > (2-\epsilon)A(F_i) - M(L_i). \tag{3}$$

We also have

$$A_t \geq A_i \geq (1-\delta)M(L_i) + \delta A(F_i)$$

so that

$$M(L_i) \leq \frac{A_t - \delta A(F_i)}{1-\delta}$$

$$A(F_i) \geq A(F_s) > (1-\tau)[(1-\epsilon)A_t] + \tau A(S_s) \tag{4}$$

From (3) we obtain

$$\begin{aligned} a_{i+1} &> (2-\epsilon)A(F_i) - [\frac{A_t - \delta A(F_i)}{1-\delta}] \\ &= [(2-\epsilon) + \frac{\delta}{1-\delta}]A(F_i) - \frac{1}{1-\delta}A_t, \end{aligned}$$

which, from (4), exceeds

$$[(2-\epsilon) + \frac{\delta}{1-\delta}][(1-\tau)(1-\epsilon)A_t + \tau A(S_s)] - \frac{1}{1-\delta}A_t$$

$$= \{[(2-\epsilon) + \frac{\delta}{1-\delta}](1-\tau)(1-\epsilon) - \frac{1}{1-\delta}]\}A_t$$

$$+\{[(2-\epsilon) + \frac{\delta}{1-\delta}]\tau\}A(S_s)$$

$$= \beta_1 A_t + \beta_2 A(S_s). \blacksquare$$

Lemma 5 $A(S_s) \leq \alpha A_t$.

Proof. By the previous lemma, each machine in S_s receives at least one job of length at least β between times s and t. Therefore the average of the heights of these machines increases by at least β. Since at time s the schedule is almost flat, all machines not in S_s have height exceeding $(1-\epsilon)A_t$. Therefore

$$\begin{aligned} A_t &\geq (1-\tau\delta)(1-\epsilon)A_t + \tau\delta(A(S_s) + \beta) \\ &= (1-\tau\delta)(1-\epsilon)A_t \\ &\quad +\tau\delta[\beta_1 A_t + (1+\beta_2)A(S_s)] \end{aligned}$$

$$\begin{aligned} A(S_s) &\leq \frac{1-(1-\tau\delta)(1-\epsilon)-\tau\delta\beta_1}{\tau\delta(1+\beta_2)}A_t \\ &= [\frac{1-\beta_1}{1+\beta_2} + \epsilon\frac{1-\tau\delta}{\tau\delta(1+\beta_2)}]A_t \\ &= \alpha A_t. \blacksquare \end{aligned}$$

Notation. $h_i(p)$ denotes the height of the schedule of machine p at time i.

Lemma 6 *If $h_j(p) > \alpha A_t$ for some machine $p \in F_r$ and some time $j \in \{r, r+1, r+2, ..., s\}$, then $h_s(p) = h_j(p)$.*

Proof. By Lemma 5, $M_j \le M_s \le A(S_s) \le \alpha A_t$ for all j such that $r \le j \le s$. It follows that after time j, p never became the shortest overall machine prior to time $s+1$, and therefore its height cannot increase. ■

Lemma 7 *$A(F_r) \le \alpha A_t$.*

Proof. It follows from Lemma 6 that for each machine $p \in F_r$, either $h_r(p) \le \alpha A_t$, or $h_s(p) = h_r(p) \le (1-\epsilon)A_t$, i.e., $p \in S_s$ (because the schedule is almost flat at time s). Therefore all machines in F_r, except those also in S_s, have height at most αA_t at time r. Every machine in S_s is in F_r. The average height of these $(\tau\delta)m$ machines at time r is no greater than their average height at time s, which is at most αA_t by Lemma 5. The remaining $\delta m - \tau\delta m$ machines in F_r have height at most αA_t at time r. It follows that

$$A(F_r) \le \tau(\alpha A_t) + (1-\tau)(\alpha A_t) = \alpha A_t. \blacksquare$$

Lemma 8 *At any time j, for each machine $p \in L_j$, either $h_j(p) \le (2-\epsilon)A(F_j)$ or p contains a job of size at least $h_j(p) - M_j$.*

Proof. When the topmost job, say, a_i, was put on p just after time $i-1 < j$, either it was put on the shortest machine in L_{i-1}, in which case

$$h_j(p) = h_i(p) \le (2-\epsilon)A(F_{i-1}) \le (2-\epsilon)A(F_j),$$

or it was put on the shortest machine overall, in which case

$$h_j(p) = h_i(p) = M_{i-1} + a_i,$$

and hence

$$a_i = h_j(p) - M_{i-1} \ge h_j(p) - M_j. \blacksquare$$

Lemma 9 *Each machine in L_r contains a job of size at least $(1-\epsilon-\alpha)A_t$.*

Proof. Since at time r the schedule is half flat, for each machine $p \in L_r$, $h_r(p) > (1-\epsilon)A_t$; and since inequality (2) implies $\alpha \le \frac{1-\epsilon}{2-\epsilon}$, we have, by Lemma 7,

$$h_r(p) > (1-\epsilon)A_t \ge (1-\epsilon)\frac{1}{\alpha}A(F_r) \ge (2-\epsilon)A(F_r).$$

Therefore, by Lemma 8, p contains a job of size at least

$$h_r(p) - M_r \ge h_r(p) - A(F_r)$$

$$\ge (1-\epsilon)A_t - \alpha A_t = (1-\epsilon-\alpha)A_t. \blacksquare$$

Now we complete the proof of Theorem 1. By Lemma 9 and inequality (2), every machine in L_r contains a job of size at least $(1-\epsilon-\alpha)A_t \geq \frac{1}{2(1-\epsilon)}A_t$. By Lemma 4 and inequality (1), every machine in S_s contains a job of size at least $\beta_1 A_t \geq \frac{1}{2(1-\epsilon)}A_t$ at time t. As shown in the proof of Lemma 7, any machine in F_r, other than those in S_s, has height at most αA_t at time r. The height at time s of that machine exceeds $(1-\epsilon)A_t$. It follows by Lemma 6 that between times r and s the machine's height never increases to a height in $(\alpha A_t, (1-\epsilon)A_t]$. Therefore, it must contain a job of size at least $(1-\epsilon-\alpha)A_t \geq \frac{1}{2(1-\epsilon)}A_t$. Thus every machine at time t contains a job of length at least $\frac{1}{2(1-\epsilon)}A_t$.

If $a_{t+1} \leq (1-\epsilon)M_t$, then

$$\begin{aligned} On_{t+1} &= M_t + a_{t+1} \\ &\leq (2-\epsilon)M_t \\ &\leq (2-\epsilon)M_{t+1} \\ &\leq (2-\epsilon)Opt_{t+1}. \end{aligned}$$

If $a_{t+1} > (1-\epsilon)M_t$, then inequality (2) implies that $1-\epsilon \geq \frac{1}{2(1-\epsilon)}$ from which we learn that $a_{t+1} > (1-\epsilon)M_t \geq \frac{1}{2(1-\epsilon)}M_t$. Therefore, among jobs $1, 2, ..., t+1$, there are at least $m+1$ jobs of length at least $\frac{1}{2(1-\epsilon)}M_t$, two of which must go on the same machine:

$$Opt_{t+1} \geq \max\{a_{t+1}, \frac{1}{1-\epsilon}M_t\}.$$

Thus

$$\begin{aligned} On_{t+1} &= M_t + a_{t+1} \\ &\leq (1-\epsilon)Opt_{t+1} + Opt_{t+1} \\ &= (2-\epsilon)Opt_{t+1}. \end{aligned}$$

In either case, we have the contradiction that completes the proof of Theorem 1. ■

An Optimal Randomized 2-Machine Algorithm

No earlier work seems to exist on randomized algorithms for the m-machine scheduling problem. We exhibit below a trivial proof that no randomized m-machine scheduling algorithm can be c-competitive for any $c < 4/3$ (if $m \geq 2$), and then we exhibit a 4/3-competitive randomized algorithm for two machines. Thus, randomization allows one to lower the 2-machine competitive ratio from 3/2 to 4/3.

Theorem 10 *There is no randomized m-machine scheduling algorithm with a competitive ratio less than $4/3$ for any $m \geq 2$.*

Proof Sketch. Consider the job sequences $m \times 1$ and $m \times 1,\ 2$. (Here $m \times 1$ denotes a sequence of m 1's.) If the algorithm schedules the m 1's on m different machines with probability p, the worst-case ratio between its makespan and the optimal makespan is at least $\max\{2-p, 1+p/2\} \geq 4/3$. ■

For the rest of this section we focus on the 2-machine case.

Definition. Define the *tall machine* and *short machine* to be the machine with the greater and lesser height to date, respectively. The *discrepancy* of a schedule is the difference between the heights of the two machines.

The 2-Machine Algorithm. The algorithm maintains the set of all possible schedules generated to date, exactly 2^{t-1} schedules after $t-1$ jobs have arrived, each with the probability that it was generated. When job t arrives, the algorithm computes the overall expected discrepancy E that would result if job t were placed on the shorter machine of each schedule, and the overall expected discrepancy E' if it were placed on the taller. If possible, the algorithm chooses a p, $0 \leq p \leq 1$, so that $p \cdot E + (1-p) \cdot E' = (1/3)(a_1 + a_2 + \cdots + a_t)$, and then flips a biased coin and in each schedule places the a_t on the shorter machine with probability p. If no such p exists, in each schedule the algorithm places the a_t on the shorter machine.

(It is possible to rework the algorithm so that after n jobs have been scheduled, the algorithm maintains a list of at most n possible schedules, not 2^n. Here's how. Consider job t. Tentatively assign job t to the short machine in each of the at most $t-1$ schedules existing after $t-1$ jobs have been scheduled. If the resulting expected discrepancy is at least $(a_1 + a_2 + \cdots + a_t)/3$, then permanently assign job t to the shorter machine in each schedule. Otherwise, start tentatively moving job t from the short to the long machine in each of the at most $t-1$ schedules. Do this one at a time, in any order. The average discrepancy will increase as a result. Stop the moment such a move increases the average discrepancy to at least $(a_1 + a_2 + \cdots + a_t)/3$; that this must happen eventually follows from the invariant below. Let P be the (critical) schedule where this occurs. Choose a $q \in [0,1]$ so that if job t is assigned to the small machine of P with conditional probability q, the resulting expected discrepancy is exactly $(a_1 + a_2 + \cdots + a_t)/3$. Use this q to assign job t to the short machine. The resulting sequence of schedules has average discrepancy exactly $(a_1 + a_2 + \cdots + a_t)/3$. The choice of P and q depends on

$a_1, a_2, ..., a_t$, but is independent of the actual sequence of random numbers chosen by the algorithm.)

The algorithm maintains the following invariant: the expected discrepancy, over *all* possible sequences of coin flips to date, is at least one third of the sum of the lengths of all jobs seen to date. We will say the invariant is *tight* if the expected discrepancy is exactly one third the sum of all processing times.

The argument that the algorithm maintains this invariant is as follows. It is certainly true before any jobs arrive. Now suppose that when job t (of size a_t) arrives, the expected discrepancy would be at most one third the sum of all job sizes (including a_t). Then the algorithm randomly assigns job t to either the shorter or taller machine, with the bias chosen precisely so that the invariant will be tight. If the expected discrepancy would exceed one third the new sum, the algorithm deterministically places job t on the shorter machine.

Definition. We divide the times when the invariant is not tight into *phases*. A job starts a new phase if its size is at least as large as the sum of all previous sizes, and after the job is added, the invariant is not tight. The phase consists of that job and all successive jobs up to but not including the first job the scheduling of which either (a) starts a new phase or (b) makes the invariant tight.

Lemma 11 *If the invariant is tight just after job r has been scheduled, then $E[On_r] \leq (4/3)Opt_r$.*

Proof. Let On_r denote the (random) makespan just after job r is scheduled, and let D_r denote its (random) discrepancy. Let Opt_r denote the optimal makespan at time r. We have

$$On_r = (1/2)[D_r + \sum_{j=1}^{r} a_j].$$

Thus

$$\begin{aligned} E[On_r] &= (1/2)[(1/3)\sum_{j=1}^{r} a_j + \sum_{j=1}^{r} a_j] \\ &= (2/3)\sum_{j=1}^{r} a_j \\ &\leq (4/3)Opt_r. \blacksquare \end{aligned}$$

Lemma 12 *Suppose that job r, of length $a_r = L$, arrives and starts a new phase. Let $S = a_1 + a_2 + \cdots + a_{r-1}$. (Thus $L \geq S$.) Then $E[On_r] \leq (4/3)Opt_r$.*

Proof. The L is added deterministically to the shorter machine, for otherwise we would have randomized to keep the discrepancy tight, and a new phase wouldn't have started. The new discrepancy of a schedule previously of discrepancy d is $L - d$, since the L is placed on the shorter machine. Thus the average discrepancy after L is added is exactly

$$E[D_r] = L - E[D_{r-1}] \leq L - (1/3)S.$$

$$E[S + L] = S + L$$

Since $D_r + (S + L) = 2On_r$, it follows that $2E[On_r] \leq (2/3)S + 2L \leq (8/3)L$. We conclude that

$$E[On_r] \leq (4/3)L \leq (4/3)Opt_r. \blacksquare$$

Lemma 13 *Let $p_1, p_2, ..., p_n$ be a sequence of nonnegative reals summing to one, and let $b_1, b_2, ..., b_n$ be nonnegative reals satisfying $b_i \leq 3t$ for all i, where $t = \sum_{i=1}^{n} b_i p_i$. Let*

$$f(x) = \sum_{i=1}^{n} |x - b_i| p_i - t.$$

Then for nonnegative x, $f(x) \leq x/3$ if and only if $x \leq 3t$.

Proof. Without loss of generality we may assume that $b_1 \leq b_2 \leq \cdots \leq b_n$. Clearly $f(0) = 0$, so $f(x) \leq x/3$ for $x = 0$. For $x \geq 3t$, $f(x) = \sum_{i=1}^{n} (x - b_i)p_i - t = x - 2t$. Therefore $f(3t) = t$. Since f has a slope of 1 for $x \geq 3t$ and the function $x/3$ has a slope of $1/3$, it follows that $f(x) > x/3$ for all $x > 3t$.

For which s, $0 < s < 3t$, if any, is $f(s) > s/3$? Note that f is continuous and piecewise linear and its slope is nondecreasing with x. Since $f(0) = 0$, the slope in $[s, 3t]$ (where the slope exists) must exceed $1/3$, for otherwise the slope is at most $1/3$ in $[0, s]$, and it is not possible that $f(s) > s/3$. It follows that $f(3t) > f(s) + (3t - s)/3 > s/3 + (1/3)(3t - s) = t$, a contradiction. $\blacksquare$

Lemma 14 *If the invariant is tight after the arrival of job s, then it is tight after the arrival of job $s + 1$ if and only if $a_{s+1} \leq a_1 + a_2 + \cdots + a_s$.*

Proof. After job s is scheduled, consider all, say, n schedules that have positive probability; their discrepancies are $b_1, b_2, ..., b_n$ with respective probabilities $p_1, p_2, ..., p_n$. Clearly $p_1 + p_2 + \cdots + p_n = 1$. Let $t = \sum_{i=1}^{n} b_i p_i$, the expected discrepancy. Since the invariant is tight after job s is scheduled, we have $t = (1/3)S$, where $S = a_1 + a_2 + \cdots + a_s$. We have $b_i \leq S = 3t$. Lemma 13 now implies that if job $s + 1$ is deterministically placed on the shorter machine, the expected discrepancy will increase by at most $(1/3)a_{s+1}$ if and only if $a_{s+1} \leq 3t = S$. Under precisely the former condition the invariant is tight after the algorithm (randomly) schedules job $s + 1$. $\blacksquare$

Lemma 15 *If the invariant is not tight after job $r+1$ is scheduled, then job $r+1$ is part of some phase.*

Proof. It suffices to prove that if the invariant is tight after job s is scheduled but not after job $s+1$ is, then job $s+1$ initiates a new phase. By Lemma 14, we have $a_{s+1} > a_1 + a_2 + \cdots + a_s$. This means that job $s+1$ initiates a new phase. ■

Lemma 16 *After the first job in a phase (say, job r) is scheduled, the makespan does not increase during the phase.*

Proof. Let $S = a_1 + a_2 + \cdots + a_{r-1}$. Let $e_1, e_2, ..., e_n$ be the possible discrepancies just before job r (of size, say, L) arrives, the e_i occurring with probability p_i. We have $\sum e_i p_i \geq S/3$. $L \geq S \geq e_i$. This means that after the L arrives, the average discrepancy is

$$\begin{aligned} \bar{D} &= \sum_{i=1}^{n} |L - e_i| p_i \\ &= \sum_{i=1}^{n} (L - e_i) p_i \\ &= L - \sum_{i=1}^{n} p_i e_i \\ &\leq L - S/3. \end{aligned}$$

Let

$$b = \frac{3}{4}[\bar{D} - \frac{S+L}{3}] \geq 0.$$

Thus

$$b \leq \frac{3}{4}((L - \frac{S}{3}) - \frac{S+L}{3})$$

$$= \frac{3}{4}(\frac{2}{3}L - \frac{2}{3}S) = \frac{L-S}{2} \leq L - S.$$

Thus $b \leq L - S$.

Let the jobs in the phase be jobs $r, r+1, r+2, ..., t$. After each job in the phase is scheduled, the expected discrepancy exceeds $1/3$ the sum of job sizes to date.

If $a_{r+1} + \cdots + a_t \leq b$, then, since $b \leq L - S$, the addition of jobs $r+1, r+2, ..., t$ does not increase the makespan. Suppose, for a contradiction, that $a_{r+1} + \cdots + a_t > b$, and choose l minimal such that $a_{r+1} + \cdots + a_l > b$; $a_{r+1} + \cdots + a_{l-1} \leq b$. Therefore in each schedule jobs $r+1, ..., l-1, l$ are all put on the same machine. Write $a_{r+1} + \cdots + a_l = b + \Delta$ with $\Delta > 0$.

We will study the performance of the algorithm when jobs $r+1, ..., l$ are replaced by a job of length b followed by a job of size Δ (the sum of their sizes is the same). In fact, we will argue that the algorithm would deterministically place both the b and the Δ on the same machine, the shorter one. Further, after doing so, the expected discrepancy would be at most 1/3 of the new sum. This will be a contradiction, since during a phase the expected discrepancy should exceed 1/3 the sum.

There are two cases: either $\Delta \leq S + L + b$ or $\Delta > S + L + b$. First, we assume $\Delta \leq S + L + b$. Imagine that at time $r+1$ one job of length b arrives. Since $b \leq L - S$, if the b were scheduled on the shorter machine deterministically, the expected discrepancy would be $\bar{D} - b$. This is exactly one third of the sum $S + L + b$ of all jobs up to and including the b, since b was chosen to be $(3/4)[\bar{D} - (S + L)/3]$, the exact value which would make this true. Lemma 14 implies that if the invariant is tight, as it is after the (fictitious) b is scheduled, then if a new job of length at most the sum of all previous job sizes is scheduled deterministically on the shorter machine, the expected discrepancy will be at most one third the new sum. Thus $\Delta \leq S + L + b$ implies that the expected discrepancy if a Δ is added after the b is at most 1/3 of the total sum, $S + L + b + \Delta$, which equals $S + L + (a_{r+1} + a_{r+2} + \cdots + a_l)$. (Note that in each schedule the Δ would be scheduled on the same machine as the b, because $b \leq L - S$.) It follows that the expected discrepancy after $a_{r+1}, a_{r+2}, ..., a_l$ are scheduled on the shorter machine is at most 1/3 of the sum of jobs 1 through l, a contradiction.

The other case is that $\Delta > S + L + b$. Thus $a_l \geq \Delta > S + L + b \geq a_1 + a_2 + \cdots + a_{l-1}$. Thus job l should not have been included in this phase.

Both cases have led to contradictions. We conclude that $a_{r+1} + \cdots + a_t \leq b$, and we have already shown that in this case the makespan does not increase. ■

Theorem 17 *The algorithm above is* 4/3*-competitive.*

Proof. The theorem follows immediately from Lemmas 11, 12, 15 and 16. ■

A 3-Machine Lower Bound

For 3-machine scheduling, one can prove a better lower bound.

Theorem 18 *The competitive ratio of every randomized,* 3*-machine scheduling algorithm is at least* 1.4*.*

Proof Sketch. We consider an adversary who presents the on-line algorithm with one of these five job sequences, all of which are prefixes of the list $1, 1, 1, 2, 1, 3, 5$:

(a) $1, 1, 1$

(b) $1, 1, 1, 2$

(c) $1, 1, 1, 2, 1$

(d) $1, 1, 1, 2, 1, 3$

(e) $1, 1, 1, 2, 1, 3, 5$

A lower bound obtained in this restricted situation is a lower bound on the competitive ratio in general.

There are 16 "reasonable" ways to schedule the jobs $1, 1, 1, 2, 1, 3, 5$ on three machines. By "reasonable," we mean that for any other way τ to schedule those seven jobs on three machines, one schedule σ of the 16 preferred schedules dominates the given schedule, in that the makespans for σ for *all* of the five job sequences are no greater than the corresponding makespans of τ. Below, we list the 16 reasonable schedules. The jobs assigned to the machines are listed in order of arrival, with those assigned to machine one preceding those assigned to machine two, which precede those assigned to machine three.

1.	$1, 2; 1, 1, 5; 1, 3$	9.	$1, 1, 1; 1, 3; 2, 5$
2.	$1, 2, 1; 1, 3; 1, 5$	10.	$1, 1, 1; 1, 5; 2, 3$
3.	$1, 2; 1, 1, 3; 1, 5$	11.	$1, 1, 5; 1, 2; 1, 3$
4.	$1, 1, 1, 1; 2, 3; 5$	12.	$1, 1, 3; 1, 2, 1; 5$
5.	$1, 1, 1; 2, 5; 1, 3$	13.	$1, 1, 5; 1, 2, 1; 3$
6.	$1, 1, 1; 2, 3; 1, 5$	14.	$1, 1, 1, 5; 1, 2; 3$
7.	$1, 1, 1; 2, 1; 3, 5$	15.	$1, 1, 2; 1, 3; 1, 5$
8.	$1, 1, 3; 1, 1, 5; 2$	16.	$1, 1, 2; 1, 1, 3; 5$

In Table 1 we list the competitive ratio that each of the 16 schedules achieves on each of the five job sequences. Let A denote the 16×5 matrix of the table. The on-line algorithm's goal is to choose $y \in \mathbb{R}$ and $p \in \mathbb{R}^{16}$ so as to minimize y, subject to $A^T p \leq y\mathbf{1}$, $\mathbf{1} \cdot p = 1$, $p \geq 0$. (Here $\mathbf{1}$ denotes both a 5-vector and a 16-vector of 1's.) In fact, the optimal solution has $y = 7/5$, $p_1 = 7/10$, $p_5 = 1/10$, $p_8 = 1/5$, and all other $p_i = 0$. (That this solution is feasible can be verified quickly by hand.) To prove that 1.4 is indeed the optimal cost, we present the dual linear program: Find $z \in \mathbb{R}$ and $q \in \mathbb{R}^5$ so as to maximize z, subject to $Aq \geq z\mathbf{1}$, $\mathbf{1} \cdot q = 1$, $q \geq 0$. (Again, $\mathbf{1}$ is both a 16- and a 5-vector.) It is easy to verify by hand that $q = [0 \;\; 0 \;\; 1/5 \;\; 3/10 \;\; 1/2]^T$ and $z = 1.4$ is a

	(a)	(b)	(c)	(d)	(e)
1	1	3/2	3/2	4/3	7/5
2	1	3/2	2	4/3	6/5
3	1	3/2	3/2	5/3	6/5
4	3	3/2	2	5/3	1
5	3	3/2	3/2	4/3	7/5
6	3	3/2	3/2	5/3	6/5
7	3	3/2	3/2	1	8/5
8	2	1	1	5/3	7/5
9	2	1	3/2	4/3	7/5
10	2	1	3/2	5/3	6/5
11	2	3/2	3/2	4/3	7/5
12	2	3/2	2	5/3	1
13	2	3/2	2	4/3	7/5
14	2	3/2	3/2	1	8/5
15	2	2	2	4/3	6/5
16	2	2	2	5/3	1

Table 1: The competitive ratios of the 16 reasonable schedules

feasible solution to the dual. This means that the optimal cost of the primal is exactly 1.4, as claimed. ■

Open Problems

The obvious questions are improvements in the upper bounds and lower bounds presented here and in [1, 2, 3], i.e., better upper and lower bounds on the deterministic competitive ratio for $m \geq 4$ machines, a better randomized lower bound for $m = 3$, any good randomized upper bound for $m \geq 3$, and any good randomized lower bound for $m \geq 4$.

Additionally, many related scheduling problems presented in [3] also have a competitive ratio of $2 - 1/m$. The natural question is whether the techniques used here are applicable to these problems. This question is relevant both to polynomial-time off-line approximation issues and to on-line versions of these problems.

References

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

[2] G. Galambos and G. Woeginger, "An On-Line Scheduling Heuristic with Better Worst Case Ratio than Graham's List Scheduling," manuscript, József Attila University, Szeged, Hungary.

[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] 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.

[6] 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.

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

[8] N. Linial, personal communication.

[9] P. R. Narayanan and R. Chandrasekaran, "Optimal On-Line Algorithms for Scheduling," manuscript, University of Texas at Dallas, 1991.