Randomized Online Scheduling on Two Uniform Machines

Leah Epstein* John Noga† Steve Seiden‡ Jiří Sgall§
Gerhard Woeginger¶

*lea@math.tau.ac.il, Department of Computer Science, Tel-Aviv University, Israel.

†noga@opt.math.tu-graz.ac.at, Institut für Mathematik, Technische Universität Graz, Steyrergasse 30, A-8010 Graz, Austria. This research has been supported by NSF grants INT9724750 and CCR9503498.

‡sseiden@acm.org, Department of Computer Science, 298 Coates Hall, Louisiana State University, Baton Rouge, LA 70803, USA. This research was done at Technische Universität Graz, supported by the START program Y43-MAT of the Austrian Ministry of Science.

§sgall@math.cas.cz, Mathematical Institute, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic and Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Praha. Partially supported by grants A1019602 and A1019901 of GA AV ČR, postdoctoral grant 201/97/P038 of GA ČR, and cooperative research grant INT-9600919/ME-103 from the NSF and MŠMT ČR.

¶gwoegi@opt.math.tu-graz.ac.at, Institut für Mathematik, Technische Universität Graz, Steyrergasse 30, A-8010 Graz, Austria. This research has been supported by the START program Y43-MAT of the Austrian Ministry of Science.

Abstract

We study the problem of online scheduling on two uniform machines with speeds 1 and $s \geq 1$. A $\phi \approx 1.61803$ competitive deterministic algorithm was already known. We present the first randomized results for this problem: We show that randomization does not help for speeds $s \geq 2$, but does help for all $s < 2$. We present a simple memoryless randomized algorithm with competitive ratio $(4-s)(1+s)/4 \leq 1.56250$. We analyze other randomized algorithms that demonstrate that the randomized competitive ratio is at most 1.52778 for any s. These algorithms are barely random, i.e., they use only a constant number of random bits. Finally, we present a $1 + s/(s^2+s+1)$ competitive deterministic algorithm for the preemptive version of this problem. For any s, it is best possible even among randomized preemptive algorithms.

Keywords: Analysis of algorithms, online algorithms, competitive ratio, scheduling.

1 Introduction

In this paper, we study online scheduling of independent jobs with the objective to minimize the *makespan* (i.e., the length of the schedule, or equivalently the last job completion time). An input to this problem consists of a sequence of jobs of different *sizes* (or *running times*), each of which has to be assigned to one of two machines. There are no dependencies between the jobs and no release times. The machines are *uniform* in that a job with running time t is processed in time t on the slow machine, and in time t/s on the fast machine. We consider the *non-preemptive* version of this problem (once a job is started, it must be processed without interruption until completion on the same machine) and the *preemptive* version (the processing of a job may be interrupted and resumed arbitrarily, and may even be switched between the two machines). In the non-preemptive version, it is never useful to introduce idle time; so the makespan is simply the maximum over machines of the sum of running times of assigned jobs.

In the online problem, jobs arrive one by one and we must assign each job before we see the next one. Particularly note that in the preemptive version all decisions about interruptions and resumptions of the current job must be made prior to the next job being revealed.

The performance of an online algorithm is measured by its *competitive ratio*, which is the largest ratio of the makespan achieved by the online algorithm to the minimal makespan. For randomized algorithms, we use expected makespan in the definition of the competitive ratio (this corresponds to the so-called oblivious adversary). Frequently, when working with randomized algorithms we will use the term makespan when we mean the expected makespan.

1.1 Our results

In this paper, we are mainly interested in the dependency of the optimal competitive ratio on the speed s.

Let us first discuss the non-preemptive version. We start by analyzing the performance of deterministic algorithms depending on the speed s. It turns out that the greedy List algorithm is best for any speed: the optimal competitive ratio is $(2s+1)/(s+1)$ for $s \leq \phi$ and $1 + 1/s$ for $s \geq \phi$ ($\phi \approx 1.61803$ is the golden ratio). Much of the material is implicit in [9], where the upper bound is proved for all s as well as the fact that this is tight for speed $s = \phi$. We include this analysis since the ideas used are essential for the rest of the paper.

We prove that randomization does not help at all for $s \geq 2$. For $1 < s < 2$ we prove lower bounds showing that the randomized competitive ratio for any s in this range is strictly larger than $4/3$, the optimal randomized competitive ratio for $s = 1$ [4].

On the other hand, for any $1 < s < 2$, even the simple *memoryless* algorithm which schedules each job independently with probability $s/2$ on the fast machine is better than any deterministic algorithm; its competitive ratio is $(4 - s)(1 + s)/4$. Is this memoryless algorithm the best possible randomized algorithm for speeds $1 < s < 2$? The answer to this question is no, and further it is no for every speed in this range. We develop two different algorithms for the case of a *small* speed $1 < s < 3/2$, and for the case of a *medium* speed $3/2 \leq s < 2$.

For small speeds, as s approaches 1, the competitive ratio of the memoryless algorithm approaches 3/2 (which is the optimal *deterministic* competitive ratio for two identical machines). On the other hand, 4/3 is the optimal *randomized* competitive ratio for two identical machines [4]. We design a randomized algorithm whose competitive ratio approaches 4/3 as s approaches 1; this is our best algorithm for small speeds, always better than the memoryless algorithm.

As our main result we develop good randomized algorithms for medium speeds. For $s = 3/2$ the memoryless algorithm schedules each job on the slow machine with probability 1/4. Its obvious drawback is that its decisions are independent of each other. Thus, if we get three big jobs, with significant probability it schedules two of them on the slow machine; this results in rather poor performance. To improve on this, we use a deterministic algorithm for scheduling on four imaginary machines. We pick one of the four machines at random and identify it with the slow machine, whereas the other three are identified with the fast machine. Hence, on average 1/4 of the load goes on the slow machine (as in the memoryless algorithm), but if in the previous example the 4-machine algorithm schedules the three big jobs on different machines, then we avoid the poor performance of memoryless algorithms. These algorithms (except for the memoryless one) are *barely random* [5], i.e., they use only a constant number of random bits (2 bits for speed 3/2).

For the preemptive version, we present a complete solution: for any $s \geq 1$, there exists a deterministic online algorithm that is $1 + s/(s^2 + s + 1)$ competitive, and randomized algorithms cannot improve on this. The same bounds have been found independently by Wen and Du [17]. Note that the optimal competitive ratio strictly decreases in the whole range of $s \geq 1$. This is in contrast to the non-preemptive version, where both for the deterministic and for the randomized case the optimal competitive ratio does increase with an increase in the speed in some range.

See Figure 1 for a comparison of all our bounds.

1.2 Motivation and related results

Most of the study of on-line algorithms for uniformly related machines deals with the case of an arbitrary number of machines. For non-preemptive scheduling on uniformly related machines the first algorithm with a constant competitive ratio was given by Aspnes *et al.* [2]; it is deterministic and its competitive ratio is 8. This was improved by Berman *et al.* [6]; they present 5.8285-competitive deterministic and 4.3111-competitive randomized algorithms. Berman *et al.* [6] also prove lower bounds of 2.4380 for deterministic non-preemptive scheduling. The best lower bound for randomized algorithms, both preemptive and non-preemptive, is 2, by Epstein and Sgall [11]. For preemptive

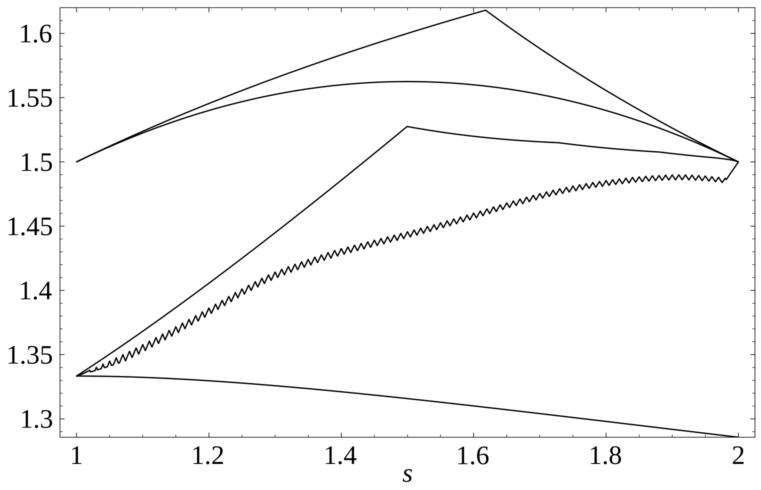

Figure 1: Deterministic, memoryless randomized, and general randomized upper bounds, the randomized lower bound, and the optimal preemptive bound (from top to bottom) for $1 \leq s \leq 2$.

scheduling no good algorithms are known. However, a recent result [18] shows that the preemptive and non-preemptive optimum differ by a factor of at most 2. Therefore, non-preemptive deterministic and randomized algorithms achieve ratios of 11.6570 and 8.6222, respectively. See [7] for a general reference on online algorithms, and [16] for a survey of online scheduling.

The problem of scheduling on two machines is less general, consequently our bounds are tighter and a more detailed study of the influence of speed can be performed.

Our main achievement is a new method of constructing randomized algorithms from deterministic ones, namely the meta-algorithm Pick One. Similar methods can be also used for identical machines, this is further elaborated in [15].

We find this problem of studying the influence of the speed interesting also for the following two reasons.

First, the problem is related to the notorious problem of randomized scheduling on m identical machines. This is a very basic problem in the area of competitive analysis, and our understanding of randomized algorithms for it is very poor [4, 14, 16]. Roughly speaking, there is an analogy between assigning jobs to the slow and fast machines in our problem and to one machine (or a few machines) and the remaining machines in the m-machine problem. Even though we do not see a direct formal connection, we believe that a better understanding of algorithms for two uniform machines will help us to attack the m-machine problem.

Second, as a general rule, our understanding of randomized algorithms is good for problems that exhibit some symmetric structure; however, once we break the symmetry, our understanding worsens rapidly. For example, in the k-server problem we have optimal algorithms for the paging problem [13] which corresponds to the uniform metric space. Once we move away from the uniform space (and from symmetry), the problem is wide open even for 2 servers [3, 10]. In our scheduling problem we fully understand the case of $s = 1$. This is the symmetric case, where the machines can arbitrarily interchange their roles during the algorithm's execution; the solution and the analysis of this case strongly relies on that fact. The case $s \neq 1$ is the non-symmetric case. We believe that the study of algorithms for accessible non-symmetric problems is an important research area, and we hope that further progress will be made.

1.3 Preliminaries

Most of our lower bounds are achieved on the *critical sequence*, which is defined so that each job (except for the first one) takes as long to run on the fast machine as all the previous jobs on the slow machine. Thus we define the running times of jobs of the critical sequence as

$$\begin{aligned} r_0 &= 1, \\ r_i &= s(s+1)^{i-1} \quad \text{for } i \geq 1. \end{aligned} \tag{1}$$

For the critical sequence ending at r_i, the optimal makespan is equal to $r_i = (s+1)^{i-1}$, and in any schedule the makespan is determined by the machine on which the last job was scheduled.

2 Deterministic scheduling

The algorithm List schedules every job on the machine where it will finish first. It was introduced as an algorithm for scheduling jobs on m identical machines and first studied by Graham [12], who showed that it is $2-1/m$ competitive for all m. List is the best possible deterministic online algorithm for $m = 2, 3$, whereas for all $m \geq 4$ better algorithms are known [1, 8].

In [9] it is proved that List is $\phi \approx 1.61803$ competitive for scheduling on two uniform machines, and that no better online algorithm is possible. The following theorem gives the exact dependence of the competitive ratio on the speed and shows that List is optimal for any speed s.

Theorem 2.1 *For any $s \geq 1$, the competitive ratio of* List *is*

$$c_{\text{LIST}} = \min\left\{\frac{2s+1}{s+1}, 1+\frac{1}{s}\right\}.$$

The competitive ratio of any non-preemptive deterministic online algorithm is at least c_{LIST}, for any $s \geq 1$.

Proof. The upper bound is from [9]. For $s \leq \phi$ the bound of $(2s+1)/(s+1)$ follows Graham's analysis of List on identical machines. For $s \geq \phi$, we consider the algorithm that schedules all jobs on the fast machine. This is clearly never better than LIST. If the makespan of this algorithm is x, the optimal algorithm has to schedule jobs with total running time x as well; it can do it only a factor of $1+1/s$ faster, since the slow machine only processes $1/s$ fraction of the jobs processed by the fast machine during the same time.

To prove the lower bound, for $s \geq \phi$, consider the sequence of two jobs 1 and s. The first job has to be scheduled on the fast machine, since otherwise the competitive ratio would be $s \geq 1 + 1/s$ (since $s \geq \phi$). After both jobs, the optimal makespan is 1. Thus the second job has to be scheduled on the fast machine as well, since otherwise the makespan is s and the competitive ratio is again s. But with both jobs on the fast machine the competitive ratio is $(s+1)/s$, and the bound follows.

For $s < \phi$, let $\alpha = (s+1-s^2)/s^2 = 1/s + 1/s^2 - 1$. Note that $0 < \alpha \leq 1/s \leq 1$. Our goal is to force the situation when the total time of jobs assigned to the fast machine is $x > 0$ and the total time of jobs on the slow machine is αx. If this happens, we continue with a job of size $(1+1/s)x$. Then the makespan of the algorithm is at least $(2/s + 1/s^2)x$ (no matter on which machine the job is scheduled), while the optimum is $(1/s + 1/s^2)x$ (scheduling only the last job on the fast machine), and the competitive ratio is at least $(2s+1)/(s+1)$. Now start the sequence with a job of size 1. If it is scheduled on the slow machine, continue with a job of size $1/\alpha$. If this job is scheduled on the fast machine, we have the desired ratio of loads $1 : \alpha$. Otherwise both jobs are scheduled on the slow machine, and a calculation yields that the competitive ratio is $1 + 1/s$. If the first job is scheduled on the fast machine, we continue with jobs of sizes α, $\alpha(1+\alpha)$, $\ldots$, $\alpha(1+\alpha)^i$, $\ldots$. None of them can be scheduled on the slow machine, as we would get the ratio of loads $1 : \alpha$. Since the sizes of jobs are a geometric sequence with ratio in $(1, 2]$, for a sufficient number of jobs it is possible to partition them into two groups so that the ratio of total sizes of jobs in the groups is arbitrarily close to $s : 1$. This means that the optimal schedule almost balances the machines and the competitive ratio is arbitrarily close to $1 + 1/s$. ∎

3 Randomized lower bounds

3.1 Large s

For $s \geq 2$, we prove that randomization does not help; the strategy of putting all jobs on the fast machine is optimal. We make use of the von Neumann-Yao principle [19]. This principle allows us to prove a lower bound for any randomized algorithm by exhibiting a distribution over inputs and showing that any deterministic algorithm incurs a cost at least c times the optimal offline cost for this distribution.

Theorem 3.1 *For $s \geq 2$, the competitive ratio of any non-preemptive randomized online algorithm is at least $c_{\text{LIST}} = 1 + 1/s$.*

Proof. For a fixed n, consider the following distribution on input instances. Choose uniformly at random $i = 0, \ldots, n-1$, and take the instance consisting of the first $i+1$ jobs of the critical sequence (i.e., ending with a job with running time r_i). We prove that for any deterministic algorithm, the average value of the competitive ratio on this distribution of inputs is at least $1 + 1/s - 1/(ns)$. The theorem follows by taking sufficiently large n.

The action of a deterministic algorithm is determined by a set $S \subseteq \{0, \ldots, n-1\}$ of the indices of jobs scheduled on the slow machine. We show that the best average competitive ratio is achieved if $S = \emptyset$; the computation essentially shows that scheduling any single job on the fast machine only improves the performance.

If a job r_i is scheduled on the slow machine, the competitive ratio after i jobs is at least s (since the optimal makespan is r_i/s but running the job on the slow machine takes r_i). If it is scheduled on the fast machine, the competitive ratio is

$$\sum_{k \le i, k \notin S} \frac{r_k}{r_i} = \sum_{k \le i} \frac{r_k}{r_i} - \sum_{k < i, k \in S} \frac{r_k}{r_i}.$$

Let $u_i = \sum_{k \le i} r_k / r_i$. For $i > 0$ we get $u_i = 1 + 1/s$ and $u_0 = 1$. The average competitive ratio is at least

$$\begin{aligned}
& \frac{1}{n}\left(\sum_{j \in S} s + \sum_{i \notin S}\left(u_i - \sum_{k<i, k \in S} \frac{r_k}{r_i}\right)\right) \\
= & \frac{1}{n}\left(\sum_{i=0}^{n-1} u_i + \sum_{j \in S}(s - u_j) - \sum_{i \notin S} \sum_{j<i, j \in S} \frac{r_j}{r_i}\right) \\
= & 1 + \frac{1}{s} - \frac{1}{ns} + \frac{1}{n} \sum_{j \in S}\left(s - u_j - \sum_{i>j, i \notin S} \frac{r_j}{r_i}\right).
\end{aligned}$$

Now we prove that the expression in the sum is non-negative for all j. For $j > 0$ we have

$$\sum_{i>j, i \notin S} \frac{r_j}{r_i} \le \sum_{i=1}^{\infty} (s+1)^{-i} = \frac{1}{s}.$$

Thus the expression is at least $s - u_j - 1/s = s - 1 - 2/s \ge 0$, since $s \ge 2$. For $j = 0$, a similar calculation yields that the expression is at least $s - 1 - (s+1)/s^2$ which is also non-negative. Therefore the average competitive ratio is at least $1 + 1/s - 1/(ns)$. ■

3.2 Small and medium s

Next, we show lower bounds for any randomized algorithm for speeds $1 < s < 2$. Unfortunately, we are not able to give a good analytical lower bound for the whole range of s.

We give lower bounds for particular speeds $s = 1 + i/100$, solving numerically a linear program for each such s. For intermediate values, we use a simple fact that if s is changed by a small factor, the competitive ratio can change only by the same factor, see Lemma 3.2. (Using this lemma with the lower bound for $s = 2$ it is possible to slightly improve the numerical bounds for s very close to 2). These bounds are not sufficient for s close to 1, since we want to prove a lower bound strictly larger than 4/3, the bound for $s = 1$. In this range we prove an analytical lower bound in Lemma 3.3.

All the lower bounds are illustrated in Figure 1.

Again, we apply the von Neumann-Yao principle. With probability p_i the input sequence contains the first i jobs of the critical sequence, for $i = 1, \ldots, n$. Solving a linear program on a computer, we can find the best lower bounds achievable by such distributions, for fixed values of s and n. There are 2^n possible deterministic algorithms for scheduling n jobs. The makespan for algorithm j after i jobs, denoted $M_{i,j}$, is computed using the following recurrence:

$$\begin{aligned} L_{0,j} &= \langle 0, 0\rangle \\ L_{i,j} &= \begin{cases} L_{i-1,\lfloor j/2\rfloor} + \langle r_{i-1}, 0\rangle & \text{for odd } j \\ L_{i-1,\lfloor j/2\rfloor} + \langle 0, r_{i-1}/s\rangle & \text{otherwise} \end{cases} \\ M_{i,j} &= \max L_{i,j} \end{aligned}$$

The expected competitive ratio incurred by algorithm j is $C_{n,j}$. This is expressed as a fixed linear combination of p_i's for each n and s:

$$C_{n,j} = \sum_{i=1}^{n} p_i \frac{M_{i,j}}{r_i} \tag{2}$$

Define the following linear program over variables c, $p_1, \ldots, p_n$: Maximize c subject to

$$\sum_{i=1}^{n} p_i = 1, \qquad c \leq C_{n,j} \quad \text{for } j = 0, \ldots, 2^n - 1.$$

Any feasible c is a lower bound on the competitive ratio. For $n = 15$ and $s = 1 + i/100$ for $i = 1, \ldots, 99$, we have calculated the optimal values of this linear program using CPLEX. We display the actual lower bounds in Table 1. The value $c(i)$ is a lower bound for any randomized online algorithm at speed $s = 1 + i/100$. The bounds are valid for all decimal places shown, i.e. the last digit is always rounded down.

Lemma 3.2 *If c_0 is a lower bound on the competitive ratio for a fixed speed s_0, then $c_0 \min\{s_0/s, s/s_0\}$ is a lower bound on the competitive ratio for any speed s.*

Proof. Suppose that there is a c-competitive algorithm ALG at speed s. First consider $s > s_0$. If we use ALG at speed s_0, its cost for any instance increases at most by a factor of s/s_0. The optimal offline cost can not decrease. Therefore, ALG is cs/s_0-competitive

i	$c(i)$	$c(i+1)$	$c(i+2)$	$c(i+3)$
1	1.33333	1.33558	1.33787	1.34022
5	1.34261	1.34505	1.34754	1.35006
9	1.35263	1.35523	1.35790	1.36059
13	1.36331	1.36608	1.36888	1.37171
17	1.37457	1.37747	1.38039	1.38335
21	1.38632	1.38932	1.39233	1.39535
25	1.39837	1.40122	1.40407	1.40679
29	1.40948	1.41210	1.41433	1.41648
33	1.41853	1.42038	1.42226	1.42411
37	1.42594	1.42772	1.42948	1.43107
41	1.43255	1.43381	1.43524	1.43654
45	1.43783	1.43912	1.44042	1.44172
49	1.44305	1.44439	1.44569	1.44705
53	1.44844	1.44984	1.45111	1.45268
57	1.45416	1.45563	1.45711	1.45865
61	1.46018	1.46172	1.46342	1.46501
65	1.46657	1.46812	1.46964	1.47113
69	1.47258	1.47401	1.47539	1.47670
73	1.47799	1.47916	1.48020	1.48125
77	1.48221	1.48311	1.48398	1.48474
81	1.48549	1.48618	1.48677	1.48737
85	1.48793	1.48839	1.48885	1.48921
89	1.48947	1.48970	1.48986	1.48993
93	1.48989	1.48970	1.48943	1.48904
97	1.48853	1.48798	1.48731	1.48652

Table 1: Lower bounds for randomized algorithms.

at speed s_0, and $c \geq c_0 s_0/s$. Now consider $s < s_0$. If we use ALG at speed s_0, its cost does not increase, and the optimal offline cost decreases at most by a factor of s/s_0. Therefore, ALG is cs_0/s-competitive at speed s_0, implying $c \geq c_0 s/s_0$. ∎

For values of s close to 2, we can use Lemma 3.2 to obtain a lower bound of $3s/4$. This is better than the numerical values in Table 1 for $s \geq 1.98$, i.e., the last three values.

For values of s close to 1, in the next lemma we use a sequence at most 4 jobs in the critical sequence, with a suitably chosen probabilities. This does not improve any of the values in Table 1, since these were obtained using the optimal probabilities in a more general case of longer sequence of jobs. The improvement concerns only the intermediate values.

Lemma 3.3 *For $1 \leq s \leq \sqrt{2}$, the competitive ratio of any non-preemptive randomized online algorithm is at least $c_{low} = (s+1)^2(s+2)/(3s^2+5s+1)$.*

Proof. We use the von Neumann-Yao principle. We give the algorithm the first $i \leq 4$ jobs of the critical sequence with probability p_i where

$$\begin{aligned}
p_1 &= 0, \\
p_2 &= \frac{s^2+s-1}{3s^2+5s+1}, \\
p_3 = p_4 &= \frac{(s+1)^2}{3s^2+5s+1}.
\end{aligned}$$

Let $\Delta_j = C_{4,j} - c_{\mathrm{low}}$, where c_{low} is the desired lower bound and $C_{4,j}$ for $j = 0, \ldots, 15$ is the competitive ratio of algorithm j, as defined by (2). We calculate Δ_i in terms of s using the above values of p_i. We find:

$$\begin{aligned}
\Delta_0 &= \frac{(s+1)(2s^2+2s-1)}{3s^2+5s+1}, \\
\Delta_1 = \Delta_4 &= \frac{(s+1)(s^2+s-1)}{3s^2+5s+1}, \\
\Delta_2 &= \frac{(s-1)(s+1)^2}{3s^2+5s+1}, \\
\Delta_3 = \Delta_5 = \Delta_6 = \Delta_{13} &= 0, \\
\Delta_7 &= \frac{(s+1)(-s^2+2)}{3s^2+5s+1}, \\
\Delta_8 &= \frac{(s+1)(2s^2+s-2)}{3s^2+5s+1}, \\
\Delta_9 &= \frac{(s-1)(s+1)(s^2+s-1)}{s(3s^2+5s+1)}, \\
\Delta_{10} &= \frac{(s-1)^2(s+1)^2}{s(3s^2+5s+1)}, \\
\Delta_{11} &= \frac{(s+1)(-s^2+2)}{s(3s^2+5s+1)}, \\
\Delta_{12} &= \frac{(s-1)(s+1)^3}{s(3s^2+5s+1)}, \\
\Delta_{14} &= \frac{s+1}{3s^2+5s+1}, \\
\Delta_{15} &= \frac{(s+1)(-s^3+3s+1)}{s(3s^2+5s+1)}.
\end{aligned}$$

All of the expressions for Δ_i are non-negative for $1 \leq s \leq \sqrt{2}$. Therefore, no algorithm is better than c_{low}-competitive for $1 \leq s \leq \sqrt{2}$. ∎

4 The memoryless algorithm

In this section we analyze the memoryless algorithm Bias: Whenever a job arrives, Bias schedules it on the fast machine with probability $s/2$, and otherwise assigns it to the slow machine. All random choices are made independently.

Theorem 4.1 *The competitive ratio of* Bias *is* $c_{\text{BIAS}} = (4-s)(1+s)/4$.

Proof. We prove that the worst case competitive ratio of Bias is achieved on a sequence of two jobs. (In fact, this is true even if the probability of assigning a job on the fast machine is a different constant than $s/2$.)

If a sequence of jobs contains three or more jobs then an optimal schedule has some two jobs scheduled on the same machine; denote their running times t, t', so that $t \geq t'$. We show that for any random choices of Bias for the other jobs, replacing the two jobs by a single job with running time $t+t'$ does not decrease the expected makespan. Since the optimal makespan does not change, the competitive ratio cannot decrease.

The two jobs can be assigned in three different ways: (i) t on the fast machine and t' on the slow machine, (ii) t' on the fast machine and t on the slow machine, and (iii) both on the same machine (fast or slow). The cases (i) and (ii) have the same probability $q = s(2-s)/4$. If we replace the two jobs by a single job of size $t+t'$, then the probability distribution changes so that instead of (i) and (ii), we schedule the new job with probability q on the fast machine, with probability q on the slow machine, and leave the distribution in case (iii) intact. To verify this, always schedule the job $t+t'$ where we previously assigned t, ignoring the previous decision for t'. To prove that the expected makespan does not decrease after the replacement, we distinguish three cases.

(I) The makespan both in (i) and (ii) is determined by the slow machine. Let l be the total time of jobs assigned to the slow machine without t and t'. The contribution of (i) and (ii) to the expected makespan in the original schedule is $q(l+t) + q(l+t')$. In the new schedule with a job of size $t+t'$ it is at least $q(l+t+t') + ql$ (it can be larger if the makespan is no longer determined by the slow machine). So the makespan does not decrease.

(II) The makespan both in (i) and (ii) is determined by the fast machine. This is identical to (I), except that all the contributions are divided by s.

(III) The makespan both in (i) and (ii) is determined by the machine with t. If we always schedule t' on the machine with t, the makespan never decreases.

Thus the worst case is a sequence with at most two jobs. In the worst case, the optimal schedule has the machines exactly balanced: otherwise we could increase the size of one of the two jobs without changing the optimum and the expected makespan of the online algorithm does not decrease. Thus the worst case sequence consists of two jobs of sizes 1 and s, and is easy to calculate that the competitive ratio on such a sequence is c_{BIAS}. ∎

5 The algorithm for small s

We present an algorithm with good competitive ratio for $1 \le s \le 3/2$. The algorithm uses as a subroutine the randomized 4/3-competitive algorithm of Bartal *et al.* [4] for scheduling on two identical machines.

The algorithm FAIR is defined as follows: We flip an unbiased coin. If it comes up heads, we schedule all jobs using the Bartal *et al.* algorithm (ignoring machine speeds) with the first job going on the fast machine. If it comes up tails, we schedule all jobs using the Bartal *et al.* algorithm with the first job going on the slow machine.

While the following upper bound holds for any s, for $s > 3/2$ we have better algorithms. In the interval $s \in [1, 3/2]$, the bound is increasing, achieving the maximum of $3/2 + 1/36 < 1.52778$ for $s = 3/2$.

Theorem 5.1 *For any $s \ge 1$,* FAIR *is c_{FAIR}-competitive, where*

$$c_{\text{FAIR}} = \frac{(3s+1)(1+s)}{6s}.$$

Proof. First we briefly describe the algorithm of Bartal *et al.* for two identical machines and introduce some notation. Given a schedule, we call the machine in that schedule with with the largest amount of processing assigned to it the *tall* machine. The other machine is the *short* machine. At each point the algorithm keeps track of all schedules $\pi_1, \ldots, \pi_i, \ldots$ that it might have chosen so far (after j jobs there are at most 2^j schedules) and of their probability distribution. In schedule π_i, we let x_i, y_i be the total load on the tall and short machine, respectively. We let p_i be the probability of π_i. When a new job arrives, let $x = \sum_i p_i x_i$ and $y = \sum_i p_i y_i$ be the expectations of the total load on the tall machine and the short machine, respectively. The algorithm schedules each job maintaining $x \ge 2y$. The intuition behind this invariant is that the algorithm should keep y, the load on the short machine, relatively low in order to be ready for the possibility of a large job arriving. For some probability p, the algorithm schedules the new job with probability p on the tall machine and with probability $1-p$ on the short machine (independently of the previous choices). The algorithm chooses the smallest p such that the invariant is preserved. For $p = 1$ (the new job is always scheduled on the tall machine) the invariant is preserved, hence "the smallest p" is well-defined. Note that this algorithm completely ignores the speed of the machines.

We may assume, without loss of generality, the sum of the sizes of all jobs is 1 (we rescale the jobs after each new one; note that the behavior of the algorithm is not affected by scaling). Let

$$g(x) = \frac{3x-1}{2}.$$

Seiden [14] presents an alternative proof that the algorithm for $s = 1$ is 4/3 competitive, and shows that the algorithm maintains the following invariants:

1. The optimal offline cost is at least $g(x)$.

2. If $g(x) > 1/2$ (i.e., $x > 2/3$) then a job of size at least $g(x)$ was scheduled.

We now begin the analysis of Fair, using the facts above. The probability that the machine with load x_i in π_i is the slow one is $1/2$. Therefore, the expected makespan in π_i is $(x_i + \max\{x_i/s, y_i\})/2$. We partition the schedules into two sets A and B according to the order of the terms in the maximum, and define the expectations x_A, y_A, x_B, y_B of x_i and y_i over these sets:

$$\begin{aligned}
A &= \{i \mid x_i < s\, y_i\},\\
B &= \{i \mid x_i \geq s\, y_i\},\\
P &= \sum_{i\in A} p_i,\\
x_A &= \frac{1}{P}\sum_{i\in A} p_i\, x_i,\\
y_A &= \frac{1}{P}\sum_{i\in A} p_i\, y_i,\\
x_B &= \frac{1}{1-P}\sum_{i\in B} p_i\, x_i,\\
y_B &= \frac{1}{1-P}\sum_{i\in B} p_i\, y_i.
\end{aligned}$$

Since we assume that the sum of all job sizes is 1, we have $y_A = 1 - x_A$ and $y_B = 1 - x_B$. The optimal offline makespan x_{opt} is bounded by $x_{\mathrm{opt}} \geq g(x)/s$ and $x_{\mathrm{opt}} \geq 1/(s+1)$. Let X be the expected makespan of Fair. We have

$$\begin{aligned}
X &= \tfrac{1}{2}\sum_{i} p_i(x_i + \max\{x_i/s, y_i\})\\
&= \tfrac{1}{2}\sum_{i\in A} p_i(x_i + y_i) + \tfrac{1}{2}\sum_{i\in B} p_i(x_i + x_i/s)\\
&= \frac{sP + (s+1)(1-P)x_B}{2s}.
\end{aligned}$$

We begin by bounding x_B:

Lemma 5.2 *For $P < \frac{2}{3}$ we have*

$$(1-P)x_B \leq \frac{1 + s(2-3P)x_{\mathrm{opt}}}{3}.$$

Proof. Since $g(x)$ is a lower bound on the makespan in any schedule for the identical machine problem, we have $x_A \geq g(x)$ and also $x_{\mathrm{opt}} \geq g(x)/s$. Thus

$$x_A \;\geq\; g(x) \;=\; \tfrac{3}{2}x - \tfrac{1}{2} \;=\; \tfrac{3}{2}Px_A + \tfrac{3}{2}(1-P)x_B - \tfrac{1}{2}.$$

This implies that, for $P < \frac{2}{3}$,

$$x_A \geq \frac{3(1-P)x_B - 1}{2-3P}.$$

Together with $x_{\text{opt}} \geq g(x)/s$ we obtain

$$\begin{aligned}
sx_{\text{opt}} &\geq g(x) \;=\; \tfrac{3}{2}Px_A + \tfrac{3}{2}(1-P)x_B - \tfrac{1}{2} \\
&\geq \tfrac{3}{2}P\frac{3(1-P)x_B - 1}{2-3P} + \tfrac{3}{2}(1-P)x_B - \tfrac{1}{2} \\
&= \frac{1}{2}\left(\frac{3P}{2-3P} + 1\right)(3(1-P)x_B - 1) \\
&= \frac{1}{2-3P}(3(1-P)x_B - 1),
\end{aligned}$$

which implies the desired bound on x_B. ■

When $P < \frac{2}{3}$, we use Lemma 5.2 and the inequality $1 \leq (s+1)x_{\text{opt}}$ to obtain

$$\begin{aligned}
X &= \frac{sP + (s+1)(1-P)x_B}{2s} \\
&\leq \frac{3sP + (s+1)(1 + s(2-3P)x_{\text{opt}})}{6s} \\
&\leq \frac{(s+1)(3s+1)}{6s}x_{\text{opt}}.
\end{aligned}$$

For $P \geq \frac{2}{3}$ we use $x_B \leq 1$ and $1 \leq (s+1)x_{\text{opt}}$:

$$\begin{aligned}
X &\leq \frac{sP + (1-P)(s+1)}{2s} \\
&= \frac{s+1-P}{2s} \\
&\leq \frac{3s+1}{6s} \\
&\leq \frac{(s+1)(3s+1)}{6s}x_{\text{opt}}.
\end{aligned}$$

In both cases it follows that $X \leq c_{\text{FAIR}} \cdot x_{\text{opt}}$. ■

6 Algorithms for medium s

We define a new class of randomized online scheduling algorithms for two uniform machines. Each instance of the class is defined by fixing an integer $\ell \geq 2$ and providing a deterministic online algorithm $\mathcal{A}$ for scheduling on ℓ identical machines $I_1, \ldots, I_\ell$. We call these machines the *imaginary* machines because they do not really exist. They simply aid in visualizing what the algorithm is doing. Given ℓ and $\mathcal{A}$ the algorithm Pick One behaves as follows:

1. Pick $k \in \{1, \ldots, \ell\}$ randomly and uniformly.

2. For each job J: Give J to $\mathcal{A}$. If $\mathcal{A}$ puts J on I_k then put J on the slow machine. Otherwise put J on the fast machine.

Note that the algorithm Fair can also be viewed as Pick One using $\ell = 2$ the randomized algorithm of Bartal *et al.* for two identical machines.

6.1 A general analysis of Pick One

Let L_i denote the load on I_i. W.l.o.g., we assume

$$L_i \leq L_{i+1} \qquad \text{for } i = 1, \ldots, \ell - 1. \tag{3}$$

So L_ℓ is the makespan of the algorithm $\mathcal{A}$ on the imaginary machines. Let T denote the size of the largest job. Let $\mathcal{L}$ denote the total size of all jobs, i.e.,

$$\mathcal{L} = \sum_{i=1}^{\ell} L_i. \tag{4}$$

W.l.o.g., we assume that the optimal offline cost for the two machine problem is 1. We have

$$1 \geq \frac{1}{s+1}\mathcal{L}, \tag{5}$$

$$1 \geq T/s, \tag{6}$$

For a general analysis of Pick One we assume that the makespan of the algorithm $\mathcal{A}$ is upper bounded in terms of T and $\mathcal{L}$:

$$L_\ell \leq f(T, \mathcal{L}) \tag{7}$$

The expected makespan of Pick One is

$$\begin{aligned} \frac{1}{\ell}\sum_{i=1}^{\ell} \max\{(\mathcal{L} - L_i)/s, L_i\} &= \frac{1}{\ell s}\left(-\mathcal{L} + \sum_{i=1}^{\ell} \max\{\mathcal{L}, (s+1)L_i\}\right) \\ &= \frac{i_0 - 1}{\ell s}\mathcal{L} + \frac{s+1}{\ell s}\sum_{i=i_0+1}^{\ell} L_i, \end{aligned} \tag{8}$$

where $i_0 = \max\{i \mid \mathcal{L} \geq (s+1)L_i\}$. Since $L_{\ell-2}$ is the third largest load and $s \leq 2$, we have $\mathcal{L} \geq 3L_{\ell-2} \geq (s+1)L_{\ell-2}$, and thus $i_0 \in \{\ell-2, \ell-1, \ell\}$.

In order to show that Pick One is c-competitive, we show that the mathematical program: maximize (8) subject to (3), (4), (5), (6) and (7) has solution at most c for each $i_0 \in \{\ell-2, \ell-1, \ell\}$. We call this mathematical program $\mathcal{P}$.

We are able to bound the value of $\mathcal{P}$ in the cases $i_0 = \ell$ and $i_0 = \ell - 2$, without any knowledge of the scheduling algorithm used.

Lemma 6.1 *For $i_0 = \ell - 2$ or $i_0 = \ell$, program $\mathcal{P}$ has solution at most*

$$\frac{(s+1)(\ell+s-2)}{\ell s}. \tag{9}$$

Proof. If $i_0 = \ell - 2$ the makespan is

$$\frac{\ell-3}{\ell s}\mathcal{L} + \frac{s+1}{\ell s}(L_{\ell-1} + L_\ell) \;\leq\; \frac{\ell+s-2}{\ell s}\mathcal{L} \;\leq\; \frac{(s+1)(\ell+s-2)}{\ell s},$$

using (5). If $i_0 = \ell$ the makespan is $(\ell - 1)\mathcal{L}/(\ell s)$, and the lemma follows again from (5). ∎

To show that Pick One is c competitive, we need therefore only show that the value of $\mathcal{P}$ is at most c for $i_0 = \ell - 1$, and that c is at least as large as (9).

It is possible to use List with Pick One, however, this does not yield any improvement over the memoryless algorithm Bias. We include the analysis of List with Pick One in Appendix A.

6.2 Analysis of Pick One with a new subroutine

We describe a new algorithm for $\ell \geq 3$ machines, which we call Prepared. Prepared is not in general a good algorithm for the ℓ machine problem, in that its competitiveness is worse than that of List. However, it does handle large jobs well. It does this by always keeping several machines lightly loaded, so that it is 'prepared' when a large job arrives. It always schedules a new job on one of the $k \leq \ell - 2$ least loaded machines. Note that when a job is assigned to a lightly loaded machine this machine will become more heavily loaded. If necessary reorder the machines so that $L_i \leq L_{i+1}$ for all i. By varying the parameters ℓ and k of the algorithm Prepared, we are able to get a family of algorithms whose competitive ratio beats Fair for all $s \in (3/2, 2)$.

For some $k \leq \ell - 2$, the algorithm has parameters $\alpha_1, \ldots, \alpha_k$ which satisfy

$$\begin{array}{rcll} 0 & \leq & \alpha_1; & \\ \alpha_i & \leq & \alpha_{i+1}, & \text{for } i = 1, \ldots, k; \\ \alpha_i & = & \dfrac{1}{\ell - i + 1} & \text{for } i = k+1, \ldots, \ell. \end{array} \tag{10}$$

The algorithm Prepared is defined as follows: if a new job J of size t arrives, then

1. For $i \leftarrow 1, \ldots, k$: if $L_{j+1} \leq \alpha_j(\mathcal{L} + t)$ for all $j = i, \ldots, k$, then put J on I_i.
2. Otherwise put J on I_{k+1}.

The next lemma gives some intuition about the role of the parameters α_i.

Lemma 6.2 *For all $i = 1, \ldots, \ell$, Prepared maintains the invariants $L_i \leq \alpha_i \mathcal{L}$.*

Proof. The bounds are trivial for $i = k+1, \ldots, \ell$. The proof for $i = 1, \ldots, k$ is by induction on the length of the job sequence.

For the basis, the stated invariants obviously hold before any jobs are processed.

For the inductive step, consider the scheduling of some job J. We let the machine loads be $L_1, \ldots, L_\ell$ before J, and $L'_1, \ldots, L'_\ell$ after J. Let $\mathcal{L} = \sum_i L_i$ and $\mathcal{L}' = \sum_i L'_i$. Assuming $L_i \leq \alpha_i \mathcal{L}$, we show that $L'_i \leq \alpha_i \mathcal{L}'$. Suppose J is scheduled on I_i. By induction, we have $L'_p = L_p \leq \alpha_p \mathcal{L} \leq \alpha_p \mathcal{L}'$ for $p = 1, \ldots, i-1$. Let $j \geq i$ be the new index of I_i, after the machines are renamed so that machine loads are sorted. We have $L'_j = L_i + t$ and $L'_p = L_{p+1}$ for $p = i, \ldots, j-1$. Since J was scheduled on I_i, by definition of PREPARED we have that $L_{p+1} \leq \alpha_p(\mathcal{L} + t)$ for $p = i, \ldots, k$. We have two cases.

First consider $j \leq k$. We have $L'_p = L_{p+1} \leq \alpha_p(\mathcal{L}+t) = \alpha_p \mathcal{L}'$ for $p = i, \ldots, j-1 < k$. Also, we have $L'_j = L_i + t \leq L_{j+1} \leq \alpha_j(\mathcal{L} + t) = \alpha_j \mathcal{L}'$. Finally, note that $L'_p = L_p \leq \alpha_p \mathcal{L} \leq \alpha_p \mathcal{L}'$ for $p = j+1, \ldots, k$.

Now consider $j > k$. We have $L'_p = L_{p+1} \leq \alpha_p(\mathcal{L} + t) = \alpha_p \mathcal{L}'$ for $p = i, \ldots, k$. ∎

The following theorem summarizes the results of our analysis of PREPARED, and determines our choice of parameters α_i and β.

Theorem 6.3 *Let $k \leq \ell - 2$ be a positive integer and let $\beta : [0,1] \mapsto [0,1]$ be any continuous function satisfying $\beta(x) \geq x$ for all x and $\beta(0) < 1$. Then there exists a unique $\psi \in [0,1]$ which satisfies*

$$\begin{aligned} \alpha_1 &= \psi; \\ \alpha_{i+1} &= \frac{\alpha_i}{\alpha_i + 1 - \beta(\alpha_1)}, \quad \textit{for } i = 1, \ldots, k-1; \\ \alpha_{k+1} &= \frac{1}{\ell - k}. \end{aligned} \tag{11}$$

Furthermore, for all $\alpha_1 \in [\psi, 1]$ if we define $\alpha_2, \ldots, \alpha_k$ using (11) then (10) are satisfied and the makespan of PREPARED *is bounded by*

$$f(T, \mathcal{L}) = \max\left\{\beta(\alpha_1)\,\mathcal{L}, (1-\alpha_1)T + \alpha_1 \mathcal{L}\right\},$$

where T is the size of the largest job.

Proof. We first prove that ψ exists. Compute $\alpha_2, \ldots, \alpha_{k+1}$ as functions of α_1, using the recurrence (11). By induction it is easy to see that the denominator of the recurrence is positive, hence each α_i is a continuous and non-negative function of α_1; in addition $\alpha_i = 0$ iff $\alpha_1 = 0$ and $\alpha_i = 1$ for $\alpha_1 = 1$. By the mean value theorem it follows that the equation $\alpha_{k+1} = 1/(\ell - k)$ (written in terms of α_1) has a solution $\psi \in [0,1]$; fix such a ψ. It remains to argue that α_i are non-decreasing for $\alpha_1 \in [\psi, 1]$. We show by induction that $\alpha_i \leq \beta$; this is sufficient since then the denominator in the recurrence is at most 1. From the definition of β we have $\alpha_1 \leq \beta(\alpha_1)$. Now $\alpha_i/(\alpha_i + 1 - \beta(\alpha_1))$ is increasing in α_i, since $\beta \leq 1$, and thus it has a maximum for $\alpha_i = \beta$, in which case it is β.

Define

$$\begin{aligned} T_1 &= 1, \\ T_i &= 1 - \alpha_{i-1}/\alpha_i, \quad \text{for } i = 2, \ldots, k+1 \\ g_i(T, \mathcal{L}) &= (1 - \alpha_i)T + \alpha_i \mathcal{L}, \\ g(T, \mathcal{L}) &= \max_{i=1,\ldots,k+1} g_i\left(\min\{T, T_i\, \mathcal{L}\}, \mathcal{L}\right). \end{aligned}$$

We claim that

$$L_\ell \leq g(T, \mathcal{L}).$$

The proof is by induction on the length of the job sequence. Consider the scheduling of a job J. Let T be the size of the largest job before J, and t be the size of J, and $T' = \max\{T, t\}$. As in the proof of Lemma 6.2, we let the machine loads be $L_1, \ldots, L_\ell$ before J, and $L'_1, \ldots, L'_\ell$ after J. Let $\mathcal{L} = \sum_i L_i$ and $\mathcal{L}' = \sum_i L'_i$.

For the basis, if J is the first job, then after it arrives $t = T' = L'_\ell = \mathcal{L}'$ and

$$L'_\ell = g_{k+1}(L'_\ell, L'_\ell) = g_{k+1}(T', \mathcal{L}') = g_{k+1}(\min\{T', T_{k+1}\, \mathcal{L}'\}, \mathcal{L}') \leq g(T', \mathcal{L}').$$

If when J is scheduled the makespan does not increase, we have

$$L'_\ell = L_\ell \leq g(T, \mathcal{L}) \leq g(\max\{T, t\}, \mathcal{L} + t) = g(T', \mathcal{L}'),$$

where the first inequality is by the inductive hypothesis and the second is by the fact that $g(T, \mathcal{L})$ is non-decreasing in both T and $\mathcal{L}$. Therefore we may assume from this point on that the makespan increases.

If J is scheduled on I_i and the makespan increases then we have

$$L'_\ell = L_i + t \leq \alpha_i \mathcal{L} + t = t + \alpha_i(\mathcal{L}' - t) = g_i(t, L').$$

First consider $i > 1$. From the definition of the algorithm, if J was scheduled on I_i then $L_{j+1} \leq \alpha_j(\mathcal{L} + t)$ for $j = i, \ldots, k$. Also, if J was not scheduled on I_{i-1} then $L_{j+1} \leq \alpha_j(\mathcal{L} + t)$ does not hold for all $j = i-1, \ldots, k$. From these two facts we have $L_i > \alpha_{i-1}(\mathcal{L} + t)$. This implies that

$$t \leq \frac{L_i}{\alpha_{i-1}} - \mathcal{L} \leq \left(\frac{\alpha_i}{\alpha_{i-1}} - 1\right)\mathcal{L}.$$

This holds if and only if

$$\frac{\alpha_i}{\alpha_{i-1}} t \leq \left(\frac{\alpha_i}{\alpha_{i-1}} - 1\right)(\mathcal{L} + t),$$

which implies

$$t \leq \left(1 - \frac{\alpha_{i-1}}{\alpha_i}\right)(\mathcal{L} + t) = T_i \mathcal{L}'.$$

Therefore

$$L'_\ell = g_i(t, L') = g_i(\min\{t, T_i\,\mathcal{L}'\}, L') \le g_i(\min\{T', T_i\,\mathcal{L}'\}, L') \le g(T', L').$$

Now consider $i = 1$. In this case,

$$g_i(t, L') \le g_{k+1}(T', L') = g_{k+1}(\min\{T', \mathcal{L}'\}, L') = g_{k+1}(\min\{T', T_{k+1}\,\mathcal{L}'\}, L') \le g(T', L').$$

It remains to prove that

$$g(T, \mathcal{L}) \le f_{\text{PREPARED}} = \max\left\{\beta\mathcal{L}, (1-\alpha_1)T + \alpha_1\mathcal{L}\right\}.$$

The second term in the above maximum is $g_1(T, \mathcal{L})$. For $i = 2, \ldots, k+1$ we have

$$\begin{aligned} g_i(\min\{T, T_i\,\mathcal{L}\}, \mathcal{L}) &\le g_i(T_i\,\mathcal{L}, \mathcal{L}) = T_i\mathcal{L} + (\mathcal{L} - T_i\,\mathcal{L})\alpha_i = (1-\alpha_i)T_i\mathcal{L} + \alpha_i\mathcal{L} \\ &= (1-\alpha_i)\left(1 - \frac{\alpha_{i-1}}{\alpha_i}\right)\mathcal{L} + \alpha_i\mathcal{L} = \mathcal{L} - \frac{\alpha_{i-1}(1-\alpha_i)}{\alpha_i}\mathcal{L} = \beta\mathcal{L}, \end{aligned}$$

using the equation (11). (This is also the reason for our choice of picking $\alpha_1, \ldots, \alpha_k$.) ∎

We choose

$$\beta(\alpha_1) = \frac{1+\alpha_1}{s+1}$$

This choice guarantees that $f_{\text{PREPARED}} = \beta(\alpha_1)\,\mathcal{L} = (1-\alpha_1)T + \alpha_1\mathcal{L}$ for the worst case value of T. Let

$$c_{\text{PICK-PREPARED-}k} = \min_{\ell \ge k+2} \frac{(s+1)(\ell - 2 + s + \alpha_1)}{\ell s}.$$

Theorem 6.4 *For all $k \ge 1$,* Pick One *with* Prepared *is $c_{\text{PICK-PREPARED-}k}$-competitive for all $1 \le s \le 2$.*

Proof. We show that for $f(T, \mathcal{L}) = f_{\text{PREPARED}}(T, \mathcal{L})$, program $\mathcal{P}$ has solution at most

$$\frac{(s+1)(\ell - 2 + s + \alpha_1)}{\ell s}. \tag{12}$$

Note that (12) is greater than (9), since $\alpha_1 > 0$. Therefore, by Lemma 6.1 it is sufficient to bound the makespan in the case $i_0 = \ell - 1$. The makespan is bounded by

$$\begin{aligned} &\frac{\ell-2}{\ell s}\mathcal{L} + \frac{s+1}{\ell s} f_{\text{PREPARED}}(T, \mathcal{L}) \\ \le\; &\frac{\ell-2}{\ell s}\mathcal{L} + \frac{s+1}{\ell s}\max\left\{\frac{s+\alpha_1}{s+1}\mathcal{L}, (1-\alpha_1)T + \alpha_1\mathcal{L}\right\} \\ \le\; &\frac{(s+1)(\ell-2)}{\ell s} + \frac{s+1}{\ell s}(s+\alpha_1) \\ =\; &\frac{(s+1)(\ell + s + \alpha_1 - 2)}{\ell s}. \end{aligned}$$

The second inequality follows from (5) and (6). The theorem now follows from the fact that the algorithm picks whichever ℓ is best for a particular s. ■

For $k = 1, 2$ we are able to solve the equations (11) explicitly. For $k = 1$ we obtain

$$\alpha_1 = \frac{1}{s(\ell - 2) + \ell - 1}.$$

For $k = 2$ we obtain

$$\begin{aligned} A &= (3s + 1)(s + 2) - (s + 1)^2 \ell, \\ \alpha_1 &= \frac{A + \sqrt{4s + A^2}}{2s}. \end{aligned}$$

For $k \geq 3$, we are unable to give a compact closed form expression for α_1. However, note that Theorem 6.3 allows us the freedom of using an approximate solution (as long as it is greater than the real one).

We display $c_{\text{PICK-PREPARED-}k}$ for $k = 1, \ldots, 4$ in Figure 2. For $s = 3/2$, the upper bound achieved for both $k = 1, 2$ is equal to $3/2 + 1/36 < 1.52778$, i.e., the same as for the algorithm FAIR. For any $s \in (3/2, 2)$, the upper bound is strictly smaller even for $k = 2$. Thus we have an algorithm with competitive ratio at most $3/2 + 1/36 < 1.52778$ for any speed.

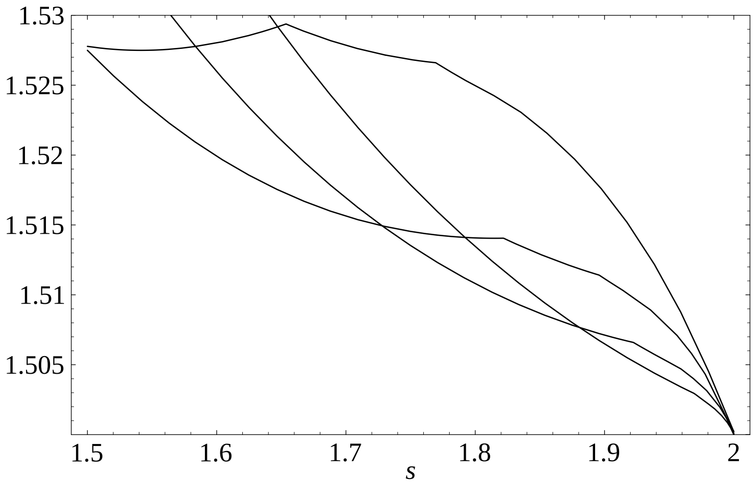

Figure 2: The curves $c_{\text{PICK-PREPARED-}k}$ with $k = 1, \ldots, 4$ for $\frac{3}{2} \leq s \leq 2$. (The curves are in decreasing order in the neighborhood of 2.)

7 Preemptive scheduling

We give tight bounds for preemptive scheduling for any speed. We show that with preemption randomization does not help: our algorithm is deterministic while the lower bound holds also for randomized algorithms.

Recall that in the preemptive case, we are allowed to interrupt a job and continue its processing on the same machine or on a different machine. However, all such decisions must be made when the job is first revealed, i.e., before we have any knowledge about future jobs.

In this model it is not a priori clear that introducing idle time is not useful. It is also possible to introduce temporarily idle time (e.g. by preempting the first job) and use this time slot for later jobs. However, our online algorithm never introduces idle time, while the lower bound holds even for algorithms with idle time (some care is needed in the formulation of the lower bound proof to achieve this).

Theorem 7.1 *There exists a deterministic online algorithm for preemptive scheduling on two machines with speeds 1 and s that is c_{pmtn}-competitive, where*

$$c_{pmtn} \;=\; 1+\frac{s}{s^2+s+1} \;=\; \frac{(s+1)^2}{s^2+s+1}.$$

For any s, no randomized online algorithm achieves a better competitive ratio than c_{pmtn}.

Proof. First we prove the upper bound. Let x be the total size of all jobs scheduled on the fast machine so far, y the total size of all jobs scheduled on the slow machine, and $t_{\max}$ the size of the largest job so far. The algorithm will maintain the following two invariants:

$$y \quad \leq \quad \frac{x}{s^2+s}; \tag{13}$$

$$\text{if } y < \frac{x}{s^2+s}, \quad \text{then} \quad x \leq c_{\text{pmtn}} \cdot t_{\max} \tag{14}$$

Note that the first invariant implies that $x/s \geq y$ and thus the makespan of our schedule is always x/s. If $y = x/(s^2+s)$, then the optimal schedule has makespan at least

$$\frac{1}{1+s}\,(x+y) \;=\; \frac{1}{1+s}\left(1+\frac{1}{s^2+s}\right)x \;=\; \frac{1}{c_{\text{pmtn}}}\cdot\frac{x}{s},$$

and thus the competitive ratio is at most c_{pmtn}. If $y < x/(s^2+s)$ then $x/s \leq c_{\text{pmtn}} \cdot t_{\max}/s$ and the algorithm is also c_{pmtn} competitive since $t_{\max}/s$ is a lower bound on the optimal makespan.

It remains to be proven that it is possible to maintain the invariant. It is true at the beginning. When a new job arrives we distinguish three cases. Let t be the size of the new job, and let x' and y' denote the loads after scheduling it.

If it is possible to schedule the new job so that $y' = x'/(s^2+s)$ and $y' \leq x/s$, we do so. The first condition guarantees that the invariant is maintained. The second condition guarantees that the portion of the job scheduled on the slow machine runs in parallel with previous jobs on the fast machine, and we do not need to introduce any idle time. This is the case when we have no big job.

If the previous case does not apply and the size of the new job t satisfies $t \geq x/s - y$, we schedule it so that $y' = x/s$. This means that the job is big and we schedule as much of it as possible on the slow machine in parallel with previous jobs on the fast machine. The invariant (13) holds since otherwise we would have scheduled the job in the previous case (putting less of the job on the slow machine). This implies that $x' \geq y'(s^2+s) = x(s+1)$. Since the invariant (13) was satisfied before this step, we have $y \leq x/(s^2+s)$ and we bound t using the previous inequality:

$$\begin{aligned} t &= (x'-x)+(y'-y) \;\geq\; x'-x+\left(\frac{x}{s}-\frac{x}{s^2+s}\right) \;=\; x'-\frac{s}{s+1}\cdot x \\ &\geq \left(1-\frac{s}{(s+1)^2}\right)x' = \frac{1}{c_{\mathrm{pmtn}}}\,x' \end{aligned}$$

and the invariant (14) is satisfied.

The remaining possibility is that $t < x/s - y$ and even after scheduling the new job on the slow machine we have $y' < x'/(s^2+s)$. This happens if a small job arrives after a big job. We schedule the job on the small machine, in parallel with the previous jobs on the fast machine. The condition above guarantees that the invariant (13) is maintained. The load of the fast machine does not change, and thus also the invariant (14) is true (the size of the largest job does not decrease).

To prove the lower bound, suppose we have a c-competitive randomized algorithm. Consider the first three jobs from the critical sequence, with running times 1, s, and s^2+s. Let T resp. T' be the expected makespan after scheduling the first two jobs, resp. all three jobs. Since the corresponding optimal schedules have length 1 and $s+1$, it must be the case that $T \leq c$ and $T' \leq c(s+1)$. After scheduling the last job, the expected time when both machines are busy is bounded by T, and even if the faster machine is used all the rest of the schedule, the expected total size of scheduled jobs is at most $T + sT' \leq c(1+s+s^2)$. Comparing with the size of the three jobs we obtain $c \geq (1+2s+s^2)/(1+s+s^2) = c_{\mathrm{pmtn}}$. ∎

8 Conclusions

In this paper, we have settled the preemptive version, both deterministic and randomized, for all speeds, the non-preemptive deterministic version for all speeds, and the non-preemptive randomized version for all speeds $s \geq 2$. In the remaining case, non-preemptive randomized scheduling with speeds $1 < s < 2$, we have developed a family of competitive algorithms and a class of almost matching lower bounds.

Settling the non-preemptive randomized case for speeds $1 < s < 2$ completely seems to be very hard. We conjecture that the optimal competitive ratio is at most 3/2 for any speed in this range. However, we do not even have a reasonable guess how the optimal competitive ratio should depend upon s or what the optimal algorithm should be.

On the critical sequence with $1 < s < 2$, even the algorithm Pick One with List would be 3/2-competitive. However, there exist other sequences on which Pick One with any deterministic scheduling algorithm on 4 machines is worse than 3/2 competitive at speed $s = 3/2$ (and considering the expected loads of the machines, at this speed it is necessary to use 4 machines). Such a sequence consists, for example, of three jobs of sizes 4, 5, and 6, in this order, possibly followed by jobs of size $3/2 \cdot 15$ and $(3/2)^2 \cdot 15$.

There are still possibilities to overcome these difficulties within the framework of Pick One: we could allow the subroutine to schedule some jobs on none or on more than one of the machines, thus changing the probability for assignments to the slow machine. Alternatively, we could use a randomized algorithm instead of a deterministic one in the subroutine. However, both modifications seem to be very hard to analyze.

Another open question concerns (preemptive and non-preemptive, deterministic and randomized) online scheduling on $m \geq 3$ machines. We conjecture that for any number $m \geq 3$ of machines and for any combination of machine speeds, the optimal competitive ratio for the deterministic preemptive version is strictly better than the optimal competitive ratio for the randomized non-preemptive version.

References

[1] S. Albers. Better bounds for online scheduling. *SIAM J. Comput.*, 29(2):459–473, 1999.

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. *J. ACM*, 44(3):486–504, 1997.

[3] Y. Bartal, M. Chrobak, and L. L. Larmore. A randomized algorithm for two servers on the line. *Inf. Comput.*, 158(1):53–69, 2000.

[4] 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.

[5] S. Ben-David, E. Dichterman, J. Noga, and S. Seiden. On the power of barely random online algorithms. Manuscript, 1998.

[6] P. Berman, M. Charikar, and M. Karpinski. On-line load balancing for related machines. In *Proc. of the 5th WADS, LNCS 1272*, pages 116–125. Springer-Verlag, 1997.

[7] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[8] B. Chen, A. van Vliet, and G. J. Woeginger. New lower and upper bounds for on-line scheduling. *Oper. Res. Lett.*, 16:221–230, 1994.

[9] Y. Cho and S. Sahni. Bounds for list schedules on uniform processors. *SIAM J. Comput.*, 9(1):91–103, 1980.

[10] M. Chrobak, L. L. Larmore, C. Lund, and N. Reingold. A better lower bound on the competitive ratio of the randomized 2-server problem. *Inf. Process. Lett.*, 1997.

[11] L. Epstein and J. Sgall. A lower bound for on-line scheduling on uniformly related machines. *Oper. Res. Lett.*, 26(1):17–22, 2000.

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

[13] L. A. McGeoch and D. D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

[14] S. S. Seiden. Randomized online multiprocessor scheduling. *Algorithmica*, 28(2):173–216, 2000.

[15] S. S. Seiden. Barely Random Algorithms for Multiprocessor Scheduling. Manuscript, 2000.

[16] J. Sgall. On-line scheduling. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, volume 1442 of *LNCS*, pages 196–231. Springer-Verlag, 1998.

[17] J. Wen and D. Du. Preemptive on-line scheduling for two uniform processors. *Oper. Res. Lett.*, 23:113–116, 1998.

[18] G. J. Woeginger. A comment on scheduling on uniform machines under chain-type precedence constraints. *Oper. Res. Lett.*, 26(3):107–109, 2000.

[19] A. C.-C. Yao. Probabilistic computations: Towards a unified measure of complexity. In *Proc. of the 18th FOCS*, pages 222–227. IEEE, 1977.

A Analysis of Prepared with List

We analyze the combination of Pick One and List, using the general analysis developed in Section 6. From [12], one can easily derive that

$$L_\ell \le f_{\text{LIST}}(T, \mathcal{L}) = \frac{\ell - 1}{\ell} T + \frac{1}{\ell}\mathcal{L}.$$

Lemma A.1 *For $f = f_{\text{LIST}}$ and $i_0 = \ell - 1$, program $\mathcal{P}$ has solution at most*

$$\frac{(s+1)(\ell^2 + \ell s - 2\ell + 1)}{\ell^2 s}. \tag{15}$$

Proof. In this case the makespan is

$$\begin{aligned}
\frac{\ell-2}{\ell}\mathcal{L} + \frac{s+1}{\ell}L_\ell &\le \frac{\ell-2}{\ell}\mathcal{L} + \frac{s+1}{\ell} f_{\text{LIST}}(T, \mathcal{L}) \\
&= \frac{\ell-2}{\ell}\mathcal{L} + \frac{s+1}{\ell}\left(\frac{\ell-1}{\ell}T + \frac{1}{\ell}\mathcal{L}\right) \\
&= \frac{\ell^2 - 2\ell + s + 1}{\ell^2}\mathcal{L} + \frac{(\ell-1)(s+1)}{\ell^2}T \\
&\le \frac{(s+1)(\ell^2 - 2\ell + s + 1)}{\ell^2 s} + \frac{(\ell-1)(s+1)}{\ell^2} \\
&= \frac{(s+1)(\ell^2 + \ell s - 2\ell + 1)}{\ell^2 s}.
\end{aligned}$$

The first inequality follows from (5) while the second follows from (6) and (7). ∎

Theorem A.2 Pick One *is $c_{\text{PICK-LIST}}$-competitive for $1 \le s \le 2$ when* List *is used, where*

$$\begin{aligned}
c(\ell) &= \frac{(s+1)(\ell^2 + \ell s - 2\ell + 1)}{\ell^2 s}, \\
c_{\text{PICK-LIST}} &= \min_{\ell \ge 2} c(\ell).
\end{aligned}$$

Proof. This follows directly from Lemmas 6.1 and A.1, and the fact that $c(\ell)$ is greater than (9) for all $\ell \ge 2$ and $1 \le s \le 2$. ∎

Note that the theorem implies that Pick One with List is c_{BIAS}-competitive for $s = 2 - 2/\ell$ for all integers $\ell \ge 2$. For any speed s, Bias is at least as good as Pick One with List: this is shown by a sequence of a number of small jobs of total size 1 followed by a job of size s. We display $c_{\text{PICK-LIST}}$ in Figure 3.

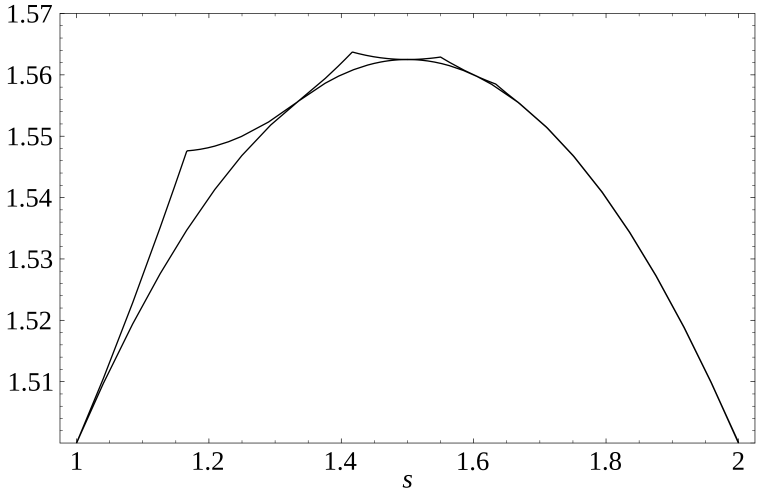

Figure 3: The functions c_{BIAS} and $c_{\text{PICK-LIST}}$ for $1 \leq s \leq 2$.